

Diffusion Monte Carlo

We have seen that the diffusion and Fokker-Planck equations can be solved using random walks to generate any desired trial wave function.

Connection with quantum mechanics

Consider the time-dependent Schrödinger equation for a free particle moving in one dimensions:

$$i\hbar \frac{\partial \psi(x, t)}{\partial t} = -\frac{\hbar^2}{2m} \frac{\partial^2 \psi(x, t)}{\partial x^2} ,$$

where m is the mass of the particle. This equation can be written

$$\frac{\partial \psi(x, t)}{\partial t} = \frac{i\hbar}{2m} \frac{\partial^2 \psi(x, t)}{\partial x^2} = \gamma_{\text{im}} \frac{\partial^2 \psi(x, t)}{\partial x^2} ,$$

which is exactly of the form of a diffusion equation, but with an *imaginary* diffusion constant

$$\gamma_{\text{im}} = \frac{i\hbar}{2m} .$$

Another way to write this equation with a real diffusion constant is to analytically continue the time $t \rightarrow -i\tau$ to imaginary values:

$$\frac{\partial \psi(x, \tau)}{\partial \tau} = \frac{\hbar}{2m} \frac{\partial^2 \psi(x, \tau)}{\partial x^2} .$$

Thus the motion of a quantum particle is equivalent to diffusion of a cloud of particles in imaginary time!

Diffusion leads the system into its ground state

Any initial wave function of the system can be expanded in a complete set of energy eigenfunctions:

$$\Psi(x, 0) = \sum_{n=0}^{\infty} c_n \psi_n(x) .$$

The solution of the real time Schrödinger equation is then

$$\Psi(x, t) = \sum_{n=0}^{\infty} c_n e^{-iE_n t/\hbar} \psi_n(x) .$$

The solution of the imaginary time equation is got by analytically continuing this solution to imaginary time $t \rightarrow -i\tau$:

$$\Psi(x, \tau) = \sum_{n=0}^{\infty} c_n e^{-E_n \tau/\hbar} \psi_n(x) .$$

As $\tau \rightarrow \infty$, each mode in this equation is exponentially damped, with higher energies damped faster than lower energies. The ground state wave function can be extracted using the following limit:

$$\lim_{\tau \rightarrow \infty} e^{E_0 \tau/\hbar} \Psi(x, \tau) = \lim_{\tau \rightarrow \infty} \sum_n c_n e^{-(E_n - E_0) \tau/\hbar} \psi_n(x) = c_0 \psi_0(x) .$$

This result is the basis of the diffusion Monte Carlo approach.

Diffusion with a potential energy term

The equations considered above were for a free particle. A free particle is not very interesting, so let's generalize this approach to a particle moving in a potential $V(x)$ for which the imaginary time equation to be solved is

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

where we have set $\hbar = 1$ and $m = 1$.

We have seen in the last lecture that if $V = 0$, then this equation can be solved using a Green's function

$$\rho(y, \tau) = \int dx G(x, y; \tau) \rho(x, 0) , \quad G(x, y; \tau) = \frac{1}{\sqrt{2\pi\tau}} e^{-(x-y)^2/(2\tau)} ,$$

for the probability density $\rho(x, \tau) = |\psi(x, \tau)|^2$. This solution preserves probability (or total number of particles in the diffusion problem).

The problem with adding the potential energy term is that it spoils this conservation of probability. This can be seen by neglecting the kinetic energy term:

$$\frac{\partial \psi(x, \tau)}{\partial \tau} = -V(x)\psi(x, \tau), \quad \psi(x, \tau) = e^{-V(x)\tau}\psi(x, 0),$$

which implies that

$$\lim_{\tau \rightarrow \infty} \psi(x, \tau) = \begin{cases} 0 & \text{where } V(x) > 0 \\ \psi(x, 0) & \text{where } V(x) = 0 \\ \infty & \text{where } V(x) < 0 \end{cases}$$

Depending on the potential, the net probability $\int dx |\psi(x, \tau)|^2$ could go to zero or to infinity!

