Faculty of Science

Course:	CSCI 3010U: Modelling and Simulation
Component:	Assignment #1
Topic:	Basic Simulation and Plotting

Collaboration Policy

These assignments contribute significantly toward your mark. They are designed to prepare you for future exams, and your project. It is in your own best interests to work on this assignment alone. However, if you are stuck on part of the assignment, collaboration is permitted to some degree. The official policy of the instructor is that it is acceptable to discuss general strategies with your fellow students, but when it comes to actually writing code you should do it entirely alone.

Any student who turns in work copied (in whole or in part) from another student will result in both (or all) students being investigated for a violation of academic integrity, which will result in failure of the assignment, a permanent record of the incident on your academic record, and possible further consequences. You'll also find the next examination to be far too difficult, since you've skipped some of my planned learning objectives.

Overview - Part 1

Julie-Ann's friends pressure her into going bungee jumping for the first time. Julie-Ann is a cautious woman, so she isn't going to jump unless she is sure it is safe to do so. She decides that she'd like you to simulate the bungee jump using Python, since you've taken a course in Computer Science, and show her the results. In the first part of the assignment, you'll simulate a free fall in the absence of air friction. In the second part, you'll include calculations for air friction. In the third part, you'll add the spring forces of the bungee cord.

Using Julie-Ann's (wearing her clothes, the harness, and bungee cord) weight of 70kg, simulate the motion of a free fall, without considering the forces of friction or the effect of the bungee. In other words (in the simulation), let's let Julie-Ann fall to her death (for science!). Only gravity will affect Julie-Ann's fall. For simplicity, use positive numbers to represent downward motion, so 'length' will represent how far from the platform Julie-Ann has travelled.

Note: *Let's assume that Julie-Ann's available falling distance is infinite, so that we can see the pattern of her fall more easily.*

Create a Python function (simulateFreeFall), which takes 3 arguments:
- mass: The mass of Julie-Ann (or any falling object)
- deltaT: The length of time (in seconds) of each time interval, for the simulation
- simulationTime: The length of time (in seconds) of the entire simulation

The function will calculate from deltaT and simulationTime how many time steps there should be. For each of these time steps, you will calculate the length (distance from the platform, in metres), velocity (in metres/s), and acceleration (which will be constant, 9.81 metres per second squared). At each time step, record each of the following values in a Python list:
- elapsedTime: deltaT * the number of time steps that have passed
- length: the distance between the object (Julie-Ann) and the platform at that time step
- velocity: the velocity of the object (Julie-Ann) at that time step
- acceleration: the acceleration on the object at that time step

The function will return a tuple containing all four of these lists (times, lengths, velocities, accelerations).

Updating the length:
When updating the length, use the current velocity and the current elapsed time, in the formula:

   Δd = v * Δt

If you use a single time step duration for Δt (deltaT), and the current velocity for v, then Δd will be the change in length, which you'll need to add to the previous length.

Updating the velocity:
When updating velocity, we'll calculate how the current acceleration will affect the value, using the following formula:

   Δv = a * Δt

If you use a single time step duration for Δt (deltaT), and the current acceleration for a, then Δv will be the change in velocity, which you'll need to add to the previous velocity.

Updating the acceleration:
This part of the assignment does not require any modification to acceleration. Julie-Ann will continue to accelerate downwards for the duration of the simulation.

Write some Python code to call your function, collecting the four input lists it returns. Use 70kg for Julie-Ann's mass, deltaT of 0.01 seconds, and simulationTime of 60 seconds. Chart the length values vs.

elapsedTime values, with red squares for markers, using MatPlotLib. Be sure to include appropriate title, x-axis labels, and y-axis labels in your chart.

