

Diffusion equation approach to quantum systems

In this approach, the ground state of the system is found by modeling a diffusion process. There are several approaches to implementing this idea: we will develop programs for the Fokker-Planck and Diffusion Monte Carlo approaches.

Diffusion and random walks

Consider a random walk on a lattice with spacing a in one dimension. The rule for the walk is that if the walker is at position x at time t , then at time $t + h$, the walker moves to the neighbor sites $x \pm a$ with equal probabilities α and remains at x with probability $1 - 2\alpha$: the sum of the three probabilities add up to 1.

Let's consider an *ensemble* of a large number of such walkers. The number density of walkers is $\rho(x, t)$, which means that, at time t , the number of walkers between x and $x + dx$ is $\rho(x, t)dx$. Note: each walker moves on a lattice, but the lattices of different walkers are in general different.

The *master equation*

$$\rho(x, t + h) - \rho(x, t) = \alpha\rho(x + a, t) + \alpha\rho(x - a, t) - 2\alpha\rho(x, t) ,$$

says that the density of walkers at x increases in one time step h due to walkers from $x \pm a$ moving to x with probability α , and decreases due to walkers moving from x to $x \pm a$ with probability α .

If h and a are both small, then we can use Taylor expansions

$$\rho(x, t + h) = \rho(x, t) + h \frac{\partial \rho}{\partial t} + \dots \quad \rho(x \pm a, t) = \rho(x, t) \pm a \frac{\partial \rho}{\partial x} + \frac{1}{2} a^2 \frac{\partial^2 \rho}{\partial x^2} + \dots$$

In the *continuum limit* $h \rightarrow 0$, $a \rightarrow 0$ with a^2/h held constant, we obtain the *diffusion equation*

$$\frac{\partial \rho}{\partial t} = \gamma \frac{\partial^2 \rho}{\partial x^2} ,$$

where

$$\gamma \equiv \lim_{h,a \rightarrow 0} \frac{\alpha a^2}{h} ,$$

is called the *diffusion constant* for the system of walkers.

Green's function for the diffusion equation

The density of walkers at time t can be computed from the initial density using the formula

$$\rho(y, t) = \int dx G(x, y; t) \rho(x, 0) , \quad G(x, y; t) = \frac{1}{\sqrt{4\pi\gamma t}} e^{-(x-y)^2/(4\gamma t)} ,$$

