Lab Assignment #2

Due 2/5/2016 at 4pm on bCourses

Some guidelines for successfully completing an E7 assignment:

- First convert the engineering problem into a function that can be effectively calculated.
  - The function should be written down using pen and paper in the language of mathematics.
- Program the function.
  - Translate the mathematical language into a programming language, i.e., the language understood by your computer.
  - Then run the program and test it to see if its outputs match those specified by the function. If they do, compute the function on the input data and check that your answers make sense and are reasonable. If they do not...go back to the beginning and look for a mistake in your math or in your coding.
- Write comments throughout your code.
  - Clear, informative comments in your code are perhaps just as important as the code itself. The purpose of the comments is to help anybody who is reading the code later to understand what is happening. Useful comments include: describing (in English) what the important variables are and what information they hold, and describing what each chunk of code does. While the functions you will write in this assignment are relatively short, it is important to develop the habit of thoroughly commenting your code.

The assignment will be partially graded by an autograder which will check the outputs of your functions. For your functions to be scored properly, it is important that the names of your functions **exactly** match the names specified in the problem statements, and input and output variables to each function are in the correct order (i.e. use the given function header exactly). Instructions for submitting your assignment are included at the end of this document.

1. Write a MATLAB function that calculates the roots of a quadratic equation $ax^2 + bx + c = 0$ of unknown x, using the quadratic formula below. The file `solve_quadratic.m` has been provided to help you get started with this function (for the rest of the functions in this assignment you will need to create the file on your own).

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Use the following header for your function:

```
function [solutions] = solve_quadratic(a, b, c)
```

The function output `solutions` should be a 1×2 double. The first element of `solutions` should be the root computed by setting the $\pm$ to $+$, and the second element of `solutions` should be the root computed by setting the $\pm$ to $-$. You may assume $b^2 - 4ac \geq 0$.

Test cases:

```
>> solutions = solve_quadratic(1, -7, 10)

solutions =

     5     2

>> solutions = solve_quadratic(5, -23, 2)

solutions =

    4.5113    0.0887
```

2. Write a MATLAB function that computes the Euclidean distances between two sets of points. The x-coordinates for the first set of points will be stored in a column vector `X_1`, the y-coordinates for the first set of points in `Y_1`, the x-coordinates for the second set of points will be stored in a column vector `X_2`, the y-coordinates for the second set of points in `Y_2`. `X_1`, `Y_1`, `X_2`, and `Y_2` are N by 1 arrays where N is the number of points in each set. Note that, for a single pair of points, (x_1, y_1) and (x_2, y_2), the Euclidean distance can be obtained from the distance formula:

$$\text{Euclidean distance} = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}. \tag{1}$$

When you implement your function, use the following function header:

```
function [D] = euclidean_distance(X_1, Y_1, X_2, Y_2)
```

where `D` is the N by 1 column vector of distances corresponding to each coordinates pair in the input vectors. Your code should match the following test case:

```
>> X_1 = [0; 0]; Y_1 = [0; 0]; X_2 = [3; 5]; Y_2 = [4; 12];
>> D = euclidean_distance(X_1, Y_1, X_2, Y_2)

D =

     5
    13
```

3. Carbon dating is a method for determining the age of an object containing the radioisotope carbon-14. Since carbon-14 is subject to radioactive decay, the number of

carbon-14 atoms in a sample will decay over time according to the following exponential decay law:

$$N(t) = N_0 e^{-\lambda t}$$

where N_0 is the number of atoms of the isotope in the original sample, $N(t)$ is the remaining number of atoms after a time t, λ is the exponential decay constant and t is the time.

Write a function that calculates the remaining fraction of carbon-14 in a sample after a certain amount of time (in years). The remaining fraction is defined as the ratio of the number of atoms at the present time ($N(t)$) to the initial number of atoms (N_0). You can re-arrange the decay formula above to find an expression for this ratio. Use the following header for your function:

```
function [fraction] = c14_dating(time)
```

Assume $\lambda = 0.00012097$ years^{-1}. Test cases:

```
>> fraction = c14_dating(10000)

fraction =

    0.2983

>> fraction = c14_dating(5730)

fraction =

    0.5000
```

4. A ball is launched in the air with initial velocity v_0 from an initial position of coordinates (x_0, y_0), as illustrated below:

