Importance Sampling Diffusion Monte Carlo

In the Diffusion Monte Carlo method, the ground state energy and wave function of the system are obtained *without* having to specify a trial wave function! The algorithm found the wave function of the ground state by itself. This is a big advantage of DMC over Variational Monte Carlo where one needs to specify a trial wave function and then vary its parameters.

However, in many cases, the DMC algorithm can be speeded up with the use of a suitable trial wave function which is called a *guide function*. This is important in problems where the potential is singular. Recall that the branching step in the DMC algorithm uses quantity

$$q = e^{-(V(R)-E_{\mathrm{T}})\Delta\tau}$$

to determine whether to kill or clone a walker at position R. If $V(R)$ is large and positive, then all walkers in the region will be killed. If $V(R)$ is very large and negative, then a very large number of clones will be created. Thus singular behavior of V can cause large fluctuations in the number of walkers, which can make the algorithm unstable.

Guide function for DMC

Instead of solving the diffusion equation

$$\frac{\partial\psi(R,\tau)}{\partial\tau} = \frac{1}{2}\nabla_R^2\psi(R,\tau) - V(R)\psi(R,\tau) \ ,$$

where R stands for the coordinates of all the particles in the system, the following function is defined

$$\rho(R,\tau) = \psi(R,\tau)\Psi_{\mathrm{T}}(R) \ ,$$

where $\Psi_{\mathrm{T}}(R)$ is a *guide* or *trial* wave function for the system which is carefully chosen to smooth any singularities in the potential. For an atomic system like Helium, $\Psi_{\mathrm{T}}(\mathbf{r}_1,\mathbf{r}_2)$ can be chosen to be a Padé-Jastrow function

$$\Psi_{\mathrm{T}}(\mathbf{r}_1,\mathbf{r}_2) = e^{-2r_1-2r_2+\frac{r_{12}}{2(1+\alpha r_{12})}} \ ,$$

which was used in the VMC method.

This modified function satisfies a Fokker-Planck type of equation:

$$\frac{\partial \rho(R,\tau)}{\partial \tau} = \frac{1}{2}\nabla_R \left[\nabla_R - \mathbf{F}(R)\right] \rho(R,\tau) - \left[E_{\rm L}(R) - E_{\rm T}\right] \rho(R,\tau) \ ,$$

with a *Fokker-Planck force function*

$$F(R) = \frac{2}{\Psi_{\rm T}(R)} \nabla_R \Psi_{\rm T}(R) \ ,$$

and a *local energy function*

$$E_{\rm L}(R) = -\frac{1}{2\Psi_{\rm T}(R)} \nabla_R^2 \Psi_{\rm T}(R) + V(R) \ .$$

Note that this local energy can be made much smoother than the potential if the guide function is carefully chosen.

For example, with the Padé-Jastrow function in the He problem,

$$\begin{aligned} E_{\rm L}(\mathbf{r}_1,\mathbf{r}_2) = &- 4 + \frac{\alpha}{(1+\alpha r_{12})} + \frac{\alpha}{(1+\alpha r_{12})^2} + \frac{\alpha}{(1+\alpha r_{12})^3} \\ &- \frac{1}{4(1+\alpha r_{12})^4} + \frac{\hat{\mathbf{r}}_{12}\cdot(\hat{\mathbf{r}}_1 - \hat{\mathbf{r}}_2)}{(1+\alpha r_{12})^2} \ . \end{aligned}$$

which is *non-singular* when r_1, r_2, or r_{12} approach zero because the cusp conditions have been satisfied. For this choice of trial function, the Fokker-Planck forces on the electrons can easily be computed:

$$\begin{aligned} \frac{2}{\Psi_{\rm T}(R)} \nabla_{r_1} \Psi_{\rm T}(R) &= -4\hat{\mathbf{r}}_1 + \frac{\hat{\mathbf{r}}_{12}}{(1+\alpha r_{12})^2} \\ \frac{2}{\Psi_{\rm T}(R)} \nabla_{r_2} \Psi_{\rm T}(R) &= -4\hat{\mathbf{r}}_2 + \frac{\hat{\mathbf{r}}_{21}}{(1+\alpha r_{12})^2} \end{aligned}$$

where $\hat{\mathbf{r}}_{ij} = (\mathbf{r}_i - \mathbf{r}_j)/r_{ij}$.