Sample output:
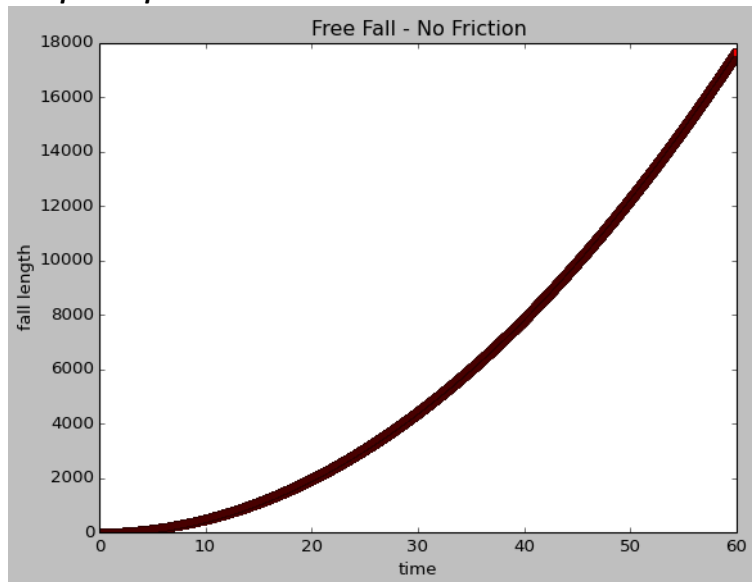
Part 1B

For this part of this assignment, you will simulate another free fall, but this time you will incorporate air friction (also known as drag). Start by making a copy of your function from part 1, and call it simulateFallFriction. Our function will have one new argument:

- surfaceArea: The surface area (in m^2) of the falling person/object

We'll perform this calculation by adding up all forces on our object. In this part, we have two forces:
1. the force due to gravity (we'll call this F_{weight})
2. the force due to air friction (we'll call this $F_{friction}$)

Updating the length and velocity:
Updating the length and the velocity will use the same procedure as before, but now the acceleration is not a constant, since it depends on air friction (which itself is dependent upon velocity).

Updating the acceleration:
We didn't really consider forces in the previous section, so we'll need to consider the force created by gravity, using the following formula:

```
Fweight = m * g
```

A simplified formula for air resistance at sea level is given below:

```
Ffriction = -0.65 * surfaceArea * v * |v|
```
        ($|v|$ - absolute value of v)

For the sake of this simulation, we'll use 0.2m^2 for Julie's surface area.

Now, the total force (F_{total} = F_{weight} + $F_{friction}$) can be used to calculate the acceleration:

```
a = Ftotal / mass
```

As before, write some Python code to call your function and collect the returned lists. Again, plot the length values vs. the elapsed time values in MatPlotLib, using blue circles for markers.

Sample Output:
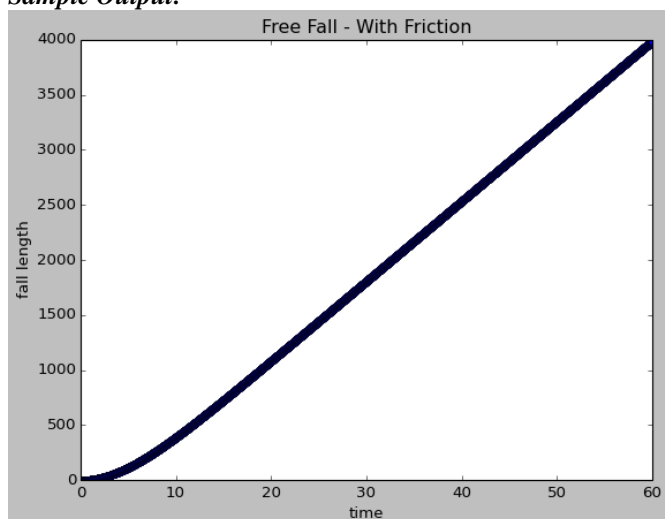
Part 1C

In this part of the assignment, we will add the effect of the spring (bungee cord) on the falling object, so that Julie-Ann can see a fairly realistic simulation of her bungee jump. Copy the function from part 2, and rename it simulateBungee to start. We'll have a new argument for this function:

- unstretchedBungeeLength: The length (in m) of the bungee/spring

Updating the length and velocity:
Updating the length and the velocity will use the same procedure as before, but now the acceleration is not a constant, since it depends on air friction (which itself is dependent upon velocity).