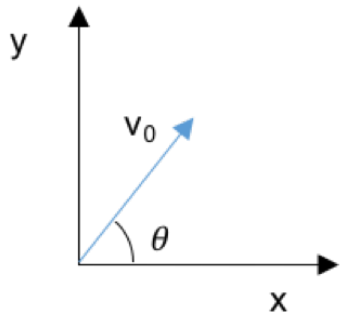

Figure 1: Ball launching at angle θ from the x-axis with initial velocity v_0

The ball is launched at an angle θ (expressed **in degrees**) from the x-axis. The equations describing the motion (i.e. the position of the ball $(x(t), y(t))$ as a function of time t) of the ball are:

$$x(t) = x_0 + v_0\, t \cos(\theta)$$
$$y(t) = y_0 + v_0\, t \sin(\theta) - \frac{1}{2} g t^2$$

where g is the gravitational acceleration ($g = 9.81$ m s^{-2}). Since θ is given in degrees, you may want to use the MATLAB functions `sind` and `cosd` instead of `sin` and `cos`.

For the following two parts, save each function you write in its own, independent `.m` file.

(a) Write a function that calculates the time at which the ball reaches the ground. Use the following function header:

```
function [time] = proj_time(y0, v0, theta)
```

Note that when the ball hits the ground, $y = 0$. Also, before writing your code, don't forget to use pen and paper to manipulate the equation(s) above and get a mathematical expression for `time`. Test case :

```
>> time = proj_time(0, 15, 40)

time =

    1.9657
```

(b) Write a function that calculates the horizontal distance traveled by the ball before it reaches the ground. You can use the function `proj_time` and the equations of motion. Use the following function header:

```
function [distance] = proj_distance(y0, v0, theta)
```

Test case:

```
>> dist = proj_distance(0, 15, 40)

dist =

   22.5873
```

5. Cell towers and triangulation. Note that part 5b will rely on the function you develop in part 5a.

(a) Write a function that draws a circle centered at the point of coordinates (x_c, y_c) and with radius r_c. Use the following function header:

```
function [] = draw_circle(xc, yc, rc)
```

Your function should return nothing, but it should plot a circle with a dashed black line for the perimeter and a red point for the center. Multiple function calls, along with the command `hold on` should plot multiple circles. For example, the code:

```
>> draw_circle(0, 0, 4);
>> hold on;
>> draw_circle(2, 2, 3);
```

should plot something similar to Figure 2. Don't worry about making the plot look *exactly* like Figure 2. In order to play with plotting options (such as line style), refer to this helpful page on using MATLAB's `plot` function: `http://www.mathworks.com/help/matlab/ref/plot.html` and/or type `help plot` in MATLAB's command window.

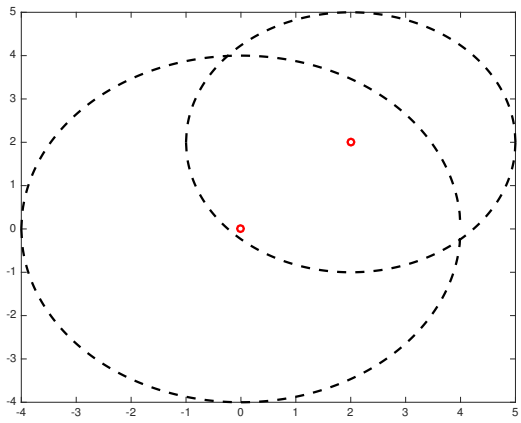

Figure 2: Draw circles example

(b) Even without GPS, cell phones can fairly accurately determine their location by using triangulation, the process by which the location can be determined using distance from a set of points with known locations. In the case of a cell phone, the set of known locations are cell towers in the service provider's network infrastructure.

For this problem, consider a set of three cell towers: towers 1, 2, and 3. We set up the coordinate system such that towers 1, 2, and 3 are located at the points of coordinates $(0, 0)$, $(0, y_2)$, and (x_3, y_3), respectively, with $0 < y_3 < y_2$ and $x_3 > 0$. Using signal propagation delays from each tower, it is determined that the distance between the cell phone and towers 1, 2, and 3 are r_1, r_2, and r_3, respectively.

Write a function that determines the cell phone's position, (x_{cell}, y_{cell}), in the coordinate system described above. To facilitate calculations, you may assume that the cell phone's position is constrained by $0 < x_{cell} < x_3$ and $0 < y_{cell} < y_2$. **Don't forget** to work this problem out with pen and paper before you start coding, as this problem will take some algebra! Use the following function header:

```
function [x_cell y_cell] = triangulate(y_2, x_3, y_3, ...
                                       r_1, r_2, r_3)
```

In the above header, `y_2`, `x_3`, `y_3`, `r_1`, `r_2`, `r_3`, `x_cell`, and `y_cell` represent the quantities y_2, x_3, y_3, r_1, r_2, r_3, x_{cell}, and y_{cell}, respectively.

