Hybrid Montecarlo Algorithm

Dibakar Sigdel

March 30, 2015

Contents

Part I

HMC: Harmonic Oscillator

1 Harmonic Oscillator

We want to solve Harmonic Oscillator problem using HMC. We know for harmonic oscillator

$$H(p,x) = \frac{p^2}{2m} + \frac{1}{2}\kappa x^2; \tag{1}$$

for simplicity we consider

$$\kappa = 1; m = 1, \therefore \omega = \sqrt{\frac{\kappa}{m}} = 1. \tag{2}$$

For Hamiltonian of the form :

$$H(p,x) = K(p) + U(x). \tag{3}$$

We remember the Hamilton's equation of motion:

$$\begin{aligned} \frac{\partial x_i}{\partial t} &= \frac{\partial H}{\partial p_i} = \frac{\partial K}{\partial p_i} \\ \frac{\partial p_i}{\partial t} &= -\frac{\partial H}{\partial x_i} = -\frac{\partial H}{\partial x_i}. \end{aligned} \tag{4}$$

For Harmonic oscillator with Hamiltonian above

$$\frac{\partial x_i}{\partial t} = p; \frac{\partial p_i}{\partial t} = -x. \tag{5}$$

We use these equation to write steps in Leap Frog Algorithm.

1.1 Leap Frog Algorithm

We implement Leap Frog method to update the momentum and position variables sequentially. We start by simulating the momentum dynamics over a small interval of time $\frac{\delta}{2}$, then the position dynamics is simulated over a slightly longer interval in time δ. Then at the end we complete the momentum simulation over another small interval of time $\frac{\delta}{2}$ so that $\mathbf{x}$ and $\mathbf{p}$ now exist at the same point in time.

Specifically, the Leap Frog method is as follows:

1. Take a half step in time to update the momentum variable:

$$p_i(t + \delta/2) = p_i(t) - (\delta/2)\frac{\partial U}{\partial x_i(t)} \tag{6}$$

2. Take a full step in time to update the position variable

$$x_i(t + \delta) = x_i(t) + \delta\frac{\partial K}{\partial p_i(t + \delta/2)} \tag{7}$$

3. Take the remaining half step in time to finish updating the momentum variable

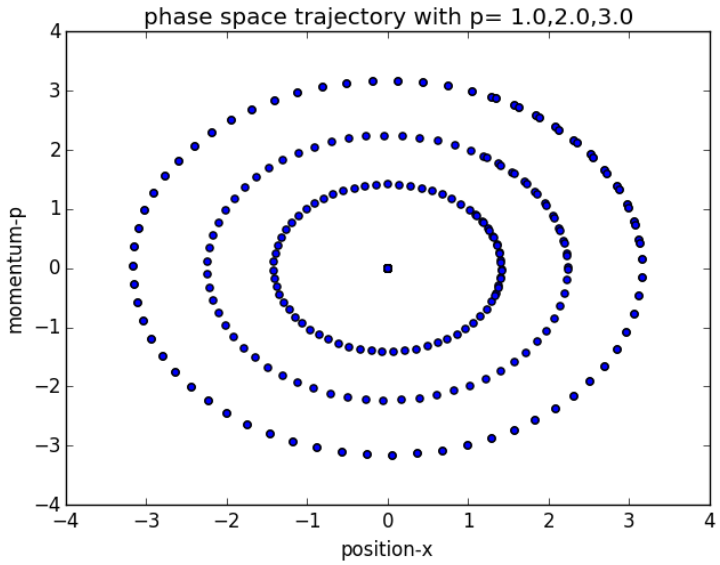
$$p_i(t + \delta) = p_i(t + \delta/2) - (\delta/2)\frac{\partial U}{\partial x_i(t + \delta)} \tag{8}$$

