Math Capstone Project

Ho Yi Alexis HO

September 2022

Acknowledgement

In this project, I will extensively use the materials and ideas provided by Prof. Qian Tiezheng. I deeply thank him for his supervision. All errors are mine.

Part I:

Given $m\ddot{x} = -\gamma\dot{x} - Kx$ (1.1) , we can set $p = \dot{x}$ (1.2), which further gives $\dot{p} = \ddot{x}$ (1.3). We then substituted these new notations into (1.1) and obtained $\dot{p} = (-\gamma p - Kx)/m$ (1.4) after some simple algebraic operations.

Given the initial position of the Brownie motion particle, $x(0) = x_0$ and its initial velocity, $p(0) = p_0$, we are interested to approximate the positions of the particle at different values of t. By applying the Taylor expansion, we have,

$$\begin{aligned}
x(h) &= x(0) + \dot{x}(0)h \\
&= x(0) + p(0)h && \textit{(by 1.2)} \\
&= x_0 + p_0 h && \textit{(by IVP condition)} \\
p(h) &= p(0) + \dot{p}(0)h \\
&= p(0) + ((-\gamma p(0) - Kx(0))/m)h && \textit{(by 1.4)} \\
&= p_0 + h(-\gamma p_0 - Kx_0)/m && \textit{(by IVP condition)} \\
x(2h) &= x(h) + \dot{x}(h)h \\
&= x(h) + p(h)h && \textit{(by 1.2)} \\
&= x(h) + h(p_0 + h(-\gamma p_0 - Kx_0)/m) && \textit{(by substitution of } p(h)\textit{)}
\end{aligned}$$

From the above procedure, we moved from time 0 to time t. WLOG, by seting $t_{n+1} = t_n + h$, we can further generalised such a procedure to,

$$\begin{aligned}
x(t_n) &= x(t_{n-1}) + \dot{x}(t_{n-1})h \\
&= x(t_{n-1}) + p(t_{n-1})h
\end{aligned}$$

[*Notice that $x(t_{n-1})$ and $p(t_{n-1})$ were already obtained from the previous iteration.*]

$$\begin{aligned} p(t_n) &= p(t_{n-1}) + \dot{p}(t_{n-1})h \\ &= p(t_{n-1}) + h(-\gamma p(t_{n-1}) - Kx(t_{n-1}))/m \\ x(t_{n+1}) &= x(t_n) + \dot{x}(t_n)h \\ &= x(t_n) + p(t_n)h \end{aligned}$$

We can observe that in the final expression of $x(t_{n+1})$, we have already calculated the $x(t_n)$ and $p(t_n)$ from the previous steps. That makes our idea of approximating $x(t)$ for all t using Python's iterative function practicable.

Forward Euler Method in Python

```
# Reference:
# https://blog.csdn.net/weixin_42376039/article/details/86485817
# https://www.epythonguru.com/2020/07/second-order-differential-equation.html
# https://apmonitor.com/pdc/index.php/Main/SolveDifferentialEquations

import matplotlib.pyplot as plt
# Ouput images with higher resolutions
import matplotlib_inline
matplotlib_inline.backend_inline.set_matplotlib_formats('svg')
from matplotlib.offsetbox import AnchoredText
from scipy.integrate import odeint
import numpy as np

# Initialized the parameters
h = 0.001 # time step used
interval_len = 10 # the length of the interval
# total num of observed time stamps N = interval_len / h
# And I will chop off the decimal places after N in cases N is a non-integer

m = 1 # mass
K = 2 # spring constant
gamma = 10 # frictional coefficient

# x_n := x(t_n)
# dx_n := the first derivative of x(t_(n-1))
x_0 = 0 # initial position of the particle
p_0 = 100 # initial velocity of the particle

# initialize the values for iterations
x_n = x_0
p_n = p_0

N_ls = [0] # The i list, (which are the index of ti, the observed timestamp)
x_n_ls = [x_n] # contains the trajectory of X at different time ti, i.e. X(t)

# N = truncate_to_int(interval_len/h) + 1
for N in range(1, int(interval_len/h)+1):
    x_n = x_n + p_n*h
    p_n = p_n + h*(-gamma*p_n - K*x_n)/m
    N_ls.append(N)
    x_n_ls.append(x_n)
```

```
# change the plot's labels in #
# i (the ith observation) to ti (observed timestamps) #
time_axis = [i * h for i in N_ls]

## Notice: for the illustration purpose ##
## points are plotted with stepsize = 1000 ##
## to contrast its discreteness with the continuous line ##
# list[start:stop:step]
# set the step size for ploting
step = 1000 # (constrast the discrete points with the continous line)
plt.plot(time_axis[0:len(time_axis):step], x_n_ls[0:len(time_axis):step],
         color='red', marker = ".", linestyle='None',
         markersize = 10, label='Numerical')
```