where $G(x, y; t)$ is a *Green's function* with the properties

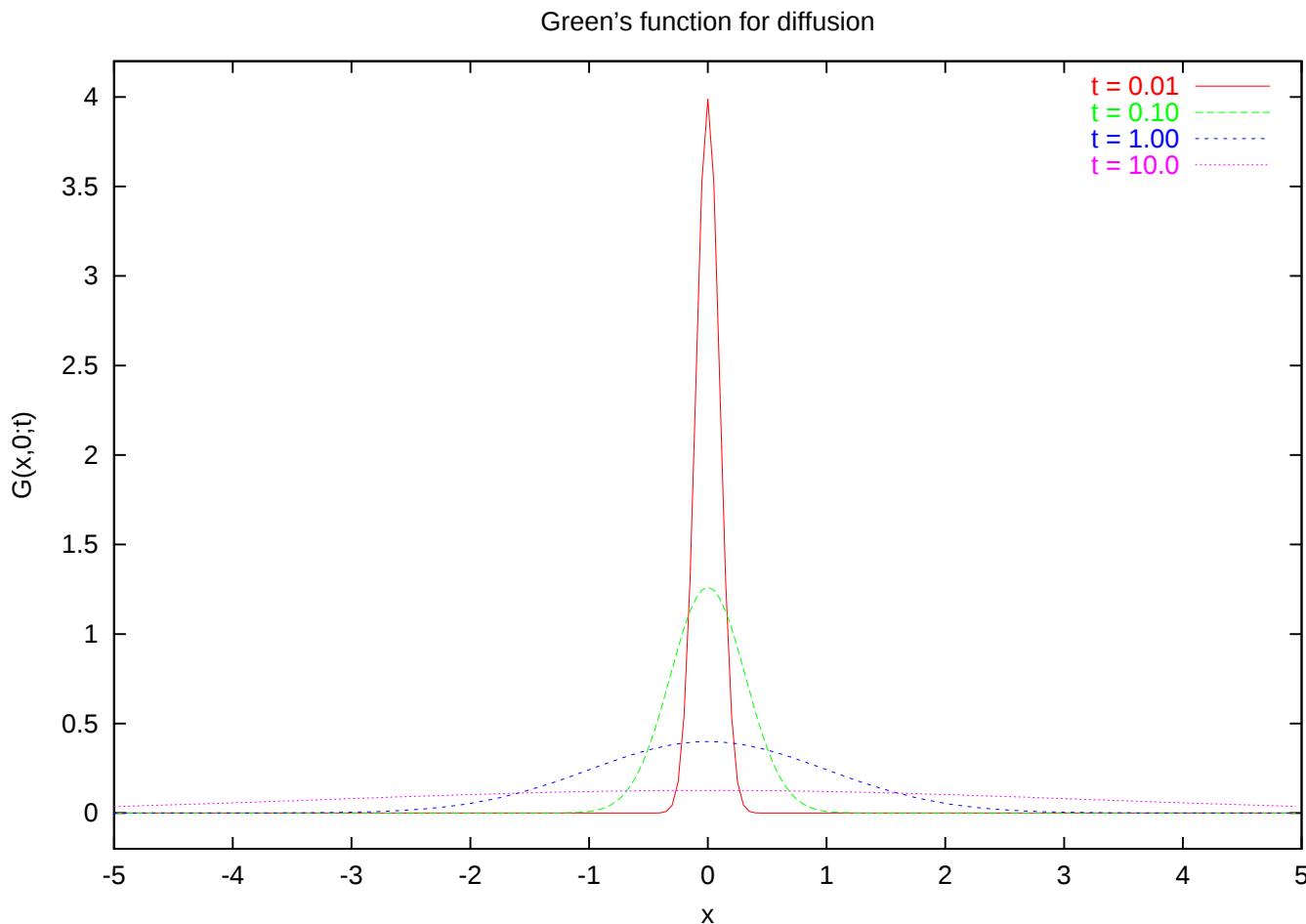
$$G(x, y; 0) = \delta(x - y) , \quad \text{and} \quad \int dx G(x, y; t) = 1 .$$

In fact, $G(x, y; t)$ is the probability that a walker at x (or y) at time $t = 0$ moves to y (or x) at time t . This provides a way of implementing the random walk:

- Choose a step size Δt in time
- A walker at $x(t)$ at time t moves to $x(t + \Delta t) = x(t) + \eta\sqrt{\Delta t}$, where η is chosen randomly from a Gaussian distribution with variance $\sigma^2 = 2\gamma$.

Let's consider this step as a trial step in the Metropolis algorithm. Do we need to make a Metropolis type test before accepting the step? The answer is no, because the step is chosen according to a probability function which drives the distribution exactly to equilibrium as a function of time t .

Another way of seeing that every step can be accepted is from the physical meaning of diffusion. Typically, we have a dilute collection of non-interacting particles in a medium which can be considered to be a heat bath at



constant temperature T . The particles undergo random thermal motion due to collisions with the molecules of the medium. The temperature of the medium determines the diffusion constant via Einstein's relation

$$\gamma = \frac{k_B T}{\beta},$$

where β is the drag coefficient, e.g., $\beta = 6\pi\eta R$ for Brownian spheres of radius R moving in fluid with kinematic viscosity η (not to be confused with the Gaussian deviate in the step). Since the diffusing particles are non-interacting, there is no energy cost when they move.

Fokker-Planck equation

Suppose we wish to find the average energy of a quantum system with a trial wave function $\Psi_T(x)$. This can be done using a diffusion process with a density of walkers $\rho(x, \tau)$ such that

$$\lim_{t \rightarrow \infty} \rho(x, t) = \rho(x) = |\Psi_T(x)|^2 .$$

Can we use the diffusion equation to generate $\rho(x)$? The problem with the diffusion equation is that the density of walkers $\rho(x, t)$ tends to a constant as $t \rightarrow \infty$.

