Cornerstone of Engineering
Fall 2020
Northeastern University
College of Engineering
Programming Lab 4
Due: November 6, 2020

Tutorials *(use command window to practice)*
1. Trajectories without Air Friction (see appendix)

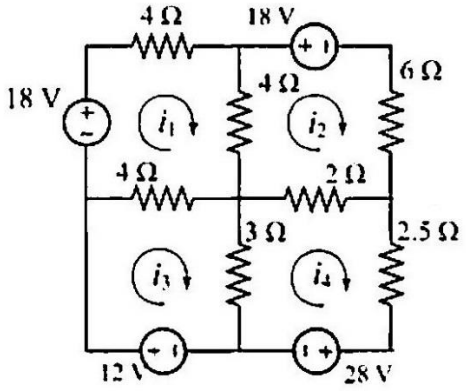
Programming Homework (need m-files)
1. Trajectories without Air Friction
Write an *m-file* called hw6_1_fml.m. You can edit the appendix Exercise file from the command window, but remember to only have in the m-file the commands that MATLAB can execute. Run the m-file in the MATLAB command window and make sure that you get the same output and plots as Exercise.

2. Use MATLAB to show that the sum of the infinite series $\sum_{n=0}^{\infty} \frac{2^n}{n!}$ Converges to e^2. Do this by computing the sum for:

a) n = 5
b) n = 10
c) n = 50

For each part create a vector n in which the first element is 0, increment is 1 and the last term is 5, 10, or 50. Then use element-by-element calculations to create a vector in which the elements are $2^n/n!$ Finally, use MATLAB's built-in function to sum of the series. Compare the values to e^2.

3. The electrical circuit shown consists of resistors and voltages sources. Determine i_1, i_2, i_3, and i_4, using the mesh current method based on Kirchhoff's voltage law



R_1=4; R_2=4; R_3=6; R_4=4; R_5=2; R_6=3; R_7=2.5
v_1=18; v_2=18; v_3=12; v_4=28
Loop1: $i_1R_1 + (i_1-i_2)R_2 + (i_1-i_3)R_4 - v_1 = 0$
Loop2: $(i_2-i_1)R_2 + i_2R_3 + (i_2-i_4)R_5 + v_2 = 0$
Loop3: $(i_3-i_1)R_4 + (i_3-i_4)R_6 - v_3 = 0$
Loop4: $(i_4-i_3)R_6 + (i_4-i_2)R_5 + i_4R_7 + v_4 = 0$

4. Biological Oxygen Demand (BOD) is a measure of the relative oxygen depletion effect of a waste contaminant and is widely used assess the amount of pollution in a water source. The BOD in the effluent (L_c in mg/L) of a rock filter without recirculation is given by:

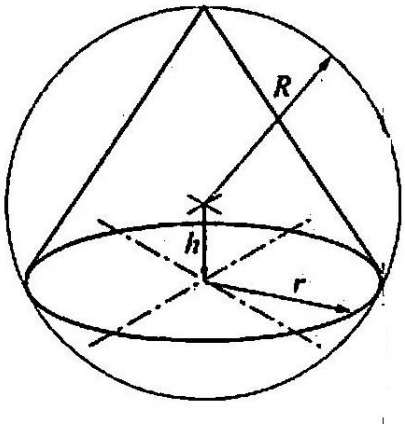
$$L_c = \frac{L_0}{1 + \frac{(2.5D^{2/3})}{\sqrt{Q}}}$$

Where L_0 is influent BOD (mg/L), D is the depth of the filter (m), and Q is the hydraulic flow rate (L/(m^2·day)).

a) Assuming Q = 300 L/(m^2·day) plot the effluent BOD as a function of the depth of the filter ($100 \leq D \leq 2000$m) for L_0 = 5, 10, 20 mg/L.
b) Make the three plots in one figure, label the axes and display a legend.
c) Estimate the depth of the filter required for each of these cases to obtain drinkable water.

5. A cone with base radius r and vertex in contact with the surface of a sphere is constructed inside a sphere, as shown in the sphere. The radius of the sphere is R = 9 in.

a) Create a polynomial expression for the volume V of the cone in terms of h.
b) Make a plot of V versus h for $-9 \leq h \leq 9$.
c) Use roots command to determine h if the volume of the cone is 500 in^3.
d) Determine the value of h that corresponds to the cone with the largest possible volume, and determine that volume.