Therefore one iteration the Leap Frog algorithm for simulating Hamiltonian dynamics of Harmonic Oscillator is:

$$\begin{aligned} p(t + \delta/2) &= p(t) - (\delta/2)x(t) \\ x(t + \delta) &= x(t) + (\delta)p(t + \delta/2) \\ p(t + \delta) &= p(t + \delta/2) - (\delta/2)x(t + \delta) \end{aligned} \tag{9}$$

```
#--------------------------------------------------------------------------
# This program plots the points on the locus of particle in phase space by
# Using Leap Frog approximation
#--------------------------------------------------------------------------

 import matplotlib.pyplot  as plt
```

```
x = [0.0  for k  in  range(101)]
p = [[0.0  for l  in  range(2)] for k  in  range(101)]
xx = [0.0  for k  in  range(101)]
pp = [0.0  for l  in  range(101)]

dt = 0.1
x[0] = 1.0
p[0][1] = 2.0

 for k  in  range(1,101):

         p[k][0] = p[k-1][1] - ((dt/2.0)*x[k-1])
         x[k] = x[k-1] + (dt*p[k][0])
         p[k][1] = p[k][0] - ((dt/2.0)*x[k])

         xx[k] = x[k]
         pp[k] = p[k][1]

plt.figure(1)
plt.scatter(xx,pp)
plt.show()
```



1.2 HMC: Algorithmic Steps

By doing HMC we can generate many (x_i, p_i) to represent the trajectory of Harmonic oscillator from where we can calculate average energy,momentum, position etc. Following arethe steps we are going to do:

1. Set time $t = 0$.

2. Generate an initial state$(\boldsymbol{x}_0, \boldsymbol{p}_0)$***.***

3. Repeat until $t = N$***.***

-Set $t = t + 1$***.***

- Sample a new initial momentum variable $\boldsymbol{p}_0$
from the momentum canonical distribution.

- Set $\boldsymbol{x}_0 = \boldsymbol{x}^{(t-1)}$.

-Run Leap Frog algorithm starting at $\boldsymbol{x}_0, \boldsymbol{p}_0$ ***for*** L ***steps and stepsize*** δ ***to obtain proposed states*** $\boldsymbol{x}^*$ ***and*** $\boldsymbol{p}^*$***.***

- Calculate metropolis acceptance probability
$\alpha = \min\left[1, \exp(-U(\mathbf{x}^*) + U(\mathbf{x}_0) - K(\mathbf{p}^*) + K(\mathbf{p}_0))\right].$

- Draw a random number u ***from Unif***$(0, 1)$***.***
If $u \leq \alpha$ ***accept the proposed state position*** $\boldsymbol{x}^*$
and set the next state in the Markov chain
$\boldsymbol{x}^{(t)} = \boldsymbol{x}^*$ ***else set*** $\boldsymbol{x}^{(t)} = \boldsymbol{x}^{(t-1)}$***.***

1.3 Histogram for Momentum Generated by using Gaussian Distribution Function

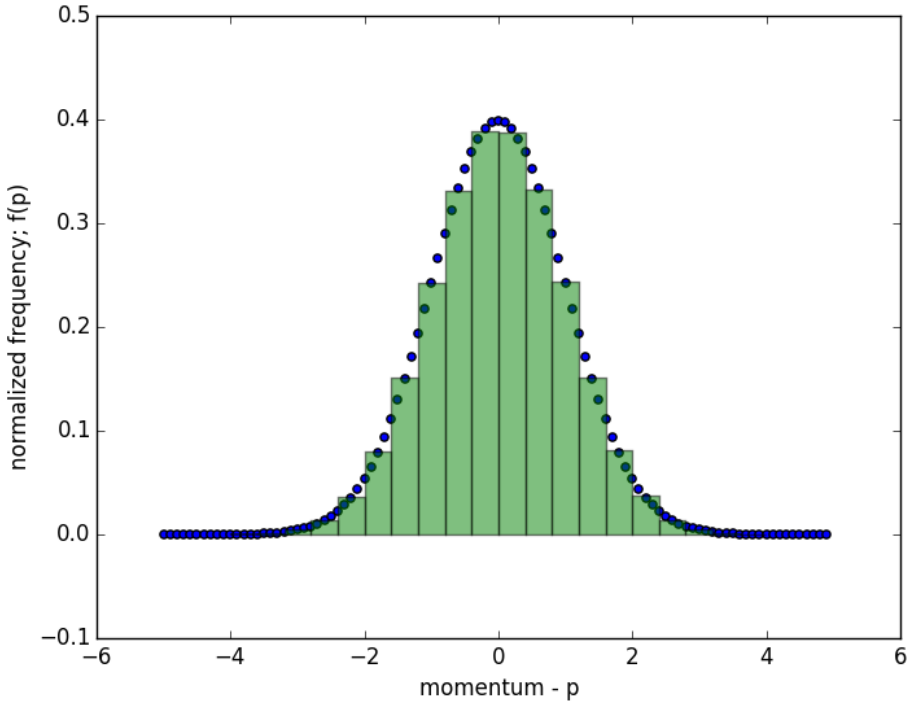
We use following momentum distribution function for generating momentum in the range p to $p + dp$.

$$f(p)dp = \frac{1}{\sqrt{2\pi}} e^{-p^2} dp \tag{10}$$

One can check the distribution of those generated momenta by ploting the histogram for distribution of momenta.