Updating the acceleration:
We now have three forces acting on our falling object:
1. the force due to gravity (we'll call this F_{weight})
2. the force due to air friction (we'll call this $F_{friction}$)
3. the force due to spring resistance (we'll call this F_{spring})

The force applied to a spring stretched by a displacement (d) is given by Hooke's Law:

$$F_{spring} = -kd \qquad \text{(Hooke's Law)}$$

In this formula, k is the spring constant, which depends on the stretchiness of the bungee cord. For our example, we'll use k = 21.7 for the spring constant. In the formula, d represents how much the spring has stretched, which can be calculated based on the length of the jump ('length') - bungee's (non-stretched) length. We'll test the function with an unstretched bungee length of 30m.

As before, write some Python code to call your function and collect the returned lists. Again, plot the length values vs. the elapsed time values in MatPlotLib, using green triangles for markers.

Sample Output:
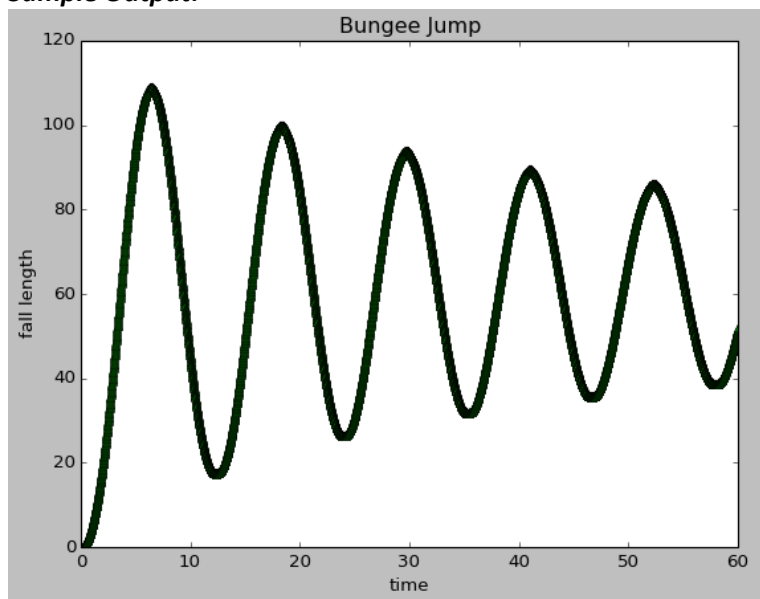
Overview – Part 2

For this part, you will write a simulation for the spread of an infectious disease. For simplicity, we will assume that the disease has a constant spread rate (0.15), fatality rate (0.025), and recovery rate (0.15).

Part 2A

For this part of this lab, you will write the function `simulateDay` that simulates a single day. This function will:

- For each susceptible person (not infected, dead, or recovered), simulate if he/she becomes infected, using the following formula:

```
infectionRate = spreadProb * numContacts * numInfected / numPeople
```

    spreadProb   – In one contact with an infected person, the probability of the disease spreading
    numContacts   – How many times does a typical person come into contact with people in a day
    numInfected   – How many infected people are there in the population
    numPeople   – How many people are there in the population (including infected, recovered, and susceptible)

- For each infected person, simulate if he/she recovers or dies

    deathProb   – The probability each day of someone infected dying from the disease
    recoverProb   – The probability each day of someone infected recovering from the disease

As an example to illustrate how to simulate is a person dies from the disease, assuming that the deathProb is a number between 0 and 1:

```
rand = random.random()
if rand < deathProb:
      # this patient died
```

The return value for this function should be a tuple containing four values:
- The number of fatalities at the end of the day
- The number of infected people at the end of the day
- The number of recovered people at the end of the day
- The number of susceptible people at the end of the day

Part 2B

In the test code, the function simulateNDays will be called with the specific parameters for the probabilities, given in the overview, the initial number of infected people (100), and the number of days to simulate (50). For the second part of this lab, you will write code to use the return values to draw a plot of the four categories of people after each day's end. As in simulateDay(), this function returns four values, but these values are lists:
- A list of fatalities after each day (red square)
- A list of infected after each day (yellow triangle)
- A list of recovered after each day (green circle)
- A list of susceptible after each day (blue plus)

Draw all four sets of values on a single plot using MatPlotLib. Use the markers and colours indicated above, beside each category of people. The resulting plot is shown in figure 8.1.

