CASINO

User's Guide Version 2.13 (2022)

Richard Needs, Mike Towler, Neil Drummond and Pablo López Ríos

September 2019

Contents

1 Introduction

CASINO is a computer program system for performing quantum Monte Carlo (QMC) electronic structure calculations that has been developed by a group of researchers initially working in the Theory of Condensed Matter group in the Cambridge University physics department, and their collaborators, over more than 20 years. It is capable of calculating incredibly accurate solutions to the Schrödinger equation of quantum mechanics for realistic systems built from atoms.

Other highly accurate methods are known that can do this of course, but what makes CASINO important is that given a big enough computer it can carry on doing this for systems containing hundreds or even thousands of atoms. These many atoms can form isolated giant molecules or groups of molecules, or they can be the repeating unit in an infinite crystal periodic in one, two, or three dimensions. Because of their inferior scaling with system size, competing methods of comparable accuracy generally struggle to handle more than a few tens of atoms. CASINO is also important because QMC is one of the few methods genuinely capable of exploiting the power of modern computer hardware. Modern versions of CASINO, by contrast, have been shown to exhibit essentially perfect parallel efficiency on machines with hundreds of thousands of cores in all of its various modes of operation.

The QMC research program that culminated in CASINO as we know it today was begun in the early 1990s by Richard Needs, who had obtained inspiration from a short period working in the United States. He developed his initial ideas in collaboration with Guna Rajagopal and Matthew Foulkes and a number of early postdocs and students. Their initial test codes were gathered together and generalized into the first QMC program capable of treating any arbitrary system (named 'CASINO' after a suggestion from Paul Kent) by Mike Towler, who had arrived in Cambridge in late 1996. The first public version of CASINO was published in 1999. Absolutely fundamental improvements and generalizations of the code were made by Neil Drummond (from 2002) and Pablo López Ríos (from 2004). Needs, Towler, Drummond and López Ríos are now considered to be principal authors of the code, and all four of them continue to be at the heart of the project today. Sadly, CASINO was thrown off the Cambridge TCM group computer systems in September 2019 at the insistence of local computer management, despite Richard still working in the group, and as a result it is no longer possible to refer to it as the 'Cambridge quantum Monte Carlo code'.

Over the years, valuable additional contributions to the software base have been made by students, postdocs, and other people working in collaboration with us. A reasonably complete set would include Andrew Porter, Randy Hood, Andrew Williamson, Dario Alfè, Gavin Brown, Chris Pickard, Rene Gaudoin, Ben Wood, Zoltán Radnai, Andrea Ma, Ryo Maezono, John Trail, Paul Kent, Nick Hine, Alexander Badinski, Matthew Brown, Ken Esler, Andrew Morris, Norbert Nemec, Robert Lee, Priyanka Seth, Bohshiang Jong, Lucian Anton, Katie Schwarz, Pascal Bugnion, Jonathan Lloyd-Williams, Elaheh Mostaani, Mike Deible, Vladimir Konjkov, Albert Defusco, Blazej Jaworowski, Marcin Szyniszewski, Ryan Hunt, David Thomas, Chung-Yuan Ren, Darryl Foo, Graham Spink, John Jumper, Katharina Doblhoff-Dier, Kayahan Saritas, Kevin Gasperich, Lars Schonenberg, Richard Dawes, Rodrigo Vargas-Hernández, Thomas Whitehead, Tom Poole, Yassmin Asiri, Clio Johnson and Ben Thorpe. There may be others. We are very grateful for their contributions.

The following citation (quoted in full) is required in any publication describing results obtained with CASINO:

> R. J. Needs, M. D. Towler, N. D. Drummond, P. López Ríos and J. R. Trail,
> J. Chem. Phys. **152**, 154106 (2020).

Further public information and resources—including forms for downloading the code—are available at the CASINO web page:

`https://vallico.net/casinoqmc/`

2 The quantum Monte Carlo method

The correlated motion of electrons plays a crucial role in the aggregation of atoms into molecules and solids, in electronic transport properties and in many other important physical phenomena. *Ab initio* electronic structure calculations, in which the properties of such correlated electron systems are computed from first principles, are a vital tool in modern condensed matter physics and molecular quantum chemistry. Currently the most popular way to include the effects of electron correlation in these calculations is *density functional theory* (DFT). This method is in principle exact, in reality fast and often very accurate, but does have a certain number of well-known limitations. In particular, with only limited knowledge available concerning the exact mathematical form of the so-called exchange–correlation (XC) functional, the accuracy of the approximate form of the theory is nonuniform and nonuniversal, and there are important classes of materials for which it gives qualitatively wrong answers.

An important and complementary alternative for situations where accuracy is paramount is the *quantum Monte Carlo* (QMC) method, which has many attractive features for probing the electronic structure of real systems. It is an explicitly many-body method, applicable to both finite and periodic systems, which takes electron correlation into account from the outset. It gives consistent, highly accurate results while at the same time exhibiting favourable scaling of computational cost with system size. This is in sharp contrast to the accurate methods used in mainstream quantum chemistry, such as configuration interaction or coupled-cluster theory, which are impractical for anything other than small molecules, and cannot generally be applied to condensed matter problems.

The use of QMC was greatly hampered over its first two decades of existence by a combination of insufficient computer power and a lack of available, efficient QMC computer programs general enough to treat a similar range of problems to regular DFT codes. Fast parallel computers are now widespread, and you are now in possession of just such a computer program, which is capable of carrying out QMC calculations for a wide range of interesting chemical and physical problems on a variety of computational hardware. As a bonus, CASINO is capable of exploiting with essentially perfect parallel efficiency as many processors as you like (within reason)—a great asset in this era of petascale computing where machines with hundreds of thousands or even millions of cores are beginning to appear. CASINO has been designed to make the power of the QMC method available to everyone and we hope you enjoy using it.

3 Miscellaneous issues

3.1 Support

CASINO has documentation and examples, etc., but it is a research code, and learning how to use it is a significant task. This is particularly the case if the user does not have relevant experience, such as familiarity with the theory of variational Monte Carlo (VMC) and diffusion Monte Carlo (DMC) calculations and knowledge of DFT or Hartree–Fock (HF) methods for atoms, molecules and solids. We have found that supporting users can take up a large amount of our time and so we have to limit the number of groups that we work with directly. Most users need quite a lot of help and their projects turn into collaborations, but of course we cannot enter into too many projects of this type as our time is limited. We often have people visiting Cambridge to learn about the codes and how to do calculations, and this seems to work well.

Beginning in 2006 we also organize annual summer schools at MDT's Tuscan monastery (see `https://vallico.net/tti/tti.html`). The scientific side of these week-long events involves over 20 hours of lectures from the main authors of the code with plenty of hands-on tuition, and people who have attended these events have generally found them to be beneficial.

Since June 2013, the best place for asking questions about CASINO is over at the CASINO discussion forum `https://vallico.net/casino-forum/`. Unless you have a good reason for thinking some particular individual can help you, this forum should be the first place you ask for help. As of June 2014, around 500 CASINO users and developers belong to it, and so there is a very good chance your question will be answered.

There used to be a CASINO Users' mailing list (`casino-users 'at' tcm.phy.cam.ac.uk`) mainly used by the CASINO administrators for announcements of, e.g., events and new versions of the code, but this was discontinued in June 2014. All such announcements are now made in the 'General

Announcements' subforum of the CASINO Forum site; a subscription to email notifications from that subforum is entirely equivalent to the old mailing list.

3.2 Legal stuff

CASINO is given away for free to any academic or individual wishing to use it for non-profit making purposes (commercial companies should make a specific email enquiry). The software may be downloaded from our public website at `https://vallico.net/casinoqmc/download-casino/`. The download procedure involves ticking a checkbox agreeing to certain legal conditions, the practical upshot of which is that users may not redistribute the code, they may not incorporate any part of it into any other program, nor may they modify it in any way whatsoever *without prior agreement of the Cambridge group* (though this is usually very easy to obtain). You may not use or even retain a copy of CASINO if you have not recorded your agreement to these conditions. A copy of the agreement can be found in the CASINO distribution in the file `CASINO/doc/academic_consent_form.pdf` or `CASINO/doc/commercial_consent_form.pdf` (the latter should be used only in the case of profit-making companies).

3.3 Getting the latest version of the code

New versions of CASINO are produced on a regular basis (nightly builds for the beta version). People downloading the software for the first time should fill in the form on the following web page: `https://vallico.net/casinoqmc/download-casino/`. After filling in the form you will be able to download the standard version of the code immediately, and a request for a CASINO login ID and password will be triggered. This ID is required to update your copy of CASINO with later updates. It may also be used to access the CASINO online discussion forum, and to publish 'Blog' or 'Project' posts on the CASINO website. Tickboxes requesting these services, as well as ones for attending the annual summer school, or for offering your services for development work, are all part of the online registration form.

In the initial download, users have the option of downloading either the most recent stable version or the 'current beta version'. The current beta is a build with the latest changes released as they happen, which may or may not be suitable for production work, but is required for development work. Subsequent updates to the code may be downloaded from the site `https://vallico.net/casinoqmc/update-casino/` using your CASINO login ID and password.

Users in the Cambridge TCM group who belong to the Unix group 'casino' may additionally copy the latest source from the directory `~casino/PUBLIC/current_beta`, and access the `git` repository at `~casino/git/CASINO`.

If you have been accepted as a developer (see `https://vallico.net/casinoqmc/how-to-become-a-developer/`) then your CASINO login ID will also give you access to the developer version of the code, which you can download from `https://vallico.net/casinoqmc/download-developer/`. The developer version is identical to the standard version in all respects, except that the standard version has 'obfuscated' source code which has deliberately been made very difficult to read. The developer version has the full documented source code.

Note that there exists a mailing list through which all licensed CASINO developers (who are listed on the page `https://vallico.net/casinoqmc/things-to-do/`) are encouraged to communicate with each other. Ask MDT if you wish to be added to or removed from this list.

The CASINO authors use the `git` revision control system, and it is possible for external developers to have (read-only) access to our `git` repository. `git`-formatted patches may be sent to Mike Towler who will (probably) incorporate them into the master repository.

It would be greatly appreciated if you could forward a copy of any article published using the results of CASINO calculations to us, both for our interest and so that we can add references to the CASINO web pages. If you wish, you may add blog posts to the CASINO public website using your CASINO login ID; this is a good way to publicize your research to the QMC community.

4 Functionality of CASINO

The CASINO program continues to be actively developed and many improvements and revisions are envisaged, but in its present state its capabilities are as follows:

- Methods:
  - Variational Monte Carlo (including optimization of wave functions by minimization of variance, energy, or the mean absolute deviation of local energies from the median (madmin); there is a super-fast variance-minimization method available for optimizing linear Jastrow parameters).
  - Diffusion Monte Carlo (branching DMC and pure DMC).
- Systems:
  - Atoms, molecules, polymers, slabs and solids (the latter three having periodic boundary conditions in one, two, and three dimensions, respectively).
  - 1D/2D/3D electron and electron–hole phases with fluid or crystal wave functions, with arbitrary cell shape/spin polarization/density (including excited-state capability). Can simulate 2D bilayers and 1D biwires.
  - Pseudopotentials or all-electron calculations.
  - Can use particles of any charge or mass.
- Wave functions:
  - Slater-Jastrow wave functions, where the Slater part may consist of multiple determinants of spin orbitals and the Jastrow factor is a sum of selectable, very flexible terms capable of describing complicated electronic correlations. Alternative specially-crafted wave function forms are available for excitonic and positronic systems.
  - Orbitals expanded in Gaussian basis sets (*s*, *sp*, *p*, *d*, *f* or *g* functions centred on atoms or elsewhere) produced by the following programs (using DFT, HF, or various multideterminant methods): CRYSTAL9X/03/06/09/14 [1, 2], GAUSSIAN9X/03/09 [3, 4], GAMESS-US, MOLPRO, CFOUR, PSI4, and TURBOMOLE. Optimizable functions may be added to these.
  - Orbitals expanded in plane waves produced by PWSCF/QUANTUM ESPRESSO [5], ABINIT [6], GP, CASTEP [7], MCEXX, and K207.
  - Orbitals expanded in 'blip' (B-spline) functions, usually generated by post-processing plane-wave orbitals though they can be produced directly by PWSCF/QUANTUM ESPRESSO.
  - Orbitals interpolated from a radial grid for atomic calculations. Optimizable functions may be added to these.
  - Slater-type orbitals produced by, e.g., ADF [8]. Optimizable functions may be added to these.
  - Use of flexible inhomogeneous backflow functions to give highly accurate trial wave functions.
  - Localized orbitals and basis functions for improved scaling with system size. CASINO scales quadratically with system size to evaluate the total energy to a specified error bar (a capability sometimes referred to as 'linear scaling' in the literature).
  - Complex wave functions and twisted boundary conditions can be used to reduce single-particle finite-size errors. Twist averaging can be performed wholly within CASINO for electron(–hole) fluid phases (and with the aid of other codes to regenerate the wave function files it can be done for real systems containing atoms).
- Expectation values:
  - Computation of excitation energies corresponding to the promotion, addition or subtraction of electrons.
  - Computation of various expectation values: density, spin-density, noncollinear spin-density matrix, one-body and two-body density matrices, condensate fraction, reciprocal space and spherical real space pair correlation functions, localization tensor, structure factor, spherically averaged structure factor and ionic populations.
  - Calculation of electron–electron interactions using either Ewald and/or our *model periodic Coulomb* interaction, which is faster and has smaller Coulomb finite-size effects.
  - Corrections to the finite-system kinetic energy from the long-ranged two-body Jastrow factor and to the interaction energy from the structure factor can be calculated for periodic systems.

- Other:
  - Grossman–Mitas DMC-DFT molecular dynamics [9] (requires access to PWSCF/QUANTUM ESPRESSO for the DFT parts of such calculations).
- Design:
  - Parallelized using MPI-2 message-passing library (this can be downgraded to MPI-1 on the very few machines which do not support the later version). MPI libraries are not required for compiling and linking on single-processor machines.
  - Second level of parallelization using OpenMP.
  - Support for accelerators such as graphics processing units (GPUs) using OpenACC.
  - Shared memory support (System V, Posix, and er. . . Blue-Gene/Q 'Posix')
  - Keyword input handled using highly flexible *electronic structure data format.*
  - Self-documenting help system.
- Resources
  - Comprehensive participatory website: `https://vallico.net/casinoqmc/`
  - Helpful online discussion forum: `https://vallico.net/casino-forum/`
  - Online library of Hartree–Fock and Dirac–Hartree–Fock pseudopotentials (some with corresponding Gaussian basis sets)

5 Installation

CASINO was designed to run on machines running a Linux/Unix operating system (this in principle includes Macs, since Mac OSX is based on BSD Unix and has a functioning bash command line). As will be explained, CASINO can also be run on Windows machines (in principle for all modern versions of Windows from XP onwards) if you first install the 'Cygwin' software which emulates a Linux-like environment to compile and run in. The following instructions assume you have your Linux environment correctly setup; Windows users who haven't already done this should first read the note in Sec. 5.4.

CASINO determines what kind of computer you are running on by looking at the value of the Unix environment variable `CASINO_ARCH`, which must be defined in your shell session. This tells CASINO to look into a particular file in a set of 'arch' files—permanently stored in the `arch/data` directory of the CASINO distribution—which contains instructions that the '`make`' shell command and various CASINO utilities can follow both for compiling the code and running calculations on the machine in question.

The various ways of setting this up—both for currently supported architectures/machines and for new unsupported ones—are described below.

The most convenient way to do this is to use the automatic utility—called '`install`'—which can help you to setup and compile CASINO on any given machine. It may be run by typing '`./install`' then pressing Enter in the base directory of the CASINO distribution, after which you should follow the prompts. You can re-run this installer any time you like to amend your configuration.

To give you an idea of what it does, note that the install script will present you with the following options:

```
[c] Compile CASINO for already-configured CASINO_ARCHs
[s] Sort/remove configured CASINO_ARCHs
[a] Auto-detect valid CASINO_ARCHs for this machine
[p] Pick a specific CASINO_ARCHs for this machine
[n] Create a new CASINO_ARCH for this machine interactively
[y] Install CASINO syntax highlighting for various text editors
[i] Install required software using package manager (requires root access)
[r] Restore the CASINO distribution directory to its original state
[q] Save configuration and quit the installer
[x] Quit the installer without saving
```

These options will be discussed in what follows.

Note for sysadmins: CASINO is not currently designed to be installed system-wide by the root user; rather, a separate copy should be installed by the user under his or her home directory. Amongst other reasons, this is because the CASINO distribution contains a huge number of utilities (with large numbers of executable files and scripts which most users of a multi-user machine will not require) along with examples and documentation which the user will wish to access.

5.1 Detailed instructions

5.1.1 Downloading the distribution

Get the CASINO distribution from `https://vallico.net/casinoqmc/download-casino/`.

You can obtain either the latest stable release version, or the 'current beta' (a nightly build containing all the latest modifications).

Most people are 'standard users' whose CASINO distribution does not contain the full documented source code; people who need to read the source with a view to developing new features will require a 'developer' version with the full documented source. For that, go here:

`https://vallico.net/casinoqmc/download-developer/`

5.1.2 Unpacking the distribution

- Change to the directory where you want the distribution to live (this is usually assumed to be your home directory, but it need not be).
- Remove/rename any existing `CASINO` directory.
- Unpack the `CASINO_vxxx.tar.gz` distribution ('`tar xvfz CASINO_vxxx.tar.gz`'). This will result in a new directory called `CASINO` containing the CASINO distribution.
- If you want to maintain different versions of the code, it may be useful to rename the `CASINO` directory to be something like `CASINO_v2.11.378` and set up a symbolic link called `CASINO` which points to it ('`ln -s CASINO_v2.11.378 CASINO`'). There is a CASINO utility *update_src* which will do this for you—just type, e.g., `update_src 2.11.378` in a directory containing a `CASINO_v2.11.378.tar.gz` archive.

5.1.3 Preliminary configuration of the machine

You need to ensure that the machine has all the relevant software installed (Fortran, C, and possibly C++ compilers, an MPI library etc.) For machines administered by other people, this should have been done for you.

Refer to the `CASINO/FAQ` file for notes on the preliminary configuration of machines that you administer yourself. This advice is duplicated online at `https://vallico.net/casinoqmc/faqs/`.

The install script can help you with installing required software using the package manager (choose the [i] option).

5.1.4 Finding or creating the `arch` file

AUTOMATIC PROCEDURE

The install script can largely do this for you. The options which concern finding or creating the arch file are [a], [p], [n] and [s].

First of all, try the 'Auto detect' [a] option. The install script may detect an exact match for the particular machine you're running on (i.e., someone else has already set CASINO up on it)—in which case, after you accept its recommendation, CASINO will simply work. The script can also suggest generic similar machines on which you can base your installation.

Alternatively, if you know which `CASINO_ARCH` you want, you can simply type in its name after selection of the [p] option.

Choosing the [n] option will take you through a guided procedure for creating your own personalized arch file (make sure you have the machine's documentation handy so you can answer the questions the script will ask you). The result of this will be the arch file describing your machine that will be placed in the `CASINO/arch/data` directory (see below); new arch files can be emailed to the CASINO developers for permanent inclusion in the distribution. On extremely complicated machines the arch file produced by install may need to be tweaked by hand.

Multiple alternative configurations are supported. This includes setting up CASINO for use with multiple compilers—each of which will have its own arch file. One may also create set-ups where multiple machines share the installation directory, such as different-architecture queues on the same cluster, or workstations sharing their home directories over a networked file system. For the latter case, run the installer on one machine of each relevant type to set it up.

Once you have all your `CASINO_ARCH`s defined, you may sort them into a preferred order (perhaps to indicate a preferred compiler) or remove them using the [s] option.

Your defined setup can then be permanently saved using the [q] option. If you're running in a bash shell, the list of defined `CASINO_ARCH`s will be stored in a hidden `.bashrc.casino` file in your home directory, which will be sourced from your `.bashrc` every time you log in. The install script also works if you use the `tcsh` or `csh` shells—though without the facility to change `CASINO_ARCH`s described below. Support for other shells is not provided.

The `.bashrc.casino` file defines a '`casinoarch`' function, which will allow you to switch between defined `CASINO_ARCH`s. For example, on the current (Jan 2013) fastest machine in the world—the Cray XK7 'Titan' at Oak Ridge National Laboratory—there are four defined `CASINO_ARCH`s:

- `linuxpc-gcc-pbs-parallel.titan.arch` (397.20 sec.)
- `linuxpc-cray-pbs-parallel.titan.arch` (456.63 sec.)
- `linuxpc-ifort-pbs-parallel.titan.arch` (477.75 sec.)
- `linuxpc-pgf-pbs-parallel.titan.arch` (490.10 sec.)

referring to the four different available compilers (Gnu, Cray, Intel, Portland), and where the time quoted is the time required to run a test DMC run (you can see it's worth paying attention to doing tests to see which compiler produces the fastest executable!). You can switch between these four arch files by typing '`casinoarch gcc`', '`casinoarch cray`', '`casinoarch ifort`', or '`casinoarch pgf`'.

As an example of what using CASINO's automated utilities saves you, note that Titan requires 'modules' to be loaded and unloaded to set up particular programming environments with different compilers. Both 'make' and the CASINO run script 'runqmc' will do this automatically on detecting the current '`CASINO_ARCH`', so there is no need for the user to understand or interact with the module system at all.

More details of all the above procedures are given in the section 'The CASINO_ARCH system' below.

MANUAL PROCEDURE

If you don't wish to use a magic script (though we strongly suggest you do), then you may attempt the following manual installation.

For most machines, a suitable `CASINO_ARCH` data file will already exist in the `/arch/data` directory. If you have installed CASINO before, you likely know which one it is. You may also look through the data files manually to see which one is likely to correspond to your machine. If your machine is similar to, but not identical, to one already set up, then you may borrow one of the data files and suitably modify it (see the instructions below and in Appendix 5).

Assuming, for example, that your `CASINO_ARCH` is '`linuxpc-ifort`', then you need to set this as a permanent environment variable in your shell. If you use, for example, the bash shell then you need to add something like

```
export CASINO_ARCH='linuxpc-ifort'
```

to your `.bashrc` file, and then `source ~/.bashrc`.

5.1.5 Compiling the code

Once you have sorted out the arch file, you may compile the code and the utilities in two different ways.

AUTOMATIC PROCEDURE

Run the install script as before and this time select the [c] option (‘`Compile CASINO for already-configured CASINO_ARCHs`’). The script will respond with a list of defined `CASINO_ARCHs` and the following text (using the Titan machine as an example):

```
 The following CASINO_ARCHs are configured (it is possible that not all of them
 can be compiled from this machine depending on your set-up):

 [1] linuxpc-gcc-pbs-parallel.titan
 [2] linuxpc-cray-pbs-parallel.titan
 [3] linuxpc-ifort-pbs-parallel.titan
 [4] linuxpc-pgf-pbs-parallel.titan

 At the prompt below enter the numbers corresponding to the CASINO_ARCHs you
 would like to compile, separated by spaces.

 You can specify which optional compile-time features to enable appending
 ':<feature>' to each number. Available <features> include:
 - 'Openmp' for building OpenMP support
 - 'Openacc' for building OpenACC support
 - 'Shm' for building the SMP shared-memory facility (of most use for
    calculations with blip or Gaussian basis sets)
 - 'OpenmpShm' for building a version with both features enabled
```

One may thus type ‘`1`’ to compile with the gcc compiler, ‘`2`’ to compile with the Cray compiler, etc. One may compile special versions such as the shared-memory version of CASINO with the gcc compiler by typing ‘`1:Shm`’.

We recommend that this automatic procedure is used, essentially for three reasons:

- One can build multiple CASINO executables with a single typed command such as ‘`1 1:Shm 1:Openmp 1:OpenmpShm 2 3:debug 4:Shm`’.
- The install script will run make in parallel as far as it can over multiple cores (this can of course be done on the command line, but the user may not know how).
- On some obscure Unix machines, the default version of ‘`make`’ will work with a sufficiently different syntax that the CASINO Makefile is not interpreted correctly. The install script will know to run an alternative version of make which we know works, such as GNU’s ‘gmake’ whereas just typing ‘make’ on the command line will not work.

MANUAL PROCEDURE

The alternative manual procedure is to sit in the base CASINO directory and type ‘`make`’.

Different versions of the code may be compiled, e.g., by:

```
make              : standard version with optimizing compiler flags
make debug        : standarc version with debug compiler flags
make Shm          : shared memory version with optimizing compiler flags
make Shm/debug    : shared memory version with debug compiler flags
make Openmp       : OpenMP version with optimizing compiler flags
make Openacc      : OpenACC version with optimizing compiler flags
make Openmp/debug : OpenMP version with debug compiler flags
make OpenmpShm    : OpenMP and Shm version with optimizing compilerflags
etc.
```

Note that the Makefile also supports all-lower-case versions of the above (e.g., make shm, make openmpshm, etc.), even though these are less legible.

The make procedure is fully documented in the preamble of the file `src/Makefile`. More generally, src/Makefile works as follows:

```
Generic form of target:
   <features>/<version>.<action>
where <version> can be:
 - opt  : use full optimization (default).
 - debug: use debugging compiler flags.
 - dev  : use full optimization, but keep object files and binary
          separate from 'opt' version.
 - prof : use profiling compiler flags (if a profiler is available).
<features> can be:
 - NoFeatures: no special features (default).
 - Openmp    : build OpenMP support.
 - Openacc   : build OpenACC support.
 - Shm       : build SHM support.
 - OpenmpShm : build OpenMP and SHM support (currently broken).
and <action> can be:
 - <empty>: perform compilation (default).
 - clean  : remove object files.
 - vclean : remove object files and binaries.

E.g., one can type:
 make
 make opt
 make NoFeatures
 make NoFeatures/opt
 make nofeatures/opt
to the same effect.

For the 'clean' and 'vclean' targets, 'all' can be used to refer
to all features or versions, e.g.:
 make all.clean          # same as 'make NoFeatures/all.vclean'
 make all/dev.vclean
 make Openmp/all.vclean
 make openmp/all.vclean
 make all/all.vclean

 NOTE: the above clean targets only work if sitting in the CASINO/src
 directory. Typing 'make clean' in the CASINO base directory is equivalent to
 'make all/all.clean' (remove all object files for all features and versions)
 and 'make all/all.vclean' (remove all object files and all binaries for all
 features and all versions). Or at least it as after version 2.13.279. Hope
 that's clear.
```

You can also type '`make info`' in the base directory to get a print out of all the compilers and compiler flags that CASINO will use, without invoking any actual compilation.

To run the code you need to add the `$HOME/CASINO/bin_qmc` directory to your shell path, e.g., through (bash):

```
export PATH="$PATH:/YOUR_HOME_DIRECTORY/CASINO/bin_qmc"
```

so that your shell can pick up all the executable CASINO utilities. (The install script will do this automatically).

5.1.6 Running the code

You run the code using the '`runqmc`' script (see Sec. 6.6). This script is able to access all the run time information in the arch file to determine how to run jobs on your machine—even to the extent of loading modules, writing batch scripts, and submitting them for you.

You can run the various utilities just by typing their name.

5.2 The CASINO_ARCH system

In this section, we give further details of how some of these things work.

As already stated, CASINO determines what kind of computer you are running on by looking at the value of the Unix environment variable `CASINO_ARCH`, which must be defined in your shell session. The '`arch/data`' subdirectory of the CASINO distribution contains architecture data files which define system-specific parameters for compiling and running the code. Include files in this directory are named `CASINO_ARCH`.arch, and contain information about which set of parameters to use, including compiler name, compiler flags, library locations, how to submit jobs, etc. `CASINO_ARCH` should be set permanently on your machine, and will be re-used when installing future versions.

While there is no fundamental difference between `CASINO_ARCH`s, we define two conceptual types for convenience, which simply differ in purpose and naming convention:

- 'Generic' `CASINO_ARCH`s are intended to represent a class of systems. Their name is typically of one of these forms:

```
- Single-processor workstations:

  <system>-<compiler> e.g. linuxpc-ifort

- Multi-processor workstations:

  <system>-<compiler>-parallel e.g. linuxpc-ifort-parallel

- Clusters with queueing systems:

  <system>-<compiler>-<queueing-system>-parallel e.g.linuxpc-ifort-pbs-parallel
```

- 'Extended' `CASINO_ARCH`s are intended to represent specific systems, and are usually modifications to existing generic `CASINO_ARCH`s. Their name is of the form:

```
<generic-name>.<specific-system-name> e.g. linuxpc-ifort.berts-weird-computer
```

  The corresponding `.arch` file is typically intended to 'include' its generic counterpart, if it exists, but again this is just a guideline.

See the files in the `CASINO/arch/data` directory for the range of both generic and extended `CASINO_ARCH` names. If you end up generating your own unique `.arch` file then you may send it to Mike Towler (`mdt26 at cantab.net`) who can incorporate it permanently into the distribution.

Very large machines in national computer facilities often have specialized setups and requires Extended `CASINO_ARCHs`. Some examples of current top-of-the-range hardware:

- Titan (Oak Ridge, USA): `linuxpc-<compiler>-pbs-parallel.titan` where `compiler` is `pgf`, `cray`, `ifort` or `gcc`.
- Hector (UK national facility): `linuxpc-⟨compiler⟩-pbs-parallel.hector3` where `⟨compiler⟩` is `pgf`, `path`, `cray` or `gcc`.
- Darwin (Cambridge HPCF facility, UK): `linuxpc-ifort-pbs-parallel.darwin2` (Westmere partition) or `linuxpc-ifort-pbs-parallel.darwin3` (Sandy Bridge partition)
- Intrepid (IBM Blue Gene/P, Argonne, USA): `bluegene-xlf-cobalt-parallel.intrepid`.
- Blue Joule (IBM Blue Gene/Q, Hartree Centre, U.K.) `bluegene-xlf-ll-parallel.bluejoule`

.

The full syntax of `.arch` files is explained in the file `CASINO/arch/README`. This information is duplicated in Appendix 5 of this manual.

5.3 Further installation notes

- The [r] option of the install script will restore the CASINO distribution directory to its original state for compilation purposes. This involves removing all binary executables, links to scripts, and compiler object files, which is achieved practically by removing the following directories: `bin_qmc`, `lib/zlib`, `src/zlib` and `utils/zlib`.

- If you use the `vim`, `emacs`, `gedit` or `nano` text editors then it is possible to configure them to highlight the syntax of CASINO's arch files and the various input files. See the `README` file in `CASINO/data/syntax` or just select the [y] option of the install script to automatically do the configuration.
- For versions of CASINO prior to 2.10, a different setup system was used, involving environment variables `QMC_ARCH` and `QMC_ID` (the latter for customization on specialized machines). These may be retained if you wish to continue to use older versions of the code, but the setup will need to be redone for version 2.10 and later.

5.4 Note for Windows users

CASINO was written for UNIX/Linux systems and is not supported directly under Windows. Nevertheless, it runs well enough under Windows using *Cygwin*, an emulation layer that provides a Linux-like compilation and execution environment for applications. Under Cygwin, CASINO works as if it's running on a Linux workstation using the GCC compiler suite. Although Cygwin is considered an emulation layer, this only concerns system calls. The calculation speed of CASINO is native, and its performance is quite acceptable.

These instructions do not cover the installation and general use of Cygwin; please refer to Cygwin's documentation for that. The instructions were tested under Windows 7 Ultimate 64-bit and Windows 8.1 Enterprise 64-bit, using Cygwin 1.7.28, with all Cygwin packages up-to-date as of March 2014, and again on a Windows 10 virtual machine using Cygwin 3.1.2 (July 2020).

Installation and compilation:

1. Download and install Cygwin from `https://www.cygwin.com` (either the 32-bit or 64-bit version).
2. Install at least the following packages and all their dependencies using Cygwin's setup program (which is also its package manager) in addition to those packages that are installed by default (switch View to 'Category'; in order to mark a package for installation, click on 'Skip' to cycle to the newest version): `gcc-core`, `gcc-g++`, `gcc-fortran`, `libgcc1`, `libgfortranXXX`, `make`, `bc`, `bzip2`, `zlib`, `zlib-devel` and `python3`.
   - Make the version numbers for the packages consistent, in particular choosing the same version for the `libgfortranXXX` package as for `gcc-core`, `gcc-fortran`, etc. At present (July 2020) this requires `XXX` = 4.
3. If you want to run in parallel (which believe me, you do, if your machine has more than 1 CPU core) then you should also install the following to get MPI support: `libopenmpi-devel`, `libopenmpifhYYY` and `openmpi`.
   - Again, ensure that the version numbers `YYY` are consistent.
   - MPI *should* just work on Cygwin straight out of the box...
4. The CASINO manual will fail to compile unless you also install various LaTeX packages. The manual is not really needed (after all you have somehow managed to read this!), but if you care deeply about compiling the manual then please install the `texlive`, `texlive-collection-basic`, `texlive-collection-latex`, `texlive-collection-latexrecommended` and `texlive-collection-plaingeneric` packages.
5. If you want to use CASINO's shared-memory facility (to enable large blip calculations) then you should install the `cygrunsrv` package. After completing the installation of packages, open the Cygwin terminal as administrator (right-click on the icon and select 'Run as administrator'). In the Cygwin terminal, type '`cygserver-config`'. Then open the `/etc/cygserver.conf` file with your favourite text editor, and uncomment and change the `kern.ipc.shm_allow_removed` value to '`yes`'. Finally, start Cygserver as a local service by typing '`net start cygserver`'. It may be necessary to reboot your machine at this point.
6. If necessary, set up your Cygwin environment so that you are using the bash shell (should be the default anyway).

7. Follow the instructions given above for downloading, unpacking, installing and configuring CASINO using the `install` script. The option [p] of `install` allows you to specify the value of `CASINO_ARCH` manually. The correct value is '`cygwin-gcc`' under Cygwin, or '`cygwin-gcc-parallel`' if, as is likely, you want to run CASINO in parallel. The auto-detect option of the `install` script should also be able to figure out the existence of these two `CASINO_ARCH`s all by itself (though because of a Cygwin bug preventing asynchronous operations, the auto-detection will run much more slowly than usual).

8. Use the install script's compilation option to compile CASINO as usual.

A further note from Marcin Szyniszewski: "X Window Server is useful if you want to use XMGRACE (e.g., in GRAPHDMC). I installed `Xming` with default settings (`https://sourceforge.net/projects/xming/`) and then installed the following packages from Cygwin's repository: `xorg-server`, `xinit`, `xterm` and `xeyes` (for testing purposes). Then I appended '`export DISPLAY=:0.0`' to my `.bashrc` file, restarted Cygwin Terminal, and made sure `Xming` was running (X icon in the tray). If everything goes well, running 'xeyes' in the terminal should show eyes following the mouse pointer. Some people use Cygwin/X X Server instead of Xming, but I found it problematic when connecting through ssh."

Further note from Neil Drummond (July 2020): "When running CASINO in parallel, Windows Firewall may ask your permission before proceeding; select both 'private' and 'public' and click on 'allow'. I also found it necessary to create a link using `ln -s /usr/bin/python3 /usr/bin/python` in order to use the utilities written in Python. Finally, if you compile the CASINO distribution with debugging flags then the LOUIS utility may fail to compile due to some issue with the -pg flag; unless you are specifically interested in running LOUIS under Cygwin with debugging flags, this issue can safely be ignored."

An alternative approach for committed Windows users is to install the distribution of Linux that you dislike least in a virtual machine, e.g., using VirtualBox (`https://www.virtualbox.org/`). An alternative lightweight virtualization option is the Windows Subsystem for Linux (WSL): see Microsoft's website for more information. We have verified that CASINO can be installed and run using the Ubuntu App from the Microsoft Store in the WSL on Windows 10. Note, however, that CASINO may be significantly slower when run on a virtual machine. Furthermore, the use of either a Linux virtual machine or the WSL still requires the user to use and maintain their Linux installation. We believe that in most cases Cygwin provides a better solution for running CASINO under Windows.

5.5 Note for Apple Mac users

CASINO can be used straightforwardly under Mac OS X.

- If you are willing to switch to bash rather than zsh, type:

  ```
  chsh -s /bin/bash
  ```

  in a terminal window.

- **Either** download and install Homebrew by following the instructions at `https://brew.sh` **or** download and install MacPorts by following the instructions at `https://www.macports.org/install.php`. The latter package manager has better support for old versions of MacOS.

- To use Homebrew to install the packages needed by CASINO, type:

  ```
  brew install gfortran openmpi grace gnuplot xquartz # git # texlive
  ```

  in a terminal. Notes:

  – If you are installing the developer version of CASINO by cloning CASINO's git repository then obviously you will need git; otherwise you do not need git.

  – If you want to compile the manual locally then you will need to install texlive. Unfortunately the texlive package is massive. There is no need to install it if you are happy to live without the manual; you can just ignore the error messages about there being no pdflatex in PATH when you compile CASINO. You could instead just download the manual as a PDF file from the CASINO website. In any case, if you are reading this, you must have got hold of the manual from somewhere!

- To use MacPorts to install the packages required by CASINO, type:

  ```
  sudo port install gcc13 +gfortran openmpi grace gnuplot # git # texlive
  sudo port select --set gcc mp-gcc13
  sudo port select --set mpi openmpi-mp-fortran
  ```

  in a terminal. See the note above about git and texlive. GCC 13 is appropriate at the time of writing. At this point you may wish to read *War and Peace*; hopefully the package manager will have nearly finished by the time you reach the end.

- To allow the use of OpenMPI you may need to change your firewall settings to allow incoming connections (at least Neil found that was necessary with MacOS 'Big Sur'). To do this, go to the Apple menu / System preferences / Security & Privacy / Firewall. Click on the padlock to make changes, go to Firewall Options, and allow incoming connections.

- Use the `install` script as normal to install CASINO, selecting `macos-gcc-parallel` as your CASINO_ARCH.

- If you do not wish to switch from zsh to bash then a minimalistic (addition to your) ∼`/.zshrc` file is as follows:

  ```
  export CASINO_ARCH=macos-gcc-parallel
  export PATH="$PATH:$HOME/CASINO/bin_qmc/utils/$CASINO_ARCH"
  ```

  (assuming you have installed CASINO in your home directory).

5.6 Discussion forum

An online discussion forum `https://vallico.net/casino-forum/` exists, where users may discuss issues relating to quantum Monte Carlo and the CASINO code. Expert users of the code are known to haunt the forum, and thus it may be useful for soliciting advice if problems are encountered in installing or using CASINO.

6 Introductory user's guide: how to use CASINO

6.1 Getting started

This section gives basic practical details for running QMC calculations with CASINO, and is intended for new users. It assumes you already know something about the theory of VMC, DMC and wave-function optimization. If you don't then please see the standard references for general details about the methods (e.g., Refs. [10], [11], [12], [13], [14], and the Theoretical Background section at the end of this manual). You might also like to have some idea of how CASINO works on parallel machines, a topic discussed in detail in Sec. 40. Following the instructions in this section will not necessarily lead to publication-quality results, but should at least allow you to play with the code and get some feel for how it works.

6.1.1 Trial wave functions

Unless you're interested in electron or electron–hole phases in the absence of an external potential, in which case you can start straight away, the first hurdle to doing research with CASINO is to generate a trial wave function using, for example, a DFT or HF calculation. Multideterminant quantum chemistry methods can also be used. This has to be done using an external program, which must either support CASINO directly, in that it is capable of writing out the wave function in a format that CASINO understands, or it must be supported indirectly by CASINO, in which case a conversion utility should be found under `~/CASINO/utils/wfn_converters/` which can convert the information in its standard to output to CASINO format. Note that 'writing out the wave function' basically means writing out the geometry, the basis set and the coefficients that define the orbitals.

The information defining the trial wave function generated by the external program lives in files whose name depends on the basis set in which the orbitals in the determinantal part of the wave function are expanded. These files are called `gwfn.data` (Gaussians), `pwfn.data` (plane waves), `bwfn.data` (blip functions), `awfn.data` (atomic orbitals given explicitly on a radial grid), `dwfn.data` (molecular orbitals for dimers given explicitly on a radial grid), or `stowfn.data` (Slater-type orbitals). These files will often be referred to generically with the name `xwfn.data`. For the case of blip orbitals, where the wave function file can get very large, you will often see the `bwfn.data` in its unformatted binary form `bwfn.data.bin` which takes up much less disk space (an older binary format `bwfn.data.b1` is also supported). The choice of basis set has been found to depend largely on personal prejudice, though some consideration should be given to issues of computational efficiency.

Gaussian, Slater-type, plane-wave and numerical atomic wave functions are taken directly from the generating code. The plane-wave DFT code PWSCF/QUANTUM ESPRESSO knows about blips, and is capable of internally transforming its plane-wave orbitals and writing out blip wave functions directly (in either binary or formatted forms). With other plane-wave DFT codes, the blip wave function files are generated by post-processing a plane-wave `pwfn.data` file by performing a CASINO calculation with **runtype** set to 'gen_blip'. It is desirable to carry out this transformation because plane waves are the worst possible basis set you could choose for QMC, since every basis function contributes at every point in space. Moreover, blips can be used to make the computer time for a CASINO calculation scale independently of system size (for energy-per-atom properties) or quadratically with system size (for total-energy properties). To achieve this, the delocalized orbitals generated by most DFT/HF programs need to undergo a linear transformation to a localized form, using the `localizer` utility before re-expanding in blips using a 'gen_blip' calculation, though in practice this facility is rarely used.

Generating trial wave functions almost always requires you to have access to one of the codes listed in Sec. 8. If you don't have access to any of these codes and have a specific system in mind, then Mike Towler is known for being able to generate trial wave functions in record time on payment of a suitable fee (just to get you started he likes, in no particular order: bright shiny things, books about romance, poems, authorship on papers for which he hasn't really done any work, ancient books from long ago about explorers with nice binding, cute cuddly toys, especially bears, anything sold by United Nuclear `https://www.unitednuclear.com/`, and money).

If you are using pseudopotentials you must be able to use the same ones in the orbital-generating code and in CASINO: see the note in the next section for details.

6.1.2 Pseudopotentials

CASINO is capable of running all-electron calculations, where core and valence electrons are explicitly included in the simulation, or pseudopotential calculations, where the core electrons are replaced by an effective potential. The latter approach is normally advantageous since the computer time required for all-electron calculations scales rather badly with atomic number Z; this scaling is improved by using pseudopotentials.

CASINO reads pseudopotential data from a file called `xx_pp.data`, where `xx` is the chemical symbol of the element in lower-case letters. The file contains a logarithmic radial grid and the values of the different angular momentum components of the pseudopotential at each grid point.

On the rare occasions when you might want to use two or more different pseudopotentials for atoms with the same atomic number (say in a surface, and in an atom or molecule absorbed on that surface), then you may use additional pseudopotentials renamed as, e.g., `xx2_pp.data`. Different types of pseudoatom are flagged in `xwfn.data` by adding multiples of 1000 to the original atomic number, e.g., atomic numbers 12, 1012 and 2012 correspond to atoms using pseudopotentials `mg_pp.data`, `mg2_pp.data` and `mg3_pp.data`, etc.

A library of pseudopotentials suitable for use with CASINO is available at:

```
https://vallico.net/casinoqmc/pplib/
```

An important point is that exactly the same pseudopotential should be used in the DFT/HF calculation that generates the trial wave function and in CASINO. Other programs do not in general understand the CASINO pseudopotential format, and so the information must somehow be transformed so that they do.

For programs using Gaussian basis sets such as GAUSSIAN9X/0X and CRYSTAL, the pseudopotential must be re-expanded in Gaussian functions multiplied by powers of r. If you are using the Cambridge pseudopotentials, such expansions (in formats suitable for these two programs) are included in the on-line library. There is a CASINO utility—`ptm`—which can manipulate pseudopotential files on grids and their Gaussian expansions in various useful ways.

As for plane-wave programs, CASTEP understands the CASINO grid format and can read such files directly. Other plane-wave programs require conversion utilities, which are included in the `~/CASINO/utils/pseudo_converters/` directory.

For atomic calculations on radial grids, `awfn.data` files generated with the Cambridge pseudopotentials are available in the on-line library.

6.1.3 The `input` file

Having prepared a trial wave-function file and (perhaps) a pseudopotential file, you need to tell CASINO exactly what to do with them. CASINO takes its instructions from a file called `input`, which contains a flexible list of keywords. These control the behaviour of the calculation, switch on and off various options and so on. Take a moment to examine the various `input` files in `~/CASINO/examples/` to get a feel for what they look like.

A complete list of input keywords, together with their definitions, is given in Sec. 7.3. Further details, including default values, may be found by using the `casinohelp` utility. This tends to be more up to date than the manual, since it interrogates CASINO directly. Type *casinohelp all* to get a list of all keywords that CASINO knows about, or *casinohelp keyword* for detailed help on a particular keyword. Type *casinohelp search text* to search for the string 'text' in all the keyword descriptions.

Although there are many keywords, the beginner can play around by changing only a few of them. Here's a (very) rough guide. Advice on good values to use will be given in the subsequent sections explaining how to do VMC, DMC and optimization calculations.

- General (system-dependent) keywords:

  NEU, NED Number of electrons of up and down spin;

  PERIODIC Whether the system is periodic or not;

  NPCELL Array of primitive cells making up the simulation cell (not required for finite systems).

- Other vital keywords:

  RUNTYPE Type of QMC calculation: 'vmc', 'vmc_dmc', 'vmc_opt', etc.;

  ATOM_BASIS_TYPE The type of orbitals to be used: 'plane-wave', 'gaussian', 'slater-type', 'blip', 'numerical', 'dimer', 'none' (for HEGs, etc.) and various special wave function types such as 'nonint_he', 'h2', and 'h3plus';

- Important VMC keywords:

  VMC_EQUIL_NSTEP Number of equilibration steps;

  VMC_NSTEP Number of VMC energy-evaluation steps;

  VMC_DECORR_PERIOD Number of steps between VMC energy-evaluation moves.

  VMC_NCONFIG_WRITE Number of VMC configurations stored for later use in DMC or optimization;

  DTVMC VMC time step (size of trial steps in random walk);

- Important optimization keywords:

  OPT_CYCLES Number of optimization+VMC cycles to perform.

  OPT_METHOD Optimization method to use: variance minimization ('varmin'), energy minimization ('emin'), etc.

- Important DMC keywords:

  DMC_TARGET_WEIGHT Target number of configurations in DMC;

  DMC_EQUIL_NSTEP Number of DMC steps in equilibration;

  DMC_STATS_NSTEP Number of DMC steps in statistics accumulation;

  DTDMC DMC time step.

6.1.4 Correlation parameter file

If you run a CASINO VMC calculation using a trial wave function consisting of only a single determinant of orbitals (referred to as an 'HFVMC' calculation), then the result will be the HF energy. If the orbitals were generated using a HF calculation, then the HFVMC energy should agree with the HF energy from the generating code. This is a good check that everything is being done correctly. Obviously, if the determinant is made up of Kohn–Sham orbitals from a DFT calculation then the total energies will not agree, because the DFT program adds an XC energy deduced from the self-consistent charge density; however, the kinetic energies should still be in agreement.

The full Slater-Jastrow trial function normally used in QMC requires the determinantal part of the wave function stored in `xwfn.data` to be multiplied by a separate 'Jastrow factor', which defines the functional form of explicit interparticle correlations. In a typical VMC calculation one might recover 60–80% of the correlation energy using such a wave function. This is not really enough to be generally useful, and in practice the main use of VMC is to prepare an accurate trial wave function to be given as input to a DMC calculation. The DMC energy does not in principle depend on the Jastrow factor, since the Jastrow factor is positive definite and the DMC energy depends only on the nodal surface (the set of points in configuration space where the many-electron wave function is zero). However, it makes the calculation vastly more efficient, and in general Jastrow factors should always be used.

The Jastrow factor is stored in a file called `correlation.data`. Again, you should look at the examples to see what these look like. The various parameters in the files are defined in Sec. 7.4.2. The adjustable parameters in the file must be optimized for a specific system, and this is the purpose of the variance-minimization procedure.

The `correlation.data` files may also contain other optimizable parameters not contained in the Jastrow factor (for example, 'backflow' parameters, or the coefficients in a determinant expansion) but you don't need to know about these yet.

6.1.5 The MPC data file

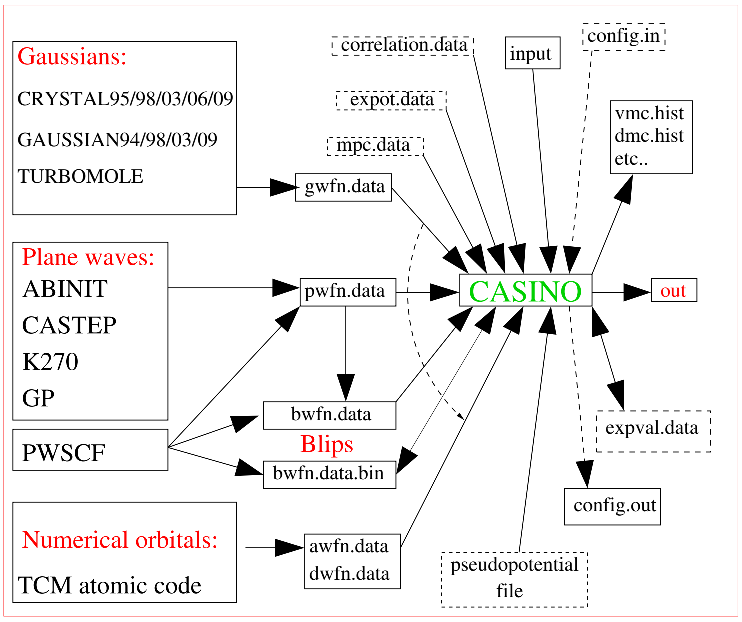
If, for a two- or three-dimensionally periodic system, you want to use the *model periodic Coulomb* (MPC) interaction to calculate the electron–electron energies instead of (or as well as) the standard Ewald interaction, then you need to generate an extra file for the given geometry before you start doing VMC/DMC calculations. This is called `mpc.data` and contains the Fourier transform of the $1/r$ interaction and the Fourier transform of the charge density. The `mpc.data` file should be prepared for a given trial wave function and geometry by setting the input keyword **runtype** to 'gen_mpc' and typing *runqmc*. Note that generation of the `mpc.data` file requires **complex_wf**=`T`, but this is not required for the subsequent use of the `mpc.data` file.

One might consider using the MPC interaction because CASINO can evaluate it much more quickly than the Ewald interaction, and also because it gives rise to smaller finite-size effects[1]. More details are given in Sec. 19.4.4.

6.1.6 Summary of input files required by CASINO

The input files (and some of the output files) required by CASINO are summarized in the figure below:

[1]The difference between the energy per atom obtained in a finite simulation cell and the energy per atom for an infinite crystal is smaller when the MPC interaction is used instead of the Ewald interaction.



The dotted-line boxes indicate optional files, and the curved dotted line indicates that one of the wave function files defined by the arrows within the sweep of the curve must be supplied (unless you are doing a model system such as the homogeneous electron gas which does not require a trial wave function generated by an external code).

A complete list of the input and output files used by CASINO, together with detailed information about their format, is given in Sec. 7.

6.2 How to do a VMC calculation

Let's begin by calculating the HF energy of a hydrogen atom. Change directory to `~/CASINO/examples/atom/hydrogen/`. You will see a `gwfn.data` file generated by a GAUSSIAN94 calculation—see the bottom of `gwfn.data` for the GAUSSIAN output—and a CASINO `input` file (note that most CASINO wave function files are supplied in compressed `.gz` form, so you may need to *gunzip* them). No pseudopotential file is supplied, so CASINO will assume you wish to do an all-electron calculation. No `correlation.data` file is supplied, because the correlation energy in a one-electron atom (zero!) is not difficult to calculate without one.

Look in the `input` file. You will see that **neu** and **ned** have been given the correct values (one spin-up electron present), **periodic** is `F`, and, as this is a finite system, the **npcell** block is not required. It is not necessary to equilibrate the electron distribution since there is only one electron, but **vmc_equil_nstep** is set to 500 to remind you that this should normally be done. The **vmc_nstep** parameter is set to 100000. Note also that the **runtype** parameter is set to 'vmc', implying that we are going to perform a VMC calculation. The **atom_basis_type** parameter is set to 'gaussian', which means that a Gaussian basis set is used to expand the orbitals in the trial wave function.

Type *runqmc*. Three files, `out`, `vmc.hist` and `config.out`, should be produced. First type *envmc*. This is a quick way to pull the VMC result out of the output file. It says:

```
ENVMC v0.60: Script to extract VMC energies from CASINO output files.
Usage: envmc [-kei] [-ti] [-fisq] [-pe] [-vee] [-vei] [-vnl] [-nc]
          [-vr] [-rel] [-ct] [-nf <no-figures>] [files]

File: ./out
      (energies in au, variances in au)
 E    = -0.49999(4) ; var  = 0.000083(4)
 Total CASINO CPU time  ::: 3.3700 seconds
```

If any warnings are present in the output file, then *envmc* will print them out, but there shouldn't be in this case. Note that, because of the stochastic nature of the method, the energy has an associated error bar (users of DFT, etc., may find this disconcerting!). The true HF energy of the hydrogen atom is exactly -0.5 a.u., so we have seemingly correctly performed our first VMC calculation!

A point about parallel machines: if you try to run this calculation on your machine then you may get a slightly different answer. The numbers above were obtained on a single processor core; if your computer has more than one core (these days most machines do, of course) and your `CASINO_ARCH` is a parallel one, then the *runqmc* script will automatically detect the number of cores on your machine and run CASINO in parallel using that number. If you wish to reproduce the single-core result, you may use `runqmc -p 1`.

For example, my (=MDT) personal machine has dual hex-core processors (i.e., 12 cores in total), though if I interrogate it using a command like

```
cat /proc/cpuinfo 2> /dev/null | grep -cE "^processor"
```

it thinks it has 24. This is because these are Intel chips with hyperthreading enabled, i.e., for each processor core that is physically present, the operating system addresses two virtual processors, and shares the workload between them when possible.

So if I execute `runqmc` then I get the following 24-core result:

```
 ENVMC v0.60: Script to extract VMC energies from CASINO output files.
Usage: envmc [-kei] [-ti] [-fisq] [-pe] [-vee] [-vei] [-vnl] [-nc]
          [-vr] [-rel] [-ct] [-nf <no-figures>] [files]

File: ./out
      (energies in au, variances in au)
 E    = -0.50000(4) ; var  = 0.000088(7)
 Total CASINO CPU time  ::: 0.4309 seconds
```

Note the behaviour here. The second run is a lot faster but gives roughly the same error bar as the first one. This is because **vmc_nstep** gives the *total* number of configuration space samples—in this case 100,000—not (as was the case with early versions of CASINO using the old **nmove** keyword) the number of samples done by each core. Giving the total number is better since it means that the input files are then independent of the number of MPI processes on which the job is run. With 24 cores, the 100000 samples to be taken will be divided by 24 (to give 4166.667). Because this is not an integer, this number is rounded up to 4167, and so $4167 \times 124 = 100008$ samples will actually be done in total. Because each processor is doing less work, the calculation as a whole is much faster (not exactly 24 times faster because the setup is a significant fraction of such a short calculation). The answer will be different, however, because 24 short random walks initialized with different random number seeds sample the space differently to 1 long random walk.

Next, look in the `out` file. We see a complete report of the calculation and at the end we see the total energy and its components. Just before the total energy, we see the 'acceptance ratio' which should be around 50% (the acceptance ratio depends on the VMC time step, and we have set **opt_dtvmc** to 1 to automagically adjust the time step so that the acceptance ratio of 50% is achieved). [2].

Next, look at the file `vmc.hist` (see Sec. 7.12). This contains mainly energy data produced at each step of the random walk (you can average over successive steps using the **vmc_ave_period** keyword to prevent this file getting too large). The `vmc.hist` file contains all the raw VMC data, which should be subjected to a statistical analysis. Note that the data in here is averaged over MPI processes, so in the example above `vmc.hist` will contain 4167 lines, and each line contains energy data averaged over 24 cores.

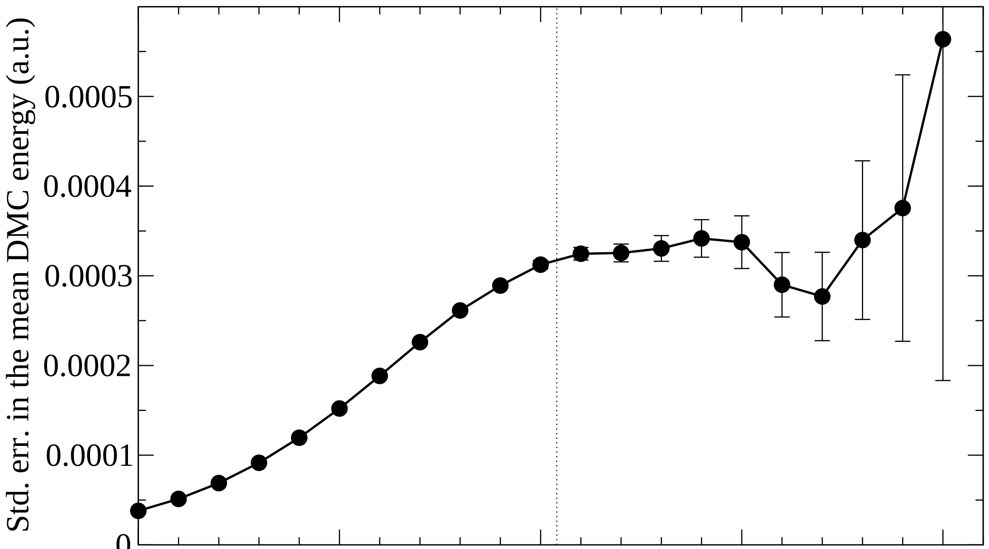
The error bar in VMC and DMC energies is affected by serial correlation (caused by the sampling points not being truly independent of each other). There are two ways of removing this—which should more or less agree with each other—and CASINO uses both when calculating total energies and their components. The first way is the *correlation-time method*—see Sec. 24.2—the corrected error bar calculated using this method is printed at the end of the output of VMC calculations (including the data from earlier runs if the current run is restarted). The second way is the *reblocking method* (see Sec. 24.1). This is done by CASINO 'on-the-fly' and the reblocked error bar is printed at the end of both VMC and DMC output files (again, this includes data from previous calculations in the case of 'continued' runs). All these numbers may also be extracted from VMC output files using the `envmc`

[2]If the time step is small then the electrons only attempt to move very short distances at each step of the random walk. Although the acceptance ratio is therefore high, the electrons are likely to bumble about in one corner of the configuration space, and serial correlation is likely to be significant. If the time step is large then the electrons attempt very large jumps—which are likely to be to regions of lower probability—and are thus likely to be rejected by the Metropolis algorithm. Hence the electrons don't move very much, and the serial correlation is again large. This is the reason why an acceptance ratio of 50% usually implies nearly optimal sampling efficiency.

utility. Both reblocking and the correlation time correction may also be performed after a calculation by using the `reblock` utility to analyse the data in `vmc.hist` or the equivalent DMC file `dmc.hist`. Although this allows a more detailed analysis of serial correlation, it is not necessary under normal circumstances to post-process the data since CASINO and the `reblock` utility should give the same answers, and the additional detail is of limited interest. One might use the utility when visualizing reblocked energy data, when doing explicit statistical studies of QMC, when calculating error bars on other expectation values stored in the `vmc.hist`/`dmc.hist` files (dipole moment, forces, ...), or just when learning how the reblocking method works.

So let's see how the reblocking method works. Type *reblock* in the directory containing the `vmc.hist` file. It will first ask you what units you want. Then it will print out a reblocking analysis (see Sec. 24.1): it will calculate an estimate of the standard error in the mean energy as a function of the length of blocks into which the data are gathered. The error bar will generally be too small for low values of the block length, and then should go up to a roughly constant value for larger block lengths. For very large block lengths the estimated error bar will oscillate as the error bar on the error bar becomes large. The constant value in the middle is the accurate error bar you want. Note that, in general, VMC calculations will reach this plateau more quickly than DMC calculations since they use a much larger time step and there is therefore less serial correlation in the data. In fact, for many VMC calculations, the standard error in the mean energy reaches its plateau at a block length of 1, so that one does not see the standard error rising significantly as the block length is increased.

A file called `reblock.plot` is normally produced as part of the output of the `reblock` utility. If you have the program XMGRACE (or GNUPLOT or python/matplotlib) set up on your system, then you can visualize the results of `reblock` by typing *plot_reblock*—you should be able to see a 'plateau' in the standard-error-versus-block-length curve, like the one shown in the figure below. (The block length is given by 2 raised to the power of the 'reblocking transformation number'.) In the example shown below, the standard error in the mean energy is 0.00035 a.u. Note that XMGRACE is a very useful thing to have, and it is used by various CASINO utilities. If you don't have it set up on your system then you can download it for free from the following website: `https://plasma-gate.weizmann.ac.il/Grace/` (it is also present in most modern Linux repositories, though not generally installed by default).



The file `config.out` contains the final positions of the electrons and the current state of the random number generator so that the VMC run can be continued if desired. To do this, set the input keyword **newrun** to `F` and rename the `config.out` file to `config.in` (in fact the `runqmc` script will normally ask if it can do this for you), then run the calculation for another **vmc_nstep** steps. All the extra data will be put onto the end of `vmc.hist` and the corrected error bar obtained from the reblocking analysis will be smaller (in principle the error bar can be made as small as desired). As I already mentioned, you need to use the *reblock* utility for continued runs since the on-the-fly reblocking will no longer work (though the correlation time correction does).

Finally, go and find a system with more than one electron and a Jastrow factor, such as the beryllium dimer in `~/CASINO/examples/molecule/be2/`. Experiment with switching **use_jastrow** between `F`

(HFVMC) and T (VMC with a Jastrow factor), and see the energy- and error-bar-lowering effect of the Jastrow factor (results shown below for 12 cores).

Without Jastrow:

```
 E    = -29.107(8) ; var  = 6.6(6)
 Total CASINO CPU time  ::: 4.0744 seconds
```

With Jastrow:

```
 E    = -29.215(2) ; var  = 0.303(2)
 Total CASINO CPU time  ::: 6.0221 seconds
```

After any CASINO calculation, you can delete all the output that CASINO produces and restore the directory to the state it was in at the start of the calculation by typing *clearup*.

6.3 Wave function optimization

The values for the parameters in the Jastrow factor used in the last part of the previous section (and other adjustable parameters such as the coefficients in a multideterminant expansion, or backflow parameters) generally need to be optimized so that we have best possible trial wave function of the given functional form. There are various methods for doing this, including minimization of the variance ('varmin'), minimization of the energy ('emin'), and minimization of the mean absolute deviation of the local energies from the median ('madmin'). Wave-function optimization is probably the most difficult part of QMC for beginners, and perseverance will help. The user might first like to take a look at Sec. 25.1, which contains a detailed summary of the theory and best practice. Here we simply summarize.

6.3.1 How to do a variance-minimization calculation

Recall that if the trial wave function were exact, then any arrangement of the electrons in configuration space would have the same value of the local energy—and thus zero variance[3]. For non-exact wave functions, this would not be the case, and the variance would be a positive number. Adjusting the shape of the non-exact wave function in order to minimize the sample variance is therefore a good idea. How do we do this? We need a set of configurations[4] distributed according to the square of the trial wave function, and the first part of a variance minimization calculation is to generate such a set. CASINO will automatically perform VMC configuration-generation runs before the optimization if keyword **runtype** is set to 'vmc_opt'. Keyword **vmc_nconfig_write** must be set to the number of configurations to generate (note again that on parallel machines this is the *total* number, not the number per MPI process)[5]. The configurations generated are passed to the variance-minimization stage, where CASINO adjusts the value of the wave-function parameters in such a way that the variance of the local energies of the previously generated configurations is minimized.

At this stage the configurations are distributed according to the original, unoptimized wave function. Thus it is often a good idea to use the new, optimized wave function to generate a new set of configurations, and then perform the minimization again. This iterated procedure can be carried on as long as desired, although there is generally only a significant change in the variational energy over the first few cycles. The number of cycles is controlled by the keyword **opt_cycles**.

The two main things to worry about are 'how many configurations to use?' and 'what parameters should be included in the Jastrow factor?'. Good advice about this is given in Sec. 25.1. Basically you should use more than 10,000 configurations in general, bearing in mind that increasing the number of configurations past a given value will not improve the wave function but will be more costly. If you have difficulties optimizing a wave function then the first things to try are (i) increasing the number of configurations used in the optimization, (ii) reducing the number of optimizable parameters, i.e., simplifying the trial wave function and (iii) reducing cutoff lengths in the Jastrow factor. On the other

[3]The local energy is given by $\Psi^{-1}(\mathbf{R})\hat{H}(\mathbf{R})\Psi(\mathbf{R})$, where $\hat{H}(\mathbf{R})$ is the Hamiltonian operator, $\Psi(\mathbf{R})$ is the trial wave function, and $\mathbf{R}$ are the electron positions.

[4]Each configuration $\mathbf{R}$ is a set of coordinates for all the electrons in the system, $\{\mathbf{r}_i\}_{i=1...N}$.

[5]Clearly, **vmc_nstep** must be greater than or equal to **vmc_nconfig_write**. You might want it **vmc_nstep** to be a few times larger than **vmc_nconfig_write** to get an acceptable error bar on the energy; this is useful for judging the success of an optimization at each stage

hand, *if optimization is going well* then you want your wave function to be as flexible as possible (more variational parameters and longer cutoff lengths).

The parameters in the Jastrow factor (see Sec. 7.4.2) are of multiple types. In systems containing atoms the expansion coefficients in the U term, the χ term and the F term together with their associated cutoffs are the ones normally varied (together with the P term in periodic systems).

So let's play. For a particular example, see `~/CASINO/examples/electron_phases/3D_fluid/`. This is a homogeneous electron gas (HEG) calculation with density parameter $r_s = 10$ and 54 electrons per cell. Make a backup copy of the Jastrow factor (e.g., `cp correlation.data correlation.data_backup` or whatever). Then, in the `correlation.data` file, find the two sections labelled 'Parameter values' and delete all the parameter lines (a first block of sixteen for the U term and a second block of seven for the P term). You have now created a 'blank' Jastrow file.

In *correlation.data* you can state which parameters should be held fixed, and which should be optimized. In this first example we don't want to optimize the nonlinear cutoff parameter, because it can take a long time to optimize. If we did want to do this then we could go to the point after the line where it says `Cutoff ; Optimizable (0=NO; 1=YES)` and set the 'Optimizable' flag to 1. If you want, you can change the number of parameters by changing the *Expansion order* for the U term. To use more parameters in a P term, you should use the MAKE_P_STARS utility.

Now edit the `input` file. First of all, set **runtype** to `vmc_opt`. Let's use 3200 configurations (**vmc_nconfig_write** = 3200). Note that the total number of VMC moves **vmc_nstep** needs to be at least as large as the number of configurations used in the optimisation. Let's set **vmc_nstep** = 6400. Make these changes now. Also, let's choose **opt_cycles** to be 4, meaning 'do four cycles of configuration generation and optimization'. Set the **opt_method** to `varmin_linjas` and make sure **opt_jastrow** is set to `T`.

Type *runqmc*. CASINO's output will be placed in the `out` file, as before. In addition, CASINO will create files called `correlation.out.1`, `correlation.out.2`, etc. These files contain the optimized Jastrow factor from each cycle of the optimization process. When CASINO has finished, type *envmc*. The output should look something like this (I ran the calculation on 4 cores):

```
ENVMC v0.80: Script to extract VMC energies from CASINO output files.
Usage: envmc [-kei] [-ti] [-fisq] [-pe] [-vee] [-vei] [-vnl] [-nc]
          [-vr] [-rel] [-ct] [-nf <no. of figures in error bars>] [files]

File: ./out
Block averages recomputed for current run.
Corrected for serial correlation using correlation time method.
Energies in au/particle, sample variances in au.

VMC #1: E     = -0.0386(1) ; var  = 0.19(1) (correlation.out.0)
VMC #2: E     = -0.05380(1) ; var  = 0.00280(9) (correlation.out.1)
VMC #3: E     = -0.05390(1) ; var  = 0.00232(6) (correlation.out.2)
VMC #4: E     = -0.05393(1) ; var  = 0.00233(8) (correlation.out.3)
VMC #5: E     = -0.05393(2) ; var  = 0.0024(2) (correlation.out.4)
       Total CASINO CPU time  ::: 438.1300 seconds
```

We see that optimizing the parameters in this way lowers the variance and total energy significantly.

You should use the output of `envmc` to choose which `correlation.out` file you want to use in subsequent calculations, e.g., for DMC or for further optimization. In general, one should choose the `correlation.out` file that gives the lowest variational energy. Therefore, in the example above, `correlation.out.4` should be copied to `correlation.data` for use in subsequent calculations.

To get an even better wave function one could try additional things, such as optimizing the cutoffs, or using more parameters, or having different functional forms between different spin types to optimize the Jastrow still further. Feel free to try this. If you wish to optimize parameters that do not appear linearly in the Jastrow exponent J, such as the cutoff lengths, then you cannot use `varmin_linjas`. You can instead use 'conventional' variance minimization (**opt_method** = `varmin`) or energy minimization (**opt_method** = `emin`).

For this example we have only used the electron–electron terms U and P in the Jastrow factor. If we were optimizing a wave function for a real system with atoms then we would also include atom-centred electron–nucleus χ terms (one for each type of atom), and (usually) electron–electron–nucleus F terms (again, one for each atom type). A three-body H term is also available, but is not widely used in studies of real systems, because the improvement to the wave function does not justify the additional computational expense. It is more useful in studies of electron gases and, especially, in systems where there is an attractive interaction beteen particles. The plane-wave term P is, of course, only applicable to periodic systems. Finally, cylindrical one-body χ_{cyl} and two-body terms U_{cyl} are available.

If you are only interested in optimizing linear parameters in CASINO's Jastrow factor (i.e., all Jastrow parameters except cutoffs and parameters in the three-body terms), then the 'varmin-linjas' method (see Sec. 25.2) is usually considerably faster than other optimization methods. However, in general it is a good idea to use energy minimization (**opt_methods** = `emin`) as a final stage of optimisation, as described next.

6.3.2 How to do an energy-minimization calculation

CASINO also includes a different optimization method, in which the variational energy is minimized. Full details of the method and its usage are given in Sec. 25.3.

Energy minimization is done in a very similar manner to standard variance minimization, and is selected by setting **opt_method** to 'emin'. As before, the process consists of cycles, each comprising a VMC configuration-generation stage followed by an optimization stage. The number of configurations that must be generated per cycle (**vmc_nconfig_write**) should usually be such that the statistical error bar is smaller than the expected lowering of the VMC energy.

There are some differences in the capabilities and behaviour of energy and variance minimization. Whereas variance minimization typically achieves all of its improvement to the wave function in one or two cycles, energy minimization will often require up to ten cycles to converge. The exception to this is that when optimizing only determinant coefficients, energy minimization should converge in at most two cycles (see Sec. 25.3 for an explanation of this). Energy-minimization cycles are usually faster than variance-minimization cycles, so that the overall time to convergence is similar for both methods. Energy minimization is also more sensitive than variance minimization to the presence of optimizable parameters which have very little effect on the wave function (it is often best to avoid including such parameters). More importantly, energy minimization has some difficulty in optimizing cutoff parameters (in the Jastrow, backflow, or orbital functions). Lastly, if the Jastrow u term is present and being optimized, starting from zeroed parameters, it is possible for the energy to increase after the first cycle of energy minimization. Subsequent cycles should lower the energy as usual. This behaviour is explained in Sec. 25.3.

6.3.3 How to do a 'madmin' calculation

'Madmin' is a variant of variance minimization where a measure of the spread of local energies other than the variance is minimized, one that is less sensitive to outliers (e.g., divergent local energies). This measure is the *mean absolute deviation (MAD) from the median energy,*

$$\mathrm{MAD} = \frac{1}{N_{\mathrm{C}}} \sum_{\mathbf{R}} \left| E_{\mathrm{L}}^{\{\alpha\}}(\mathbf{R}) - \bar{E}_m \right| , \tag{1}$$

where $\bar{E}_m$ is the *median* of the set of local energies $\{E_{\mathrm{L}}\}$. This actually works very well, and minimizing the MAD generally gives a lower energy than minimizing the variance. Having a 'robust' measure of the spread turns out to be important when e.g., optimizing parameters that affect the nodal surface where $\Psi = 0$ (such as multideterminant expansion coefficients, parameters in orbitals and backflow functions). Optimization of such parameters is difficult because the local energy diverges where the wave function is zero, and so the unreweighted variance diverges whenever the nodal surface moves through a configuration.

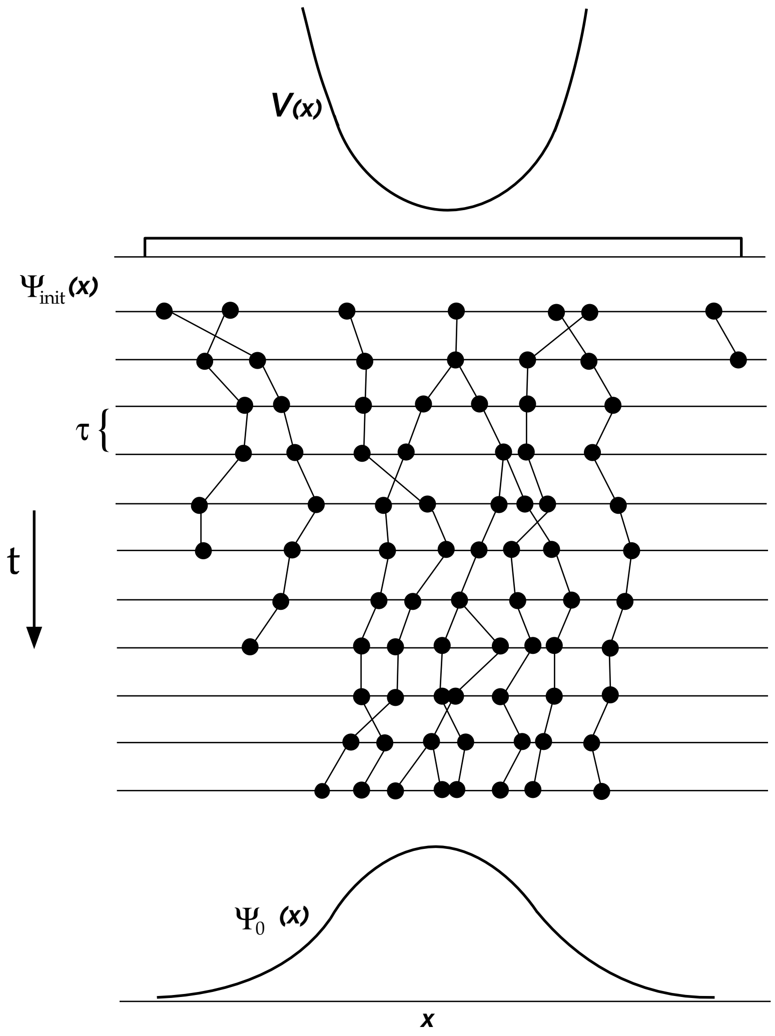
To use this method, you do essentially the same procedure as in a varmin calculation but set **opt_method** = 'madmin'.

6.4 How to do a DMC calculation

DMC calculations are the main point of doing QMC. They are generally extremely accurate—comparable to or better than the best quantum chemistry correlated wave-function techniques—and yet remain applicable to very large systems. However, they require an accurate trial wave function in order to be efficient. This is normally taken to be a Slater-Jastrow wave function with the parameters in the Jastrow factor optimized by one or more of the above minimization procedures.

So we begin by assuming we have an `input` file, an `xwfn.data` file containing the determinantal part of the wave function and an *optimized* `correlation.data` file containing the Jastrow factor. A DMC calculation then consists of three basic steps: (i) VMC configuration generation; (ii) DMC equilibration; and (iii) DMC statistics accumulation.

The wave function in DMC is not represented analytically, but by the time-dependent distribution of a set of configurations (or 'walkers'). The shape of the many-electron wave function in configuration space is built up by moving, killing or duplicating individual walkers according to certain rules, and thus the population of walkers fluctuates.



The first step in DMC is to generate these configurations in their initial distribution (i.e., according to the VMC trial wave function). This is done through a VMC calculation in exactly the same way as we generated configurations for variance minimization in the previous section. Therefore one should again set **vmc_nconfig_write** to be the desired number of configurations, and **vmc_nstep** to be $\geq$ **vmc_nconfig_write**. (Setting the decorrelation period **vmc_decorr_period** to a higher value than in VMC is less important than in variance minimization, since the correlations will disappear as the DMC calculation evolves).

Note—and this is a common point of confusion—that the **runtype** flag should generally be set to 'vmc_dmc' at the start of a DMC calculation, not, as might be thought, to 'dmc' [6].

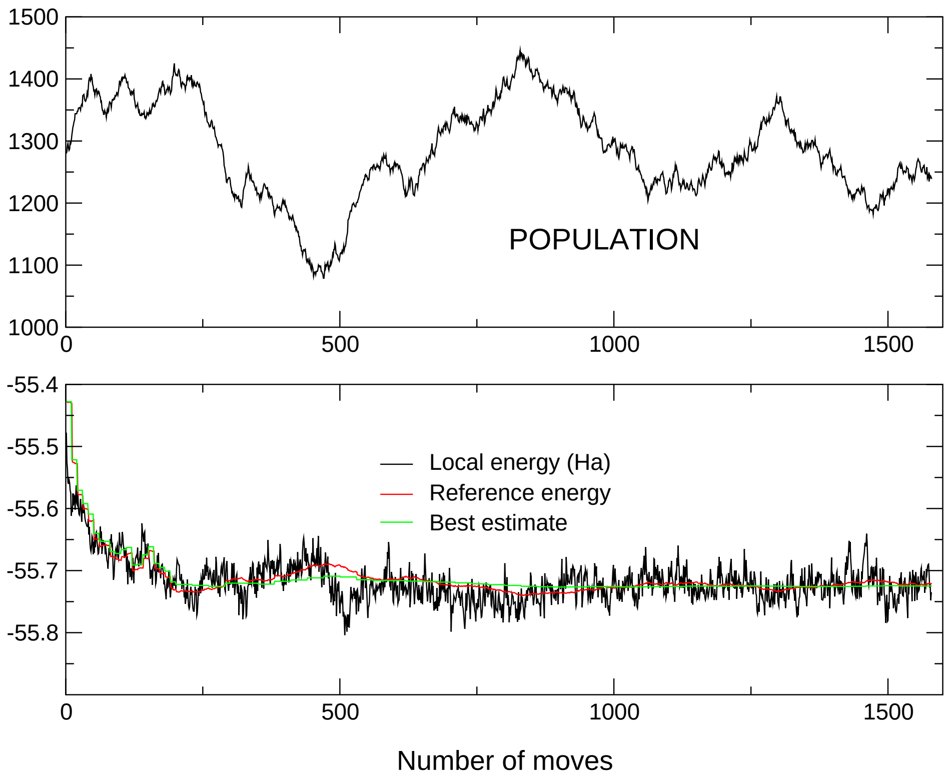
[6]Sometimes one does wish to do DMC only (by using already generated VMC configurations, or by continuing an existing DMC run) in which case one should set **runtype** to either 'dmc_equil' (perform DMC equilibration), 'dmc_stats' (perform DMC statistics DMC equilibration), or 'dmc_dmc' (perform DMC equilibration, then statistics accumulation). In earlier versions of the code, we used **runtype** = 'dmc' with the now-deprecated keyword **iaccumulate** = `T` or `F` to indicate whether stats accumulation was activated or not. This usage is now deprecated and, unless **iaccumulate** is specifically defined in input, **runtype** = 'dmc' is just a synonym for 'dmc_dmc'. If you don't know what DMC equilibration or statistics accumulation are, read on.

An appropriate number of configurations to use for DMC might be 1000 or more. You can sometimes get away with using a few hundred for small systems. Notice that parallelization is automatic—configurations will be distributed among the MPI processes.

Following configuration generation the distribution of walkers is allowed to propagate in imaginary time (see e.g. Ref. [11]) according to the DMC rules. During a certain period of equilibration, the distribution will change until the walkers are distributed according to the ground-state wave function of the system, subject to the constraint that the wave function has the same nodes as the trial function that we started with. This part of the process is called 'DMC equilibration'. The best estimate of the energy will fall from the initial VMC value to around the correct ground-state energy during this process. After equilibration, the best estimate of the energy will be roughly correct, and we now propagate for a long period of imaginary time in order to accumulate enough energy data to estimate the DMC energy with a sufficiently low error bar. This is the 'statistics-accumulation' part [7].

As usual, the basic output of CASINO is sent to a file called `out`—this includes the final total energy and its components with reblocked error bars. The other important output file is `dmc.hist`, which contains the energy and other data as a function of move number (see Sec. 7.12 for precise definitions of what it contains). The data in this file can be analysed in detail using the `reblock` utility, as was done above for the data in the `vmc.hist` file (there is no equivalent of the `envmc` utility for plucking the most important numbers out of the `out` file—as this is only really useful in wave function optimization). In most circumstances it is not generally necessary to use the utility as the reblocked error bars are computed on the fly. As a precaution against runs being prematurely ended—due to power cuts, time limits, etc.—a file `dmc.status` is written and overwritten at the end of each DMC block. At any time during the run, this file contains the complete statistical evaluation as it would be written if the run had finished here. The `out` file only contains this data once at the end, if the run goes to completion (at which point the `dmc.status` file is deleted).

The progress of a DMC simulation is easily visualized by means of the utility `graphdmc`, which reads the `dmc.hist` file and calls the plotting program XMGRACE to show you the resulting pretty picture. Typing *graphdmc* will produce something like this:



[7]Given our ability to reblock the CASINO data and discard an arbitrary number of initial steps, one might wonder why it is necessary at all to run equilibration steps in DMC. In fact, the algorithm in the equilibration and statistics accumulation phases is slightly different. In the equilibration phase, the underlying mean DMC energy is generally expected to decrease as time progresses, whereas in the statistics accumulation phase it is steady. Hence the 'best estimate of the energy' in equilibration is taken over a narrow window of the most recent **ebest_av_window** (=25 by default) moves, whereas in statistics accumulation the best estimate is the average over all moves since the start of statistics accumulation. The best estimate of the energy is used in the local-energy limiting schemes when computing the branching factors: see later.

This picture shows the results of the simulation of an antiferromagnetic NiO crystal with 1280 configurations. The upper panel shows how the population fluctuates as the simulation progresses. There is a feedback process in operation to limit the population fluctuations, so the population should just oscillate around the total initial number of configurations that we chose (1280 in this case). If you have a colour-printed manual or you are looking at the PDF version, in the bottom panel you will see a red line, a green line and a rapidly oscillating black line. The black line is the instantaneous value of the local energy averaged over the current population of configurations; the red line is the *reference energy* $E_{\rm T}$, which is adjusted to control the feedback process that keeps the population in check; and the green line is the *best estimate* of the DMC energy as the simulation progresses.

You should note that, in the picture above, the best estimate of the energy falls from its initial value (the energy from the VMC configuration-generation run) to a much lower, constant value (around -55.72 a.u.) as the wave function evolves to the ground state. You should look for the point at which the energy becomes roughly constant (at around 500 moves in this case). This splits the graph quite neatly into an equilibration phase and a statistics-accumulation phase. When we average energies to produce the final energy and error bar, we should only include those moves between 500 and the end of the run. More detailed information about choosing the number of moves for equilibration is given in Sec. 24.3.

To see DMC in action, go to `~/CASINO/examples/molecule/h2/RHF/dmc/` and let's calculate the DMC energy of a hydrogen molecule. The input files should already be set up correctly (though as before you may need to `gunzip` the `gwfn.data` file). We see that an equilibrated VMC run is set to go for **vmc_nstep**= 1000 moves in order to produce **vmc_nconfig_write**= 1000 configurations or walkers [8]. For DMC equilibration, we run for **dmc_equil_nstep**= 2000 moves, and for the statistics accumulation we run here for **dmc_stats_nstep**= 100000 moves (note that unlike **vmc_nstep**, these are *per process* quantities, since in DMC the parallelization is done over configurations, not steps. Study the definition of these keywords carefully in Sec. 7.3). The **dmc_stats_nblock** keyword is set to 5; this does not mean we do 5 blocks of 100000 moves each; rather, that we do 100000 moves and we checkpoint the data (write to output etc.) 5 times over the course of that calculation, i.e., every 20000 moves. The **dmc_target_weight** parameter is set to 1000.0—the same as **vmc_nconfig_write**, but note that they are allowed to differ. The **dtdmc** parameter is the DMC time step—note that it is much smaller than the VMC time step (it needs to be small because the DMC Green's function is only exact in the limit that the time step goes to zero). A typical value in a DMC simulation with all-electron ions might be 0.003, while a typical value in a simulation with pseudopotentials might be 0.02. Note that for accurate work, you need to consider extrapolating your DMC results to zero time step (see the utility `extrapolate_tau`, described in Sec. 10). Now type *runqmc*.

When CASINO has finished running, you can type *reblock* in the directory containing the `dmc.hist` file. `reblock` will read these files and then starting asking you questions. How many initial lines of data do you wish to discard? Usually the same as the number of lines of equilibration data—which is printed out—though look at the '`graphdmc`' plot to be sure. Find the move number where the green line in the plot becomes approximately constant (e.g., 501 in the NiO case). Then, what units do you want the answer in? Whatever you want. `reblock` will then show you the correlation-time analysis of the error bar, and start the reblocking analysis, where you need to choose a block length. Choose a value where the 'Std err of mean' column starts to plateau (again, the `plot_reblock` utility can help you visualize this). A value in the thousands might be typical; the value increases as you reduce the time step. `reblock` will then print out the final DMC energies and reblocked error bars (which should agree with the ones in the `out` file), together with an analysis of the population fluctuations, effective time steps and acceptance ratios.

In the case of molecular hydrogen, for my short 5-minute run the final energy is -1.174322 ± 0.00018 a.u. The exact energy is -1.1744757 a.u., which is within our reblocked error bar, so this result is pretty good without paying particular attention to getting it absolutely right—this is due to the system being exactly solvable in DMC due to the lack of nodes in the wave function.

[8] What is a typical number of DMC walkers? We usually say 'around 1000'. Of course this is a bit like the length of a piece of string. Nevertheless, for most systems with most wave functions on most computers it ensures that population-control bias—see later—is completely negligible, but still lets you gather data for a sufficiently long period in imaginary time that you can see a plateau on your reblock plot without heroic computational effort. If one has to give a single number for the target weight, clearly 1000 configurations is as sensible a suggestion as any. Obviously if you want to run on 20000 cores or whatever then you need a larger number of configurations.

6.5 How to perform QMC calculations for periodic systems

Suppose you wish to use QMC to study condensed matter rather than isolated atoms or molecules. For example, you might be interested in the bulk properties of a crystalline solid. Obviously, in a QMC simulation you can only study a small 'simulation cell' of such a crystal. Hopefully this cell will be sufficiently large that you are able to obtain the bulk properties of the material. Fortunately the boundary conditions on the simulation cell can be chosen to maximise the rate of convergence of the bulk properties with respect to the size of the cell. *Periodic boundary conditions* are a highly efficient and convenient choice, and are ubiquitous in computational studies of condensed matter.

According to Bloch's theorem, single-particle orbitals in a periodic crystal can be written as $\psi_{\mathbf{k}}(\mathbf{r}) = u_{\mathbf{k}}(\mathbf{r}) \exp(i\mathbf{k} \cdot \mathbf{r})$, where $u_{\mathbf{k}}(\mathbf{r})$ has the periodicity of the primitive cell and $\mathbf{k}$ lies in the first Brillouin zone of the primitive cell. The allowed Bloch vectors $\mathbf{k}$ are determined by the boundary conditions on the simulation supercell. Under periodic boundary conditions the allowed $\mathbf{k}$ are the reciprocal lattice points of the supercell. Under so-called *twisted* boundary conditions the allowed $\mathbf{k}$ are the reciprocal lattice points of the supercell offset by a constant vector $\mathbf{k}_{\rm s}$ in the first Brillouin zone of the simulation cell.

In an explicitly correlated method such as QMC the need to describe long-range correlations forces one explicitly to construct a simulation supercell consisting of several primitive cells. By contrast, in single-particle methods such as DFT, the problem can be reduced to a single primitive cell with a grid of $\mathbf{k}$ points, and the simulation supercell is never explicitly constructed.

You could generate a trial wave function for use in your QMC calculation by performing a DFT calculation for the simulation supercell at a single $\mathbf{k}$ point. However this would be inefficient: the translational symmetry within the simulation cell would not be exploited to reduce memory requirements in both the DFT calculation and the subsequent QMC calculation. Instead you should generate the trial wave function for a simulation supercell consisting of $l \times m \times n$ primitive cells by performing a DFT calculation in a single primitive cell with an $l \times m \times n$ mesh of $\mathbf{k}$ points. The KVEC_MAKER utility will help you to generate an appropriate $\mathbf{k}$-vector grid: simply run the utility and answer the questions.

In summary, your DFT calculation should be set up to generate a set of orbitals for a single primitive unit cell with an $l \times m \times n$ $\mathbf{k}$-point grid, whilst the QMC calculation should be set up for a simulation supercell consisting of $l \times m \times n$ primitive cells. In the `input` file, the number of up- and down-spin electrons **neu** and **ned** should be appropriate for the supercell, whilst the **npcell** block, which specifies the size of the supercell, should contain 'l m n'.

Note that QMC methods cannot exploit symmetry (apart from time-reversal symmetry), and hence a complete $l \times m \times n$ grid of $\mathbf{k}$ vectors must be supplied in general. (How you do this depends on your DFT code.) If you are using a grid of $\mathbf{k}$ vectors with time-reversal symmetry then your trial wave function in the subsequent QMC calculations can be chosen to be real, and the **complex_wf** `input` keyword should be set to `F`. In this case you only need to supply one out of each $\pm\mathbf{k}$ pair. If you do not have time-reversal symmetry you must supply the complete grid of $\mathbf{k}$ vectors and set the **complex_wf** keyword to `T`. The KVEC_MAKER utility will ask you whether time-reversal symmetry is to be applied

Typically you will need to perform QMC calculations in two different supercell sizes and extrapolate your results to the limit of infinite cell size. How exactly you proceed depends a good deal on what you are calculating. The problem of finite-size errors in QMC calculations is discussed extensively in Ref. [15].

6.6 How to run the code: RUNQMC

The `runqmc` script (which should be in your path after installing CASINO) is designed to reduce the effort of doing QMC calculations to just entering one command, and that command is essentially the same on all different kinds of computer. It is most useful when using parallel machines with a batch queueing system, since it detects most common errors that a user may make in setting up a calculation and such faults are thus detected immediately rather than when CASINO actually starts running (which may be many hours or even days later). See the `utils/runqmc/README` file for more information.

`runqmc` can run calculations on single- and multiprocessor workstations, and on clusters. To run the CASINO calculation set up in the current directory, simply type *runqmc*. This will automatically—and possible a little dangerously—occupy all cores on a workstation, or the maximum permitted allocation (in both number of cores and walltime) on a cluster. To change this, or to specify other calculation parameters, there are various options available.

An example on a big parallel machine:

```
runqmc --nproc=224256 --walltime=10h --shm=12
```

will run the code in parallel using 224256 MPI processes (distributed among computational nodes depending on the number of cores per node in the machine, which runqmc knows about) using shared memory over groups of 12 processes with a walltime limit of 10 hours (which may affect what queue your job is put in).

Note that this is equivalent to

```
runqmc -p 224256 -T 10h -s
```

(provided the machine has 12 cores per node) i.e., most options come with a longer version (with two hyphens and an equals sign) and a shorter easier-to-type version (with one hyphen, and zero or more spaces before the option value).

A really important point which trips quite a few people up: on machines with batch queue systems everyone is used to manually creating batch scripts which tell the machine how to run the code, how many cores, time limits etc, which is then manually *qsub*bed to the batch queue. **Do not insert a call to runqmc in such a script** (© the CASINO Wiki on the former UK national facility 'Hector', written by the sysadmins). The purpose of *runqmc* is to **create** this batch script, and *runqmc* will then submit it all by itself. OK? We appreciate this is unusual, but it works.

If you can't remember the basic parameters of the machines that you're running on (e.g., the number of cores per node, queue-dependent walltime limits, the maximum number of cores etc.) then typing `runqmc --info` or `runqmc -i` will print to the screen all relevant parameters available from the machine's arch file. This may help you decide on what values to use for the run time parameters when invoking `runqmc`.

6.6.1 Other facilities

(1) `runqmc` can run calculations in multiple directories, by simply giving the directory names at the end of the command line, e.g.,

```
runqmc -p 256 -T 10h diamond beta_tin
```

will run the two calculations in directories 'diamond' and 'beta_tin', where *each* of them uses 256 cores.

Running, e.g., 2 calculations on a 4-core workstation will result in 2 cores being used for each of them (unless otherwise specified via '`-p`'). Running multiple calculations on clusters is only possible for specific machines; multiple calculations on clusters will use a single batch queue slot and a single script. This is useful for large clusters where runs on large numbers of cores are cheaper per core-hour by policy, but is otherwise a bad idea since different calculations will take different times to complete but the longest will be charged for.

If no directories are specified, the calculation in the current directory will be run.

(2) In order to run on a cluster log-in node, use the '`--no-cluster`' option, which will make `runqmc` behave as if the machine was a workstation. **You should check with your system administrator to see if this is OK**.

(3) If you would like to produce a submission script on a cluster without running CASINO (for verification purposes), use the '`--check-only`' flag.

(4) In order to use the OpenMP capabilities of CASINO, use the flag '`--tpp=⟨threads-per-process⟩`'. `runqmc` will automatically adjust the number of processes per node to leave one OpenMP thread per core. If you would like to modify this behaviour, use the '`--ppn=⟨processes-per-node⟩`' flag and set your own value.

You need to have compiled the OpenMP version with '`make Openmp`' in order to access this feature.

(5) In order to use the shared-memory capabilities of CASINO, use the flag '`--shm`'. `runqmc` will tell CASINO to share orbital coefficients (blip or Gaussian only) and some other data among all cores in each node; to modify this behaviour use instead '`--shm=⟨number-of-cores-in-SHM-group⟩`'.

You need to have compiled the SHM version with '`make shm`' in order to access this feature ('`make OpenmpShm`' is also available).

(6) Individual machines can have user-defined options `--user.option` in order to cope with particular idiosyncrasies. See the end of the Usage section below.

(7) A similar script '`runpwscf`' is available for running the PWSCF plane-wave DFT-code, which also uses the same `CASINO_ARCH` system for determining what computer we're running on. It has the same set of optional arguments as `runqmc`, apart from the obviously QMC-specific ones (plus some extra ones for controlling PWSCF behaviour). Note that if your CASINO arch file defines a command for running CASINO, then it must include a tag `&BINARY_ARGS&` following the `&BINARY&` tag; this is because the PWSCF executable takes command line arguments such as `-pw2casino`, `-npool` etc., which are not required by CASINO.

(8) A script '`runqmcmd`' is available which runs coupled DFT-DMC molecular dynamics calculations (see section6.7). This exploits both '`runpwscf`' and '`runqmc`' to alternately run PWSCF DFT calculations and CASINO QMC calculations.

(9) A script '`twistav_pwscf`' is available for running 'twist-averaged' calculations where new `xwfn.data` files are repeatedly generated by PWSCF for different twist angles (offsets of the k-point grid from the origin)—see Sec. 28. This is useful in reducing finite-size effects in periodic systems. Again, this facility exploits both the '`runpwscf`' and '`runqmc`' scripts.

6.6.2 Usage

Run `runqmc --help` to display the full set of options available to your specific machine.

```
runqmc [<options>] [[--] <directories>]
```

The core set of options is as follows:

Options available on all machines

```
 --force | -f
   Run the calculation without checking for presence/correctness of input
   files.

 --check-only | -c
   Stop before running the calculation.  In clusters, this option can be used
   to produce the batch submission script for manual checking; '--check-only
   --force' would only produce a submission script in these systems.

 --unlock | -u
   Ignore existing lock files and run the calculation regardless.  Using this
   option will remove stale lock files left over by a runqmc instance that died
   without being able to unlock the directory.

 --version=<version> | --opt | --dev | --debug|-d | --prof
   Select the binary version <version>, which ought to be one of 'opt', 'dev',
   'debug' or 'prof'.  --opt, --dev, --debug and --prof are equivalent to
   the respective --version=<version> option.  -d sets <version> to 'debug'.
   <version> is set to 'opt' by default.

 --continue
   Continue a previous run which provides continuation info.  This requires
   using the MAX_CPU_TIME and/or MAX_REAL_TIME input keywords.  This option
   is not available when running multiple jobs.

 --auto-continue
   Start and automatically continue a run until it finishes.  This requires
   using the MAX_CPU_TIME and/or MAX_REAL_TIME input keywords, and
   partitioning the job into multiple blocks (CASINO can emergency stop
   at the end of a block if there is not sufficient time remaining to do at
   least one more block).  This option is not available when running multiple
   jobs.

 --home=<home> | --chome=<home> | -H <home>
   Set the location of the CASINO installation to <home>.  By default, <home>
   is determined by the location of this script, or set to \$HOME/CASINO if
   unsuccessful.

 --binary=<binary> | -b <binary>
   Set the binary name to use to <binary> instead of 'casino'.  This only needs
   to be used for custom compilations with the option 'EXECUTABLE=<binary>'.

 --tpp=<tpp> | -t <tpp>
   Set the number of OpenMP threads per process to <tpp>.  This requires having
   compiled the code with OpenMP support, as in 'make Openmp'

 --acc
   Enable the use of accelerators (GPUs) using OpenACC.  This requires having
   compiled the code with OpenACC support, as in 'make Openacc'

 --help | -h
   Display this help.  If the CASINO_ARCH can be determined and exists, the
   help will display options specific to the current manchine, else all options
   will be displayed.

 --verbosity=<verbosity> | -v | -q
   Set the verbosity level of the machine set-up process to <verbosity>.  By
   default <verbosity> is 0.  '-v' increases the verbosity level by 1, and
   '-q' decreases it by 1.

  --info | -i
   Report back various machine-dependent parameters whose value one might like
   to know when deciding how to run a job, then stop. This is useful
   for inquiring what runqmc believes to be, e.g., the number of cores per node,
   according to the currently activated arch file.
```

Options available on workstations

```
 --background | -B
   Run CASINO in the background, returning control to the shell after starting
   the run.  This has the same effect as '${0##*/} & disown', whereby the
   CASINO process is detached from the shell, so if one wants to stop the run
   'kill' or 'killall' must be used.  It is safe to log out after running with
   this option, the calculation will continue---no need for nohup/disown.
   Running multiple jobs causes them to run in the background whether this
   option is specified or not.

 --print-out | -P
   Print out the output of CASINO as it is being run.  Implies --background.
   [CTRL]-[C] will stop the print-out, and the CASINO job will remain in the
   background.  This option is ignored when running multiple jobs.

 --debugger=<debugger> | --gdb|-g | --valgrind
   Run the code through the a debugger.  This automatically sets <version> to
   'debug' if no version had been selected.  --gdb and -g are equivalent to
   --debugger="gdb -q".  --valgrind is equivalent to --debugger="valgrind"
   Note that specific debuggers tend to work better with specific compilers,
   for example the gdb debugger tends works better with GCC's gfortran than
   with Intel's ifort.
```

Options available on parallel workstations and clusters

```
 --no-mpi | -1
   Run the binary directly without invoking mpirun etc.  This option is applied
   before any others, and makes this script behave as if the machine was a
   single-core machine.
 --nproc=<nproc> | -p <nproc>
   Set the number of MPI processes to <nproc>.  This will have to be consistent
   with <tpp>, <nnode>, <ppn> and the machine information in the relevant
   .arch file.

 --ppn=<ppn>
   Set the number of MPI processes per physical, multi-core node to <ppn>.
   This will have to be consistent with <tpp>, <nproc>, <nnode>, <ppn> and the
   machine information in the relevant .arch file.

 --shm[=<numablk>] | --shmem[=<numablk>] | -s
   Enable shared memory (which affects, e.g., storage of orbital
   coefficients in blip or Gaussian mode and some other large arrays).
   If <numablk> is provided, set the
   number of processes among which to share memory to <numablk>, otherwise
   shared memory is used across all cores on the same node by default. This
   option requires having compiled the code with shared memory support, e.g.,
   with 'make shm' or 'make openmpshm.

 --diagram | -D
   Draw a diagram of the processes and threads on each node to the terminal
   during set-up.
```

Options available on clusters

```
 --no-cluster | -l
   Run the calculation directly on the login node of a cluster without
   producing a submission script.  This option is applied before any others,
   and makes this script behave as if the machine was a multi-core workstation.

 --nnode=<nnode> | -n <nnode>
   Set the number of physical, (possibly) multi-core nodes to use to <nnode>.
   This will have to be consistent with <tpp>, <nnode>, <ppn> and the machine
   information in the relevant .arch file.

 --walltime=<walltime> | -T <walltime>
   Set the wall-time limit for the run to <walltime>, given in the format
   [<days>d][<hours>h][<minutes>m][<seconds>s].

 --coretime=<coretime> | -C <coretime>
   Set the core-time (i.e., wall-time times number of reserved cores) limit for
   the run to <coretime>, given in the format
   [<days>d][<hours>h][<minutes>m][<seconds>s].

 --name=<name> | -N <name>
   Set the submission script name and job name to <name>.
```

Options available on some individual machines

If you type `runqmc --help` you may see appended to the above list a set of additional user-definable options—these all take the form of '`--user.option`' (and obviously a `CASINO_ARCH` must be defined in order to see these). Such options are defined in the relevant `.arch` file using the syntax of Appendix 5. The arch system was designed to be readily extensible on a per-machine basis without having to modify any part of the script system that handles it, and thus only variables associated with core functionality are pre-defined, e.g., even the (widely applicable but Linux-specific) nice value of a job on a workstation is handled as `--user.nice`.

Examples of such options include but are not limited to:

```
--user.account
 Name of account under which to run the job (note that it's very useful to
alias runqmc='runqmc --user.account=my_account ' on machines which require it).

--user.nice
```

```
 Unix 'nice' value to use for the job on workstations.

--user.queue
 Name of queue on which to run the job.

--user.mem
 (Over-)estimate of the memory usage per compute node in Mb.

--user.shmemsize
 When running in shared memory mode [on a Blue Gene machine] you are required
  to set the size of the shared memory partition when launching a job (through
 the value of an environment variable named BG_SHAREDMEMPOOLSIZE (Blue Gene/P)
 or BG_SHAREDMEMSIZE (Blue Gene/Q)). The value of this number (in Mb) may be
 set on the command line as the value of user.shmemsize. See the CASINO FAQ for
 further details. Note that setting --user.shmemsize implies an Shm
 calculation; there is no need to use -s/--shm/--shmem as well unless you want
 to change 'numablk' (the number of processes among which to share memory)
 from the default of all cores on the node.

--user.messagesize
 Size of MPI message in bytes below which the 'eager' (truly non-blocking)
 protocol is used instead of the 'rendezvous' (partially non-blocking)
 protocol. Perfect parallel scaling at very large numbers of MPI processes
 (> c.20,000) with CASINO supposedly requires the non-blocking message
 passing to be truly asynchronous (i.e., communication takes place at the same
 time as communication). The default value for MESSAGESIZE on [a
 Blue Gene/P machine] (1200 bytes) is generally too small, as CASINO config
 messages typically range up to around 50000 bytes. Thus some advantage may
 be gained on large numbers of cores by setting MESSAGESIZE to be larger than
 the default.
```

6.7 How to run coupled DFT-DMC molecular dynamics calculations: the `runqmcmd` script

Coupled DFT-DMC molecular dynamics—which involves a correlated sampling procedure that couples the stochastic imaginary-time electronic and real-time nuclear trajectories—is popularly supposed to result in highly accurate energies for a full dynamical trajectory at a fraction of the cost of doing a full QMC calculation for each nuclear configuration. The basic ideas are outlined in Grossman and Mitas's paper—Ref. [9].

In my (MDT's) opinion, this technique is often misrepresented, both by the authors of the GM paper, and by others. It is claimed, for example, in the original paper that:

'*This continuous evolution of the QMC electrons results in highly accurate total energies for the full dynamical trajectory at a fraction of the cost of conventional, discrete sampling.*'

'[It provides]*an improved, significantly more accurate total energy for the full dynamical trajectory.*'

'*This approach provides the same energies as conventional, discrete QMC sampling and gives error bars comparable to separate, much longer QMC calculations.*'

A casual reading of the above sentences might lead to the conclusion that we get effectively the same results as conventional DMC at each point along the trajectory. As we are only doing, e.g., 3 DMC stats accumulation per step, we thus appear to be getting 'something for nothing'. What are we missing?

(1) The point about GM-MD—and Lubos Mitas has agreed with this in an email to me—is that its true purpose is to calculate *thermodynamic averages*, an example being given in the paper of the heat of vaporization of H_2O. This is calculated as an average over the evolution path of the ions (at a given $T > 0$) and QMC and thermodynamic averages are done at the same time. Think of it as a method of Monte Carlo sampling a distribution in 3N-dimensional configuration space that is changing shape in time. Each nuclear configuration is sampled via far fewer (e.g., three moves times the number of walkers) electron configurations than usual. It is not intended that accurate energies for all nuclear configurations are calculated, only an accurate Monte-Carlo-sampled 'thermodynamic average' as the molecule (or whatever) vibrates or otherwise moves.

(2) If you want accurate answers and small error bars for the energies at each point along the MD trajectory then—done under normal conditions (number of DMC walkers etc.)—the method actually does not save you any time at all, apart from that spent doing DMC equilibration. It gives extremely poor answers with huge error bars for the individual MD points; if you want proper DMC accuracy then you have to do the usual amount of statistical accumulation work.

(3) The only reason they are able to claim '*the same energies* [and error bars] *as conventional, discrete QMC sampling*' is because they use extremely large populations of walkers (a technique which is difficult to scale to large systems). The requirement for a large number of walkers is not stressed in the paper; it is merely stated that '*we chose a number of walkers such that the statistical fluctuations in the GM-MD energies are one-tenth the size of the variation in the total energy as a function of the MD time* with no emphasis that this is considerably more than usual.

There is also a big question over whether the configurations read in at the restart can be properly equilibrated in so few moves in the case when the DMC wave functions involve genuinely new physics. MDT has a discussion document in circulation which covers these issues in more detail.

Bearing in mind these reservations, the technique has been implemented in CASINO as follows.

The `runqmcmd` script is used to automate DMC-MD molecular dynamics calculations using CASINO and the PWSCF DFT code (part of the QUANTUM ESPRESSO package, available at `https://www.quantum-espresso.org`). PWSCF must be version 4.3 or later.

```
runqmcmd [--help --nproc_dft=I --splitqmc[=N] --startqmc=M
          --dft_only/--qmc_only [<runqmc/runpwscf options>]
```

We first generate a full DFT trajectory. We then do a full QMC calculation for the initial nuclear configuration of the trajectory, followed by a series of very quick (a few moves) DMC calculations for each point along the trajectory, where each such calculation is *restarted* from the `config.out` of the previous one with slightly different nuclear coordinates.

This script works by repeatedly calling the `runpwscf` and `runqmc` scripts which know how to run PWSCF/CASINO on any individual machine. Almost all optional arguments to this script are the same as for `runpwscf/runqmc` and are passed on automatically to these subsidiary run scripts (the –background/-B option is also used by runqmcmd, and for the same purpose). Type '`runpwscf --help`' or '`runqmc --help`' to find out what these options are (or see the previous section of the manual). There is a short list of optional flags specific to `runqmcmd` which are described below.

It is assumed that PWSCF lives in `$HOME/espresso` and CASINO lives in `$HOME/CASINO`. There are override options available if this is not the case.

If you are running on a multi-user machine with an account to be charged for the calculations, you might consider aliasing `runqmcmd` using, e.g., `alias runqmcmd="runqmcmd --user.account=CPH005mdt "` or whatever.

In general you should do something like the following:

Setup the PWSCF input ('`in.pwscf`') and the CASINO input ('`input`', etc., but no wave function file) in the same directory. For the moment we assume you have an optimized Jastrow from somewhere. Have the PWSCF setup as '`calculation = "md"`', and '`nstep = 100`' or whatever. The `runqmcmd` script will then run PWSCF once to generate 100 `xwfn.data` files, then it will run CASINO on each of the `xwfn.data`. The first will be a proper DMC run with full equilibration (using the values of **dmc_equil_nstep**, **dmc_stats_nstep** etc). The second and subsequent steps (with slightly different nuclear positions) will be restarts from the previous converged `config.in`; each run will use new keywords **dmcmd_equil_nstep** and **dmcmd_stats_nstep** (with the number of blocks assumed to be 1. The latter values are used if new keyword **dmc_md** is set to `T`, and they should be very small).

It is recommended that you set **dmc_spacewarping** and **dmc_reweight_conf** to `T` in CASINO input when doing such calculations.

The calculation can be run through `pwfn.data`, `bwfn.data` or `bwfn.data.bin` (and obsolete `bwfn.data.b1`) formats as specified in the `pw2casino.dat` file (see elsewhere).

Default behaviour of runqmcmd (on all machines)

Note: in what follows, **nmdstep** is the value of the PWSCF input keyword '**nstep**', while `xwfn.data` refers to whatever wave function file is specified in the `pw2casino.dat` file (either `bwfn.data.b1/bwfn.data.bin` [default], `bwfn.data` or `pwfn.data`).

For a complete DMC-MD run, the following three steps are performed in sequence:

(A) Generate **nmdstep**+1 `xwfn.data.$` files, where $ is a sequence number from 0 to **nmdstep**.

(B) Do a full DMC run on `xwfn.data.0`

(C) Temporarily modify the CASINO `input` file, by changing **dmc_md** from `F` to `T`, and **runtype** from `vmc_dmc` to `dmc_dmc`. Run **nmdstep** restarted QMC runs on `xwfn.data.1` to `xwfn.data.[nmdstep]`, each restarting from the previous.

On batch queue systems, `runqmcmd` will by default do *two* batch script submissions, the first—handled by the `runpwscf` script—executing step (A), and the second—handled by the `runqmc` script—executing steps (B–C).

In principle, this wastes some unnecessary time (the time spent waiting for the QMC batch script to start) but this is unavoidable if `runqmcmd` uses separate `runpwscf` and `runqmc` scripts to handle the DFT and QMC calculations. This may be changed in the future, if anyone can be arsed.

Note that all calculations will be done on the number of cores requested on the command line (with the `--nproc/-p` flag) irrespective of whether they are DFT or QMC calculations. You may override this for the DFT calculations by using the `--nproc_dft` flag to runqmcmd.

Modifications to default behaviour (on all machines)

```
runqmcmd --dft_only
```

Execute only step (1), generating **nmdstep**+1 `xwfn.data.$` files.

Essentially the same thing can be done by executing '`runpwscf --qmc`' but doing that would bypass a few error traps.

```
runqmcmd --qmc_only
```

Execute only steps (B & C). This requires that the **nmdstep**+1 `xwfn.data.$` files already exist; if they don't the script will whinge and die.

```
runqmcmd --startqmc=M
```

Start the chain of QMC runs with file `xwfn.data.M` If $M = 0$, the first run will be a full QMC run with **dmc_md**=`F`, otherwise if $M > 0$ then all runs will be short restarted ones with **dmc_md**=`T`. (Note that for $M > 0$, **dmc_md** and **runtype** in the input file will be temporarily 'modified' as described above, no matter what values they currently have).

Modifications to default behaviour (batch machines only)

On batch machines, there is an additional complication due to the walltime limits on particular queues which may require full DMC-MD runs to be split into sections. The following flags may be used to do this.

```
runqmcmd --splitqmc
```

Do step A (DFT run), step B (initial QMC run) and step C (chain of remaining QMC restarted jobs) as three separate batch script submissions (i.e., no longer combine B and C).

```
runqmcmd --splitqmc=N
```

As (3) but split step C into N separate batch script submissions.

Example: **nmdstep**=1005, and `runqmcmd --splitqmc=4` will result in 1 step B job plus four sets of step C jobs with 251, 251, 251, 252 steps.

Note finally that there are a couple of simple utilities (`extr_casino and extr_pwscf` that extract the DFT/QMC energies from the output of a runqmcmd run.

7 Files used by CASINO

A complete list of the input files is given in Sec. 7.1, while a corresponding list of the output files is given in Sec. 7.2. By *input files* we mean all files that may be read by CASINO, including files that may also be written to. By *output files* we mean those files that are only written to, even where the data is appended to an existing file.

7.1 Complete list of the input files

7.1.1 The principal input files

`input` This is the main input parameter file.

`correlation.data` This file contains all *optimizable* parameters together with accompanying data (for example, the parameters used to define a Jastrow factor or backflow function).

`xwfn.data` This file contains the data that defines the geometry, the orbitals and, if appropriate, the determinant expansion coefficients calculated by the generating code. '`x`' indicates one of the various different basis sets supported by CASINO. The filenames supported at present are: `pwfn.data` (plane-wave basis), `gwfn.data` (Gaussian basis), `awfn.data` (numerical atomic orbitals on a grid), `dwfn.data` (numerical molecular orbitals for dimers), `bwfn.data` (blip basis) and `stowfn.data` (Slater-type orbitals).

`bwfn.data.bin` New (11/2011) format binary blip wave function file (much smaller than the human-readable `bwfn.data`). Created and written automatically on reading a formatted `bwfn.data` file. These files are portable between same-endianness machines and compilers. Where portability is not possible, the code should error out nicely thanks to an endianness check included in the binary file.

`bwfn.data.b1` Old format binary blip wave function file—deprecated but still completely supported. Note that PWSCF is capable of producing single `bwfn.data.b1` files directly; it is intended to one day convert it produce the newer `.bin` files. (Note that an old method of splitting blip data over processors used additional files such as `bwfn.data.b2`, `bwfn.data.b3`, etc., hence the notation. With the advent of a proper shared-memory facility, support for these additional files has been completely removed).

Here is a list of benefits of the newer `.bin` format over the `.b1`:

- Extensible and flexible, by using labels for groups of data and a format version string at the start of the file (instead of by means of hacks).
- Portable between different compilers on same-endianness machines
- Detection of different-endianness machines
- Reduced size for spin-polarized systems (up to 50% smaller)
- Up to 4 times faster to write (to quote a figure, the actual amount will vary depending on the case)

`mpc.data` This file contains data used by the MPC interaction. This includes the Fourier components of the charge density corresponding to the Slater wave function and the Fourier components of the $1/r$ Coulomb interaction treated within the minimum-image convention.

`config.in` This is the file used to store configurations and other data (such as random-number-generator states) between different stages of a calculation (VMC-optimization, optimization-VMC, VMC-DMC). It also contains data to continue VMC and DMC runs.

`x_pp.data` (where `x` is the chemical symbol of an element in lower-case letters.) This file contains the pseudopotential data for the corresponding element. In certain rare circumstances one may wish to use multiple pseudopotentials for elements with the same atomic number (e.g., Mg with He core on a surface, and Mg with Ne core in the bulk). Different types of pseudoatom are flagged in `xwfn.data` by adding multiples of 1000 to the original atomic number, e.g., atno 12, 1012, 2012 and these different pseudopotentials are stored in files called, e.g., `mg_pp.data`, `mg2_pp.data`, `mg3_pp.data` etc.

`expval.data` This file contains data that allow the estimation of various expectation values. For example, this file may contain the charge density, spin density, spin density matrix, pair-correlation function, localization tensor, structure factor, one-electron density matrix or two-electron density matrix.

`expot.data` This file contains a specification of any external potential (for example, the potential for an inhomogeneous electron gas calculation, or the potential due to an external electric field), and data defining the orbitals associated with such potential. It is also used to specify the magnetic vector potential.

7.1.2 Other input files

`config.backup` If a population-explosion catastrophe occurs during a DMC simulation, and the input keyword **dmc_trip_weight** is set, then this file is read and the simulation jumps back to an earlier point. This file is not read at the start of a DMC simulation, and is not required under normal circumstances.

`expval.backup` As `config.backup`, but for the expval.data file (if accumulation of expectation values other than the energy is flagged in input).

7.2 Complete list of the output files

7.2.1 The principal output files

The files listed below are the 'standard' output files, which users are likely to encounter.

`out` This file contains the main output of CASINO.

`correlation.out[.`*n*`]` This is the `correlation.data` file generated by the nth cycle of CASINO's optimization procedure.

`vmc.hist` This file contains all of the energy components calculated during VMC. This file can be analysed using the `reblock` utility.

`dmc.hist` This file contains the energy components and important simulation parameters at each iteration in a DMC simulation. This file can be analysed using the `reblock` utility.

`config.out` This is the name under which the `config.in` file is produced. It will be automatically renamed by `runqmc` provided `config.in` does not exist; otherwise, `runqmc` will alert the user and exit.

`dmc.status` This file is written and overwritten at the end of each block of a DMC simulation. At any time during the run, this file contains the complete statistical evaluation as it would be written if the run was finished here. The `out` file only contains this data once at the end, if the run is finished properly.

`movie.out` This file contains the data for making a 'movie' of the simulation.

`lineplot.dat` This file contains data such as orbital values plotted along a straight line in space.

`2Dplot.dat` This file contains visualization data given on a 2D plane.

`3Dplot.dat` This file contains visualization data given over a 3D volume.

7.2.2 Other output files

The files listed below are only likely to be of interest to developers.

`local_energy.dat` This file contains information related to the cusp correction to Gaussian orbitals.

`orbitals.dat`**,** `gradients.dat` **and** `laplacians.dat` These files contain information relating to the cusp correction to Gaussian orbitals.

`random.log` This file holds information about the state of the random-number generator.

`jastrow_value_u.dat`**,** `jastrow_deriv_u.dat` **and** `jastrow_sderiv_u.dat` These files contain the value, first derivative and second derivative, respectively, of the u term of the Jastrow factor with respect to electron separation.

`jastrow_value_chi_x.dat`**,** `jastrow_deriv_chi_x.dat` **and** `jastrow_sderiv_chi_x.dat` These files contain the value, first derivative and second derivative, respectively, of the χ term of the Jastrow factor (for set `x`) with respect to electron–nucleus distance.

`jastrow_value_f_x.dat` This file contains the value of f (for set `x`) as an electron is moved along a straight line.

`jastrow_value_p.dat` This file contains the value of p as an electron is moved along a straight line.

`jastrow_lap_p.dat` This file contains the Laplacian of p as an electron is moved along a straight line.

`jastrow_value_q.dat` This file contains the value of q as an electron is moved along a straight line.

`jastrow_lap_q.dat` This file contains the Laplacian of q as an electron is moved along a straight line.

`jastrow_value_ucyl.dat` This file contains the value of $u_{\rm cyl}$ as an electron is moved along a straight line.

`jastrow_lap_ucyl.dat` This file contains the Laplacian of $u_{\rm cyl}$ as an electron is moved along a straight line.

`jastrow_value_chicyl.dat` This file contains the value of $\chi_{\rm cyl}$ as an electron is moved along a straight line.

`jastrow_lap_chicyl.dat` This file contains the Laplacian of $\chi_{\rm cyl}$ as an electron is moved along a straight line.

`bfconfig.dat` Dump of configuration used for backflow plot.

`bfconfigx.dat` Dump of quasiparticle configuration used for backflow plot.

`bffield.dat` Plot of the backflow-displacement vector field on a plane.

`bfeta_s.dat` Plot of the backflow η function for spin-dependence `s`.

`bfmu_s_set.dat` Plot of the backflow μ function for spin-dependence `s` and set `set`.

`bfphi_s_set.dat` 3D plot of the backflow $\Phi + \Theta$ function for spin-dependence `s` and set `set` (on a plane).

`lsqfun.dat` Plot of the variance against the value of a parameter. This file can only be produced by modifying the code.

`emin.log` Matrix algebra log produced by an energy minimization run.

`btilde.log`, `bhtilde.log`, `SVD_cpts_0.log` **and** `SVD_cpts_1.log` Full matrix logs produced by an energy-minimization run.

`ft_of_jastrow.dat` Fourier transform of u and p terms in Jastrow factor for a periodic system.

7.3 Basic input file: `input`

The file `input` contains all the parameters needed to control the QMC calculation. A complete list of the `input` parameters is given below. Further details, including default values, may be found by using the `casinohelp` utility. Type *casinohelp all* to get a list of all keywords that CASINO knows about, or *casinohelp keyword* for detailed help on a particular keyword. Type *casinohelp search text* to search for the string *text* in all the keyword descriptions. Create a blank file containing the keyword **input_example** and type *runqmc*—this will create a sample `input` file containing all valid keywords and their default values in the correct format.

The `input` file is meant to be very flexible and the list of understood keywords can vary with time without breaking anything. The `input` file is based on an early version of the *electronic-structure data format* (ESDF) developed for the CASTEP code. The main points are summarized below. See the comments at the top of the `esdf.f90` module for more information.

- Each line is of the form 'keyword : value'. There *must* be a space either side of the colon.
- The parameters are divided into types: 'string'/'integer'/'single'/'double'/'physical'/'boolean'. Variables of 'physical' type must be supplied with a unit, such as 'hartree', 'eV', 'rydberg' or 'Joules/Megaparsec'. All reasonable physical units are understood.
- The parameter names are case-insensitive (e.g., **RunType** is equivalent to **runtype**) and punctuation-insensitive (**run_type** is equivalent to **run-type** and **runtype**). Punctuation characters are '.', '_' and '-'.

- Some of the parameters are of 'block' type, which means that multiple parameters must be supplied, which may be spread over several lines. See, e.g., 'qmc_plot'.

Note that the `input` file is never written to by CASINO (though there are a couple of utilities, such as the `runqmcmd` script, which temporarily manipulate it). Files generated by other codes, which may contain large amounts of data, are not included in `input`.

7.3.1 List of current input keywords

Here is the current list of the input parameters in alphabetical order (a list of some older keywords is given after this). Note that the *casinohelp* facility is always up to date (it runs CASINO to find out what it knows about)—but this manual may not be.

ALIMIT (*Real*) Parameter required by DMC drift-velocity- and energy-limiting schemes when **limdmc**=2, 3, 4 or 6. A value of 0.5 (default) is generally appropriate. The DMC energy is insensitive to the precise value of **alimit**. **alimit** is ignored if **nucleus_gf_mods** is set to `T`. See Sec. 13.5.

ALLOW_NOCHI_ATOMS (*Logical*) If this keyword is set to `T` then CASINO will issue a warning message when some atoms are not included in any sets of χ or f terms in the Jastrow factor and μ and Φ terms in the backflow function. Otherwise, CASINO halts with an error if some atoms are not included in these terms.

ALLOW_SLAVE_WRITE (*Logical*) The ALLOW_SLAVE_WRITE flag can be used to allow/disallow slave process output to the main output file. The default is to allow it. The ability to turn this off can be useful when you're running on a million cores.

ALPHALIMIT (*Real*) Parameter required by DMC, in ZSGMA branching limiting scheme (i.e., when **limdmc**=4). A value of 0.2 (default) is generally appropriate. The limit for $\tau \to 0$ of the DMC energy is insensitive to the precise value of **alphalimit**, but it affects the finite τ DMC energy and the stability.

ATOM_BASIS_TYPE (*Text*) This selects the basis set in which the atom-centred orbitals are expanded, or more generally, the 'type of orbital' to be used. If the orbitals are to be read from disk, this implicitly selects which file to read the orbitals from. Possible values are:
'none' (default): no atoms are present, so no externally generated orbitals are read in;
'plane-wave': use a plane-wave basis set; the orbitals are read in from `pwfn.data`;
'gaussian': use a Gaussian basis set; the orbitals are read in from `gwfn.data`;
'slater-type': use Slater-type orbitals; the orbitals are read in from `stowfn.data`;
'numerical': use orbitals tabulated on a grid (atomic systems only); the orbitals are read in from `awfn.data`;
'dimer': use orbitals tabulated on a grid (molecular dimers only); the orbitals are read in from `dwfn.data`;
'blip': use a blip basis set; the orbitals are read in from `bwfn.data`.

Some special wave function types are also available:
'nonint_he': use exact orbitals for a noninteracting helium atom.
'h2' or 'h3plus': wave functions for the H_2 molecule or the H_3^+ molecular ion where each orbital is the sum over hydrogen nuclei of a parameter-less exponential centred at each nucleus.

For free-particle and external-potential-related orbitals, set **atom_basis_type** to 'none' and use the input block **free_particles**.

In a 'gen_blip' calculation, **atom_basis_type** should be set to 'plane-wave', since the geometry and orbitals will be read from `pwfn.data`.

BACKFLOW (*Logical*) Turns on backflow corrections (see Sec. 23). Backflow parameters are read from `correlation.data` and, if optimized (**opt_backflow** = `T`), written to `correlation.out`.

BF_SPARSE (*Logical*) Setting **bf_sparse** to `T` will result in the Woodbury formula being used to update the inverse Slater matrices and determinants instead of recomputing the determinants entirely. This is advantageous only if the backflow functions are short-ranged with respect to the size of the system.

BLIP_CALC_KE (*Logical*) Choose whether to perform a numerical evaluation of the norm squared and kinetic energy of blip orbitals, to be compared with the kinetic energies evaluated in a plane-wave basis, in a blip-generation calculation.

BLIP_NBAND_MAX (*Block*) This block consists of a single line with a list of **nspin** integers, these being the maximum numbers of bands to be transformed from a plane-wave basis to a blip basis in a blip-generation calculation. If a negative integer is given then all bands for that spin will be re-represented in a blip basis (this is the default behaviour).

BLIP_NRANDPOINTS (*Integer*) Number of random points for the Monte Carlo evaluation of the overlap between blip and plane-wave orbitals in a blip-generation calculation. By default this is zero, meaning that the overlap test is skipped.

BLIP_MPC (*Logical*) If **blip_mpc** is set to `T` and one is using the MPC interaction in a system that is periodic in all three dimensions and consists of only electrons then the long-range portion of the MPC potential will be evaluated using blip functions. In some systems setting this to `T` can greatly speed up the calculation. The default is `F`.

BLIP_XMUL (*Real*) Multiplicity of the blip grid in a blip-generation calculation (i.e., the amount by which the blip grid is finer in each spatial dimension than the corresponding grid of **G** vectors in the `pwfn.data` file). See Sec. **??** for more information. The default value of 2.0 is often appropriate.

BLIP_PERIODICITY (*Integer*) Orbitals expanded in a blip basis can be periodic in zero, one, two or three dimensions. **blip_periodicity** specifies the number of dimensions in which the orbitals are periodic. Note that if **blip_periodicity** is 1 then the system is assumed to be periodic in the x direction, while if **blip_periodicity** is 2 then the system is periodic in the (x, y) plane. In all cases, the simulation cell is the parallelepiped defined by the lattice vectors placed at the origin. 'Lattice vectors' in nonperiodic directions should be orthogonal to lattice vectors in periodic directions. Note that **k** points may only be used in periodic directions. See Sec. 9.

BLOCK_TIME (*Physical*) If **block_time** is set to a value greater than 0.0, then the number of blocks of moves implied by the value of **vmc_nblock**, **dmc_equil_nblock**, or **dmc_stats_nblock** will be ignored. Instead, CASINO will do all of the things it normally does at the end of a block approximately every **block_time** minutes of CPU time.

For VMC, the actions performed after a block are: (1) write data to the `out` file, `vmc.hist` file, and possibly the `expval.data` file; (2) write the current VMC state plus any accumulated configurations to the `config.out` file (this latter only if the **checkpoint** input keyword is increased to 2 from its default value of 1—otherwise `config.out` is only written after the end of the final block).

For DMC, the actions performed after a block are: (1) write data to `out`, `dmc.hist`, and possible `expval.data` (the latter not during equilibration); (2) make a backup copy of the `config.out/expval.data` file (if catastrophe protection is turned on with the **dmc_trip_weight** keyword); (3) Write the `dmc.status` file (except after the final block); (4) Write the current state of the system, and all configurations in the current population to the `config.out` file (note that by setting the **checkpoint** keyword to 0, this step can be skipped until the end of the final block, or skipped completely if **checkpoint**=-1, but this is not the default).

Note that the above actions can take a significant amount of time (especially if they involve writing to disk) so it is better to do them as infrequently as possible (i.e., large value of **block_time**). Obviously if the stopping criterion (number of moves, target error bar, etc.) implies that the run will stop before **block_time** minutes have elapsed, then the total run time can be shorter than **block_time**.

Note that using **block_time** implies that multiple repetitions of the same run will not necessarily lead to the same answer in parallel calculations (since the number of runs done in **block_time** seconds is defined by the master and what happens on the other slaves can mess around with their random number sequences in an unpredictable way).

Note finally that **block_time** is a physical parameter with dimensions of time, and the units must be specified as, e.g., 1 day, 24 hr, 1440 min, or 86400 s.

BSMOOTH (*Logical*) If **bsmooth** is set to `T` then localized orbitals are interpolated smoothly to zero beyond their cutoff radius. Otherwise, they are truncated abruptly. See Sec. 27 for more information. It is recommended that **bsmooth** be set to `F`, which is the default.

CEREFDMC (*Real*) Constant used in updating the reference energy in the DMC algorithm. See Sec. 13.4.

CHECK_ADERIVS (*Logical*) Check that analytical derivatives of the wave function and its gradient and Laplacian with respect to wave function parameters are coded correctly, by comparing with numerical derivatives.

CHECKPOINT (*Integer*) This integer-valued keyword determines how much CASINO should worry about saving checkpoint data to `config.out` files (which can take a not insignificant amount of time, especially with large systems done on many cores, and can reduce the parallel efficiency—since the slower blocking redistribution algorithm must be used at the end of every block when we write out a config file). **checkpoint** can take four values:
'2': save data after every block in both VMC and DMC, and save the state of the random number generator in OPT runs.
'1' [default]: as '2', but save data in VMC only after the last block when **runtype** = `vmc_opt`, `opt_vmc` or `vmc_dmc` (still after every block if **runtype**=`vmc`).
'0': only save data at the end of the run, for continuation purposes. This is safe only if used in conjunction with the **max_cpu_time** or **max_real_time** keywords (since then the `config.out` file will be automatically written if CASINO sees the job is about to run into an imposed time limit, even if we have not completed the full number of requested blocks).
'-1': do not write **config.out** file at all, ever (DMC only). Note this value should be chosen only if you *know* that the job will fit in any imposed time limit and that such a run will be long enough to give an acceptably small error bar, since it will be impossible to subsequently continue the run.
Note **checkpoint**$= 0$ or -1 clashes with the DMC catastrophe-recovery facility, for which each DMC block needs to be checkpointed. The value of **checkpoint** is thus set to 1 regardless of the input value if **dmc_trip_weight** > 0.

CHECKPOINT_NCPU (*Integer*) This keyword can be used to specify how to group CPUs for reading `config.in` checkpoint files. Having many CPUs access the same file at the same time is not a good idea; therefore we form groups of **checkpoint_ncpu** MPI processes in which only one of them accesses data. The default value is the total number of MPI processes (**nprocs** internally), but depending on the hardware you run on you may want to set **checkpoint_ncpu** to a different value (between 1 and **nprocs**). Note that in the case that **nprocs** is not exactly divisible by **checkpoint_ncpu**, then the remainder will be distributed over the existing groups, and some of the groups will therefore contain **checkpoint_ncpu+1** MPI processes.

CHECKWFN (*Logical*) Enable a numerical check of the analytic orbital derivatives coded in the various routines such as `gauss_per/gauss_mol/bwfdet/pwfdet`, etc. This is primarily intended to help developers ensure that they have coded up the analytical derivatives of new forms of wave function correctly.

COMPLEX_WF (*Logical*) If **complex_wf** is set to `T` then a complex Slater wave function will be used. The orbital-evaluation routines will not attempt to construct real orbitals by forming linear combinations of complex orbitals. It is necessary to set **complex_wf** to `T` in periodic calculations if twisted boundary conditions are to be applied or if twist averaging is to be performed. Note that complex arithmetic is somewhat slower than real arithmetic and hence **complex_wf** should be set to `F` wherever possible. Information about the use of twisted boundary conditions can be found in Sec. 28.

CON_LOC (*Character*) The `config.out` and `config.in` configuration-data files are written to and read from the directory specified by **con_loc**. By default **con_loc** is the directory in which CASINO is run.

COND_FRACTION (*Logical*) If **cond_fraction** is set to `T`, then an improved estimator of the spherically averaged two-particle density matrix, from which one-body contributions are subtracted, will be computed. This is currently only available if the system is homogeneous. See Sec. 35.

COND_FRACTION_MOM (*Logical*) If **cond_fraction** is set to `T`, then an improved estimator of the Fourier transform of the two-particle density matrix, from which one-body contributions are subtracted, will be computed. This is currently only available if the system is homogeneous. See Sec. 35.

CONTACT_DEN (*Logical*) If this flag is set then CASINO will accumulate the spatial overlap between electrons and a positron. At present this is only relevant in studies of positronic molecules.

CONV_BINARY_BLIPS (*Logical*) In November 2011, a new format binary blip file `bwfn.data.bin` was introduced, which is now written out by default when CASINO reads in formatted `bwfn.data` files. The previous binary format—`bwfn.data.b1`—is still supported, not least because at the time of the introduction of the new format, DFT codes such as PWSCF still produced the old-format `b1` file natively (without the intermediate formatted file ever having existed). By default, `b1` files are treated exactly as `bin` files. If the value of **conv_binary_blips** is set to `T`, then after reading in a `b1` file, the data will be converted and written out as `bwfn.data.bin` and the old `b1` file will be deleted, prior to continuing the calculation as normal. This can save disk space since `bin` files are generally smaller than `b1` files and they can be read in somewhat faster (which is advantageous if the same `bin` file is to be used in multiple calculations). The `bin` files are also more portable. Default value is `F`.

CUSP_CORRECTION (*Logical*) When expanded in a basis set of Gaussian functions, the electron–nucleus cusp that should be present in all-electron calculations is not represented correctly. However, when the **cusp_correction** flag is activated, the s-type Gaussian basis functions centred on each atom are replaced within a small sphere by a function which ensures that the electron–nucleus cusp condition is obeyed. This procedure greatly reduces fluctuations in the local energy in all-electron Gaussian calculations. See Sec. 16 for more details.

CUSP_INFO (*Logical*) If **cusp_correction** is set to `T` for an all-electron Gaussian basis set calculation, then CASINO will alter the orbitals inside a small radius around each nucleus in such a way that they obey the electron–nucleus cusp condition. If **cusp_info** is set to `T` then information about precisely how this is done will be printed to the `out` file. Be aware that in large systems this may produce a lot of output. Furthermore, if you create a file called `orbitals.in` containing an integer triplet specifying which orbital/ion/spin you want, the code will print graphs of the specified orbital, radial gradient, Laplacian and 'one-electron local energy' to the files `orbitals.dat`, `gradients.dat`, `laplacians.dat` and `local_energy.dat`. These graphs may be viewed using XMGRACE or similar plotting programs. See Sec. 16 for more details.

CUSP_THRESHOLD (*Real*) If the magnitude of the s component of a Gaussian orbital is less than this threshold then it will not be cusp corrected. See Sec. 16.

CUSTOM_SPAIR_DEP (*Block*) This input block can be used to create new spin-pair groupings for the Jastrow factor, etc. For example, if one were studying a paramagnetic fluid bilayer, with spin-up and spin-down electrons in one plane (spins 1 and 2) and spin-up and spin-down electrons on the other plane (spins 3 and 4) then one would want sets of $u(r_{ij})$ terms for same-plane, same-spin pairs, same-plane, opposite-spin pairs and opposite-plane pairs. Here is an example:

```
%block custom_spair_dep
  no_spair_deps 1    # Number of custom spin dependences
  spair_dep -1 3     #Label (-1,-2,-3,...) and number of spin groups
  1-1,2-2,3-3,4-4
  1-2,3-4
  1-3,1-4,2-3,2-4
%endblock custom_spair_dep
```

To use this spin-pair dependence, the 'spin-dependence' flag in e.g. the u term of the Jastrow factor in `correlation.data` would be set to '-1'. All spin-pairs must be included in a group. See also **custom_ssingle_dep**.

