Assignment 04

***Due Date**: As per the deadline on Avenue dropbox titled 'A04'. Time required to do this assignment is about 2 hour. The deadline is set to the end of the week to give students with special accommodations adequate time to complete the assignment. No further extensions will be given.*

Part A: Modeling login attempts

You are a computational expert who is working for a network security firm. The security analyst in your group is monitoring the login attempts of a server for a client. She notices that between midnight and 6 am there is a variation in the number of attempts and wants to model the data. With such modeling we can do some resource allocation and address the issues. You are the only one on the team with programming abilities and so the team is counting on you. The login data, ϕ(n,t), is a function of login attempts (n) and time (t). . However, for ease of handling the data is presented to you via a transformation, and therefore as a function of time alone (S(t)). The data is handed to you in a file `A04_sfwr_data_01.txt`. In S(t), a large negative number implies a low login attempt and positive value implies a high attempt. The 120 data points are a function of time as well. For the convenience of plotting and analyzing, the time data, t, is in the interval [0,120]. Further, the security analyst can tell you that this data is periodic.

An equation that you propose to use to model the data is

$$\hat{S}(t) = \left(-A\sin\left(\frac{2\pi}{T}t\right) + \mu\right)\ e^{B\left(\frac{2\pi}{T}t\right)} \qquad (1)$$

Where $\hat{S}(t)$represents the estimated value of the login attempts, T is the time period of the data, μ is some mean, and A and B are constants. It is believed that the three values, μ , A and B are in the interval [0,2]. The objective is to determine these values such that the model $\hat{S}(t)$ matches S(t) as closely as possible. To evaluate the closeness of this fit, you decide to use the mean square error as

$$MSE = \frac{1}{N}\sum_{t=1}^{N}\left(\hat{S}(t) - S(t)\right)^2, \qquad (2)$$

With the above model and metrics in place, you want to do the following:

1. The data from the file `A04_sfwr_data_01.txt` is loaded using the **getInputs**() function and is stored in the array **S_true**.
2. Define the time period, **T**, of the signal based on the information given above. Define the range of the parameters, μ , A and B using the variable names **mu**, **A** and **B**. These definitions should be through the **setParameters**() function.
3. Conduct a brute force search over these parameters' range to find out the optimal combination of **mu**, **A** and **B** that results in a MSE of less than 1.0. This searching is done through the **getFit**() function. The function **evaluateModel**() is used to calculate equation (1) for each combination of these parameters.
   Hint: For the search, use small variations of upto 0.05 for these parameters.

4. The **getFit**() function also keeps track of how the MSE has been reducing over the search in the array **MSE_trend**.
5. The pseudo code of main program is structured as follows:

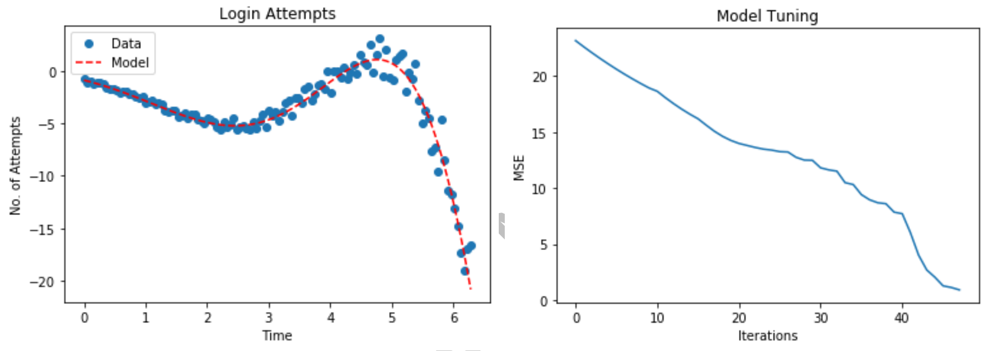
```
getInputs()
setParameters()
getFit()
print the optimal values of mu, A and B
print the mse
plot results
```

The output should be printed only from the main program.