In the diffusion Monte Carlo method, this problem with the potential energy term is solved by modifying the equation as follows:

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

where the quantity E_T is adjusted as a function of τ so that the probability (number of walkers in the diffusion approach) remains constant. If in the limit $\tau \rightarrow \infty$ the solution $\psi(x, \tau) \rightarrow \psi(x)$ becomes independent of τ , i.e., $\partial \psi / \partial \tau = 0$, then

$$-\frac{1}{2} \frac{d^2 \psi(x)}{dx^2} + V(x)\psi(x) = E_T \psi(x),$$

that is, $\psi(x, \tau)$ tends to an eigenfunction of the quantum mechanical problem, and E_T is the energy eigenvalue!

Diffusion Monte Carlo algorithm

The DMC algorithm is based on the ideas that the kinetic energy term can be represented by diffusion of random walkers, and the potential energy causes the number of walkers at a given point x to grow or decay. A simple form of the algorithm is as follows:

- **Initialization:** Choose a time step $\Delta\tau$ and a target number N_T of random walkers which are randomly located in a region where the wave function is expected to be large. Also choose a value for the parameter E_T .
- **Time Step:** The following two operations are carried out on each of the current number N of walkers:
 - **Diffusion Step:** The kinetic energy shifts the walker to a new position with a step chosen at random from a Gaussian distribution with variance Δt , exactly as in the case of a free particle.
 - **Branching Step:** The potential energy, modified by the E_T parameter, causes a growth or decay in the number of walkers. This effect is implemented by computing

$$q = e^{-\Delta\tau[V(x)-E_T]} .$$

The value of q determines whether this walker dies, survives, or is cloned. Note that $q > 0$. Let $\lfloor q \rfloor$ be its integer part. Then $q - \lfloor q \rfloor$ lies between 0 and 1. The walker is replaced with $\lfloor q \rfloor$ identical copies with probability $1 - (q - \lfloor q \rfloor)$ and $\lfloor q \rfloor + 1$ copies with probability $q - \lfloor q \rfloor$.

- **Adjusting the value of E_T :** At the end of the time step, the number of walkers N will have changed due to branching. If $N > N_T$, then we need to *increase* E_T which will tend to *reduce* q and hence tend to kill walkers. Conversely, if $N < N_T$, then *reducing* E_T will *increase* q and hence tend to generate more clones. The textbook suggests

$$E_T \longrightarrow E_T + \alpha \ln \left(\frac{N_T}{N} \right) ,$$

where α is a small positive parameter.

Diffusion Monte Carlo program for the 3-D harmonic oscillator

The following program implements the DMC algorithm outlined above for the 3-D harmonic oscillator which has ground state energy and wave function

$$E_0 = \frac{3}{2} , \quad \psi_0 = \frac{e^{-r^2/2}}{(2\pi)^{3/2}} ,$$

using units with $m = \omega = \hbar = 1$.

```
// Diffusion Monte Carlo program for the 3-D harmonic oscillator

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

using namespace std;

int seed = -987654321;           // for ran2 and gasdev
const int DIM = 3;              // dimensionality of space
```

Potential energy function

This function evaluates the potential energy of the harmonic oscillator in D dimensions given the position $\mathbf{r}$ of the oscillator.

```
double V(double *r) {           // harmonic oscillator in DIM dimensions
    double rSqd = 0;
    for (int d = 0; d < DIM; d++)
        rSqd += r[d] * r[d];
    return 0.5 * rSqd;
}

double dt;                      // Delta_t set by user
double E_T;                     // target energy
```

```
// random walkers
int N;           // current number of walkers
int N_T;        // desired target number of walkers
double **r;     // x,y,z positions of walkers
bool *alive;    // is this walker alive?
```

Dynamical adjustment of array storage

Since the number of walkers N will change with time, we can either allocated large-enough arrays to accomodate this growth, or we can grow the arrays dynamically if necessary while the program is running. The following function is called when the N might have changed and we wish to check whether an index is legal. If the array is too small to accomodate that index, it is replaced with a larger array with the values of the original elements preserved.

```
void ensureCapacity(int index) {

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

    if (index < maxN)        // no need to expand array
        return;             // do nothing

    int oldMaxN = maxN;      // remember the old capacity
    if (maxN > 0)
        maxN *= 2;          // double capacity
    else
        maxN = 1;
    if (index > maxN - 1)    // if this is not sufficient
        maxN = index + 1;   // increase it so it is sufficient
```

```
// allocate new storage
double **rNew = new double* [maxN];
bool *newAlive = new bool [maxN];
for (int n = 0; n < maxN; n++) {
    rNew[n] = new double [DIM];
    if (n < oldMaxN) {        // copy old values into new arrays
        for (int d = 0; d < DIM; d++)
            rNew[n][d] = r[n][d];
        newAlive[n] = alive[n];
        delete [] r[n];      // release old memory
    }
}
delete [] r;                // release old memory
r = rNew;                   // point r to the new memory
delete [] alive;
alive = newAlive;
}
```

