

Week 6: Importance Sampling and Metropolis-Hastings' algorithm

Morten Hjorth-Jensen Email morten.hjorth-jensen@fys.uio.no^{1,2}

¹Department of Physics and Center for Computing in Science Education, University of Oslo, Oslo, Norway

²Department of Physics and Astronomy and Facility for Rare Ion Beams, Michigan State University, East Lansing, Michigan, USA

February 6-10, 2023

Overview of week 6, February 6-10

Topics.

- Importance Sampling, practical aspects, discussion of code and computational issues
- Fokker-Planck and Langevin equations
- [Video of lecture](#)

Importance sampling, what we are planning this week

Last week we discussed the Metropolis algorithm with and without importance sampling and derived the Metropolis-Hastings algorithm.

We replaced the brute force Metropolis algorithm with a walk in coordinate space biased by the trial wave function. This approach is based on the Fokker-Planck equation and the Langevin equation for generating a trajectory in coordinate space. The link between the Fokker-Planck equation and the Langevin equations and importance sampling is non-trivial and perhaps almost impossible to explain without a full semester course in for example statistical mechanics and transport theory. However, we will try.

Detailed content

The plans for this week are:

1. First we want to give you an overarching view on the equations we need to program in order to perform importance sampling with a simple Python code that demonstrates the essential elements.

2. Secondly, we will go through the computational elements we need in order to program the local energy, the trial wave functions and the so-called **quantum force** in the most efficient way (computationally).

Teaching Material, videos and textbooks

1. You may find the following [Lecture on the Langevin and Fokker-Planck equations](#) as a useful introduction.
2. The textbook of [Gardiner on Stochastic Methods](#) is a classic and chapters 4 and 5 are the most relevant ones in order to get a better understanding of these equations. The textbook can be downloaded via UiO's subscription to Springer. Alternatively, the [link here](#) gives you an earlier version.
3. Another excellent reference on topics like Brownian motion, Markov chains, the Fokker-Planck equation and the Langevin equation is the text by [Van Kampen](#)

Overview of what needs to be coded

For a diffusion process characterized by a time-dependent probability density $P(x, t)$ in one dimension the Fokker-Planck equation reads (for one particle /walker)

$$\frac{\partial P}{\partial t} = D \frac{\partial}{\partial x} \left(\frac{\partial}{\partial x} - F \right) P(x, t),$$

where F is a drift term and D is the diffusion coefficient.

Importance sampling

The new positions in coordinate space are given as the solutions of the Langevin equation using Euler's method, namely, we go from the Langevin equation

$$\frac{\partial x(t)}{\partial t} = DF(x(t)) + \eta,$$

with η a random variable, yielding a new position

$$y = x + DF(x)\Delta t + \xi\sqrt{\Delta t},$$

where ξ is gaussian random variable and Δt is a chosen time step. The quantity D is, in atomic units, equal to 1/2 and comes from the factor 1/2 in the kinetic energy operator. Note that Δt is to be viewed as a parameter. Values of $\Delta t \in [0.001, 0.01]$ yield in general rather stable values of the ground state energy.

Importance sampling

The process of isotropic diffusion characterized by a time-dependent probability density $P(\mathbf{x}, t)$ obeys (as an approximation) the so-called Fokker-Planck equation

$$\frac{\partial P}{\partial t} = \sum_i D \frac{\partial}{\partial \mathbf{x}_i} \left(\frac{\partial}{\partial \mathbf{x}_i} - \mathbf{F}_i \right) P(\mathbf{x}, t),$$

where \mathbf{F}_i is the i^{th} component of the drift term (drift velocity) caused by an external potential, and D is the diffusion coefficient. The convergence to a stationary probability density can be obtained by setting the left hand side to zero. The resulting equation will be satisfied if and only if all the terms of the sum are equal zero,

$$\frac{\partial^2 P}{\partial \mathbf{x}_i^2} = P \frac{\partial}{\partial \mathbf{x}_i} \mathbf{F}_i + \mathbf{F}_i \frac{\partial}{\partial \mathbf{x}_i} P.$$