CUSTOM_SSINGLE_DEP (*Block*) This input block can be used to create new spin-single groupings for the Jastrow factor, etc. Here is an example:

```
%block custom_ssingle_dep
  no_ssingle_deps 1    # Number of custom spin dependences
  ssingle_dep -1 2     # Label (-1,-2,-3,...) and number of spin groups
  1,2
  3,4
%endblock custom_ssingle_dep
```

To use this spin-single dependence, the 'spin-dependence' flag in e.g. the χ term of the Jastrow factor in `correlation.data` would be set to '-1'. All spins must be included in a group. See also **custom_spair_dep**.

CUSTOM_STRIPLET_DEP (*Block*) This input block can be used to create new spin-triplet groupings for the Jastrow factor, etc. Since their is no automated generation of spin-triplets, this block is necessary whenever using a Jastrow H term. Here is an example:

```
%block custom_striplet_dep
  no_striplet_deps 1
  striplet_dep -1 3
  1=1=3,1=1=4,2=2=3,2=2=4,1=3=3,1=4=4,2=3=3,2=4=4
  1=1=1,1=1=2,1=2=2,2=2=2,3=3=3,3=3=4,3=4=4,4=4=4
  1=2-3,1=2-4,1-3=4,2-3=4
%endblock custom_striplet_dep
```

DBARRC (*Integer*) **dbarrc** is the number of moves between full recalculation of the cofactor (='DBAR') matrices. Basically every time an electron move is accepted in equilibration/VMC/DMC, the `update_dbar` routine is called which updates these matrices using the efficient Eq. (26) of Ref. [16]. As a numerical precaution that the (unstable!) update procedure is working, every **dbarrc** accepted moves the DBAR matrices and determinant are recomputed from scratch from the orbitals in the Slater matrix. If the new DBAR differs by too much from the old updated DBAR, then the program ought to be stopped (but in fact isn't, since it happens at least once per simulation and is very irritating). In principle one can boost the value of **dbarrc** up to a fairly large number before this happens, and this is a good idea since reevaluation of the matrix costs quite a lot. The default value is 100,000. Tests show that for systems with 1024 particles per spin channel it may be safe to do up to 1,000,000 updates with accuracy better than single precision. See Sec. 18. NOTE: *This keyword had a different meaning earlier in the life of* CASINO *where it was appropriate to set it to a value of, e.g., 10. Re-using an old input file with such a setting now can cause the code to slow down by 1–2 orders of magnitude without the user necessarily understanding why (real world examples have been observed of people doing this). It is therefore now forbidden to set DBARRC to a value lower than the default; if the user has a genuine reason for wishing to do this he/she may search for the error trap in the source code and comment it out.*

DENSITY (*Logical*) If **density** is set to `T` then the charge density is accumulated and written to the `expval.data` file. See Sec. 35.

DIPOLE_MOMENT (*Logical*) If this flag is set to `T` then CASINO will accumulate the expectation value of the electric dipole moment $\mathbf{p}$. It will also evaluate the expectation value of $|\mathbf{p}|^2$. The data $(\mathbf{p}_x, \mathbf{p}_y, \mathbf{p}_z)$ and $|\mathbf{p}|^2$ are written to `vmc.hist` or `dmc.hist` like energy components, rather than into `expval.data`, and their value and error bars are determined by reblocking. The dipole moment is well-defined in aperiodic systems, or in aperiodic directions in 1D- and 2D-periodic systems.

Note that the CASINO `reblock` utility reports only the components and not the magnitude of the dipole moment in order to allow the user to decide how to deal with the symmetry. Suppose that symmetry dictates the dipole moment will point in the x direction. The y and z components should be zero, but there will be some noise when they are evaluated in QMC. If you work out $|\mathbf{p}| = \sqrt{|\mathbf{p}_x|^2 + |\mathbf{p}_y|^2 + |\mathbf{p}_z|^2}$ then you will get something larger than $|\mathbf{p}_x|$, tending to $|\mathbf{p}_x|$ in the limit of perfect sampling (i.e., a biased estimate with finite sampling). You will also get larger error bars on $|\mathbf{p}|$ than $|\mathbf{p}|_x$. See Sec. 35.8 for more details and for an example see J. Chem. Phys. **127**, 124306 (2007).

DMC_AVE_PERIOD (*Integer*) Number of consecutive local energies that are averaged together in DMC before writing them to the `dmc.hist` file. The only effect of this keyword is reduce the number of lines in `dmc.hist` by a factor of 1/**dmc_ave_period**. Note that if **dmc_equil_nstep** or **dmc_state_nstep** are not divisible by **dmc_ave_period**, they will be rounded up to the nearest integer multiple of it.

DMC_DECORR_PERIOD (*Integer*) Length of the inner decorrelation loop in DMC. The algorithm will perform **dmc_decorr_period** configuration moves between successive evaluations of expectation values other than the energy. Setting **dmc_decorr_period** to a value greater than 1 should reduce the serial correlation of the data. Notice that **dmc_decorr_period** differs from its VMC counterpart in that in DMC local energies are calculated at intermediate steps (they must), and these additional values are averaged into the energy data. Therefore, for calculations which do not require expectation values other than the energy, changing **dmc_decorr_period** from 1 to some value x is equivalent to setting multiplying both **dmc_equil/stats_nstep** and **dmc_ave_period** by x. In a preliminary DMC calculation, **dmc_decorr_period** specifies the frequency with which configurations are written out.

DMC_DTEFF_METHOD (*Integer*) Method used to evaluate the DMC effective time step (see Sec. 13.5). Possible values: 0 (effective time step is the same as the actual time step); 1 (the mean squared diffusive distance for accepted moves is evaluated as the weighted average of squared diffusive distances for proposed moves); and 2 (the mean squared diffusive distance for accepted moves is evaluated as the unweighted mean of the squared diffusive distances for accepted moves). We recommend the default value of 1.

DMC_EQUIL_FIXPOP (*Real*) If VMC and DMC energy are too different, the population increases during the initial phase of equilibration before the reference energy can counteract. This parameter (between 0.0 and 1.0) specifies an initial fraction of the equilibration phase during which the population and total weight are fixed to the target weight. Setting this parameter to e.g. 0.5 will prevent such explosions and should have negligible effect on equilibration time.

DMC_EQUIL_NBLOCK (*Integer*) Number of blocks into which the DMC equilibration phase is divided (if **dmc_equil_nstep** is not divisible by **dmc_equil_nblock**, then the number of steps will be increased to the nearest multiple of the number of blocks). Note that having multiple blocks does not increase the amount of data collected, merely the frequency with which data is written to files; the final answer should be essentially the same, irrespective of the number of blocks. Specifically, at the end of each equilibration block, the following significant actions are performed:
(1) Write MPI process- and config-averaged data to `dmc.hist` (one line for each step in the current block).
(2) Print monitoring data to the `out` file (block-averaged quantities).
(3) Make a backup copy of the `config.out` file (if catastrophe protection is turned on with the **dmc_trip_weight** keyword).
(4) Write the `dmc.status` file.
(5) Write the current state of the system, and all configurations in the current population to the `config.out` file (note that by setting the **checkpoint** keyword to 0, this step can be skipped until the end of the final block, or skipped completely if **checkpoint**=-1, but this is not the default).
Note that if accumulating expectation values other than the energy, data is not written to the `expval.data` file after each block, as it would be during the statistics accumulation phase. Also, having too many blocks will make the code slower, and if the run is not massively long it is perfectly in order to have only one DMC equil block (which is the default).

DMC_EQUIL_NSTEP (*Integer*) Number of DMC steps performed on each MPI process in the DMC equilibration phase, and consequently, the total number of local energy samples (averaged over configurations and MPI processes) written to the `dmc.hist` file. The equilibration phase may be partitioned into **dmc_equil_nblock** blocks, but this does not affect the total number of steps (just how frequently stuff is written out). However, if **dmc_equil_nstep** is not divisible by the number of blocks, then it will be rounded up to the nearest multiple of **dmc_equil_nblock**. Furthermore, **dmc_ave_period** consecutive local energies may be averaged together in DMC before writing them to the `dmc.hist` file (hence reducing its size), but again, if **dmc_equil_nstep** is not divisible by **dmc_ave_period**, it will be rounded up to the nearest multiple of it. Note the difference in parallel behaviour compared to **vmc_nstep**, which is not a per-process quantity; this is because the DMC phase is parallelized over configurations.

DMC_EREF_METHOD (*Integer*) **dmc_eref_method** selects the method used to evaluate the DMC reference energy E_{T}. Possible values are: (1) the algorithm in Ref. [17] (default); (2) like 1, but with the mixed estimate of the energy replaced by the growth estimator; (3) like 1, but with the mixed estimate of the energy replaced by the mean limited local energy. We recommend you use the default value **dmc_eref_method**=1.

DMC_INIT_EREF (*Physical*) If set, **dmc_init_eref** defines the initial reference energy for a DMC calculation. If unset, the VMC energy is used instead (default). This keyword is ignored if the initial configurations come from DMC, in which case the previous DMC best estimate of the energy is used instead.

DMC_LOCAL_DUMP (*Logical*) Setting **dmc_local_dump** causes the local energies and force components for each configuration to be written to disk. Each MPI process writes its own dump, `dmc_local_dump_`*iproc*`.dat`, containing configuration weights and relevant local values obtained every **dmc_decorr_period**th step.

DMCMD_EQUIL_NSTEP (*Integer*) Total number of DMC steps performed in the DMC equilibration stage when we are doing a non-initial step in a DMC molecular dynamics calculation (we already have a quasi-converged wave function for a slightly different nuclear configuration). The number of blocks is assumed to be 1.

DMCMD_STATS_NSTEP (*Integer*) Total number of DMC steps performed in the DMC stats accumulation stage when we are doing a non-initial step in a DMC molecular dynamics calculation (we already have a quasi-converged wave function for a slightly different nuclear configuration). The number of blocks is assumed to be 1.

DMC_MD (*Logical*) If DMC_MD is `T` then in a DMC calculation we assume we are doing molecular dynamics and that we are restarting from a converged wave function for a slightly different nuclear configuration. In practice, all this means is that the number of steps performed are given by **dmcmd_equil_nstep** and **dmcmd_stats_nstep**, rather than **dmc_equil_nstep/dmc_stats_nstep** (the number of blocks is assumed to be 1 in the MD case, and the value of **block_time** is ignored). The number of moves necessary will be greatly reduced from the normal case. See also **dmc_reweight_conf** and **dmc_spacewarping**. The necessary manipulations are automated by the `runqmcmd` script.

DMC_METHOD (*Integer*) **dmc_method** selects which version of DMC to use: (1) the particle-by-particle algorithm; (2) the configuration-by-configuration algorithm. Method 1 is the default. The DMC algorithm is discussed at length in Sec. 13.

DMC_NCONF_PRELIM (*Integer*) This is the approximate total number of configurations (summed over all MPI processes) to generate in a preliminary DMC calculation (for reducing the cost of equilibration). See Sec. 40.3.6.

DMC_NORM_CONSERVE (*Logical*) Use the norm-conserving DMC algorithm [18]. This eliminates fluctuations in the total population. Experimental algorithm: use with caution.

DMC_NTWIST (*Integer*) Number of random 'twists' or offsets to the grid of **k** vectors to be sampled during DMC statistics accumulation. If **dmc_ntwist** is 0 then the twist angle is not changed during DMC. After each change of twist angle, the set of configurations needs to be re-equilibrated: hence a value needs to be specified for **dmc_reequil_nstep**. Setting **dmc_ntwist** greater than 0 requires the use of a complex wave function (**complex_wf** : `T`). Note that the usual keywords define the run length for a single twist angle, thus the run length is increased by a factor of **dmc_ntwist**. DMC twist averaging can only be carried out within CASINO for electron(–hole) fluid phases at present (for real systems containing atoms, see the utilities in the `CASINO/utils/twist` directory). See Sec. 28.

DMC_POPRENORM (*Logical*) Control the DMC configuration population by randomly deleting or copying configurations after branching with the reference energy set equal to the best estimate of the ground-state energy. This can be used to maintain a constant population of configurations per MPI process, provided the value of **dmc_target_weight** is an integer multiple of the number of MPI processes. Note that non-integer values of **dmc_target_weight** are not allowed when using **dmc_poprenorm**. Note also that **dmc_poprenorm** is not in general recommended because of the population control errors it can theoretically introduce, though in general these are likely to be small.

DMC_REEQUIL_NBLOCK (*Integer*) Number of blocks into which the total re-equilibration run-length is divided when doing a twist-averaged DMC run. Currently, a re-equilibration only takes place when the twist angle is changed. DMC twist averaging can only be carried out within CASINO for electron(–hole) fluid phases at present (for real systems containing atoms, see the `twistav_xxx` utilities in the `CASINO/utils/twist` directory). See Sec. 28.

DMC_REEQUIL_NSTEP (*Integer*) Total number of steps performed in the re-equilibration stage when doing a twist-averaged DMC run. A re-equilibration only takes place when the twist angle is changed. Notice that this number will be rounded up to the nearest multiple of **dmc_reequil_nblock** times **dmc_ave_period**. DMC twist averaging can only be carried out within CASINO for electron(–hole) fluid phases at present (for real systems containing atoms, see the `twistav_xxx` utilities in the `CASINO/utils/twist` directory). See Sec. 28.

DMC_REWEIGHT_CONF (*Logical*) Update walker weights read in from config.in. Weights of walkers are recomputed after reading `config.in` to correct for a modified wave function. This allows continuous QMC-MD computations as described in PhysRevLett.94.056403.

DMC_SPACEWARPING (*Logical*) Electronic positions are adjusted to follow the ionic positions when adapting an existing population to a new wave function. The method follows the description in Phys. Rev. B **61**, R16291 and is typically combined with **dmc_reweight_conf**.

DMC_STATS_NBLOCK (*Integer*) Number of blocks into which the DMC statistics accumulation phase is divided (if **dmc_stats_nstep** is not divisible by **dmc_stats_nblock**, then the number of steps will be increased to the nearest multiple of the number of blocks). Note that having multiple blocks does not increase the amount of data collected, merely the frequency with which data is written to files; the final answer should be the same, irrespective of the number of blocks. Specifically, at the end of each accumulation block, the following significant actions are performed:
(1) Write MPI process- and config-averaged data to `dmc.hist` (one line for each step in the current block).
(2) Write MPI process- and config-averaged data to the expval.data file (if accumulating expectation values other than the energy).
(3) Print monitoring data to the `out` file (block-averaged quantities).
(4) Make a backup copy of the `config.out` file (if catastrophe protection is turned on with the **dmc_trip_weight** keyword).
(5) Make a backup copy of the `expval.data` file (if it exists, and if catastrophe protection is turned on with the **dmc_trip_weight** keyword).
(6) Write the `dmc.status` file.
(7) Write the current state of the system, and all configurations in the current population to the `config.out` file (note that by setting the **checkpoint** keyword to 0, this step can be skipped until the end of the final block, or skipped completely if **checkpoint**=-1, but this is not the default).
Note that having too many blocks will make the code slower, and if the run is not massively long it is perfectly in order to have only one DMC stats block (which is the default).

DMC_STATS_NSTEP (*Integer*) Number of DMC steps performed on each MPI process in the statistics accumulation phase, and consequently, the total number of local energy samples (averaged over configurations and processes) written to the `dmc.hist` file. The accumulation phase may be partitioned into **dmc_stats_nblock** blocks, but this does not affect the total number of steps (just how frequently stuff is written out). However, if **dmc_stats_nstep** is not divisible by the number of blocks, then it will be rounded up to the nearest multiple of **dmc_stats_nblock**. Furthermore, **dmc_ave_period** consecutive local energies may be averaged together in DMC before writing them to the `dmc.hist` file (hence reducing its size), but again, if **dmc_stats_nstep** is not divisible by **dmc_ave_period**, it will be rounded up to the nearest multiple of it. Note the difference in parallel behaviour compared to **vmc_nstep**, which is not a per-process quantity; this is because the DMC phase is parallelized over configurations.

DMC_TARGET_WEIGHT (*Real*) Total target weight in DMC, summed over all MPI processes. This is synonymous with the 'target population' of configurations, except that **dmc_target_weight** is allowed to be non-integer. Typically **dmc_target_weight** will be the same as **vmc_nconfig_write** when **runtype=** `vmc_dmc`, though it does not have to be.[9]

[9]It may seem bizarre to allow non-integer total target weights, but a possible use for this is in increasing the parallel efficiency when you have a very small population per MPI process. Suppose your target weight is 1 configuration per process and you are running on 100000 processes. Half the time your total population will be a bit higher than 100000, and when this happens nearly all of your processes will spend half their time twiddling their thumbs waiting for the small number of processes that have two configurations to finish the iteration. So around 25% of the computer time is wasted. If instead you set your target weight to 0.98 configurations per process then it is very unlikely that any processes will have two configurations. Instead you have on average 2% of your processes sitting idle, which is sad, but still more efficient than having large numbers of processes wait for a small number of over-burdened processes.

DMC_TRIP_WEIGHT (*Real*) In the course of a DMC simulation, it is possible for a configuration 'population explosion' to occur. If **dmc_trip_weight** is set to 0.0 then nothing will be done about this. If **dmc_trip_weight**$>$ 0 then the algorithm will attempt to restart the block (with a different random number sequence) if the iteration weight exceeds **dmc_trip_weight**. A general suggestion for its value would be three times **dmc_target_weight** (but see the discussion later in the manual in Sec. 13.9 about this).

DTDMC (*Real*) Time step for DMC run (atomic units). The DMC time step must be small, as the DMC Green's function is only exact in the limit of zero time step: see Sec. 13. Typically the DMC time step is about two orders of magnitude smaller than the VMC time step, and the DMC move-acceptance ratio should be about 99.9%. For accurate work, one must always investigate time-step bias by plotting the DMC energy against the value of **dtvmc**. Provided the time step is sufficiently small that the root-mean-square distance diffused by each particle at each step is much less than the shortest physically relevant length scale, one can expect to find the time-step bias in the DMC energy to be linear; hence it is straightforward to extrapolate to zero time step. The EXTRAPOLATE_TAU utility and MAKE_E_V_DT script exist to help with the extrapolation to zero time step.

DTVMC (*Real*) Time step for VMC run (atomic units). The form of the VMC transition-probability density used by CASINO is discussed in Sec. 12. As described in Sec. 12.4, **dtvmc** should be chosen so that the overall move acceptance ratio is close to 50%. In most normal systems, the appropriate value is between 0.1 a.u. and 0.6 a.u. If **dtvmc** is given a sensible starting value (and, for normal systems, anything in this range is sensible), setting **opt_dtvmc** to 1 will cause the time step to be optimized automatically. For very low-density systems, a much larger value of **dtvmc** is appropriate.

DTVMCS (*Block*) Use this keyword to specify a VMC time step for each particle family explicitly, as well as to determine whether to optimize each of them individually. The contents of this block override the values of **dtvmc** and **opt_dtvmc**. One line is to be written for each 'family' of particles, the format of each line being:
DTVMC OPT_DTVMC
where 'DTVMC' is the value of the time step, and 'OPT_DTVMC' can be 0 or 1, indicating whether to optimize the corresponding time step or not.

DTVMC_SHIFT (*Real*) **dtvmc_shift** is an optional shift in the VMC transition probability, which can be used to 'encourage' electrons to be more mobile. **dtvmc_shift** is expressed in units of the square root of **dtvmc**.

E_OFFSET (*Physical*) This keyword gives a constant shift $E_{\rm offset}$ in the total energy per electron such that the final result will be $E = E_{\rm calc} - E_{\rm offset}$. The default is zero. This allows the user to add any constant contributions to the total energy that are not calculated within CASINO.

EBEST_AV_WINDOW (*Integer*) Averaging window for calculating the ground-state energy during equilibration. During DMC equilibration the best estimate of the ground-state energy is taken to be the average local energy over the last **ebest_av_window** moves. The default of 25 is usually sufficient.

EDIST_BY_ION, EDIST_BY_IONTYPE (*Block*) The **edist_by_ion** block allows fine control of the initial distribution of the electrons before equilibration starts. The standard algorithm shares out the electrons amongst the various ions weighted by the pseudo-charge/atomic number of the ion. Each electron is placed randomly on the surface of a sphere surrounding its parent ion. There are certain situations, for example a simple crystal with a very large lattice constant, where the standard algorithm in the `points` routine may give a bad initial distribution, which cannot be undone by equilibrating for a reasonable amount of time. This keyword allows a user-defined set of electron/ion associations to be supplied. The syntax is to supply $N_{\rm ion}$ lines within the block which look like, e.g., 1 4 4, where the three numbers are: the ion sequence number; the number of up-spin electrons associated with this ion; the number of down-spin electrons associated with this ion. Alternatively one may use the **edist_by_iontype** keyword block, where you replace the ion sequence number with the ion type sequence number and the information is supplied only for each particular type of ion.

EMIN_MIN_ENERGY (*Physical*) This keyword sets a minimum energy threshold for energy minimization, used to reject low-quality wave functions which produce spurious low VMC energy estimates. The value of **emin_min_energy** should (ideally) be set slightly below the ground-state energy. The ground-state energy is often not known, in which case a good estimate can sometimes be supplied. If this keyword is not set manually, CASINO will supply an automatic guess derived from the preceding VMC run. See Sec. 25.3.7 for more details.

EMIN_XI_VALUE (*Real*) This keyword sets the value of the ξ parameter used to control semi-orthogonalization in energy minimization. It should rarely be changed by users. See Sec. 25.3 for details.

ESUPERCELL (*Logical*) By default total energies and their components in periodic systems are printed as energies per primitive cell. Switching this flag to `T` forces printing of energies per simulation cell in the output file.

EWALD_CHECK (*Logical*) CASINO and the wave-function generating program should be able to calculate the same value for the nuclear repulsion energy, given the same crystal structure. By default CASINO computes the Ewald interaction and compares it with the value given in the wave-function file. If they differ by more than 10^{-5}, then CASINO will stop and complain. If you have a justifiable reason for doing so (e.g., you have turned off periodicity), you may turn off this check by setting **ewald_check** to `F`.

EWALD_CONTROL (*Real*) This is the percentage increase (from the default) of the cutoff radius for the reciprocal space sum in the Ewald interaction, which is used for calculating electrostatic interactions between particles in periodic systems. Its default value is zero. Increasing **ewald_control** will cause more vectors to be included in the sum, the effect of which is to increase the range of the Ewald γ parameter over which the energy is constant (the default γ should lie somewhere in the middle of this range). This need only be done in exceptional circumstances and the default should be fine for the general user. See Sec. 19.4 for information about the Ewald method.

EXPOT (*Logical*) If **expot** is set to `T` then an external potential is read from the file `expot.data` and included as a summed contribution to the total energy. See Sec. 7.11.

EXPVAL_CUTOFF (*Physical*) **expval_cutoff** is the energy cutoff for G-vectors used in the evaluation of expectation values accumulated in reciprocal space (e.g., the density, spin density, pair-correlation function, etc.). The value of **expval_cutoff** is ignored if an `expval.data` file is already present, in which case the G-vector set(s) given therein are used instead. If you set it to zero, then the program will suggest a value. The default is 3 a.u. See Sec. 7.13.

EXPVAL_ERROR_BARS (*Logical*) If this flag is set, CASINO will, where practicable, accumulate the additional quantities required to evaluate error bars on requested expectation values. This will increase the size of the `expval.data` file and slow down the calculation slightly. At present this functionality is limited to the structure factor.

EXPVAL_KGRID (*Block*) This block contains a specification of one or more **k**-point grids defined in one, two or three dimensions. A one-dimensional grid is defined by a line AB, a two-dimensional grid by a plane AB–AC and a three-dimensional grid by a parallelepiped AB–AC–AD, together with an appropriate number of **k**-points along each direction. These grids may be used in the calculation of various expectation values, if the appropriate keywords are set to `T` in `input`. The block consists of the following lines:
Line 1: Number of grids defined in this block;
Line 2: Which expectation value uses this grid? (2 for spherical structure factor);
Line 3: Dimensionality d of current **k**-grid (1–3);
Line 4: Coordinates of **k**-point A (a.u.);
Line 5: Coordinates of **k**-point B (a.u.), number of points along AB;
Line 6: [If $d = 2$ or 3] Coordinates of **k**-point C (a.u.), number of points along AC;
Line 7: [If $d = 3$] Coordinates of **k**-point D (a.u.), number of points along AD.
Repeat lines 2 to 7 for each additional grid.
Note that spherically averaged quantities require a one-dimensional grid, irrespective of the dimensionality of the system.

FINITE_SIZE_CORR (*Logical*) Calculate finite-size corrections to the kinetic energy and the electron–electron interaction energy in periodic systems using the method described in Ref. [19] and Ref. [15]. See Secs. 29 and 30 for further information.

FIX_HOLES (*Logical*) This keyword is used to define the reference points for the exciton-exciton separation when using the 'BIEX3' wave function. Setting **fix_holes** to `T` means that the two holes are fixed at a distance `xx_sep` apart. The default is `F`, in which case the centres of mass of the two excitons are fixed instead. If BIEX3 is not being used then this keyword is ignored.

FIXED_PARTICLES (*Block*) When setting up a model system one can place fixed, charged particles within the simulation cell by using the **fixed_particles** block. This can be used to study, e.g., electron–hole complexes in the presence of fixed donor and acceptor ions. The block consists of one line for each fixed particle, where the lines are of the form '⟨charge⟩ $\langle x\rangle$ $\langle y\rangle$ $\langle z\rangle$', where x, y and z are the Cartesian components of the fixed charge's position. The charge must be an integer. Periodic repeats of the fixed particle are generated automatically.

FORCES (*Logical*) Calculate atomic forces in VMC/DMC. Forces are only implemented for Gaussian basis sets and only work in pseudopotential calculations (in order to eliminate the electron–nucleus singularity).

FORCES_INFO (*Integer*) Controls the amount of information calculated/displayed during force calculations:
'2': display no additional information; the Hellmann–Feynman force is evaluated with the d-channel of the pseudopotential chosen to be local and the s-d and p-d channels nonlocal (default);
'5': calculate and display two additional Hellmann–Feynman force estimators, where the s- and p-channels of the pseudopotential components are chosen to be local.

FREE_PARTICLES (*Block*) This block sets the parameters that define the behaviour of the orbitals which are not atom-related in a system. The geometry of the system can be given using 'r_s $\langle r_\mathrm{s}\rangle$', 'dimensionality $\langle d\rangle$', 'periodicity $\langle P\rangle$' and 'cell_geometry' (followed by d lines with d reals corresponding to the unscaled cell vectors). For 2D or 1D systems one can also specify that the electrons are confined to different layers (wires in 1D) using 'heg_nlayers ⟨no. layers⟩' and 'heg_zlayer ⟨layer⟩ $\langle z\rangle$', with species being assigned to layers using 'heg_layer ⟨spin⟩ ⟨layer⟩'. In 1D, one can also specify the y-coordinate of a wire using 'heg_ylayer ⟨layer⟩ $\langle y\rangle$'. These parameters are only required if **atom_basis_type**='none' (which it is by default) in the `input` file). The number and type of the orbitals can be given using lines with the syntax 'particle $\langle i\rangle$: $\langle n\rangle$ orbitals ⟨orb⟩ [orb-options]' (if all determinants contain the same orbital type) or 'particle $\langle i\rangle$ det ⟨det⟩ : $\langle n\rangle$ orbitals ⟨orb⟩ [orb-options]', where ⟨det⟩ is the term in the multideterminant expansion, $\langle i\rangle$ must be 1, 2 or a number given in the **particles** block (1 and 2 are up- and down-spin electrons), $\langle n\rangle$ is the number of free particles/orbitals belonging to the ⟨det⟩th determinant and '⟨orb⟩ [orb-options]' is one of the following: 'free', 'crystal siteset $\langle s\rangle$', 'harmonic', 'pairing $\langle j\rangle$ [+ $\langle m\rangle$ orbitals free]', 'sdw' or 'expot $\langle set\rangle$', $\langle j\rangle$ being the particle type with which $\langle i\rangle$ is paired, $\langle m\rangle$ is the number of unpaired particles of type $\langle i\rangle$, and $\langle set\rangle$ being an orbital set in `expot.data`. If the orbitals have optimizable parameters, these must be provided in `correlation.data`. Wigner-crystal geometry is specified using the keywords 'crystal_type ⟨type⟩ $\langle n\rangle$ siteset[s] [repeat $\langle r\rangle$]' (type = 'cubic', 'fcc', 'bcc', 'rectangular', 'hexagonal' or 'triangular', which must match 'dimensionality' and 'cell_geometry', or 'manual'), and 'siteset $\langle s\rangle$ [antiferro[magnetic]] offset $\langle x\ y\ z\rangle$' for predefined lattices (in primitive-cell fractional coordinates), and 'siteset $\langle s\rangle$ manual $\langle n\rangle$ site[s]' followed by $\langle n\rangle$ lines of the form $\langle x\ y\ z\rangle$ defining the sites for manual lattices (in primitive-cell fractional coordinates). In nonperiodic directions the site positions and offsets are in absolute rather than fractional coordinates. If a complex wave function is used, i.e., **complex_wf** is set to `T`, then an offset to the grid of **k** vectors for fluid phases may be specified using 'k_offset $\langle k_x\rangle$ $\langle k_y\rangle$ $\langle k_z\rangle$', where k_x, k_y and k_z are the Cartesian components of the offset. Alternatively, the offset can be specified in terms of fractions of supercell reciprocal lattice vectors using 'k_offset_frac $\langle k_1\rangle$ $\langle k_2\rangle$ $\langle k_3\rangle$'. The offset is translated into the first Brillouin zone of the simulation cell. Using a nonzero offset corresponds to using twisted boundary conditions. If you are studying a 1D HEG or a 2D HEG of finite width b, you can specify that width using 'quasi_1D b' and the model of transverse confinement using 'transverse_model m', where $m=1$ softens the Coulomb interaction to $1/\sqrt{r^2+b^2}$, $m=2$ assumes a quadratic confining potential and hence a Gaussian distribution of transverse coordinates with standard deviation b and $m=3$ imposes hard-wall boundary conditions on a cylinder of radius b. You can select whether the finite width is described using an effective 1D interaction that is averaged over transverse motion or by explicit sampling of the ground-state distribution in the transverse direction. The former and latter methods are selected using 'transverse_method 0' and 'transverse_method 1', respectively. It's not quite as difficult to use this input block as it may appear from the above: see the examples in `~/CASINO/examples/electron_phases` and `~/CASINO/examples/electron_hole_phases`. Information about the associated blocks of optimizable parameters in `correlation.data` can be found in Secs. 7.4.9 and 7.4.10.

FUTURE_WALKING (*Logical*) If this flag is set to `T` then future walking will be used to evaluate pure estimators in DMC. See Sec. 37.

GAUTOL (*Real*) Tolerance for Gaussian orbital evaluation. The contribution of a Gaussian is neglected if its value is less than $10^{-\mathbf{gautol}}$.

GROWTH_ESTIMATOR (*Logical*) Turn on calculation of the growth estimator of the total energy in DMC calculations. A statistically significant difference between the mixed estimator and the growth estimator for the energy normally implies the presence of time-step bias. Other than that, the growth estimator is not generally useful, because the statistical error in the growth estimator is substantially greater than the error in the mixed estimator. See Sec. 13.8 for more information.

IBRAN (*Logical*) If set to `T` then weighting and branching is allowed in DMC. Setting **ibran**=`F` may be used to check the DMC algorithm, as it then reduces to a VMC algorithm in which the DMC drift-diffusion Green's function is the transition probability density.

INITIAL_CONFIG (*Block*) Use this keyword if you want to specify the initial VMC configuration to use instead of the random one generated by the `points` routine. It is possible to specify the positions of only some of the particles. The format of each line in this block is:
$\sigma\ i\ x\ y\ z$
where σ is the spin index of the particle, i is the index of the particle within its spin channel and (x, y, z) is the position of the particle.

INPUT_EXAMPLE (*Logical*) If **input_example** is `T` then an example of a CASINO `input` file with all currently known keywords and their default values will be written out. A modified version of this can be used as an `input` file in future runs.

INT_SF (*Logical*) If **int_sf** is set to `T` then the electron–electron interaction energy for a periodic system will be calculated in terms of the structure factor. The structure factor should either have been accumulated in a previous run and stored in an available `expval.data` file, or its accumulation should be flagged for the current run. Using this method the total interaction energy can be separated into Hartree and XC terms. This feature is not currently documented in the manual.

HARTREE_XC (*Logical*) Flag the computation of separate Hartree and exchange-correlation (XC) parts of the electron-electron interaction energy for a periodic system. This may be done in two different ways, namely the structure factor method and the MPC method. The computation thus requires either (1) structure factor information from a previously accumulated expval.data file or from setting **structure_factor**=T, or (2) the MPC interaction to be active (through **interaction**=mpc, mpc_ewald or ewald_mpc). If both these things are true then both methods will be used to compute the hartree/XC energies (the resulting numbers should agree reasonably closely). If neither are true, then this keyword has no effect. The default is T. Note that the MPC version only works with 3D periodicity.

INTERACTION (*Integer*) Type of interaction between particles. **interaction** can take the following values:
'none': noninteracting particles;
'coulomb': Coulomb interaction;
'ewald': periodic Coulomb interaction computed using Ewald summation;
'mpc': periodic Coulomb interaction computed using the MPC method;
'ewald_mpc': compute and report both Ewald and MPC results, but use Ewald in DMC propagation;
'mpc_ewald': compute and report both Ewald and MPC results, but use MPC in DMC propagation;
'manual': compute a user-defined interaction (see the **manual_interaction** block input keyword);
'ewaldpp', 'ewaldpp_mpc', 'mpc_ewaldpp': as their above counterparts, but using an electron-electron pseudopotential for the Ewald interaction, whose parameters [see Eq. (4) of Ref. [20]] must be specified in the **manual_interaction** block;
'ewald_kel': use the "Ewaldized" (i.e., periodic) Keldysh interaction for 2D semiconductors,

which is equal to the usual Coulomb Ewald interaction plus the difference of the finite Keldysh potential and the Coulomb $1/r$ potential summed over periodic images; the required r_* parameter is specified in the **manual_interaction** block.

The values 'coulomb' and 'ewald' can be used interchangeably, although 'coulomb' should strictly refer to aperiodic systems and 'ewald' to periodic systems.

The MPC interaction is generally significantly faster than the Ewald interaction and should give smaller finite-size effects. The MPC interaction is not currently implemented for 1D systems, however. Furthermore, we recommend using 'ewald_mpc' rather than 'mpc' or 'mpc_ewald', as there is some evidence that the MPC interaction can distort the XC hole. See Sec. 19.4 for information about the Ewald interaction, Sec. 19.4.4 for information about the MPC and Sec. 20 for information about 'manual' interactions.

ISOTOPE_MASS (*Real*) This keyword can be used to define a nuclear mass in unified atomic mass units (u) if you need to override the default value used in CASINO (which is averaged over isotopes according to their abundances). The default (given in the table in Sec. 34) is used if **isotope_mass** is set to zero. The unified atomic mass unit (u) in this sense means 'the ratio of the average mass per atom of the element to 1/12 of the mass of ^{12}C'. This is only relevant if **relativistic** is set to `T`. See Sec. 34.

JASBUF (*Logical*) If **jasbuf** is `T` then the one-body (χ and q) terms in the Jastrow factor for each electron in each configuration are buffered in DMC: this saves time at the expense of memory. Clearly this will have no effect in systems without one-body terms in the Jastrow factor.

JASTROW_PLOT (*Block*) This utility allows the user to plot the $u(r_{ij})$, $\chi(r_i)$, $f(r_i, r_j, r_{ij})$, $p(\mathbf{r}_{ij})$, $u_{\rm cyl}(\mathbf{r}_{ij})$, $\chi_{\rm cyl}(\mathbf{r}_i)$ and $q(\mathbf{r}_i)$ terms in the Jastrow factor. The first line is a flag specifying whether the Jastrow factor is to be plotted (0=NO, 1=YES); the second line holds the spin of particle $i = 1, 2, \ldots$; the third line holds the spin of particle $j = 1, 2, \ldots$. Optionally, another three lines may be given: the fourth line holds the (x, y, z)-position of particle j; the fifth line holds a vector with the direction in which i is moved; and the sixth line holds the position vector of a point on the straight line along which electron i moves. If lines 4–6 are not given, default values will be inserted. The nucleus is assumed to lie at the origin. All 6 (or 3) lines must be present, even if only χ is to be plotted: the redundant information about particle j will be ignored. If χ is plotted then `jastrow_value_chi_?.dat`, `jastrow_deriv_chi_?.dat` and `jastrow_sderiv_chi_?.dat` contain the value, derivative and second derivative of $\chi(r_i)$ against r_i for each set of χ terms. Likewise for u. If f is plotted, the `jastrow_value_f_?.dat` files contain the value of f against the distance from the point given in line 6. Likewise for p and q. All terms present in the Jastrow factor in `correlation.data` will be plotted. If wave-function optimization has gone wrong, a common indication is that $u(r_{ij})$ does not increase monotonically to 0. If you encounter unexpected population-control problems in DMC, this is a good test to apply. See Sec. 22.1 for information on CASINO's Jastrow factor.

KE_FORGIVE (*Logical*) CASINO performs numerical tests to determine whether the kinetic energies computed during the run will be correct. If **ke_forgive** is set to `F`, CASINO will regard this as an error and stop. The default is `T`. Note that although the procedure is generally stable, there may be cases in which poor numerics causes failures. See also **ke_verbose**.

KE_VERBOSE (*Logical*) CASINO performs numerical tests to determine whether the kinetic energies computed during the run will be correct. Such tests are carried out after VMC equilibration, and will only produce concise output about the outcome. However, if the flag **ke_verbose** is set to `T`, CASINO will print out information throughout the process. The default is `F`. See also **ke_forgive**.

KWARN (*Logical*) The **kwarn** flag is relevant only in calculations using a plane-wave basis set. If the flag is set to `T`, then CASINO will issue a warning whenever the kinetic energy calculated from the supplied orbitals differs from the DFT kinetic energy given in the `pwfn.data` file by more than an internal tolerance (usually set to 10^{-6}). If the flag is `F`, then CASINO will stop with an error message on detecting this condition. Note that in cases where the DFT calculation which generated the orbitals used fractional occupation numbers, the kinetic energy mismatch is very likely to occur since QMC deals in principle only with integer occupation numbers, hence the existence of this flag. Furthermore, the calculation of the kinetic energy is based on the assumption that the orbitals are orthogonal; hence **kwarn** should be set to `T` if nonorthogonal localized plane-wave orbitals are used.

LCUTOFFTOL (*Real*) This is used to define the cutoff radius for the local part of the pseudopotential. It is the maximum deviation of the local potential from $-Z/r$ at the local cutoff radius. See Sec. 19.3.

LIMDMC (*Integer*) Set modifications to Green's function in DMC (see Sec. 13.5). May take values:
0: no modifications applied;
1: Depasquale *et al.* scheme [21];
2: Umrigar *et al.* scheme [17];
3: Langfelder–Rothstein–Vrbik mods [22];
4: Umrigar mods to drift velocity, Zen–Sorella–Alfè mods to energy [23], with a cut in the branching of $E_{\rm cut} = \alpha\sqrt{N/\tau}$. The parameter α is set by **alphalimit**
5: Experimental Zen–Alfé scheme;
6: Scheme of Umrigar *et al.* applied to the single-electron local energies rather than the total local energy. We recommend a value of 4, which is the default. This scheme must be used if the **nucleus_gf_mods** flag is set to `T`.

LOC_TENSOR (*Logical*) If **loc_tensor** is set to `T` then the localization tensor will be accumulated in the `expval.data` file (periodic systems only). See Sec. 35.

LWDMC (*Logical*) Enable weighted DMC, where each configuration carries a weight that is simply multiplied by the branching factor after each move; only if the weight of a configuration goes outside certain bounds (above **wdmcmax** or below **wdmcmin**) is it allowed to branch or be combined with another configuration. This should reduce excessive population fluctuations, which is generally held to be a good thing. Note that setting **lwdmc=T** means that your population will generally fluctuate around a value other than **dmc_target_weight** (after an initial transient); the chances of being killed if your weight is below 1 or duplicated if your weight is above 1 depend on the values of **wdmcmin** and **wdmcmax**, and in general this is not symmetrical. See Sec. 13.4.

LWDMC_FIXPOP (*Logical*) This flag activates the **lwdmc** variant with fixed population. By interpreting **wdmcmin** and **wdmcmax** relative towards the current population the population and the total weight are decoupled. The population is nearly fixed while the total weight fluctuates as usual. While this generally reduces the statistical efficency of the DMC algorithm, it is a simple way to eliminate population explosions or extinction in cases of small population and large population fluctuation. WARNING: this is *not* a solution for walkers trapped in singular points of the wave functions, nor is it a solution for populations that get trapped in high-energy states. Be careful about this option when you do not know the reason for the population problems in the first place.

MAGNETIC_FIELD (*Logical*) Apply an external magnetic field using the magnetic vector potential specified in `expot.data`. See Sec. 39.

MAKEMOVIE (*Logical*) Plot the particle positions every **movieplot** moves (see Sec. 11).

MANUAL_INTERACTION (*Block*) When **interaction** is set to 'manual', this block is used to specify the form and parameters of the desired interaction. The format is either

```
square_well
Height : -2.0
Width : 3.0
```

or

```
poschl_teller
Mu : 12
V_0 : -1.0
```

or

```
hard_sphere
D : 0.88
```

or

```
polynomial
order : 3
cutoff : 5.3
c_0 : 1
c_1 : 1
c_2 : 1
```

or

```
logarithmic
rstar : 1.0
```

or

```
keldysh
rstar : 1.0
```

or

```
dipole
d^2 : 1.0
```

or

```
tilted_dipole
d^2 : 1.0
theta : 1.57
```

or

```
pseudodipole
order : 3
cutoff : 5.3
c_0 : 1
c_1 : 1
c_2 : 1
d^2 : 1.0
```

or

```
tilted_pseudodipole
order : 3
cutoff : 5.3
c_0 : 1
c_1 : 1
c_2 : 1
d^2 : 1.0
theta : 1.57
```

or

```
ewald_kel
rstar : 1.7858
```

or

```
clifford
```

See Sec. 20 for information on the available interactions and their parameters.

MAX_CPU_TIME (*Physical*) If the CPU time elapsed since the start of a QMC simulation exceeds **max_cpu_time** and a suitable point in the algorithm is reached, then CASINO will halt gracefully. This should make it easier to carry out, e.g., multiple DMC runs on a computer with a queueing system, particularly when used with the `--continue` or `--auto-continue` options of `runqmc`.

The way this works is that at the end of each block of moves, CASINO will check whether doing one more block will exceed the time limit. If so, it will perform an emergency stop, writing to the output file any changes to the input file that must be made in order to restart the job (in a form readable both by humans and by runqmc). The user must therefore define the block length appropriately—most usefully via the **block_time** keyword—such that the time taken per block is a sufficiently small fraction of **max_cpu_time**.

Note that in DMC, if **checkpoint**=0 in input and there is a job time limit, it is strongly recommended that **max_cpu_time** is used to ensure the `config.out` file is written out if the required CPU time is longer than the time limit. **max_cpu_time** is a physical parameter with dimensions of time, and the units must be specified as, e.g., 1 day, 24 hr, 1440 min, or 86400 s. See also the **max_real_time** keyword.

MAX_REAL_TIME (*Physical*) If the wall-clock time elapsed since the start of a QMC simulation exceeds **max_real_time** and a suitable point in the algorithm is reached, then CASINO will halt gracefully. This should make it easier to carry out, e.g., multiple DMC runs on a computer with a queueing system, particularly when used with the `--continue` or `--auto-continue` options of `runqmc`.

The way this works is that at the end of each block of moves, CASINO will check whether doing one more block will exceed the time limit. If so, it will perform an emergency stop, writing to the output file any changes to the input file that must be made in order to restart the job (in a form readable both by humans and by runqmc). The user must therefore define the block length appropriately—most usefully via the **block_time** keyword—such that the time taken per block is a sufficiently small fraction of **max_real_time**.

Note that in DMC, if **checkpoint**=0 in input and there is a job time limit, it is strongly recommended that **max_real_time** is used to ensure the `config.out` file is written out if the required time is longer than the time limit. **max_real_time** is a physical parameter with dimensions of time, and the units must be specified as, e.g., 1 day, 24 hr, 1440 min, or 86400 s. See also the **max_cpu_time** keyword.

MAX_REC_ATTEMPTS (*Integer*) This is the maximum number of times that DMC will attempt to restart a block if it continues to encounter population-explosion catastrophes. Relevant only if the **dmc_trip_weight** keyword is set to a nonzero value. See the discussion in Sec. 13.9 for more details.

MOLGSCREENING (*Logical*) Toggle on and off the use of screening in Gaussian basis set calculations of molecules, i.e., the division of space into boxes and the preparation of lists of which Gaussian basis functions have a significant weight in each box. The use of screening should speed up the calculation of large molecules. The screening information can take up a reasonable amount of memory; hence the existence of this keyword.

MOM_DEN (*Logical*) If set to `T` the momentum density will be accumulated. Exclusively for HEGs at the moment.

MOVIECELLS (*Logical*) If `F` then CASINO will plot the unit cell when making a movie; if `T` then nearest-neighbour cells in the (x, y)-plane will also be written (see Sec. 11).

MOVIEPROC (*Integer*) Plot the particle positions on MPI process **movieproc** (see Sec. 11).

MOVIEPLOT (*Integer*) Plot the particle positions every **movieplot** moves (see Sec. 11).

MPC_CUTOFF (*Physical*) **mpc_cutoff** is the energy cutoff for G-vectors used in (a) the fast Fourier transform (FFT) of the MPC interaction and (b) the FFT of the one-particle density required when generating the `mpc.data` file. The program will suggest a value for **mpc_cutoff** if the existing value is unsuitable, or if the user inputs a value of zero. The default is 30 a.u. See Sec. 19.4.4.

NED (*Integer*) For real systems containing atoms, **ned** is the total number of spin-down electrons referenced by the many-body wave function (for periodic systems, this is the number of spin-down electrons in the simulation cell, rather than the underlying primitive cell). The number of spin-up electrons is given by the keyword **neu**.
Note that in the presence of addition or subtraction excitations, **ned** refers to the state of the system AFTER the required number of electrons have been added or removed. For model electron(–hole) phases such as the HEG, set **ned** to zero and use the **free_particles** block to define the number of spin-down electrons.

NEIGHPRINT (*Integer*) **neighprint**= n will generate a printout of the first n stars of neighbours of each atom in the primitive cell, with the relevant interatomic distances given in both Ångstrom and a.u. If $n = 0$ or if you have an atom-free electron or electron–hole fluid phase, then the keyword has no effect. Note that activating cusp corrections when using a Gaussian basis (the default) will trigger a neighbour analysis irrespective of the value of this keyword.

NEU (*Integer*) For real systems containing atoms, **neu** is the total number of spin-up electrons referenced by the many-body wave function (for periodic systems, this is the number of spin-up electrons in the simulation cell, rather than the underlying primitive cell). The number of spin-down electrons is given by the keyword **ned**.
Note that in the presence of addition or subtraction excitations, **neu** refers to the state of the system AFTER the required number of electrons have been added or removed. For model electron(–hole) phases such as the HEG, set **neu** to zero and use the **free_particles** block to define the number of spin-up electrons.

NEWRUN (*Logical*) Specifies whether a new run or a continuation of an old one is to be performed:

`T`: (VMC) **vmc_equil_nstep** Metropolis steps are performed on a set of randomly generated configurations before accumulation of statistics begins.;
(DMC) A set of VMC configurations is read from a `config.in` file and the initial best estimate of the energy EBEST is calculated as the mean energy of these configurations. In the special case of **dmc_reweight_configs**=`T`, `config.in` may also contain *DMC* configurations which are reweighted to a new wave function and EBEST shifted by the current energy difference between the wave functions.;

`F`: (VMC) Continuation of an old run. A set of old (and presumably equilibrated) electron positions are read from a `config.in` file, and no Metropolis equilibration steps are performed before accumulation of statistics.;
(DMC) Continuation of an old run. A set of DMC configurations is read from a `config.in` file and EBEST is not recomputed but taken to have the value written on the end of the `config.in` file (presumably by a previous DMC run, either equilibration or accumulation).

NHD (*Integer*) Like **nhu**, but for spin-down particles.

NHU (*Integer*) Number of spin-up fermions other than electrons in real systems. For example, if you are interested in positronic molecules then **nhu** should be set to 1, and the up-spin positron ('spin' 3) should be defined appropriately in the **particles** block. Likewise for muonic systems. At present only the Gaussian routines can be used to return orbitals for species other than electrons. (There exists a modified version of the GAUSSIAN code that can be used to generate orbitals for positronic molecules.) If you want to use a different basis or study muons or something then you will need to make the appropriate changes first. For model systems such as electron-hole gases, etc., please use the **free_particles** block to define the number of spin-up holes.

NLCUTOFFTOL (*Real*) This is used to define the cutoff radius for the nonlocal parts of the pseudopotential. It is defined as the maximum deviation of the nonlocal potentials from the local potential at the nonlocal cutoff radius. See Sec. 19.3.

NNJAS_B_AMP (*Real*) Like **nnjas_w_amp**, but for biases.

NNJAS_W_AMP (*Real*) Amplitude for random default weights in neural network Jastrow terms. Neural network weights that are not specified in the correlation.data file are chosen from a uniform distribution on the interval [−NNJAS_W_AMP,NNJAS_W_AMP).

NON_LOCAL_GRID (*Integer*) **non_local_grid** selects the grid for nonlocal integration, ranging from coarse (low **non_local_grid** value) to fine (high **non_local_grid** value). The value is assumed to be the same for all atoms if it is controlled through this keyword; alternatively you can provide values of **non_local_grid** for particular species by specifying values for **nlrule1** at the top of the corresponding pseudopotential files. **non_local_grid** can take values between 1 and 7, the default being 4 (used if **non_local_grid** is negative or not supplied). See Sec. 19.2 for more information.

NPCELL (*Block*) Vector of length 3 giving the number of primitive cells in each dimension that make up the simulation cell. N.B., for the 1D-periodic case, **npcell**(2) and **npcell**(3) must be 1, and for the 2D slab case, **npcell**(3) must be 1. To construct more general simulation supercells, please use the **scell_matrix** keyword.

NUCLEUS_GF_MODS (*Logical*) This keyword is the switch for enabling the use of the modifications to the DMC Green's function for the presence of bare nuclei, suggested in Ref. [17], in order to reduce time-step errors in all-electron calculations. See Sec. 13.6.

ONEP_DENSITY_MAT (*Logical*) If **onep_density_mat** is set to `T`, then the spherically averaged one-particle density matrix will be computed and written to the `expval.data` file. This is only possible if the system is homogeneous for the moment. See Sec. 35.

ON_TOP_PAIR (*Block*) This block contains two lines consisting of the particle type and index (integers) of each of two particles to be forced to stay on top of each other throughout a VMC calculation. This is intended for the evaluation of recombining-pair momentum densities and "one-body" density matrices in hole-in-HEG systems—wrong values will be reported for other expectation values, including the energy. Note that the time step of the first-specified particle applies to the pair. This block must not be used in DMC or optimization runs.

OPT_BACKFLOW (*Logical*) Optimize backflow parameters in wave-function optimization. See Sec. 23.

OPT_COMPLEX (*Logical*) This flag determines whether the imaginary part of the local energy is taken into account in 'varmin', 'madmin' and 'varmin_linjas' calculations when a complex trial wave function is used. **opt_complex** is `T` by default.

OPT_CYCLES (*Integer*) Number of cycles of configuration generation and optimization to be carried out if **runtype**='vmc_opt' or 'opt_vmc'. For variance minimization, 3–6 cycles is typical; for energy minimization, 5–10 cycles is usual unless only determinant coefficients are being optimized, in which case 1–2 cycles will be enough.

OPT_DET_COEFF (*Logical*) Optimize the coefficients of the determinants in wave-function optimization.

OPT_DTVMC (*Integer*) This keyword may take three possible values: **opt_dtvmc**=0 (default) turns off optimization of the VMC time step, while **opt_dtvmc**=1 causes the time step to be optimized during equilibration in order to achieve an acceptance ratio of (roughly) 50%. See Sec. 12.4. The value of **opt_dtvmc** is ignored if the input block **dtvmcs** is supplied.
CASINO can also maximize the diffusion constant with respect to **dtvmc**. This can be enabled by setting **opt_dtvmc**=2. In a first stage, **dtvmc** is varied to get an acceptance ratio of 50%, so as to have decent statistics to perform the diffusion-constant maximization stage. This last option is only useful for **vmc_method**=3, where it is the default.

OPT_FIXNL (*Logical*) If this keyword is set to `T` then the nonlocal energy will be fixed where possible in optimization. This is recommended for variance minimization (and `T` by default) as it greatly improves the speed of the optimization process, and possibly the accuracy of the optimization as well. For energy minimization, it is `F` by default, as the speed increase is small.

OPT_GEMINAL (*Logical*) During optimization, allow the optimization of the geminal coefficient matrix.

OPT_INFO (*Integer*) Controls amount of information displayed and/or written out during wave-function optimization. Variance minimization: (1) display no information; (2) display energies at each function evaluation; (3) as (2), but calculate weights as well; (4) write out configurations and their energies, etc., as they are read in. Energy minimization: (1) little information; (2) basic information; (3) full information, write matrix algebra log file; (4) also write full matrices to log files; (5) also write SVD component matrices to log files, if SVD used.

OPT_JASTROW (*Logical*) Optimize the Jastrow factor in wave-function optimization.

OPT_MAXEVAL (*Logical*) Maximum number of evaluations of the variance during variance minimization (default 200).

OPT_MAXITER (*Integer*) Largest permitted number of `nl2sol` or global EMIN iterations (default: 10).

OPT_METHOD (*Text*) There are currently four optimization methods implemented in CASINO: (1) variance minimization ('varmin'); (2) minimization of the mean absolute deviation of the set of local energies from the median ('madmin'); (3) energy minimization ('emin'); (4) an accelerated variance minimization technique for parameters that appear linearly in the Jastrow factor ('varmin_linjas'). The first three methods are capable of optimizing the Jastrow factor, orbitals, backflow functions, and determinant coefficients. The fourth method can only be used to optimize linear parameters in the Jastrow factor. There are other keywords that affect the behaviour of each of these methods. The default value of **opt_method** is 'varmin'.

OPT_NOCTF_CYCLES (*Integer*) Supplying a positive integer X for this keyword will cause all 'shallow' parameters (cut-off lengths in the Jastrow factor, backflow transformation and orbitals) to remain fixed for the final X cycles of a multi-cycle optimization run. This is potentially useful for energy minimization, which can be adversely affected by the presence of optimizable cut-off parameters. **opt_noctf_cycles** defaults to 0 (i.e., cut-offs are never fixed).

OPT_ORBITALS (*Logical*) Optimize parameters in the orbitals in wave-function optimization (relevant, e.g., if **use_orbmods**=`T`).

OPT_PLAN (*Block*) Allows specifying different parameters for each optimization cycle for **runtype** 'vmc_opt', 'opt_vmc' or 'opt'. The block has one line per optimization cycle (the block length overrides the value of **opt_cycles**), each containing the cycle index followed by any number of blank-separated `<keyword>=<value>` assignments. Valid keywords are:

- `method`: sets **opt_method** to `<value>` for the cycle (string)
- `reweight`: sets **vm_reweight** to `<value>` for the cycle (Boolean)
- `w_max`: sets **vm_w_max** to `<value>` for the cycle (real)
- `w_min`: sets **vm_w_min** to `<value>` for the cycle (real)
- `sample_hf`: sets **vmc_sample_hf** to `<value>` for the cycle (Boolean)
- `jastrow`: sets **opt_jastrow** to `<value>` for the cycle (Boolean)
- `backflow`: sets **opt_backflow** to `<value>` for the cycle (Boolean)
- `det_coeff`: sets **opt_det_coeff** to `<value>` for the cycle (Boolean)
- `orbitals`: sets **opt_orbitals** to `<value>` for the cycle (Boolean)
- `geminal`: sets **opt_geminal** to `<value>` for the cycle (Boolean)
- `maxiter`: sets **opt_maxiter** to `<value>` for the cycle (Boolean)
- `fix_cutoffs`: determines whether to fix cut-offs (`T`) or not (`F`) for the cycle (Boolean; analogous to **opt_noctf_cycles**)

Input keywords will remain at their provided/default values for all cycles for which they are not modified by the corresponding **opt_plan** line.

OPT_STRICT (*Logical*) Setting **opt_strict**=`T` will cause CASINO to stop a vmc_opt or opt_vmc run if the VMC energies are incremented within a 99.7% confidence interval during two consecutive cycles. Prevents wastage of CPU time in times of scarcity. Default value is `F`.

ORB_NORM (*Real*) Allows user to change normalization of orbitals by multiplying all of them by this constant. Of course this should have no effect on the energy, but it can be useful if the Slater determinant starts going singular, as it might for some very low-density systems.

ORBBUF (*Logical*) Setting **orbbuf**=`T` turns on orbital buffering in DMC. This is an efficiency device in which buffered copies of orbitals/gradients/Laplacians are kept for later reuse. This has a significant memory cost. Orbital buffering should always be used unless you start running out of memory; hence the ability to turn it off.

PAIR_CORR (*Logical*) Set **pair_corr** to `T` to accumulate the reciprocal-space pair-correlation function in the `expval.data` file. Currently restricted to periodic systems. Note that you also need to give the position and type of a fixed particle using the **pcf_rfix** block (unless the density is homogeneous). See Sec. 35.

PAIR_CORR_SPH (*Logical*) If **pair_corr_sph** is set to `T` then the spherically averaged real-space pair-correlation function will be accumulated in the `expval.data` file (via a process of 'binning' the electron-electron separations). This currently works for periodic homogeneous systems and finite isotropic systems such as electron-hole bilayers. For periodic systems with atoms you can use the **pair_corr** keyword instead which gives you the full (non-spherically averaged) pair-correlation function accumulated in reciprocal space. See Sec. 35.

PARTICLES (*Block*) Using the **particles** block the user can define quantum particles (other than electrons, which can be introduced using **neu** and **ned**) to be used in the QMC calculation. The format of each line is '$\langle i \rangle$ $\langle$charge/$|e|\rangle$ $\langle$mass/$m_e\rangle$ $\langle$spin/$\hbar\rangle$ $\langle$name$\rangle$'. A negative value of the mass indicates that the following three lines give an anisotropic 3×3 mass tensor [currently unused]. CASINO decides whether each particle type is a fermion or a boson (based on the spin), and selects the appropriate way to combine the one-particle orbitals (symmetric combination [not currently implemented] or antisymmetric Slater determinants). The particles defined here can be assigned orbitals using the **free_particles** block.

PCF_RFIX (*Block*) This block contains two lines. The first line gives the type of particle to be fixed during accumulation of the pair correlation function $g(r, r')$; the second line gives the coordinates of the position $\mathbf{r}$ at which to fix it (in a.u.). This applies to the reciprocal-space pair-correlation function (PCF) activated with the **pair_corr** input keyword. It also applies *in principle* to the spherical real space PCF activated with the **pair_corr_sph** input keyword, in the sense that the format of `expval.data` allows it, but the accumulation of the spherical PCF with fixed particles has not yet been implemented. See Sec. 35.

PCFS_RCUTOFF (*Physical*) Radius of region to be considered when accumulating the pair-correlation function. The default value in periodic systems (the Wigner–Seitz cell radius) is generally appropriate; however, for finite systems such as excitonic complexes, **pcfs_rcutoff** should be set to something rather larger than the size of the complex. Note that units (e.g., bohr) should be supplied after the value of **pcfs_rcutoff**. Enter a negative value or omit the keyword to use the default value. If an `expval.data` file is present then the value given in `expval.data` will be used and the `input` keyword **pcfs_rcutoff** will be ignored. See Sec. 35.

PCFS_NBINS (*Physical*) Number of bins to be used when accumulating the real-space pair-correlation function. Enter a negative value or omit the keyword to use the default value. If an `expval.data` file is present then the value given in `expval.data` will be used and the `input` keyword **pcfs_nbins** will be ignored. See Sec. 35.

PERIODIC (*Logical*) `T` if and only if the system is periodic in either 1, 2 or 3 dimensions.

PERMIT_DEN_SYMM (*Logical*) If this flag is set to T then he symmetry of the self-consistent field (SCF) charge density (in `mpc.data`) will be imposed on the QMC charge-density data used in the MPC interaction (with the justification that imposing an exact condition on the charge density can't hurt) and also when writing the QMC density to `expval.data`. It is possible however that DMC will break the symmetry of the SCF calculation; in this case the user should turn off **permit_den_symm**.

PLOT_BACKFLOW (*Block*) This block allows a plot of the backflow transformation to be made (see Sec. 23) just after VMC equilibration. The block should contain 2 lines, plus an optional line for plotting the Φ term: (1) '0' or '1' to (de-)activate this facility; (2) 'kspin', 'knumber' and 'zposition'; (3) value of fixed electron–nucleus distance r_{iI}. This will produce various files: `bfconfig.dat` (reference config), `bfconfigx.dat` (associated quasi-particle config), `bfions.dat` (coordinates of nuclei for which backflow terms exist), `bfeta_`$\langle$`s`$\rangle$`.dat` (η vs. r_{ij} for each spin-pair type $\langle$s$\rangle$), `bfmu_`$\langle$`s`$\rangle$`_`$\langle$`set`$\rangle$`.dat` (μ vs r_i for each spin type $\langle$s$\rangle$ in each set $\langle$set$\rangle$), `bfphi.dat` [contribution of Φ to the 3D backflow displacement on electron j vs. 2D projection of r_{jI} on the plane defined by electron j, ion I in set $\langle$set$\rangle$, and electron i at distance r_{iI} from the nucleus with spin such that $\langle$s$\rangle$ is the spin-pair type of (i, j)], and `bffield.dat` (3D backflow displacement on electron (kspin,knumber) vs. its 2D position on the plane $z =$ zposition).

PLOT_EXPVAL (*Block*) This block allows the `plot_expval` utility to be told about the geometrical region over which a particular expectation value is to be plotted (which can be a line AB / plane AB-AC / volume AB-AC-AD). The structure of the block is as follows:
Line 1: dimensionality d (1, 2 or 3);
Line 2: No. of points along the d directions;

Line 3: x, y and z coordinates of point A;
Line 4: x, y and z coordinates of point B;
Line 5: x, y and z coordinates of point C (if required);
Line 6: x, y and z coordinates of point D (if required).
The data will be plotted in a format suitable for XMGRACE in the file `lineplot.dat` or in a format suitable for GNUPLOT in `2Dplot.dat` or `3Dplot.dat` which can be quickly visualized using the `plot_2D` utility. This block is ignored by CASINO itself. Linear combinations of different `lineplot.dat` and `2Dplot.dat` files, e.g. for extrapolated estimation, can be taken using the `combine_plot_data` utility.

POPSTATS (*Logical*) If **popstats** is `T` then the variance of the local energies sampled in DMC will be evaluated.

POPULATION (*Logical*) If **population** is `T` then the population of electrons associated with each ion is estimated by evaluating the electronic charge in Voronoi polyhedra about the ions. See Sec. 35.9 for more information.

POSTFIT_VMC (*Logical*) If **postfit_vmc** is set to T then an extra VMC calculation will be performed with the final optimized wave function when **runtype**=vmc_opt or opt_vmc, to enable one to see the effect of the final optimization on the energy, etc. This is done by default. Unless postfit_keep_cfg is set to T, this final VMC run will not generate any configurations.

POSTFIT_KEEP_CFG (*Logical*) If **postfit_keep_cfg** is set to `T` then the configurations generated in the post-fit VMC calculation will be written to `config.out`. The default is F.

PRIMITIVE_CELL (*Block*) Sometimes the 'primitive lattice vectors' in the `xwfn.data` file do not correspond to the true primitive cell. If the actual primitive lattice vectors are needed (e.g., for accumulation of the charge density), then override values can be supplied in the **primitive_cell** block.

PRINTGSCREENING (*Logical*) Before doing a periodic Gaussian calculation, CASINO prepares lists of potentially significant (primitive) cells and sites in each such cell which could contain Gaussians having a nonzero value in a reference primitive cell centred on the origin. Zero is defined as $10^{-\mathbf{gautol}}$. Turning on the **printgscreening** flag prints out the important information about this screening.

PSI_S (*Text*) If **psi_s** is 'none' then the Slater wave function is set to one; if **psi_s** is 'slater' (default) then an expansion in one or more Slater determinants is used; if **psi_s** is 'exmol' then a specially crafted wave function for excitonic and positronic molecules is used; if **psi_s** is 'mahan' then a custom wave function for studying a single positive impurity in a HEG is used (see Sec. 33).

QMC_DENSITY_MPC (*Logical*) If this flag is set to T then the QMC charge data at the end of the `expval.data` file will be used to compute the MPC interaction (see Sec. 19.4.4), rather than the default SCF density in the `mpc.data` file. This is likely to be useful in cases such as the Wigner crystal where the Hartree–Fock charge density is very different to the true charge density (it is too localized) as opposed to, say, the Fermi fluid where the Hartree–Fock charge density is exact. Note that using this option is likely to increase the time taken to evaluate the MPC interaction; in both DFT and QMC cases, the code counts backwards from the end of the list of G vectors and discards all those before the first non-zero one (where zero is defined by some threshold like 1.d-6). In the HF/DFT case this tends to give a large reduction in the size of the vector to be evaluated. However, the random noise in the QMC density coefficients is likely to exceed the zero threshold for all G, and the vector will likely be untruncated. It is important therefore to use a value for **expval_cutoff** which is not too large when using this facility, in order that the total number of G vectors in the expansion is not too large.

QMC_PLOT (*Block*) This utility allows you to plot the value of certain quantities along a line *AB* / plane *AB-AC* / volume *AB-AC-AD*. The data will be plotted in a format suitable for XMGRACE in the file `lineplot.dat` or in a format suitable for GNUPLOT in `2Dplot.dat` or `3Dplot.dat`. The block has the following format:
Line 1: what to plot (either 'orb', 'orb_gradx', 'orb_grady', 'orb_gradz', 'orb_lap', 'wfn', 'nodes', 'energy', 'eipot' or 'expot');
Line 2: dimensionality (1, 2 or 3);
Line 3: no. of points along each direction;
Lines 4–: (x, y, z) coordinates of point A; of point B; of point C (if required); of point D (if

required).
Three additional lines have to be added for orbital plots:
Line A1: number of orbitals ($N_{\rm orb}$);
Line A2: $N_{\rm orb}$ integers identifying the orbitals to be plotted;
Line A3: $N_{\rm orb}$ integers identifying the spin/species for each of the orbitals.
For wave-function and local-energy plots, the coordinate defined by lines 4– refer to electron 1 by default. The coordinates of all remaining electrons are taken from a configuration obtained by VMC equilibration (without Jastrow factor). Alternatively the positions of several electrons may be fixed. The following line(s) must be added:
Line B1: number of fixed electrons (≥ 0);
Line B2 and onwards (if the number of fixed electrons is nonzero): spin, number and (x, y, z) coordinates for each of the fixed electrons.
The default of moving the first electron can be overridden by adding the lines:
LINE B3: number of electrons to move (= 1 or 2);
LINE B4: spin, number of first electron to move;
LINE B5: spin, number and offset of second electron relative to the first.
For electron–ion-potential, external-potential and node plots, no lines have to be added.

RANDOM_SEED (*String*) This keyword determines which random seed to use for the 'ranlux' random-number generator. The default value of **random_seed** is 'timer', which causes the system timer to be used as the seed. If **random_seed** is set to 'standard', the seed 314159265 is used. If the value of **random_seed** is an integer, that integer will be used as the random seed. The seed is printed to the output file so that calculations using **random_seed**='timer' can be reproduced afterwards. Note that, if **random_seed** is an integer or 'standard' or 'timer' then, when restarting from a previous calculation the value of **random_seed** is ignored (except for any initial setup, such as evaluation of the twist-averaged Hartree–Fock energy of a homogeneous electron gas), and the random-number sequence will be continued from the saved state of the random-number generator stored in the `config.in` file. However, if **random_seed** is 'timer_reset' then the generator will be re-initialized from the system clock after the `config.in` file is read. This might be useful, for example, if a prior test has revealed that the standard sequence will lead to a configuration giving rise to a population explosion.

RANLUXLEVEL (*Integer*) To generate the parallel streams of pseudo-random numbers for its stochastic algorithms, CASINO uses an implementation of the `ranlux` algorithm. This is an advanced pseudo-random number generator based on the `rcarry` algorithm proposed in 1991 by Marsaglia and Zaman. `rcarry` used a subtract-and-borrow algorithm with a period on the order of 10^{171} but still had detectable correlations between numbers. Martin Luescher proposed the `ranlux` algorithm in 1993; `ranlux` generates pseudo-random numbers using `rcarry` but throws away numbers to destroy correlations. `ranlux` trades execution speed for quality through the choice of a 'luxury level' given in CASINO by the **ranluxevel** input keyword. By choosing a larger luxury setting one gets better random numbers slower. By the tests available at the time it was proposed, `ranlux` at its higher settings appears to give a significant advance in quality over previous generators. The luxury setting must be in the range 0–4. Level 0: equivalent to the original `rcarry` of Marsaglia and Zaman, very long period, but fails many tests. Level 1: considerable improvement in quality over level 0, now passes the gap test, but still fails spectral test. Level 2: passes all known tests, but theoretically still defective. Level 3 [DEFAULT]: any theoretically possible correlations have very small chance of being observed. Level 4: highest possible luxury, all 24 bits chaotic.

RANPRINT (*Integer*) Setting this keyword to a value greater than zero will cause the first **ranprint** numbers generated by the CASINO random number generator to be printed to a file `random.log`. On parallel machines the numbers generated on all MPI processes are printed. The run script should pick out `random.log` files from different stages of a calculation (e.g., VMC configuration generation / DMC equilibration / DMC statistics accumulation) and rename them appropriately. Since 1.2011 (for more than 100 processes) the check on random seeds being equal on different MPI processes is only performed when **RANPRINT** is greater than zero.

REDIST_GRP_SIZE (*Integer*) In the branch-and-redistribute algorithm (which does redistribution of configurations across MPI processes in DMC), we must decide which pairs of MPI processes are involved in configuration transfers, and how many configurations are to be transferred in each operation. There is an optimal algorithm for doing this (involving looking at individual configuration multiplicities and the exact excess or deficit of configurations relative to a target on each MPI process). If we consider all the MPI processes then this algorithm scales linearly with the number of MPI processes, eventually becoming so expensive that for a fixed number of configurations the code actually becomes slower if we increase the number of MPI processes. We therefore parallelize the algorithm; to do this we form groups of processes ('redist groups') of size **redist_grp_size** (plus some remainder). When calculating the vector of instructions, only transfers within these groups are contemplated, and the cost for working out what to send where no longer increases with the number of MPI processes (above a certain size).

RELATIVISTIC (*Logical*) If **relativistic** is `T`, then relativistic corrections to the energy are calculated using perturbation theory. Note that this can only be done for closed-shell systems at present. See Sec. 34 for further details.

RNG_RESTART_SAFE (*Logical*) We would like, e.g., a 1000-move VMC run, and two 500-move VMC runs linked together by a restart to give the same answer, in the sense that we end up with the same `vmc.hist` file. Unfortunately they do not in general since the pseudorandom number sequence is affected by the restart. This is because random numbers are generated something like 63 at a time and stored in a buffer until needed (this buffer being refilled when necessary). In the normal way of saving a point in the random number sequence, any unused numbers in the buffer are discarded, which means the final answer will be different to the unrestarted case. If the keyword **rng_restart_safe** is `T` (which is actually now the default) then the whole current buffer is saved in the final `config.out` file as well as the current state of the random sequence (necessarily fixed at the end of the current buffer). This allows multiple step runs to give the same answer as single step runs, at the expense of slightly larger configuration files.

RUNTYPE (*Text*) This keyword specifies the type of QMC run to be carried out. It can take the following values: 'vmc' (perform a single VMC simulation); 'dmc_equil' (perform DMC equilibration); 'dmc_stats' (perform DMC statistics accumulation); 'dmc' (perform DMC equilibration, then statistics accumulation); 'vmc_dmc' (perform VMC, then DMC equilibration, then DMC statistics accumulation); 'vmc_dmc_equil' (perform VMC, then DMC equilibration); 'opt' (perform a single wave-function optimization calculation); 'vmc_opt' (perform **opt_cyles** cycles of VMC and optimization, alternately); 'opt_vmc' (perform **opt_cycles** cycles of optimization and VMC, alternately); 'gen_mpc' (generate an `mpc.dat` file enabling the use of the MPC interaction; requires **complex_wf**=`T`); 'gen_blip' (generate a blip representation, stored in `bwfn.data` or `bwfn.data.bin`, of the plane-wave orbitals stored in `pwfn.data`); 'gen_gpcc' (generate a `gpcc.casl` file with a Jastrow factor term reproducing the e-n cusp correction from the GPCC facility; see Sec. 7.8.1); 'gen_gpcc_single' and 'gen_gpcc_simple' (like 'gen_gpcc', but make assumptions to simplify the term); 'gen_mdet_casl' (generate an `mdet.casl` file for use by the **det_compress** utility); 'plot' (perform plot specified by block **qmc_plot**). NOTE: in earlier versions of the code, we used '**runtype** : dmc' with the now-redundant keyword '**iaccumulate** : `T` or `F`' to indicate whether stats accumulation was activated or not. This usage is now deprecated and, unless **iaccumulate** is specifically defined in input, then '**runtype** : dmc' is just a synonym for '**runtype** : dmc_dmc'.

SCELL_MATRIX (*Block*) If the simulation-cell lattice vectors are $\{\mathbf{a}_i\}$ and the primitive-cell lattice vectors are $\{\mathbf{p}_j\}$ then we may write $\mathbf{a}_i = \sum_j S_{ij}\mathbf{p}_j$, where the S_{ij} are integers. The 3×3 integer matrix S is given in the **scell_matrix** `input` block. This is a generalization of **npcell** (which simply gives the diagonal elements of S in the special case that S is diagonal). Only one of **npcell** and **scell_matrix** should be present in the `input` file. You can use the `supercell` utility to construct S.

SHM_SIZE_NPROC (*Integer*) In Shm calculations on Blue Gene machines one needs to know in advance the number of MB of shared memory required, so that one may set the BG_SHAREDMEMSIZE environment variable (which can be done by means of the `--user.shemsize` argument to runqmc). CASINO will calculate this number and print it to output at the end of the setup process (within the scope of **testrun**=T). However, the amount of shared memory required depends on the number of MPI processes per node (or per shared memory partition). If one ultimately wishes to run on, say, half a million cores, it may be desirable to execute a test run on just a few cores on your personal laptop, rather than waiting a week for the full job to sit in a queue. For the purposes of computing the size of the shared memory partition, one may therefore set the number of desired processes/node by setting **shm_size_nproc**, and this value will be used in computation of the shared memory size rather than the actual number of processes/node being used in the test run (unless **shm_size_nproc=0**—which is the default). Note that the CASINO test run must be done in Shm mode.

SMALL_TRANSFER (*Logical*) If **small_transfer** is set to `T`, the DBAR matrices and any potentially large optional data are not transferred across MPI processes in DMC configuration redistribution. The default is `F`. Set to `T` if you run into problems with parallel transfers.

SPARSE (*Logical*) CASINO is capable of using sparse matrix algebra in some algorithms for efficiency purposes. For systems which are definitely not sparse (orbitals not well localized) then attempting to use sparse algorithms might actually slow things down. Thus, until we work out a better way, you can toggle this behaviour with the **sparse** flag. In these algorithms a matrix element is considered to be zero if it less than the value of the input keyword **sparse_threshold**.

SPARSE_THRESHOLD (*Real*) CASINO sometimes uses sparse matrix algebra for efficiency purposes. In a future version, matrix elements will be taken to be zero if they are less than **sparse_threshold**. This keyword has no effect at present, however.

SP_BLIPS (*Logical*) The single-particle orbitals that appear in the determinants can require a great deal of memory when expanded in a blip basis. When **sp_blips**=`T`, CASINO represents the blip coefficients using single-precision real or complex numbers, which will halve the memory required. This parameter is only relevant when **atom_basis_type**='blip'. The default value is `F`.

SPIN_DENSITY (*Logical*) Setting **spin_density** to `T` will activate the accumulation of separate up- and down-spin densities in the `expval.data` file. See Sec. 35.

SPLOT (*Logical*) If **splot** is `T` then the line plotter will use just the s component of orbitals. Useful for analysing Gaussian cusp corrections (see Sec. 16).

STOP_METHOD (*Text*) The **stop_method** keyword defines how VMC and DMC runs are to be terminated. It may take the values '`nstep`', '`target_error`', or '`small_error`'.

The classic method is '`nstep`' which means simply: perform the number of VMC/DMC steps implied by the input keywords **vmc_nstep** or **dmc_stats_nstep** then stop. The error bar then is what it is (it may be too large or smaller than required).

Note that in the VMC case **stop_method**s other than '`nstep`' are used only for pure VMC calculations, i.e., for **runtype**='`vmc`'. The value of **stop_method** is implicitly assumed to be '`nstep`' for the VMC stage of optimization or DMC calculations.

If **stop_method** = 'target_error' then the run will continue until the error bar on the total energy (corrected on the fly for serial correlation) is approximately equal to that defined by the **target_error** input keyword, subject to the constraint that the *estimated* CPU time required on the master process (summed over restarts if necessary) will not exceed **stop_time**. CASINO is able to approximately estimate the required time by analysing how the error bar decreases as a function of the number of moves, and as soon as it is reasonably confident that the desired **target_error** is too small and cannot be reached, then the code will stop (in a restartable condition). On halting in this manner, an estimate of the CPU time required to get a range of error bars will be written to the output file. Note that the method used to estimate the required time assumes the validity of the central limit theorem, which is only approximately valid in most cases.

If **stop_method** = '*small_error*', CASINO will attempt to make the error bar as small as possible in a 'reasonable time' defined by the value of the **stop_time** keyword. 'As small as possible' means what it says, but taking account of the fact that there is an error bar on the error bar and it is somewhat pointless to reduce the error bar below its significant precision.

Note in both the last two cases CASINO has a minimum run length needed to get a reasonable estimate of the variance.

STOP_TIME (*Text*) If **stop_method** = '`target_error`' then the run will continue until the error bar on the total energy (corrected on the fly for serial correlation) is approximately equal to that defined by the **target_error** input keyword, subject to the constraint that the *estimated* CPU time required on the master process (summed over restarts if necessary) will not exceed **stop_time**. CASINO is able to approximately estimate the required time by analysing how the error bar decreases as a function of the number of moves, and as soon as it is reasonably confident that the desired **target_error** is too small and cannot be reached, then the code will stop (in a restartable condition). On halting in this manner, an estimate of the CPU time required to get a range of error bars will be written to the output file. Note that the method used to estimate the required time assumes the validity of the central limit theorem, which is only approximately valid in most cases.

If **stop_method** = '`small_error`', CASINO will attempt to make the error bar as small as possible in a 'reasonable time' defined by the value of **stop_time**. 'As small as possible' means what it says, but taking account the fact that there is an error bar on the error bar and it is somewhat pointless to reduce the error bar below its significant precision.

Note that in both cases the code will block pointlessly short stop times, where 'pointless' is currently and arbitrarily defined to be less than 60 seconds.

STRUC_FACTOR_SPH (*Logical*) If **structure_factor_sph** is set to `T` then the spherically averaged structure factor will be accumulated in the `expval.data` file (homogeneous systems only). See Sec. 35.

STRUCTURE_FACTOR (*Logical*) If **structure_factor** is set to `T` then the structure factor will be accumulated in the `expval.data` file (periodic systems only). See Sec. 35.

TARGET_ERROR (*Text*) If **stop_method** = '`target_error`' then the run will continue until the error bar on the total energy (corrected on the fly for serial correlation) is approximately equal to that defined by the **target_error** keyword, subject to the constraint that the *estimated* CPU time required on the master process (summed over restarts if necessary) will not exceed **stop_time**. CASINO is able to approximately estimate the required time by analysing how the error bar decreases as a function of the number of moves, and as soon as it is reasonably confident that the desired **target_error** is too small and cannot be reached, then the code will stop (in a restartable condition). On halting in this manner, an estimate of the CPU time required to get a range of error bars will be written to the output file. Note that the method used to estimate the required time assumes the validity of the central limit theorem, which is only approximately valid in most cases. Note also that the code will block infeasibly small target errors, where 'infeasible' is currently and arbitrarily defined to be less than 10^{-6} a.u.

TESTRUN (*Logical*) If this flag is `T` then CASINO will read input files, print information and stop.

TIMING_INFO (*Logical*) Setting **timing_info** to `T` will turn on the collection of subroutine timings. Note, however, that the timing routines can adversely affect performance on certain computers, especially for small systems; hence timers are deactivated by default. See Appendix A.6. If **timing_info** is `T` and CASINO halts to report an error message, CASINO will be able to provide some traceback information about where that error occurs.

TPDMC (*Integer*) **tpdmc** ($T_{\rm p}$) is the number of time steps for which the effects of changes in the (theoretically constant) reference energy should be undone in order to estimate the DMC energy at a given point. It is assumed that the best estimates of the DMC energy separated by an amount greater than this are not correlated by fluctuations in the reference energy. Thus $T_{\rm p}$ should exceed the timescale of fluctuations in the reference energy. A suggested value is $T_{\rm p} = 10/\tau$, where τ is the time step. If you set it to 9999 in the input, then the code will automatically use this value; if you set it to 0 then the reweighting scheme for population-control biasing will not be used. The latter is the default since time-step bias is generally not a problem in practice, and the reweighting scheme is an additional source of noise. See Sec. 13.7.

TWOP_DENSITY_MAT (*Logical*) If **twop_density_mat** is set to `T`, then the spherically averaged diagonal term of the two-particle density matrix will be computed. This is only possible if the system is homogeneous for the moment, and will increase the cost of the calculation significantly. See Sec. 35.

TWOP_DM_MOM (*Logical*) If **twop_dm_mom** is set to `T`, then the Fourier transform of the two-particle density matrix will be computed. This is only possible if the system is homogeneous for the moment, and will increase the cost of the calculation significantly. See Sec. 35.

USE_DETLA (*Logical*) If set to `T`, the *determinant localization approximation* (DLA) [24] to non-local terms of pseudopotentials will be used. Use of the determinant locality approximation in conjunction with the T-move scheme is discussed in Sec. **??**.

USE_GJASTROW (*Logical*) If set to `T`, the *gjastrow* Jastrow factor will be used. This Jastrow factor is defined in a `parameters.casl` file. The default is to use the standard Drummond-Towler-Needs Jastrow factor if a `correlation.data` file is present, else use the *gjastrow* Jastrow factor.

USE_GPCC (*Logical*) If this is set to `T` then short-ranged functions will be added to the orbitals to ensure that the Kato cusp conditions are satisfied.

USE_JASTROW (*Logical*) Use a wave function of the Slater-Jastrow form, where the Jastrow factor exp(J) is an optimizable object that multiplies the determinant part in order to introduce correlations in the system. The Jastrow factor is read from the 'JASTROW' block in `correlation.data` (see Sec. 7.4.2). The form of CASINO's Jastrow factor is described in Sec. 22. If **use_jastrow** is `F` then the Slater wave function will not be multiplied by the Jastrow factor.

USE_ORBMODS (*Logical*) If **use_orbmods** is set to `T` then the orbital-modification block in `correlation.data` will be read, and the modification functions will be added to the numerical atomic orbitals. This only applies if **atom_basis_type** is set to 'numerical', 'gaussian' or 'slater-type'. See Secs. 7.4.7 and 7.4.8. To optimize the corresponding parameters, use **opt_orbitals**.

USE_TMOVE (*Logical*) If **use_tmove** is `T` then the Casula nonlocal pseudopotential scheme [25] will be used in DMC. So-called 'T-moves' will be performed in order to give a DMC energy that is greater than or equal to the ground-state energy. This violates the detailed-balance principle at finite time steps, but greatly improves the stability of the DMC algorithm when nonlocal pseudopotentials are used. The advantages of T-moves are that they restore the variational principle and help to prevent population explosions; the disadvantages of T-moves are that the magnitude of the error due to the locality approximation is generally larger, although always positive, and the time-step bias is generally worse. [This latter problem is alleviated, to some extent, by using a symmetric branching factor [26] as opposed to the asymmetric one suggested in Ref. [25]. This advice was implemented in CASINO in June 2014.]. T-moves are now (as of 2018) used by default. T-moves in CASINO are discussed in Sec. **??**.

VIRTUAL_NCONFIG, VIRTUAL_NNODES, VIRTUAL_NODE (*Integers*) Number of configurations, number of virtual nodes and virtual node number in virtual parallel variance minimization. These parameters are not to be set manually.

VM_E_GUESS (*Physical*) If **vm_use_e_guess** is `T` then **vm_e_guess** should be supplied as an estimate of the ground-state energy. (Note that energy units should be supplied.) See Sec. 25.1.

VM_FILTER (*Logical*) This keyword activates filtering of configurations in variance minimization by making the weights (artificially) energy-dependent, i.e., $W_i = W(|E_i - E_{\rm ave}|)$. This method uses two parameters: **vm_filter_thres** and **vm_filter_width**. See Sec. 25.1.

VM_FILTER_THRES, VM_FILTER_WIDTH (*Real*) When limiting outlying configurations in variance minimization (by setting the **vm_filter** flag to `T`), the maximum deviation from the average energy at which the (artificial) weight of a configuration $W_i = W(|E_i - E_{ave}|)$ is kept equal to unity is **vm_filter_thres** times the square root of the unreweighted variance. Outside this limit, the weight is brought to zero using a gaussian of width **vm_filter_width** times the square root of the unreweighted variance. By default, **vm_filter_thres** is 4.0 and **vm_filter_width** is 2.0. See Sec. 25.1.

VM_FORGIVING (*Logical*) Do not whinge about calculated configuration energies not agreeing with those read in. Recommended to be set to `F`.

VM_LINJAS_ITS (*Integer*) **vm_linjas_its** specifies the maximum number of iterations to be performed if **vm_linjas_method** is 'CG', 'SD', 'BFGS', 'CG_MC', 'BFGS_MC' or 'GN_MC'. If **vm_linjas_method** is 'MC', 'LM', 'CG_MC', 'BFGS_MC', or 'GN_MC' then it (also) specifies the number of line minimizations to be performed. See Sec. 25.2. Setting **vm_linjas_its** to 0 gives default behaviour, which is usually adequate.

VM_LINJAS_METHOD (*Text*) **vm_linjas_method** specifies the method used to minimize the quartic least-squares function in the varmin-linjas optimization scheme. **vm_linjas_method** should be one of: 'CG' (conjugate gradients), 'MC' (Monte Carlo), 'LM' (line minimization), 'SD' (steepest descents), 'BFGS' (Broyden-Fletcher-Goldfarb-Shanno), 'BFGS_MC' (BFGS and Monte Carlo), 'CG_MC' (conjugate gradients and Monte Carlo), 'GN' (Gauss-Newton) or 'GN_MC' (Gauss-Newton and Monte Carlo). (See Sec. 25.2). 'BFGS' is the default method, although in case of difficulty, it is worth trying 'GN'.

VM_REWEIGHT (*Logical*) If **vm_reweight** is `T` then the reweighted variance-minimization algorithm will be used. If `F` then the unreweighted algorithm will be used. See Sec. 25.1. Unreweighted variance minimization is recommended.

VM_SMOOTH_LIMITS (*Logical*) When set to `T`, the optimizing routine used in variance minimization is sent a smoothed version of the set of parameters. This only affects those which are to remain bounded, such as Jastrow cutoffs. The result is a set of parameters that can vary in the range $(-\infty, +\infty)$, which is more convenient than ignoring out-of-range values without the minimizer knowing. A suitable hyperbolic function is used for mapping 'limited' values into 'extended' ones and *vice versa*.

VM_USE_E_GUESS (*Logical*) Setting this flag to `T` will cause the 'variance' in variance minimization to be evaluated using **vm_e_guess** in place of the average energy of the configuration set, in an attempt to combine energy minimization with variance minimization. Otherwise the least-squares function will simply be the variance of the configuration local energies. See Sec. 25.1.

VM_W_MAX (*Real*) Maximum value that a configuration weight may take during reweighted variance minimization (i.e., when **vm_reweight** is `T`). Set **vm_w_max** to 0 if you do not wish to limit the weights; otherwise it should be greater than 1.

VM_W_MIN (*Real*) Minimum value that a configuration weight may take during reweighted variance minimization (i.e., when **vm_reweight** is `T`). **vm_w_min** should have a value between zero and one. Note that the limiting is not applied if **vm_w_max** = 0.

VMC_AVE_PERIOD (*Integer*) Number of consecutive local energies that are averaged together in VMC before writing them to the `vmc.hist` file. The only effect of this keyword is reduce the size of `vmc.hist`: the number of lines written in a VMC calculation is **vmc_nstep**/(**vmc_ave_period** × number of MPI processes).

VMC_DECORR_PERIOD (*Integer*) Length of the inner decorrelation loop in VMC. The algorithm will perform **vmc_decorr_period** configuration moves between successive evaluations of the local energy (and other quantities to be averaged) in order to ensure the configurations that are used are not significantly correlated with each other. Setting **vmc_decorr_period** to a value greater than 1 should reduce the serial correlation of the data, but the length of the run will be increased.

It is normally stated that typical values might be 3 or 4 for a pure VMC calculation, and greater than 10 during a config-generation run for optimization or DMC (though this depends on the system, and clearly setting the decorrelation period to a higher value in DMC config generation is less important than in wave function optimization, since the correlations will disappear as the DMC calculation evolves). The defaults are 3 for VMC calculations and 15 for config generation runs. A slight complication is that if **vmc_nstep** is greater than **vmc_nconfig_write** (as it might be if you need more moves than the number of desired configurations to calculate a VMC energy with a small enough error bar), then CASINO is able to exploit the extra moves to space the config writes further apart, and it is no longer necessary for the inner decorrelation loop to be so long in config generation runs. In such a case, **vmc_decorr_period** should be taken to represent the *minimum* number of steps separating config writes; internally the length of the decorrelation loop will be reduced as far as practical without going below the VMC default of 3. Note that **vmc_nstep** refers to the number of moves at which energies are evaluated or configurations are (sometimes) written out; this is not affected by the value of **vmc_decorr_period**. The value of **vmc_decorr_period** is ignore during equilibration.

Note that CASINO is able to automatically determine **vmc_decorr_period** to maximize the run efficiency. This is done by adding an extra set of moves after equilibration to compute an estimation of the correlation time of the local energies. The feature is enabled by setting **vmc_decorr_period**=0.

VMC_EQUIL_NSTEP (*Integer*) Total number of equilibration steps in VMC; should normally be at least a few thousand. Equilibration is only performed if **newrun**=`T`, thus the value of **vmc_equil_nstep** is ignored on restarts. This is a single-process quantity, that is, all MPI processes run **vmc_equil_nstep** equilibration steps. The value of **vmc_decorr_period** is ignored during equilibration.

VMC_IONJUMP (*Real*) In the special case of nearly separated molecule fragments (e.g., for intermolecular forces) electrons may get trapped in the energetically less favourable fragment for a long time during a VMC run due to the large distance between the fragments. Setting **vmc_ionjump** to a small, nonzero probability will cause the VMC routine to try a long-distance jump from one ion to another once in a while, improving the sampling of disjoint areas of the configuration space. Detailed balance is preserved.

VMC_LOCAL_DUMP (*Logical*) Setting **vmc_local_dump** causes the local energies and force components for each configuration to be written to disk. Each MPI process writes its own dump, `vmc_local_dump_`*iproc*`.dat`, containing configuration weights (if **vmc_sampling** is not 'standard') and relevant local values obtained every **vmc_decorr_period**th step.

VMC_METHOD (*Integer*) **vmc_method** selects which version of VMC to use: (1) propose single-electron moves at the accept/reject stage and evaluate configuration energies at the end of the configuration move; or, (3) propose entire configuration moves at the accept/reject stage and add a weighted sum of the old and new energies to the accumulation arrays. Method 1 is the default and is recommended. The algorithms are discussed in greater detail in Sec. 12.

VMC_NBLOCK (*Integer*) Setting **vmc_nblock** is one of two ways of specifying the number of blocks into which the total VMC run is divided post-equilibration (the other way being to specify **block_time**). The number of blocks determines how often the output, history and configuration/checkpoint files are written to disk. More specifically, at the end of each block (1) the MPI process- and block-averaged energies and a short 'report' are written to `out`, (2) the MPI process-averaged energies for each step in the current block are appended to `vmc.hist` (and other quantities to `expval.data`), and (3) the current VMC state plus any accumulated configurations are written to the `config.out` file (this latter only if the **checkpoint** input keyword is increased to 2 from its default value of 1—otherwise `config.out` is only written after the end of the final block). Note that the total energy and error bar should be effectively independent of **vmc_nblock** (provided it is ensured that the random number sequence is independent of the number of blocks, which it has not been at various periods in CASINO's history, though it should be now). Note also that the value of **vmc_nblock** is ignored if **vmc_ntwist** or **block_time** are greater than zero. The default value of **vmc_nblock** is 1.

VMC_NCONFIG_WRITE (*Integer*) Total number of configurations to be written out to the `config.out` file in VMC for later use (wave-function optimization or DMC). This number must be $\leq$ **vmc_nstep** (though you may want to set **vmc_nstep** to be significantly greater than **vmc_nconfig_write** to get an acceptable error bar on the energy; this is useful for, e.g., judging the success of an optimization after each stage). Since each MPI process always does the same number of steps, then **vmc_nconfig_write** (and **vmc_nstep**) will be rounded up to the nearest multiple of the number of MPI processes (e.g., **vmc_nconfig_write**$=20$ will be rounded up internally to 24 on 12 processes, and 24 configurations will be written to `config.out`—2 from each MPI process). Note that the `config.out` file will still be written even if **vmc_nconfig_write** is zero, since this file is used to store the current state of the system at the end of every VMC block (equivalent to writing one config, though of course multiple MPI processes write multiple configurations to save the state). Writing of `config.out` may be suppressed completely with an appropriate value for the CHECKPOINT keyword, and the data will be held in memory between different stages of the calculations.

VMC_NSTEP (*Integer*) Total number of VMC steps summed over all MPI processes; this corresponds to the total number of particle configurations for which the energy (and other quantities to be averaged) are calculated. Note that because adjacent moves are likely to be serially correlated, there is also an inner decorrelation loop of length **vmc_decorr_period**, so the total number of configuration moves attempted in a VMC run following equilibration is **vmc_nstep**$\times$**vmc_decorr_period**. On parallel machines, each MPI process will do the same number of steps and for each step the energy is averaged over the processes and written to the `vmc.hist` file (which will ultimately contain **vmc_nstep/nprocs** lines, though **vmc_ave_period** adjacent lines may be averaged over to reduce the file size). This means that if **vmc_nstep** is not divisible by the number of MPI processes then it will internally be rounded up to the nearest multiple of the number of processes (example: on a 12-core machine, given **vmc_nstep**$=20$ in `input`, CASINO will round up **vmc_nstep** to 24; each core will then do two steps and a total of two records will be written to `vmc.hist`, each of which is an average of 12 energies). On a single-core machine with **vmc_nstep**=20, CASINO will move the single config 20 times, and 20 records will be written to `vmc.hist`. Note the **vmc_nblock** or **block_time** keywords may be used to vary the frequency with which checkpointing is done, i.e., how often we write the data to disk; this does not affect the total number of VMC steps and expectation values such as average energy should be independent of it.

VMC_NTWIST (*Integer*) Number of different 'twists' or offsets to the grid of k vectors to be applied during a twist-averaged VMC run. Note that the usual keywords define the run-length for a single twist angle, thus the run-length is increased by a factor of **vmc_ntwist**. Note also that if **vmc_ntwist** is greater than zero, the values of **vmc_nblock** and **block_time** are ignored. Setting this keyword to a value greater than zero requires using a complex wave function. See Sec. 28. (Note twist-averaging wholly within `casino` can currently be done only for electron(–hole) fluid phases; for real systems with atoms one needs to couple with an external code to regenerate the wave function after each twist. The various `twistav_xxx` and `twistanalysis` scripts in `CASINO/utils/twist` can help with this). See also Sec. 28.

VMC_OPTIMUM_E0 (*Physical*) This keyword controls the centre parameter used for optimum VMC sampling, which is enabled by setting **vmc_sampling** to 'optimum' or 'HF optimum'. By default this is 0 hartree. It should be set to an estimate of the ground-state energy of the system under consideration.

VMC_OPTIMUM_EW (*Physical*) This keyword controls the width parameter used for optimum VMC sampling, which is enabled by setting **vmc_sampling** to 'optimum' or 'HF optimum'. By default this is 100 hartree. It should be set to an estimate of the expected width of the local energy distribution.

VMC_REEQUIL_NSTEP (*Integer*) Total number of steps to take for a VMC re-equilibration when doing a twist-averaged VMC run. Currently, a re-equilibration only takes place when the twist angle is changed. Notice that this is a single-process quantity: all MPI processes run **vmc_equil_nstep** equilibration steps. Also notice that the value of **vmc_decorr_period** is ignored in re-equilibrations. Electron(–hole) fluid phases only. See Sec. 28.

VMC_SAMPLE_HF (*Logical*) Setting **vmc_sample_hf** to `T` causes CASINO to ignore the Jastrow factor during the accept/reject step, which results in the HF wave function being sampled. The Jastrow factor is still used to evaluate the local energy, so the reported VMC energy is an estimate of $\langle E_{\rm SJ}\rangle_{|\Psi_{\rm HF}|^2}$, which is the reference energy in similarity-trasformed FCIQMC calculations. This keyword lets one run consistency checks between VMC and ST-FCIQMC, and optimize Jastrow factors by minimizing the spread of $E_{\rm SJ}$ over $|\Psi_{\rm HF}|^2$—set **opt_method** to `varmin`, `madmin`, or `varmin_linjas` for this.

VMC_SAMPLING (*Text*) This keyword allows the use of alternative sampling distributions instead of the square of the trial wave function in VMC and wave-function optimization. Each sampling distribution has its own set of advantages and disadvantages. Possible values of this keyword are: 'standard' (use the square of the wave function) (default), 'optimum' (use the optimum sampling distribution), 'HF optimum' (use the optimum sampling distribution for the HF wave function), and 'efficient' (use an inexpensive probability distribution for speed, currently only available in multideterminant calculations). See Sec. 26 for more information. For 'optimum' and 'HF optimum' sampling, see description of keywords **vmc_optimum_e0** and **vmc_optimum_ew**.

WDMCMIN, WDMCMAX (*Real*) These are the minimum and maximum allowed weights in weighted DMC. See entry for **lwdmc**.

WRITE_BINARY_BLIPS (*Logical*) The formatted blip data file `bwfn.data` can be very large. Consequently reading this file can be slow. Setting **write_binary_blips** to `T` will cause CASINO to write the binary file `bwfn.data.bin` provided there are no pre-existing `bwfn.data.bin` files in the run directory. When CASINO is run again it will first attempt to read `bwfn.data.bin` rather then `bwfn.data`, so start up will be faster. If one wishes to change the occupied orbitals delete the existing binary file `bwfn.data.bin` in the run directory. In a blip-generation calculation, this parameter determines whether a formatted `bwfn.data` file or an unformatted `bwfn.data.bin` file will be produced. This parameter is only relevant when the **atom_basis_type** = `blip` or when **runtype** is 'gen_blip'. Default value is `T`.

WRITEOUT_DMC_HIST (*Logical*) If **writeout_dmc_hist** is set to `T` (default) then the energy components, etc., are written to the file `dmc.hist` during a DMC simulation.

WRITEOUT_VMC_HIST (*Logical*) If **writeout_vmc_hist** is set to `T` (default) then the energy components etc. are written to the file `vmc.hist` during a VMC simulation. Furthermore, the `config.out` file required to continue a VMC calculation will only be produced if **writeout_vmc_hist** is `T`.

XC_CORR_METHOD (*Integer*) If **xc_corr_method** is set to 1, the XC finite-size correction will be evaluated by determining the coefficient of k^2 in the structure factor; if it is set to 2, the XC correction will be evaluated by fitting the whole structure factor. Method 1 (the default) is recommended. See Sec. 30 for further information.

7.3.2 Old format input keywords

CASINO understands an alternative set of keywords left over from its early development days which are listed below. You must either use the set of keywords below or their equivalents in the above list; it is not allowed to mix both. If you choose to use this 'old format' set, CASINO will tell you what the equivalent current keywords and values are.

Please note that support for these keywords will be **removed** in a version of CASINO expected to be published around January 2015. A utility called INPUT_KW_CONV exists to help you change the 'old' keyword set to the 'new format' set (and, somewhat pointlessly, *vice versa*).

CORPER Same as **vmc_decorr_period**.

CORPER_DMC Same as **dmc_decorr_period**.

NBLOCK Same as **vmc_nblock**, or same as **vmc_ntwist** when doing twist-averaging runs.

NBLOCK_DMC_EQUIL Same as **dmc_equil_nblock**.

NBLOCK_DMC_STATS Same as **dmc_stats_nblock**.

NBLOCK_DMCT_EQUIL Same as **dmc_reequil_nblock**.

NCONFIG Same as **dmc_target_weight**/*nproc*.

NCONFIG_PRELIM Same as **dmc_nconf_prelim**/*nproc*.

NDMCAVE Same as **dmc_ave_period**$*$**dmc_decorr_period**.

NEQUIL Same as **vmc_equil_nstep**.

NEQUIL_TA Same as **vmc_reequil_nstep**.

NMOVE Same as **vmc_nstep**/(**vmc_nblock**$*$**dmc_ave_period**$*$*nproc*).

NMOVE_DMC_EQUIL Same as **dmc_equil_nstep**/(**dmc_equil_nblock**$*$**dmc_decorr_period**).

NMOVE_DMC_STATS Same as **dmc_stats_nstep**/(**dmc_stats_nblock**$*$**dmc_decorr_period**).

NMOVE_DMCT_EQUIL Same as **dmc_reequil_nstep**.

NUM_DMC_TWISTS Same as **dmc_ntwist**.

NVMCAVE Same as **vmc_ave_period**.

NWRCON Same as **vmc_nconfig_write**/*nproc*.

PARALLEL_KEYWORDS When set to `total`, modifies the meaning of the other keywords described in this list in a way equivalent to setting $nproc = 1$ in the definitions (this was a first attempt at simplifying the VMC/DMC input; alternatively the 'new' keyword set can be used, and hence this keyword has now been flagged as REDUNDANT).

TRIP_POPN Same as **dmc_trip_weight**/*nproc*.

VMC_TWIST_AV Flag set whenever **vmc_ntwist**> 0.

7.4 Optimizable-parameter file: `correlation.data`

7.4.1 The `correlation.data` file

The `correlation.data` file contains all the optimizable parameters in the trial wave function that allow the QMC wave function to improve upon the wave function produced by the generating code. Specifically, the file contains the parameters that define

- a *Jastrow factor* (see Sec. 7.4.2)
- a *backflow function* (see Sec. 7.4.4)
- determinant-expansion coefficients (see Sec. 7.4.5)
- modifications to numerical atomic orbitals (see Sec. 7.4.7)
- modifications to HF/DFT molecular orbitals represented in a Gaussian basis (see Sec. 7.4.8)
- parameters relating to HEGs and pairing wave functions (see Secs. 7.4.9 and 7.4.10).
- a custom 'Mahan' wave function for describing impurity-in-HEG systems (see Sec. 7.4.11).

The sets may be given in any order, and not all sets need to be given. The `correlation.data` file may also contain a header block, with information about the contents. The format is:

```
START HEADER
  Correlation data for MgSiO3 calculations.
  Can have several lines, if desired.
END HEADER
```

A very detailed specification of the format of this file can be found in Appendix D.

Note that CASINO also contains a second implementation of Jastrow factor made up of terms of arbitrary rank, spin-dependencies and functional bases, with fully implemented analytic derivatives. This is stored in a file called `parameters.casl`, where 'casl' refers to the CASINO *Serialization Language*; see Sec. 7.7. Users may activate this Jastrow by setting the **use_gjastrow** input flag to `T`.

7.4.2 Jastrow factor

The format of the Jastrow-factor data set is shown below. This represents the state of the data set in `correlation.data` *before* optimization: the variable parameters are not given explicitly and are therefore assumed by CASINO to be zero. Here are 3 of the 6 possible Jastrow terms, any of which can be added or removed.

```
START JASTROW
 Title
   Title of system goes here.
 Truncation order
   3
 START U TERM
  Number of sets
    1
  START SET 1
   Spherical harmonic l,m
     0 0
   Expansion order
     8
   Spin-dep params (0->uu=dd=ud; 1->uu=dd/=ud; 2->uu/=dd/=ud)
     1
   Cutoff (a.u.)     ;  Optimizable (0=NO; 1=YES)
     0.d0                          1
   Parameter values  ;  Optimizable (0=NO; 1=YES)
  END SET 1
 END U TERM
 START CHI TERM
  Number of sets; labelling (1->atom in s cell; 2->atom in p cell; 3->species)
    2  1
  START SET 1
   Spherical harmonic l,m
     0 0
   Number of atoms in set
     1
   Labels of the atoms in this set
     1
   Impose electron-nucleus cusp (0=NO; 1=YES)
     0
   Expansion order
     6
   Spin-dep params (0->u=d; 1->u/=d)
     0
   Cutoff (a.u.)     ;  Optimizable (0=NO; 1=YES)
     0.d0                            0
   Parameter values  ;  Optimizable (0=NO; 1=YES)
  END SET 1
  START SET 2
   Spherical harmonic l,m
     0 0
   Number of atoms in set
     4
   Labels of the atoms in this set
     2 3 4 5
   Impose electron-nucleus cusp (0=NO; 1=YES)
     0
   Expansion order
     6
   Spin-dep params (0->u=d; 1->u/=d)
     0
   Cutoff (a.u.)     ;  Optimizable (0=NO; 1=YES)
     0.d0                            0
   Parameter values  ;  Optimizable (0=NO; 1=YES)
  END SET 2
 END CHI TERM
 START F TERM
  Number of sets; labelling (1->atom in s cell; 2->atom in p cell; 3->species)
    2  1
  START SET 1
   Number of atoms in set
     1
   Labels of the atoms in this set
     1
   Prevent duplication of u term (0=NO; 1=YES)
     0
   Prevent duplication of chi term (0=NO; 1=YES)
     0
   Electron-nucleus expansion order
     2
   Electron-electron expansion order
     2
   Spin-dep params (0->uu=dd=ud; 1->uu=dd/=ud; 2->uu/=dd/=ud)
     0
   Cutoff (a.u.)     ;  Optimizable (0=NO; 1=YES)
     0.d0                            0
   Parameter values  ;  Optimizable (0=NO; 1=YES)
  END SET 1
  START SET 2
   Number of atoms in set
     4
   Labels of the atoms in this set
     2 3 4 5
   Prevent duplication of u term (0=NO; 1=YES)
     0
```

```
     Prevent duplication of chi term (0=NO; 1=YES)
       0
     Electron-nucleus expansion order
       2
     Electron-electron expansion order
       2
     Spin-dep params (0->uu=dd=ud; 1->uu=dd/=ud; 2->uu/=dd/=ud)
       0
     Cutoff (a.u.)     ;  Optimizable (0=NO; 1=YES)
       0.d0                           0
     Parameter values  ;  Optimizable (0=NO; 1=YES)
    END SET 2
   END F TERM
  END JASTROW
```

There are two additional terms which may be of benefit in periodic systems:

```
  START P TERM
   Spin dep (0->uu=dd=ud; 1->uu=dd/=ud; 2->uu/=dd/=ud)
     1
   Number of simulation-cell G-vectors (NB, cannot have both G & -G)
     6
   G-vector (in terms of rec latt vects) ; label
      0      0     -1                      1
      0     -1      0                      1
     -1      0      0                      1
      1      1      1                      1
      1      1      0                      2
      1      0      1                      2
   Parameter value ; Optimizable (0=NO; 1=YES)
  END P TERM
  START Q TERM
   Spin dep (0->u=d; 1->u/=d)
     0
   Number of primitive-cell G-vectors (NB, cannot have both G & -G)
     10
   G-vector (in terms of rec latt vects) ; label
      0     -1      1                     -2
     -1      1      0                     -2
     -1      0      1                     -2
      1      1      1                     -1
     -1      0      0                      1
      0      0     -1                      1
      0     -1      0                      1
      1      1      2                      2
      2      1      1                      2
      1      2      1                      2
   Parameter value ; Optimizable (0=NO; 1=YES)
  END Q TERM
```

A general three-body polynomial H term exists:

```
 START H TERM
 Number of sets
   1
 START SET 1
 Spherical harmonic l,m
   0 0
 Expansion order N_h
   3
 Spin dep (0->uu=dd=ud; 1->uu=dd/=ud; 2->uu/=dd/=ud)
   -1
 Cutoff (a.u.)     ;  Optimizable (0=NO; 1=YES)
   0.d0                           1
 Parameter values  ;  Optimizable (0=NO; 1=YES)
```

```
END SET 1
END H TERM
```

where use is made of spin-triplet dependencies as defined in the **custom_striplet_dep** input block. Alternatively, in homogeneous systems, a three-body W term can be included:

```
 START W TERM
 Number of sets
   1
 START SET 1
 Spherical harmonic l,m
   0 0
 Expansion order N_w
   6
 Spin dep (0->uu=dd=ud; 1->uu=dd/=ud; 2->uu/=dd/=ud)
   0
 Cutoff (a.u.)     ;  Optimizable (0=NO; 1=YES)
   0.d0                        1
 Parameter values  ;  Optimizable (0=NO; 1=YES)
 END SET 1
 END W TERM
```

In general we recommend the use of the three-body H term rather than the W term.
In 2D-periodic systems a cylindrical two-body term $u_{\rm cyl}$ can be used:

```
START UCYL TERM
Expansion order N_ucylrho
  3
Expansion order N_ucylz
  3
Spin dep (0->uu=dd=ud; 1->uu=dd/=ud; 2->uu/=dd/=ud)
  1
Axis polar angle theta   ;  Optimizable (0=NO; 1=YES)
 0.00000000000000000             0
Axis azimuthal angle phi ;  Optimizable (0=NO; 1=YES)
  0.0000000000000000             0
Radial cutoff (a.u.)     ;  Optimizable (0=NO; 1=YES)
  0.0                            0
Axial cutoff (a.u.)      ;  Optimizable (0=NO; 1=YES)
  0.0                            0
Parameter values  ;  Optimizable (0=NO; 1=YES)
END UCYL TERM
```

In 2D-periodic systems a cylindrical one-body term $\chi_{\rm cyl}$ can be used:

```
START CHICYL TERM
Expansion order N_chicylrho
  3
Expansion order N_chicylz
  3
Spin dep (0->u=d; 1->u/=d)
  1
Origin x, y, z (3 lines) ;  Optimizable (0=NO; 1=YES)
  0.0000000000000000             1
  0.0000000000000000             1
  0.0000000000000000             1
Axis polar angle theta   ;  Optimizable (0=NO; 1=YES)
  0.0000000000000000             1
Axis azimuthal angle phi ;  Optimizable (0=NO; 1=YES)
  0.0000000000000000             1
Radial cutoff (a.u.)     ;  Optimizable (0=NO; 1=YES)
  0.0                            1
Axial cutoff (a.u.)      ;  Optimizable (0=NO; 1=YES)
```

```
  0.0                                1
Parameter values  ;  Optimizable (0=NO; 1=YES)
END CHICYL TERM
END JASTROW
```

There is an (experimental) neural network Jastrow term:

```
START NNJAS TERM
Number of layers (excluding input layer)
          3
Number of neurons in each layer (INC input layer, where #neurons is #features)
          6           6           6          1
Activation functions for each layer
 ELU ELU IDENTITY
Input features
 U0C2_SD1_ST1 U0C2_SD1_ST2 CHI0C2_IT1 CHI0_C2_IT2 U2C2 CHI2C2
Feature parameters ;    Optimizable (0=NO; 1=YES)
  4.0           0        ! L_U0C2
  4.0           0        ! L_U0C2
  4.0           0        ! L_CHI0C2
  4.0           0        ! L_CHI0C2
  4.0           0        ! L_U2C2
  4.0           0        ! L_CHI2C2
Weights and biases ;    Optimizable (0=NO; 1=YES)
END NNJAS TERM
```

Notes:

- There are 8 standard types of term available: u (isotropic electron–electron terms), χ (isotropic electron–nucleus terms), f (isotropic electron–electron–nucleus terms), p (plane-wave expansions in electron–electron separation), q (plane-wave expansions in electron position), H/W (isotropic, homogeneous three-electron term), $u_{\rm cyl}$ (cylindrical electron–electron terms) and $\chi_{\rm cyl}$ (cylindrical electron–nucleus terms). All of these terms are optional: e.g., omitting all the lines from 'START Q TERM' to 'END Q TERM' will give a Jastrow factor with no q terms. In many cases (particularly in the presence of pseudopotentials) only u and χ terms are needed, and this is good since higher-order terms such as f-functions can become very expensive in large systems.

- The 'truncation order' should be either 2 or 3. If it is of value C then the Jastrow factor is C times differentiable everywhere; it must therefore be at least 2 for the kinetic-energy integrand to be well-defined. If it is 3 then the local energy is continuous in configuration space (assuming the orbitals are smooth). This is not strictly required, and leads to the loss of some variational freedom, but it makes the numerical optimization of cutoff lengths easier. In general, we recommend setting $C = 3$.

- In a future release, the u and χ terms may be made anisotropic; there are place-holders for this in the Jastrow data set. At present, however, both must be isotropic. There should only be one set of u terms, and the spherical harmonic l and m values should therefore be set to 0.

- The 'expansion order' of u determines the number of parameters. Typically it is given a value between 4 and 8.

- The 'spin-dep params' line allows the user to specify whether the same u-parameters are to be used for parallel- and antiparallel-spin electron-pairs. If the value is set to 0 then the same parameters are used for parallel and antiparallel pairs (this option should *not* be used in general); if the value is set to 1 then different parameters are used for parallel and antiparallel pairs; if it is set to 2 then different parameters are used for up-spin, down-spin and opposite-spin pairs: this is useful for spin-polarized systems.[10]

- The cutoff length is given a default value if it is set to 0. Note that it is possible to optimize the cutoff length using variance minimization. This can be useful, because the choice of cutoff

[10]When particles other than electrons are present in the systems, introduced via the **particles** block input keyword, these definitions change a little. Information on the particle and particle-pair splittings are given near the beginning of the `out` file.

length has a significant effect on the optimized energy and variance. Unfortunately, optimizing the cutoff length is numerically difficult: variance minimization will take many more iterations to converge if the cutoff is optimizable. Setting the truncation order to 3 can help somewhat.

- It is possible to specify exactly which of the expansion coefficients ('parameter values') are optimizable and which are not. One does not need to specify all of the expansion coefficients: any that are not listed are assumed to be zero. If all of the parameter values in all of the Jastrow terms are blank, as is the case in the example given above, then the following default behaviour will be used: if one is studying a homogeneous system, or a 3D-periodic system or is using energy minimisation, a simple default for u will be chosen that satisfies the Kato cusp conditions; otherwise, only the Slater wave function will be used for the first configuration generation run when performing wave-function optimization.[11]

- Different χ functions are used for different species of ion: the 'number of sets' can be chosen to be equal to the number of chemically distinct species. One can optionally specify how the atoms in each set are to be labelled. The 'labelling' flag can be set to '1', meaning that each atom is labelled by its number within the simulation cell, or '2', meaning that groups of atoms are labelled by their number within the primitive cell, or '3', meaning that groups of atoms are labelled by their species.

- The ions in each set are specified by giving a list of the numbers that label them. If the labelling flag is set to '1' (the default if the labelling flag is absent: see above) then, in the first primitive cell, the atom labels are the same as the labels used in the `xwfn.data` file that specifies the geometry of the system; the atoms in the subsequent primitive cells are labelled in the same order as the first. If the labelling flag is set to '2' then the atom labels are the same as those used in the `xwfn.data` file (so the same χ functions are used for translationally equivalent atoms within the supercell). If the labelling flag is set to '3' then the atom labels refer to species (in the order in which the species occur in the `xwfn.data` file), so that all atoms of the same species have the same χ functions.

- It is possible to make the Jastrow factor enforce the electron–nucleus cusp condition. This should only be done if the χ-set contains bare nuclei and the orbitals do not satisfy the cusp condition. With a Gaussian basis set, it is much better to use the in-built cusp correction algorithm activated with the `input` keyword **cusp_correction** than to use the Jastrow. Likewise, numerical atomic orbitals already satisfy the electron–nucleus cusp conditions.

- There are two spin-dependence options for χ: if it is 0 then the same parameters are used for up- and down-spin electrons; if it is 1 then different parameters are used.

- Similar comments to those made for u apply to the 'spherical harmonic' labels, the cutoff length and the parameter values of χ.[12]

- The f function contains terms that approximately duplicate the u and χ terms: additional constraints can be placed on f to remove these terms if desired. Duplication of u and χ should generally be permitted, however.

- The number of f-parameters grows very rapidly with the electron–electron and electron–nucleus expansion orders, which should normally be either 2 or 3. Note also that calculating f-functions can be very expensive in large systems, particularly if the cutoffs are allowed to get too big.

- The spin-dependence options for f are exactly the same as for u.

- Optionally, one can specify a "secondary spin dependence" for f. This is a single-particle spin-dependence that is used to decide whether $f(r_i, r_j, r_{ij})$ should be symmetric under exchange of r_i and r_j; f will be symmetric if and only if i and j belong to the same single-particle group. f must be symmetric under exchange of same-spin particles, but is allowed to be asymmetric under exchange of unlike particles. By default, f is always symmetric. An example of a situation in which f should be allowed to be asymmetric would be for electron–positron–nucleus f terms in a non-spin-polarised positronic molecule. In this case the spin dependence line in each f set could be given as

[11]The u-parameters are listed in the following order for electron systems: coefficients for spin-up pairs; coefficients for antiparallel pairs (if spin-dependence is 1 or 2); coefficients for spin-down pairs (if spin-dependence is 2). For each spin-pair, the number of parameters is equal to the expansion order.

[12]For electron systems, the parameter values are given for spin-up electrons, then (if the spin-dependence is 1) for spin-down electrons. For each spin-type, the number of parameters is equal to the expansion order.

Table 1: Feature labels for the NNJAS Jastrow term and the corresponding features. The cutoff lengths $\{L\}$ are optimizable parameters.

Label	Form
`U0C2[_SD⟨spin-dep⟩_ST⟨spin-type⟩]`	$\sum_{i>j}(1+2r_{ij}/L)(r_{ij}-L)^2\Theta(L-r_{ij})$
`U2C2[_SD⟨spin-dep⟩_ST⟨spin-type⟩]`	$\sum_{i>j} r_{ij}^2(r_{ij}-L)^2\Theta(L-r_{ij})$
`CHI0C2[_SD⟨spin-dep⟩_ST⟨spin-type⟩][_IT⟨ion-type⟩]`	$\sum_I\sum_i(1+2r_{iI}/L)(r_{iI}-L)^2\Theta(L-r_{iI})$
`CHI2C2[_SD⟨spin-dep⟩_ST⟨spin-type⟩][_IT⟨ion-type⟩]`	$\sum_I\sum_i r_{iI}^2(r_{iI}-L)^2\Theta(L-r_{iI})$

```
Spin-dep params
  1 1
```

where the first "1" chooses the spin-pair dependence in which the same-spin pairs are grouped, opposite-spin electron pairs are grouped and electron–positron pairs are grouped, while the second "1" is a spin-single dependence in which the electrons are grouped and the positron is in a group by itself. This choice of spin-dependence and secondary spin-dependence implies that there would be symmetric f terms for same-spin pairs, symmetric f terms for opposite-spin electron pairs, and asymmetric f terms for electron-positron pairs.

- The p and q terms can only be present in periodic systems. The p term makes a small but significant improvement to the wave function in most cases, whereas the q term does not appear to help much. It is especially important to use a p term if the finite-size correction to the kinetic energy is to be calculated (see Sec. 29). Note that q should only be used if the origin is a centre of inversion symmetry of the charge density.

- The spin-dependence options for p and q are the same as those of u and χ respectively.

- For p, a list of simulation-cell **G**-vectors must be provided. These are specified in terms of the reciprocal-lattice vectors. Only one out of each **G** and −**G** should be specified. The same parameter value is used for **G**-vectors with the same label. Note that the `make_p_stars` utility can be used to generate p terms to paste into the `correlation.data` file.

- For q, a list of primitive-cell **G**-vectors must be provided. Again, **G**-vectors with the same label have the same parameter value. It is possible to specify a negative relationship between parameter values by using a negative label for the appropriate **G**-vectors.

- For $u_{\rm cyl}$ terms, separate cutoff lengths and expansion orders for the radial and axial terms must be given. See Sec. 22.5.

- For $\chi_{\rm cyl}$ terms, separate cutoff lengths and expansion orders for the radial and axial terms must be given. See Sec. 22.6.

- The direction of the cylindrical axis for the $u_{\rm cyl}$ and $\chi_{\rm cyl}$ terms is specified by giving the spherical polar and azimuthal angles θ and ϕ of the cylindrical axis with respect to the Cartesian axes. For a 2D-periodic system, the cylindrical axis should point in the z direction; hence θ should be fixed at zero. For a 1D-periodic system, the axis should point in the x direction; hence θ should be fixed at $\pi/2$ and ϕ should be fixed at zero. For nonperiodic systems the appropriate direction of the cylindrical axis may be clear on symmetry grounds; if not, the values of θ and ϕ can be optimized. For 3D-periodic systems the values of θ, ϕ and the cutoff lengths should be chosen with care; CASINO does not currently check whether they are compatible with the periodic boundary conditions.

- There is an experimental neural network Jastrow term given in an 'NNJAS' block. In this term, electron-position-dependent 'features' provide the input layer to a fully connected neural network, with the weights and biases being optimizable parameters. The currently available activation functions for each layer are: `IDENTITY` ($x=y$), `TANH` ($x=\tanh(y)$), `SOFTPLUS` ($x=\log(1+\exp(y))$), `ELU` ($x=y\Theta(y)+[\exp(y)-1]\Theta(-y)$), `SILU` ($x=y/[1+\exp(-y)]$) and `CUBIC` ($x=y+y^3$). We suggest using `IDENTITY` for the output layer and `ELU` for hidden layers. The currently available input features are listed in Table 1. Other input features can be implemented on request.

7.4.3 Nonstandard Jastrow terms

Additional Jastrow terms are available for model 2D or 3D (only EXJAS) excitonic systems: these are called 'BIEX1', 'BIEX2', 'BIEX3' and 'EXJAS'.

The BIEX1 term is appropriate for a biexciton consisting of two opposite-spin electrons and two opposite-spin holes moving in 2D, where the electrons are confined to one layer and the holes are confined to another. This models a spatially indirect biexciton in a coupled quantum well III–V heterostructure. The form of the BIEX1 term is defined in Ref. [27]. The BIEX2 term is a simpler version of BIEX1. The BIEX3 term provides a Jastrow factor for two interacting 2D excitons whose centres of mass are pinned a fixed distance apart: see Ref. [28].

The EXJAS term is a flexible two-body term for describing bound charge-carrier complexes in 2D or 3D systems including gapped 2D semiconductors (transition-metal dichalcogenides, indium/gallium chalcogenides, phosphorene, etc.). The screened interaction (in 2D only) between the charge carriers is generally of the Keldysh form discussed in Sec. 20.6.1, although the EXJAS term can also be used for the unscreened $1/r$ Coulomb interaction (in both 2D and 3D) between charge carriers. The EXJAS term can be used between any pairs of free or fixed particles; hence one can study excitons, trions, biexcitons, donor-bound excitons, donor-bound trions, donor-bound biexcitons, ... The EXJAS term satisfies the Kato cusp conditions for Coulomb interactions and the analogue of the Kato cusp conditions for the Keldysh interaction (see Sec. 20.6.1). At long range the EXJAS two-body Jastrow term falls off as $-r$, resulting in a wave function that is exponentially localised; thus the complex being simulated is in principle forced to be bound (although other terms in the Jastrow factor can try to fight this).

The form of the two-body EXJAS terms for particles interacting via the Keldysh interaction is

$$u_{\rm exjas}(r) = \frac{[c_1 + \Gamma \log(r) + c_2 r] r^2}{1 + c_3 r^2}, \tag{2}$$

where c_1, c_2 and c_3 are optimizable parameters, with $c_2 \leq 0$ and $c_3 \geq 0$, and $\Gamma = -q_i q_j m_i m_j / [2 r_* (m_i + m_j)]$ for distinguishable pairs of particles of charge q_i and q_j and mass m_i and m_j, with r_* being the parameter defined in Sec. 20.6.1. The same wave-function form is used for the logarithmic interaction, but the choice of units implies that r_* is absent from the expression for Γ. For indistinguishable particles of mass m and charge q, $\Gamma = -mq^2/(8r_*)$ (again without r_* for the logarithmic interaction). If the interaction between the charge carriers is of Coulomb $1/r$ form then

$$u_{\rm exjas}(r) = \frac{\Gamma r + c_1 r^2}{1 + c_2 r}, \tag{3}$$

where $c_1 \leq 0$ and $c_2 \geq 0$ are optimizable parameters, and $\Gamma = 2 q_i q_j m_i m_j / [(d-1)(m_i + m_j)]$ for distinguishable pairs of particles of mass m_i and m_j and charge q_i and q_j and d is the dimensionality (2 or 3). For indistinguishable pairs of particles of charge q and mass m, $\Gamma = q^2 m/2$.

At present the EXJAS term is only implemented for the situation in which all free particles are distinguishable. In this case the ground-state wave function is nodeless and hence DMC is exact.

The format of the EXJAS Jastrow block in `correlation.data` is

```
START EXJAS TERM
Pair spin-dependence
          1
Single spin-dependence
          1
Parameter value ; Optimizable (0=NO; 1=YES)
END EXJAS TERM
```

Spin-dependences for the 'pair' terms between free particles and the 'single' terms between free and fixed particles are chosen in the same way that one would choose the spin-dependences for the u and χ terms, respectively.

When the EXJAS term is used, the charge carriers should be defined in the **free_particles** and **fixed_particles** blocks in `input`. In the former block, the particles should be chosen to be 'free', because their wave function is entirely of Jastrow form. If fixed particles are present then the χ and f terms in the Jastrow factor can be used.

The varmin-linjas method cannot be used to optimise free parameters in the BIEX or EXJAS terms, although it can be used to optimise other terms in the Jastrow factor in the presence of the BIEX and EXJAS terms.

Another nonstandard Jastrow term 'UHARM' is available for pairwise correlations between heavy particles with repulsive interactions (i.e., situations in which the Born–Oppenheimer approximation is appropriate for the heavy particles). This is a pairwise Jastrow term of the form $U_{\rm harm} = \sum_{i>j} u_{\rm harm}(r_{ij})$, where $u_{\rm harm}(r) = -c(r-r_0)^2$. The optimizable parameter r_0 is indicative of the equilibrium bond length, while the parameter c is related to the vibrational frequency. The Jastrow factor $\exp(U_{\rm harm})$ arising from this term is a Gaussian vibrational ground-state wave function of interparticle distance. The canonical example of the sort of system for which UHARM is useful is a dihydrogen molecule, formed of two protons and two electrons; in this case it would be appropriate to have a UHARM term between the protons.

The format of the UHARM Jastrow block (for a D_2 molecule) is

```
START UHARM TERM
Spin dep (0->uu=dd=ud; 1->uu=dd/=ud; 2->uu/=dd/=ud)
  1
r0 parameter values  ;  Optimizable (0=NO; 1=YES)
  1.0000000000000000               0        ! r0_1
  1.0000000000000000               0        ! r0_2
  1.3489599285818805               1        ! r0_3
c parameter values  ;  Optimizable (0=NO; 1=YES)
  0.0000000000000000               0        ! c_1
  0.0000000000000000               0        ! c_2
  33.608103257293003               1        ! c_3
END UHARM TERM
```

Notes:

- Both the set of r_0 parameters and the set of c parameters can be left blank, in which case default values will be supplied. By default c is fixed at zero for all except the heaviest pair of particle types with opposite charges.
- The UHARM term introduces a cusp in the wave function at the coalescence point; this cusp violates the cusp conditions, which could in principle create problems for DMC. In practice, in the circumstances in which the UHARM term is to be used, the particles involved should remain close to r_0 and should not approach each other.
- For particle pairs that do not involve heavy nuclei, the corresponding c value should be fixed at zero.

7.4.4 Backflow parameters

The parameters in the backflow functions are supplied to CASINO in a data set in the `correlation.data` file. The format is analogous to that of the Jastrow-factor data set. Please see the notes in Sec. 7.4.2 for information on the 'truncation order', 'spin dependence', etc.

```
START BACKFLOW
Title
AE Carbon atom - empty parameter set
Truncation order
  3
START ETA TERM
Expansion order
  8
Spin dep (0->uu=dd=ud; 1->uu=dd/=ud; 2->uu/=dd/=ud)
  1
Cutoff radii ;       Optimizable (0=NO; 1=YES; 2=YES BUT NO SPIN-DEP)
  0.d0                           1
Parameter ;          Optimizable (0=NO; 1=YES)
END ETA TERM
START MU TERM
Number of sets; labelling (1->atom in s cell; 2->atom in p cell; 3->species)
```

```
  1   2
 START SET 1
 Number of atoms in set
  1
 Labels of the atoms in this set
   1
 Type of e-N cusp conditions (0->PP/cuspless AE; 1->AE with cusp)
  1
 Expansion order
  6
 Spin dep (0->u=d; 1->u/=d)
  1
 Cutoff (a.u.)     ;  Optimizable (0=NO; 1=YES)
  4.5                          1
 Parameter values  ;  Optimizable (0=NO; 1=YES)
 END SET 1
 END MU TERM
 START PHI TERM
 Number of sets; labelling (1->atom in s cell; 2->atom in p cell; 3->species)
  1  2
 START SET 1
 Number of atoms in set
  1
 Labels of the atoms in this set
   1
 Type of e-N cusp conditions (0=PP; 1=AE)
  1
 Apply no-curl constraint (0=NO; 1=YES)
  1
 Electron-nucleus expansion order N_eN
  3
 Electron-electron expansion order N_ee
  3
 Spin dep (0->uu=dd=ud; 1->uu=dd/=ud; 2->uu/=dd/=ud)
  0
 Cutoff (a.u.)     ;  Optimizable (0=NO; 1=YES)
  0.d0                         1
 Parameter values  ;  Optimizable (0=NO; 1=YES)
 END SET 1
 END PHI TERM
 START AE CUTOFFS
 Nucleus ; Set ; Cutoff length     ;  Optimizable (0=NO; 1=YES)
 END AE CUTOFFS
 START PI TERM
 Spin dep (0->uu=dd=ud; 1->uu=dd/=ud; 2->uu/=dd/=ud)
          0
 Number of simulation-cell G-vectors (NB, cannot have both G & -G)
          9
 G-vector (in terms of rec latt vects) ; label
      0     0     1       1
      1     0     0       1
      0     1     0       1
      1     0     1       2
      1     0    -1       2
      0     1     1       2
      0     1    -1       2
      1     1     0       2
      1    -1     0       2
 Parameter value ; Optimizable (0=NO; 1=YES)
 END PI TERM
 END BACKFLOW
```

Detailed information about each of the backflow functions can be found in Sec. 23.

Notes:

- The polynomial form of η requires the specification of an expansion order as shown above. The two lines have to be removed when using any of the other two forms.
- The η function has separate cutoff lengths for each of the spin-dependencies. The first one must be specified; every cutoff length not listed is initially set to the value of the first one. Set the 'Optimizable' flag to 2 to constrain all cutoff lengths to be the same during optimization too.
- 'Type of e-N cusp conditions' in μ and Φ is used to specify whether to regard the atoms in the set as all-electron (AE) or pseudo-atoms (PP). PP cusp conditions are less restrictive than AE ones. The user should set this flag to 0 only if a pseudopotential is being used for the atom in question.
- 'Apply no-curl constraint' in Φ is used to reduce the number of parameters in Φ and Θ when this number is too large for optimization to succeed. In principle, this flag should always be set to zero for the best results.
- It is recommended that **vm_filter** be set to `T` in the `input` file when optimizing backflow parameters using unreweighted variance minimization.
- The `AE CUTOFFS` section will appear automatically in systems with all-electron atoms, and all cutoffs will be flagged as optimizable by default.
- The plane-wave term Π is of the form of the gradient of the Jastrow P term. The `PI TERM` block in the `BACKFLOW` section of `correlation.data` is of exactly the same format as the `P TERM` block in the `JASTROW` section. The Π term can be generated using the MAKE_P_STARS utility (and replacing "P TERM" with "PI TERM").

7.4.5 Excitations and multideterminant-expansion coefficients

The `xwfn.data` file contains data that define the orbitals produced by the wave-function generating code. The file also specifies a *reference configuration* (the form of the Slater wave function), which may consist of one or more determinants. The 'MDET' block in the `correlation.data` file allows one either to use the reference configuration or to specify excitations, additions or subtractions from the reference configuration. Furthermore, it enables the user to construct an expansion in several determinants with optimizable expansion coefficients. The MDET block consists of a title followed by an arbitrary number of lines (up to 'END MDET'), which contain tokens used to specify the excitations and the multideterminant configuration.

The simplest case is when we want to use exactly the reference configuration specified in `xwfn.data` (this is the default behaviour if no MDET block is supplied). In this case, the single token 'GS' is used:

```
START MDET
 Title
   MDET example: use reference configuration in xwfn.data file.
 Multideterminant/excitation specification (see manual)
   GS
END MDET
```

Promotions of electrons are specified using the keyword 'PR'. Several changes to the reference configuration can be made at once. Here is an example of a single-determinant ('SD') excited-state calculation:

```
START MDET
 Title
   MDET example: use a single-determinant excited state.
 Multideterminant/excitation specification (see manual)
   SD
   DET 1 2 PR 3 4 5 6
END MDET
```

For electron gas, and additionally for plane-wave and blip basis calculations, it is also currently possible to explicitly specify additions and subtractions. For example, if one were to increment the `input` parameter **neu**, the additional electron in a single-determinant plane-wave or blip calculation would be added to the orbital with smallest associated eigenvalue (supplied in the `xfwn.data` file by wave function generating code). If one wishes to explicitly specify the band and **k**-point to which the additional spin-up electron is added, then one may use an MDET block of the form:

```
START MDET
 Title
   MDET example: use a single-determinant excited state.
 Multideterminant/excitation specification (see manual)
   SD
   DET 1 2 PL 3 4
END MDET

START MDET
 Title
   MDET example: single-determinant addition.
 Multideterminant/excitation specification (see manual)
   SD
   DET 1 2 PL 3 4
END MDET
```

Where the token 'PL' implies 'plus'. In this case, an extra electron would be added to band 3 at **k**-point 4 (if it is unoccupied, else an error will be raised). If one were to decrement the `input` parameter **neu**, then by default the electron would cease to occupy the band with the highest associated eigenvalue (as per the `xwfn.data` information). Similarly with electron additions, one could control where this electron is removed from by use of an MDET block of the form:

```
START MDET
 Title
   MDET example: single-determinant subtraction.
 Multideterminant/excitation specification (see manual)
   SD
   DET 1 2 MI 1 8
END MDET
```

Where the token 'MI' implies 'minus'. In this case, an electron would be removed from band 1 at **k**-point 8. Addition and subtraction tokens are particularly useful for generating triplet excitations, where the promotion specification will move electrons of a given spin between orbitals and **k**-points, two separate addition and subtraction specifications can facilitate the creation of a triplet. For example, in the below MDET block, we have specified a triplet excitation between bands 4 and 6 at **k**-point 1:

```
START MDET
 Title
   MDET example: triplet excitation at k-point 1.
 Multideterminant/excitation specification (see manual)
   SD
   DET 1 1 MI 4 1
   DET 1 2 PL 6 1
END MDET
```

This should of course be accompanied by some changes to the `input` parameters: a decrease of **neu** and an increase of **ned**.

