# The Path Integral Monte Carlo Method

The Path Integral formulation of quantum mechanics was suggested by Dirac Rev. Mod. Phys. 17, 195-199 (1945), and extensively developed by Feynman, Rev. Mod. Phys. 20, 367-387 (1948).

The method relates quantum mechanics of particles that move in in $d$ spatial dimensions to classical statistical mechanics of a corresponding system in $d+1$ spatial dimensions, where the extra dimension can be viewed as an imaginary time for the quantum system.

The Path Integral Monte Carlo (PIMC) method then uses classical Monte Carlo (Topic 2) to compute the properties of the quantum system. The PIMC method can be used to compute time-dependent properties of the quantum system as well as properties of an ensemble of quantum systems in thermal equilibrium at finite temperature.

A good review of the PIMC method with many applications can be found in Ceperley Rev. Mod. Phys. 67, 279-355 (1995).

## Path Integral Formulation in Imaginary Time

Consider a system of $N$ particles with positions $\mathcal{R} = \{\mathbf{r}_i\}$ in $d = 3$ dimensions and Hamiltonian function

$$H = -\hbar^2 \sum_i \frac{1}{2m_i} \frac{\partial^2}{\partial \mathbf{r}_i^2} + V(\mathcal{R}) \ .$$

A formal expression for the time evolution of the wavefunction of the system in the coordinate representation is

$$\psi(\mathcal{R}, t) = \left\langle \mathcal{R} \left| e^{-itH/\hbar} \right| \psi \right\rangle \ .$$

There are two problems with this formal solution that complicate numerical application using Monte Carlo methods:

- The exponential is complex valued and cannot be used as a real positive definite probability distribution.
- $H$ is a quantum mechanical operator in an infinite dimensional Hilbert space, and the kinetic and potential energy operators do not commute.

## Continuing to Imaginary Time

The first problem is solved by continuing to imaginary time $\tau = -it$ as was done in the DMC method. A connection is made to statistical mechanics by defining a partition function

$$Z(\beta) = \int d^{Nd}\mathcal{R} \left\langle \mathcal{R} \left| e^{-\tau H/\hbar} \right| \mathcal{R} \right\rangle = \mathrm{Tr} e^{-\beta H} \ , \qquad \beta = \frac{\tau}{\hbar} \ .$$

This partition function defines the quantum statistical mechanics of the system at temperature $k_{\mathrm{B}}T = 1/\beta = \hbar/\tau$.

## Discretizing the Time Dimension

The second problem is solved by replacing the finite time interval $t$ with a lattice of $M$ small time steps of size $\Delta\tau = \tau/M$. There are $M - 1$ intermediate time steps. At each intermediate time step a complete set of coordinate eigenstates

$$\mathbf{1} = \int d^{Nd}\mathcal{R}_i \ |\mathcal{R}_i\rangle\langle\mathcal{R}_i| \ , \qquad i = 1, \ldots, M-1$$

is inserted to factor the time evolution operator. The partition function is decomposed along the time direction into $M$ segments of duration $\tau$

$$\begin{aligned} Z(\beta) &= \int d^{Nd}\mathcal{R} \left\langle \mathcal{R} \left| e^{-\tau H/\hbar} \right| \mathcal{R} \right\rangle \\ &= \int d\mathcal{R}_0 \int d\mathcal{R}_1 \ldots \int d\mathcal{R}_{M-1} \left\langle \mathcal{R}_0 \left| e^{-\Delta\tau H/\hbar} \right| \mathcal{R}_{M-1} \right\rangle \ldots \\ &\qquad \times \left\langle \mathcal{R}_2 \left| e^{-\Delta\tau H/\hbar} \right| \mathcal{R}_1 \right\rangle \left\langle \mathcal{R}_1 \left| e^{-\Delta\tau H/\hbar} \right| \mathcal{R}_0 \right\rangle \end{aligned}$$

where the integration over $\mathcal{R}_M = \mathcal{R}_0 = \mathcal{R}$ completes the original operator trace.