Algorithm for guided DMC

As in the Fokker-Planck VMC calculation, the Fokker-Planck equation in this problem can be solved using an *approximate* Green's function:

$$G(R', R; \Delta\tau) = \frac{1}{\sqrt{2\pi\Delta\tau}} \exp\left[-\frac{\left(R' - R - \frac{1}{2}F(R)\Delta\tau\right)^2}{2\Delta\tau}\right] ,$$

and the $\mathcal{O}(\Delta\tau^2)$ error corrected using a Metropolis procedure with a test ratio

$$w = \frac{G(R, R'; \Delta\tau)\rho(R')}{G(R', R; \Delta\tau)\rho(R)} \simeq \frac{G(R, R'; \Delta\tau)\Psi_{\mathrm{T}}^2(R')}{G(R', R; \Delta\tau)\Psi_{\mathrm{T}}^2(R)} .$$

The strategy for solving the problem combines the Green's function approach to the Fokker-Planck type equation with the branching process used in the simple DMC algorithm:

- **Initialization:** Choose a target number of walkers, and a step size $\Delta\tau$. The walkers are randomly placed in the 6-dimensional configuration space of the two electrons.
- **Monte Carlo Steps:** After some number of thermalization steps, the average energy is measured over time. For each time step:
  - **Iteration:** For each of the N walkers, do the following:
    - **Diffusion step:** A trial move to a new position is generated for the walker

      $$R' = R + \frac{1}{2}\Delta\tau F(R) + \eta\sqrt{\Delta\tau}$$

      where η is chosen randomly from a Gaussian with unit variance.
    - **Metropolis test:** The trial move is accepted if the test ratio w exceeds a uniform random number between 0 and 1. If the Metropolis test succeed, then
      - **Branching process:** Evaluate the branching factor

        $$q = e^{-\Delta\tau\left[\frac{E_{\mathrm{L}}(R') + E_{\mathrm{L}}(R)}{2} - E_{\mathrm{T}}\right]} .$$

The walker is killed, survives, or is cloned as in the simple DMC algorithm.

- **Adjust N:** As in the simple DMC algorithm, the number of walkers is stabilized at the target value by adjusting

$$E_{\rm T} \longrightarrow E_{\rm T} + \alpha \ln\left(\frac{N_{\rm T}}{N}\right) ,$$

where α is a small positive parameter. The value of $E_{\rm T}$ is accumulated as a measure of the ground state energy.

- **Clean up:** The killed walkers are removed from the ensemble.

- **Compute averages:** The ground state energy is the average of $E_{\rm T}$ values over the Monte Carlo Steps.

Guided DMC Program for Helium

gdmc-he.cpp

```
// Guide Function Diffusion Monte Carlo for Helium Atom

#include <cmath>
#include <cstdlib>
#include <iostream>
#include "rng.h"

using namespace std;

int seed = -987654321;           // seed for ran2 and gasdev

int N;                           // current number of walkers
int N_T;                         // desired target number of walkers
const int DIM = 6;               // dimension of R = (r1, r2)
double **R;                      // walker positions in 6-D space
```

```
    bool *alive;                    // is this walker alive?
```

Note that we use a 6-D vector R to represent the position coordinates of the two electrons. The following function handles memory allocation as in dmc.cpp.

gdmc-he.cpp

```
    void ensureCapacity(int index) {

        static int maxN = 0;        // remember size of the arrays

        if (index < maxN)
            return;                 // additional storage not needed

        int oldN = maxN;            // remember old capacity to copy values
        if (maxN > 0)
            maxN *= 2;              // double capacity
        else
            maxN = 1;
        if (index > maxN - 1)       // if this is not enough
            maxN = index + 1;       // increase to make it enough

        // allocate new storage
        double **newR = new double* [maxN];
        bool *newAlive = new bool [maxN];
        for (int n = 0; n < maxN; n++) {
            newR[n] = new double [DIM];
            if (n < oldN) {
                for (int d = 0; d < DIM; d++)
                    newR[n][d] = R[n][d];
                newAlive[n] = alive[n];
                delete [] R[n];       // release old memory
            }
        }

        // delete old storage and point to new
        delete [] R;
        R = newR;
        delete [] alive;
        alive = newAlive;
    }

    double ESum, ESqdSum;             // accumulators for observables

    void zeroAccumulators() {
        ESum = ESqdSum = 0;
    }

    double dt;                        // time step Delta_t set by user
    double E_T;                       // target energy

    void initialize() {

        // create target number of walkers
        N = N_T;
        for (int n = 0; n < N; n++) {
            ensureCapacity(n);
            for (int d = 0; d < DIM; d++)
                R[n][d] = ran2(seed) - 0.5;
            alive[n] = true;
        }

        // set target energy close to VMC result
        E_T = -2.85;
    }
```

Computing the electron-nucleus and electron-electron separations

gdmc-he.cpp

```
    void findSeparations(double *R, double& r1, double& r2, double&r12) {

        // find electron-nucleus and electron-electron separations
        r1 = r2 = r12 = 0;
        for (int e1 = 0; e1 < 3; e1++) {
            int e2 = e1 + 3;           // second electron indices
            r1 += R[e1] * R[e1];
            r2 += R[e2] * R[e2];
            r12 += (R[e1] - R[e2]) * (R[e1] - R[e2]);
        }
        r1 = sqrt(r1);
        r2 = sqrt(r2);
        r12 = sqrt(r12);
    }
```

