Forces and Correlated Sampling in DMC

T.B. Poole[1] W.M.C. Foulkes[1]
J.S. Spencer[1,2] P.D. Haynes[1,2]

[1]Department of Physics
Imperial College London

[2]Department of Materials
Imperial College London

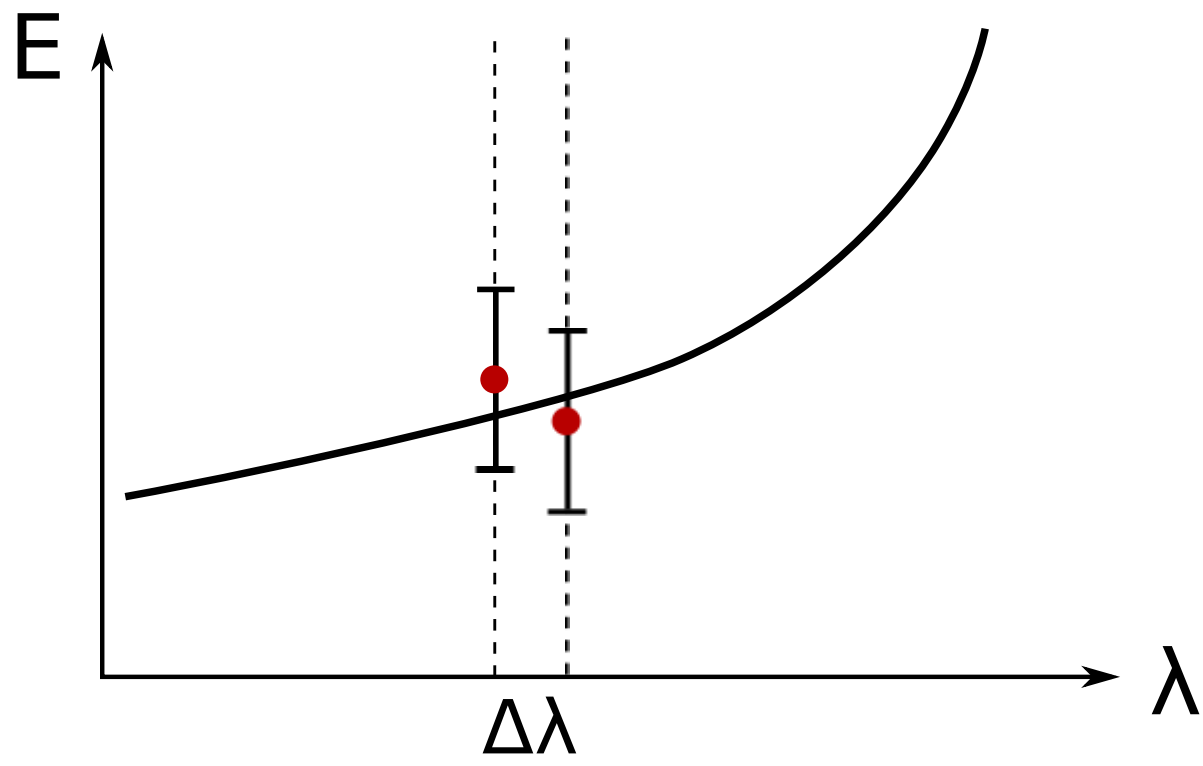
Quantum Monte Carlo in the Apuan Alps VIII
28th July 2013

Outline

Outline

What is the problem?

Why are derivatives so difficult to obtain using Monte Carlo methods?



Relative error in finite difference estimate of $dE/d\lambda$ diverges as $\Delta\lambda \to 0$.

Why do derivatives matter?

- Forces are derivatives, as are other linear response parameters.
- Quick and accurate DMC derivatives of the total energy with respect to trial function parameters would enable DMC wavefunction optimization.

Outline

What are the solutions?

Three main approaches to this problem are discussed in the literature:

1. Derive and evaluate an analytic expression for the force.
2. Correlated sampling:
   - By using the same random numbers in the runs at λ and $\lambda + \Delta\lambda$, ensure that the statistical errors in the energies at λ and $\lambda + \Delta\lambda$ are as similar as possible.
   - Statistical correlations are usually a problem; here they are an advantage.
3. Differentiate your QMC code line by line to obtain an exact numerical algorithm to evaluate the derivative of its output.

Analytic derivatives

The Hellmann-Feynman theorem

$$
\begin{aligned}
\frac{d\langle\Psi|\hat{H}|\Psi\rangle}{d\lambda} &= \left\langle \Psi \middle| \frac{d\hat{H}}{d\lambda} \middle| \Psi \right\rangle + \left\langle \frac{d\Psi}{d\lambda} \middle| \hat{H} \middle| \Psi \right\rangle + \left\langle \Psi \middle| \hat{H} \middle| \frac{d\Psi}{d\lambda} \right\rangle \\
&= \left\langle \Psi \middle| \frac{d\hat{H}}{d\lambda} \middle| \Psi \right\rangle + E\left\langle \frac{d\Psi}{d\lambda} \middle| \Psi \right\rangle + E\left\langle \Psi \middle| \frac{d\Psi}{d\lambda} \right\rangle \\
&= \left\langle \Psi \middle| \frac{d\hat{H}}{d\lambda} \middle| \Psi \right\rangle + E\frac{d\langle\Psi|\Psi\rangle}{d\lambda} \\
&= \left\langle \Psi \middle| \frac{d\hat{H}}{d\lambda} \middle| \Psi \right\rangle
\end{aligned}
$$

Analytic derivatives

The Hellmann-Feynman theorem

$$
\begin{aligned}
\frac{d\langle\Psi|\hat{H}|\Psi\rangle}{d\lambda} &= \left\langle\Psi\left|\frac{d\hat{H}}{d\lambda}\right|\Psi\right\rangle + \left\langle\frac{d\Psi}{d\lambda}\left|\hat{H}\right|\Psi\right\rangle + \left\langle\Psi\left|\hat{H}\right|\frac{d\Psi}{d\lambda}\right\rangle \\
&= \left\langle\Psi\left|\frac{d\hat{H}}{d\lambda}\right|\Psi\right\rangle + E\left\langle\frac{d\Psi}{d\lambda}\middle|\Psi\right\rangle + E\left\langle\Psi\middle|\frac{d\Psi}{d\lambda}\right\rangle \\
&= \left\langle\Psi\left|\frac{d\hat{H}}{d\lambda}\right|\Psi\right\rangle + E\frac{d\langle\Psi|\Psi\rangle}{d\lambda} \\
&= \left\langle\Psi\left|\frac{d\hat{H}}{d\lambda}\right|\Psi\right\rangle
\end{aligned}
$$

The red line only follows if $\hat{H}\Psi = E\Psi$. Otherwise you have to keep the Pulay terms, which depend on wavefunction derivatives.

More precisely . . .