Notes:

- 'DET 1 2 PR 3 4 5 6' means 'in determinant 1, spin 2 (down), promote an electron from band 3, **k**-point 4, to band 5, **k**-point 6'.
- For nonperiodic systems, the **k**-point indices should just be set to 1.
- For electron gas calculations, the band indices should just be set to 1.
- If one subtracts a spin-up electron then the value of the `input` parameter **neu** should be decreased accordingly. Likewise for spin-down electrons. Similarly if one adds an electron.
- Addition and subtraction tokens should not exist for the same spin and same determinant number. In this case, one should use promotion to generate the corresponding excitation.

- The addition specification will take an electron from the highest occupied orbital, and add it to the user-specified orbital.
- The subtraction specification will take an electron from the user-specified orbital, and add it to the lowest unoccupied orbital.

A multideterminant expansion is specified by the keyword 'MD':

```
START MDET
 Title
   MDET example: use a multideterminant expansion.
 Multideterminant/excitation specification (see manual)
   MD
   3
   1.d0   1   0
   0.5d0  2   1
 * 1.0d0  2   1
   DET 2 1 PR 4 5 6 7
   DET 3 2 PR 4 5 6 7
END MDET
```

Notes:

- The '3' in the line after 'MD' specifies that there are three determinants in the expansion.
- The next three lines contain the determinant expansion coefficients for the three determinants (1, 0.5 and 0.5). Each expansion coefficient is followed by a 'label' and then an 'optimizable' flag.
- The 'optimizable' flag must be 0 or 1, specifying that the expansion coefficient is fixed or free to be optimized.
- All coefficients with the same label must have the same 'optimizable' flag. The ratios of these coefficients will be fixed during the optimization. There is therefore only one optimizable parameter in the example above, since the coefficients of the second and third determinants are constrained to be equal.
- At least one determinant expansion coefficient must be fixed.
- The asterisk '`*`' preceding the coefficient of the third determinant indicates that the value is relative to the coefficient of the first determinant in the 'label', i.e., $c_3/c_2 = 1$ in this case, so $c_3 = 0.5$. If the asterisk is ommitted, the number would be interpreted as the coefficient of the third determinant. Using the asterisk notation is advantageous in that a configuration state function can be fully specified when its overall coefficient is set to zero. On output, the MDET block is always written using the asterisk notation.
- The excitation specifications are of the same form as for the 'SD' case. For example, 'DET 2 1 PR 4 5 6 7' means 'in determinant 2 promote an electron of spin 1 (up) from band 4, **k**-point 5 to band 6, **k**-point 7'.

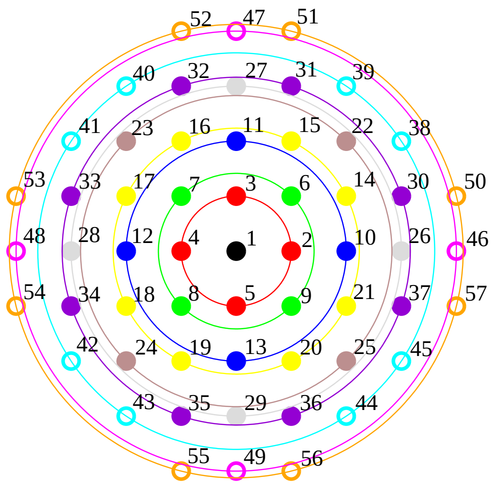
Multideterminant expansions and promotion excitations as described above are fully functional for Gaussian (`gwfn.data`), plane-wave (`pwfn.data`) and blip (`bwfn.data`) orbitals. They may also be used with numerical atomic (`awfn.data`) orbitals, although the excitation information must also be supplied in the `awfn.data` file (see Sec. 7.10.4).

If blip or plane-wave orbitals are used then the user may specify a phase angle for a particular band and **k** point and a particular determinant and spin using

```
ORB_PHASE <determinant> <spin> <band> <k point> <phase angle>
```

where the phase angle is given in radians. The phase angle is used in the construction of a real orbital when the band is occupied at **k** but not $-\mathbf{k}$, as described in Sec. 15. Specifying a phase has no effect when the band is occupied at both **k** and $-\mathbf{k}$, or when the band is unoccupied at **k**.

If one is studying a HEG and a complex wave function is used (i.e., **complex_wf** = `T`) then the orbitals in the Slater determinants are of the form $\exp(i\mathbf{k}\cdot\mathbf{r})$, where the $\{\mathbf{k}\}$ are simulation-cell reciprocal-lattice points offset by the constant $\mathbf{k}_{\rm offset}$ vector specified in the **free_particles** block in the `input` file. Before the occupancy of the plane-wave orbitals is worked out, the $\mathbf{k}$ vectors are sorted into increasing order of azimuthal angle $\phi_{\mathbf{k}}$ (innermost), then polar angle $\theta_{\mathbf{k}}$, then magnitude $|\mathbf{k}|$ (outermost). Hence one can specify excitations of electrons unambiguously. An example of the labelling of the $\{\mathbf{k}\}$ for a 2D HEG is shown below. Note that adding and subtracting electrons to or from particular states in a HEG is achieved by increasing or decreasing the electron number and making appropriate promotions. Note also that the angular sorting of the $\mathbf{k}$ vectors is not performed when the wave function is real, i.e., when **complex_wf** = `F`.



The labelling of the $\mathbf{k}$-vectors in a 2D HEG with a square cell. The filled points indicate the doubly occupied states in a 74-electron paramagnetic ground state. If one wishes to create an excitation of an up-spin electron from state 25 to state 47 in determinant 1, say, then one should include 'DET 1 1 PR 1 25 1 47' in the excitation specification in the MDET block in `correlation.data`.

7.4.6 Named special k-points

CASINO knows about various special points within the Brillouin zones of solids. One can use known special point labels in place of $\mathbf{k}$-point indices in excitation specifications, and if CASINO is able to identify a lattice in a periodic calculation with a gaussian, blip or plane-wave basis, it will print the labels (if any) of any special points it detects in the $\mathbf{k}$–point grid, as well as the name of the lattice.

In the following sections, three-dimensional coordinates for special points are given such that harmony with the source code is kept. Obviously, a one-dimensional Brillouin zone is not three dimensional, and the trailing zeroes are to be ignored. Supposing that the lattice vectors can generally be given by $\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3$, we will take

$$\begin{aligned} a &= \sqrt{\mathbf{v}_1 \cdot \mathbf{v}_1} \quad , \quad \alpha = \arccos\left(\mathbf{v}_1 \cdot \mathbf{v}_3/ac\right) \\ b &= \sqrt{\mathbf{v}_2 \cdot \mathbf{v}_2} \quad , \quad \beta = \arccos\left(\mathbf{v}_2 \cdot \mathbf{v}_3/bc\right) \\ c &= \sqrt{\mathbf{v}_3 \cdot \mathbf{v}_3} \quad , \quad \gamma = \arccos\left(\mathbf{v}_1 \cdot \mathbf{v}_2/ab\right) \end{aligned} \qquad (4)$$

with angles, as given, in radians.

One-dimensional Lattices

There is only one one-dimensional Bravais lattice. In this case CASINO designates special points (in the form "label" → "fractional reciprocal lattice coordinates") as:

$$\begin{aligned} \Gamma\ (\text{“G”}) &\rightarrow (0.0, 0.0, 0.0) \\ \text{X} &\rightarrow (0.5, 0.0, 0.0) \end{aligned}$$

Two-dimensional lattices

In two-dimensions there are five Bravais lattices; square, hexagonal, rectangular (primitive), rectangular (centred) and (the most general) oblique. CASINO recognises square, hexagonal, rectangular (primitive) and [currently the Γ ("G") point only] oblique lattices.

Square $a = b$, $\gamma = 90°$:

$$\begin{aligned} \Gamma\ (\text{“G”}) &\rightarrow (0.0, 0.0, 0.0) \\ \text{M} &\rightarrow (0.5, 0.5, 0.0) \\ \text{X} &\rightarrow (0.0, 0.5, 0.0) \\ \text{Y} &\rightarrow (0.5, 0.0, 0.0) \end{aligned}$$

Hexagonal (variant I) $a = b$, $\gamma = 60°$:

$$\begin{aligned} \Gamma\ (\text{“G”}) &\rightarrow (0.0, 0.0, 0.0) \\ \text{K} &\rightarrow \left(\frac{2}{3}, \frac{1}{3}, 0.0\right) \\ \text{M} &\rightarrow (0.5, 0.0, 0.0) \end{aligned}$$

Hexagonal (variant II) $a = b$, $\gamma = 120°$:

$$\begin{aligned} \Gamma\ (\text{“G”}) &\rightarrow (0.0, 0.0, 0.0) \\ \text{K} &\rightarrow \left(\frac{2}{3}, \frac{2}{3}, 0.0\right) \\ \text{M} &\rightarrow (0.5, 0.0, 0.0) \end{aligned}$$

Rectangular (primitive) $a \neq b$, $\gamma = 90°$:

$$\begin{aligned} \Gamma\ (\text{“G”}) &\rightarrow (0.0, 0.0, 0.0) \\ \text{S} &\rightarrow (0.5, 0.5, 0.0) \\ \text{X} &\rightarrow (0.0, 0.5, 0.0) \\ \text{Y} &\rightarrow (0.5, 0.0, 0.0) \end{aligned}$$

(incomplete) Oblique $a \neq b$, $\gamma \neq 90°$:

$$\Gamma\ (\text{“G”}) \rightarrow (0.0, 0.0, 0.0)$$

Three-dimensional lattices

In three dimensions there are fourteen Bravais lattices, which can be grouped into seven crystal systems: cubic, orthorhombic, tetragonal, monoclinic, rhombohedral, triclinic, and hexagonal. CASINO knows about the following crystal structures.

Simple cubic $a = b = c$, $\alpha = \beta = \gamma = 90°$:

$$\begin{aligned} \Gamma\ (\text{“G”}) &\rightarrow (0.0, 0.0, 0.0) \\ \text{M} &\rightarrow (0.5, 0.5, 0.0) \\ \text{X} &\rightarrow (0.0, 0.5, 0.0) \\ \text{R} &\rightarrow (0.5, 0.5, 0.5) \end{aligned}$$

Face centred cubic $a = b = c$, $\alpha = \beta = \gamma = 60°$:

$$\begin{aligned} \Gamma\ (\text{“G”}) &\rightarrow (0.0, 0.0, 0.0) \\ \text{X} &\rightarrow (0.0, 0.5, 0.5) \\ \text{L} &\rightarrow (0.5, 0.5, 0.5) \\ \text{W} &\rightarrow (0.25, 0.75, 0.5) \\ \text{U} &\rightarrow (0.25, 0.625, 0.625) \\ \text{K} &\rightarrow (0.375, 0.75, 0.375) \end{aligned}$$

Body centred cubic $a = b = c$, $\alpha = \beta = \gamma = 109.47\ldots°$:

$$\begin{aligned} \Gamma\ (\text{“G”}) &\rightarrow (0.0, 0.0, 0.0) \\ \text{H} &\rightarrow (-0.5, 0.5, 0.5) \\ \text{P} &\rightarrow (0.25, 0.25, 0.25) \\ \text{N} &\rightarrow (0.0, 0.5, 0.0) \end{aligned}$$

Hexagonal (variant I) $a = b = c$, $\alpha = \beta = 90°, \gamma = 60°$:

$$\begin{aligned}
\Gamma\ (\text{“G”}) &\rightarrow (0.0, 0.0, 0.0) \\
\text{A} &\rightarrow (0.0, 0.0, 0.5) \\
\text{K} &\rightarrow \left(\frac{2}{3}, \frac{1}{3}, 0.0\right) \\
\text{H} &\rightarrow \left(\frac{2}{3}, \frac{1}{3}, 0.5\right) \\
\text{M} &\rightarrow (0.5, 0.0, 0.0) \\
\text{L} &\rightarrow (0.5, 0.0, 0.5)
\end{aligned}$$

Hexagonal (variant II) $a = b = c$, $\alpha = \beta = 90°, \gamma = 120°$:

$$\begin{aligned}
\Gamma\ (\text{“G”}) &\rightarrow (0.0, 0.0, 0.0) \\
\text{A} &\rightarrow (0.0, 0.0, 0.5) \\
\text{K} &\rightarrow \left(\frac{2}{3}, \frac{2}{3}, 0.0\right) \\
\text{H} &\rightarrow \left(\frac{2}{3}, \frac{2}{3}, 0.5\right) \\
\text{M} &\rightarrow (0.5, 0.0, 0.0) \\
\text{L} &\rightarrow (0.5, 0.0, 0.5)
\end{aligned}$$

Somewhat generically, if a lattice is not identified, it will be given the name specified “unclassified” by CASINO. If this happens, the Γ point (labelled as G in excitation specifications) will still always be indexed (with obvious value. . .).

Excitation specification example

Suppose one wishes to make a promotion of a spin-up electron from band 4 at the Γ point (in Si, for example) to band 5 at the X point. Assuming that both Γ and X are present in the grid of **k**-vectors (they would both be present in a calculation for a $2 \times 2 \times 2$ supercell, with $\mathbf{k}_s = \mathbf{0}$). Using an MDET block of the form

```
  START MDET
   Title
     MDET example: single-determinant subtraction.
   Multideterminant/excitation specification (see manual)
     SD
     DET 1 1 PR 4 G 5 X
 (or DET 1 1 PR 4 g 5 x)
  END MDET
```

would achieve the required goal. In this way, one need not know the particular **k**-point indices which label the Γ and X points.

7.4.7 Atomic orbital modification functions

The format of the data set used to specify atomic orbital modifications in `correlation.data` is as follows:

```
  START ORBMODS
   Title
     Orbital modification functions for 2s and 2p orbitals of neon
   Spin-dependence (0->u=d; 1->u/=d)
     0
   Number of modification functions
     2
   START MODFN 1
    Quantum numbers n and l
      2 0
    Expansion order
      3
    Parameters in modification function ;  Optimizable (0=NO; 1=YES)
   END MODFN 1
   START MODFN 2
    Quantum numbers n and l
      2 1
    Expansion order
      4
    Parameters in modification function ;  Optimizable (0=NO; 1=YES)
   END MODFN 2
  END ORBMODS
```

Notes:

- Atomic orbital modification functions can only be used if **atom_basis_type**='numerical' (i.e., numerical atomic orbitals are used) and the **use_orbmods** keyword is set to `T`. When **atom_basis_type**='numerical', the system consists of a single, isolated atom, with the nucleus lying at the origin.
- The modification functions are of the form

$$w_{nl}(r) = \left(c_{0nl} + c_{1nl}r + \cdots + c_{Nnl}r^N\right) \exp\left(\frac{-A_{nl}r^2}{1+B_{nl}r}\right) r^l, \tag{5}$$

  where A_{nl}, B_{nl} and $\{c_{0nl}, \ldots, c_{Nnl}\}$ are optimizable parameters. n and l are the principal and orbital-angular-momentum quantum numbers of the orbitals that are modified using w_{nl}.
- Each modification function w_{nl} is added to the corresponding HF radial function, which is found by interpolation from the data in `awfn.data`. The resulting radial function can then be multiplied by the spherical harmonic $Y_{lm}(\theta, \phi)$ to give the corresponding orbitals for each of the $2l + 1$ possible values of the magnetic quantum number m. (In fact the radial function is multiplied by appropriate linear combinations of spherical harmonics to give real-valued orbitals.)
- For s orbitals, the value of c_{1n0} is determined by the electron–nucleus Kato cusp condition: $c_{1n0} = -Zc_{0n0}$, where Z is the atomic number of the atom in an all-electron calculation and Z is zero in a pseudopotential calculation.
- If the spin-dependence flag is set to 1 then different parameters are used for spin-up and spin-down electrons; if it is set to zero then the same parameters are used for spin-up and spin-down electrons.
- If an orbital with quantum numbers n and l occurs in any determinant in the wave function, but no corresponding modification function is given in `correlation.data`, then a warning is printed and the unmodified orbital is used.
- The 'expansion order' specifies the value of N in Eq. (5). For s orbitals, the number of free c parameters is N multiplied by the number of independent spin types; for orbitals with higher angular momenta the number of c parameters is $N+1$ multiplied by the number of independent spin types. The expansion order must be at least 1.
- If parameter values are not supplied (as is the case in the example above) then default values are used: by default $A_{nl} = 1$, $B_{nl} = 0$ and $c_{inl} = 0$. If too many parameters are given then the extra ones are ignored.
- Note that we must have $A_{nl} > 0$ and $B_{nl} \geq 0$, otherwise the corresponding orbital is unnormalizable.

7.4.8 Molecular orbital modifications

The format of the data set used to specify molecular orbital modifications in `correlation.data` is as follows:

```
 START MOLORBMODS
  Title
   Molecular orbital modifications for the XXX molecule
  START GAUSSIAN MO COEFFICIENTS
   Parameter values  ;  Optimizable (0=NO; 1=YES)
  END GAUSSIAN MO COEFFICIENTS
  START GAUSSIAN EXPONENTS
   Parameter values  ;  Optimizable (0=NO; 1=YES)
  END GAUSSIAN EXPONENTS
  START CONTRACTION COEFFICIENTS
   Parameter values  ;  Optimizable (0=NO; 1=YES)
  END CONTRACTION COEFFICIENTS
 END MOLORBMODS
```

Each of the three blocks for coefficients, exponents and primitives can be present or absent independently.

Notes:

- The success of MOLORBMOD optimization is likely to vary from system to system (it is perhaps best for small molecules, since the algorithm was designed with such systems in mind). One of the main problems is that regions of the configuration space where the potential energy is large and the wave function is near zero are not sampled at all in a VMC run. The subsequent optimization run then has no incentive to keep the wave function near zero and produces large fluctuations in the corresponding coefficients, thus severely degrading the wave function.
- Only the optimization of coefficients has been tested successfully. The optimization of exponents and primitive corrections remains to be explored.

7.4.9 Free orbitals

Model systems such as homogeneous electron(–hole) gases generally require bespoke orbitals, which are often simple analytical forms that describe the qualitative state of the system, e.g., plane waves for fluid phases or simple Gaussian functions centred on lattice sites for Wigner crystals. The nature of the orbitals to be used is specified in the **free_particles** block in the `input` file. Optimizable parameters relating to these orbitals are placed in a FREE_ORBS block in `correlation.data`. A self-explanatory example for a Wigner crystal is as follows:

```
 START FREE_ORBS
 Title
  Wigner crystal
 START WIGNER CRYSTAL
  Type of WC orbitals (1: gaussian, 2: plane-waves)
    1
  Spin dependence (0 -> eu=ed ; 1 -> eu/=ed)
    0
  Parameter ;          Optimizable (0=NO; 1=YES)
   0.560882906179200              1       ! exp_1
 END WIGNER CRYSTAL
 END FREE_ORBS
```

For fluid phases the orbitals are plane waves with no optimizable parameters and hence no FREE_ORBS block is needed. For particles in a harmonic trap (with quadratic potentials defined in `expot.data`), the orbitals are harmonic oscillator eigenfunctions in the confined directions. When interactions between the particles are present the Gaussian exponents in the harmonic-oscillator may be regarded as optimizable parameters. The format of the block describing the orbitals is

```
START FREE_ORBS
Title
  Harmonic oscillator orbitals
START HARMONIC
Numbers of confined directions (grouped by equivalence)
  2 1
Spin dependence (0 -> eu=ed ; 1 -> eu/=ed)
  0
Use same Gaussian exponents for each level (0=NO; 1=YES)
  1
Gaussian exponent(s) ; Optimizable (0=NO; 1=YES)
```

```
  0.6             1        ! exp_1,1
  0.8             1        ! exp_3,1
 END HARMONIC
 END FREE_ORBS
```

Notes:

- In the above example the ground-state wave function is $\exp(-0.6(x^2+y^2)-0.8z^2)$ for each electron.
- Excited-state wave functions will be implemented when needed!

7.4.10 Pairing wave functions

There are four basic different types of pairing orbitals available in CASINO, and any linear combination of them is allowed. In the following, $\mathbf{r}$ represents the vector between the particles being paired, $\mathbf{r} = \mathbf{r}_i - \mathbf{r}_j$.

- The plane-wave pairing orbital is

$$\phi(\mathbf{r}) = \sum_{l=1}^{N_{\rm pw}} p_l \exp(i\mathbf{k}_l \cdot \mathbf{r}) \ , \tag{6}$$

  where the optimizable parameters are the linear p_l coefficients. The $\mathbf{k}_l$ vectors are the reciprocal lattice vectors of the simulation cell. Notice that the coefficients for all plane waves in the same star are constrained to be the same (they need not be supplied more than once in `correlation.data`).

- The Gaussian pairing orbital is

$$\phi(\mathbf{r}) = \sum_{l=1}^{N_g} \alpha_l \exp(-\beta_l r^2) \ , \tag{7}$$

  where α_l and β_l are the optimizable parameters. One can optionally constrain the parameters so that they correspond to the Gaussian expansion of an exponential,

$$\phi(\mathbf{r}) \approx \exp\left(-r/R_{\rm ex}\right) \ , \tag{8}$$

  in which case $R_{\rm ex}$ is the only optimizable parameter, and the α_l and β_l are varied accordingly.

- The polynomial pairing orbital is

$$\phi(\mathbf{r}) = \left(\frac{L_{\rm p} - r}{L_{\rm p}}\right)^{C_{\rm p}} \sum_{n=0}^{N_{\rm p}} a_n r^n \ , \tag{9}$$

  where $N_{\rm p}$ and $C_{\rm p}$ are the order and truncation order of the polynomials, respectively, $L_{\rm p}$ is the cutoff radius and a_n are the polynomial coefficients.

- The Slater pairing orbital is

$$\phi(\mathbf{r}) = \sum_{s=1}^{N_{\rm s}} c_s \exp\left(-\frac{a_s r^2}{1+b_s r}\right) + \sum_{p=1}^{N_{\rm P}} (\mathbf{C}_p \cdot \mathbf{r}) \exp\left(-\frac{A_p r^2}{1+B_p r}\right) \tag{10}$$

  where $N_{\rm s}$ and $N_{\rm p}$ are the orders of the S-type and P-type expansions, respectively, and c_s, a_s, b_s, $\mathbf{C}_p$, A_p and B_p are optimizable parameters.

If more than one of these forms is defined inside the 'PAIRING' section of the 'FREE_ORBS' block in `correlation.data`, they are added together to form the final orbital. In the following example all four forms are combined.

```
START PAIRING
Spin-pair dependence (CURRENTLY HAS NO EFFECT)
  0
START PLANE-WAVE TERM
Number of plane waves
  3
Parameter ;          Optimizable (0=NO; 1=YES)
  1.3                        1
  2.0                        1
  2.1                        0
END PLANE-WAVE TERM
START GAUSSIAN TERM
Number of Gaussians
  3
Fit to exponential
  F
Parameter ;          Optimizable (0=NO; 1=YES)
 0.132550237642873               1       ! g_1,1
 0.534558971701745               1       ! exp_1,1
-0.193823021668688               1       ! g_2,1
 1.057634525976201               1       ! exp_2,1
 0.532728037567208               1       ! g_3,1
 1.821074256123948               1       ! exp_3,1
END GAUSSIAN TERM
START POLYNOMIAL TERM
Order of polynomials
  3
Truncation Order of polynomial
  3
Parameter ;          Optimizable (0=NO; 1=YES)
 0.774223432432566                1       ! L_p
 0.345435431123389                1       ! alpha_0,1
 6.477832908479002                1       ! alpha_2,1
 5.666348347747839                1       ! alpha_3,1
END POLYNOMIAL TERM
START SLATER TERM
Order of S-type expansion
  1
Order of P-type expansion
  1
Parameter ;          Optimizable (0=NO; 1=YES)
 0.999999304258297                0       ! S_c_1,1
  1.12494764624888                1       ! S_a_1,1
  1.29982910271845                1       ! S_b_1,1
 0.000000000000000E+000           0       ! P_c_1,1,1
 0.000000000000000E+000           0       ! P_c_2,1,1
 1.004203030584676E-003           1       ! P_c_3,1,1
 0.872921256476300                1       ! P_a_1,1
 0.971277687984812                1       ! P_b_1,1
END SLATER TERM
END PAIRING
```

Note that it is possible to construct "partial" pairing determinants for systems with more electrons than holes, in which the excess electrons occupy plane-wave orbitals. To run calculations with partial pairing determinants, use the syntax "`+` m `orbitals free`" in the **free_particles** block in the input file; see Sec. 7.3. By default the shortest k-vectors will be populated in the usual order, which can be modified by specifying excitations in `correlation.data`.

7.4.11 Mahan wave function parameters

The parameters for the Mahan wave function are supplied to CASINO in a data set in the `correlation.data` file. The format is analogous to that of the Jastrow-factor and backflow data sets. Please see the notes in Sec. 7.4.2 for information on the 'truncation order', 'spin dependence', etc. Detailed information about the Mahan wave function can be found in Sec. 33.

```
START MAHAN
Title
26e HEG + hole --- example correlation.data file
START ETA_G TERM
Expansion order
  6
Spin-dep (0->u=d ; 1->u/=d)
  0
Number of optimizable stars
  2
Truncation orders
  3.00000000000000            ! C_2
  3.00000000000000            ! C_2
Cutoff radii; Optimizable (0=NO; 1=Yes)
  9.63252167718146                1       ! L_2
  9.63252167718146                1       ! L_2
Parameters; Optimizable (0=NO; 1=Yes)
END ETA_G TERM
START U_G TERM
Expansion order
  6
Spin-dep (0->u=d ; 1->u/=d)
  0
Number of optimizable stars
  2
Truncation orders
  3.00000000000000            ! C_2
  3.00000000000000            ! C_2
Cutoff radii; Optimizable (0=NO; 1=Yes)
  9.63252167718146                1       ! L_2
  9.63252167718146                1       ! L_2
Parameters; Optimizable (0=NO; 1=Yes)
END U_G TERM
END MAHAN
```

Notes:

- Both the η_G and u_G functions contain optimizable parameters for each star of electrons except the first. The number of optimizable stars for each function is therefore one less than the total number of stars of electrons.
- The η_G and u_G functions have separate cutoff lengths and truncation orders for each of the optimizable stars. These must be included, one per line, for each optimizable star in the block as above.
- It is possible to run without the u_G block; simply delete the lines from 'START U_G TERM' to 'END U_G TERM', inclusive, above.
- The spin-dependence must be `0` for both functions, as only this option has been implemented so far.
- The lists of parameter values above are blank; CASINO will supply suitable default values for the Mahan parameters not listed, usually assigning the value zero, except where the cusp conditions require non-zero values.

7.5 Pseudopotential file: `xx_pp.data`

CASINO can carry out all-electron or pseudopotential calculations, but it is normally advantageous to replace the core electrons by a pseudopotential. CASINO will automatically treat an atom as all-electron unless there exists a pseudopotential file `xx_pp.data` in the directory in which CASINO is run, where `xx` is the symbol for the element in question in lower case. This file contains the different angular momentum components of the pseudopotential given on a radial grid and some auxiliary information in the following format:

```
LSDA Pseudopotential in real space for Si
Atomic number and pseudo-charge
14 4d0
Energy units (rydberg/hartree/ev):
rydberg
Angular momentum of local component (0=s,1=p,2=d..) for DFT and QMC
0 2
NLRULE override (1) VMC/DMC (2) config gen (0 ==> input/default value)
0 0
 Number of grid points
        2476
 R(i) in atomic units
  0.000000000000000E+000
  7.178690774405650E-012
.
   39.6567798705125
   40.0553371342383
 r*potential (L=0) in Ry
     0.000000000000000E+00
     0.131424063731862E-10
.
     -8.00000000000000
     -8.00000000000000
 r*potential (L=1) in Ry
     0.000000000000000E+00
    -0.574393968349227E-10
.
     -8.00000000000000
     -8.00000000000000
 r*potential (L=2) in Ry
     0.000000000000000E+00
    -0.128711823920867E-09
.
     -8.00000000000000
     -8.00000000000000
Core-polarization terms
0.1650 0.5446        ! alpha (a.u.) rbaree (a.u.) usually set to 0.5*(rbars+rbarp)
0.5216 0.5676 0.7172 ! rbar for s,p,d  (a.u.)
```

The **nlrule** parameters control the grid on which the angular integration is performed (see Sec. 19.2) and are normally specified in the file `input`. The values given in `input` are the default for all atoms in the system but they can be overridden for particular elements by setting the parameters in this file to nonzero values. As many angular-momentum components as desired can be supplied, but they must be in the order $l = 0, 1, 2, \ldots$. Unless you explicitly wish to use a core-polarization potential (see Sec. 19.3), the few lines at the bottom relating to this should be omitted. At present the $\bar{r}_l$ parameters for the core-polarization potential should be supplied for $l = 0$, 1 and 2 only.

CASINO pseudopotential library: `https://vallico.net/casinoqmc/pplib/`

On the rare occasions when you might want to use two or more different pseudopotentials for atoms with the same atomic number (say in a surface, and in an atom or molecule absorbed on that surface), then you may use additional pseudopotentials renamed as, e.g., `xx2_pp.data`. Different types of pseudoatom are flagged in `xwfn.data` by adding multiples of 1000 to the original atomic number, e.g., atomic numbers 12, 1012 and 2012 correspond to atoms using pseudopotentials `mg_pp.data`, `mg2_pp.data` and `mg3_pp.data`, etc.

7.6 MPC-interaction file: `mpc.data`

This contains the Fourier transform of the function $f(\mathbf{r})$ defined in Sec. 19.4.4. It is required when using the model periodic Coulomb (MPC) interaction to compute the electron–electron interactions. Before doing a QMC calculation with the MPC interaction, this file should be generated from the trial wave function given in the `[x]wfn.data` file by using `runqmc` to run CASINO with **runtype** set to 'gen_mpc' in the input file. The data can be visualized in real space using the supplied `plot_mpc` utility.

The format of the `mpc.data` file is as follows:

```
 START MPC DATA
  Title
    Title of system goes here
  File version
    1  [this should be an integer that signifies spec version]

  START DENSITY DATA
   Self consistent charge density in reciprocal space
   Real space primitive cell translation vectors (au)
      PA1X   PA1Y   PA1Z
      PA2X   PA2Y   PA2Z
      PA3X   PA3Y   PA3Z
   Number of atoms in the primitive cell
      nbasis
   Positions of atoms (au)
      atno(1)   basis(1,1)   basis(2,1)   basis(3,1)
      atno(2)   basis(1,2)   basis(2,2)   basis(3,2)
      ...
      atno(n)   basis(1,n)   basis(2,n)   basis(3,n)
   Energy cutoff used for G-vectors (au)
      energy_cutoff
   Number of particle types (1=electrons,2=electrons/holes)
    2
   Number of G-vectors
      den_ngvec
   G-vectors (au)
      gvec(1,1)   gvec(2,1)   gvec(3,1)
      gvec(1,2)   gvec(2,2)   gvec(3,2)
      ...
      gvec(1,n)   gvec(2,n)   gvec(3,n)
   START SET 1
    Particle type
     1
    Complex charge density (real part, imaginary part)
       Re{den_sc(1)}   Im{den_sc(1)}
       ...
       Re{den_sc(n)}   Im{den_sc(n)}
   END SET 1
   START SET 2
    Particle type
     2
    Complex charge density (real part, imaginary part)
       Re{den_sc(1)}   Im{den_sc(1)}
       ...
       Re{den_sc(n)}   Im{den_sc(n)}
   END SET 2
  END DENSITY DATA

  START EEPOT DATA
   Real space simulation cell translation vectors (au)
      A1X      A1Y      A1Z
      A2X      A2Y      A2Z
      A3X      A3Y      A3Z
   Multiples of primitive translation vectors
      2 2 2
   Energy cutoff used for G-vectors
      energy_cutoff
   Number of G-vectors
      eepot_ngvec
   G-vectors (au) and Fourier coefficients
      gvec(1,1)   gvec(1,2)   gvec(1,3)   fg(1)
      gvec(2,1)   gvec(2,2)   gvec(2,3)   fg(2)
      ...
      gvec(n,1)   gvec(n,2)   gvec(n,3)   fg(n)
  END EEPOT DATA

 END MPC DATA
```

Notes:

1. The 'file version' is an integer, which is always increased if the specification for this file changes.
2. The charge-density Fourier coefficients may be complex.
3. The EEPOT Fourier coefficients must be real (by definition).
4. This format allows for the existence of different types of particles.
5. Both the primitive-cell and simulation-cell translation vectors are included so that consistency checks can be made. Furthermore, this allows the data to be scaled appropriately if they are reused for a system of different size.
6. The normalization of the density should be such that the $\mathbf{G} = \mathbf{0}$ component is equal to the number of particles per primitive cell. This makes the density independent of simulation-cell size.

Note: it is not necessary to regenerate this file for different supercell sizes, provided the lattice vectors in the 'EEPOT' set are obtainable from the lattice vectors in `[x]wfn.data` through scaling by a single parameter.

7.7 The CASL file format

CASINO is starting to make use of the CASL serialization language and file format for data interchange. The CASL serialization language defines a syntax to specify structured data by means of keyword-value associations. The file format is inspired by YAML, and is reminiscent of Python code in its use of indentation.

A CASL file, usually named *filename*`.casl`, could look like this:

```
keyword1: value1
keyword2: value2
keyword3:
  keyword4: value4
  keyword5: [ keyword6: value6, value7 ]
value8
```

where:

- `keyword1` is the label of a scalar item of value `value1`.
- `keyword3` is the label of a block item that contains the items labelled `keyword4` and `keyword5`.
- `keyword5` is the label of another block item, represented in-line using square brackets and commas, that contains the item labelled `keyword6` and an unnamed item of value `value7`.
- `value7` and `value8` are the values of two unnamed items. Unnamed items are intended to be accessed by sequential order. Unnamed items cannot be block items.

CASL keywords are case- and whitespace- insensitive. The `:` and `,` characters must have a whitespace or newline immediately after them to be considered syntactically active.

The specific keyword-value structures available in a given CASL block depends on the block, as described below, but the underlying syntax is the same for all of them.

Some additional notes on the CASL format:

- All named items are identified by their name, not the order in which they appear.

- All unnamed items are identified by the order in which they appear.
- The parser treats scalar values as strings, therefore it does not issue data type mismatch errors. After parsing, specific scalars may be allowed to have one of several data types, or may be restricted to a single data type.
- Blank lines are ignored.
- Anything following a hash sign "`#`" in a line is ignored, e.g., `Keyword: value # comment`.
- Whole blocks can be commented out by prepending "`%!`" to their name, for example `%!This block: [ is, ignored ]`. This is called a "syntactical comment"; the block is still parsed and subject to the CASL syntax rules, but then it and its contents are discarded after reading them.
- Lines can be divided at any blank that does not follow a colon "`:`", and continuation lines must all have the same indentation which must be greater than that of the line they are continuing. For example,

```
Keyword: really long value
     which does not fit in
     a single line.
Other keyword: value
```

- Named scalars with an empty value are allowed, e.g., `Empty scalar:`.
- Empty blocks are allowed and can be declared only in inline form, e.g., `Empty block: [ ]`.
- The parser interprets "`(`", "`)`", "`{`", "`}`" and "`"`" as grouping characters, and will not match any syntactical tokens inside them. For example, in `Complex variables: [ a: (0.0, 0.0), b: (5.0, -2.0) ]`, the value of `a` is "`(0.0, 0.0)`", not "`(0.0`", etc.

7.8 Wave function parameter file: `parameters.casl`

The `parameters.casl` file is intended as a replacement for `correlation.data`; when new wave function parameters are introduced, they are added to `parameters.casl`.

Each wave function section goes in a different top-level block in `parameters.casl`. As is the case with the `correlation.data` file, optimized wave function parameters are written to disk during optimization to files called `parameters.`x`.casl`, where x is the optimization cycle number.

7.8.1 Generic Jastrow construction framework

Generic Jastrow factors constructed using the framework described and showcased in Ref. [29] are declared in the `JASTROW` top-level block of the `parameters.casl` file. We refer to this Jastrow factor implementation as *gjastrow*. The *gjastrow* allows constructing arbitrary terms involving n electrons and m nuclei using any of the available basis functions.

In a system of N electrons and M nuclei, an n-electron, m-nuclei *gjastrow* Jastrow factor term is of the form,

$$J_{n,m}(\mathbf{R}) = \sum_{i_1<\cdots<i_n}^{N} \sum_{I_1<\cdots<I_m}^{M} \sum_{\{\nu_{i_\alpha i_\beta}\}_{\alpha<\beta}^{n}}^{p} \sum_{\{\mu_{i_\alpha I_\gamma}\}_{\alpha,\gamma}^{n,m}}^{q} \lambda_{\{\nu\}\{\mu\}}^{\{P\}\{S\}} \prod_{\alpha<\beta}^{n} \Phi_{\nu_{i_\alpha i_\beta}}^{P_{i_\alpha i_\beta}}(\mathbf{r}_{i_\alpha i_\beta}) \prod_{\alpha,\gamma}^{n,m} \Theta_{\mu_{i_\alpha I_\gamma}}^{S_{i_\alpha I_\gamma}}(\mathbf{r}_{i_\alpha I_\gamma}) \; , \quad (11)$$

where

- i_α and I_γ are the electron and nucleus indices,
- ν_{ij} and μ_{iI} are the electron-electron and electron-nucleus expansion indices (there are $n(n-1)/2$ and nm such indices, respectively),
- P_{ij} and S_{iI} are integers representing electron-electron and electron-nucleus pair types, which define the spin-pair and spin-nucleus dependencies of the parameters,
- $\lambda_{\{\nu\}\{\mu\}}^{\{P\}\{S\}}$ are the linear parameters, where the superindices $\{P\}\{S\}$ select the appropriate parameter *channel* for the electrons and nuclei under consideration,

- $\Phi_\nu^P(\mathbf{r})$ and $\Theta_\mu^S(\mathbf{r})$ are the ν-th electron-electron and μ-th electron-nucleus basis functions; if they contain optimizable parameters, P and S select the appropriate parameter set for the corresponding electron-electron or electron-nucleus pair, and
- p and q are the electron-electron and electron-nucleus expansion orders, respectively.

For convenience we factorize Φ and Θ into a basis function times an optional cut-off function which does not depend on the expansion index, $\Phi_\nu^P(\mathbf{r}) = \phi_\nu^P(\mathbf{r}) f^P(\mathbf{r})$ and $\Theta_\mu^S(\mathbf{r}) = \theta_\nu^S(\mathbf{r}) g^S(\mathbf{r})$.

The `JASTROW` CASL block may contain a string-valued `Title` scalar item, and then one block item named `TERM` x per term in the Jastrow factor.

The following is a sample Jastrow factor definition for a system consisting of a graphene layer of 50 carbon atoms, and a hydrogen atom, with 201 electrons:

```
JASTROW:
  Title: H on graphite
  TERM 1:
    Rank: [ 2, 0 ]
    Rules: [ 1-1=2-2 ]
    e-e basis: [ Type: natural power, Order: 9 ]
    e-e cutoff:
      Type: alt polynomial
      Constants: [ C: 3 ]
  TERM 2:
    Rank: [ 1, 1 ]
    Rules: [ Z, 1=2 ]
    e-n basis: [ Type: natural power, Order: 9 ]
    e-n cutoff:
      Type: alt polynomial
      Constants: [ C: 3 ]
  TERM 3:
    Rank: [ 2, 1 ]
    Rules: [ 1-Z6=2-Z6, 1-Z1=2-Z1, 1-1=1-2=2-2 ]
    e-e basis: [ Type: natural power, Order: 3 ]
    e-n basis: [ Type: natural power, Order: 3 ]
    e-n cutoff:
      Type: alt polynomial
      Constants: [ C: 3 ]
```

This example serves us to introduce how the *gjastrow* is defined:

- `Rank` is a block item containing two integer-valued unnamed scalars which define the number of electrons n and of nuclei m involved in the term. In the above example, term 1 is an e–e term (u term), term 2 is an e–n term (χ term), and term 3 is an e–e–n term (f term).
- `Rules` is a block item containing any number of *grouping rules*. These define the P and S matrices by specifying which particles, particle pairs, nuclei, or particle-nucleus pairs are to be regarded as equivalent, thus determining the *parameter channels* required for each group of particles and nuclei present in the system. For example, in term 1 above we force up-spin–up-spin electron pairs (`1-1`) to be equivalent to down-spin–down-spin electron pairs (`2-2`). In term 2 we make all nuclei with the same Z equivalent (`Z`), thus grouping all carbon atoms, and up-spin and down-spin electrons are regarded as equivalent (`1=2`). In term 3 we request that the pair formed by up-spin electrons and carbon atoms (`1-Z6`) be treated as equivalent to that formed by down-spin electrons and carbon atoms (`2-Z6`), that up-electron–hydrogen be equivalent to down-electron–hydrogen (`1-Z1=2-Z1`), and that up-up, up-down and down-down electron pairs be equivalent (`1-1=1-2=2-2`). Different rule specifications may yield the same results; for example, in term 3 above we could have specified `Rules: [ Z, 1=2 ]` to the same effect. See below for the channel splitting achieved by the rules in this example, and further down for a fuller description of rules.
- `e-e basis`, `e-e cutoff`, `e-n basis` and `e-n cutoff` are blocks containing the necessary details to define which basis function the Jastrow factor term is expanded into and which cut-off functions, if any, are applied. `Type` is a string-valued scalar item which chooses the basis functions, a list of which can be found later in this section; "`none`" is a valid value. `Order` is an integer-valued

scalar item which selects the expansion order to be used for the e–e and e–n basis functions (not available for cut-offs). Note that basis functions are indexed starting from 1 in the *gjastrow*, not 0 as is done in the Jastrow factor defined in the `correlation.data` file. Some basis functions contain constants, which must be declared in a `Constants` block, and some contain optimizable parameters, which are declared in a `Parameters` block. For example, term 3 above uses natural powers for the basis functions with an expansion order of 3 (i.e., r^0, r^1 and r^2 are the basis functions), no electron-electron cut-off function, and the `Alt polynomial` cut-off function for the electron-nucleus cut-offs, with a truncation order constant `C` of 3.

Note that there are no parameters defined in the above example. After optimization, the `parameters.casl` file is populated with the optimized parameters, resulting in, for example:

```
JASTROW:
  Title: H on graphite
  TERM 1:
    Rank: [ 2, 0 ]
    Rules: [ 1-1=2-2 ]
    e-e basis: [ Type: natural power, Order: 9 ]
    e-e cutoff:
      Type: alt polynomial
      Constants: [ C: 3 ]
      Parameters:
        Channel 1-1:
          L: [ 11.352023756975679, optimizable, limits: [ 0.50000000000000000,
               11.645421058099613 ] ]
        Channel 1-2:
          L: [ 11.352023756975679, optimizable, limits: [ 0.50000000000000000,
               11.645421058099613 ] ]
    Linear parameters:
      Channel 1-1:
        c_2: [ -8.1899486572157494E-005, optimizable ]
        c_3: [ 5.1184567229701015E-006, optimizable ]
        c_4: [ 1.8827600991665889E-006, optimizable ]
        c_5: [ -8.7213298031161725E-007, optimizable ]
        c_6: [ 1.1125342957423828E-007, optimizable ]
        c_7: [ 7.3652204644128116E-009, optimizable ]
        c_8: [ -2.2870664337997183E-009, optimizable ]
        c_9: [ 1.1138335392155449E-010, optimizable ]
      Channel 1-2:
        c_2: [ -2.1396334655704384E-004, optimizable ]
        c_3: [ 6.0234443089333439E-005, optimizable ]
        c_4: [ -1.4480264142099425E-005, optimizable ]
        c_5: [ 4.1747844178600060E-006, optimizable ]
        c_6: [ -1.1839016098212222E-006, optimizable ]
        c_7: [ 2.0853997270042511E-007, optimizable ]
        c_8: [ -1.8336515915250190E-008, optimizable ]
        c_9: [ 6.1723787079640366E-010, optimizable ]
  TERM 2:
    Rank: [ 1, 1 ]
    Rules: [ Z, 1=2 ]
    e-n basis: [ Type: natural power, Order: 9 ]
    e-n cutoff:
      Type: alt polynomial
      Constants: [ C: 3 ]
      Parameters:
        Channel 1-n1:
          L: [ 6.0700477330152580, optimizable, limits: [ 0.50000000000000000,
               11.645421058099613 ] ]
        Channel 1-n51:
          L: [ 3.7337378814861637, optimizable, limits: [ 0.50000000000000000,
               11.645421058099613 ] ]
    Linear parameters:
      Channel 1-n1:
        c_2: [ -2.7080127539095265E-003, optimizable ]
        c_3: [ 9.1801079749774387E-004, optimizable ]
        c_4: [ -2.1689705811039064E-003, optimizable ]
        c_5: [ 2.0357040900787675E-003, optimizable ]
        c_6: [ -9.1125883025703280E-004, optimizable ]
        c_7: [ 2.0898033728911013E-004, optimizable ]
        c_8: [ -2.3699765748611267E-005, optimizable ]
        c_9: [ 1.0225781416020435E-006, optimizable ]
      Channel 1-n51:
        c_2: [ -1.6245605563919808E-003, optimizable ]
        c_3: [ -3.9939657824912926E-003, optimizable ]
        c_4: [ 1.9993939109752307E-002, optimizable ]
        c_5: [ -4.1150138201955976E-002, optimizable ]
        c_6: [ 3.8713128982571968E-002, optimizable ]
        c_7: [ -1.9167699173502323E-002, optimizable ]
        c_8: [ 4.8731383835212486E-003, optimizable ]
        c_9: [ -5.0240854042510926E-004, optimizable ]
  TERM 3:
    Rank: [ 2, 1 ]
    Rules: [ 1-Z6=2-Z6, 1-Z1=2-Z1, 1-1=1-2=2-2 ]
    e-e basis: [ Type: natural power, Order: 3 ]
    e-n basis: [ Type: natural power, Order: 3 ]
    e-n cutoff:
      Type: alt polynomial
      Constants: [ C: 3 ]
      Parameters:
        Channel 1-n1:
          L: [ 5.8227027247545742, optimizable, limits: [ 0.50000000000000000,
               11.645421058099613 ] ]
        Channel 1-n51:
          L: [ 2.8697151605618441, optimizable, limits: [ 0.50000000000000000,
               11.645421058099613 ] ]
    Linear parameters:
      Channel 1-1-n1:
        c_1,2,2: [ -5.9197710909612964E-007, optimizable ]
        c_1,3,2: [ -3.5400241136773510E-007, optimizable ]
        c_1,3,3: [ -1.0268299593198822E-007, optimizable ]
        c_3,1,1: [ -1.0458640505715438E-006, optimizable ]
        c_3,2,1: [ 2.5749897254695416E-007, optimizable ]
        c_3,2,2: [ -2.0195608511354725E-008, optimizable ]
        c_3,3,2: [ -1.5698588458262213E-008, optimizable ]
        c_3,3,3: [ 1.1525211735158285E-008, optimizable ]
      Channel 1-1-n51:
        c_1,2,2: [ -2.0702547828979545E-003, optimizable ]
        c_1,3,2: [ -7.0607939146236269E-003, optimizable ]
        c_1,3,3: [ 7.2263427882358795E-003, optimizable ]
        c_3,1,1: [ 8.3725219208974691E-005, optimizable ]
        c_3,2,1: [ 1.2560947012457320E-003, optimizable ]
        c_3,2,2: [ -3.6945139699129980E-003, optimizable ]
        c_3,3,2: [ 4.8378566108180792E-004, optimizable ]
        c_3,3,3: [ -1.3437750629367575E-003, optimizable ]
```

In this example note the following:

- The parameters in the basis functions and cut-off functions appear in their respective blocks under the `Parameters` sub-block, grouped by particle-pair/particle–nucleus channel. The linear parameters of the Jastrow factor appear in the `Linear parameters` block grouped by their full n-particle–m-nuclei channels.
- The parameters in the basis functions and cut-off functions have names that depend on the selected `Type`. The linear parameters are always named `c_`*indices*.
- All parameter declaration blocks have the parameter value as their first unnamed item and the string `optimizable` or `fixed` as their second unnamed item. The parameter value can be set to `default`, or, if the parameter has both an upper limit and a lower limit, a string such as `50%` will initialize the parameter value relative to the limits. It is possible to omit the second unnamed item, or both of them.

- Channels are named after the group of particles and nuclei with the lowest indices that belongs in the channel, e.g., the electron-electron-nucleus linear-parameter channel in term 3 above that involves two any-spin electrons and a hydrogen nucleus is called `1-1-n51`.
- The values of all optimizable parameters can be limited using the `Limits` sub-block, as in the case of the `L` cut-off lengths above. Limits are not shown for parameters which have no default limits, but the `Limits` block can be added manually for them if necessary, e.g., `c_1,3,3: [ Limits: [ -0.01, 0.01 ] ]`. The special values `Inf`, `-Inf` and `default` are allowed in place of the limits, or, if the parameter has both a default upper limit and a default lower limit, a string such as `37.5%` will redefine the corresponding limit relative to the default limits. Default parameter limits can only be redefined to reduce the parameter range, not to extend it.

The available basis functions are:

- **Natural power basis**
  - `Type: natural power`
  - Description: a natural power basis for localized Jastrow terms.
  - Functional form:
$$\phi_k(\mathbf{r}) = r^{k-1} \tag{12}$$
  - Constants: none
  - Parameters: none
  - Notes:
    * This is the natural-power basis used in the standard CASINO Jastrow factor for the U, χ, f, H, W and D terms.
- **Cosine basis**
  - `Type: cosine`
  - Description: a cosine basis for periodic systems.
  - Functional form:
$$\phi_k(\mathbf{r}) = \sum_{\substack{\mathbf{G}\in k\text{th} \\ \text{star}}} \cos(\mathbf{G}\cdot\mathbf{r}) \tag{13}$$
  - Constants: the $\mathbf{G}$ vectors in terms of the reciprocal lattice vectors, expressed as integers grouped in *periodicity*-sized vectors, in turn organized in stars. For example,

```
Constants:
  Star 1:
    G_1: [ 0, 0, 0 ]
  Star 2:
    G_1: [ 0, 0, 1 ]
    G_2: [ 1, 0, 0 ]
    G_3: [ 0, 1, 0 ]
  Star 3:
    G_1: [ 1, 0, 1 ]
    G_2: [ 1, 0, -1 ]
    G_3: [ 0, 1, 1 ]
    G_4: [ 0, 1, -1 ]
    G_5: [ 1, 1, 0 ]
    G_6: [ 1, -1, 0 ]
```

    There are *periodicity*×*number-of-G-vectors* integer-valued constants. If left unspecified, CASINO will generate the shortest $\mathbf{G}$ vectors up to the requested number of stars (which is what one almost always wants).
  - Parameters: none
  - Notes:
    * This is the cosine basis used in the standard CASINO Jastrow factor for the P term.
    * The expansion order is the number of stars, not the number of $\mathbf{G}$ vectors as in the standard Jastrow.

    - ∗ $\mathbf{G} = \mathbf{0}$ is included by default in the set of $\mathbf{G}$ vectors. If used in a two-body term without a cut-off, the $\mathbf{G} = \mathbf{0}$ basis function will be flagged as redundant (since it is a constant) and its linear coefficient will be fixed at a value of zero.

- **Cosine basis with k-cutoff**
  - Type: `cosine with k-cutoff`
  - Description: a cosine expansion where the coefficients of the cosines are computed as a polynomial function of the modulus of the $\mathbf{G}$ vectors, intended to improve transferability between different system sizes.
  - Functional form:

$$\phi_k(\mathbf{r}) = \frac{1}{\Omega} \sum_{s}^{n_s} |\mathbf{G}_s|^{p_0+\delta_p(k-1)} \left(|\mathbf{G}_s| - k_c\right)^C \sum_{\substack{\mathbf{G}\in s\text{th}\\ \text{star}}} \cos\left(\mathbf{G}\cdot\mathbf{r}\right) \tag{14}$$

    where Ω is the volume/area/length of the simulation cell in 3D/2D/1D.
  - Constants: p_0, δ_p (reals), k_c (real, in a.u.) and C (integer), called "`p_0`", "`delta_p`", "`k_cut`" and "`C`". The k-cutoff k_c has a default value of 1 a.u. (not necessarily sensible, k_c should be set by hand); the truncation order C has a default value of 1; the exponent offset p_0 and exponent step δ_p have respective default values of -2 and 1 in 3D-periodic systems, $-3/2$ and $1/2$ in 2D-periodic systems, and 0 and 1 in other cases.
  - Parameters: none
  - Notes:
    - ∗ This basis is designed to construct a p-like e–e term where the coefficients of the cosines are not the optimizable parameters, but are themselves a parametrized function α of $|\mathbf{G}|$. The ij contribution to such a term looks like

$$p(\mathbf{r}) = \sum_{s}^{n_s} \alpha(|\mathbf{G}_s|) \sum_{\substack{\mathbf{G}\in s\text{th}\\ \text{star}}} \cos\left(\mathbf{G}\cdot\mathbf{r}\right) \ . \tag{15}$$

      If we choose to parametrize α as a polynomial of $|\mathbf{G}|$ times a polynomial cut-off,

$$\alpha(|\mathbf{G}|) = \left(|\mathbf{G}| - k_c\right)^C \sum_k c_k \left|\mathbf{G}\right|^k \ , \tag{16}$$

      substituting into the previous equation and swapping the summation signs gives the expression for ϕ_k above, with $\{c_k\}$ turned into linear parameters of the Jastrow factor.
    - ∗ Note that, unlike for the `cosine` basis, the expansion order is not the number of stars n_s in the cosine expansion. n_s is the number of stars up to $|G| = k_c$ and is calculated automatically, while the expansion order is the number of terms in the polynomial expansion of $\alpha(|\mathbf{G}|)$.
    - ∗ The $\mathbf{G}$ vectors are calculated automatically for this functional basis and cannot be provided by the user. $\mathbf{G} = \mathbf{0}$ is never included in this set of $\mathbf{G}$ vectors.

- **ν basis**
  - Type: `nu`
  - Description: basis functions that interpolate between the isotropic, radially symmetric natural powers at short radius and functions with the simulation cell symmetry at large distances [30].
  - Functional form:

$$\phi_k(\mathbf{r}) = \left| \sum_i \mathbf{A}_i\cdot\mathbf{A}_i f^2(w_i) + 2\sum_{j>k} \mathbf{A}_j\cdot\mathbf{A}_k g(w_j) g(w_k) \right|^{k/2} \ , \tag{17}$$

    where $w_i = \mathbf{B}_i\cdot\mathbf{r}$,

$$f(w) = |w| \left(1 - \frac{|w/\pi|^3}{4}\right) \ , \tag{18}$$

$$g(w) = w\left(1 - \frac{3}{2}|w/\pi| + \frac{1}{2}|w/\pi|^2\right) \ , \tag{19}$$

$\{\mathbf{B}\}$ are the reciprocal lattice vectors of the simulation cell and (optionally) all symmetry-equivalent vectors, and $\{\mathbf{A}\}$ are their real-space counterparts.

- Constants: the $\{\mathbf{A}\}$ and $\{\mathbf{B}\}$ vectors in atomic units grouped in 3-dimensional vectors, for example,

```
Constants:
  a_1: [ 0.61833763416878673, 0.0000000000000000, 0.0000000000000000 ]
  a_2: [ 0.0000000000000000, 0.61833763416878673, 0.0000000000000000 ]
  a_3: [ 0.0000000000000000, 0.0000000000000000, 0.61833763416878673 ]
  b_1: [ 1.6172394250986690, 0.0000000000000000, 0.0000000000000000 ]
  b_2: [ 0.0000000000000000, 1.6172394250986695, 0.0000000000000000 ]
  b_3: [ 0.0000000000000000, 0.0000000000000000, 1.6172394250986695 ]
```

  There are 6×*number-of-B-vectors* real-valued constants. If left unspecified, CASINO will use the optimal set of vectors for simple cubic, FCC, BCC, 3D hexagonal, square and 2D hexagonal lattices, and the "basic", *dimensionality*-sized set of vectors for other lattices.
- Parameters: none
- Notes:
  * This basis function can only be used in periodic systems.
  * This basis function is intended to capture the same correlations as the natural power basis with polynomial cutoff combined with the cosine basis.
  * This basis function is locally isotropic at $r = 0$, and can therefore be used to apply Kato cusp conditions.
  * This basis function does not require a cutoff function, automatically satisfying periodic boundary conditions in the simulation cell.

- **Fractional power basis**
  - `Type: r/(r^b+a) power` (also `r/(r+a) power` and `1/(r+a) power`, see notes below)
  - Description: powers of fractions which tend to a constant as $r \to \infty$.
  - Functional form:
$$\phi_k(\mathbf{r}) = \left(\frac{r}{r^b + a}\right)^{k-1} \tag{20}$$
  - Constants: none
  - Parameters: a and b (reals), called "`a`" and "`b`", whose default values are 3 and 1.3, respectively, and have default limits of $a \in [1.1 \times 10^{-8}, +\infty)$, and $b \in [1, +\infty)$
  - Notes:
    * This is a good functional basis for finite systems, especially for atoms where it performs significantly better than natural powers with polynomial cut-offs.
    * Also available are a basis of powers of $r/(r + a)$ ("`Type: r/(r+a) power`"), and one with powers of $1/(r + a)$ ("`Type: 1/(r+a) power`"). According to our tests, these alternatives seem to be slightly inferior to the powers of $r/(r^b + a)$.
    * It is often useful to impose stricter limits on the non-linear parameters for this basis to avoid issues during optimization, for example with

```
a: [ limits: [ 1, 8 ] ]
b: [ limits: [ 1.1, 2 ] ]
```

- **Exponential power basis**
  - `Type: exp power`
  - Description: powers of $[1 - \exp(-ar)]/a$ which tend to a constant as $r \to \infty$.
  - Functional form:
$$\phi_k(\mathbf{r}) = \left(\frac{1 - \exp(-ar)}{a}\right)^{k-1} \tag{21}$$
  - Constants: none

  - Parameters: a (real), called "a", whose default value is 1 and has default limits of $a \in [1.1 \times 10^{-8}, +\infty)$
  - Notes:
    * This is similar to the fractional power basis in nature.

- **Natural polynomial basis**
  - `Type: natural polynomial`
  - Description: a polynomial, optionally broken up into pieces if `Order` is greater than 1.
  - Functional form:

$$\phi_k(\mathbf{r}) = r^{k_0+\sum_{p=1}^{k-1} k_p} \sum_{l=k_0+\sum_{p=1}^{k-1} k_p+1}^{k_0+\sum_{p=1}^{k} k_p} c_l r^l \tag{22}$$

  - Constants: the exponent offset k_0 (integer) and the `Order` lengths of the sub-expansions in which the polynomial is split, k_p, which are specified as, e.g.,

```
Order: 2
Constants:
  k_0: 0
  Split: [ 3, 3 ]
```

    which results in the basis functions $\phi_1 = 1 + c_1 r + c_2 r^2$ and $\phi_2 = r^3 + c_4 r^4 + c_5 r^5$.
  - Parameters: the polynomial coefficients c_l, called `c_`*l*. Note that the first coefficient in each basis function is not variable to avoid redundancies with the linear coefficients of the Jastrow term, and is fixed at a value of one.
  - Notes:
    * The most straight-forward use of this basis is with an `Order` of one, where `Split` contains a single integer which is the order of the polynomial minus one.
    * The natural power basis is recovered by setting $k_0 = 0$ and $k_p = 1 \;\; \forall\, p$, in which case a `Parameters` block need not be given.
    * This functional basis is mainly intended to be used with a low `Order` in the construction of terms with high `Rank`, i.e., those involving many electrons and/or nuclei. Compared with the natural power basis, the use of a polynomial reduces the cost of evaluating the Jastrow factor substantially by reducing the number of possible products of basis functions, at the cost of giving up the variational freedom of having independent coefficients for each product of natural powers. Splitting the polynomial into pieces allows one to recover some of this variational freedom.

- **Natural power basis for dot-product terms**
  - `Type: natural power vectorial` (also `natural polynomial vectorial`, see notes below)
  - Description: a natural power basis for dot-product terms.
  - Functional form:

$$\phi_k(\mathbf{r}) = r^{\mathrm{INT}\left(\frac{k-1}{d}\right)} \frac{r_{\mathrm{MOD}(k,d)}}{r} \tag{23}$$

    where d is the dimensionality of the system and r_i is the ith Cartesian component of $\mathbf{r}$.
  - Constants: none
  - Parameters: none
  - Notes:
    * This basis is intended to be used in conjunction with the `Dot-product` indexing constraint (see below).
    * There exists an analogous `natural polynomial vectorial` basis.
    * For example, the W term in the standard CASINO Jastrow factor translates into:

```
TERM 2:
  Rank: [ 3, 0 ]
  Rules: [ 1=2 ]
  Indexing:
    e-e dot product: T
  e-e basis:
    Type: natural polynomial vectorial
    Order: 3
    Constants:
      k0: 1
      Split: [ 7 ]
  e-e cutoff:
    Type: alt polynomial
    Constants: [ C: 2 ]
```

- **RPA electron-electron Jastrow factor form**
  - Type: `RPA`
  - Description: the RPA form of the electron-electron Jastrow factor.
  - Functional form:
    $$\phi_1(\mathbf{r}) = -\left[1 - \exp(-r/F)\right]/r \tag{24}$$
    in three dimensions, and
    $$\phi_1(\mathbf{r}) = -\left[1 - \exp\left(-r/2F - \sqrt{r/F}\right)\right]/\sqrt{r} \tag{25}$$
    in two dimensions.
  - Constants: none
  - Parameters: F (real), called "`F`", whose default value is 1 and has default limits of $F \in [10^{-7}, +\infty)$.
  - Notes:
    * The maximum expansion order for this basis is 1.
    * In periodic systems it is recommended to combine this basis function with a cuspless cut-off function (e.g., `gaussian` or `spline`) in order to preserve the short-range behaviour of the RPA Jastrow factor.

- **Specialized cusp condition functions**
  - Type: `logarithmic cusp` / `dipole cusp` / `tilted dipole cusp`
  - Description: function to enforce cusp conditions with non-Coulomb interactions
  - Functional forms (respectively for `logarithmic cusp` / `dipole cusp` / `tilted dipole cusp`):
    $$\begin{aligned} \phi_1(\mathbf{r}) &= r^2 \log r \\ \phi_1(\mathbf{r}) &= 1/\sqrt{r} \\ \phi_1(\mathbf{r}) &= \left[a_0(\theta) + a_2(\theta)(x^2/r^2 - 1)\right]/\sqrt{r} \end{aligned} \tag{26}$$
    where $a_0(\theta)$ and $a_2(\theta)$ are fixed functions of the tilt angle θ.
  - Constants: none
  - Parameters: for `tilted dipole cusp`, the tilt angle θ, called `theta`, of default value zero and given in radians.
  - Notes:
    * The `logarithmic cusp` / `dipole cusp` / `tilted dipole cusp` are intended for use with the `logarithmic` / `dipole` / `2D_tilted_dipole` interactions defined in the `manual_interaction` block; see Sec. 20.
    * The maximum expansion order for these bases is 1.
    * In periodic systems these bases must be combined with a cuspless cut-off function (e.g., `gaussian` or `spline`) in order to preserve the cusp behaviour.

- **Half-integer power basis**
  - Type: `half-integer power`
  - Description: a half-integer power basis.
  - Functional form:

$$\begin{aligned}\phi_k(\mathbf{r}) &= r^{s(k)/2} \quad , \quad s(k) > 0 \\ & \log(r) \quad , \quad s(k) \neq 0\end{aligned}$$

  - Constants: the exponents $s(k)/2$ (reals), named `k`k, which are by default set to $s(k) = k/2 - 1$ (i.e., generating the basis $r^{-1/2}$, $\log(r)$, $r^{1/2}$, etc.). Exponents are rounded to the nearest half integers, and can be positive, negative, or zero.
  - Parameters: none
  - Notes:
    * This basis has been used for work on cusp conditions in the presence of non-Coulomb interactions (this basis does not declare its coalescence bahaviour, which makes it useful for messing about), but it is likely not to be very useful in other contexts.

- **CHAMP 2-body Jastrow factor basis**
  - Type: `CHAMP 2-body`
  - Description: CHAMP Jastrow factor basis for 2-body terms
  - Functional form: given a CHAMP core scaling function $f(r)$, the 2-body scaling function is $u(r) = (1 - f(r))/\kappa$, and the two-body basis is

$$\begin{aligned}\phi_1(\mathbf{r}) &= 1 \,, \\ \phi_2(\mathbf{r}) &= \frac{u(r)}{1 + a\kappa u(r)} \,, \\ \phi_k(\mathbf{r}) &= [u(r)]^{k-1} \quad , \quad k > 2 \,,\end{aligned}$$

    where κ is a scale parameter and a is a nested scale parameter. The following core scaling functions are available via the `iscale` integer constant:

`iscale=2`:	$f = e^{-\kappa r}$
`iscale=3`:	$f = e^{-\kappa r - (\kappa r)^2/2 - (\kappa r)^3/3}$
`iscale=4`:	$f = \frac{1}{1+\kappa r}$
`iscale=5`:	$f = 1 - \frac{\kappa r}{[1+(\kappa r)^3]^{1/3}}$

  - Constants: the choice of core scaling function f, named `iscale`, which is an integer of default value 4.
  - Parameters: the scale parameter κ, named `kappa`, which is 0.5 by default and must be positive, and the nested scale parameter a in ϕ_2, named `a`, which is 2.0 by default and must satisfy $a > -1$.
  - Notes:
    * In CHAMP one uses the same linear parameters for parallel- and antiparallel-spin electron channels, except for the cusp-setting prefactor of ϕ_2 and parameter a, both of which are spin-pair dependent. The purpose of the nested scaling parameter a in this context is to provide variational freedom to compensate for the rigidity of the otherwise spin-pair-independent form. Note that the *gjastrow* in CASINO does not currently support constraining the linear parameters to be spin-pair-independent during optimization if the basis function parameters are spin-pair-dependent, so this is not too useful here – one can still convert CHAMP-optimized Jastrows for use in CASINO VMC and DMC runs without problems.
    * The a parameter is redundant when the coefficient of ϕ_2 is zero, so, e.g., for cuspless e-n terms it should be explicitly declared as `fixed`.
    * The main purpose of the CHAMP bases in CASINO is to be able to use CHAMP-optimized Jastrow factors in CASINO for collaboration projects.

- **CHAMP 2-body Jastrow factor basis with cutoff**

- Type: `CHAMP 2-body cutoff`
- Description: CHAMP Jastrow factor bases with built-in cutoff for 2-body terms
- Functional form: given a CHAMP core scaling function $f(r)$, the 2-body scaling function is $u(r) = (1 - f(r))/\kappa$, and the two-body basis is

$$
\begin{aligned}
\phi_1(\mathbf{r}) &= \Theta(L-r) \ , \\
\phi_2(\mathbf{r}) &= \left\{ \frac{u(r)}{1+au(r)} - \frac{u_c}{1+au_c} \right\} \Theta(L-r) \ , \\
\phi_k(\mathbf{r}) &= \left\{ [u(r)]^{k-1} - u_c^{k-1} \right\} \Theta(L-r) \quad , \quad k>2 \ ,
\end{aligned}
$$

  where κ is a scale parameter, a is a nested scale parameter, L is the cutoff length, $f_c = f(L)$, $u_c = u(L)$, and Θ is the Heaviside step function. The following core scaling functions are available via the `iscale` integer constant:

  `iscale=6`: $f = e^{-\kappa r p(r)}$
  `iscale=7`: $f = \frac{1}{1+\kappa r p(r)}$

  where $p(r) = 1 - \frac{r}{L} + \frac{1}{3}\left(\frac{r}{L}\right)^2$.
- Constants: the choice of core scaling function f, named `iscale`, which is an integer of default value 7.
- Parameters: the scale parameter κ, named `kappa`, which is 0.5 by default and must be positive; the nested scale parameter a in ϕ_2, named `a`, which is 2.0 by default and must satisfy $a > -1$, and the cutoff length L, named `r_c`, which defaults to 6.0 in finite systems and to 99.9% of the radius of the sphere inscribed in the Wigner-Seitz cell in periodic systems.
- Notes:
  * The comments on the `CHAMP 2-body` basis above also apply here.
  * These basis functions go to zero smoothly to second order at L.
  * Functions 6 and 7 reduce to functions 2 and 4 of the `CHAMP 2-body` basis as $L \to \infty$.

• **CHAMP 3-body Jastrow factor basis**

- Type: `CHAMP 3-body`
- Description: CHAMP Jastrow factor bases for 3-body terms
- Functional form: given a CHAMP core scaling function $f(r)$, the 3-body scaling function is $v = f$. The 3-body basis is

$$
\phi_k(\mathbf{r}) = [v(r)]^{k-1} \ ,
$$

  where κ is a scale parameter. The following core scaling functions are available via the `iscale` integer constant:

`iscale=2`:	$f = e^{-\kappa r}$	`iscale=12`:	$f = e^{-\kappa r^2}$
`iscale=3`:	$f = e^{-\kappa r - (\kappa r)^2/2 - (\kappa r)^3/3}$		
`iscale=4`:	$f = \frac{1}{1+\kappa r}$	`iscale=14`:	$f = \frac{1}{1+\kappa r^2}$
`iscale=5`:	$f = 1 - \frac{\kappa r}{[1+(\kappa r)^3]^{1/3}}$		

  Functions with `iscale`>10 are cuspless variants of their `iscale`-10 counterparts.
- Constants: the choice of core scaling function f, named `iscale`, which is an integer of default value 4.
- Parameters: the scale parameter κ, named `kappa`, which is 0.5 by default and must be positive.
- Notes:
  * In order to optimize separate e-e, e-n, and e-e-n terms using the CHAMP bases it is advisable to ensure that low rank contributions to the e-e-n term are removed to avoid redundant parameters – i.e., coefficients `c_1,1,`k, `c_1,`k`,1`, `c_`k`,1,1`, should be declared as `c_...: [ 0, fixed ]` in all channels of the e-e-n term.

• **CHAMP 3-body Jastrow factor basis with cutoff**

- Type: `CHAMP 3-body cutoff`
- Description: CHAMP Jastrow factor basis with built-in cutoff for 3-body terms
- Functional form: given a CHAMP core scaling function $f(r)$, the 3-body scaling function is $v = (f - f_c)/(1 - f_c)$. The 3-body basis is

$$\phi_k(\mathbf{r}) \quad = \quad [v(r)]^{k-1}\,\Theta(L-r)\ ,$$

  where κ is a scale parameter, L is the cutoff length, $f_c = f(L)$, and Θ is the Heaviside step function. The following core scaling functions are available via the `iscale` integer constant:

`iscale=6`:	$f = e^{-\kappa r p(r)}$	`iscale=16`:	$f = e^{-\kappa r^2 q(r)}$
`iscale=7`:	$f = \frac{1}{1+\kappa r p(r)}$	`iscale=17`:	$f = \frac{1}{1+\kappa r^2 q(r)}$

  where $p(r) = 1 - \frac{r}{L} + \frac{1}{3}\left(\frac{r}{L}\right)^2$ and $q(r) = 1 - \frac{4}{3}\frac{r}{L} + \frac{1}{2}\left(\frac{r}{L}\right)^2$. Functions with `iscale`$> 10$ are cuspless variants of their `iscale`-10 counterparts.
- Constants: the choice of core scaling function f, named `iscale`, which is an integer of default value 7.
- Parameters: the scale parameter κ, named `kappa`, which is 0.5 by default and must be positive, and the cutoff length L, named `r_c`, which defaults to 6.0 in finite systems and to 99.9% of the radius of the sphere inscribed in the Wigner-Seitz cell in periodic systems.
- Notes:
  * The comments on the `CHAMP 3-body` basis above also apply here.
  * These basis functions go to zero smoothly to second order at L.

• **CHAMP 3-body Jastrow factor basis without parameters**

- Type: `CHAMP 3-body fix`
- Description: CHAMP Jastrow factor bases for 3-body terms without optimizable parameters
- Functional form: given a CHAMP core scaling function $f(r)$, the 3-body scaling function is $v = f$. The 3-body basis is

$$\phi_k(\mathbf{r}) \quad = \quad [v(r)]^{k-1}\ ,$$

  where κ is a fixed scale parameter. The following core scaling functions are available via the `iscale` integer constant:

`iscale=2`:	$f = e^{-\kappa r}$	`iscale=12`:	$f = e^{-\kappa r^2}$
`iscale=3`:	$f = e^{-\kappa r-(\kappa r)^2/2-(\kappa r)^3/3}$		
`iscale=4`:	$f = \frac{1}{1+\kappa r}$	`iscale=14`:	$f = \frac{1}{1+\kappa r^2}$
`iscale=5`:	$f = 1 - \frac{\kappa r}{[1+(\kappa r)^3]^{1/3}}$		

  Functions with `iscale`> 10 are cuspless variants of their `iscale`-10 counterparts.
- Constants: the choice of core scaling function f, named `iscale`, which is an integer of default value 4, and the scale parameter κ, named `kappa`, which is 0.5 by default and must be positive
- Parameters: none
- Notes:
  * This basis is identical to `CHAMP 3-body` but with parameter κ converted to a (channel-independent) constant. This allows us to treat the products of some basis functions at coalescence points as equal, which we cannot guarantee in the presence of optimizable parameters, so that the cusp constraints match those used by CHAMP.

• **CHAMP 3-body Jastrow factor basis with cutoff without parameters**

- Type: `CHAMP 3-body cutoff fix`
- Description: CHAMP Jastrow factor basis with built-in cutoff for 3-body terms without optimizable parameters

- Functional form: given a CHAMP core scaling function $f(r)$, the 3-body scaling function is $v = (f - f_c)/(1 - f_c)$. The 3-body basis is

  $$\phi_k(\mathbf{r}) \quad = \quad [v(r)]^{k-1}\,\Theta(L-r) \; ,$$

  where κ is a fixed scale parameter, L is the fixed cutoff length, $f_c = f(L)$, and Θ is the Heaviside step function. The following core scaling functions are available via the `iscale` integer constant:

`iscale=6`:	$f = e^{-\kappa r p(r)}$	`iscale=16`:	$f = e^{-\kappa r^2 q(r)}$
`iscale=7`:	$f = \frac{1}{1+\kappa r p(r)}$	`iscale=17`:	$f = \frac{1}{1+\kappa r^2 q(r)}$

  where $p(r) = 1 - \frac{r}{L} + \frac{1}{3}\left(\frac{r}{L}\right)^2$ and $q(r) = 1 - \frac{4}{3}\frac{r}{L} + \frac{1}{2}\left(\frac{r}{L}\right)^2$. Functions with `iscale`$> 10$ are cuspless variants of their `iscale`-10 counterparts.
- Constants: the choice of core scaling function f, named `iscale`, which is an integer of default value 7, the scale parameter κ, named `kappa`, which is 0.5 by default and must be positive, and the cutoff length L, named `r_c`, which defaults to 6.0 in finite systems and to 99.9999% of the radius of the sphere inscribed in the Wigner-Seitz cell in periodic systems.
- Parameters: none
- Notes:
  * This basis is identical to `CHAMP 3-body cutoff` but with parameters κ and L converted to (channel-independent) constants. This allows us to treat the products of some basis functions at coalescence points as equal, which we cannot guarantee in the presence of optimizable parameters, so that the cusp constraints match those used by CHAMP.

The available cut-off functions are:

- **Polynomial cut-off function**
  - `Type: polynomial`
  - Description: a polynomial of selectable truncation order C that goes smoothly to zero at the cut-off length L, and goes to 1 at $r = 0$.
  - Functional form:
    $$f(\mathbf{r}) = (1 - r/L)^C \Theta(L - r) \tag{27}$$
  - Constants: the truncation order C (integer), called `C`, with a default value of 3.
  - Parameters: the cut-off length L, called `L`, which in periodic systems has an upper limit of the radius of the sphere inscribed in the Wigner-Seitz cell, and has a default value which depends on various system characteristics.
  - Notes:
    * This is the cut-off function used in CASINO for the functions in the backflow transformation, and in the standard Jastrow factor for the H three-body term.
    * This cut-off function is technically equivalent to `alt polynomial` below, but is thought to improve numerical stability during optimization in some cases.
- **Alternative polynomial cut-off function**
  - `Type: alt polynomial`
  - Description: a polynomial of selectable truncation order C that goes smoothly to zero at the cut-off length L.
  - Functional form:
    $$f(\mathbf{r}) = (r - L)^C \Theta(L - r) \tag{28}$$
  - Constants: the truncation order C (integer), called `C`, with a default value of 3.
  - Parameters: the cut-off length L, called `L`, which in periodic systems has an upper limit of the radius of the sphere inscribed in the Wigner-Seitz cell, and has a default value which depends on various system characteristics.
  - Notes:
    * This is the cut-off function used in the standard Jastrow factor for the u, χ, f, W and D terms.

- **Gaussian cut-off function**
  - `Type: gaussian`
  - Description: Gaussian cut-off function (with hard cut-off at long distance).
  - Functional form:
$$f(\mathbf{r}) = \exp(-r^2/L^2)\Theta(L_{\rm hard} - r) \tag{29}$$
  - Constants: the hard cut-off length $L_{\rm hard}$ (real), called `L_hard`, which has a default value of the radius of the sphere inscribed in the Wigner-Seitz cell in periodic systems, or 1000 a.u. in finite systems.
  - Parameters: the Gaussian width L, called `L`, which has a default value of 3 a.u.
- **Smoothed-out step cut-off function**
  - `Type: spline`
  - Description: smoothed-out step cut-off function which is minimally invasive at short range.
  - Functional form:
$$\begin{aligned} f(\mathbf{r}) &= 1 \quad , \quad r < xL \\ & h_C\left(\frac{r-xL}{L-xL}\right) \quad , \quad xL < r < L \\ & 0 \quad , \quad r > L \end{aligned} \tag{30}$$
    where $h_C(s)$ is a polynomial such that $h_C(0) = 1$, $h_C(1) = 0$, and the first C derivatives at 0 and 1 are zero.
  - Constants: the spline order C (integer), named `C`, of default value 2, constrained to $0 \leq C \leq 3$.
  - Parameters: the hard cut-off length L (real), called `L`, which in periodic systems has an upper limit of the radius of the sphere inscribed in the Wigner-Seitz cell and has a default value which depends on various system characteristics, and the relative length of the constant portion of the function x (real), called `x`, which is limited to $x \in [0.05, 0.95]$ and has a default value of 0.5.
- **Anisotropic cut-off function**
  - `Type: anisotropic polynomial`
  - Polynomial cut-off times a spherical harmonic to construct anisotropic terms.
  - Functional form:
$$f(\mathbf{r}) = (1 - r/L)^C\Theta(L-r)\sum_i c_i \prod_\beta^d \left[\frac{\mathbf{r}\cdot\hat{\mathbf{u}}_\beta}{r}\right]^{p_\beta^{(i)}} \tag{31}$$
    where d is the dimensionality and $\hat{\mathbf{u}}_\beta$ is the βth unit vector of a potentially rotated reference frame.
  - Constants: The truncation order C (integer), called "`C`", with a default value of 3, the coefficients c_i and the exponents $p_\beta^{(i)}$ (reals), and optionally the $\hat{\mathbf{u}}$ vectors (which can be given unnormalized) that define any required rotated reference frames—see below for an example.
  - Parameters: the cut-off length L, called `L`, which in periodic systems has an upper limit of the radius of the sphere inscribed in the Wigner-Seitz cell, and has a default value which depends on various system characteristics.
  - Notes:
    * For e–e cut-offs it is possible to specify one rotated reference frame, while for e–n cut-offs up to one reference frame per nucleus can be defined. If no rotated reference frames are given, the standard one is used.
    * The sum of the exponents $\sum_\beta p_\beta^{(i)}$ should be the same for all terms in the sum for the function to be a valid spherical harmonic.
    * For example,

```
Constants:
  C: 3
  xyz_1: [ c:  3.0, p: [ 2, 1, 1 ] ]
  xyz_2: [ c: -1.0, p: [ 0, 3, 1 ] ]
  Frame 1:
    Atoms: [ 1, 2 ]
    u_1: [  1.0, 1.0, 0.0 ]
    u_2: [ -1.0, 1.0, 0.0 ]
    u_3: [  0.0, 0.0, 1.0 ]
  Frame 1:
    Atoms: [ 3, 4 ]
    u_1: [ 1.0,  0.0, 0.0 ]
    u_2: [ 0.0,  1.0, 1.0 ]
    u_3: [ 0.0, -1.0, 1.0 ]
```

defines two rotated reference frames, one for atoms 1 and 2 and another for atoms 3 and 4. This function pre-multiplies the Jastrow factor term by the spherical harmonic $\left(3x^2yz - y^3z\right)/r^4$ times the isotropic cut-off function $(1 - r/L)^3$.

- **Quasi-cusp function**
  - `Type: quasicusp`
  - Quasi-cusp function for multi-layer/multi-wire systems.
  - Functional form:

    $$f(\mathbf{r}) = \left(\sqrt{r^2+z^2} - \sqrt{L^2-z^2}\right)\left[1 - 6(r/L)^2 + 8(r/L)^3 - 3(r/L)^4\right]\Theta(L-r) \qquad (32)$$

    where z is the inter-layer separation.
  - Constants: none
  - Parameters: the cut-off length L, called `L`, which in periodic systems has an upper limit of the radius of the sphere inscribed in the Wigner-Seitz cell, and has a default value of 1 a.u.
  - Notes:
    * This function is intended to be used in a `Rank: [ 2, 0 ]` term without an `e-e basis`. It allows transitioning smoothly from, e.g., a single-layer 2D electron-hole gas to the bilayer case by introducing a cuspless wave function feature which tends to the electron-hole cusp as $z \to 0$. It should not be used in other circumstances since it is not a particularly good cut-off function.

- **Orbital cusp correction function**
  - `Type: orbital cusp`
  - Function designed to replicate the effect of orbital-based cusp-correction schemes from within the Jastrow factor.
  - Functional form:

    $$f(\mathbf{r}) = \left[\log\left(e^{\sum_{k=0}^{4}\alpha_k r^k} + C\right) - \log\phi_{\rm s}(r)\right]\Theta(L-r) \qquad (33)$$

    where $\phi_{\rm s}(r)$ is the spherical average of the orbital being cusp corrected (this is in practice a linear combination of $n_{\rm orb}$ orbitals, $\phi_{\rm s}(r) = \sum_{j=1}^{n_{\rm orb}} a_j\phi_j(r)$, each evaluated as a spline interpolation of values on a grid provided as contants).
  - Constants: the number of orbitals $n_{\rm orb}$, called "`norb`" (integer), the tabulation grid length $L_{\rm grid}$, called "`L_grid`" (real), the number of points $n_{|rmgrid}$ on the tabulation grid, called "`ngrid`" (integer), the nuclear charge Z, called "`Z`" (real), the function offset C, called "`C`" (real), and the $n_{\rm orb} \times (n_{\rm grid}+1)$ tabulated orbital values $\phi_j(r_i)$ at $r_i = (i/n_{\rm grid})/L_{\rm grid}$ for $i = 0, \ldots, n_{\rm grid}$, called "`Orbital` $<j>$`:phi_`$<i>$".
  - Parameters: the cusp radius L, called `L`, which is constrained to be between $L_{\rm grid}/n_{\rm grid}$ and $L_{\rm grid}$, coefficient α_0, named `alpha_0` (to be internally determined by the Kato cusp condition), and the orbital mixing coefficients a_i for $i = 2, \ldots, n_{\rm orb}$, called "`a_`$<i>$" (NB, a_1 is fixed to 1).
  - Notes:

    * This function is intended to be generated by CASINO using `runtype: gen_gpcc`, with `cusp_correction: F` and `use_gpcc: T`. After moving the contents of the resulting `gpcc.casl` to the Jastrow factor in the `parameters.casl` file, using the Jastrow-based cusp correction requires setting `cusp_correction: F` and `use_gpcc: F`.
    * In tests, the Jastrow-based cusp correction yields very similar results to the orbital cusp correction; in fact, the ability to optimize the cusp radius gives the Jastrow-based cusp correction a slight edge in some cases.
    * Using just the 1*s* orbital in the Jastrow-based cusp correction appears to suffice in small molecules.

There exist CASL keywords to control the behaviour of the *gjastrow* terms. The full keyword listing is:

- `Rank: [` *n* `,` *m* `]`
  Defines the number of electrons *n* and nuclei *m* involved in the term.

- `Rules: [` *rule1* `,` *rule2* `, ...]`
  Declares the symmetries that define parameter channels for two-body and many-body parameters. The empty ruleset, `Rules: [ ]`, corresponds to all particles and nuclei being regarded as different. By default CASINO generates a set of rules based on the basic symmetries of the system. To declare rules without removing the default rules, the syntax `Rules: [ default,` *rule1* `, ...]` can be used. Rules can be specified in any order. The available rule formats are:

Example	Effect
`1=2`	Makes two particle types equivalent
`1-1=1-2`	Makes two particle-particle pairs equivalent
`n1=n2`	Makes two nuclei equivalent
`1-n1=2-n1`	Makes two particle-nucleus pairs equivalent
`N`	Makes periodic images of all nuclei in the primitive cell equivalent
`N1`	Makes nucleus `n1` equivalent to all its periodic images
`N1=N2`	Makes two nuclei equivalent to each other, and to all their periodic images (same as `[ N1, N2, n1=n2 ]`)
`1-N1=2-N2`	Makes two particle-nucleus pairs equivalent, and each nucleus equivalent to all its periodic images (same as `[ N1, N2, 1-n1=2-n2 ]`)
`Z`	Makes nuclei with the same atomic number equivalent
`Z6`	Makes nuclei with a certain atomic number (6) equivalent
`Z1=Z11`	Makes all the nuclei with the given atomic numbers (1 and 11) equivalent (note that this is in general not terribly sensible)
`1-Z1=2-Z11`	Makes two particle-nuclei pairs equivalent
`!1`	Removes a particle type from a term
`!n1`	Removes a nucleus from a term
`!N1`	Removes a nucleus and all its periodic images from a term
`!Z1`	Removes all nuclei with the given atomic number from a term

- `e-e basis` *(block)*
  `e-n basis` *(block)*
  `e-e cutoff` *(block)*
  `e-n cutoff` *(block)*
  Define the type, expansion order, constants and parameters of the basis functions and cut-offs of the term. See above for details.

- `e-e cusp:` *boolean*
  `e-n cusp:` *boolean*
  By default CASINO applies the e–e cusp conditions on the first declared term which is capable of describing an e–e cusp, forcing any other such terms to be cuspless at coalescence points, and makes all terms capable of describing an e–n cusp cuspless. Setting `e-e cusp` and/or `e-n cusp` to `T` changes this behaviour.

  Note that it is rarely possible to apply cusp conditions to anything other than e–e and e–n terms. However CASINO is capable of applying cusp conditions to higher-rank terms when it is possible.

Note that the use of these flags could in principle could result in multiple terms having cusp conditions imposed on, which may or may not be sensible. CASINO will issue warnings if the cusp condition for any e–e or e–n pair is imposed more than once.

- `Waive e-e cusp: [` *e-e-pair1* `,` *e-e-pair2* `, ... ]`
  `Waive e-n cusp: [` *e-n-pair1* `,` *e-n-pair2* `, ... ]`
  When a term is used to impose the e–e and/or e–n cusp conditions, by default all e–e and/or e–n pairs in the term are forced to have cusps. The `Waive e-e cusp` and `Waive e-n cusp` blocks allow defining a list of e–e and e–n pairs which are to be kept cuspless. For example, an e–e term in an electronic system with `Waive e-e cusp: [ 1-1, 2-2 ]` would have cuspless channels for same-spin electrons and apply a cusp to the anti-parallel-spin channel (`1-2`) only.

- `Indexing` *(block)*
  The construction of some terms require imposing constraints on the linear parameters. These constraints could take the form of a linear system involving the parameters, for example. "Indexing constraints" are a class of constraints which instead manipulate the list of available parameter index sets, which is an efficient way of eliminating linear parameters.

  An index value of zero is given the special meaning of removing a basis function and its associated cut-off function in some of the indexing constraints below, allowing further flexibility in the construction of terms. This is only useful when a cut-off function is used or if the functional basis does not contain the function $\phi = 1$.

  The `Indexing` block may contain the following keyword-value pairs:

  - `Maximum sum:` *integer*
    The `Maximum sum` indexing constraint allows the construction of terms following a Boys-Handy-style indexing, where sum of the linear parameter indices is constrained not to exceed a certain value. This type of indexing is an efficient way of constructing affordable high-rank terms to accurately describe small assemblies of particles.

  - `All masks:` *boolean*
    If set to `T`, indices are allowed to take the value zero. For example this allows pure e–e, e–e–e, e–n and e–e–n contributions in an e–e–e–n term.

    In the generation of the list of valid index sets, boolean masks determining which indices are allowed to be non-zero are used. By default the only mask used is a vector filled with *true*. "`All masks`" refers to the fact that all possible masks are used in the generation of the list of index sets.

  - `e-e dot product:` *boolean*
    `e-n dot product:` *boolean*
    If `e-e dot product` (or `e-n dot product`) is set to `T`, the list of valid index sets will contain only index sets where all but two e–e (or e–n) indices are zero, i.e., the index generator uses only masks which contain two *true*s. Also, the two non-zero indices are required to have the same remainder when divided by the dimensionality of the system, and a linear constraint is imposed to equate parameters whose two non-zero indices divided by the dimensionality give the same integer. Used in conjunction with a *vectorial* basis this can be used to construct dot-product terms, such as the W and D terms in the standard CASINO Jastrow factor.

    Specifying `e-e dot product: no repeat` or `e-n dot product: no repeat` applies the above constraints and additionally removes parameters which correspond to dot products of vectors involving a repeated electron or nucleus such as $\mathbf{r}_{iI} \cdot \mathbf{r}_{iJ}$. This enables building, e.g., an e-e-n-n term around a dot product of the form $\mathbf{r}_{iI} \cdot \mathbf{r}_{jJ}$ to explicitly describe dipole-dipole interactions, without including spurious lower-rank contributions.

    Note that, by construction, the use of `e-e dot product` and `e-n dot product` is incompatible with the use of `All masks`.

- `Linear parameters` *(block)*
  Defines the linear parameters of the Jastrow factor term. See above for details.

  Note that the name given to channels inside this block is used as a model for cusps. E.g., if a cusp-imposing electron-electron term has the rule `1=2`, then both parallel (`1-1`) and antiparallel (`1-2`) electron pairs will have the same cusp condition applied; if one explicitly provides a `Linear parameters` block containing an empty `Channel 1-2` block, the cusp value will be that corresponding to antiparallel-spin electrons. Note that by default the first-occurring pair in a natural enumeration will be used as the model (i.e., `1-1`, unless there is only one up-spin electron in the system in which case `1-2` is used instead).

In addition to the parameters, the `Channel` blocks may contain the following keywords to further fine-tune the cusp behaviour:

– `e-e cusp:` *boolean*
  `e-n cusp:` *boolean*
  Per-channel override of e–e and e–n cusp condition application flags (see analogous per-term flags above).

– `e-e cusp model:` *string*
  `e-n cusp model:` *string*
  Per-channel override of e–e and e–n cusp model, e.g., `1-2` or `2-n3`.

– `e-e cusp value:` *real*
  `e-n cusp value:` *real*
  Per-channel override of e–e and e–n cusp value, e.g., `0.5` or `-6.0`. (This is the target value of $\frac{1}{\Psi}\frac{d\Psi}{dr}\Big|_{r=0}$, sometimes referred to as the Kato Γ value elsewhere in this manual).

7.8.2 Multi-geminal wave functions

Geminal and multi-geminal wave function parameters are defined in the `GEMINAL` top-level block of the `parameters.casl` file. More information about the specifics of these wave functions and on the format of the CASL block is available on request.

7.9 Compressed multi-determinant expansions: `mdet.casl` and `cmdet.casl`

CASINO will produce an `mdet.casl` file when **runtype** is set to `gen_mdet_casl` while using a multi-determinant expansion (see Sec. 7.4.5). This file describes the multi-determinant expansion (referring to the orbitals as integer indices but not specifying what they are), and is used by the **det_compress** utility to produce a cost-efficient compressed version of the expansion.

The `mdet.casl` file looks like this:

```
MDET:
  Title: Ne atom (numerical orbitals)
  Expansion:
    Term 1:
      Coefficient: [ -0.98314330000000005, Group: 1 ]
      Spin 1: [ 1, 2, 3, 4, 5 ]
      Spin 2: [ 1, 2, 3, 4, 5 ]
    Term 2:
      Coefficient: [ 2.9915557225902840E-003, Group: 2 ]
      Spin 1: [ 1, 2, 3, 4, 5 ]
      Spin 2: [ 1, 2, 3, 4, 6 ]
    Term 3:
      Coefficient: [ 2.9915557225902840E-003, Group: 2 ]
      Spin 1: [ 1, 2, 3, 4, 5 ]
      Spin 2: [ 1, 2, 4, 5, 7 ]
    Term 4:
      Coefficient: [ 2.9915557225902840E-003, Group: 2 ]
      Spin 1: [ 1, 2, 3, 4, 6 ]
      Spin 2: [ 1, 2, 3, 4, 5 ]
    Term 5:
      Coefficient: [ 2.9915557225902840E-003, Group: 2 ]
      Spin 1: [ 1, 2, 4, 5, 7 ]
      Spin 2: [ 1, 2, 3, 4, 5 ]
    Term 6:
      Coefficient: [ -2.9915557225902840E-003, Group: 2 ]
      Spin 1: [ 1, 2, 3, 4, 5 ]
      Spin 2: [ 1, 2, 3, 5, 8 ]
    Term 7:
      Coefficient: [ -2.9915557225902840E-003, Group: 2 ]
      Spin 1: [ 1, 2, 3, 5, 8 ]
      Spin 2: [ 1, 2, 3, 4, 5 ]
    ...
```

The **det_compress** utility will produce a `cmdet.casl` file describing the original, de-duplicated and compressed expansion. If a `cmdet.casl` is present in the directory where CASINO is being run, it will be used automatically.

The `cmdet.casl` file looks like this:

```
CMDET:
  Title: Ne atom (numerical orbitals)
  Original expansion:
    Term 1:
      Coefficient: [ -0.98314330000000005, Group: 1 ]
      Spin 1: [ 1, 2, 3, 4, 5 ]
      Spin 2: [ 1, 2, 3, 4, 5 ]
    Term 2:
      Coefficient: [ 2.9915557225902840E-003, Group: 2 ]
      Spin 1: [ 1, 2, 3, 4, 5 ]
      Spin 2: [ 1, 2, 3, 4, 6 ]
    ...
  Deduplicated coefficients:
    c_1: [ 1 ]
    c_2: [ 2 ]
    ...
    c_103: [ -124 ]
    c_104: [ 125, 150 ]
    ...
    c_764: [ 993 ]
    c_765: [ 994 ]
    c_766: [ 995 ]
    c_767: [ 996 ]
    ...
  Compressed expansion:
    Orbital pool:
      Orbital 1:
        Component 1: [ 1 ]
      Orbital 2:
        Component 1: [ 2 ]
      ...
      Orbital 47:
        Component 1: [ 2 ]
        Component 2: [ -9, Num: [ 764 ], Den: [ 1 ] ]
        Component 3: [ -51, Num: [ 766 ], Den: [ 1 ] ]
      Orbital 48:
        Component 1: [ -9, Num: [ 765 ], Den: [ 767 ] ]
        Component 2: [ -51 ]
      ...
    Expansion:
      Term 1:
        Coefficient: [ Num: [ -1 ] ]
        Spin 1: [ 1, 2, 3, 4, 5 ]
        Spin 2: [ 1, 3, 4, 5, 47 ]
      Term 2:
        Coefficient: [ Num: [ -767 ] ]
        Spin 1: [ 1, 3, 4, 5, 48 ]
        Spin 2: [ 1, 2, 3, 4, 5 ]
      ...
```

For more details on the compression algorithm see Ref. [31]. The file `utils/det_compress/README` in the distribution also contains important information on the method and the format of these files.

7.10 Orbital files: `awfn.data`, `bwfn.data`, `dwfn.data`, `gwfn.data`, `pwfn.data` and `stowfn.data`

These files contain the geometry and the orbital and determinant data produced by the wave-function generating code. The data can be given in a Gaussian basis set (`gwfn.data`), in a plane-wave basis set (`pwfn.data`) or in a blip function basis set (`bwfn.data`). The `awfn.data` file contains a trial wave function for an atom with the orbitals given explicitly on a radial grid.

Without going into details of specific formats, the files basically contain the following information:

- Basic information about the trial wave-function generating calculation (e.g., DFT/HF/etc.), including total energy and components;
- Geometry of the system;
- Details of the **k**-space net used by the program that generated the trial wave function (only if the systems is periodic);
- Details of the basis set:
  - Exponents, contraction coefficients and shell types (Gaussians);
  - G-vectors of the plane waves;
  - Blip grid;
- Multideterminant properties of the trial wave function (if any);
- Specification of the orbitals:
  - Gaussian coefficients;
  - Plane-wave coefficients;
  - Blip coefficients;
- Eigenvalue spectrum (used to work out which orbitals to occupy and, crudely, whether the system is a metal or an insulator);
- The output file from the program that generated the trial wave function (not read by CASINO—for reference only).

These files are generated automatically by various utilities available for different electronic structure programs (see Secs. 8 and 9). The format of the different files should be clear from looking at the various examples for different dimensionalities and basis sets.

If correlated sampling is introduced into CASINO, it may be necessary to define several `xwfn.data` files. These will be named `xwfn.data.1`, `xwfn.data.2`, `xwfn.data.3`, ...

7.10.1 `gwfn.data` file specification

The `gwfn.data` file contains orbitals expanded in a Gaussian basis set. The original specification is below, followed by an actual example:

```
Dimensions of allocated arrays are shown.

For the larger fields I use the following formatting.

 % strings (title,code,method,functional) may be up to 80 characters long.
 * free format
 $ FORMAT(3(1PE20.13)) - for reals naturally grouped in triples (e.g. coords)
 & FORMAT(4(1PE20.13)) - other reals (e.g. gaussian exponents)
 @ FORMAT(8I10)        - lengthy lists of integers

use e.g. write(IO,format=&)(array(i),i=1,size)

First line of file is TITLE field.
```

```
  MDT 1997

--------------------------------------------------------------------------------

  % TITLE (the title)

  BASIC INFO
  ----------
  % CODE:
    name of code producing this file
    (i.e. CRYSTAL95/98/03/06/09/14/17 GAUSSIAN94/98/03/09, TURBOMOLE)
  % METHOD:
    Comment - RHF/ROHF/UHF/DFT/S-DFT/CI/etc.
  % FUNCTIONAL
    DFT functional name. If method not DFT then ‘none’.
  * PERIODICITY:
    system dimension 0,1,2,3 => molecule, polymer, slab, solid
  * SPIN_UNRESTRICTED:
    .true. or .false. (i.e. different orbitals for different spins)
  * EIONION:
    nuclear-nuclear repulsion energy (au/atom)
  * NUM_ELECTRONS:
    number of electrons per primitive cell

  GEOMETRY
  --------
  * NUM_ATOMS:
    number of atoms per primitive cell
  $ BASIS(3,NUM_ATOMS):
    atomic positions(au)
  @ ATNO(NUM_ATOMS):
    atomic numbers for each atom (add 200 to flag atom with pseudopotential)
  & VALENCE_CHARGE(NUM_ATOMS):
    valence charges for each atom (=atno for all-electron case or e.g. H pseudo)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% PERIODIC INSERT%%%%%%%%%%%%%%%%%%%%%%%%%%%
  $ LATTICE_VECTORS(3,3):
    primitive lattice vectors (au)

  K SPACE NET
  -----------
  * NUM_K:
    no. of k points in BZ
  * NUM_REAL_K:
    no. of ‘real’ k points
    (all components of ‘real’ k points are either zero or half a
     reciprocal lattice vector)
  $ KVEC(3,NUM_K):
    k point coordinates (a.u.)
    NB: coordinates of ‘real’ k points must occupy the first num_real_k
    positions in kvec.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% END PERIODIC INSERT%%%%%%%%%%%%%%%%%%%%%%%

  BASIS SET
  ---------
  * NUM_CENTRES
    number of centres with associated Gaussian functions
    (i.e. number of atoms + number of nonatom Gaussian sites) per primitive cell
  * NUM_SHELLS:
    number of shells per primitive cell
  * NUM_AO:
    number of basis functions (’AO’) per primitive cell
  * NUM_PRIMS:
    number of Gaussian primitives per primitive cell
  * HIGHEST_ANG_MOM:
    highest angular momentum of shells (max 4 for periodic, 5 for finite system)
  @ SHELL_AM(NUM_SHELLS):
```

```
  code for shell type
  s=1, sp=2, p=3, d= 4, f= 5 etc. (harmonic representation)
                d=-4, f=-5      (cartesian representation - not implemented)
@ NUMPRIMS_IN_SHELL(NUM_SHELLS):
  Number of primitive Gaussians in each shell
@ FIRST_SHELL(NUM_CENTRES+1)
  Sequence number of first shell on each Gaussian centre.
  Allows e.g.
  do n=1,num_centres
   do shell=first_shell(n),first_shell(n+1)-1
    blah.
   enddo
  enddo
  to loop over shells on each centre
  Note dimension.
& EXPONENT(NUM_PRIMS):
  exponents of Gaussian primitives
& C_PRIM(NUM_PRIMS)
  contraction coefficients 'normalized' without m-dependent normalization
  (see note above)
& C_PRIM2(NUM_PRIMS) **must be omitted if no sp shells in basis**
  2nd contraction coefficients 'normalized' without m-dependent normalization
  (i.e. p coefficient for sp shells, zero otherwise)
  (see note above)
$ SHELL_POS(3,NUM_SHELLS)
  positions of shells (not necessarily atom-centred)

MULTIDETERMINANT INFORMATION
----------------------------

% GS - Ground state calculation

or

% SD - Single determinant inc. excitations/additions/subtractions

  Example:
         SD                    Single det calculation
         DET 1 1 PR 2 1 5 1    Promote electron in band 2 kpoint 1
                               to band 5 kpoint 1 in determinant 1,
                               spin 1 ("up")

or

% MD - Multiple determinants
         MD                    Multideterminant
         3                     3 determinants
         1.d0                  Determinant 1 prefactor
         2.d0                  Determinant 2 prefactor
        -1.d0                  Determinant 3 prefactor
         DET 3 1 PR 2 1 5 1    Promotion as examples above


ORBITAL COEFFICIENTS
--------------------

& CK(NUM_REAL_K*NUM_AO*NUM_AO + NUM_COMPLEX_K*NUM_AO*NUM_AO)
  block as follows
   ----spin (spin-polarized calcs only)---------------------------------
  /                                                                     \
   ----k (for solids)-------------------------------
  /                                                 \
   ----bands-(solids)_or MOs (molecules)------
  /                                           \
   -AO-basis functions grouped by shell-
  /                                     \
  Complex coefficients are given as 2 adjacent real numbers (real,imaginary).
  Ordering of d orbital coefficients;
  z2, xz, yz, x2-y2, xy
  Ordering of f,g,..: m=0,1,-1,2,-2,3,-3,4,-4.....

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% PERIODIC INSERT%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
EIGENVALUES
-----------
* k  K_INDEX  KX  KY  KZ
& List of the eigenvalues of the MOs at this k point.
  (likewise for the remainning k points)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% END PERIODIC INSERT%%%%%%%%%%%%%%%%%%%%%%%%%%
OTHER STUFF
-----------

* Anything you like can be written here - intended for copy of input/output
  files from DFT/HF/etc. program.
```

Note the potential issues with 'normalizing' contraction coefficients, e.g., with the CRYSTAL program the contraction coefficients are multiplied by that part of the normalization factor which is independent of m; the CRYSTAL output does not in fact contain the m-dependent part at all. This must be artificially multiplied into the orbital coefficients by the CRYSTAL to CASINO converter. Authors of other interfaces must ensure this is done properly.

A second issue is that for d functions only (not f or g), the scale factors of the real solid harmonics (e.g., the 3 in $3xy$) are not explicitly evaluated by the CASINO orbital evaluator, but must be pre-multiplied into the orbital coefficients. Again, the CRYSTAL to CASINO converter does this.

These issues are discussed with reference to the CRYSTAL code in MDT's document in `CASINO/examples/generic/gauss_dfg/README` which also contains a 'test suite' to verify that higher angular momentum functions are being treated correctly.

The expressions for the real solid harmonics assumed by CASINO are given in the table on p. 170 of Pisani, Dovesi and Roetti, *Hartree–Fock ab initio treatment of crystalline systems* (Springer-Verlag, 1988). This can be found online.

An example of a `gwfn.data` file for a silicon crystal is given below:

```
Silicon RHF

BASIC INFO
----------
Generated by:
 CRYSTAL2003
Method:
 RHF
DFT functional:
 None
Periodicity:
 3
Spin unrestricted:
 .false.
Nuclear repulsion energy (au/atom):
 -4.200509061764448
Number of electrons per primitive cell
 8

GEOMETRY
--------
Number of atoms
 2
Atomic positions (au)
 1.2824155389174E+00 1.2824155389174E+00 1.2824155389174E+00
-1.2824155389174E+00-1.2824155389174E+00-1.2824155389174E+00
```

```
Atomic numbers for each atom
 214 214
Valence charges for each atom
 4.0000000000000E+00 4.0000000000000E+00
Primitive lattice vectors (au)
 0.0000000000000E+00 5.1296621556694E+00 5.1296621556694E+00
 5.1296621556694E+00 0.0000000000000E+00 5.1296621556694E+00
 5.1296621556694E+00 5.1296621556694E+00 0.0000000000000E+00

K SPACE NET
-----------
Number of k points
 8
Number of ‘real’ k points on BZ edge
 8
k point coordinates (au)
 0.0000000000000E+00 0.0000000000000E+00 0.0000000000000E+00
-3.0621828087817E-01 3.0621828087817E-01 3.0621828087817E-01
 3.0621828087817E-01-3.0621828087817E-01 3.0621828087817E-01
 3.0621828087817E-01 3.0621828087817E-01 3.0621828087817E-01
 3.0621828087817E-01 3.0621828087817E-01-3.0621828087817E-01
 6.1243656175634E-01 0.0000000000000E+00 0.0000000000000E+00
 0.0000000000000E+00 6.1243656175634E-01 0.0000000000000E+00
 0.0000000000000E+00 0.0000000000000E+00 6.1243656175634E-01

BASIS SET
---------
Number of Gaussian centres
 2
Number of shells per primitive cell
 10
Number of basis functions (’AO’) per primitive cell
 42
Number of Gaussian primitives per primitive cell
 10
Highest shell angular momentum (s/p/d/f/g... 1/2/3/4/5...)
 3
Code for shell types (s/sp/p/d/f... 1/2/3/4/5...)
         2         2         2         4         2         2         2         2
         4         2
Number of primitive Gaussians in each shell
         1         1         1         1         1         1         1         1
         1         1
Sequence number of first shell on each centre
         1         6        11
Exponents of Gaussian primitives
 2.0871000000000E+00 1.1464000000000E+00 3.4240000000000E-01 4.4610000000000E-01
 1.1830000000000E-01 2.0871000000000E+00 1.1464000000000E+00 3.4240000000000E-01
 4.4610000000000E-01 1.1830000000000E-01
’Normalized’ contraction coefficients without m-dependent normalization
 3.0098420484637E+00 1.9203853959954E+00 7.7586574622601E-01 9.7474724462951E-01
 3.4964298562377E-01 3.0098420484637E+00 1.9203853959954E+00 7.7586574622601E-01
 9.7474724462951E-01 3.4964298562377E-01
2nd ’normalized’ contraction coefficients (p coeff for sp shells, 0.0 otherwise)
 8.6965165911299E+00 4.1123159952884E+00 9.0799499001157E-01 0.0000000000000E+00
 2.4051778078084E-01 8.6965165911299E+00 4.1123159952884E+00 9.0799499001157E-01
 0.0000000000000E+00 2.4051778078084E-01
Position of each shell (au)
 1.2824155389174E+00 1.2824155389174E+00 1.2824155389174E+00
 1.2824155389174E+00 1.2824155389174E+00 1.2824155389174E+00
 1.2824155389174E+00 1.2824155389174E+00 1.2824155389174E+00
 1.2824155389174E+00 1.2824155389174E+00 1.2824155389174E+00
 1.2824155389174E+00 1.2824155389174E+00 1.2824155389174E+00
-1.2824155389174E+00-1.2824155389174E+00-1.2824155389174E+00
-1.2824155389174E+00-1.2824155389174E+00-1.2824155389174E+00
-1.2824155389174E+00-1.2824155389174E+00-1.2824155389174E+00
```

```
-1.2824155389174E+00-1.2824155389174E+00-1.2824155389174E+00
-1.2824155389174E+00-1.2824155389174E+00-1.2824155389174E+00

MULTIDETERMINANT INFORMATION
----------------------------
GS

ORBITAL COEFFICIENTS
---------------------------
 7.9927874568142E-02 0.0000000000000E+00 0.0000000000000E+00 0.0000000000000E+00
-2.8262251738009E-01 0.0000000000000E+00 0.0000000000000E+00 0.0000000000000E+00
<snip>
-1.8480536384432E-03 0.0000000000000E+00 0.0000000000000E+00 0.0000000000000E+00
 0.0000000000000E+00 0.0000000000000E+00 3.1778950736229E-01 0.0000000000000E+00

EIGENVALUES
-----------
k      1    0.00000000    0.00000000    0.00000000
-8.3654527718958E-01-2.0855050412594E-01-2.0855050412594E-01-2.0855050412594E-01
 1.1891116371242E-01 1.1891116371242E-01 1.1891116371242E-01 1.6642614221042E-01
 4.8905871448045E-01 5.5993323168592E-01 5.5993323168592E-01 7.0605030728642E-01
 8.6499950558517E-01 8.6499950558517E-01 8.6499950558517E-01 1.0845740016273E+00
 1.0845740016273E+00 1.0845740016273E+00 1.1874293233021E+00 1.1874293233021E+00
 1.3682921598936E+00 1.3682921598936E+00 1.3682921598936E+00 1.5285057402088E+00
 1.5285057402088E+00 1.5285057402088E+00 3.0863962899448E+00 3.0863962899448E+00
 3.0863962899448E+00 3.1999760552302E+00 3.6143223221555E+00 3.6143223221555E+00
 3.6143223221555E+00 3.6584331265112E+00 9.0595232556836E+00 9.0998780449430E+00
 9.0998780449430E+00 9.0998780449430E+00 9.4963012296477E+00 9.4963012296477E+00
 9.4963012296477E+00 9.5221533702082E+00
k      2   -0.30621828    0.30621828    0.30621828
-8.3195540313868E-01-2.4528906228096E-01-2.1458434393753E-01-2.1458434393753E-01
 1.2112567282366E-01 1.2873111845318E-01 1.2873111845318E-01 1.9164631417601E-01
 5.0097442550864E-01 5.5884325681737E-01 5.5884325681737E-01 6.8605998196272E-01
 8.7087516518368E-01 8.7097496240349E-01 8.7097496240349E-01 1.0564198656000E+00
 1.0687921848831E+00 1.0687921848831E+00 1.1911137685880E+00 1.1911137685880E+00
 1.3813695578425E+00 1.3957621401173E+00 1.3957621401173E+00 1.5386574220184E+00
 1.5386574220184E+00 1.5619812446828E+00 3.0953070972747E+00 3.1171873104694E+00
 3.1171873104694E+00 3.1553855866583E+00 3.6172352384823E+00 3.6172352384823E+00
 3.6473503138099E+00 3.7104355682694E+00 9.0295682518751E+00 9.1131656229938E+00
 9.1189605126448E+00 9.1189605126448E+00 9.4985126051165E+00 9.4985126051165E+00
 9.5096928174714E+00 9.5640262470037E+00
k      3    0.30621828   -0.30621828    0.30621828
<snip>

Input and output files for this calculation (not read by CASINO)
================================================================================

<deleted>
```

7.10.2 pwfn.data file specification

The `pwfn.data` file contains orbitals expanded in a plane-wave basis set. An example is shown below:

```
  Si diamond

  BASIC INFO
  ----------
  Generated by:
   CASTEP
  Method:
   DFT
  DFT Functional
   LDA
  Pseudopotential
   LDA Trouiller-Martin (1551 coeff)
```

```
 Plane-wave cutoff (au)
  7.5
 Spin polarized:
  F
 Total energy (au per primitive cell)
  -7.84635517057440
 Kinetic energy (au per primitive cell)
   3.24780615624099
 Local potential energy (au per primitive cell)
  -1.03508540683458
 Nonlocal potential energy (au per primitive cell)
  0.144898714904598
 Electron-electron energy (au per primitive cell)
  -1.80295658630558
 Ion-ion energy (au per primitive cell)
  -8.40101804857984
 Number of electrons per primitive cell
   8

 GEOMETRY
 --------
 Number of atoms per primitive cell
  2
 Atomic numbers and positions of atoms (au)
  14   1.2824155389173550   1.2824155389173550   1.2824155389173550
  14  -1.2824155389173550  -1.2824155389173550  -1.2824155389173550
 Primitive lattice vectors (au)
  0.000000000000000E+000   5.12966215566942        5.12966215566942
   5.12966215566942       0.000000000000000E+000   5.12966215566942
   5.12966215566942        5.12966215566942       0.000000000000000E+000

 G VECTORS
 ---------
 Number of G-vectors
         341
 Gx Gy Gz (au)
  0.000000000000000E+000  0.000000000000000E+000  0.000000000000000E+000
<snip>
   2.44974624702536        2.44974624702536        2.44974624702536

 WAVE FUNCTION
 -------------
 Number of k-points
   8
 k-point # ; # of bands (up spin/down spin) ; k-point coords (au)
   1 4 0 0.0 0.0 0.0
 Band, spin, eigenvalue (au)
   1 1 -2.432668987064920E-002
 Eigenvector coefficients
 -0.953978956601082       OR      (1.123456789, 9.876543210)  if complex
<snip>
  0.000000000000000E+000
 Band, spin, eigenvalue (au)
   2 1 0.417639684278199
 Eigenvector coefficients
 -1.576227515378431E-012
<snip>
  0.000000000000000E+000
 Band, spin, eigenvalue (au)
   3 1 0.417639711691157
 Eigenvector coefficients
  6.348405363267598E-008
<snip>
  0.000000000000000E+000
 Band, spin, eigenvalue (au)
   4 1 0.417639745897256
 Eigenvector coefficients
 -2.030042411030856E-009
<snip>
  0.000000000000000E+000
 k-point # ; # of bands (up spin/down spin) ; k-point coords (au)
   2 4 0 -0.3062182808781698 -0.3062182808781698 0.3062182808781698
 Band, spin, eigenvalue (au)
   1 1 6.261478573034787E-002
 Eigenvector coefficients
  0.683590846620266
<snip>
  0.000000000000000E+000
 Band, spin, eigenvalue (au)
           2           1  0.156856931414136
etc. for all 8 k points with 4 bands per k point in this case.

OTHER STUFF
-----------

* Anything you like can be written here - intended for copy of input/output
  files from the PW DFT program.
```

7.10.3 `bwfn.data` file specification

The `bwfn.data` file contains orbitals represented by blip functions (though these files can get so large that they are normally converted by CASINO to the binary format `bwfn.data.bin`. An alternative binary format `bwfn.data.b1` is produced natively by the PWSCF code.) An example of a formatted `bwfn.data` file is shown below:

```
 Bulk Si

 BASIC INFO
 ----------
 Generated by:
  PWSCF
 Method:
  DFT
 DFT Functional:
  unknown
 Pseudopotential
  unknown
 Plane-wave cutoff (au)
  7.5
 Spin polarized:
  F
 Total energy (au per primitive cell)
  -62.76695352625196
 Kinetic energy (au per primitive cell)
  0.E+0
 Local potential energy (au per primitive cell)
  0.E+0
 Non local potential energy(au per primitive cell)
  0.E+0
 Electron electron energy (au per primitive cell)
  0.E+0
 Ion ion energy (au per primitive cell)
  -67.208144397949283
 Number of electrons per primitive cell
  64

 GEOMETRY
 --------
 Number of atoms per primitive cell
  16
 Atomic number and position of the atoms(au)
```

```
    14     -1.282415538750     -1.282415538750     -1.282415538750
    14     -1.282415538750      3.847246616250      3.847246616250
     :
     :
    14     11.541739848750      6.412077693750      6.412077693750
    14     11.541739848750     11.541739848750     11.541739848750
Primitive lattice vectors (au)
     10.259324310000     10.259324310000      0.000000000000
      0.000000000000     10.259324310000     10.259324310000
     10.259324310000      0.000000000000     10.259324310000

G VECTORS
---------
Number of G-vectors
 2085
Gx Gy Gz (au)
      0.000000000000      0.000000000000      0.000000000000
     -0.306218280918     -0.306218280918     -0.306218280918
     -0.306218280918     -0.306218280918      0.306218280918
         :
         :
      2.143527966427      2.755964528263      1.531091404591
      1.531091404591      3.368401090099      0.918654842754
Blip grid
  20  20  20

WAVE FUNCTION
-------------
Number of k-points
 1
k-point # ; # of bands (up spin/down spin) ; k-point coords (au)
   1  32   0  0.0000000000000000  0.0000000000000000  0.0000000000000000
Band, spin, eigenvalue (au), localized
    1    1     -0.015443213499 F
Blip coefficients
     -0.347534087942
     -0.517756216863
     -0.528529751776
       :
       :
     -0.450993755155
      0.024823593723
     -0.022992059309
Band, spin, eigenvalue (au)
    2    1     -0.009057759900
Blip coefficients
     -0.097602672632
     -0.176031268912
     -0.188226226118
       :
       :
    (for 32 bands)
```

7.10.4 `awfn.data` file specification

The `awfn.data` file contains atomic orbitals represented numerically on a radial grid. An example is shown below:

```
Atomic Be wave function in real space
Atomic number
 4
Total number of orbitals
 2
The 1s(2)2s(2) [1S] state electronic configuration
Number of up, down spin electrons
```

```
  2  2
 States
 1  1  0  0                 % label of spin up electron, quantum number n, l, m
 2  2  0  0
 1  1  0  0                 % label of spin down electron, n, l, m
 2  2  0  0
 Radial grid (a.u.)
    301                     % Number of radial grid points are given
     0.000000000000000E+00  % Distance from centre of atom r
     0.457890972218354E-02
     0.477372816593466E-02
     0.497683553179352E-02
     0.518858448775392E-02
     0.540934270674177E-02
     0.563949350502860E-02
      :
      :
     0.108431746952108E+04
     0.113045182349640E+04
     0.117854905151603E+04
 Orbital # 1 [1s]
  0 1 0                      % spin type [0=unpolarized, 1=up, 2=down], n, l
     0.000000000000000E+00   % r * Value of trial wave function at point r
     0.659528705936585E-01
     0.687054582044431E-01
     0.715705591983797E-01
      :
      :
     0.000000000000000E+00
     0.000000000000000E+00
     0.000000000000000E+00
 Orbital # 2 [2s]
  0 2 0
     0.000000000000000E+00
     0.120187930416515E-01
     0.125203784849961E-01
     0.130424628552784E-01
      :
      :
     0.000000000000000E+00
     0.000000000000000E+00
     0.000000000000000E+00
     0.000000000000000E+00
```

7.10.5 `dwfn.data` file specification

The `dwfn.data` file contains molecular dimer orbitals represented numerically on a radial grid. An example is shown below (given in full in `CASINO/examples/numerical_dimer/o2`):

```
Tabulated dimer wave functions in real space
 O2 R=2.283
Nuclei-Nuclei distance
 2.283
Atomic numbers
 8 8
Total number of orbitals
 9
Number up/downspin electrons, determinants
 9 7 12
States
 1  1  0  0
 2  2  0  0
 3  3  0  0
 4  4  0  0
 :
  :
  4  4  0  0
  5  5  0  0
  6  8  1 -1
  7  8  1  1
 Grid, niXmu
  169 193
  0.186999562713678E-01 0.221327072304881E-01
 Orbital number 1 (sigma g)
    0.860779272842285E+01
    0.859406979250750E+01
    0.855304113006462E+01
    0.848512471670455E+01
  :
  :
    0.833254905465229E-32
    0.556321049814705E-32
    0.389883298880131E-32
    0.334392266810106E-32
 Orbital number 2 (sigma u)
   -0.860882690581978E+01
   -0.859510169013196E+01
   -0.855406620855265E+01
   -0.848613849691955E+01
   -0.839200713356398E+01
etc. etc.
```

7.10.6 `stowfn.data` file specification

The `stowfn.data` file contains Slater-type orbitals for atoms, molecules, polymers, slabs or solids. Slater-type orbitals have the form

$$\psi_j(\mathbf{r}_i) = \sum_k c_{jk}\phi_k\left(\mathbf{r}_i - \mathbf{R}_{I(k)}\right) \ , \tag{34}$$

where ϕ_k is an atomic orbital centred at nucleus $I(k)$ of the form

$$\phi_k(\mathbf{r}) = \rho_k(r) Y_{l_k m_k}(\theta,\phi) \ , \tag{35}$$

where $\rho_k(r) = \exp(-\zeta_k r) r^{n_k}$ is a radial function, l_k and m_k are angular-momentum quantum numbers, and Y_{lm} is a normalized real spherical harmonic.

An example `stowfn.data` file is shown here:

```
CH4 molecule

BASIC INFO
----------
Generated by:
 ADF
Periodicity:
 0
Spin unrestricted:
 .false.
Nuclear repulsion energy (au/atom):
 2.696727236672644
Number of electrons
 10

GEOMETRY
--------
Number of atoms
 5
Atomic positions (au)
 0.0000000000000E+00 0.0000000000000E+00 0.0000000000000E+00
 1.1849716639362E+00 1.1849716639362E+00 1.1849716639362E+00
```

```
-1.1849716639362E+00-1.1849716639362E+00 1.1849716639362E+00
 1.1849716639362E+00-1.1849716639362E+00-1.1849716639362E+00
-1.1849716639362E+00 1.1849716639362E+00-1.1849716639362E+00
Atomic numbers for each atom
         6         1         1         1         1
Valence charges for each atom
 6.0000000000000E+00 1.0000000000000E+00 1.0000000000000E+00 1.0000000000000E+00
 1.0000000000000E+00

BASIS SET
---------
Number of STO centres
 5
Position of each centre (au)
 0.0000000000000E+00 0.0000000000000E+00 0.0000000000000E+00
 1.1849716639362E+00 1.1849716639362E+00 1.1849716639362E+00
-1.1849716639362E+00-1.1849716639362E+00 1.1849716639362E+00
 1.1849716639362E+00-1.1849716639362E+00-1.1849716639362E+00
-1.1849716639362E+00 1.1849716639362E+00-1.1849716639362E+00
Number of shells
 51
Sequence number of first shell on each centre
         1        16        25        34        43
Code for shell types (s/sp/p/d/f/g 1/2/3/4/5/6)
         1         1         1         1         1         1         1         3
         3         3         3         4         4         5         5         1
         1         1         1         1         3         3         4         4
         1         1         1         1         1         3         3         4
         4         1         1         1         1         1         3         3
         4         4         1         1         1         1         1         3
         3         4         4
Order of radial prefactor r in each shell
         0         0         0         1         1         1         1         0
         0         0         0         0         0         0         0         0
         0         0         0         0         0         0         0         0
         0         0         0         0         0         0         0         0
         0         0         0         0         0         0         0         0
         0         0         0         0         0         0         0         0
         0         0         0
Exponent in each STO shell
 1.1400000000000E+01 6.6000000000000E+00 4.8500000000000E+00 5.9000000000000E+00
 2.5500000000000E+00 1.6500000000000E+00 1.1600000000000E+00 5.1500000000000E+00
 2.4000000000000E+00 1.3000000000000E+00 7.8000000000000E-01 2.5000000000000E+00
 1.5000000000000E+00 4.0000000000000E+00 2.0000000000000E+00 3.3000000000000E+00
 2.0000000000000E+00 1.4000000000000E+00 1.0000000000000E+00 7.1000000000000E-01
 2.0000000000000E+00 1.0000000000000E+00 2.5000000000000E+00 1.5000000000000E+00
 3.3000000000000E+00 2.0000000000000E+00 1.4000000000000E+00 1.0000000000000E+00
 7.1000000000000E-01 2.0000000000000E+00 1.0000000000000E+00 2.5000000000000E+00
 1.5000000000000E+00 3.3000000000000E+00 2.0000000000000E+00 1.4000000000000E+00
 1.0000000000000E+00 7.1000000000000E-01 2.0000000000000E+00 1.0000000000000E+00
 2.5000000000000E+00 1.5000000000000E+00 3.3000000000000E+00 2.0000000000000E+00
 1.4000000000000E+00 1.0000000000000E+00 7.1000000000000E-01 2.0000000000000E+00
 1.0000000000000E+00 2.5000000000000E+00 1.5000000000000E+00
Number of basis functions ('AO')
 127
Number of molecular orbitals ('MO')
         5

MULTIDETERMINANT INFORMATION
----------------------------
GS

ORBITAL COEFFICIENTS (normalized AO)
------------------------------------
 1.6988226730536E-02 2.5547553149910E-01 8.0520564294501E-01-7.6530531473020E-02
-1.0796538835168E-02 1.8427415395707E-02-2.8375885251793E-02 0.0000000000000E+00
```

```
 0.0000000000000E+00 0.0000000000000E+00 0.0000000000000E+00 0.0000000000000E+00
   :
   :
-4.4321439503424E-02-2.6719231703795E-03-1.4778027220273E-03 1.4778027220273E-03
-5.8839059140813E-04-3.2662239102706E-20 9.3447725407852E-04 4.2896698516357E-03
-4.2896698516357E-03-1.7516507584000E-03-9.7236150153889E-20
```

While some of the above is self-explanatory, the following notes should help with making `stowfn.data` files from external codes:

- The molecular orbital block lists c_{jk} with molecular orbital index j in the outer loop and atomic orbital index k in the inner loop. In Fortran notation, this is
  `((coeff(iao,imo), iao=1,nao), imo=1,nmo)`
- Atomic orbitals `1..nao` are sorted, starting from the outer loop and moving in, by centre, then by shell, then by quantum number m (see below), i.e.,
  `iao` ≡ `(mval_in_shell,ishell_in_centre,icentre)`.
- Shells are groups of atomic orbitals sharing the same ζ, n, and l values (specified in the `Exponent ...`, `Order...`, and `Code ...` blocks in the file).
- m values are given in the following order:
  - $l = 0$: $m = 0$
  - $l = 1$: $m = 1, -1, 0$
  - $l = 2$: $m = -2, -1, 1, 0, 2$
  - $l = 3$: $m = 0, 1, -1, 2, -2, 3, -3$
  - $l = 4$: $m = 0, 1, -1, 2, -2, 3, -3, 4, -4$
  - orbitals with $l > 4$ are not currently supported.

  See, e.g., `https://en.wikipedia.org/wiki/Table_of_spherical_harmonics#Real_spherical_harmonics` for the full form of Y_{lm}.

7.11 External-potential file: `expot.data`

This file contains a representation of an external potential and of the orbitals associated with it. The potential is built up as a sum over one-dimensional sets, each of which is a potential of a stated representation (see the list below) varying as a function of **r** either isotropically or in a stated direction. Each individual potential set may be finite or periodic (with certain restrictions depending on whether the system itself is finite or one-, two-, or three-dimensionally periodic). Periodic external potentials must be commensurate with the underlying lattice if the system is periodic along axes not orthogonal to the potential direction.

The specification of the file is as follows (comments in square brackets):

```
START HEADER
User may put any comments here. CASINO will ignore this section.
END HEADER

START VERSION
 1
END VERSION

START EXPOT
Title
 title
Number of sets
 1
START SET 1
 Particle types affected
  All
 Periodicity [APERIODIC/PERIODIC]
```

```
  APERIODIC
 Type of representation
  SQUARE
 Direction [ISOTROPIC/A1/A2/A3/X/Y/Z/CUSTOM]
  ISOTROPIC
  [If CUSTOM then add direction vector after keyword e.g. CUSTOM 1.0 1.0 0.0]
 Number of such potentials to add
  1
 Origin(s) for each potential (au)
  0.0 0.0 0.0
 Potential
  ... insert potential specification of appropriate type (see below)
END SET 1
 ... insert as many sets as required, consistently with 'Number of sets' above
END EXPOT

START EXPOT_WFN
Title
 title
Number of sets
 1
START SET 1
 Periodicity [APERIODIC/PERIODIC]
  PERIODIC
 Type of orbitals
  FOURIER
 Direction [ISOTROPIC/A1/A2/A3/X/Y/Z/CUSTOM]
  Z
  [If CUSTOM then add direction vector after keyword e.g. CUSTOM 1.0 1.0 0.0]
 Origin of orbitals
  0.d0 0.d0 0.d0
 Orbitals
  ... insert orbital specification of appropriate type (see below)
END SET 1
 ... insert as many sets as required, consistently with 'Number of sets' above
END EXPOT_WFN
```

Notes:

1. The **Type of representation** may take the following values for APERIODIC potentials (the form of the corresponding **Potential** section is also given):

   'SQUARE' Square well/barrier potentials centred at specified points. 'Height' can be set to `INF` or `-INF` for generating infinite barriers or wells, respectively.

   ```
   Potential
    Width (au)
     1.d0
    Height (au)
     -2.d0
   ```

   'LINEAR' Linear potential corresponding to constant force field F $(= -dV/dx)$ passing through 0 at specified point. Note that (i) that the charge of particles is not taken into account and (ii) the potential energy of the nuclei is not calculated. Use the 'ELECTRIC_FIELD' representation if this is desired.

   ```
   Potential
    Gradient (au)
     1.d0
   ```

   'ANALYTIC' Analytic aperiodic functions (for example, Gaussians, harmonic wells) centred at specified points. Type of function to be specified in the **Potential** section:

   ```
   Potential
    Function type and defining parameters [(choose one; extra types definable)]
     GAUSSIAN 1.d0 2.d0 0.d0  ! c,a,b in c*exp(-a*r^2)+b
     HARMONIC 2.d0 0.d0       ! a,b in ar^2+b
     SLAB 20.0 2.0            ! Slab width, 2D r_s param.
   ```

'GAUSSIAN' Aperiodic Gaussian expansion.

```
Potential
 Number of Gaussians
  2
 Coefficients and exponents
  1.d0 6.d0
  1.d0 2.d0
```

'NUMERICAL' Numerical representation on a grid.

```
Potential
 Number of grid points
  2000
 Grid point ; function value
  0.01 2.345
  0.02 2.456
  ...
  2.d0 0.000
```

'BLIP' Blip (B-spline) representation. The blip grid spans the region defined by the cell vectors, as is the case for blip orbitals.

```
Potential
 Blip grid
  100 100 100
 Blip coefficients avc_ix,iy,iz
  1.0      ! coeff_1,1,1
  2.0      ! coeff_1,1,2
  ...
  100.0    ! coeff_100,100,100
```

'ELECTRIC_FIELD' Linear potential corresponding to constant electric field E. Unlike 'LINEAR' this representation takes account of particle charge and calculates the electrostatic energy of the nuclei. For charged complexes in the absence of fixed nuclei, the electrostatic potential of the total charge at the centre of mass of the system is subtracted.

```
Potential
 Electric Field (au)
  1.d0
```

2. The **Type of representation** may take the following values for PERIODIC potentials (the form of the corresponding **Potential** section is also given):

'CONSTANT' Constant potential.

```
Potential
 Constant
  0.d0
```

'SQUARE_PERIODIC' Periodically repeated square well/barrier potential ('square wave'). 'Height' can be set to `INF` or `-INF` for generating infinite barriers or wells, respectively.

```
Potential
 Repeat distance (au)
  3.d0
 Width (au)
  1.d0
 Height (au)
  -2.d0
```

'SAWTOOTH' Periodically repeated linear potential corresponding to electric field E $(= -dV/dx)$. ('Sawtooth wave.')

```
Potential
 Repeat distance (au)
  3.d0
 Electric field (au)
  1.d0
```

'COSINE' Cosine wave of given wavelength and amplitude.

```
Potential
 Amplitude
  1.d0
 Wavelength (au)
  3.d0
```

'ANALYTIC_PERIODIC' Simple analytic aperiodic function (for example, Gaussians, harmonic) periodically repeated. Type of function to be specified in the **Potential** section:

```
Potential
 Repeat distance (au)
  3.d0
 Function type and defining parameters [(choose one; extra types definable)]
  GAUSSIAN 1.d0 2.d0 0.d0  ! c,a,b in c*exp(-a*r^2)+b
  SLAB 20.0 2.0            ! Slab width, 2D r_s param.
```

'FOURIER' 1D Fourier series $a_0/2 + \sum_{i=1}^{n} \left[a_n \cos\left(2\pi nx/L\right) + b_n \sin\left(2\pi nx/L\right)\right]$

```
Potential
 Period L (au)
  3.d0
 Symmetry [ODD/EVEN/NONE]
  NONE
 Number of terms n (excluding a0)
  2
 Fourier coeffs [a_0 then a_n, b_n pairs; omit all a or b if symm ODD/EVEN]
  1.d0
  1.d0 3.d0
  2.d0 4.d0
```

'GAUSSIAN_PERIODIC' Periodic Gaussian expansion.

```
Potential
 Repeat distance (au)
  3.d0
 Number of Gaussians
  2
 Coefficients and exponents
  1.d0 6.d0
  1.d0 2.d0
```

'NUMERICAL_PERIODIC' Periodic numerical representation on a grid.

```
Potential
 Repeat distance
  3.d0 au
 Number of grid points
  2000
 Grid point ; function value
  0.01 2.345
  0.02 2.456
  ...
  2.d0 0.000
```

'BLIP_PERIODIC' Periodic blip (B-spline) representation. The blip grid spans the primitive cell.

```
Potential
 Blip grid
  100 100 100
 Blip coefficients avc_ix,iy,iz
  1.0      ! coeff_1,1,1
  2.0      ! coeff_1,1,2
  ...
  100.0    ! coeff_100,100,100
```

3. The **Type of orbitals** may take the following value (more to be added on request/need):

'FOURIER' One-dimensional Fourier series in the z-direction, plane waves in the XY plane.

```
Period L (au)
 20.0
Symmetry (even/odd/none)
 NONE
Number of terms n (excluding a0)
 15
Number of bands
 2
START BAND 1
 Occupation
  13
 Fourier coefficients (a0 ; a_i,b_i {i=1,n}. Omit a/b if symm ODD/EVEN)
  0.08197602811796                    [a0]
  0.00000000000000   0.00000000011629  [a1 b1]
 -0.07488206414295  -0.00000000158436  [a2 b2]
  ... [13 more rows, according to the value of n=15 given above]
END BAND 1
START BAND 2
 Occupation
  11
 Fourier coefficients (a0 ; a_i,b_i {i=1,n}. Omit a/b if symm ODD/EVEN)
  ...
END BAND 2
```

4. The **file version** is an integer, which is always increased if the specification for this file changes.

5. Where more than one set is given, the potentials defined in the different sets are added to give the final potential at a particular point.