Importance sampling

The drift vector should be of the form $\mathbf{F} = g(\mathbf{x}) \frac{\partial P}{\partial \mathbf{x}}$. Then,

$$\frac{\partial^2 P}{\partial \mathbf{x}_i^2} = P \frac{\partial g}{\partial P} \left(\frac{\partial P}{\partial \mathbf{x}_i} \right)^2 + P g \frac{\partial^2 P}{\partial \mathbf{x}_i^2} + g \left(\frac{\partial P}{\partial \mathbf{x}_i} \right)^2.$$

The condition of stationary density means that the left hand side equals zero. In other words, the terms containing first and second derivatives have to cancel each other. It is possible only if $g = \frac{1}{P}$, which yields

$$\mathbf{F} = 2 \frac{1}{\Psi_T} \nabla \Psi_T,$$

which is known as the so-called *quantum force*. This term is responsible for pushing the walker towards regions of configuration space where the trial wave function is large, increasing the efficiency of the simulation in contrast to the Metropolis algorithm where the walker has the same probability of moving in every direction.

Importance sampling

The Fokker-Planck equation yields a (the solution to the equation) transition probability given by the Green's function

$$G(y, x, \Delta t) = \frac{1}{(4\pi D \Delta t)^{3N/2}} \exp \left(-(y - x - D \Delta t F(x))^2 / 4 D \Delta t \right)$$

which in turn means that our brute force Metropolis algorithm

$$A(y, x) = \min(1, q(y, x)),$$

with $q(y, x) = |\Psi_T(y)|^2 / |\Psi_T(x)|^2$ is now replaced by the [Metropolis-Hastings algorithm](#) as well as [Hasting's article](#),

$$q(y, x) = \frac{G(x, y, \Delta t) |\Psi_T(y)|^2}{G(y, x, \Delta t) |\Psi_T(x)|^2}$$

Code example for the interacting case with importance sampling

We are now ready to implement importance sampling. This is done here for the two-electron case with the Coulomb interaction, as in the previous example. We have two variational parameters α and β . After the set up of files

```
# Common imports
import os

# Where to save the figures and data files
PROJECT_ROOT_DIR = "Results"
FIGURE_ID = "Results/FigureFiles"
DATA_ID = "Results/VMCQdotImportance"

if not os.path.exists(PROJECT_ROOT_DIR):
    os.mkdir(PROJECT_ROOT_DIR)

if not os.path.exists(FIGURE_ID):
    os.makedirs(FIGURE_ID)

if not os.path.exists(DATA_ID):
    os.makedirs(DATA_ID)

def image_path(fig_id):
    return os.path.join(FIGURE_ID, fig_id)

def data_path(dat_id):
    return os.path.join(DATA_ID, dat_id)

def save_fig(fig_id):
    plt.savefig(image_path(fig_id) + ".png", format='png')

outfile = open(data_path("VMCQdotImportance.dat"), 'w')
```

we move on to the set up of the trial wave function, the analytical expression for the local energy and the analytical expression for the quantum force.

```
# 2-electron VMC code for 2dim quantum dot with importance sampling
# Using gaussian rng for new positions and Metropolis- Hastings
# No energy minimization
from math import exp, sqrt
from random import random, seed, normalvariate
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
from matplotlib.ticker import LinearLocator, FormatStrFormatter
import sys
from numba import jit, njit

# Trial wave function for the 2-electron quantum dot in two dims
def WaveFunction(r, alpha, beta):
    r1 = r[0,0]**2 + r[0,1]**2
    r2 = r[1,0]**2 + r[1,1]**2
    r12 = sqrt((r[0,0]-r[1,0])**2 + (r[0,1]-r[1,1])**2)
    deno = r12/(1+beta*r12)
```

```

        return exp(-0.5*alpha*(r1+r2)+deno)

# Local energy for the 2-electron quantum dot in two dims, using analytical local energy
def LocalEnergy(r,alpha,beta):

    r1 = (r[0,0]**2 + r[0,1]**2)
    r2 = (r[1,0]**2 + r[1,1]**2)
    r12 = sqrt((r[0,0]-r[1,0])**2 + (r[0,1]-r[1,1])**2)
    deno = 1.0/(1+beta*r12)
    deno2 = deno*deno
    return 0.5*(1-alpha*alpha)*(r1 + r2) + 2.0*alpha + 1.0/r12+deno2*(alpha*r12-deno2+2*beta*deno-1)

# Setting up the quantum force for the two-electron quantum dot, recall that it is a vector
def QuantumForce(r,alpha,beta):

    qforce = np.zeros((NumberParticles,Dimension), np.double)
    r12 = sqrt((r[0,0]-r[1,0])**2 + (r[0,1]-r[1,1])**2)
    deno = 1.0/(1+beta*r12)
    qforce[0,:] = -2*r[0,:]*alpha*(r[0,:]-r[1,:])*deno*deno/r12
    qforce[1,:] = -2*r[1,:]*alpha*(r[1,:]-r[0,:])*deno*deno/r12
    return qforce