- The HFT holds in VMC if:
  - the wave function has been energy optimized; and
  - the Hilbert space spanned by the basis set is independent of λ.

- The HFT does not hold in fixed-node DMC unless the nodal surface is independent of λ.

Problems of the analytic approach

- The naive HFT force estimator has infinite variance.
  (Assaraf-Caffarel improved estimators and Filippi-Umrigar space-warp transformations help.)
- The Pulay terms are hard to evaluate in VMC.
- They are even harder to evaluate in DMC, since they depend on derivatives of the fixed-node ground state, which is sampled but not known explicitly.

Outline

Algorithmic differentiation

- AD is well known in some fields and is becoming quite widely used in finance, but has barely penetrated into materials simulation.
- A couple of years ago, Sorella and Capriotti (a former physicist now working for a bank) proposed using AD to evaluate forces and other derivatives in QMC.
- Their paper was fascinating but hard to follow. This sparked us to try to understand it.

Basic idea

Excluding control-flow statements, lines of code in a computer program ($y = 2 * z$; $x = \sin(y + z)$; ...) can all be viewed as little functions, which are fed inputs and produce outputs:

$$
\begin{aligned}
\text{Inputs:} \quad & \boldsymbol{x}^0 \\
\text{Statement 1:} \quad & \boldsymbol{x}^1 = \boldsymbol{f}^1(\boldsymbol{x}^0) \\
\text{Statement 2:} \quad & \boldsymbol{x}^2 = \boldsymbol{f}^2(\boldsymbol{x}^1) \\
& \ldots\ldots\ldots \\
& \ldots\ldots\ldots \\
\text{Outputs:} \quad & \boldsymbol{x}^M = \boldsymbol{f}^M(\boldsymbol{x}^{M-1})
\end{aligned}
$$

(The vectors $\boldsymbol{x}^0$, $\boldsymbol{x}^1$, ..., generally have different numbers of components.)

Outline

Forward AD

$$\begin{aligned} \boldsymbol{x}^m &= \boldsymbol{f}^m(\boldsymbol{x}^{m-1}) \\ x_i^m &= f_i^m(\boldsymbol{x}^{m-1}) \end{aligned}$$

If we vary the program inputs, $x_j^0 \to x_j^0 + dx_j^0$, the variations propagate through the program:

$$\begin{aligned} dx_i^1 &= \frac{\partial f_i^1(\boldsymbol{x}^0)}{\partial x_j^0} dx_j^0 \\ dx_i^2 &= \frac{\partial f_i^2(\boldsymbol{x}^1)}{\partial x_j^1} dx_j^1 \\ &\dots \\ &\dots \\ dx_i^M &= \frac{\partial f_i^M(\boldsymbol{x}^{M-1})}{\partial x_j^{M-1}} dx_j^{M-1} \end{aligned}$$

(summation convention in force)

Matrix notation

$$
\begin{aligned}
d\boldsymbol{x}^1 &= \boldsymbol{J}^1 d\boldsymbol{x}^0 \\
d\boldsymbol{x}^2 &= \boldsymbol{J}^2 d\boldsymbol{x}^1 \\
&\dots \\
d\boldsymbol{x}^M &= \boldsymbol{J}^M d\boldsymbol{x}^{M-1}
\end{aligned}
$$

$$
d\boldsymbol{x}^M = \boldsymbol{J}^M \boldsymbol{J}^{M-1} \dots \boldsymbol{J}^1 d\boldsymbol{x}^0 = \boldsymbol{J} d\boldsymbol{x}^0
$$

Notes

- The matrices $\boldsymbol{J}^m$ need not be square; the number of active variables changes as the program runs.
- $\boldsymbol{J}^m$ depends on the vector $\boldsymbol{x}^{m-1}$, which describes the state of the program (the values of all active variables) before step m begins.

Example

Lines of code	Lines of differentiated code
$\ldots$	$\ldots$
$y = 2 * z$	$dy = 2 * dz$
$x = \sin(y + 3 * z)$	$dx = \cos(y + 3 * z) * dy + 3 * \cos(y + 3 * z) * dz$
$\ldots$	$\ldots$
if $(x > 0)$	$dy = \Theta(x)dx$
$y = x$	
else	
$y = 0$	

Programming forward AD

$$
\begin{aligned}
x_i^m &= f_i^m(\boldsymbol{x}^{m-1}) \\
dx_i^m &= \sum_j \frac{\partial f_i^m(\boldsymbol{x}^{m-1})}{\partial x_j^{m-1}} dx_j^{m-1}
\end{aligned}
$$

- For every line of the original code, add a new line to propagate the differentials.
- A single line of code normally changes only one variable x_i^m, and only a few inputs x_j^{m-1} affect its value. The matrix $\partial f_i^m / \partial x_j^{m-1}$ is mostly the identity.
- The differential lines are not much more complicated than the original lines, so ...

Running a code in forward AD mode is only a few times slower than running it without AD.

Use of forward AD

Forward AD is best when the code has one input but many outputs:

- Choose $d\boldsymbol{x}^0 = (dx_1^0)$, which is a one-component vector.
- Run the code in forward AD mode to obtain the vector

$$d\boldsymbol{x}^M = \boldsymbol{J} d\boldsymbol{x}^0 \qquad dx_i^M = J_{i,1} dx_1^0$$

  where i ranges over all output variables.
- One run of the AD code yields the derivatives of *every* output x_i^M with respect to the single input x_1^0.
- Since dx_i^M is a linear function of dx_1^0, might as well set dx_1^0 to 1. The outputs dx_i^M are then *exactly* equal to $\partial x_i^M / \partial x_1^0$.

AD in QMC

Unfortunately, in a QMC simulation, we have many inputs (atomic positions, wavefunction parameters) but only one output (the energy). Forward AD is very inefficient.

Need reverse or adjoint AD

Outline

Adjoint AD

$$d\boldsymbol{x}^M = \boldsymbol{J}^M \boldsymbol{J}^{M-1} \ldots \boldsymbol{J}^1 d\boldsymbol{x}^0 = \boldsymbol{J} d\boldsymbol{x}^0$$

- For simplicity, assume that the output vector $d\boldsymbol{x}^M = (dx_1^M)$ has only one component (as in QMC).
- Introduce a one-component unit row vector $b\tilde{\boldsymbol{x}}^M$ and dot it with both sides:

$$b\tilde{\boldsymbol{x}}^M d\boldsymbol{x}^M = b\tilde{\boldsymbol{x}}^M \boldsymbol{J}^M \boldsymbol{J}^{M-1} \ldots \boldsymbol{J}^1 d\boldsymbol{x}^0$$

- Because $b\tilde{\boldsymbol{x}}^M$ is a one-component unit vector, the left-hand side is equal to dx_1^M.

$$dx_1^M = b\tilde{\boldsymbol{x}}^M d\boldsymbol{x}^M = b\tilde{\boldsymbol{x}}^M \boldsymbol{J}^M \boldsymbol{J}^{M-1} \ldots \boldsymbol{J}^1 d\boldsymbol{x}^0$$