The Padé-Jastrow guide function

$$\Psi_{\mathrm{T}}(\mathbf{r}_1, \mathbf{r}_2) = e^{-2r_1 - 2r_2 + \frac{r_{12}}{2(1+\alpha r_{12})}}$$

gdmc-he.cpp

```
double alpha = 0.15;                // Pade-Jastrow wave function parameter

double Psi_T(double *R) {

    // value of guide function
    double r1, r2, r12;
    findSeparations(R, r1, r2, r12);
    double Psi_T = - 2 * r1 - 2 * r2 + r12 / (2 * (1 + alpha * r12));
    return exp(Psi_T);
}
```

The local energy

$$E_{\mathrm{L}}(\mathbf{r}_1, \mathbf{r}_2) = -4 + \frac{\alpha}{(1+\alpha r_{12})} + \frac{\alpha}{(1+\alpha r_{12})^2} + \frac{\alpha}{(1+\alpha r_{12})^3} - \frac{1}{4(1+\alpha r_{12})^4} + \frac{\hat{\mathbf{r}}_{12} \cdot (\hat{\mathbf{r}}_1 - \hat{\mathbf{r}}_2)}{(1+\alpha r_{12})^2} \, .$$

gdmc-he.cpp

```
double E_L(double *R) {

    // value of local energy for guide function
    double r1, r2, r12;
    findSeparations(R, r1, r2, r12);
    double dotProd = 0;
    for (int e1 = 0; e1 < 3; e1++) {
        int e2 = e1 + 3;            // second electron indices
        dotProd += (R[e1] - R[e2]) / r12 * (R[e1] / r1 - R[e2] / r2);
    }
```

```
        double denom = 1 / (1 + alpha * r12);
        double denom2 = denom * denom;
        double denom3 = denom2 * denom;
        double denom4 = denom2 * denom2;
        double E_L = - 4 + alpha * (denom + denom2 + denom3)
                     - denom4 / 4 + dotProd * denom2;
        return E_L;
    }
```

The Fokker-Planck force

$$\frac{2}{\Psi_{\rm T}(R)}\nabla_{r_1}\Psi_{\rm T}(R) = -4\hat{\mathbf{r}}_1 + \frac{\hat{\mathbf{r}}_{12}}{(1+\alpha r_{12})^2}$$
$$\frac{2}{\Psi_{\rm T}(R)}\nabla_{r_2}\Psi_{\rm T}(R) = -4\hat{\mathbf{r}}_2 + \frac{\hat{\mathbf{r}}_{21}}{(1+\alpha r_{12})^2}$$

gdmc-he.cpp

```
    void findForce(double *R, double *F) {

        // find Fokker-Planck forces
        double r1, r2, r12;
        findSeparations(R, r1, r2, r12);
        for (int d = 0; d < DIM; d++)
            F[d] = 0;
        double denom2 = 1 / (1 + alpha * r12);
        denom2 *= denom2;
        for (int e1 = 0; e1 < 3; e1++) {
            int e2 = e1 + 3;
            F[e1] += - 4 * R[e1] / r1 + denom2 * (R[e1] - R[e2]) / r12;
        F[e2] += - 4 * R[e2] / r2 + denom2 * (R[e2] - R[e1]) / r12;
    }
}
```

The Fokker-Planck Green's function

$$G(R', R; \Delta\tau) = \frac{1}{\sqrt{2\pi\Delta\tau}} \exp\left[-\frac{\left(R' - R - \frac{1}{2}F(R)\Delta\tau\right)^2}{2\Delta\tau}\right] .$$

gdmc-he.cpp

```
double G(double *RPrime, double *R) {

    // value of Fokker-Planck Green's function exponential
    double F[DIM];
    findForce(R, F);
    double G = 0;
    for (int d = 0; d < DIM; d++) {
        double dR = RPrime[d] - R[d] - F[d] * dt / 2;
        G += dR * dR;
    }
    return exp(- G / (2 * dt));
}

int nTrials;                    // number of Metropolis tests
int nAccept;                    // number of acceptances

void oneMonteCarloStep(int n) {
```

```
// define position variables for this walker
double R[DIM];
for (int d = 0; d < DIM; d++)
    R[d] = ::R[n][d];
```

Trial shift in R

$$R' = R + \frac{1}{2}\Delta\tau F(R) + \eta\sqrt{\Delta\tau}\ .$$

gdmc-he.cpp

```
// trial shift walker to new position
double F[DIM], RPrime[DIM];
findForce(R, F);
for (int d = 0; d < DIM; d++)
    RPrime[d] = R[d] + F[d] * dt / 2 + gasdev(seed) * sqrt(dt);
```