6. The **Direction** parameter gives the direction along which a particular potential varies. This may be along one of the three lattice vectors (periodic systems), along the x, y or z axes, or along a custom direction given in input. If the **Direction** parameter is 'ISOTROPIC' then the potential varies radially as a function of distance from the given point.

7. In the case of the Fourier expansion, complex coefficients need to satisfy $c_{\mathbf{G}} = c^*_{-\mathbf{G}}$ if the potential is to be real. This will be checked for. With pure real coefficients the option exists of omitting the imaginary part of the Fourier coefficients section in order to save disk space.

8. For periodic types the external potential will be checked for being commensurate with the underlying lattice.

9. Clearly, the total number of possible forms of external potential is infinite, and the associated geometries can be arbitrarily complex. It is therefore envisaged that new representations and, if necessary, new ways of describing the geometry will be added to the `expot.data` file on a case-by-case basis. At some stage, the ability to do full 2D and 3D Fourier expansions of the external potential will be added.

10. The orbital bands will be filled in order, e.g., the second band will be started once the first one is full. This is in contrast with an energy-based filling method which could also be implemented (on request). All of the orbitals in a band have the z dependency given by the Fourier series, while a different xy plane-wave will multiply each of them. Be sure to have odd occupation numbers to keep translational symmetry—CASINO will stop otherwise—and to occupy your bands with the right numbers to ensure isotropy as well, if required—CASINO will perform no checks in this sense.

11. Fourier-expanded orbitals are pretty inefficient if what you want to represent is a set of orbitals localized in the z direction. Numerical orbitals would be a neat alternative, and are easy to implement, but as none of us is interested in this so far, this is unavailable. Contact us if you would like to use such a thing.

12. Used with care, the plane-wave terms p and q in the Jastrow factor, together with the u term, can often provide variational freedom that reflects the geometry of the external potential in periodic systems. If you believe that your choice of external potential necessitates the development of new types of term in the Jastrow factor then please discuss with the developers.

13. Note that the external potential is only applied to the quantum particles that are simulated; constant potential energy contributions arising from, e.g., interactions with fixed nuclei are not calculated by CASINO and must be added by hand (except when the 'ELECTRIC_FIELD' potential is used; in this case the ionic energy contributions are included).
14. Magnetic fields are also specified in the `expot.data` file. See Sec. 39.

7.12 Raw QMC data files: `vmc.hist` and `dmc.hist`

The raw VMC and DMC energy data are stored in files called `vmc.hist` and `dmc.hist`, respectively. Although the files have different names, they have the same format. The `.hist` files contain the local energy and its components as calculated during the QMC simulation; by averaging these, one obtains estimates of the total energy, etc. Note that the raw data in the `.hist` files are the averages of the local energies across the MPI processes of a parallel machine. Furthermore, the VMC local energies are averaged over **vmc_ave_period** evaluations before being written out, while the DMC local energies are averaged over the current population of configurations. One cannot, therefore, calculate the variance of the energy by computing the variance of the total-energy data in `vmc.hist` and `dmc.hist`. The energy data can be analysed using the `reblock` utility (or `quickblock` if the file is very large). The data in `vmc.hist` and `dmc.hist` can be extracted for plotting using the `plot_hist` utility.

An example of a `vmc.hist` file is given below.

```
# Title
# My CASINO vmc.hist file.
# File version number
# 1
# QMC method (VMC, DMC, PIMC, AFMC or RMC)
# VMC
# Electron-electron interaction type (interaction keyword)
# ewald
# Constant (ion-ion) energy
# 2.34
# Number of electrons (and other particles) in simulation
# 48
# Number of atoms per primitive cell
# 12
# Number of primitive cells
# 1
# Basis type (from atom_basis_type keyword - formerly btype)
# gaussian
# Periodic (0=NO; 1=YES)
# 0
# Number of data columns (excluding iteration number)
# 6
# Data items (For DMC, must have WEIGHT,POP,ETOT_PROP,EREF,EBEST as 1st 5)
# ETOT
# KEI
# TI
# VCPPEE
# VCPPEI
# VCPPE
# Raw QMC data
1  1.2  2.3  3.6  4.7  5.89 6.3
2  1.3  2.5  3.8  4.8  5.5  6.43
3  1.2  2.3  3.4  4.3  5.67 6.4
4  1.23 2.3  3.4  4.5  5.7  6.7
```

Notes:

- It is not envisaged that ordinary users will ever need to look directly at the contents of this file.
- The header lines of the file start with '#' so that it is easy to plot the raw data using XMGRACE. This is used by the `graphdmc` utility, for example. Note that `graphdmc` requires the first five columns of DMC to be as indicated in the comment line in the example above.

- The file version number is an integer which is increased each time the format of the `.hist` files changes.
- The total energy in `vmc.hist` or `dmc.hist` uses the Ewald or MPC interaction energy, as determined by the value of **interaction**.
- The first column is always the iteration number. This is not included in the number of data columns, and no label for the iteration number needs to be supplied.
- All energies are quoted in a.u. per primitive cell (per electron for electron phases, per particle for electron–hole phases). Information about the different components of the energy and how they are calculated may be found in Secs. 19–19.4.4.
- The following column labels may be used:

  ETOT Total local energy;

  KEI K (Used to evaluate the local kinetic energy: see Sec. 19.1);

  TI T (Used to evaluate the local kinetic energy: see Sec. 19.1);

  EWALD Electron–electron interaction energy ($1/r$ or Ewald);

  LOCAL Local electron–ion interaction energy (plus external potential energy);

  NONLOCAL Nonlocal electron–ion interaction energy.

  SHORT Short-range part of MPC;

  LONG Long-range part of MPC;

  CPPEI Electron–ion core-polarization potential (CPP) term;

  CPPE Electron part of CPP term;

  CPPEE Electron–electron part of CPP;

  MASSPOL Mass-polarization term ('relativistic');

  MASSVEL Mass-velocity term (relativistic);

  DARWINEN electron–nucleus Darwin term (relativistic);

  DARWINEE Electron–electron Darwin term (relativistic);

  RETARD Retardation term (relativistic).

  WEIGHT (DMC only) Total weight of all configurations;

  NCONF (DMC only) Number of configurations at this time step;

  EREF (DMC only) Reference energy;

  EBEST (DMC only) Best estimate of the ground-state energy (mixed estimate);

  ACC (DMC only) Acceptance ratio at this time step;

  TEFF (DMC only) Effective time step;

  DIPOLE1–3 x-, y- and z-components of the dipole moment.

  DIPOLESQ Squared magnitude of the dipole moment.

  FUTURE0–10 (DMC only) Future-walking data.

- The end of equilibration and the start of statistics accumulation in DMC is indicated by the line '`#### START STATS`'. If there is more than one occurrence of this line in the file then the last occurrence is taken to mark the boundary (though recent—10/2011—modifications to the code have sought to reduce the likelihood of this happening, since it can be confusing).

7.13 Expectation-value file: `expval.data`

The `expval.data` file may contain a large variety of data sets. This specification describes the format of the density, spin density, spin-density matrix, reciprocal-space pair-correlation function, spherically averaged pair-correlation function, structure factor, spherically averaged structure factor, one-electron density matrix, two-electron density matrix, momentum density and ionic population sets. When new expectation values are implemented in CASINO, appropriate data sets can be added to this file. Each set is optional. The overall format is the same as that of `correlation.data`, as described in Appendix

D. The `expval.data` file also acts as an input file. If a data set is already present and CASINO is asked to accumulate data for that particular set then the newly accumulated data will be added to the existing data.

The format of the `expval.data` file is as follows. (Note that the text in square brackets below should not actually appear in the file.)

```
START HEADER
 User may put any comments in here. CASINO will ignore this section
 and is required to copy it verbatim if this file is written to.
END HEADER

START EXPVAL
 Title
   Some system
 File version
   1
 Number of particle types (e.g. 2=electrons,4=electrons+holes)
   4
 Number of each type of particle
   nele(1) nele(2) nele(3) nele(4)
 Dimensionality
   3
 Periodicity
   3
 Primitive translation vectors (au)
  -1.2345  1.2345  1.2345
   1.2345 -1.2345  1.2345
   1.2345  1.2345 -1.2345
 Supercell matrix
   S(1,1) S(2,2) S(3,3) S(1,2) S(1,3) S(2,1) S(2,3) S(3,1) S(3,2)
 Volume of simulation cell (area/length for 2D/1D)
   volume
 Radius of sphere inscribed in Wigner-Seitz cell of simulation cell
   1.2345678
 Number of available G-vector sets
   1

 START GVECTOR SET 1
  Energy cutoff used to generate set
    e_cutoff
  Number of G-vectors in set
    ng
  Primitive/supercell reciprocal lattice vectors (au)
    g1(1) g1(2) g1(3)
    g2(1) g2(2) g2(3)
    g3(1) g3(2) g3(3)
  G-vector components (Gx, Gy, Gz) in atomic units
    gvec(1,1)   gvec(2,1)   gvec(3,1)
    ...
    gvec(1,ng)   gvec(2,ng)   gvec(3,ng)
 END GVECTOR SET 1

 START DENSITY
  Accumulation carried out using
    VMC/DMC
  Use G-vector set
    1
  Number of sets
    2
  START SET 1
   Particle type (1=electron,2=hole)
     1
   Total weight
     expval_den_weight_total
   Complex charge-density coefficients (real part, imaginary part)
```

```
    Re{expval_den_total(1)}   Im{expval_den_total(1)}
    ...
    Re{expval_den_total(ng)}   Im{expval_den_total(ng)}
 END SET 1
 START SET 2
  Particle type (1=electron,2=hole)
    2
  Total weight
    expval_den_weight_total
  Complex charge-density coefficients (real part, imaginary part)
    Re{expval_den_total(1)}   Im{expval_den_total(1)}
    ...
    Re{expval_den_total(ng)}   Im{expval_den_total(ng)}
 END SET 2
END DENSITY

START SPIN DENSITY
 Accumulation carried out using
   VMC/DMC
 Use G-vector set
   1
 Number of sets
   4
 START SET 1
  Particle type (1=up elec,2=dn elec,3=up hole, 4=dn hole)
    1
  Total weight
    expval_sden_weight_total
  Complex spin density coefficients (real part, imaginary part)
    Re{expval_sden_total(1)}   Im{expval_sden_total(1)}
    ...
    Re{expval_sden_total(ng)}   Im{expval_sden_total(ng)}
 END SET 1
 START SET 2
 ...
 ...
 END SET 4
END SPIN DENSITY

START SPIN-DENSITY MATRIX
 Accumulation carried out using
   VMC/DMC
 Use G-vector set
   1
 Number of sets
   2
 START SET 1
  Particle type (1-4)
    1
  Total weight
    expval_sdenmat_weight_total
  Component
    1 1
  Complex coefficients (real part, imaginary part)
    Re{expval_sdenmat_total(1)}      Im{expval_sdenmat_total(1)}
    ...
    Re{expval_sdenmat_total(ng)}   Im{expval_sdenmat_total(ng)}
  Component
    1 2
  Complex coefficients (real part, imaginary part)
    ...
  Component
    2 1
  Complex coefficients (real part, imaginary part)
    ...
  Component
```

```
       2 2
     Complex coefficients (real part, imaginary part)
       ...
    END SET 1
    START SET 2
     Particle type
       3
     ...
    END SET 2
   END SPIN-DENSITY MATRIX

   START RECIPROCAL-SPACE PCF
    Accumulation carried out using
      VMC/DMC
    Fixed particle type (1-4)
      1
    Fixed particle coordinate (in atomic units)
      Rx   Ry   Rz
    Use G-vector set
      2
    ...
    [format of the rest identical to spin density]
    ...
   END RECIPROCAL-SPACE PCF

[example below is for fixed particle mode in inhomogeneous system]
   START SPHERICAL PCF
    Accumulation carried out using
      VMC/DMC
    Number of bins
      nbin
    Cutoff radius (in atomic units)
      rcutoff [usually the wigner_seitz radius]
    Accumulation mode (1=fixed particle, 2=homogeneous system)
      1
    Fixed particle type (1-4)
      1
    Fixed particle position
      Rx   Ry   Rz
    Number of sets
      2
    START SET 1
     Types of particles in pair (1-4, 1-4)
       1 1
     Total weight
       upbin_numall
     Bin contents
       matuptotbin(1)
       ...
       matuptotbin(nbin)
    END SET 1
    START SET 2
     Types of particles in pair (1-4, 1-4)
       1 2
     Total weight
       dnbin_numall
     Bin contents
       matdntotbin(1)
       ...
       matdntotbin(nbin)
    END SET 2
   END SPHERICAL PCF

[this example is for homogeneous system]
   START SPHERICAL PCF
    Accumulation carried out using
      VMC/DMC
    Number of bins
      nbin
    Cutoff radius (in atomic units)
      rcutoff [usually the wigner_seitz radius]
    Accumulation mode (1=fixed particle, 2=homogeneous system)
      2
    Number of sets
      3
    START SET 1
     Types of particles in pair
       1 1
     Total weight
       no. of accum steps
     Bin contents
       corrdata(1,1,1)
       ...
       corrdata(nbin,1,1)
    END SET 1
    START SET 2
     Types of particles in pair
       1 2
       ...
    END SET 2
    START SET 3
     Types of particles in pair
       2 2
       ...
    END SET 3
   END SPHERICAL PCF

   START STRUCTURE FACTOR
    Accumulation carried out using
      VMC/DMC
    Use G vector set
      1
    Number of sets for rho_a(G)*rho_b(-G)
      3
    START SET 1
     Types of particles in pair
      1 1
     Total weight
      1000.0
     rho_a(G)*rho_b(-G)
      16.0
      4.12345
      ...
      1.0123
    END SET 1
    START SET 2
     Types of particles in pair
       1 2
       ...
    END SET 2
    START SET 3
     Types of particles in pair
       2 2
       ...
    END SET 3
    Number of sets for spin density part
      2
    START SET 1
     Particle type
      1
     Total weight
      1000.0
     Complex spin density coefficients (real part, imaginary part)
      4.0 0.0
      0.12345 0.0
      ...
      1.0 0.0
    END SET 1
    START SET 2
     Particle type
       2
       ...
    END SET 2
     Total weight
       expval_sden_weight_total
     Complex spin density coefficients (real part, imaginary part)
       Re{expval_sden_total(1)}   Im{expval_sden_total(1)}

   END STRUCTURE FACTOR

   START SPHERICAL STRUCTURE FACTOR
    Accumulation carried out using
      VMC/DMC
    Radial k point grid : (k1, k2, nk)
      0.d0 10.d0 11
    Number of sets
      1
    START SET 1
     Particle type
      1
     Total weight
      expval_sf_sph_weight_total
     k, SF(k)
      0.d0 0.0
      1.d0 0.12345
      ...
      10.d0 1.0
    END SET 1
   END SPHERICAL STRUCTURE FACTOR

   START ONE-PARTICLE DENSITY MATRIX
    Accumulation carried out using
     VMC
    Number of sets
     1
    Number of bins
     100
    Number of random points to sample
     20
    START SET 1
     Number of particle types in set
      2
     Particle types
      1 2
     r,Weight(r),One-Particle-DM(r)
      (...)
    END SET 1
   END ONE-PARTICLE DENSITY MATRIX

   START TWO-PARTICLE DENSITY MATRIX
    Accumulation carried out using
     VMC
    Number of sets
     1
    Number of bins
     100
    Number of random points to sample
     20
    START SET 1
     Number of particle-pair types in set
      2
     Particle-pair types
      1 4  2 3
     r,Weight(r),Two-Particle-DM(r)
      (...)
    END SET 1
   END TWO-PARTICLE DENSITY MATRIX

   START MOMENTUM DENSITY
   Accumulation carried out using
    VMC
   Number of sets
    1
   Number of random points to sample
    2
   START SET 1
   Number of particle types in set
    2
   Particle types
    1 2
   Weight for this set
    1000.000000
   |k|/k_F,MOM_DEN(k)**2,MOM_DEN(k)
    (...)
   END SET 1
   END MOMENTUM DENSITY

   START FINITE DENSITY
    Not yet documented.
   END FINITE DENSITY

   START MOLECULAR DENSITY
   Accumulation carried out using
    VMC
   Grid size
    100 100 1
   Coordinates of A
      3.0000000000000000   3.0000000000000000  2.9999999999999999E-002
   Coordinates of B
     -3.0000000000000000  -3.0000000000000000 -2.9999999999999999E-002
   Nstep, Total weight, Total weight^2
      1000000.0000000000  1000000.0000000000  1000000.0000000000
   START SET 1
   n_i,(n_i)**2
    (...)
   END SET 1
   START SET 2
   n_i,(n_i)**2
    (...)
   END SET 2
   END MOLECULAR DENSITY

   START POPULATION
   Accumulation carried out using
    VMC
   Number of sets
    1
   Number of ions
    2
   START SET 1
   Number of particle types in set
    2
   Particle types
    1 2
   Weight for this set
    1000.000000
   ion, population^2, population
     (...)
   END SET 1
   END POPULATION

   START LOCALIZATION TENSOR
    Not yet documented.
   END LOCALIZATION TENSOR

  END EXPVAL
```

1. General notes:

   (a) Information about the expectation values can be found in Sec. 35.

   (b) The 'file version' is an integer, which is always increased if the specification for this file changes.

   (c) The data in this file should always be normalized. For example, they should be divided by the total weight, which is usually the total number of accumulation steps. However, the total weight must always be included alongside the data so that further accumulation steps can be carried out. The correct way to perform additional accumulation is to set

   $$\text{new tot. weight} = \text{old tot. weight} + \text{new weight} \qquad (36)$$

   $$\text{new tot. data} = \frac{\text{old tot. weight} \times \text{old tot. data} + \text{new weight} \times \text{new data}}{\text{new tot. weight}}. \qquad (37)$$

   The 'new total weight' and 'new total data' should be written out to `expval.data`.

   (d) Sets of reciprocal space vectors ('G-vectors') are specified at the start of the file and are then referenced by the different sets in the file, because expectation values for different operators are often accumulated using the same set of G-vectors.

   (e) Each of the subsequent data sets is optional. If a data set is present when this file is read as input, but CASINO is not accumulating further data for this set, then the set must be copied verbatim on output.

   (f) It is permitted to have, for example, the VMC-accumulated charge density present in this file, to be used for the MPC interaction in a DMC calculation whilst accumulating, for example, the pair-correlation function.

2. Notes on the 'density' data set:

   (a) There should be one set of data present for each type of particle (electron, hole).

   (b) The normalization of the data for a given particle type is such that the $\mathbf{G} = \mathbf{0}$ component is equal to the total number of particles of that type in the primitive cell. This is naturally the result if the data is accumulated during the simulation, then divided by the total weight (that is, the number of accumulation steps) then divided by the number of primitive cells in the simulation cell.

   (c) The QMC charge density may be used in the MPC interaction, if desired.

3. Notes on the 'spin density' data set:

   (a) The format of this section is the same as that of the 'density' section, except that there are two sets of data for each particle type, to allow for up and down spins.

4. Notes on the 'spin-density matrix' data set:

   (a) These data can be obtained when using the noncollinear-spin mode of CASINO. There are separate 2×2 spin-density matrices for each particle type.

   (b) Due to the way in which noncollinear spins are implemented in CASINO, it would be possible in principle to have a separate set of data for up-like electrons, down-like electrons, up-like holes and so on. In practice this does not make sense, and only one type of electron and one type of hole should be used.

(c) The PLOT_EXPVAL utility will give you the option to plot either the spin-density matrix itself or the magnetization density, which is derived from the spin-density matrix.

5. Notes on the 'molecular density' data set:

   (a) There should be one set of data present for each type of particle (electron, hole).

   (b) The PLOT_EXPVAL utility allows the user to perform a cylindrical or spherical average of the molecular density.

6. Notes on the 'reciprocal-space pair-correlation function' data set:

   (a) This data set contains the Fourier coefficients of the pair-correlation function, accumulated with the position of one particle being fixed.

   (b) Apart from information on the type and location of the fixed particle, the format of this data set is identical to the 'spin density' set.

7. Notes on the 'spherical pair-correlation function' data set:

   (a) This set contains a real-space representation of the pair-correlation function, obtained by binning particle separations.

   (b) There are two slightly different ways in which spherical PCFs may be accumulated. For homogeneous and finite systems, we accumulate the spherical PCF based on the relative distances of all pairs of particles. For inhomogeneous and periodic systems, we must accumulate the spherical PCF as the distance of all other particles from one fixed particle.

   (c) The format of this set is flexible enough to allow for both methods. Two examples are given above to show the two possible scenarios.

   (d) When the data is for fixed-particle accumulation, the first of the two particle types appearing on the 'Types of particles in pair' line must be that of the fixed particle.

8. Notes on the 'population' data set:

   (a) The ionic populations are determined by partitioning space into Voronoi polyhedra about the ions and binning the number of times each electron is found in each ion's polyhedron. The results quoted are the mean numbers of electrons in each ion's Voronoi polyhedron.

   (b) The population (number of electrons) for each ion is reported in `expval.data`. There is no need to run PLOT_EXPVAL.

   (c) By using two sets (one for spin-up and one for spin-down electrons), spin-resolved populations can be accumulated.

   (d) Please see the comments in Sec. 35.9 about the interpretation and use of the population data.

8 Generating CASINO trial wave functions with other programs

The following third-party codes have support for producing CASINO trial wave function `xwfn.data` files. The exact degree of support will depend on time; in particular the interfaces may stop working if the developers release a new version of their code without bothering to check that CASINO support still works. If you find this is the case, please let us know. All voluntary contributions to maintaining these interfaces is gratefully accepted, since doing so is very boring indeed. Our personal support for the codes can be greatly enhanced if we have someone working in Cambridge who is a dedicated user of the code in question. When they leave, our group knowledge may become non-existent (e.g., currently, none of us know anything about the Gaussian code).

Example input files for some of the codes below can be found in the distribution directory examples/wfn_generation. Instructions are in accompanying `README`s.

A summary of the current status of the various interfaces—which supplements and may be more up-to-date than the information given here—can be found on the CASINO web site here:

`https://vallico.net/casinoqmc/interfaces/`

and a page of advice for developers who wish to create their own interfaces is here:

`https://vallico.net/casinoqmc/interface_development/`

Developers should note that for Gaussian basis set codes, there is a special test suite in the directory `examples/generic/gauss_dfg`. The `README` file there should be consulted for conventions on solid harmonics/normalization etc., and calculations should be run using the provided set of input files to test the treatment of d, f and g functions.

8.1 ABINIT

Website: `https://abinit.github.io/abinit_web/`.

To produce a wave function suitable for use as a CASINO trial wave function, certain ABINIT parameters must be set correctly. To tell ABINIT to write the wave function in CASINO format, set **prtwf** to 2 in the ABINIT input file. CASINO (and QMC methods generally) can only take advantage of time-reversal symmetry, and not the full set of symmetries of the crystal structure. Therefore, ABINIT must be instructed to generate **k** points not just in the irreducible Brillouin zone, but in a full half of the Brillouin zone (using time-reversal symmetry to generate the other half). Additionally, unless instructed otherwise, ABINIT avoids the need for internal storage of many of the coefficients of its wave functions for **k** points that have the property $2\mathbf{k} = \mathbf{G}_{\rm latt}$, where $\mathbf{G}_{\rm latt}$ is a reciprocal lattice vector, by making use of the property that $c_{\mathbf{k}}(\mathbf{G}) = c^*_{\mathbf{k}}(-\mathbf{G} - \mathbf{G}_{\rm latt})$. ABINIT must be instructed not to do this in order to output the full set of coefficients for use in CASINO. See the ABINIT theoretical background documents `ABINIT/Infos/Theory/geometry.pdf` and `ABINIT/Infos/Theory/1WF.pdf` for more information.

The first of these requirements is met by setting the ABINIT input variable **kptopt** to 2 (see `ABINIT/Infos/varbas.html#kptopt`) and the second by setting **istwfk** to 1 for all the **k** points (see `ABINIT/Infos/vardev.html#istwfk`). Since CASINO is typically run with relatively small numbers of **k** points, this is easily done by defining an array of '1's in the input file.

For example, for the eight **k** points generated with 'ngkpt 2 2 2', we add the following lines to the ABINIT input file:

```
# Turn off special storage mode for time-reversal k-points
istwfk 1 1 1 1 1 1 1 1
# Use only time reversal symmetry, not full set of symmetries.
kptopt 2
# Write a CASINO pwfn.data wave function file
prtwf 2
```

Other useful input variables of relevance to the plane waves ABINIT will produce include **ecut**, **nshiftk**, **shiftk**, **nband**, **occopt**, **occ**, **spinat** and **nsppol** (see relevant input variable documents in `ABINIT/Infos/`). If ABINIT is run in multiple dataset mode, the different wave functions for the various datasets are exported as `pwfn1.data`, `pwfn2.data`, ..., `pwfnn.data` where the numbers are the contents of the contents of the input array **jdtset** (defaults to 1,2,...,**ndtset**).

The exporter does not currently work on multiple processors if **k**-point parallelism is chosen. ABINIT does not store the full wave function on each processor but rather splits the **k** points between the processors, so no one processor could write out the whole file. The sort of plane wave DFT calculations usually required to generate QMC trial wave functions execute very rapidly anyway and will generally not require a parallel machine. The `outqmc` routine currently bails out with an error if this combination of modes is selected.

If there is any doubt about the output of this routine, the first place to look is the log file produced by ABINIT: if there are any warnings about incorrectly normalized orbitals or noninteger occupation numbers there is probably something set wrong in the input file.

There exists a utility called ABINIT_SC to convert ABINIT pseudopotentials, as long as they are in grid-based formats, i.e., **pspcod**=1 or **pspcod**=6, to CASINO `x_pp.data` format. The utility does not currently work for the non-grid-based or outdated pseudopotential formats (2,3,4 and 5).

8.2 ADF

See `https://www.scm.com`.

The converter `adf2stowf.py` was developed for version 2008.01 of the ADF code. It takes the output of a molecular ADF calculation and produces a CASINO trial wave function in a Slater-type basis. The cusp condition is implemented as a simple linear constraint, which is enforced by projection in the converter. The effect of this projection may be significant for small basis sets.

In our experience, the ZORA/QZ4P basis set has been found to perform well.

ADF does not support pseudopotentials, but uses the frozen-core approximation to reduce the computational time. Importing frozen-core states is supported in principle. However, frozen-core states are not implemented in CASINO (no QMC algorithm for this is available at present). Therefore, ADF frozen-core states are used as regular orbitals within CASINO.

The ADF converter and the Slater-type orbital implementation in CASINO have not yet been tested heavily, so expect bugs and limitations, which we could try to fix when reported (but note that if you're using the code at all, you almost certainly know more about it than we do, since the author of the converter has left physics behind and the collective experience amongst the main CASINO developers has therefore been reduced to zero.)

8.3 ATSP2K

ATSP2K is a numerical orbital multi-determinant (and Hartree–Fock) atomic package (see `https://nlte.nist.gov/cgi-bin/MCHF/download.pl?d=ATSP2K`). Output from this program may be converted into CASINO's `awfn.data` numerical atomic orbital format by the converter '`extractdet`' (written by John Trail) which is provided as a CASINO utility. At a numerical level the QTSP2K code works with radial orbitals and Configuration State Functions (CSFs) rather than determinants constructed from 3-d orbitals, hence this conversion is non-trivial.

Further information is provided in the file `CASINO/utils/wfn_converters/a2sp2k/Extractdet/README`. Note that the converter is not automatically installed by the CASINO build system (not least because it requires an ATSP2K library to compile); it must therefore be compiled and installed manually.

8.4 CASTEP

Website: `https://www.castep.org`

The modern CASTEP [7] is an entirely new Fortran 90 version of the venerable Cambridge plane-wave program which it is designed to replace. It is freely available to academics worldwide. Mail Chris Pickard (cjp20 @ cam.ac.uk) to discuss CASTEP and how to get hold of a copy.

We assume that the reader is familiar with CASTEP. (It takes about five minutes to learn how to perform basic CASTEP calculations.) To generate a trial wave function using CASTEP:

1. Download the pseudopotentials that you need from `https://vallico.net/casinoqmc/pplib/`. You need to download the 'tabulated' file `pp.data`, which should be renamed as '`x_pp.data`', where `x` is the chemical symbol in lower-case letters, and the 'CASINO `awfn.data`' file for the ground state, which should be renamed as '`x_pp.awfn`'. In your CASTEP `seedname.cell` file allocate the pseudopotential to species 'X' using

   ```
   %block SPECIES_POT
   X x_pp.data
   %endblock SPECIES_POT
   ```

   Note that both `x_pp.data` and `x_pp.awfn` must be present.

2. If you want to relax the geometry using these pseudopotentials, do so at this point. (However, it is generally best not to use CASINO's HF pseudopotentials for DFT geometry optimization.)

3. Specify your list of **k** points explicitly using the **kpoints_list** block in `seedname.cell` (otherwise CASTEP will apply symmetry operations to the **k** points in the grid). If you intend to use a complex wave function then you should supply the complete list of **k** points; otherwise you should only supply one out of each $\pm\mathbf{k}$ pair. For example, the orbitals for a real wave function in a supercell consisting of $2 \times 2 \times 2$ primitive cells could be constructed using the following **k**-point grid:

```
%block kpoints_list
0.25 -0.25 -0.25      0.25
0.25 -0.25  0.25      0.25
0.25  0.25 -0.25      0.25
0.25  0.25  0.25      0.25
%endblock kpoints_list
```

This is a $2 \times 2 \times 2$ **k**-point mesh offset so that Γ is exactly in between the grid points, with only one out of each $\pm\mathbf{k}$ being given, and equal weight being assigned to each **k** point in the DFT calculation. The **symmetry_generate** flag should be given in `seedname.cell`.

4. Run a CASTEP single-point energy calculation (not geometry optimization, etc.) using *castep ⟨seedname⟩*.

5. Then run the CASTEP2CASINO converter, which is part of the CASTEP distribution using *castep2casino ⟨seedname⟩*.[13]

6. Rename `seedname.casino` as `pwfn.data`.

7. The `pwfn.data` can be read by CASINO, but it's usually best to convert to a blip basis by performing a preliminary blip-conversion CASINO calculation by setting **runtype** to 'gen_blip'. (See Sec. 9.)

The AE_PP_MAKER utility allows the user to generate a CASTEP-format `.recpot` file holding a bare Coulomb potential, to fool CASTEP into performing an all-electron calculation. This is useful for accurate calculations featuring light elements such as hydrogen, helium and perhaps lithium. Note that CASINO does not require a pseudopotential file for all-electron calculations, but you will certainly want to set **use_gpcc** to `T` to apply cusp corrections to your all-electron plane-wave or blip orbitals.

8.5 CFOUR

"*CFOUR (Coupled-Cluster techniques for Computational Chemistry) is a program package for performing high-level quantum chemical calculations on atoms and molecules. The major strength of the program suite is its rather sophisticated arsenal of high-level ab initio methods for the calculation of atomic and molecular properties. Virtually all approaches based on Møller-Plesset (MP) perturbation theory and the coupled-cluster approximation (CC) are available; most of these have complementary analytic derivative approaches within the package as well.*"

Website: `https://cfour.uni-mainz.de/cfour/`

Support for this Gaussian basis set quantum chemistry code is provided through the `molden2qmc` utility by Mike Deible and Vladimir Konjkov, which can convert files written in the quasi-standard MOLDEN format into CASINO's `gwfn.data` format.

CFOUR's implementation of MOLDEN is somewhat special... and requires various vectors to be reordered before the standard CASINO `molden2qmc` converter will work. To do this, replace the file `/libr/reorderdf.f` in the CFOUR distribution with the file `utils/wfn_converters/cfour/reorderdf.f` from the CASINO distribution, then recompile CFOUR.

To generate a `gfwn.data` file, sit in a directory containing suitable MOLDEN output, type '`molden2qmc`', then follow the prompts.

See the file `CASINO/examples/generic/gauss_dfg/RESULTS` for the current status of the interface.

8.6 CRYSTAL

[See `~/CASINO/utils/wfn_converters/crystal_9x/`, `/crystal_03/`, `/crystal_06`, `/crystal_09`, `/crystal_14` and `/crystal_17`.]

[13]If you are using an old version of CASTEP then you will need to insert the line 'continuation : ⟨seedname⟩' in the `seedname.param` file before running CASTEP2CASINO.

8.6.1 The CRYSTAL program

CRYSTAL is a commercially available quantum-mechanical package (free for UK academics?) which is able to calculate the electronic structure of both molecules and systems with periodic boundary conditions in 1, 2 or 3 dimensions (polymers, slabs or crystals) using either HF or DFT. The latest version of the program was written by Roberto Dovesi, Vic Saunders, Carla Roetti, Roberto Orlando, Claudio Zicovich-Wilson, F. Pascale, Bartolomeo Civalleri, Klaus Doll, Nic Harrison, Ian Bush, Philippe D'Arco, Miquel Llunell, Mauro Causà, Y. Noel, L. Maschio, A. Erba, M. Rerat and Silvia Casassa.

The official web page can be found at `https:/www.crystal.unito.it`. See also `https://vallico.net/mike_towler/crystal.html` and the various links from there for more information from a CASINO perspective (though this page has not been maintained for many years).

CASINO supposedly supports the official releases CRYSTAL95, CRYSTAL98, CRYSTAL03, CRYSTAL06, CRYSTAL09, CRYSTAL14, CRYSTAL17 (but not 88 or 92).

With the now very obsolete CRYSTAL95 or CRYSTAL98 you must use a separate utility *crysgen98*—included with the CASINO distribution—which transforms the contents of various CRYSTAL temporary Fortran files into a CASINO `gwfn.data` file.

CRYSTAL03 was supposed to contain a facility to write out a CASINO `gwfn.data` file directly. The necessary routines were written by MDT following a specification agreed by the Daresbury authors of CRYSTAL but its incorporation into CRYSTAL was then vetoed by the Torino group. MDT was then asked to change the routines into a separate utility, but necessary information regarding the CRYSTAL source code was not available. The practical upshot of this is that CASINO is only supported in MDT's private version of CRYSTAL03 which he is not allowed to distribute. Best to use CRYSTAL06 or CRYSTAL09 instead.

CRYSTAL06 is able to write out all the information required by external programs but as this is not specific to CASINO a utility *crysgen06* is required to convert the data into CASINO `gwfn.data` format.

CRYSTAL09 is able to write out all the information required by external programs but as this is not specific to CASINO a utility *crysgen09* is required to convert the data into CASINO `gwfn.data` format (the size of the overlap matrix in the API was changed in going from 06 to 09 for some reason, hence the different utility).

CRYSTAL14 is essentially the same as CRYSTAL09 as far as CASINO is concerned, though the conversion utility is mildly changed and renamed as *crysgen14*. Note that the initial release of CRYSTAL14 contained several bugs which meant that the data written by its API was incorrect; if CASINO users downloaded their version of CRYSTAL14 before 30th Jan 2014, they should download it again in order for the interface to work.

CRYSTAL17 is essentially the same as CRYSTAL14 as far as CASINO is concerned, though the conversion utility is mildly changed and renamed as *crysgen14*.

If you want to use pseudopotentials with CRYSTAL you can use the Trail-Needs ones on the CASINO website, which are given in the CRYSTAL format. For some of these a Gaussian basis set in the appropriate format which is optimized with respect to the pseudopotential is included in the 'Further data' column.

8.6.2 Generating `gwfn.data` files with CRYSTAL95/98

These early versions of CRYSTAL are absolutely not designed to communicate with other programs so we will need to jump through some hoops in order to get things to work.

Basically, MDT's *crysgen98* utility is used to convert the output of CRYSTAL95 or CRYSTAL98 into a `gwfn.data` file readable by CASINO. We will assume in what follows that you know how to use CRYSTAL. You need to run the program using the *runcrystal* script, which lives in `~/CASINO/wfn_converters/crystal9x/run_script/`, using the `-qmc` flag as an argument (this should be set up automatically for you during utilities compilation). The publicly available `run` script on MDT's CRYSTAL website does not contain this flag, so you need to use the script included with CASINO instead. You may need to change some environment variable definitions in the *run* and *crysgen98* scripts to get them to work properly on your system.

If you want to use your own copy of CRYSTAL95 or CRYSTAL98 to generate QMC wave functions, you will need to make some minor modifications to the source code (see the accompanying `~/CASINO/wfn_converters/crystal_to_qmc/README_CHANGES` file). If you are working in Cambridge, you can use the programs that have already been so modified by MDT—please ask.

(NOTE: 4/2003—the above changes involve turning off the use of symmetry in k space—for some *metallic* calculations this seems to cause a problem (Gilat net?) unless you also manually turn off all symmetry in the input file with the keyword **symmremo**. This is unlikely to be fixed in the future.)

CRYSTAL will produce three binary files in the scratch directory you define in the run script—namely `fort.12` (basis set, geometry, common variables), `fort.10` (orbital coefficients) and `fort.30` (eigenvalues). These files would normally be deleted at the end of a CRYSTAL run. Including the -qmc flag as an argument to the *run* script means these files will be grabbed and renamed as `silicon.f12`, `silicon.f10` and `silicon.f30` (or whatever). They will be kept in the scratch directory since they can become very large.

Sit in the directory where these three files live and type *crysgen98*. This script will run the *crystaltoqmc* program, which will ask you some questions and then generate the `gwfn.data` file.

crysgen06/09 will ask you for the size of the Monkhorst–Pack **k**-point net in the CRYSTAL calculation, and the desired size of supercell in the QMC calculation. These need not be the same (if not, you are 'plucking' to use the local vernacular), but the former must be divisible by the latter. For example, a 12×12×12 MP net in CRYSTAL will allow you to generate `gwfn.data` for 1×1×1, 2×2×2, 3×3×3, 4×4×4 and 6×6×6 supercell cases. Note that it is desirable to carry out the calculation on a higher density **k**-point grid than you actually need so that the orbital coefficients are calculated accurately.

Note that for polymer and slab calculations the last one and two numbers in the MP net and supercell specifications should be 1 to reflect the fact that the system is not periodic in those dimensions. For example, polymer (12×1×1 $\longrightarrow$ 1×1×1, 2×1×1, 3×1×1, 4×1×1, 6×1×1) slab (12×1×1 $\longrightarrow$ 1×1×1, 2×2×1, 3×3×1, 4×4×1, 6×6×1). For molecular calculations, just imagine you have a 1×1×1 MP net.

So, to summarize, to produce `gwfn.data` from the CRYSTAL input file `dna`:

```
% run -qmc dna
```

The -qmc flag invokes generation of relevant QMC files in temp.

```
% cd /temp/mdt
```

... or whatever the temp directory is called in your CRYSTAL *runscript.*

`% crysgen06` or `% crysgen09`

Then answer the questions.

```
% mv gwfn.data ~ ; rm dna.*
```

Then run CASINO.

8.6.3 Generating `gwfn.data` files with CRYSTAL03

NOTE: THE FOLLOWING INSTRUCTIONS **WILL NOT WORK** WITH THE STANDARD RELEASE OF CRYSTAL03—SEE DISCUSSION ABOVE. THE BEST WAY AROUND THIS NOW IS TO USE CRYSTAL06/09.

The `gwfn.data` file can be generated directly using the CRYSTAL03 *properties* program.

Note that this facility only exists in the official version of CRYSTAL03 in binaries produced after December 2003, and that some early versions of CRYSTAL03 had a broken pseudopotential evaluator (which should now be fixed).

The CRYSTAL *run* script included with CASINO will actually force the production of `gwfn.data` automatically if you invoke it with the `-qmc` flag when running the calculations, so all you need to do is type *run -qmc input_filename*. If the calculation is periodic, the script will ask you how many different supercell sizes N you wish to generate, and then to input N sets of integer triplets indicating the supercell sizes (these must be a subset of the MP shrinking factors in the CRYSTAL input file). For example,

```
% run -qmc h
```

Number of different QMC supercell sizes to calculate? (Maximum 5)

```
% 1
```

Size of cell 1? (e.g., 2 2 2)

% `2 2 2`

Put the script in the background with 'Ctrl-Z' then 'bg'.

If you want to do this by hand, rather than letting the *run* script do it for you, then the relevant part of the CRYSTAL properties input file looks like this (for periodic systems):

```
QMC
2                      ! want to generate 2 supercell sizes i.e.
2 2 2                  ! a 2x2x2 one
3 3 3                  ! and a 3x3x3 one
END
```

For molecules, only the keyword 'QMC' is required with no additional input.

8.6.4 Generating `gwfn.data` files with CRYSTAL06/09/14/17

Note this is only possible with versions of CRYSTAL06 from 1_0_2 onwards, since the 1_0_1 version does not recognize the required **CRYAPI_OUT** keyword. Note also that the initial release of CRYSTAL14 contained a broken **CRYAPI_OUT** function; CASINO users must therefore use a version of CRYSTAL14 produced after 4th March 2014 when this error was fixed.

The entire process is automated by running CRYSTAL06/09/14/17 with our supplied script *runcrystal*. If you choose to do this, then simply typing `runcrystal -qmc input_filename` will run the SCF calculation as normal then invoke all the necessary post-processing to generate the gwfn.data file automatically. The CRYSTAL17 version is run by default; CRYSTAL06 may be selected using the -06 flag, CRYSTAL09 with the -09 flag, and CRYSTAL14 with the -14 flag.

The only complication is in periodic systems where, before executing the CRYSTAL code, the run script will ask you how many different supercell sizes you require. You then need to input that many integer triplets (divisors of the CRYSTAL 'shrinking factors') to specify the size of each of the supercells. A `gwfn.data` file will then be generated for each required cell (appropriately labelled with the integer triplet—rename them before running with CASINO).

Running the `gwfn.data` generation procedure manually

If you choose to run CRYSTAL06/09/14/17 without our `runcrystal` script (say, with the scripts provided by the Torino people) then you will need to carry out the following manual procedure to generate the `gwfn.data` file(s).

Note that `runcrystal` is an old csh script that does not support the CASINO architecture system (like, e.g., `runqmc`, `runpwscf`, `runlouis`). It is hoped that a modernized bash version which does this might appear in the future.

Molecules

After completing a converged DFT/HF run with the CRYSTAL06/09/14/17 *crystal* binary, the *properties* program needs to be run to produce the formatted data files needed by the CASINO *crysgen06/09/14/17* utilities. A minimal input file for the *properties* program in the case of a nonperiodic system is:

```
NEWK
1 0
CRYAPI_OUT
END
```

After successfully running *properties*, the directory should contain the files `GRED.DAT` and `KRED.DAT`. Note that GRED.DAT has a slightly different format from CRYSTAL09 onwards.

Running the CASINO utility `crysgen06`, `crysgen09`, `crysgen14` or `crysgen17` in the same directory reads these files and writes `gwfn.data` for use in CASINO.

1D/2D/3D periodic systems

For periodic systems, the k-point mesh used within CRYSTAL should be an integer multiple of the supercell size that is used in CASINO. Within CRYSTAL06/09/14/17, the mesh is defined using the **SHRINK** keyword followed by 1, 2 or 3 integers depending on the number of periodic dimensions.

Next, the input for the *properties* program needs one additional line after the **NEWK** keyword. To simply output the full k-point mesh that was used with CRYSTAL, use the following:

```
NEWK
0 0
1 0
CRYAPI_OUT
END
```

After running properties successfully, the directory should contain the files `GRED.DAT` and `KRED.DAT`

IMPORTANT: In periodic systems the program *crysgen17* (or *crysgen06/09/14*) now needs to know some parameters related to the size of the supercells required (the converter can output several sizes of supercells in one run). These can be taken from a file called `crysgen.dat` (which is normally automatically produced by the runqmc script)—but if this file is not present *crysgen* will ask you to input this data on the command line.

For reference, a typical `crysgen.dat` indicating that three supercells of sizes 222, 333 and 444 should be generated might look like:

```
QMC
3
2 2 2
3 3 3
4 4 4
END
```

The first number it reads after the QMC line is the number N of different super cell sized is should produce. After this, it takes N triples of integers each specifying one size of a supercell. These supercells have to be integer dividers of the **k**-point mesh that was output by the *properties* program. After all N triples have been given, the individual `gwfn.data` files are produced for use by CASINO.

Differences between CRYSTAL03 and CRYSTAL06/09/14/17 input files

The CRYSTAL03 and CRYSTAL06/09/14/17 codes have minor irritating and mutually incompatible differences in their input formats. The differences that CASINO users are most likely to encounter using old example input files from our distributions are as follows:

- In CRYSTAL06/09/14/17 there are now three sections to the input file rather than four. To update earlier input files, delete the superfluous END statement after the third section.
- The shrinking factors of the Monkhorst and Gilat nets are now given by the SHRINK keyword followed by, e.g., ‘8 16’, rather than the old three numbers at the start of the fourth section.
- The **TOLSCF** keyword—which is common in input files round here—no longer works. Replace this block with the TOLDEE block which contains one number (the usual energy threshold, e.g., 7).
- In CRYSTAL09 the format of user-defined INPUT pseudopotentials changed. If you happen to be using a pseudopotential taken from our online library before February 2012, then it will cease to work, and CRYSTAL09 will stop with an error message ‘RADIAL POWER IN INPUT PSEUDO OUT OF RANGE’. The solution is to add a zero to the line after ‘INPUT’ in each pseudopotential. The CASINO online library and examples were updated to reflect this change in February 2012. See Question D4 of the CASINO FAQ.

Parallel CRYSTAL

The `runcrystal` script supplied with the CASINO distribution supports parallel running of the CRYSTAL program (on machines without batch queues)—simply use the `-np` flag to indicate the desired number of cores. You also need to manually set up the name of the parallel run command at the top of the runcrystal script to be ‘`mpirun -np`’ or whatever it is.

The standard version of CRYSTAL will only run on a single core. To run in parallel you must have produced a `Pcrystal` (replicated data, small Ncore) or `MPPcrystal` (distributed data, large Ncore) binary from the set of pre-compiled object files and Makefiles that the Torino people distribute (*my experience: in the* `build/Xmakes/Linux-ifort-11.1_emt64.inc` *file, set* `MPIBIN` *to be the location* *of your* `mpif90` *compiler (e.g.,* `/opt/openmpi_intel/bin`*), set* `MKLPATH` *to be the location of your MKL libraries (e.g.,* `/opt/intel/mkl/lib/intel64`*) and change the apparently incorrect default value of* `F90` *from '*`ifort`*' to '*`mpif90`*' or it won't pick up* `mpif.h` *etc. Then type 'make' in the* `build` *directory.)*[14].

Note when running parallel CRYSTAL with the `runcrystal` script, there is no need to rename the input file to be '`INPUT`' (the script will do this for you). You do however have to add the '`MPP`' keyword to the bottom of the CRYSTAL input file if you are running the distributed data `MPPcrystal`.

A summer school introduction to parallel CRYSTAL by Ian Bush is available online at: `https://www.crystal.unito.it/mssc2009/lezioni/friday/t25_bush.pdf`.

8.7 DALTON

Website: `https://www.daltonprogram.org/`

Support for this Gaussian basis set quantum chemistry code is provided through the `molden2qmc` utility by Mike Deible and Vladimir Konjkov, which can convert files written in the quasi-standard MOLDEN format into CASINO's `gwfn.data` format.

To generate a `gfwn.data` file, sit in a directory containing suitable MOLDEN output, type '`molden2qmc`', then follow the prompts.

See the file `CASINO/examples/generic/gauss_dfg/RESULTS` for the current status of the interface.

8.8 GAMESS-US

"GAMESS *is a program for ab initio molecular quantum chemistry. Briefly,* GAMESS *can compute SCF wave functions ranging from RHF, ROHF, UHF, GVB, and MCSCF. Correlation corrections to these SCF wave functions include Configuration Interaction, second order perturbation Theory, and Coupled-Cluster approaches, as well as the Density Functional Theory approximation. Excited states can be computed by CI, EOM, or TD-DFT procedures.*"

Website: `https://www.msg.chem.iastate.edu/gamess/`

There are two alternatives for the CASINO interface to the GAMESS-US package:

(1) An old set of perl scripts distributed with the main CASINO distribution, the main one of which is called '`gamess2qmc`' These live, along with a `README` file with instructions and some basic examples, in the directory `CASINO/utils/wfn_converters/gamess`. They were originally written by Alexander Badinski many years ago. Support for f functions was added in 2015 by Kevin Gasperich. The script has considerable limitations, but providing you have some idea of what you're doing and you're not doing anything weird it is not so difficult to make it work.

(2) Albert Defusco of the University of Pittsburgh has implemented native CASINO support into GAMESS independently of the `gamess2qmc` script. This is certainly available in the current standard GAMESS distribution; the facility has been evolving and it might be best to say you should use the most recent version available (I—MDT—have been made aware [July 2015] of a problem occurring when basis functions are automatically discarded because of quasi-linear dependence. . . last I heard a fix was being worked on).

8.9 GAUSSIAN94/98/03/09

[See `~/CASINO/utils/wfn_converters/gaussian9x-03/`.]

GAUSSIAN is an extremely large and widely used commercially available quantum chemistry package. More details are available from `https://gaussian.com/`.

`gaussiantoqmc` is a utility to read the wave function from the output of a GAUSSIAN94/98/03/09 (G94/G98/G03/G09) calculation and output it in a form compatible with CASINO. `gaussiantoqmc` was originally written by Andrew Porter (2000).

If `gaussiantoqmc` does not work for some reason, you may like to ask Katharina Doblhoff-Dier (`k.doblhoff-dier 'at' umail.leidenuniv.nl`), who has written her own converter for this purpose which overcomes some infelicities in the CASINO-supplied version.

[14]Isn't the CASINO `install` script cool?

The `gaussiantoqmc` code *requires* the existence of a formatted checkpoint file (produced by putting FormCheck=(MO,Basis) in the route section of the GAUSSIAN job file for G94/G98, produced automatically by G03 and G09). It expects this file to have a '.Fchk' suffix ('.fchk' for G09). The output file of the GAUSSIAN job is also *required.* It is assumed that this has a '.out' suffix. If the original GAUSSIAN job file is present then it will be appended to the end of the QMC input file (as is the GAUSSIAN output file).

Note that although more recent versions of GAUSSIAN contains a limited facility for treating *periodic* systems, the `gaussiantoqmc` converter does not support this, since it was written before the periodic functionality was introduced. People who wish to do this should use the (superior) CRYSTAL program instead (or volunteer to update the converter).

If you want to use pseudopotentials with GAUSSIAN you can use the Trail-Needs ones on the CASINO website, which are given in the GAUSSIAN format. For some of these a Gaussian basis set in the appropriate format which is optimized with respect to the pseudopotential is included in the 'Further data' column.

Note that there is an associated utility called `egaussian` which extracts the SCF energy (and components) from a GAUSSIAN output file.

8.9.1 How to use GAUSSIANTOQMC

If you have a GAUSSIAN job file called dna (say), and you run it to produce `dna.out` and `Test.FChk`, then you must:

```
% mv Test.FChk dna.Fchk
```

run `gaussiantoqmc` . . . and follow the prompts

```
% mv dna.qmc gwfn.data
```

run CASINO.

The code should automatically detect what sort of GAUSSIAN job it is and give you the opportunity to construct an excited-state wave function if applicable.

It can deal with the following sorts of calculation:

- HF and DFT ground states, open and closed shell.
- CIS excited states, open and closed shell.
- CASSCF states. Getting GAUSSIAN to output these can be problematic for large calculations. It is possible that `gaussiantoqmc` will get confused if you use some combination of IOps other than those described in Sec. 8.9.3.
- Time-dependent HF (TD-HF) or DFT (TD-DFT) excited states.

If the user chooses to output a CIS or TD-DFT wave function, they are given the option of resumming it. The wave function must also be resummed if the user wishes to analyse its composition. As distributed, `gaussiantoqmc` will not do this analysis but if you wish to switch this option on then the flag **analyse_cis** in `cis_data.f90` must be set to `T` and `gaussiantoqmc` recompiled.

With this flag set the code evaluates the percentage contribution of each single excitation to the CIS expansion. Degenerate virtual and occupied orbitals are identified and their contributions summed. The final output takes the form of files called `fromi_j.dat` where i–j indicates a range of degenerate occupied orbitals (if $i = j$ then i is a nondegenerate orbital). These files detail all excitations out of the specified orbitals along with a percentage giving their contribution to the CIS expansion as a whole. The sum of the percentages (final column) from each of the `fromi_j.dat` should be 100 if everything is working OK.

8.9.2 Other features of GAUSSIANTOQMC

The code also contains some crude normalization and plotting routines that are really just debugging aids. By setting the flag **test**=`T` in `gaussiantoqmc.f90` and recompiling, the user is given the option of plotting and testing the normalization of individual molecular orbitals. The axis along which the plotting is done is set in `wfn_construct.f90` and this must be hacked if the user wishes to change things.

8.9.3 Getting GAUSSIAN to do what you want

In principle, GAUSSIAN can do an awful lot of things. In fact, some of these things seem to require magical incantations. These will be described in this section, broken down into the different calculations to which they apply. The comments on G98 refer to revision A9 and may depend on which version is used.

General bits and pieces

Some points to note:

- Both G94 and G98 appear to have formatting errors when printing out the Gaussians used in the ECP expansion—large exponent values are replaced with stars. This does not affect the subsequent calculation.
- G94's ECP (pseudopotential) package will not accept expansions containing more than 13 Gaussians per angular-momentum channel.
- G98's ECP integral package crashes when one attempts to do a large (both in terms of basis and ECP expansion) calculation with symmetry switched on. The solution to this is to switch symmetry off using 'Nosymm'.
- One cannot use basis functions containing g and higher basis functions with a pseudopotential in G94.
- Obviously, `gaussiantoqmc` needs the molecular orbitals (MOs) produced by the calculation. It gets these from the formatted checkpoint file which is produced by putting 'Formcheck=(Basis,MO)' in the route section of the GAUSSIAN job. Alternatively, it may be obtained from the binary checkpoint file (`.chk`) using the `formchk` utility: see the GAUSSIAN manual.
- Many (but not all) of the IOps mentioned here are described on GAUSSIAN's website at `https://gaussian.com/iops/`.

CIS

- Performing a 'HF test' for an excited state: It is possible to get GAUSSIAN to output a breakdown of the energy of a CIS excited state which may be compared with the results of a determinant-only VMC run. The key to this is the density used to perform the population analysis and other post-SCF calculations. By default, GAUSSIAN uses the density produced by the original SCF run. To get the kinetic, nuclear-nuclear potential and electron-nuclear potential energies you must tell it to use the one-particle CI density via 'density=RhoCI'. You must also specify the excited state that you are interested in via 'Root=N' in the CIS options. With all of this done properly, GAUSSIAN produces some output like:

  ```
  N-N= 6.9546274D+00 E-N=-2.3781323D+01 KE= 3.3771062D+00
  ```

  (units of a.u.), which is hidden in the density analysis right at the end of the output.
- Some trouble has been encountered with the 'Add=N' option to CIS (which reads converged excited states from the checkpoint file and then calculates N more). The IOp alternative which does work is IOp(9/49=2) (use guess vectors from the checkpoint file) combined with IOp(9/39=N) (make N additional guesses to those present).
- Using the '50–50' option to CIS to calculate singlet and triplet excitations simultaneously can cause problems. It appears best to do the singlet ('singlets') and triplet ('triplets') calculations separately.
- For QMC, we want the complete CIS expansion. GAUSSIAN may be persuaded to output all excited states with coefficients $> 10^{-N}$ by using IOp(9/40=N). Typically $N = 5$ is good enough. `gaussiantoqmc` outputs the sum of the square of the coefficients so that the user can see how complete the wave function is. (Standard GAUSSIAN output has the coefficients normalized so that the sum of their squares for a complete expansion would be unity.)

CISD

Although `gaussiantoqmc` cannot read a CISD wave function it might be worth mentioning that IOp(9/6)=N is equivalent to MAXCYCLE=N for such a calculation.

CASSCF

- As described in Foresman and Frisch's book [32], getting CASSCF to converge for a singlet state is difficult. The following procedure normally works:
  1. Run a ROHF calculation for the lowest triplet state of the system and save the checkpoint file.
  2. Run CASSCF for the second triplet state ('Nroot=2') taking the initial guess from the checkpoint file. (GAUSSIAN calculates the first and second triplets but converges on the second.)
  3. Run CASSCF for the first triplet ('Nroot=1') taking the initial guess from the checkpoint file.
  4. Run CASSCF for the singlet excited state ('Nroot=2') taking the initial guess from the triplet checkpoint file.
  5. Finally, run CASSCF for the singlet ground state ('Nroot=1').
- By default, GAUSSIAN uses spin configurations (combinations of Slater determinants) in a CASSCF calculation. It is best to converge the CASSCF state that you want using this option. However, for input to the QMC code, the wave function must be in terms of Slater determinants. In principle, the 'SlaterDet' option to CASSCF will do this, but I never succeeded in getting it to work. Instead, specifying IOp(4/21=10) does the trick, as does IOp(4/46=3).
- For large CASSCF calculations on the alpha cluster Columbus, the diagonalization method must be changed by specifying IOp(5/51=1).
- In large CASSCF calculations where very many determinants are involved, GAUSSIAN currently only prints the first fifty determinants in the expansion (those with the largest coefficients). The significance of the truncation may be judged by looking at the sum of the squares of the coefficients that `gaussiantoqmc` outputs when it reads the wave function. Unfortunately, this truncation prevents a full 'HF test' (i.e., running the wave function in VMC without a Jastrow factor and checking that the result agrees with that of GAUSSIAN), but the energy returned by such a test should be *above* that reported by GAUSSIAN.
- As well as this truncation to fifty determinants, GAUSSIAN has a formatting error which means that if the index number of a determinant is greater than 99,999 then it is replaced by stars and is thus useless. `gaussiantoqmc` deals with this by simply throwing away such configurations which further truncates the expansion.
- GAUSSIAN switches to direct mode for large CASSCF calculations and in doing so automatically stops printing the definitions of the Slater determinants used in the calculation. In order to reconstruct the wave function we do of course need to know what the Slater determinants are. GAUSSIAN may be persuaded to print them by using IOp(4/46=3) IOp(4/21=10) IOp(4/21=100). The first two of these both tell GAUSSIAN to use Slater determinants (I specified both to be on the safe side) and the last one in theory tells it to 'just print the configurations' although in fact it still proceeds to do the calculation as well.
- Restarting a CASSCF calculation from a previously converged run fails when symmetry is switched on. Use 'Nosymm' to avoid this problem.

TD-HF and TD-DFT

As far as converting the resulting wave function for use in CASINO is concerned, these two methods are no different to CIS apart from the issue of normalization. In CIS, the default output is normalized so that the sum of the squares of the coefficients is equal to unity. In a TD-HF or TD-DFT calculation (which involves solving a non-Hermitian eigenvalue problem) a different scheme is used which essentially means that the sum of the squares of the coefficients is arbitrary.

It should also be noted that GAUSSIAN cannot do gradients within TD-DFT yet and so cannot relax excited states.

8.9.4 Summary of routines used in GAUSSIANTOQMC

Routine	Purpose
analyse_cis_state	Break the selected CIS/TD-DFT state down into excitations from each distinct occ. MO
awk_like	Module and subroutines to give AWK-like functionality. Used in parsing the GAUSSIAN output files
cas_wfn	Brings each of the CAS configurations into maximum coincidence with the reference configuration and (optionally) calls `resum_cas`
cas_write	Outputs the CAS wave function (i.e., the determinant expansion) to the `gwfn.data` file
cis_data	MODULE—holds data defining the CIS and other multi-determinant wave functions
con_coeffs	Multiplies the common part of shell normalization factors into the contraction coefficients and adjusts their storage for improved accessibility
fatal	Echoes a string and then kills the program
g94_wave function	MODULE—holds the data about the type of GAUSSIAN run as well as the data defining the MOs etc.
gaussiantoqmc	Main driver unit
g_d_type	Evaluates a primitive d-type Gaussian basis function at a specified location in 3D space
g_s_type	Evaluates a primitive s-type Gaussian basis function at a specified location in 3D space
get_gauss_version	Reads the GAUSSIAN output file and identifies whether it is from GAUSSIAN94 or GAUSSIAN98
integ_params	MODULE—holds parameters defining granularity of plotting and integration grids as well as which MO to plot/test
max_coincidence	Brings a CIS configuration into maximum coincidence with a specified 'reference' determinant
normalization_check	Tests the normalization of a specified MO
normalise_ci	Normalizes the CIS expansion. Not necessary for QMC but keeps things tidy and output gives an idea of how complete the expansion is
numsrt	Sort an array into descending (numeric) order and (optionally) keep track of reordering
numsrt_2way	As for `numsrt` but has additional argument to specify ascending or descending order
numsrt_signchange	As for `numsrt` but returns an associated sign change given by multiplying by -1 for each exchange
pack_evcoeffs	Stores the alpha and beta eigenvector coefficients separately and multiply in remaining normalization factors (which differ between d_{xx},$d_{x^2-y^2}$ etc.)
paramfile	MODULE—contains `define_pi` and also defines conversion factors for Hartree to eV and Bohr to Ångstrom
psi	Evaluates an MO of a given spin at a specified point in space

Routine	Purpose
qmc_write	Writes the `gwfn.data` file
re_sum	Resums a CIS/TD-DFT wave function
read_G9xout	Reads the output file produced by the GAUSSIAN job. Gets the nuclear-nuclear potential energy, CIS/TD-DFT/CAS expansion (if present) and HF eigenvalues
read_fchk	Reads the formatted checkpoint file produced by the GAUSSIAN job. Gets the MOs etc.
rejig	MODULE—contains `numsrt` and hence prototypes it which is necessary because it has an *optional* argument
resum_cas	Partially resums a CAS expansion
set_parameter_values	Sets the value of Pi and related constants
shell_centres	Identifies the positions of the distinct shell centres and store the first shell index corresponding to each
sum_degen_excite	Called by `analyse_cis`. Loops over excitations out of a given range (i–j) of (degenerate) occ. MOs and sums those that correspond to degenerate final (virtual) MOs. Outputs the results to a `fromi_j.dat` file.
user_control	Calls the major reading routines and asks the user about excited states and resumming
wfn_construct	Plot an MO (debugging option). Called by `wfn_test`
wfn_test	Asks user about plotting and normalization testing. Optionally calls `wfn_construct` and `normalization_check`.

8.9.5 Confession

The guy who wrote this left Cambridge years ago and nobody here understands how this works or has ever used GAUSSIAN.

Here is a sequence of emails between me (MDT) and Katie Schwarz which might help for some things.

```
Dear Mike Towler,
I am a graduate student in Richard Hennig's group, and we have been studying
the benzene dimer with CASINO.  I have recently started performing CISD
calculations in Gaussian03 for CASINO, and I spent a very long time trying
to figure out how to output all of the determinant coefficients.  I emailed
Gaussian, and they suggested using the IOp 9/28, which has worked for me.
Richard mentioned that this may also be useful to you (or to people who use
the CASINO manual), so I am sending this information along.
Hopefully it will save someone from scrolling through the IOps!
-Katie Schwarz

Dear Katie,

Thanks for this info.

However - I have to confess I've never used Gaussian or the converter (which
was written by some student who disappeared about ten years ago) and I'm not
quite sure what your email means. Could you elaborate?

Also, someone just asked me whether the converter will work in the
unrestricted CI case. Did you happen to notice whether this is the case?  (just
to save me the trouble of analyzing the source code - I presume the info is not
in the manual..).

Best wishes,
Mike

Hi Mike,

To try to answer your question about gaussiantoqmc and the unrestricted CI
case: I don't think that gaussiantoqmc will work for full, unrestricted CI, and
it definitely doesn't work for unrestricted CISD.  I personally couldn't make
```

```
gaussiantoqmc work for multi-determinant wave functions, but according to the
manual, it works for CASSCF and CIS files.

To explain what I wrote earlier, Gaussian has a list of internal options (IOps,
https://www.gaussian.com/iops.htm) which allow users more flexibility in
calculations than the standard Gaussian keywords.  The keywords are
well-documented on the Gaussian site, but the IOps are just listed by number on
a separate site.  In order to print all of the wave function coefficients for
the single and double excitations in CISD, the IOp 9/28 is necessary.  The
CASINO manual does not list this IOp, and other users might benefit from a
mention of it.

Because I wanted to use Gaussian's CISD calculations, I wrote a short unix
shell script to convert the Gaussian output to a CASINO-readable form for the
correlation.data file.  The script is currently messy, but if it would help
other users, I can clean it up and send it to you once I finish testing it.

Actually, do you know of researchers using CISD calculations with CASINO, or do
most users use CASSCF wave functions?

Thanks,

-Katie
```

8.10 GP

We used to support an old Lawrence Livermore code which, when we played with it years ago, was called GP. It also appears to have been called JEEP, and has now morphed into something called QBOX which has a website at `http://qboxcode.org/`.

There is a converter called '`jeep_to_pwfn`' supplied with CASINO which used to read in GP '`jeep.wf`' and '`atoms.sys`' files to produce a CASINO `pwfn.data` file. It is highly unlikely that this still works with modern versions of QBOX. If anyone out there would like to update this information, or to make the converter work in a modern context, then they would be very welcome.

8.11 MCEXX

MCEXX is a code written by A. Görling, S. Rohra, P. Carrier, A. Hesselmann, H. Schulz and E. Trushin at the University Erlangen Nuremberg in Germany. It is a plane wave electronic structure code for conventional Kohn–Sham calculations (LDA/GGAs), Hartree–Fock calculations, DFT calculations with hybrid functionals, exact-exchange (EXX) Kohn–Sham calculations, and calculations treating electron correlation within the random-phase approximation (RPA). Spin-orbit interactions, non-collinear spin, and accompanying magnetization currents can be treated. With explicitly temperature-dependent functionals calculations for the very high temperatures relevant in warm dense matter physics (e.g., in the context of nuclear fusion or matter in stars) can be carried out. An interface between MCEXX and CASINO was implemented in November 2013. For further information contact `andreas.goerling @ fau.de`.

8.12 MOLPRO

Website: `https://www.molpro.net`

Support for this Gaussian basis set quantum chemistry code is provided through the `molden2qmc` utility by Mike Deible and Vladimir Konjkov, which can convert files written in the quasi-standard MOLDEN format into CASINO's `gwfn.data` format.

To generate a `gfwn.data` file, sit in a directory containing suitable MOLDEN output, type '`molden2qmc`', then follow the prompts.

See the file `CASINO/examples/generic/gauss_dfg/RESULTS` for the current status of the interface.

8.13 ORCA

Website: `https://www.faccts.de/orca/`

Support for this Gaussian basis set quantum chemistry code is provided through the `molden2qmc` utility by Mike Deible and Vladimir Konjkov, which can convert files written in the quasi-standard MOLDEN format into CASINO's `gwfn.data` format.

To generate a `gfwn.data` file, sit in a directory containing suitable MOLDEN output, type '`molden2qmc`', then follow the prompts.

See the file `CASINO/examples/generic/gauss_dfg/RESULTS` for the current status of the interface.

There is an online forum for ORCA here: `https://orcaforum.kofo.mpg.de/app.php/portal`, where the developers can give you direct support.

8.14 PSI-4

Website: `https://psicode.org/`

Support for this Gaussian basis set quantum chemistry code is provided through the `molden2qmc` utility by Mike Deible and Vladimir Konjkov, which can convert files written in the quasi-standard MOLDEN format into CASINO's `gwfn.data` format.

To generate a `gfwn.data` file, sit in a directory containing suitable MOLDEN output, type '`molden2qmc`', then follow the prompts.

See the file `CASINO/examples/generic/gauss_dfg/RESULTS` for the current status of the interface.

8.15 PWSCF/Quantum Espresso

Website: `https://www.quantum-espresso.org`

This is probably the best-supported free plane-wave DFT package. PWSCF supports CASINO directly (properly so only from version 4.3). Note that versions 5.1 and 5.1.1 of the code were released with an (easily fixable) bug that affected its CASINO converter—see the note at the end of this section.

The text below is taken from the `README_pwscf` file in the `utils/wfn_converters/pwscf` directory, hence the repetition.

The `utils/wfn_converters/pwscf` directory contains a run script `runpwscf` which may be used to run the PWSCF program on all the architectures that CASINO supports (it uses the same architecture information in `CASINO/arch`). It understands more or less the same set of command line flags as `runqmc`, one important addition being the `--qmc/-w` option which toggles the creation of wave function files. The script assumes the two distributions are in `$HOME/CASINO` and `$HOME/espresso`; if this is not the case the defaults can be changed with the `--chome` and `--ehome flags`. Fine-grained control over parallelism in PWSCF is done by defining the number of images, pools, task groups and linear algebra groups (see the Espresso documentation); these may be specified with the `--image`, `--npool`, `--ntg` and `--ndiag` flags to `runpwscf`.

Note that if your CASINO arch file defines a command for running CASINO and PWSCF (such as `SCRIPT_RUN: mpirun -np &NPROC& &BINARY&`), then it must include a tag `&BINARY_ARGS&` following the `&BINARY&` tag. This is because the PWSCF executable takes command line arguments such as `-pw2casino`, `-npool` etc., which are not required by CASINO. For this reason an arch file which works for CASINO may not necessarily work for PWSCF without this modification. To check this on a batch machine, type '`runpwscf --qmc --check-only`' and examine the resulting '`pw.x`' batch file. If the line containing, e.g., the `mpirun` command above does not have '`-pw2casino`' following '`pw.x`', then you will need to add `&BINARY_ARGS&` to your arch file.

The interface between PWSCF and CASINO is provided through a file with a standard format containing geometry, basis set and orbital coefficients. For SCF calculations, the name of this file may be `pwfn.data`, `bwfn.data` or `bwfn.data.b1` depending on user requests (see below). If the files are produced from an MD run, the files have a suffix '.1', '.2', '.3', etc., corresponding to the sequence of time steps.

CASINO support is implemented by three routines in the `PW` directory of the espresso distribution:

- `pw2casino.f90`: the main routine,

- `pw2casino_write.f90`: writes the CASINO `xwfn.data` file in various formats,
- `pw2blip.f90`: does the plane wave to blip conversion, if requested.

Relevant behaviour of PWSCF may be modified through an optional auxiliary input file, named `pw2casino.dat` (see below).

In some versions prior to 4.3, this functionality was provided through separate post-processing utilities available in the `PP` directory: these are no longer supported. For QMC-MD runs, PWSCF, etc., previously needed to be 'patched' using the patch script `PP/pw2casino-MDloop.sh`; this is no longer necessary.

Note that 'twist-averaged' calculations may be done with PWSCF/CASINO using the `twistav_pwscf` script. See Sec. 28.3.5.

How to generate `xwfn.data` files with PWSCF

Use the '`-pw2casino`' option when invoking PW.X, e.g.:

`pw.x -pw2casino < input_file > output_file`

The `xfwn.data` file will then be generated automatically.

If running using the supplied RUNPWSCF script, then one would type (with assumed `in.pwscf` and `out.pwscf` i/o files):

`runpwscf --qmc` OR `runpwscf -w`

On parallel machines, one could type, e.g.,

`runpwscf --qmc -p 128`

to run the calculation on 128 cores, or whatever.

PWSCF is capable of doing the plane wave to blip conversion (see Sec. 9) directly (a CASINO blip-generation calculation with **runtype** set to 'gen_blip' is not required) and so by default, PWSCF produces the 'binary blip wave function' file `bwfn.data.b1`.

Various options may be modified by providing a file '`pw2casino.dat`' with the following format:

```
&inputpp
blip_convert=.true.
blip_binary=.true.
blip_single_prec=.false.
blip_multiplicity=1.d0
n_points_for_test=0
/
```

Some or all of the five keywords may be provided, in any order. The default values are as given above (and these are used if the `pw2casino.dat` file is not present).

The meanings of the keywords are as follows:

blip_convert Re-expand the converged plane-wave orbitals in localized blip functions prior to writing the CASINO wave-function file. This is almost always done, since wave functions expanded in blips are considerably more efficient in QMC calculations. If **blip_convert** is `F`, a `pwfn.data` file is produced (orbitals expanded in plane waves); if **blip_convert** is `T`, either a `bwfn.data` file or a `bwfn.data.b1` file is produced, depending on the value of **blip_binary** (see below).

blip_binary If `T`, and if **blip_convert** is also `T`, write the blip wave function as an unformatted binary `bwfn.data.b1` file. This is much smaller than the formatted `bwfn.data` file, but is not generally portable across all machines.

blip_single_prec If `F`, the orbital coefficients in `bwfn.data` or `bwfn.data.b1` are written out in double precision; if the user runs into hardware limits **blip_single_prec** can be set to `T`, in which case the coefficients are written in single precision, reducing the memory and disk requirements at the cost of a small amount of accuracy.

blip_multiplicity The quality of the blip expansion; this is **xmul** in Sec. 9.

n_points_for_test Number of points in overlap test: see Sec. 9.

Pseudopotentials in PWSCF and CASINO

DFT trial wave functions produced by PWSCF must be generated using the same pseudopotential as in the subsequent QMC calculation. This requires the use of tools to switch between the different file formats used by the two codes.

CASINO uses the 'CASINO-tabulated format', PWSCF officially supports the UPF (version 2) format (though it will read other 'deprecated' formats).

It should be noted that ultrasoft and projector-augmented-wave pseudopotentials cannot be used with the CASINO code.

There are two options for switching between the various file formats:

(1) `casino2upf`/`upf2casino` (written by Simon Binnie)

Converts CASINO tabulated format to and from UPF version 2 (UPFv2) format.

This is included in the Quantum Espresso distribution (see directory `upftools`).

In the CASINO distribution, see in addition the `README` and `INSTRUCTIONS` files in the `utils/pseudo_converters/pwscf/casino2upf` directory.

Note the following pitfall. The `casino2upf` utility marks any UPF files it creates as having been generated using Hartree–Fock (since they generally are). If you do not supply a value for the **input_dft** keyword in the 'system' section of the PWSCF input file, then PWSCF will attempt to use the functional specified in the pseudopotential file, i.e., it will try to do a Hartree–Fock calculation, and—given that this is only possible with PWSCF if you compiled it having invoked *configure* with the `--enable-exx` flag—then the code may stop and whine about not having been compiled with support for hybrid functionals. This can be confusing. Solution: specify **input_dft** in the input file.

(2) `casino2gon` (written by John Trail)

Converts CASINO tabulated format to the (deprecated) GON format.

This is included in the `utils/pseudo_converters/pwscf/casino2gon` directory in the CASINO distribution.

Which utility to use?

Since UPFv2 is the current official format for PWSCF, one would normally use the `casino2upf` converter (though as of 3.2011 PWSCF will still read `.gon` files).

The `casino2gon` alternative is useful when you need to do interpolation, i.e., use a non-standard grid or wave functions on a different grid. In particular it can take `pp_gaussian` or `pp_gamess` as input as well as `pp.data` (see the CASINO pseudopotential website).

8.15.1 Erroneous versions

Do not use unmodified versions 5.1 or 5.1.1 of PWSCF (extant from April to December 2014) to generate QMC trial wave functions as they contained a bug introduced by the developers that affected the CASINO converter routine. SVN development versions from December 2014 contain a fix for this bug, as will subsequent releases.

To fix the bug manually you need to replace line 366 of `PW/src/pw2casino_write.f90`

```
CALL get_buffer (evc, nwordwfc, iunwfc, ikk )
```

with

```
IF( nks > 1 ) CALL get_buffer (evc, nwordwfc, iunwfc, ikk )
```

8.16 TURBOMOLE

TURBOMOLE is a quantum-chemical program package, initially developed in the group of Prof. Dr. Reinhart Ahlrichs at the University of Karlsruhe and at the Forschungszentrum Karlsruhe, and now run by a commercial company. It provides all standard and state of the art methods for ground state calculations (Hartree–Fock, DFT, MP2, CCSD(T)), excited state calculations at different levels (full

RPA, TDDFT, CIS(D), CC2, ADC(2), . . .), geometry optimizations, transition state searches, and molecular dynamics calculations.

Website: `https://www.turbomole.org/`

The CASINO converter/interface was implemented a long time ago (2001) by Stefan Grimme of the University of Muenster, and has almost certainly stopped working. A piece of code which was to be incorporated into the TURBOMOLE source is provided in `CASINO/utils/wfn_converters/turbomole_old`. (Martin Korth of the University of Ulm was the last person known to possibly have his own private version which works).

In modern times however, support for this code is now also provided through the `molden2qmc` utility by Mike Deible nd Vladimir Konjkov, which can convert files written in the quasi-standard MOLDEN format into CASINO's `gwfn.data` format.

To generate a `gfwn.data` file, sit in a directory containing suitable MOLDEN output, type '`molden2qmc`', then follow the prompts.

See the file `CASINO/examples/generic/gauss_dfg/RESULTS` for the current status of the MOLDEN interface for TURBOMOLE.

8.17 2DHF

2DHF is a numerical orbital Hartree–Fock diatomic molecule code, available from this website: `http://fizyka.umk.pl/~jkob/software/2dhf/`. Unfortunately access to the website appears to be forbidden (October 2024).

Output from this program may be converted to CASINO's numerical diatomic orbital format `dwfn.data` format using the converter `uf2dwfn` (written by John Trail). This is supplied as a CASINO utility, but is not automatically compiled and setup by the CASINO build system and must be installed manually. For further details, see the instructions in the file:

`CASINO/utils/wfn_converters/2dhf/Uf2dwfn/README`.

8.18 Unsupported programs

If the program in question uses Gaussian, plane-wave, Slater or blip basis sets (or represents orbital in a grid for atoms or dimers) then feel free to create your own converter to write the output of the program in the appropriate `[x]wfn.data` format and send it to us (mdt26 at cantab.net) for inclusion in future releases. If you are the author/owner of the electronic structure package in question, then you may like to add internal support so that the code can write out CASINO wave functions directly. We can provide advice about this (there is in fact some on the CASINO web site; see `https://vallico.net/casinoqmc/interface_development/`).

If your program uses some other basis set which requires a new CASINO orbital evaluator to be written, then this could be a major project. Please ask.

Note that support for any Gaussian basis set program that supports the quasi-standard MOLDEN file format can in principle be added relatively easily via the standard MOLDEN interface of Mike Deible and Vladimir Konjkov. Vladimir is known to have at least looked at adding NWCHEM and/or Q-CHEM.

8.19 Request for help

A final remark: support for plane-wave DFT programs is currently either 'advanced' (PWSCF—in that the program understands about blip functions and can do the necessary PW$\longrightarrow$blip transformations internally, and it can do DMC-MD calculations)—or 'basic' (CASTEP, ABINIT, GP, MCEXX). These latter codes are only capable of writing out plane-wave `pwfn.data` files. If anyone would like to add the blip stuff to e.g., CASTEP or ABINIT then please volunteer: the routines can be essentially nicked from PWSCF, since we wrote them. Any volunteers coming forward to improve interfaces to other codes would be very welcome.

9 Using CASINO with blip functions

Blip functions were devised by Mike Gillan and implemented in CASINO by Dario Alfè [33] (the original implementation has since been significantly improved by many contributors).

Plane-wave DFT codes such as CASTEP can be used to produce CASINO `pwfn.data` files with the orbitals expanded in plane waves, but it is inefficient to use these directly in CASINO (though you can if you want). Re-expanding the orbitals in a basis set of localized 'blip functions' on a grid makes the code run faster and scale better with system size than it does with plane waves, though blips can require a lot of memory and disk space.

Performing a CASINO calculation with **runtype** set to 'gen_blip' will transform the `pwfn.data` files generated by a plane-wave DFT package into `bwfn.data` or `bwfn.data.bin` files. The PWSCF software is capable of generating blip `bwfn.data` files (and their binary equivalents) directly, and 'gen_blip' calculations are unnecessary.

The quality of the blip expansion (i.e., the fineness of the blip grid) can be improved by increasing the grid multiplicity parameter **blip_xmul** `input` parameter (in PWSCF the **xmul** parameter is given as input in the `pw2casino.dat file`). A suitable default value is 2.0. Increasing the grid multiplicity results in a greater number of blip coefficients and therefore larger memory requirements, but the CPU time should be unchanged. For very accurate work, one may want to experiment with **blip_xmul** larger than 2.0. Note, however, that it might be more efficient to keep **blip_xmul** at 2.0 and increase the plane wave cutoff instead.

If you wish CASINO to perform a Monte Carlo evaluation of the overlap between the blip and plane-wave orbitals, set the **blip_nrandpoints** parameter to a positive value. CASINO will then sample the wave function, the Laplacian and the gradient at the specified number of random points in the simulation cell and compute the overlap of the blip orbitals with the original plane-wave orbitals:

$$\alpha = \frac{\langle BW|PW\rangle}{\sqrt{\langle BW|BW\rangle\,\langle PW|PW\rangle}} \tag{38}$$

The closer α is to 1, the better the blip representation. By increasing **blip_xmul**, or by increasing the plane-wave cutoff, one can make α as close to 1 as desired. The blip utility will as you for the number of points to be used (1000 is a good bet); for PWSCF the number of points is given in the `pw2casino.dat` file (the keyword '**n_points_for_test**').

If you set the **blip_calc_ke** to `T` then CASINO will calculate the kinetic energies of your orbitals (in the plane-wave and blip representations). Obviously, the kinetic energies should be in good agreement. This test can take some time to run, however, and is often skipped.

Finally a blip-generation calculation allows you to translate the atoms in an appropriate manner for a reduced-periodicity QMC calculation. For example, if you would like to study a molecule and you have performed your plane-wave DFT calculation with the molecular coordinates centred on the origin, you should set **blip_periodicity** to 0; `casino` will then translate the atoms (and plane-wave coefficients) in the blip-generation calculation so that the molecule is centred in the middle of the blip grid, which spans the first simulation cell. Note that if you change the periodicity of your calculation for whatever reason then you should redo the blip-generation calculation.

When CASINO is run, the formatted `bwfn.data` file is converted into the unformatted and much smaller `bwfn.data.bin` file. If you update the formatted `bwfn.data` file then please be sure to delete the `bwfn.data.bin` file(s); otherwise the `bwfn.data.bin` file(s) will be read and the (updated) `bwfn.data` file will be ignored.

An older format binary blip file—`bwfn.data.b1`—is still supported. Note that PWSCF can bypass the conversion to binary step in CASINO, as it is capable of producing single `bwfn.data.b1` files directly (in fact, it does this by default). It is intended that PWSCF will presently be converted to produce `.bin` files instead of `.b1`, and by the time you read this, that may have already been done.

If your machine has CPUs with multiple cores, the set of blip (or Gaussian) coefficients can be shared among the cores in each CPU, thus saving a significant amount of memory. This feature, which is implemented for both UNIX System V Inter-Process Communication and POSIX shared memory, must be enabled at compile-time by setting in the relevant `Makefile` include files the flags `NEED_SHM` to `yes` and `CPP_FLAGS_SHM` to `-DSHM_SYSV` or `-DSHM_POSIX` for System V or POSIX version, respectively. (Once set up correctly. Just type 'make shm' or 'make openmpshm'.)

The gain in speed with respect to plane waves should be of the order of $N_{\rm PW}/64$, where $N_{\rm PW}$ is the number of plane waves in the plane-wave basis.

Localized plane-wave orbitals can also be represented in terms of blips. This is discussed in Sec. 27.

10 Utilities provided with the CASINO distribution

A large variety of little programs which do useful things are provided in the `~/CASINO/utils/` directory. Although you should probably refer to the `README` files in each of the subdirectories for up-to-date and more comprehensive information, here we provide a reasonably current list of them:

- `abinit_to_casino_pp`, `casino_to_abinit_pp`: Converts pseudopotentials for the ABINIT program into CASINO format, and *vice versa*.
- `ae_pp_maker`: generate an all-electron pseudopotential in the CASTEP `.recpot` format.
- `billy`: Shell script for optimizing basis sets and geometrical parameters with CRYSTAL95/98/03/06/09/14/17. Reasonably vital for developing decent trial wave functions for CASINO with these programs. See the documentation in `~/CASINO/utils/billy/` (this was the first proper computer program that baby MDT ever wrote). The `opt_crystal` utility is a more modern version of this and is normally to be preferred since it tends to give a better answer—though beware it tends to be much slower than billy.
- `casinohelp`: Simple script to invoke the CASINO help system. Usage:

```
casinohelp <casino_keyword>     : tells you the definition and type of keyword
casinohelp search <text>        : finds <text> in descriptions of all keywords
casinohelp all                  : lists all possible keywords
casinohelp basic                : lists all basic level keywords
casinohelp inter                : lists all intermediate level keywords
casinohelp expert               : lists all expert level keywords
```

- `casinostyle_checker`: Checks that the `.f90` files supplied as command-line arguments adhere to the CASINO style guidelines (see Sec. A.3); obviously this is only relevant for developers of CASINO.
- `champ_to_casino_pp`: Converts pseudopotentials for the CHAMP QMC program into CASINO format.
- `change_inputs`: Script to allow users to change, add or remove keywords in large numbers of `input` files. Similar to `modify_inputs`, but aimed at users rather than maintainers. Type `change_inputs -h` for more information.
- `clearup`: Script for cleaning up after CASINO by removing output and indicator files, etc. It attempts to 'reset' directories to a suitable starting point.
- `clearup_twistav`: Script for cleaning up after a twist-averaging calculation using `twistav_pwscf` or `twistav_castep`.
- `crysgen06/09/14/17`, `crystaltoqmc`: The `crysgen06/09/14/17` scripts drive the program `crystaltoqmc`, which takes the output of the corresponding versions of the CRYSTAL program and produces a CASINO `gwfn.data` file. See Sec. 8.6.2.
- `det_compress`: Given an `mdet.casl` file describing a multi-determinant expansion, produce a compressed multi-determinant expansion and write it to `cmdet.casl`. See Secs. 7.9 and 21. DET_COMPRESS is an imported, cross-licensed utility; refer to https://github.com/plopezrios/det_compress for the upstream.
- `dfit`: Draw a polynomial through a set of energy points and find the minimum (used by the `billy` utility, but is independent of it).
- `dmc_change_dt`: Supply a directory containing input files for a DMC calculation as a command-line argument, and this utility will copy the directory and modify the `input` file to increase the DMC time step by a factor of four, decrease the target population by a factor of four and decrease the number of steps by a factor of two. This is intended to help users perform DMC calculations at different time steps and then extrapolate to zero time step and infinite population with near-optimal efficiency [34]. The time step etc. can be changed by other amounts if desired.

- `egaussian`: Utility for extracting energies etc. from the output of the GAUSSIAN code. See Sec. 8.9.

- `envmc`: Utility to extract VMC energies, energy components, standard errors and time taken from the standard output file. Examples of the use of `envmc` are given in Sec. 6.2 and Sec. 6.3.1. See `~/CASINO/utils/envmc/` for further information. Note that `envmc` is short for 'VMC energy' so it doesn't work for DMC, OK?

- `extrapolate_tau`: Program to extrapolate DMC energies to zero time step by fitting a polynomial form to the DMC energy as a function of time step. For sufficiently small time steps, the bias in the DMC energy should be linear in the time step. To use, create a text file containing DMC time steps, energies and error bars, arranged in three columns (MAKE_E_V_DT may help with this); then type *extrapolate_tau* and answer the questions. The name of the file holding the energy against time step can be supplied as a command-line argument to `extrapolate_tau`.

- `extr_casino`/`extr_pwscf`: Simple utilities for extracting energies, etc., from the output of DMC-MD simulations carried out using `runqmcmd`.

- `finsize`: Calculates post-run table of kinetic-energy and XC-energy finite-size corrections and reports the finite-size-corrected Ewald and/or MPC energies. The utility requires an `expval.data` file containing a structure factor, as well as the CASINO `input` and `out` files.

- `format_configs`: Format an unformatted `config.in` file to give a formatted `config.in_formatted`, or *vice versa* (it similarly treats `config.out` and `config.backup` files). This enables one to read its contents, e.g., for development purposes. Furthermore, it allows one to transfer the architecture-specific `config.in` file between different computer systems, since the formatted file can be read by any machine.

- `gaussiantoqmc`: This program takes the output of GAUSSIAN and produces a CASINO `gwfn.data` file. See Sec. 8.9 for information.

- `get_exciton_binding`: Calculate an excitonic energy gap and (optionally) an exciton binding energy using QMC results held in directories supplied as command-line arguments. The first directory specified should hold a ground-state calculation, followed by a directory containing a promotion calculation, optionally followed by directories holding the addition and subtraction calculations required to give the corresponding quasiparticle gap. Energies are read from the results of the REBLOCK utility stored in `reblock.results`, unless the option '-out' is given, in which case energies will be read from the `out` file or the `dmc.status` file.

- `get_qp_gap`: Calculate a quasiparticle energy gap using QMC results held in directories supplied as command-line arguments. The first directory specified should hold a ground-state calculation, followed by directories holding the addition and subtraction calculations required to give the quasiparticle gap. Energies are read from the results of the REBLOCK utility stored in `reblock.results`, unless the option '-out' is given, in which case energies will be read from the `out` file or the `dmc.status` file.

- `graphdmc`: Plot energy versus move data from a DMC run (i.e., the numbers in `dmc.hist`) in an XMGRACE plot. If for any reason some lines have been removed from the `dmc.hist` file then type *graphdmc -x* to generate a new set of line numbers when the plot is made. An example of the use of `graphdmc` is given in Sec. 6.4.

- `haltqmc`: stop a CASINO job, tidy up and analyse the output (check for warnings, move `config.out` to `config.in`, remove indicator files, run REBLOCK) and update the `input` file for the next stage of the calculation.

- `input_kw_conv` Program for converting between the 'old' (e.g., **nmove**, **nconfig**) and 'new' (e.g., **vmc_nstep**, **dmc_target_weight**) keyword sets. Note that support for the old sets will be removed in early Jan 15, at which point this utility might be more heavily used.

- `update_src` Script to extract a CASINO tar archive and set it up in a directory `CASINO_vxx.xx.xx` with a `CASINO` symbolic link pointing at it.

- `heg_hfta` Program for evaluating twist-averaged and non-twist-averaged Hartree–Fock energies for 1D, 2D and 3D HEGs at finite $N_\uparrow$ and $N_\downarrow$. For 3D HEGs, (twist-averaged) perturbative relativistic corrections are evaluated. The program is parallelised using OpenMP. See `CASINO/utils/heg_hfta/README` for more information.

- `ion_dist` Program for automatic generation of the **edist_by_ion** block in the `input` file for CASINO. Currently works only for antiferromagnetic Wigner crystals that have been generated by the CRYSTAL program (since the numbers are generated from an analysis of `gwfn.data`). See documentation in `~/CASINO/utils/ion_dist/`. Current holder of 'Most Useless Utility' award.

- `jeep_pp`: Convert a pseudopotential from the format used by GP into the format used by CASINO.

- `jeep_to_pwfn`: This program takes the output of GP version 1.8.0 and produces a CASINO `pwfn.data` file.

- `localizer`: Utility to read in a `pwfn.data` file containing extended orbitals expanded in plane waves and generate localized orbitals, also represented in a plane-wave basis. See Sec. 27.

- `louis`: MDT's pilot wave quantum trajectory code, included with CASINO to help with the development of pilot-wave QMC. Also includes the utilities `runlouis`, `plot_louis`, `louishelp`, `louisplot2d`, `louisplot3d` and `modify_louis_inputs`, which do the obvious things. See utils/louis/README for more details.

- `make_E_v_dt`: Create a file holding DMC energy and error bar against time step, for plotting or for use with EXTRAPOLATE_TAU. Supply the names of directories holding completed DMC jobs as command-line arguments.

- `kvec_maker`: Utility for generating the **k**-vector grid for the orbitals in a QMC calculation; the results can be supplied to CASTEP or PWSCF to generate the orbitals for a particular supercell. Simply run the program and answer the questions!

- `make_new_mpc`: Convert CASINO 1.x `density.data` and `eepot.data` files into CASINO 2.x `mpc.data`. Located in the `converters_2.0 directory`.

- `make_p_stars`: Utility for helping users construct p terms in the Jastrow factor and π terms in the backflow function. Type *make_p_stars* and answer the questions. Note that p should have the symmetry of the simulation cell rather than the primitive cell. See Secs. 22 and 23.1.4.

- `mcta_hf`: Utility to compute the Monte Carlo twist averaged Hartree–Fock energy components for HEGs. Refer to the `utils/heg_mcta/README` file for usage information.

- `mcta_post_process`: Utility to perform post-processing analysis on Monte Carlo twist averaged data for HEGs. This utility uses a control variate technique to reduce the statistical error bar of the QMC energy; see Sec. 28.3.2. Refer to the `utils/heg_mcta/README` file for usage information. The results of `mcta_post_process` should approximately agree with the results of `twistanalysis_castep`.

- `menugrep`: Script to facilitate the process of editing lines containing items found by `grep`: useful for editing the CASINO source code. See documentation in `~/CASINO/utils/menugrep/`.

- `combine_plot_data`: Program to scale or to take a linear combination of the data in one or more `lineplot.dat` or `2Dplot.dat` files. The files to be combined should be given as command-line arguments, with each filename followed by the scalar that multiplies the data. For example, to combine two `lineplot.dat` files holding VMC and DMC charge densities to form the extrapolated estimate, one could use '`combine_plot_data lineplot_dmc.dat 2 lineplot_vmc.dat -1 > lineplot_extrap.dat`'. If the data have error bars then please use the flag '`-e`' to let combine_plot_data know this.

- `modify_inputs`: Simultaneously modify all `input` files in subdirectories of the current directory (e.g., `~/CASINO/examples/`) by adding or deleting keywords or by changing the value of a keyword. I (MDT) can remember a time when I did this by hand (with 100+ examples); that was entertaining.

- `molden2qmc`: A wave function converter which can produce CASINO gwfn.data files from files in the MOLDEN format produced by quantum chemistry codes such as MOLPRO, PSI-4, Turbomole, ORCA, . . .

- `movie2avi`: Utility to generate animations from `movie.out` files using povray to render the frames and ffmpeg/avconv to produce the video file. Refer to `utils/movie2avi/README` for instructions, or equivalently run the script with `--help`.

- mpr: Pretty printing script for source-code listings, which saves trees (assuming a2ps has been set up correctly for your system).
- multirun: Script for running several CASINO runs sequentially changing any set of parameters in the input file for each of them.
- nstring: Generate integer number sequences. Useful for strings required in correlation.data files if you're using Type 1 labelling of the sets of atoms in the χ and f functions.
- plot_2D: This script generates 2D plots of trial wave functions and particle positions using data produced by **qmc_plot**. The script uses GNUPLOT. See the documentation in ~/CASINO/utils/plot_2D/ for further information.
- plot_bffield, plot_bfphi: These scripts generate 2D plots of the backflow transformation and the inhomogeneous ϕ term (electron–electron–nucleus) of the backflow transformation. They make use of GNUPLOT. See documentation in ~/CASINO/utils/plot_backflow/.
- plot_expval: General program to read a CASINO-produced expval.data file and visualize the results. Converts the data to standard 1D lineplot.dat files which can be visualized with XMGRACE, or 2Dplot.dat/3Dplot.dat files for higher dimensional data which can be visualized by GNUPLOT (via the CASINO plot_2D utility).
- plot_hist: General program to read vmc.hist and dmc.hist files and plot selected quantities as a function of move number.
- plot_reblock: This script uses XMGRACE to plot the reblocked standard error in the mean energy against reblocking transformation number, which is contained in the reblock.plot file produced by the reblock utility. An example of the use of this script is given in Sec. 6.2.
- plot_mpc: Reads a CASINO mpc.data file (containing a Fourier representation of the density of the Slater wave function and a Fourier representation of $1/r$ treated with the minimum image convention). It can convert the data to standard 1D lineplot.dat files which can be visualized with XMGRACE, or 2Dplot.dat/3Dplot.dat files for higher dimensional data which can be visualized by GNUPLOT (via the CASINO plot_2D utility).
- ptm: Manipulate pseudopotentials on radial grids or as Gaussian expansions. See documentation in ~/CASINO/utils/ptm/.
- quad_fit: Program for carrying out a quadratic fit to a set of data, in order to find a local extremum. Simply type *quad_fit* followed by the name of a file containing columns of data in the form x, y or $x, y, \delta y$.
- quickblock: Simple reblocking utility intended for analysing very large dmc.hist files (since it only looks at one column at a time). Alternative to reblock. quickblock allows reblocking of data other than the energy in the dmc.hist files. If a file called dmc.hist is present in the directory then quickblock will assume that you want to analyse the total energy in that file. Otherwise it will ask you for the name of the file (and the column of data) to analyse.
- reblock: Perform full statistical analysis and reblocking of QMC data in vmc.hist and dmc.hist (note the reblocked error bars are computed by CASINO on the fly and are reported at the end of the out file and in the temporary dmc.status file). Information about the reblocking algorithm is given in Sec. 24. Further information can be found in the documentation in CASINO/utils/reblock/.
- remaining_time: Simple script for estimating the amount of wallclock time until a CASINO calculation completes its current phase (e.g., completes equilibration or completes statistics accumulation). Supply one or more CASINO out files as command-line arguments. This is intended to help users manage large numbers of jobs.
- runcrystal: Run script for carrying out CRYSTAL9X and CRYSTAL03/06/09/14/17 calculations. Its use is discussed in Sec. 8.6.2.
- runqmc: General run script for carrying out CASINO calculations on any kind of computer. See Sec. 6.6.

- `runqmcmd`: Script for carrying out QMC molecular dynamics calculations with CASINO and the PWSCF program. Works by calling the `runqmc` and `runpwscf` scripts, ensuring that it can run on any kind of machine. See Sec. 6.7.
- `runpwscf`: General run script for carrying out PWSCF calculations on any kind of computer.
- `runvp`: Utility to run a calculation using several workstations as if they formed a cluster.
- `set_random_seed`: script to assign a different random seed to each of the CASINO `input` files listed as command-line arguments.
- `supercell`: constructs simulation supercells that maximize the radius of the sphere that can be inscribed in the Wigner-Seitz cell of the supercell. Further information can be found in the documentation in `CASINO/utils/supercell/README`.
- `tcm_comps`: Script for listing the jobs running on each of the computers in TCM (Cambridge interest only).
- `tidy_bib`: Tidy up a bibtex file downloaded from a journal website. Supply the bibtex file as a command-line argument.
- `treat`: TREAT is the tail-regression estimator analysis toolbox. The tail-regression estimator is a statistical analysis tool designed to deal with heavy-tailed distributions, such as the local energy and local force distributions from VMC and DMC. Refer to `CASINO/utils/treat/README.md` for usage. See https://arxiv.org/abs/1903.07684 for information on the tail-regression estimator. TREAT is an imported, cross-licensed utility; refer to https://github.com/plopezrios/treat for the upstream.
- `twistanalysis`: Script for analysing the results generated by `twistav_castep` or `twistav_pwscf`. See Sec. 28.3.4.
- `twistanalysis_heg_external`: program for analysing the results of a Monte Carlo external twist-averaging calculation for the HEG.
- `twistanalysis_heg_internal`: program for analysing the results of a Monte Carlo internal twist-averaging calculation for the HEG. Supply one or more `vmc.hist` or `dmc.hist` files as command-line arguments. Uses a slightly different approach to `mcta_post_process`, but the results should be in statistical agreement. See Sec. 28.3.2.
- `twist_average_dipole`: program for twist averaging out-of-plane dipole moments in 2D-periodic systems, e.g. to calculate the polarizability of graphene and other 2D materials. Use `twistav_castep` or `twistav_pwscf` to set up a twist-averaged calculation for a 2D-periodic system with an external electric field in the z direction, then perform VMC and DMC at each twist and tidy up and analyse these calculations using `haltqmc -r`; finally obtain the twist-averaged dipole moment and polarizability by typing '`twist_average_dipole twist????/?MC*/`'.
- `twistav_castep`: Script for carrying out twist averaging for real systems, using the CASTEP plane-wave DFT code in conjunction with CASINO. See Sec. 28.3.4.
- `twistav_pwscf`: Script for carrying out twist averaging for real systems, using the PWSCF plane-wave DFT code in conjunction with CASINO. See Sec. 28.3.5.
- `update_hist`: A utility for converting CASINO version 1 `.hist` files to CASINO version 2 `.hist` files (as if there might be anyone left who might want to do this).

11 Making movies with CASINO

I think somebody did this once way back in the 20th century. And it was rubbish, but this is supposed to be a comprehensive manual.

11.1 How to make movies

In the input file the following keywords control the movie-making process:

```
# MOVIES
makemovie         : T                #*! Make movie (Boolean)
movieplot         : 1                #*! Plot every * moves (Inte
movieproc         : 0                #*! Process to plot (Integer)
moviecells        : F                #*! Plot nn cells (Boolean)
```

Set the keyword **makemovie** to `T` to enable the movie-making facility. You could set **movieplot** to an integer greater than 1 so that the particle positions are only written out every **movieplot** moves. The MPI process which plots the particle positions are chosen by the keyword **movieproc**, which has to be zero or a positive integer less than the total number of MPI processes. If **moviecells** is set to `F` then the unit cell will be plotted, if set to `T` then nearest-neighbour cells in the (x, y)-plane will also be written. Type *runqmc* and an output file called `movie.out` will be produced. The format of the `movie.out` file is as follows:

```
           4
   Input geometry
  C    0.000000    0.000000    0.000000    5.000000
  H   -1.407651   -1.138185    0.054434   -1.000000
  H    0.894354    0.554315    1.263301   -1.000000
  H    0.528074    1.081535   -0.755823   -1.000000
           4
   Input geometry
  C    0.000000    0.000000    0.000000    5.000000
  H   -1.407651   -1.138185    0.054434   -1.000000
  H    0.894354    0.554315    1.263301   -1.000000
  H    0.528074    1.081535   -0.755823   -1.000000
(etc.)
```

Notes:

1. The `movie.out` file follows the standard (x, y, z) molecular format.
2. Line 1 indicates the total number n of ions and particles.
3. Line 2 is a comment.
4. The following n lines consist of five columns. Column 1 specifies the type of particle (`H`=electron, `O`=hole and `C`=other atoms). Columns 2, 3 and 4 are the x, y and z coordinates of the particle. Column 5 specifies the charge of the particle.
5. This information is then repeated. The total number of sets of geometry contained in `movie.out` is controlled by the keywords **vmc_equil_nstep**, **vmc_nstep** and **movieplot** in the `input` file.
6. For column 1, the '`H`=electron, `O`=hole and `C`=other atoms' convention is somewhat confusing, but is necessary because particles have to take on element symbols in order for the standard visualization programs to read the `movie.out` file.
7. Column 5 is not read by the visualization programs. It is only there so that different types of particle can be distinguished.

11.2 Visualization

Having generated the `movie.out` file we are able to visualize the results using VMD or JMOL. (This information probably needs updating!)

11.2.1 VMD

VMD (Visual Molecular Dynamics) is a molecular visualization program. It supports computers running MacOS-X, Unix, or Windows, is distributed free of charge, and includes source code. VMC can be downloaded from `https://www.ks.uiuc.edu/Research/vmd/`.

- Type *vmd.* A 'VMD console' and a 'VMD Display' window will appear.
- In the 'VMD console' window type *menu main on.* An extra 'VMD Main' menu bar will appear.
- On the 'VMD Main' menu bar, click on **File** → **New Molecule**. A 'Molecule File Browser' will appear.
- In the 'Molecule File Browser', browse for the file `movie.out` and choose the file type to be **xyz**. Click **Load** to open the file.
- On 'VMD Main', click on **Graphics** → **Representations**. A 'Graphical Representations' menu bar will pop up. Choose **CPK** as the drawing method, the bond resolution to be **1** and the sphere resolution to be **15**. Click **Apply**.
- On 'VMD Main', click on **Extensions** → **vmdmovie**. (For version 1.8.3, click on **Extensions** → **Visualization** → **Movie Maker**.) A 'VMD Movie Generator' will pop up.
- In **Movie Settings** choose **Trajectory**. The movie can be saved in the AVI or MPEG format. Choose by clicking on **Format** and tick the preferred format. Then check whether the name of the temporary directory suggested is right (this is where the RGB files are created). Note that this directory should be free of RGB files belonging to other users. If this has to be changed then click on the **Set working directory** button and browse for the directory.
- Type in the name of the movie in the box provided. Click on the **Make Movie** button.
- The movie will be displayed in the **Open GL Display** screen. The `.mpg` or `.avi` movie file will be produced in the working directory being specified. They have to be viewed with other viewers, for example *mpeg_play* for `.mpg` files.

For an example movie made with VMD (a CASINO VMC simulation of cyclohexane) see `https://casinoqmc.net/downloads/cyclohexane2.mpg`.

11.2.2 JMOL

JMOL is a free, open source molecule viewer. It supports computers running Windows, Mac OS X and Linux/Unix systems. Jmol can be downloaded from jmol.sourceforge.net/

- Type *jmol.* Click on **File**→ **Open**. Browse for the file `movie.out` and click **Open**.
- Click on **Display** and untick the box for **Bonds**.
- Click on **Extras** → **Animate**. An animation tool bar will appear. To start the movie click on the 'play' symbol.

12 Detailed information: the VMC method

12.1 Evaluating expectation values

The expectation value of the Hamiltonian $\hat{H}$ with respect to the trial wave function Ψ can be written as

$$\langle \hat{H} \rangle = \frac{\int E_{\rm L}({\bf R})|\Psi({\bf R})|^2 \, d{\bf R}}{\int |\Psi({\bf R})|^2 \, d{\bf R}}, \tag{39}$$

where $E_{\rm L}({\bf R}) = \Psi^{-1}({\bf R})\hat{H}({\bf R})\Psi({\bf R})$ is the *local energy.* We can evaluate this expectation value by using the Metropolis algorithm [35] to generate a sequence of configurations $\bf R$ distributed according to $|\Psi({\bf R})|^2$ and averaging the corresponding local energies.

12.2 The sampling algorithm

The implementation of VMC in CASINO involves making trial moves, whether of a single electron or of the entire configuration, and accepting or rejecting the moves in accordance with the Metropolis algorithm. The Metropolis transition probability density is Gaussian of variance τ, where τ is the VMC time step (**dtvmc**).

In the electron-by-electron algorithm, each step consists in proposing individual moves for each of the electrons and subject each move to a separate accept/reject step. In the configuration-by-configuration algorithm one configuration move is proposed per step, and is accepted or rejected as a whole.

Another difference in the CASINO implementation of these two methods is that in the configuration-by-configuration the local energy (and all other expectation values) are evaluated both before and after the move, and it is the average of the two, weighted by the acceptance probability, that enters the accumulation arrays. In the electron-by-electron algorithm we only evaluate the energy after having moved the configuration.

The configuration-by-configuration algorithm has the disadvantage of suffering from long correlation times, which makes it in practice more expensive than the electron-by-electron algorithm in virtually all cases. Thus we will restrict our discussion below to the electron-by-electron algorithm.

The **vmc_method** input parameter selects which is to be used: a value of 1 means electron-by-electron, whereas a value of 3 means configuration-by-configuration.[15]

The local energy does not have to be calculated every configuration move. In particular, energies are not calculated during the first **vmc_equil_nstep** moves of a VMC simulation when the Metropolis algorithm has yet to reach its equilibrium. Furthermore, because the configurations are serially correlated, the expense of calculating the energy at every configuration move is unjustified: it is more efficient to calculate the energy once every **vmc_decorr_period** moves, where typically the input parameter **vmc_decorr_period** might be in the range 4–20.

Note that it is not necessary to write out every local energy calculated. **vmc_ave_period** successive energies can be averaged over before being written out to `vmc.hist`: this helps ensure that the `vmc.hist` file is not excessively large when long production calculations are carried out. Furthermore, on parallel machines the local energies computed on each MPI process are averaged over before being written out.

The input parameter **vmc_nstep** gives the number of energy-calculating steps in the VMC simulation. In parallel machines, each MPI process goes over **vmc_nstep**/*nproc* energy-calculating steps.

12.3 Two-level sampling

The Metropolis acceptance probability for a move from $\mathbf{R}'$ to $\mathbf{R}$ in the standard algorithm[16] is

$$p(\mathbf{R} \leftarrow \mathbf{R}') = \min\left\{1, \frac{|\Psi(\mathbf{R})|^2}{|\Psi(\mathbf{R}')|^2}\right\} = \min\left\{1, \frac{|D_\uparrow^2(\mathbf{R})D_\downarrow^2(\mathbf{R})|\exp[2J(\mathbf{R})]}{|D_\uparrow^2(\mathbf{R}')D_\downarrow^2(\mathbf{R}')|\exp[2J(\mathbf{R}')]}\right\}. \tag{40}$$

It is straightforward to show that if detailed balance [11] in configuration space is satisfied then the resulting ensemble of configurations will be distributed according to the square of the trial wave function.

However, CASINO employs a two-level sampling algorithm, which has been shown to be considerably more efficient [36].

Let us define the first-level acceptance probability

$$p_1(\mathbf{R} \leftarrow \mathbf{R}') = \min\left\{1, \frac{|D_\uparrow^2(\mathbf{R})D_\downarrow^2(\mathbf{R})|}{|D_\uparrow^2(\mathbf{R}')D_\downarrow^2(\mathbf{R}')|}\right\} \tag{41}$$

and the second-level acceptance probability

$$p_2(\mathbf{R} \leftarrow \mathbf{R}') = \min\left\{1, \frac{\exp[2J(\mathbf{R})]}{\exp[2J(\mathbf{R}')]}\right\}. \tag{42}$$

[15]There used to be a **vmc_method**=2, but after extensive testing we concluded that it did not offer any advantage over the other methods and was hard to support.

[16]In the electron-by-electron algorithm, $\mathbf{R}$ and $\mathbf{R}'$ differ only in the coordinates of a single electron.

In the two-level algorithm we accept trial moves from $\mathbf{R}'$ to $\mathbf{R}$ with probability $p_1(\mathbf{R} \leftarrow \mathbf{R}')p_2(\mathbf{R} \leftarrow \mathbf{R}')$. It can be shown that, provided detailed balance in configuration space is satisfied, this procedure also results in an ensemble of configurations distributed according to the square of the trial wave function.

The two-level approach is computationally advantageous because the Metropolis accept/reject step can be carried out in two stages: if the 'first level' is accepted (with probability p_1) then we compute the Jastrow factors of the new and old configurations and determine whether the 'second level' is accepted (with probability p_2). Thus, if a move is rejected at the first level then we do not have to compute the Jastrow factors for the new and old configuration[17].

12.4 Optimal value of the VMC time step

Ideally, the VMC time step **dtvmc** should be chosen such that the correlation period (as determined by reblocking analysis) of the resulting configuration local energies is minimized. For large time steps the move rejection probability is high, which obviously leads to serial correlation. On the other hand, for low time steps the configuration does not move very far at a given step, and so serial correlation is again large.

A rule of thumb for choosing an appropriate **dtvmc** is that the average acceptance probability should be about 50%. If you use at least 500 steps[18] in the VMC equilibration phase, then CASINO will automatically optimize the VMC time step to achieve an acceptance ratio of 50% (provided **opt_dtvmc** is set to 1, which is the default).

Notice that one could determine the optimal time step by analysing the correlation length of the local-energy sequence, but this is just too expensive and complicated and would provide little (if any) benefit over the simple, inexpensive '50% rule'.

13 Detailed information: the DMC method

See, e.g., Refs. [11], [12], [13] and [14] for general information about DMC. Here we present some more detailed information about the implementation of DMC within CASINO. Our algorithm follows that of Umrigar, Nightingale and Runge [17] (referred to simply as UNR), with some additional developments due to Umrigar and Filippi [37]. Throughout this section we assume for simplicity that the trial wave function is real, though CASINO is perfectly capable of dealing with complex wave functions.

13.1 Imaginary-time propagation

Let $\mathbf{R}$ be a point in the configuration space of an N-electron system. The importance-sampled DMC method propagates the distribution $f(\mathbf{R},t) = \Psi(\mathbf{R})\Phi(\mathbf{R},t)$ in imaginary time t, where Ψ is the trial wave function and Φ is the DMC wave function. The importance-sampled imaginary-time Schrödinger equation may be written in integral form,

$$f(\mathbf{R},t) = \int G(\mathbf{R} \leftarrow \mathbf{R}', t-t')f(\mathbf{R}',t')\, d\mathbf{R}' \;, \tag{43}$$

where the Green's function $G(\mathbf{R} \leftarrow \mathbf{R}', t-t')$ satisfies the initial condition

$$G(\mathbf{R} \leftarrow \mathbf{R}', 0) = \delta(\mathbf{R} - \mathbf{R}') \;. \tag{44}$$

The Green's function used in DMC is an approximation to the exact form which is accurate for short time steps, $\tau = t - t'$ (**dtdmc**),

$$G_{\mathrm{DMC}}(\mathbf{R} \leftarrow \mathbf{R}', \tau) = G_{\mathrm{D}}(\mathbf{R} \leftarrow \mathbf{R}', \tau)\, G_{\mathrm{B}}(\mathbf{R} \leftarrow \mathbf{R}', \tau) \;, \tag{45}$$

where

$$G_{\mathrm{D}}(\mathbf{R} \leftarrow \mathbf{R}', \tau) = \frac{1}{(2\pi\tau)^{3N/2}} \exp\left(-\frac{(\mathbf{R} - \mathbf{R}' - \tau\mathbf{V}(\mathbf{R}'))^2}{2\tau}\right) \tag{46}$$

[17]It is better to use the Slater wave function to compute the first-level acceptance probability because p_1 is much lower (in general) than the corresponding acceptance probability for the Jastrow part (p_2); hence we are less likely to need to compute the second level acceptance probability than if the situation were reversed.

[18]2000 steps are required for configuration-by-configuration VMC.

is the drift-diffusion Green's function and

$$G_{\rm B}(\mathbf{R} \leftarrow \mathbf{R}', \tau) = \exp\left(-\frac{\tau}{2}[E_{\rm L}(\mathbf{R}) + E_{\rm L}(\mathbf{R}') - 2E_{\rm T}]\right) \tag{47}$$

is the branching Green's function. $E_{\rm T}$ is the reference energy, which acts as a renormalization factor (see Sec. 13.4). $E_{\rm L}$ is the local energy,

$$E_{\rm L}(\mathbf{R}) = \Psi^{-1}\hat{H}\Psi, \tag{48}$$

where $\hat{H}$ is the Hamiltonian and $\mathbf{V}$ is the drift vector,

$$\mathbf{V}(\mathbf{R}) = \Psi^{-1}\nabla\Psi. \tag{49}$$

Note that $\mathbf{V} = (\mathbf{v}_1, \ldots, \mathbf{v}_N)$, where $\mathbf{v}_i$ is the drift vector of electron i.

13.2 The ensemble of configurations

The f distribution is represented by an ensemble of electron configurations, which are propagated according to rules derived from the Green's function of Eq. (45). $G_{\rm D}$ represents a drift-diffusion process while $G_{\rm B}$ represents a branching process. The branching process leads to fluctuations in the population of configurations and/or fluctuations in their weights.

We will introduce labels for the different configurations α present at each time step m. From now on $\mathbf{R}$ represents the electron coordinates of a particular configuration in the ensemble and i labels a particular electron.

In the CASINO implementation of DMC, electrons can moved one at a time (electron-by-electron, **dmc_method**= 1, default) or all at once (configuration-by-configuration, **dmc_method**= 2). The former is much more efficient and is the standard procedure for large systems, but most of the algorithms described in the literature are for moving all electrons at once.

13.3 Drift and diffusion

We now discuss the practical implementation of the drift-diffusion process using the *electron-by-electron* algorithm in which electrons are moved one at a time, as used in CASINO.

To implement the drift-diffusion step, each electron i in each configuration α is moved from $\mathbf{r}'_i(\alpha)$ to $\mathbf{r}_i(\alpha)$ in turn according to

$$\mathbf{r}_i = \mathbf{r}'_i + \chi + \tau\mathbf{v}_i(\mathbf{r}_1, \ldots, \mathbf{r}_{i-1}, \mathbf{r}'_i, \ldots, \mathbf{r}'_N), \tag{50}$$

where χ is a three-dimensional vector of normally distributed numbers with variance τ and zero mean. $\mathbf{v}_i(\mathbf{R})$ denotes those components of the total drift vector $\mathbf{V}(\mathbf{R})$ due to electron i.

Hence each electron i is moved from $\mathbf{r}'_i$ to $\mathbf{r}_i$ with a transition probability density of

$$t_i(\mathbf{r}_1, \ldots, \mathbf{r}_{i-1}, \mathbf{r}_i \leftarrow \mathbf{r}'_i, \mathbf{r}'_{i+1}, \ldots, \mathbf{r}'_N) = \frac{1}{(2\pi\tau)^{3/2}} \exp\left(\frac{(\mathbf{r}_i - \mathbf{r}'_i - \tau\mathbf{v}_i(\mathbf{r}_1, \ldots, \mathbf{r}_{i-1}, \mathbf{r}'_i, \ldots, \mathbf{r}'_N))^2}{2\tau}\right). \tag{51}$$

For a complete sweep through the set of electrons, the transition probability density for a move from $\mathbf{R}' = (\mathbf{r}'_1, \ldots, \mathbf{r}'_N)$ to $\mathbf{R} = (\mathbf{r}_1, \ldots, \mathbf{r}_N)$ is simply the probability that each electron i moves from $\mathbf{r}'_i$ to $\mathbf{r}_i$. So the transition probability density for the configuration move is

$$T(\mathbf{R} \leftarrow \mathbf{R}') = \prod_{i=1}^{N} t_i(\mathbf{r}_1, \ldots, \mathbf{r}_{i-1}, \mathbf{r}_i \leftarrow \mathbf{r}'_i, \mathbf{r}'_{i+1}, \ldots, \mathbf{r}'_N). \tag{52}$$

In the limit of small time steps, the drift velocity $\mathbf{V}$ is constant over the (small) configuration move. Evaluating the product in this case, we find that the transition probability density is

$$T(\mathbf{R} \leftarrow \mathbf{R}') = G_D(\mathbf{R} \leftarrow \mathbf{R}', \tau), \tag{53}$$

so that the drift-diffusion process is described by the drift-diffusion Green's function G_D.

At finite time steps, however, the approximation that the drift velocity is constant leads to the violation of the detailed balance condition.

We may enforce the detailed balance condition on the DMC Green's function by means of a Metropolis-style accept/reject step introduced by Ceperley *et al.* [38]. This has been shown to greatly reduce time-step errors [39]. The move of the ith electron of a configuration is accepted with probability

$$\begin{aligned}
& \min\left\{1, \frac{t_i(\mathbf{r}_1,\ldots,\mathbf{r}_{i-1},\mathbf{r}_i' \leftarrow \mathbf{r}_i,\mathbf{r}_{i+1}',\ldots,\mathbf{r}_N')\Psi^2(\mathbf{r}_1,\ldots,\mathbf{r}_i,\mathbf{r}_{i+1}',\ldots,\mathbf{r}_N')}{t_i(\mathbf{r}_1,\ldots,\mathbf{r}_{i-1},\mathbf{r}_i \leftarrow \mathbf{r}_i',\mathbf{r}_{i+1}',\ldots,\mathbf{r}_N')\Psi^2(\mathbf{r}_1,\ldots,\mathbf{r}_{i-1},\mathbf{r}_i',\ldots,\mathbf{r}_N')}\right\} \\
= & \min\Bigg\{1, \exp\Bigg[\Big(\mathbf{r}_i' - \mathbf{r}_i + \frac{\tau}{2}[\mathbf{v}_i(\mathbf{r}_1,\ldots,\mathbf{r}_{i-1},\mathbf{r}_i',\ldots,\mathbf{r}_N') - \mathbf{v}_i(\mathbf{r}_1,\ldots,\mathbf{r}_i,\mathbf{r}_{i+1}',\ldots,\mathbf{r}_N')]\Big) \\
& \qquad \cdot \big(\mathbf{v}_i(\mathbf{r}_1,\ldots,\mathbf{r}_{i-1},\mathbf{r}_i',\ldots,\mathbf{r}_N') + \mathbf{v}_i(\mathbf{r}_1,\ldots,\mathbf{r}_i,\mathbf{r}_{i+1}',\ldots,\mathbf{r}_N')\big)\Bigg] \\
& \qquad \times \frac{\Psi^2(\mathbf{r}_1,\ldots,\mathbf{r}_i,\mathbf{r}_{i+1}',\ldots,\mathbf{r}_N')}{\Psi^2(\mathbf{r}_1,\ldots,\mathbf{r}_{i-1},\mathbf{r}_i',\ldots,\mathbf{r}_N')}\Bigg\} \\
\equiv & \; a_i(\mathbf{r}_1,\ldots,\mathbf{r}_{i-1},\mathbf{r}_i \leftarrow \mathbf{r}_i',\mathbf{r}_{i+1}',\ldots,\mathbf{r}_N'). \qquad (54)
\end{aligned}$$

This leads to the single-electron detailed balance condition

$$\begin{aligned}
& s_i(\mathbf{r}_1,\ldots,\mathbf{r}_{i-1},\mathbf{r}_i \leftarrow \mathbf{r}_i',\mathbf{r}_{i+1}',\ldots,\mathbf{r}_N')\Psi^2(\mathbf{r}_1,\ldots,\mathbf{r}_{i-1},\mathbf{r}_i',\ldots,\mathbf{r}_N') \\
= & \; s_i(\mathbf{r}_1,\ldots,\mathbf{r}_{i-1},\mathbf{r}_i' \leftarrow \mathbf{r}_i,\mathbf{r}_{i+1}',\ldots,\mathbf{r}_N')\Psi^2(\mathbf{r}_1,\ldots,\mathbf{r}_i,\mathbf{r}_{i+1}',\ldots,\mathbf{r}_N'),
\end{aligned} \qquad (55)$$

where

$$\begin{aligned}
s_i(\mathbf{r}_1,\ldots,\mathbf{r}_{i-1},\mathbf{r}_i \leftarrow \mathbf{r}_i',\mathbf{r}_{i+1}',\ldots,\mathbf{r}_N') = & \; a_i(\mathbf{r}_1,\ldots,\mathbf{r}_{i-1},\mathbf{r}_i \leftarrow \mathbf{r}_i',\mathbf{r}_{i+1}',\ldots,\mathbf{r}_N') \\
& \times t_i(\mathbf{r}_1,\ldots,\mathbf{r}_{i-1},\mathbf{r}_i \leftarrow \mathbf{r}_i',\mathbf{r}_{i+1}',\ldots,\mathbf{r}_N'), \qquad (56)
\end{aligned}$$

is the effective single-electron transition probability density, once the accept/reject step has been introduced.

Hence we find that the effective transition probability density for the entire configuration move satisfies

$$\begin{aligned}
S(\mathbf{R} \leftarrow \mathbf{R}') & = \prod_{i=1}^{N} s_i(\mathbf{r}_1,\ldots,\mathbf{r}_{i-1},\mathbf{r}_i \leftarrow \mathbf{r}_i',\mathbf{r}_{i+1}',\ldots,\mathbf{r}_N') \\
& = \prod_{i=1}^{N} s_i(\mathbf{r}_1,\ldots,\mathbf{r}_{i-1},\mathbf{r}_i' \leftarrow \mathbf{r}_i,\mathbf{r}_{i+1}',\ldots,\mathbf{r}_N')\frac{\Psi^2(\mathbf{r}_1,\ldots,\mathbf{r}_i,\mathbf{r}_{i+1}',\ldots,\mathbf{r}_N')}{\Psi^2(\mathbf{r}_1,\ldots,\mathbf{r}_{i-1},\mathbf{r}_i',\ldots,\mathbf{r}_N')} \\
& = S(\mathbf{R}' \leftarrow \mathbf{R})\frac{\Psi^2(\mathbf{R})}{\Psi^2(\mathbf{R}')}. \qquad (57)
\end{aligned}$$

And so detailed balance in configuration space is satisfied.

It is more efficient to use an electron-by-electron algorithm than the (perhaps more straightforward) *configuration-by-configuration* algorithm in which moves of entire configurations are proposed and then accepted or rejected. This is because, for a given time step, a configuration will travel further on average if the accept/reject step is carried out for each electron in turn. For example, it is clear that it is very unlikely for a configuration not to be moved at all in an electron-by-electron algorithm. Hence the sampling of configuration space in an electron-by-electron algorithm is more efficient.

After each move of each electron we check whether the configuration has crossed the nodal surface (by checking the sign of the Slater part of the trial wave function). If it has then the move is rejected. This has been found to be the least-biased method of imposing the fixed-node approximation [11].

13.4 Branching and population control

The branching Green's function can be implemented by altering the population of configurations and/or their weights. At the start of the calculation one chooses a target population, M_0, and the actual population $M_{\rm tot}(m)$ [see Eq. (61)] is controlled so that it does not deviate too much from M_0. The population control is principally exerted by altering the reference energy, $E_{\rm T}(m)$. Large changes in $E_{\rm T}$ can lead to a bias and therefore it is varied smoothly over the simulation.

For each move of all the electrons in configuration α the branching factor is calculated as:

$$M_b(\alpha,m) = \exp\left[\left(-\frac{1}{2}\left\{S(\mathbf{R}_{\alpha,m}) + S(\mathbf{R}'_{\alpha,m})\right\} + E_{\rm T}(m)\right)\tau_{\rm eff}(\alpha,m)\right] \tag{58}$$

where $\tau_{\rm eff}$ is the effective time step for configuration α at time step m (see Sec. 13.5), S is the local energy (we denote it by S because it is usually a modified version of $E_{\rm L}$; see Sec. 13.5). Unless weighted DMC is used (i.e., unless **lwdmc**=`T`), the number of copies of this configuration that continue to the next time step is given by:

$$M(\alpha,m) = {\rm INT}\{\eta + M_b(\alpha,m)\}, \tag{59}$$

where η is a random number drawn from a uniform distribution on the interval [0,1].

In weighted DMC, each configuration carries a weight that is simply multiplied by $M_b(\alpha,m)$ after each move; only if the weight of a configuration goes outside certain bounds (above **wdmcmax** or below **wdmcmin**) is it allowed to branch or be combined with another configuration.

Throughout this section we denote the best estimate of the ground-state energy at time step α by $E_{\rm best}(m)$. At the start of a DMC run we set $E_{\rm best}(0) = E_{\rm T}(0) = E_{\rm V}$, where $E_{\rm V}$ is the average local energy of the initial configurations. During equilibration $E_{\rm best}$ is updated after each iteration as the average local energy over the previous **ebest_av_window** moves. During accumulation $E_{\rm best}$ is set equal to the current value of the mixed estimator of the energy, given by Eq. (79), with $\hat{A} = E_{\rm L}(\alpha,m)$. The algorithms differ because we wish to discard data from the start of the simulation during equilibration. $E_{\rm T}$ is updated after every iteration as

$$E_{\rm T}(m+1) = E_{\rm best}(m) - \frac{g^{-1}}{\tau_{\rm EFF}(m)}\log\left(\frac{M_{\rm tot}(m)}{M_0}\right), \tag{60}$$

where $g^{-1} = \min\{1,\tau c_{E_{\rm T}}\}$, $c_{E_{\rm T}}$ is a constant that can be set in the `input` file (**cerefdmc**), although the default value of 1 a.u. rarely needs to be altered, M_0 =**dmc_target_weight** is the target number of configurations (in our implementation it is allowed to take non-integer values) and

$$M_{\rm tot}(m) = \sum_{\alpha=1}^{N_{\rm config}(m)} w_\alpha(m), \tag{61}$$

where $N_{\rm config}(m)$ is the number of configurations and w_α is the weight of configuration α. Note that $E_{\rm best}(m)$ is the best energy at time step m while $E_{\rm T}(m+1)$ is the trial energy to be used in the next time step. $\tau_{\rm EFF}$ is the current best estimate over all configurations and time steps of the mean effective time step, calculated using Eq. (79) with $\hat{A} = \tau_{\rm eff}(\alpha,m)$. Note that g is the imaginary time scale (in units of time steps) over which the population attempts to return to M_0, while $c_{E_{\rm T}}$ is the inverse of the amount of imaginary time in a.u. over which the population attempts to return to M_0.

13.5 Modifications to the Green's Function

13.5.1 The effective time step

Time-step errors can be reduced and the stability of the DMC algorithm improved by modifying the Green's function. An important modification is to introduce an effective time step, $\tau_{\rm eff}$, into the branching factor [39]. When the accept/reject step is included, the mean distance diffused by each electron each move (which should go as the square root of the time step) is reduced because some moves are rejected. When calculating branching factors it is therefore more accurate to use a time step appropriate for the actual distance diffused. Umrigar and Filippi [37] have suggested using an effective time step for each configuration at each time step. The effective time step is given by

$$\tau_{\rm eff}(\alpha,m) = \tau\frac{\sum_i p_i \Delta r_{{\rm d},i}^2}{\sum_i \Delta r_{{\rm d},i}^2}\ , \tag{62}$$

where the averages are over all attempted moves of the electrons i in configuration α at time step m. The $\Delta r_{{\rm d},i}$ are the diffusive displacements [i.e., the distances travelled by the electrons *without* the drift-displacement: see Eq. (50)] and p_i is the acceptance probability of the electron move [see Eq. (54)]. The values averaged over the current run are written in the output file.

We calculate $\tau_{\rm EFF}(m)$ using Eq. (79) with $\hat{A} \equiv \tau_{\rm eff}(\alpha,m)$.

13.5.2 Drift-vector and local-energy limiting

The drift vector diverges at the nodal surface and a configuration which approaches a node can exhibit a very large drift, resulting in an excessively large move in the configuration space. One can improve the Green's function by cutting off the drift vector when its magnitude becomes large. The total drift vector is defined in Eq. (49).

We use the smoothly cut-off drift vector suggested by UNR [17]. For each electron with drift vector $\mathbf{v}_i$, we define the smoothly cut-off drift vector:

$$\tilde{\mathbf{v}}_i = \frac{-1+\sqrt{1+2a|\mathbf{v}_i|^2\tau}}{a|\mathbf{v}_i|^2\tau}\mathbf{v}_i \ , \tag{63}$$

where a is a constant that can be chosen to minimize the bias. The value of $a =$ **alimit** can be entered by the user if **nucleus_gf_mods** is set to `F` (the default is $a = 1$); otherwise a will be calculated as described in Sec. 13.6.

In the UNR [17] scheme the modified local energy, $S(\alpha, m)$, is given by

$$S(\alpha,m) = E_{\rm best}(m) - [E_{\rm best}(m) - E_{\rm L}(\alpha,m)]\frac{|\tilde{\mathbf{V}}|}{|\mathbf{V}|} \ , \tag{64}$$

where $\tilde{\mathbf{V}}(\alpha,m) = (\tilde{\mathbf{v}}_1,\ldots,\tilde{\mathbf{v}}_N)$. Note that we define S slightly differently from Ref. [17]. S is used only in the branching factor. When evaluating the average energy, we sum the unlimited local energies, $E_{\rm L}$. In practice, even though the local energy is calculated at the end of the configuration move, the limiting scheme of Eq. (64) is applied to the single-electron drift velocities. The ratio of drift velocities is therefore calculated as

$$\frac{\bar{V}(\mathbf{R})}{V(\mathbf{R})} = \frac{\left(\bar{v}_1^2(\mathbf{R}) + \cdots + \bar{v}_N^2(\mathbf{R})\right)^{1/2}}{V(\mathbf{R})}. \tag{65}$$

Finally, a very simple option is simply to cut off the local energy drift vector when their magnitudes becomes large using the method of Depasquale *et al.* [21],

$$\begin{aligned} S(\mathbf{R}) &= E_{\rm V} + {\rm sign}[2/\sqrt{\tau}, E_{\rm L}(\mathbf{R}) - E_{\rm V}] \quad \text{for } |E_{\rm L}(\mathbf{R}) - E_{\rm V}| > 2/\sqrt{\tau} \\ \tilde{\mathbf{v}}_i(\mathbf{R}) &= {\rm sign}[1/\tau, \mathbf{v}_i] \qquad\qquad\qquad\qquad \text{for } |\mathbf{v}_i| > 1/\tau \ . \end{aligned} \tag{66}$$

In the paper of Depasquale *et al.* [21] they recommend using the variational energy for $E_{\rm V}$, but we use the best estimate of the energy. We prefer the UNR scheme.

The limited single-electron drift velocities are calculated at the same time as the local energy of the configuration, after all of the electron positions in the configuration have been updated.

13.6 Modifications to the DMC Green's function at bare nuclei

13.6.1 Modifications to the limiting of the drift velocity

The limiting of the drift velocity is intended to remove the divergence at the nodal surface; interference with the cusps at bare nuclei is an undesirable side-effect, which may introduce bias. In order to distinguish between nodes and nuclei, UNR make the a-parameter in their limiting scheme dependent on electron position. Immediately before the limited drift velocity of an electron at $\mathbf{r}'$ is calculated, a is evaluated as

$$a(\mathbf{r}') = \frac{1}{2}\left(1 + \hat{\mathbf{v}}\cdot\mathbf{e}_z\right) + \frac{Z^2z^2}{10(4+Z^2z^2)}, \tag{67}$$

where $\hat{\mathbf{v}}$ is the unit vector in the direction of the unlimited drift velocity, $\mathbf{e}_z$ is the unit vector from the closest bare nucleus to the electron, z is the distance of the electron from the nucleus and Z is the atomic number. This formula makes a small (and hence the limiting weak) if the electron is both close to the nearest nucleus and drifting towards it.

13.6.2 Preventing electrons from overshooting nuclei

In its immediate vicinity, the single-electron drift velocity is always directed towards a bare nucleus. Therefore, drifting particles should never cross the nucleus; rather, they should end up on top of it.

In order to impose this condition at finite time steps, we work in cylindrical polar coordinates with the z-axis lying along the line from the nucleus to the electron. Let the position of the closest nucleus be $\mathbf{R}_Z$.

The position of the electron relative to the nucleus is

$$\mathbf{r}' - \mathbf{R}_Z = z'\mathbf{e}_z, \tag{68}$$

while the limited drift velocity can be resolved as

$$\bar{\mathbf{v}} = \bar{v}_z\mathbf{e}_z + \bar{v}_\rho\mathbf{e}_\rho, \tag{69}$$

where $\mathbf{e}_\rho$ is a unit vector orthogonal to $\mathbf{e}_z$.

The new z-coordinate after drifting for one time step is

$$z'' = \max\{z' + \bar{v}_z\tau, 0\}, \tag{70}$$

which cannot lie beyond the nucleus.

The drift in the radial direction over one time step is

$$\rho'' = \frac{2\bar{v}_\rho\tau z''}{z' + z''}. \tag{71}$$

The new radial coordinate is approximately $\bar{v}_\rho\tau$ when far from the nucleus, but it is forced to go to zero as the nucleus is approached. Hence, if the electron attempts to overshoot the nucleus, it will end up on top of it. So time-step errors caused by drifting across nuclei are eliminated.

Let the electron position at the end of the drift process be $\mathbf{r}'' = z''\mathbf{e}_z + \rho''\mathbf{e}_\rho$.

13.6.3 Diffusion close to a bare nucleus

Close to a nucleus, f is proportional to the square of the hydrogenic 1s orbital (assuming the trial wave function has the correct behaviour). This cusp cannot be reproduced by Gaussian diffusion at finite time steps. In fact, starting from the nucleus, we would like our electron to take a random step $\mathbf{w}$ distributed according to $\exp(-2Z|\mathbf{w}|)$.