```

The Monte Carlo sampling includes now the Metropolis-Hastings algorithm, with the additional complication of having to evaluate the **quantum force** and the Green's function which is the solution of the Fokker-Planck equation.

```

# The Monte Carlo sampling with the Metropolis algo
# jit decorator tells Numba to compile this function.
# The argument types will be inferred by Numba when function is called.
@jit()
def MonteCarloSampling():

    NumberMCcycles= 100000
    # Parameters in the Fokker-Planck simulation of the quantum force
    D = 0.5
    TimeStep = 0.05
    # positions
    PositionOld = np.zeros((NumberParticles,Dimension), np.double)
    PositionNew = np.zeros((NumberParticles,Dimension), np.double)
    # Quantum force
    QuantumForceOld = np.zeros((NumberParticles,Dimension), np.double)
    QuantumForceNew = np.zeros((NumberParticles,Dimension), np.double)

    # seed for rng generator
    seed()
    # start variational parameter loops, two parameters here
    alpha = 0.9
    for ia in range(MaxVariations):
        alpha += .025
        AlphaValues[ia] = alpha
        beta = 0.2
        for jb in range(MaxVariations):
            beta += .01
            BetaValues[jb] = beta
            energy = energy2 = 0.0
            DeltaE = 0.0
            #Initial position
            for i in range(NumberParticles):
                for j in range(Dimension):

```

```

        PositionOld[i,j] = normalvariate(0.0,1.0)*sqrt(TimeStep)
wfold = WaveFunction(PositionOld,alpha,beta)
QuantumForceOld = QuantumForce(PositionOld,alpha, beta)

#Loop over MC MCcycles
for MCcycle in range(NumberMCcycles):
    #Trial position moving one particle at the time
    for i in range(NumberParticles):
        for j in range(Dimension):
            PositionNew[i,j] = PositionOld[i,j]+normalvariate(0.0,1.0)*sqrt(TimeStep)*\
                QuantumForceOld[i,j]*TimeStep*D
            wfnew = WaveFunction(PositionNew,alpha,beta)
            QuantumForceNew = QuantumForce(PositionNew,alpha, beta)
            GreensFunction = 0.0
            for j in range(Dimension):
                GreensFunction += 0.5*(QuantumForceOld[i,j]+QuantumForceNew[i,j])*\
                    (D*TimeStep*0.5*(QuantumForceOld[i,j]-QuantumForceNew[i,j])-\
                    PositionNew[i,j]+PositionOld[i,j])

            GreensFunction = exp(GreensFunction)
            ProbabilityRatio = GreensFunction*wfnew**2/wfold**2
            #Metropolis-Hastings test to see whether we accept the move
            if random() <= ProbabilityRatio:
                for j in range(Dimension):
                    PositionOld[i,j] = PositionNew[i,j]
                    QuantumForceOld[i,j] = QuantumForceNew[i,j]
                wfold = wfnew
            DeltaE = LocalEnergy(PositionOld,alpha,beta)
            energy += DeltaE
            energy2 += DeltaE**2
        # We calculate mean, variance and error (no blocking applied)
        energy /= NumberMCcycles
        energy2 /= NumberMCcycles
        variance = energy2 - energy**2
        error = sqrt(variance/NumberMCcycles)
        Energies[ia,jb] = energy
        outfile.write('%f %f %f %f %f\n' %(alpha,beta,energy,variance,error))
return Energies, AlphaValues, BetaValues