Your program output should be in the following format:

Test Case 1: (Note: your data in the file is different from the one shown here.)

```
The value of A is:  1.00
The value of B is:  0.50
The value of mu is: 0.90
The MSE predicted by our model is:  0.899
```



Part B: Some variation is observed!

While the model seemed to work and your firm was able to minimize the intrusion attempts with some security measures, the hackers decided to evolve their attempts pattern. As a result, the above model no longer worked optimally. Your colleagues rush to you with a new data set (`A04_sfwr_data_03.txt`) for which the model was not very optimal. They want you to minimize the MSE further.

You decide to use a slightly modified model by introducing an additional parameter C. This way you can reuse most of the code as well! This refined model is

$$\hat{S}(t) = \left(-A \sin\left(\frac{2\pi}{T}t\right) + \mu\right) e^{B\left(\frac{\frac{2\pi}{T}t}{C}\right)} \quad (3)$$

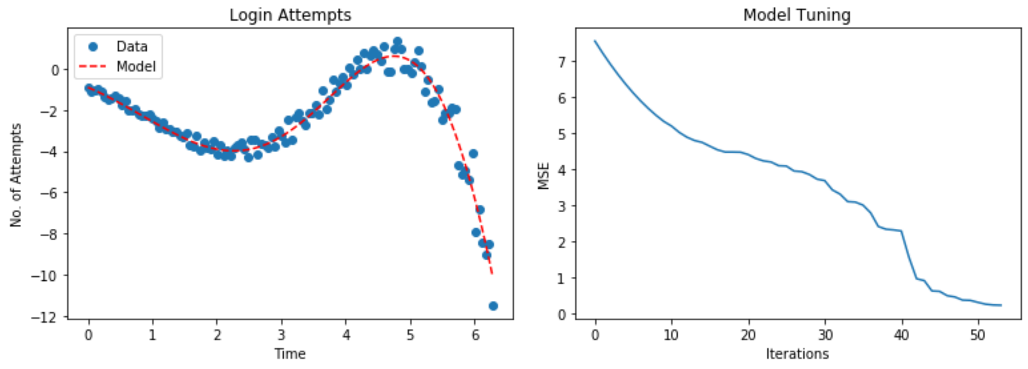
This parameter C is also in the range [0, 2]. With this definition, you want to optimize the model in Equation (3) using the same 5 steps as in part A. Further, you decide to use a refined benchmark of MSE for convergence, i.e., MSE<=0.5. In addition to the other data that was being printed, the pseudocode will now print the optimal value of the parameter C as well.

Once again, the output should be printed only from the main program.

Your program output should be in the following format:

Test Case 2: (Note: your data in the file is different from the one shown here.)

```
The value of A is:  1.00
The value of B is:  0.50
The value of C is:  1.30
The value of mu is:  0.90
The MSE predicted by our model is:  0.225
```



Part C: Towards a generic model

Within days of the revised model, you are told that the pattern of attempts is modified a bit. Your colleague wants to know if there is a way to enhance the model and improve the accuracy further, i.e., even smaller MSE value. Since this client is very critical for the company, you decide to take on the challenge. After much deliberation, you propose to use the following equation:

$$\hat{S}(t) = \left(-A\sin\left(\frac{2\pi}{T}t + \delta\right) + \mu\right)\, e^{B\left(\frac{\frac{2\pi}{T}t+\delta}{C}\right)} \tag{4}$$