However, we only want to diffuse in this fashion when the electron is likely to cross the nucleus. Let Π be the plane with normal $\mathbf{e}_z$ that contains the nucleus. For the usual Gaussian diffusion process, the probability that an electron drifts (assuming that nuclear overshoot is permitted) and diffuses across Π, is

$$\tilde{q} = 1 - \tilde{p} = \frac{1}{2}\,\mathrm{erfc}\left(\frac{z + \bar{v}_z\tau}{\sqrt{2\tau}}\right). \tag{72}$$

So, with probability $\tilde{p}$, we sample $\mathbf{w}$ from

$$g_1(\mathbf{w}) = (2\pi\tau)^{-3/2}\exp\left(-\frac{|\mathbf{w}|^2}{2\tau}\right), \tag{73}$$

and set the new electron position to be $\mathbf{r} = \mathbf{r}'' + \mathbf{w}$; otherwise, we sample[19] $\mathbf{w}$ from

$$g_2(\mathbf{w}) = \frac{\zeta^3}{\pi}\exp(-2\zeta|\mathbf{w}|), \tag{74}$$

and set $\mathbf{r} = \mathbf{R}_Z + \mathbf{w}$. We have defined ζ by

$$\zeta = \sqrt{Z^2 + \frac{1}{\tau}}, \tag{75}$$

which reduces to Z for large time steps, giving the desired cusp; however, this choice of ζ causes the second moments of g_1 and g_2 to be equal to $\mathcal{O}(\tau)$. Hence the Green's function remains correct to $\mathcal{O}(\tau)$.

The single-electron Green's function for the move from $\mathbf{r}'$ to $\mathbf{r}$ is given by

$$g(\mathbf{r} \leftarrow \mathbf{r}') = \tilde{p}g_1(\mathbf{r} - \mathbf{r}'') + \tilde{q}g_2(\mathbf{r} - \mathbf{R}_Z). \tag{76}$$

In order to calculate the Green's function for the reverse move, need to perform all of the steps above (apart from the random diffusion), starting at point $\mathbf{r}$ and ending up at $\mathbf{r}'$.

[19] In order to sample $\mathbf{w}$ from $g_2(\mathbf{w})$, we sample the cosine of the polar angle uniformly on $[-1, 1]$, the azimuthal angle uniformly on $[0, 2\pi]$ and the magnitude w from $4\zeta^3w^2\exp(-2\zeta w)$. This is achieved by sampling r_1, r_2 and r_3 uniformly on $[0, 1]$ and setting $w = -\log(r_1r_2r_3)/2\zeta$; see reference [40] for further information.

13.6.4 Using the modifications in CASINO

These three modifications to the DMC Green's function are applied if the **nucleus_gf_mods** keyword is set to `T`. Note that they can only be used if bare nuclei are actually present!

13.7 Evaluating expectation values of observables

The *reference energy* $E_{\rm T}$ is varied to maintain a reasonably steady population. However, this procedure can result in a bias in the estimate of expectation values, especially for small populations. To remove this one can evaluate expectation values using the method of UNR [17].

Using the label m for time step, Eq. (45) becomes

$$f(\mathbf{R}, m) = \int G_{\rm DMC}(\mathbf{R} \leftarrow \mathbf{R}', \tau) f(\mathbf{R}', m-1)\, d\mathbf{R}' \; . \tag{77}$$

Clearly, in the absence of the accept/reject step, the effect of including the (time-step-dependent) reference energy $E_{\rm T}(m)$ in $G_{\rm DMC}$ can be 'undone' by multiplying the right-hand-side of Eq. (77) by $\exp[-\tau E_{\rm T}(m)]$. In a similar fashion the effect of including the reference energy from the previous time step can be eliminated by multiplying by $\exp[-\tau E_{\rm T}(m-1)]$. When the accept/reject step is present, we can approximately undo the effect of the reference energy by using our best estimate of the effective time step $\tau_{\rm EFF}$ (See Sec. 13.5) in the 'undoing' factors.

Continuing this process, we may eliminate the effect of changing the reference energy from $f(m)$ by multiplying it by

$$\Pi(m) = \prod_{m'=0} \exp\left[-\tau_{\rm EFF} E_{\rm T}(m-m')\right] \; , \tag{78}$$

where in principle the product runs over all previous time steps. In practice it is sufficient to include $T_{\rm p}$ (=**tpdmc**) terms in the product, provided that $T_{\rm p}$ is greater than the number of iterations over which the DMC data are correlated by fluctuations in the reference energy: $T_{\rm p} = {\rm NINT}(10/\tau)$ is generally sufficient (the estimation of correlation periods is discussed in Sec. 24.3). Let $\Pi(m, T_{\rm p}) = \prod_{m'=0}^{T_{\rm p}} \exp[-\tau_{\rm EFF}(m) E_{\rm T}(m-m')]$. Then the mixed estimator of the expectation value of a (local) operator $\hat{A}$ may be written as:

$$\begin{aligned} \frac{\langle\Psi|\hat{A}|\Phi\rangle}{\langle\Psi|\Phi\rangle} &= \frac{\int \Psi(\mathbf{R})\hat{A}(\mathbf{R})\Phi(\mathbf{R})\, d\mathbf{R}}{\int \Psi(\mathbf{R})\Phi(\mathbf{R})\, d\mathbf{R}} \\ &\approx \frac{\sum_{m'=1}^{m} \Pi(m', T_{\rm p}) \sum_{\alpha=1}^{N_{\rm config}(m')} w_\alpha(m') \hat{A}(\alpha, m')}{\sum_{m'=1}^{m} \Pi(m', T_{\rm p}) \sum_{\alpha=1}^{N_{\rm config}(m')} w_\alpha(m')}, \end{aligned} \tag{79}$$

where $w_\alpha(m')$ is the weight of configuration α at the end of time step m'. (For unweighted DMC, w_α is simply the branching factor.) Note that if we choose $\hat{A}(\alpha, m) = \Psi^{-1}(\mathbf{R}_{\alpha,m})\hat{H}\Psi(\mathbf{R}_{\alpha,m})$ then Eq. (79) gives us our mixed estimator of the ground state energy at time step m. This is used as our 'best estimate' of the ground state energy, $E_{\rm best}$ (see Sec. 13.4).

The terms in the Π weights are exponential functions of the reference energy; hence the Π weights are potentially very large (or small). However, it can be seen that any constant contribution to the reference energy will cancel in Eq. (79). Therefore, in practice, we evaluate the Π-weights as:

$$\Pi(m, T_{\rm p}) = \prod_{m'=0}^{T_{\rm p}-1} \exp\left[\tau_{\rm EFF}(1) E_{\rm V} - \tau_{\rm EFF}(m) E_{\rm T}(m-m')\right], \tag{80}$$

where $E_{\rm V}$ is the variational energy. This is necessary in order to avoid floating point errors.

Note that in practice population-control bias is usually negligible if more than a few hundred configurations are used, and that the Π weights are an additional source of statistical noise. For this reason we do not usually use the UNR Π-weighting scheme [i.e., $\Pi(m, T_{\rm p}) = 1$ for all m in all formulae involving the Π weights]. The scheme is not used when **tpdmc** is set to 0, which is the default.

13.8 Growth estimator of the energy

The total weight of a DMC simulation at time $t = m\tau$ is given by:

$$W(t) \equiv \int f(\mathbf{R},t)\,d\mathbf{R} \approx \sum_{\alpha=1}^{N_{\rm config}(m)} w_\alpha(m) \equiv M_{\rm tot}(m). \tag{81}$$

Suppose the DMC simulation is equilibrated, so that $f(\mathbf{R},t) = \Psi(\mathbf{R})\phi_0(\mathbf{R})$. Then

$$\begin{aligned} W(t+\tau) &= \int\int G(\mathbf{R} \leftarrow \mathbf{R}',\tau) f(\mathbf{R}',\tau)\,d\mathbf{R}'\,d\mathbf{R} \\ &= \int\int \Psi(\mathbf{R})\langle\mathbf{R}|\exp[-\tau(\hat{H}-E_{\rm T})]|\mathbf{R}'\rangle\Psi^{-1}(\mathbf{R}')\Psi(\mathbf{R}')\phi_0(\mathbf{R}')\,d\mathbf{R}\,d\mathbf{R}' \\ &= \langle\Psi|\exp[-\tau(\hat{H}-E_{\rm T})]|\phi_0\rangle = W(t)\exp[-\tau(E_0-E_{\rm T})]. \end{aligned}$$

So

$$\begin{aligned} E_0 &= -\frac{1}{\tau}\log\left(\frac{W(t+\tau)\exp(-\tau E_{\rm T})}{W(t)}\right) \\ &\approx -\frac{1}{\tau}\log\left(\frac{\exp[-E_{\rm T}(m+1)\tau]M_{\rm tot}(m+1)}{M_{\rm tot}(m)}\right). \end{aligned} \tag{82}$$

This is the single-iteration growth estimator.

By taking the expectation value of the argument of the logarithm and using our estimate of the effective time step, we obtain a much less noisy estimate of the ground state[20]:

$$\begin{aligned} E_{\rm growth}(m) &= -\frac{1}{\tau_{\rm EFF}(m)}\log\left(\frac{\sum_{m'=1}^{m}\Pi(m',T_{\rm p})M_{\rm tot}(m')\left(\frac{\exp[-E_{\rm T}(m'+1)\tau_{\rm EFF}(m')]M_{\rm tot}(m'+1)}{M_{\rm tot}(m')}\right)}{\sum_{m'=1}^{m}\Pi(m',T_{\rm p})M_{\rm tot}(m')}\right) \\ &= -\frac{1}{\tau_{\rm EFF}(m)}\log\left(\frac{\sum_{m'=1}^{m}\Pi(m',T_{\rm p})\exp[-E_{\rm T}(m'+1)\tau_{\rm EFF}(m')]M_{\rm tot}(m'+1)}{\sum_{m'=1}^{m}\Pi(m',T_{\rm p})M_{\rm tot}(m')}\right). \end{aligned} \tag{83}$$

Equation (83) is used to evaluate the growth estimator of the energy in CASINO if the **growth_estimator** flag is set to `T` in the input file. The error bar on the growth estimator is always much larger than the error on the mixed estimator in practice, so we do not normally use the growth estimator.

13.9 Automatic block-resetting

Numerous schemes for preventing population control catastrophes due to the occurrence of 'persistent electrons' have been investigated. Of these, the one that seems to perform best in practice involves returning to an earlier point in the simulation and changing the random number sequence.

If the **dmc_trip_weight** input variable is set to a nonzero value, then a `config.backup` file will be created. This contains a copy of the `config.out` file from the beginning of the previous block. If the total weight at any given iteration exceeds **dmc_trip_weight**, then the data from `config.backup` will be read in, the last block of lines will be erased from the `dmc.hist` file and the random number generator will be called a few times so that the configurations go off on new random walks, hopefully avoiding the catastrophe that led to the population explosion in the first place. (If **dmc_trip_weight** is exceeded in the first or second blocks, then the initial `config.in` file will be read in instead of `config.backup`.)

Note that if the accumulation of expectation values other than the energy is flagged in input, and the calculation is in a DMC statistics accumulation phase, then the resulting `expval.data` file will be subject to the same treatment (through a saved '`expal.backup`' file).

If a block has to be reset more than **max_rec_attempts** times then the program will abort with an error.

Great care should be taken when choosing a value for **dmc_trip_weight**. It should be sufficiently large that it cannot interfere with normal population fluctuations: this would lead to population control

[20]Note that by bringing the average inside the logarithm we introduce a small bias into $E_{\rm growth}$.

biasing. (Note that the population often grows rapidly at the start of equilibration: again, it must be ensured that automatic block resetting does not interfere with this natural process.) On the other hand, **dmc_trip_weight** should be sufficiently small that persistence is dealt with quickly and that there is insufficient time for a population of configurations containing a persistent electron to stabilize. Choosing larger block lengths (by decreasing the value of **dmc_equil_nblock** and **dmc_stats_nblock**) allows the program to return to an earlier point in the simulation, increasing the likelihood that the catastrophe will be avoided.

Population-control catastrophes should not occur under normal circumstances. The following have been found to lead to catastrophic behaviour: (i) the use of a trial wave function that does not satisfy the electron–nucleus cusp conditions, e.g., when a non-cusp-corrected Gaussian basis is used; (ii) the use of certain nonlocal pseudopotentials in the absence of the T-move scheme [25, 26] (do not set **use_tmove** to `F` in order to use the T-move scheme); (iii) the severe truncation of localized orbitals, especially when a smooth truncation scheme is used; (iv) the use of an inadequate basis set to represent the orbitals; (v) pathological trial wave functions, resulting from optimizations that have gone awry (use **jastrow_plot** to examine the behaviour of $u(r_{ij})$: it should increase monotonically to 0). All of these circumstances should be avoided if possible.

Note finally that, in large systems, the likelihood of population explosions may be reduced if you use (unreweighted) variance minimization rather than energy minimization to optimize the wave function. The reason for this is related to the fact that energy minimisation doesn't care much about what happens near nodes, since those regions do not contribute much to the energy expectation value. However, the divergent local energies there make a big difference to the stability of the DMC algorithm. If you're using backflow then you have to use energy minimisation, but then you're unlikely to be looking at a big system.

13.10 Determinant locality approximation and T moves

13.10.1 Nonlocal pseudopotentials in DMC

Nonlocal pseudopotentials are problematic in DMC calculations, because the DMC wave function is not directly accessible. The standard approach for dealing with nonlocal pseudopotentials is the so-called locality approximation [41], in which the pseudopotential is localised by applying it to the trial wave function and then dividing by the trial wave function. However this replacement of a term in the Hamiltonian is problematic because it is a nonvariational approximation. Furthermore, the localised pseudopotential diverges at the nodes of the trial wave function, leading to instabilities in the branching DMC algorithm. These problems have been successfully addressed by Casula and co-workers [25, 26], who introduced the so-called T-move scheme to restore the variational principle and eliminate the instabilities.

A remaining issue is that the localised pseudopotential depends on the trial wave function, so that different DMC programs may obtain different results, and there is a systematic tendency to find lower DMC energies for systems for which a variationally better trial wave function can be found. This issue was addressed by Zen and co-workers [24], who introduced the so called determinant locality approximation (DLA), in which the nonlocal pseudopotential is localised by a well-defined Slater-determinant wave function, which is independent of QMC wave-function optimisation.

We may combine the T-move scheme and the DLA, retaining the advantages of both methods.

13.10.2 Partial localisation by a Slater determinant

Definitions Let $\hat{V}$ be the nonlocal pseudopotential operator.

Let the importance-sampled matrix elements of the pseudopotential operator be $V_{\mathbf{R},\mathbf{R}'} = \Psi(\mathbf{R})\langle\mathbf{R}|\hat{V}|\mathbf{R}'\rangle\Psi^{-1}(\mathbf{R}')$, where $\{|\mathbf{R}\rangle\}$ is the position basis and $\Psi(\mathbf{R}) = \langle\mathbf{R}|\Psi\rangle$ is the trial wave function.

Let $V^+_{\mathbf{R},\mathbf{R}'} = \max\{V_{\mathbf{R},\mathbf{R}'}, 0\}$ and $V^-_{\mathbf{R},\mathbf{R}'} = \min\{V_{\mathbf{R},\mathbf{R}'}, 0\}$, so that $V_{\mathbf{R},\mathbf{R}'} = V^+_{\mathbf{R},\mathbf{R}'} + V^-_{\mathbf{R},\mathbf{R}'}$.

The factor in the importance-sampled DMC Green's function due to the nonlocal pseudopotential is

$$\begin{aligned} G_{\rm NL}(\mathbf{R} \leftarrow \mathbf{R}', \tau) &= \Psi(\mathbf{R})\langle\mathbf{R}|\exp(-\tau\hat{V})|\mathbf{R}'\rangle\Psi^{-1}(\mathbf{R}') & (84) \\ &= \delta_{\mathbf{R},\mathbf{R}'} - \tau V_{\mathbf{R},\mathbf{R}'} + \mathcal{O}(\tau^2) & (85) \end{aligned}$$

$$= \frac{\delta_{\mathbf{R},\mathbf{R}'} - \tau V^-_{\mathbf{R},\mathbf{R}'} - \tau V^+_{\mathbf{R},\mathbf{R}'}}{1 - \tau \sum_{\mathbf{R}''} V^-_{\mathbf{R}'',\mathbf{R}'}} \exp\left(-\tau \sum_{\mathbf{R}''} V^-_{\mathbf{R}'',\mathbf{R}'}\right) + \mathcal{O}(\tau^2), \quad (86)$$

where we have assumed that space is discretised so that we sum over positions and the overlap of spatial basis vectors is a Kronecker delta rather than a Dirac delta function.

Let $S(\mathbf{R}) = \langle \mathbf{R}|S\rangle$ be a Slater determinant wave function. In the DLA, S is used to localise $\hat{V}$. Here, we instead make the T-move determinant locality approximation (TMDLA) for the positive part of the importance-sampled pseudopotential matrix elements:

$$V^+_{\mathbf{R},\mathbf{R}'} \approx \delta_{\mathbf{R},\mathbf{R}'} \sum_{\mathbf{R}''} \max\left\{0, S(\mathbf{R}'')\langle \mathbf{R}''|\hat{V}|\mathbf{R}'\rangle S^{-1}(\mathbf{R}')\right\} \equiv \delta_{\mathbf{R},\mathbf{R}'} V^+_{\mathrm{TMDLA}}(\mathbf{R}'). \quad (87)$$

Note that the $\max\{0,\ldots\}$ in Eq. (87) is necessary in order that the TMDLA is exact for a local pseudopotential; see below. The nonlocal pseudopotential factor in the DMC Green's function becomes

$$\begin{aligned} G_{\mathrm{NL}}(\mathbf{R} \leftarrow \mathbf{R}', \tau) &\approx \frac{\delta_{\mathbf{R},\mathbf{R}'} - \tau V^-_{\mathbf{R},\mathbf{R}'}}{1 - \tau \sum_{\mathbf{R}''} V^-_{\mathbf{R}'',\mathbf{R}'}} \exp\left(-\tau V^+_{\mathrm{TMDLA}}(\mathbf{R}')\right) \exp\left(-\tau \sum_{\mathbf{R}''} V^-_{\mathbf{R}'',\mathbf{R}'}\right) + \mathcal{O}(\tau^2) \quad (88) \\ &\approx \frac{\delta_{\mathbf{R},\mathbf{R}'} - \tau V^-_{\mathbf{R},\mathbf{R}'}}{1 - \tau \sum_{\mathbf{R}''} V^-_{\mathbf{R}'',\mathbf{R}'}} \exp\left(-\tau \left(V^+_{\mathrm{TMDLA}}(\mathbf{R}') + \sum_{\mathbf{R}''} V^-_{\mathbf{R}'',\mathbf{R}'}\right)\right) + \mathcal{O}(\tau^2). \quad (89) \end{aligned}$$

Equation (89) shows how the TMDLA should be implemented in DMC. The T-move step is governed by the first factor and is exactly the same as the usual T-move scheme; however, the energy in the branching factor is modified. We have to check the sign of the importance-sampled matrix elements $V_{\mathbf{R},\mathbf{R}'}$ and evaluate Eq. (87) for positive elements, while for negative elements we evaluate the contribution to the local energy using the usual locality approximation.

By making either the DLA or the TMDLA we lose the important property that the DMC energy is exact when the trial wave function is exact.

Reduction to the usual T-move scheme Suppose that $|\Psi\rangle = |S\rangle$, i.e., the trial wave function is just a Slater determinant. Then

$$V^+_{\mathrm{TMDLA}}(\mathbf{R}') = \sum_{\mathbf{R}''} \max\left\{0, \Psi(\mathbf{R}'')\langle \mathbf{R}''|\hat{V}|\mathbf{R}'\rangle \Psi^{-1}(\mathbf{R}')\right\} = \sum_{\mathbf{R}''} V^+_{\mathbf{R}'',\mathbf{R}'}. \quad (90)$$

Hence, in this case,

$$\begin{aligned} V^+_{\mathrm{TMDLA}}(\mathbf{R}') + \sum_{\mathbf{R}''} V^-_{\mathbf{R}'',\mathbf{R}'} &= \sum_{\mathbf{R}''} \left(V^+_{\mathbf{R}'',\mathbf{R}'} + V^-_{\mathbf{R}'',\mathbf{R}'}\right) \quad (91) \\ &= \sum_{\mathbf{R}''} V_{\mathbf{R}'',\mathbf{R}'} \quad (92) \\ &= \sum_{\mathbf{R}''} \frac{\langle \mathbf{R}''|\hat{V}|\mathbf{R}'\rangle \Psi(\mathbf{R}'')}{\Psi(\mathbf{R}')} \quad (93) \\ &= \sum_{\mathbf{R}''} \frac{\langle \mathbf{R}'|\hat{V}|\mathbf{R}''\rangle\langle \mathbf{R}''|\Psi\rangle}{\Psi(\mathbf{R}')} = \frac{\langle \mathbf{R}'|\hat{V}|\Psi\rangle}{\Psi(\mathbf{R}')}, \quad (94) \end{aligned}$$

where we have assumed the nonlocal potential $\hat{V}$ to have time-reversal symmetry. Hence, in the case that $|\Psi\rangle = |S\rangle$, the nonlocal potential energy in the branching factors is just the usual locality approximation (i.e., the nonlocal potential is effectively localised by application to the trial wave function). Thus the TMDLA reduces to the usual T-move scheme.

Simplification for local pseudopotentials Now suppose that $|\Psi\rangle$ and $|S\rangle$ are different, but the pseudopotential is in fact local, so that $\langle \mathbf{R}|\hat{V}|\mathbf{R}'\rangle = V(\mathbf{R}')\delta_{\mathbf{R},\mathbf{R}'}$ for some $V(\mathbf{R})$. Then $V_{\mathbf{R},\mathbf{R}'} = V(\mathbf{R}')\delta_{\mathbf{R},\mathbf{R}'}$ and so $V^+_{\mathbf{R},\mathbf{R}'} = \delta_{\mathbf{R},\mathbf{R}'} \max\{V(\mathbf{R}'), 0\}$ and $V^-_{\mathbf{R},\mathbf{R}'} = \delta_{\mathbf{R},\mathbf{R}'} \min\{V(\mathbf{R}'), 0\}$. Hence, in this case, the T-move part of the Green's function is

$$\frac{\delta_{\mathbf{R},\mathbf{R}'} - \tau V^-_{\mathbf{R},\mathbf{R}'}}{1 - \tau \sum_{\mathbf{R}''} V^-_{\mathbf{R}'',\mathbf{R}'}} = \delta_{\mathbf{R},\mathbf{R}'}, \quad (95)$$

i.e., there are no T moves. Also,

$$V^+_{\mathrm{TMDLA}}(\mathbf{R}') = \sum_{\mathbf{R}''} \max\left\{0, S(\mathbf{R}'')V(\mathbf{R}')\delta_{\mathbf{R}'',\mathbf{R}'}S^{-1}(\mathbf{R}')\right\} = \max\{0, V(\mathbf{R}')\}. \tag{96}$$

Likewise,

$$\sum_{\mathbf{R}''} V^-_{\mathbf{R}'',\mathbf{R}'} = \min\{0, V(\mathbf{R}')\}. \tag{97}$$

Hence the energy in the branching factor is simply $V(\mathbf{R}')$, as expected. Hence the TMDLA shows the correct behaviour in the case that the pseudopotential is local.

13.10.3 Jastrow-factor independence

Partitioning of importance-sampled matrix elements Suppose the trial wave function is of Slater-Jastrow form, i.e., $\Psi(\mathbf{R}) = \exp(J(\mathbf{R}))S(\mathbf{R})$, where $\exp(J)$ is a Jastrow correlation factor. Then

$$V_{\mathbf{R},\mathbf{R}'} = S(\mathbf{R})\langle\mathbf{R}|\hat{V}|\mathbf{R}'\rangle S^{-1}(\mathbf{R}')\exp(J(\mathbf{R}) - J(\mathbf{R}')). \tag{98}$$

Hence $V_{\mathbf{R},\mathbf{R}'} \geq 0$ if and only if $S(\mathbf{R})\langle\mathbf{R}|\hat{V}|\mathbf{R}'\rangle S^{-1}(\mathbf{R}') \geq 0$, i.e., partitioning of importance-sampled matrix elements into $V^+_{\mathbf{R},\mathbf{R}'}$ and $V^-_{\mathbf{R},\mathbf{R}'}$ is independent of the Jastrow factor J.

On the other hand, if $\Psi(\mathbf{R})$ is a Slater-Jastrow-backflow or multideterminant wave function then the nodal surface of $\Psi(\mathbf{R})$ differs from that of $S(\mathbf{R})$ and hence the partitioning of importance-sampled matrix elements of the potential is different from the partitioning that would arise from the Slater wave function. More generally, the sign of the matrix elements and hence the partitioning depends on the nodal surface of the trial wave function.

Ordinary DLA The ordinary DLA [24] is

$$V_{\mathbf{R},\mathbf{R}'} \approx \delta_{\mathbf{R},\mathbf{R}'}\sum_{\mathbf{R}''} S(\mathbf{R}'')\langle\mathbf{R}''|\hat{V}|\mathbf{R}'\rangle S^{-1}(\mathbf{R}') = \delta_{\mathbf{R},\mathbf{R}'}\frac{\langle\mathbf{R}'|\hat{V}|S\rangle}{S(\mathbf{R}')}. \tag{99}$$

Within this approximation, the Green's function factor $G_{\mathrm{NL}}(\mathbf{R} \leftarrow \mathbf{R}', \tau) = \delta_{\mathbf{R},\mathbf{R}'}\exp(-\tau\langle\mathbf{R}'|\hat{V}|S\rangle S^{-1}(\mathbf{R}'))$ is independent of the trial wave function. Effectively we have replaced the nonlocal pseudopotential with an approximate local potential that depends on the Slater determinant but not the Jastrow factor.

The property of wave-function-independence is lost in the TMDLA [Eq. (87)], because the partitioning of $V_{\mathbf{R},\mathbf{R}'}$ into $V^+_{\mathbf{R},\mathbf{R}'}$ and $V^-_{\mathbf{R},\mathbf{R}'}$ depends on $|\Psi\rangle$. However, so long as $\Psi(\mathbf{R})$ is of Slater-Jastrow form, the partitioning (and hence the DMC energy) is independent of Jastrow factor.

Where $\Psi(\mathbf{R})$ is not of Slater-Jastrow form, the fixed-node DMC energy already depends on the nodal surface of $\Psi(\mathbf{R})$, and so the additional dependence of the DMC energy on the nodal surface of Ψ through the TMDLA probably does not matter.

Restoring trial-wave-function-independence to the TMDLA Dependence on the nodal surface of $\Psi(\mathbf{R})$ is all-but-inevitable for the importance-sampled Green's function of Eq. (84), except in the special case in which $\hat{V}$ is (approximated to be) local.

One could introduce a different approximation by using the Green's function factor

$$G_{\mathrm{NL}}(\mathbf{R} \leftarrow \mathbf{R}', \tau) \approx S(\mathbf{R})\langle\mathbf{R}|\exp(-\tau\hat{V})|\mathbf{R}'\rangle S^{-1}(\mathbf{R}') \tag{100}$$

for the nonlocal pseudopotential. This eliminates dependence on $|\Psi\rangle$ (and the T-move scheme could be applied just as if the trial wave function were $|S\rangle$). However, this approximation would immediately break the detailed-balance condition.

The importance-sampled DMC Green's function is

$$\begin{aligned} G_{\mathrm{DMC}}(\mathbf{R} \leftarrow \mathbf{R}') &= \Psi(\mathbf{R})\langle\mathbf{R}|\exp(-\tau\hat{H})|\mathbf{R}'\rangle\Psi^{-1}(\mathbf{R}') & (101)\\ &\approx \int \Psi(\mathbf{R})\langle\mathbf{R}|\exp(-\tau\hat{T})|\mathbf{R}''\rangle\Psi^{-1}(\mathbf{R}'')\Psi(\mathbf{R}'')\langle\mathbf{R}''|\exp(-\tau\hat{V})|\mathbf{R}'\rangle\Psi^{-1}(\mathbf{R}')\,d\mathbf{R}'' \\ &\quad + \mathcal{O}(\tau^2), & (102)\end{aligned}$$

where $\hat{T}$ is the kinetic-energy operator. If in this expression one replaces the $\Psi(\mathbf{R}'')\langle\mathbf{R}''|\exp(-\tau\hat{V})|\mathbf{R}'\rangle\Psi^{-1}(\mathbf{R}')$ factor with $S(\mathbf{R}'')\langle\mathbf{R}''|\exp(-\tau\hat{V})|\mathbf{R}'\rangle S^{-1}(\mathbf{R}')$ then (i) one makes an uncontrolled $\mathcal{O}(\tau)$ error in the Green's function and (ii) the detailed-balance condition

$$|\Psi(\mathbf{R})|^2 G_{\rm DMC}(\mathbf{R}' \leftarrow \mathbf{R}) = |\Psi(\mathbf{R}')|^2 G_{\rm DMC}(\mathbf{R} \leftarrow \mathbf{R}') \tag{103}$$

is no longer satisfied.

If the potential $\hat{V}$ is (approximated to be) local, i.e., $\hat{V} \approx \sum_{\mathbf{R}} |\mathbf{R}\rangle V(\mathbf{R})\langle\mathbf{R}|$, then

$$\langle\mathbf{R}''|\exp(-\tau\hat{V})|\mathbf{R}'\rangle = \delta_{\mathbf{R}'',\mathbf{R}'}\exp(-\tau V(\mathbf{R}')), \tag{104}$$

so that the $S(\mathbf{R}'')\,S^{-1}(\mathbf{R}')$ cancels out and hence there is no issue. This is the case in the ordinary DLA and also in the ordinary locality approximation.

One can certainly make the approximation of using $S(\mathbf{R}'')\langle\mathbf{R}''|\exp(-\tau\hat{V})|\mathbf{R}'\rangle S^{-1}(\mathbf{R}')$ with the T-move scheme and see what happens empirically. This should just correspond to what one gets if one sets **use_tmove**=`T` and **use_detla**=`T` together in CASINO before March 2023, in which case only the Slater wave function is used in the wave-function ratios for the nonlocal pseudopotential.

This approximation cannot in general be cast as a change in the Hamiltonian, because if one writes the missing factors in terms of Jastrow operators $\exp(J(\hat{\mathbf{R}}))$, they do not contain the time step and one cannot look at small-τ behaviour, i.e., one cannot easily simplify

$$\Psi(\mathbf{R}'')\langle\mathbf{R}''|\exp(-\tau\hat{V})|\mathbf{R}'\rangle\Psi^{-1}(\mathbf{R}') = S(\mathbf{R}'')\langle\mathbf{R}''|\exp(J(\hat{\mathbf{R}}))\exp(-\tau\hat{V})\exp(J(\hat{\mathbf{R}}))|\mathbf{R}'\rangle S^{-1}(\mathbf{R}') \tag{105}$$

into something of the form $S(\mathbf{R}'')\langle\mathbf{R}''|\exp(-\tau\hat{W})|\mathbf{R}'\rangle S^{-1}(\mathbf{R}')$. It is not truly analogous to the ordinary T-move scheme in which part of the nonlocal pseudopotential (with negative matrix elements) is treated exactly.

This approximation with a Slater-Jastrow wave function will not give the same result as the T-move scheme with just a Slater wave function ($|\Psi\rangle = |S\rangle$), because the latter corresponds to a well-defined modification to the Hamiltonian whereas Eq. (100) does not.

13.10.4 Mixed estimate of the energy

Equation (89) gives the expression for the DMC Green's function due to a nonlocal pseudopotential in the TMDLA. It involves a local energy contribution

$$E_{\rm L}^{\rm TMDLA}(\mathbf{R}') = V_{\rm TMDLA}^{+}(\mathbf{R}') + \sum_{\mathbf{R}''} V_{\mathbf{R}'',\mathbf{R}'}^{-} \tag{106}$$

and corresponds to a Hamiltonian $\hat{H}_{\rm TMDLA}$ with potential-energy matrix elements

$$\langle\mathbf{R}|\hat{V}_{\rm TMDLA}|\mathbf{R}'\rangle = \left\{ \begin{array}{ll} \langle\mathbf{R}|\hat{V}|\mathbf{R}'\rangle & \text{if } V_{\mathbf{R},\mathbf{R}'} \leq 0 \\ \delta_{\mathbf{R},\mathbf{R}'} V_{\rm TMDLA}^{+}(\mathbf{R}') & \text{otherwise} \end{array} \right. . \tag{107}$$

The result of propagation to large imaginary time using the DMC algorithm with $\hat{H}_{\rm TMDLA}$ is an eigenfunction $|\phi_0^{\rm TMDLA}\rangle$, with corresponding eigenvalue $E_0^{\rm TMDLA}$, which may be obtained by averaging $E_{\rm L}^{\rm TMDLA}(\mathbf{R})$ over the mixed distribution generated by the DMC algorithm.

14 Evaluation of Gaussian orbitals in the Slater wave function

CASINO can handle Gaussian basis sets up to and including angular momentum $l = 4$ (s, p, sp, d, f and g functions). Suppose we have N Gaussians $g_{w=1,\ldots,N}$ located at positions $\mathbf{r}_w$ in the primitive cell. Let $\mathbf{R}$ label the other primitive cells. There are therefore copies of the basic set of Gaussian functions located at positions $\mathbf{r}_w + \mathbf{R}$. We want to evaluate the Bloch orbitals at some point $\mathbf{r}$. The orbital is labelled by band ν and $\mathbf{k}$ point,

$$\phi_{\nu,\mathbf{k}}(\mathbf{r}) = \sum_w C_{\nu,\mathbf{k}} \sum_{\mathbf{R}} g_w(\mathbf{r} - \mathbf{r}_w + \mathbf{R})\exp[i\mathbf{k}\cdot\mathbf{R}]\,. \tag{108}$$

CASINO implicitly assumes the following about the $\mathbf{k}$ points when in Gaussian mode:

- The $\mathbf{k}$ points form a grid,

$$\mathbf{k}_{lmn} = \frac{l}{q_1}\mathbf{b}_1 + \frac{m}{q_2}\mathbf{b}_2 + \frac{n}{q_3}\mathbf{b}_3 + \mathbf{k}_{\mathrm{s}} \ , \tag{109}$$

  where the q_i are integers, the $\mathbf{b}_i$ are the primitive reciprocal lattice vectors, $0 \leq l \leq q_1 - 1$, $0 \leq m \leq q_2 - 1$ and $0 \leq n \leq q_3 - 1$. In CRYSTAL the offset $\mathbf{k}_{\mathrm{s}}$ is always zero, but CASINO does not assume this.

- CASINO deals only with real orbitals, which can be formed by making linear combinations of the states at $\mathbf{k}$ and $-\mathbf{k}$. The list of $\mathbf{k}$ points in the file `gwfn.data` must contain only one of each $(\mathbf{k}, -\mathbf{k})$ pair; the presence of the other is assumed. To make a many-body Bloch wave function satisfying the condition that it is multiplied by a phase factor under the replacement $\mathbf{r}_i \to \mathbf{r}_i + \mathbf{R}_{\mathrm{s}}$, where $\mathbf{R}_{\mathrm{s}}$ is a translation vector of the simulation cell, the offset must satisfy $\mathbf{k}_{\mathrm{s}} = \mathbf{G}_{\mathrm{s}}/2$, where $\mathbf{G}_{\mathrm{s}}$ is a reciprocal lattice vector of the simulation cell [42]: see Sec. 15.

- Finite systems are treated as if they had a single $\mathbf{k}$ point.

15 Constructing real orbitals

The Bloch orbitals at an arbitrary point in $\mathbf{k}$-space are complex. If the set of wave vectors consists of $\pm\mathbf{k}$ pairs then one can always construct a set of real orbitals spanning the same space as the original complex set. A necessary and sufficient condition for the mesh of Eq. (109) to consist of $\pm\mathbf{k}$ pairs is that $\mathbf{k}_{\mathrm{s}} = \mathbf{G}_{\mathrm{s}}/2$, where $\mathbf{G}_{\mathrm{s}}$ is a reciprocal lattice vector of the simulation cell lattice. It is four times more efficient to use real orbitals than complex ones because it takes four multiplications to evaluate the product of two complex numbers but only one multiplication for the product of two real numbers. An orbital satisfying Bloch's theorem can be written as

$$\phi_{\mathbf{k}}(\mathbf{r}) = u_{\mathbf{k}}(\mathbf{r})e^{i\mathbf{k}\cdot\mathbf{r}} \ , \tag{110}$$

where $u_{\mathbf{k}}$ has the periodicity of the primitive lattice. The function $\phi^*_{\mathbf{k}}$ is a Bloch function with wave vector $-\mathbf{k}$. Therefore we can make two real orbitals from $\phi_{\mathbf{k}}$ and $\phi^*_{\mathbf{k}}$ as follows:

$$\begin{aligned} \phi_+(\mathbf{r}) &= \frac{1}{\sqrt{2}}\left[\phi_{\mathbf{k}}(\mathbf{r}) + \phi^*_{\mathbf{k}}(\mathbf{r})\right] \ , \\ \phi_-(\mathbf{r}) &= \frac{1}{\sqrt{2}i}\left[\phi_{\mathbf{k}}(\mathbf{r}) - \phi^*_{\mathbf{k}}(\mathbf{r})\right] \ . \end{aligned} \tag{111}$$

The orbitals ϕ_+ and ϕ_- are orthogonal if $\phi_{\mathbf{k}}$ and $\phi^*_{\mathbf{k}} = \phi_{-\mathbf{k}}$ are orthogonal, which is true unless $\mathbf{k} - (-\mathbf{k}) = \mathbf{G}_{\mathrm{p}}$, i.e., their wave vectors differ by a reciprocal lattice vector of the primitive lattice. In this case ϕ_+ and ϕ_- are linearly dependent and we must use only one of them. Therefore the scheme is:

$$\begin{aligned} &\text{Case 1.} \quad \text{If } \mathbf{k} \neq \frac{\mathbf{G}_{\mathrm{p}}}{2} \text{ use } \phi_+ \text{ and } \phi_- \ . \\ &\text{Case 2.} \quad \text{If } \mathbf{k} = \frac{\mathbf{G}_{\mathrm{p}}}{2} \text{ use } \phi_+ \text{ or } \phi_- \ . \end{aligned} \tag{112}$$

In the second case, if one of ϕ_+ or ϕ_- is zero then obviously one must use the other one.

It may happen that we have in our $\mathbf{k}$-point grid the vectors $\mathbf{k}$ and $-\mathbf{k}$ which are not related by a reciprocal lattice vector of the primitive lattice. We may wish to occupy only one of the orbitals from these $\mathbf{k}$ points. We can then form a real orbital as $\cos(\theta)\phi_+ + \sin(\theta)\phi_-$, where θ is a phase angle between zero and 2π. If both $\mathbf{k}$ and $-\mathbf{k}$ orbitals are supplied, CASINO chooses one, which in general is sufficient to generate all linearly independent orbitals.

Note that, if **complex_wf** is set to `T`, then the Slater wave function is complex, and CASINO makes no attempt to construct real orbitals. See Sec. 28 for information about the use of complex wave functions under twisted boundary conditions.

16 Cusp corrections for Gaussian orbitals

Gaussian basis sets are unable to describe the cusps in the single-particle orbitals at the nuclei that would be present in the exact single-particle orbitals, because the Gaussian basis functions have zero gradient at the nuclei on which they are centred. This leads to divergences in the local energy at the nuclei, which should be removed. This can either be done using the Jastrow factor (which is generally a poor method) or by using the cusp correction scheme described here [43].

In this scheme the molecular orbitals are modified so that each of them obeys the cusp condition at each nucleus. This ensures that the local energy remains finite whenever an electron is in the vicinity of a nucleus, although it generally has a discontinuity at the nucleus.

16.1 Electron–nucleus cusp corrections

The Kato cusp condition [44] applied to an electron at $\mathbf{r}_i$ and a nucleus of charge Z at the origin is

$$\left(\frac{\partial\langle\Psi\rangle}{\partial r_i}\right)_{r_i=0} = -Z\langle\Psi\rangle_{r_i=0} \ , \tag{113}$$

where $\langle\Psi\rangle$ is the spherical average of the many-body wave function about $\mathbf{r}_i = 0$. For a determinant of orbitals to obey the Kato cusp condition at the nuclei it is sufficient for every orbital to obey Eq. (113) at every nucleus. We need only correct the orbitals which are nonzero at a particular nucleus because the others already obey Eq. (113). This is sufficient to guarantee that the local energy is finite at the nucleus provided at least one orbital is nonzero there. In the unlikely case that all of the orbitals are zero at the nucleus then the probability of an electron being at the nucleus is zero and it is not important whether Ψ obeys the cusp condition.

An orbital, ψ, expanded in a Gaussian basis set can be written as

$$\psi = \phi + \eta \ , \tag{114}$$

where ϕ is the part of the orbital arising from the s-type Gaussian functions centred on the nucleus in question (which, for convenience is at $\mathbf{r} = 0$), and η is the rest of the orbital. The spherical average of ψ about $\mathbf{r} = 0$ is given by

$$\langle\psi\rangle = \phi + \langle\eta\rangle \ . \tag{115}$$

In our scheme we seek a corrected orbital, $\tilde{\psi}$, which differs from ψ only in the part arising from the s-type Gaussian functions centred on the nucleus, i.e.,

$$\tilde{\psi} = \tilde{\phi} + \eta \ . \tag{116}$$

The correction, $\tilde{\psi} - \psi$, is therefore spherically symmetric about the nucleus. We now demand that $\tilde{\psi}$ obeys the cusp condition at $\mathbf{r} = 0$,

$$\left(\frac{d\langle\tilde{\psi}\rangle}{dr}\right)_0 = -Z\langle\tilde{\psi}\rangle_0 \ . \tag{117}$$

Note that $\langle\eta\rangle$ is cuspless because it arises from the Gaussian basis functions centred on the origin with nonzero angular momentum, whose spherical averages are zero, and the tails of the Gaussian basis functions centred on other sites, which must be cuspless at the nucleus in question. We therefore obtain

$$\left(\frac{d\tilde{\phi}}{dr}\right)_0 = -Z\left(\tilde{\phi}(0) + \eta(0)\right) \ . \tag{118}$$

We use Eq. (118) as the basis of our scheme for constructing cusp-corrected orbitals.

16.2 Cusp correction algorithm

We apply a cusp correction to each orbital at each nucleus at which it is nonzero. Inside some cusp correction radius $r_{\rm c}$ we replace ϕ, the part of the orbital arising from s-type Gaussian functions centred on the nucleus in question, by

$$\tilde{\phi} = C + \mathrm{sgn}[\tilde{\phi}(0)]\exp[p(r)] = C + R(r). \tag{119}$$

In this expression $\mathrm{sgn}[\tilde{\phi}(0)]$ is ± 1, reflecting the sign of $\tilde{\phi}$ at the nucleus, and C is a shift chosen so that $\tilde{\phi} - C$ is of one sign within $r_{\rm c}$. This shift is necessary since the uncorrected s-part of the orbital

ϕ may have a node where it changes sign inside the cusp correction radius, and we wish to replace ϕ by an exponential function, which is necessarily of one sign everywhere. The polynomial p is given by

$$p = \alpha_0 + \alpha_1 r + \alpha_2 r^2 + \alpha_3 r^3 + \alpha_4 r^4 \ , \tag{120}$$

and we determine α_0, α_1, α_2, α_3, and α_4 by imposing five constraints on $\tilde{\phi}$. We demand that the value and the first and second derivatives of $\tilde{\phi}$ match those of the s-part of the Gaussian orbital at $r = r_{\rm c}$. We also require that the cusp condition is satisfied at $r = 0$. We use the final degree of freedom to optimize the behaviour of the local energy in a manner to be described below. However, if we impose such a constraint directly the equations satisfied by the α_i cannot be solved analytically. This is inconvenient and we found that a superior algorithm was obtained by imposing a fifth constraint which allows the equations to be solved analytically, and then searching over the value of the fifth constraint for a 'good solution'. To this end we chose to constrain the value of $\tilde{\phi}(0)$. With these constraints we have:

1. $$\ln|\tilde{\phi}(r_{\rm c}) - C| = p(r_{\rm c}) = X_1; \tag{121}$$

2. $$\frac{1}{R(r_{\rm c})}\frac{d\tilde{\phi}}{dr}\bigg|_{r_{\rm c}} = p'(r_{\rm c}) = X_2; \tag{122}$$

3. $$\frac{1}{R(r_{\rm c})}\frac{d^2\tilde{\phi}}{dr^2}\bigg|_{r_{\rm c}} = p''(r_{\rm c}) + p'^2(r_{\rm c}) = X_3; \tag{123}$$

4. $$\frac{1}{R(0)}\frac{d\tilde{\phi}}{dr}\bigg|_{0} = p'(0) = -Z\left(\frac{C + R(0) + \eta(0)}{R(0)}\right) = X_4; \tag{124}$$

5. $$\ln|\tilde{\phi}(0) - C| = p(0) = X_5. \tag{125}$$

Although the constraint equations are nonlinear, they can be solved analytically, giving

$$\begin{aligned}
\alpha_0 &= X_5 \\
\alpha_1 &= X_4 \\
\alpha_2 &= 6\frac{X_1}{r_{\rm c}^2} - 3\frac{X_2}{r_{\rm c}} + \frac{X_3}{2} - 3\frac{X_4}{r_{\rm c}} - 6\frac{X_5}{r_{\rm c}^2} - \frac{X_2^2}{2} \\
\alpha_3 &= -8\frac{X_1}{r_{\rm c}^3} + 5\frac{X_2}{r_{\rm c}^2} - \frac{X_3}{r_{\rm c}} + 3\frac{X_4}{r_{\rm c}^2} + 8\frac{X_5}{r_{\rm c}^3} + \frac{X_2^2}{r_{\rm c}} \\
\alpha_4 &= 3\frac{X_1}{r_{\rm c}^4} - 2\frac{X_2}{r_{\rm c}^3} + \frac{X_3}{2r_{\rm c}^2} - \frac{X_4}{r_{\rm c}^3} - 3\frac{X_5}{r_{\rm c}^4} - \frac{X_2^2}{2r_{\rm c}^2}.
\end{aligned} \tag{126}$$

Our procedure is to solve Eq. (126) using an initial value of $\tilde{\phi}(0) = \phi(0)$. We then vary $\tilde{\phi}(0)$ so that the 'effective one-electron local energy',

$$\begin{aligned}
E_{\rm L}^s(r) &= \tilde{\phi}^{-1}\left[-\frac{1}{2}\nabla^2 - \frac{Z_{\rm eff}}{r}\right]\tilde{\phi} \\
&= -\frac{1}{2}\frac{R(r)}{C + R(r)}\left[\frac{2p'(r)}{r} + p''(r) + p'^2(r)\right] - \frac{Z_{\rm eff}}{r},
\end{aligned} \tag{127}$$

is well-behaved. Here the effective nuclear charge $Z_{\rm eff}$ is given by

$$Z_{\rm eff} = Z\left(1 + \frac{\eta(0)}{C + R(0)}\right), \tag{128}$$

which ensures that $E_{\rm L}^s(0)$ is finite when the cusp condition of Eq. (124) is satisfied.

We use an 'ideal' effective one-electron local energy curve given by

$$\frac{E_{\rm L}^{\rm ideal}(r)}{Z^2} = \beta_0 + \beta_1 r^2 + \beta_2 r^3 + \beta_3 r^4 + \beta_4 r^5 + \beta_5 r^6 + \beta_6 6 r^7 + \beta_7 r^8. \tag{129}$$

The values chosen for the coefficients were $\beta_1 = 3.25819$, $\beta_2 = -15.0126$, $\beta_3 = 33.7308$, $\beta_4 = -42.8705$, $\beta_5 = 31.2276$, $\beta_6 = -12.1316$, $\beta_7 = 1.94692$, obtained by fitting to the data for the $1s$ orbital of the carbon atom. The value of β_0 depends on the particular atom and its environment. The ideal effective one-electron local energy for a particular orbital is chosen to have the functional form of $E_{\rm L}^{\rm ideal}(r)$, but with the constant value β_0 chosen so that the effective one-electron local energy is continuous at $r_{\rm c}$. Hydrogen is treated as a special case as the $1s$ orbital of the isolated atom is only half-filled, and we use $E_{\rm L}^{\rm ideal}(r) = \beta_0$.

We wish to choose $\tilde{\phi}(0)$ so that $E_{\rm L}^s(r)$ is as close as possible to $E_{\rm L}^{\rm ideal}(r)$ for $0 < r < r_{\rm c}$, i.e., the effective one-electron local energy is required to follow the 'ideal' curve as closely as possible. In our current implementation we find the best $\tilde{\phi}(0)$ by minimizing the maximum square deviation from the ideal energy, $\left[E_{\rm L}^s(r) - E_{\rm L}^{\rm ideal}(r)\right]^2$, within this range. Beginning with $\tilde{\phi}(0) = \phi(0)$, we first bracket the minimum then refine $\tilde{\phi}(0)$ using a simple golden section search.

We use an automatic procedure for choosing appropriate values of the cusp correction radii. The maximum possible cusp correction radius is taken to be $r_{\rm c,max} = 1/Z$. The actual value of $r_{\rm c}$ is then determined by a universal parameter $c_{\rm c}$ (**cusp_control** in the `input` file) for which a default value of 50 was found to be reasonable. The cusp correction radius $r_{\rm c}$ for each orbital and nucleus is set equal to the largest radius less than $r_{\rm c,max}$ at which the deviation of the effective one-electron local energy calculated with ϕ from the ideal curve has a magnitude greater than $Z^2/c_{\rm c}$. Appropriate polynomial coefficients α_i and the resulting maximum deviation of the effective one-electron local energy from the ideal curve are then calculated for this $r_{\rm c}$. As a final refinement one might then allow the code to vary $r_{\rm c}$ over a relatively small range centred on the initial value, recomputing the optimal polynomial cusp correction at each radius, in order to optimize further the behaviour of the effective one-electron local energy. This is done by default in the implementation.

When a Gaussian orbital can be readily identified as, for example, a $1s$ orbital, it generally does not have a node within $r_{\rm c,max}$. In many cases, however, some of the molecular orbitals have small s-components which may have nodes close to the nucleus. The possible presence of nodes inside the cusp correction radius complicates the procedure because the effective one-electron local energy diverges there. One could simply force the cusp correction radius to be less than the radius of the node closest to the nucleus, but in practice nodes can be very close to the nucleus and such a constraint severely restricts the flexibility of the algorithm. In practice we define small regions around each node where the effective one-electron local energies are not taken into account during the minimization, and from which the cusp correction radius is excluded.

The Gaussian cusp correction is activated through the input keyword **cusp_correction**—this has an effect only if the system contains at least one all-electron atom. In periodic systems it has proved difficult to implement this scheme efficiently, and while it works perfectly well the performance of the code is significantly affected. Check the timings. More information about the cusp correction of each orbital at each nucleus can be produced with the keyword **cusp_info** which can be useful in fine tuning. Clearly this can produce a lot of output, so beware.

17 General-purpose cusp corrections

A scheme for modifying real, Gaussian orbitals so that they satisfy the Kato cusp conditions is described in Sec. 16 and Ref. [43]. An extension of this scheme can be used to enforce the Kato cusp conditions on complex orbitals expanded in any smooth basis set. Both cusp-correction schemes are present in CASINO. We refer to the former as the *Gaussian* cusp-correction scheme and the latter as the *general-purpose* (GP) cusp-correction scheme.

For simplicity, we consider only the case of a single nucleus of charge Z at the origin in the following discussion.

In the Gaussian cusp-correction scheme, the s-type basis functions are replaced by radial functions in the vicinity of the bare nucleus. These functions satisfy the cusp conditions and make the single-particle local energy resemble an 'ideal' curve [43]. In the GP scheme, instead of *replacing* part of the orbital, we *add* a spherically symmetric function of constant phase to the orbital. The function added to uncorrected orbital $\psi(\mathbf{r})$ is

$$\Delta\psi(r) = \exp(i\theta_0)\Delta\phi(r) = \exp(i\theta_0)\left[C + \exp\left[p(r)\right] - \phi(r)\right]\Theta(r_{\rm c} - r), \tag{130}$$

where Θ is the Heaviside function, C is a real constant, $r_{\rm c}$ is a cutoff length, $\theta_0 = \arg[\psi(\mathbf{0})]$,

$$\phi(r) = \mathrm{Re}\left(\frac{\exp(-i\theta_0)}{4\pi}\int \psi(\mathbf{r})\, d\Omega\right) , \tag{131}$$

and

$$p(r) = \alpha_0 + \alpha_1 r + \alpha_2 r^2 + \alpha_3 r^3 + \alpha_4 r^4, \tag{132}$$

where the $\{\alpha\}$ are real constants to be determined. In practice $\phi(r)$ is calculated by cubic spline interpolation: the spherical averaging of the uncorrected orbital is performed at the outset on a grid of radial points. C is chosen so that $\phi(r) - C$ is positive everywhere within the Bohr radius of the nucleus.

The uncorrected orbital may be written as

$$\psi(\mathbf{r}) = \exp(i\theta_0)\phi(r) + \eta(\mathbf{r}), \tag{133}$$

where $\eta(\mathbf{r})$ consists of the $l > 0$ spherical harmonic components of $\psi(\mathbf{r})$, together with the phase-dependence of the $l = 0$ component. Note that $\exp(i\theta_0)\phi(0) = \psi(\mathbf{0})$, and hence $\eta(\mathbf{0}) = 0$. Let

$$\tilde{\phi}(r) = \phi(r) + \Delta\phi(r). \tag{134}$$

These are real, spherically symmetric functions.

We may now apply the scheme of Ma *et al.* to determine $\{\alpha\}$ and $r_{\rm c}$, with $\exp(i\theta_0)\phi$ and $\exp(i\theta_0)\tilde{\phi}$ playing the roles of the uncorrected and corrected s-type Gaussian functions centred on the nucleus in question [43]. The constant phase $\exp(i\theta_0)$ cancels out of the equations determining the $\{\alpha\}$ (Eqs. (9)–(13) in Ref. [43]), so the determination of the $\{\alpha\}$ and $r_{\rm c}$ is exactly as described in Ma *et al.*, except that we do not need to modify Z at the nuclei when more than one atom is present, because $\eta(\mathbf{0}) = 0$.

Each orbital is corrected at each all-electron ion in the simulation cell. If twisted boundary conditions are used then the phase of the uncorrected orbital at the nucleus in simulation cell $\mathbf{R}_{\rm s}$ is $\exp[i(\theta_0 + \mathbf{k}_{\rm s} \cdot \mathbf{R}_{\rm s})]$, where $\mathbf{k}_{\rm s}$ is the simulation-cell Bloch vector. [So we can evaluate the cusp correction using minimum-image distances, then multiply by $\exp(i\mathbf{k}_{\rm s} \cdot \mathbf{R}_{\rm s})$, where $\mathbf{R}_{\rm s}$ is the difference of the actual and minimum-image positions of the electron relative to the ion.]

To use the scheme, set **use_gpcc** to `T`. Note that **cusp_correction** (which activates the Gaussian cusp correction) must be set to `F`.

18 Wave-function updating

Consider the Slater wave function

$$\Psi_{\rm S}(\mathbf{R}) = D^{\uparrow}(\mathbf{r}_1, \ldots, \mathbf{r}_{N_\uparrow}) D^{\downarrow}(\mathbf{r}_{N_\uparrow+1}, \ldots, \mathbf{r}_N) , \tag{135}$$

where $D^{\uparrow}$ and $D^{\downarrow}$ are Slater determinants for the spin-up and spin-down electrons respectively. We will need to calculate the ratio of the new wave function to the old when, for example, the ith spin-up electron is moved from $\mathbf{r}_i^{\rm old}$ to $\mathbf{r}_i^{\rm new}$. The wave-function ratio can be written as

$$q^{\uparrow} = \frac{D^{\uparrow}(\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_i^{\rm new}, \ldots, \mathbf{r}_N)}{D^{\uparrow}(\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_i^{\rm old}, \ldots, \mathbf{r}_N)} . \tag{136}$$

A direct calculation of the determinants in $q^{\uparrow}$ at every move by LU decomposition is time-consuming, and instead we use an updating method. We define the Slater matrix $\mathcal{D}^{\uparrow}$ via

$$\mathcal{D}^{\uparrow}_{jk} = \psi_j(\mathbf{r}_k) , \tag{137}$$

where ψ_j is the jth one-electron orbital of the spin-up Slater determinant and $\mathbf{r}_k$ is the position of the kth spin-up electron. The transpose of the inverse of $\mathcal{D}^{\uparrow}$, which we call $\overline{\mathcal{D}}^{\uparrow}$, may be expressed in terms of the cofactors and determinant of $\mathcal{D}^{\uparrow}$,

$$\overline{\mathcal{D}}^{\uparrow}_{jk} = \frac{\mathrm{cof}(\mathcal{D}^{\uparrow}_{jk})}{\det(\mathcal{D}^{\uparrow})} . \tag{138}$$

The move of electron i changes only the ith column of $\mathcal{D}^{\uparrow}$ and so does not affect any of the cofactors associated with this column. The new Slater determinant may be expanded in terms of these cofactors and the result divided by the old Slater determinant to obtain

$$q^{\uparrow} = \frac{\det(\mathcal{D}^{\uparrow,\text{new}})}{\det(\mathcal{D}^{\uparrow,\text{old}})} = \sum_j \psi_j(\mathbf{r}_i^{\text{new}})\overline{\mathcal{D}}_{ji}^{\uparrow,\text{old}} \ . \tag{139}$$

If the $\overline{\mathcal{D}}^{\uparrow}$ matrix is known, one can compute $q^{\uparrow}$ in a time proportional to N.

Evaluating $\overline{\mathcal{D}}^{\uparrow}$ using LU decomposition takes of order N^3 operations; but once the initial $\overline{\mathcal{D}}^{\uparrow}$ matrix has been calculated it can be updated at a cost proportional to N^2 using the formulae

$$\overline{\mathcal{D}}_{ji}^{\uparrow,\text{new}} = \frac{1}{q^{\uparrow}}\overline{\mathcal{D}}_{ji}^{\uparrow,\text{old}} \ , \tag{140}$$

in the case where $k = i$, and

$$\overline{\mathcal{D}}_{jk}^{\uparrow,\text{new}} = \overline{\mathcal{D}}_{jk}^{\uparrow,\text{old}} - \frac{1}{q^{\uparrow}}\overline{\mathcal{D}}_{ji}^{\uparrow,\text{old}} \sum_m \psi_m(\mathbf{r}_i^{\text{new}})\overline{\mathcal{D}}_{mk}^{\uparrow,\text{old}} \ , \tag{141}$$

when $k \neq i$.

19 Evaluating the local energy

The local energy is given by

$$E_{\text{L}}(\mathbf{R}) = \sum_{i=1}^{N} -\frac{1}{2}\Psi^{-1}(\mathbf{R})\nabla_i^2\Psi(\mathbf{R}) + \sum_{i=1}^{N} V(\mathbf{R}) + \sum_{i=1}^{N} \Psi^{-1}(\mathbf{R})\hat{V}_{\text{nl},i}^{\text{ps}}\Psi(\mathbf{R}) + V_{\text{CPP}}(\mathbf{R}) + \sum_{i>j}^{N} v_{\text{e−e}}(\mathbf{R}) \ , \tag{142}$$

where the terms are the kinetic energy, the local part of the external potential energy, the nonlocal part of the potential energy, the core polarization potential energy (if present) and the electron–electron interaction energy. The evaluation of the kinetic energy is discussed in Sec. 19.1. The evaluation of the nonlocal energy is discussed in Sec. 19.2, while the core-polarization potential energy is discussed in Sec. 19.3. The local part of the external potential energy is divided into a short range part around each ion, which is evaluated directly, and a long range Coulomb part which is evaluated using the Ewald potential in periodic systems (see Sec. 19.4) or simply as a sum of $1/r$ potentials in finite systems. The electron–electron interaction energy is evaluated either using the Ewald interaction or the MPC interaction (see Sec. 19.4.4) in periodic systems, or simply as a sum of $1/r$ potentials in finite systems.

19.1 Evaluating the kinetic energy

The kinetic part of the local energy, K, can be expressed as a sum of contributions from each electron,

$$K = \sum_{i=1}^{N} K_i = \sum_{i=1}^{N} -\frac{1}{2}\Psi(\mathbf{R})^{-1}\nabla_i^2\Psi(\mathbf{R}) \ . \tag{143}$$

Because of the exponential form of the Jastrow factor, it is convenient to re-express K_i in terms of the logarithm of Ψ. We define

$$T_i = -\frac{1}{4}\nabla_i^2\left(\ln|\Psi|\right) = -\frac{1}{4}\frac{\nabla_i^2\Psi}{\Psi} + \frac{1}{4}\left(\frac{\nabla_i\Psi}{\Psi}\right)^2 \ , \tag{144}$$

and the drift vector $\mathbf{F}_i$,

$$\mathbf{F}_i = \frac{1}{\sqrt{2}}\nabla_i\left(\ln|\Psi|\right) = \frac{1}{\sqrt{2}}\frac{\nabla_i\Psi}{\Psi} \ . \tag{145}$$

Therefore

$$K_i = 2T_i - |\mathbf{F}_i|^2 \ . \tag{146}$$

In VMC an integration by parts shows that

$$\langle K\rangle = \langle|\mathbf{F}|^2\rangle = \langle T\rangle \ , \tag{147}$$

where the angle brackets denote averages over the variational distribution, $|\Psi(\mathbf{R})|^2$. Equation (147) provides a useful consistency check for VMC calculations but note that it does not hold exactly within DMC, except in the limit of perfect importance sampling. In VMC the kinetic energy may be evaluated using any of the three estimators in Eq. (147). CASINO automatically uses $\langle K\rangle$ for the evaluation of the total energy, because this normally leads to the lowest variance. However, the lowest variance of the kinetic energy itself is often obtained from $\langle T\rangle$. In DMC the three estimators are not exactly equivalent and $\langle K\rangle$ should always be used as the kinetic-energy estimate.

For the Slater-Jastrow wave function of Eq. (135) we have

$$\nabla_i(\ln|\Psi|) = \frac{\nabla_i D^{\sigma_i}}{D^{\sigma_i}} + \nabla_i J \ , \tag{148}$$

$$\nabla_i^2(\ln|\Psi|) = \frac{\nabla_i^2 D^{\sigma_i}}{D^{\sigma_i}} - \left(\frac{\nabla_i D^{\sigma_i}}{D^{\sigma_i}}\right)^2 + \nabla_i^2 J \ . \tag{149}$$

The terms involving Slater determinants may be evaluated by expanding D^{σ_i} in terms of the cofactors of the ith column of the Slater matrix $\mathcal{D}^{\sigma_i}$. If electron i has spin up, for example, the required expansion is

$$D^{\uparrow} = \det(\mathcal{D}^{\uparrow}) = \sum_j \psi_j(\mathbf{r}_i)\,\mathrm{cof}(\mathcal{D}^{\uparrow}_{ji}) \ . \tag{150}$$

Since all the cofactors appearing in this equation are independent of $\mathbf{r}_i$, we obtain

$$\frac{\nabla_i D^{\uparrow}}{D^{\uparrow}} = \sum_j (\nabla_i\psi_j(\mathbf{r}_i))\,\overline{\mathcal{D}}^{\uparrow}_{ji} \ , \tag{151}$$

$$\frac{\nabla_i^2 D^{\uparrow}}{D^{\uparrow}} = \sum_j \left(\nabla_i^2\psi_j(\mathbf{r}_i)\right)\overline{\mathcal{D}}^{\uparrow}_{ji} \ . \tag{152}$$

When moving electron i from $\mathbf{r}_i^{\rm old}$ to $\mathbf{r}_i^{\rm new}$, it is useful to be able to evaluate the kinetic energy at the new position before updating the $\overline{\mathcal{D}}$ matrix. Since the cofactors in Eq. (150) are independent of $\mathbf{r}_i$, Eqs. (151) and (152) become

$$\frac{\nabla_i D^{\uparrow,\rm new}}{D^{\uparrow,\rm new}} = \frac{1}{q^{\uparrow}}\sum_j (\nabla_i\psi_j(\mathbf{r}_i^{\rm new}))\,\overline{\mathcal{D}}^{\uparrow,\rm old}_{ji} \ , \tag{153}$$

$$\frac{\nabla_i^2 D^{\uparrow,\rm new}}{D^{\uparrow,\rm new}} = \frac{1}{q^{\uparrow}}\sum_j \left(\nabla_i^2\psi_j(\mathbf{r}_i^{\rm new})\right)\overline{\mathcal{D}}^{\uparrow,\rm old}_{ji} \ , \tag{154}$$

where $q^{\uparrow} = D^{\uparrow,\rm new}/D^{\uparrow,\rm old}$.

19.2 Evaluating the nonlocal pseudopotential energy

The action of the nonlocal pseudopotential on the wave function can be written as a sum of contributions from each electron and each angular momentum channel. The contribution to the local energy made by the nonlocal pseudopotential is

$$\begin{aligned} V_{\rm nl} &= \Psi^{-1}\hat{V}_{\rm nl}\Psi \\ &= \sum_i \Psi^{-1}\hat{V}^{\rm ps}_{{\rm nl},i}\Psi = \sum_i V_{{\rm nl},i} \ , \end{aligned} \tag{155}$$

where for simplicity we consider the case of a single atom placed at the origin. $V_{{\rm nl},i}$ may be written as [16]

$$V_{{\rm nl},i} = \sum_l V^{\rm ps}_{{\rm nl},l}(r_i)\,\frac{2l+1}{4\pi}\int P_l\left[\cos(\theta'_i)\right]\frac{\Psi(\mathbf{r}_1,\ldots,\mathbf{r}_{i-1},\mathbf{r}'_i,\mathbf{r}_{i+1},\ldots,\mathbf{r}_N)}{\Psi(\mathbf{r}_1,\ldots,\mathbf{r}_{i-1},\mathbf{r}_i,\mathbf{r}_{i+1},\ldots,\mathbf{r}_N)}\,d\Omega_{\mathbf{r}'_i}, \tag{156}$$

where P_l denotes a Legendre polynomial.

CASINO currently performs the nonlocal projections for $l = 0,1,2,3,4$ only (though any number of higher angular momentum functions may be omitted if desired, and doing just that can considerably speed up the code). Many standard pseudopotentials use just $l = 0, 1, 2$ (s, p, d). The integral over the surface of the sphere in Eq. (156) is evaluated numerically. The $\mathbf{r}'$ dependence of the many-body wave function is expected to have predominantly the angular momentum character of the orbitals in the Slater part of the wave function. A suitable integration scheme is therefore to use a quadrature rule that integrates products of spherical harmonics exactly up to some maximum value $l_{\max}$. The quadrature grids currently available are listed in Table 2. To avoid bias the orientation of the axes is chosen randomly each time such an integral is evaluated.

Note that the CPP and force modules still require that pseudopotentials contain only s, p and d components (this should be fixable by a friendly developer on request).

non_local_grid	$l_{\max}$	N_{p}
1	0	1
2	2	4
3	3	6
4	5	12
5	5	18
6	7	26
7	11	50

Table 2: Quadrature grids for the nonlocal integration. **non_local_grid** (also known as **nlrule**) is the label for the rule, $l_{\max}$ is the maximum value of l which is integrated exactly and N_{p} is the number of points in the grid.

Within a VMC calculation it is often possible to use a low-order quadrature rule because the error cancels over the run, but higher accuracy is required for wave-function optimization and DMC calculations, which are biased by errors in the nonlocal integration. In principle the nonlocal energy should be summed over all the ionic cores and all electrons in the system. However, since the nonlocal potential of each ion is short ranged, one need only sum over the few atoms nearest to each electron.

Exact sampling of the nonlocal energy with DMC is problematic and we use the *localization approximation* in which the nonlocal operator acts on the trial wave function in exactly the same way as in VMC. The error introduced by this approximation is proportional to $(\Psi - \Psi_0)^2$ [41], where Ψ_0 is the exact wave function.

19.3 The core-polarization potential energy

Core-polarization potentials (CPPs) account for the polarization of the pseudo-ion cores by the fields of the other charged particles in the system. The polarization of the pseudo-ion cores by the fields of the valence electrons is a many-body effect which includes some of the core-valence correlation energy [45]. In the CPP approximation the polarization of a particular core is determined by the electric field at the nucleus. The electric field acting on a given ion core at $\mathbf{R}_I$ due to the other ion cores at $\mathbf{R}_J$ and the electrons at $\mathbf{r}_i$ is

$$\mathbf{F}_I = -\sum_{J \neq I} Z_J \frac{\mathbf{R}_{JI}}{|\mathbf{R}_{JI}|^3} + \sum_i \frac{\mathbf{r}_{iI}}{|\mathbf{r}_{iI}|^3} , \tag{157}$$

where $\mathbf{R}_{JI} = \mathbf{R}_J - \mathbf{R}_I$ and $\mathbf{r}_{iI} = \mathbf{r}_i - \mathbf{R}_I$. The CPP energy is then

$$V_{\mathrm{CPP}} = -\frac{1}{2} \sum_I \alpha_I \mathbf{F}_I \cdot \mathbf{F}_I , \tag{158}$$

where α_I is the dipole polarizability of core I.

Equation (157) assumes a classical description, which is valid when the valence electrons are far from the core. When a valence electron penetrates the core, the classical result is a very poor approximation, diverging at the nucleus. To remove this unphysical behaviour each contribution to the electric field in Eq. (157) is multiplied by a cutoff function $f(r_{iI}/\bar{r}_I)$, which tends to unity at large r_{iI}. A further possible modification is to allow the one-electron term in Eq. (158), which takes the form $-\alpha_I/(2r_{iI}^4)$, to depend on the angular momentum component, l, so that $\bar{r}_I$ in the cutoff function is replaced by

$\bar{r}_{lI}$. With these modifications the CPP energy operator becomes

$$V_{\rm CPP} = -\frac{1}{2}\sum_I \alpha_I \left[\sum_i \frac{1}{r_{iI}^4}\sum_l f\left(\frac{r_{iI}}{\bar{r}_{lI}}\right)^2 \hat{P}_l + \sum_i\sum_{j\neq i}\frac{\mathbf{r}_{iI}\cdot\mathbf{r}_{jI}}{r_{iI}^3 r_{jI}^3} f\left(\frac{r_{iI}}{\bar{r}_I}\right) f\left(\frac{r_{jI}}{\bar{r}_I}\right)\right.$$
$$\left. -2\sum_i\sum_{J\neq I}\frac{\mathbf{r}_{iI}\cdot\mathbf{R}_{JI}}{r_{iI}^3 R_{JI}^3} f\left(\frac{r_{iI}}{\bar{r}_I}\right) Z_J + \left(\sum_{J\neq I}\frac{\mathbf{R}_{JI}}{R_{JI}^3} Z_J\right)^2\right] , \qquad (159)$$

where $\hat{P}_l$ is the projector onto the lth angular momentum component of the ith electron with respect to the Ith ion.

We use the cutoff function [46, 45],

$$f(x) = \left(1 - e^{-x^2}\right)^2 . \qquad (160)$$

For efficient evaluation, Eq. (159) is written as

$$V_{\rm CPP} = -\frac{1}{2}\sum_I \alpha_I |\bar{\mathbf{F}}_I|^2 + \frac{1}{2}\sum_I \alpha_I \sum_i \frac{1}{r_{iI}^4}\sum_{l=0}^{lmx}\left[f\left(\frac{r_{iI}}{\bar{r}_I}\right)^2 - f\left(\frac{r_{iI}}{\bar{r}_{lI}}\right)^2\right]\hat{P}_l , \qquad (161)$$

where

$$\bar{\mathbf{F}}_I = -\sum_{J\neq I} Z_J \frac{\mathbf{R}_{JI}}{|\mathbf{R}_{JI}|^3} + \sum_i \frac{\mathbf{r}_{iI}}{|\mathbf{r}_{iI}|^3} f\left(\frac{r_{iI}}{\bar{r}_I}\right) , \qquad (162)$$

and the maximum angular momentum is $lmx = 2$. In our approach the cutoff parameter for all angular momenta $l > 2$ is $\bar{r}_I$, which is slightly different from Shirley and Martin [45] who use $\bar{r}_{2I}$.

Equation (161) contains 5 parameters for each ion, $\alpha_I, \bar{r}_{0I}, \bar{r}_{1I}, \bar{r}_{2I}$ and $\bar{r}_I$, whose values are entered at the end of the `xx_pp.data` file; see Sec. 7.5. Suitable values of the parameters are given in the paper by Shirley and Martin [45]. If $\bar{r}_{0I} = \bar{r}_{1I} = \bar{r}_{2I} = \bar{r}_I$, the second term in Eq. (161) is zero and it is not calculated. The second term in Eq. (161) is short-ranged because $f(x) \to 1$ at large x. This term is calculated in real space.

The second term in Eq. (161) is added to the pseudopotential and the core radii **lcutofftol** and **nlcutofftol** are determined from the resulting potential. The electric field evaluation is activated by the presence of core-polarization terms in the pseudopotential files; they are not calculated by default since they may be expensive, especially when periodic boundary conditions are used.

In periodic boundary conditions the electric fields are evaluated directly from the analytic first derivatives of the Ewald potential: see Sec. 19.4. Calculations using CPPs may be 5–10% slower than ones without CPPs in periodic systems.

Note: the evaluation of the first derivatives of the periodic potential in 1D polymers has not yet been implemented, and thus the core-polarization energy cannot be evaluated in such systems.

19.4 Evaluation of infinite Coulomb sums

19.4.1 3D Ewald interaction

In three dimensionally periodic systems, the periodic potential of a neutralized lattice of point charges may be evaluated using the Ewald method [47, 48]. Consider the periodic charge density consisting of a unit point charge at $\mathbf{r}_j$ in every simulation cell plus a uniform cancelling background,

$$\rho_j(\mathbf{r}) = \sum_{\mathbf{R}}\left(\delta(\mathbf{r}-\mathbf{r}_j-\mathbf{R}) - \frac{1}{\Omega}\right) , \qquad (163)$$

where $\mathbf{R}$ denotes the lattice translation vectors and Ω is the volume of the simulation cell. The Ewald formula for the periodic potential corresponding to this charge density is

$$\begin{aligned} v_{\rm E}(\mathbf{r},\mathbf{r}_j) &= \sum_{\mathbf{R}}\frac{{\rm erfc}\left(\gamma^{\frac{1}{2}}|\mathbf{r}-(\mathbf{r}_j+\mathbf{R})|\right)}{|\mathbf{r}-(\mathbf{r}_j+\mathbf{R})|} - \frac{\pi}{\Omega\gamma} \\ &\quad + \frac{4\pi}{\Omega}\sum_{\mathbf{G}\neq\mathbf{0}}\frac{\exp\left(-G^2/4\gamma\right)}{G^2}\exp(i\mathbf{G}\cdot(\mathbf{r}-\mathbf{r}_j)) , \end{aligned} \qquad (164)$$

where $\mathbf{G}$ denotes the reciprocal lattice translation vectors. The value of $v_{\rm E}(\mathbf{r},\mathbf{r}_j)$ is in principle independent of the screening parameter γ and this also holds in practice provided enough vectors are included in the sums.[21] The larger the value of γ the more rapidly convergent is the real space sum, but the more slowly convergent is the reciprocal space sum. A compromise is required to minimize the overall computational cost. In CASINO, this parameter is set to $(2.8/\Omega^{1/3})^2$ which approximately minimizes the cost for a wide variety of Bravais lattices [49].

The full periodic potential of the simulation cell is obtained by adding the potentials of all the N charges and their cancelling backgrounds (which sum to zero because the cell has no net charge),

$$v(\mathbf{r}) = \sum_{j=1}^{N} q_j v_{\rm E}(\mathbf{r},\mathbf{r}_j) \ . \tag{165}$$

The potential acting on the charge at $\mathbf{r}_i$ is therefore

$$v(\mathbf{r}_i) = \sum_{j(\neq i)}^{N} q_j v_{\rm E}(\mathbf{r}_i,\mathbf{r}_j) + q_i v_{\rm M} \ , \tag{166}$$

where the Madelung constant

$$\begin{aligned} v_{\rm M} &= \lim_{\mathbf{r}\to\mathbf{r}_i}\left(v_{\rm E}(\mathbf{r},\mathbf{r}_i) - \frac{1}{|\mathbf{r}-\mathbf{r}_i|}\right) & (167)\\ &= \sum_{\mathbf{R}\neq\mathbf{0}} \frac{\operatorname{erfc}(\gamma^{\frac{1}{2}}R)}{R} - \frac{2\gamma^{\frac{1}{2}}}{\pi^{\frac{1}{2}}} - \frac{\pi}{\Omega\gamma} + \frac{4\pi}{\Omega}\sum_{\mathbf{G}\neq\mathbf{0}} \frac{\exp(-G^2/4\gamma)}{G^2} & (168)\end{aligned}$$

is the potential acting on the charge at $\mathbf{r}_i$ due to its own images and cancelling background. The full Ewald potential energy appearing in the QMC Hamiltonian is therefore

$$\begin{aligned} U(\mathbf{r}_1,\ldots,\mathbf{r}_N) &= \frac{1}{2}\sum_{i=1}^{N}\sum_{\substack{j=1\\(j\neq i)}}^{N} q_i q_j v_{\rm E}(\mathbf{r}_i,\mathbf{r}_j) + \frac{v_{\rm M}}{2}\sum_{i=1}^{N} q_i^2 & (169)\\ &= \frac{1}{2}\sum_{i=1}^{N}\sum_{\substack{j=1\\(j\neq i)}}^{N} q_i q_j \left(v_{\rm E}(\mathbf{r}_i,\mathbf{r}_j) - v_{\rm M}\right) \ , & (170)\end{aligned}$$

where we have used the charge neutrality condition, $q_i = -\sum_{j(\neq i)} q_j$. The interaction $v_{\rm E}(\mathbf{r}_i,\mathbf{r}_j) - v_{\rm M}$ approaches $1/|\mathbf{r}_i-\mathbf{r}_j|$ as $\mathbf{r}_i \to \mathbf{r}_j$ and is independent of the choice of the zero of potential.

The gradient of the 3D Ewald potential [Eq. (164)] is required for evaluation of the core-polarization contribution to the total energy (Sec. 19.3). It is given by

$$\begin{aligned} \nabla v_{\rm E}(\mathbf{r},\mathbf{r}_j) &= -\sum_{\mathbf{R}} \frac{\mathbf{r}-(\mathbf{r}_j+\mathbf{R})}{|\mathbf{r}-(\mathbf{r}_j+\mathbf{R})|^2}\left(\frac{\operatorname{erfc}\left(\gamma^{\frac{1}{2}}|\mathbf{r}-(\mathbf{r}_j+\mathbf{R})|\right)}{|\mathbf{r}-(\mathbf{r}_j+\mathbf{R})|} + \frac{2\gamma^{\frac{1}{2}}}{\pi^{\frac{1}{2}}}\exp(-\gamma|\mathbf{r}-(\mathbf{r}_j+\mathbf{R})|^2)\right)\\ &- \frac{4\pi}{\Omega}\sum_{\mathbf{G}\neq\mathbf{0}} \mathbf{G}\frac{\exp\left(-G^2/4\gamma\right)}{G^2}\sin\left(\mathbf{G}\cdot(\mathbf{r}-\mathbf{r}_j)\right) \ . & (171)\end{aligned}$$

19.4.2 2D Ewald interaction

Infinite Coulomb sums in systems which are periodic in two dimensions (the xy-plane, according to CASINO) are performed using the standard 2D Ewald method originally developed by Parry [50]. One way to derive the relevant formula is to take the infinite separation limit of the 3D sum for a periodic stack of finite-width slabs. CASINO uses this algorithm when treating two-dimensional slabs of atoms with local Gaussian basis sets (useful in modelling surfaces) and also when treating 2D electron and electron–hole phases (either as strict 2D planes, 2D slabs with finite thickness, or strict 2D bilayers). In

[21] Increasing the input parameter **ewald_control** will increase the number of reciprocal vectors included in the sum, the effect of which is to increase the range of γ over which the energy is constant. The default value of γ should normally lie in the middle of the constant energy region for the default number of reciprocal vectors, so playing with **ewald_control** is normally unnecessary.

the case of periodic arrays of slabs separated by a finite vacuum gap (necessary when using plane-wave basis sets), the regular 3D algorithm is used.

Consider a finite width slab with a charge density periodic in two dimensions consisting of a unit point charge at $\mathbf{r}_j$ in every simulation cell plus a uniform cancelling background.

$$\rho_j(\mathbf{r}) = \sum_{\mathbf{R}} \left(\delta(\mathbf{r} - \mathbf{r}_j - \mathbf{R}) - \frac{1}{A} \right) \ , \tag{172}$$

where $\mathbf{R}$ now denotes the 2D lattice translation vectors in the xy-plane, and A is the area of the simulation cell in that plane. The Parry formula for the periodic potential corresponding to this charge density is

$$\begin{aligned} v_{\rm E}^{2D}(\mathbf{r},\mathbf{r}_j) &= \sum_{\mathbf{R}} \frac{\mathrm{erfc}\big(\gamma^{\frac{1}{2}}|\mathbf{r}-(\mathbf{r}_j+\mathbf{R})|\big)}{|\mathbf{r}-(\mathbf{r}_j+\mathbf{R})|} \\ &+ \frac{\pi}{A}\sum_{\mathbf{G}\neq\mathbf{0}} \frac{\exp(i\mathbf{G}\cdot(\mathbf{r}-\mathbf{r}_j))}{G}\left[\exp(zG)\mathrm{erfc}\left(\frac{G}{2\gamma^{\frac{1}{2}}}+z\gamma^{\frac{1}{2}}\right)+\exp(-zG)\mathrm{erfc}\left(\frac{G}{2\gamma^{\frac{1}{2}}}-z\gamma^{\frac{1}{2}}\right)\right] \\ &- \frac{2\pi}{A}\left[\mathrm{erf}(z\gamma^{\frac{1}{2}})z+\frac{\exp(-\gamma z^2)}{(\gamma\pi)^{\frac{1}{2}}}\right] \ , \end{aligned} \tag{173}$$

where $\mathbf{G}$ denotes the reciprocal lattice translation vectors in the xy-plane, and z is the z-component of the $\mathbf{r}-\mathbf{r}_j$ vector. In two dimensions CASINO sets the screening parameter γ to $(2.4/A^{1/2})^2$ which again should approximately minimize the cost over different Bravais lattices.

The full periodic potential of the simulation cell is obtained by following a procedure analogous to that described for the 3D case, with the Madelung constant $v_{\rm M}$ given by

$$v_{\rm M} = \lim_{\mathbf{r}\to\mathbf{r}_i}\left(v_{\rm E}(\mathbf{r},\mathbf{r}_i)-\frac{1}{|\mathbf{r}-\mathbf{r}_i|}\right) \tag{174}$$

$$= \sum_{\mathbf{R}\neq\mathbf{0}}\frac{\mathrm{erfc}\left(\gamma^{\frac{1}{2}}R\right)}{R} - \frac{2\gamma^{\frac{1}{2}}}{\pi^{\frac{1}{2}}} + \frac{2\pi}{A}\sum_{\mathbf{G}\neq\mathbf{0}}\frac{\mathrm{erfc}\left(\frac{G}{2\gamma^{\frac{1}{2}}}\right)}{G} - \frac{2\pi^{\frac{1}{2}}}{A\gamma^{\frac{1}{2}}} \ . \tag{175}$$

The first derivatives of the 2D Ewald potential, required for the evaluation of the core-polarization energy in 2D slabs, are different in directions parallel and perpendicular to the plane of the slab. The x and y derivatives are given by

$$\begin{aligned} \frac{\partial v_{\rm E}^{2D}(\mathbf{r},\mathbf{r}_j)}{\partial\lambda} &= -\sum_{\mathbf{R}}\frac{(\mathbf{r}-(\mathbf{r}_j+\mathbf{R}))_\mu}{|\mathbf{r}-(\mathbf{r}_j+\mathbf{R})|^2}\left[\frac{\mathrm{erfc}\big(\gamma^{\frac{1}{2}}|\mathbf{r}-(\mathbf{r}_j+\mathbf{R})|\big)}{|\mathbf{r}-(\mathbf{r}_j+\mathbf{R})|}+\frac{2\gamma^{\frac{1}{2}}}{\pi^{\frac{1}{2}}}\exp(-\gamma|\mathbf{r}-(\mathbf{r}_j+\mathbf{R})|^2)\right] \\ &- \frac{\pi}{A}\sum_{\mathbf{G}\neq\mathbf{0}}\frac{G_\mu\sin(\mathbf{G}\cdot(\mathbf{r}-\mathbf{r}_j))}{G}\left[\exp(zG)\mathrm{erfc}\left(\frac{G}{2\gamma^{\frac{1}{2}}}+z\gamma^{\frac{1}{2}}\right)+\exp(-zG)\mathrm{erfc}\left(\frac{G}{2\gamma^{\frac{1}{2}}}-z\gamma^{\frac{1}{2}}\right)\right] \ . \end{aligned} \tag{176}$$

where $\lambda = x$ or y, and the z derivative is given by

$$\begin{aligned} \frac{\partial v_{\rm E}^{2D}(\mathbf{r},\mathbf{r}_j)}{\partial z} &= -\sum_{\mathbf{R}}\frac{z}{|\mathbf{r}-(\mathbf{r}_j+\mathbf{R})|^2}\left[\frac{\mathrm{erfc}\big(\gamma^{\frac{1}{2}}|\mathbf{r}-(\mathbf{r}_j+\mathbf{R})|\big)}{|\mathbf{r}-(\mathbf{r}_j+\mathbf{R})|}+\frac{2\gamma^{\frac{1}{2}}}{\pi^{\frac{1}{2}}}\exp(-\gamma|\mathbf{r}-(\mathbf{r}_j+\mathbf{R})|^2)\right] \\ &+ \frac{\pi}{A}\sum_{\mathbf{G}\neq\mathbf{0}}\cos(\mathbf{G}\cdot(\mathbf{r}-\mathbf{r}_j))\left[\exp(zG)\mathrm{erfc}\left(\frac{G}{2\gamma^{\frac{1}{2}}}+z\gamma^{\frac{1}{2}}\right)+\exp(-zG)\mathrm{erfc}\left(\frac{G}{2\gamma^{\frac{1}{2}}}-z\gamma^{\frac{1}{2}}\right)\right] \\ &- \frac{2\pi}{A}\mathrm{erf}(z\gamma^{\frac{1}{2}}) \ . \end{aligned} \tag{177}$$

Note finally that in things such as 2D bilayer systems (electrons in one layer, holes in the other, say) there is an additional 'capacitor term' due to interaction of the backgrounds.

19.4.3 1D Coulomb interaction

Coulomb sums in systems that are periodic in one dimension (the x-direction, according to CASINO) are performed using an algorithm based on the Euler–Maclaurin summation formula. Physically, a point charge is assumed to be accompanied by a line of neutralizing background parallel to the periodic axis, so that each periodic cell is electrically neutral and hence the electrostatic potential at each point in space away from the line of neutralizing charge is given by an absolutely convergent series. The first few terms in the series are evaluated explicitly; the rest of the series is approximated using a second-order Euler-Maclaurin formula. The unwanted contribution of the neutralizing background to the Coulomb potential diverges logarithmically as the line of neutralizing charge is approached. This logarithmic divergence is subtracted out of the electrostatic potential, thus removing the effects of the neutralizing background from the electrostatic potential energy of an electroneutral cell.

Such systems are done infrequently enough that it probably isn't worth writing out all the theory here. Please refer to Ref. [51], particularly noting Eq. (4.8) of that paper. Note, however, that the constant term $2\log(a)/a$ in Eq. (4.8), where a is the supercell lattice constant, is omitted from the 1D-periodic Coulomb interaction in CASINO. This term was added "by hand" to the Coulomb potential $V(r)$ in Ref. [51] [see the discussion after Eq. (2.5) of that paper] in order to give the property $V(sr) = V(r)/|s|$ for all nonzero s, and the term has no effect in a system with an explicitly electroneutral basis. However, for a system with an implicit neutralizing background, this term breaks the size consistency of the Coulomb sums and introduces a logarithmic divergence into the interaction energy as a function of system size.

The number of terms in the explicit sum over periodic images is controlled by the **ewald_control** parameter. The default number of terms in the explicit sum is $2 \times 3 + 1 = 7$, leading to fractional errors in Coulomb interactions of between 10^{-6} and 10^{-5}. If starting a new project on 1D-periodic systems it might be worthwhile to investigate increasing **ewald_control**.

Note that the first derivatives of the 1D Coulomb interaction have never been implemented in CASINO, so core-polarization potentials may not be used in one-dimensionally periodic systems with atoms.

19.4.4 MPC interaction

The model periodic Coulomb (MPC) interaction [52, 53, 42] is used to reduce finite size effects in periodic calculations. The exact MPC interaction operator is

$$\begin{aligned}\hat{H}^{\rm exact}_{e-e} &= \sum_{i>j} f(\mathbf{r}_i-\mathbf{r}_j) + \sum_i \int_{\rm WS} \rho(\mathbf{r})\left[v_{\rm E}(\mathbf{r}_i-\mathbf{r}) - f(\mathbf{r}_i-\mathbf{r})\right]\,d\mathbf{r} \\ &\quad -\frac{1}{2}\int_{\rm WS} \rho(\mathbf{r})\rho(\mathbf{r}')\left[v_{\rm E}(\mathbf{r}-\mathbf{r}') - f(\mathbf{r}-\mathbf{r}')\right]\,d\mathbf{r}\,d\mathbf{r}' \ , \end{aligned} \tag{178}$$

where $f(\mathbf{r})$ is the $1/r$ Coulomb interaction treated within the 'minimum-image' convention, which corresponds to reducing the vector $\mathbf{r}$ into the Wigner-Seitz (WS) cell of the simulation cell, $v_{\rm E}$ is the Ewald potential, and ρ is the electronic charge density from the many-electron wave function Ψ. The electron–electron interaction energy is

$$E^{\rm exact}_{\rm e-e} = \langle\Psi|\hat{H}^{\rm exact}_{e-e}|\Psi\rangle \ , \tag{179}$$

which gives

$$\begin{aligned} E^{\rm exact}_{\rm e-e} &= \frac{1}{2}\int_{\rm WS} \rho(\mathbf{r})\rho(\mathbf{r}')v_{\rm E}(\mathbf{r}-\mathbf{r}')\,d\mathbf{r}\,d\mathbf{r}' \\ &\quad + \left(\int_{\rm WS} |\Psi|^2 \sum_{i>j} f(\mathbf{r}_i-\mathbf{r}_j)\,\Pi_k\,{\rm d}\mathbf{r}_k - \frac{1}{2}\int_{\rm WS} \rho(\mathbf{r})\rho(\mathbf{r}')f(\mathbf{r}-\mathbf{r}')\,d\mathbf{r}\,d\mathbf{r}'\right) \ , \end{aligned} \tag{180}$$

where the first term on the right-hand side is the Hartree energy and the term in brackets is the XC energy. We can see that the Hartree energy is calculated with the Ewald interaction while the XC energy (expressed as the difference between a full Coulomb term and a Hartree term) is calculated with the cutoff interaction f.

In a DMC calculation we require the local energy at every step, but we only know the DMC charge density, ρ, at the end of the run. Normally we have a good approximation to the charge density, $\rho_{\rm A}$, either from an independent particle calculation or a VMC calculation. We can avoid the need to know ρ exactly by constructing a new interaction operator which involves only $\rho_{\rm A}$,

$$\hat{H}_{e-e} = \sum_{i>j} f(\mathbf{r}_i-\mathbf{r}_j) + \sum_i \int_{\rm WS} \rho_{\rm A}(\mathbf{r})\left[v_{\rm E}(\mathbf{r}_i-\mathbf{r}) - f(\mathbf{r}_i-\mathbf{r})\right]\,d\mathbf{r}$$

$$- \frac{1}{2}\int_{\mathrm{WS}} \rho_{\mathrm{A}}(\mathbf{r})\rho_{\mathrm{A}}(\mathbf{r}')\left[v_{\mathrm{E}}(\mathbf{r}-\mathbf{r}') - f(\mathbf{r}-\mathbf{r}')\right] d\mathbf{r}\, d\mathbf{r}' \; . \tag{181}$$

The interaction energy becomes

$$\begin{aligned} E_{\mathrm{e-e}} &= \langle\Psi|\hat{H}_{e-e}|\Psi\rangle \\ &= \int_{\mathrm{WS}} \rho(\mathbf{r})\rho_{\mathrm{A}}(\mathbf{r}')v_{\mathrm{E}}(\mathbf{r}-\mathbf{r}')\, d\mathbf{r}\, d\mathbf{r}' - \frac{1}{2}\int_{\mathrm{WS}} \rho_{\mathrm{A}}(\mathbf{r})\rho_{\mathrm{A}}(\mathbf{r}')v_{\mathrm{E}}(\mathbf{r}-\mathbf{r}')\, d\mathbf{r}\, d\mathbf{r}' \\ &\quad + \int_{\mathrm{WS}} |\Psi|^2 \sum_{i>j} f(\mathbf{r}_i - \mathbf{r}_j)\, \Pi_k\, d\mathbf{r}_k - \int_{\mathrm{WS}} \rho(\mathbf{r})\rho_{\mathrm{A}}(\mathbf{r}')f(\mathbf{r}-\mathbf{r}')\, d\mathbf{r}\, d\mathbf{r}' \\ &\quad + \frac{1}{2}\int_{\mathrm{WS}} \rho_{\mathrm{A}}(\mathbf{r})\rho_{\mathrm{A}}(\mathbf{r}')f(\mathbf{r}-\mathbf{r}')\, d\mathbf{r}\, d\mathbf{r}'. \end{aligned} \tag{182}$$

Noting that

$$E_{\mathrm{e-e}} = E_{\mathrm{e-e}}^{\mathrm{exact}} - \frac{1}{2}\int_{\mathrm{WS}} \left[\rho(\mathbf{r}) - \rho_{\mathrm{A}}(\mathbf{r})\right]\left[\rho(\mathbf{r}') - \rho_{\mathrm{A}}(\mathbf{r}')\right]\left[v_{\mathrm{E}}(\mathbf{r}-\mathbf{r}') - f(\mathbf{r}-\mathbf{r}')\right] d\mathbf{r}\, d\mathbf{r}' \; , \tag{183}$$

we see that the error due to this approximation is second order in $(\rho - \rho_{\mathrm{A}})$, and in addition the operator $(v_{\mathrm{E}} - f)$ becomes very small as the size of the simulation cell goes to infinity. The error term is therefore usually small and is neglected although it could be calculated after the simulation. We use the MPC expressions of Eqs. (181) and (182) in both VMC and DMC calculations.

The first term of the Hamiltonian of Eq. (181) is evaluated in real space and the second term in Fourier space. The third term is a constant which is evaluated in reciprocal space at the start of the calculation. Introducing the Fourier transformed quantities,

$$f_{\mathbf{G}} = \frac{1}{\Omega}\int_{\mathrm{WS}} f(\mathbf{r})e^{i\mathbf{G}\cdot\mathbf{r}} d\mathbf{r} \; , \tag{184}$$

$$\rho_{\mathbf{G}} = \frac{1}{\Omega}\int_{\mathrm{WS}} \rho(\mathbf{r})e^{i\mathbf{G}\cdot\mathbf{r}} d\mathbf{r} \; , \tag{185}$$

where Ω is the volume of the cell, and noting that the Fourier transform of the Ewald interaction is $4\pi/(\Omega G^2)$, we have

$$\hat{H}_{e-e} = \sum_{i>j} f(\mathbf{r}_i - \mathbf{r}_j) + \Omega \sum_i \sum_{\mathbf{G}\neq 0} \left[\frac{4\pi}{\Omega G^2} - f_{\mathbf{G}}\right] \rho_{A,\mathbf{G}} e^{-i\mathbf{G}\cdot\mathbf{r}_i} - \Omega \sum_i f_{\mathbf{G}=0}\rho_{A,\mathbf{G}=0} - C \; , \tag{186}$$

where

$$C = \frac{\Omega^2}{2}\sum_{\mathbf{G}\neq 0}\left[\frac{4\pi}{\Omega G^2} - f_{\mathbf{G}}\right]\rho^*_{A,\mathbf{G}}\rho_{A,\mathbf{G}} - \frac{\Omega^2}{2} f_{\mathbf{G}=0}\rho^*_{A,\mathbf{G}=0}\rho_{A,\mathbf{G}=0} \tag{187}$$

The calculation of $f_{\mathbf{G}}$ is achieved using the following scheme developed by Randy Hood. The integrand in Eq. (184) diverges at the origin and we separate out the divergent behaviour by writing

$$f(\mathbf{r}) = g(\mathbf{r}) + h(\mathbf{r}) \tag{188}$$

where

$$g(\mathbf{r}) = \begin{cases} y(r) & r < L \\ 1/r & r > L \; , \end{cases} \tag{189}$$

$$\tag{190}$$

$$h(\mathbf{r}) = \begin{cases} 1/r - y(r) & r < L \\ 0 & r > L \; , \end{cases} \tag{191}$$

and L is the radius of the largest sphere which is contained within the WS cell and $y(r)$ is chosen to be

$$y(r) = -\frac{r^2}{2L^3} + \frac{3}{2L} \; , \tag{192}$$

so that both g and h have continuous first derivatives at $r = L$. The Fourier transform of $h(\mathbf{r})$ is calculated analytically as

$$\begin{aligned} h_{\mathbf{G}} &= \frac{1}{\Omega}\int_0^L \int_{-1}^{+1} \left(\frac{1}{r} + \frac{r^2}{2L^3} - \frac{3}{2L}\right) 2\pi r^2 e^{iGr\cos(\theta)}\, d\cos(\theta)\, dr \\ &= \frac{4\pi}{\Omega G^2} + \frac{12\pi}{\Omega L^2 G^4}\left[\cos GL - \frac{\sin(GL)}{GL}\right] \; , \end{aligned} \tag{193}$$

from which the $G = 0$ value can be extracted as

$$h_{\mathbf{G}=0} = \frac{2\pi L^2}{5\Omega} \ , \tag{194}$$

The Fourier transform of $g(\mathbf{r})$ must be evaluated numerically. The gradient of $g(\mathbf{r})$ is discontinuous at the boundary of the WS cell. The errors in the Fourier components obtained using the fast Fourier transform (FFT) method therefore fall off slowly, going approximately as $N_{\rm grid}^{-2}$, where $N_{\rm grid}$ is the number of FFT grid points along each lattice vector. It is assumed that the FFT Fourier coefficients of g satisfy

$$g_{\mathbf{G}}^{N_{\rm grid}} = a_{\mathbf{G}} N_{\rm grid}^{-4} + b_{\mathbf{G}} N_{\rm grid}^{-2} + c_{\mathbf{G}}, \tag{195}$$

where the $\{a_{\rm G}\}$, $\{b_{\rm G}\}$ and $\{c_{\rm G}\}$ are constants. Note that $c_{\rm G} = g_{\mathbf{G}}^{\infty}$, the value of the Fourier coefficient in the limit of an infinite grid. FFTs are carried out at three different grid sizes in order to determine the $\{g_{\rm G}^{\infty}\}$.

To investigate whether the extrapolation has been successful, $a_{\mathbf{G}}$ is set to 0, and the FFT data for the two finest grids are used to extrapolate to infinite system size. CASINO prints out the $\mathbf{G} = \mathbf{0}$ coefficients obtained by these two extrapolation methods (referred to as 'quadratic' and 'linear' extrapolation, respectively). CASINO also displays the extrapolated coefficients for which the disagreement between the two schemes is largest. The user should examine these data, and verify that the extrapolation has been successful.

Once the extrapolated Fourier coefficients of g have been obtained by running CASINO with **runtype** set to 'gen_mpc', they are added to the analytic Fourier coefficients of f and stored in the file `mpc.data`. Note that, for historical reasons, the coefficients stored in `mpc.data` are actually $\Omega f_{\mathbf{G}}$.

Running a 'gen_mpc' calculation also evaluates the Fourier components of the charge density, using the orbitals in `xwfn.data` to evaluate the density. This assumes that the orbitals are (i) orthogonal and (ii) consistently normalized by the program that generated them. The **complex_wf** keyword must be set to `T` and any cusp corrections must be switched off in a 'gen_mpc' calculation, in order to preserve the normalization of the orbitals. The orbitals used to evaluate the charge density are the orbitals that are occupied in the first determinant in a multideterminant wave function.

20 Model interactions

20.1 Manual interactions block

When using a model particle–particle interaction, the **interaction** keyword should be set to 'manual'. This should be done, for instance, when describing the interaction between fermions in a Bose–Einstein condensate. The form of the interaction is set in the **manual_interaction** block. Currently, the first line of the block can be set to 'square_well', 'poschl_teller', 'hard_sphere', 'polynomial', 'logarithmic', 'keldysh' or 'clifford' (other model interactions could easily be implemented). The **manual_interactions** block is also used to provide the screening parameter r_* required when the **interaction** is 'ewald_kel'.

20.2 Square-well interaction

For the square well the subsequent lines in the **manual_interaction** block should be set to: `width` and `height`, where `height` is magnitude of the interaction, V_0, (negative for a well) and `width` is the extent of the interaction of the radial well, L.

$$\begin{aligned} v(r) &= V_0 \quad r \le L \\ &= 0 \quad r > L \end{aligned} \tag{196}$$

The interaction exists only between particles of opposite spin (to simulate zero-range interactions). Note that the square well is the one used by Astrakharchik *et al.* [54].

20.3 Modified Pöschl-Teller interaction

For the modified Pöschl-Teller interaction the well has the form

$$v(r) = \frac{V_0\mu^2}{\mu_m \cosh^2(\mu r)}, \tag{197}$$

where r is the distance between the two particles, μ_m is the reduced mass and V_0 is the strength of the interaction. As with the square well, the interaction exists only between particles of opposite spin. Note that currently CASINO only calculates this for particles of the same mass; hence Eq. (197) has the substitution $\mu_m = m/2$, which is the same as the method of Carlson *et al.* [55]. Again $V_0 < 0$ implies a potential well.

20.4 Hard sphere

The condition that the wave function goes linearly to zero whenever two hard-sphere particles come into contact with each other is imposed by including a term $u(r) = \log\{\tanh[(\frac{r}{D} - 1)/(1 - \frac{r}{L})]\}$ in the two-body Jastrow factor, where D is the hard-sphere diameter, and L is a cutoff length normally chosen to be the Wigner-Seitz radius of the simulation cell. Moves that bring two hard-sphere particles within D of each other are rejected. Note that, unlike the square well and Pöschl-Teller interactions, the hard sphere potential acts between all particles (not just particles with anti-parallel spin) unless the **op_spins** keyword is present in the **manual_interaction** block.

20.5 Polynomial

This allows the specification of a spherically symmetric interparticle interaction $V(r) = \Theta(r - L)p(r)$, where p is a polynomial, L is a cutoff distance and Θ is the Heaviside step function. The interaction acts only between opposite-spin particles. This can be used to implement interparticle pseudopotentials [56]. The input block takes the following format,

```
%block manual_interaction
polynomial
order : 5
cutoff : 2.0
c_0 : 1.0
c_2 : -0.5
c_4 : 0.3
c_5 : -0.1
%endblock manual_interaction
```

The `order` keyword should be the order of the polynomial (the highest power present in the polynomial), and the keyword `c_i` identifies the coefficient for the term r^i in the polynomial. Omitted coefficients are given a default value of zero.

20.6 Interactions between charges in 2D semiconductors

20.6.1 Theoretical background

Suppose a charge density $\rho(x, y)\delta(z)$ is placed in a 2D semiconductor at $z = 0$. The resulting electric displacement field is $\mathbf{D} = \mathbf{E}/(4\pi) + \mathbf{P} = -\nabla\phi/(4\pi) + \mathbf{P}$, where $\mathbf{E}$ is the electric field, ϕ is the electrostatic potential and $\mathbf{P} = \mathbf{P}_\perp(x, y)\delta(z)$ is the polarization field, where $\mathbf{P}_\perp(x, y)$ is the in-plane polarization. Note that $\mathbf{P}_\perp$ has no component in the z direction because the charge lies in-plane. Now $\nabla \cdot \mathbf{D} = \rho\delta(z)$ by Gauss's law, and hence

$$\nabla^2\phi = -4\pi\rho\delta(z) + 4\pi\nabla \cdot \mathbf{P} = -4\pi\rho\delta(z) + 4\pi(\nabla \cdot \mathbf{P}_\perp)\delta(z). \tag{198}$$

But $\mathbf{P}_\perp(x, y) = -\kappa\nabla[\phi(x, y, 0)]$, where κ is the in-plane susceptibility of the material. So

$$\nabla^2\phi = -4\pi\rho\delta(z) - 4\pi\kappa \left(\nabla^2[\phi(x, y, 0)]\right) \delta(z). \tag{199}$$

If we take the Fourier transform of Eq. (199), using $\mathbf{q}$ for the wavevector in the (x, y) plane and k for the wavenumber in the z direction and rearrange, we find

$$\phi(\mathbf{q}, k) = 4\pi \frac{\rho(\mathbf{q}) - \kappa q^2 \phi(\mathbf{q}, z = 0)}{q^2 + k^2}. \tag{200}$$

But

$$\begin{aligned} \phi(\mathbf{q}, z = 0) &= \frac{1}{2\pi} \int \phi(\mathbf{q}, k)\, dk \\ &= \frac{2\pi}{q} \left[\rho(\mathbf{q}) - \kappa q^2 \phi(\mathbf{q}, z = 0)\right]. \end{aligned} \tag{201}$$

Rearranging for $\phi(\mathbf{q}, z = 0)$, we find the in-plane electrostatic potential to be

$$\phi(\mathbf{q}, z = 0) = \frac{2\pi\rho(\mathbf{q})}{q(1 + 2\pi\kappa q)}. \tag{202}$$

Hence the interaction between charges in a 2D semiconductor is [57]

$$v(q) = \frac{2\pi}{q(1 + 2\pi\kappa q)} = \frac{2\pi}{q(1 + r_* q)}, \tag{203}$$

where $r_* \equiv 2\pi\kappa$. Taking the Fourier transform, we find

$$v(r) = \frac{\pi}{2r_*} \left[H_0(r/r_*) - Y_0(r/r_*)\right], \tag{204}$$

where $H_n(x)$ is a Struve function and $Y_n(x)$ is a Bessel function of the second kind. This expression was first obtained by Keldysh [58].

At long range ($r \gg r_*$), the effective interaction reduces to the usual Coulomb form: $v(r) \approx 1/r$. On the other hand, at short range ($r \ll r_*$), the effective interaction is approximately logarithmic:

$$v(r) \approx [\log(2r_*/r) - \gamma]/r_*, \tag{205}$$

where γ is Euler's constant [57].

In order to satisfy the analogue of the Kato cusp conditions (i.e., to make the local energy nondivergent at coalescence points), we require that the two-body Jastrow term between two distinguishable particles of mass m_i and m_j and charge q_i and q_j must go as $-q_i q_j m_i m_j r^2 \log(r)/[2r_*(m_i + m_j)]$ at short range. The two-body Jastrow term between pairs of indistinguishable particles of mass m and charge q must go as $-mq^2 r^2 \log(r)/(8r_*)$ at short range. These cusp conditions can be imposed using the 'EXJAS' Jastrow term in CASINO: see Sec. 7.4.3.

When using the logarithmic interaction, it is simpler to use a system of units in which $\hbar = e = \mu = 4\pi\epsilon r_* = 1$, where ϵ is the absolute permittivity and μ is the electron–hole reduced mass. In these units, r_* vanishes from the denominator of Eq. (205). The resulting energies are in units of $E_0 = e^2/(4\pi\epsilon r_*)$, and lengths are in units of $\sqrt{2} r_0$, where $r_0 = \sqrt{4\pi\epsilon r_* \hbar^2/(2e^2\mu)}$. In these units the energy only depends on the electron–hole mass ratio and the r_* value; furthermore, the dependence of the energy on the r_* value is a trivial, constant offset of $\sum_{i>j} q_i q_j \log(r_*)$. CASINO assumes that these "logarithmic excitonic units" are used.

20.6.2 Choosing the logarithmic or Keldysh interaction manual interactions

Two 'manual interactions' are available for 2D semiconductors: 'logarithmic' and 'keldysh', where the former evaluates the logarithmic approximation of Eq. (205) while the latter evaluates the full interaction of Eq. (204). The value of the r_* parameter must be set in the **manual_interaction** block as `rstar`.

20.6.3 Evaluating the Keldysh interaction

Equation (204) is approximated using a Taylor expansion in r/r_* at small r and an expansion in r_*/r at large r. The small-r expansion is used for $r/r_* <= 18$ and the large-r expansion is used for $r/r_* > 18$. This allows us to evaluate the interaction accurately to at least 8 digits of precision using double-precision arithmetic, as demonstrated by comparing with results obtained using higher precision. The small-r expansion of Eq. (204) is

$$v(r) = \frac{1}{r_*}\sum_{i=0}^{\infty} \frac{(-1)^{i+1}(r/r_*)^{2i}}{\prod_{j=1}^{i}(2j)^2}\left[\log\left(\frac{r}{2r_*}\right)+\gamma-\sum_{k=1}^{i}\frac{1}{k}\right]+\frac{1}{r_*}\sum_{i=0}^{\infty}\frac{(-1)^i(r/r_*)^{2i+1}}{\prod_{j=0}^{i}(2j+1)^2}. \tag{206}$$

The large-r expansion of Eq. (204) is

$$v(r) = \frac{1}{r}\sum_{i=0}^{\infty}(-1)^i\prod_{j=0}^{i-1}(2j+1)^2\left(\frac{r_*}{r}\right)^{2i}. \tag{207}$$

This series does not converge for finite values of r; hence only a few terms should be used to approximate the potential at finite r.

20.6.4 Multiple layers of 2D semiconductor: `2D_multilayer_int`

For an N-layer heterostructure of 2D semiconductors, the interactions between charges within layers and between layers can be determined by solving Poisson's equation, analogously with the monolayer case. Consider such a heterostructure of 2D semiconductors comprised of N parallel layers (labelled $i = \{1, 2, \ldots, N\}$), each having in-plane susceptibility κ_i and being located at fixed z-coordinates $z = d_i$. Suppose also that this heterostructure is immersed in an isotropic medium of dielectric constant ϵ. In this general case we will assume no symmetry whatsoever, and therefore we must determine $N(N-1)/2$ interlayer and N intralayer charge–charge interaction potentials.[22]

In order to determine the intralayer potential between pairs of charges in layer j, we must consider the presence of a test charge density, which is confined to layer j:

$$\rho^j_{tot}(\mathbf{r}, z) = \rho^j(\mathbf{r})\delta(z-d_j). \tag{208}$$

The electric displacement field throughout the rest of space due to this density, and because of screening in the other layers is then

$$\begin{aligned}\mathbf{D} = & -\ \epsilon\nabla\phi(\mathbf{r},z)\\ & -\ \sum_i \kappa_i[\nabla_\parallel\phi(\mathbf{r},d_i)]\delta(z-d_i),\end{aligned} \tag{209}$$

which, upon use of Gauss's law, yields

$$\begin{aligned}\rho^j(\mathbf{r},z)\delta(z-d_j) = & -\ \epsilon\nabla^2\phi(\mathbf{r},z)\\ & -\ \sum_i \kappa_i[\nabla^2_\parallel\phi(\mathbf{r},d_i)]\delta(z-d_i).\end{aligned} \tag{210}$$

After taking the Fourier transform of this expression we find

$$\begin{aligned}\rho^j(\mathbf{q})\exp(-ikd_j) = \epsilon(q^2+k^2)\phi(\mathbf{q},k) + \\ q^2\sum_i \kappa_i\phi(\mathbf{q},d_i)\exp(-ikd_i),\end{aligned} \tag{211}$$

which, after Fourier inversion in the k variable only, gives

$$\begin{aligned}\rho^j(\mathbf{q})\exp(-q|z-d_j|) = 2\epsilon q\phi(\mathbf{q},z) + \\ q^2\sum_i \kappa_i\phi(\mathbf{q},d_i)\exp(-q|z-d_i|).\end{aligned} \tag{212}$$

Evaluating Eq. (212) at each of the layer z coordinates d_l ($l = \{1, 2, \ldots, N\}$), we can write

$$\begin{aligned}\rho^j\exp(-q|d_l-d_j|) = q(2\epsilon-\kappa_l q)\phi(\mathbf{q},d_l) + \\ q^2\sum_{i\neq l} \kappa_i\phi(\mathbf{q},d_i)\exp(-q|d_l-d_i|),\end{aligned} \tag{213}$$

[22]There are $\binom{N}{2} = N(N-1)/2$ possible pairings of individual layers, each with a corresponding interaction potential between the two layers in question. Additionally, there are N individual layers, and therefore N intralayer interaction potentials describing interactions between charges in the same layer.

which is a matrix equation

$$\rho_l^j(\mathbf{q}) = M_{li}(\mathbf{q})\phi_i(\mathbf{q}), \tag{214}$$

with the following definitions

$$\begin{aligned} \rho_l^j(\mathbf{q}) &= \rho^j(\mathbf{q})\exp\left(-q|d_l - d_j|\right), \\ \phi_l(\mathbf{q}) &= \phi(\mathbf{q}, d_l), \\ M_{li} &= \begin{cases} q(2\epsilon + \kappa_l q) & \text{if } i = l \\ q^2\kappa_i \exp\left(-q|d_l - d_i|\right) & \text{otherwise} \end{cases}. \end{aligned} \tag{215}$$

The solution to the matrix Eq. (214) is a set of ϕ_i which are equal to $\rho^j(\mathbf{q}) \times v_{ji}(\mathbf{q})$, with $v_{ji}(\mathbf{q})$ the Fourier components of the interaction potential between layer j and layer i. If $j = i$, then this is just the intralayer interaction in layer j. This procedure should, in general, be repeated for $j = 1, 2, \ldots, N$—however, if there is sufficient symmetry (e.g. a mirror symmetry about an imaginary plane through the centre of the heterostructure), only a subset of j values will require solution of Eq. (214).

For completeness, we note that the same analysis applies in the case that the surrounding dielectric medium is anisotropic, having dielectric tensor

$$\tilde{\epsilon} = \text{diag}(\epsilon_\parallel, \epsilon_\parallel, \epsilon_\perp) \tag{216}$$

with the caveat that the substitutions

$$d \quad \rightarrow D = \sqrt{\epsilon_\parallel/\epsilon_\perp}d, \tag{217}$$

$$\epsilon \quad \rightarrow \bar{\epsilon} = \sqrt{\epsilon_\parallel\epsilon_\perp}, \tag{218}$$

are also made.

The multilayer interaction can be used by specifying an appropriate manual interactions block. For example, suppose we have set up a multilayer system with charge in layers 1, 2, 3, and 4 of an electron gas system (using **heg_zlayer** to specify layer z coordinates, and **heg_layer** to specify particle layer occupation). We might specify the multilayer interaction with a manual interaction block of the form:

```
%block manual_interaction
keldysh_multilayer
rstar1 : 100.d0
rstar2 : 100.d0
rstar3 : 100.d0
rstar4 : 30.d0
grid_size : 10000
grid_mult : 10.d0
%endblock manual_interaction
```

or, equivalently, we could use the **rstardef** keyword to set the r_* parameter associated with any unspecified layers to a fixed value:

```
%block manual_interaction
keldysh_multilayer
rstardef : 100.d0
rstar4 : 30.d0
grid_size : 10000
grid_mult : 10.d0
%endblock manual_interaction
```

the two grid parameters **grid_size** and **grid_mult** may be used, as well as the bilayer equivalents (**bilayer_grid_size** and **bilayer_grid_mult**), to specify the spatial extent and density of the grid on which the interactions are evaluated. Their given values (10000 and 10.0, respectively) are the defaults.

20.6.5 The bilayer case: `2D_bilayer_int`

In the bilayer case ($N = 2$), it is possible to solve the matrix Eq. (214) by hand (analytical solution is also possible for $N = 3$ and 4, but no higher). With $j = 1$ labelling the position of our test charge density, the intra- (ϕ_1) and inter-layer (ϕ_2) potentials read

$$\begin{aligned}\phi_1(\mathbf{q}) &= \frac{(1+r_{*2}q)\exp(qd) - r_{*2}q\exp(-qd)}{2\epsilon q\left[(1+r_{*1}q)(1+r_{*2}q)\exp(qd) - r_{*1}r_{*2}q^2\exp(-qd)\right]}, \\ \phi_2(\mathbf{q}) &= \frac{1}{2\epsilon q\left[(1+r_{*1}q)(1+r_{*2}q)\exp(qd) - r_{*1}r_{*2}q^2\exp(-qd)\right]},\end{aligned} \tag{219}$$

where we have made the definition $r_{*i} = \kappa_i/2\epsilon$. An important difference between the intralayer interaction and the monolayer screened interaction (the Keldysh interaction) is that the intralayer interaction between charges accounts for the screening provided by the (empty) second layer.

CASINO is capable of evaluating these potentials in real space, by calculation of their 1D Hankel transform, which is equivalent to 2D Fourier transformation in the case of radial symmetry. If one wishes to use the exact bilayer interactions, one should use the manual interaction specifiers, with the interaction type `2D_bilayer_int`, and two manual interaction parameters, r_{*1} (the r_* parameter for layer 1) and r_{*2} (the r_* parameter for layer 2). If one would like to use the adapted interactions (in the presence of an anisotropic dielectric tensor as specified in the preceeding section), then one will have to manually scale one's input parameter values, and use a sensible set of excitonic units. Keldysh or logarithmic cusp conditions are applied to Jastrow parameter terms in cases where the interactions become singular (i.e. for the intralayer cases at zero separation).

20.6.6 Approximate multilayer potentials

If there are multiple layers of 2D semiconductor then it can be shown that the interaction between charge carriers at large in-plane distances is of monolayer Keldysh form [Eq. (204)], but with an effective r_* value that is the sum of the r_* values for the individual layers, plus the absolute value of the interlayer separation. This approximation breaks down at short range. It is valid provided the layer separation is small compared with the physical size of the excitonic complex being studied. Meanwhile, if the interaction is `keldysh` or `logarithmic` and multiple layers are present then the interaction between charge carriers will be of monolayer Keldysh form with an effective r_* value as described above. Note that the r_* value supplied to CASINO in the **manual_interactions** block should be the sum of the r_* values for each of the individual 2D semiconductors present in the multilayer.

20.6.7 Ewald–Keldysh interaction for periodic 2D systems

The difference between the Keldysh interaction and the Coulomb $1/r$ interaction goes as r^{-3}. If this difference is summed over all periodic images of a pair of particles then we get an absolutely convergent series which gives a Keldysh correction to the ordinary 2D Ewald energy. The resulting Ewald–Keldysh energy is appropriate for modelling homogeneous electron(–hole) gases in 2D semiconductors. To use the Ewald–Keldysh interaction in CASINO, set **interaction** to 'ewald_kel' and add a **manual_interaction** block to `input` with two lines (e.g., 'ewald_kel', then 'rstar : 1.7858').

20.7 Dipolar interaction

The dipolar interaction potential models particles moving in 2D interacting via magnetic or electric dipole moments in the out-of-plane z direction. (If you know what you are doing you can also use these interactions in 3D, but CASINO still assumes that the confinement of the particles is such that 2D cusp conditions are appropriate.) The dipolar interaction is selected by writing `dipole` in the first line of the `manual_interaction` block in the `input` file. The dipole strength d^2 (i.e., the square of the dipole moment) is specified in the `manual_interaction` block using keyword `d^2`.

A generalised version of the dipolar interaction for particles with dipole moments aligned at polar angle θ and azimuthal angle ϕ is also available. This interaction is selected by writing `tilted_dipole` as the first line of the `manual_interaction` block in the `input` file, and the dipole strength d^2 and the angles θ and ϕ are specified using the keywords `d^2`, `theta` and `phi`, respectively. For a tilted dipolar interaction, one can optionally supply two additional parameters `c_6` and `c_12` defining an additional pairwise Lennard-Jones potential $(c_{12}r^{-6} - c_6)r^{-6}$, intended to prevent collapse of the gas of particles due to the attractive region of the tilted dipolar potential.

Note that a special "cusp" condition appropriate for these interactions may be applied to the *gjastrow* Jastrow factor by including a `dipole cusp` term in the `parameters.casl` file. For the standard

Jastrow factor, "cusp" conditions (see Sec. 32) are applied whenever a two-body U term is included in the Jastrow factor. (Note that the cusp conditions are not exact for tilted dipolar interactions, unless they are regularized by the Lennard-Jones potential.)

20.8 Pseudodipolar interaction

A pseudopotential for the dipolar interaction is also available. The interaction potential is a polynomial in r for $r < L$ and d^2/r^3 for $r > L$. This interaction can be used by writing `pseudodipole` as the first line of the `manual_interaction` block in the `input` file, which takes the same parameters as the `polynomial` interaction described above and the interaction strength d^2 using keyword `d^2`.

A pseudopotential for the anisotropic tilted dipolar interaction is also available, whose input block takes the form

```
%block manual_interaction
tilted_pseudodipole
order : 5
tilt_order : 4
c_0 : 1
c_5 : 1
b_0 : 2
b_4 : 1
d^2 : 0.25
theta : 0.615
%endblock manual_interaction
```

The `tilt_order` keyword gives the order of the anisotropic part of the polynomial, with coefficients for $r^i \cos 2\phi$ given by `b_`i, and ϕ is the angle between the inter-particle separation and the x-axis.

20.9 Clifford interaction

The Clifford interaction is an alternative to the Ewald interaction for periodic Coulomb systems. At present it is available for homogeneous electron gases. Let $\mathbf{r} = \sum_{i=1}^{3} x_i \mathbf{e}_i$ be the separation of two electrons. The separation vector of the two electrons in the Clifford torus embedding space is

$$x'_i = \sum_{j=1}^{3} 2B_{ij}^{-1} \sin\left(\sum_{k=1}^{3} \frac{1}{2} B_{jk} x_k\right) , \tag{220}$$

where B_{ij} is the jth Cartesian component of the ith simulation-cell reciprocal lattice vectors. The Coulomb interaction between the two electrons is then taken to be $v(\mathbf{r}) = 1/|\mathbf{r}'|$. This is a generalization of Eq. (23) of Ref. [59]. Note that at short range, $\mathbf{r}' = \mathbf{r} + \mathcal{O}(r^3/a^2)$, where a is the lattice parameter, so that the short-range interaction converges to $1/r$. Furthermore, it is easy to show that $v(\mathbf{r} + \mathbf{R}) = v(\mathbf{r})$ for all simulation-cell lattice vectors $\mathbf{R}$.

For HEG systems, an additional constant term due to (i) the electrostatic energy of the neutralizing background and (ii) the interaction between the electrons and the background should be included in the total energy [59]. Evaluating this term requires the numerical calculation of low-dimensional integrals. It is currently left to the user to evaluate this constant energy contribution using an external program; it is not included in the energy reported by CASINO when the Clifford interaction is used.

21 Multi-determinant expansions

CASINO is capable of using multi-determinant expansions of the form

$$\Psi_{\mathrm{MD}}(\mathbf{R}) = \sum_n c_n D_n^{\uparrow} D_n^{\downarrow} , \tag{221}$$

where $\{c_n\}$ are the determinant coefficients. Multi-determinant expansions are widely used in quantum Chemistry methods and successfully describe correlations missing from the single-determinant wave function for small systems. However, the length of the expansion required to achieve a given error in the energy grows rapidly with system size, and for large systems multi-determinant expansions are impractical.

In QMC one has the ability to use Jastrow factors (see Sec. 22), which do a good job at describing electronic correlation. As a consequence, it is possible to use small multi-determinant expansions in QMC and attain an excellent description of electronic correlations.

21.1 Compressed multi-determinant expansions

It is possible to apply certain algebraic tricks to a multi-determinant expansion in order to reduce it to a much shorter expansion where the determinants are populated with modified orbitals. This substantially reduces the cost of using large expansions in QMC. The compression method is described in Ref. [31].

To use a multi-determinant wave function in CASINO, you will need to:

- Run CASINO with **runtype** set to `gen_mdet_casl` to produce an `mdet.casl` file describing the multi-determinant expansion.
- Run the **det_compress** utility to produce a `cmdet.casl` file describing the compressed multi-determinant expansion.
- Run CASINO as usual with the `cmdet.casl` file produced by **det_compress** in the directory where CASINO is being run.

22 The Jastrow factor

The Slater-Jastrow wave function is

$$\Psi(\mathbf{R}) = \exp\left[J\right] \sum_n c_n D_n^{\uparrow} D_n^{\downarrow} \;, \tag{222}$$

where the D_n are determinants of up and down spin orbitals and J is the Jastrow factor.

22.1 General form of CASINO's Jastrow factor

CASINO uses the form of Jastrow factor proposed in Ref. [60]. CASINO's Jastrow factor is the sum of homogeneous, isotropic electron–electron terms u, a homogeneous, isotropic electron–electron–electron term W, isotropic electron–nucleus terms χ centred on the nuclei, isotropic electron–electron–nucleus terms f, also centred on the nuclei and, in periodic systems, plane-wave expansions of electron–electron separation and electron position, p and q. The form is

$$\begin{aligned} J(\{\mathbf{r}_i\},\{\mathbf{r}_I\}) \quad = \quad & \sum_{i=1}^{N-1}\sum_{j=i+1}^{N} u(r_{ij}) + W(\{r_{ij}\}) + \sum_{I=1}^{N_{\rm ions}}\sum_{i=1}^{N}\chi_I(r_{iI}) + \sum_{I=1}^{N_{\rm ions}}\sum_{i=1}^{N-1}\sum_{j=i+1}^{N} f_I(r_{iI},r_{jI},r_{ij}) \\ & + \sum_{i=1}^{N-1}\sum_{j=i+1}^{N} p(\mathbf{r}_{ij}) + \sum_{i=1}^{N} q(\mathbf{r}_i), \end{aligned} \tag{223}$$

where N is the number of electrons, $N_{\rm ions}$ is the number of ions, $\mathbf{r}_{ij}=\mathbf{r}_i-\mathbf{r}_j$, $\mathbf{r}_{iI}=\mathbf{r}_i-\mathbf{r}_I$, $\mathbf{r}_i$ is the position of electron i and $\mathbf{r}_I$ is the position of nucleus I. In periodic systems the electron–electron and electron–nucleus separations, $\mathbf{r}_{ij}$ and $\mathbf{r}_{iI}$, are evaluated under the minimum-image convention. Note that u, χ, f, p and q may also depend on the spins of electrons i and j.

The plane-wave term, p, will describe similar sorts of correlation to the u term. In periodic systems the u term must be cut off at a distance less than or equal to the radius of the sphere inscribed in the WS cell of the simulation cell and therefore the u function includes electron pairs over less than three quarters of the simulation cell. The p term adds variational freedom in the 'corners' of the simulation cell, which could be important in small cells. The p term can also describe anisotropic correlations, such as might be encountered in a layered compound. It is expected that the u term will be considerably more important than the p term, which cannot describe the electron–electron cusps and is therefore best limited to describing longer-ranged correlations. The q term will describe similar electron–nucleus correlations to the χ_I terms.

22.2 The u, χ and f terms in the Jastrow factor

The u term consists of a complete power expansion in r_{ij} up to order $r_{ij}^{C+N_u}$ which satisfies the Kato cusp conditions at $r_{ij} = 0$, goes to zero at the cutoff length, $r_{ij} = L_u$, and has $C - 1$ continuous derivatives at L_u:

$$u(r_{ij}) = (r_{ij} - L_u)^C \times \Theta(L_u - r_{ij}) \times \left(\alpha_0 + \left[\frac{\Gamma_{ij}}{(-L_u)^C} + \frac{\alpha_0 C}{L_u}\right] r_{ij} + \sum_{l=2}^{N_u} \alpha_l r_{ij}^l\right), \tag{224}$$

where Θ is the Heaviside function and $\Gamma_{ij} = 1/2$ if electrons i and j have opposite spins and $\Gamma_{ij} = 1/4$ if they have the same spin. In this expression C determines the behaviour at the cutoff length. If $C = 2$, the gradient of u is continuous but the second derivative and hence the local energy is discontinuous, and if $C = 3$ then both the gradient of u and the local energy are continuous.

The form of χ is

$$\chi_I(r_{iI}) = (r_{iI} - L_{\chi I})^C \times \Theta(L_{\chi I} - r_{iI}) \times \left(\beta_{0I} + \left[\frac{-Z_I}{(-L_{\chi I})^C} + \frac{\beta_{0I} C}{L_{\chi I}}\right] r_{iI} + \sum_{m=2}^{N_\chi} \beta_{mI} r_{iI}^m\right). \tag{225}$$

It may be assumed that $\beta_{mI} = \beta_{mJ}$ where I and J are equivalent ions. The term involving the ionic charge Z_I enforces the electron–nucleus cusp condition.

The expression for f is the most general expansion of a function of r_{ij}, r_{iI} and r_{jI} that does not interfere with the Kato cusp conditions and goes smoothly to zero when either r_{iI} or r_{jI} reach cutoff lengths:

$$f_I(r_{iI}, r_{jI}, r_{ij}) = (r_{iI} - L_{fI})^C (r_{jI} - L_{fI})^C \Theta(L_{fI} - r_{iI}) \Theta(L_{fI} - r_{jI}) \sum_{l=0}^{N_{fI}^{\mathrm{eN}}} \sum_{m=0}^{N_{fI}^{\mathrm{eN}}} \sum_{n=0}^{N_{fI}^{\mathrm{ee}}} \gamma_{lmnI} r_{iI}^l r_{jI}^m r_{ij}^n. \tag{226}$$

Various restrictions are placed on γ_{lmnI}. To ensure the Jastrow factor is symmetric under electron exchanges it is demanded that $\gamma_{lmnI} = \gamma_{mlnI}$ $\forall\, I, m, l, n$. If ions I and J are equivalent then it is demanded that $\gamma_{lmnI} = \gamma_{lmnJ}$. The condition that the f term has no electron–electron cusps is

$$\left(\frac{\partial f}{\partial r_{ij}}\right)_{\substack{r_{ij}=0 \\ r_{iI}=r_{jI}}} = 0, \tag{227}$$

which implies that

$$\sum_{l=0}^{N_{fI}^{\mathrm{eN}}} \sum_{m=0}^{N_{fI}^{\mathrm{eN}}} \gamma_{lm1I} r_{iI}^{l+m} (r_{iI} - L_{fI})^{2C} = 0\ , \tag{228}$$

for all r_{iI}. Hence, $\forall\, k \in \{0, \ldots, 2N_{fI}^{\mathrm{eN}}\}$,

$$\sum_{l,m\ :\ l+m=k} \gamma_{lm1I} = 0. \tag{229}$$

The condition that the f term has no electron–nucleus cusps is

$$\left(\frac{\partial f}{\partial r_{iI}}\right)_{\substack{r_{iI}=0 \\ r_{ij}=r_{jI}}} = 0, \tag{230}$$

which gives

$$\sum_{m=0}^{N_{fI}^{\mathrm{eN}}} \sum_{n=0}^{N_{fI}^{\mathrm{ee}}} (C\gamma_{0mnI} - L_{fI}\gamma_{1mnI})(-L_{fI})^{C-1} r_{jI}^{m+n} (r_{jI} - L_{fI})^C = 0, \tag{231}$$

for all r_{jI}. It is therefore required that, $\forall\, k' \in \{0, \ldots, N_{fI}^{\mathrm{eN}} + N_{fI}^{\mathrm{ee}}\}$,

$$\sum_{m,n\ :\ m+n=k'} (C\gamma_{0mnI} - L_{fI}\gamma_{1mnI}) = 0. \tag{232}$$

22.3 The p and q terms in the Jastrow factor

The p term takes the cuspless form

$$p(\mathbf{r}_{ij}) = \sum_A a_A \sum_{\mathbf{G}_A^+} \cos(\mathbf{G}_A \cdot \mathbf{r}_{ij}) \ , \tag{233}$$

where the $\{\mathbf{G}_A\}$ are the reciprocal lattice vectors of the simulation cell belonging to the Ath star of vectors that are equivalent under the full symmetry group of the Bravais lattice, and '+' means that, if $\mathbf{G}_A$ is included in the sum, $-\mathbf{G}_A$ is excluded. The p term is important if the finite-size correction to the kinetic energy is to be calculated (see Sec. 29).

For systems with inversion symmetry the q term takes the cuspless form

$$q(\mathbf{r}_i) = \sum_B b_B \sum_{\mathbf{G}_B^+} \cos(\mathbf{G}_B \cdot \mathbf{r}_i), \tag{234}$$

where the $\{\mathbf{G}_B\}$ are the reciprocal lattice vectors of the primitive unit cell belonging to the Bth star of vectors that are equivalent under the space-group symmetry of the crystal, and the '+' means that, if $\mathbf{G}_B$ is included in the sum, $-\mathbf{G}_B$ is excluded. The q term is rarely of use in practice.

22.4 The three-body W term

The W term is given by

$$W = \sum_{j,k,l}^{N} \bar{\delta}_{jk}\bar{\delta}_{jl}\bar{\delta}_{kl}\mathbf{s}_{lj} \cdot \mathbf{s}_{lk} \ , \tag{235}$$

$$\mathbf{s}_{jk} = w_{jk}\mathbf{r}_{jk} \ , \tag{236}$$

where $w_{jk} = w(r_{jk})$ is a suitably parametrized function of the distance between electrons j and k, $\mathbf{r}_{jk} = \mathbf{r}_j - \mathbf{r}_k$, and the symbol $\bar{\delta}_{jk}$ (*no-delta* of j, k) is short-hand for $1 - \delta_{jk}$.

The *core* function w_{ij} is parametrized in CASINO as

$$w_{ij} = w(r_{ij}) = f_C(r_{ij}; L_w) \sum_{l=0}^{n} c_l r_{ij}^l \ . \tag{237}$$

where n is the order of the expansion, c_l are the expansion coefficients and $f_C(r_{ij}; L_w) = (r_{ij} - L_w)^C$ is the usual cutoff function.

22.5 The $u_{\rm cyl}$ term

We may include two-body terms of the form

$$u_{\rm cyl}(\mathbf{r}_{ij}) = \sum_{l=0}^{N_{u\rho}} \sum_{m=0}^{N_{uz}} \varepsilon_{lm}\rho_{ij}^l |z_{ij}|^m (\rho_{ij} - L_{u\rho})^C (|z_{ij}| - L_{uz})^C \Theta\left(L_{u\rho} - \rho_{ij}\right) \Theta(L_{uz} - |z_{ij}|), \tag{238}$$

where ρ_{ij} and z_{ij} are the radial and axial cylindrical polar coordinates of $\mathbf{r}_{ij}$, the $\{\varepsilon_{lm}\}$ and the cutoff lengths $L_{u\rho}$ and L_{uz} are variational parameters, and $N_{u\rho}$ and N_{uz} determine the amount of variational freedom. The parameters ε_{lm} may be different for parallel and antiparallel-spin pairs of electrons. This two-body correlation function has cylindrical symmetry and is symmetric under exchange of electrons. To ensure the wave function has a continuous derivative we must have $\varepsilon_{1m} = \varepsilon_{0m}C/L_{u\rho}$ and $\varepsilon_{l1} = \varepsilon_{l0}C/L_{uz}$; all the other parameters may be varied freely. The largest value that $L_{u\rho}$ can take is the radius of the largest circle that can be inscribed in the Wigner–Seitz cell of the simulation supercell.