- We can obtain dx_1^M by stepping backwards through the program, working out the quantities

$$\begin{aligned} b\tilde{\boldsymbol{x}}^{M-1} &= b\tilde{\boldsymbol{x}}^M \boldsymbol{J}^M \\ b\tilde{\boldsymbol{x}}^{M-2} &= b\tilde{\boldsymbol{x}}^{M-1} \boldsymbol{J}^{M-1} \\ &\ldots \\ b\tilde{\boldsymbol{x}}^0 &= b\tilde{\boldsymbol{x}}^1 \boldsymbol{J}^1 \end{aligned}$$

and then evaluating

$$dx_1^M = b\tilde{\boldsymbol{x}}^0 d\boldsymbol{x}^0$$

- Dotting $b\tilde{\boldsymbol{x}}^0$ with $d\boldsymbol{x}^0$ to obtain dx_1^M is unnecessary. Since

$$dx_1^M = b\tilde{\boldsymbol{x}}^0 d\boldsymbol{x}^0$$

the components of $b\tilde{\boldsymbol{x}}^0$ are already the partial derivatives of the single output with respect to the n^0 inputs:

$$\frac{dx_1^M}{dx_i^0} = b\tilde{x}_i^0$$

One adjoint AD run generates the derivatives of the output with respect to all inputs. (All of the forces in one go.)

- Adjoint AD uses the same $\boldsymbol{J}^m$ matrices as forward AD:
  - First run the code forwards, generating and storing the few non-trivial elements of each matrix.
  - Then use this information to step backwards, evaluating $b\tilde{\boldsymbol{x}}^m$ for decreasing values of m.
- The computational effort is not much more than for forward AD, but the memory requirements can be shocking.

Outline

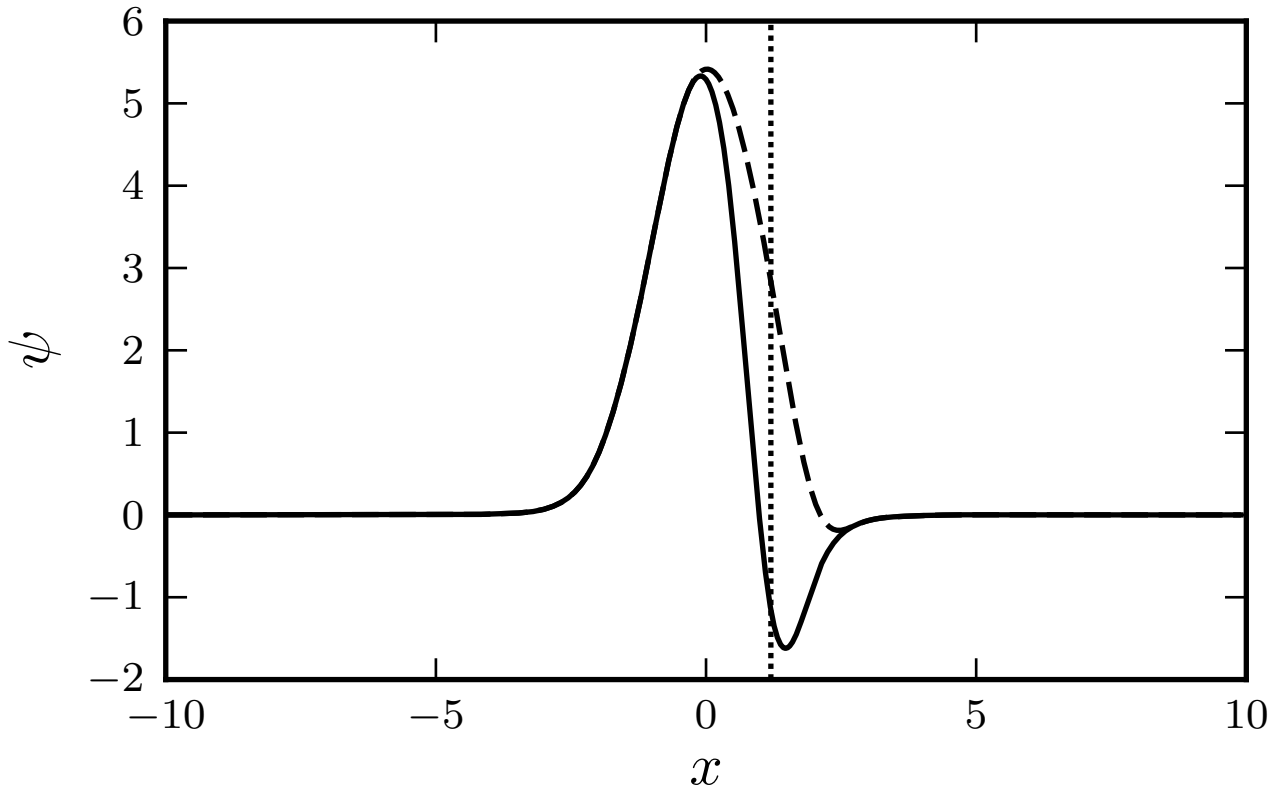
Ambitions

- As far as we are able to tell, Sorella and friends have only used AAD to evaluate derivatives of the trial wavefunction in VMC simulations.
- Tom decided to differentiate the complete DMC algorithm. This was ambitious because the books say that Monte Carlo algorithms in general cannot be differentiated!