To set up a valid criterion to verify our numerical approximation, we will also derive the analytical solutions for (1.1). Moving terms in (1.1) gives us $m\ddot{x} + \gamma\dot{x} + Kx = 0$ (1.5). Since m, γ and K are all constants, we recognize (1.5) as a homogeneous, linear, second order differential equation. The characteristic equation of (1.5), hence, is $mw^2 + rw + K = 0$. Solving this quadratic equation gives $w = (-\gamma \pm \sqrt{\gamma^2 - 4mK})/2m$ so we will further separate our discussion into three cases.

Case 1: When $r^2 - 4mK > 0$, then the roots of the characteristic equation, w_1 and w_2 are real and distinct. The general solution of (1.5) is, hence, $x(t) = c_1 e^{w_1 t} + c_2 e^{w_2 t}$ (1.6). Given $x(0) = x_0$, we obtained $c_1 + c_2 = x_0$ (1.7). Given $\dot{x}(0) = p(0) = p_0$, we first took first derivative w.r.t t on both sides of (1.6) and attained $\dot{x} = c_1 w_1 e^{w_1 t} + c_2 w_2 e^{w_2 t}$. Further evaluating $\dot{x}$ at point 0 gives, $\dot{x}(0) = c_1 w_1 + c_2 w_2 = p_0$ (1.8). By equating (1.7) and (1.8) and using the Cramer's rule, we obtained,

$$\begin{cases} c_1 + c_2 = x_0 \\ c_1 w_1 + c_2 w_2 = p_0 \end{cases} \Rightarrow \begin{cases} c_1 = (x_0 w_2 - p_0)/(w_2 - w_1) \\ c_2 = (p_0 - w_1 x_0)/(w_2 - w_1) \end{cases}$$

The general solution of (1.5) under IVP condition is, therefore, $x(t) = (x_0 w_2 - p_0)/(w_2 - w_1) e^{w_1 t} + (p_0 - w_1 x_0)/(w_2 - w_1) e^{w_2 t}$, where $w_{1,2} = (-\gamma \pm \sqrt{\gamma^2 - 4mK})/(2m)$.

Case 2: When $\gamma^2 - 4mK = 0$, then the roots of the characteristic equation, w_1, w_2 are real and but repeated (i.e.$w_1 = w_2 = w$). The general solution of (1.5) is, hence, $x(t) = c_1 e^{wt} + c_2 t e^{wt}$ (1.9). Given $x(0) = x_0$, we obtained $c_1 = x_0$ (2.0). Given $\dot{x}(0) = p(0) = p_0$, we first took first derivative w.r.t t on both sides of (1.9) and attained $\dot{x} = c_1 w e^{wt} + c_2(wte^{wt} + e^{wt})$. Further evaluating $\dot{x}$ at point 0 gives, $\dot{x}(0) = c_1 w + c_2 = p_0$ (2.1). By equating (2.0) and (2.1), we obtained $c_2 = p_0 - x_0 w$. The general solution of (1.9), hence, is $x(t) = x_0 e^{wt} + (p_0 - x_0 w) t e^{wt}$, where $w = (-\gamma + \sqrt{\gamma^2 - 4mK})/(2m)$.

