Quantum Monte Carlo Methods

In this topic, we will study simple quantum mechanical systems of particles which have bound states. The simplest such system is the quantum harmonic oscillator in one dimension. To find the energy eigenstates, we solve the time-independent Schrödinger equation

$$H\psi(x) = \left[-\frac{\hbar^2}{2m}\frac{\mathrm{d}^2}{\mathrm{d}x^2} + \frac{1}{2}m\omega^2x^2\right]\psi(x) = E\psi(x) ,$$

subject to boundary conditions

$$\lim_{x\to\pm\infty}\psi(x) = 0 .$$

If you are not familiar with quantum mechanics, you can just view this as an interesting example of an eigenvalue problem for a second-order ordinary differential equation. The solution of this equation is the *wave function* of the particle. The interpretation of the wave function $\psi(x)$ is that

$$|\psi(x)|^2 \ \mathrm{d}x$$

is the *probability* of finding the particle between position x and position $x + \mathrm{d}x$.

Such a simple one-dimensional problem can easily be solved numerically using deterministic algorithms described in Appendix A.7.1 of Thijssen's book. In fact, the harmonic oscillator problem can be solved exactly. Solutions which satisfy the boundary conditions exist only for discrete *eigenvalues* of the energy

$$E_n = \left(n + \frac{1}{2}\right)\hbar\omega \qquad n = 0, 1, 2, 3, \ldots$$

and the normalized energy *eigenfunctions* are given by

$$\psi_n(x) = \left(\frac{m\omega}{\pi\hbar}\right)^{\frac{1}{4}}\frac{1}{\sqrt{2^n n!}}H_n\left(x\sqrt{\frac{m\omega}{\hbar}}\right)e^{-m\omega x^2/(2\hbar)} ,$$

where H_n are Hermite polynomials

$$H_0(y) = 1 , \qquad H_1(y) = 2y , \qquad H_2(y) = 4y^2 - 2 , \qquad \text{etc.}$$

Exact solutions have been found only for a very small number of problems which can essentially be reduced to one-dimensional ordinary differential equations. Another example is the hydrogen atom which consists of a proton and an electron interacting through a Coulomb force.

The variational theorem

The eigenfunctions of a quantum mechanical problem are *complete*. This means that any wave function $\Psi(x)$ can be expressed as a linear superposition

$$\Psi(x) = \sum_n c_n \psi_n(x) \ ,$$

where c_n are complex numbers. According to the rules of quantum mechanics, the average energy of a particle with this wave function is given by

$$\langle E \rangle = \frac{\int \mathrm{d}x \ \Psi^*(x) H \Psi(x)}{\int \mathrm{d}x \ \Psi^*(x)\Psi(x)} \ .$$

The *variational theorem* states that $\langle E \rangle \geq E_0$ for *any* Ψ, and $\langle E \rangle = E_0$ if and only if $\Psi(x) = c_0\psi_0(x)$. It is easy to see this if we use the fact that the eigenfunctions $\psi_n(x)$ can be chosen to be *orthonormal*

$$\int \mathrm{d}x \ \psi_n^*(x)\psi_{n'}(x) = \delta_{nn'} \ ,$$

$$\langle E \rangle = \frac{\sum_{n,n'} c_n^* E_{n'} c_{n'} \int \mathrm{d}x \ \psi_n^*(x)\psi_{n'}(x)}{\sum_{n,n'} c_n^* c_{n'} \int \mathrm{d}x \ \psi_n^*(x)\psi_{n'}(x)} = \frac{\sum_n |c_n|^2 E_n}{\sum_n |c_n|^2} = E_0 + \frac{\sum_n |c_n|^2 (E_n - E_0)}{\sum_n |c_n|^2} \ ,$$

we have used the eigenvalue equation $H\psi_{n'} = E_{n'}\psi_{n'}$. Because $E_n - E_0 > 0$, the second term in the last expression is positive and larger than zero *unless* all $c_n = 0$ for all $n \neq 0$.

The *variational method* is based on this important theorem: to estimate the ground state energy and wave function, choose a *trial wave function* $\Psi_{\mathrm{T},\alpha}(x)$ which depends on a parameter α. The expectation value $\langle E \rangle$ will depend on the parameter α, which can be *varied* to *minimize* $\langle E \rangle$. This energy and the corresponding $\Psi_{\mathrm{T},\alpha}(x)$ then provide the best estimates for the ground state energy and wave function.

Mean-field variational methods

For most quantum systems which involve more than two particles, i.e., atomic nuclei and electrons, numerical methods must be used. *Deterministic variational methods* can be used to solve Schrödinger's equation for many-particle systems. These methods typically replace the effects of the many particles by an average *mean field*: each particle is then acted on by this field, thus reducing the problem to an effective one-particle system. This problem must be solved *self-consistently* because the mean field is determined by the positions of the particles, and the motion of the particles is determined by the mean field!