22.6 The $\chi_{\rm cyl}$ term

We may include one-body terms of the form

$$\chi_{\rm cyl}(\mathbf{r}_i) = \sum_{l=0}^{N_{\chi\rho}} \sum_{m=0}^{N_{\chi z}} \omega_{lm}\rho_i^l |z_i|^m (\rho_i - L_{\chi\rho})^C (|z_i| - L_{\chi z})^C \Theta\left(L_{\chi\rho} - \rho_i\right) \Theta(L_{\chi z} - |z_i|), \tag{239}$$

where ρ_i and z_i are the radial and axial cylindrical polar coordinates of $\mathbf{r}_i - \mathbf{r}_0$, the $\{\omega_{lm}\}$ and the cutoff lengths $L_{\chi\rho}$ and $L_{\chi z}$ are variational parameters, and $N_{\chi\rho}$ and $N_{\chi z}$ determine the amount of variational freedom. $\mathbf{r}_0$ is an optimizable origin for the $\chi_{\rm cyl}$ term. The parameters ω_{lm} may be different for spin-up and spin-down electrons. This Jastrow term has cylindrical symmetry. To ensure the wave function has a continuous derivative we must have $\omega_{1m} = \omega_{0m} C/L_{\chi\rho}$ and $\omega_{l1} = \omega_{l0} C/L_{\chi z}$; all the other parameters may be varied freely. The largest value that $L_{\chi\rho}$ can take is the radius of the largest circle that can be inscribed in the Wigner–Seitz cell of the simulation supercell.

23 Backflow transformations

The most widely used form of the trial wave function $\Psi_{\rm T}$ is the Slater-Jastrow (SJ) form

$$\Psi_{\rm T} = e^J \Psi_{\rm S} \; , \tag{240}$$

where the Jastrow correlation factor e^J is an optimizable function of the particle coordinates, $J = J(\{\mathbf{r}_i\})$, and the Slater part $\Psi_{\rm S}$ is in general a multideterminant expansion,

$$\begin{aligned} \Psi_{\rm S} &= \sum_{k=1}^{N_D} c_k \prod_{j=1}^{N_{\rm S}} D^{(jk)} \; , & (241) \\ D^{(jk)} &= D^{(jk)}(\{\mathbf{r}_i\}) \; , & (242) \end{aligned}$$

where $\{c_k\}$ are the expansion coefficients, and $D^{(jk)}$ is the Slater determinant of the one-electron orbitals of electrons of spin j in the kth term of such expansion.

Backflow corrections in QMC are capable of introducing further correlations in $\Psi_{\rm T}$ by substituting the coordinates in the Slater determinants by a set of collective coordinates $\mathbf{x}_i(\{\mathbf{r}_j\})$, given by

$$\mathbf{x}_i = \mathbf{r}_i + \boldsymbol{\xi}_i(\{\mathbf{r}_j\}) \; , \tag{243}$$

where $\boldsymbol{\xi}_i$ is the backflow displacement of particle i, which depends on the configuration of the system $\{\mathbf{r}_j\}$, and contains optimizable parameters that can be fed into a standard method like variance minimization.

While the use of a Jastrow factor does not modify the nodal surface of the wave function, backflow transformations do change it.

23.1 The generalized backflow transformation

The form of the backflow displacement $\boldsymbol{\xi}_i$ in homogeneous systems has traditionally been taken as [61]

$$\boldsymbol{\xi}_i^{(\rm e-e)} = \sum_{j\neq i}^{N} \eta_{ij} \mathbf{r}_{ij} \; , \tag{244}$$

where $\eta_{ij} = \eta(r_{ij})$ is an appropriate function of interparticle distance. Equation (244) can be regarded as the most general two-body coordinate transformation for a homogeneous system.

Notice that the above expression implicitly assumes that there is a set of preferred directions in the system, given by the electron–electron vectors $\{\mathbf{r}_{ij}\}$. In a system with nuclei a new set of preferred directions is introduced, the electron–nucleus vectors $\{\mathbf{r}_{iI}\}$. Following the same idea, one is led to introduce an electron–nucleus contribution to $\boldsymbol{\xi}_i$, of the form:

$$\boldsymbol{\xi}_i^{(\rm e-N)} = \sum_{I}^{N_{\rm ion}} \mu_{iI} \mathbf{r}_{iI} \; , \tag{245}$$

where $\mu_{iI} = \mu(r_{iI})$. However, this is a one-electron term; to be consistent with the order of the η term, it is necessary to introduce an inhomogeneous two-electron term (i.e., an electron–electron–nucleus term), which would be given by

$$\boldsymbol{\xi}_i^{(\rm e-e-N)} = \sum_{j\neq i}^{N} \sum_{I}^{N_{\rm ion}} \left(\Phi_i^{jI} \mathbf{r}_{ij} + \Theta_i^{jI} \mathbf{r}_{iI} \right) \; , \tag{246}$$

where $\Phi_i^{jI} = \Phi(r_{iI}, r_{jI}, r_{ij})$ and $\Theta_i^{jI} = \Theta(r_{iI}, r_{jI}, r_{ij})$.

These functions must be cut off at given lengths for efficiency. We use a simple truncation function,

$$f(r; L) = \left(\frac{L - r}{L}\right)^C \Theta(L - r) \ , \tag{247}$$

where L is a cutoff length, C is the truncation order and Θ denotes the Heaviside function.

Notice that the above can be easily extended to inhomogeneities other than atoms by introducing any other coordinates appearing in the potential energy into the expressions. This is not implemented in CASINO yet.

23.1.1 The η function

Three forms of the homogeneous backflow function have been tested with CASINO. Following extensive tests, no advantages were found from using the rational and Gaussian forms, and only the polynomial form remains in the code,

$$\eta_{ij} = f(r_{ij}; L_\eta) \sum_{k=0}^{N_\eta} c_k r_{ij}^k \ , \tag{248}$$

where N_η is the expansion order and $\{c_l\}$ are the optimizable coefficients. We expect this simple form to be flexible and efficient. The natural polynomial basis is used to improve the speed of evaluation, and rounding errors are not likely to appear for $N_\eta + C < 20$ [23].

23.1.2 The μ function

We use the following polynomial expansion for μ,

$$\mu_{iI} = f(r_{iI}; L_{\mu,I}) \sum_{k=0}^{N_{\mu,I}} d_{k,I} r_{iI}^k \ , \tag{249}$$

where $L_{\mu,I}$ is the cutoff length for ion I, $N_{\mu,I}$ is the expansion order and $\{d_{k,I}\}$ are the optimizable coefficients.

23.1.3 The Φ and Θ functions

Our choice for Φ_i^{jI} and Θ_i^{jI} are the following power expansions,

$$\Phi_i^{jI} = f(r_{iI}; L_{\Phi,I}) f(r_{jI}; L_{\Phi,I}) \sum_{k=0}^{N_{\mathrm{eN},I}} \sum_{l=0}^{N_{\mathrm{eN},I}} \sum_{m=0}^{N_{\mathrm{ee},I}} \varphi_{klm,I} r_{iI}^k r_{jI}^l r_{ij}^m \ , \tag{250}$$

$$\Theta_i^{jI} = f(r_{iI}; L_{\Phi,I}) f(r_{jI}; L_{\Phi,I}) \sum_{k=0}^{N_{\mathrm{eN},I}} \sum_{l=0}^{N_{\mathrm{eN},I}} \sum_{m=0}^{N_{\mathrm{ee},I}} \theta_{klm,I} r_{iI}^k r_{jI}^l r_{ij}^m \ , \tag{251}$$

where $N_{\mathrm{eN},I}$ and $N_{\mathrm{ee},I}$ are the expansion orders, $L_{\Phi,I}$ are the truncation lengths and $\varphi_{klm,I}$ and $\theta_{klm,I}$ are optimizable coefficients.

23.1.4 The Π function

The backflow Π term is of the form of the gradient of the Jastrow P term. Specifically, the contribution to the backflow displacement $\boldsymbol{\xi}_i$ is

$$\sum_{j \neq i}^{N} \boldsymbol{\pi}(\mathbf{r}_{ij}), \tag{252}$$

[23] It is estimated that numerical problems arise in the evaluation of polynomials beyond order ~ 20 when using double-precision arithmetics. More complicated polynomial forms (e.g., Chebyshev polynomials) should be used to be able to exceed this limit.

where $\mathbf{r}_{ij} = \mathbf{r}_i - \mathbf{r}_j$ and

$$\boldsymbol{\pi}(\mathbf{r}_{ij}) = \sum_A \alpha_A \sum_{\mathbf{G}_A^+} - \sin(\mathbf{G}_A \cdot \mathbf{r}_{ij})\mathbf{G}_A, \tag{253}$$

where the $\{\mathbf{G}_A\}$ are the reciprocal lattice vectors of the simulation cell belonging to the Ath star of vectors that are equivalent under the full symmetry group of the Bravais lattice, and '+' means that, if $\mathbf{G}_A$ is included in the sum, $-\mathbf{G}_A$ is excluded. The $\{\alpha_A\}$ are optimizable parameters.

Like the Jastrow P term, the Π term is intended to capture long-range behaviour, allowing the description of correlation between electrons in the "corners" of the Wigner-Seitz cell of the simulation cell. The Π term is cuspless and irrotational by construction.

23.2 Constraints on the backflow parameters

In SJ wave functions it is common practice to impose the electron–electron cusp conditions on the parameters in the Jastrow factor and the electron–nucleus ones on the orbitals in the Slater determinant. Backflow can modify the cusp conditions; we have constrained the backflow parameters so that they do not.

When AE atoms are present, it can be shown that the electron–nucleus cusp conditions cannot be fulfilled unless there is no homogeneous backflow term present. However, this issue can be bypassed by smoothly truncating $\eta(r_{ij})$ around such nuclei. This is automatically done by CASINO.

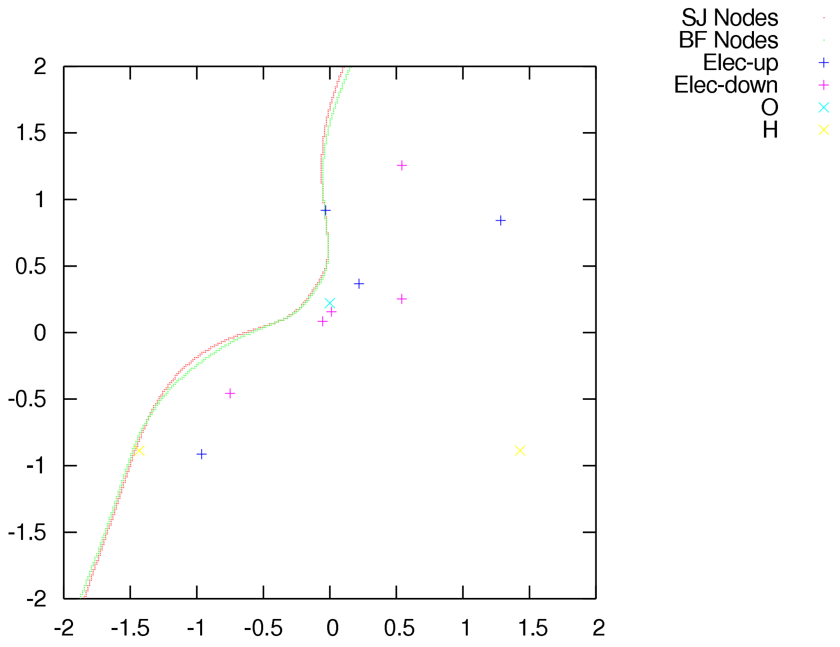
An additional set of constraints can be added in order to satisfy the relation

$$\boldsymbol{\xi}_i(\{\mathbf{r}_j\}) = \nabla_i Y(\{\mathbf{r}_j\}) \ , \tag{254}$$

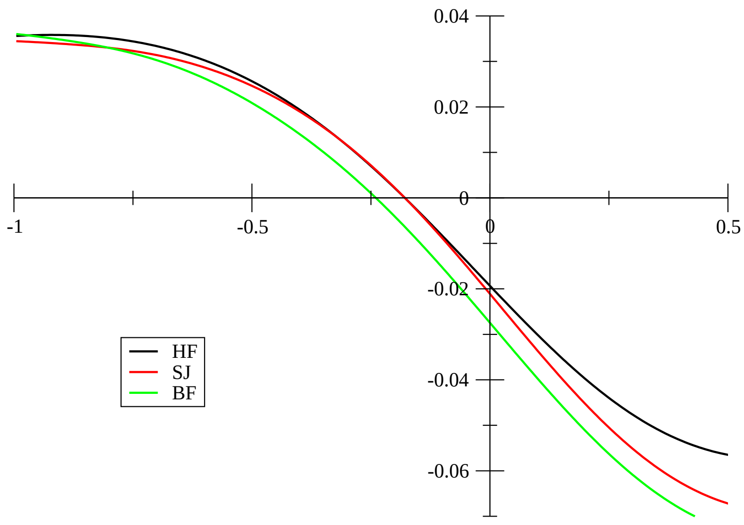
where $Y(\{\mathbf{r}_j\})$ is an object called *backflow potential*, which appears in the derivation of backflow of Ref. [62]. This equation is already satisfied by both the electron–electron and the electron–nucleus terms by definition, and it can be imposed on the electron–electron–nucleus backflow functions by using an appropriate set of constraints, which correspond to the 'no-curl' flag in the *phi term* in `correlation.data`. It is found that the 'no-curl' conditions do not give particularly good backflow functions; it has been left in CASINO as a means of drastically reducing the number of free parameters in cases where this is absolutely necessary.

23.3 Improving the nodes of Ψ_{T}

CASINO's implementation of backflow has the ability to improve the nodal surface of the trial wavefunction with respect to that of the SJ form. There is an additional computational expense in going from SJ to backflow (BF), which is a factor of ~ 2 for the AE neon atom—based on the CPU time taken to achieve a given error bar in the mean energy—in both VMC and DMC; other systems may vary, but we expect backflow to be a reasonably efficient way of reducing the fixed-node error in most cases.



Plot of the wave-function nodes of a water molecule, performed by moving an electron in the molecular plane (XY) while keeping all others fixed in an arbitrary configuration (generated using HF-VMC).



Plot of the HF, SJ and BF wave-functions across a node in a water molecule, corresponding to moving (x, y) from $(-1, -1)$ to $(-1, 0.5)$ in the previous graph.

It should be noticed, however, that the application of backflow has a greater effect on VMC than on DMC. For instance, BF-VMC applied on AE neon has been found to recover a 53% of the correlation energy missing at the SJ-VMC level, while BF-DMC achieves about 38% with respect to SJ-DMC. The case of AE Carbon is even more pronounced, the figures being 45% for VMC and a mere 17% for DMC.

The reason for this is clear: the backflow parameters are optimized within VMC, where the nodal error is far less important than the *bulk* error—*bulk* meaning 'the wave-function away from the nodes' in this context—simply because of the difference in the size of the two regions in configuration space. Theoretically, it is even possible to worsen the DMC results by using a wave-function that improves the VMC energies, but this case has not been found in practice. It is unknown whether a different optimization scheme may be used to systematically improve the nodal surface of a wave-function regardless of its *bulk*.

Even so, there are two solid points that encourage the use of backflow on many problems:

- It reduces the fixed-node error by a statistically significant amount in VMC and DMC at little additional cost in many cases.

- BF-VMC energies are often found to be very close to the SJ-DMC ones. The VMC method is thus turned into a powerful, yet simple tool delivering highly reliable results.

24 Statistical analysis of data

24.1 The reblocking method

Each configuration generated by the QMC algorithms is related to a configuration from the previous iteration, so the raw QMC data in the `vmc.hist` and `dmc.hist` files are serially correlated. A naïve calculation of the variance of the energy (and hence the standard error in the mean) is an underestimate, because successive local energies are more similar on average than they would be if the configurations were independent.

The effects of serial correlation can be eliminated by gathering successive data points into blocks and averaging over the data in each block [63]. The variance of the set of block averages can then be calculated. If the block length is greater than the correlation length between data points then the block averages are uncorrelated and an unbiased estimate of their variance is obtained. Thus, if the estimated standard error in the mean energy is plotted against block length then it should, for large enough blocks, be distributed about a constant value, which is the true standard error in the mean. If it does not reach such a plateau then there is insufficient data to estimate the standard error in the energy estimate.

Each iteration is equally weighted in a VMC calculation; however, for DMC, each iteration is weighted by the total weight of the configurations multiplied by the Π-weight. In either case, let the iteration weights be w_i, the total number of data points be M and the energy from iteration i be e_i.

Consider the bth reblocking transformation, in which the block length is $B_b = 2^{b-1}$. The data range may be divided into M_b blocks, the last of which is usually incomplete.

For each block j, the block weight is

$$W_{bj} = \sum_{i \in j} w_i, \tag{255}$$

and the corresponding block energy is

$$E_{bj} = \frac{\sum_{i \in j} e_i w_i}{W_{bj}}. \tag{256}$$

The 'reblocked' energy is

$$\begin{aligned} E_b &= \frac{\sum_j E_{bj} W_{bj}}{\sum_j W_{bj}} \\ &= \frac{\sum_i e_i w_i}{\sum_i w_i} \equiv E, \end{aligned} \tag{257}$$

which is therefore independent of reblocking transformation. On the other hand, the reblocked variance is

$$\sigma_b^2 = \frac{\sum_j W_{bj} (E_{bj} - E)^2}{\sum_j W_{bj} - \frac{\sum_j W_{bj}^2}{\sum_j W_{bj}}}, \tag{258}$$

which *does* depend on the reblocking transformation number.

The number of blocks at the bth reblocking transformation is $N_b = M/B_b$. (Note that N_b is not necessarily an integer, because the last block may be incomplete.) The standard error in the energy estimate at reblocking transformation number b is

$$\delta_b = \frac{\sigma_b}{\sqrt{N_b}}, \tag{259}$$

and the error in δ_b may be estimated as

$$\epsilon_b = \frac{\delta_b}{\sqrt{2(N_b - 1)}}. \tag{260}$$

The reblocking procedure is performed 'on-the-fly' by CASINO itself for total energies and their components, and the reblocked error bars should be printed out at the end of the `out` file. The algorithm for determining the best block size is based on that given in Ref. [64], giving a robust algorithm that offers an apparently optimal tradeoff between statistical and systematic error in the error bar. If CASINO thinks the reblocking procedure has not converged (as it won't for shorter runs) then it will say so and print out additional information.

The separate `reblock` utility may be used to post-process the data in the `vmc.hist` or `dmc.hist` file and produce a table of δ_b and ϵ_b against b; the user can then manually look for a region in which the standard error δ_b has reached a plateau as a function of b, and choose an appropriate reblocking transformation number, which is then used to calculate the error bars on all the different components of energy.

The `reblock` utility will be invoked automatically if you use '`haltqmc -r`' to tidy up your output files.

24.2 Estimate of the correlation time given by CASINO

The correlation time of the energy is computed and shown for every block in a VMC calculation, and also when using the `reblock` utility. The correlation time measures the average number of Monte Carlo steps between two uncorrelated values of the energy, and should be unity for optimal statistics. Note that in this context by 'Monte Carlo step' we mean 'every step for which an energy is stored'. For example, in a VMC calculation, every **vmc_decorr_period**×**vmc_ave_period** configuration moves produce a single datum for later analysis.

The definition of the correlation time τ of an observable H is

$$\tau = \int_{-\infty}^{+\infty} A(t)dt = \int_{-\infty}^{+\infty} \frac{\langle (H_{t'} - \langle H_{t''}\rangle_{t''})(H_{t'+t} - \langle H_{t''}\rangle_{t''})\rangle_{t'}}{\sigma_H^2}\, dt, \tag{261}$$

where $A(t)$ is the value of the autocorrelation function at an interval of t, $\sigma_H^2 = \langle (H_{t'} - \langle H_{t''}\rangle_{t''})^2\rangle_{t'}$ is the variance of the expectation values, and the latter are taken with respect to their subscript, which we shall remove for the sake of clarity. For a discrete set of values, equally spaced by an amount $\Delta t = 1$,

$$\tau = \sum_{t=-\infty}^{+\infty} A(t) = 1 + 2\sum_{t=1}^{+\infty} A(t) = 1 + 2\sum_{t=1}^{+\infty} \frac{\langle (H_{t'} - \langle H\rangle)(H_{t'+t} - \langle H\rangle)\rangle}{\sigma_H^2} \tag{262}$$

and for a finite set of length N,

$$\tau = 1 + 2\sum_{t=1}^{N-1} \frac{\langle (H_{t'} - \langle H\rangle)(H_{t'+t} - \langle H\rangle)\rangle}{\sigma_H^2}. \tag{263}$$

Numerically, the problem with this expression is that if averages are used instead of proper expectation values (which are, of course, unknown), great fluctuations will appear at the tail of the autocorrelation function. This problem is solved by introducing a cutoff in the summation [65]:

$$\tau(t_{\max}) = 1 + 2\sum_{t=1}^{t_{\max}} \frac{\overline{(H_{t'} - \overline{H})(H_{t'+t} - \overline{H})}}{\hat{\sigma}_H^2}, \tag{264}$$

where the numerator is the average of the measured values of its arguments over the configurations indexed by $1 \le t' \le N - t$, and $\hat{\sigma}_H^2$ is the variance of these measures. One possibility for setting the cutoff is to check against the self-consistent inequality $t_{\max} < 3\tau(t_{\max})$ while computing the sum, and truncate it as soon as it stops being true. This allows an estimate of the error in above expression to be calculated:

$$\epsilon_\tau(t_{\max}) = \tau\sqrt{\frac{2(2t_{\max} + 1)}{N}}. \tag{265}$$

The reblocking method and the correlation time are in principle equally valid methods for estimating the error in the energy. Each of them has its own disadvantages, though: the plot of reblocked standard errors can often become noisy before the plateau is reached, preventing accurate determination of the optimal reblocking length, whereas the error in the correlation time decays very slowly with the number of energies in the sample. It is recommended that both measures of serial correlation be taken into account for optimal results.

The corrected error bar from the correlation time analysis is printed at the end of all VMC calculations (both for single runs and for sequences of continued calculations).

24.3 Estimating equilibration times and correlation periods

The root-mean-square distance diffused by a particle in a period T of imaginary time is $\sqrt{2N_D DAT}$, where A is the move acceptance ratio (which is usually close to 1 in DMC and 1/2 in VMC), N_D is the dimensionality of the system (which is usually 3, unless a strict 2D or 1D system is being studied) and $D = 1/2m$ is the diffusion constant, where m is the particle mass (note that $D = 1/2$ for electrons). We expect that correlation effects will disappear when the particles have diffused through distances in excess of the largest physically relevant length-scale λ. Let $T = N_{\rm move} \times \tau$, where $N_{\rm move}$ is the number of moves and τ is the time step. Then the number of moves over which we expect correlation effects to be present is

$$N_{\rm move} = \frac{\lambda^2}{2N_D D\tau A}. \tag{266}$$

The number of equilibration moves should be substantially larger than the above estimate of the correlation period in order to ensure that all of the transient effects due to the initial distribution die away. The required equilibration period is often greater than one might expect by simply examining the variation of the total energy with time.

In practical QMC calculations, with sensible choices of time step, we often find the VMC correlation period to be about 5 configuration moves and the DMC correlation period to be about 1000 moves.

25 Wave-function optimization

Optimization of the trial wave function is a crucial part of a VMC or DMC calculation. CASINO allows optimization of the parameters in the Jastrow factor, the coefficients of the determinants in a multideterminant wave function, the parameters in the backflow functions, pairing parameters in electron–hole gases and parameters in the orbitals for certain electron and electron–hole phases as well as modification functions for atomic orbitals. All optimizable parameters are contained in the file `correlation.data`. Furthermore, each optimizable parameter is followed by a flag indicating whether the parameter is fixed ('0') or free to be optimized ('1'). See Sec. 7.4 for information on `correlation.data`.

There are two methods available within CASINO for wave function optimization: variance minimization and energy minimization. Both methods can be used to optimize any or all of the parameters mentioned above. In addition to the 'standard' variance-minimization method, there also exists a much faster version, which can be used when only parameters which appear linearly in the Jastrow factor are being optimized.

25.1 Variance minimization: the standard method

Consider a real trial wave function $\Psi(\mathbf{R})$, where $\mathbf{R}$ is a point in the electron configuration space. In VMC the energy is written as

$$E = \frac{\int |\Psi(\mathbf{R})|^2 E_{\rm L}(\mathbf{R})\, d\mathbf{R}}{\int |\Psi(\mathbf{R})|^2\, d\mathbf{R}}, \tag{267}$$

where the *local energy*, $E_{\rm L}$, is

$$E_{\rm L}(\mathbf{R}) = \Psi(\mathbf{R})^{-1}\hat{H}(\mathbf{R})\Psi(\mathbf{R}), \tag{268}$$

and $\hat{H}$ is the Hamiltonian. The variance of the energy is

$$\sigma^2 = \frac{\int |\Psi(\mathbf{R})|^2 |E_{\rm L}(\mathbf{R}) - E|^2\, d\mathbf{R}}{\int |\Psi(\mathbf{R})|^2\, d\mathbf{R}}. \tag{269}$$

We write the trial wave function as $\Psi^{\{\alpha\}}(\mathbf{R})$, to denote that it depends on a set of free parameters, $\{\alpha\}$. Consider a set of $N_{\rm C}$ configurations $\{\mathbf{R}\}$ distributed according to $\left|\Psi^{\{\alpha_0\}}(\mathbf{R})\right|^2$ for some fixed parameter set $\{\alpha_0\}$. The variance σ^2 is then estimated for any given parameter set $\{\alpha\}$ using a correlated-sampling procedure, which gives rise to the *reweighted variance*,

$$\sigma_w^2 = \frac{T_{\{\alpha_0\}}^{\{\alpha\}}}{\left(T_{\{\alpha_0\}}^{\{\alpha\}}\right)^2 - \sum_{\mathbf{R}} \left(W_{\{\alpha_0\}}^{\{\alpha\}}(\mathbf{R})\right)^2} \sum_{\mathbf{R}} \left| E_{\mathrm{L}}^{\{\alpha\}}(\mathbf{R}) - \bar{E}_w \right|^2 W_{\{\alpha_0\}}^{\{\alpha\}}(\mathbf{R}), \tag{270}$$

where the *reweighted energy* is

$$\bar{E}_w = \frac{1}{T_{\{\alpha_0\}}^{\{\alpha\}}} \sum_{\mathbf{R}} \mathrm{Re}\left(E_{\mathrm{L}}^{\{\alpha\}}(\mathbf{R})\right) W_{\{\alpha_0\}}^{\{\alpha\}}(\mathbf{R}), \tag{271}$$

which is an estimate of E, and the total weight is

$$T_{\{\alpha_0\}}^{\{\alpha\}} = \sum_{\mathbf{R}} W_{\{\alpha_0\}}^{\{\alpha\}}(\mathbf{R}), \tag{272}$$

and the weights W are

$$W_{\{\alpha_0\}}^{\{\alpha\}}(\mathbf{R}) = \left| \frac{\Psi^{\{\alpha\}}(\mathbf{R})}{\Psi^{\{\alpha_0\}}(\mathbf{R})} \right|^2 . \tag{273}$$

The nodal surface of the trial wave function is independent of parameters in the Jastrow factor, so the weights cannot diverge when such parameters change. For parameters that do affect the nodal surface, however, it is possible for the weights to diverge.

The *unreweighted variance* as a function of parameter set $\{\alpha\}$ is defined to be

$$\sigma_u^2 = \frac{1}{N_{\mathrm{C}} - 1} \sum_{\mathbf{R}} \left| E_{\mathrm{L}}^{\{\alpha\}}(\mathbf{R}) - \bar{E}_u \right|^2 , \tag{274}$$

where the *unreweighted energy* is

$$\bar{E}_u = \frac{1}{N_{\mathrm{C}}} \sum_{\mathbf{R}} \mathrm{Re}\left(E_{\mathrm{L}}^{\{\alpha\}}(\mathbf{R})\right) . \tag{275}$$

The reweighted and unreweighted variances are identical when the same set of configurations is used and $\{\alpha\} = \{\alpha_0\}$. However, for any given $\{\alpha_0\}$ they are different functions of $\{\alpha\}$, and there is no reason to expect that their minima coincide with each other, or that either minimum should coincide with that of the (reweighted) energy.

Both σ_w^2 and σ_u^2 are non-negative, but are zero when $\Psi^{\{\alpha\}}$ is an eigenstate of $\hat{H}$. The reweighted and unreweighted variances are therefore reasonable cost functions for wave-function optimizations. The reweighted energy is also a reasonable cost function. However, the problem with the reweighted energy and variance is that the weights W may vary rapidly as the parameters change, especially for large systems, which leads to instabilities in optimization procedures [66]. From these considerations we conclude that the cost function with the most suitable mathematical properties for the stable optimization of wave functions within the correlated-sampling approach is the unreweighted variance.

The usual variance-minimization procedure is to generate a set of electron configurations $\{\mathbf{R}\}$ distributed according to $\left|\Psi^{\{\alpha_0\}}(\mathbf{R})\right|^2$ using VMC, and then to minimize the reweighted or unreweighted energy variance over this set. Since the variance landscape depends on the distribution of configurations, several *cycles* of configuration generation and optimization are normally carried out, with the optimized wave function from the previous cycle being used in each VMC configuration-generation phase. We usually iterate several times and choose the wave function that gives the lowest *variational energy*. In the limit of perfect sampling, the reweighted variance is equal to the actual variance, and is therefore independent of the configuration distribution, so that the optimized parameters would not change over successive cycles of reweighted variance minimization. This is not the case for unreweighted variance minimization; nevertheless, by carrying out a number of cycles, a 'self-consistent' parameter set may be obtained.

In CASINO the minimization of the variance is carried out by the routine `nl2sno` in module `nl2sol`, which performs an unconstrained minimization (without requiring derivatives) of a sum of m squares of functions which contain n variables, where $m \geq n$. (Information on the minimization routine can be found in Ref. [67]).

Before carrying out this process, the user must decide how they wish to parametrize the trial wave function, and with how many parameters. They must also decide on the number of configurations to be used in the optimization. These choices are system specific, and depend on the level of accuracy to which the user wishes to work.

A systematic approach to deciding on an appropriate number of variational parameters is to start by optimizing a few parameters, then to add more and re-optimize, and so on, until the decrease in energy resulting from the inclusion of additional parameters is small compared with the energy difference that the user wishes to resolve. Note that the error bars on the VMC energy must be smaller still, so that the user can make accurate judgements about the energy differences.

It is clearly desirable for the VMC-generated configurations to be completely uncorrelated. This can be achieved by giving **vmc_decorr_period** a large value (e.g., 10). Reblocking VMC energies in a preliminary VMC run will allow the user to determine the correlation period for VMC energies, which in turn suggests a suitable value for **vmc_decorr_period**. It is also essential that the VMC configuration-generation run is fully equilibrated. Since VMC equilibration is usually computationally inexpensive, this should be straightforward enough. The utility `plot_hist` can be used to verify that the VMC energies have equilibrated.

The user may choose whether to optimize the Jastrow factor, determinant-expansion coefficients, or the pairing parameters and orbital coefficients, by setting the **opt_jastrow**, **opt_detcoeff** and **opt_orbitals** flags as appropriate. For most applications, it is only necessary to optimize the Jastrow factor. If only linear parameters in CASINO's Jastrow factor are to be optimized then the 'varmin-linjas' method should be used: see Sec. 25.2.

The user may choose between the reweighted or unreweighted variance-minimization algorithms by choosing the **vm_reweight** keyword to be `T` or `F`, respectively. As shown in Ref. [68], the unreweighted variance-minimization has several desirable properties, which often make it a more useful technique than reweighted variance minimization: (i) the unreweighted algorithm is numerically more stable; (ii) in general the unreweighted variance has a simple functional form and only a single minimum in the space of linear Jastrow parameters; (iii) for a large number of model systems it can be demonstrated that the wave functions generated by unreweighted variance minimization iterated to self-consistency have a lower variational energy than wave functions optimized by reweighted variance minimization.

If reweighted variance minimization is performed then it is possible to limit the values that the weights can take, in an attempt to improve the stability. The **vm_w_max** and **vm_w_min** parameters can be used to specify the maximum and minimum values that the weights can take.

When optimizing parameters that affect the nodal surface, the local energies of configurations may diverge as the nodal surface is moved. The affected configurations will then have a disproportionate effect on the value of the unreweighted variance. It is therefore desirable to remove configurations from the optimization procedure when their local energies deviate substantially from the mean local energy. This can be achieved by introducing an effective weight for each configuration, which is a function of the deviation of the local energy from the mean local energy,

$$f(E_{\rm L}) = \left\{ \begin{array}{cc} 1 & \left|E_{\rm L}-\bar{E}\right|/\sigma_{E_{\rm L}} < T \\ \exp\left[-\left(\frac{\left|E_{\rm L}-\bar{E}\right|/\sigma_{E_{\rm L}}-T}{W}\right)^2\right] & \left|E_{\rm L}-\bar{E}\right|/\sigma_{E_{\rm L}} > T \end{array} \right. , \tag{276}$$

where T is a user-defined threshold, W is a user-defined filter width, $E_{\rm L}$ is the local energy of a configuration, $\bar{E}$ is the average energy, and $\sigma_{E_{\rm L}}$ is the square root of the unreweighted variance. To activate the configuration-filtering scheme, turn **vm_filter** to `T` in `input`; the T parameter corresponds to the keyword **vm_filter_thres** and W corresponds to **vm_filter_width**. The default values of $T = 4$ and $W = 2$ are found to work well in most cases.

Another variant of variance minimization which has proven to be very robust is the minimization of the mean-absolute-deviation (MAD) of the local energies with respect to the median local energy $\bar{E}_m$,

$$\text{MAD} = \frac{1}{N_{\rm C}} \sum_{\mathbf{R}} \left| E_{\rm L}^{\{\alpha\}}(\mathbf{R}) - \bar{E}_m \right| , \tag{277}$$

which is found to be adequate for parameters which affect the nodes of the trial wave function. To use this optimization method, set **opt_method** to `madmin` in the input file.

It has been suggested [69] that using an estimate of the ground-state energy in place of the mean (or median) energy in the expression for the variance can improve the variance-minimization algorithm by adding an element of energy minimization. This can be achieved by setting **vm_use_E_guess** to `T` and supplying the energy estimate using **vm_E_guess**. However, we have very rarely obtained any advantage by doing this.

It is possible to choose how much information CASINO will provide during the optimization process. Setting **opt_info** to 1 will provide no information during the minimization; setting it to 2 will provide a list of the parameters, the mean energy and variance at each iteration [default]; 3 will give additional information about the numerical derivatives that are computed between iterations to build the Jacobian; 4 will add the mean and variance of the weights to the output (note that the weights are then computed but not actually used unless **vm_reweight** is set); 5 provides an enormous amount of detail and is only likely to be of use for development or debugging purposes.

When carrying out variance minimization, one usually observes a sharp fall in both the variance and the energy at the first cycle. Thereafter the energy 'bounces around'. Note that the energies of subsequent cycles may change by more than the statistical error bars on the individual VMC runs, because the reoptimization of the parameters constitutes an additional source of variance.

If one wishes to obtain an 'adequate' wave function without spending a lot of time on the optimization process then it is advisable to (i) use unreweighted variance minimization; (ii) err on the side of underparametrization, as this reduces the chance of encountering instabilities; (iii) use as many configurations as is practicable, but certainly more than 10,000; (iv) carry out a few (e.g., 4) variance-minimization cycles, in order to check that the optimization process was successful.

If one wishes to obtain a highly accurate wave function then the following approach is often successful: (i) use unreweighted variance minimization; (ii) systematically investigate the use of different numbers of parameters and different forms of parametrization; (iii) ensure that the VMC error bars are much smaller than the energy differences to be resolved; (iv) carry out a large number of variance-minimization cycles (e.g., 20) and choose the `correlation.data` file that gives the lowest variational energy.

25.2 Variance minimization: the 'varmin-linjas' method

25.2.1 Background

Consider the linear parameters in CASINO's Jastrow factor (the expansion coefficients $\{\alpha_l\}$, $\{\beta_m\}$, $\{\gamma_{lmn}\}$, $\{a_A\}$, $\{b_B\}$ and ε_{lm} in the u, χ, f, p, q, $u_{\rm cyl}$ and $\chi_{\rm cyl}$ terms, respectively). The local energy of a single configuration can be shown to be a quadratic function of the linear parameters; hence the variance of the local energies of a fixed, finite set of configurations is a quartic function of the parameters. But this is precisely the quantity that is minimized in an unreweighted variance-minimization calculation. The process of variance minimization can therefore be greatly simplified and accelerated if only linear Jastrow parameters are to be optimized.

In an ordinary variance-minimization calculation, the VMC method is used to generate a set of configurations distributed according to the initial trial wave function. During the optimization process, the variance of the local energies of this configuration set is computed for different sets of parameters, and the variance is minimized with respect to the parameters. By contrast, in the 'varmin-linjas' method, the quartic expansion coefficients of the unreweighted variance are accumulated directly in VMC: there is no need to write out a set of configurations. Furthermore, when the unreweighted variance—referred to as the *least-squares function* (LSF)—is evaluated during the subsequent optimization stage, there is no need to sum repeatedly over a set of configurations: the quartic LSF can be evaluated directly.

The fact that the LSF can be evaluated as a quartic function of the parameters gives two significant advantages over the standard variance-minimization algorithm: (i) the LSF can be evaluated extremely rapidly (typically thousands of times per second); furthermore the CPU time required is *independent* of the system size; and (ii) the LSF along any line in parameter space is a simple quartic polynomial, so that the exact, *global* minimum of the LSF along that line can be computed analytically.

The method has two drawbacks: (i) only linear Jastrow parameters can be optimized in this fashion; and (ii) the number of quartic coefficients to be evaluated and stored in memory grows as the fourth power of the number of parameters to be optimized.

Detailed information about the varmin-linjas method can be found in Ref. [68].

25.2.2 Using the varmin-linjas method

A CASINO variance-minimization calculation using the varmin-linjas method is carried out in exactly the same way as an ordinary variance-minimization calculation except that:

1. The **opt_method** keyword in `input` should be set to `varmin_linjas`.
2. If desired, the user may change the method used to minimize the LSF with respect to the set of parameters by using the **vm_linjas_method** keyword. This can take the values: 'CG' (*conjugate gradients*); 'SD' (*steepest descents*); 'GN' (*Gauss-Newton*); 'MC' (*Monte Carlo line minimization*); 'LM' (*simple line minimization*); 'CG_MC' (*alternate conjugate gradients and Monte Carlo line minimization*); 'BFGS' (*Broyden-Fletcher-Goldfarb-Shanno*); 'BFGS_MC' (*alternate BFGS and Monte Carlo line minimization*); or 'GN_MC' (alternate Gauss-Newton and Monte Carlo line minimization). If the **vm_linjas_method** keyword is omitted then the BFGS method will be used by default. If you experience difficulty optimizing a large set of parameters then the Gauss-Newton method is worth trying. The BFGS method seems to be the most efficient method in general, however.
3. If desired, the user can change both the maximum number of iterations and the number of line minimizations to be performed by means of the **vm_linjas_its** keyword. If this keyword is omitted, or it is given a negative value, then a default number of iterations will be performed.

The cutoff lengths in the Jastrow factor are important variational parameters, and some attempt to optimize them should always be made. It is recommended that a (relatively cheap) calculation using the standard variance-minimization method should be carried out in order to optimize the cutoff lengths, followed by an accurate optimization of the linear parameters using the varmin-linjas method. For some systems, good values of the cutoff lengths can be supplied immediately (for example, in periodic systems at high density with small simulation cells, the cutoff length L_u should be set equal to the radius of the sphere inscribed in the WS cell of the simulation cell), and one can make use of the varmin-linjas method straight away.

25.3 Energy minimization

25.3.1 Motivation

There are several reasons why optimizing the energy instead of the variance is desirable:

- Since trial wave functions generally cannot exactly represent an eigenstate, the energy and variance minima do not coincide. Energy minimization should therefore produce lower VMC energies. This might in turn yield lower DMC energies, although it is not clear how improvements in the energy (or variance) relate to improvements in the nodal surface. (In practice, lower VMC energies usually lead to lower DMC energies.)
- Energy-optimized wave functions have been shown to give better estimates of other expectation values [70, 71, 72].
- It is known that the variance of the DMC wave function is proportional to the difference between the ground-state energy and the VMC energy [73, 74].

The energy minimization method used in CASINO is especially accurate when optimizing parameters which appear linearly in the wave function (i.e., determinant coefficients). When only optimizing such parameters, the global energy minimum is found, usually in one cycle.

Sections 25.3.2 and 25.3.3 briefly summarize the theory of the method, and Sec. 25.3.7 explains its use. The theoretical background given here is far from exhaustive. For a much more thorough discussion, see Refs. [75, 76, 77, 78].

25.3.2 Basic theory

Consider a (generally complex) trial wave function Ψ which depends upon a set of p real, variable parameters $\boldsymbol{\alpha}$. If the cycle $\Psi^{(n)} \rightarrow \Psi^{(n+1)}$ involves parameter changes $\boldsymbol{\alpha}^{(n)} \rightarrow \boldsymbol{\alpha}^{(n)} + \delta\boldsymbol{\alpha}^{(n)}$, then we can Taylor-expand $\Psi^{(n+1)} = \Psi(\boldsymbol{\alpha}^{(n+1)})$ as:

$$\Psi(\boldsymbol{\alpha}^{(n+1)}) = \Psi(\boldsymbol{\alpha}^{(n)}) + \sum_{i=1}^{p} \delta\alpha_i^{(n)} \left.\frac{\partial\Psi}{\partial\alpha_i}\right|_{\boldsymbol{\alpha}^{(n)}} + O\left(\left[\delta\boldsymbol{\alpha}^{(n)}\right]^2\right) \tag{278}$$

$$= \Psi_{\text{lin}}^{(n+1)} + O\left(\left[\delta\boldsymbol{\alpha}^{(n)}\right]^2\right), \tag{279}$$

where $\Psi_{\text{lin}}^{(n+1)}$ is the linear sum

$$\Psi_{\text{lin}}^{(n+1)} = \sum_{i=0}^{p} a_i\phi_i, \tag{280}$$

with the coefficients $\{a_i\}$ and the basis functions $\{\phi_i\}$ defined as:

$$a_i = \begin{cases} 1 & i=0 \\ \delta\alpha_i^{(n)} & i \neq 0 \end{cases} \tag{281}$$

$$\phi_i = \begin{cases} \Psi(\boldsymbol{\alpha}^{(n)}) & i=0 \\ \left.\frac{\partial\Psi}{\partial\alpha_i}\right|_{\boldsymbol{\alpha}^{(n)}} & i \neq 0. \end{cases} \tag{282}$$

The form of $\Psi_{\text{lin}}^{(n+1)}$ allows $\{a_i\}$ to be optimized using diagonalization (the freedom of normalization in the resulting eigenvectors of coefficients can be exploited to demand that $a_0 = 1$). The energy minimization method in CASINO makes the approximation that

$$\Psi^{(n+1)} \simeq \Psi_{\text{lin}}^{(n+1)}, \tag{283}$$

and determines the parameter changes $\{\delta\alpha_i\}$ at each cycle by optimizing $\{a_i\}$ using diagonalization, and taking the eigenvector of coefficients corresponding to the lowest eigenvalue. This approach is motivated by the existence of a formulation of diagonalization for VMC which incorporates a zero-variance principle similar to that enjoyed by variance minimization [78].

In the standard derivation of diagonalization, $\Psi_{\text{lin}}^{(n+1)}$ would be optimized by demanding that the variational energy be stationary with respect to the variable parameters:

$$\frac{\partial}{\partial\alpha_i} \frac{\langle\Psi_{\text{lin}}^{(n+1)}|\hat{H}|\Psi_{\text{lin}}^{(n+1)}\rangle}{\langle\Psi_{\text{lin}}^{(n+1)}|\Psi_{\text{lin}}^{(n+1)}\rangle} = 0, \tag{284}$$

which leads to the generalized eigenproblem

$$(\mathbf{H} + \mathbf{H}^{\mathrm{T}})\mathbf{a} = E(\mathbf{S} + \mathbf{S}^{\mathrm{T}})\mathbf{a}, \tag{285}$$

where:

$$S_{ij} = \frac{\langle\phi_i|\phi_j\rangle}{\langle\Psi_{\text{lin}}^{(n+1)}|\Psi_{\text{lin}}^{(n+1)}\rangle}; \tag{286}$$

$$H_{ij} = \frac{\langle\phi_i|\hat{H}|\phi_j\rangle}{\langle\Psi_{\text{lin}}^{(n+1)}|\Psi_{\text{lin}}^{(n+1)}\rangle}. \tag{287}$$

Since $\mathbf{H}^{\mathrm{T}} = \mathbf{H}^*$, there is more than one possible way to VMC-estimate $\mathbf{H} + \mathbf{H}^{\mathrm{T}}$. For example, the choice

$$\left\langle \left(\frac{\phi_i}{\phi_0}\right)^* \left(\frac{\hat{H}\phi_j}{\phi_0}\right) + \left(\frac{\phi_j}{\phi_0}\right)^* \left(\frac{\hat{H}\phi_i}{\phi_0}\right) \right\rangle_{|\phi_0|^2} \tag{288}$$

ensures that the VMC estimate of the $\mathbf{H}$ matrix is symmetric, but does not ensure that it is real. The different choice

$$\left\langle \left(\frac{\phi_i}{\phi_0}\right)^* \left(\frac{\hat{H}\phi_j}{\phi_0}\right) + \left(\frac{\phi_i}{\phi_0}\right) \left(\frac{\hat{H}\phi_j}{\phi_0}\right)^* \right\rangle_{|\phi_0|^2} \tag{289}$$

ensures that it is real, but not necessarily symmetric. (The exact $\mathbf{H}+\mathbf{H}^{\mathrm{T}}$ is both real and symmetric.)

In Ref. [78], Nightingale and Melik-Alaverdian show that it is possible to rederive diagonalization as a least-squares fit, over a finite number of VMC configurations, to the ideal situation in which the basis functions $\{\phi_i\}$ span an invariant subspace of the Hamiltonian. Because this approach incorporates finite-sampling from the beginning, it removes the ambiguity over how to VMC-estimate $\mathbf{H}+\mathbf{H}^{\mathrm{T}}$. The analysis introduced in Ref. [78] (and used in Ref. [75, 76, 77]) assumes the wave function is real, but it is not difficult to extend it to the complex case: what follows is a very concise outline.

Assume that the basis functions span an invariant subspace of the Hamiltonian, i.e.,

$$\hat{H}|\phi_i\rangle = \sum_{j=0}^{p} \mathcal{E}_{ji}|\phi_j\rangle \quad \forall\, i, \tag{290}$$

meaning that $\Psi_{\mathrm{lin}}^{(n+1)}$ will be an eigenstate of $\hat{H}$ if $\mathbf{a}$ is an eigenvector of $\boldsymbol{\mathcal{E}}$. In general, $\{\phi_i\}$ are complex, but we choose to restrict $\{\mathcal{E}_{ji}\}$ to be real. Equation (290) cannot be exactly satisfied, so instead we seek an approximate solution for $\boldsymbol{\mathcal{E}}$ by minimizing:

$$\chi^2 = \sum_{\sigma=1}^{M}\sum_{i=0}^{p}\left|\hat{H}\phi_i(\mathbf{R}_\sigma) - \sum_{j=0}^{p}\mathcal{E}_{ji}\phi_j(\mathbf{R}_\sigma)\right|^2 . \tag{291}$$

Demanding that

$$\frac{\partial \chi^2}{\partial \mathcal{E}_{pq}} = 0 \quad \forall\, p, q, \tag{292}$$

and seeking the eigenvalues and eigenvectors of the resulting approximation to $\boldsymbol{\mathcal{E}}$, leads to exactly the same eigenproblem as in Eq. (285). The difference is that the VMC estimate of $\mathbf{H}+\mathbf{H}^{\mathrm{T}}$ is now specified, as:

$$\left\langle \left(\frac{\phi_i}{\phi_0}\right)^{*}\left(\frac{\hat{H}\phi_j}{\phi_0}\right) + \left(\frac{\phi_i}{\phi_0}\right)\left(\frac{\hat{H}\phi_j}{\phi_0}\right)^{*}\right\rangle_{|\phi_0|^2} . \tag{293}$$

Using this expression to estimate $\mathbf{H}+\mathbf{H}^{\mathrm{T}}$ gives the method a zero-variance principle: if the basis functions $\{\phi_i\}$ genuinely do span an invariant subspace of the Hamiltonian, exact diagonalization will be achieved for any number of configurations greater than p. In practice, this condition never holds true. Nevertheless, using this expression to VMC-estimate $\mathbf{H}+\mathbf{H}^{\mathrm{T}}$ massively reduces the statistical noise in the diagonalization process. As noted above, this expression does not guarantee that the estimate of $\mathbf{H}+\mathbf{H}^{\mathrm{T}}$ is symmetric. This means that the eigenvalues $\{E\}$ and $\{\mathbf{a}\}$ may not be real, but, since their imaginary parts only arise from statistical noise, we can simply discard them.

Using this method to optimize the trial wave function relies on the accuracy of the approximation made in Eq. (283). If all the parameters being optimized appear linearly in the wave function, the approximation is exact, and the global minimum of the energy will be found in one cycle.[24] If other parameters are included in the optimization, the energy will converge to a local minimum over several cycles, provided that the central approximation holds true.

25.3.3 Stabilization

The basic method described above can encounter two main problems: (i) linear dependencies in the basis functions $\{\phi_i\}$; (ii) parameter changes too large for the approximation of Eq. (283) to be valid. The first problem, which arises from redundancies in the parameters, leads to the algorithm attempting to invert a singular matrix. Exact redundancies in the parameters should be avoided when setting up the wave function, but parameters which are approximately redundant (to within the numerical precision of the computer) are harder to detect. The solution, as described in Ref. [78], is to use Singular Value Decomposition [79] for matrix inversions. The practical problems of redundant parameters are discussed in more detail in Sec. 25.3.7. The second problem can cause the optimization to diverge. Two techniques are used in CASINO to prevent divergence and stabilize the algorithm: semi-orthogonalization of the basis functions, and 'level-shifting' [80, 81] in either the $\mathbf{H}+\mathbf{H}^{\mathrm{T}}$ matrix or the $(\mathbf{S}+\mathbf{S}^{\mathrm{T}})^{-1}(\mathbf{H}+\mathbf{H}^{\mathrm{T}})$ matrix. Both ideas were first introduced to the method by Umrigar *et al.* [75, 76].

[24] In fact, a second cycle may sometimes improve the accuracy of the optimized parameters. The parameters resulting from the first cycle will be inexact because of statistical noise. The second cycle uses an improved wave function, which will result in lower noise when determining the second cycle's parameters.

25.3.4 Semi-orthogonalization

Consider multiplying Ψ by a (complex) renormalizing factor $A(\boldsymbol{\alpha})$ which does not depend on $\mathbf{R}$:

$$\tilde{\Psi}(\mathbf{R},\boldsymbol{\alpha}) = A(\boldsymbol{\alpha})\Psi(\mathbf{R},\boldsymbol{\alpha}). \tag{294}$$

The same parameter set minimizes the energy expectation value taken using both Ψ and $\tilde{\Psi}$, so we are free to choose any $A(\boldsymbol{\alpha})$. However, the energy expectation values taken using $\Psi_{\rm lin}$ and $\tilde{\Psi}_{\rm lin}$ are minimized by different parameter sets. We can choose $A(\boldsymbol{\alpha})$ to improve the parameter changes. The new basis $\{\tilde{\phi}_i\}$ is given by:

$$\begin{aligned}
\tilde{\phi}_0 &= A(\boldsymbol{\alpha}^{(n)})\phi_0; & (295)\\
\tilde{\phi}_i &= \left.\frac{\partial(A\Psi)}{\partial\alpha_i}\right|_{\boldsymbol{\alpha}^{(n)}} \\
&= A(\boldsymbol{\alpha}^{(n)})\phi_i + \phi_0 \left.\frac{\partial A}{\partial\alpha_i}\right|_{\boldsymbol{\alpha}^{(n)}} \quad (i \neq 0). & (296)
\end{aligned}$$

Following Ref. [75, 76], we choose $A(\boldsymbol{\alpha}^{(n)}) = 1$ (such that $\tilde{\phi}_0 = \phi_0$), and

$$\left.\frac{\partial A}{\partial\alpha_i}\right|_{\boldsymbol{\alpha}}^{(n)} = A_i(\boldsymbol{\alpha}^{(n)}) = -\frac{\langle\Lambda|\phi_i\rangle}{\langle\Lambda|\Psi^{(n)}\rangle} \quad \forall\, i. \tag{297}$$

The basis $\{\tilde{\phi}_i\}$ is 'semi-orthogonalized', in the sense that, while the basis functions are not orthogonal to one another, they are all orthogonal to a chosen wave function Λ. Umrigar *et al.* suggest choosing

$$\Lambda = \xi\frac{\Psi^{(n)}}{||\Psi^{(n)}||} + \sigma(1-\xi)\frac{\Psi_{\rm lin}^{(n+1)}}{||\Psi_{\rm lin}^{(n+1)}||}, \tag{298}$$

where $0 \leq \xi \leq 1$ is a parameter which can be chosen freely, and σ takes the value 1 when the angle between $\Psi^{(n)}$ and $\Psi_{\rm lin}^{(n+1)}$ is acute, and -1 when it is obtuse, i.e.,

$$\sigma = 1 \times {\rm sgn}\left[{\rm Re}(\langle\Psi^{(n)}|\Psi_{\rm lin}^{(n+1)}\rangle)\right]. \tag{299}$$

The primary purpose of this choice of Λ is simply to reduce the size of the parameter changes, although it also achieves certain other conditions for particular choices of ξ. This can be seen using geometrical arguments, presented and explained[25] in Ref. [77]. Although it is possible to vary ξ in CASINO (see Sec. 25.3.7), the default value of 0.5 is believed to be effective in all circumstances.

To use the semi-orthogonalized basis $\{\tilde{\phi}_i\}$, an expression for $A_i(\boldsymbol{\alpha}^{(n)})$ is needed. From expanding Eq. (297):

$$A_i = \frac{-\xi\bar{S}_{0i}D - \sigma(1-\xi)(\bar{S}_{0i} + \sum_{j=1}^p \delta\alpha_j\bar{S}_{ji})}{\xi D + \sigma(1-\xi)(1 + \sum_{j=1}^p \delta\alpha_j\bar{S}_{j0})}, \tag{300}$$

where

$$D = \sqrt{1 + \sum_{j=1}^p \delta\alpha_j(\bar{S}_{j0} + \bar{S}_{0j}) + \sum_{j,k=1}^p \delta\alpha_j\delta\alpha_k\bar{S}_{jk}}, \tag{301}$$

and $\bar{S}_{ij}$ is the VMC estimate of S_{ij} (note that, unlike for H_{ij}, there is no ambiguity over how to do these VMC estimates). These expressions differ very slightly from those appearing in Refs. [75, 76, 77], because here the wave function is allowed to be complex.

25.3.5 Matrix manipulation (level-shifting)

We can rewrite Eq. (285) as a simple eigenproblem:

$$(\mathbf{S} + \mathbf{S}^{\rm T})^{-1}(\mathbf{H} + \mathbf{H}^{\rm T})\mathbf{a} = E\mathbf{a}. \tag{302}$$

[25] The treatment in Ref. [77] does not include σ, but its necessity (noted by Umrigar *et al.*) is clear when one considers the case where the angle between $\Psi^{(n)}$ and $\Psi_{\rm lin}^{(n+1)}$ is obtuse. Also, in Ref. [77], Ψ is assumed to be real, although the extension to the complex case is simple, and the correct expressions are given here.

If the $(\mathbf{S}+\mathbf{S}^{\mathrm{T}})^{-1}(\mathbf{H}+\mathbf{H}^{\mathrm{T}})$ matrix had the form

$$\begin{pmatrix} \lambda_0 & 0 & 0 & 0 \\ 0 & \lambda_1 & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & \lambda_p \end{pmatrix}, \quad \lambda_0 < \lambda_i \quad \forall\, i = 1 \ldots p, \tag{303}$$

then there would be no change at all in the variable parameters $\boldsymbol{\alpha}$. It is possible to bring any $(\mathbf{S}+\mathbf{S}^{\mathrm{T}})^{-1}(\mathbf{H}+\mathbf{H}^{\mathrm{T}})$ matrix qualitatively closer to this 'no-change' condition using 'level-shifting' [80, 81], in which a positive real constant L is added to all the diagonal elements of the matrix except the first:

$$\begin{pmatrix} \lambda_0 & \bullet & \bullet & \bullet \\ \bullet & \lambda_1 & \bullet & \bullet \\ \bullet & \bullet & \ddots & \bullet \\ \bullet & \bullet & \bullet & \lambda_p \end{pmatrix} \rightarrow \begin{pmatrix} \lambda_0 & \bullet & \bullet & \bullet \\ \bullet & \lambda_1 + L & \bullet & \bullet \\ \bullet & \bullet & \ddots & \bullet \\ \bullet & \bullet & \bullet & \lambda_p + L \end{pmatrix}. \tag{304}$$

The approximation in Eq. (283) fails when the parameter changes are too large. Level-shifting can be used to reduce the size of the parameter changes, stabilizing the optimization. In CASINO, the best value for the constant L is determined automatically by performing a line minimization of the VMC energy estimate with respect to L. (Where possible, correlated sampling is used, both to reduce computational effort and to increase the statistical precision of energy differences.) As noted by Umrigar *et al.* [75, 76], level-shifting can equally well be applied to the $\mathbf{H}+\mathbf{H}^{\mathrm{T}}$ matrix.

In principle one would find the optimal value of parameter L by minimizing the mean energy of the VMC configurations using the resulting parameter values. However this could result in choosing a parameter set which yields an unphysically low mean energies for the fixed set of configurations. Such parameter sets are usually associated with very large variance of the local energy, and Ref. [76] suggests minimizing a linear combination of the energy and variance. This is of course problematic in that these quantities have dimensions of energy and squared energy, respectively, necessarily requiring system-dependent mixing parameters to produce equivalent target functions in different systems. CASINO instead optimizes L by minimizing the mean energy plus three standard deviations, which accomplishes the same goal in a universal manner.

25.3.6 Correlated sampling

As in the case of variance minimization, CASINO uses correlated sampling during energy minimization, performing several iterations of the basic algorithm on the same set of VMC configurations at different values of the wave function parameters. When starting from an "empty" Jastrow factor, variance minimization uses VMC configurations distributed according to a Jastrow-less wave function and introduces the "empty" Jastrow factor (usually containing non-zero cusp-enforcing parameters) in the first optimization cycle. Energy minimization is found to be sensitive to this initial mismatch between the configurations and the wave function, and in CASINO the initial configurations are instead generated using the "empty" Jastrow factor.

25.3.7 Using energy minimization

Energy minimization is used very similarly to standard variance minimization. After selecting it by setting **opt_method** to 'emin', the user must choose the number of optimization cycles (**opt_cycles**), and the number of optimization configurations per cycle (**vmc_nconfig_write**). As for variance minimization, **vmc_decorr_period** should be chosen so as to generate approximately uncorrelated configurations, and the number of configurations required is usually similar to the number required for variance minimization; increasing the number of configurations beyond typical values for variance minimization does however reduce the likelihood of unstable behaviour.

As described above, some of the candidate wave functions during level-shifting may be so poor that they result in unphysically low energy estimates which is actually spurious. In addition to using the mean energy plus three standard deviations as the target function in obtaining L, CASINO also implements a mean-energy threshold below which parameter sets are rejected, controlled by the keyword **emin_min_energy**. This parameter also affects diagonalization, with eigenvalues being thrown away if they are below **emin_min_energy**.

The default value of **emin_min_energy** is the VMC energy minus three times the square root of the variance of the local energies. This default value works well in practice, but in very special cases one might want to explicitly set **emin_min_energy** to ensure that the target ground-state energy is above it. For example, this would be the case of the hydrogen molecule with full quantum nuclear motion using exponential orbitals and a Jastrow factor, in which case the dissociated state (H+H) can be exactly described by the wave function, while the bound state (H_2) can be described only approximately. If a VMC cycle is run on the H+H wave function, the variance of the local energy will be zero, and the subsequent energy minimization cycle would set **emin_min_energy** to exactly the energy of H+H (-1 Hartree), effectively blocking the ability to optimize the wave function parameters to describe the lower-energy H_2 molecule (~ -1.165 Hartree). In this case it would be advisable to set **emin_min_energy** to, *e.g.*, `-1.2 hartree`.

Keyword **emin_xi_value**, mainly intended for developer use, controls the value of ξ in Eq. (298); the default value of 0.5 performs well in all tests.

As mentioned in Sec. 25.3.3, the presence of parameters which the energy is a shallow function of may adversely affect the optimization. CASINO offers the possibility of fixing parameters internally flagged as 'shallow' during the final **opt_noctf_cycles** cycles of a multi-cycle optimization run. However, if cut-off lengths or other 'shallow' parameters need to be optimized it is usually a better idea to increase the number of VMC configurations used in the configuration-generation stage.

Finally, a brief list of possibilities to investigate if an energy minimization run appears to misbehave:

- Having trouble optimizing from an "empty" Jastrow factor? You might want to start your optimization with a single cycle of, *e.g.*, **opt_method**: `madmin`.
- Are enough configurations being used? The stability of energy minimization depends on how many VMC configurations are used (set by **vmc_nconfig_write**), and the optimal value is usually larger than for variance minimization. Typically, if you can afford to use 10^5–10^6 configurations, it is a good idea to do so.
- Are the VMC configurations serially correlated? If the reported VMC correlation time is significantly greater than 1, you should increase **vmc_decorr_period**.
- Is **emin_min_energy** too high? (See above.)

26 Alternative sampling strategies

26.1 Summary

In VMC we directly use Monte Carlo sampling to estimate expectation values defined as the quotient of two integrals over $3N$ dimensional space, such as the total energy expectation value, $E_{\rm tot} = \langle\Psi|\hat{H}|\Psi\rangle/\langle\Psi|\Psi\rangle$. Estimation of integrals using Monte Carlo methods can generally be performed using samples drawn from any of a wide range of distributions. The usual VMC choice of $P(\mathbf{R}) = \lambda|\Psi^2(\mathbf{R})|$ (with λ an unknown and un-required normalization prefactor) is an analytically convenient choice, but is not in any sense optimal [82]. This leaves open the possibility of choosing a different sampling distribution, such as one that is optimal in the sense of providing the minimum statistical error for a given number of samples, or one that is maximally efficient in the sense of providing a given accuracy for less computational cost than other choices.

Unfortunately there are some strong limitations on the distribution functions that we can use. Firstly, we require the 'estimate' to be an *estimate*, that is, to converge to the true value with increasing sample size due to the law of large numbers (LLN) being valid. Secondly, we usually require meaningful error bars to be available for a finite sample size estimate, that is that the central limit theorem (CLT) is valid in some form. The validity of either of these conditions must be arrived at analytically, and if they are not true then the conventional formulae for the sample mean and standard error provide numerical results that are unrelated to the estimate and error. Examples of when this occurs are provided by force estimates [83] and parameter optimization [84].

In what follows, estimates constructed using an arbitrary distribution are described. Following this, three alternative choices of sampling distribution are summarized. The usual VMC choice is referred to as 'standard sampling', whereas the alternative methods are referred to as 'alternative sampling'.

26.2 Alternative sampling

For estimating the Hamiltonian expectation value using an arbitrary sampling distribution, $P(\mathbf{R})$, the estimate is composed of the quotient of two sample means

$$\bar{E}_{\rm tot} = \frac{\sum w(\mathbf{R}) E_{\rm L}(\mathbf{R})}{\sum w(\mathbf{R})}, \tag{305}$$

where $w(\mathbf{R}) = |\Psi^2|/P(\mathbf{R})$. In the limit of large sample size, this is a value drawn from a normal distribution with mean $E_{\rm tot}$ and variance

$$\sigma_E^2 = \frac{1}{N} \frac{\int |\Psi^2(\mathbf{R})|/w(\mathbf{R}) d^3\mathbf{R} \int w(\mathbf{R}) |\Psi^2(\mathbf{R})|^2 (E_{\rm L}(\mathbf{R}) - E_{\rm tot})^2 \, d^3\mathbf{R}}{\left[\int |\Psi^2(\mathbf{R})|^2 d^3\mathbf{R}\right]^2}, \tag{306}$$

whose estimate is

$$\bar{\sigma_E^2} = \frac{N}{N-1} \frac{\sum w(\mathbf{R})^2 \left(E_{\rm L}(\mathbf{R}) - \bar{E}_{\rm tot}\right)^2}{\left[\sum w(\mathbf{R})\right]^2} \tag{307}$$

as long as the *bivariate* CLT is valid. For standard sampling as described below, the random error is normal for energy estimates, but is not normal for most other estimates such as the energy surface implicit in optimization. For the three alternative sampling strategies, the random error is normal for most estimates, including the energy surface implicit in optimization.

Note that the estimates given above are not the sample mean and standard error of any set of independent identically distributed random variables. In the `input` file this form of alternative sampling is activated by setting the **vmc_sampling** keyword to one of the values described below.

26.2.1 Standard sampling

For the standard choice of sampling distribution the weight w is constant, and the usual estimates and standard error are recovered. This can be activated for alternative sampling by setting **vmc_sampling** to 'standard' (the default).

26.2.2 Optimum sampling

Since we have an analytic expression for random error we may seek the sampling distribution that provides the lowest statistical error for a given sample size. A somewhat generalized version of this distribution is given by

$$P(\mathbf{R}) = \lambda |\Psi^2(\mathbf{R})| \left[(E_{\rm L}(\mathbf{R}) - E_0)^2 + 2\epsilon^2 \right]^{\frac{1}{2}}, \tag{308}$$

where (E_0, ϵ) are user-supplied parameters (**vmc_optimum_e0** and **vmc_optimum_ew**). The best values for these parameters are $E_0 = E_{\rm tot}$ and $\epsilon = 0$, which provide the lowest possible statistical error for a given number of samples, but using a rough estimate with an accompanying error usually brings us close to this. It is worth emphasizing that the choice of parameters does not bias the estimate—for example, any E_0 with $\epsilon \to \infty$ provides a valid sampling distribution (standard sampling)—but it is advantageous to have $\epsilon \approx |E_{\rm tot} - E_0|$. A particular special case that should be avoided is $\epsilon = 0$, for which both the LLN and CLT are invalid and the quotient of weighted means given above is not an estimate (unless E_0 is exactly equal to the quantity we are estimating).

If no values are provided by the user they are set *ad hoc* to $E_0 = 0$ and $\epsilon = 100$, which usually gives similar accuracy to standard sampling. A rule of thumb is to use the best VMC total energy estimate available so far (e.g., from a test calculation), or to use the best *ab initio* total energy estimate for E_0 and 10% of this for ϵ. During optimization **vmc_optimum_e0** and **vmc_optimum_ew** are updated after each cycle to the best estimates available so far.

Although this sampling strategy reduces random error in the estimate for a given system to close to the best value possible with a given sample size, it can be more expensive; for example, for all-electron atomic calculations with accurate trial wave functions such sampling results in a $\times 2$ increase in computational cost for a given accuracy due to long correlation times in the Metropolis algorithm for these systems [84].

This sampling scheme can be activated by setting **vmc_sampling** to 'optimum', together with values for E_0 and ϵ via the keywords **vmc_optimum_e0** and **vmc_optimum_ew**, respectively (notice that these are of type 'physical' and require energy units to be specified, as in '37.45 hartree').

26.2.3 Simplified optimum sampling

We may attempt to achieve a more accurate estimate for a given computational cost by approximating the above optimum distribution with something that increases the fixed-sample-size error a little, but that allows the sample size to be increased. A computationally cheaper candidate implemented in the code is referred to as *simplified optimum sampling*, and corresponds to sampling the optimum distribution associated with a *simplified* version of the actual trial wave function. Note that this simplification occurs only in the sampling distribution; the expectation value estimated is still that of the full trial wave function.

A number of different forms of trial wave functions are implemented in the code, and we limit ourselves to a multideterminant expansion combined with a Jastrow factor and backflow. For these it is natural to take the dominant determinant, remove the Jastrow factor, and remove the backflow to provide a computationally undemanding wave function, Φ, whose optimum sampling distribution,

$$P(\mathbf{R}) = \lambda|\Phi^2(\mathbf{R})|\left[(E_{\rm L}(\mathbf{R})[\phi] - E_0)^2 + 2\epsilon^2\right]^{\frac{1}{2}}, \tag{309}$$

may be used, in which the local energy is that of the simplified trial wave function.

A good choice of parameters are as suggested for optimum sampling, but with ϵ replaced by a rough overestimate of the correlation energy in the system such as 10% of best total energy available so far. During optimization E_0 is updated, whereas ϵ is left unchanged.

This sampling strategy reduces random error by a smaller amount than optimum sampling for a given sample size, but is computationally cheaper. For example, for accurate trial wave functions in all-electron atomic calculations the efficiency is comparable to standard sampling [84].

It may be activated by setting **vmc_sampling** to 'HF optimum', together with values for E_0 and ϵ via the keywords **vmc_optimum_e0** and **vmc_optimum_ew**, as above.

26.2.4 Efficient sampling

We may reduce computational expense further at the cost of an increase in fixed-sample-size error, and still obtain a net increase in performance when compared to standard sampling. This is achieved by avoiding the evaluation of the Jastrow factor, backflow function, multideterminant expansion and local energy that appears in the previous distributions and using the form

$$P(\mathbf{R}) = \lambda\left[\left(D_1^{\uparrow}(\mathbf{R})D_1^{\downarrow}(\mathbf{R})\right)^2 + \left(D_2^{\uparrow}(\mathbf{R})D_2^{\downarrow}(\mathbf{R})\right)^2\right]. \tag{310}$$

This is an arbitrary choice that is *not* the mod-square of any fermionic trial wave function, is not an approximation to any physical quantity, and does not possess a nodal surface. This choice is not unique, but has been specifically chosen such that the LLN and CLT are valid and for the increase in fixed-sample-size error to be manageable (note that the validity of the CLT relies on the presence of *two* terms in this expansion, and dropping the second term results in an invalid estimate.) As implemented we do not sum the mod-square of the first two determinants, but the mod-square of the first two configuration state functions (CSFs), in order to avoid naturally the coincidence of nodal surfaces for the two terms through symmetry.

For such a choice we achieve a net gain in computational efficiency even though the fixed-sample-size error is greater, since more samples are accessible for a given computational budget. For example, to achieve a given accuracy in all-electron atomic carbon calculations, standard sampling takes $\times 25$ longer than efficient sampling [84].

Efficient sampling can be activated by setting **vmc_sampling** to 'efficient'. The first two CSFs are included in the sum as above.

26.2.5 Optimization

Optimization proceeds as for standard sampling, with the same control keywords and with variance and energy minimization defined using estimates of the same integral expressions using the different sampling distributions provided above. The only significant change arises for variance minimization. It is often not clear whether to interpret the optimized quantity in terms of a Monte Carlo estimate or as a least-squares fitting with random sample points, hence many interpretations arise that are equivalent for standard sampling but differ for general sampling. In addition, many of the variations of variance minimization available are specifically designed to prevent the failure of the CLT during optimization, something that does not occur for any of the alternative sampling methods presented here. As it stands the code minimizes only one quantity in variance minimization: the Monte Carlo estimate of the actual error in the estimated energy, $\tilde{\sigma}^2_{\bar{E}}$. Note that this has not been tested, and a wide range of other options may perform better, such as minimizing an estimate of the optimum or standard error.

27 Use of localized orbitals and bases in CASINO

27.1 Theoretical background

27.1.1 Introduction

The rate-determining step in practical QMC calculations is the evaluation of the orbitals in the Slater part of the trial wave function after each electron has moved[26]. We explain how the amount of CPU time spent evaluating the orbitals after each electron move can be made essentially independent of system size.

27.1.2 'Standard' QMC

Let the number of electrons in our system be N. In 'standard' QMC the orbitals are HF or DFT eigenfunctions extending over the entire system. On the other hand, the basis functions used to represent those orbitals are usually localized functions. Each time an electron is moved, all $\mathcal{O}(N)$ occupied orbitals must be evaluated; however, only the $\mathcal{O}(1)$ basis functions in the vicinity of the electron need to be computed to evaluate each orbital. So, the time taken to carry out a configuration move of all N electrons in the system is $\mathcal{O}(N^2)$.

If the orbitals are represented by extended basis functions, such as plane waves, then the time taken for a configuration move scales as $\mathcal{O}(N^3)$, because every basis function in every orbital has to be computed for every electron.

27.1.3 Localized orbitals

It is easy to show that a nonsingular linear transformation of a set of orbitals can only change the normalization of the Slater determinant of those orbitals. For insulating materials there exist straightforward and efficient algorithms for determining linear transformations to highly localized sets of orbitals [85, 86].

Highly localized orbitals can be truncated to zero at a certain distance from their centres without introducing a substantial bias. Therefore, each time an electron is moved, only a few orbitals need to be evaluated; the others must be zero because the electron lies outside their truncation radii. So the number of orbitals to be computed after each electron move is $\mathcal{O}(1)$ [87].

The truncation of the orbitals results in small discontinuities in the Slater wave function. These are potentially serious for QMC because they result in the presence of Dirac delta functions in the kinetic-energy integrand. The delta functions cannot be sampled, so their contribution to the total energy is lost. However, in practice the resulting bias is extremely small, provided the truncation radii are sufficiently large. In fact the bias can be made arbitrarily small by increasing the truncation radii.

The discontinuities can be avoided by bringing the localized orbitals smoothly to zero over a thin, spherical skin. Surprisingly, it has been shown that the bias resulting from the use of smooth truncation schemes is larger than the bias that occurs if the orbitals are truncated abruptly at their cutoff radii [88]. The reason for this is that the smooth truncation schemes introduce a new, small length scale (the skin thickness) into the problem, and the local kinetic energy in the truncation region is extremely large. A time step that is physically reasonable for the system being simulated is much too large for the length-scale introduced by the truncation region. This results in large time-step bias and frequent

[26]The time spent updating the cofactor matrices will in principle dominate in the limit of large system size, but this limit is not usually reached in practice. Note that, throughout this section, we assume that electron-by-electron QMC algorithms are used.

population-explosion catastrophes. We therefore recommend truncating localized orbitals abruptly: the **bsmooth** input parameter should be set to `F`.

27.1.4 Localized bases

Suppose the basis functions are zero outside fixed radii about their centres. Then the only functions that have to be evaluated when an electron is moved are those with the electron inside their radii. So the number of basis functions to be computed simply depends on the local environment of the electron and is therefore independent of system size.

Gaussian basis functions can be regarded as localized if they are truncated to zero outside a certain radius. This is done by default in CASINO: Gaussian functions $\exp(-ar^2)$ are assumed to be zero when $\exp(-ar^2) < 10^{-G_\mathrm{T}}$, where G_T is the **gautol** input parameter. Gaussian basis sets cannot yet be used in conjunction with localized orbitals, however.

Alternatively, orbitals can be represented numerically using blip functions, which also constitute a localized basis. If the orbitals are localized then the memory requirements are greatly reduced (by a factor of $\mathcal{O}(N)$), because we only have to store the blip coefficients needed to evaluate the orbital within its truncation radius.

27.1.5 'Linear-scaling' QMC

If the numbers of orbitals and basis functions to be evaluated are both $\mathcal{O}(1)$ then the CPU time for a configuration move scales as $\mathcal{O}(N)$. This is what is meant by 'linear-scaling QMC'. Note, however, that the number of configuration moves required to achieve a given error bar on the total energy scales as $\mathcal{O}(N)$; hence, in practice, the CPU time for 'linear-scaling' QMC calculations scales as $\mathcal{O}(N^2)$.

27.2 Using CASINO to carry out 'linear-scaling' QMC calculations

27.2.1 Generation of Bloch orbitals

At present the Bloch orbitals must be represented in a plane-wave basis. A plane-wave DFT code should be used to generate a `pwfn.data` file as though an ordinary QMC calculation with a plane-wave basis were to be carried out. Note that only one **k** point may be used: the Γ point. This is not a severe restriction since, for large systems, one would usually only carry out calculations at Γ anyway.

27.2.2 Generation of localized orbitals

The `localizer` code should be used to carry out the linear transformation to a localized set of orbitals. The code has the following features:

- `localizer` implements the method described in Ref. [85].
- `localizer` requires a `pwfn.data` file holding the Bloch orbitals represented in a plane-wave basis and a `centres.dat` file of format:

```
Number of centres
 <N>
Display coefficients of linear transformation (0=NO; 1=YES)
 <iprint>
Use spherical (1) or parallelepiped (2) localization regions
 <icut>
x,y & z coords of centres ; radius ; no. orbs on centre (up & dn)
 <pos(1,1)> <pos(2,1)> <pos(3,1)> <radius(1)> <norbs_up(1)> <norbs_dn(1)>
 ...
 <pos(1,N)> <pos(2,N)> <pos(3,N)> <radius(N)> <norbs_up(N)> <norbs_dn(N)>
```

  where 'N' is the total number of localization centres. If 'iprint' is 1 then the coefficients of the linear combination are written to `stdout`; otherwise they are not. 'icut' can take values 1 or 2, specifying that the localization regions are spherical or parallelepiped-shaped, respectively. 'pos(i,j)' is the ith Cartesian component of the position vector of the jth centre. 'radius(j)' is the cutoff radius for the jth localization centre. 'norbs_up(j)' and 'norbs_dn(j)' are the number of spin-up and spin-down orbitals to be localized on the jth centre, respectively. Note that if a parallelepiped-shaped localization region is used then the shape of the parallelepiped is defined by the lattice vectors, but the distance to each face is given by the cutoff radius.

- The choice of localization centres requires some chemical intuition. See the discussion in Ref. [85]. At some point in the future, the optimization of the localization centres will be enabled. Note that in some highly symmetric molecules, symmetric choices of localization centres can lead to two localized orbitals being identical. This problem can be avoided by breaking the symmetry of the localization centres.
- Note that the orbitals will become linearly dependent as two centres approach one another. In the limit that two centres are located in the same place, the orbitals localized on those centres will be identical. (One should instead define a single centre and increase the number of orbitals localized on that centre.)
- `localizer` produces a `pwfn.data.localized` file that contains the localized orbitals represented in the same plane-wave basis as that specified in `pwfn.data`.
- `localizer` can only work with Bloch orbitals at Γ. Since it is intended for use in large systems, this should not be a serious restriction.
- One can only choose different numbers of localized orbitals for spin-up and spin-down electrons if `pwfn.data` contains spin-polarized data.
- If the number N of localized orbitals specified in `centres.dat` is less than the number M of orbitals included in `pwfn.data` then only the first N orbitals are included in the localization transformation; the remainder are left as extended orbitals. The orbitals in `pwfn.data` are usually arranged in ascending order of eigenvalue, so the orbitals left out of the localization transformation are those with the highest eigenvalue. If you want to leave particular orbitals of lower eigenvalue out of the transformation, you could edit the `pwfn.data` file to put the orbitals that are to be localized at the start of the file.

27.2.3 Generation of a blip representation of the localized orbitals

A truncated blip representation of a set of localized orbitals can be generated using a blip-generation calculation (**runtype**='gen_blip'), which was described in Sec. 9. In a blip-generation calculation, `casino` reads in (i) a `pwfn.data` file containing plane-wave orbital data (`pwfn.data_localized` should be renamed `pwfn.data` beforehand) and (ii) a `centres.dat` file. Note that the latter file is optional. CASINO produces a `bwfn.data` file holding a blip representation of the localized orbitals. The format of the `centres.dat` file is the same as that which is read by `localizer`, except that two lines may added to the end of the file:

```
Number of centres
 <N>
Display coefficients of linear transformation (0=NO; 1=YES)
 <iprint>
Use spherical (1) or parallelepiped (2) localization regions
 <icut>
x,y & z coords of centres ; radius ; no. orbs on centre (up & dn)
 <pos(1,1)> <pos(2,1)> <pos(3,1)> <radius(1)> <norbs_up(1)> <norbs_dn(1)>
 ...
 <pos(1,N)> <pos(2,N)> <pos(3,N)> <radius(N)> <norbs_up(N)> <norbs_dn(N)>
Minimum skin thickness (a.u.)       <- OPTIONAL LINE
 0.d0                               <- OPTIONAL LINE
```

- If a `centres.dat` file is not present when a blip-generation calculation is performed then the orbitals in `pwfn.data` will be represented by a blip grid spanning the entire simulation cell. Likewise, any orbitals that are not included in the localization transformation described in `centres.dat` will be represented by a blip grid that spans the entire cell.
- If one requests that the kinetic energy be calculated then CASINO will calculate the kinetic energies of the orbitals represented by plane waves and the truncated orbitals represented by blips. Note that the kinetic energies of nonorthogonal orbitals cannot be summed to obtain the

HF kinetic energy. Nevertheless, the kinetic energies of the plane-wave and blip orbitals should be in agreement. Casino will also calculate the ratio of the squared norm of the truncated orbitals to that of the original plane-wave ones. This fraction should be very close to 1. For example, percentages in excess of 99.7% are typical.

- The smallest array of blip grid points that completely spans the requested localization region (including the 'minimum skin thickness') is worked out for each state. The skin thickness is then given the largest possible value such that the localization region is contained within the region spanned by the localized blip grid. The requested localization radius (excluding the skin thickness) is referred to as the *inner truncation radius* and the localization radius (including the actual skin thickness) is referred to as the *outer truncation radius*. In the subsequent QMC calculation, if **bsmooth** is `T` then the orbital is brought smoothly to zero between these radii. If **bsmooth** is `F` then the abrupt truncation of the orbital occurs at the outer truncation radius. It is recommended that **bsmooth** be set to `F` and that the minimum skin thickness be set to 0 (which is the default if it is not specified).

27.2.4 Running a QMC calculation with localized orbitals

The `bwfn.data` file containing the truncated, localized orbitals can be read by CASINO. The usual input files (`input`, `correlation.data` and pseudopotentials) should also be supplied. The **atom_basis_type** input parameter should be set to 'blip'.

Please note the following:

- It is possible to use blip orbitals to study systems that are finite, or periodic in just the x direction, or in the (x, y) plane, or are periodic in all three dimensions by setting the **blip_periodicity** keyword to 0, 1, 2 or 3, respectively. The **periodic** keyword must be set to `T` if the system is periodic in any direction. Note that if reduced-periodicity systems are studied then the atoms should be placed in the centre of the unit cell spanned by the blip grid, i.e., the unit cell defined by the lattice vectors.
- When excitations are specified, it should be noted that band indices start at the first nonlocalized band. For example, if all bands apart from the highest occupied molecular orbital (HOMO) are localized then the HOMO is band 1. Likewise, when CASINO reports the number of bands occupied at a given $\mathbf{k}$ point, it does not include the number of localized orbitals.
- Multideterminant calculations involving localized orbitals are possible, but all localized orbitals must be occupied in every determinant. (It only makes sense to carry out a localization transformation between the set of occupied orbitals.)
- The **bsmooth** input parameter is used to specify whether localized orbitals with spherical truncation surfaces should be abruptly or smoothly truncated. It is recommended that **bsmooth** be set to `F`.
- For large systems it is advisable to set the **sparse** input parameter to `T` in order to exploit the fact that most of the orbital values are zero when updating cofactor matrices used in the evaluation of the Slater determinants.

28 Twist averaging in QMC

28.1 Periodic and twisted boundary conditions

Suppose we wish to calculate the energy per particle of a periodic solid. In one-electron theories we can often reduce the problem to the primitive unit cell and integrate over the first Brillouin zone. Reduction to the primitive cell is not possible in many-body calculations, however, because correlation effects may be long-ranged; hence such calculations must be performed in periodic simulation cells consisting of several primitive cells.

Suppose the simulation cell is of volume Ω and contains N electrons, and let $\{\mathbf{r}_1, \ldots, \mathbf{r}_N\}$ be the electron coordinates. The Hamiltonian $\hat{H}$ must satisfy

$$\hat{H}(\mathbf{r}_1, \ldots, \mathbf{r}_i + \mathbf{R}_\mathrm{s}, \ldots, \mathbf{r}_N) = \hat{H}(\mathbf{r}_1, \ldots, \mathbf{r}_i, \ldots, \mathbf{r}_N) \qquad \forall i \in \{1, \ldots, N\}, \tag{311}$$

where $\mathbf{R}_\mathrm{s}$ is a simulation-cell lattice vector. This translational symmetry leads to the many-body Bloch condition

$$\Psi_{\mathbf{k}_\mathrm{s}}(\mathbf{r}_1, \ldots, \mathbf{r}_N) = U_{\mathbf{k}_\mathrm{s}}(\mathbf{r}_1, \ldots, \mathbf{r}_N) \exp\left(i\mathbf{k}_\mathrm{s} \cdot \sum_i \mathbf{r}_i \right), \tag{312}$$

where U has the periodicity of the simulation cell for all electrons [89]. The use of a nonzero simulation-cell Bloch vector $\mathbf{k}_\mathrm{s}$ is sometimes described as the application of *twisted* boundary conditions [90].

In a finite simulation cell subject to periodic boundary conditions, each single-particle orbital is usually taken to be of Bloch form $\psi_\mathbf{k}(\mathbf{r}) = \exp[i\mathbf{k} \cdot \mathbf{r}]u_\mathbf{k}(\mathbf{r})$, where $u_\mathbf{k}$ has the periodicity of the primitive cell and $\mathbf{k}$ lies on the grid of integer multiples of the simulation-cell reciprocal-lattice vectors within the first Brillouin zone of the primitive cell, the grid being offset from the origin by the simulation-cell Bloch vector $\mathbf{k}_\mathrm{s}$. For metallic systems some of the ground-state orbitals are nonanalytic functions of $\mathbf{k}$, because the occupancy depends on $\mathbf{k}$. Instead of integrating over single-particle orbitals inside the Fermi surface to calculate the HF kinetic and exchange energies, one sums over a discrete set of $\mathbf{k}$ vectors when a finite cell is used. As the system size is increased, the fineness of the grid of single-particle Bloch $\mathbf{k}$ vectors increases, and the HF energy changes abruptly as shells of orbitals pass through the Fermi surface.

One usually finds that the fluctuations in the QMC energy due to single-particle finite-size effects are proportional to the corresponding fluctuations in the HF kinetic energy or the DFT energy. Hence HF or DFT energy data can be used to extrapolate QMC energies to infinite system size. Note that a judicious choice of $\mathbf{k}_\mathrm{s}$ (e.g., the Baldereschi point [91] for insulators) can greatly reduce single-particle finite-size errors [89]. The common choice of $\mathbf{k}_\mathrm{s} = \mathbf{0}$ generally maximizes shell-filling effects and is therefore usually the worst possible value for estimating the total energy, although it does maintain the correct symmetry of the system.

Twist averaging within the canonical ensemble means taking the average of expectation values over all simulation-cell Bloch vectors $\mathbf{k}_\mathrm{s}$ in the first Brillouin zone of the simulation cell, i.e., over all offsets to the grid of $\mathbf{k}$ vectors, keeping the number of electrons fixed [90]. At the HF level, the effect of twist averaging is to replace sums over the discrete set of single-particle orbitals by integrals over the volume of reciprocal space defined by placing copies of the simulation-cell Brillouin zone around each ground-state occupied $\mathbf{k}$ point. The boundary of this region tends to the Fermi surface in the limit of infinite system size and the region has the correct volume at all system sizes. Twist averaging removes a large part of the error caused by the use of a discrete set of orbitals, although some residual errors remain because the shape of the Fermi surface is not quite correct. At the single-particle level this gives a small positive bias to the KE, in which shell-filling effects are still visible.

28.2 Using twisted boundary conditions in CASINO

28.2.1 Complex wave functions and the fixed-phase approximation

The procedure described in Sec. 15 for constructing real orbitals cannot be carried out if orbitals at arbitrary $\mathbf{k}$ points are used. In general the trial wave function must be complex under twisted boundary conditions. To specify that a complex wave function is to be used in CASINO the **complex_wf** keyword should be set to `T`. Using complex arithmetic causes CASINO to run more slowly, so **complex_wf** should be set to `F` wherever possible.

The use of a complex wave function makes very little difference to the VMC and wave-function optimization algorithms. Since the expectation values of Hermitian operators are real, only the real parts of the local-energy components are calculated and gathered.

The use of a complex trial wave function necessitates the use of the fixed-*phase* approximation in DMC [92], in which DMC wave function is constrained to have the same phase as the trial wave function. In fact it turns out that the fixed-node algorithm needs very few modifications to allow the use of complex wave functions within the fixed-phase approximation: the real part of the drift vector is used when proposing trial electron moves; it is not possible to reject node-crossing electron moves[27]; and, as in VMC, only the real parts of the local energies are gathered. The fixed-phase approximation is discussed at greater length in Sec. 39.

[27]In fact rejecting moves that cross the nodal surface has almost no effect on the time-step bias in systems with real wave functions anyway.

28.3 Monte Carlo twist averaging within CASINO

With CASINO twist-averaging can be done in two ways depending on the type of system. For electron(–hole) fluid phases the procedure can be performed wholly within CASINO. Unfortunately this is not the case for real systems with atoms since a new `xwfn.data` file is required for each twist angle, and such calculations must therefore be done in conjunction with an external wave-function generating code. These two methods are described below.

28.3.1 Electron(–hole) fluid phases

For electron-hole fluid phases [90] the following procedure is used. At specified intervals during the simulation, a new **k**-vector offset is chosen at random, then a short period of equilibration is carried out before statistics accumulation is resumed.

To specify an offset $\mathbf{k}_\mathrm{s}$ to the grid of **k** vectors for an electron(–hole) fluid, use the **k_offset** keyword (or the **k_offset_frac** keyword) in the **free_particles** block in `input`. This should be followed by the Cartesian components of the offset vector. For example:

```
%block free_particles
r_s 3
dimensionality 3
cell_geometry
0.0 0.5 0.5
0.5 0.0 0.5
0.5 0.5 0.0
particle 1 det 1 : 9 orbitals free
particle 2 det 1 : 9 orbitals free
k_offset 0.1 0.2 0.3
%endblock free_particles
```

Note that ground-state calculations for electron(–hole) fluids with real wave functions must have 'magic' numbers of electrons, such that there are no partially filled shells of **k** vectors. If the wave functions is complex, however, then any number of electrons may be used. This reflects the fact that for general twisted boundary conditions each shell contains just one **k** vector.

Anyway, to specify that twist averaging is to be used in VMC, set **vmc_ntwist** to the required number of twist angles to sample. After each change of the **k**-vector offset, **vmc_reequil_nstep** VMC equilibration moves will be carried out. Note that **vmc_reequil_nstep** can be much smaller than **vmc_equil_nstep**, because the distribution of configurations obtained at different **k**-vector offsets should be similar. A typical value for **vmc_reequil_nstep** might be 100.

To specify that twist averaging is to be used in DMC, set **dmc_ntwist** to the desired number of twist angles to sample. In the statistics-accumulation phase of DMC the following process will occur **dmc_ntwist** times: (i) a random **k**-vector offset will be chosen, (ii) **dmc_reequil_nstep** DMC equilibration moves will be carried out, but no data will be written to `dmc.hist` and (iii) **dmc_stats_nstep** statistics-accumulation moves will be carried out. So the total amount of data generated is **dmc_ntwist** $\times$ **dmc_stats_nstep**. Note that the number of post-twist-change equilibration moves **dmc_reequil_nstep** can be much less than the number of equilibration moves **dmc_equil_nstep** because the configuration distribution at one twist angle is a reasonably good approximation to the configuration distribution at another.

Obviously, if twist averaging is to be used in either VMC or DMC then **complex_wf** should be set to `T`. The twist-averaged HF energy of the HEG will be calculated and written to the `out` file for single-determinant ground-state calculations. One can check that **vmc_reequil_nstep** and **dmc_reequil_nstep** are sufficiently large by setting **use_jastrow**, **backflow** and **ibran** to `F` and comparing the VMC and DMC energies with the twist-averaged HF value (they should be the same).

Note that one cannot expect to find a plateau when performing reblocking analysis until the block length is greater than the number of data points between changes of twist angle, i.e., greater than **vmc_nstep** for VMC data and **dmc_stats_nstep** for DMC data. In any case, we recommend using the `mcta_post_process`, `twistanalysis_heg_external` and `twistanalysis_heg_internal` utilities to analyse the result of a Monte Carlo twist averaged calculation for the HEG; see Sec. 28.3.2.

28.3.2 Statistical analysis of Monte Carlo twist averaged (MCTA) calculations

MCTA-QMC calculations exhibit jumps in the local energy when the twist angle changes, resulting in large sample variances and difficulties to converge the result to the target error bar. In the case of the homogeneous electron gas (HEG) it is possible to model the jumps in the local energy and reduce their effect very significantly, since these jumps correlate very strongly with the HF energy components [93]. An example of this is shown in Fig. 1 for an MCTA-VMC calculation.

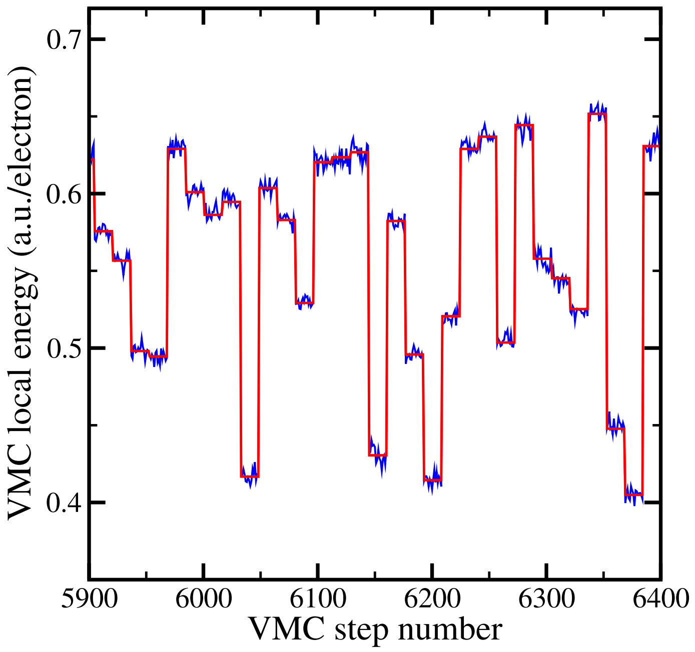


Figure 1: Local energies in a MCTA-VMC calculation (blue) and offsets induced by twist-angle changes (red).

Suppose we have knowledge of a function of the twist angle F that mirrors the jumps in the local energy as closely as possible. Then we can define a modified local energy $\tilde{E}_L$,

$$\tilde{E}_L(\mathbf{R};\mathbf{k}_{\text{offset}}) = E_{\text{L}}(\mathbf{R};\mathbf{k}_{\text{offset}}) - F(\mathbf{k}_{\text{offset}}) \; , \tag{313}$$

whose variance is substantially smaller than that of E_{L}. In this context, F is called a *control variate*. We can then evaluate the QMC energy as

$$E_{\text{QMC}} = \langle \tilde{E}_L \rangle + \langle F \rangle \; , \tag{314}$$

with squared error bar

$$\sigma^2_{E_{\text{QMC}}} = \sigma^2_{\langle \tilde{E}_L \rangle} + \sigma^2_{\langle F \rangle} \; . \tag{315}$$

Given that the sample variance of $\tilde{E}_L$ is smaller than that of E_{L}, this error bar is much smaller than that obtained from simply averaging E_{L}, provided that $\langle F \rangle$ has been evaluated to an accuracy such that $\sigma_{\langle F \rangle} \ll \sigma_{\langle \tilde{E}_L \rangle}$.

In the case of the HEG we can set F to

$$F(\mathbf{k}_{\text{offset}}) = aK^{\text{HF}}(\mathbf{k}_{\text{offset}}) + bX^{\text{HF}}(\mathbf{k}_{\text{offset}}) \; , \tag{316}$$

where $K^{\text{HF}}(\mathbf{k}_{\text{offset}})$ and $X^{\text{HF}}(\mathbf{k}_{\text{offset}})$ are the finite-system HF kinetic and exchange energies at the given $\mathbf{k}_{\text{offset}}$, and a and b are free parameters. Minimizing the variance of $\tilde{E}_L(\mathbf{R};\mathbf{k}_{\text{offset}})$ with respect to a and b yields

$$a = \frac{c_2c_3 - c_4c_5}{c_5^2 - c_1c_2} \; , \tag{317}$$

and

$$b = \frac{c_1c_4 - c_3c_5}{c_5^2 - c_1c_2} \; , \tag{318}$$

where

$$
\begin{aligned}
c_1 &= \langle (K^{\rm HF})^2 \rangle - \langle K^{\rm HF} \rangle^2 \,, & (319)\\
c_2 &= \langle (X^{\rm HF})^2 \rangle - \langle X^{\rm HF} \rangle^2 \,, & (320)\\
c_3 &= \langle E_{\rm L} K^{\rm HF} \rangle - \langle E_{\rm L} \rangle \langle K^{\rm HF} \rangle \,, & (321)\\
c_4 &= \langle E_{\rm L} X^{\rm HF} \rangle - \langle E_{\rm L} \rangle \langle K^{\rm HF} \rangle \,, & (322)\\
c_5 &= \langle K^{\rm HF} X^{\rm HF} \rangle - \langle K^{\rm HF} \rangle \langle K^{\rm HF} \rangle \,. & (323)
\end{aligned}
$$

With the above, one can obtain the QMC energy as

$$E_{\rm DMC} = \langle \tilde{E}_L \rangle + \langle F \rangle \,, \tag{324}$$

with squared error bar

$$\sigma^2_{E_{\rm DMC}} = \sigma^2_{\langle \tilde{E}_L \rangle} + \sigma^2_{\langle F \rangle} \,. \tag{325}$$

Ideally $\langle F \rangle$ should be evaluated to an accuracy such that $\sigma_{\langle F \rangle} \ll \sigma_{\langle \tilde{E}_{\rm L} \rangle}$, and should not be evaluated from the set of twist angles used in the MCTA-QMC calculation.

The `mcta_post_process` script implements this method for HEGs. A similar method could be applied to any system where the jumps in the local energy are accurately modelled as a function of quantities that can be computed inexpensively.

An alternative (equivalent) way of viewing the situation is that we wish to fit the function

$$E_i = \bar{E} + b(K_i^{\rm HF} - \bar{K}^{\rm HF}) + c(X_i^{\rm HF} - \bar{X}^{\rm HF}) \tag{326}$$

to the QMC energy per particle at each twist i, where $K_i^{\rm HF}$ and $X_i^{\rm HF}$ are the HF kinetic and exchange energies per particle at that twist, and $\bar{K}^{\rm HF}$ and $\bar{X}^{\rm HF}$ are the averages of these quantities, which can be calculated with very high precision. There are three parameters to be determined by fitting: b, c, and the twist-averaged energy $\bar{E}$. The parameter b is an approximation to the inverse of the quasiparticle effective mass. This is the approach used by the `twistanalysis_heg_external` and `twistanalysis_heg_internal` utilities.

One can use this approach to remove some of the residual single-particle errors in the twist-averaged energy by replacing $\bar{K}^{\rm HF}$ with the infinite-system HF kinetic energy per particle $K_\infty^{\rm HF}$, which is straightforward to calculate by pen and paper. One should not replace $\bar{X}^{\rm HF}$ with its infinite-system value, however, as there is also a long-range finite-size error in the exchange energy.

If the number of re-equilibration steps between twists is sufficiently large and/or the number of steps at each twist is sufficiently large, we can assume the mean energy at each twist to be statistically independent. If we also assume the error in the QMC energy at each twist to be normally distributed about the optimal fitting function with equal standard deviations, we can use the covariance matrix from the fit together with the assumption that the χ^2 function is equal to the number of degrees of freedom (the number of twists minus the number of fitting parameters) to estimate the error in $\bar{E}$. If there is a non-negligible error bar associated with the estimates of $\bar{K}^{\rm HF}$ and $\bar{X}^{\rm HF}$ then it can incorporated into the fitted parameters by Gaussian propagation of errors.

If the re-equilibration period after changes to twist is less than the longest correlation period then the approach taken by `mcta_post_process` is preferred, as it allows for reblocking analysis of the corrected energy data. However, one should aim to avoid this situation, as it can also lead to bias. Generally `mcta_post_process` and `twistanalysis_heg_internal` should give very similar answers.

28.3.3 Real systems

For real systems one can perform the DFT/HF wave-function generation calculations at arbitrary grids of **k** points, then use the wave-function converters in the usual fashion to generate CASINO input files, then read in and use those wave functions in CASINO, provided that **complex_wf** is set to `T`. This functionality is not yet available for Gaussian orbitals, however.

We provide a couple of scripts which quasi-automate this procedure: `twistav_pwscf` for use with the PWSCF plane-wave DFT code, and `twistav_castep` for use in conjunction with the CASTEP plane-wave DFT code, the `castep2casino` utility and the CASINO program. Note that these scripts just produce the correct data files for you to analyse manually; they do not analyse the results themselves. How you want to analyse the data is up to you. Do you (i) just average the QMC energies and include the random error from twist averaging with the statistical error or (ii) do you use the DFT energies to correct the QMC energies before averaging or (iii) some other way... As an example, in Ref. [94] option (ii) was chosen.

It is not generally necessary to re-optimize the Jastrow factor and backflow functions for the different twists; just a single `correlation.data` file should be supplied.

28.3.4 Monte Carlo twist averaging for real systems using CASINO and CASTEP

This is automated with the `twistav_castep` utility. This script repeatedly offsets the grid of **k** vectors in the CASTEP `.cell` file and runs CASTEP, then `castep2casino`, then CASINO. The CASTEP and CASINO output files are put in directories called `twist0001`, `twist0002`, etc. The progress of the calculations is reported in a file called `STATUS`. All the necessary CASTEP and CASINO input files must be set up in the directory in which the calculation is run. The complete list of **k** vectors must be specified (note that both members of a $\pm\mathbf{k}$ pair must be given), and **complex_wf** must be `T` in the `input` file. By default, the offsets to the grid of **k** vectors are chosen randomly. If one wishes to specify a set of offsets, to compare different phases of a material for example, then the '`-twists`' flag should be used. The specified twists should be placed in a file called '`twists.data`' containing a CASTEP-style input block named `TWISTS`.

By default the script uses `runqmc` to run the CASINO calculations. If the script is to be used on a machine with a queueing system then the '`-batch`' flag should be set and the command for running CASINO should be specified with, e.g., '`-casino "runqmc --nproc=4 --walltime=1h5m"`'. The script doesn't currently let you submit the CASTEP jobs to a queue, but the CASTEP runs are relatively quick. The number of twists to use is specified using the '`-ntwist`' flag (default 12 at the time of writing). For more information on available options, type *twistav_castep -help.*

A script called `twistanalysis` is available to analyse DMC results generated by `twistav_castep`. This script is also capable of "twist-reblocking" these same results. The set of twists is divided up into blocks with size equal to a factor of the total number of twists. Then, within each of these blocks the script extrapolates the DMC results at each twist to zero time step, and fits

$$E_{\rm DMC}(t) = \langle E_{\rm DMC}\rangle + c\left[E_{\rm DFT}(t) - E_{\rm DFT}^{\rm fine}\right] \tag{327}$$

to the DMC energy $E_{\rm DMC}(t)$ at each twist t. $\langle E_{\rm DMC}\rangle$ and c are fitting parameters, with the former being the twist-averaged DMC energy. $E_{\rm DFT}(t)$ is the DFT energy from the wave-function generation calculation at a particular twist t, while $E_{\rm DFT}^{\rm fine}$ is the DFT energy with a large Monkhorst–Pack grid. Following fitting the data within each block to Eq. (327) the twist-averaged results are then averaged to give the twist-reblocked result. The results of this twist-reblocking procedure are printed to the file '`twistblocked_energy.results`', where each row contains: the number of blocks, the twist-reblocked energy, the standard error in the twist-reblocked energy, and the error in the standard error. In the case of there being one block the error printed is error resulting from fitting the data to Eq. (327). The script performs twist-reblocking for all possible divisors of the total number of twists, provided each block contains at least three twists to allow fitting of Eq. (327); it is therefore recommended to use a total of number of twists that has many divisors, e.g. 12,24,48. One can perform twist-averaging, in a single block, without using a DFT control variate (i.e., without fitting to Eq. (327)) by supplying the flag '`--unweighted`'. In this case, the twist-averaged energy is simply the mean of the energies at the different twists and the error is the standard error calculated over all of the twists.

To use `twistanalysis`:

1. Use `twistav_castep` to generate the input files for CASINO (i.e., use CASTEP and CASTEP2CASINO but not CASINO by using the `-justcastepblip` argument to `twistav_castep`).

2. In each `twistxxxx` directory, create two or more directories that contain DMC calculations at that twist with different time steps. (These directories should contain: `input`, `dmc.hist` and `dmc.status`.)

3. The user will be asked to enter the results of a DFT calculation with a fine **k**-point mesh. In all other regards, this DFT calculation should be the same as the wave-function-generation calculations. The `twistanalysis` script currently assumes that the density-mixing algorithm is used in CASTEP (no fixed-occupancy calculations).

4. In the directory containing the `twistxxxx` directories, run `twistanalysis`. The twist-averaged results will be placed in the file `twistanalysis.results`. In the ideal case of a perfect fit

(i.e., perfect correlation between the fluctuations in the DFT energy and the DMC energy as a function of twist $\mathbf{k}_\mathrm{s}$), the reduced χ^2 value of the fit would be 1. In practice it is often one or two orders of magnitude larger, depending on how small you make your error bars.

5. Beware of the following: `twistanalysis` will only run `reblock` in each DMC directory if the `dmc.hist` file has been modified more recently than the `reblock.results` file; hence if you rerun `twistanalysis` and make a different choice of energy units then you could end up with different energy units in the different `reblock.results` files and hence nonsense results when you twist average. You should therefore be careful to make the same choice of energy units throughout or, if you really have to change this, touch all of the `dmc.hist` files before rerunning `twistanalysis`. To be honest, I think the best policy in general is to stick to Hartree a.u. per unit cell throughout your calculations, and only convert to eV or kcal mol^{-1} or megatonnes of TNT or whatever at the point at which you write your paper.

The `clearup_twistav` script can be used to clear up the output from a CASTEP twist-averaging run.

28.3.5 Monte Carlo twist averaging for real systems using CASINO and PWSCF

This is automated with the `twistav_pwscf` utility. The support provided by this script is currently more advanced than its CASTEP equivalent since it uses the standard CASINO architecture system (meaning it should run automatically on any system, batch or otherwise) and the full range of run time options are available. PWSCF itself by default works with binary `bwfn.data.b1` files which obviates the need for using huge bwfn.data files as in CASTEP, and no multi-step conversions (involving e.g., `castep2casino` and blip-conversion CASINO calculations) are required.

Usage:

```
twistav_pwscf [--help --nproc_dft=I --splitqmc[=N] --startqmc=M
              --dft_only/--qmc_only --ntwist=L [<runqmc/runpwscf options>]
```

This script is used to automate the collection of twist-averaged data using CASINO and the PWSCF DFT code (part of the Quantum espresso package, available at `https://www.quantum-espresso.org`). PWSCF must be version 4.3 or later.

This script works by repeatedly calling the `runpwscf` and `runqmc` scripts which know how to run PWSCF/CASINO on any individual machine. With the exception of those listed above, almost all optional arguments to this script are the same as for `runpwscf`/`runqmc` and are passed on automatically to these subsidiary run scripts (the `--background`/`-B` option is also used by `twistav_pwscf`, and for the same purpose). Type '`runpwscf --help`' or '`runqmc --help`' to find out what these options are. The short list of optional flags specific to `twistav_pwscf` are described below.

It is assumed that PWSCF lives in `$HOME/espresso` and CASINO lives in `$HOME/CASINO`. There are override options available if this is not the case.

If you are running on a multi-user machine with an account to be charged for the calculations, you might consider aliasing `twistav_pwscf` as `alias twistav_pwcf="twistav_pwscf --user.account=CPH005mdt "` or whatever.

To run a twist-averaged calculation you should in general do something like the following:

Setup the PWSCF input ('`in.pwscf`') and the CASINO input ('`input`', etc., but no wave function file) in the same directory. For the moment we assume you have an optimized Jastrow from somewhere (the same one will be used irrespective of the twist angle).

Have the PWSCF input file set up with **calculation** `= scf`, both **nosym** and **noinv** (system section) set to `T`, **wf_collect** `= T` (control section), and **verbosity** to `high` (control section). Some required **k**-point information is not printed in output without the latter. The `in.pwscf` file must also contain a K_POINTS block written *using the 'crystal' format*, i.e., something like:

```
K_POINTS crystal
8
  0.250000   0.250000   0.250000   0.1250000
 -0.250000   0.250000   0.250000   0.1250000
  0.250000  -0.250000   0.250000   0.1250000
 -0.250000  -0.250000   0.250000   0.1250000
  0.250000   0.250000  -0.250000   0.1250000
 -0.250000   0.250000  -0.250000   0.1250000
  0.250000  -0.250000  -0.250000   0.1250000
 -0.250000  -0.250000  -0.250000   0.1250000
```

This block will be manipulated by `twistav_pwscf` and the runscripts that it calls.

In your CASINO `input` file, **complex_wf** must be `T`. Note that the VMC/DMC runs for each twist can be relatively short and need not be fully converged; the idea is that we collect enough data to achieve an acceptable error bar when the data is averaged over all twist angles. If a normal run without twist-averaging takes N moves to arrive at an acceptable error bar, then each twist angle might be run for around $N/ntwist$ moves.

The `twistav_pwscf` script will then repeatedly run PWSCF to generate '*ntwist*' `xwfn.data` files (*ntwist* default = 12, or change with optional argument `--ntwist=xx`), then it will run CASINO on each of the `xwfn.data`. The CASINO `out` files, `xwfn.data` files, `config.out` and `vmc.hist/dmc.hist` files will be renamed with an appropriate integer suffix.

The calculation can be run through `pwfn.data`, `bwfn.data` or `bwfn.data.b1` formats as specified in the `pw2casino.dat` file (see CASINO and PWSCF documentation). When PWSCF is upgraded to produce new-format `bwfn.data.bin` files, then this script will need to be changed; ask MDT to do so.

If you wish to do the (fast) DFT wave function generation calculations and the (slow) QMC calculations on different machines, for example to avoid batch queue waiting time, then use the `--dft_only` option to generate the full set of `xwfn.data` files, transfer these to the more powerful machine, then run on that using the `--qmc_only` option—see below.

The `clearup_twistav` script can be used to clear up the output from a PWSCF twist-averaging run.

Default behaviour of twistav_pwscf (on all machines):

Note: in the following *ntwist* is 12, or the value of the optional argument `--ntwist`, while `xwfn.data` refers to whatever wave function file is specified in the `pw2casino.dat` file (either `bwfn.data.b1` [default], `bwfn.data` or `pwfn.data`).

For a complete twist-averaging run, the following steps are performed in sequence:

(A) PWSCF generates *ntwist* `xwfn.data.$` files, where $ is a sequence number from 1 to *ntwist*. Each succeeding run will have a different twist.
(B) CASINO runs a relatively short VMC or VMC-DMC run on each of the `xwfn.data`.

On batch queue systems, `twistav_pwscf` will by default do *two* batch script submissions, the first—handled by the `runpwscf` script—executing step (A), and the second—handled by the `runqmc` script—executing step (B).

In principle, this wastes some unnecessary time (the time spent waiting for the QMC batch script to start) but this is unavoidable if `twistav_pwscf` uses separate `runpwscf` and `runqmc` scripts to handle the DFT `xwfn` generation and QMC calculations. This may be changed in the future if anyone thinks it's worth it.

Note that usually all calculations will be done on the number of cores requested on the command line (with the `--nproc or -p flag`), irrespective of whether they are DFT or QMC calculations. Since typically DFT calculations in fact require fewer cores, you may override this for the DFT calculations by using the `--nproc_dft` flag to `twistav_pwscf`.

Modifications to default behaviour (on all machines)

```
(1) twistav_pwscf --dft_only  : execute only step (A), generating ntwist
                                xwfn.data.$ files. Usually used if you want
                                to run DFT and QMC on different machines.

(2) twistav_pwscf --qmc_only  : execute only step (B). Usually used if you want
                                to run DFT and QMC on different machines.
```

This latter option requires that the *ntwist* `xwfn.data.$` files already exist; if they don't the script will whinge and die.

```
(3) twistav_pwscf --startqmc=M : Start the chain of QMC runs with file xwfn.data.M .
```

Modifications to default behaviour (batch machines only)

On batch machines, there is an additional complication because of the presence of walltime limits. Full twist-averaging runs might need to be split into multiple sections if all *ntwist* QMC calculations run one after another would exceed the walltime limit. The following method may be used to do this.

```
(4) twistav_pwscf --splitqmc=N : Split step B into N separate sequential batch
     script submissions, run one after another. If no value is supplied
     [--splitqmc] the run will be split into two.
```

Example: *ntwist*=13, and `twistav_pwscf --splitqmc=4` will result in four step B batch submissions with 3, 3, 3, 4 twists. Recall that the QMC calculations for each twist angle are considerable shorter than normal, and the entire twist-averaging run should not take much longer than a standard calculation done with a constant twist angle.

Note there is no facility for splitting step A into sections, i.e., all DFT wave function generation runs will always be run in a single batch script submission. This is because we assume the DFT runs are fast and you have adequate job time limits. If this is not the case then simply do multiple sets of `twistav_pwscf` runs.

The use of ensemble jobs (which are anyway supported only on some machines) to run the multiple short QMC runs simultaneously is currently not supported; it could in principle be implemented. However, if you are choosing to run *ntwist* calculations simultaneously on *ntwist*$\times M$ cores rather than running sequentially on M cores, then why not just run the sequential run on *ntwist* $\times M$ cores using fewer moves (that way you avoid multiplying the queueing time—which can be weeks on overcrowded machines—by *ntwist*.

The `twistav_pwscf` script needs to run in the background throughout the sequence of calculations, so make sure it stays alive. Logging out is inadvisable on some machines.

Note finally that we expect to generalize the `twistanalysis` utility to PWSCF very soon; if this is urgent, tell MDT.

There is a set of input files demonstrating how to setup twist-averaging calculations with PWSCF in the `examples/crystal/twistav/PWSCF` directory of the main CASINO distribution. If attempting to do twist-averaging on a complicated batch machine, users are advised to first use this set of files to verify that everything works before concentrating on their own calculations.

29 Finite-size correction to the kinetic energy

29.1 Finite-size correction due to long-ranged correlations

Consider a periodic system of N particles. Suppose there are N_s species present, each with mass m_α and charge q_α, and let N_α be the number of particles of type α. Let $\mathbf{r}_{i\alpha}$ be the position vector of the ith particle of type α. Let the simulation supercell have volume Ω.

Suppose the ground-state trial wave function Ψ can be written as the product of a part involving long-ranged two-body correlations $u_{\alpha\beta}$ and a part consisting of everything else, Ψ_s [19, 15]:

$$\Psi(\mathbf{R}) = \Psi_\mathrm{s}(\mathbf{R}) \exp\left(\sum_{\alpha=1}^{N_\mathrm{s}} \sum_{\beta=\alpha+1}^{N_\mathrm{s}} \sum_{i=1}^{N_\alpha} \sum_{j=1}^{N_\beta} u_{\alpha\beta}(\mathbf{r}_{i\alpha} - \mathbf{r}_{j\beta}) + \sum_{\alpha=1}^{N_\mathrm{s}} \sum_{i=1}^{N_\alpha - 1} \sum_{j=i+1}^{N_\alpha} u_{\alpha\alpha}(\mathbf{r}_{i\alpha} - \mathbf{r}_{j\alpha}) \right), \tag{328}$$

where $u_{\alpha\beta}(\mathbf{r})$ has the periodicity of the simulation cell and $u_{\alpha\beta}(\mathbf{r}) = u_{\alpha\beta}(-\mathbf{r})$. Note that, throughout this section, we use u to denote the two-body Jastrow factor; in fact this is the sum of the u and p terms in CASINO's Jastrow factor.

Let the Fourier transform of $u_{\alpha\beta}$ be

$$\tilde{u}_{\alpha\beta}(\mathbf{G}) = \int_\Omega u_{\alpha\beta}(\mathbf{r}) \exp(-i\mathbf{G}\cdot\mathbf{r})\, d\mathbf{r}, \tag{329}$$

where the $\{\mathbf{G}\}$ are the simulation-cell reciprocal lattice vectors. Let

$$\tilde{\rho}_\alpha(\mathbf{G};\mathbf{R}) = \sum_{i=1}^{N_\alpha} \exp(-i\mathbf{G}\cdot\mathbf{r}_{i\alpha}) \tag{330}$$

be the Fourier transform of the density operator $\rho_\alpha(\mathbf{r};\mathbf{R}) = \sum_{i=1}^{N_\alpha} \delta(\mathbf{r}-\mathbf{r}_{i\alpha})$. Then

$$\begin{aligned}
\Psi(\mathbf{R}) &= \Psi_{\rm s}(\mathbf{R}) \exp\left(\frac{1}{\Omega} \sum_{\alpha=1}^{N_{\rm s}-1} \sum_{\beta=\alpha+1}^{N_{\rm s}} \sum_{\mathbf{G}} \tilde{u}_{\alpha\beta}(\mathbf{G}) \tilde{\rho}^*_\alpha(\mathbf{G};\mathbf{R}) \tilde{\rho}_\beta(\mathbf{G};\mathbf{R}) \right. \\
&\qquad \left. + \frac{1}{2\Omega} \sum_{\alpha=1}^{N_{\rm s}} \sum_{i=1}^{N_\alpha} \sum_{\mathbf{G}} \tilde{u}_{\alpha\alpha} \tilde{\rho}^*_\alpha(\mathbf{G};\mathbf{R}) \tilde{\rho}_\beta(\mathbf{G};\mathbf{R}) - \frac{1}{2\Omega} \sum_{\alpha=1}^{N_{\rm s}} N_\alpha \sum_{\mathbf{G}} \tilde{u}_{\alpha\alpha}(\mathbf{G}) \right) \\
&= \Psi_{\rm s}(\mathbf{R}) \exp\left(\frac{1}{2\Omega} \sum_{\alpha=1}^{N_{\rm s}} \sum_{\beta=1}^{N_{\rm s}} \sum_{\mathbf{G}\neq\mathbf{0}} \tilde{u}_{\alpha\beta}(\mathbf{G}) \tilde{\rho}^*_\alpha(\mathbf{G};\mathbf{R}) \tilde{\rho}_\beta(\mathbf{G};\mathbf{R}) + K \right), \qquad (331)
\end{aligned}$$

where K is independent of $\mathbf{R}$.

If we assume that only electrons are present, the 'TI' kinetic-energy estimator [11] may be written as

$$T(\mathbf{R}) = \frac{-1}{4} \nabla^2 \log(\Psi) = T_{\rm s}(\mathbf{R}) - \frac{1}{8\Omega} \sum_{\alpha=1}^{N_{\rm s}} \sum_{\beta=1}^{N_{\rm s}} \sum_{\mathbf{G}\neq\mathbf{0}} \tilde{u}_{\alpha\beta}(\mathbf{G}) \nabla^2 \left[\tilde{\rho}^*_\alpha(\mathbf{G};\mathbf{R}) \tilde{\rho}_\beta(\mathbf{G};\mathbf{R}) \right], \qquad (332)$$

where $T_{\rm s} = -\nabla^2 \log(\Psi_{\rm s}(\mathbf{R}))/4$ [15].

It can easily be shown that

$$\nabla^2 \left[\tilde{\rho}^*_\alpha(\mathbf{G};\mathbf{R}) \tilde{\rho}_\beta(\mathbf{G};\mathbf{R}) \right] = -2|\mathbf{G}|^2 \left[\tilde{\rho}^*_\alpha(\mathbf{G};\mathbf{R}) \tilde{\rho}_\beta(\mathbf{G};\mathbf{R}) - N_\alpha \delta_{\alpha\beta} \right]. \qquad (333)$$

Hence the kinetic energy is

$$\langle T(\mathbf{R}) \rangle = \langle T_{\rm s} \rangle + \frac{1}{4\Omega} \sum_{\mathbf{G}\neq\mathbf{0}} |\mathbf{G}|^2 \left(\sum_{\alpha=1}^{N_{\rm s}} \sum_{\beta=1}^{N_{\rm s}} \tilde{u}_{\alpha\beta}(\mathbf{G}) \left\langle \tilde{\rho}^*_\alpha(\mathbf{G};\mathbf{R}) \tilde{\rho}_\beta(\mathbf{G};\mathbf{R}) \right\rangle - \sum_{\alpha=1}^{N_{\rm s}} N_\alpha \tilde{u}_{\alpha\alpha}(\mathbf{G}) \right). \qquad (334)$$

The Fourier transform of the translationally averaged structure factor is

$$\tilde{S}_{\alpha\beta}(\mathbf{k}) = \frac{1}{N} \left(\left\langle \tilde{\rho}_\alpha(\mathbf{k};\mathbf{R}) \tilde{\rho}^*_\beta(\mathbf{k};\mathbf{R}) \right\rangle - \left\langle \tilde{\rho}_\alpha(\mathbf{k};\mathbf{R}) \right\rangle \left\langle \tilde{\rho}^*_\beta(\mathbf{k};\mathbf{R}) \right\rangle \right). \qquad (335)$$

The charge density has the periodicity of the primitive lattice and therefore $\langle \tilde{\rho}_\alpha(\mathbf{k};\mathbf{R}) \rangle$ is only nonzero for $\mathbf{G}$ vectors of the primitive lattice. In particular, the second term in the Fourier-transformed structure factor is zero for small $\mathbf{k}$, which is the regime of relevance here. Hence

$$\langle T \rangle = \langle T_{\rm s} \rangle + \frac{N}{4\Omega} \sum_{\mathbf{G}\neq\mathbf{0}} |\mathbf{G}|^2 \sum_{\alpha=1}^{N_{\rm s}} \sum_{\beta=1}^{N_{\rm s}} \tilde{u}_{\alpha\beta}(\mathbf{G}) \tilde{S}^*_{\alpha\beta}(\mathbf{G}) - \frac{1}{4\Omega} \sum_{\mathbf{G}\neq\mathbf{0}} |\mathbf{G}|^2 \sum_{\alpha=1}^{N_{\rm s}} N_\alpha \tilde{u}_{\alpha\alpha}(\mathbf{G}). \qquad (336)$$

In the infinite-system limit, the sums over $\mathbf{G}$ in Eq. (336) should be replaced by integrals over $\mathbf{k}$ [19]. The leading-order finite-size corrections are due to the differences between these integrals and sums. The relevant theory can be found in Ref. [15]. The RPA suggests that we can approximate $\bar{u}_{\alpha\alpha}(k)$ by $-4\pi(A_\alpha/k^2 + B_\alpha/k)$ at small k in 3D, where A_α and B_α are constants. In 2D we can approximate $\bar{u}_{\alpha\alpha}(k)$ by $-a_\alpha/k^{3/2} - b_\alpha/k$. In each case the constants are determined by fitting the model of $\bar{u}_{\alpha\alpha}$ to its values at the first few nonzero stars of $\mathbf{G}_{\rm s}$ vectors.

We evaluate the finite-size correction to the kinetic energy in 3D as

$$\Delta T = \sum_{\alpha=1}^{N_{\rm s}} \left[\frac{N_\alpha \pi A_\alpha}{\Omega} + \frac{C_{\rm 3D} N_\alpha B_\alpha}{\Omega^{4/3}} \right], \qquad (337)$$

where

$$C_{\rm 3D} = \frac{\Omega^{4/3}}{4} \lim_{\alpha\to 0} \left[\frac{1}{\pi\alpha^2} - \frac{4\pi}{\Omega} \sum_{\mathbf{G}_{\rm s}\neq\mathbf{0}} G_{\rm s} \exp(-\alpha G_{\rm s}^2) \right] \qquad (338)$$

is a lattice-specific constant. This constant is evaluated by numerically by CASINO. We use three different values of α and extrapolate the term in square brackets quadratically to $\alpha = 0$. For a given value of α we evaluate the sum over $\mathbf{G}_{\rm s}$ explicitly inside a sphere of radius $Q_{\rm c}$, and approximate the sum by an integral outside Q_c. Q_c is chosen to be sufficiently large that the summand is very small at this radius. In 2D, we evaluate the finite-size correction as

$$\Delta T = \sum_{\alpha=1}^{N_\mathrm{s}} \frac{C_\mathrm{2D} N_\alpha a_\alpha}{4\pi P^{5/4}}, \tag{339}$$

where P is the simulation-cell area and

$$C_\mathrm{2D} = \frac{P^{5/4}}{2} \lim_{\alpha\to 0} \left[\frac{\Gamma(5/4)}{2\alpha^{5/4}} - \frac{2\pi}{P} \sum_{\mathbf{G}_\mathrm{s}\neq\mathbf{0}} \sqrt{G_\mathrm{s}} \exp(-\alpha G_\mathrm{s}^2) \right], \tag{340}$$

where Γ is the Gamma function.

The leading term is $\mathcal{O}(1)$ in 3D, so the error in the kinetic energy per electron falls off as $\mathcal{O}(N^{-1})$. The leading term is $\mathcal{O}(N^{-1/4})$ in 2D, so the error in the kinetic energy per electron falls off as $\mathcal{O}(N^{-5/4})$.

29.2 Fourier transformation of CASINO's two-body Jastrow factor

29.2.1 The two-body Jastrow factor

CASINO's two-body Jastrow factor consists of two terms: u and p [60]. For any given spin-type, u is of the form

$$u(\mathbf{r}) = (r - L_u)^C \Theta(L_u - r) \sum_{l=0}^{N_u} \alpha_l r^l, \tag{341}$$

where L_u, C and $\{\alpha\}$ are parameters, Θ is the Heaviside function, and r is the magnitude of electron–electron distance evaluated within the minimum-image convention. (Throughout this section we omit the spin indices α and β.)

The Fourier transformation of the two-body Jastrow factor is just the sum of the Fourier transforms of u and p. Note that the Fourier transform of u is spherically symmetric, whereas spherical averaging may have to be performed for the Fourier transform of p.

29.2.2 Fourier transformation of p

The p term in the Jastrow factor is

$$p(\mathbf{r}) = \sum_A a_A \sum_{\mathbf{G}_A^+} \cos(\mathbf{G}_A\cdot\mathbf{r}) = \sum_A \sum_{\mathbf{G}_A} \frac{1}{2} a_A \exp(i\mathbf{G}_A\cdot\mathbf{r}), \tag{342}$$

where A denotes a set of symmetry-equivalent simulation-cell $\mathbf{G}$ vectors and '+' means that, if $\mathbf{G}$ is included in the sum, $-\mathbf{G}$ is excluded [60]. So the Fourier coefficient for $\mathbf{G}_A$ is $\Omega a_A/2$ in 3D, $A a_A/2$ in 2D and $L a_A/2$ in 1D.

29.2.3 Fourier transformation of u in 3D

Suppose $\mathbf{k}\neq\mathbf{0}$. Then

$$\begin{aligned} \tilde{u}(\mathbf{k}) &= \int_\Omega u(\mathbf{r}) \exp(-i\mathbf{k}\cdot\mathbf{r})\, d\mathbf{r} \\ &= \frac{4\pi}{k} \int_0^{L_u} r u(r) \sin(kr)\, dr \\ &= \frac{4\pi}{k} \sum_{l=0}^{N_u} \alpha_l \sum_{m=0}^{C} \binom{C}{m} (-L_u)^{C-m} \int_0^{L_u} r^{m+l+1} \sin(kr)\, dr. \end{aligned} \tag{343}$$

Let $I_n(k) = \int_0^{L_u} r^n \sin(kr)\, dr$. Then $I_0 = [1 - \cos(kL_u)]/k$ and $I_1 = -L_u \cos(kL_u)/k + \sin(kL_u)/k^2$, and, for $n \geq 2$,

$$I_n(k) = \frac{1}{k} \left[\frac{n}{k} \left(L_u^{n-1} \sin(kL_u) - (n-1) I_{n-2}(k) \right) - L_u^n \cos(kL_u) \right]. \tag{344}$$

Hence we can rapidly evaluate $I_n(k)$ for all n required (that is, up to $n = N_u + C + 1$).

For $\mathbf{k} = \mathbf{0}$ we have

$$\tilde{u}(\mathbf{0}) = 4\pi \sum_{l=0}^{N_u} \alpha_l \sum_{m=0}^{C} \binom{C}{m} (-L_u)^{C-m} \frac{L_u^{l+m+3}}{l+m+3}. \tag{345}$$

29.2.4 Fourier transformation of u in 2D

An analytic expression for $\tilde{u}(\mathbf{k})$ is not available in two dimensions. We therefore evaluate $\tilde{u}(\mathbf{k})$ numerically using a fast Fourier transform.

29.2.5 Fourier transformation of u in 1D

$$\begin{aligned}\tilde{u}(k) &= \int_{-L/2}^{L/2} u(|x|)\exp(-ikx)\,dx \\ &= 2\sum_{l=0}^{N_u} \alpha_l \int_0^{L_u} (x-L_u)^C x^l \cos(kx)\,dx \\ &= 2\sum_{l=0}^{N_u} \alpha_l \sum_{n=0}^{C} \binom{C}{n} (-L_u)^{C-n} J_{n+l}(k),\end{aligned} \tag{346}$$

where L is the length of the simulation cell and $J_n = \int_0^{L_u} x^n \cos(kx)\,dx$. Suppose $k \neq 0$. Then $J_0(k) = \sin(kL_u)/k$, $J_1(k) = L_u \sin(kL_u)/k + [\cos(kL_u) - 1]/k^2$ and, for $n \geq 2$

$$J_n(k) = \frac{1}{k}\left[\frac{n}{k}\left(L_u^{n-1}\cos(kL_u) - (n-1)J_{n-2}(k)\right) + L_u^n \sin(kL_u)\right]. \tag{347}$$

Hence we can rapidly evaluate $J_n(k)$ for all n required (from $n = 0$ to $n = N_u + C$). If $k = 0$ then $J_n = L_u^{n+1}/(n+1)$.

29.3 Fitting form for the long-ranged two-body Jastrow factor (3D)

29.3.1 The 'RPA–Kato' Jastrow factor

Consider the infinite-system 'RPA–Kato' two-body Jastrow factor [11] for pairs of particles of type α, which satisfies the Kato cusp condition and has the long-ranged $1/r$ decay predicted by the RPA [95],

$$u_{\alpha\alpha}(\mathbf{r}) = -\frac{A_\alpha}{r}\left[1 - \exp(-r/F_\alpha)\right], \tag{348}$$

where A_α is a free parameter and $F_\alpha^2 = c_\alpha A_\alpha$ is determined by the cusp conditions, where $c_\alpha = 2/(q_\alpha^2 m_\alpha)$. The Fourier transformation of this two-body Jastrow factor is

$$\tilde{u}_{\alpha\alpha}(\mathbf{k}) = -4\pi A_\alpha \left(\frac{1}{k^2} - \frac{1}{k^2 + 1/(c_\alpha A_\alpha)}\right). \tag{349}$$

Note that this has the k^{-2} divergence predicted by the RPA [95]. If the value of $\tilde{u}_{\alpha\alpha}(\mathbf{k})$ is known at a single point $\mathbf{k}$ then the parameter A_α may be evaluated as

$$A_\alpha = \frac{-k^2 \tilde{u}_{\alpha\alpha}(\mathbf{k})}{4\pi + c_\alpha k^4 \tilde{u}_{\alpha\alpha}(\mathbf{k})}. \tag{350}$$

The missing contribution to the infinite-system kinetic energy in Eq. (336) is approximately

$$\Delta T \approx -\sum_{\alpha=1} N_{\rm s} \frac{N_\alpha}{4(2\pi)^3 m_\alpha} \int_0^Q 4\pi k^2 \times k^2 \tilde{u}_{\alpha\alpha}(k)\,dk, \tag{351}$$

where $Q = (6\pi^2/\Omega)^{1/3}$ is the radius of the sphere with volume $(2\pi)^3/\Omega$.

If one inserts the RPA–Kato form for $\tilde{u}_{\alpha\alpha}$ into Eq. (351), noting that it is already spherically symmetric, then one obtains

$$\Delta T = \sum_{\alpha=1}^{N_s} \frac{N_\alpha}{2\pi m_\alpha} \left(\frac{Q}{c_\alpha} - \frac{\tan^{-1}\left(\sqrt{c_\alpha A_\alpha} Q\right)}{c_\alpha \sqrt{c_\alpha A_\alpha}} \right). \tag{352}$$

Making use of the Taylor expansion of $\tan^{-1}$, it is found that the leading-order correction to the kinetic energy is

$$\Delta T = \sum_{\alpha=1}^{N_s} \frac{\pi N_\alpha A_\alpha}{\Omega m_\alpha} + \mathcal{O}(N^{-2/3}), \tag{353}$$

where the first term is independent of N, so the error in the kinetic energy per particle falls off as $\mathcal{O}(N^{-1})$.

29.3.2 Application to the HEG

The Jastrow factor of Eq. (348) for a HEG of density parameter r_s has $A_\alpha = 1/\omega_p$ where $\omega_p = \sqrt{3/r_s^3}$ is the plasma frequency [11]. Note that $\Omega = 4\pi r_s^3 N/3 = 4\pi N/\omega_p^2$, and $m_\alpha = 1$. Hence the leading-order correction to the kinetic energy is $\Delta T = \omega_p/4$, as found by Chiesa *et al.* [19].

29.4 Applying the correction scheme in practice

The steps carried out by CASINO in order to calculate the kinetic-energy correction for a 3D system are as follows:

1. Calculate the Fourier transformation of $u_{\alpha\alpha}(r) + p_{\alpha\alpha}(\mathbf{r})$ in the Jastrow factor for each spin α and perform spherical averaging over $\mathbf{G}$ vectors of equal length.
2. For each α, determine the parameters $A_{\alpha\alpha}$ and $B_{\alpha\alpha}$ by a least-squares fit to the values of $\tilde{u}_{\alpha\alpha} + \tilde{p}_{\alpha\alpha}$ at the first few nonzero stars of $\mathbf{G}$ vectors.
3. Use Eq. (337) to calculate the kinetic-energy correction.
4. As a check, calculate $A_{\alpha\alpha}$ in Eq. (352) using the values of $\tilde{u}_{\alpha\alpha} + \tilde{p}_{\alpha\alpha}$ at the smallest nonzero star of $\mathbf{G}$ vectors [Eq. (350)]. Then evaluate the kinetic-energy correction using Eq. (352). The user can verify that the two estimates of the kinetic-energy are in reasonable agreement.

Similar steps are carried out for a 2D system is available in this case.

In order to calculate the finite-size correction to the kinetic energy, the user should set the **finite_size_corr** flag to `T`. The finite-size correction will only be calculated if the `correlation.data` file contains a nonempty Jastrow factor with either p or u terms. The finite-size correction is displayed near the top of the `out` file.

Note that it is usually essential to use p terms in the Jastrow factor when calculating the kinetic-energy correction. Even if the p term only gives a small decrease in the total energy, it is needed in order to get the shape of the long-ranged two-body Jastrow factor correct. To generated a blank p term to paste into the Jastrow factor in `correlation.data`, the `make_p_stars` utility can be used.

30 Finite-size correction to the interaction energy

An alternative to using the MPC interaction is to calculate corrections to the XC energy from an accumulated structure factor (essentially as described in Ref. [19]). Accumulation of the structure factor is automatically activated if **finite_size_corr** is set to `T`, and the finite-size correction to the interaction energy is written out at the end of the `out` file.