Case 3: When $\gamma^2 - 4mK < 0$, then the roots of the characteristic equation, w_1 and w_2 are complex and it is in the form of $w_{1,2} = \lambda \pm \mu i$. The general solution

of (1.5) is, hence, $x(t) = c_1 e^{\lambda t} cos(\mu t) + c_2 e^{\lambda t} sin(\mu t)$(2.2). Given $x(0) = x_0$, we obtained $c_1 = x_0$ (2.3). Given $\dot{x}(0) = p(0) = p_0$, we first took first derivative w.r.t t on both sides of (2.2) and attained
$\dot{x} = c_1 e^{\lambda t}[-\mu \sin(\mu t) + \lambda \cos(\mu t)] + c_2 e^{\lambda t}[\mu \cos(\mu t) + \lambda \sin(\mu t)]$. Further evaluating $\dot{x}$ at point 0 gives, $\dot{x}(0) = c_1\lambda + c_2\mu = p_0$ (2.4). By equating (2.3) and (2.4), we obtained $c_2 = (p_0 - x_0\lambda)/\mu$. The general solution of (1.9), hence, is $x(t) = x_0 e^{\lambda t} cos(\mu t) + (p_0 - x_0\lambda) e^{\lambda t} sin(\mu t)/\mu$, where
$w_{1,2} = (-\gamma \pm \sqrt{\gamma^2 - 4mK})/(2m) = \lambda \pm \mu i$.

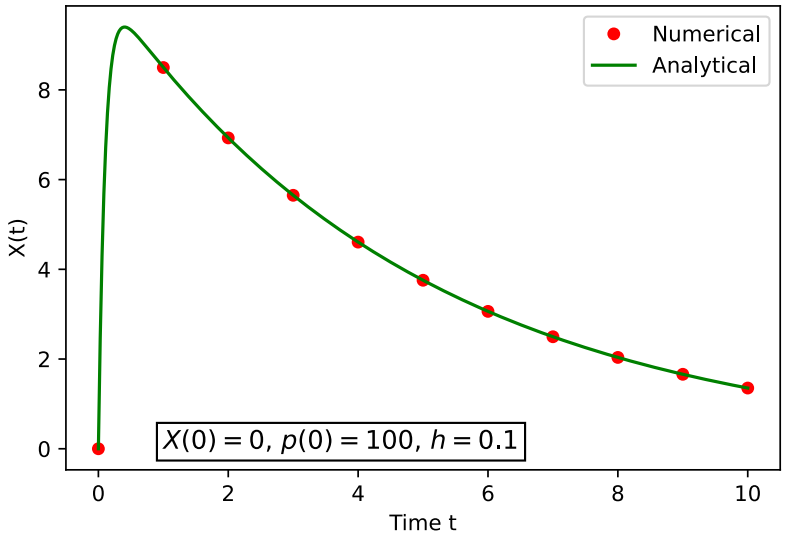

Figure 1: $m = 1$, $K = 2$, $\gamma = 10$ (Overdamped)

An Analytical Approach to solve ODEs in Python

```
## We will plot the analytical ODE sol in a continous line ##
## to represent the ground truth ##
# odeint() can define more than one 1st order differential equations
    # And we can solve even higher order ODEs by using multiple 1st order ODEs.
        def derivatives(initial_values, time_interval):
            x_0 = initial_values[0]
            dxdt_0 = initial_values[1]
            sec_dxdt_0 = -gamma/m*dxdt_0-K/m*x_0
            return(dxdt_0, sec_dxdt_0)
        initial_values = [x_0, p_0]
        time_interval = np.linspace(0, interval_len, N)
        ODE_sol = odeint(derivatives, initial_values, time_interval)
        ODE_sol = ODE_sol[:,0]
```