The problem with these methods is that they do not take into account *many-particle* effects and *correlations* between particles in a simple way.

Quantum Monte Carlo methods use random numbers and *random walks* to try to improve on deterministic variational methods.

Variational Monte Carlo (VMC)

In the Variational Monte Carlo method, a trial wave function $\Psi_{\mathrm{T},\alpha}$, which depends on a set of variational parameters $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_S)$, is carefully chosen.

An efficient way must be found to evaluate the expected value of the energy

$$\langle E \rangle = \frac{\int \mathrm{d}R\ \Psi^*_{\mathrm{T},\alpha} H \Psi_{\mathrm{T},\alpha}}{\int \mathrm{d}R\ \left|\Psi_{\mathrm{T},\alpha}\right|^2} ,$$

where $R = (\mathbf{r}_1, \ldots, \mathbf{r}_N)$ are the positions of the particles in the system. The problem is that this multi-dimensional integral must be evaluated many many times as the program searches the α parameter space for the minimum $\langle E \rangle$.

Monte Carlo methods can be used to evaluate multi-dimensional integrals much more efficiently than deterministic methods. The key to using a Monte Carlo method is to define a *positive definite weight function* which is used to sample the important regions of the multi-dimensional space. In the VMC method, the weight function is taken to be

$$\rho(R) = \frac{\left|\Psi_{\mathrm{T},\alpha}(R)\right|^2}{\int \mathrm{d}R\ \left|\Psi_{\mathrm{T},\alpha}\right|^2} .$$

The expected value of the energy can then be written

$$\langle E\rangle = \frac{\int \mathrm{d}R\ |\Psi_{\mathrm{T},\alpha}|^2 \frac{H\Psi_{\mathrm{T},\alpha}}{\Psi_{\mathrm{T},\alpha}}}{\int \mathrm{d}R\ |\Psi_{\mathrm{T},\alpha}|^2} = \int \mathrm{d}R\ \rho(R)E_L(R)\ ,$$

where the *local energy* $E_L(R)$ is defined by

$$E_L(R) = \frac{H\Psi_{\mathrm{T},\alpha}(R)}{\Psi_{\mathrm{T},\alpha}(R)}\ .$$

The variational wave function $\Psi_{\mathrm{T},\alpha}(R)$ is usually chosen to be real and non-zero (almost) everywhere in the region of integration. In evaluating the ground state of the system, it can generally be chosen to be real and positive definite.

The VMC strategy is to generate a random set of points $\{R_i\}$, $i = 1, \ldots, M$ in configuration space that are distributed according to $\rho(R)$. Then

$$\langle E\rangle = \frac{1}{M}\sum_{i=1}^{M} E_L(R_i)\ .$$

VMC Program for the Harmonic Oscillator

Thijssen's textbook suggests a simple variational trial wave function for the Harmonic Oscillator:

$$\Psi_{\mathrm{T},\alpha}(x) = e^{-\alpha x^2}\ .$$

Let's choose units so that $m = 1$, $\hbar = 1$, and $\omega = 1$. The Hamiltonian operator in these units is

$$H = -\frac{\mathrm{d}^2}{\mathrm{d}x^2} + \frac{1}{2}x^2\ ,$$

from which the local energy can be derived:

$$E_L(x) = \alpha + x^2\left(\frac{1}{2} - 2\alpha^2\right)\ .$$

Note that when $\alpha = 1/2$ we obtain the exact ground state energy and eigenfunction.

The following program `vmc.cpp` implements the VMC method outlined above.

vmc.cpp

```
// Variational Monte Carlo for the harmonic oscillator

#include <cmath>
#include <cstdlib>
#include <iostream>
#include <fstream>
#include "rng.h"
using namespace std;

int seed = -123456789;          // for random number generator
```

The program uses N Metropolis random walkers

The weight function

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

can easily be generated using a single Metropolis random walker, as was done in Topic 3 to evaluate a Gaussian integral. However, in more complex problems, it is conventional to use a large number of independent random walkers that are started at random points in the configuration space. This is beacuse the weight function can be very complicated in a multi-dimensional space: a single walker might have trouble locating all of the peaks in the distribution; using a large number of randomly located walkers improves the probability that the distribution will be correctly generated.

vmc.cpp

```
int N;                          // number of walkers
double *x;                      // walker positions
double delta;                   // step size
```