```

The main part here contains the setup of the variational parameters, the energies and the variance.

```

#Here starts the main program with variable declarations
NumberParticles = 2
Dimension = 2
MaxVariations = 10
Energies = np.zeros((MaxVariations,MaxVariations))
AlphaValues = np.zeros(MaxVariations)
BetaValues = np.zeros(MaxVariations)
(Energies, AlphaValues, BetaValues) = MonteCarloSampling()
outfile.close()
# Prepare for plots
fig = plt.figure()
ax = fig.gca(projection='3d')
# Plot the surface.
X, Y = np.meshgrid(AlphaValues, BetaValues)
surf = ax.plot_surface(X, Y, Energies,cmap=cm.coolwarm,linewidth=0, antialiased=False)
# Customize the z axis.
zmin = np.matrix(Energies).min()

```

```

zmax = np.matrix(Energies).max()
ax.set_zlim(zmin, zmax)
ax.set_xlabel(r'\alpha$')
ax.set_ylabel(r'\beta$')
ax.set_zlabel(r'\langle E \rangle$')
ax.zaxis.set_major_locator(LinearLocator(10))
ax.zaxis.set_major_formatter(FormatStrFormatter('%.02f'))
# Add a color bar which maps values to colors.
fig.colorbar(surf, shrink=0.5, aspect=5)
save_fig("QdotImportance")
plt.show()

```

Importance sampling, program elements

The general derivative formula of the Jastrow factor is (the subscript C stands for Correlation)

$$\frac{1}{\Psi_C} \frac{\partial \Psi_C}{\partial x_k} = \sum_{i=1}^{k-1} \frac{\partial g_{ik}}{\partial x_k} + \sum_{i=k+1}^N \frac{\partial g_{ki}}{\partial x_k}$$

However, with our trial state function written in a way which can be reused later as

$$\Psi_C = \prod_{i < j} g(r_{ij}) = \exp \left\{ \sum_{i < j} f(r_{ij}) \right\},$$

the gradient needed for the quantum force and local energy is easy to compute. The function $f(r_{ij})$ will depend on the system under study. In the equations below we will keep this general form.

Importance sampling, program elements

In the Metropolis/Hasting algorithm, the *acceptance ratio* determines the probability for a particle to be accepted at a new position. The ratio of the trial wave functions evaluated at the new and current positions is given by (OB for the onebody part)

$$R \equiv \frac{\Psi_T^{new}}{\Psi_T^{old}} = \frac{\Psi_{OB}^{new} \Psi_C^{new}}{\Psi_{OB}^{old} \Psi_C^{old}}$$

Here Ψ_{OB} is our onebody part (Slater determinant or product of boson single-particle states) while Ψ_C is our correlation function, or Jastrow factor. We need to optimize the $\nabla \Psi_T / \Psi_T$ ratio and the second derivative as well, that is the $\nabla^2 \Psi_T / \Psi_T$ ratio. The first is needed when we compute the so-called quantum force in importance sampling. The second is needed when we compute the kinetic energy term of the local energy.

$$\frac{\nabla \Psi}{\Psi} = \frac{\nabla(\Psi_{OB} \Psi_C)}{\Psi_{OB} \Psi_C} = \frac{\Psi_C \nabla \Psi_{OB} + \Psi_{OB} \nabla \Psi_C}{\Psi_{OB} \Psi_C} = \frac{\nabla \Psi_{OB}}{\Psi_{OB}} + \frac{\nabla \Psi_C}{\Psi_C}$$

Importance sampling

The expectation value of the kinetic energy expressed in atomic units for electron i is

$$\langle \hat{K}_i \rangle = -\frac{1}{2} \frac{\langle \Psi | \nabla_i^2 | \Psi \rangle}{\langle \Psi | \Psi \rangle},$$

$$\hat{K}_i = -\frac{1}{2} \frac{\nabla_i^2 \Psi}{\Psi}.$$

Importance sampling

The second derivative which enters the definition of the local energy is

$$\frac{\nabla^2 \Psi}{\Psi} = \frac{\nabla^2 \Psi_{OB}}{\Psi_{OB}} + \frac{\nabla^2 \Psi_C}{\Psi_C} + 2 \frac{\nabla \Psi_{OB}}{\Psi_{OB}} \cdot \frac{\nabla \Psi_C}{\Psi_C}$$