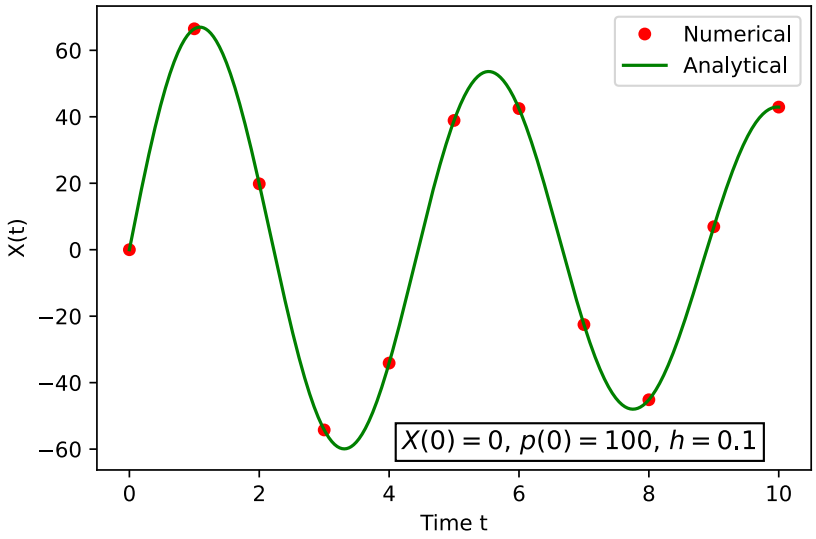

Figure 2: $m = 1$, $K = 2$, $\gamma = 10$ (Underdamped)

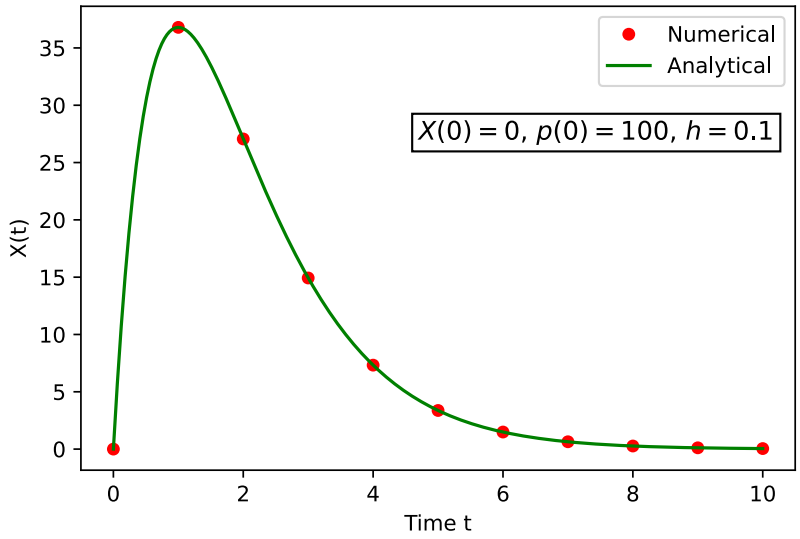

Figure 3: $m = 1$, $K = 1$, $\gamma = 2$ (Marginal: Repeated Roots)

