PHYS 310 Homework 3

Khalil El Achi

Matrix Squaring Process

To establish the matrix squaring of the harmonic to go from x to x' through an intermediate points:

$$\rho(x, x', \beta) = \sum_k \rho(x, x_k, \beta)\rho(x_k, x', \beta)$$

The code will define the density function matrix of all possibilities between two vectors. The harmonic function will be derived as the density function nestled between the two potentials. The code that illustrates the harmonic transition is coded as such

```
import numpy as np
import matplotlib.pyplot as plt
from tqdm import tqdm
import random

def density(x1, x2, beta): #define the density function as a matrix of
    ↪ all the possibilities between x1 vector and x2 vector
        den = np.zeros((len(x1),len(x2)))
        for i in range(len(x1)):
        for j in range(len(x2)):
        den[i,j] = (1 / np.sqrt(2 * np.pi * beta)) * np.exp(-(x1[i] - x2[j
            ↪ ])**2 / (2 * beta))
        return den

def harmonic(x, beta, k): #define the harmonic function as the density
    ↪ function nestled between the exponential of the potentials.
return np.exp(-0.5 * k * x) * density(x, x, beta) * np.exp(-0.5 * k * x)

x = np.linspace(0,1,1000)
beta = 0.01
k = .005
dx = x[1]-x[0]
h = harmonic(x,beta,k)
```

```
    for _ in range(5):
    plt.imshow(h)
    plt.show()
    h = h*h * dx
    plt.imshow(h)
    plt.show()
```

The code outputs the following graphs representing a concentrated density function at the beginning but then getting slimmer after each iteration, i.e. after each matrix squaring:

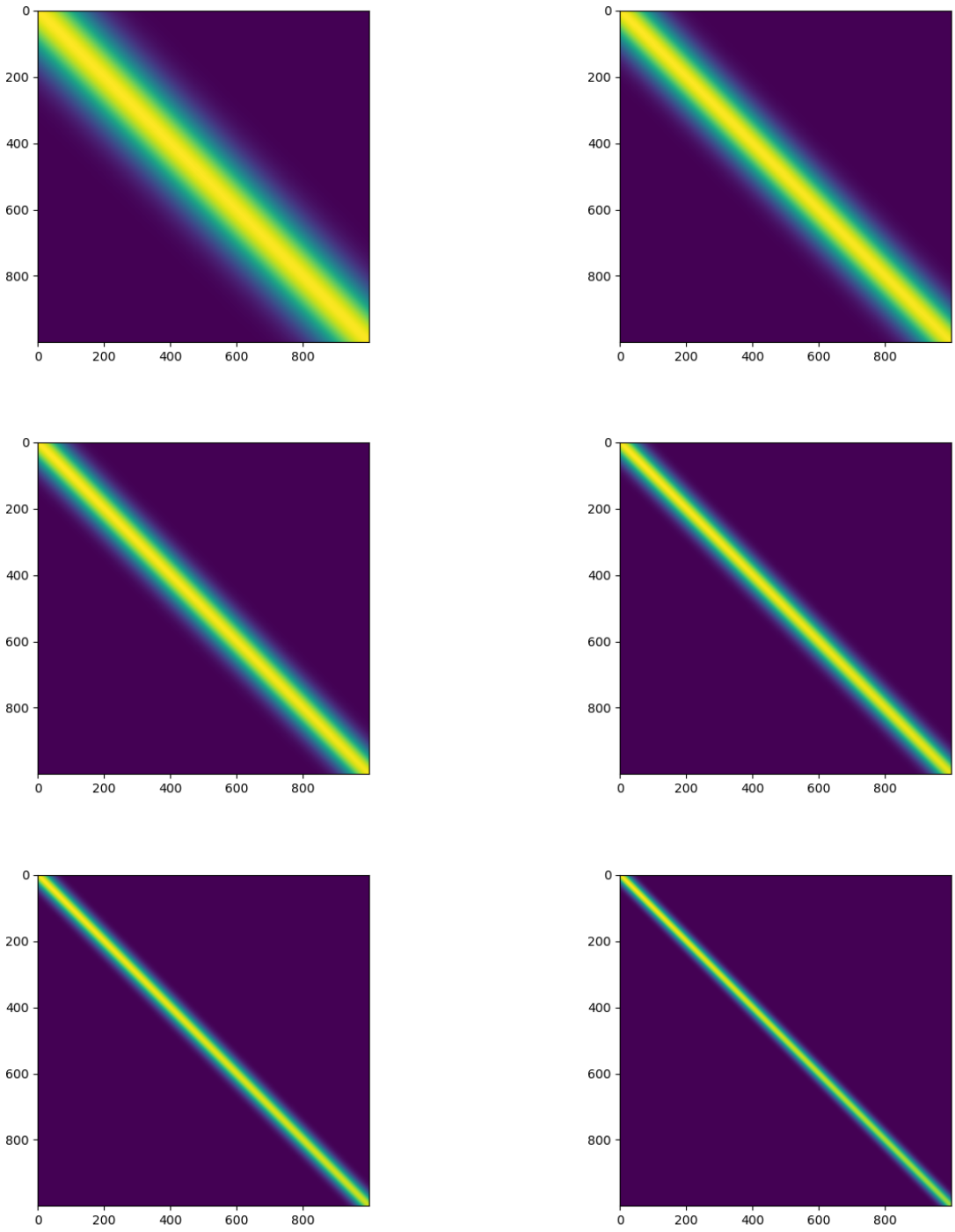

Figure 1: The Gaussian transition of the harmonic function going from x to x'