Variables to measure observables

Variables are introduced to accumulate E_L values and compute the Monte Carlo average and error estimate. The probability distribution is accumulated in a histogram with bins of size `dx` in the range $-10 \leq x \leq 10$.

vmc.cpp

```
double eSum;                    // accumulator to find energy
double eSqdSum;                 // accumulator to find fluctuations in E
double xMin = -10;              // minimum x for histogramming psi^2(x)
double xMax = +10;              // maximum x for histogramming psi^2(x)
double dx = 0.1;                // psi^2(x) histogram step size
double *psiSqd;                 // psi^2(x) histogram
int nPsiSqd;                    // size of array

void zeroAccumulators() {
    eSum = eSqdSum = 0;
    for (int i = 0; i < nPsiSqd; i++)
        psiSqd[i] = 0;
}
```

Initialization

The following function allocates memory to hold the positions of the walkers and distributes them uniformly at random in the range $-0.5 \leq x \leq 0.5$. The step size δ for the Metropolis walk is set to 1.

vmc.cpp

```
void initialize() {

    x = new double [N];
    for (int i = 0; i < N; i++)
        x[i] = ran2(seed) - 0.5;
    delta = 1;

    nPsiSqd = int((xMax - xMin) / dx);
    psiSqd = new double [nPsiSqd];

    zeroAccumulators();
}
```

Probability function and local energy

The following function evaluates the ratio

$$w = \frac{\rho\left(x_{\text{trial}}\right)}{\rho\left(x\right)} ,$$

which is used in the Metropolis algorithm: if $w \geq 1$ the step is accepted unconditionally; and if $w < 1$ the step is accepted only if w is larger than a uniform random deviate between 0 and 1.

vmc.cpp

```
    double alpha;                  // trial function is exp(-alpha*x^2)

    double p(double xTrial, double x) {

        // compute the ratio of rho(xTrial) / rho(x)
        return exp(- 2 * alpha * (xTrial*xTrial - x*x));
    }

    double eLocal(double x) {

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

```
}
```

One Metropolis step

One Metropolis step is implemented as follows:

- Choose one of the N walkers at random
- The walker takes a trial step to a new position that is Gaussian distributed with width δ around the old position. The function gasdev defined in rng.h returns a Gaussian deviate with unit width: multiplying this by a step size δ yields a Gaussian deviate with $\sigma = \delta$. This choice of trial step is suggested by the programs on the author's web site.

vmc.cpp

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

void MetropolisStep() {

    // chose a walker at random
    int n = int(ran2(seed) * N);

    // make a trial move
    double xTrial = x[n] + delta * gasdev(seed);

    // Metropolis test
    if (p(xTrial, x[n]) > ran2(seed)) {
        x[n] = xTrial;
        ++nAccept;
    }

    // accumulate energy and wave function
```

```
        double e = eLocal(x[n]);
        eSum += e;
        eSqdSum += e * e;
        int i = int((x[n] - xMin) / dx);
        if (i >= 0 && i < nPsiSqd)
            psiSqd[i] += 1;
    }
```

As usual, when we have multiple walkers, one Monte Carlo Step is conventionally defined as N Metropolis steps:

vmc.cpp

```
void oneMonteCarloStep() {

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

Steering the computation with the `main` function

vmc.cpp

```
int main() {

    cout << " Variational Monte Carlo for Harmonic Oscillator\n"
         << " -----------------------------------------------\n";
    cout << " Enter number of walkers:  ";
    cin >> N;
    cout << " Enter parameter alpha:  ";
    cin >> alpha;
    cout << " Enter number of Monte Carlo steps:  ";
    int MCSteps;
    cin >> MCSteps;

    initialize();
```

As in all Monte Carlo calculations, some number of steps are taken and discarded to allow the walkers to come to "equilibrium." The thermalization phase is also used to adjust the step size so that the acceptance ratio is approximately 50%. If δ is too small, then too many steps will be accepted; and conversely, if δ is too large, then too many steps will be rejected. Multiplying δ by one half of the acceptance ratio will increase δ if the ratio is larger than 0.5, and decrease δ if the ratio is smaller than 0.5.

vmc.cpp