## Approximation for Short Time Evolution

The Baker-Campbell-Hausdorff theorem states that

$$e^{\mathcal{A}} e^{\mathcal{B}} = e^{\mathcal{C}}$$

if and only if

$$\mathcal{C} = \mathcal{A} + \mathcal{B} + \frac{1}{2}[\mathcal{A}, \mathcal{B}] + \cdots$$

Applying this to the short-time evolution operator with

$$\mathcal{C} = -\Delta\tau H/\hbar \,, \qquad \mathcal{B} = -\Delta\tau V/\hbar \,, \qquad \mathcal{A} = -\Delta\tau K/\hbar \,,$$

where the kinetic energy operator

$$K = -\hbar^2 \sum_i \frac{1}{2m_i} \frac{\partial^2}{\partial \mathbf{r}_i^2} \ ,$$

we see that the commutators in the BCH formula are of order $(\Delta\tau)^2$ and higher. It seems reasonable to neglect these corrections in the limit of very small $\Delta\tau$, and this can be justified rigorously.

With this approximation the evolution in the first time step becomes

$$\left\langle \mathcal{R}_1 \middle| e^{-\Delta\tau H/\hbar} \middle| \mathcal{R}_0 \right\rangle \simeq \left\langle \mathcal{R}_1 \middle| e^{-\Delta\tau K/\hbar} e^{-\Delta\tau V/\hbar} \middle| \mathcal{R}_0 \right\rangle = \left\langle \mathcal{R}_1 \middle| e^{-\Delta\tau K/\hbar} \middle| \mathcal{R}_0 \right\rangle e^{-\Delta\tau V(\mathcal{R}_0)/\hbar} \ .$$

To evaluate the matrix element of the kinetic energy operator, insert two complete sets of momentum eigenstates

$$\begin{aligned} \left\langle \mathcal{R}_1 \middle| e^{-\Delta\tau K/\hbar} \middle| \mathcal{R}_0 \right\rangle &= \int d\mathcal{P}_0 \int d\mathcal{P}_1 \left\langle \mathcal{R}_1 \middle| \mathcal{P}_1 \right\rangle \left\langle \mathcal{P}_1 \middle| e^{-\Delta\tau K/\hbar} \middle| \mathcal{P}_0 \right\rangle \left\langle \mathcal{P}_0 \middle| \mathcal{R}_0 \right\rangle \\ &= \left( \frac{m}{2\pi\hbar\Delta\tau} \right)^{Nd/2} \exp\left[ -\frac{m\Delta\tau}{2\hbar} \sum_i \left( \frac{\mathbf{r}_{i1} - \mathbf{r}_{i0}}{\Delta\tau} \right)^2 \right] \ , \end{aligned}$$

where we have assumed for simplicity that all $N$ particles have the same mass $m_i = m$. Note that this is just a product of $Nd$ free particle diffusion Green functions from the previous lecture

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

one for each of the $Nd$ coordinates.

## Path Integral and Classical Action

The partition function can now be expressed entirely in terms of classical variables and free of any explicit quantum mechanical operators:

$$Z(\beta) \simeq \left(\frac{m}{2\pi\hbar\Delta\tau}\right)^{MNd/2} \int d\mathcal{R}_0 \int d\mathcal{R}_1 \ldots \int d\mathcal{R}_{M-1} \\ \times \exp\left\{-\frac{\Delta\tau}{\hbar}\sum_{j=0}^{M-1}\left[\frac{m}{2}\left(\frac{\mathcal{R}_{j+1}-\mathcal{R}_j}{\Delta\tau}\right)^2 + V(\mathcal{R}_j)\right]\right\} ,$$

The argument of the exponential is just the classical action (continued to imaginary time), the expression is essentially the same as that derived in Section 2.5 of Sakurai, Quantum Mechanics, for one-dimensional motion

$$\langle x_M, t_M | x_1, t_1\rangle = \lim_{M\to\infty}\left(\frac{m}{2\pi\hbar\Delta t}\right)^{(M-1)/2} \int dx_{M-1} \ldots \int dx_2 \Pi_{j=2}^{M} \exp\left[\frac{iS(j,j-1)}{\hbar}\right] \\ = \int_{x_1}^{x_M} \mathcal{D}\left[x(t)\right] \exp\left[\frac{i}{\hbar}\int_{t_1}^{t_M} dt\, L_{\text{classical}}(x,\dot{x})\right] ,$$

where $L_{\text{classical}} = K - V$. This is Feynman's Path Integral formulation of quantum mechanics. The probability amplitude for the quantum particle to travel from point $x_1$ at time $t_1$ to point $x_M$ at time

$t_M$ is got by summing the imaginary exponential of the classical action in units of $\hbar$ for all possible paths between the two points.