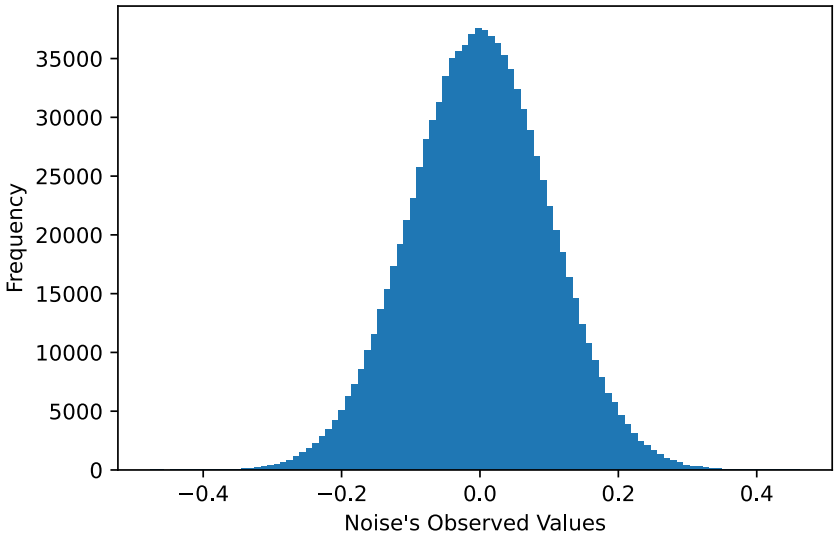

Figure 4: Histogram of Noise following $\mathrm{N}(\mu = 0,\, \sigma^2 = 0.1^2)$

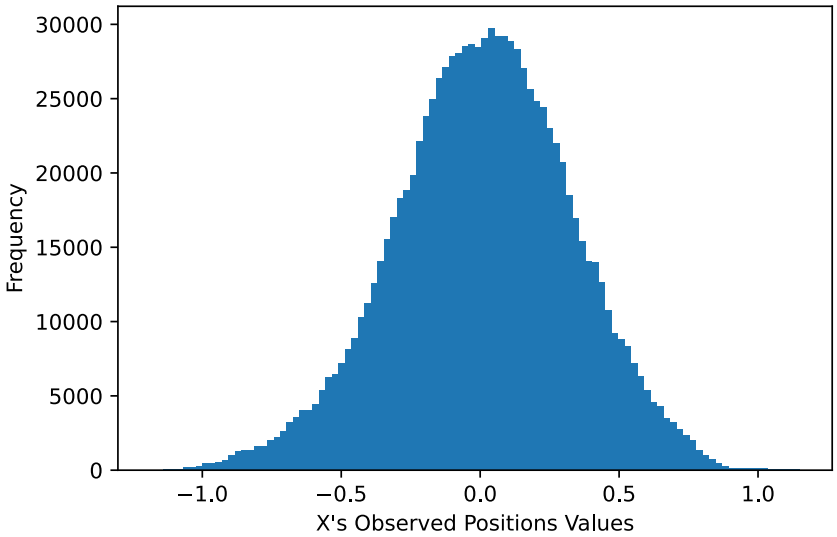

Figure 5: Histogram of X(t) with Noise

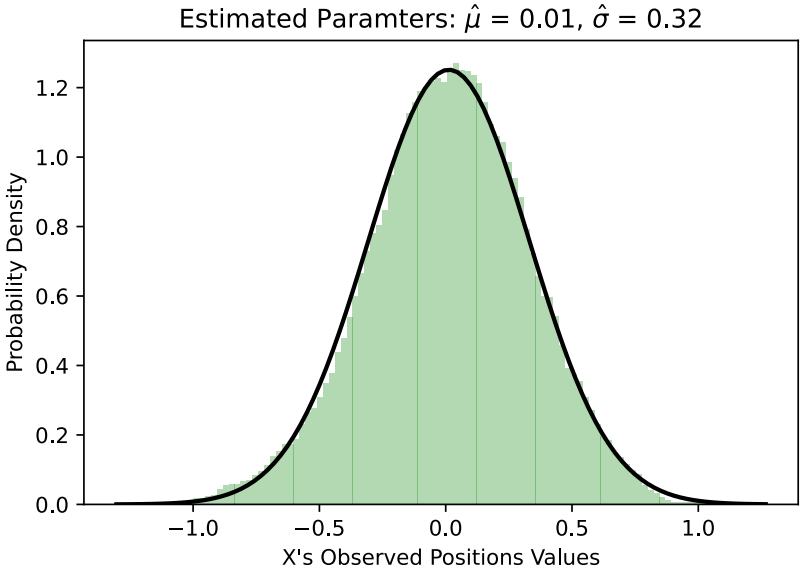

Figure 6: Estimated Distribution of X(t) with Noise

```
plt.plot(time_interval,ODE_sol, linestyle ='-',
        color='green', label='Analytical' )
plt.text(1, 0, '$X(0) = 0$, $p(0) = 100$, $h = 0.1$', fontsize = 12,
         bbox = dict(facecolor='none', edgecolor='black', pad = 3))
plt.xlabel("Time t")
plt.ylabel("X(t)")
plt.legend()
plt.savefig(r'C:\Users\alexi\Desktop\Plots\Overdamped.svg')
```