System and algorithm

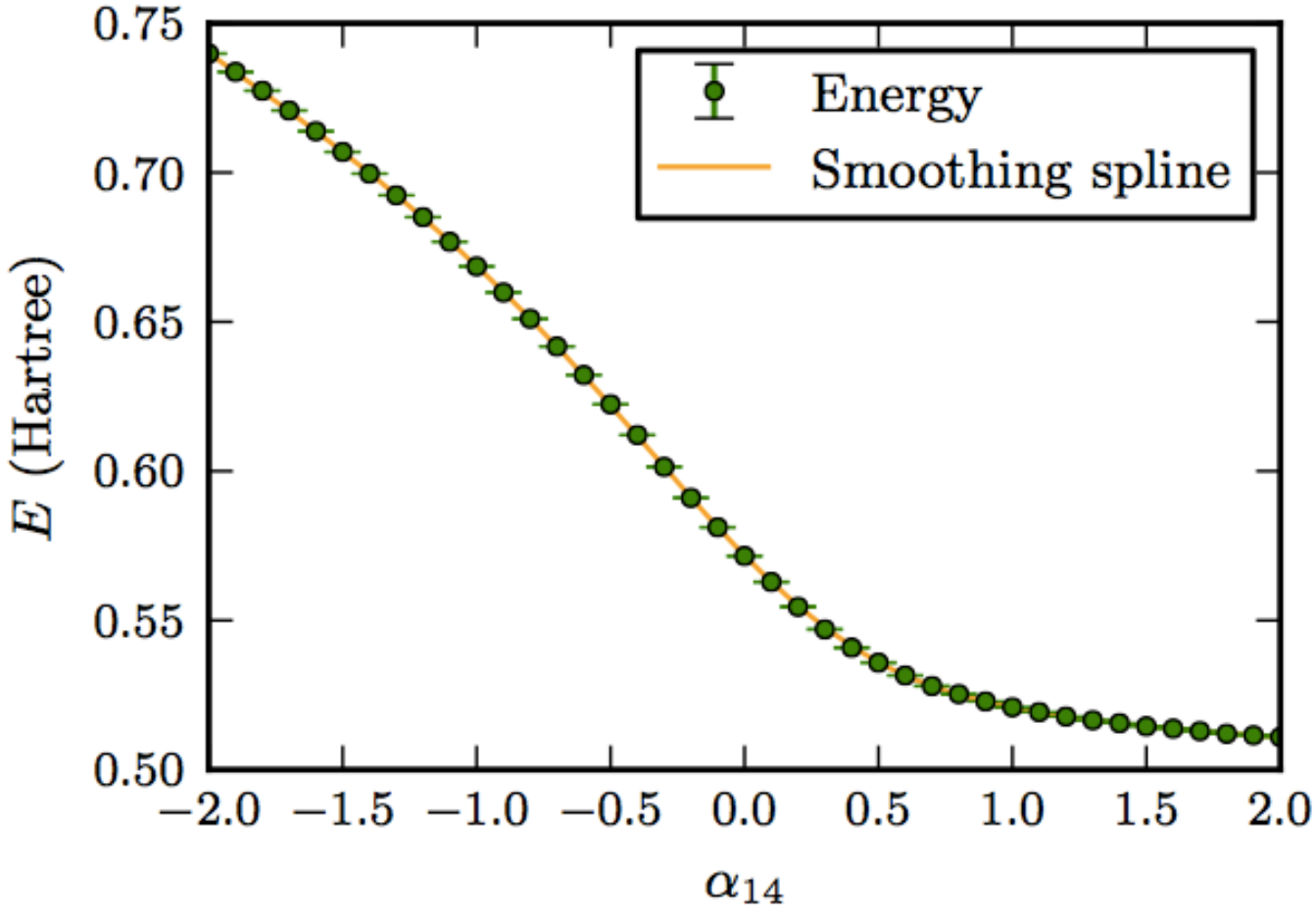
- One electron in a 1D harmonic well.
- Aim is direct DMC trial function optimization.
- Trial function consisting of 25 blips. Starting function had two unnecessary nodes.
- No Metropolis rejection step.

Starting trial function

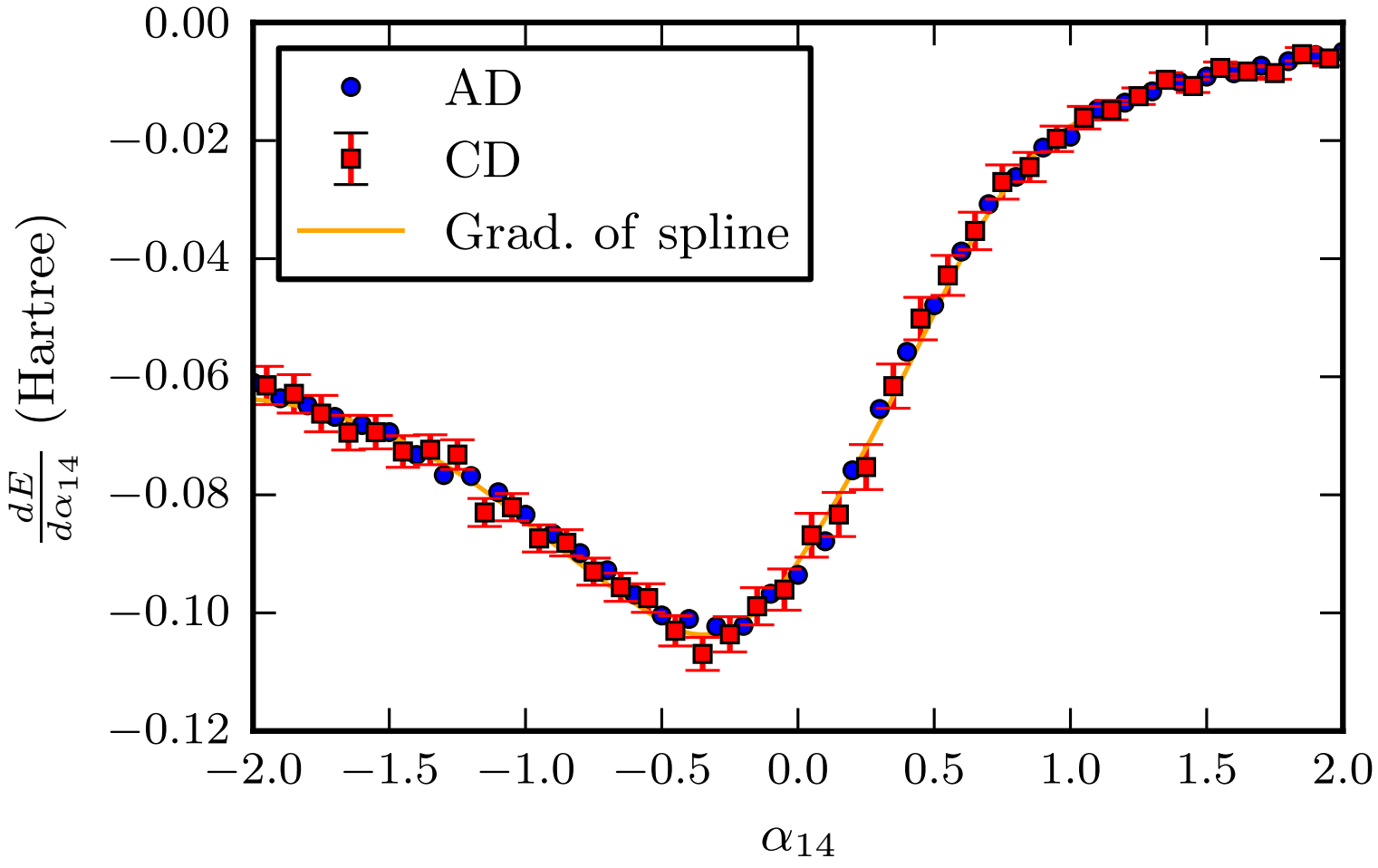


The form of the wave function used for the single electron harmonic well calculations. The dotted line represents the centre of the blip function with coefficient α_{14}, which is varied between -2.0, giving the solid line, and 2.0, giving the dashed line.