The partition function is got by continuing to imaginary time, and integrating also over $x_1 = x_M$, i.e., by summing over all possible periodic paths with period $\tau$.

## Path Integral Monte Carlo Algorithm

The expression

$$\sum_{j=0}^{M-1} \left[ \frac{m}{2} \left( \frac{\mathcal{R}_{j+1} - \mathcal{R}_j}{\Delta\tau} \right)^2 + V(\mathcal{R}_j) \right]$$

in the final formula for the partition function can be viewed as the potential energy function for a system of $MNd$ classical particles with coordinates $\{\mathcal{R}_j\}$, one for each of the $d$ position vector components of each the $N$ quantum particles at each of the $M$ imaginary time steps.

The $Nd$ particles at each time step are coupled by the the potential energy function $V$ of the quantum problem, and the particles at neighboring time steps are coupled by a classical harmonic oscillator forces with force constant $m/\Delta\tau$.

The system can now be simulated using the methods developed in Topic 2. The temperature $k_{\text{B}}T = 1/\beta$ and $d+1$ dimensional volume are fixed, and the forces are conservative. The simulation will give the finite temperature properties of the classical system in the canonical ensemble.

In the limit of zero temperature, i.e. large $\beta = \tau/\hbar$ the classical system will tend to its lowest energy state. The corresponding quantum system will then be in its ground state, and the classical particle positions, averaged over the $M$ time slices, will the distributed according to ground state wavefunction of the $N$ quantum particles.

To obtain a reasonable approximation to the ground state, the temperature is chosen so that $k_{\mathrm{B}}T$ is much smaller than the level spacing of the low-lying energy eigenstates of the quantum system.

## PIMC Code for the Harmonic Oscillator

The simple harmonic oscillator provides a good illustration of the PIMC method as shown in this PIMC Java Applet

The time-sliced path integral (discretized partition function) can be evaluated analytically and the efficiency and error estimates of PIMC simulation can be checked against these exact results. This is done for example in M.F. Herman et al., J. Chem. Phys. 76, 5150-5155 (1982).

The Lagrangian in imaginary time is the energy of an oscillator in the finite temperature canonical ensemble of $M$ "atoms"

$$-L = E = K + V = \frac{m}{2}\left(\frac{dx}{d\tau}\right)^2 + \frac{1}{2}m\omega_0^2 x^2 \, .$$

Choose units such that $\hbar = m = \omega_0 = 1$. The energy associated with the ensemble oscillator at time slice $j$ is

$$E(x_j, j\Delta\tau) = \frac{1}{2}\left(\frac{x_{j+1} - x_j}{\Delta\tau}\right)^2 + V(x_j) \, .$$

Herman et al., show that this formula for the energy is unstable when it is used to estimate fluctuations in the average energy. They introduce and alternative formula based on the Virial theorem

$$2\langle K(x)\rangle = \left\langle x\frac{dV}{dx}\right\rangle \, ,$$

which is generally preferred in PIMC simulations and is used in the following code.