Case 1.1 We consider the system $m\ddot{x} = -\gamma\dot{x}$ (1.1.1). In such a system, the acceleration $\ddot{x}$ is negatively correlated with the velocity $\dot{x}$ with a coefficient, $-\gamma/m$. The opposite sign between $\ddot{x}$ and $\dot{x}$ indicates the deceleration. Imagine a particle moving on a frictional surface. The inertia, which is proportional to the self-mass, drives the particle moving forward while the frictional force drags the particle backward and attempts to stop the particle's forward motion. That explains why m appears on the fraction's denominator to compensate for the opposite effect brought by γ. From the above force analysis, we recognize that frictional force is the only force applied to this system, as inertia is not a force. Meanwhile, we can also draw the same conclusion by comparing the classical Newton's second law, $F = ma$, with the equation (1.1.1), which gives $F = -\gamma\dot{x}$. By observing the coefficient $-\gamma$, we again concluded that the frictional force is the only contributing force in our system. We, thus, call such a system overdamped to reflect the high energy loss caused by the frictional force.

We let $p = \dot{x}$ to reduce the ODE's higher orders in equation (1.1.1). The equation

becomes

$$\dot{p} = -\frac{\gamma}{m}p$$

with the solution

$$p(t) = p(0)e^{-\frac{\gamma}{m}t}$$

The solution is in the basic form of exponential decay, with exponential decay constant $\lambda = -\gamma/m$. Recall that the characteristic decay time $\tau_p = 1/\lambda = m/\gamma$. When we let $t = \tau_p$, $p(\tau_p) = p(0)e^{-1}$. If we treat $p(0)$ as the initial population, then after the time, τ_p, the initial population is roughly reduced to its original's -0.367 times. The initial population almost goes to extinction; that is where the name of characteristic decay time comes from. In our oscillator's context, the initial population becomes the velocity p. τ_p, hence, is the stop time. The particle stops after the time scale τ_p since no velocity remains. To understand how fast the velocity will be all used up by the particle, we need a benchmark for comparison. Therefore, we need to consider another system illustrated in Case 1.3.

Case 1.3 In Case 1.1, we considered a world with the frictional force alone and obtained the system's stop time τ_p. To realize which benchmark to compare, we must be aware that we were previously in a system filled with obstacles that strive to stop the particle's motions through frictional force. The mirrored extreme case, hence, is an energetic environment replete with the driving force. This time, we only want the presence of $F = -Kx$, where K is the spring constant; we let the spring be the force generator. Enlightened of Newton's second law, we consider $m\ddot{x} = -Kx$ (1.3.1). Since there is no damping force, e.g., friction but only the spring's elastic force. We can imagine that the system continues indefinitely. Such a phenomenon is not a coincidence but a necessary result of the simple harmonic motion, whose standard equation form is described by (1.3.1).

Introducing Noise to X's trajectory in Python

```
# Initialized the parameters
h = 0.001 # time step
interval_len = 1000 # the length of the interval

m = 2 # mass
K = 5 # spring constant
gamma = 10 # frictional coefficient

mu, sigma = 0, 0.1 # mean and standard deviation
noise_sample = np.random.normal(mu, sigma, N)
plt.hist(noise_sample, bins = 100)
plt.xlabel("Noise's Observed Values")
plt.ylabel("Frequency")
plt.show()

x_noise_i = x_0
x_noise_ls = [x_noise_i]
for i in range(1, int(interval_len/h)+1):
    x_noise_i = x_noise_i - K/gamma*x_noise_i*h + noise_sample[i-1]/gamma
```

```
        x_noise_ls.append(x_noise_i)
    plt.hist(x_noise_ls, bins = 100)
    plt.xlabel("X's Observed Positions Values")
    plt.ylabel("Frequency")
    plt.show()
```

After setting up the basic framework, we now introduce the noise to the the current system for a more realistic simulation. The governing equation, hence, turns to be $\gamma\dot{x} = -Kx + \xi$ (3.1), where ξ represents the noise. In Physics, we regard these noises as constant kicks. These kicks guarantee the particle will never stop at a fixed point, even at its equilibrium state. We assume the noise in our model has zero mean, i.e., $< \xi >= 0$, considering noise's mean reversion property in many real-life cases. We expect noises not to be a self-perpetuating force in favor of any specific direction but instead as a random disturbance to the system. In our model, noises oscillate with positive and negative signs and are supposed to be a neutral force in the sense that they overall cancel out each other's effect with a zero mean. We additionally play the universe's role for now and leave further explanations to this article's next section. At this moment, we directly assume a specified autocorrelation function to the noise to ease the discussion. That is

$$R_\xi(t, t') =< \xi(t)\xi(t') >= \Gamma\delta(t - t') \tag{3.2}$$

where $\Gamma = 2\gamma k_B T$ with $k_B T$ being the thermal energy.