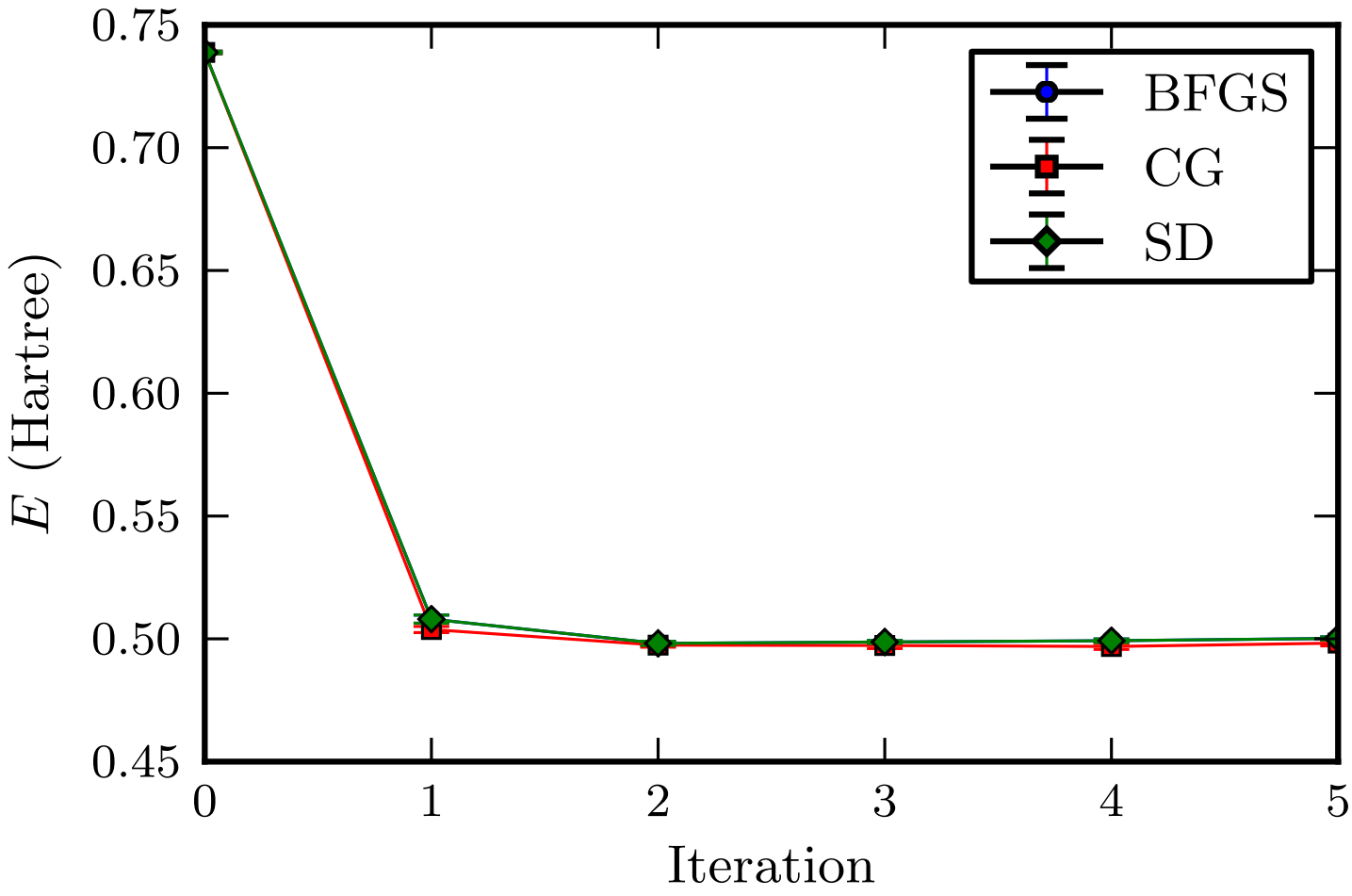
Energy


Energy
Smoothing spline
0.75
0.70
0.65
0.60
0.55
0.50
E (Hartree)
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
α_{14}

Derivative


AD
CD
Grad. of spline
0.00
−0.02
−0.04
−0.06
−0.08
−0.10
−0.12
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
$\frac{dE}{d\alpha_{14}}$ (Hartree)
α_{14}

Optimization

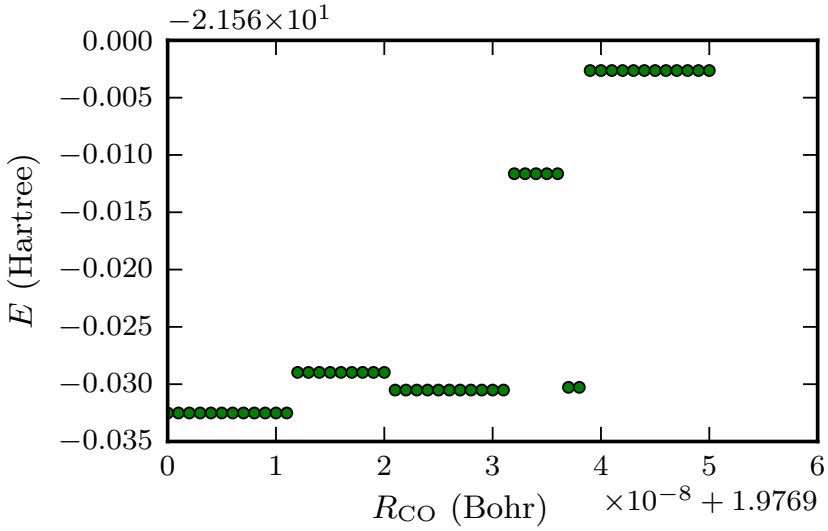


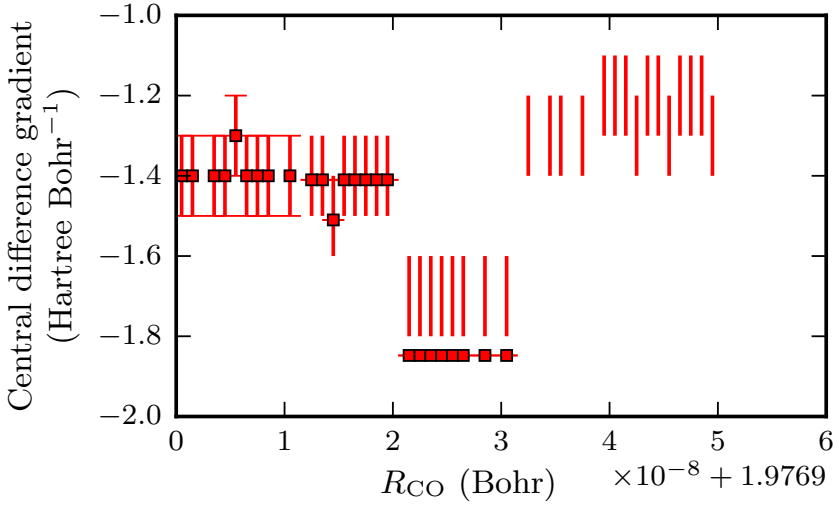
[It is just as easy and not much slower to calculate and optimize many derivatives simultaneously.]