1 Monte-Carlo for the Naive-Harmonic Path

The simulation starts again by defining the density function $\rho(x1, x2, \beta/N)$ but restricts it to two discrete points outputting a single value rather than a matrix.
The density function will be computed at high temperature of small β by computing it at $\Delta_r = \beta/N$.
A random integer number k is generated from 0 to N-2, where N is the number of x components, or the length of my transition.
Then when k is zero, k_- is defined back as $N-1$ and the probabilities of the predicted transition are calculated as such:

$$\pi_a = \rho(x_1, x_2, \frac{\beta}{N}) e^{\frac{\beta}{N} V(x_a)} \rho(x_2, x_3, \frac{\beta}{N})$$

$$\pi_b = \rho(x_2, x_3, \frac{\beta}{N}) e^{\frac{\beta}{N} V(x_b)} \rho(x_3, x_4, \frac{\beta}{N})$$

If the ratio of the probability satisfies the following condition: $P(a \to b) = min(1, \frac{pi_b}{\pi_a})$ then the predicted transition is accepted.

The code that will implement the simulation goes as such:

```
## Harmonic Path

def rho(x1,x2,beta):
        return (1 / np.sqrt(2 * np.pi * beta)) * np.exp(-(x1 - x2)
            ↪ **2 / (2 * beta))

def randomeWalk(x,beta,d):
        N = len(x)
        delta = beta / N
        k = np.random.randint(0,N-2)
        k_plus = k +1
        k_minus = k - 1
        if k_minus == -1: k_minus = N-1
        x_k = x[k] + random.uniform(-d,d)
        pi_a = rho(x[k_minus],x[k],delta) * rho(x[k], x[k_plus],
            ↪ delta) * np.exp(-0.5 * delta * x[k]**2)
        pi_b = rho(x[k_minus],x_k,delta) * rho(x_k, x[k_plus],
            ↪ delta) * np.exp(-0.5 * delta *  x_k**2)
        if random.uniform(0,1) < pi_b/pi_a:
                x[k] = x_k
        return x

x = np.linspace(0,1,1000)
beta = 4
d = 0.1
for i in tqdm(range(10000000)):
        x = randomeWalk(x,beta,d)
```

```
        plt.hist(x,bins=80, weights= 1/np.full(len(x),len(x)))
        plt.show()
```

The code outputs the following predicted solution of the harmonic function:

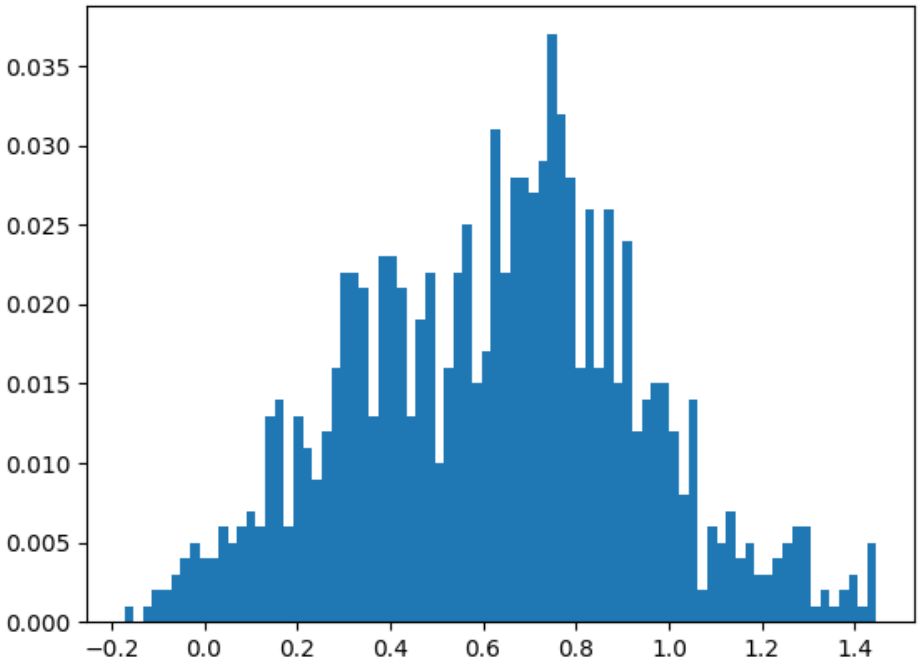

Figure 2: The predicted solution of the Harmonic Function using the Monte-Carlo Simulation