The particle now moves consistently due to the introduction of noise. Abandoning the world with a still particle with zero variance, we are now interested in studying the distribution of this particle's trajectory under its equilibrium state. But to achieve our goal, firstly, we need to find the particle's trajectory's expression. Since we can regard $\int_{(i-1)h}^{ih} \dot{x}(t)\,\mathrm{d}t$ as Δdistance = velocity $\times$ Δtime, we write,

$$\begin{aligned} x_i &= x_{i-1} + \int_{(i-1)h}^{ih} \dot{x}(t)\,\mathrm{d}t \\ &= x_{i-1} + \int_{(i-1)h}^{ih} (-\frac{k}{r}x + \frac{\xi(t)}{r})\,\mathrm{d}t \qquad \textit{(by substitutions of (3.1))} \\ &= x_{i-1} - \frac{k}{r}\int_{(i-1)h}^{ih} x(t)\,\mathrm{d}t + \frac{1}{r}\int_{(i-1)h}^{ih} \xi(t)\,\mathrm{d}t \end{aligned}$$

For the second term in the above step, we can again solve it by applying Forward Euler Method. $x(t)$ is a solution of the ODE. It is, hence, differentiable. Otherwise, it will be meaningless to discuss the ODE when $\dot{x}$ does not exist and do so other higher order derivative terms, e.g., $\ddot{x}$ as a consequence. $x(t)$'s differentiability implies $x(t)$ is a continuous function, which satisfies the condition of using the first fundamental theorem of calculus. We let $F = \int_0^x x(t)\,\mathrm{d}t$. The

theorem states that F, an indefinite integral is also an antiderivative of $x(t)$, i.e. $F'(t) = x(t)$. Along with Newton–Leibniz axiom, we use Talyor expansion to further simplify the integral to,

$$\int_{(i-1)h}^{ih} x(t)\,\mathrm{d}t = F(ih) - F((i-1)h) = F'((i-1))h = x(i-1)h$$

Introducing a new notation $\zeta_i = \int_{(i-1)h}^{ih} \xi(t)\,\mathrm{d}t$ and defining ζ_i on the interval i finally gives the core equation we used to perform the iteration task in Python,

$$x_i = x_{i-1} - \frac{k}{r} x_{i-1} h + \frac{\zeta_i}{r}$$

Find the Best Fit Normal Distribution to $X(t)$

```
# Reference:
# https://stackoverflow.com/questions/20011122/fitting-a-normal-distribution-to-1d-data
import numpy as np
from scipy.stats import norm

## Find the best fitted normal distribution to X(t) ##
mu, std = norm.fit(x_noise_ls) # x_noise_ls: contains the trajectory of X(t)

## Plot the histogram of X(t) ##
# density: transfer the frequence on y-axis into probability density
# alpha: constrols the histogram's transparency
plt.hist(x_noise_ls, bins = 100, density=True, alpha=0.3, color='green')

xmin, xmax = plt.xlim() # set the boundary for x_axis
# larger N is, the higher the resolutions of (smoother) #
# the fitted line of the PDF values curve #
selected_pts_on_x_axis = np.linspace(xmin, xmax, 100)
# 100 is the number of equally spaced points, N

corresponding_pdf_values = norm.pdf(selected_pts_on_x_axis , mu, std)
plt.plot(selected_pts_on_x_axis, corresponding_pdf_values, 'k', linewidth=2)
# k:= (col = black)
title = "Estimated Paramters: $\hat{\mu}$ = %.2f, $\hat{\sigma}$ = %.2f" % (mu, std)
# %.2f: rounds up to 2 decimal places

plt.title(title)
plt.xlabel("X's Observed Positions Values")
plt.ylabel("Probability Density")
plt.savefig(r'C:\Users\alexi\Desktop\Plots\Bestfit.svg')
```