Can the DMC algorithm be differentiated?

- Imagine running two parallel DMC simulations with slightly different Hamiltonians but the same starting point and stream of random numbers.
- As the difference between the two Hamiltonians tends to zero, you have to wait longer and longer for the two simulations to make a different branching decision.
- CASINO sets the walker weights back to unity after each branching decision. Hence, in a short run, it forces the two infinitesimally different simulations to use the same weights.

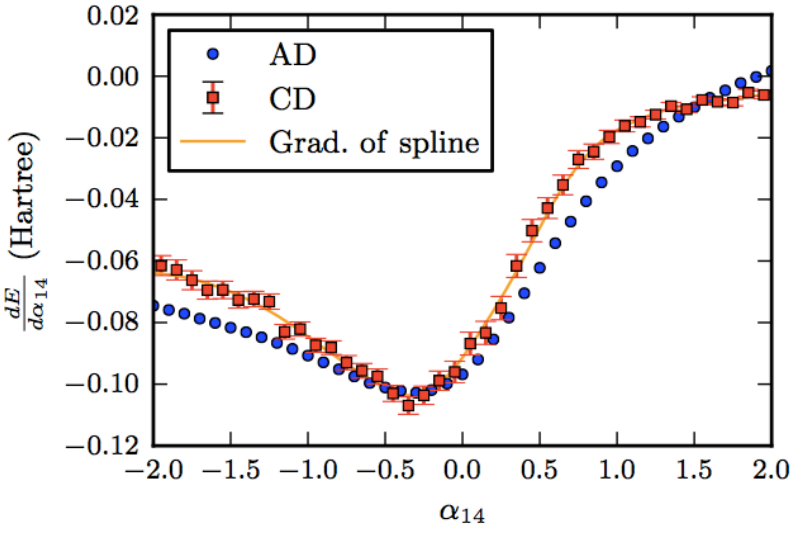
CASINO calculations of CO molecule

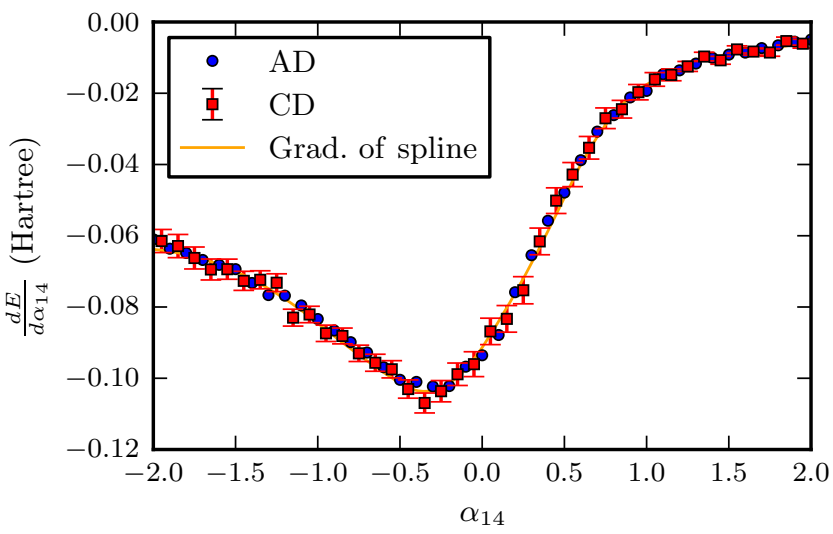




Smooth derivatives in AD simulations

To circumvent this problem in our AD simulations, we had to take great care to maintain the expected value of the weight derivative across branching decisions.





Conclusions

- AAD is elegant and powerful.
- Early results are quite encouraging; it might be possible to make AAD work in a real DMC code.
- The task of differentiating, say, CASINO, is terrifying. The result would be a nightmare to maintain.
- Experts suggest starting again from scratch!

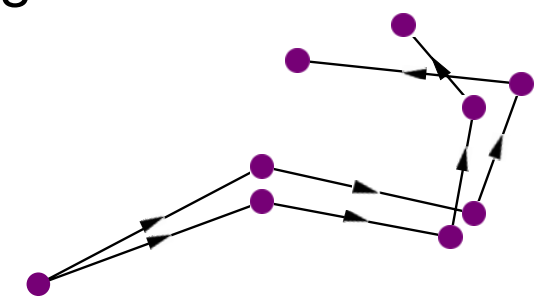
Outline

Pure DMC correlated sampling algorithm

- Consider two pure DMC random walks with slightly different Hamiltonians, $\hat{H}$ $(= \hat{H}^{\lambda=1})$ and $\hat{H}^{\lambda}$.
- The drift-diffusion steps are

$$\boldsymbol{r}_{n+1}^{\lambda} = \boldsymbol{r}_{n}^{\lambda} + \boldsymbol{v}^{\lambda}(\boldsymbol{r}_{n}^{\lambda})\tau + \boldsymbol{\xi}\sqrt{\tau}$$

- Correlate the Gaussian displacements $\boldsymbol{\xi}$. Drift velocities and accumulated weights differ.

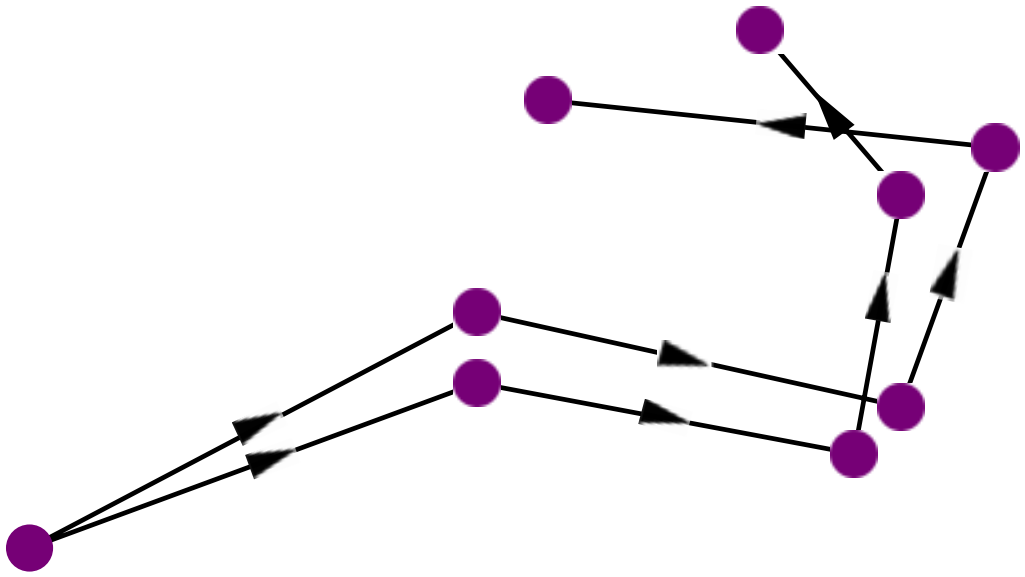


Advantages

- Why do analytic HFT-based approaches suffer from divergences?
- All points sampled are chosen using the unperturbed Hamiltonian. Walkers may therefore visit regions near the perturbed nodal surface or nuclei, where the perturbed local energy and drift velocity diverge.
- Using two random walks solves this problem: each DMC walk only visits regions of configuration space appropriate for its own Hamiltonian.