There is a modified diffusion equation, however, for which the density of walkers tends to a non-constant function as $t \rightarrow \infty$. This is the *Fokker-Planck* equation

$$\frac{\partial \rho(x, t)}{\partial t} = \frac{1}{2} \frac{\partial}{\partial x} \left[\frac{\partial}{\partial x} - F(x) \right] \rho(x, t) ,$$

where the “Force” is determined by the desired density at $t = \infty$:

$$F(x) = \frac{1}{\rho(x)} \frac{d\rho(x)}{dx} .$$

Note that the Fokker-Planck equation can be written

$$\frac{\partial \rho(x, t)}{\partial t} = \frac{1}{2} \left[\frac{\partial^2 \rho(x, t)}{\partial x^2} - \frac{\rho(x, t)}{\rho(x)} \frac{d^2 \rho(x)}{dx^2} + \frac{\rho(x, t)}{\rho^2(x)} \left(\frac{d\rho(x)}{dx} \right)^2 - \frac{1}{\rho(x)} \frac{d\rho(x)}{dx} \frac{\partial \rho(x, t)}{\partial x} \right] .$$

The right hand side vanishes if $\rho(x, t) = \rho(x)$, and hence $\partial \rho / \partial t$ tends to zero, and the density becomes independent of time in this limit.

Approximate Green's function for the Fokker-Planck equation

Unfortunately, a closed form for the Green's function cannot be obtained for arbitrary $F(x)$. However, we can use the following *approximate* form

$$G(x, y; \Delta t) = \frac{1}{\sqrt{2\pi\Delta t}} e^{-[y-x-F(x)\Delta t/2]^2/(2\Delta t)} ,$$

which has the necessary properties

$$\lim_{\Delta t \rightarrow 0} G(x, y; \Delta t) = \delta(x - y) , \quad \int dy G(x, y; \Delta t) = 1 ,$$

for a probability interpretation. This function does not however obey the Fokker-Planck equation: but if Δt is small, then the error is of order Δt^2 .

To implement a random walk with this Green's function, suppose the walker is at position x at time t , then he moves to y which is Gaussian-distributed around $x + F(x)\Delta t/2$ with variance Δt :

$$x(t + \Delta t) = x(t) + F(x)\Delta t/2 + \eta\sqrt{\Delta t} ,$$

where η is chosen from a Gaussian distribution with unit variance.

Metropolis correction to Fokker-Planck walk

If the Green's function for the Fokker-Planck equation were exact, then the random walk algorithm given above would inevitably drive any initial distribution to the desired equilibrium distribution $\rho(x)$, just as in the case of the diffusion equation. However, the Green's function is *not* exact, and there will be errors in the evolution of the distribution of $\mathcal{O}(\Delta t^2)$. If Δt is chosen small, the errors will be small, but it will take a long time to generate $\rho(x)$.

These errors can be corrected using the Metropolis algorithm. The Fokker-Planck step from $x(t)$ to $x(t + \Delta t)$ is considered as a *trial step* for the Metropolis algorithm. If the Green's function were exact, then we would have for the ratio of step probabilities

$$\frac{T(x \rightarrow y)}{T(y \rightarrow x)} = \frac{G(x, y; \Delta t)}{G(y, x; \Delta t)} = \frac{\rho(y)}{\rho(x)} .$$

Since the Green's function is not exact, we need to push the ratio

$$w = \frac{T(y \rightarrow x)\rho(y)}{T(x \rightarrow y)\rho(x)} = \frac{G(y, x; \Delta t)\rho(y)}{G(x, y; \Delta t)\rho(x)}$$

toward unity. If $T(x \rightarrow y)$ is too small compared with $T(y \rightarrow x)$, this will tend to make $w > 1$: in this case the step from x to y should always be accepted. Conversely, if $T(x \rightarrow y)$ is too large compared with $T(y \rightarrow x)$, then accepting the step conditionally by checking whether w is larger than a uniform deviate will tend to correct the problem.

Fokker-Planck equation approach to VMC

The algorithms described above are coded in the following program `vmc-fp.cpp`. The system chosen is the Harmonic Oscillator moving in one dimension. For trial wave function, we choose a Gaussian:

$$\Psi_T(x) \sim e^{-\alpha x^2}, \quad \rho(x) \sim |\Psi_T(x)|^2 \sim e^{-2\alpha x^2},$$

where α is a variational parameter. The “Force” in the Fokker-Planck equation is then

$$F(x) = \frac{1}{\rho(x)} \frac{d\rho(x)}{dx} = -4\alpha x.$$

Note that this “Force” acts towards the equilibrium position $x = 0$ of the oscillator, which is reasonable!

```
// Fokker-Planck equation approach to VMC
```