Include in your plot a legend, so that the observer can know at a glance what each shape represents. To include a legend for four plots, use the following code:

```
plt.legend(['Fatalities','Infected','Recovered','Susceptible'], loc='upper right')
```
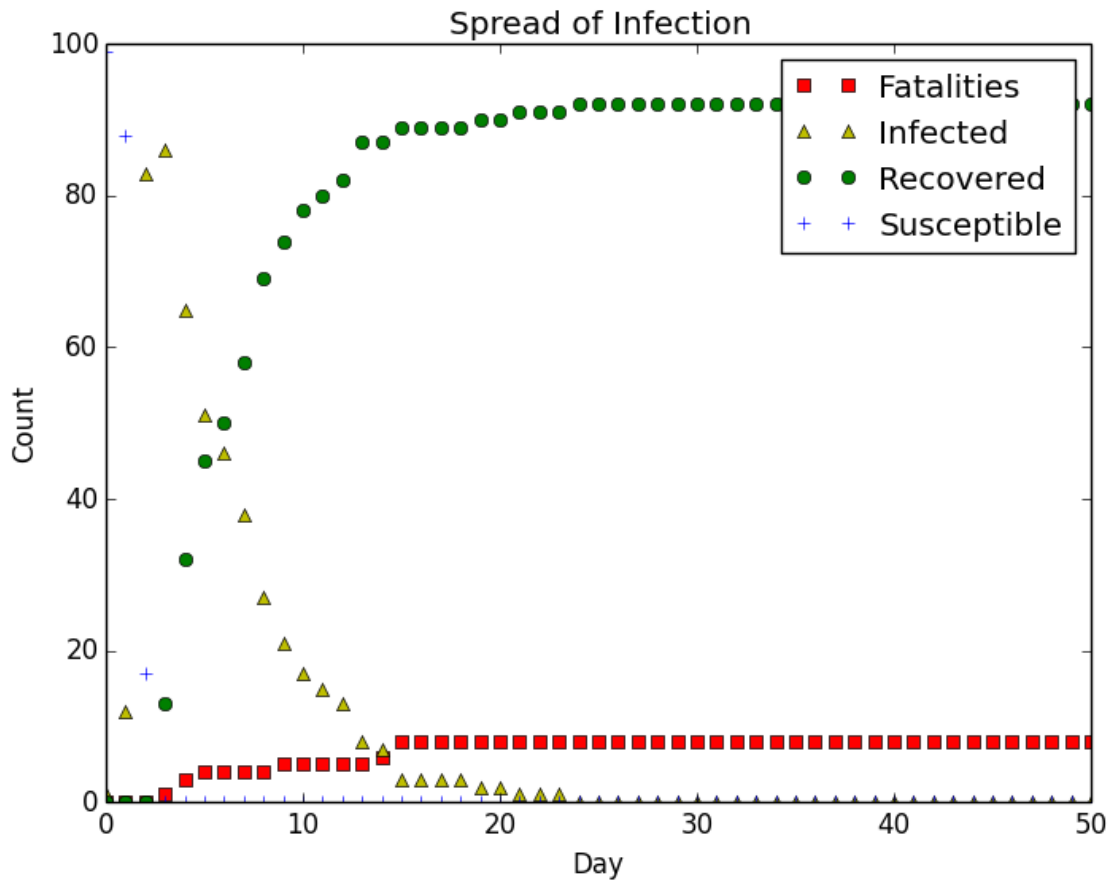

Sample Output – A Plot of fatalities, infected, recovered, susceptible people by day

How to Submit

Submit the Python file (containing all parts of the assignment) to the corresponding drop box on Blackboard. Name your file using the naming convention Lastname_FirstName_100200300_Asmt1.py (e.g. Fortier_Randy_100539147_Asmt1.py), and include a comment section at the top that contains your full name, your student ID, and a description of the assignment. Include comments for each part of the assignment, as well.