Chaos

What about chaos? Surely the two walkers will separate exponentially fast and the correlation between them will be lost?



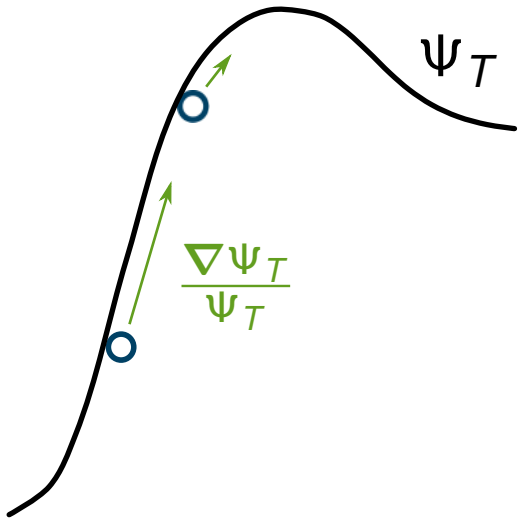
Not necessarily!

Why stochastic resonance?

Fahy and Hamann (1992) showed that in some cases the common Gaussian noise can drive initially distinct trajectories together and keep them close. By coupling DMC configurations by the diffusive components of their dynamics, the drift velocity

$$\boldsymbol{v}(\boldsymbol{r}) = \frac{\boldsymbol{\nabla}\Psi_T}{\Psi_T}$$

coheres configurations occupying concave regions of Ψ_T.



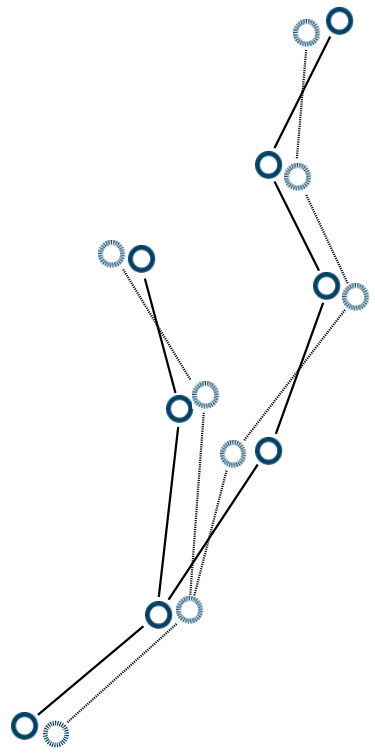
Outline

Branching

> “Everyone knows that pure DMC is next to useless”

Then don’t use it!

- Allow both walks to branch, but correlate the branching decisions.

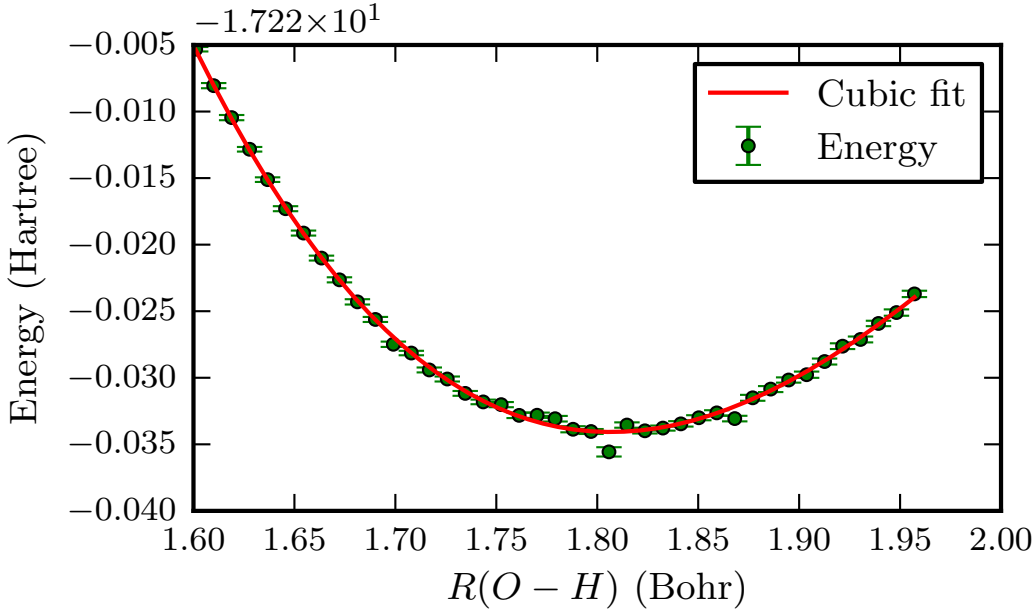


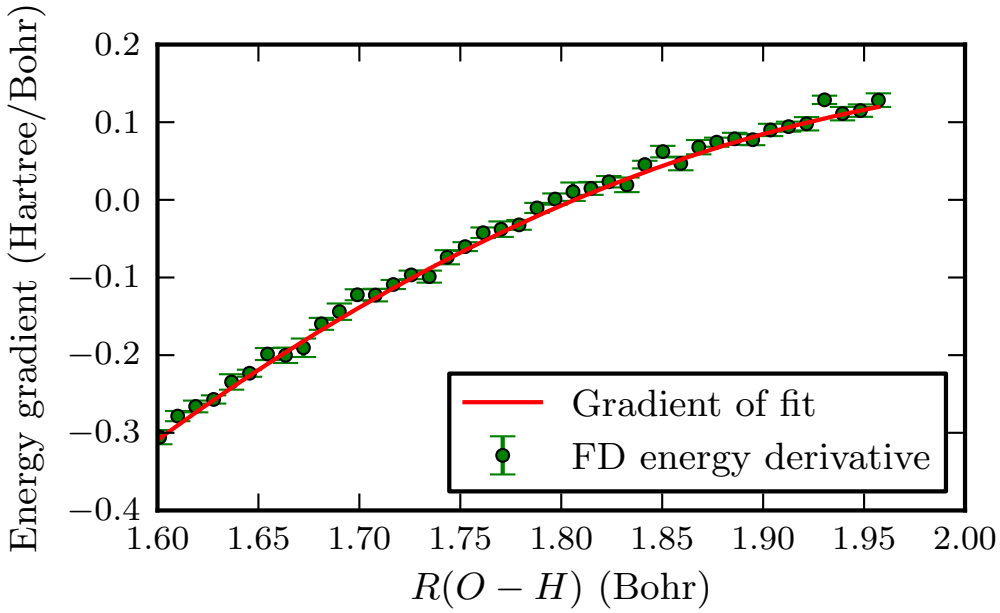
- The perturbed walk branches in exactly the same way as the unperturbed walk.
- The small errors introduced can be dealt with exactly by careful reweighting.
- Rejection step can also be dealt with by reweighting.
- Method has a zero-variance zero-bias principle.