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

```
using namespace std;

int N;                // number of walkers
double *x;            // positions of walkers
double alpha;        // variational parameter
double tStep;        // time step
int seed = -987654321; // for random ran2 and gasdev

double eSum;          // accumulator to find energy
double eSqdSum;       // accumulator to find fluctuations in E

void initialize() {

    x = new double [N];
    for (int i = 0; i < N; i++)
        x[i] = qadran() - 0.5;
    tStep = 0.1;
}

void zeroAccumulators() {
    eSum = eSqdSum = 0;
}

double eLocal(double x) {

    // compute the local energy
    return alpha + x * x * (0.5 - 2 * alpha * alpha);
}
```

```
int nAccept;           // accumulator for number of accepted steps
```

The changes from the simple VMC program `vmc.cpp` are in the following function. The chosen walker is provisionally moved to to a new position

$$y = x + \eta\sqrt{\Delta t} + \frac{1}{2}F(x)\Delta t ,$$

where

$$F(x) = -4\alpha x .$$

```
void MetropolisStep(int n) {
    // make a trial move
    double x = ::x[n];           // :: chooses the global x
    double Fx = - 4 * alpha * x;
    double y = x + gasdev(seed) * sqrt(tStep) + Fx * tStep / 2;
```

The Metropolis test involves the ratio

$$w = \frac{G(y, x; \Delta t)\rho(y)}{G(x, y; \Delta t)\rho(x)} = \frac{e^{-(x-y-F(y)\Delta t/2)^2/(2\Delta t)} e^{-2\alpha y^2}}{e^{-(y-x-F(x)\Delta t/2)^2/(2\Delta t)} e^{-2\alpha x^2}} ,$$

```
// compute ratio for Metropolis test
double rhoRatio = exp( - 2 * alpha * (y * y - x * x));
double oldExp = y - x - Fx * tStep / 2;
double Fy = - 4 * alpha * y;
```

```
double newExp = x - y - Fy * tStep / 2;
double GRatio = exp( -(newExp * newExp - oldExp * oldExp) / (2 * tStep));
double w = rhoRatio * GRatio;

// Metropolis test
if (w > ran2(seed)) {
    ::x[n] = x = y;
    ++nAccept;
}

// accumulate energy and wave function
double e = eLocal(x);
eSum += e;
eSqdSum += e * e;
}

void oneMonteCarloStep() {

    // perform N Metropolis steps
    for (int n = 0; n < N; n++) {
        MetropolisStep(n);
    }
}
```

The `main` function is almost identical to that in `vmc.cpp`. Following Thijssen's Fortran program, time step size Δt is adjusted in the thermalization phase of the Monte Carlo so that the acceptance ratio for the Metropolis tests is 90%. This is reasonable because the Metropolis test would not be needed were it not for the fact that the Green's function is not exact.

```
int main() {

    cout << " Fokker-Planck approach to VMC: Harmonic Oscillator\n"
         << " -----\n"
         << " Enter number of walkers:  ";
    cin >> N;
    cout << " Enter variational parameter alpha:  ";
    cin >> alpha;
    cout << " Enter number of Monte Carlo steps:  ";
    int MCSteps;
    cin >> MCSteps;

    initialize();

    // perform 20% of MCSteps as thermalization steps
    // and adjust time step size so acceptance ratio ~90%
    int thermSteps = int(0.2 * MCSteps);
    int adjustInterval = int(0.1 * thermSteps) + 1;
    nAccept = 0;
    cout << " Performing " << thermSteps << " thermalization steps ..."
         << flush;
    for (int i = 0; i < thermSteps; i++) {
        oneMonteCarloStep();
        if ((i+1) % adjustInterval == 0) {
            tStep *= nAccept / (0.9 * N * adjustInterval);
            nAccept = 0;
        }
    }
    cout << "\n Adjusted time step size = " << tStep << endl;
```

```
// production steps
zeroAccumulators();
nAccept = 0;
cout << " Performing " << MCSteps << " production steps ..." << flush;
for (int i = 0; i < MCSteps; i++)
    oneMonteCarloStep();

// compute and print energy
double eAve = eSum / double(N) / MCSteps;
double eVar = eSqdSum / double(N) / MCSteps - eAve * eAve;
double error = sqrt(eVar) / sqrt(double(N) * MCSteps);
cout << "\n <Energy> = " << eAve << " +/- " << error
    << "\n Variance = " << eVar << endl;
}
```