We discuss here how to calculate these quantities in an optimal way,

Importance sampling

We have defined the correlated function as

$$\Psi_C = \prod_{i < j} g(r_{ij}) = \prod_{i < j} g(r_{ij}) = \prod_{i=1}^N \prod_{j=i+1}^N g(r_{ij}),$$

with $r_{ij} = |\mathbf{r}_i - \mathbf{r}_j| = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2}$ in three dimensions or $r_{ij} = |\mathbf{r}_i - \mathbf{r}_j| = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$ if we work with two-dimensional systems.

In our particular case we have

$$\Psi_C = \prod_{i < j} g(r_{ij}) = \exp \left\{ \sum_{i < j} f(r_{ij}) \right\}.$$

Importance sampling

The total number of different relative distances r_{ij} is $N(N-1)/2$. In a matrix storage format, the relative distances form a strictly upper triangular matrix

$$\mathbf{r} \equiv \begin{pmatrix} 0 & r_{1,2} & r_{1,3} & \cdots & r_{1,N} \\ \vdots & 0 & r_{2,3} & \cdots & r_{2,N} \\ \vdots & \vdots & 0 & \ddots & \vdots \\ \vdots & \vdots & \vdots & \ddots & r_{N-1,N} \\ 0 & 0 & 0 & \cdots & 0 \end{pmatrix}.$$

This applies to $\mathbf{g} = \mathbf{g}(r_{ij})$ as well.

In our algorithm we will move one particle at the time, say the k th-particle. This sampling will be seen to be particularly efficient when we are going to compute a Slater determinant.

Importance sampling

We have that the ratio between Jastrow factors R_C is given by

$$R_C = \frac{\Psi_C^{\text{new}}}{\Psi_C^{\text{cur}}} = \prod_{i=1}^{k-1} \frac{g_{ik}^{\text{new}}}{g_{ik}^{\text{cur}}} \prod_{i=k+1}^N \frac{g_{ki}^{\text{new}}}{g_{ki}^{\text{cur}}}.$$

For the Pade-Jastrow form

$$R_C = \frac{\Psi_C^{\text{new}}}{\Psi_C^{\text{cur}}} = \frac{\exp U_{\text{new}}}{\exp U_{\text{cur}}} = \exp \Delta U,$$

where

$$\Delta U = \sum_{i=1}^{k-1} (f_{ik}^{\text{new}} - f_{ik}^{\text{cur}}) + \sum_{i=k+1}^N (f_{ki}^{\text{new}} - f_{ki}^{\text{cur}})$$

Importance sampling

One needs to develop a special algorithm that runs only through the elements of the upper triangular matrix \mathbf{g} and have k as an index.

The expression to be derived in the following is of interest when computing the quantum force and the kinetic energy. It has the form

$$\frac{\nabla_i \Psi_C}{\Psi_C} = \frac{1}{\Psi_C} \frac{\partial \Psi_C}{\partial x_i},$$

for all dimensions and with i running over all particles.

Importance sampling

For the first derivative only $N - 1$ terms survive the ratio because the g -terms that are not differentiated cancel with their corresponding ones in the denominator. Then,

$$\frac{1}{\Psi_C} \frac{\partial \Psi_C}{\partial x_k} = \sum_{i=1}^{k-1} \frac{1}{g_{ik}} \frac{\partial g_{ik}}{\partial x_k} + \sum_{i=k+1}^N \frac{1}{g_{ki}} \frac{\partial g_{ki}}{\partial x_k}.$$

An equivalent equation is obtained for the exponential form after replacing g_{ij} by $\exp(f_{ij})$, yielding:

$$\frac{1}{\Psi_C} \frac{\partial \Psi_C}{\partial x_k} = \sum_{i=1}^{k-1} \frac{\partial g_{ik}}{\partial x_k} + \sum_{i=k+1}^N \frac{\partial g_{ki}}{\partial x_k},$$

with both expressions scaling as $\mathcal{O}(N)$.