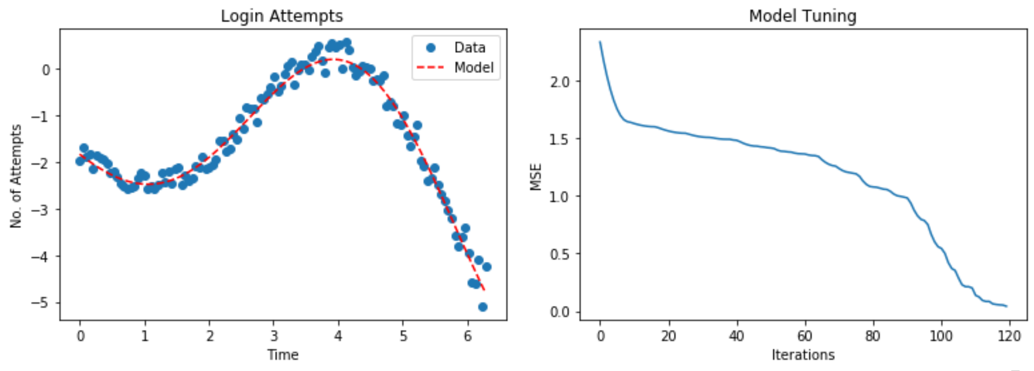
You introduce a new parameter, δ that is in the range [0,1.5]. In your program you call this variable **shift**. The new dataset is given to you in the file `A04_sfwr_data_05.txt`. As before, you decide to undertake the brute force search using almost the same pseudocode to optimize the parameters in Equation (4). This time you set an even stringent benchmark for yourself, MSE <=0.1. You want your code to print the optimal values of all the parameters.

Once again, the output should be printed only from the main program.

Your program output should be in the following format:

Test Case 3: (Note: your data in the file is different from the one shown here.)

```
The value of A is:  1.00
The value of B is:  0.20
The value of C is:  1.30
The value of mu is:  0.90
The value of shift is:  0.80
The MSE predicted by our model is:  0.042
```



Submission Requirements: Submit a single word or PDF file containing the python program for each part along with the corresponding sample screenshots of the output when you run the program, to the dropbox titled **A04** in the *Assignments* section on Avenue. Do not submit a screenshot of the program itself. This will result in a grade of 0. Copy and paste your code into a text editor such as MS word instead. Only a screenshot of the program output is allowed.

Assignment 03

***Due Date**: As per the deadline on Avenue dropbox titled 'A03'. Time required to do this assignment is about 1 hour. The deadline is set to the end of the week to give students with special accommodations adequate time to complete the assignment. No further extensions will be given.*

Part A: Dropping a Ball!

We all drop the ball occasionally (pun intended)☺ Let us study how it bounces back though.

The process: Assume that the ball is dropped from a certain height (h) with an initial speed of u m/s. The ball travels down and as it is acted upon by the gravitational force (g ~ 10m/s^2), it hits the ground with a higher final speed v. This final speed is calculated using the relation v = u+gt. Upon hitting the ground the ball rebounds. Let us assume that it rebounds without any loss of energy, i.e., it rebounds with the same speed v. As the ball moves up, it decelerates at the rate of -g (i.e. g ~ -10m/s^2) and eventually when it get back to the original height of h, it stops. It then starts falling back to the ground, emulating the initial fall. The distance that the ball rises/falls is d = ut + 0.5gt^2. (Note: If the initial dropping velocity is 0 m/s then the ball must rise to the exact same initial height. If the ball it thrown with a certain downward speed, then it rebounds to a greater height than the initial height.)

Now, write a python program that simulates this process by doing the following: Using the **getInput**() function obtain the following values from the user: initial height and initial speed. (Both will be a positive number). Use the **motionSimulator**() function to simulate the falling and rising of the ball. In this you will have to increment time by small steps of 0.04s and determine the precise position of the ball for these timesteps, plotting it as a function of time. You must print the following data onto the screen each time the ball reaches the extremum:

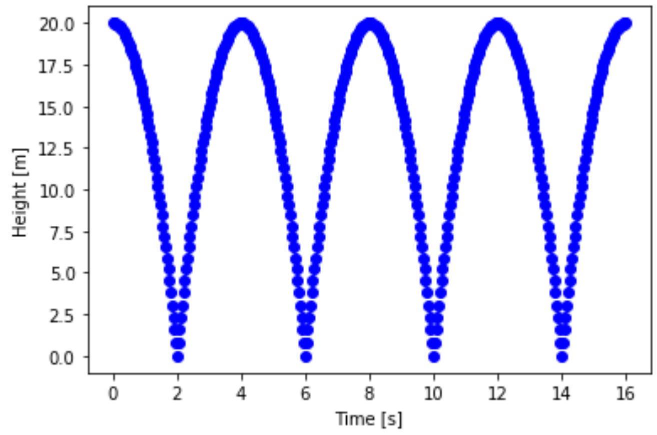
1. Position (Top or Bottom indicated by T and B, respectively).
2. The speed at that position
3. Total number of timesteps to be simulated (use 401 timesteps)
4. The time interval needed to make that motion from the previous extremum.
5. The total time that the ball has been bouncing for since the beginning of the simulation.
6. The cumulative number of 0.04s timesteps that have been simulated.
7. A graph showing the bouncing of the ball through the entire duration
   Hint: You will need the pyplot library to plot graphs. While a detailed introduction to pyplot is in https://matplotlib.org/stable/tutorials/introductory/pyplot.html, a relevant code that you need for this exercise is:

```
import matplotlib.pyplot as plt
plt.plot(time,current_height,'bo')
plt.xlabel('Time [s]')
plt.ylabel('Height [m]')
```

Use floats for all your variables except the number of timesteps. **The output should be printed only from the main program. Your program output should be in the following format:**

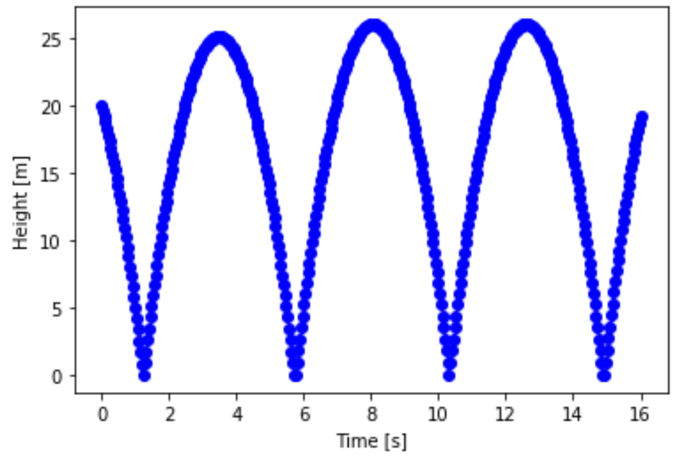
Test Case 1: (h=20, u=0)

```
B: 20.00  2.00  2.00    50
T:  0.00  2.00  4.00   100
B: 20.00  2.00  6.00   150
T:  0.00  2.00  8.00   200
B: 20.00  2.00 10.00   250
T:  0.00  2.00 12.00   300
B: 20.00  2.00 14.00   350
T:  0.00  2.00 16.00   400
```



Test Case 2: (h=20, u=10)

```
B: 22.40  1.24  1.24    31
T:  0.00  2.24  3.48    87
B: 22.80  2.28  5.76   144
T:  0.00  2.28  8.04   201
B: 22.80  2.28 10.32   258
T:  0.00  2.28 12.60   315
B: 23.20  2.32 14.92   373
```



Part B: Energy Loss

There is an unrealistic assumption in part A. The ball will always lose some energy when it hits the ground. As a result it will not be able to get back to the same height. So, let us fix the problem. Copy and paste the code from part A since you will need most of it. Use the **energyLoss**() function that determines the rebound speed (v) using the principle of conservation of energy. For a given bounce, when the ball is at the highest point (y), its potential energy is E_potential = mgy, where m is the mass of the ball (not needed), and g is as in part A. When it hits the ground, this energy converts itself to the kinetic energy E_kinetic = $0.5mv^2$, where v is the rebound speed. Assume that the loss of the potential energy is Ep when the ball hits the ground and calculate the rebound speed (v) in this function. Ep is obtained from the **getInput()** function as a decimal value (example, Ep = 0.1 for 10% loss).