31 Electron–hole systems

CASINO has the ability to include positively charged particles of variable mass (holes) in the simulation in addition to electrons. Currently these may only be used in electron–hole phases without an external potential, but the code needs only a few trivial changes for these things to be able to wander around inside real crystals (useful for studying positron problems—contact Mike Towler if you want this to be implemented).

In this section the changes required to the CASINO code and to the basic equations in the presence of holes are discussed. These largely stem from the possibility of having a variable mass ratio between the positively and negatively charged particles. The basic differences are:

- The diffusion Green's function, Eq. (46), becomes,
$$\begin{aligned} G_{\mathrm{D}}(\mathbf{R}\leftarrow\mathbf{R}',\tau) &= \frac{1}{(4\pi D_e\tau)^{3N_e/2}}\exp\left(-\frac{(\mathbf{R}_e-\mathbf{R}'_e-2\tau D_e\mathbf{V}_e(\mathbf{R}'_e))^2}{4D_e\tau}\right) \\ &\times\frac{1}{(4\pi D_h\tau)^{3N_h/2}}\exp\left(-\frac{(\mathbf{R}_h-\mathbf{R}'_h-2\tau D_h\mathbf{V}_h(\mathbf{R}'_h))^2}{4D_h\tau}\right), \end{aligned} \tag{354}$$
where e and h denote electron and hole quantities, N_e and N_h are the numbers of electrons and holes, the diffusion constants are defined as $D_e = 1/(2m_e)$ and $D_h = 1/(2m_h)$, where m_e and m_h are the electron and hole masses in atomic units (i.e., in units of the mass of the electron).

- When particle i is moved, Eq. (50) becomes,
$$\mathbf{r}_i = \mathbf{r}'_i + \chi + 2D_i\tau\mathbf{v}_i(\mathbf{R}'), \tag{355}$$
where χ is a 3D vector of normally distributed numbers with variance $2D_i\tau$ and zero mean.

- The probability of accepting this move, Eq. (54) is then,
$$p_i \simeq \min\left\{1, \exp\left([\mathbf{r}'_i-\mathbf{r}_i+\tau D_i\left(\mathbf{v}_i(\mathbf{R}')-\mathbf{v}_i(\mathbf{R})\right)]\cdot[\mathbf{v}_i(\mathbf{R}')+\mathbf{v}_i(\mathbf{R})]\right)\frac{\Psi(\mathbf{R})^2}{\Psi(\mathbf{R}')^2}\right\}. \tag{356}$$

- The effective time step, Eq. (62), is given by,
$$\tau_{\mathrm{eff}}(\alpha, m) = \tau\frac{\sum_i m_i p_i \Delta r^2_{\mathrm{d},i}}{\sum_i m_i \Delta r^2_{\mathrm{d},i}} \tag{357}$$

- The drift vector limiting, Eq. (63), takes the form,
$$\tilde{\mathbf{v}}_i = \frac{-1+\sqrt{1+4D_i a|\mathbf{v}_i|^2\tau}}{2a|\mathbf{v}_i|^2 D_i\tau}\mathbf{v}_i\ . \tag{358}$$

- Separate Jastrow factors must be defined for the electron–electron, hole–hole and electron–hole interactions. The general form of the cusp condition for Coulomb interactions is,
$$\left.\frac{1}{\Psi}\frac{d\Psi}{dr}\right|_{r=0} = \frac{2q_iq_j\mu_{ij}}{d\pm1}, \tag{359}$$
where q_i and q_j are the charges in units of the charge of the electron, $\mu_{ij} = m_im_j/(m_i+m_j)$ is the reduced mass and d is the dimensionality. The minus sign is used for distinguishable particles (e.g., anti-parallel-spin electrons or electron and holes) and the plus sign for indistinguishable particles (e.g., parallel-spin electrons).

- Backflow transformations for the pairing wave-function have to be carefully rederived.

- The kinetic energy term in the local energy is modified to include the mass,
$$K = \sum_{i=1}^{N} K_i = \sum_{i=1}^{N} -\frac{1}{2m_i}\Psi(\mathbf{R})^{-1}\nabla_i^2\Psi(\mathbf{R})\ . \tag{360}$$
Similarly,
$$T_i = -\frac{1}{4m_i}\nabla_i^2\left(\ln|\Psi|\right) = -\frac{1}{4m_i}\frac{\nabla_i^2\Psi}{\Psi} + \frac{1}{4m_i}\left(\frac{\nabla_i\Psi}{\Psi}\right)^2, \tag{361}$$
and for the drift vector $\mathbf{F}_i$,
$$\mathbf{F}_i = \frac{1}{\sqrt{2m_i}}\nabla_i\left(\ln|\Psi|\right) = \frac{1}{\sqrt{2m_i}}\frac{\nabla_i\Psi}{\Psi}\ . \tag{362}$$

32 Derivation of the short-range two-body behaviour in dipolar gases

Suppose a gas of particles moving in 2D interact via the pairwise dipolar potential d^2/r^3, where d^2 is the dipole strength. Consider a coalescing pair of particles of reduced mass μ and relative position $\mathbf{r}$. The contribution to the Hamiltonian from the relative motion of the coalescing pair (with all other coordinates fixed) is

$$\hat{H}' = -\frac{1}{2\mu}\nabla^2 + \frac{d^2}{r^3}. \tag{363}$$

Let the wave function be $\Psi(\mathbf{r};\mathbf{R}) = \mathcal{U}(r)S(\mathbf{R})$, where $\mathcal{U}(r) = e^{u(r)}$ is the two-body Jastrow factor for the coalescing pair and S is the rest of the many-body wave function, which is smooth at the coalescence point (smooth and antisymmetric for same-spin particles). The contribution to the local energy from the coalescing pair is

$$E'_{\mathrm{L}} = \frac{\hat{H}'\Psi}{\Psi} = -\frac{1}{2\mu}\left(\frac{\nabla^2\mathcal{U}}{\mathcal{U}} + 2\frac{\nabla\mathcal{U}}{\mathcal{U}}\cdot\frac{\nabla S}{S} + \frac{\nabla^2 S}{S}\right) + \frac{d^2}{r^3}. \tag{364}$$

We require that $\mathcal{U}(r)$ be such that the leading-order divergent contributions to the local energy are cancelled. For antiparallel spins, $(\nabla S)/S$ is $\mathbf{R}$-dependent and $\mathcal{O}(r^0)$. Hence we cannot remove the $\nabla\mathcal{U}\cdot\nabla S/(\mathcal{U}S)$ term, but we can afterwards verify that it does not affect the leading-order divergences. So, for antiparallel spins in 2D, we require that

$$-\frac{1}{2\mu\mathcal{U}}\left(\frac{d^2\mathcal{U}}{dr^2} + \frac{1}{r}\frac{d\mathcal{U}}{dr}\right) + \frac{d^2}{r^3} = 0. \tag{365}$$

Let $a_{d2} = \sqrt{8d^2\mu}$, and let $x = a_{d2}/\sqrt{r}$. Then

$$x^2\frac{d^2\mathcal{U}}{dx^2} + x\frac{d\mathcal{U}}{dx} - x^2\mathcal{U} = 0. \tag{366}$$

This is the zeroth-order modified Bessel equation, with exponentially decaying solution

$$\mathcal{U} = K_0(x) = K_0(a_{d2}/\sqrt{r}) \approx \exp(-a_{d2}/\sqrt{r}), \tag{367}$$

where in the last approximate equation we have noted the leading-order behaviour at small separations r. Hence the $\nabla\mathcal{U}\cdot\nabla S/(\mathcal{U}S)$ term in the local energy [Eq. (364)] contributes an $\mathcal{O}(r^{-3/2})$ divergence. This is a considerable improvement on the $\mathcal{O}(r^{-3})$ divergence due to the dipolar interaction. Furthermore, upon spherical averaging the $\nabla\mathcal{U}\cdot\nabla S/(\mathcal{U}S)$ divergence disappears, so that for a typical configuration the remaining divergence has only a small prefactor.

For same-spin fermions, $S = ax + by + \mathcal{O}(r^3)$ for some a and b. Hence $\nabla S = a\mathbf{e}_x + b\mathbf{e}_y + \mathcal{O}(r^2)$ and $\nabla^2 S = \mathcal{O}(r)$. Also $\nabla\mathcal{U} = r^{-1}(d\mathcal{U}/dr)\mathbf{r}$. So

$$2\frac{\nabla\mathcal{U}(r)}{\mathcal{U}}\cdot\frac{\nabla S}{S} = \frac{2}{\mathcal{U}r}\frac{d\mathcal{U}}{dr}[1+\mathcal{O}(r^2)]. \tag{368}$$

The beyond-leading-order term here depends on all the particle positions and hence cannot easily be removed. To remove leading-order divergences in the local energy [Eq. (364)] we require that

$$-\frac{1}{2\mu\mathcal{U}}\left(\frac{d^2\mathcal{U}}{dr^2} + \frac{3}{r}\frac{d\mathcal{U}}{dr}\right) + \frac{d^2}{r^3} = 0. \tag{369}$$

Let $\mathcal{U} = r^{-1}\mathcal{W}$ and $x = a_{d2}/\sqrt{r}$. Then

$$x^2\frac{d^2\mathcal{W}}{dx^2} + x\frac{d\mathcal{W}}{dx} - (x^2+2^2)\mathcal{W} = 0. \tag{370}$$

This is the second-order modified Bessel equation, with exponentially decaying solution $\mathcal{W}(x) = K_2(x)$. Hence

$$\mathcal{U}(r) = r^{-1}K_2(a_{d2}/\sqrt{r}) \approx \exp(-a_{d2}/\sqrt{r}), \tag{371}$$

where in the last approximate equation we have noted the leading-order behaviour at small separations r. So the $\mathbf{R}$-dependent neglected term in Eq. (368) leads to an $\mathcal{O}(r^{-1/2})$ divergence in the local energy at same-spin coalescence points. Upon spherical averaging the remaining $\nabla\mathcal{U}\cdot\nabla S/(\mathcal{U}S)$ divergence disappears, so that for a typical configuration the divergence has only a small prefactor.

In summary, the pairwise Jastrow term that imposes the leading-order short-range behaviour in a dipolar gas is $u_{\rm A}(r)=\log[K_0(a_{d2}/\sqrt{r})]$ for opposite-spin pairs and $u_{\rm A}(r)=\log[K_2(a_{d2}/\sqrt{r})/r]$ for same-spin pairs.

To truncate the pairwise dipolar Jastrow term $u_{\rm A}(r)$ at finite range, we offset $u_{\rm A}(r)$ by the constant $-u_{\rm A}(L_u)$ and multiply the result by a smooth cutoff function $u_{\rm C}(r)=(1-r^3/L_u^3)^{C-1}\,\Theta(L_u-r)$, where L_u is the cutoff length of the usual polynomial Jastrow u term [see Eq. (224)] and Θ is the Heaviside function. The fact that the cutoff function deviates from 1 by $\mathcal{O}(r^3)$ at short range ensures that the cutoff function does not combine with the leading-order $r^{-1/2}$ behaviour of $u_{\rm A}$ to introduce new divergent terms into the local kinetic energy. The truncated Jastrow term is C times differentiable at $r=L_u$.

If a Lennard-Jones potential $c_{12}r^{-12}$ is used to regularize the tilted dipolar interaction then a two-body Jastrow term $u_{\rm B}(r)=-(\sqrt{2c_{12}\mu}/5)r^{-5}$, which cancels the leading-order divergence in the potential energy, is automatically included in the Jastrow factor. This term is smoothly truncated by multiplication by $u_{\rm C}(r)$.

33 Mahan wave function module

The trial wave functions used in CASINO consist of a product of two terms: the Jastrow factor, and a term controlled by the keyword **psi_s** in the input file. In general, both components can contain optimizable parameters. The Jastrow factor is symmetric with respect to particle interchange (in fact, it is always positive), often accounts for the cusp conditions (see Sec. 22) and usually recovers the largest part of the correlation energy. The other component of the trial wave function, $\Psi_{\rm S}$, enforces the required symmetry properties under particle interchange, provides the part of the trial wave function acted upon by a backflow transformation and can often significantly improve the trial wave function. Importantly, only the $\Psi_{\rm S}$ component of the trial wave function can affect the nodal surface, meaning that even the DMC energy can be improved *via* optimizing this component of the trial wave function.

Several types of $\Psi_{\rm S}$ components are implemented in CASINO, including Slater determinants of plane waves and several types of pairing orbitals (see Sec. 7.4.10), geminal wave functions and a form tailored to the description of excitonic molecules called exmol. The choice of which form to use should be guided by the kind of system being studied. In this section, we describe a new form of $\Psi_{\rm S}$ suitable for studying a positive impurity (such as a hole or positron) immersed in a homogeneous electron gas.

The trial wave function takes the form

$$\Psi_{\rm T}(\mathbf{R})=e^{J(\mathbf{R})}\Psi_{\rm S}\left[\mathbf{X}(\mathbf{R})\right]\ , \tag{372}$$

where $\mathbf{R}$ is the set of particle coordinates, $e^{J(\mathbf{R})}$ is a Jastrow factor that describes correlations between the particles, $\mathbf{X}(\mathbf{R})$ is the set of backflow-transformed coordinates, and $\Psi_{\rm S}$, the 'Mahan' part of the wave function, is a product of Slater determinants containing orbitals which pair each of the electrons with the hole,

$$\Psi_{\rm S}(\mathbf{R})=\det\left[\phi_i(\mathbf{r}_j^{\uparrow}-\mathbf{r}_h)\right]\det\left[\phi_i(\mathbf{r}_j^{\downarrow}-\mathbf{r}_h)\right]\ , \tag{373}$$

where $\mathbf{r}_j^{\sigma}$ is the position vector of the jth electron of spin σ and $\mathbf{r}_h$ is the position vector of the hole. Backflow significantly improves the quality of this trial wave function form, but if backflow is not being used, simply consider an identity transformation $\mathbf{X}(\mathbf{R})=\mathbf{R}$ above.

Flexible pairing orbitals whose parameters are optimized within VMC are used in the determinants.

$$\phi_i(\mathbf{r})=\exp\left[u_{G_i}(r)\right]\exp\left[i\mathbf{G}_i\cdot(r-\eta_{G_i}(r))\hat{\mathbf{r}}\right]\ , \tag{374}$$

where $\mathbf{G}_i$ is the ith shortest reciprocal lattice vector and $\hat{\mathbf{r}}$ is the unit vector in the direction of $\mathbf{r}$. The orbital-dependent electron-hole Jastrow function u_G is of the form

$$u_G(r)=\left(1-\frac{r}{L_{u,G}}\right)^C\Theta(r-L_{u,G})\sum_{l=0}^{n_u}c_{G,l}r^l\ , \tag{375}$$

where C is an integer truncation order, Θ is the Heaviside step function, n_u is the expansion order, and L_G and $c_{G,l}$ are optimizable parameters. The orbital-dependent electron-hole backflow function

η_G is of the form

$$\eta_G(r) = \left(1 - \frac{r}{L_{\eta,G}}\right)^C \Theta(r - L_{\eta,G}) \sum_{l=0}^{n_\eta} d_{G,l} r^l \ , \tag{376}$$

where n_η is the expansion order and $L_{u,G}$ and $d_{G,l}$ are optimizable parameters. The functions in Eq. (375) and (376) are smoothly truncated at optimizable distances $L_{u,G}$ and $L_{\eta,G}$ which are constrained to be less than or equal to the radius of the largest circle that can be inscribed in the Wigner-Seitz cell. To avoid redundancy, we set $u_0 = 0$, as one of the u_G functions factorizes out and can be absorbed into the global Jastrow factor. Both the electron-electron and electron-hole cusps are enforced *via* the Jastrow factor, and so the determinants of orbitals are constrained to be cuspless at electron-hole coalescence points.

The wave functions have currently been implemented for a simulation-cell Bloch vector of $\mathbf{k}_s = \mathbf{0}$ only, and have been tested in paramagnetic systems (containing equal numbers of up- and down-spin electrons) only, although it would not be difficult to go beyond either limitation.

To use this form for Ψ_S, one should set input keyword **psi_s** to `mahan` and provide a 'MAHAN' block in `correlation.data` following the outline in Sec. 7.4.11. Complex wave functions should be used (**complex_wfn** set to `T`) and to optimize the orbitals, set **opt_orbs** to `T`.

34 Relativistic corrections to energies

Relativistic corrections to the nonrelativistic Hamiltonian can be calculated to order c^{-2}, where c is the velocity of light, using first-order perturbation theory [96, 97]. This method works well for atoms of low nuclear charge Z when the relativistic corrections are small, but is unsatisfactory when Z is large.

In CASINO the perturbative relativistic corrections can be calculated for VMC or DMC[28] by setting the **relativistic** flag in the `input` file to `T`. By default the relativistic corrections are not calculated.

First we consider the mass-polarization term ε_1, which accounts for the correction due to the finite total nuclear mass to order $1/M$, where M is the total nuclear mass in a.u. (NB, this isn't actually a relativistic term as such.) If there is just one nucleus present then all finite-mass effects are accounted for to $\mathcal{O}(M^{-1})$; otherwise some finite-mass effects are neglected. The estimator used for this term is

$$\varepsilon_1 = \frac{1}{M} \sum_{i<j} \mathbf{v}_i \cdot \mathbf{v}_j, \tag{377}$$

where $\mathbf{v}_i(\mathbf{R}) = \Psi(\mathbf{R})^{-1} \nabla_i \Psi(\mathbf{R})$ is the drift vector of electron i. By default CASINO uses nuclear masses averaged over isotopes, which are listed in Table 3. If a nuclear mass for a specific isotope is required, the default setting can be overridden by the **isotope_mass** keyword in the `input` file.

The relativistic terms can be written as a sum of the mass-velocity term, Darwin terms and the retardation term. The mass-velocity term ε_2 arises from the relativistic variation of mass with velocity, and an estimator can be written as

$$\varepsilon_2 = -\frac{1}{8c^2} \sum_i \left(\nabla_i \cdot \mathbf{v}_i + |\mathbf{v}_i|^2\right)^2. \tag{378}$$

The spread of electronic charge is described by the electron–nucleus and electron–electron Darwin terms

$$\varepsilon_3 + \varepsilon_4 = \sum_I \sum_i \frac{Z_I \pi}{2c^2} \delta(\mathbf{r}_{iI}) + \sum_{i<j} \frac{\pi}{c^2} \delta(\mathbf{r}_{ij}), \tag{379}$$

where Z_I is the atomic number of ion I. Hence the expectation value, valid in both VMC and DMC, is

$$\langle \varepsilon_3 + \varepsilon_4 \rangle = \sum_I \frac{Z_I \pi}{2c^2} \rho(\mathbf{r}_I) + \frac{\pi}{c^2} \left\langle \sum_{i<j} \delta(\mathbf{r}_{ij}) \right\rangle, \tag{380}$$

[28]Note that the estimators used for the relativistic corrections are approximate in DMC, in the same way that the TI and FISQ kinetic-energy estimators are approximate. The error in the DMC relativistic corrections is of the same order as the error in the VMC corrections, and the order of the error is not reduced by the usual extrapolated-estimation procedure.

where $\rho(\mathbf{r}_I)$ is the electronic charge density at ion I. For a homogeneous, periodic system, $\left\langle \sum_{i<j} \delta(\mathbf{r}_{ij}) \right\rangle = \rho^2 \Omega g(0)/2$, where $g(r)$ is the pair-correlation function and Ω is the cell volume. It is almost certainly best to evaluate the electron–electron Darwin term for a HEG using the contact PCF. Alternatively, note that $\nabla^2(1/r) = -4\pi\delta(\mathbf{r})$. Likewise, $\nabla^2 v_E(\mathbf{r}) = -4\pi\delta(\mathbf{r})$ in a periodic system, where v_E is the Ewald interaction. Applying this to the VMC expectation value of Eq. (379), we obtain the following estimator for the Darwin terms:

$$\varepsilon_3' + \varepsilon_4' = \frac{1}{4c^2} \left[\sum_i (\nabla_i \cdot \mathbf{v}_i + 2|\mathbf{v}_i|^2) \right] \times \left[-\sum_I \sum_j \frac{Z_I}{r_{jI}} - \sum_{j<k} \frac{1}{r_{jk}} \right]. \tag{381}$$

In a periodic system, the Ewald interaction $v_E(\mathbf{r})$ should be used in place of $1/r$ in the last factor of Eq. (381). Gathering adequate statistics with this estimator can be difficult, because it diverges whenever particles approach each other or a nucleus.

For systems such as the HEG where there is a neutralizing background of charge density ρ_0, there is an associated 'Darwin constant': $\pi\rho_0 N/(2c^2)$. This is the analogue of the electron–nucleus Darwin term.

The last term, known as the retardation term ε_5, arises from the interaction between spin magnetic moments which are not mutually penetrating. An estimator for this term is

$$\varepsilon_5 = -\frac{1}{2c^2} \sum_{i<j} \left[\frac{(\mathbf{r}_{ij} \cdot \mathbf{v}_i)(\mathbf{r}_{ij} \cdot \mathbf{v}_j)}{r_{ij}^3} + \frac{\mathbf{v}_i \cdot \mathbf{v}_j}{r_{ij}} \right], \tag{382}$$

where $\mathbf{r}_{ij} = \mathbf{r}_i - \mathbf{r}_j$. For periodic systems the term is evaluated using minimum-image separations. The resulting finite-size errors are small [97].

Calculations for the beryllium atom [96] show that the total relativistic correction to the energy is approximately 0.00239 a.u., with the mass-velocity term having the greatest contribution of 0.0145 a.u., followed by the Darwin terms of 0.0119 a.u.

35 Expectation values computable by CASINO

Although the total energy is indeed the expectation value of the Hamiltonian, the term 'expectation value' in CASINO is generally intended to refer to all other observables that the program is able to calculate. In the current version of the code there are twelve of these, and in this section we will describe how to calculate and plot them, and outline some of the theoretical details.

The thirteen currently available expectation values are as follows (with the letters in brackets indicating whether they are implemented for ATOMs, MOLecules, FINite systems without fixed nuclei such as isolated biexcitons, PERiodic systems containing atoms, and HOMogeneous systems such as the HEG):

- Density - **density** (ATOM, PER)
- Spin-density - **spin_density** (ATOM, PER)
- Reciprocal-space pair-correlation function - **pair_corr** (PER, HOM)
- Spherical real-space pair-correlation function - **pair_corr_sph** (FIN, HOM)
- Structure factor - **structure_factor** (PER, HOM)
- Spherically averaged structure factor - **struc_factor_sph** (HOM)
- One-body density matrix (OBDM) - **onep_density_mat** (HOM)
- Two-body density matrix (TBDM) - **twop_density_mat** (HOM)
- Condensate fraction estimator ($\sim$ TBDM – OBDM^2) - **cond_fraction** (HOM)
- Momentum density - **mom_den** (HOM)
- Two-body momentum density (Fourier transform of TBDM) - **twop_dm_mom** (HOM)

1 H 1.00794																	2 He 4.00260
3 Li 6.941	4 Be 9.012187											5 B 10.811	6 C 12.0107	7 N 14.00674	8 O 15.9994	9 F 18.99840	10 Ne 20.1797
11 Na 22.98977	12 Mg 24.3050											13 Al 26.98154	14 Si 28.0855	15 P 30.97376	16 S 32.066	17 Cl 35.4527	18 Ar 39.948
19 K 39.0983	20 Ca 40.078	21 Sc 44.95591	22 Ti 47.867	23 V 50.9415	24 Cr 51.9961	25 Mn 54.93805	26 Fe 55.845	27 Co 58.93320	28 Ni 58.6934	29 Cu 63.546	30 Zn 65.39	31 Ga 69.723	32 Ge 72.61	33 As 74.92160	34 Se 78.96	35 Br 79.904	36 Kr 83.80
37 Rb 85.4678	38 Sr 87.62	39 Y 88.90585	40 Zr 91.224	41 Nb 92.90638	42 Mo 95.94	43 Tc 98.0	44 Ru 101.07	45 Rh 102.90550	46 Pd 106.42	47 Ag 107.8682	48 Cd 112.411	49 In 114.818	50 Sn 118.710	51 Sb 121.760	52 Te 127.60	53 I 126.90447	54 Xe 131.29
55 Cs 132.90545	56 Ba 137.327	71 Lu 174.967	72 Hf 178.49	73 Ta 180.9479	74 W 183.84	75 Re 186.207	76 Os 190.23	77 Ir 192.217	78 Pt 195.078	79 Au 196.96655	80 Hg 200.59	81 Tl 204.3833	82 Pb 207.2	83 Bi 208.98038	84 Po 209.0	85 At 210.0	86 Rn 222.0
87 Fr 223.0	88 Ra 226.0	103 Lr 262.0	104 Rf 261.0	105 Db 262.0	106 Sg 263.0	107 Bh 264.0	108 Hs 265.0	109 Mt 268.0	110 Ds	111 Uuu	112 Uub	113 Uut	114 Uuq	115 Uup	116 Uuh	117 Uus	118 Uuo

Lanthanoids	57 La 138.9055	58 Ce 140.116	59 Pr 140.90765	60 Nd 144.24	61 Pm 145.0	62 Sm 150.36	63 Eu 151.964	64 Gd 157.25	65 Tb 158.92534	66 Dy 162.50	67 Ho 164.93032	68 Er 167.26	69 Tm 168.93421	70 Yb 173.04
Actinoids	89 Ac 227.0	90 Th 232.0381	91 Pa 231.03588	92 U 238.0289	93 Np 237.0	94 Pu 244.0	95 Am 243.0	96 Cm 247.0	97 Bk 247.0	98 Cf 251.0	99 Es 252.0	100 Fm 257.0	101 Md 258.0	102 No 259.0

Table 3: The default nuclear masses used in CASINO.

- Condensate fraction estimator in momentum space - **cond_fraction_mom** (HOM)
- Localization tensor - **loc_tensor** (PER)
- Dipole moment - **dipole_moment** (MOL)
- Population - **population** (ATOM, MOL, FIN, PER)

By default these observables are not accumulated during a VMC/DMC simulation; to do this one must set to **T** the corresponding input keyword (the bold terms in brackets in the above list). With the exception of the dipole moment (for which the required information is stored in the `vmc.hist`/`dmc.hist` file) the activation of any of the above keywords will flag the creation of an `expval.data` file (see Sec. 7.13) wherein the required data will be accumulated.

The data in the `expval.data` file is stored in independent sets corresponding to each observable. If a data set is already present at the start of a calculation, then any newly accumulated data will be added to the existing data. The `expval.data` file also contains basic information about the system, plus all the **G**-vector sets necessary to represent any reciprocal-space quantities.

At the end of the calculation, the data in `expval.data` can usually be visualized using the **plot_expval** utility. The use of this program is fairly self-explanatory. Type 'plot_expval' in any directory containing an `expval.data` file, and the utility will read the data then ask you a series of questions designed to elicit information about exactly what kind of plot you want. It will then write out the data in a file readable by standard plotting programs such as XMGRACE (for 1D data) or GNUPLOT (for 2D/3D data). A CASINO shell-script—`plot_2D`—is available which will call GNUPLOT with appropriate arguments.

Note that, for operators that do not commute with the Hamiltonian, the error in the usual DMC mixed estimator will be linear in the error in the wave function. However, the error in the extrapolated estimator $2\mathbf{p}_{\mathrm{DMC}} - \mathbf{p}_{\mathrm{VMC}}$ will be quadratic in the error in the wave function (Here $\mathbf{p}_{\mathrm{VMC}}$ and $\mathbf{p}_{\mathrm{DMC}}$ are the VMC and DMC estimates of the expectation value using the same wave function). One may also use the future walking technique (see Sec. 37) to obtain better estimates for such observables.

After a short summary of relevant theoretical results, each expectation value will now be described in turn.

35.1 Basics

35.1.1 Fourier transforms

Define the Fourier transform and its inverse by

$$\tilde{f}(\mathbf{G}) = \frac{1}{\Omega}\int_{\Omega} f(\mathbf{r})\, e^{i\mathbf{G}\cdot\mathbf{r}}\, d\mathbf{r} \tag{383}$$

$$f(\mathbf{r}) = \sum_{\mathbf{G}} \tilde{f}(\mathbf{G})\, e^{-i\mathbf{G}\cdot\mathbf{r}}, \tag{384}$$

where the set of **G** vectors are the reciprocal lattice vectors of the simulation cell lattice. Using this definition, the Kronecker delta is

$$\delta_{\mathbf{G},\mathbf{G}'} = \frac{1}{\Omega}\int e^{i(\mathbf{G}'-\mathbf{G})\cdot\mathbf{r}}\, d\mathbf{r}, \tag{385}$$

and the Dirac delta function is

$$\delta(\mathbf{r}-\mathbf{r}') = \frac{1}{\Omega}\sum_{\mathbf{G}} e^{-i\mathbf{G}\cdot(\mathbf{r}-\mathbf{r}')}. \tag{386}$$

The $3N$-dimensional Fourier transform of the wave function is

$$\tilde{\Psi}(\mathbf{k}_1,\ldots,\mathbf{k}_N) = \frac{1}{\Omega^N}\int \Psi(\mathbf{r}_1,\ldots,\mathbf{r}_N)\, e^{i\mathbf{k}_1\cdot\mathbf{r}_1}\ldots e^{i\mathbf{k}_N\cdot\mathbf{r}_N}\, d\mathbf{r}_1\ldots d\mathbf{r}_N \tag{387}$$

$$\Psi(\mathbf{r}_1,\ldots,\mathbf{r}_N) = \sum_{\mathbf{k}_1,\ldots,\mathbf{k}_N} \tilde{\Psi}(\mathbf{k}_1,\ldots,\mathbf{k}_N)\, e^{-i\mathbf{k}_1\cdot\mathbf{r}_1}\ldots e^{-i\mathbf{k}_N\cdot\mathbf{r}_N}. \tag{388}$$

35.1.2 Translational averaging

Consider a function $f(\mathbf{r},\mathbf{r}')$ whose Fourier transform is

$$\tilde{f}(\mathbf{k},\mathbf{k}') = \frac{1}{\Omega^2}\int f(\mathbf{r},\mathbf{r}')\, e^{i\mathbf{k}\cdot\mathbf{r}} e^{i\mathbf{k}'\cdot\mathbf{r}'}\, d\mathbf{r}\, d\mathbf{r}'. \tag{389}$$

The translational average of $f(\mathbf{r},\mathbf{r}')$ is

$$f^{\mathrm{T}}(\mathbf{r}) = \frac{1}{\Omega}\int f(\mathbf{r}'',\mathbf{r}''')\,\delta(\mathbf{r}''-\mathbf{r}'''-\mathbf{r})\, d\mathbf{r}''d\mathbf{r}'''. \tag{390}$$

The Fourier transform of the translational average is

$$\begin{aligned}\tilde{f}^{\mathrm{T}}(\mathbf{k}) &= \frac{1}{\Omega}\int f^{\mathrm{T}}(\mathbf{r})\, e^{i\mathbf{k}\cdot\mathbf{r}}\, d\mathbf{r} &(391)\\ &= \frac{1}{\Omega^2}\tilde{f}(\mathbf{k},-\mathbf{k}). &(392)\end{aligned}$$

Therefore the Fourier transform of $f(\mathbf{r},\mathbf{r}')$ and the Fourier transform of its translational average are related by

$$\tilde{f}(\mathbf{k},-\mathbf{k}) = \Omega^2\tilde{f}^{\mathrm{T}}(\mathbf{k}). \tag{393}$$

The operations of translational averaging and Fourier transforming commute.

The function $f(\mathbf{r},\mathbf{r}')$ has the periodicity of the simulation cell,

$$f(\mathbf{r},\mathbf{r}'+\mathbf{R}) = f(\mathbf{r},\mathbf{r}'), \tag{394}$$

where $\mathbf{R}$ is a translation vector of the simulation cell lattice. Equation (394) implies that $f^{\mathrm{T}}(\mathbf{r})$ is also periodic,

$$f^{\mathrm{T}}(\mathbf{r}+\mathbf{R}) = f^{\mathrm{T}}(\mathbf{r}). \tag{395}$$

The Fourier transforms of $f^{\mathrm{T}}(\mathbf{r})$ and $f(\mathbf{r},\mathbf{r}')$ are therefore nonzero only on the reciprocal lattice vectors of the simulation cell lattice.

35.1.3 Rotational averaging

The rotational average of a function is

$$f^{\mathrm{R}}(r) = \frac{1}{4\pi r^2}\int f(\mathbf{r}')\,\delta(|\mathbf{r}'|-r)\, d\mathbf{r}'. \tag{396}$$

The rotationally averaged function $f^{\mathrm{R}}(r)$ is no longer periodic. The Fourier transform of the rotational average is

$$\begin{aligned}\tilde{f}^{\mathrm{R}}(k) &= \frac{1}{\Omega}\int f^{\mathrm{R}}(r)\, e^{i\mathbf{k}\cdot\mathbf{r}}\, d\mathbf{r} &(397)\\ &= \frac{1}{\Omega}\int_0^\infty f^{\mathrm{R}}(r)\, 4\pi r^2\, \frac{\sin(kr)}{kr}\, dr. &(398)\end{aligned}$$

The operations of rotational averaging and Fourier transforming commute because

$$\frac{1}{4\pi k^2\Omega}\int f(\mathbf{r})\, e^{i\mathbf{k}'\cdot\mathbf{r}}\delta(|\mathbf{k}'|-k)\, d\mathbf{k}'\, d\mathbf{r} = \int \frac{1}{4\pi r^2\Omega} f(\mathbf{r}')\,\delta(|\mathbf{r}'|-r)\, e^{i\mathbf{k}\cdot\mathbf{r}}\, d\mathbf{r}'\, d\mathbf{r}. \tag{399}$$

In practise, two-point correlation functions such as the pair correlation function and density matrix are calculated by summing over the contributions from pairs of particles whose separation is defined by the minimum image convention. We accumulate contributions from all pairs of particles whose separation is less than or equal to the radius L_{WS} of the sphere inscribed within the WS cell of the simulation cell. We then accumulate

$$\begin{aligned}\tilde{f}^{\mathrm{R}}_{\mathrm{WS}}(k) &= \frac{1}{\Omega}\int f^{\mathrm{R}}(r)\,\Theta(L_{\mathrm{WS}}-r)\, e^{i\mathbf{k}\cdot\mathbf{r}}\, d\mathbf{r} &(400)\\ &= \frac{1}{\Omega}\int_0^\infty f^{\mathrm{R}}(r)\,\Theta(L_{\mathrm{WS}}-r)\, 4\pi r^2\, \frac{\sin(kr)}{kr}\, dr, &(401)\end{aligned}$$

which is the convolution of $\tilde{f}(k)$ with the Fourier transform of the Heaviside function. Rather than perform the deconvolution it is probably better just to calculate the translational average in reciprocal space on the reciprocal lattice vectors of the simulation cell, averaging over the length of the vectors as the calculation proceeds.

A spherical average can be performed in reciprocal space to obtain

$$\tilde{f}^{\mathrm{TR}}(G) = \frac{1}{N_G} \sum_{i=1}^{N_G} \tilde{f}^{\mathrm{T}}(\mathbf{G}_i), \tag{402}$$

where N_G is the number of reciprocal lattice vectors of length G. The spherical average can be accumulated as the calculation proceeds.

35.2 Density and spin density

*K*eywords: **density**, **spin_density**

35.2.1 Periodic systems

The charge density operator for spin α is

$$\rho_\alpha(\mathbf{r}) = \sum_{i=1}^{N_\alpha} \delta(\mathbf{r} - \mathbf{r}_{i\alpha}). \tag{403}$$

The Fourier transform of the charge density operator is

$$\begin{aligned} \tilde{\rho}_\alpha(\mathbf{k}) &= \frac{1}{\Omega} \int \sum_{i=1}^{N_\alpha} e^{i\mathbf{k}\cdot\mathbf{r}} \delta(\mathbf{r} - \mathbf{r}_{i\alpha})\, d\mathbf{r} && (404) \\ &= \frac{1}{\Omega} \sum_{i=1}^{N_\alpha} e^{i\mathbf{k}\cdot\mathbf{r}_{i\alpha}}. && (405) \end{aligned}$$

The charge density for spin α is

$$\begin{aligned} n_\alpha(\mathbf{r}) &= \frac{\int |\Psi|^2 \rho_\alpha(\mathbf{r})\, d\mathbf{R}}{\int |\Psi|^2\, d\mathbf{R}} && (406) \\ &= \frac{\int |\Psi|^2 \sum_{i=1}^{N_\alpha} \delta(\mathbf{r} - \mathbf{r}_{i\alpha})\, d\mathbf{R}}{\int |\Psi|^2\, d\mathbf{R}}. && (407) \end{aligned}$$

The Fourier transform of the charge density is

$$\tilde{n}_\alpha(\mathbf{k}) = \frac{1}{\Omega} \int n_\alpha(\mathbf{r})\, e^{i\mathbf{k}\cdot\mathbf{r}}\, d\mathbf{r}. \tag{408}$$

So the charge density may be accumulated by summing $\exp(i\mathbf{G}\cdot\mathbf{r})$ for each primitive-cell $\mathbf{G}$ vector after each single electron move from $\mathbf{r}' \longrightarrow \mathbf{r}$. At the end of the simulation, we then divide by the total weight (e.g., the number of accumulation steps in VMC) to get the average of each $n(\mathbf{G})$. The Fourier coefficients are normalized such that $n(\mathbf{G} = 0)$ is the number of electrons in the primitive cell (which is obtained from the above average $n(G)$ through division by the number of primitive cells in the simulation cell).

35.2.2 Atomic densities

*K*eywords: **density** or **spin_density** (both produce the same thing)

In the single-atom case, the charge density is obviously spherically symmetric, $\rho(\mathbf{r}) = \rho(r)$. The spherical average of the charge density is

$$\rho_{\mathrm{sph}}(r) = \frac{1}{4\pi r^2} \sum_{i=1}^{N} \frac{\int f(\mathbf{R}) \delta(|\mathbf{r}_i| - r) d\mathbf{R}}{\int f(\mathbf{R}) d\mathbf{R}}\ , \tag{409}$$

where $f(\mathbf{R})$ is the distribution of configurations. The charge density is normalized such that $\int 4\pi r^2 \rho_{\rm sph}(r)dr = N$.

The analytical behaviour of $\rho(r)$, $\rho_{\rm sph}(r)$ at $r \to 0$ and $r \to \infty$ is known,

$$\rho'_{\rm sph}(0) = -2Z\rho_{\rm sph}(0) \ , \tag{410}$$

where Z is the atomic number, and

$$\rho_{\rm sph}(r) \underset{r\to\infty}{\to} Ar^{2B}\exp\left(-2\sqrt{2I}r\right) \ , \tag{411}$$

where A is a constant, $B = (Z - N + 1)/\sqrt{2I} - 1$, and I is the ionization energy.

The method used to calculate the charge density in QMC is presented below for the general case of the charge density, but it is easily applied to quantities such as the spherical average of the charge density.

Suppose we are performing a QMC calculation, sampling a distribution $f(\mathbf{R})$. In the case of VMC, $f(\mathbf{R}) = |\Psi(\mathbf{R})|^2$, while in DMC the distribution is $f(\mathbf{R}) = \Psi(\mathbf{R})\Phi(\mathbf{R})$. The charge density can be evaluated in QMC as

$$\rho(\mathbf{r}) \approx \bar{\rho}(\mathbf{r}) = \frac{\sum_{m=1}^{M} w_m \rho(\mathbf{R}_m)}{\sum_{m=1}^{M} w_m} \ , \tag{412}$$

where $\{\mathbf{R}_m\}_{m=1}^{M}$ are the M configurations visited during the simulation and w_m is the weight of each of them, which is constant in VMC and varies from configuration to configuration in DMC. The 'approximately equal' sign becomes an 'equal' sign in the limit of perfect sampling, that is, when $M \to \infty$.

We can identify $\sum_i \delta(\mathbf{r}_i - \mathbf{r})$ with $\rho(\mathbf{R})$, resulting in

$$\rho(\mathbf{r}) \approx \bar{\rho}(\mathbf{r}) = \frac{\sum_{m=1}^{M}\sum_i w_m \delta(\mathbf{r}_{i,m} - \mathbf{r})}{\sum_{m=1}^{M} w_m} \ . \tag{413}$$

However, Eq. (413) is not useful for accumulating $\bar{\rho}(\mathbf{r})$ since Dirac delta functions cannot be evaluated outside an integral.

We overcome this by accumulating data in *bins* of finite width. Let us partition three-dimensional space into $N_{\rm exp}$ disjoint regions or *bins* $\{\Omega_{\rm p}\}$, and let us define the step function

$$\Theta_{\rm p}(\mathbf{r}) = \left\{ \begin{array}{ll} 1 & \text{if } \mathbf{r} \in \Omega_{\rm p} \\ 0 & \text{if } \mathbf{r} \notin \Omega_{\rm p} \end{array} \right. \ . \tag{414}$$

Denoting the volume of region $\Omega_{\rm p}$ by the same symbol, $\Omega_{\rm p}$, we can construct an orthogonal functional basis $h_{\rm p}(\mathbf{r}) = \Omega_{\rm p}^{-1/2}\Theta_{\rm p}(\mathbf{r})$. Then, Eq. (413) becomes

$$\bar{\rho}(\mathbf{r}) \approx \sum_{p=1}^{N_{\rm exp}} \frac{\sum_{m=1}^{M}\sum_i w_m \Theta_{\rm p}(\mathbf{r}_{i,m})}{\Omega_{\rm p}\sum_{m=1}^{M} w_m}\Theta_{\rm p}(\mathbf{r}) \ , \tag{415}$$

that is, the function inside $\Omega_{\rm p}$ is approximated by a single value

$$\rho_{\rm p} = \frac{\sum_{m=1}^{M}\sum_i w_m \Theta_{\rm p}(\mathbf{r}_{i,m})}{\Omega_{\rm p}\sum_{m=1}^{M} w_m} \ . \tag{416}$$

When plotting a binned estimator, it is convenient to draw a single point per bin rather than a horizontal line, and this point should be located at the geometrical centre of the bin,

$$\mathbf{r}_{\rm p} = \frac{1}{\Omega_{\rm p}}\int_{\Omega_{\rm p}} \mathbf{r}d\mathbf{r} \ . \tag{417}$$

If any symmetry constraints are imposed, *e.g.*, the function is spherically symmetric and it is accumulated in spherical bins along a radial grid, the above expression should be modified so that the integrand remains the coordinate on the left-hand side. For example, to calculate the radius $r_{\rm p}$ at which we should plot $\rho_{\rm p} \approx \bar{\rho}(r_{\rm p})$ in the spherical annulus $\Omega_{\rm p}$, we would use the expression

$$r_{\rm p} = \frac{1}{\Omega_{\rm p}}\int_{\Omega_{\rm p}} r d\mathbf{r} = \frac{4\pi}{4\pi\left(b^3 - a^3\right)/3}\int_a^b r^3 dr = \frac{\left(b^4 - a^4\right)/4}{\left(b^3 - a^3\right)/3} = \frac{3}{4}\frac{b^3 + b^2a + ba^2 + a^3}{b^2 + ba + a^2} \ , \tag{418}$$

where a and b are the inner and outer radii of Ω_{p}, respectively.

Using similar analysis, we can calculate the standard error in $\bar{\rho}$, $\sigma_{\bar{\rho}}(\mathbf{r})$, given by

$$\sigma_{\bar{\rho}}^2(\mathbf{r}) = \frac{\sum_{m=1}^{M} w_m(\rho(\mathbf{R}_m) - \bar{\rho}(\mathbf{R}_m))^2}{M\left(\sum_{m=1}^{M} w_m - \frac{\sum_{m=1}^{M} w_m^2}{\sum_{m=1}^{M} w_m}\right)} . \tag{419}$$

Note that contribution to the error from serial correlation has been neglected; the configurations are assumed to be independent.

We are also interested in expectation values that represent a quantity associated with the relative vectorial position of two particles (the ith and the jth particles, for instance) being fixed at $\mathbf{r}$, such as the intracule density, $h(\mathbf{r})$.

The spherical average of the intracule density, $h_{\mathrm{sph}}(r)$ is

$$h_{\mathrm{sph}}(r) = \frac{1}{2}\frac{1}{4\pi r^2}\sum_{i\neq j}\frac{\int f(\mathbf{R})\delta(|\mathbf{r}_{ij}| - r)d\mathbf{R}}{\int f(\mathbf{R})d\mathbf{R}} , \tag{420}$$

such that $\int 4\pi r^2 h_{\mathrm{sph}}(r)dr = N(N-1)/2$.

This accumulation process in this case is analogous to that in the single-particle case, and one only needs to include a factor of 1/2 and replace $\mathbf{r}_i$ with $\mathbf{r}_{ij}$ in the formulae in this section to adapt them to the two-particle case.

The n^{th} radial moment μ_n of a function $g(\mathbf{r})$ is defined as

$$\mu_n = \int |\mathbf{r}|^n g(\mathbf{r})d\mathbf{r} = 4\pi\int_0^\infty r^{n+2} g_{\mathrm{sph}}(r)dr . \tag{421}$$

The moments of the charge density are given by $g(\mathbf{r}) = \rho(\mathbf{r})$ and the moments of the intracule density are given by $g(\mathbf{r}) = h(\mathbf{r})$. They are calculated for $n = -2, -1, 1, 2, 3$ and written to rad_mom.dat.

The systems we deal with usually contain several indistinguishable particles for which the expectation value of any $\mathbf{r}_i$-dependent observable should give the same result. It is possible to take statistical advantage of the presence of indistinguishable particles by averaging any QMC-accumulated quantities over all particles of the same type.

Moreover, in cases where different particle classes are equivalent (*e.g.*, when there are the same number of up- and down-spin electrons in a system without magnetic fields, provided $g(\mathbf{r}_i)$ does not itself depend on the spin of the particles) one can average over classes as well.

For $\mathbf{r}_{ij}$-dependent expectation values it is also possible to make use of the presence of indistinguishable particles to achieve better statistics. In this case one has to average any QMC-accumulated quantities over all particle pairs of the same relative type (*e.g.*, all parallel-spin electron pairs).

This average has been omitted in this document for simplicity. However, it is important to do this in practice for improved statistics.

35.2.3 Molecular density

*K*eywords: **density** or **spin_density** in an aperiodic system (both produce the same thing).

The charge density may be accumulated by binning on a regular $l \times m \times n$ mesh aligned with the Cartesian axes, spanning a cuboid with diagonally opposite vertices at points A and B. The grid dimensions l, m and n and the Cartesian coordinates of points A and B are specified in the 'MOLECULAR DENSITY' block in `expval.data`. One set is written out for each species of particle. The PLOT_EXPVAL utility will allow the user to perform spherical or cylindrical averaging of the molecular charge density.

35.2.4 Spin-density matrix

*K*eywords: **density** or **spin_density** in a non-collinear spin system (both produce the same thing).

The spin-density matrix is a two-by-two matrix which is the generalization of the spin density in a noncollinear-spin system (see Sec. 38). It is diagonal for the case of collinear spins, with the two diagonal elements being equal to the spin densities.

Let $\mathbf{x}_i \equiv \{\mathbf{r}_i, s_i\}$, where $\mathbf{r}_i$ and s_i are the spatial and spin coordinates of electron i. We use the notation $\int f(\mathbf{x})\,d\mathbf{x} \equiv \sum_s \int f(\{\mathbf{r},s\})\,d\mathbf{r}$.

The spin-density matrix is defined as

$$\rho_{\rm sdm}(\mathbf{r};s,s') = \frac{N\int \Psi^*(\{\mathbf{r},s'\},\mathbf{x}_2,\ldots,\mathbf{x}_N)\Psi(\{\mathbf{r},s\},\mathbf{x}_2,\ldots,\mathbf{x}_N)\,d\mathbf{x}_2\ldots d\mathbf{x}_N}{\int |\Psi|^2\,d\mathbf{x}_1\ldots d\mathbf{x}_N}. \tag{422}$$

Note that $\rho_{\rm sdm}(\mathbf{r};s,s') = \rho^*_{\rm sdm}(\mathbf{r};s',s)$.

The diagonal elements of the spin-density matrix, which are real, give the spin density:

$$\rho_{\rm s}(\mathbf{r};s) = \rho_{\rm sdm}(\mathbf{r};s,s). \tag{423}$$

The magnetization density operator is defined as

$$\hat{\mathbf{M}}(\mathbf{r}) = 2\sum_i \delta(\mathbf{r}-\mathbf{r}_i)\hat{\mathbf{s}}_i, \tag{424}$$

where $\hat{\mathbf{s}}_i$ is the spin operator for electron i. Let $\mathbf{M}(\mathbf{r}) = \langle \hat{\mathbf{M}}(\mathbf{r})\rangle$.

So

$$\begin{aligned} M_z(\mathbf{r}) &= \frac{2\sum_i \int \Psi^*\delta(\mathbf{r}-\mathbf{r}_i)\hat{s}_{iz}\Psi\,d\mathbf{x}_1\ldots d\mathbf{x}_N}{\int |\Psi|^2\,d\mathbf{x}_1\ldots d\mathbf{x}_N} \\ &= \frac{2N\sum_{s_1}\int \Psi^*(\{\mathbf{r},s_1\},\mathbf{x}_2,\ldots,\mathbf{x}_N)\hat{s}_{1z}\Psi(\{\mathbf{r},s_1\},\mathbf{x}_2,\ldots,\mathbf{x}_N)\,d\mathbf{x}_2\ldots d\mathbf{x}_N}{\int |\Psi|^2\,d\mathbf{x}_1\ldots d\mathbf{x}_N} \\ &= N\frac{\int |\Psi(\{\mathbf{r},\uparrow\},\mathbf{x}_2,\ldots,\mathbf{x}_N)|^2\,d\mathbf{x}_2\ldots d\mathbf{x}_N - \int |\Psi(\{\mathbf{r},\downarrow\},\mathbf{x}_2,\ldots,\mathbf{x}_N)|^2\,d\mathbf{x}_2\ldots d\mathbf{x}_N}{\int |\Psi|^2\,d\mathbf{x}_1\ldots d\mathbf{x}_N} \\ &= \rho_{\rm sdm}(\mathbf{r};\uparrow,\uparrow) - \rho_{\rm sdm}(\mathbf{r};\downarrow,\downarrow) = \rho_\uparrow(\mathbf{r}) - \rho_\downarrow(\mathbf{r}). \end{aligned} \tag{425}$$

We also have

$$\begin{aligned} M_x(\mathbf{r}) &= \frac{2\sum_i \int \Psi^*\delta(\mathbf{r}-\mathbf{r}_i)\hat{s}_{ix}\Psi\,d\mathbf{x}_1\ldots d\mathbf{x}_N}{\int |\Psi|^2\,d\mathbf{x}_1\ldots d\mathbf{x}_N} \\ &= N\frac{\sum_{s_1}\int \Psi^*(\{\mathbf{r},s_1\},\mathbf{x}_2,\ldots,\mathbf{x}_N)[\hat{s}_{1+}+\hat{s}_{1-}]\Psi(\{\mathbf{r},s_1\},\mathbf{x}_2,\ldots,\mathbf{x}_N)\,d\mathbf{x}_2\ldots d\mathbf{x}_N}{\int |\Psi|^2\,d\mathbf{x}_1\ldots d\mathbf{x}_N} \\ &= N\left[\frac{\int \Psi^*(\{\mathbf{r},\uparrow\},\mathbf{x}_2,\ldots,\mathbf{x}_N)\Psi(\{\mathbf{r},\downarrow\},\mathbf{x}_2,\ldots,\mathbf{x}_N)\,d\mathbf{x}_2\ldots d\mathbf{x}_N}{\int |\Psi|^2\,d\mathbf{x}_1\ldots d\mathbf{x}_N}\right. \\ &\qquad \left.\frac{+\int \Psi^*(\{\mathbf{r},\downarrow\},\mathbf{x}_2,\ldots,\mathbf{x}_N)\Psi(\{\mathbf{r},\uparrow\},\mathbf{x}_2,\ldots,\mathbf{x}_N)\,d\mathbf{x}_2\ldots d\mathbf{x}_N}{\int |\Psi|^2\,d\mathbf{x}_1\ldots d\mathbf{x}_N}\right] \\ &= \rho_{\rm sdm}(\mathbf{r};\downarrow,\uparrow) + \rho_{\rm sdm}(\mathbf{r};\uparrow,\downarrow) = 2{\rm Re}\left[\rho_{\rm sdm}(\mathbf{r};\downarrow,\uparrow)\right], \end{aligned} \tag{426}$$

where we have used the raising and lowering properties of the ladder operators $\hat{s}_+ = \hat{s}_x + i\hat{s}_y$ and $\hat{s}_- = \hat{s}_x - i\hat{s}_y$. In a similar fashion,

$$M_y(\mathbf{r}) = 2{\rm Im}\left[\rho_{\rm sdm}(\mathbf{r};\downarrow,\uparrow)\right]. \tag{427}$$

To evaluate the spin-density matrix in VMC, we rewrite it as

$$\begin{aligned} \rho_{\rm sdm}(\mathbf{r};s,s') &= \sum_i \frac{\int |\Psi|^2\,\delta(\mathbf{r}-\mathbf{r}_i)\,\delta_{s_i,s'}\,\frac{\Psi(\ldots,\{\mathbf{r}_i,s\},\ldots)}{\Psi(\ldots,\{\mathbf{r}_i,s'\},\ldots)}\,d\mathbf{x}_1\ldots d\mathbf{x}_N}{\int |\Psi|^2\,d\mathbf{x}_1\ldots d\mathbf{x}_N} \\ &= \left\langle \sum_i \delta(\mathbf{r}-\mathbf{r}_i)\,\delta_{s_i,s'}\,\frac{\Psi(\ldots,\{\mathbf{r}_i,s\},\ldots)}{\Psi(\ldots,\{\mathbf{r}_i,s'\},\ldots)}\right\rangle. \end{aligned} \tag{428}$$

In practice we gather the Fourier components of the spin-density matrix, which are

$$\tilde{\rho}_{\rm sdm}(\mathbf{G};s,s') = \frac{1}{\Omega}\left\langle \sum_i \exp(-i\mathbf{G}\cdot\mathbf{r}_i)\,\delta_{s_i,s'}\,\frac{\Psi(\ldots,\{\mathbf{r}_i,s\},\ldots)}{\Psi(\ldots,\{\mathbf{r}_i,s'\},\ldots)}\right\rangle, \tag{429}$$

where $\mathbf{G}$ is a reciprocal lattice vector of the simulation cell.

After each configuration move, we can evaluate a contribution to two out of the four components of $\tilde{\rho}_{\rm sdm}$ for each electron. Given that the ith electron has spin value s_i, we can record $\tilde{\rho}_{\rm sdm}(\mathbf{G}; s, s_i)$ for both values of s by evaluating the ratio of the wave functions for those two values.

CASINO evaluates the Fourier components of the spin-density matrix if the **spin_density** keyword is set to `T` in a noncollinear-spin calculation. The Fourier components are stored in the `expval.data` file. The PLOT_EXPVAL utility reads the Fourier components in the `expval.data` file and enables the user to plot the components of the spin-density matrix along lines in real space. It also asks the user if he or she wishes to plot the components of the magnetization density along lines in real space.

35.3 Reciprocal-space and spherical real-space pair-correlation functions

*K*eywords: **pair_corr**, **pair_corr_sph**

The two-particle density matrix is defined as

$$\gamma^{(2)}_{\alpha\beta}(\mathbf{r},\mathbf{r}';\mathbf{r}'',\mathbf{r}''') = N_\alpha(N_\beta-\delta_{\alpha\beta})\frac{\int \Psi^*(\mathbf{r},\mathbf{r}',\mathbf{r}_3,\ldots,\mathbf{r}_N)\Psi(\mathbf{r}'',\mathbf{r}''',\mathbf{r}_3,\ldots,\mathbf{r}_N)\,d\mathbf{r}_3\ldots d\mathbf{r}_N}{\int \Psi^*(\mathbf{r}_1,\mathbf{r}_2,\ldots,\mathbf{r}_N)\Psi(\mathbf{r}_1,\mathbf{r}_2,\ldots,\mathbf{r}_N)\,d\mathbf{R}}, \quad (430)$$

where $\mathbf{r}$ and $\mathbf{r}''$ are α-spin electron coordinates and $\mathbf{r}'$ and $\mathbf{r}'''$ are β-spin electron coordinates. The diagonal elements of the two-particle density matrix,

$$\gamma^{(2)}_{\alpha\beta}(\mathbf{r},\mathbf{r}';\mathbf{r},\mathbf{r}') = N_\alpha(N_\beta-\delta_{\alpha\beta})\frac{\int \Psi^*(\mathbf{r},\mathbf{r}',\mathbf{r}_3,\ldots,\mathbf{r}_N)\Psi(\mathbf{r},\mathbf{r}',\mathbf{r}_3,\ldots,\mathbf{r}_N)\,d\mathbf{r}_3\ldots d\mathbf{r}_N}{\int \Psi^*(\mathbf{r}_1,\mathbf{r}_2,\ldots,\mathbf{r}_N)\Psi(\mathbf{r}_1,\mathbf{r}_2,\ldots,\mathbf{r}_N)\,d\mathbf{R}}, \quad (431)$$

are of special interest. The normalization has been chosen so that

$$\int \gamma^{(2)}_{\alpha\beta}(\mathbf{r},\mathbf{r}';\mathbf{r},\mathbf{r}')\,d\mathbf{r}\,d\mathbf{r}' = N_\alpha(N_\beta-\delta_{\alpha\beta}). \quad (432)$$

The sum over spin indices gives

$$\sum_{\alpha\beta} N_\alpha(N_\beta-\delta_{\alpha\beta}) = N(N-1), \quad (433)$$

where $N = N_\alpha + N_\beta$.

The pair correlation functions, $g_{\alpha\beta}(\mathbf{r},\mathbf{r}')$, are related to the diagonal elements of the two-particle density matrix by

$$\gamma^{(2)}_{\alpha\beta}(\mathbf{r},\mathbf{r}';\mathbf{r},\mathbf{r}') = n_\alpha(\mathbf{r})\,n_\beta(\mathbf{r}')\,g_{\alpha\beta}(\mathbf{r},\mathbf{r}'). \quad (434)$$

The pair correlation functions are given by

$$\begin{aligned} g_{\alpha\beta}(\mathbf{r},\mathbf{r}') &= \frac{1}{n_\alpha(\mathbf{r})n_\beta(\mathbf{r}')}\frac{\int|\Psi|^2\sum_{i=1}^{N_\alpha}\delta(\mathbf{r}_{i=1}^{N_\alpha}-\mathbf{r})\sum_{j=1\ (j\neq i\text{ if }\alpha=\beta)}^{N_\beta}\delta(\mathbf{r}_{j\beta}-\mathbf{r}')\,d\mathbf{R}}{\int|\Psi|^2\,d\mathbf{R}} & (435)\\ &= \frac{N_\alpha(N_\beta-\delta_{\alpha\beta})}{n_\alpha(\mathbf{r})n_\beta(\mathbf{r}')}\frac{\int|\Psi|^2\delta(\mathbf{r}_{i\alpha}-\mathbf{r})\,\delta(\mathbf{r}_{j\beta}-\mathbf{r}')\,d\mathbf{R}}{\int|\Psi|^2\,d\mathbf{R}}. & (436)\end{aligned}$$

35.3.1 The total or spin-averaged pair correlation function

$$g(\mathbf{r},\mathbf{r}') = \sum_{\alpha,\beta}\frac{n_\alpha(\mathbf{r})n_\beta(\mathbf{r}')}{n(\mathbf{r})n(\mathbf{r}')}\,g_{\alpha\beta}(\mathbf{r},\mathbf{r}'). \quad (437)$$

35.3.2 Properties of the pair correlation functions

The pair correlation functions satisfy the following properties:

$$\begin{aligned} g_{\alpha\beta}(\mathbf{r},\mathbf{r}') &\geq 0 & (438)\\ g_{\alpha\beta}(\mathbf{r},\mathbf{r}') &= g_{\beta\alpha}(\mathbf{r}',\mathbf{r}) & (439)\\ g_{\alpha\beta}(\mathbf{r},\mathbf{r}') &= 0 \quad \text{for}\ \ \alpha=\beta,\ \ \mathbf{r}=\mathbf{r}' & (440)\\ \int n_\beta(\mathbf{r}')\,[g_{\alpha\beta}(\mathbf{r},\mathbf{r}')-1]\,d\mathbf{r}' &= -\delta_{\alpha\beta}. & (441)\end{aligned}$$

35.3.3 Homogeneous and isotropic systems

Suppose we specialize to a homogeneous and isotropic system where $g_{\alpha\beta}(\mathbf{r},\mathbf{r}') = g_{\alpha\beta}(|\mathbf{r}-\mathbf{r}'|) = g_{\alpha\beta}(r)$, and $n_\alpha(\mathbf{r}) = N_\alpha/\Omega$, where Ω is the volume of the simulation cell. Performing a translational average of $g_{\alpha\beta}$ we obtain

$$\begin{aligned} g_{\alpha\beta}(\mathbf{r}) &= \frac{1}{\Omega}\int g_{\alpha\beta}(\mathbf{r}'',\mathbf{r}''')\,\delta(\mathbf{r}''-\mathbf{r}'''-\mathbf{r})\,d\mathbf{r}''d\mathbf{r}''' \qquad (442) \\ &= \frac{\Omega}{N_\alpha N_\beta}\frac{\int |\Psi|^2 \sum_{i=1}^{N_\alpha}\sum_{j=1\ (j\neq i\ \text{if}\ \alpha=\beta)} \delta(\mathbf{r}_{i\alpha}-\mathbf{r}_{j\beta}-\mathbf{r})\,d\mathbf{R}}{\int |\Psi|^2\,d\mathbf{R}}. \qquad (443) \end{aligned}$$

Performing a rotational average we obtain

$$\begin{aligned} g_{\alpha\beta}(r) &= \frac{1}{4\pi r^2}\int g_{\alpha\beta}(\mathbf{r}')\,\delta(|\mathbf{r}'|-r)\,d\mathbf{r}' \qquad (444) \\ &= \frac{\Omega}{4\pi r^2 N_\alpha N_\beta}\frac{\int |\Psi|^2\ \sum_{i=1}^{N_\alpha}\sum_{j=1\ (j\neq i\ \text{if}\ \alpha=\beta)}^{N_\beta} \delta(|\mathbf{r}_{i\alpha}-\mathbf{r}_{j\beta}|-r)\,d\mathbf{R}}{\int |\Psi|^2\,d\mathbf{R}}. \qquad (445) \end{aligned}$$

The rotationally and translationally averaged pair correlation functions can be evaluated by collecting in bins; see Sec. 35.3.5.

The sum rule of Eq. (441) gives

$$\frac{N_\beta}{\Omega}\int [g_{\alpha\beta}(r)-1]\,4\pi r^2\,dr \quad = \quad -\delta_{\alpha\beta}. \tag{446}$$

35.3.4 Translational and rotational averaging of $g_{\alpha\beta}(\mathbf{r},\mathbf{r}')$

The translational average of $g_{\alpha\beta}(\mathbf{r},\mathbf{r}')$ is

$$\begin{aligned} g_{\alpha\beta}(\mathbf{r}) &= \frac{1}{\Omega}\int g_{\alpha\beta}(\mathbf{r}'',\mathbf{r}''')\,\delta(\mathbf{r}''-\mathbf{r}'''-\mathbf{r})\,d\mathbf{r}''\,d\mathbf{r}''' \qquad (447) \\ &= \frac{1}{\Omega}\int \frac{|\Psi|^2\sum_{i=1}^{N_\alpha}\delta(\mathbf{r}_{i\alpha}-\mathbf{r}'')\sum_{j=1\ (j\neq i\ \text{if}\ \alpha=\beta)}^{N_\beta}\delta(\mathbf{r}_{j\beta}-\mathbf{r}''')\,\delta(\mathbf{r}''-\mathbf{r}'''-\mathbf{r})\,d\mathbf{r}''\,d\mathbf{r}'''}{\int|\Psi|^2\,d\mathbf{R}\qquad n_\alpha(\mathbf{r}'')\,n_\beta(\mathbf{r}''')}\,d\mathbf{R} \qquad (448) \\ &= \frac{1}{\Omega}\int \frac{|\Psi|^2\sum_{i=1}^{N_\alpha}\sum_{j=1\ (j\neq i\ \text{if}\ \alpha=\beta)}^{N_\beta}\delta(\mathbf{r}_{i\alpha}-\mathbf{r}_{j\beta}-\mathbf{r})}{\int|\Psi|^2\,d\mathbf{R}\qquad n_\alpha(\mathbf{r}_{i\alpha})\,n_\beta(\mathbf{r}_{j\beta})}\,d\mathbf{R}. \qquad (449) \end{aligned}$$

The rotational average of $g_{\alpha\beta}(\mathbf{r})$ is

$$\begin{aligned} g_{\alpha\beta}(r) &= \frac{1}{4\pi r^2\Omega}\int g_{\alpha\beta}(\mathbf{r}')\,\delta(|\mathbf{r}'|-r)\,d\mathbf{r}' \qquad (450) \\ &= \frac{1}{4\pi r^2\Omega}\int \frac{|\Psi|^2\sum_{i=1}^{N_\alpha}\sum_{j=1\ (j\neq i\ \text{if}\ \alpha=\beta)}^{N_\beta}\delta(|\mathbf{r}_{i\alpha}-\mathbf{r}_{j\beta}|-r)}{\int|\Psi|^2\,d\mathbf{R}\qquad n_\alpha(\mathbf{r}_{i\alpha})\,n_\beta(\mathbf{r}_{j\beta})}\,d\mathbf{R}. \qquad (451) \end{aligned}$$

As well as the above averages one can calculate the pair correlation $g_{\alpha\beta}(\mathbf{r},\mathbf{r}')$ where an electron of spin α is fixed at a particular position $\mathbf{r}$. Suppose we write $g_{\alpha\beta}(\mathbf{r}',\mathbf{r}')$ as

$$\begin{aligned} g_{\alpha\beta}(\mathbf{r},\mathbf{r}') &= \frac{\int|\Psi|^2\sum_{i=1}^{N_\alpha}\delta(\mathbf{r}_{i\alpha}-\mathbf{r})\sum_{j=1\ (j\neq i\ \text{if}\ \alpha=\beta)}^{N_\beta}\delta(\mathbf{r}_{j\beta}-\mathbf{r}')\,d\mathbf{R}}{\int|\Psi|^2\sum_{i=1}^{N_\alpha}\delta(\mathbf{r}_{i\alpha}-\mathbf{r})\,d\mathbf{R}\,n_\beta(\mathbf{r}')} \qquad (452) \\ &= \frac{N_\alpha(N_\beta-\delta_{\alpha\beta})}{N_\alpha}\frac{\int|\Psi|^2\delta(\mathbf{r}_{1\alpha}-\mathbf{r})\delta(\mathbf{r}_{2\beta}-\mathbf{r}')\,d\mathbf{R}}{\int|\Psi|^2\delta(\mathbf{r}_{1\alpha}-\mathbf{r})\,d\mathbf{R}\quad n_\beta(\mathbf{r}')}. \qquad (453) \end{aligned}$$

We now define the probability distribution $|\Psi^{\mathrm{F}}|^2$ as

$$|\Psi^{\mathrm{F}}|^2 \quad = \quad \int|\Psi|^2\delta(\mathbf{r}_{1\alpha}-\mathbf{r})\,d\mathbf{r}_{1\alpha}. \tag{454}$$

Writing Eq. (452) in terms of $|\Psi^{\mathrm{F}}|^2$ we obtain

$$\begin{aligned} g_{\alpha\beta}(\mathbf{r},\mathbf{r}') &= (N_\beta-\delta_{\alpha\beta})\frac{\int|\Psi^{\mathrm{F}}|^2\delta(\mathbf{r}_{2\beta}-\mathbf{r}')\,d\mathbf{r}_2\ldots d\mathbf{r}_N}{\int|\Psi^{\mathrm{F}}|^2\,d\mathbf{r}_2\ldots d\mathbf{r}_N\quad n_\beta(\mathbf{r}')} \qquad (455) \\ &= \frac{n_{\alpha\beta}(\mathbf{r},\mathbf{r}')}{n_\beta(\mathbf{r}')}, \qquad (456) \end{aligned}$$

where $n_{\alpha\beta}(\mathbf{r},\mathbf{r}')$ is the charge density of the $N_\beta - \delta_{\alpha\beta}$ electrons of spin β calculated with an electron of spin α held fixed at $\mathbf{r}$. We can now average $g_{\alpha\beta}(\mathbf{r},\mathbf{r}')$ over the surface of a sphere of radius ρ centred on $\mathbf{r}$. The vector $\mathbf{r}'-\mathbf{r}$ can be written in spherical polar coordinates as (ρ,θ,ϕ). The required pair correlation function is then

$$g^{\mathrm{F}}_{\alpha\beta}(\mathbf{r},\rho) = \frac{1}{4\pi\rho^2}\int g_{\alpha\beta}(\mathbf{r},\mathbf{r}')\,\delta(|\mathbf{r}'-\mathbf{r}|-\rho)\,d\Omega, \tag{457}$$

where $d\Omega = \sin\theta\,d\theta\,d\phi$. This gives

$$\begin{aligned} g^{\mathrm{F}}_{\alpha\beta}(\mathbf{r},\rho) &= \frac{1}{4\pi\rho^2}\int g_{\alpha\beta}(\mathbf{r},\mathbf{r}')\,\delta(|\mathbf{r}'-\mathbf{r}|-\rho)\,d\Omega & (458)\\ &= \frac{1}{4\pi\rho^2}\frac{\int|\Psi^{\mathrm{F}}|^2\sum_{j=1}^{N_\beta-\delta_{\alpha\beta}}\delta(|\mathbf{r}_{j\beta}-\mathbf{r}'|-\rho)\,d\mathbf{r}_2\ldots d\mathbf{r}_N}{\int|\Psi^{\mathrm{F}}|^2\,d\mathbf{r}_2\ldots d\mathbf{r}_N \qquad n_\beta(\mathbf{r}')}\,d\Omega. & (459)\end{aligned}$$

This type of averaging was used by Maezono *et al.* [98]. The pair correlation functions can be evaluated in ρ bins for each $\mathbf{r}$ of interest. Equations (449), (451) and (459) contain one or two charge densities in the denominator. When one or both of these charge densities is very small there will be a large amount of noise in the quantity evaluated. We expect $g_{\alpha\beta}(\mathbf{r},\mathbf{r}')$ to tend to $1-\delta_{\alpha\beta}/N_\beta$ as $\mathbf{r}-\mathbf{r}'$ becomes large, while for small values of $\mathbf{r}-\mathbf{r}'$ we expect it be less than one. One strategy for coping with the noise would be to set a lower limit on the charge density $n_\beta(\mathbf{r}')$ below which the contribution to Eq. (459) is not accumulated.

35.3.5 Collecting in bins

Assume the system is homogeneous and isotropic. The pair correlation function $g_{\alpha\beta}(r)$ can be collected in bins of width Δ, giving

$$g_{\alpha\beta}(r_n) = \frac{\Omega}{\Omega_n}\left\langle\frac{N^n_{\alpha\beta}}{N_\alpha N_\beta}\right\rangle, \tag{460}$$

where $N^n_{\alpha\beta}$ is the number of α,β-spin pairs whose separation falls within the nth bin (the pairs $i\alpha, j\alpha$ and $j\alpha, i\alpha$ should be counted separately, and similarly for the parallel β-spins), and Ω_n is the volume of the nth bin. Note that if we consider a single bin, so that $\Omega=\Omega_n$, then $N^n_{\alpha\beta} = N_\alpha(N_\beta-\delta_{\alpha\beta})$. This confirms that the normalization of Eq. (460) is correct, but note that our formulae disagree with Eq. (35) of Ortiz and Ballone [99].

The volume of the nth bin, Ω_n, is given by

$$\Omega_n = \frac{4}{3}\pi\left[n^3-(n-1)^3\right]\Delta^3 = 4\pi\left(n^2-n+1/3\right)\Delta^3. \tag{461}$$

One could define r_n to be the centre of the nth bin, but a better choice is [100]

$$r_n = \frac{3\Delta}{4}\frac{\left[n^4-(n-1)^4\right]}{\left[n^3-(n-1)^3\right]} = \frac{3\Delta}{4}\frac{\left[4n^3-6n^2+4n-1\right]}{\left[3n^2-3n+1\right]}. \tag{462}$$

If g were to be linear across each bin and if g were sampled exactly without statistical error then Eqs. (460) and (461) would reproduce the exact value at each point r_n.

In two dimensions one requires the area of the nth circular strip,

$$A_n = \pi\left[n^2-(n-1)^2\right]\Delta^2 = \pi\Delta^2(2n-1). \tag{463}$$

We want to average $f(r)=ar+b$ over a strip, and find the corresponding radius r_n. Therefore

$$\begin{aligned} f(r_n) &= \frac{\int_{(n-1)\Delta}^{n\Delta} f(r)\,2\pi r\,dr}{\int_{(n-1)\Delta}^{n\Delta} 2\pi r\,dr}\\ &= \frac{\left[\frac{1}{3}ar^3+\frac{1}{2}br^2\right]_{(n-1)\Delta}^{n\Delta}}{\left[\frac{1}{2}r^2\right]_{(n-1)\Delta}^{n\Delta}}\\ &= a\frac{2[(n\Delta)^3-((n-1)\Delta)^3]}{3[(n\Delta)^2-((n-1)\Delta)^2]}+b\\ &= ar_n+b.\end{aligned}$$

Hence

$$r_n = \frac{2\Delta}{3}\frac{\left[n^3-(n-1)^3\right]}{\left[n^2-(n-1)^2\right]} = \frac{2\Delta}{3}\frac{[3n^2-3n+1]}{[2n-1]}. \qquad (464)$$

35.4 Structure factor and spherically averaged structure factor

*K*eywords: **structure_factor**, **struc_factor_sph**

One may define the following correlation function,

$$S'_{\alpha\beta}(\mathbf{r},t;\mathbf{r}',t') \;=\; \langle\Psi|\rho_\alpha(\mathbf{r},t)_H\,\rho_\beta(\mathbf{r}',t')_H|\Psi\rangle, \qquad (465)$$

where ρ is the density operator, and the subscript H denotes that we are using the Heisenberg picture. In fact one normally considers a modified correlation function

$$S_{\alpha\beta}(\mathbf{r},t;\mathbf{r}',t') \;=\; \langle\Psi|\rho_\alpha(\mathbf{r},t)_H\,\rho_\beta(\mathbf{r}',t')_H|\Psi\rangle - n_\alpha(\mathbf{r})n_\beta(\mathbf{r}'), \qquad (466)$$

because S tends to zero as $|t-t'|\to\infty$, so that the Fourier transform of S exists in the time domain. S is known as the structure factor. We are mostly interested in the static structure factor, which is Eq. (466) evaluated at equal times,

$$S_{\alpha\beta}(\mathbf{r},\mathbf{r}') \;=\; \langle\Psi|\rho_\alpha(\mathbf{r})_H\,\rho_\beta(\mathbf{r}')_H|\Psi\rangle - n_\alpha(\mathbf{r})n_\beta(\mathbf{r}'), \qquad (467)$$

which has the form of a covariance. The covariance might show a lower variance than the individual terms in Eq. (467), so it might be better to evaluate them together. Note that as $|\mathbf{r}-\mathbf{r}'|\to\infty$ we expect the electrons to be uncorrelated, so that in this limit $S_{\alpha\beta}(\mathbf{r},\mathbf{r}')\to 0$. The static structure factor can be written as

$$S_{\alpha\beta}(\mathbf{r},\mathbf{r}') \;=\; \frac{\int|\Psi|^2\sum_{i=1}^{N_\alpha}\delta(\mathbf{r}_{i\alpha}-\mathbf{r})\sum_{j=1}^{N_\beta}\delta(\mathbf{r}_{j\beta}-\mathbf{r}')\,d\mathbf{R}}{\int|\Psi|^2\,d\mathbf{R}} - n_\alpha(\mathbf{r})n_\beta(\mathbf{r}'). \qquad (468)$$

35.4.1 Relationship between $S_{\alpha\beta}$ and $g_{\alpha\beta}$

The static structure factor is related to the pair correlation function by

$$S_{\alpha\beta}(\mathbf{r},\mathbf{r}') \;=\; n_\alpha(\mathbf{r})n_\beta(\mathbf{r}')\,[g_{\alpha\beta}(\mathbf{r},\mathbf{r}')-1] + n_\alpha(\mathbf{r})\delta_{\alpha\beta}\,\delta(\mathbf{r}-\mathbf{r}'), \qquad (469)$$

see Dreizler and Gross, page 276. Using Eq. (441) we find that $S_{\alpha\beta}(\mathbf{r},\mathbf{r}')$ satisfies

$$\int S_{\alpha\beta}(\mathbf{r},\mathbf{r}')\,d\mathbf{r}\,d\mathbf{r}' = 0. \qquad (470)$$

The Fourier transform of $S_{\alpha\beta}$ is

$$\tilde{S}_{\alpha\beta}(\mathbf{k},\mathbf{k}') \;=\; \frac{1}{\Omega^2}\int S_{\alpha\beta}(\mathbf{r},\mathbf{r}')\,e^{i\mathbf{k}\cdot\mathbf{r}}e^{i\mathbf{k}'\cdot\mathbf{r}'}\,d\mathbf{r}\,d\mathbf{r}' \qquad (471)$$

$$=\; \langle\tilde{\rho}_\alpha(\mathbf{k})\tilde{\rho}_\beta(\mathbf{k}')\rangle - \tilde{n}_\alpha(\mathbf{k})\tilde{n}_\beta(\mathbf{k}'). \qquad (472)$$

The translational average of $S_{\alpha\beta}(\mathbf{r},\mathbf{r}')$ is

$$S_{\alpha\beta}(\mathbf{r}) \;=\; \frac{1}{\Omega}\int S_{\alpha\beta}(\mathbf{r}'',\mathbf{r}''')\,\delta(\mathbf{r}''-\mathbf{r}'''-\mathbf{r})\,d\mathbf{r}''d\mathbf{r}''' \qquad (473)$$

$$=\; \frac{1}{\Omega}\frac{\int|\Psi|^2\sum_{i=1}^{N_\alpha}\sum_{j=1}^{N_\beta}\delta(\mathbf{r}_{i\alpha}-\mathbf{r}_{j\beta}-\mathbf{r})\,d\mathbf{R}}{\int|\Psi|^2\,d\mathbf{R}} - \frac{1}{\Omega}\int n_\alpha(\mathbf{r}')n_\beta(\mathbf{r}'-\mathbf{r})\,d\mathbf{r}'. \qquad (474)$$

The first term in Eq. (474) is the translational average of $n_\alpha(\mathbf{r})n_\beta(\mathbf{r}')g_{\alpha\beta}(\mathbf{r},\mathbf{r}') + n_\alpha(\mathbf{r})\delta_{\alpha\beta}\,\delta(\mathbf{r}-\mathbf{r}')$, while the second is the translational average of $-n_\alpha(\mathbf{r})n_\beta(\mathbf{r}')$. $S_{\alpha\beta}(\mathbf{r})$ satisfies

$$\int S_{\alpha\beta}(\mathbf{r})\,d\mathbf{r} = 0. \qquad (475)$$

The Fourier transform of $S_{\alpha\beta}(\mathbf{r})$ is

$$\begin{aligned}\tilde{S}_{\alpha\beta}(\mathbf{k}) &= \frac{1}{\Omega}\int S_{\alpha\beta}(\mathbf{r})\, e^{i\mathbf{k}\cdot\mathbf{r}}\, d\mathbf{r} && (476)\\ &= \frac{1}{\Omega^2}\langle\tilde{\rho}_\alpha(\mathbf{k})\tilde{\rho}_\beta(-\mathbf{k})\rangle - \frac{1}{\Omega^2}\tilde{n}_\alpha(\mathbf{k})\tilde{n}_\beta(-\mathbf{k}) && (477)\\ &= \frac{1}{\Omega^2}\tilde{S}_{\alpha\beta}(\mathbf{k},-\mathbf{k}), && (478)\end{aligned}$$

as expected from Eq. (393). Using Eqs. (472) and (478) one can show that $\tilde{S}_{\alpha\beta}(\mathbf{k}=0,\mathbf{k}'=0)=0$ and $\tilde{S}_{\alpha\beta}(\mathbf{k}=0)=0$.

The rotational average of $S_{\alpha\beta}(\mathbf{r})$ is

$$\begin{aligned}S_{\alpha\beta}(r) &= \frac{1}{4\pi r^2\Omega}\int S_{\alpha\beta}(\mathbf{r}')\,\delta(|\mathbf{r}'|-r)\,d\mathbf{r}' && (479)\\ &= \frac{1}{4\pi r^2\Omega}\frac{\int|\Psi|^2\sum_{i=1}^{N_\alpha}\sum_{j=1}^{N_\beta}\delta(|\mathbf{r}_{i\alpha}-\mathbf{r}_{j\beta}|-r)\,d\mathbf{R}}{\int|\Psi|^2\,d\mathbf{R}} && (480)\\ &\quad -\frac{1}{4\pi r^2\Omega}\int n_\alpha(\mathbf{r}'')n_\beta(\mathbf{r}''-\mathbf{r}')\,\delta(|\mathbf{r}'|-r)\,d\mathbf{r}'d\mathbf{r}''. && (481)\end{aligned}$$

The Fourier transform of $S_{\alpha\beta}(r)$ is

$$\begin{aligned}S_{\alpha\beta}(k) &= \frac{1}{\Omega^2}\frac{\int|\Psi|^2\sum_{i=1}^{N_\alpha}\sum_{j=1}^{N_\beta}\frac{\sin(k|\mathbf{r}_{i\alpha}-\mathbf{r}_{j\beta}|)}{k|\mathbf{r}_{i\alpha}-\mathbf{r}_{j\beta}|}\,d\mathbf{R}}{\int|\Psi|^2\,d\mathbf{R}} && (482)\\ &\quad -\frac{4\pi}{\Omega}\sum_{\mathbf{G}}\frac{1}{kG}\,\tilde{n}_\alpha(\mathbf{G})\,\tilde{n}_\beta(-\mathbf{G})\int_0^\infty \sin(kr)\sin(Gr)\,dr. && (483)\end{aligned}$$

Unfortunately the integral in the second term is undefined. However, we can use the argument that, for large enough $\mathbf{r}$, $S_{\alpha\beta}(\mathbf{r})\to 0$ because the effects of exchange and correlation will tend to zero. The Fourier transforms of $S_{\alpha\beta}(\mathbf{r})$ and $S_{\alpha\beta}(r)$ are therefore well defined. The contributions from the first and second terms in Eq. (481) must therefore cancel at large distances. In practise we will include only pairs of particles whose separation is within the radius of the sphere inscribed in the WS cell of the simulation cell, $L_{\rm WS}$. We can therefore include only pairs of particles whose separation is less than $L_{\rm WS}$ and set the upper limit on the integral to $L_{\rm WS}$, in which case it can be evaluated,

$$\int_0^{L_{\rm WS}}\sin(kr)\sin(Gr)\,dr = \frac{1}{2}\left[\frac{\sin(k-G)L_{\rm WS}}{k-G}-\frac{\sin(k+G)L_{\rm WS}}{k+G}\right]. \quad (484)$$

For $k-G$ small or $k+G$ small the relevant sin function should be expanded, as described by Rene Gaudoin. In this method pairs of electrons in the 'corners' of the WS cell whose separation is larger than $L_{\rm WS}$ are not included so that $S_{\alpha\beta}(k=0)\neq 0$. This would need to be corrected afterwards. The structure factor $S_{\alpha\beta}(k)$ should satisfy

$$S_{\alpha\beta}(k\to\infty)\to\delta_{\alpha\beta}. \quad (485)$$

35.4.2 Homogeneous and isotropic systems

For a homogeneous and isotropic system, we have from Eq. (469),

$$S^{\rm H}_{\alpha\beta}(\mathbf{r}) = \frac{N_\alpha}{\Omega}\frac{N_\beta}{\Omega}\left[g_{\alpha\beta}(\mathbf{r})-1\right]+\delta_{\alpha\beta}\,\delta(\mathbf{r})\frac{N_\alpha}{\Omega}. \quad (486)$$

The Fourier transform of $S^{\rm H}_{\alpha\beta}(\mathbf{r})$ is

$$\begin{aligned}S^{\rm H}_{\alpha\beta}(\mathbf{k}) &= \frac{1}{\Omega}\int\left[\frac{N_\alpha}{\Omega}\frac{N_\beta}{\Omega}\left[g_{\alpha\beta}(\mathbf{r})-1\right]+\delta_{\alpha\beta}\,\delta(\mathbf{r})\frac{N_\alpha}{\Omega}\right]e^{i\mathbf{k}\cdot\mathbf{r}}\,d\mathbf{r} && (487)\\ &= \frac{1}{\Omega}\int\frac{N_\alpha}{\Omega}\frac{N_\beta}{\Omega}\left[g_{\alpha\beta}(\mathbf{r})-1\right]e^{i\mathbf{k}\cdot\mathbf{r}}\,d\mathbf{r}+\delta_{\alpha\beta}\frac{N_\alpha}{\Omega^2}. && (488)\end{aligned}$$

As the system is homogeneous and isotropic $S^{\rm H}_{\alpha\beta}(\mathbf{k})$ is a function of k only, and we have

$$S^{\rm H}_{\alpha\beta}(k) = \frac{1}{\Omega}\int\frac{N_\alpha}{\Omega}\frac{N_\beta}{\Omega}\left[g_{\alpha\beta}(r)-1\right]4\pi r^2\frac{\sin(kr)}{kr}\,dr+\delta_{\alpha\beta}\frac{N_\alpha}{\Omega^2}. \quad (489)$$

Using the sum rules of Eq. (446) we find that

$$S^{\mathrm{H}}_{\alpha\beta}(k=0) \quad = \quad 0 \tag{490}$$

$$S^{\mathrm{H}}_{\alpha\beta}(k\to\infty) \quad \to \quad \delta_{\alpha\beta}\frac{N_\alpha}{\Omega^2}. \tag{491}$$

35.5 One-body density matrix, two-body density matrix and condensate fraction

*K*eywords: **onep_density_mat**, **twop_density_mat**, **cond_fraction**

Density matrices are important tools to determine properties of systems, such as the phase of an electron–hole system.

35.5.1 Definition of the density matrices

The one-body density matrix (OBDM) is defined as

$$\rho^{(1)}_\alpha(\mathbf{r}_1;\mathbf{r}'_1) = N_\alpha \frac{\int |\Psi(\mathbf{R})|^2 \frac{\Psi(\mathbf{r}'_1)}{\Psi(\mathbf{r}_1)} d\mathbf{r}_2 \ldots d\mathbf{r}_N}{\int |\Psi(\mathbf{R})|^2 \, d\mathbf{R}} \; , \tag{492}$$

and the two-body density matrix (TBDM) is

$$\rho^{(2)}_{\alpha\beta}(\mathbf{r}_1,\mathbf{r}_2;\mathbf{r}'_1,\mathbf{r}'_2) = N_\alpha(N_\beta-\delta_{\alpha\beta}) \frac{\int |\Psi(\mathbf{R})|^2 \frac{\Psi(\mathbf{r}'_1,\mathbf{r}'_2)}{\Psi(\mathbf{r}_1,\mathbf{r}_2)} d\mathbf{r}_3 \ldots d\mathbf{r}_N}{\int |\Psi(\mathbf{R})|^2 \, d\mathbf{R}} \; , \tag{493}$$

where α and β are the spin indices (or particle indices, in general) corresponding to $\mathbf{r}_1$ and $\mathbf{r}_2$, respectively, and the irrelevant dependencies of the wave function have been omitted (i.e., $\Psi(\mathbf{r}'_1,\mathbf{r}'_2) \equiv \Psi(\mathbf{r}'_1,\mathbf{r}'_2,\mathbf{r}_3,\ldots,\mathbf{r}_N)$). Normalization is such that

$$\sum_\alpha \int \rho^{(1)}_\alpha(\mathbf{r}_1;\mathbf{r}'_1)\delta(\mathbf{r}_1-\mathbf{r}'_1)d\mathbf{r}_1 d\mathbf{r}'_1 = N \; , \tag{494}$$

and

$$\sum_{\alpha\beta} \int \rho^{(2)}_{\alpha\beta}(\mathbf{r}_1,\mathbf{r}_2;\mathbf{r}'_1,\mathbf{r}'_2)\delta(\mathbf{r}_1-\mathbf{r}'_1)\delta(\mathbf{r}_2-\mathbf{r}'_2)d\mathbf{r}_1 d\mathbf{r}_2 d\mathbf{r}'_1 d\mathbf{r}'_2 = N(N-1) \; . \tag{495}$$

We are interested in the QMC accumulation of the density matrices for the case of an isotropic, homogeneous system. Hence we require the translational and rotational average of Eqs. (492) and 493. The translational average is

$$\rho^{(1)}_\alpha(\mathbf{r}') = \frac{N_\alpha}{\Omega} \frac{\int |\Psi(\mathbf{R})|^2 \frac{\Psi(\mathbf{r}_1+\mathbf{r}')}{\Psi(\mathbf{r}_1)} d\mathbf{R}}{\int |\Psi(\mathbf{R})|^2 \, d\mathbf{R}} \; , \tag{496}$$

and

$$\rho^{(2)}_{\alpha\beta}(\mathbf{r}') = \frac{N_\alpha(N_\beta-\delta_{\alpha\beta})}{\Omega^2} \frac{\int |\Psi(\mathbf{R})|^2 \frac{\Psi(\mathbf{r}_1+\mathbf{r}',\mathbf{r}_2+\mathbf{r}')}{\Psi(\mathbf{r}_1,\mathbf{r}_2)} d\mathbf{R}}{\int |\Psi(\mathbf{R})|^2 \, d\mathbf{R}} \; , \tag{497}$$

where Ω is the volume of the simulation cell. The rotational average of Eqs. (496) and (497) is

$$\rho^{(1)}_\alpha(r) = \frac{N_\alpha}{\Omega S(r)} \frac{\int |\Psi(\mathbf{R})|^2 \frac{\Psi(\mathbf{r}_1+\mathbf{r}')}{\Psi(\mathbf{r}_1)} \delta(|\mathbf{r}'|-r) d\mathbf{R} d\mathbf{r}'}{\int |\Psi(\mathbf{R})|^2 \, d\mathbf{R}} \; , \tag{498}$$

and

$$\rho^{(2)}_{\alpha\beta}(r) = \frac{N_\alpha(N_\beta-\delta_{\alpha\beta})}{\Omega^2 S(r)} \frac{\int |\Psi(\mathbf{R})|^2 \frac{\Psi(\mathbf{r}_1+\mathbf{r}',\mathbf{r}_2+\mathbf{r}')}{\Psi(\mathbf{r}_1,\mathbf{r}_2)} \delta(|\mathbf{r}'|-r) d\mathbf{R} d\mathbf{r}'}{\int |\Psi(\mathbf{R})|^2 \, d\mathbf{R}} \; , \tag{499}$$

where $S(r)$ is $4\pi r^2$ in 3D, $2\pi r$ in 2D and 1 in 1D.

35.5.2 QMC accumulation

The integral over $\mathbf{R}$ in Eqs. (496) and 497 can be approximated by a Monte Carlo average, using

$$\frac{\int |\Psi(\mathbf{R})|^2 f(\mathbf{R}) d\mathbf{R}}{\int |\Psi(\mathbf{R})|^2 d\mathbf{R}} = \frac{1}{M} \sum_{i=1}^{M} f(\mathbf{R}_i) \ , \tag{500}$$

where $\{\mathbf{R}_i\}_{i=1}^{i=M}$ is a set of configurations distributed according to $|\Psi(\mathbf{R})|^2$.

The estimate of the value of the rotational average of a function $f(\mathbf{r})$ in the bth bin Ω_b, is

$$f_b = \frac{\int_{r_{b-1}}^{r_b} \left[\int f(\mathbf{r}')\delta(|\mathbf{r}'| - r) d\mathbf{r}'\right] dr}{\int_{r_{b-1}}^{r_b} S(r) dr} = \frac{\int_{\Omega_b} f(\mathbf{r}') d\mathbf{r}'}{\int_{\Omega_b} d\mathbf{r}'} = \frac{1}{m_b} \sum_{i=1}^{m_b} f(\mathbf{r}'_i) \ , \tag{501}$$

where $\{\mathbf{r}'_i\}$ is a set of points uniformly distributed in the simulation cell, and m_b is the number of such points falling into the bth bin.

Hence the estimate of the translational-rotational average of the OBDM is

$$\rho^{(1)}_{\alpha}(r_b) \approx \frac{N_\alpha}{\Omega M m_b} \sum_{i=1}^{M} \sum_{j=1}^{m_b} \frac{\Psi(\mathbf{r}^i_1 + \mathbf{r}'_j)}{\Psi(\mathbf{r}^i_1)} \ , \tag{502}$$

and for the TBDM we have

$$\rho^{(2)}_{\alpha\beta}(r_b) \approx \frac{N_\alpha(N_\beta - \delta_{\alpha\beta})}{\Omega^2 M m_b} \sum_{i=1}^{M} \sum_{j=1}^{m_b} \frac{\Psi(\mathbf{r}^i_1 + \mathbf{r}'_j, \mathbf{r}^i_2 + \mathbf{r}'_j)}{\Psi(\mathbf{r}^i_1, \mathbf{r}^i_2)} \ . \tag{503}$$

In order to improve the statistics, one can take advantage of the antisymmetry of the wave function.

35.5.3 Condensate fraction

A well-known limit of the fermionic TBDM for $\alpha \neq \beta$ and $N_\alpha = N_\beta$ is [101, 102]

$$\lim_{|\mathbf{r}| \to \infty} \rho^{(2)}_{\alpha\beta}(\mathbf{r}_1, \mathbf{r}_2; \mathbf{r}_1 + \mathbf{r}, \mathbf{r}_2 + \mathbf{r}) = c N_\alpha |\varphi(|\mathbf{r}_2 - \mathbf{r}_1|)|^2 \ , \tag{504}$$

where c is the condensate fraction and $\varphi(r)$ is a function such that $\int |\varphi(r)|^2 d\mathbf{r} = 1/\Omega$. The OBDM is zero in this limit.

Applying the limit to Eq. (499) and substituting, we can estimate c as

$$c = \frac{\Omega^2}{N_\alpha} \lim_{r \to \infty} \rho^{(2)}_{\alpha\beta}(r) \ . \tag{505}$$

35.5.4 Improved estimators

Consider a system of two distinguishable particles with wave function $\Psi(\mathbf{r}_1, \mathbf{r}_2) = \phi(\mathbf{r}_1)\phi(\mathbf{r}_2)$. The two particles are independent from one another. Equations (492) and 493 can combined, giving

$$\rho^{(2)}_{\alpha\beta}(\mathbf{r}_1, \mathbf{r}_2; \mathbf{r}_1 + \mathbf{r}', \mathbf{r}_2 + \mathbf{r}') = \rho^{(1)}_{\alpha}(\mathbf{r}_1; \mathbf{r}_1 + \mathbf{r}')\rho^{(1)}_{\beta}(\mathbf{r}_2, \mathbf{r}_2 + \mathbf{r}') \ , \tag{506}$$

from which we can see that in the independent-particle case the TBDM does nothing but reflect the one-body properties of the system.

An opposite example is a system of two completely paired particles with wave function $\Psi(\mathbf{r}_1, \mathbf{r}_2) = \delta_\epsilon(\mathbf{r}_1 - \mathbf{r}_2)$, where $\delta_\epsilon(\mathbf{r})$ is a localized function of range ϵ that tends to the Dirac delta as $\epsilon \to 0$. In this case, the OBDM is zero for all $|\mathbf{r}'| > \epsilon$, whereas the TBDM is

$$\rho^{(2)}_{\alpha\beta}(\mathbf{r}_1, \mathbf{r}_2; \mathbf{r}_1 + \mathbf{r}', \mathbf{r}_2 + \mathbf{r}') = \frac{|\Psi(\mathbf{r}_1, \mathbf{r}_2)|^2}{\int |\Psi(\mathbf{r}_1, \mathbf{r}_2)|^2 d\mathbf{r}_1 d\mathbf{r}_2} \ . \tag{507}$$

In this case the value of the TBDM is completely due to two-body effects.

To improve the estimation of the condensate fraction c it is necessary to eliminate the one-body effects of the form of Eq. (506), while keeping the pure two-body effects of Eq. (507) intact. One such approach [103] is to subtract $\rho_\alpha^{(1)}(\mathbf{r}_1;\mathbf{r}_1+\mathbf{r}')\rho_\beta^{(1)}(\mathbf{r}_2,\mathbf{r}_2+\mathbf{r}')$ from $\rho_{\alpha\beta}^{(2)}(\mathbf{r}_1,\mathbf{r}_2;\mathbf{r}_1+\mathbf{r}',\mathbf{r}_2+\mathbf{r}')$. It is clear from Eq. (506) that this removes one-body effects, and does not bias the value of c because the OBDM is zero when Eq. (504) holds. The condensate fraction could then be estimated as

$$c = \frac{\Omega^2}{N_\alpha} \lim_{r\to\infty} \left\{ \rho_{\alpha\beta}^{(2)}(r) - \left[\rho_\alpha^{(1)}(r)\rho_\beta^{(1)}(r) \right] \right\} , \tag{508}$$

where the approximation $\left[\rho_\alpha^{(1)}(\mathbf{r})\rho_\beta^{(1)}(\mathbf{r})\right]^R \approx \rho_\alpha^{(1)}(r)\rho_\beta^{(1)}(r)$ has been used.

Another method, proposed here, is to define a modified TBDM,

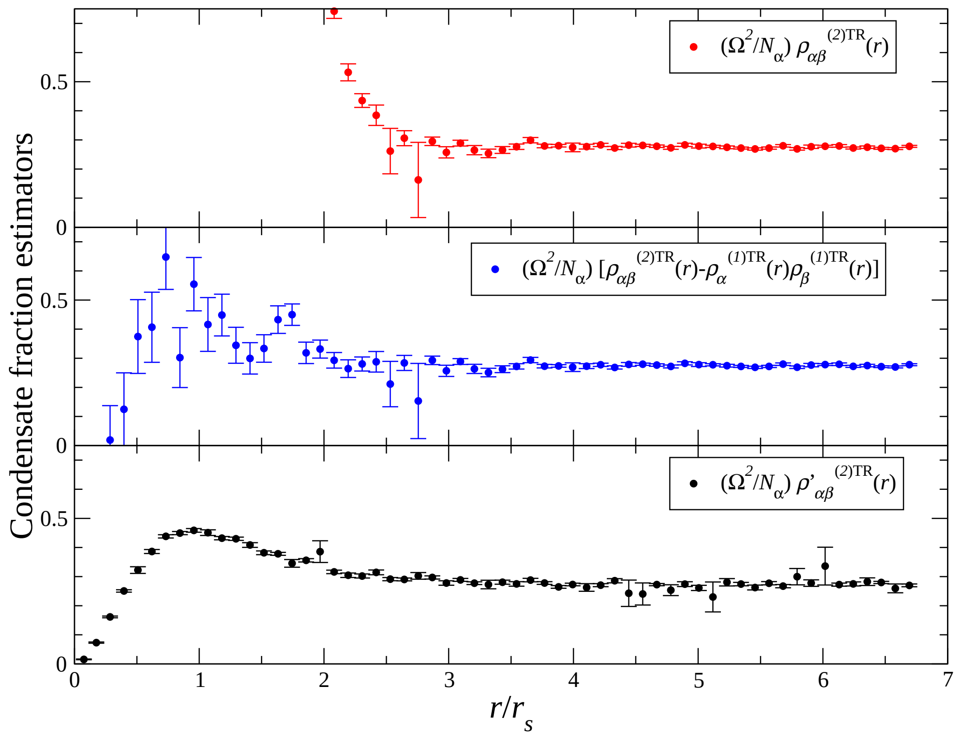
$$\tilde{\rho}_{\alpha\beta}^{(2)}(\mathbf{r}_1,\mathbf{r}_2;\mathbf{r}_1',\mathbf{r}_2') = N_\alpha(N_\beta-\delta_{\alpha\beta}) \frac{\int |\Psi(\mathbf{r}_1,\mathbf{r}_2)|^2 \left[\frac{\Psi(\mathbf{r}_1',\mathbf{r}_2')}{\Psi(\mathbf{r}_1,\mathbf{r}_2)} - \frac{\Psi(\mathbf{r}_1',\mathbf{r}_2)}{\Psi(\mathbf{r}_1,\mathbf{r}_2)} \frac{\Psi(\mathbf{r}_1,\mathbf{r}_2')}{\Psi(\mathbf{r}_1,\mathbf{r}_2)} \right] d\mathbf{r}_3 \ldots d\mathbf{r}_N}{\int |\Psi(\mathbf{R})|^2 \, d\mathbf{R}} , \tag{509}$$

and compute the condensate fraction as

$$c = \frac{\Omega^2}{N_\alpha} \lim_{r\to\infty} \tilde{\rho}_{\alpha\beta}^{(2)}(r) , \tag{510}$$

which also achieves the same purposes, but benefits from correlated sampling and is somewhat cheaper to evaluate, as the wave function updates required for the OBDM can be re-used in the evaluation of the TBDM.

The three condensate fraction estimators have been computed for a two-dimensional electron–hole bilayer ($r_s = 5$, $d = 1$, $N_e = N_h = 58$), and are represented in the figure below. From top to bottom, the TBDM estimator [Eq. (505)], the TBDM-OBDM estimator [Eq. (508)], and the modified-TBDM estimator [Eq. (510)]. The advantages of Eq. (510) are evident in the short-range region, while the long-range region seems to display a slightly noisier behaviour than the other two.



35.5.5 Speed issues

The condensate fraction is very expensive to calculate. The accumulation parameters can be tuned to reduce the cost of the run. Unfortunately due to the design of the `expval.data` file format the process is somewhat fiddly.

You will need to:

- Starting from an existing `expval.data` file, delete the CONDENSATE FRACTION block and replace it with:

  ```
  START CONDENSATE FRACTION
  Accumulation carried out using
   DMC
  Number of sets
   1
  Number of bins
   100
  Number of random points to sample
   20
  Fraction of particle pairs to sample at random
   0.0500
  START SET 1
  Number of particle-pair types in set
   4
  Particle-pair types
   1 3  2 3  1 4  2 4
  Weight(r),CF(r)**2,CF(r)
  END SET 1
  END CONDENSATE FRACTION
  ```

  This will only accumulate the electron-hole expectation value and ignore the electron-electron and hole-hole components which are computed by default.

- In the above we have modified the '`Fraction of particle pairs`' parameter. By default, the code goes over all 58^2 (=3364) electron-hole pairs, but it can go over a random sample of them instead. With the above '`Fraction of particle pairs`' value of 0.05, the code will sample a random 5% of the 58^2 electron-hole pairs (=168) at each step, reducing the cost of the evaluation per DMC step by a factor of 20. This will also affect the statistics of the result, of course, but overall it is advantageous to use a value much less than 1.0 here.

- If you want, modify the '`Number of random points`' parameter. This controls the number of random electron-hole pair displacements sampled for each electron-hole pair at each step. In our opinion the default value of 20 is generally sensible.

- If you want more or less resolution in the final plot, change the '`Number of bins`' from 100 to whichever value you'd like. The finer the grid, the poorer the statistics. In my opinion the default value of 100 is generally sensible.

- If you are accumulating any other expectation values in the `expval.data` file, make sure you delete the data lines in all sets, as above, to start the accumulation afresh.

- Make a copy of `expval.data` in case you need to start over, since `expval.data` is overwritten when CASINO is run.

- Now run your calculation.

Note that this assumes that you are accumulating the **cond_fraction** estimator, which is a non-standard estimator with clear statistical advantages near the origin, but tends to exhibit outliers in the tail (the important bit to compute the condensate fraction) and has never been used in the literature. It is also possible to accumulate the standard estimator by toggling the **onep_density_mat** and **twop_density_mat** input keywords. If you choose to plot the two-body density matrix in the `plot_expval` utility, it will offer to subtract the one-particle density matrix, giving the standard estimator of the condensate fraction. You may want to modify the default accumulation parameters for these expectation values as above, giving:

```
START ONE-PARTICLE DENSITY MATRIX
Accumulation carried out using
 DMC
Number of sets
 2
Number of bins
```

```
 100
Number of random points to sample
 20
START SET 1
Number of particle types in set
 2
Particle types
 1 2
Weight(r),OBDM(r)**2,OBDM(r)
END SET 1
START SET 2
Number of particle types in set
 2
Particle types
 3 4
Weight(r),OBDM(r)**2,OBDM(r)
END SET 2
END ONE-PARTICLE DENSITY MATRIX

START TWO-PARTICLE DENSITY MATRIX
Accumulation carried out using
 DMC
Number of sets
 1
Number of bins
 100
Number of random points to sample
 20
Fraction of particle pairs to sample at random
 0.0500
START SET 1
Number of particle-pair types in set
 4
Particle-pair types
 1 3  2 3  1 4  2 4
Weight(r),TBDM(r)**2,TBDM(r)
END SET 1
END TWO-PARTICLE DENSITY MATRIX
```

35.6 One- and two-body momentum densities

*K*eywords: **mom_den**, **twop_dm_mom**, **cond_fraction_mom**

The one-body and two-body momentum densities are the Fourier transforms of the one- and two-body density matrices, respectively, and are explicitly calculated as such. Note that it is not a good idea to Fourier-transform the spherical real-space version of these expectation values described above, because they ignore the "corners" of the simulation cell, biasing the small-k limit of the Fourier transforms. The momentum-space version of these expectation values does not have this problem.

The k-vectors of the transformation are the reciprocal-lattice vectors of the simulation cell. Notice that for a homogeneous system these vectors are affected by keyword **k_offset**, hence running different accumulations with different **k_offset** values allows evaluating the momentum density on an arbitrarily fine grid.

35.7 Localization tensor

*K*eyword: **loc_tensor**

An early success of quantum mechanics was to explain the distinction between metal and non-metal using band theory. The system is metallic if the conduction and valence band overlap and more than one band is partly filled, while in non-metallic systems all the bands are fully occupied or empty. Exceptions arise in some systems in which the band is partially filled but the Coulomb interaction between the electrons is sufficiently strong that the electrons are localized. An alternative view to band theory would help to distinguish the character of the system. It is argued by Kohn [104] that the system is insulating as a result of wave function localization in the configuration space. The development of the Berry-phase theory of polarization [105, 106, 107, 108, 109] further advanced Kohn's idea and provided tools to measure the localization of the electrons and the polarization. This Berry-phase approach solves the problem that the polarization is ill-defined in an extended system by computing the quantity directly from the many-body wave function instead of the electron positions. Souza and Martin [108] provided the expressions for the localization tensor and the polarization in terms of a many-body Wannier wave function. This wave function is linked to Kohn's wave function: for an insulating system it is localized in configuration space in the thermodynamic limit. Souza and Martin [108] also showed that the localization tensor is related to a frequency integral of the conductivity. The conductivity formula implies that for a system with non-vanishing conductivity the localization tensor is infinite, otherwise it has a finite value. In [110], the off-diagonal elements of the localization tensor are used to calculate the DC conductivity in the transverse direction for a quantum hall fluid.

The localization tensor and polarization are written in terms of a many-body operator

$$Z_N^{(\alpha)} = \left\langle \Psi | e^{i\mathbf{G}_\alpha \cdot \mathbf{X}(\mathbf{R_m})} | \Psi \right\rangle \ , \tag{511}$$

where G_α is a simulation cell reciprocal lattice vector and X is the sum of electron positions $\sum_{i=1}^{n} r_i$ of a configuration R_m.

The Berry-phase polarization is given by

$$P_\alpha = \frac{N}{V} \left\langle r_\alpha \right\rangle_{\mathrm{c}} \ , \tag{512}$$

where $\left\langle r_\alpha \right\rangle_{\mathrm{c}}$ is the expectation value of the electron distribution given by

$$\left\langle r_\alpha \right\rangle_{\mathrm{c}} = \frac{1}{NG_\alpha} \Im \log Z_N^{(\alpha)} \ . \tag{513}$$

Equation (512) measures the polarization current of the system in response to an adiabatic change of the Hamiltonian by approximating the Coulomb interaction as a first-order perturbation.

The localization tensor can be interpreted as a measure of the quadratic spread of a charge distribution. This gives an indication of how well the electrons are localized in the simulation cell according to the wave function. This is written as

$$\left\langle r_\alpha^2 \right\rangle_{\mathrm{c}} = \frac{-1}{NG_\alpha^2} \log |Z_N^{(\alpha)}|^2 \ , \tag{514}$$

where N is the number of electrons in the simulation cell.

The off-diagonal elements of the localization tensor are defined by

$$\left\langle r_\alpha r_\beta \right\rangle_{\mathrm{c}} = \frac{-1}{NG_\alpha G_\beta} \log \frac{|Z_N^{(\alpha)}||Z_N^{(\beta)}|}{|Z_N^{(\alpha\beta)}|} \ , \tag{515}$$

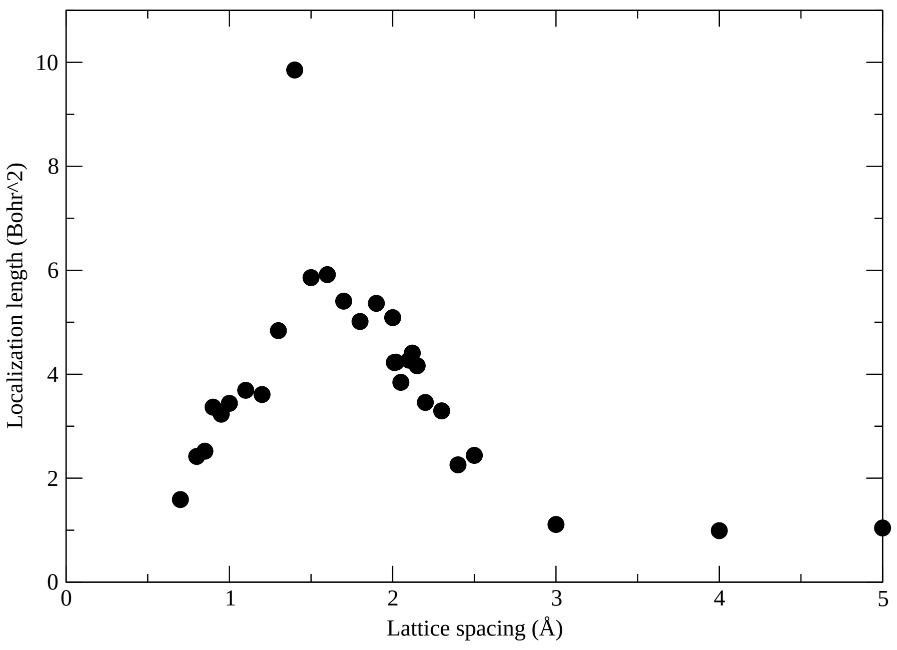
where $Z_N^{(\alpha\beta)}$ is defined as $\left\langle \Psi | e^{-i(\mathbf{G}_\alpha - \mathbf{G}_\beta)\cdot \mathbf{X}(\mathbf{R_m})} | \Psi \right\rangle$.

In QMC, the localization tensor and polarization are calculated from Z_N with the periodic boundary conditions imposed. This is done by summing the electron positions of each configuration R_m to calculate $e^{-i\mathbf{G}_\alpha \cdot \mathbf{X}(\mathbf{R_m})}$. This is then averaged over configurations generated by VMC or DMC to give mean

$$\bar{z}_N = \frac{1}{M} \sum_{m=1}^{M} Z_N(R_m) \ , \tag{516}$$

to ensure (516) tends to (511) and the statistical error is small, a large number of steps must be taken to sample the configuration. The localization tensor diverges when Z_N is zero. Numerically, Z_N would never be zero, but could become small so that the localization tensor becomes very large when approaching metal-insulator transition from the insulating side [111]. As the difference between $\bar{z}$ and $Z_N(R_m)$ becomes large when approaching the divergence, it is often useful to examine the error bar in Z_N to determine when the localization tensor diverges.

The figure shows the localization length of a linear chain of antiferromagnetic hydrogen atoms as a function of lattice spacing. The localization length has a maximum at 1.4 Å, indicating that the chain becomes metallic at around this lattice spacing.



35.8 Dipole moment (molecules only)

*K*eyword: **dipole_moment**

The electric dipole moment of a finite set of point charges of charge q_i and position vector $\mathbf{r}_i$ is defined to be

$$\mathbf{p} = \sum_i q_i \mathbf{r}_i . \tag{517}$$

It is easy to show that the value of **p** is independent of the origin if the overall system is electrically neutral.

We may use QMC to estimate the mean dipole moment of a molecule by averaging **p** over the set of configurations generated by the VMC or DMC algorithms. The mean value of $|\mathbf{p}|^2$ may be determined in a similar fashion. This will be done if the **dipole_moment** keyword is set to `T` in the `input` file. The raw data will be written to the `vmc.hist` or `dmc.hist` file (not the `expval.data` file!) and the error bars on components of the dipole moment should be evaluated using the `reblock` utility.

Note that the CASINO `reblock` utility reports only the components and not the magnitude of the dipole moment in order to allow the user to decide how to deal with the symmetry. Suppose that symmetry dictates the dipole moment will point in the x direction. The y and z components should be zero, but there will be some noise when they are evaluated in QMC. If you work out $|\mathbf{p}| = \sqrt{|\mathbf{p}_x|^2 + |\mathbf{p}_y|^2 + |\mathbf{p}_z|^2}$ then you will get something larger than $|\mathbf{p}_x|$, tending to $|\mathbf{p}_x|$ in the limit of perfect sampling (i.e., a biased estimate with finite sampling). You will also get larger error bars on $|\mathbf{p}|$ than $|\mathbf{p}|_x$.

For an example application to the water molecule see J. Chem. Phys. **127**, 124306 (2007).

35.9 Population

*K*eyword: **population**

The electronic charge within Voronoi polyhedra about the ions in a system with atoms can be calculated. This can be used to give an idea of the number of electrons associated with each ion.

For each electronic configuration sampled, for each ion I, CASINO will count the number $n_{I\alpha}$ of electrons of spin α for which I is the closest ion. The mean of $n_{I\alpha}$ and $n_{I\alpha}^2$ are reported in the 'population' block in `expval.data`.

Health warning: population analysis is at best qualitative. Simple Voronoi partitioning of the charge density does not necessarily lead to chemically sensible partial charges. A more accurate approach is to use Voronoi deformation density analysis. For example, for an HCl molecule, one could calculate the mean numbers of electrons in Voronoi polyhedra using CASINO for (i) HCl and (ii) HX, where X is a null pseudopotential placed where the Cl nucleus used to be. The partial charge on H is the difference of the two.

36 Atomic forces

The total atomic force is defined as the negative energy gradient with respect to the atomic position within the Born-Oppenheimer approximation, where the atomic positions are treated as parameters rather than dynamical variables. Section 36.1 states the force expression in VMC. Section 36.2 reports the exact and approximate expressions for the forces in DMC under two different localization schemes. Section 36.5 describes the implementation in CASINO and gives some practical advice.

36.1 Forces in the VMC method

We write the valence Hamiltonian for a many-electron system as

$$\hat{H} = \hat{H}_{\rm loc} + \hat{W}, \tag{518}$$

where $\hat{H}_{\rm loc}$ consists of the kinetic energy, the Coulomb interaction between the electrons and the local pseudopotential, and $\hat{W}$ is the nonlocal pseudopotential operator. We now consider a general parameter λ, e.g., a nuclear coordinate, which is used to vary the Hamiltonian, and upon which the trial wave function $\Psi_{\rm T}$ depends. Taking the derivative of the VMC energy $E_{\rm VMC}$ with respect to λ gives

$$F_{\rm VMC} = -\frac{\int \Psi_{\rm T}\Psi_{\rm T}\left(\frac{\hat{H}'\Psi_{\rm T}}{\Psi_{\rm T}}\right)\,dV}{\int \Psi_{\rm T}\Psi_{\rm T}\,dV} - 2\frac{\int \Psi_{\rm T}\Psi_{\rm T}(E_{\rm L}-E_{\rm VMC})\frac{\Psi_{\rm T}'}{\Psi_{\rm T}}\,dV}{\int \Psi_{\rm T}\Psi_{\rm T}\,dV}. \tag{519}$$

We use the notation $\alpha' = d\alpha/d\lambda$ where α can be a function or an operator. The first term in Eq. (519) is the Hellmann–Feynman theorem (HFT) force [112, 113] and the others are the Pulay terms [114].

36.2 Forces in the DMC method

In the DMC method, we may use two different pseudopotential localization approximation (PLA) schemes to evaluate the nonlocal action of $\hat{W}$ on the DMC wave function Φ. In these schemes $\hat{H}$ is replaced by an effective Hamiltonian [115, 25],

$$\hat{H}_A = \hat{H}_{\rm loc} + \frac{\hat{W}\Psi_{\rm T}}{\Psi_{\rm T}}, \qquad \hat{H}_B = \hat{H}_{\rm loc} + \frac{\hat{W}^+\Psi_{\rm T}}{\Psi_{\rm T}} + \hat{W}^-. \tag{520}$$

The nonlocal pseudopotential operator $\hat{W}^+$ corresponds to all positive matrix elements $\langle \mathbf{r}'|\,\hat{W}^+\,|\mathbf{r}'\rangle$, and $\hat{W}^-$ corresponds to all negative matrix elements [25], where $\mathbf{r}$ is the $3N$-dimensional position vector for the N-electron system and N is the total number of electrons. Following Ref. [116], these two approximations are referred to as the full-PLA (FPLA) and semi-PLA (SPLA) when using $\hat{H}_A$ and $\hat{H}_B$, respectively. The corresponding fixed-node DMC ground-state wave functions are denoted by Φ_A and Φ_B.

The DMC energy can be written in the form

$$E_{\rm D} = \frac{\int \Phi\hat{H}\Psi\,dV}{\int \Phi\Psi\,dV}, \tag{521}$$

which includes the mixed DMC ($\Psi = \Psi_{\rm T}$) and pure DMC ($\Psi = \Phi$) estimates of the energy. In all later expressions, Φ stands for either Φ_A or Φ_B, and $\hat{H}$ for either $\hat{H}_A$ or $\hat{H}_B$. Taking the derivative of the DMC energy with respect to λ gives

$$\frac{dE_{\rm D}}{d\lambda} = \frac{\int \Phi\hat{H}'\Psi\,dV}{\int \Phi\Psi\,dV} + \frac{\int \left[\Phi'\left(\hat{H}-E_{\rm D}\right)\Psi + \Phi\left(\hat{H}-E_{\rm D}\right)\Psi'\right]\,dV}{\int \Phi\Psi\,dV}, \tag{522}$$

for both the mixed and pure DMC methods. The first term in Eq. (522) is the HFT force [112, 113] and the other terms are Pulay terms [114].

36.3 The mixed DMC forces

The total force in the mixed DMC method, $F^{\rm tot}_{\rm mix}$, is obtained by setting $\Psi = \Psi_{\rm T}$ in Eq. (522). After some rearrangements, we arrive at

$$F^{\rm tot}_{\rm mix} = F^{\rm HFT}_{\rm mix} + F^{\rm P}_{\rm mix} + F^{\rm V}_{\rm mix} + F^{\rm N}_{\rm mix}, \tag{523}$$

with

$$F^{\rm HFT}_{\rm mix} = -\frac{\int \Phi\Psi_{\rm T}\left(\frac{\hat{W}'\Psi_{\rm T}}{\Psi_{\rm T}}\right)dV}{\int \Phi\Psi_{\rm T}\,dV} - \frac{\int \Phi\Psi_{\rm T}V'_{\rm loc}\,dV}{\int \Phi\Psi_{\rm T}\,dV} + Z_\alpha \sum_{\beta\,(\beta\neq\alpha)} Z_\beta \frac{\mathbf{R}_\alpha - \mathbf{R}_\beta}{|\mathbf{R}_\alpha - \mathbf{R}_\beta|^3} \tag{524}$$

$$F^{\rm P}_{\rm mix} = -\frac{\int \Phi\Psi_{\rm T}\left[\frac{\hat{W}\Psi'_{\rm T}}{\Psi_{\rm T}} - \left(\frac{\hat{W}\Psi_{\rm T}}{\Psi_{\rm T}}\right)\frac{\Psi'_{\rm T}}{\Psi_{\rm T}}\right]dV}{\int \Phi\Psi_{\rm T}\,dV} \tag{525}$$

$$F^{\rm V}_{\rm mix} = -\frac{\int \Phi\Psi_{\rm T}\left[\frac{\Phi'}{\Phi}\frac{\left(\hat{H}-E_{\rm D}\right)\Psi_{\rm T}}{\Psi_{\rm T}}\right]dV}{\int \Phi\Psi_{\rm T}\,dV} \tag{526}$$

$$F^{\rm N}_{\rm mix} = -\frac{\int \Phi\Psi_{\rm T}\left[\frac{\left(\hat{H}-E_{\rm D}\right)\Psi'_{\rm T}}{\Psi_{\rm T}}\right]dV}{\int \Phi\Psi_{\rm T}\,dV}. \tag{527}$$

$F^{\rm HFT}_{\rm mix}$ is the mixed DMC HFT force and the other expressions are Pulay terms. The HFT force in Eq. (524) contains two contributions from the pseudopotential, one from its local part $V_{\rm loc}$ and one from its nonlocal part $\hat{W}$, and a third contribution from the nucleus–nucleus interaction. In this nucleus–nucleus interaction term, $\mathbf{R}_\alpha$ represents the 3-dimensional position vector of the αth nucleus, and Z_α is the associated charge. The three Pulay terms in Eqs. (525)–(527) are identified as follows: $F^{\rm P}_{\rm mix}$ results from the PLA and is therefore called the pseudopotential Pulay term, $F^{\rm V}_{\rm mix}$ is the volume term, and $F^{\rm N}_{\rm mix}$ is called the mixed DMC nodal term since it can be written as an integral over the nodal surface [117]. Note that all terms in Eqs. (524)–(527) take the same form under both localization schemes; the only difference is the distribution ($\Psi_{\rm T}\Phi_A$ or $\Psi_{\rm T}\Phi_B$) used to evaluate the expectation values. A simple way to understand this is to note that $\hat{H}$ always acts on the trial wave function $\Psi_{\rm T}$ and $\hat{H}_A\Psi_{\rm T} = \hat{H}_B\Psi_{\rm T}$.

In mixed DMC simulations, it is straightforward to evaluate the contributions to the force, except for the volume term $F^{\rm V}_{\rm mix}$, because it depends on the derivative of the DMC wave function, Φ'. Since it is unclear how to evaluate Φ' in mixed DMC calculations, we use the Reynolds' approximation [118, 119],

$$\frac{\Phi'}{\Phi} \simeq \frac{\Psi'_{\rm T}}{\Psi_{\rm T}}, \tag{528}$$

which is exact on the nodal surface [see Eqs. (4) and (16) of Ref. [117]] but introduces an error of first order in $(\Psi_{\rm T} - \Phi)$ away from the nodal surface.

36.4 The pure DMC forces

The total force in the pure DMC method, $F^{\rm tot}_{\rm pure}$, is obtained by setting $\Psi = \Phi$ in Eq. (522). After some manipulations, we obtain

$$F^{\rm tot}_{\rm pure} = F^{\rm HFT}_{\rm pure} + F^{\rm P}_{\rm pure} + F^{\rm N}_{\rm pure}, \tag{529}$$

with

$$F^{\rm HFT}_{\rm pure} = \left\{ \begin{array}{c} -\dfrac{\int \Phi_A\Phi_A\left(\frac{\hat{W}'\Psi_{\rm T}}{\Psi_{\rm T}}\right)dV}{\int \Phi_{\rm A}\Phi_{\rm A}\,dV} \\ -\dfrac{\int \Phi_B\Phi_B\left(\frac{(\hat{W}^+)'\Psi_{\rm T}}{\Psi_{\rm T}} + \frac{(\hat{W}^-)'\Phi_{\rm B}}{\Phi_{\rm B}}\right)dV}{\int \Phi_B\Phi_B\,dV} \end{array} \right\} \tag{530}$$

$$-\frac{\int \Phi\Phi V'_{\rm loc}\,dV}{\int \Phi\Phi\,dV} + Z_\alpha \sum_{\beta\,(\alpha\neq\beta)} Z_\beta \frac{\mathbf{R}_\alpha - \mathbf{R}_\beta}{|\mathbf{R}_\alpha - \mathbf{R}_\beta|^3}$$

$$F^{\rm P}_{\rm pure} = \begin{cases} -\dfrac{\int \Phi_A\Phi_A \left[\frac{\hat{W}\Psi'_{\rm T}}{\Psi_{\rm T}} - \left(\frac{\hat{W}\Psi_{\rm T}}{\Psi_{\rm T}}\right)\frac{\Psi'_{\rm T}}{\Psi_{\rm T}}\right] dV}{\int \Phi_A\Phi_A\,dV} \\ -\dfrac{\int \Phi_B\Phi_B \left[\frac{\hat{W}^+\Psi'_{\rm T}}{\Psi_{\rm T}} - \left(\frac{\hat{W}^+\Psi_{\rm T}}{\Psi_{\rm T}}\right)\frac{\Psi'_{\rm T}}{\Psi_{\rm T}}\right] dV}{\int \Phi_B\Phi_B\,dV} \end{cases} \tag{531}$$

$$F^{\rm N}_{\rm pure} = -\frac{1}{2}\frac{\int_\Gamma |\nabla_{\mathbf{r}}\Phi|\Phi'\,dS}{\int \Phi\Phi\,dV}. \tag{532}$$

$F^{\rm HFT}_{\rm pure}$ is the pure DMC HFT force, $F^{\rm P}_{\rm pure}$ is the pure DMC pseudopotential Pulay term, and the pure DMC nodal term $F^{\rm N}_{\rm pure}$ is an integral over the nodal surface Γ (defined by $\Psi_{\rm T} = 0$). Where terms appear in braces, the upper one refers to the FPLA and the lower to the SPLA. The form of the nodal term in Eq. (532) is independent of the localization scheme. The nodal term involves the gradient $\nabla_{\mathbf{r}}\Phi$ evaluated at the nodal surface Γ. Reference [117] shows that this gradient is defined as its limit when approaching the nodal surface from within a nodal pocket (where Φ is nonzero). The derivation of the nodal term from Eq. (522) can be found in Refs. [120, 117].

Although the HFT force $F^{\rm HFT}_{\rm pure}$ under the FPLA can be calculated in the pure DMC method, it is not straightforward to evaluate the action of the nonlocal operator $(\hat{W}^+)'$ on the unknown DMC wave function Φ_B in $F^{\rm HFT}_{\rm pure}$ under the SPLA scheme. Therefore, the following localization approximation,

$$\frac{(\hat{W}^-)'\Phi_B}{\Phi_B} \simeq \frac{(\hat{W}^-)'\Psi_{\rm T}}{\Psi_{\rm T}}, \tag{533}$$

is used in the evaluation of $F^{\rm HFT}_{\rm pure}$ under the SPLA scheme which introduces an error of first order in $(\Psi_{\rm T} - \Phi)$.

Another complication arises in the pure DMC nodal term $F^{\rm N}_{\rm pure}$ in Eq. (532) because it involves the evaluation of an integral over the nodal surface. It is unclear how to evaluate such an integral in a standard DMC simulation. The following relationship suggested in Ref. [117],

$$F^{\rm N}_{\rm pure} = 2\,F^{\rm N}_{\rm mix} + \mathcal{O}[(\Psi_{\rm T} - \Phi)^2], \tag{534}$$

allows the approximate evaluation of $F^{\rm N}_{\rm pure}$ as twice its mixed DMC counterpart while introducing an error of second order in $(\Psi_{\rm T} - \Phi)$. Since $F^{\rm N}_{\rm pure}$ is defined as a volume integral in Eq. (527), the pure DMC nodal term $F^{\rm N}_{\rm pure}$ can then be calculated as a volume integral in a standard DMC simulation. Equation (534) is an application of the standard extrapolation technique [11], as in this case the variational estimate of the nodal term is zero [117].

36.5 Implementation of forces in CASINO

It is well-known that the HFT estimator has an infinite variance when the bare Coulomb potential is used to describe the electron–nucleus interaction. Different routes have been proposed to address this problem. Assaraf *et al.* [121, 122] added a term to the HFT force which has a zero mean value but greatly reduces the variance of the estimator. Chiesa *et al.* [123] developed a method to filter out the part of the electron density that gives rise to the infinite variance. Using soft pseudopotentials also eliminates the infinite variance problem [116], and this method is used in the CASINO code.

Since the atomic force equals the negative total derivative of the DMC energy with respect to a nuclear position λ, all previous force expressions involve total derivatives, in particular $\Psi'_{\rm T}$. Unfortunately, it is not straightforward to calculate total derivatives in VMC and DMC. A different route is to approximate all total derivatives by their partial derivatives, which introduces an error of first order in $(\Psi_{\rm T} - \Phi)$. We expect, however, this approximation to be rather accurate for the following reason: taking the total derivative of the VMC or DMC energy with respect to λ gives

$$\frac{dE}{d\lambda} = \frac{\partial E}{\partial\lambda} + \sum_i \frac{\partial E}{\partial c_i}\frac{dc_i}{d\lambda}, \tag{535}$$

where the c_i are the parameters in $\Psi_{\rm T}$ and the Hamiltonian. The sum on the right-hand side of Eq. (535) is neglected when all total derivatives are replaced with partial derivatives in all previous force expressions. This is exact in VMC when the wave function is optimized using energy minimization. In DMC, we can assume that the energy is approximately minimized with respect to the c_i. Therefore, we expect that the parameters c_i have little effect on both the VMC and DMC energy, i.e., $\partial E/\partial c_i$ is small. Therefore, neglecting the sum in Eq. (535) or, equivalently, replacing all total derivatives with partial derivatives in the expressions above, is expected to be a good approximation.

We use the analytic expressions derived in Ref. [116] for evaluating the HFT force. The Pulay terms may also be evaluated using analytic expressions, but to make the code more easily adaptable to other forms of trial wave function we use a finite-difference approach. This introduces an error which is linear in the infinitesimal nuclear displacement, Δ. We find that $\Delta \approx 10^{-7}$ Å minimizes the resulting error in the Pulay terms, which is about seven orders of magnitude smaller than the estimated values of the total forces. See Ref. [124] for more information.

The `input` keyword to activate the calculation of forces in VMC and DMC is **forces**. The keyword **forces_info** may be used to generate different levels of output. The default value is 2, which is recommended for speed. When 5 is chosen, two additional estimates of the Hellmann–Feynman force are calculated, which are only useful for debugging. These two additional estimates pick either the *s*-component or the *p*-component of the pseudopotential as the local one (the default is to pick the *d*-component as the local one). When calculating forces in DMC, the keyword **future_walking** must be chosen. See also Sec. 37.

Forces are only implemented for the Gaussian basis set and are only properly tested for molecules. When the molecule is linear (planar), the force algorithm in CASINO is optimized for geometries along the x-axis (x, y plane). See also the CASINO output. The forces are only implemented and tested for pseudopotentials. The implementation of forces in CASINO should in principle also work for all-electron calculations when the zero-variance estimators are chosen. These are the estimators 'Total Forces+ZV' in VMC and 'Total Forces (mixed)' in DMC. Also, since the SPLA scheme in DMC calculations requires an additional approximation in Eq. (533), we may expect that the FPLA scheme may give better results. For some small molecules and using large Gaussian basis sets, however, we find that the two localization schemes give very similar total forces. See also Refs. [125, 124].

36.6 Explanation of the force estimators printed by CASINO

In VMC: 'Total Force(dloc)' corresponds to Eq. (519) and the d-channel of the pseudopotential components is chosen local (default in CASINO); 'HFT Force(dloc)' is the first term in Eq. (519); 'Wave function Pulay term' is the second term in Eq. (519); 'Pseudopotential Pulay term' corresponds to Eq. (525) calculated in VMC, should have a zero average value, and is evaluated to check whether this condition is satisfied; 'Total Force+ZV(dloc)' is the same as 'Total Force(dloc)' with the zero-variance term added to reduce the statistical error; 'Zero-variance term' is the zero-variance term evaluated separately and should always have a zero average value; 'VMC NT' is a part of a total force estimator and should be added to the DMC estimator 'Total Force(purHFT,purNT,dloc)'; see Ref. [126].

In DMC: 'Total Force(purHFT, mixNT,dloc)' corresponds to Eq. (529); 'Total Force(purHFT, purNT,dloc)' is a total force estimator when the VMC estimator 'VMC NT' is added, see Ref. [126]; 'HFT Force(pur,dloc)' is the first term in Eq. (529); 'Nodal Term(mix)' is the third term in Eq. (529) calculated as twice times the mixed nodal term, as stated in Eq. (534); 'Nodal Term(pur)' equals the third term in Eq. (529) calculated as a pure estimator and is part of 'Total Force(purHFT,purNT,dloc)'; 'Pseudopotential Pulay Term(pur)' is the second term in Eq. (529); 'Total Force(mix,dloc)' corresponds to Eq. (523); 'HFT Force(mix,dloc)' is the first term in Eq. (523).

37 The future-walking method

The standard DMC algorithm generates the 'mixed' probability distribution $\Psi_{\rm T}\Phi$ which can be used to calculate unbiased estimates of an operator $\mathcal{A}$ that commutes with the Hamiltonian, $\hat{H}$. If, however, the operator $\mathcal{A}$ does not commute with $\hat{H}$, the 'pure' probability distribution $\Phi\Phi$ is required to obtain unbiased estimates. The simplest method to calculate pure estimates is to use the extrapolation estimator (2 time the mixed estimate minus the variational estimate) which introduces an error in the pure estimate that is of second order in $(\Psi_{\rm T} - \Phi)$. Exact pure estimates can be obtained, for example, by using the future-walking (FW) method which can straightforwardly be implemented in a standard DMC algorithm and will be discussed here, or by the reptation quantum Monte Carlo method. The FW method is used in CASINO with the keyword **future_walking**.

The basic idea of FW is to rewrite the pure estimate of a local operator $\mathcal{A}$ as

$$\frac{\langle\Phi|\mathcal{A}|\Phi\rangle}{\langle\Phi|\Phi\rangle} = \frac{\langle\Phi|\,\mathcal{A}\frac{\Phi}{\Psi_{\rm T}}\,|\Psi_{\rm T}\rangle}{\langle\Phi_0|\,\frac{\Phi_0}{\Psi_{\rm T}}\,|\Psi_{\rm T}\rangle} \simeq \frac{\sum_j \mathcal{A}(\mathbf{r}_j)\omega_j(\mathbf{r}_j)}{\sum_j \omega_j(\mathbf{r}_j)}, \tag{536}$$

with weights

$$\omega_j(\mathbf{r}_j) = \frac{\Phi(\mathbf{r}_j)}{\Psi_{\rm T}(\mathbf{r}_j)}. \tag{537}$$

For the Eq. (536) to be satisfied, $\mathcal{A}$ must be a local operator. Once the weights are known, the pure estimate can be calculated as an average over the local quantity $\mathcal{A}_j\omega_j$ with samples drawn from the mixed distribution generated by a standard DMC simulation. We will show in the next section, that these weights can be obtained from the asymptotic number of descendants of a walker $\mathbf{r}_j$. We then give a description of the FW algorithm that is now implemented in CASINO.