Importance sampling

Using the identity

$$\frac{\partial}{\partial x_i} g_{ij} = -\frac{\partial}{\partial x_j} g_{ij},$$

we get expressions where all the derivatives acting on the particle are represented by the *second* index of g :

$$\frac{1}{\Psi_C} \frac{\partial \Psi_C}{\partial x_k} = \sum_{i=1}^{k-1} \frac{1}{g_{ik}} \frac{\partial g_{ik}}{\partial x_k} - \sum_{i=k+1}^N \frac{1}{g_{ki}} \frac{\partial g_{ki}}{\partial x_i},$$

and for the exponential case:

$$\frac{1}{\Psi_C} \frac{\partial \Psi_C}{\partial x_k} = \sum_{i=1}^{k-1} \frac{\partial g_{ik}}{\partial x_k} - \sum_{i=k+1}^N \frac{\partial g_{ki}}{\partial x_i}.$$

Importance sampling

For correlation forms depending only on the scalar distances r_{ij} we can use the chain rule. Noting that

$$\frac{\partial g_{ij}}{\partial x_j} = \frac{\partial g_{ij}}{\partial r_{ij}} \frac{\partial r_{ij}}{\partial x_j} = \frac{x_j - x_i}{r_{ij}} \frac{\partial g_{ij}}{\partial r_{ij}},$$

we arrive at

$$\frac{1}{\Psi_C} \frac{\partial \Psi_C}{\partial x_k} = \sum_{i=1}^{k-1} \frac{1}{g_{ik}} \frac{\mathbf{r}_{ik}}{r_{ik}} \frac{\partial g_{ik}}{\partial r_{ik}} - \sum_{i=k+1}^N \frac{1}{g_{ki}} \frac{\mathbf{r}_{ki}}{r_{ki}} \frac{\partial g_{ki}}{\partial r_{ki}}.$$

Importance sampling

Note that for the Pade-Jastrow form we can set $g_{ij} \equiv g(r_{ij}) = e^{f(r_{ij})} = e^{f_{ij}}$ and

$$\frac{\partial g_{ij}}{\partial r_{ij}} = g_{ij} \frac{\partial f_{ij}}{\partial r_{ij}}.$$

Therefore,

$$\frac{1}{\Psi_C} \frac{\partial \Psi_C}{\partial x_k} = \sum_{i=1}^{k-1} \frac{\mathbf{r}_{ik}}{r_{ik}} \frac{\partial f_{ik}}{\partial r_{ik}} - \sum_{i=k+1}^N \frac{\mathbf{r}_{ki}}{r_{ki}} \frac{\partial f_{ki}}{\partial r_{ki}},$$

where

$$\mathbf{r}_{ij} = |\mathbf{r}_j - \mathbf{r}_i| = (x_j - x_i)\mathbf{e}_1 + (y_j - y_i)\mathbf{e}_2 + (z_j - z_i)\mathbf{e}_3$$

is the relative distance.

Importance sampling

The second derivative of the Jastrow factor divided by the Jastrow factor (the way it enters the kinetic energy) is

$$\left[\frac{\nabla^2 \Psi_C}{\Psi_C} \right]_x = 2 \sum_{k=1}^N \sum_{i=1}^{k-1} \frac{\partial^2 g_{ik}}{\partial x_k^2} + \sum_{k=1}^N \left(\sum_{i=1}^{k-1} \frac{\partial g_{ik}}{\partial x_k} - \sum_{i=k+1}^N \frac{\partial g_{ki}}{\partial x_i} \right)^2$$

Importance sampling

But we have a simple form for the function, namely

$$\Psi_C = \prod_{i < j} \exp f(r_{ij}),$$

and it is easy to see that for particle k we have

$$\frac{\nabla_k^2 \Psi_C}{\Psi_C} = \sum_{ij \neq k} \frac{(\mathbf{r}_k - \mathbf{r}_i)(\mathbf{r}_k - \mathbf{r}_j)}{r_{ki} r_{kj}} f'(r_{ki}) f'(r_{kj}) + \sum_{j \neq k} \left(f''(r_{kj}) + \frac{2}{r_{kj}} f'(r_{kj}) \right)$$