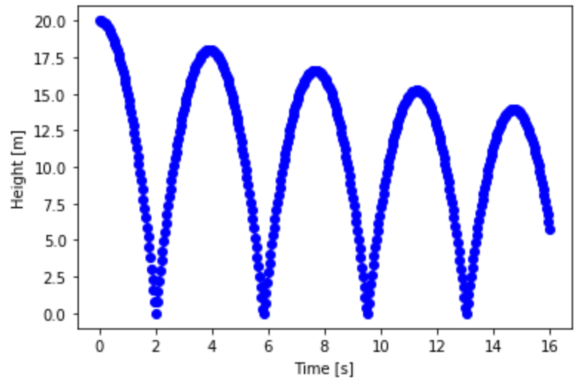
As in part A, using the **getInput**() function obtain the following values from the user: initial height and initial speed. (Both will be a positive number). Use the **motionSimulator**() function to simulate the falling and rising of the ball. In this you will have to increment time by small steps of 0.04s and determine the precise position of the ball for these timesteps, plotting it as a function of time. You must print the following data onto the screen each time the ball reaches the extremum:

1. Position (Top or Bottom indicated by T and B, respectively).
2. The speed at that position
3. Total number of timesteps to be simulated (use 401 timesteps)
4. The time interval needed to make that motion from the previous extremum.
5. The total time that the ball has been bouncing for since the beginning of the simulation.
6. The cumulative number of 0.04s timesteps that have been simulated.
7. A graph showing the bouncing of the ball through the entire duration.

The output should be printed only form the main program. Try different combinations of inputs to understand how the ball bounces. **Your program output should be in the following format:**

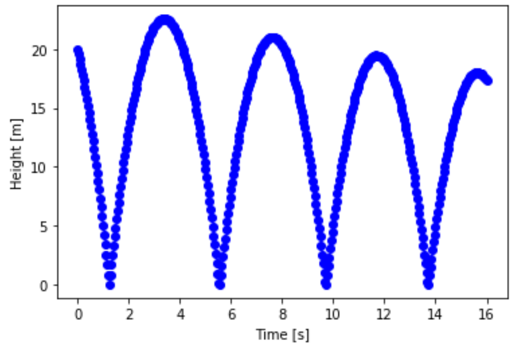
Test Case 1: (h=20, u=0, Ep = 0.1)

```
B: 18.97  2.00  2.00    50
T:  0.00  1.92  3.92    98
B: 18.21  1.92  5.84   146
T:  0.00  1.84  7.68   192
B: 17.46  1.84  9.52   238
T:  0.00  1.76 11.28   282
B: 16.70  1.76 13.04   326
T:  0.00  1.68 14.72   368
```



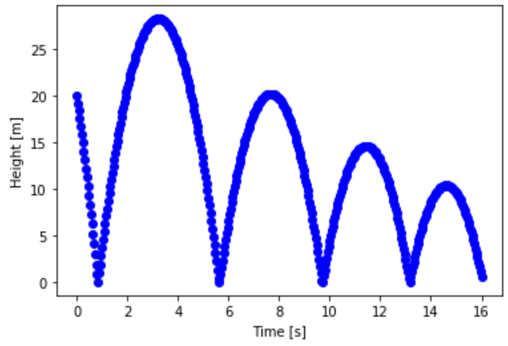
Test Case 2: (h=20, u=10, Ep=0.1)

```
B: 21.25  1.24  1.24    31
T:  0.00  2.16  3.40    85
B: 20.49  2.16  5.56   139
T:  0.00  2.08  7.64   191
B: 19.73  2.08  9.72   243
T:  0.00  2.00 11.72   293
B: 18.97  2.00 13.72   343
T:  0.00  1.92 15.64   391
```



Test case 3: (h=20,u=20,Ep=0.3)

```
B: 23.76  0.84  0.84    21
T:  0.00  2.40  3.24    81
B: 20.08  2.40  5.64   141
T:  0.00  2.04  7.68   192
B: 17.07  2.04  9.72   243
T:  0.00  1.72 11.44   286
B: 14.39  1.72 13.16   329
T:  0.00  1.44 14.60   365
```



Submission Requirements: Submit a single word or PDF file containing the python program for each part along with the corresponding sample screenshots of the output when you run the program, to the dropbox titled **A03** in the *Assignments* section on Avenue. Do not submit a screenshot of the program itself. This will result in a grade of 0. Copy and paste your code into a text editor such as MS word instead. Only a screenshot of the program output (see sample of test case results including the figure) is allowed.