```
    // perform 20% of MCSteps as thermalization steps
    // and adjust step size so acceptance ratio ~50%
    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) {
            delta *= nAccept / (0.5 * N * adjustInterval);
            nAccept = 0;
        }
    }
    cout << "\n Adjusted Gaussian step size = " << delta << endl;
```

Once the system has thermalized, the accumulators for observables are initialized and the production steps are taken.

vmc.cpp

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

Finally the average value of the energy and the Monte Carlo error estimate are printed, and the probability distribution $\sim \psi_0^2(x)$ is written in the form of a histogram to a file.

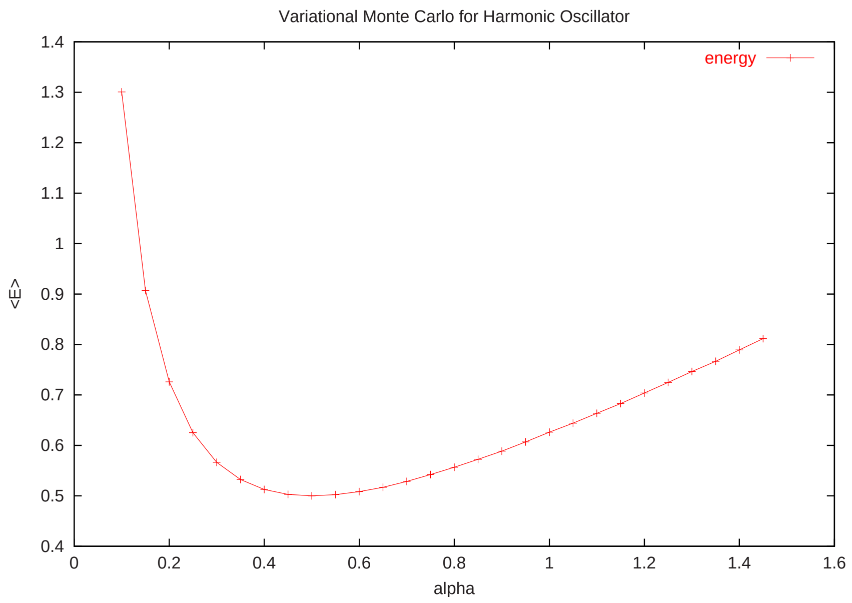
vmc.cpp

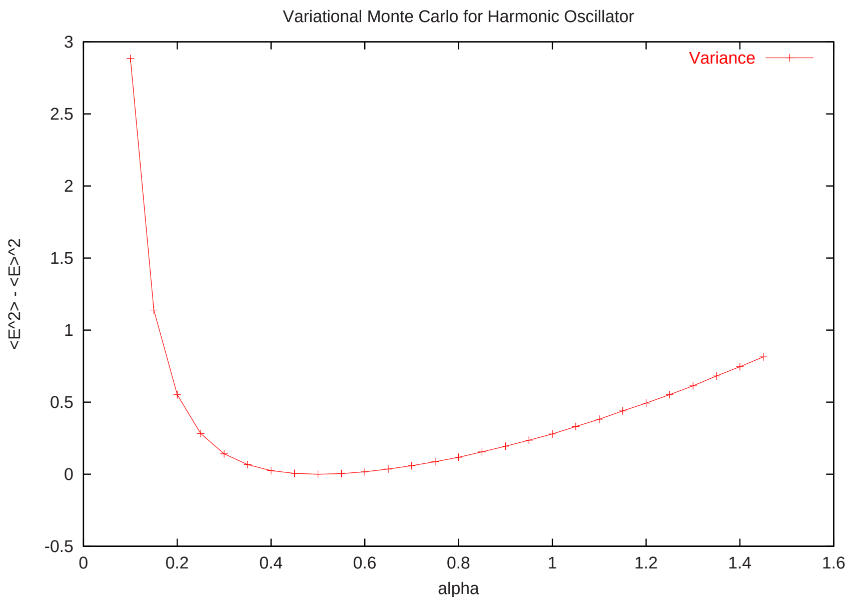
```
// compute and print energy
double eAve = eSum / double(N) / MCSteps;
double eVar = eSqdSum / double(N) / MCSteps - eAve * eAve;
double error = sqrt(eVar) / sqrt(double(N) * MCSteps);
cout << "\n <Energy> = " << eAve << " +/- " << error
     << "\n Variance = " << eVar << endl;

// write wave function squared in file
ofstream file("psiSqd.data");
double psiNorm = 0;
for (int i = 0; i < nPsiSqd; i++)
    psiNorm += psiSqd[i] * dx;
for (int i = 0; i < nPsiSqd; i++) {
    double x = xMin + i * dx;
    file << x << '\t' << psiSqd[i] / psiNorm << '\n';
}
file.close();
```

```
        cout << " Probability density written in file psiSqd.data" << endl;
    }
```

The following plots show results for the average energy $\langle E\rangle$ and its variance $\langle E^2\rangle - \langle E\rangle^2$ as functions of the variational parmeter α. Runs were performed with $N = 300$ walkers and `MCSteps` $=$ 10,000.





As might be expected, the average energy is minimum $\langle E\rangle = 1/2$, and the variance is zero, at $\alpha = 1/2$ which corresponds to the exact solution for the harmonic oscillator ground state.