We need to measure the energy, its variance, and the wave function of the ground state.

```
// observables
double ESum;                // accumulator for energy
double ESqdSum;             // accumulator for variance
double rMax = 4;            // max value of r to measure psi
const int NPSI = 100;       // number of bins for wave function
double psi[NPSI];           // wave function histogram

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

```
    for (int i = 0; i < NPSI; i++)
        psi[i] = 0;
}

void initialize() {
    N = N_T;                // set N to target number specified by user
    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;
    }
    zeroAccumulators();
    E_T = 0;                // initial guess for the ground state energy
}
```

One Diffusion Monte Carlo step

The following function implements the Diffusion Monte Carlo step algorithm on a particular walker. Recall that

- A Gaussian diffusive step is taken with step size $\sqrt{\Delta t}$.
- A branching step is implemented with the walker dying, surviving or being cloned, depending on its potential energy.

```
void oneMonteCarloStep(int n) {

    // Diffusive step
    for (int d = 0; d < DIM; d++)
```

```
    r[n][d] += gasdev(seed) * sqrt(dt);

// Branching step
double q = exp(- dt * (V(r[n]) - 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] = r[n][d];
    alive[N] = true;
    ++N;
}

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

One time step Δt

One time step Δt consists in the following:

- One DMC step is performed on each walker in turn.
- To make the living walkers easier to access, dead walkers are removed from the arrays.

- E_T is adjusted to drive N towards N_T .
- Data is accumulated to measure $\langle E \rangle$, its variance, and the ground state wave function.

```
void oneTimeStep() {  
  
    // DMC step for each walker  
    int N_0 = N;  
    for (int n = 0; n < N_0; n++)  
        oneMonteCarloStep(n);  
  
    // 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;  
  
    // adjust E_T  
    E_T += log(N_T / double(N)) / 10;  
  
    // measure energy, wave function  
    ESum += E_T;  
    ESqdSum += E_T * E_T;  
}
```

```

for (int n = 0; n < N; n++) {
    double rSqd = 0;
    for (int d = 0; d < DIM; d++)
        rSqd = r[n][d] * r[n][d];
    int i = int(sqrt(rSqd) / rMax * NPSI);
    if (i < NPSI)
        psi[i] += 1;
}
}

```

The main function to steer the calculation

The user specifies the number of walkers, the time step size, and number of time steps. After initialization, 20% of the specified number of time steps are run to equilibrate the walkers. Then the production steps are taken. The Monte Carlo wave function and the exact wave function, both normalized unity in the plotting interval, are output to a file.

```

int main() {

    cout << " Diffusion Monte Carlo for the 3-D Harmonic Oscillator\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);
for (int i = 0; i < thermSteps; i++)
    oneTimeStep();

// 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;
double psiNorm = 0, psiExactNorm = 0;
double dr = rMax / NPSI;
for (int i = 0; i < NPSI; i++) {
    double r = i * dr;
    psiNorm += r * r * psi[i] * psi[i];
    psiExactNorm += r * r * exp(- r * r);
}
psiNorm = sqrt(psiNorm);
psiExactNorm = sqrt(psiExactNorm);
ofstream file("psi.data");
for (int i = 0; i < NPSI; i++) {
```

```
        double r = i * dr;
        file << r << '\t' << r * r * psi[i] / psiNorm << '\t'
            << r * r * exp(- r * r / 2) / psiExactNorm << '\n';
    }
    file.close();
}
```

Output of the program

```
Diffusion Monte Carlo for the 3-D Harmonic Oscillator
-----
Enter desired target number of walkers:  300
Enter time step dt:  0.05
Enter total number of time steps:  4000
<E> = 1.49113 +/- 0.0127478
<E^2> - <E>^2 = 0.650031
```