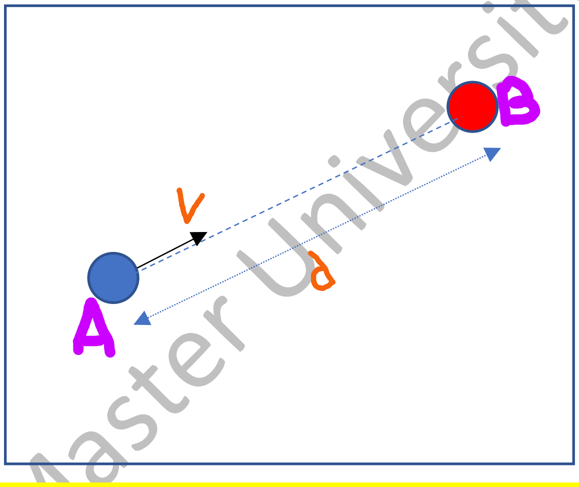
Assignment 2

***Due Date**: As per the deadline on Avenue dropbox titled 'A02'. Time required to do this assignment is about 1 hour. The deadline is set to the end of the week to give students with special accommodations adequate time to complete the assignment. No further extensions will be given.*

Suppose object A is traveling towards object B with a speed *v*. A and B are separated by a distance *d*. If A is decelerating by a value of *a* , we want to determine if it will hit the object B. The user must provide the following values *d, v* and *a*. *d* is in the range [5,10], *a* is in the range [-100,0] and v is in the range [1,10]. The distance travelled by the object A in time *t* (a positive number less than 10 that is also to be received from the user) can be calculated as

$$s = \max(0, vt + 0.5at^2)$$

If *s* is greater than or equal to *d* then the object will collide. Write a python program that does the following: (i) determines if the objects collide for a given set of *d, v, a* and t. (ii) for a given value of *d and v*, and starting with a = -50, determines the critical value of *a* at which A will just touch B.



Hint: To find the critical value of a, keep increasing *a* by a small amount (example, 0.2), and for each value of a scan through a long range of t to conclusively establish if the object A hits B or not. Note: All values are floating point data.

Submission Requirements: Submit a single word or PDF file containing the python program and sample screenshots of the output when you run the program to the dropbox titled **A02** in the *Assignments* section on Avenue. Do not submit a screenshot of the program itself. This will result in a grade of 0. Copy and paste your code into a text editor such as MS word instead. Only a screenshot of the program output is allowed.

Assignment 1

***Due Date**: As per the deadline on Avenue dropbox titled 'A01'. Time required to do this assignment is about 1 hour. The deadline is set to the end of the week to give students with special accommodations adequate time to complete the assignment. No further extensions will be given.*

1. Write a python program for the *Rock Paper Scissors Spock Lizard* game. The rules of the game are:

   Spock beats scissors and rock, but loses to paper and lizard.
   Lizard beats Spock and paper, but loses to rock and scissors.
   Rock beats scissors and lizard, but loses to paper and Spock.
   Paper beats rock and Spock, but loses to scissors and lizard.
   Scissors beats paper and lizard, but loses to rock and Spock.

   The program will have two players and each of them produce a random outcome, i.e., Rock, paper, scissor, spock, or lizard. Based on each player's outcome, determine the winner.

Submission Requirements: Submit a single word or PDF file containing the python program and sample screenshots of the output when you run the program to the dropbox titled **A01** in the *Assignments* section on Avenue. Do not submit a screenshot of the program itself. This will result in a grade of 0. Copy and paste your code into a text editor such as MS word instead. Only a screenshot of the program output is allowed.