(Filippi and Umrigar considered a similar method but dismissed it because they thought the weights would accumulate horribly. In small systems, at least, this does not appear to be the case — presumably because of stochastic resonance.)

Outline

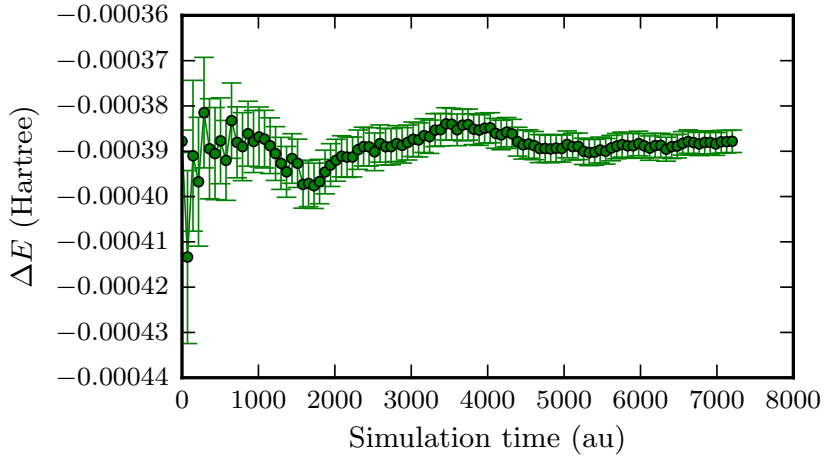
Symmetric stretch of the water molecule

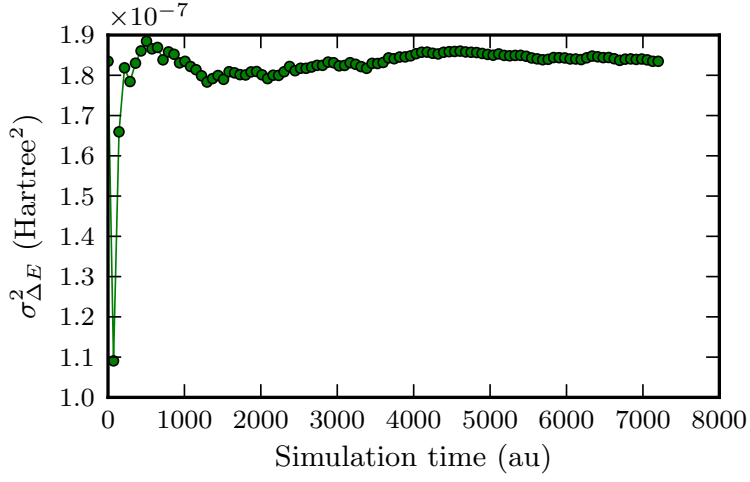




Stability

A common feature of reweighting approaches is an increase in the variance of the estimator with simulation time, arising from the exponentially diverging weights. Not here.



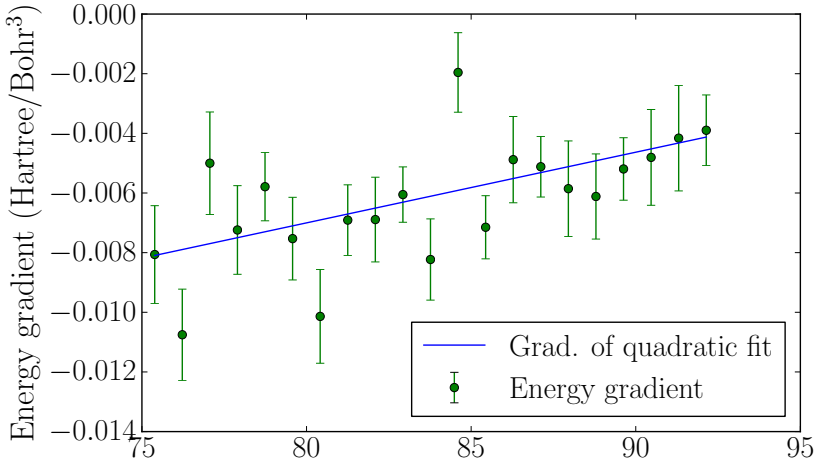


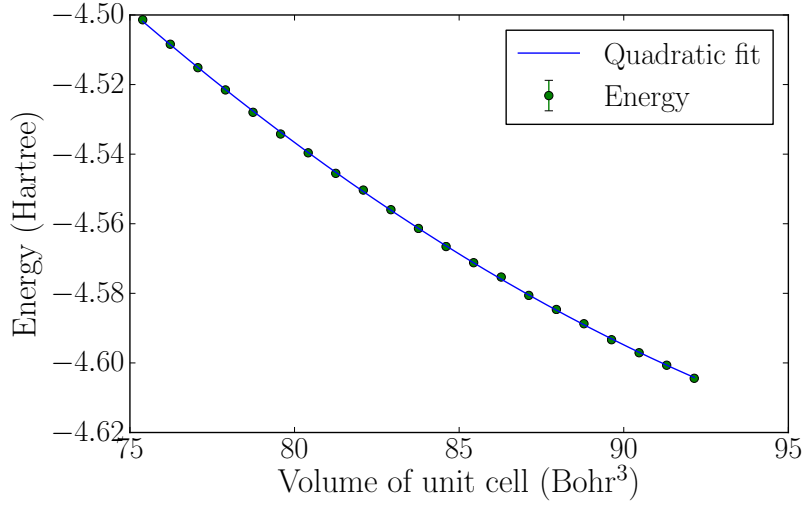
Caveats

- Sadly, stochastic resonance does not always occur. It depends on the system.
- In solid H, for example, we see stochastic resonance at 1 atmosphere but not at the very high pressures found in diamond anvil cells.
- The noise in the calculated finite differences is much worse when there is no resonance.

Pressure of solid H

8 atoms in cell; Cmca structure; high density.





Conclusions

- DMC correlated sampling is by no means as hopeless as has often been assumed.
- Taking the $\Delta\lambda \rightarrow 0$ limit of DMC correlated sampling yields an algorithm to generate exact derivatives, not finite differences.
- This has some similarities to the method introduced last year by Per, Snook and Russo. We expect it to produce results more or less equivalent to those that would be obtained using AD.
- We are still investigating what to do when stochastic resonance does not occur.