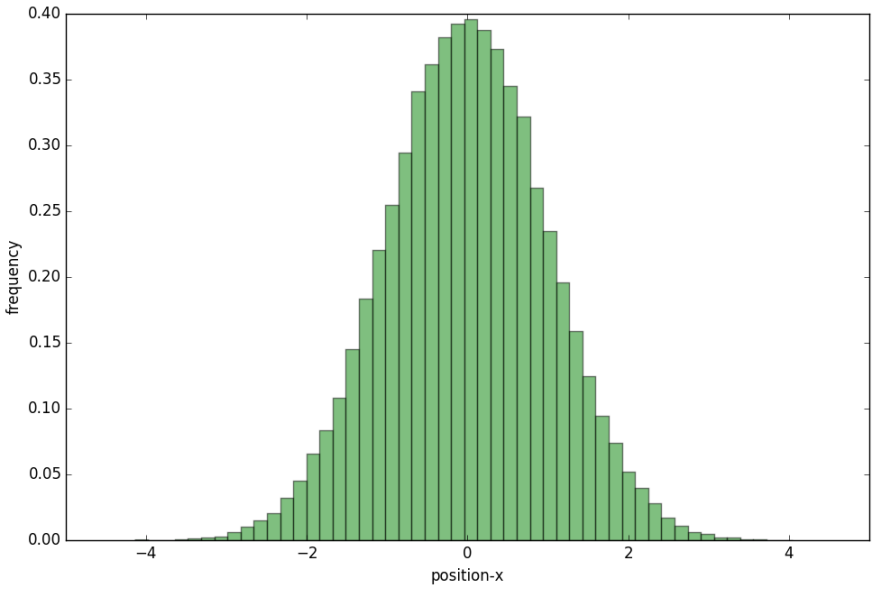
```
#-----------------------------------------------------------------------------------------
# This program plots the histogram for momentum generated by using gaussian function.
#By: Dibakar Sigdel    Date: March-3,2015
#-----------------------------------------------------------------------------------------



 import numpy  as np
 import math  as math
 import matplotlib.pyplot  as plt
 import random  as random

t = np.arange(-5.0,5.0,0.1)

def fx(p):
    fxx = (1/(math.sqrt(2*math.pi)))*math.exp(-(p*p)/2 )
     return fxx

n = len(t)
y = [0.0  for k  in  range(n)]
 for k  in  range(n):
   y[k] = fx(t[k])

N= 1000000
p = [0.0  for k  in  range(N+1)]
 for k  in  range(N+1):
    p[k] = random.gauss(0.0,1.0)
```

```
num_bins =20
plt.figure(11)
plt.xlabel("momentum - p")
plt.ylabel("normalized frequency; f(p)")
plt.scatter(t,y)
plt.hist(p,num_bins, normed= 1.0, facecolor='green', alpha = 0.5)
plt.show()
```