In addition to returning the coordinates of the cell phone, the function should also plot circles of radii r_1, r_2, and r_3 centered at the locations of cell towers 1, 2, and 3, respectively. (Use your function from part a for this.) Finally, the function should plot a triangular point at the cell phone's location.

Test case:

```
>> y_2 = 3; x_3 = 8; y_3 = 1; r_1 = 2.24; r_2 = 1.41; r_3 = 7.07;
>> [x_cell, y_cell] = triangulate(y_2, x_3, y_3, r_1, r_2, r_3)

x_cell =

    1.0018

y_cell =

    2.0049
```

For the test case described above, the function should generate a plot similar to Figure 3.

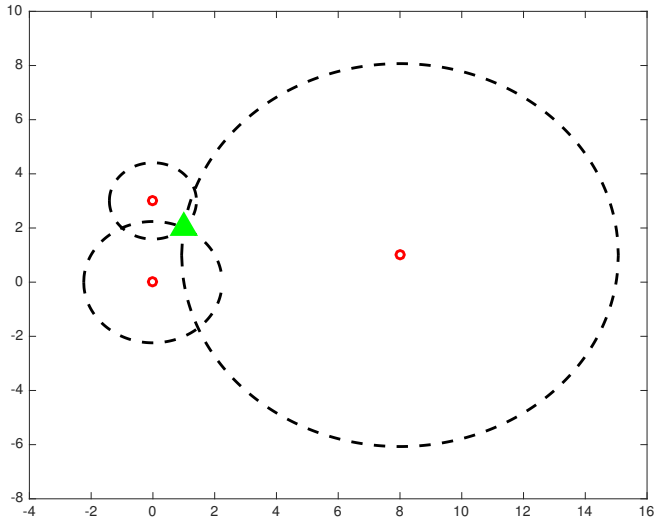

Figure 3: Triangulate example

Snap!

Snap! (formerly BYOB) is a visual, drag-and-drop programming language, as seen in lecture. Snap! has extended capabilities over Scratch which make it suitable for a serious introduction to computer science for high school or college students (e.g. it is used for CS 10 here at Berkeley). Snap! runs in your browser - if you have any trouble, try a different browser (they recommend Google Chrome).

6. Complete this tutorial (from CS 10). In this tutorial you will create an account, try out some test programs, and write your own Kaleidoscope program. (You can ignore the parts about pair programming. Answering the quiz questions may be helpful - but answers are not recorded.) You do not need to submit anything for this problem, but it will be helpful for the next problem.

7. For the next part, you will create a simulation of a molecule as it travels in a liquid or a gas - this is known as a random walk, which describes the Brownian motion of a molecule. (It may help to read this page on random blocks and this page on randomly moving a character.)

   (a) Think about how you want to structure your code before you start. Your sprite should start from the center and then perform a 2D random walk of a particle. First, change the costume for your sprite from the default arrow shape, by drawing a square or circle (using the Paint tool under the Costumes tab) of a reasonable size.

   Your sprite should do the following:

   i. randomly walk forever starting from the center of the stage (stop the program with the red button)
   ii. have a random step size and random direction
   iii. drag the pen around to show its path, and gradually change the pen color as it goes
   iv. play a random note at each step (values between 40 and 90 are good) for 0.1 beats
   v. pause between steps by 0.1 seconds (in addition to playing the note for 0.1 beats)
   vi. make sure it bounces back if it hits the edge of the stage

   For this problem, you will submit:

   (a) a link to your final Random Walk program (copy the URL). Make sure you have saved your program to the cloud. Submit your URL by filling out this Google Form.

   (b) a pdf named `<Firstname>_<Lastname>_Snap.pdf` that contains a screenshot of your final browser window. You can right click on the Scripts workspace and click 'clean' and it will order your blocks neatly. If you need more room you can expand the workspace with the sliding bar.

A full Snap reference manual can be found here but you shouldn't need it unless you try to get fancy. If you want more details see this page.

What to hand-in

You will submit a single zip file containing the following collection of files for this assignment:

```
solve_quadratic.m
euclidean_distance.m
c14_dating.m
proj_time.m
proj_distance.m
draw_circle.m
triangulate.m
<Firstname>_<Lastname>_Snap.pdf
```

Create a zip file with the above 8 files in it named `<Firstname>_<Lastname>_Lab2.zip` and submit this zip file to bCourses. Don't forget to submit your URL for your Snap! project via the Google form as well.