37.1 Derivation of the FW method

We show that the weight ω_j in Eq. (537) can be interpreted as the asymptotic number of descendents from the walker $\mathbf{r}_j$. We write the importance-sampled Schrödinger equation as

$$f(\mathbf{r},\tau) = \int \mathcal{G}(\mathbf{r} \leftarrow \mathbf{r}',\tau) f(\mathbf{r}',0) d\mathbf{r}', \tag{538}$$

where $\mathcal{G}$ is the importance-sampled Green's function. When the initial walker $\mathbf{r}_j$ is represented by a δ-function, $f(\mathbf{r}',0) = \delta(\mathbf{r}' - \mathbf{r}_j)$, Eq. (538) reduces to

$$f(\mathbf{r},\tau) = G(\mathbf{r} \leftarrow \mathbf{r}_j,\tau). \tag{539}$$

This can be interpreted as the transition probability of the walker to move from $\mathbf{r}_j$ to $\mathbf{r}$ in time τ. We write the importance-sampled Green's function in its spectral expansion [10],

$$f(\mathbf{r},\tau) = G(\mathbf{r} \leftarrow \mathbf{r}_j,\tau) = \frac{\Psi_{\rm T}(\mathbf{r})}{\Psi_{\rm T}(\mathbf{r}_j)} \sum_{n=0}^{\infty} \exp[-\tau(E_n - E_{\rm Ref})]\Phi_n(\mathbf{r})\Phi_n(\mathbf{r}_j), \tag{540}$$

where Φ_n and E_n are eigenfunctions and eigenvalues of $\hat{H}$, respectively, and $E_{\rm Ref}$ is a constant. When considering the limit of $\tau \to \infty$ and integrating over the final position $\mathbf{r}$, we obtain

$$\int \lim_{\tau\to\infty} f(\mathbf{r},\tau)\, d\mathbf{r} \;=\; \langle\Psi_{\rm T}|\Phi_0\rangle \exp[-\tau(E_0 - E_{\rm Ref})] \frac{\Phi_0(\mathbf{r}_j)}{\Psi_{\rm T}(\mathbf{r}_j)} \tag{541}$$

$$=\; \langle\Psi_{\rm T}|\Phi\rangle\, \frac{\Phi_0(\mathbf{r}_j)}{\Psi_{\rm T}(\mathbf{r}_j)}. \tag{542}$$

When $E_{\rm Ref} = E_0$, all contributions from the excited-states decay away in the penultimate equation in the limit $\tau \to \infty$. The left-hand side of the penultimate equation can be interpreted as the number of descendents from walker $\mathbf{r}_j$ for asymptotic τ,

$$N(\tau \to \infty) = \int \lim_{\tau\to\infty} f(\mathbf{r},\tau)\, d\mathbf{r}. \tag{543}$$

Combining the last two equations, we find

$$N(\tau \to \infty) = \langle\Psi_{\rm T}|\Phi\rangle\, \frac{\Phi(\mathbf{r}_j)}{\Psi_{\rm T}(\mathbf{r}_j)}, \tag{544}$$

where $\langle\Psi_{\rm T}|\Phi\rangle$ is a constant. This is the important relationship between the ratio $\Phi/\Psi_{\rm T}$ and the asymptotic number of walkers descended from the initial walker $\mathbf{r}_j$. When this relationship is inserted in the pure estimator of Eq. (536), the constants cancel in the numerator and denominator. Hence, the weights ω_j can be calculated in a DMC simulation from the asymptotic number of walkers. Since this technique involves taking information from a later time in the simulation to evaluate quantities at an earlier time, this method is called *future-walking* or *forward-walking* method. We introduce the *future-walking time* $\tau_{\rm FW}$ (or $N_{\rm FW}$ when given in time steps) that is necessary to project out the ratio of wave functions in Eq. (544). See Sec. 37.3 for a discussion of $\tau_{\rm FW}$.

37.2 The FW algorithm

Different implementations exist to calculate the asymptotic number of descendents in FW. The tagging algorithm introduced by Barnett *et al.* [127], for example, assigns a label to each walker which uniquely identifies all its ancestors. The asymptotic number of descendents of a walker $\mathbf{r}_j$ at time t is then determined by searching through all labels of the walkers at time $t+\tau_{\rm fw}$ and counting the walkers that descend from $\mathbf{r}_j$. A more elegant algorithm was proposed by Casulleras and Boronat (CB) [128] which evaluates the product $\mathcal{A}_j\omega_j$ and the sum $\sum_j \omega_j$ in Eq. (536) instead of calculating the weight ω_j and quantity $\mathcal{A}_j$ for each walker individually. We use this idea for the FW implementation in CASINO but chose a slightly improved version to the one originally proposed by CB.

To each walker $\mathbf{r}_j$, we assign a linked list L_j (or an order set) of $N_{\rm FW}$ elements, where $N_{\rm FW}$ is the total number of FW time steps. Each element in the list is a scalar or vector. At each time step, the local quantities (forces, energies, etc.) evaluated at $\mathbf{r}_j$ are written into one element, which is then added to the linked list on one side. Simultaneously, one element is deleted on the other side. Following this updating method, the nth element in the linked list always contains quantities that were evaluated n time steps ago so that the last element in the linked list was calculated $N_{\rm FW}$ time steps ago. When the walker drifts, diffuses and branches in a standard DMC simulation, the linked list is copied when the walker is copied, and the linked list is deleted when the walker is deleted. The result of this procedure is crucial: all walkers that have a common ancestor from $N_{\rm FW}$ time steps ago also have the same $N_{\rm FW}$th element in the linked list. Therefore, the numerator of the pure estimator in Eq. (536) at each time step can be written as

$$\sum_j \mathcal{A}(\mathbf{r}_j)\omega(\mathbf{r}_j) = \sum_j L_j(N_{\rm FW}), \tag{545}$$

where $L_j(N_{\rm FW})$ is the $N_{\rm FW}$th element of the linked list associated with walker $\mathbf{r}_j$. Hence, the sum of the product $\mathcal{A}_j\omega_j$ can be calculated as the sum over the $N_{\rm FW}$th elements of all linked lists for a given time step. Similarly, the sum over all weights in the denominator of Eq. (536) reduces to the population number $N_{\rm pop}$ at a given time step,

$$\sum_j \omega(\mathbf{r_j}) = N_{\rm pop}. \tag{546}$$

So far, we assumed a DMC simulation without reweighting. Since CASINO uses by default the reweighting scheme (**ibran**=`T`), the additional weights p_j from the branching factor need to be included in the pure estimate,

$$\frac{\sum_j p_j\omega_j\mathcal{A}(\mathbf{r}_j)}{\sum_j p_j} = \frac{\sum_j p_j L_j(N_{\rm FW})}{\sum_j p_j}. \tag{547}$$

The difference between the original algorithm by CB and the one implemented in CASINO and presented above is that the former averages over all $N_{\rm FW}$ elements in one linked list and only stores the averages for each linked list. In particular, after $N_{\rm FW}$ initial time steps, all elements of the linked lists are averaged, and additional $N_{\rm FW}$ time steps are required before these averaged values are used to calculate the pure estimate. The advantage of this CB algorithm is that it does not require the storage of the linked lists. The disadvantage is that the contribution to the pure estimate is only evaluated as an average over one block, which makes it impossible to reblock the data and to properly determine the statistical error bar. The FW implementation chosen in CASINO, in contrast, keeps and writes out the contributions to the pure estimate for each time step, which can be used to properly decorrelate the statistical data. The additional required storage of the linked lists is negligible for the systems tested so far.

37.3 Some practical advice

In principle, the FW estimator of Eq. (536) is only exact when the FW time is infinite. In practice, however, this is not possible: if the FW time is too long, it is very likely that most asymptotic walkers are descendents from only a few (possibly just one) walkers, since the total population is kept around a target population. Therefore, most wave-function ratios will be zero and the FW estimate is only calculated from a few (possibly just one) independent samples, and the FW estimate is inaccurate. In contrast, if the FW time is too short, higher-states in Eq. (544) may not have decayed away. Therefore, an optimal FW time must be chosen. For parallel computing, a larger target population is possible than on a single computer, resulting in a larger optimal FW time.

For calculations with a target population of around 10,000, a FW time of 10 a.u. was found to be sufficient in calculations for some small molecules [116, 117, 125] and no significant changes in the pure FW estimates were found for longer FW times. Therefore, the FW time is currently hardwired to 10 a.u. in CASINO.

38 Noncollinear-spin systems

38.1 Wave functions for noncollinear-spin systems

In a *noncollinear-spin* system, the particles of interest can have spin directions that are not parallel to the global quantization axis and/or the spin direction can vary with position in space. To treat such a system, it is not possible to assign a definite spin to each particle. Instead, the full four-dimensional position–spin coordinates of the particles must be considered.

CASINO can perform VMC calculations on noncollinear-spin systems by evaluating the energy expectation value

$$E = \frac{\left\langle \Psi(\mathbf{X})|\hat{H}|\Psi(\mathbf{X})\right\rangle}{\langle \Psi(\mathbf{X})|\Psi(\mathbf{X})\rangle} = \frac{\sum_{\mathbf{S}} \int |\Psi(\mathbf{R},\mathbf{S})|^2 E_{\rm L}(\mathbf{R},\mathbf{S})\, d\mathbf{R}}{\sum_{\mathbf{S}} \int |\Psi(\mathbf{R},\mathbf{S})|^2\, d\mathbf{R}}, \tag{548}$$

where $\mathbf{X}$ is the $4N$-dimensional vector of position and spin coordinates for all the particles in the system, $\mathbf{R}$ is their real-space positions and $\mathbf{S}$ is their spin coordinates. This is achieved by extending the standard Metropolis algorithm, so that there is a *spin-flip* step which may change the spin coordinate $\mathbf{s}$ in addition to the real-space position $\mathbf{r}$ of each particle. This extension is supported for VMC methods 1 and 3.

CASINO supports noncollinear calculations for Slater–Jastrow(–backflow) wave functions of the form

$$\Psi(\mathbf{X}) = \exp[J(\mathbf{R})] \begin{vmatrix} \psi_1(\mathbf{x}_1) & \cdots & \psi_1(\mathbf{x}_N) \\ \vdots & & \vdots \\ \psi_N(\mathbf{x}_1) & \cdots & \psi_N(\mathbf{x}_N) \end{vmatrix}. \tag{549}$$

The Jastrow factor and backflow function can only depend on the real-space positions of the particles, but the single-particle orbitals depend on both position- and spin-coordinates. Equivalently, we can say that the single-particle orbitals in the determinant can be arbitrary two-component spinors.

There are no specific keywords in the input files that control whether CASINO performs a noncollinear calculation or not. If noncollinear mode is supported for the selected system type and the input data about the wave function indicates noncollinear orbitals, CASINO automatically performs a noncollinear VMC calculation.

DMC does not support noncollinear spins.

38.2 Spiral spin-density waves in the HEG

Currently the only noncollinear-spin system type that can be studied using CASINO is a spiral spin-density wave state in the HEG. In such a system, the single-particle orbitals are of the spinor form

$$\underline{\psi}_{\mathbf{k}}(\mathbf{r}) = \frac{1}{\sqrt{\Omega}} e^{i\mathbf{k}\cdot\mathbf{r}} \begin{pmatrix} \cos\left(\frac{1}{2}\theta_{\mathbf{k}}\right) e^{-i\frac{1}{2}\mathbf{q}\cdot\mathbf{r}} \\ \sin\left(\frac{1}{2}\theta_{\mathbf{k}}\right) e^{+i\frac{1}{2}\mathbf{q}\cdot\mathbf{r}} \end{pmatrix}, \tag{550}$$

where $\mathbf{k}$ is the familiar plane-wave-vector, $\mathbf{q}$ is a constant vector (*magnetization wave-vector*) which is the same for all orbitals, and the values $\theta_{\mathbf{k}}$ are independent parameters for each orbital. For each orbital $\underline{\psi}_{\mathbf{k}}$, an electron can also occupy the orbital orthogonal to it, obtained by the replacement $\theta_{\mathbf{k}} \to \theta_{\mathbf{k}} + \pi$. A determinant of orbitals of the above form gives rise to a static, spiral spin density, with wave-vector $\mathbf{q}$.

Setting up a calculation of this form is very similar to a standard electron fluid calculation. In addition to the usual parameters, the `input` file must contain a definition of the magnetization wavevector in the **free_particles** block and the `correlation.data` file must contain a block specific to the SDW system, giving a definition of the occupied single-particle orbitals. For an example, see `~/CASINO/examples/electron_phases/3D_fluid_sdw`.

The calculation of the spin-density matrix and hence magnetization density in a spiral spin density wave is described in Sec. 35.2.4.

39 Magnetic fields and the fixed-phase approximation

39.1 Hamiltonian when an external magnetic field is present

Consider a many-particle system in the presence of an external magnetic field $\mathbf{B}(\mathbf{r}) = \nabla \times \mathbf{A}(\mathbf{r})$. The Hamiltonian is

$$\hat{H} = \sum_{i=1}^{N} \frac{1}{2m_i} (\hat{\mathbf{p}}_i - q_i \mathbf{A}_i)^2 + V, \tag{551}$$

where $\hat{\mathbf{p}}_i = -i\nabla_i$, the $\{m_i\}$ and the $\{q_i\}$ are the masses and charges of the particles, $V(\mathbf{R})$ is the usual electrostatic potential energy and $\mathbf{A}_i \equiv \mathbf{A}(\mathbf{r}_i)$ is the magnetic vector potential for particle i. The Hamiltonian does not have time-reversal symmetry in general, so that the eigenfunctions are complex.

39.2 VMC in the presence of an external magnetic field

Let Ψ be a complex trial many-body wave function. The expectation value of $\hat{H}$ with respect to Ψ is

$$\frac{\langle \Psi | \hat{H} | \Psi \rangle}{\langle \Psi | \Psi \rangle} = \frac{\int |\Psi|^2 E_{\rm L} \, d\mathbf{R}}{\int |\Psi|^2 \, d\mathbf{R}}, \tag{552}$$

where the complex local energy is

$$E_{\rm L} = \frac{\hat{H}\Psi}{\Psi} = \sum_{i=1}^{N} \frac{1}{2m_i} \left(-\frac{\nabla_i^2 \Psi}{\Psi} + q_i^2 |\mathbf{A}_i|^2 + 2iq_i \mathbf{A}_i \cdot \frac{\nabla_i \Psi}{\Psi} + iq_i \nabla_i \cdot \mathbf{A}_i \right) + V. \tag{553}$$

$\hat{H}$ is Hermitian, so its expectation value is real.[29] Hence only the real part of the local energy should be averaged when computing the VMC energy. On the other hand, the full complex local energy is needed when evaluating the variance of the local energy.

$$\sigma^2 = \frac{1}{N_{\rm C} - 1} \sum_{\mathbf{R}} |E_{\rm L}(\mathbf{R}) - \bar{E_{\rm L}}|^2, \tag{554}$$

where $N_{\rm C}$ is the number of configurations sampled, because Ψ is eigenfunction of $\hat{H}$ if and only if the complex local energy is constant (in which case the imaginary part of the local energy is zero). It is possible for the real part of the local energy to be constant, but the imaginary part to vary, in which case Ψ is not an eigenstate of the Hamiltonian. Therefore the variance of the real part of the local energy is not a good objective function for wave-function optimization.

It is not currently possible to use magnetic fields or complex wave functions in conjunction with linear-least-squares energy minimization in CASINO.

39.3 DMC in the presence of an external magnetic field

39.3.1 Fixed-phase Schrödinger equation

Let $\Phi = |\Phi| \exp(i\psi)$ be a complex many-Fermion wave function. The phase ψ must change by π whenever two particles are exchanged. Then

$$\begin{aligned} \exp(-i\psi)\hat{H}\Phi &= \left[\sum_{i=1}^{N} \frac{1}{2m_i} \left(-\nabla_i^2 + (q_i \mathbf{A}_i - \nabla_i \psi)^2 \right) + V \right] |\Phi| \\ &\quad + i \left[\sum_{i=1}^{N} \frac{1}{2m_i} \left(2[q_i \mathbf{A}_i - \nabla_i \psi] \cdot \nabla_i + \nabla_i \cdot [q_i \mathbf{A}_i - \nabla_i \psi] \right) \right] |\Phi| &(555) \\ &\equiv \left[\hat{\mathcal{H}}_\psi + i\hat{\mathcal{K}}_\psi \right] |\Phi|. &(556) \end{aligned}$$

[29]Averaging the complex local energy over a finite number of configurations distributed as $|\Psi|^2$ gives a small imaginary component in the mean energy, which vanishes in the limit of a large number of samples.

Noting that $|\Phi|\hat{\mathcal{K}}_\psi|\Phi| = \sum_{i=1}^{N} \nabla_i \cdot [|\Phi|^2(q_i\mathbf{A}_i - \nabla_i\psi)]/(2m_i)$, it is easy to show that $\left\langle |\Phi| \left| \hat{\mathcal{K}}_\psi \right| |\Phi| \right\rangle = 0$. Hence

$$\left\langle \Phi \left| \hat{H} \right| \Phi \right\rangle = \left\langle |\Phi| \left| \hat{\mathcal{H}}_\psi \right| |\Phi| \right\rangle . \tag{557}$$

So the ground-state eigenvalue of the fixed-phase Schrödinger equation $\hat{\mathcal{H}}_\psi|\phi_0| = E_0|\phi_0|$ is equal to the expectation value of the Hamiltonian $\hat{H}$ with respect to $\phi_0 = |\phi_0|\exp(i\psi)$, which is greater than or equal to the Fermionic ground-state energy of $\hat{H}$ by the variational principle, becoming equal in the limit that the fixed phase ψ is exactly equal to that of the Fermionic ground state [92].

39.4 Importance sampling

The fixed-phase imaginary-time Schrödinger equation is

$$\left(\hat{\mathcal{H}}_\psi - E_\mathrm{T}\right)|\Phi| = -\frac{\partial|\Phi|}{\partial t}, \tag{558}$$

where E_T is the reference energy. In the large-time limit the ground-state eigenfunction $|\phi_0|$ of the fixed-phase Hamiltonian is projected out.

Let the DMC wave function Φ have the same phase $\exp(i\psi)$ as the trial wave function Ψ. Then $f \equiv \Phi^*\Psi = |\Phi||\Psi|$ is real. Substitute $|\Phi| = |\Psi|^{-1}f$ into Eq. (558) and rearrange to obtain the importance-sampled fixed-phase imaginary-time Schrödinger equation,

$$\sum_{i=1}^{N} \frac{1}{2m_i}\left[-\nabla_i^2 f + 2\nabla_i \cdot (\mathrm{Re}(\mathbf{V}_i)f)\right] + [\mathrm{Re}(E_\mathrm{L}) - E_\mathrm{T}]f = -\frac{\partial f}{\partial t}, \tag{559}$$

where $\mathbf{V}_i = \Psi^{-1}\nabla_i\Psi = |\Psi|^{-1}\nabla_i|\Psi| + i\nabla_i\psi$ is the complex drift velocity. This is a straightforward generalization of the usual fixed-node importance-sampled imaginary-time Schrödinger equation, with the real part of the drift velocity appearing in the drift–diffusion term and the real part of the local energy appearing in the branching term. After equilibration the algorithm produces configurations distributed as $\phi_0^*\Psi = |\phi_0||\Psi|$. Noting that $\mathrm{Re}(E_\mathrm{L}) = |\Psi|^{-1}\hat{\mathcal{H}}_\psi|\Psi|$, the mixed estimate of the energy is equal to the pure estimate:

$$\begin{aligned}\frac{\int \phi_0^*\Psi \mathrm{Re}(E_\mathrm{L})\,d\mathbf{R}}{\int \phi_0^*\Psi\,d\mathbf{R}} &= \frac{\int |\phi_0||\Psi|\mathrm{Re}(E_\mathrm{L})\,d\mathbf{R}}{\int |\phi_0||\Psi|\,d\mathbf{R}} \\ &= \frac{\left\langle |\phi_0| \left| \hat{\mathcal{H}}_\psi \right| |\Psi| \right\rangle}{\langle |\phi_0| \,|\, |\Psi| \rangle} = E_0 = \frac{\left\langle |\phi_0| \left| \hat{\mathcal{H}}_\psi \right| |\phi_0| \right\rangle}{\langle |\phi_0| \,|\, |\phi_0| \rangle} = \frac{\langle \phi_0|\hat{H}|\phi_0\rangle}{\langle \phi_0|\phi_0\rangle}.\end{aligned} \tag{560}$$

For operators that do not commute with the Hamiltonian the mixed estimate is not equal to the pure estimate. Extrapolated estimation can be used in the same fashion as for real wave functions.

39.5 Applying magnetic fields in CASINO

At present it is only possible to apply uniform magnetic fields in CASINO, although it would be straightforward to generalize this. The vector potential is written as $\mathbf{A}(\mathbf{r}) = \mathbf{A}_0 + A_1\mathbf{r}$, where A_1 is a 3×3 matrix. Clearly it is possible to satisfy the Coulomb gauge condition $\nabla \cdot \mathbf{A} = 0$ with this form. To apply an external magnetic field, the **complex_wf** keyword must be set to `T` in the `input` file and the magnetic vector potential must be given in a **UNIFORM MAGNETIC FIELD** block in the `expot.data` file. The format is:

```
START UNIFORM MAGNETIC FIELD
Vector A0
 1.0 0.0 0.0
Matrix A1
 0.0 0.0 0.0
 1.0 0.0 0.0
 0.0 0.0 0.0
END UNIFORM MAGNETIC FIELD
```

Please note that it is crucial to use a trial wave function with the correct phase behaviour corresponding to the vector potential; otherwise the calculated energies will be nonsense.

40 CASINO on parallel computers

Diffusion Monte Carlo codes are parallelized, at the most basic level, by dividing the set of walkers/-configurations as evenly as possibly amongst the MPI processes, with each process then propagating its own walkers independently. In writing a code like CASINO from scratch, one might choose to do this using one of three language-independent APIs (applications programming interfaces), namely MPI, or OpenMP or OpenACC. There are pros and cons to either method. MPI involves explicit message passing, where the programmer might do things like '`call mpi_send`' to send some data between particular parallel processes, or '`call mpi_reduce`' to sum a quantity over the processes. OpenMP involves marking likely looking loops with compiler directives and using an appropriate compiler that supports OpenMP; on reaching this loop a 'master thread' will fork the appropriate number of 'slave threads' and the task is divided among them. OpenACC, which is focussed on the use of accelerators such as GPUs, involved compiler directives in a similar manner to OpenMP.

That said, CASINO was originally written so long ago (it was parallelized in the mid 1990s) that OpenMP had not yet been published and thus the choice of MPI was made for us. More recently-developed—and, of course, just not as good ;-)—codes such as QMCPACK have chosen OpenMP for their basic parallelization operations, but this is not necessarily an advantage (particularly since it can be difficult to get OpenMP to scale beyond 8 threads). In more recent years, a second level of parallelization was added to CASINO, where it uses OpenMP to parallelize over stuff like electrons and/or orbitals. One can thus in principle choose to run in 'hybrid mode' where one might fork N OpenMP threads in a multicore node with N cores, using MPI to communicate between nodes. In practice, it is usually better to use multiple MPI processes even within the multicore node and not to use openMP at all; the use of OpenMP would only be considered in large systems with many electrons where it makes sense to 'split a configuration' over multiple cores, perhaps to extend the number of usable processors that may be used if you insist on some definite minimum number of moves with no less than some fixed error bar. Even where you do use OpenMP, we find that in practice using more than 4 OpenMP threads per MPI process doesn't really buy you very much (this issue will be discussed in more detail in what follows).

One must carefully consider the use of memory in parallel calculations; in particular if some large array - such as the orbital coefficients - does not change during the calculation and has the same set of values on every processor, then it is hugely wasteful for each process to have its own copy of that array. Nevertheless, that is what CASINO will do when running in basic MPI mode. On a machine that is physically capable of it (and this is what shared-memory multicore nodes are basically for, so that's most machines nowadays) one would usually prefer to allocate the array in *shared memory* and, e.g., have only one copy of the array which all processes can access. Machines are available currently where running sixty-four MPI processes per node is reasonable; in such a case for a 2Gb blip file, the use of shared memory reduces the memory requirement from more than 128 Gb down to 2Gb. Clearly shared memory is practically obligatory on such a machine. Note that OpenMP multithreads use shared memory natively, but MPI programs require some additional magic, and this requires the code to be compiled with System V or Posix shared memory support (note this is *not* done by default). Further practical details are given later in this section.

The biggest machines in the world now have more than a million processors. Some techniques (such as DFT) have difficulty exploiting more than a thousand processors because of the large amount of interprocessor communication required, so an appropriate question is: how do the various different tasks done by a QMC code (VMC, DMC, optimization, . . .) scale with the number of processors, and consequently, how many processors can we successfully exploit? We begin with a theoretical analysis of precisely that point.

40.1 VMC in parallel

The VMC algorithm is perfectly parallel: no interprocessor communication is required during simulations. Each MPI process carries out an independent random walk using a different random-number sequence, and the results are averaged at the end of each block, so that running for a length of time T on P MPI processors generates the same amount of data as running for time PT on a single processor (assuming the equilibration time to be negligible). VMC should therefore scale to an arbitrarily large number of processors.

Note that, although the energy obtained by running for time T on P MPI processes should be in statistical agreement with that obtained by running for time PT on a single processor, the results will not be exactly equal, because the random walks are different in the two cases.

40.2 Optimization in parallel

40.2.1 Standard variance minimization

The VMC stages of a variance-minimization calculation are perfectly parallel, as described above. In the optimization stages, the configuration set is distributed evenly between the MPI processes. The master process broadcasts the current set of optimizable parameters, then each MPI process calculates the local energy of each of its configurations and reports the energies (and weights, if required) to the master. The CPU time required to evaluate the local energies of the configuration set usually far exceeds the time spent communicating (reporting one or two numbers per configuration to the master and receiving a handful of parameter values at each iteration). In particular the time spent evaluating the local energies increases with system size, whereas the time spent on interprocessor communication is independent of system size. So the standard variance minimization method is essentially perfectly parallel.

Note that the number of processor communications could easily be reduced further if each MPI process were simply to report the sum of its local energies and the sum of the squares of the local energies to the master.

40.2.2 Variance minimization for linear Jastrow parameters

The VMC stage of the optimization (including the construction and accumulation of the quartic coefficients) is perfectly parallel. The optimization itself is carried out in serial on the master process. However, this stage typically takes a fraction of a second, and is independent of system size. So the varmin-linjas scheme is essentially perfectly parallel.

40.2.3 Energy minimization

The VMC stages of an energy minimization are perfectly parallel, as described above. For the matrix algebra stages, the configurations are divided evenly between the MPI processes, each of which separately generates one section of the full matrices. The full matrices are then gathered on the master process, where the matrix algebra is done. The time taken to do the matrix algebra is usually insignificant in comparison to the time taken in VMC and matrix generation. The time taken in interprocessor communication is recorded and written out during energy minimization, and is typically at maximum a few percent of the total time spent in an iteration (and often much less than one percent). Overall, energy minimization is very nearly perfectly parallel.

40.3 DMC in parallel

40.3.1 Parallelization strategy

When performing DMC on a parallel machine using the standard algorithm, the population of configurations is usually distributed evenly over the set of MPI processes. DMC is *not* perfectly parallel because the populations on different MPI processes interact via the population-control mechanism. The population of configurations on each process fluctuates, and the cost of each time step is determined by the process with the largest population. It is therefore necessary to even up the distribution of configurations between MPI processes from time to time (‘load balancing’). Unfortunately, transferring configurations between processes is costly, and this is normally stated to be the principal limitation on the scaling of the DMC method with the number of processors. (There is also a small cost associated with the communication required to decide on a reference energy after each time step, and to average the energy over the MPI processes, but we assume this to be negligible henceforth.)

Note that with the release of CASINO 2.8 in Feb 2011, it was shown that the effective cost of configuration transfers can be reduced to essentially zero by using fancy tricks such as asynchronous communication (on machines that support it) and the use of subgroups of cores for redistribution. The discussion that follows ignores the possibility of doing these tricks, but we retain it for completeness as a discussion of the *theoretical* scaling.

40.3.2 Behaviour of the population on each MPI process

Let $T_{\text{redist}}\tau$ be the redistribution period, i.e., we redistribute the configuration population after every T_{redist} time steps τ. At any given time the population on an MPI process p must be increasing or decreasing exponentially, because the mean energy e_p of the configuration population on that process is unlikely to be exactly equal to the reference energy E_{T}. We assume that $e_p - E_{\text{T}}$ remains roughly constant over the redistribution period, i.e., that the autocorrelation period is much longer than the redistribution period. At the start of the redistribution period the population $C_p(1)$ on each process is the same. At the end of the redistribution period, the expected population on MPI process p is $C_p(T_{\text{redist}}) = C_p(1)\exp[-(e_p - E_{\text{T}})T_{\text{redist}}\tau]$. Hence $\bar{C}(T_{\text{redist}}) \approx \bar{C}(1)\exp[-(\bar{E} - E_{\text{T}})T_{\text{redist}}\tau] + \mathcal{O}(T_{\text{redist}}^2\tau^2)$, where the bar denotes an average over the MPI processes, and so the average growth or decay of the population is the same as that of the entire population (which should be small, because E_{T} is chosen so as to ensure this).

40.3.3 Optimal redistribution period

Let A be the cost of propagating a single configuration over one time step. Let B be the cost of transferring a single configuration between MPI processes.

Let q be the process with the largest number of configurations, i.e., with the lowest energy $e_q \equiv \min\{e_p\}$. Both the cost of propagating configurations and the cost of transferring configurations are determined by process q. The expected number of configurations on process q at the end of the redistribution period (i.e., after T_{redist} time steps) is $\max\{C_p(T_{\text{redist}})\} \approx \bar{C}(T_{\text{redist}}) + cT_{\text{redist}} + \mathcal{O}(T_{\text{redist}}^2)$, where $c = \bar{C}(1)(\bar{E} - \min\{e_p\})\tau$. NB, $\langle \bar{C}(1)\rangle = N_{\text{C}}/P$, where N_{C} is the target population and P is the number of MPI processes, and $\langle c\rangle$ is a positive constant.

At the end of the redistribution period, cT_{redist} configurations are to be transferred from process q. Hence the average cost of transferring configurations per time step is $B\langle c\rangle$, which is independent of T_{redist}.

The average cost per time step of waiting for the process q with the greatest number of configurations to finish propagating all its excess configurations is

$$\frac{A\langle c\rangle\,[0+1+\cdots+(T_{\text{redist}}-1)]}{T_{\text{redist}}} = \frac{A\langle c\rangle(T_{\text{redist}}-1)}{2}. \tag{561}$$

So the total average cost per time step in DMC is

$$T = \frac{AN_{\text{C}}}{P} + \frac{A\langle c\rangle(T_{\text{redist}}-1)}{2} + B\langle c\rangle. \tag{562}$$

Clearly the redistribution period should be chosen to be as small as possible to minimize T. Numerical tests confirm that increasing the redistribution period only acts to slow down calculations. One should therefore choose $T_{\text{redist}} = 1$, i.e., redistribution should take place after every time step. We assume this to be the case henceforth (the previously existing keyword **redist_period** which allowed the user to do otherwise has now anyway been deleted).

40.3.4 Scaling of the DMC algorithm with the number of processors

The cost A of propagating each configuration scales as N^α. For typical systems, where extended orbitals represented in a localized basis are used and the CPU time is dominated by the evaluation of the orbitals, $\alpha = 2$ [11]. The use of localized orbitals can improve this to $\alpha = 1$ [87]. For very large systems, or systems in which the orbitals are trivial to evaluate, the cost of updating the determinants will start to dominate: this gives $\alpha = 3$ with extended orbitals and $\alpha = 2$ with localized orbitals. Hence the average cost of propagating all the configurations over one time step, which is approximately the same on each MPI process, is

$$T_{\text{CPU}} \approx a\frac{N^\alpha N_{\text{C}}}{P}, \tag{563}$$

where a is a constant which depends on both the system being studied and the computer being used.

Let σ_{e_p} be the standard deviation of the set of energies on the different MPI processes. We assume that $\langle e_p\rangle - \langle\min\{e_p\}\rangle \propto \sigma_{e_p} \propto \sqrt{NP/N_{\text{C}}}$. Hence $\langle c\rangle \propto \sqrt{NN_{\text{C}}/P}$. The cost B of transferring a single configuration is proportional to the system size N. Hence the cost of load-balancing is

$$T_{\text{comm}} \approx b\sqrt{\frac{N_{\text{C}}N^3}{P}}, \tag{564}$$

where the constant b depends on the system being studied, the wave-function quality and the computer architecture. Note that good trial wave functions will lead to smaller population fluctuations and therefore less time spent load-balancing.

Clearly one would like to have $T_{\rm CPU} \gg T_{\rm comm}$, as the DMC algorithm would be perfectly parallel in this limit. The ratio of the cost of load-balancing to the cost of propagating the configurations is

$$\frac{T_{\rm comm}}{T_{\rm CPU}} = \frac{b}{a}\left(\frac{N_{\rm C}}{P}\right)^{-1/2} N^{3/2-\alpha}. \tag{565}$$

It is immediately clear that by increasing the number of configurations per MPI process $N_{\rm C}/P$ the fraction of time spent on interprocessor communication can be made arbitrarily small. In practice the number of configurations per MPI process is limited by the available memory, and by the fact that carrying out DMC equilibration takes longer if more configurations are used. Increasing the number of configurations does not affect the efficiency of DMC statistics accumulation, so, assuming that equilibration remains a small fraction of the total CPU time, the configuration population should be made as large as memory constraints will allow.

For $\alpha > 3/2$ (which is always the case except in the regime where the cost of evaluating localized orbitals dominates), the fraction of time spent on interprocessor communication falls off with system size. Hence processor-scaling tests on small systems may significantly underestimate the maximum usable number of processors for larger problems.

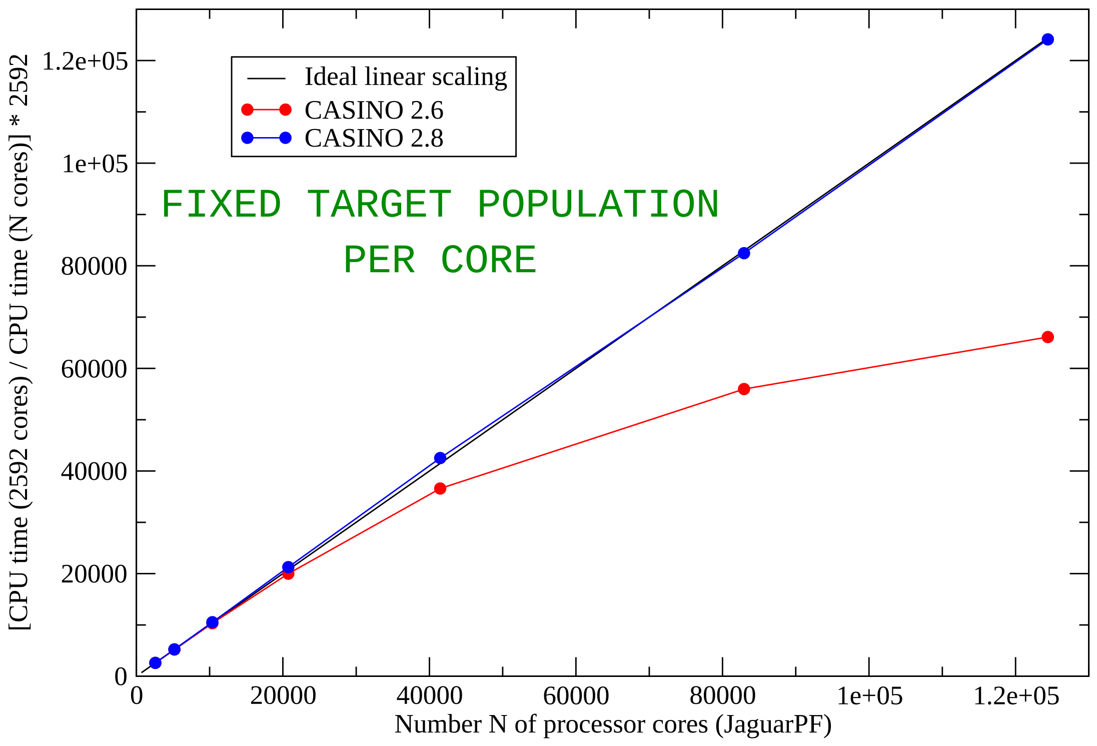
In summary, if a user wishes to assess the parallel performance of the DMC algorithm then it is best to perform tests using the system for which production calculations are to be performed. The number of configurations should be made as large as possible.

Other tricks which serve to reduce communication are the use of weighted DMC (**lwdmc** keyword) to reduce branching (with the default weight limits of 0.5 and 2.0) and the disabling of the transfer of large arrays (such as inverse Slater matrices) between MPI processes by using the **small_transfer** keyword.

That was the formal discussion; the additional practical points in the following section should be noted.

40.3.5 Parallel DMC in practice

The new redistribution algorithm introduced in CASINO 2.8 reduces the cost of configuration transfers to practically zero. This means that, when doing statistics accumulation under the right conditions, the CASINO DMC algorithm now has essentially perfect scaling out to at least a hundred thousand cores and beyond, as shown in the following diagram obtained from calculations on 120000+ cores of a Cray XT5 and discussed in Ref. [129].



The largest calculations of which we are aware (Feb 2014) have been done by MDT on up to 524288 cores of Japan's K computer where a similar scaling was achieved.

Note, however, that perfect linear scaling may require that the combination of your hardware and MPI implementation is capable of genuinely asynchronous non-blocking MPI, i.e., that commands like MPI_ISEND actually do what they are supposed to (in some MPI implementations this functionality is 'faked') and also that the machine is genuinely capable of doing communications at the same time as computing stuff. Understanding the extent to which this is true on particular machines with particular implementations of MPI requires further study.

It's also important to write to disk as little as possible; use longer blocks (**dmc_stats_nblock**) and turn off checkpointing (using the **checkpoint** input keyword) to stop config.out files being written at the end of the block. Before you object that this is cheating, it's actually perfectly reasonable and safe even in practical calculations. If your machine has a batch queue system with fixed time limits you can use the `--auto-continue`/`--continue` flags to the `runqmc` script, in combination with, e.g., the **max_cpu_time** keyword in input, and the calculation will emergency stop if it approaches the time limit, then resubmit itself and restart.

How to achieve the linear scaling in practice? Let's say that one requires 500 million samples of the configuration space to compute the answer with the required error bar. Using 10 configurations/core on 10000 cores will thus require 5000 DMC statistics accumulation moves per core to get the required number of samples. What happens if I double the number of cores to 20000 and want to double the speed of the calculation? I can either halve the number of configurations per core to 5 or I can halve the number of moves to 2500 in order to get my 100 million samples. To achieve the best scaling with processor number it is important that you do the latter; this (a) makes sure that each process continues to have enough 'work to do' compared to the time spent doing communication by maintaining a constant number of configurations per core, and (b) reduces the required number of instances of load-balancing.

Note that are some caveats here: this halving of the number of moves cannot continue indefinitely since, e.g., we need a minimum number of moves in order to reblock the data, so as you increase the processor number at a certain point you will have to start choosing the less efficient option of reducing the number of configurations per core. Note also that if you insist your answer has no *less* than some required error bar, and a minimum number of moves are performed, then for any given system there is a maximum number of cores that you can usefully exploit. Using more cores than that will merely serve to make the error bar smaller than you need, and is thus pointless. (Note that to some extent you can use OpenMP to exploit more processors than the maximum by having 1 config per MPI process, and getting this moved by multiple OpenMP threads each running on separate cores).

A particularly important caveat which we have ignored so far is the following: *DMC equilibration time cannot be reduced towards zero by using more cores.* And when the equilibration time becomes comparable to the statistics accumulation time (which *is* reduced by using more cores) our scaling analysis will be affected. This problem, and what you can do about it, is discussed in more detail in the following section.

CASINO's support for GPUs using OpenACC is still experimental. Please contact Neil Drummond for more information.

40.3.6 Reducing the computational expense of DMC equilibration

It is always necessary to discard the results of propagating the configuration population over the first few correlation periods (the equilibration phase) of a DMC simulation. Suppose the target population is large. Because the number of statistics-accumulation steps required to achieve a given error bar is relatively small in this case, equilibration can be a large fraction of the total computational effort. Indeed, for some calculations, DMC equilibration accounts for as much as half of the total CPU time. By contrast, if one uses a small population and runs for a large number of correlation periods until one has generated the required amount of data, equilibration is a small fraction of the total run time. Unfortunately, when running on a large number of cores, the configuration population is necessarily large, since (unless you use OpenMP to split them between cores) each core must have at least one configuration on average.

CASINO is capable of using an alternative approach for equilibrating a DMC calculation with a large target population P. Instead of using VMC to generate P initial configurations, VMC can be used to generate a small number of configurations q, which are then used in a preliminary DMC calculation to generate P configurations. In this preliminary DMC calculation with q configurations, equilibration is followed by $(P/q)T_{\rm corr}$ time steps of statistics accumulation, where $T_{\rm corr}$ is the correlation period in terms of time steps, during which a total of P configurations are written out. Following this, a P-configuration DMC statistics-accumulation calculation is performed. Note that q must be sufficiently large that population-control bias is negligible in the preliminary DMC calculation.

For example, if one wanted to use a population of 10,000 configurations in a DMC calculation, one could carry out a preliminary DMC calculation with 1,000 configurations, until 10,000 independent configurations were generated, then use these as the initial population for the main 10,000-configuration run (which would not need to be equilibrated). The computational effort of equilibration would be reduced nearly tenfold.

If the data generated in the preliminary DMC calculation were discarded altogether, this approach would still lead to a substantial reduction in the computational effort, because the DMC equilibration would be performed with a small population. However, one can further reduce the waste of computational effort: the data generated in the statistics accumulation phase of the preliminary DMC calculation can be averaged with the data generated in the main DMC calculation. Since some of the configurations generated in the preliminary calculation are reused in the initial population of the main DMC calculation, some care must be taken to ensure that the error bar is not underestimated. The most effective way of combining the data from the preliminary and main calculations is still being investigated, and it is left up to the user to combine the data 'by hand' for the time being.

The drawbacks of this method are (i) having an extra stage to the calculation; (ii) the fact that the preliminary DMC calculation would probably have to be run on a relatively small number of processors compared with the main calculation; and (iii) the fact that population-control bias might be a problem if the population q in the preliminary calculation is very small.

In order to perform a preliminary DMC calculation (i.e., to write out configurations during DMC statistics accumulation), the **dmc_nconf_prelim** keyword should be set to the total number of configurations to be written out (summed over all MPI processes). The configurations are written out once every **dmc_decorr_period** iterations, which should therefore be given a suitably large value. The total number of statistics-accumulation steps carried out should be sufficiently large that the required number of configurations can be written out; however, the statistics-accumulation run will in fact keep going until all the configurations have been written out, going beyond the specified number of steps if necessary. Note that the statistics-accumulation phase of a preliminary DMC run cannot be continued.

Whilst on the subject of reducing the expense of DMC equilibration, we point out that the length of the equilibration period can be reduced if the initial configuration population is already partially equilibrated. For example, if performing DMC calculations at a few different time steps, the initial configuration population can be generated at one particular time step, then reused in the others. The number of equilibration steps needed to re-equilibrate the population when the time step is changed is small compared with the number of steps required to equilibrate the population from scratch.

40.4 Shared memory support

Shared memory is required to avoid redundant copies of large arrays on multicore nodes (which form the basic computing unit of most modern computers). Not using it can multiply the memory required per node by up to 64 on current architectures, so its use is basically obligatory.

Shared memory is implicit when parallelizing using OpenMP. For programs parallelized using MPI such as CASINO, one needs to explicitly compile the code with support for shared memory (the new MPI3 standard will support it natively, but CASINO doesn't yet support MPI3). A future version of CASINO will do this by default when the hardware allows it (i.e., almost always).

To compile with shared memory support, you proceed as follows. If compiling from the command line, type '`make Shm`' (or '`make -j N Shm`', where *N* is a suitable number of cores, to compile faster in parallel)—the capital S is not obligatory. If you're compiling the code using the install script, then it will present you with a list of possible `CASINO_ARCH`s and ask which of them you want to compile. If you want to compile—for some strange reason—`CASINO_ARCH`s number 1 and 2 in the list with shared memory support, and number 3 without, then you would respond '`1:Shm 2:Shm 3`'.

How to run with the shared memory executable? You include `-s`, `--shm`, or `--shmem` as an argument to the `runqmc` script. This by default sets the number of processes among which to share memory to be all cores on the same node. If you wish to share memory among a non-default number of processes, then you need to set the `CASINO_NUMABLK` environment variable to that number. This is done automatically by `runqmc` if you invoke it as either '`runqmc --shm=<numablk>`' or '`runqmc --shmem=<numablk>`' (the `-s` form always uses the default number).

This can be useful on 'NUMA nodes' (i.e., nodes with Non-Uniform Memory Access). Simplifying somewhat, let's say that a 32-core node consists of 4 physical 8-core processors plugged into a board, and each 8-core processor can access its own local memory faster than the memory local to the other 3 processors. Then—if you have enough memory available—it would be faster to run with 4 copies of your shared memory arrays, and each one will be shared by all cores on a processor. In practice most people don't bother reading the documentation deeply enough to realize that this is likely to benefit them, and end up not doing it. (Probably not enough practical timing tests have been done to determine how much this kind of thing matters with CASINO).

How does all this work? There are two shared memory APIs in common use: System V and Posix—and CASINO is able to use either. On most machines CASINO uses System V.

In CASINO this functionality is implemented using (1) a low level C routine (`alloc_shm.c`) which contains the actual Posix or System V commands that do the allocating and deallocating of shared memory, (2) A Fortran module `shalloc_smp.f90` defining a 'shallocate' function entirely analogous to the normal Fortran allocate function; this determines the *type* of the array to be allocated in shared memory (integer, double precision, single precision, complex, etc.) and at how many dimensions it has, then calls the stuff in `alloc_shm` appropriately. (3) A 'fake' Fortran module `shalloc_nonsmp.f90` for when you don't want to use shared memory mode (some machines, such as the Japanese K computer, physically won't allow it). This simply allocates the array using a normal Fortran allocate statement for each MPI process. Then a non-Shm computer without System V/Posix won't get confused by trying to '`call shallocate`'.

A programmer can in principle just replace all required allocations with 'shallocations' and work as normal. In practice you also need to worry about synchronization, e.g., if the master process on a node is responsible for filling the shared memory array with numbers, make sure that no other process on that node tries to read it before the array has been filled. This may require the use of node level barrier calls ('`call shallocate_barrier`' in CASINO).

To add support for shared memory to a CASINO arch file, you need to set `SUPPORT_SHM = yes` and `CFLAGS_SHM` to reference the appropriate API, i.e., one of `-DSHM_SYSV`, `-DSH_POSIX`, or `-DSHM_POSIX_BGQ`.

The reason for the existence of the `POSIX_BGQ` option is discussed in the next section.

One final point about Shm for developers: note that in general a shared memory segment is not automatically released by the OS when the program that created it terminates. This is a result of the OS only knowing about the existence of processes, not about what the process is doing. If one process dies the OS does not know whether another process is accessing a given segment or not. Thus the program itself must specifically delete the segments at the end, otherwise they will hang around and more and more of the system memory will be occupied by unused shared memory segments. Of course this ceases to be a problem on machines where the used partition is rebooted after every calculation (e.g., Blue Gene machines). However…

40.4.1 The Blue Gene problem

IBM Blue Gene machines (currently, Blue Gene/Qs) really, really think they're special, and they love doing everything just a little bit differently to every other machine out there; they are, in short, specially designed to make life hard for you—especially when you try to do shared memory. Here are some lovely features of these machines that you will enjoy:

(1) You have to 'guess' in advance how much of the memory on a node will be required for *shared* memory. Even we don't know that, and we wrote the program—a regular user probably wouldn't have a clue (though see below and the FAQ, question B8).

(2) It takes the total memory on a node, subtracts the required shared memory, then (approximately!) divides the remainder by the number of MPI processes on the node, giving a maximum amount of non-shared memory that may be used by *any individual MPI process.* On a Blue Gene/Q with 16Gb per node, running with the recommended 64 processes per node (`runqmc --ppn=64`'), this is somewhat less than 250Mb per process, which is not good—especially if, as with CASINO—the master process can sometimes require considerably more memory than the slaves. If you run out of available memory per process, you would hope—in the second decade of the twenty-first century—that there would be a soft pinging noise, a little blue light would come on, and a gentle error message 'Out of memory, I'm afraid' would appear. Instead the BG/Q will effectively vomit all over your shoes, spew a vast number of error messages to standard error—none of which, of course, have anything to do with running out of memory—and essentially CASINO will just appear to crash in a massively inelegant way. If this happens, try running with the `--ppn` flag set to a smaller value, which will reduce the number of MPI processes per core.

(3) However, to make *efficient* use of a processor core on a BG/Q, you are supposed to run more than 1 thread/process per core (see the CASINO FAQ, question B9). Running only 1 MPI process per core means very sloooow code compared to a modern Cray XK7 or something.

(4) There is no System V (at least not on the BG/Qs I've used), and the Blue Gene/Q implementation of Posix Shm appears to be full of bugs, to the extent that we had to completely reimplement the Posix part of `alloc_shm.c` using an alternative algorithm where the memory allocation is controlled by a linked list of pointers to block of memory (this may be accessed by setting `CFLAGS_SHM` in the machine's CASINO arch file to be `-DSHM_POSIX_BGQ`). Just for the record, Blue Gene Posix does not remove unlinked files, `ftruncate` produces unexpected results, `mmap` does not use the offset argument, etc., etc. Note that CASINO's original implementation of shared memory required only the shallocation of a single vector (the array of blip coefficients) at the start of a calculation, which then remained throughout. Blue Gene/Q Posix can cope with this. It was only when we began to try to repeatedly allocate and deallocate things in late 2013 that the deficiencies became apparent.

To help you with the above, we have implemented a few things. OK, so on Blue Gene/Qs one needs to know in advance the number of MB of shared memory required, so that one may set the BG_SHAREDMEMSIZE environment variable (this is/was called BG_SHAREDMEMPOOLSIZE on Blue Gene/Ps). If the required amount of shared memory is, say, 100Mb then this can be set appropriately and passed to the compute nodes simply by using a `--user.shemsize=100` argument to `runqmc`. (Never attempt to run on a Blue Gene without the runqmc script, trust me.) Note that if you include `--user.shmemsize` as a command-line argument, you may omit the usual `-s` which toggles shared memory.

How do you know what value to use for `user.shmemsize`? For small systems on not too many processors where you're not likely to run out of memory, try using 1.2 times the size of the `bwfn.data.bin` (or `bwfn.data.b1`) file for blip calculations, and 2.2 times the size of the `gwfn.data` file in Gaussian calculations. To set the value exactly without guessing, note that CASINO will calculate this number and print it to output at the end of the setup process (within the scope of **testrun**=T). This means you can do a quick preliminary calculation to find it out (search for, e.g., 'shared memory' in the output file). However, the amount of shared memory required depends on the number of cores per shared memory node (or the number of MPI processes per node if running more or less than one process per core). If one ultimately wishes to run on, say, half a million cores, it may be desirable to execute the test run on just a few cores on your personal machine, rather than waiting a week for the full job to sit in a queue of a national supercomputer. For the purposes of computing the size of the shared memory partition, one may therefore set the number of desired processes per node by setting the keyword **shm_size_nproc** in the CASINO input file, and this value will be used in the computation of the shared memory size rather than the actual number of processes per node being used in the test run (unless **shm_size_nproc** =0—which is the default). Note that the CASINO test run must be done in Shm mode, i.e., with `runqmc -s`.

If you're setting up the code on a new Blue Gene, feel free to follow the hideously complicated template in, e.g., `CASINO/arch/data/bluegene-xlf-cobalt-parallel.mira.arch`.

40.5 Using CASINO on the Knights Landing manycore processor

The Intel Knights Landing (KNL) processor is the second generation of Intel's Xeon Phi range. The KNL has up to 72 physical cores per processor. These cores each support up to four hyperthreads, and have two vector processing units. The memory on each node is divided into a relatively large conventional memory and a high-bandwidth on-chip multichannel dynamic RAM (MCDRAM), which is often configured to act as a large cache. It is straightforward to use CASINO on the KNL processor.

Our experience with running CASINO on the ARCHER KNL cluster (`http://www.archer.ac.uk/documentation/knl-guide/`) suggests the following:

- The Cray Fortran compiler appears to give slightly better performance on the KNL than either ifort or gfortran.
- The use of four hyperthreads per physical core improves the per-node performance of CASINO. To use multiple hyperthreads, use e.g. '--ppn=256' as an argument of RUNQMC to request 256 processes per node (if there are 64 physical cores per node).
- The per-node performance of the ARCHER KNL cluster (with 64 physical cores per node) is about the same as the per-node performance of ARCHER itself (Cray XC-30 with 24 cores per node), provided the Cray compiler and hyperthreading are used on the KNL.
- For blip calculations, shared memory should be used. There does not seem to be a significant speedup from squeezing all the data onto the MCDRAM as opposed to using the MCDRAM as a cache. Use the '--user.memmode' flag to RUNQMC to specify how the MCDRAM is to be used (on the ARCHER KNL cluster this can take values 'quad_100' or 'quad_0'; see the ARCHER documentation for more information on the memory modes available).

40.6 OpenMP support

40.6.1 Introduction

In addition to MPI, CASINO also has a secondary level of parallelization using OpenMP.

Current and near-future processors have a hierarchical architecture due to limitations in the amount of power that can be reasonably delivered to and dissipated from each processing unit [130]. One approach to the different levels in the hierarchy is to use multiple simultaneous approaches to parallelism, with an OpenMP-like level parallelizing across the cores in one or a few CPUs and an MPI-like level parallelizing across the entire system.

For a pure-MPI QMC calculation with P processors, the total computation time t is roughly given by $t \approx MCt_{\rm c}/P$, where M is number of steps, C is number of configurations and $t_{\rm c}$ is the average time to move one configuration at each step. However on very large computers one can be in a situation where the desired C and P are such that $P > C$, which means that there will be MPI processes with no configurations in them (and thus idle), which is a waste of resources.

The second level of parallelism becomes useful when $P > C$. Running multiple OpenMP threads on multiple cores allows keeping C small, effectively reducing t_c in the cost formula above.

40.6.2 Implementation basics and performance

The general strategy of this implementation is to use OpenMP parallelism for the loops whose trip counts scale with the number of electrons or atoms. In the QMC algorithm the basic logical units that need to be parallelized are routines like

- orbital evaluation routines (only for blips, currently)
- Jastrow factor evaluation routines,
- inverse Slater matrix updating routine,
- potential energy evaluation routine,
- electron–electron and electron–nucleus distance evaluation routines,
- etc.

Extensive performance tests were done on pseudopotential systems that use the `blip3d` and `blip3dgamma`. The best performance obtained on an AMD quadcore CPU was for a system of 1024 electrons. The speedup factor was close to 1.5 for 2 OpenMP threads and close to 2 for 4 OpenMP threads. Larger systems had `update_dbar` as an OpenMP bottleneck.

Note that performance benefits using OpenMP are basis-set dependent, since the orbital evaluation is only parallelized when using blips. If using Gaussians or whatever, then one should still see some benefit from speedup of the Jastrow and other routines.

40.6.3 Using OpenMP

To use the OpenMP feature, compile the code with `make Openmp` on a supported architecture (or respond, e.g., '`1:OpenMP` when using the compilation option of the *install* script). Then use the option `--tpp` *`threads-per-process`* in the `runqmc` command line to specify the number of OpenMP threads per process.

E.g. to run two processes with two threads each you would type `runqmc --nproc=2 --tpp=2`, ideally on a 4-core machine. By default on batch-queueing systems the number of cores reserved for the job will be `nproc * tpp`.

Note finally that when analysing timing data from Openmp runs you need to look at the 'Real Time' data, rather than the 'CPU time' data, since the CPU time is summed over all OMP threads.

40.6.4 Using OpenMP with Shm

To use the OpenMP feature in combination with System V or Posix shared memory, compile the code with `make OpenmpShm` (or `make openmpshm`) on a supported architecture (or respond, e.g., '`1:OpenmpShm` when using the compilation option of the *install* script). Then use the option `--tpp` *`threads-per-process`* in the `runqmc` command line to specify the number of OpenMP threads per process, in conjunction with the usual '-s' flags.

You may need to think about the required MPI process-binding flags if using shared memory together with OpenMP (see Sec. 40.6.5).

40.6.5 MPI process binding

It is not usually necessary to supply MPI process-binding flags to `mpirun` (i.e., the default behaviour is fine), but occasionally it is necessary to specify how MPI processes are to be bound. As an example, suppose your compute node has two CPU sockets, with each CPU having eight cores (so there are 16 cores on the compute node). Usually the NUMA nodes (see Sec. 40.4) correspond to the CPU sockets. Hence if you want to run eight MPI processes per compute node with two threads each using shared memory then you want to assign and bind four MPI processes to the first socket and four to the second socket, and you want to make sure that the MPI ranks of the four bound to the first socket are 0, 1, 2 and 3, and that the ranks assigned to the second socket are 4, 5, 6 and 7. This can be achieved in OpenMPI by adding the flags "`--map-by socket --rank-by core --bind-to socket`" to `mpirun`. On the other hand, if you want to run one MPI process with 16 threads then you cannot bind the process to a socket, otherwise it will only be able to spawn eight threads. In this case you should not attempt to bind the process, i.e., you should include the `mpirun` flag "`--bind-to none`".

Binding flags are not currently set by default, except on the Lancaster high-end-computing cluster (see `CASINO/arch/data/machine/lancaster.arch`). Please edit your `.arch` files to include binding flags as and when you need them.

40.7 OpenACC support

CASINO's support for GPUs via OpenACC is still experimental. Please contact Neil Drummond for more information.

A Appendix 1: Programming guide for CASINO

A.1 Making changes to the CASINO source code

Many people from around the world have contributed to CASINO, and we are very grateful for this. However, we request that users should obtain the agreement of the developers before making alterations to either the main source code or the utilities supplied in the CASINO distribution, as per the user agreement / download form. Furthermore, we cannot guarantee that your changes will subsequently be included in the CASINO distribution, or that they will remain there. Any contributions made to the CASINO code may be edited by the developers or indeed by other contributors.

The following comments apply to any changes that you make to the existing source code of CASINO, and to any utilities that you wish to add to the CASINO distribution.

A.2 Languages

The main CASINO source code and many of the utilities are written in Fortran, which **must** conform to the Fortran 2003 standard; later standards are not necessarily supported by all compilers. There are also a couple of simple C routines in the main code, for shared memory etc. Utilities not in Fortran should generally be written as bash shell scripts (or possibly tcsh or csh, though this is deprecated). To test the validity of bash scripts, please consider using shellcheck (`https://github.com/koalaman/shellcheck`). We have one C++ pseudopotential conversion utility. The ADF converter in `utils/wfn_converters/adf` and a couple of other utilities are written in Python. Please understand that CASINO is meant to compile and run out of the box on any computer in the world, and the fewer languages we are dependent on the easier this is to achieve.

If you would like to add utilities written in Python then please try to ensure that your code is as portable as possible, e.g., it should not rely on any difficult-to-obtain libraries, nor use any bleeding-edge language features, nor assume some particular Python virtual environment. The code should be written in Python 3. If you are adding a critically important utility that will be required almost everywhere that CASINO is used then it is probably safest to stick to Fortran and bash.

A.3 Style

CASINO has a Fortran 2003 'style' which should be adhered to when writing code, both for the main source and for the Fortran utilities. This is because it is desirable that the package has a homogeneous look and feel (and because searching for text strings then works consistently). Everybody has their own style. Yours is different and may even be better, but we've decided on one for CASINO and there it is. If you don't write your code like this, the likelihood is that MDT or someone else will reformat it for you, and they will probably accidentally delete a crucial minus sign while correcting your routine, an error which may take two weeks of your life to track down (and this could be much better spent doing other things). You can get the idea just by looking at the developer version source code in `CASINO/src` (the standard version is 'obfuscated' and practically unreadable) but let's emphasize the main points:

- Don't use more than one module per physical file, as Hitachi compilers won't (or didn't used to) allow this!
- Capitalization outside of character context: use upper-case letters for keywords whose use is primarily at compile time (statements that delimit program and subprogram boundaries, declaration statements of variables). Use lower-case letters for everything else, including the bulk of run-time code. Description of routines is to be enclosed in a little box with a description of what it does and an author. Later changes are to be documented at the bottom of this box. For example:

```
 SUBROUTINE rubbish
!-----------------------------------------------------------------------!
! Routine to write words to the screen.
!
!
!
! MDT 8.2000
!
!
!
! Changes:
!
! --------
!
! 3/2001 MDT - added capability to write 'Hello'
!
! 5/2001 MDT - added additional capability to write 'Donkey!'
!
!-----------------------------------------------------------------------!
  USE dsp
  USE parallel
  IMPLICIT NONE
  INTEGER i,j,k,some_integer
  REAL(dp) a
  CHARACTER(llength)my_name

  i=0
  do j=1,10
   write(6,*)'Hello'
   do k=1,1000000
    i=i+1
    write(6,*)'Donkey!'
   enddo
  enddo

! etc.

 END SUBROUTINE rubbish
```

- Contents of if, do and case blocks should be indented by *one* space.
- In general don't put spaces between words, e.g.,

  ```
  if(something_is_true)a=b+c
  ```

  not

  ```
  if ( something_is_true )  a  =  b  +  c
  ```

- Do not use '.eq.', '.ne.', '.lt.', '.le.', etc.; use the Fortran 90 versions instead, i.e., '==', '/=', '<', '<=', etc.

- Use a maximum of 80 characters per line. If you go over this, use '&' continuation characters at the end of the line *and* at the beginning of the next one (don't include spaces between the ampersand and the text).
- There should be two blank lines between subroutines in a given physical file.
- Error conditions are to be handled using the `errstop` and `errwarn` routines (in the `utilities.f90` module).
- Use 'endif' and 'enddo', not 'end if' and 'end do'.
- No double colons should be used in simple variable declarations, e.g.,

  ```
  INTEGER ialloc
  ```

  except where required, e.g.,

  ```
  INTEGER,INTENT(in) :: n
  ```

- In lists of declared variables, adhere to the following order:
  - INTENTed dummy arguments first, order: in/out/inout

    ```
    INTEGER,INTENT(in)
    (INTEGER,INTENT(out) etc.)
    (INTEGER,INTENT(inout) etc.)
    REAL(sp),INTENT(in)
    REAL(dp),INTENT(in)
    COMPLEX(sp),INTENT(in)
    COMPLEX(dp),INTENT(in)
    LOGICAL,INTENT(in)
    CHARACTER(12),INTENT(in)
    TYPE(xx),INTENT(in)
    ```

  - followed by things which are not arguments

    ```
    INTEGER
    REAL(sp)
    REAL(dp)
    COMPLEX(sp)
    COMPLEX(dp)
    LOGICAL
    CHARACTER(12)
    TYPE
    ```

  Within each class, put standard variables on the first line, followed by things with attributes like ALLOCATABLE, PARAMETER, etc., on subsequent lines, in whatever order seems aesthetically pleasing.

  Note also the CHARACTER(12), not CHARACTER*12, which is not in the Fortran 90 standard.

- Don't use tab characters anywhere in the code.
- All units for reading and writing are to be allocated unit numbers using the standard `open_units` procedure, of which you can several examples throughout the source.
- Initial letters of comments are to be in capitals. Comments are to be spelt correctly and end with a full stop/period (if they form a complete sentence). Comments must be useful.

```
! This is a legitimate comment.

! this is an illegitimate comment                             X

! tihs one is evn more illetigimate as i cant spell           XX

! allocate a
XXX (useless!)
  allocate(a(1))
```

- If you add a module USE statement anywhere, don't forget to change the dependency list in the `Makefile`. This can be done automatically by running `update-makefile-tool` in the `CASINO/src` directory.

- Module 'USE' statements should be in alphabetical order, followed by 'USE ONLY's in alphabetical order:

  ```
  USE a
  USE b
  USE c
  USE a1, ONLY : x
  USE b1, ONLY : x
  USE c1, ONLY : x
  ```

- Don't use 'return' at the end of each routine (unless necessary); just use END SUBROUTINE or END FUNCTION etc.

- Don't use GOTO statements if you can avoid them (which you can).

Note that the `casinostyle_checker` utility exists to help you ensure that your program or modifications to CASINO source files conforms to the style guidelines above. Simply supply the `.f90` files that you wish to check as command-line arguments.

A.3.1 C routines

The CASINO source contains two C routines `etime.c` and `alloc_shm.c`. Please make reference to these to get some idea of how to format new C routines. The only important point is not to use new-style C comments designated by `//`—there are still C compilers that we supposedly support which do not allow this. Use the old-style `/* */` form instead.

A.4 Content

CASINO uses the `git` revision control system (see `https://git-scm.com`). There is an official CASINO document written by PLR which explains the most important features of this system, including how to submit content for inclusion into the main distribution. It can be found in the `CASINO/doc` directory (*git_guide.pdf*) and probably online somewhere.

Access to the developer git repository may be granted by sending a request to Mike Towler (mdt26 at cantab.net) and then following the instructions at `https://vallico.net/casinoqmc/git-repository/`.

Note that modifications to CASINO will *not* be accepted unless the patches are done on the most recent development version of the code via the standard git mechanism.

Even if modifications are made in the correct manner, you may find that they are still not incorporated into the public release. If you want this to happen, there are a variety of ways to lower the energy barrier for the inclusion of your routines. The general theme here is to try to reduce the amount of effort that MDT has to perform to validate and check your code. If your submission includes lots of things from the 'Good things' list, and none at all from the 'Bad things' list, then inclusion is semi-automatic and will normally take place within days. We are aware that it requires a lot of work to do the things on the Good list but of course we will have to do it if you don't!

A.4.1 Good things

- Well-written, absolutely standard Fortran 90/95/2003 in the standard CASINO format. Do not use features from later releases of the Fortran standard, or non-standard extensions.

- The results of extensive testing of your routine with a range of examples, with before and after numbers (including timings). This should be done with the autotest suite (see the `README` file in `examples/TEST` and the discussion below).

- A detailed description of whatever your code is supposed to do. (I shouldn't have to go through each line of your code at the same detailed level you did, or I might as well have written it myself.)

- A bit of TeX for inclusion in the manual (if the code requires the user to know something over and above what the current manual describes). This should be well-written, comprehensive, and helpful.
- Some example input/output demonstrating the new capabilities, which MDT can just run and look at and perhaps incorporate into the `CASINO/examples` directory, without having to invent his own.
- Evidence of testing on serial and parallel machines. Remember that in general only the master processor should write to the output file. Also, CASINO is supposed to compile and work on single-processor machines *without* an installed MPI (Message Passing Interface) library. Use the fake `comms_serial.f90` module supplied both in the `utils` directory and with the main code (in general you only need worry about this if you introduce a call to an MPI routine which is not already used in CASINO and which has not been 'faked').
- If you use MPI calls when doing the parallel bits, try to ensure they are included in the MPI-1 standard since, somewhat unbelievably, there are some computers in the world which claim not to be able to support the 14-year-old MPI-2 standard. Currently, the only MPI-2 routine used in CASINO is `MPI_GATHER_IN_PLACE`, and PLR has implemented a fake version of this in the `comms_parallel_mpi1.f90` routine which can automatically be used on machines which do not have an MPI-2 installation (if their `CASINO_ARCH` file tells them so).
- Your routine uses the standard CASINO facilities for handling errors (`errstop`/`errwarn` etc.). These can be found in the `run_control.f90` module.
- Your routine implements something that we desperately want or is in the `TODO` list.

A.4.2 Bad things

- We do not sacrifice speed for additional functionality. Think of another way to write it if it slows the code down.
- We do not like to include things unless they are completely general. The development of general, complex electronic structure codes can be set back years by people choosing to implement functionality which works only for their current project. Nongeneral algorithms tend to get completely thrown away and re-implemented, which wastes the time of both parties. It is not usually much harder to write a general program than it is to write a specific one (we understand this is not always true!).
- CASINO has to be able to compile out of the box on any machine, and we cannot assume that every user has the ability to install particular libraries. You should not use *any* calls to external libraries that are not included in the CASINO distribution apart from MPI, BLAS and LAPACK. If you want to use any other library or third-party code in your addition to CASINO then you must ensure that there are **absolutely no licence restrictions** on including the library or third-party code in the CASINO distribution; you can then place the source code for the library in `CASINO/lib`.
- When writing to the output file, the new routines should not write out lots of weird arrays (whose meaning is understood only by the author) in an incredibly scruffy format full of spelling mistakes. In general, be as beautiful and informative as you can when writing to output, or MDT will just have to make it so (which takes ages). If there are all sorts of special cases which require different output, then so be it—`select case` and `if` blocks are very useful in this regard.
- Prior agreement for modification of the code should have been obtained. Unsolicited submissions are not accepted and can lead to moody lack of cooperation on our part.

A.5 Testing and debugging CASINO

Here is some advice on how to go about testing and debugging CASINO after you have made some modifications.

- Type *make debug* in `~/CASINO/src/` in order to compile the code with debugging options set. The resulting binary can be run by the runscript using `runqmc --debug` or `runqmc -d`.
- Remember to check that CASINO still works with OpenMP and OpenACC when you modify the code.
- The best compiler for debugging on PCs is NAG. MDT has recently discovered that the Cray compiler is pretty good too.
- Running GCC with Valgrind is also very effective for debugging, and can find memory issues missed even by the NAG compiler. Run CASINO using `runqmc --debug --valgrind`. If you wish to debug using Valgrind in parallel then you should first compile your own version of the OpenMPI library supplying the "–with-valgrind" option to the `configure` script; even then you should expect to see lots of (hopefully unimportant!) memory leaks that occur in OpenMPI, *not* in CASINO.
- It is a good idea to try running the code using several different compilers on several different architectures: this maximizes the chance of finding problems. There are bugs which require a combination of compilers to locate successfully. Note that you should always test that your modifications work on parallel computers, if possible.
- If you encounter a problem, try changing options in the `input` file before looking in the code. For example, does the problem only occur when a Jastrow factor is used? Does it only occur when the cusp correction is used? Try to obtain the simplest set of circumstances capable of reproducing the problem.
- Track down bugs either by (i) inserting write statements in the code and recompiling or (ii) using a debugger (such as gdb) in conjunction with binaries that have been compiled with the debugging flags set, which you can use by passing the `--debugger=`*debugger-binary* option to `runqmc`.
- Any utilities you write should also be tested by compiling and running with full debugging flags. Use, e.g., *man f90* to find out about the available debugging flags for your compiler.
- If you get stumped and have to send a bug report to us in Cambridge, then please make sure that everything we need to reproduce the problem is attached or otherwise available. Our record so far is an exchange of 24 emails—beginning with one stating 'CASINO doesn't work!'—before we discovered precisely what it was the user was complaining about (that is, what the problem was, rather than the solution). We shall leave it to you to guess which distinguished London-based professor we are referring to.

A.6 Performance

A.6.1 Internal timers

CASINO can print a detailed timing analysis at the end of the output file when it has finished a calculation. This information can be useful to the programmer in deciding what optimizations to do to make the program run faster. It is generally disabled by default since, annoyingly, the process of calculating the timings slows down the code significantly. To activate the detailed analysis, set **timing_info**=`T` in the input file.

The total time is printed as 'CPU' time, and the ratio between 'System' and 'User' time is displayed as well. These timings are broken down according to specific tasks, although by no means all activities are timed—only those thought likely to contribute significantly. For very small systems such as atoms a significant amount of CPU time will remain unaccounted for since the traditional heavyweight tasks are very fast. For medium and large systems, often the calculation of the orbitals and their derivatives will dominate, closely followed by Jastrow factors and Ewald interactions. In systems with pseudopotentials, the nonlocal integration is very expensive indeed (since it calls the orbital evaluation routine a lot) and this expense will increase the larger the nonlocal cutoff radius (lower **nlcutofftol** in the `input` file to see the effect of this).

The 'User' time is the time spent executing the instructions that make up the user code, including, e.g., calls to subroutine libraries linked to the code.

The 'System' time is the time spent in *kernel* mode, processing I/O requests, for example, or other situations which require the intervention of the operating system.

Together these comprise the total CPU time. Note that the User time is usually significantly greater than the System time. Indeed, a high ratio of system to user CPU time may indicate a problem. Exceptions such as page faults, misaligned memory references, and floating-point exceptions consume a large amount of system time. Note that time spent doing things like waiting for input from the terminal, seeking on a disk, or rewinding a tape does not appear in the CPU time; these situations do not require the CPU and thus it can (and generally does) work on something else while waiting.

The elapsed ('wallclock' or 'real') time is simply the total real time that passed during the execution of the program. This will typically be greater than the total CPU time due primarily to sharing the CPU with other programs. In addition, programs that perform a large number of I/O operations requiring more memory bandwidth than is available on the machine (i.e., the CPU spends significant time waiting for data to arrive from memory, or is paged or swapped) will show a relatively low ratio of CPU time used to elapsed time.

A.6.2 Profiling tools

To optimize the code it can be helpful to use tools such as gprof to investigate bottlenecks. Compile CASINO using `make prof` and run it using `runqmc --prof`, then use gprof to examine the resulting `.gmon` file. If you want to check that CASINO is making good use of your machine's memory cache then you can investigate using Cachegrind (a Valgrind tool).

Lots of other performance and optimization tools are available. If you find a tool that demonstrably allows you to improve CASINO then please let us know!

A.7 Bug reports

We cannot of course guarantee that CASINO is free of bugs, and if you find one, please tell us. If you don't know what the problem is yourself, than please include as much detailed information you can about the nature of the error in order to ensure a quick fix. In particular, please send us all the relevant input files to enable us to reproduce your problem.

A.8 Requests for new features

We are of course always happy to discuss such requests.

B Appendix 2: Automatic testing of CASINO

B.1 Overview

A facility for automatically testing CASINO can be found in `CASINO/examples/TEST`. This directory contains another directory called `Input`, which holds a large number of examples designed to test most aspects of CASINO, and a script called `autotest`, which will run CASINO for each of those examples in turn. This facility is intended to help developers ensure that they do not inadvertently break the code when adding new features.

B.2 Running the set of examples

The first time that `autotest` is run, a stable, trusted version of CASINO should be used to create a library of standard output files, with which subsequent results can be compared. Type `./autotest --library` to achieve this. (Note that the CASINO binary compiled with debugging flags will be used by default.)

Subsequently, just type *./autotest* to run the examples and compare the output with that in the library. The script will warn you if CASINO fails or halts unexpectedly, or if the output differs from that in the library (don't worry if it scrolls off the screen too fast; all output is saved in the `autotest.log` file). Note that rounding errors often affect the last couple of digits in the results of optimization calculations. Moreover, there are often trivial differences between the results produced with different compilers, so, when testing the code, it is best to use the same compiler as was used to generate the library.

The `autotest` script uses `runqmc` to run CASINO, and one may pass options to `runqmc` directly using the syntax '`--`⟨`extra-runqmc-options`⟩ (i.e., use a double hyphen to separate `autotest` options from the ones to be passed to `runqmc`, as in '`runqmc --nproc=100`'). This allows, for example, running `Shm` and `OpenMP` autotests, controlling the number of cores used on of parallel autotests, running the `debug`, `opt`, or `dev` versions of the code, and so on.

To skip a particular example use, e.g., '`--not carbon_atom`'. To run the test for a particular example only, use e.g. '`--only neon_atom`'. Use `--missing` to only run tests which have not been run already, or were interrupted; use `--error` to only run tests which have not been run already, or were interrupted, or resulted in an error; use `--mismatch` to only run tests which have not been run already, or were interrupted, or did not match the library output. If you want to get rid of all the results (including the library results), either remove the `Library` and `Output` directories by hand, or type `./autotest --clean`. Type `./autotest --help` to get a full list of options.

The facility is good at picking up minor bugs when used in conjunction with the NAG compiler with debugging flags. The facility should be run after any substantial modification of CASINO.

B.3 Adding a new example

If a new feature is added to CASINO then an appropriate example should be added to the automatic testing facility. Create a new directory in `Input` with name describing the system, and create one or more subdirectories in it with names indicative of the run type; see other examples for guidance. Let's refer to these directories as `system` and `system/run`, respectively. Place common files (e.g., `xwfn.data` files, pseudopotentials, etc) under `system`, and files which change from run to run (e.g., `input`) in the individual `system/run` subdirectories.

Note that the examples should run very quickly; it doesn't matter whether good results will be produced. The great majority of bugs will show up when the results of the modified version of CASINO are compared with the 'standard' version.

The **random_seed** keyword should be set to a definite value (usually 'standard') in `input` files for `autotest`. Otherwise, it default to 'timer' and a different random-number sequence will be used every time `autotest` is run.

B.4 Using `git-bisect` with autotest

Bisecting is a very simple, yet powerful, feature of `git` which aids locating the commit at which a regression was introduced, and thus (hopefully) determine how to fix it.

Suppose you have a clean working directory (no modifications over the last commit), and when you run the autotest you find a crash, an output mismatch, a large increase in CPU time or any other undesirable circumstance for a particular example, `system/run`. You know that this did not happen a few hundred commits ago when you last ran the autotest at commit `123456`, say.

In that case you may want to use `git-bisect`. Open a new terminal and change into `~/CASINO`, so that you have one terminal to run the autotest (the 'autotest terminal') and another to do version switching and compiling (the 'git terminal'). In the 'git terminal' type

```
git bisect start
git bisect bad         # the current commit has the bug
git bisect good 123456 # an old commit (123456) does not have the bug
```

`git` will then jump to an intermediate commit. Type `make` to recompile CASINO at that point, then switch to the 'autotest terminal' and type, for example,

```
./autotest --only system/run
```

Then switch to the 'git terminal' and, if the problem is not present in this run, type

```
git bisect good
```

else type

```
git bisect bad
```

and type `make` again to rebuild the code at the next commit `git` decides to test, and so on. If at any point you land on a commit which you would rather skip (it won't compile, or it has a known problem which won't let you see whether the bug you are looking for is there or not), type

```
git bisect skip
```

and `git` will pretend the commit is not there.

After a few bisects `git` should be able to tell you which commit introduced the regression. To return to the initial state (and this can be done at any point during the bisect operation), type

```
git bisect reset
```

For more information see the `git-bisect` *manpage*.

C Appendix 3: Converting CASINO v1.x input files to CASINO v2.x format

There is a utility called `update_input` included with CASINO which should be capable of updating most input files from version prior to 2.0 into ones suitable for the current code. As the changeover took place ages ago (2003?) hardly anyone will want to do this, and this appendix is in serious danger of being deleted. However, if you really do want to do such conversions, and have any problems using this script, then here are some useful tips.

- The formats of the Jastrow and backflow sets in the `correlation.data` file are exactly the same as the formats of the old `jastrow.data` and `backflow.data` files: therefore you simply need to copy the contents of the old files into a file called `correlation.data`. If you don't have a `backflow.data` file then you can simply rename `jastrow.data` as `correlation.data`.
- If your calculation uses the archaic `jasfun.data` file of CASINO's old Jastrow factor or the `swfn.data` file used to store 'spline' orbitals then it is not possible to continue your calculation using version 2, as both the old Jastrow factor and the spline representation of orbitals have been dropped from CASINO. You must therefore use an archived version of CASINO version 1.
- Any `blwfn.data` files should be regenerated. The file format can also be updated by hand—ask Neil to do this.
- Delete any `eepot.data` and `density.data` files and regenerate them as a `mpc.data` file using the **runtype** = 'gen_mpc' option. Alternatively there is a `make_new_mpc` utility which will do a direct conversion.
- The **hist_update** utility should be used to convert old-format `vmc.hist`, `dmc.hist` and `dmc.hist2` files to the new-format `vmc.hist` and `dmc.hist` files.
- The following changes should be made to the `input` file (it is probably best to copy an `input` file from the new examples and just change the few parameters in the new 'system-specific' section at the top):
  1. **irun** = 1 should be replaced by **runtype** = 'vmc' and **use_jastrow** = `F`; **irun** = 2 should be replaced by **runtype** = 'vmc' and **use_jastrow** = `T`; **irun** = 3 should be replaced by **runtype** = 'dmc'; **irun** = 4 should be replaced by **runtype** = 'opt'; **irun** = 5 or 6 should be replaced by **runtype** = 'gen_mpc'.
  2. The number of variance-minimization cycles is specified by the **opt_cycles** keyword. A post-fit VMC calculation will be performed if the **postfit_vmc** keyword is set to `T`. This only has effect if the **runtype** = 'vmc_opt' or 'opt_vmc'.
  3. The keyword **opt_pairing** has been renamed to **opt_orbitals**. Its meaning is unchanged.

4. **btype** = 7 should be replaced by **btype** = 4 (and in version 2.1 **btype** was marked redundant and replaced by the **atom_basis_tye** keyword).
5. The **wavefunction** block in the `input` file is no longer used. If you used it to specify multideterminant or excited-state wave functions then you should move the data to an 'MDET' block in `correlation.data`. The format of the latter is the same as that of the former except that (i) the 'MDET' block has a title (and comment lines introducing the title and multideterminant data) and (ii) the multideterminant expansion coefficients must be followed by an integer label and an 'optimizable' flag. See Sec. 7.4.5.
6. The keyword **energy_cutoff** has been replaced with separate **mpc_cutoff** and **expval_cutoff** keywords specifying the cutoff values for the G vector sets used in `mpc.data` generation and expectation value accumulation.
7. The keywords **denft_threshold**, **dbar_tolerance**, **vm_mode**, **use_newjas**, **use_coeff_file** and all keywords previously flagged as redundant have been deleted.
8. The keyword **use_newopt** should be removed; to use the 'newopt' method (now varmin-linjas), set the keyword **opt_method** to `varmin-linjas`.

D Appendix 4: Specification of the format of the `correlation.data` file

1. The `correlation.data` file contains: (i) the data specifying the Jastrow factor (the old `jastrow.data`); (ii) the data specifying the backflow function (the old `backflow.data`); (iii) the data specifying the orbitals occupied in each determinant together with determinant expansion-coefficient data; (iv) the data specifying the geometry and wave function of electron-gas and electron–hole-gas systems (these data may be considered to be optimizable parameters). Any new optimizable parameters, even ones relating to the orbitals specified in `xwfn.data`, should be placed in `correlation.data`.

2. The precise definitions of the Jastrow and backflow data sets are given in Secs. 7.4.2 and 7.4.4, respectively.

3. If excitation data or determinant expansion-coefficient data are given in `correlation.data` then this overrides the ground-state definition given in the `xwfn.data` file.

4. The `correlation.data` file uses the following format:

   (a) The data should be broken up into 'sets'. For example, there should be separate sets of data specifying the Jastrow factor and the backflow function.

   (b) Each distinct set of data XXX should begin with a line 'START XXX' and end with a line 'END XXX'.

   (c) Sets of data may be nested.

   (d) Any blank lines immediately before a line 'START XXX' or after 'END XXX' should be ignored. No other blank lines are allowed in the file.

   (e) Parameter values should be specified by (i) a line of text defining the parameter(s); (ii) the unformatted parameter value(s) (it is possible to have more than one parameter per line and there may be several lines of parameters). The precise format that is required should be clearly specified by the line of text.

   (f) Each major set of data (that is, one that is not nested within another) should have a title as the first input parameter.

   (g) The order in which major sets are supplied should not matter. Major data sets that are not required by CASINO may be omitted.

   (h) All lines with optimizable parameters are to contain a flag specifying whether the parameters in that line are free to be varied (1) or not (0). This flag should be given after the parameter value(s) on that line.

   (i) Any comments after the desired parameter value (and optimizable flag) should be ignored.

(j) If a set of parameters is the last item in data set XXX (that is, immediately before 'END XXX') then it may be acceptable for the user to omit some or any of the parameters. Any parameters that are not specified are assumed to take value 0, and, if relevant, the corresponding 'optimizable' flag is assumed to be 1 ('true').

(k) When CASINO produces a `correlation.out` file, all optimizable parameters should be written out, followed by their optimizable flag (as given in `correlation.data`, or '1' by default if not specified in `correlation.data`), optionally followed by a '!' and then a brief definition of the parameter.

(l) It is acceptable to have input parameter values in `correlation.data` which specify that default values are to be substituted. However, the *actual* parameter values should always be written out to `correlation.out`.

5. If correlated sampling is introduced into CASINO, it may be necessary to define several `correlation.data` files. These should be named `correlation.data.1`, `correlation.data.2`, `correlation.data.3`, ...

6. `correlation.out` is of the same format as `correlation.data`. It is produced by CASINO during the optimization of the wave function, and contains the optimized parameter values.

7. Optionally, a header may be given. Comments may be provided over several lines between 'START HEADER' and 'END HEADER'. The header should not be nested within any other data sets.

8. Optionally, a version number can be given. This is a whole number placed between the lines 'START VERSION' and 'END VERSION'. The version number will be increased each time the format of the `correlation.data` file is changed. The version-number set should not be nested within any other data sets.

E Appendix 5: CASINO system-specific data files

The `arch/data` subdirectory contains architecture data files which define system-specific parameters for compiling and running CASINO. Include files in this directory are named `$CASINO_ARCH.arch`, where `CASINO_ARCH` is an environment variable which should be defined in your shell session and determines which set of parameters to use, including compiler name, flags, library locations, how to submit jobs, etc.

This system is documented here, and in `CASINO/arch/README` (of which the following is a copy).

```
I. Types of CASINO_ARCHs
========================
While there is no fundamental difference between CASINO_ARCHs, we define two
conceptual types for convenience, which simply differ in purpose and naming
convention:

* "Generic" CASINO_ARCHs are intended to represent a class of systems.  Their
  name is typically of one of these forms:
  - Single-processor workstations:

    <system>-<compiler>

  - Multi-processor workstations:

    <system>-<compiler>-parallel

  - Clusters with queueing systems:

    <system>-<compiler>-<queueing-system>-parallel

* "Extended" CASINO_ARCHs are intended to represent specific systems, and are
  usually modifications to existing generic CASINO_ARCHs.  Their name is of the
  form:
```

```
  <generic-name>.<specific-system-name>

  The corresponding .arch file is typically intended to "include" its generic
  counterpart, if it exists, but again this is just a guideline.

See the files in the CASINO/arch/data directory for examples of both generic
and extended CASINO_ARCH names.


II. Data file format
====================
The architecture data file is structured in three sections:

* The first section contains architecture parameters ("tags", see below).

* The second section contains any optional 'include' statements, always
  of the form:

    include $(INCBASE)/<file-name>.arch

* The third section contains Makefile definitions (see below).

The file is designed to be 'include'd by the Makefiles in the distribution,
hence the lines in the first section start with the hash character '#' so that
they are ignored by 'make'.  The first section is read by the 'runqmc' script
and by the CASINO/arch/arch_info.sh assessment tool.

When editing .arch files, you are encouraged to use the syntax highlighting
file provided with CASINO, see CASINO/data/syntax for more info.


II.1. Data file tags
====================
The beginning of architecture data files contain tags which can be used to
identify the machine it is designed for, and to specify how to run CASINO on
that machine.  Tag contents can be single- or multi-line.

Single-line tags are specified this way:

  #-! TAG-NAME: tag-value
  #-!  tag-value-continue
  #-!  ...
  #-! ANOTHER-TAG_NAME: ...

(no indentation before '#-!', *one* space between '#-!' and TAG-NAME, no space
between TAG-NAME and the colon; one or more spaces between the colon and
tag-value; two or more spaces between '#-!' and tag-value-continue).  Leading
and trailing blanks are removed from tag-value and continuation lines, and
lines are concatenated with a single blank between them to form the final
value.

Multiline tags are specified this way:

  #-! TAG-NAME:
  #-!  tag-value-line-1
  #-!  tag-value-line-2
  #-!  ...
  #-! ANOTHER-TAG-NAME: ...

(same format restrictions as above, and two or more spaces between '#-!' and
tag-value-line-i).  Leading and trailing blanks are removed from each line
read.

Note that tags are inherited via "include" statements, read in the order in
which the include statements are written, with the tags in a file overriding
those in the included files regardless of the relative placement of tags
```

and include statements.

Below are tables containing the full list of tags, classified according to their purpose. The values under the 'L' heading specify if the tag's value is supposed to be a single line ('S') or multiple lines ('M').

II.1.i. Tags for manual CASINO_ARCH detection

TAG NAME	L	Description & examples
DESCRIPTION	S	human-readable description of target system(s)
MAINTAINER	S	name and email of maintainer. E.g., Mike Towler <mdt26 @ cantab.net>
DATE	S	date written
COMMENT	S	any other relevant comments
QUEUEING_SYSTEM	S	(TYPE=cluster only) queuing system. E.g., PBS

II.1.ii. Tags for automatic CASINO_ARCH detection

TAG NAME	L	Description & examples
ARCH	S	comma-separated list of allowed architecture patterns as obtained from ‘uname -m‘. E.g., i?86, x86_64
KERNEL	S	comma-separated list of allowed kernel name patterns as obtained from ‘uname -s‘. E.g., Linux, GNU/kFreeBSD
OS	S	comma-separated list of allowed operating system names as obtained from ‘uname -o‘. E.g., GNU/*
DISTRIBUTION	S	comma-separated list of allowed Linux distribution strings, as obtained from (in order of decreasing preference): - ‘lsb_release -ds‘ - ‘head -n 1 /etc/redhat-release‘ - ‘head -n 1 /etc/SuSE-release‘ - Debian GNU-Linux ‘head -n 1 /etc/debian_version‘ E.g., *buntu *, *SUSE 11.3*
HOSTNAME	S	comma-separated list of allowed host name patterns of the target computer. E.g., pc*.tcm.phy.cam.ac.uk, cluster.tcm.* An empty value means that the hostname should not be checked against anything.
DOMAIN	S	comma-separated list of allowed domain name patterns of the target computer. An empty value means that the domain should not be checked against anything. This is provided since some distinct machines have the same hostname but different domains.

```
F90_VERSION              S  comma-separated list of allowed compiler version
                            patterns that should be matched.  E.g.,
                              <= 2.1, 3.2.*, 4.0.*, > 4.1

COMMAND_CHECK_F90_VERSION M  bash code block that outputs the compiler
                            version, which will be matched against
                            F90_VERSION.  E.g.,
                              set -- $(&F90& --version | head -n 1)
                              echo ${*:$#}
                            The code may make use of the &F90& variable,
                            which is replaced by the value of the Makefile
                            variable F90.

COMMAND_CHECK_F90        M  bash code block that checks if the compiler
                            is indeed the desired compiler (e.g., to avoid
                            false positives with the usual 'f90', 'mpif90',
                            etc).  The code should print '1' if the check
                            succeeds, anything else otherwise.  E.g.,
                              set -- $(&F90& --version | head -n 1)
                              [ "${*:1:2}" = "GNU Fortran" ] && echo 1
                            The code may make use of the &F90& variable,
                            which is replaced by the value of the Makefile
                            variable F90.

CC_VERSION               S  | like their *F90* counterparts, but for the C
COMMAND_CHECK_CC_VERSION  M  > compiler
COMMAND_CHECK_CC         M  |

CXX_VERSION              S  | like their *F90* counterparts, but for the C++
COMMAND_CHECK_CXX_VERSION M  > compiler
COMMAND_CHECK_CXX        M  |
-------------------------------------------------------------------------------


II.1.iii. Run-time tags
=======================

  If a run-time tag's name is preceded by an asterisk (e.g., '*RUN_SINGLE'),
  the contents are interepreted as multi-line bash code whose standard output
  provides the value of the tag -- in the case of single-line tags, multiple
  output lines are appended a blank and concatenated, and in the case of
  multi-line tags, each output line becomes an individual line in the tag's
  value.

  See 'variable substitution' section below for a list of allowed
  run-time variables.

TAG NAME            L  Description & examples
------------------------------------------------------------------------------
FORCE_PATH          S  comma-separated list of path patterns under one of
                       which calculations must be run.  E.g.,
                         /work, /scratch*
                       would allow calculations under /work/john/heg,
                       /scratch/mdt/h2o, or /scratch_large/benzene but not
                       under /work1/dna, for example.

RUN_SINGLE          S  command to run direct single-processor CASINO
                       calculations.  This is '&BINARY&' by default.

RUN_PARALLEL        S  (TYPE=parallel|cluster only) command to run direct
                       multi-processor CASINO calculations.  This is
                       'mpirun -np &NPROC& &BINARY&' by default.

CLUSTER_RUN_MODE    S  (TYPE=cluster only) determines if the cluster requires a
                       batch script to submit a job ('batch', default, triggers
                       use of SCRIPT_HEAD, SCRIPT_RUN and SUBMIT_SCRIPT), or
```

```
                        if a command for submitting a job to the queue is
                        available ('direct', triggers use of RUN_CLUSTER).

RUN_CLUSTER          S  (TYPE=cluster only) only used if CLUSTER_RUN_MODE is
                        'direct'; command to submit a CASINO job to the cluster
                        queue directly.  Undefined by default.  E.g., 'bgrun
                        -mode VN -np &NPROC& -exe &BINARY&'

CORES_PER_NODE       S  (TYPE=parallel|cluster only) number of cores on the
                        workstation, or on the login node of the cluster; '1'
                        by default (but note that gnulinux.arch contains code
                        that counts the number of cores on the current node).

CORES_PER_NODE_CLUSTER  S  (TYPE=cluster only) number of cores per node in
                        the cluster compute nodes, if it differs from the
                        number of cores in the login node - if defined, its
                        value overrides that of CORES_PER_NODE in clusters.

SCRIPT_HEAD          M  (TYPE=cluster only) only used if CLUSTER_RUN_MODE is
                        'batch'; header of submission script.

SCRIPT_RUN           M  (TYPE=cluster only) only used if CLUSTER_RUN_MODE is
                        'batch'; line (or set of lines) in the submission script
                        where CASINO is run, optionally surrounded by extra
                        bash code as required by the machine.

SUBMIT_SCRIPT        S  (TYPE=cluster only) only used if CLUSTER_RUN_MODE is
                        'batch'; command to submit the submission script.  This
                        is 'qsub &SCRIPT&' by default.

ALLOWED_NCORE        S  (TYPE=cluster only) blank-separated list of allowed
                        number of CPU cores to reserve or %<number> to specify
                        "any integer multiple of <number>".  E.g.,
                          1 2 4 8 16 32 %3

ALLOWED_NNODE        S  (TYPE=cluster only) blank-separated list of allowed
                        number of physical nodes to reserve or %<number> to
                        specify "any integer multiple of <number>".  E.g.,
                          1 2 4 8 16 32 %3

MIN_NCORE            S  (TYPE=cluster only) minimum/maximum number of CPU cores
MAX_NCORE               that can be reserved.

MIN_NNODE            S  (TYPE=cluster only) minimum/maximum number of physical
MAX_NNODE               nodes that can be reserved.

MIN_NNODE_ENSEMBLE   S  (TYPE=cluster only) minimum number of physical nodes
                        for a single job in an ensemble of multiple jobs (e.g.
                        certain Blue-Gene Qs cannot run jobs of less than 128
                        nodes because of the hardware).

TIME_FORMAT          S  (TYPE=cluster only) a string determining how the
                        &WALLTIME& variable is to be constructed when specifying
                        the requested job time in the submission script.  In
                        this variable, D, H, M, and S are evaluated to days,
                        hours, minutes and seconds, respectively; if the letters
                        are repeated, the respective number is padded with
                        zeroes on the left to fill the number of digits given by
                        the number of repetitions. E.g., for a CASINO-formatted
                        walltime of 5h43m31s,
                          H:MM:SS
                        would make &WALLTIME& expand to "5:43:31", while
                          DD:HH:M:SSSS
                        would make &WALLTIME& expand to "00:05:43:0031", and
                          MMMM minutes SS seconds
                        would make &WALLTIME& expand to "0343 minutes 31
```

```
                       seconds".

MIN_WALLTIME        S  (TYPE=cluster only) minimum/maximum wall time that can
MAX_WALLTIME           be requested, in CASINO format.  E.g.,
                         1d4h51m

WALLTIME_CODES      S  (TYPE=cluster only) blank-separated list of associations
                       between strings and associated wall times in CASINO
                       format.  The strings will be used as the &WALLTIME&
                       variable on machines which force discrete job times and
                       uses custom codes to identify them.  E.g.,
                         u=24h t=12h s=6h

ALLOWED_WALLTIME    S  (TYPE=cluster only) blank-separated list of wall times
                       that can be requested on a machine which forces discrete
                       job times.  One need not supply this if WALLTIME_CODES
                       is specified.  If both are, the intersection of both
                       lists will take effect.  If MAX_*TIME is specified,
                       both the limits and the discrete list constrain the
                       available runlengths.

MIN_CORETIME        S (TYPE=cluster only) minimum/maximum sum of time on all
MAX_CORETIME          requested cores, in CASINO format.  If both MIN_WALLTIME
                      and MIN_CORETIME, or MAX_WALLTIME and MAX_CORETIME, are
                      specified, the most restrictive value takes effect.  This
                      is particularly useful if there is an accounting credit
                      system in place on the machine, so one can provide a
                      *MAX_CORETIME tag which returns the time remaining in
                      the user's account.

MAX_NJOBS           S (TYPE=cluster only) On some machines there is a maximum
                      number of jobs that may be flagged by a single runqmc
                      command (e.g. on Titan only 100 aprun processes are
                      permitted per job submission script). The maximum may
                      be specified using this tag, so that runqmc can complain
                      about this problem.

RELPATHNAMES        S (TYPE=cluster only) If set to 'yes', this flags the
                      existence of a machine with completely different
                      filesystems on the login nodes and the compute nodes
                      (and which therefore requires 'staging' of the CASINO
                      input and output files).  This necessitates the use of
                      relative pathnames rather than absolute pathnames and
                      a more elaborate clean-up procedure.

SCRIPTCSH           S (TYPE=cluster only) This should be set to 'yes' on
                      extremely unusual machines which insist that batch
                      scripts be written in csh, instead of the standard
                      bash. The resulting batch scripts may have reduced
                      functionality; in particular they do not yet support
                      twist-averaging calcs.

MAKE_EXECUTABLE     S Non-standard versions of 'make' available on some
                      machines may not be able to compile CASINO. If this
                      is the case, the need to use an alternative version
                      of make (such as GNU 'gmake', which definitely works) can
                      be specified by setting the value of the MAKE_EXECUTABLE
                      tag. Merely setting a shell alias make='gmake' will not
                      work as the alias is not available to the install script
                      (though the alias might be necessary if you want to
                      compile CASINO by hand, rather than via the install
                      script).
------------------------------------------------------------------------------


II.2. Variable substitution
```

```
===========================

The variable substitution system in the .arch files is very flexible.
In the runtime section, any tag is allowed to depend on any variable, and
an appropriate evaluation order will be computed.  Together with the use
of INTERNAL and USER variables (see below), this enables the .arch system
to cater for very complex set-up requirements.


II.2.i. List of variables available to automatic-detection tags
================================================================
- &F90& : value of the F90 makefile tag
- &CC&  : value of the CC makefile tag
- &CXX& : value of the CXX makefile tag


II.2.ii. List of variables available to run-time tags
=====================================================
- &TYPE&            : TYPE of the machine (not necessarily as defined in the
                      Makefile section of the .arch file (see below); clusters
                      can be used as workstations, and multi-processor
                      workstations can run in non-MPI mode)
- &F90&             : value of the F90 makefile tag
- &CC&              : value of the CC makefile tag
- &CXX&             : value of the CXX makefile tag
- &OUT&             : output file name (used for stdout and stderr)
- &NPROC&           : (TYPE=parallel|cluster only) number of processes to run
                      (per job)
- &NNODE&           : (TYPE=cluster only) number of physical nodes to use (per
                      job)
- &NCORE&           : (TYPE=parallel|cluster only) number of CPU cores to
                      reserve (per job) -- this is exactly NNODE*CORES_PER_NODE,
                      and is provided as a (redundant) convenience
- &NJOB&            : number of simultaneous jobs being run
- &NPROC_TOTAL&     : number of processes/nodes/cores used in total for all
  &NNODE_TOTAL&       simultaneous jobs
  &NCORE_TOTAL&
- &PPN&             : (TYPE=parallel|cluster only) number of processes to run
                      per physical node (per job)
- &TPP&             : (TYPE=parallel|cluster only) number of threads to run per
                      process
- &TPN&             : (TYPE=parallel|cluster only) number of threads to run per
                      physical node (per job); this is TPP*PPN
- &WALLTIME&        : (TYPE=cluster only) wall time limit
- &SCRIPT&          : (TYPE=cluster only) submission script
- &BINARY&          : full binary pathname
- &BINARY_ARGS&     : list of command line arguments to be used with the
                      binary executable. CASINO does not require such
                      arguments, but other codes which make use of the
                      CASINO architecture system do (e.g., to run PWSCF, one
                      might write 'pw.x -pw2casino -npool 4 < in.pwscf >>
                      out.pwscf') where everything after the pw.x counts as
                      a command line argument.

II.2.iii. Environment variables
===============================
&ENV.<variable>& will expand to the value of environment variable <variable>
(under the directory under which the runscript is invoked, so be careful with
the usage of, e.g., ENV.PWD, etc).

Example:
  #PBS -M &ENV.USER&


II.2.iv. Internal variables
===========================
```

These are only available to run-time tags.

Internal variables are defined and set within the .arch file. Its value
is defined via the runtime tag INTERNAL.<variable>, and &INTERNAL.<variable>&
is the variable which expands to its value. These are useful for defining
often-used intermediate values from a single block of code.

See example in section II.4.

II.2.v. User variables

These are only available to run-time tags.

User variables are custom variables which can be set from the command line,
or take their values from the defaults defined in the .arch file:

```
- The tag USER.DESCRIPTION.<variable> gives a description of the purpose of
  this variable which is displayed to the user when --help is requested.
- The tag USER.DEFAULT.<variable> defines the default value of the variable.
- The tag USER.ALLOWED.<variable> defines a set of blank-separated allowed
  values that the variable can take -- if USER.DEFAULT.<variable> is not
  specified, the first value in this list becomes the default value.
- The tag USER.MIN.<variable> defines the minimum value that integer variable
  <variable> can take.
- The tag USER.MAX.<variable> defines the maximum value that integer variable
  <variable> can take.
- The variable &USER.<variable>& expands to the value of variable.
```

II.3. Makefile section

There are many variables that can be defined in the .arch files to modify
the build process, although only a few of them are absolutely necessary to
get CASINO to compile.

```
* Required variables:
  - TYPE      : omit or set to 'single' for single-processor machines, set to
                'parallel' for multi-processor workstations and set to
                'cluster' for clusters with batch-queueing systems
  - F90       : name of the Fortran compiler binary
  - CC        : name of the C compiler binary
  - NEED_ETIME: omit or set to 'no' if the Fortran compiler supports the ETIME
                extension, set to 'yes' otherwise

* Recommended variables:
  - CXX          : name of the C++ compiler binary (used by a single converter
                   utility at present, which will be skipped from compilation
                   if no C++ compiler is available)
  - FFLAGS_opt   : flags for the Fortran compiler with full optimization
  - FFLAGS_debug : flags for the Fortran compiler with full debugging
  - CFLAGS_opt   : flags for the C compiler with full optimization
  - CXXFLAGS_opt : flags for the C++ compiler with full optimization

* Optional variables controlling features:
  - HAVE_BLAS        : omit HAVE_BLAS/HAVE_LAPACK or set them to 'no' if you
    HAVE_LAPACK        would like to use the BLAS/LAPACK libraries provided with
    LDBLAS_yes         the CASINO distribution. Otherwise, set the HAVE_
    LDLAPACK_yes       variables to 'yes' and the LDBLAS_yes/LDLAPACK_yes
                       variables to the '-l' flags required to link the
                       optimized BLAS/LAPACK libraries in your system.
  - SUPPORT_OPENMP   : omit SUPPORT_OPENMP or set it to 'no' if your compiler
    FFLAGS_OPENMP_yes  does not support OpenMP or you wish to disable the
                       ability to compile it, else set SUPPORT_OPENMP to 'yes'
                       and set FFLAGS_OPENMP_yes to the flags required by your
                       compiler to enable OpenMP extensions, e.g., '-openmp'
  - SUPPORT_OPENACC  : omit SUPPORT_OPENACC or set it to 'no' if your compiler
    FFLAGS_OPENACC_yes does not support OpenACC or you wish to disable the
                       ability to compile it, else set SUPPORT_OPENACC to 'yes'
                       and set FFLAGS_OPENACC_yes to the flags required by your
                       compiler to enable OpenACC extensions, e.g., '-openacc'
  - SUPPORT_SHM      : omit SUPPORT_SHM or set it to 'yes' if your machine
    CFLAGS_SHM         supports either SysV or POSIX SHM.  Set CFLAGS_SHM to
                       the C compiler flags to compile SHM support - these are
                       expected to be either -DSHM_SYSV or -DSHM_POSIX, which
                       select which version of SHM to use.

* Optional variables controlling the compile/run environment:
  - ENVIRONMENT_COMMAND: single-line command to run before compiling and
                         running.  This is useful where environment variables
                         need to be set (e.g., 'export MPI=OpenMPI') or in
                         clusters where the 'module' environment
                         handling system is used (e.g., 'module load
                         default-infinipath').  Multiple commands separated
                         by semicolons can be given.  The only variables
                         that may be referenced are pre-existing environment
                         variables, and should be referenced using the syntax
                         '${variable}' rather than '$variable'; e.g.:
                           export PATH=${PATH}:/path/to/f90
                         is valid, while
                           export PATH=$PATH:/path/to/f90
                         and
                           export PATH1=/path/to/f90 ;\
                           export PATH=${PATH}:${PATH1}
                         are not valid.  (The reason for this restriction is
                         the peculiarities of how 'make' and 'sh' interact.)

* Variables controlling Fortran and MPI libraries:
  - MPI_VERSION   : major version of the MPI library.  Allowed values are 1
                   and 2 (default).
  - ISOVAR_VERSION : version of iso_varying_string[1-2].f90. Allowed values
                   are 1 (default) and 2. The option 2 should only be used with
                   compilers which refused to compile iso_varying_string1.f90
                   (such as the Hitachi ofort90 compiler).
  - LIB_PATH      : '-L' options to add library search paths, e.g.,
                   '-L/usr/opt/mpi/lib' (add one '-L' option for each path to
                   be added)
  - INCLUDE_DIR   : '-I' options to add include search paths, e.g.,
                   '-I/usr/opt/mpi/include' (add one '-I' option for each path
                   to be added)
  - LDLIBS_all    : '-l' options to link libraries, e.g., '-lmpi' (add one '-l'
                   option for each library to be linked)

* Variables defining a different compilation parameters for the utilities.
  This is useful when one needs to cross-compile the sources to run on the
  compute nodes of a cluster, but still requires a native compiler to be
  able to build the utilities. The main keyword is:
  - UTILS_MODE : set to 'native' if the utilities require a compiler
                 different from the one used for main binary, leave empty
                 otherwise.

  The following tags are of the form <X>_NATIVE, and have the same meaning
  as their <X> counterparts, but only used for the utilities if UTILS_MODE
  is set to 'native':
  - F90_NATIVE FFLAGS_opt_NATIVE FFLAGS_all_NATIVE
    LDF90_NATIVE LDFLAGS_opt_NATIVE LDFLAGS_all_NATIVE
    CC_NATIVE CFLAGS_opt_NATIVE CFLAGS_all_NATIVE
    LDC_NATIVE LDCFLAGS_opt_NATIVE LDCFLAGS_all_NATIVE
    CXX_NATIVE CXXFLAGS_opt_NATIVE CXXFLAGS_all_NATIVE
    LDCXX_NATIVE LDCXXFLAGS_opt_NATIVE LDCXXFLAGS_all_NATIVE
    HAVE_BLAS_NATIVE LDBLAS_yes_NATIVE LDBLAS_no_NATIVE
    HAVE_LAPACK_NATIVE LDLAPACK_yes_NATIVE LDLAPACK_no_NATIVE
    LDLIBS_opt_NATIVE LDLIBS_all_NATIVE
    ENVIRONMENT_COMMAND_NATIVE

* Rarely needed variables:
  - CFLAGS_F90_INTERFACE : flags required by the C compiler to correctly
                           interface with the Fortran compiler.  The value will
                           depend on the requirements of the Fortran compiler,
                           not on the C compiler.  Possible values are:
                            <empty>
                            -DF90_CAPITALS
                            -DF90_DOUBLE_UNDERSCORE
                            -DF90_DOUBLE_UNDERSCORE -DF90_CAPITALS
                            -DF90_NO_UNDERSCORE
                            -DF90_NO_UNDERSCORE -DF90_CAPITALS
  - FFLAGS_libs          : flags to compile the provided BLAS/LAPACK, if
                           different from FFLAGS_opt
  - FFLAGS0_libs         : non-optimized flags to compile sensitive functions
                           in the provided BLAS/LAPACK, if different from '-O0'
  - MODNAME_BUG          : omit or set to '0' if your compiler does *not*
                           suffer from the 'modname' bug, else set to '1'
  - FFLAGS_dev           : flags for the Fortran compiler for the dev and prof
    FFLAGS_prof            versions
  - FFLAGS_all           : flags for the Fortran compiler for all versions;
                           these are appended to those for the individual
                           versions
  - LDFLAGS_opt          : flags for the linker (if different from those for
    LDFLAGS_debug          the compiler) for the individual versions
    LDFLAGS_dev
    LDFLAGS_prof
  - FFLAGS_OPENMP_no     : flags required by your compiler to *disable* OpenMP
                           support, in compilers where it is enabled by default
  - CFLAGS_OPENMP_no     : flags required by your C compiler to *disable*
                           OpenMP support, in compilers where it is enabled by
                           default
  - LDFLAGS_OPENMP_yes   : flags required by your linker to enable OpenMP
                           support, if different from those for the compiler
  - LDFLAGS_OPENMP_no    : flags required by your linker to *disable* OpenMP
                           support, if different from those for the compiler
  - FFLAGS_OPENACC_no    : flags required by your compiler to *disable* OpenACC
                           support, in compilers where it is enabled by default
  - LDFLAGS_OPENACC_yes  : flags required by your linker to enable OpenACC
                           support, if different from those for the compiler
  - LDFLAGS_OPENACC_no   : flags required by your linker to *disable* OpenACC
                           support, if different from those for the compiler
  - LDLIBS_opt           : '-l' options for the individual versions; LDLIBS_all
    LDLIBS_debug           will be appended to these
    LDLIBS_dev
    LDLIBS_prof
  - CFLAGS_opt           : flags for the C compiler for the individual versions
    CFLAGS_debug
    CFLAGS_dev
    CFLAGS_prof
  - CFLAGS_all           : flags for the C compiler for all versions; these are
                           appended to those for the individual versions
  - CXXFLAGS_opt         : flags for the C++ compiler for the individual
    CXXFLAGS_debug         versions
    CXXFLAGS_dev
    CXXFLAGS_prof
  - CXXFLAGS_all         : flags for the C++ compiler for all versions; these
                           are appended to those for the individual versions
  - LDC                  : linker, linker flags and linker '-l' options for
    LDCFLAGS_opt           compiling pure C applications.  LDC defaults to CC
    LDCFLAGS_all           and LDCFLAGS_* default to CFLAGS_*.
    LDCLIBS_opt
    LDCLIBS_all
  - LDCXX                : linker, linker flags and linker '-l' options for
    LDCXXFLAGS_opt         compiling pure C++ applications.  LDCXX defaults to
    LDCXXFLAGS_all          CXX and LDCXXFLAGS_* defaults to CXXFLAGS_*.
    LDCXXLIBS_opt
    LDCXXLIBS_all
  - AR                    : archiver command to create a static library for
                            BLAS/LAPACK.
  - NATIVE_WINDOWS        : flags the intention to compile Native Windows
                            executables under Cygwin (and hence the use of the
                            CASINO/build-tools/winwrap script which converts
                            Linux filenames to Unix ones and eliminates
                            symbolic links. [equivalent to the old 'IS_CYGWIN']

II.4. Example
=============
The following exemplifies the use of command substitution of internal
variables and of user variables:

  #-! *INTERNAL.NICE_BE_NICE:
  #-!  (($(nice)<=15)) && echo 15 || nice
  #-! *INTERNAL.NICE_BE_UNNICE:
  #-!  nice
  #-! *INTERNAL.CPUFREQ_IGNORES_NICE:
  #-!  [ "$(head -n 1\
  #-!   /sys/devices/system/cpu/cpu0/cpufreq/ondemand/ignore_nice_load\
  #-!   2> /dev/null)" = 0 ] && echo no || echo yes
  #-! USER.DESCRIPTION.NICE: Nice-value for the job on workstations.
  #-! *USER.DEFAULT.NICE:
  #-!  case "&INTERNAL.CPUFREQ_IGNORES_NICE&" in
  #-!  no) echo "&INTERNAL.NICE_BE_NICE&" ;;
  #-!  *) echo "&INTERNAL.NICE_BE_UNNICE&" ;;
  #-!  esac
  #-! *USER.MIN.NICE:
  #-!  nice
  #-! USER.MAX.NICE: 19
  #-! RUN_SINGLE: nice -n &USER.NICE& &BINARY&
  #-! RUN_PARALLEL: nice -n &USER.NICE& mpirun -np &NPROC& &BINARY&


III. Concepts
=============
It is important to define the following concepts which are implicit in the
descriptions above.

- CORE: each processing unit in a machine is a CORE.  Single-processor
  workstations only have one core.

- NODE: each computational unit which runs its own instance of the operating
  system in the machine is a node.  Nodes may have one or multiple cores.
  Workstations may have multiple *cores*, but by definition only have one node.

  It is assumed that a node has an amount of memory that is available to all
  of its cores, and that shared-memory mode and OpenMP will always run within
  a node (i.e., memory is shared among cores in a node, and OpenMP threads
  for a given process will spawn within the node that process runs on).
```

F Appendix 6: Switching between double- and single-precision arithmetic in CASINO

By default, CASINO uses double-precision arithmetic everywhere. Almost the only exception that users are likely to encounter is the optional use of single-precision arrays for blip coefficients when the **sp_blips** keyword is set to T.

However, if you wish, it is possible to use single-precision arithmetic throughout.

If you would like to try doing this, sit in the CASINO distribution and run the `CASINO/build-tools/switch_casino_precision` script. You will then need to completely recompile CASINO.

There are a number of known issues with running CASINO using single-precision arithmetic:

- Single-precision BLAS and LAPACK routines are not currently included in the CASINO distribution, so you must use an external BLAS/LAPACK library. E.g., use `CASINO_ARCH=linuxpc-gcc-parallel.openblas` rather than `CASINO_ARCH=linuxpc-gcc-parallel`.
- The main issue: throughout the CASINO source there are many places in which tolerance parameters for various numerical tasks are given fixed values; these tolerances are generally appropriate for double-precision arithmetic, not single-precision arithmetic. Known cases in which `autotest` fails as a result:
  - Numerical Hankel transforms for multilayer Keldysh interactions fail.
- The 'general-purpose cusp correction' scheme fails.

Use of single-precision arithmetic in CASINO is still highly experimental. If you are running CASINO using single-precision arithmetic then please check everything very carefully! Preliminary tests indicate that single-precision arithmetic results in speedups of 11% and 50% for a 186-electron HEG and a 922-electron HEG, respectively, with Slater-Jastrow wave functions and `CASINO_ARCH=linuxpc-gcc-parallel.openblas`.

References

[1] V.R. Saunders, R. Dovesi, C. Roetti, M. Causà, N.M. Harrison, R. Orlando and C.M. Zicovich-Wilson, CRYSTAL98 User's Manual (University of Torino, Torino, 1998).

[2] V.R. Saunders, R. Dovesi, C. Roetti, R. Orlando, C.M. Zicovich-Wilson, N.M. Harrison, K. Doll, B. Civalleri, I. Bush, P. D'Arco and M. Llunell, CRYSTAL2003 User's Manual (University of Torino, Torino, 2003).

[3] GAUSSIAN 98 (Revision A.7), M.J. Frisch, G.W. Trucks, H.B. Schlegel, G.E. Scuseria, M.A. Robb, J.R. Cheeseman, V.G. Zakrzewski, J.A. Montgomery, R.E. Stratmann, J.C. Burant, S. Dapprich, J.M. Millam, A.D. Daniels, K.N. Kudin, M.C. Strain, O. Farkas, J. Tomasi, V. Barone, M. Cossi, R. Cammi, B. Mennucci, C. Pomelli, C. Adamo, S. Clifford, J. Ochterski, G.A. Petersson, P.Y. Ayala, Q. Cui, K. Morokuma, D.K. Malick, A.D. Rabuck, K. Raghavachari, J.B. Foresman, J. Cioslowski, J.V. Ortiz, B.B. Stefanov, G. Liu, A. Liashenko, P. Piskorz, I. Komaromi, R. Gomperts, R.L. Martin, D.J. Fox, T. Keith, M.A. Al-Laham, C.Y. Peng, A. Nanayakkara, C. Gonzalez, M. Challacombe, P.M.W. Gill, B.G. Johnson, W. Chen, M.W. Wong, J.L. Andres, M. Head-Gordon, E.S. Replogle and J.A. Pople, Gaussian, Inc., Pittsburgh PA, 1998.

[4] GAUSSIAN 03, Revision B.03, M.J. Frisch, G.W. Trucks, H.B. Schlegel, G.E. Scuseria, M.A. Robb, J.R. Cheeseman, J.A. Montgomery, Jr., T. Vreven, K.N. Kudin, J.C. Burant, J.M. Millam, S.S. Iyengar, J. Tomasi, V. Barone, B. Mennucci, M. Cossi, G. Scalmani, N. Rega, G.A. Petersson, H. Nakatsuji, M. Hada, M. Ehara, K. Toyota, R. Fukuda, J. Hasegawa, M. Ishida, T. Nakajima, Y. Honda, O. Kitao, H. Nakai, M. Klene, X. Li, J.E. Knox, H.P. Hratchian, J.B. Cross, C. Adamo, J. Jaramillo, R. Gomperts, R.E. Stratmann, O. Yazyev, A.J. Austin, R. Cammi, C. Pomelli, J.W. Ochterski, P.Y. Ayala, K. Morokuma, G.A. Voth, P. Salvador, J.J. Dannenberg, V.G. Zakrzewski, S. Dapprich, A.D. Daniels, M.C. Strain, O. Farkas, D.K. Malick, A.D. Rabuck, K. Raghavachari, J.B. Foresman, J.V. Ortiz, Q. Cui, A.G. Baboul, S. Clifford, J. Cioslowski, B.B. Stefanov, G. Liu, A. Liashenko, P. Piskorz, I. Komaromi, R.L. Martin, D.J. Fox, T. Keith, M.A. Al-Laham, C.Y. Peng, A. Nanayakkara, M. Challacombe, P.M.W. Gill, B. Johnson, W. Chen, M.W. Wong, C. Gonzalez and J.A. Pople, Gaussian, Inc., Pittsburgh PA, 2003.

[5] P. Giannozzi *et al.*, J. Phys. Cond. Mat. **21**, 395502 (2009); `https://www.quantum-espresso.org`

[6] *First-principles computation of material properties: the* ABINIT *software project*, X. Gonze, J.-M. Beuken, R. Caracas, F. Detraux, M. Fuchs, G.-M. Rignanese, L. Sindic, M. Verstraete, G. Zerah, F. Jollet, M. Torrent, A. Roy, M. Mikami, P. Ghosez, J.-Y. Raty and D.C. Allan, Comp. Mat. Science **25**, 478 (2002).

[7] M.D. Segall, P.L.D. Lindan, M.J. Probert, C.J. Pickard, P.J. Hasnip, S.J. Clark and M.C. Payne, J. Phys.: Cond. Matt. **14** 2717 (2002).

[8] The Amsterdam Density Functional package (`https://www.scm.com`).

[9] J.C. Grossman and L. Mitas, Phys. Rev. Lett. **94**, 056403 (2005).

[10] B.L. Hammond, W.A. Lester, Jr. and P.J. Reynolds, *Monte Carlo methods in ab initio quantum Chemistry*, (World Scientific, Singapore, 1994).

[11] W.M.C. Foulkes, L. Mitas, R.J. Needs and G. Rajagopal, Rev. Mod. Phys. **73**, 33 (2001).

[12] R.J. Needs, M.D. Towler, N.D. Drummond and P. López Ríos, J. Phys.: Condens. Matter **22**, 023201 (2010).

[13] R.J. Needs, M.D. Towler, N.D. Drummond, P. López Ríos and J.R. Trail, J. Chem. Phys. **152**, 154106 (2020).

[14] *Quantum Monte Carlo, or, how to solve the many-particle Schrödinger equation accurately whilst retaining favourable scaling with system size*, M.D. Towler, in *Computational Methods for Large Systems* (Wiley, 2011). Also available on the CASINO website.

[15] N.D. Drummond, R.J. Needs, A. Sorouri and W.M.C. Foulkes, Phys. Rev. B **78**, 125106 (2008).

[16] S. Fahy, X.W. Wang and S.G. Louie, Phys. Rev. B **42**, 3503 (1990).

[17] C.J. Umrigar, M.P. Nightingale and K.J. Runge, J. Chem. Phys. **99**, 2865 (1993).

[18] A. Jones, A. Thompson, J. Crain, M.H. Müser and G.J. Martyna, Phys. Rev. B **79**, 144119 (2009).

[19] S. Chiesa, D.M. Ceperley, R.M. Martin and M. Holzmann, Phys. Rev. Lett. **97**, 076404 (2006).

[20] J. H. Lloyd-Williams, R. J. Needs, and G. J. Conduit, Phys. Rev. B **92**, 075106 (2015).

[21] M.F. Depasquale, S.M. Rothstein and J. Vrbik, J. Chem. Phys. **89**, 3629 (1988).

[22] P. Langfelder, S.M. Rothstein, and J. Vrbik, J. Chem. Phys. **107**, 8525 (1997).

[23] A. Zen, S. Sorella, M.J. Gillan, A. Michaelides and D. Alfè, Phys. Rev. B **93**, 241118(R) (2016).

[24] A. Zen, J.G. Brandenburg, A. Michaelides, and D. Alfè, J. Chem. Phys. **151**, 134105 (2019).

[25] M. Casula, Phys. Rev. B **74**, 161102 (2006).

[26] M. Casula, S. Moroni, S. Sorella and C. Filippi, J. Chem. Phys. **132**, 154113 (2010).

[27] M.Y.J. Tan, N.D. Drummond and R.J. Needs, Phys. Rev. B **71**, 033303 (2005).

[28] R.M. Lee, N.D. Drummond and R.J. Needs, Phys. Rev. B **79**, 125308 (2009).

[29] P. López Ríos, P. Seth, N.D. Drummond and R.J. Needs, Phys. Rev. E **86**, 036703 (2012).

[30] T.M. Whitehead, M.H. Michael, and G.J. Conduit, Phys. Rev. B **94**, 035157 (2016).

[31] G.L. Weerasinghe, P. López Ríos, and R.J. Needs, Phys. Rev. E **89**, 023304 (2014).

[32] J.B. Foresman and M.J. Frisch, *Exploring Chemistry with Electronic Structure Methods*, Gaussian, Inc., Pittsburgh, PA, 2nd edition (1996).

[33] D. Alfè and M.J. Gillan, Phys. Rev. B **70**, 161101 (2004).

[34] R. M. Lee, G. J. Conduit, N. Nemec, P. López Ríos and N. D. Drummond, Phys. Rev. E **83**, 066706 (2011).

[35] N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.M. Teller and E. Teller, J. Chem. Phys. **21**, 1087 (1953).

[36] M. Dewing, J. Chem. Phys. **113**, 5123 (2000).

[37] C.J. Umrigar and C. Filippi, unpublished.

[38] D.M. Ceperley, M.H. Kalos and J.L. Lebowitz, Macromolecules **14**, 1472 (1981).

[39] P.J. Reynolds, D.M. Ceperley, B.J. Alder and W.A. Lester, Jr., J. Chem. Phys. **77**, 5593 (1982).

[40] C.J. Everett and E.D. Cashwell, *A Third Monte Carlo Sampler*, Los Alamos Technical Report No. LA-9721-MS (1983).

[41] L. Mitas, E.L. Shirley and D.M. Ceperley, J. Chem. Phys. **95**, 3467 (1991).

[42] P.R.C. Kent, R.Q. Hood, A.J. Williamson, R.J. Needs, W.M.C. Foulkes and G. Rajagopal, Phys. Rev. B **59**, 1917 (1999).

[43] A. Ma, M.D. Towler, N.D. Drummond and R.J. Needs, J. Chem. Phys. **122**, 224322 (2005).

[44] T. Kato, Commun. Pure Appl. Math. **10**, 151 (1957); R.T. Pack and W.B. Brown, J. Chem. Phys. **45**, 556 (1966).

[45] E.L. Shirley and R.M. Martin, Phys. Rev. B **47**, 15413 (1993).

[46] W. Müller, J. Flesch and W. Meyer, J. Chem. Phys. **80**, 3297 (1984).

[47] P.P. Ewald, Ann. Phys. **64**, 253 (1921).

[48] M.P. Tosi, in *Solid State Physics*, Vol. 16, edited by H. Ehrenreich and D. Turnbull (Academic, New York, 1964), p. 1.

[49] V.R. Saunders, C. Freyria-Fava, R. Dovesi, L. Salasco and C. Roetti, Mol. Phys. **77**, 629 (1992).

[50] D.E. Parry, Surf. Sci. **49**, 433 (1975); erratum, Surf. Sci. **54**, 195 (1976).

[51] V.R. Saunders, C. Freyria-Fava, R. Dovesi and C. Roetti, Comp. Phys. Commun. **84**, 156 (1994).

[52] L.M. Fraser, W.M.C. Foulkes, G. Rajagopal, R.J. Needs, S.D. Kenny and A.J. Williamson, Phys. Rev. B **53**, 1814 (1996).

[53] A.J. Williamson, G. Rajagopal, R.J. Needs, L.M. Fraser, W.M.C. Foulkes, Y. Wang and M.-Y. Chou, Phys. Rev. B (Rapid Communications) **55**, 4851 (1997).

[54] G.E. Astrakharchik, J. Boronat, J. Casulleras and S. Giorgini, Phys. Rev. Lett. **93**, 200404 (2004).

[55] J. Carlson, S.-Y. Chang V.R. Pandharipande and K.E. Schmidt, Phys. Rev. Lett. **91**, 050401 (2003).

[56] P.O. Bugnion, R.J. Needs and G.J. Conduit, Phys. Rev. A **90**, 033626 (2014).

[57] B. Ganchev, N.D. Drummond, I. Aleiner and V. Fal'ko, Phys. Rev. Lett. **114**, 107401 (2015).

[58] L.V. Keldysh, JETP Lett. **29**, 658 (1979).

[59] E. Alves, G. L. Bendazzoli, S. Evangelisti and J. A. Berger, Phys. Rev. B **103**, 245125 (2021).

[60] N.D. Drummond, M.D. Towler and R.J. Needs, Phys. Rev. B **70**, 235119 (2004).

[61] Y. Kwon, D.M. Ceperley and R.M. Martin, Phys. Rev. B **48**, 12037 (1993).

[62] M. Holzmann, D.M. Ceperley, C. Pierleoni and K. Esler, Phys. Rev. E **68**, 046707 (2003).

[63] H. Flyvbjerg and H.G. Petersen, J. Chem. Phys. **91**, 461 (1989).

[64] U. Wolff, Comput. Phys. Commun. **156**, 143 (2004).

[65] W. Janke, *Statistical Analysis of Simulations: Data Correlations and Error Estimation*, published in J. Grotendorst, D. Marx, A. Muramatsu (Eds.), *Quantum Simulations of Many-Body Systems: From Theory to Algorithms*, John von Neumann Institute for Computing, Jülich, NIC Series **10**, 423 (2002).

[66] P.R.C. Kent, R.J. Needs and G. Rajagopal, Phys. Rev. B **59**, 12344 (1999).

[67] J.E. Dennis, D.M. Gay and R.E. Welsch, *Algorithm 573: NL2SOL—An Adaptive Nonlinear Least-Squares Algorithm*, ACM Transactions on Mathematical Software **7**, 369 (1981).

[68] N.D. Drummond and R.J. Needs, Phys. Rev. B **72**, 085124 (2005).

[69] C. Filippi and C.J. Umrigar, J. Chem. Phys. **105**, 213 (1996).

[70] M. Snajdr and S.M. Rothstein, J. Chem. Phys. **112**, 4935 (2000).

[71] F.J. Gálvez, E. Buendía and A. Sarsa, J. Chem. Phys. **115**, 1166 (2001).

[72] A. Badinski and R.J. Needs, Phys. Rev. E **76**, 036707 (2007).

[73] D.M. Ceperley, J. Stat. Phys. **43**, 815 (1986).

[74] A. Ma, N.D. Drummond, M.D. Towler and R.J. Needs, Phys. Rev. E **71**, 066704 (2005).

[75] C.J. Umrigar, J. Toulouse, C. Filippi, S. Sorella and R.G. Hennig, Phys. Rev. Lett. **98**, 110201 (2007).

[76] J. Toulouse and C.J. Umrigar, J. Chem. Phys. **126**, 084102 (2007).

[77] M.D. Brown, PhD Thesis, University of Cambridge, Cambridge (2007).

[78] M.P. Nightingale and V. Melik-Alaverdian, Phys. Rev. Lett. **87**, 043401 (2001).

[79] W.H. Press, S.A. Teukolsky, W.T. Vetterling and B.P. Flannery, *Numerical Recipes in Fortran 77, Second Edition*, (Cambridge University Press, 1992).

[80] V.R. Saunders and I.H. Hillier, Int. J. Quant. Chem. **7**, 699 (1973).

[81] M.F. Guest and V.R. Saunders, Mol. Phys. **28**, 819 (1974).

[82] J.R. Trail, Phys. Rev. E **77**, 016703 (2008); Phys. Rev. E **77**, 016704 (2008).

[83] A. Badinski, P.D. Haynes, J.R. Trail and R.J. Needs, J. Phys.: Condens. Matter **22**, 074202 (2010).

[84] J.R. Trail and R. Maezono, J. Chem. Phys. **133** 174120 (2010).

[85] D. Alfè and M.J. Gillan, J. Phys.: Cond. Matt. **16**, L305 (2004).

[86] F.A. Reboredo and A.J. Williamson, Phys. Rev. B **71**, 121105 (2005).

[87] A.J. Williamson, R.Q. Hood and J.C. Grossman, Phys. Rev. Lett. **87**, 246406 (2001).

[88] N.D. Drummond, PhD Thesis, University of Cambridge, Cambridge (2004).

[89] G. Rajagopal, R.J. Needs, S. Kenny, W.M.C. Foulkes and A. James, Phys. Rev. Lett. **73**, 1959 (1994); G. Rajagopal, R.J. Needs, A. James, S. Kenny and W.M.C. Foulkes, Phys. Rev. B **51**, 10591 (1995).

[90] C. Lin, F.H. Zong and D.M. Ceperley, Phys. Rev. E **64**, 016702 (2001).

[91] A. Baldereschi, Phys. Rev. B **7**, 5212 (1973).

[92] M.D. Jones, G. Ortiz and D.M. Ceperley, Int. J. Quant. Chem. **64**, 523 (1997); G. Ortiz, D.M. Ceperley and R.M. Martin, Phys. Rev. Lett. **71**, 2777 (1993); G. Ortiz and D.M. Ceperley, Phys. Rev. Lett. **75**, 4642 (1995).

[93] G.G. Spink, N.D. Drummond and R.J. Needs, Phys. Rev. B **88**, 085121 (2013).

[94] R. Maezono, N.D. Drummond, A. Ma and R.J. Needs, Phys. Rev. B **82**, 184108 (2010).

[95] M.S. Becker, A.A. Broyles and T. Dunn, Phys. Rev. **175**, 224 (1968).

[96] S.D. Kenny, G. Rajagopal and R.J. Needs, Phys. Rev. A **51**, 1898 (1995).

[97] S.D. Kenny, G. Rajagopal, R.J. Needs, W.-K. Leung, M.J. Godfrey, A.J. Williamson and W.M.C. Foulkes, Phys. Rev. Lett. **77**, 1099 (1996).

[98] R. Maezono, M.D. Towler, Y. Lee and R.J. Needs, Phys. Rev. B **68**, 165103 (2003).

[99] G. Ortiz and P. Ballone, Phys. Rev. B **50**, 1391 (1994).

[100] L.M. Fraser, PhD thesis, Imperial College, London (1995).

[101] C.N. Yang, Rev. Mod. Phys. **34**, 694 (1962).

[102] S. De Palo, F. Rapisarda and G. Senatore, Phys. Rev. Lett. **88**, 206401 (2002).

[103] G.E. Astrakharchik, J. Boronat, J. Casulleras and S. Giorgini, Phys. Rev. Lett. **95**, 230405 (2005).

[104] W. Kohn, Phys. Rev. A **133**, 171 (1964).

[105] R.D. King-Smith and D. Vanderbilt, Phys. Rev. B **47**, R1651 (1993).

[106] D. Vanderbilt and R.D. King-Smith, Phys. Rev. B **48**, 4442 (1993).

[107] R. Resta and S. Sorella, Phys. Rev. Lett. **82**, 370 (1999).

[108] I. Souza, T. Wilkens and R. Martin, Phys. Rev. B **62**, 1666 (2000).

[109] M. Veithen, X. Gonze and Ph. Ghosez, Phys. Rev. B **66**, 235113 (2002).

[110] R. Resta, Phy. Rev. Lett. **95**, 196805 (2005).

[111] N.D.M. Hine and W.M.C. Foulkes, J. Phys. Condens. Matter **19**, 506212 (2007).

[112] H. Hellmann, *Einführung in die Quantenchemie*, 285 (1937).

[113] R.P. Feynman, Phys. Rev. **56**, 340 (1939).

[114] P. Pulay, Mol. Phys. **17**, 197 (1969).

[115] M.M. Hurley and P.A. Christiansen, J. Chem. Phys. **86**, 1069 (1987).

[116] A. Badinski and R.J. Needs, Phys. Rev. E **76**, 36707 (2007).

[117] A. Badinski, P.D. Haynes and R.J. Needs Phys. Rev. B **77**, 85111 (2008).

[118] P.J. Reynolds, R.N. Barnett, B.L. Hammond and W.A. Lester, Jr., J. Stat. Phys. **43**, 1017 (1986).

[119] P.J. Reynolds, R.N. Barnett, B.L. Hammond, R.M. Grimes and W.A. Lester, Jr., Int. J. Quant. Chem. **29**, 589 (1986).

[120] F. Schautz and H.-J. Flad, J. Chem. Phys. **110**, 11700 (1999).

[121] R. Assaraf and M. Caffarel, Phys. Rev. Lett. **83**, 4682 (1999).

[122] R. Assaraf and M. Caffarel, J. Chem. Phys. **113**, 4028 (2000).

[123] S. Chiesa, D.M. Ceperley and S. Zhang, Phys. Rev. Lett. **94**, 36404 (2005).

[124] A. Badinski, PhD thesis, University of Cambridge (2008).

[125] A. Badinski and R.J. Needs, unpublished (2008).

[126] A. Badinski and R.J. Needs, unpublished (2008).

[127] R.N. Barnett, P.J. Reynolds and W.A. Lester, Jr., J. Comp. Phys. **96**, 258 (1991).

[128] J. Casulleras and J. Boronat, Phys. Rev. B **52**, 3654 (1995).

[129] *Petascale computing opens up new vistas for quantum Monte Carlo*, M.J. Gillan, M.D. Towler and D. Alfè, Psi-k Scientific Highlight of the Month (Feb 2011).
Available at `https://psi-k.net/download/highlights/Highlight_103.pdf`.

[130] `https://www.darpa.mil/ipto/personnel/docs/ExaScale_Study_Initial.pdf` (2008).