1.4 Simulation of Harmonic Oscillator using HMC

One can plot the histogram for distribution of position of Harmonic oscillator where positions are generated through HMC.




```
#-----------------------------------------------------------------------
# Hamiltonian Dynamics of Harmonic Oscillator(m= 1.0,k = 1.0)
# Date 2015-Jan-25
# By Dibakar Sigdel
#-----------------------------------------------------------------------



 import matplotlib.pyplot  as plt
 import math  as math
 import random  as random


# Gussian distribution function for momentum
def fx(t):
    fxx = (1/(math.sqrt(2*math.pi)))*math.exp(-(t*t)/2 )
     return fxx

#Drawing of momentum from a Gussian distribution of momentum
def drawp():
    r = random.gauss(0.0,1.0)
```

```
    return(r)

#leap_frog algorithm
def leap_frog(N,dt,ix,ip):
       x = [0.0  for k  in  range(N+1)]
       p = [[0.0  for l  in  range(2)] for k  in  range(N+1)]
       x[0] = ix
       p[0][1] = ip
        for k  in  range(1,N+1):
              p[k][0] = p[k-1][1] - ((dt/2.0)*x[k-1])
              x[k] = x[k-1] + (dt*p[k][0])
              p[k][1] = p[k][0] - ((dt/2.0)*x[k])

        return x[N],-p[N][1]


def hamiltonian(x,p):
   H = x*x/2.0 + p*p/2.0
    return H




#************************************************************


N = 100
dt = 0.01
steps =100000
xx = [0.0  for k  in  range(steps)]
pp = [0.0  for k  in  range(steps)]
H = [0.0  for k  in  range(steps)]
x0 = 1.0
chain = 1
 while chain < steps:
       p0 = drawp()
       xt,pt = leap_frog(N,dt,x0,p0)
       frac = (math.exp(-xt*xt/2.0 - pt*pt/2.0))\
            /(math.exp(-x0*x0/2.0 - p0*p0/2.0))

       alpha = min(1,frac)
       rn = random.uniform(0.0,1.0)

        if rn < alpha:
           xx[chain] = xt
           pp[chain] = pt
           H[chain] = hamiltonian(xt,pt)
           x0 = xt
        else:
           xx[chain] = x0
           pp[chain] = p0
           H[chain] = hamiltonian(x0,p0)
       chain = chain+1


t = [1.0*k  for k  in  range (steps)]
num_bins = 50
```

```
plt.figure(12)
#plt.scatter(t,H)
#plt.scatter(t,xx)
plt.hist(xx,num_bins, normed= 1.0, facecolor='green', alpha = 0.5)
plt.xlabel("position-x")
plt.ylabel("frequency")
plt.show()
```

Part II

II: Random Matrices

2 Random Hermitian Matrices generated by using Gaussian distribution

We want to generate traceless Harmitian matrices such that they obey the distribution function

$$p(H)dH = e^{-\frac{1}{2}\mathrm{Tr}[H^2]}dH \tag{11}$$

Let a traceless Hermitian matrix is of the form

$$H = \begin{pmatrix} h_{11} & h_{12} & h_{13} & .... & h_{1n} \\ h_{21} & h_{22} & h_{23} & .... & h_{2n} \\ ... & ... & ... & ... & ... \\ ... & ... & ... & ... & ... \\ h_{n1} & h_{n2} & h_{n3} & ... & h_{NN} \end{pmatrix}; h^*_{i,j} = h_{j,i}; \forall i,j = 1...N \tag{12}$$

with

$$\sum_{i=1}^{N} h_{ii} = 0 \tag{13}$$

Then obviously

$$\mathrm{Tr}(H^2) = \sum_{i=1}^{N} h_{ii}^2 + 2\sum_{j>i}^{N} |h_{ij}|^2 \tag{14}$$

Thus distribution function becomes

$$P(\{h_{ij}\}) = \prod_{i=1}^{N} e^{-\frac{1}{2}|h_{ii}|^2} \sum_{j>i}^{N} e^{-|h_{ij}|^2} \delta\left(\sum_{i=1}^{N} h_{ii}\right) \tag{15}$$

Using standard Gaussian integral

$$\int_{\infty}^{+\infty} e^{-\alpha x^2} dx = \sqrt{\frac{\pi}{\alpha}}, \tag{16}$$

2.1 Gaussian random numbers

Gaussian distribution is given by

$$P(x)dx = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)}{2\sigma^2}} \tag{17}$$

Box-Muller (1958) transformation: It allows us to transform uniformly distributed random variables, to a new set of random variables with a Gaussian (or Normal) distribution with $\mu = 0$ and $\sigma = 1$.

The most basic form of the transformation looks like:

$$\begin{aligned} y_1 &= \sqrt{(-2ln(x_1))}\cos(2\pi x_2) \\ y_2 &= \sqrt{(-2ln(x_1))}\sin(2\pi x_2) \end{aligned} \tag{18}$$

We start with two independent random numbers, x_1 and x_2, which come from a uniform distribution (in the range from 0 to 1). Then apply the above transformations to get two new independent random numbers which have a Gaussian distribution with zero mean and a standard deviation of one.