Monte Carlo Simulation to Find the Frajectory of $X(t)$

```
import matplotlib.pyplot as plt
import random
import math
kBT = 0.5
K = 5 # spring constant
E = lambda x: (K*x**2/2) # elastic potential energy
x = 0 # initial position
```

```
chain = [x]
for i in range(1, 100000):
    x_c = x + random.uniform(-1, 1) # trial step size
    z = random.uniform(0, 1)
    prob = math.exp(-(E(x_c)-E(x))/kBT)
    if E(x_c) < E(x):
        x = x_c
    elif z <= prob:
        x = x_c
    else:
        x = x
    chain.append(x)
plt.hist(chain, bins = 100)
plt.xlabel("X's Observed Positions Values")
plt.ylabel("Frequency")
plt.savefig(r'C:\Users\alexi\Desktop\Plots\MC_hist.svg')
```

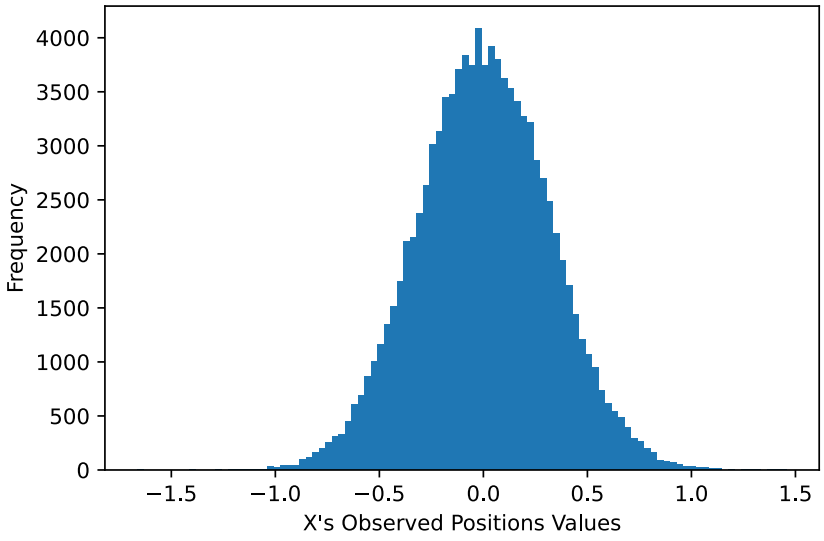

Figure 7: Histogram of $X(t)$ generated from MC Simulation

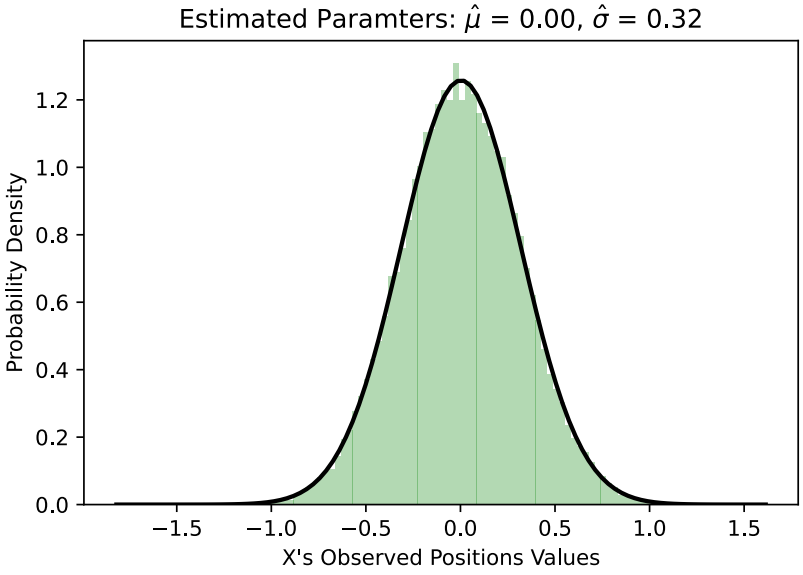

Figure 8: Estimated Distribution of $X(t)$ generated from MC Simulation