pimc.cpp

```
// Path Integral Monte Carlo program for the 1-D harmonic oscillator

#include <cmath>
#include <cstdlib>
#include <fstream>
#include <iostream>
using namespace std;

#include "gsl.hpp"

double V(double x)           // potential energy function
{
    // use units such that m = 1 and omega_0 = 1
    return 0.5 * pow(x, 2.0);
}

double dVdx(double x)        // derivative dV(x)/dx used in virial theorem
{
    return x;
}

double tau;                  // imaginary time period
```

```
int M;                       // number of time slices
double Delta_tau;            // imaginary time step
vector<double> x;            // displacements from equilibrium of M "atoms"

int n_bins;                  // number of bins for psi histogram
double x_min;                // bottom of first bin
double x_max;                // top of last bin
double dx;                   // bin width
vector<double> P;            // histogram for |psi|^2

double delta;                // Metropolis step size in x
int MC_steps;                // number of Monte Carlo steps in simulation

void initialize()
{
    Delta_tau = tau / M;
    x.resize(M);
    x_min = -x_max;
    dx = (x_max - x_min) / n_bins;
    P.resize(n_bins);
    cout << " Initializing atom positions using gsl::ran_uniform()" << endl;
    for (int j = 0; j < M; ++j)
        x[j] = (2 * gsl::ran_uniform() - 1) * x_max;
}
```

```
bool Metropolis_step_accepted(double& x_new)
{
    // choose a time slice at random
    int j = int(gsl::ran_uniform() * M);
    // indexes of neighbors periodic in tau
    int j_minus = j - 1, j_plus = j + 1;
    if (j_minus < 0) j_minus = M - 1;
    if (j_plus > M - 1) j_plus = 0;
    // choose a random trial displacement
    double x_trial = x[j] + (2 * gsl::ran_uniform() - 1) * delta;
    // compute change in energy
    double Delta_E = V(x_trial) - V(x[j])
        + 0.5 * pow((x[j_plus] - x_trial) / Delta_tau, 2.0)
        + 0.5 * pow((x_trial - x[j_minus]) / Delta_tau, 2.0)
        - 0.5 * pow((x[j_plus] - x[j]) / Delta_tau, 2.0)
        - 0.5 * pow((x[j] - x[j_minus]) / Delta_tau, 2.0);
    if (Delta_E < 0.0 || exp(- Delta_tau * Delta_E) > gsl::ran_uniform()) {
        x_new = x[j] = x_trial;
        return true;
    } else {
        x_new = x[j];
        return false;
    }
```

```
}

int main()
{
    cout << " Path Integral Monte Carlo for the Harmonic Oscillator\n"
         << " -----------------------------------------------------\n";

    // set simulation parameters
    cout << " Imaginary time period tau = " << (tau = 10.0)
         << "\n Number of time slices M = " << (M = 100)
         << "\n Maximum displacement to bin x_max = " << (x_max = 4.0)
         << "\n Number of histogram bins in x = " << (n_bins = 100)
         << "\n Metropolis step size delta = " << (delta = 1.0)
         << "\n Number of Monte Carlo steps = " << (MC_steps = 100000)
         << endl;

    initialize();
    int therm_steps = MC_steps / 5, acceptances = 0;
    double x_new = 0;
    cout << " Doing " << therm_steps << " thermalization steps ...";
    for (int step = 0; step < therm_steps; ++step)
        for (int j = 0; j < M; ++j)
            if (Metropolis_step_accepted(x_new))
                ++acceptances;
```

```
cout << "\n Percentage of accepted steps = "
     << acceptances / double(M * therm_steps) * 100.0 << endl;

double E_sum = 0, E_sqd_sum = 0;
P.clear();
acceptances = 0;
cout << " Doing " << MC_steps << " production steps ...";
for (int step = 0; step < MC_steps; ++step) {
    for (int j = 0; j < M; ++j) {
        if (Metropolis_step_accepted(x_new))
            ++acceptances;
        // add x_new to histogram bin
        int bin = int((x_new - x_min) / (x_max - x_min) * n_bins);
        if (bin >= 0 && bin < M)
            P[bin] += 1;
        // compute Energy using virial theorem formula and accumulate
        double E = V(x_new) + 0.5 * x_new * dVdx(x_new);
        E_sum += E;
        E_sqd_sum += E * E;
    }
}

// compute averages
double values = MC_steps * M;
```

```
    double E_ave = E_sum / values;
    double E_var = E_sqd_sum / values - E_ave * E_ave;
    cout << "\n <E> = " << E_ave << " +/- " << sqrt(E_var / values)
         << "\n <E^2> - <E>^2 = " << E_var << endl;
    ofstream ofs("pimc.out");
    E_ave = 0;
    for (int bin = 0; bin < n_bins; ++bin) {
        double x = x_min + dx * (bin + 0.5);
        ofs << " " << x << '\t' << P[bin] / values << '\n';
        E_ave += P[bin] / values * (0.5 * x * dVdx(x) + V(x));
    }
    ofs.close();
    cout << " <E> from P(x) = " << E_ave << endl;
    cout << " Probability histogram written to file pimc.out" << endl;

    return 0;
}
```