The Metropolis acceptance test

$$w = \frac{G(R, R'; \Delta\tau)\rho(R')}{G(R', R; \Delta\tau)\rho(R)} \simeq \frac{G(R, R'; \Delta\tau)\Psi_{\rm T}^2(R')}{G(R', R; \Delta\tau)\Psi_{\rm T}^2(R)}\ .$$

gdmc-he.cpp

```
// Metropolis acceptance test
double w = Psi_T(RPrime) / Psi_T(R);
w *= w * G(R, RPrime) / G(RPrime, R);
++nTrials;
if (w > ran2(seed)) {
    for (int d = 0; d < DIM; d++)
```

```
            ::R[n][d] = RPrime[d];
        ++nAccept;
    } else {
        return;                 // don't do branching step below
    }
```

The branching process

The textbook has two different suggestions for the branching test:

$$q = e^{-\Delta\tau\left[\frac{E_{\rm L}(R')+E_{\rm L}(R)}{2} - E_{\rm T}\right]} ,$$

on page 334, and

$$q = e^{-\Delta\tau\left[E_{\rm L}(R') - E_{\rm T}\right]} ,$$

on page 333, which we use here.

gdmc-he.cpp

```
    // branching step
    double q = exp(- dt * (E_L(RPrime) - E_T));

    int survivors = int(q);
    if (q - survivors > ran2(seed))
        ++survivors;

    // append survivors-1 copies of the walker to the end of the array
    for (int i = 0; i < survivors - 1; i++) {
        ensureCapacity(N);
        for (int d = 0; d < DIM; d++)
            ::R[N][d] = RPrime[d];
        alive[N] = true;
        ++N;
    }

    // if survivors is zero, then kill the walker
    if (survivors == 0)
        alive[n] = false;
}

void oneTimeStep() {

    // DMC step for each walker
    int N_0 = N;
    for (int n = 0; n < N_0; n++)
        oneMonteCarloStep(n);

    // adjust E_T
    E_T += log(N_T / double(N)) / 10;

    // remove all dead walkers from the arrays
    int newN = 0;
    for (int n = 0; n < N; n++)
    if (alive[n]) {
        if (n != newN) {
            for (int d = 0; d < DIM; d++)
                R[newN][d] = R[n][d];
            alive[newN] = true;
        }
        ++newN;
    }
    N = newN;

    // measure energy, wave function
    ESum += E_T;
    ESqdSum += E_T * E_T;
}

int main() {

    cout << " Guide Function Diffusion Monte Carlo for Helium Atom\n"
         << " -----------------------------------------------------\n";
    cout << " Enter desired target number of walkers:  ";
    cin >> N_T;
    cout << " Enter time step dt:  ";
    cin >> dt;
    cout << " Enter total number of time steps:  ";
    int timeSteps;
    cin >> timeSteps;

    initialize();

    // do 20% of timeSteps as thermalization steps
    int thermSteps = int(0.2 * timeSteps);
    int adjustInterval = int(0.1 * thermSteps) + 1;
    nTrials = nAccept = 0;
    cout << " Performing " << thermSteps << " thermalization steps ..."
         << flush;
    for (int i = 0; i < thermSteps; i++) {
```

```
        oneTimeStep();
        if ((i+1) % adjustInterval == 0) {
            dt *= nAccept / (0.9 * nTrials);
            nTrials = nAccept = 0;
        }
    }
    cout << "\n Adjusted time step size = " << dt << endl;

    // production steps
    zeroAccumulators();
    for (int i = 0; i < timeSteps; i++) {
        oneTimeStep();
    }

    // compute averages
    double EAve = ESum / timeSteps;
    double EVar = ESqdSum / timeSteps - EAve * EAve;
    cout << " <E> = " << EAve << " +/- " << sqrt(EVar / timeSteps) << endl;
    cout << " <E^2> - <E>^2 = " << EVar << endl;
}
```

Results from `gdmc-he.cpp`

```
 Guide Function Diffusion Monte Carlo for Helium Atom
 ----------------------------------------------------
 Enter desired target number of walkers:  1000
 Enter time step dt:  0.01
 Enter total number of time steps:  4000
 Performing 800 thermalization steps ...
```

```
Adjusted time step size = 0.0231441
<E> = -2.90311 +/- 0.0111302
<E^2> - <E>^2 = 0.495522
```

These results can be compared with

Method	**Energy (Hartrees)**
Wavefunction $e^{-2r_1-2r_2}$	-2.75
$e^{-\alpha(r_1-r_2)}$ with $\alpha = 27/16$	-2.84765
Hartree-Fock (Chapter 4)	-2.8617
Padé-Jastrow `vmc-he.cpp`	-2.878
Hylleraas (1929) 10 variational parameters	-2.90363
Pekeris (1959) 1,078 variational parameters	-2.90372
Experiment	-2.90372