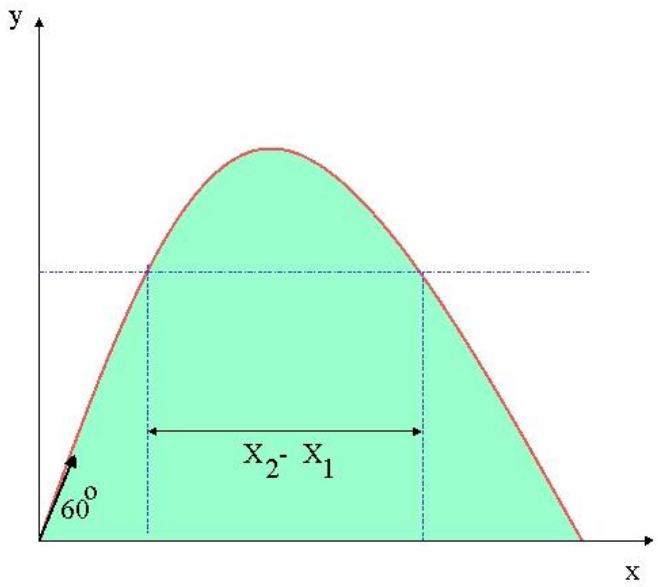
To Turn In via Blackboard:

1. Word file with cover page, tutorials, outputs, answers to any questions, and results and discussion
2. Source code e-file (upload to the blackboard, the source code should be saved as phw41_xxx.m, phw42_xxx.m, …etc., where xxx is your initial)

Appendix: Introduction to MATLAB

Trajectories without Air Friction

This exercise will use MATLAB to calculate and plot the frictionless trajectories of a projectile such as you have studied in physics. You will learn to use MATLAB to define variable names, use trig function, solve a quadratic equation, define an array, use array multiplication and exponentiation (".*" and ".^" in MATLAB) to calculate arrays of *x*- and *y*- points, and then plot these points to see the trajectory.



The problem is illustrated in the figure. A ball is projected at an angle of 60□ with an initial velocity of 100 m/sec and traces out a parabolic trajectory as shown. We want to find the distance x_2-x_1 for several different heights, *y*.

As you know from physics, the initial velocity can be broken into a horizontal and vertical component, V_{0x} and V_{0y}, and then the usual kinematic equations can be used for the *x*- and *y*- motion. Since there is no force in the *x*-direction, the *x*-component (distance) is given by:

$$x = v_{ox}\, t = v_o \cos(60)\, t$$

In the *y*-direction (height) there is a constant acceleration, g = -9.8 m/s^2, as well as the initial velocity, $V_{0y}=V_0 \sin(60)$, and the height of the ball is given by:

$$y = v_{oy}\, t + 1/2\, gt^2$$

(remember that *g* is a negative number). The instantaneous *y*-velocity is given by:

$$v_y = v_{oy} + gt$$

At the top of the trajectory V_y =0, so the time to reach the top can be found from $t_{top} = -v_{oy} / g$.

In the exercise we will select several heights (values of *y*), and solve the quadratic equation:

$$1/2\, gt^2 + v_{oy}\, t - y = 0$$

to find the times to reach the given height. In general there will be two times at which the ball will be at a height *y*. (Why is this? Are there any exceptions?) For each given height *y*, we will find both times, t_1 and t_2, and the horizontal distance between these points, $x_2 - x_1 = v_{ox}\,(t_2 - t_1)$. We will do this problem using an array for *y* so that we can solve for four different heights at once. We will then calculate the *x*- and *y*-coordinates for 101 values of time and plot the *y* vs. *x* positions to show the trajectory.

Exercise:

The MATLAB Command Window at a prompt like this >>. Follow the steps below by typing in the program below. The % sign is a comment, and is ignored by the program. You must fill in the comments where they are missing, that is, where there is a % but no comment following it. You do not have to type in all of the comments that are there for MATLAB assistance. Print out copies of the three plots to turn in.

```
% Your Name
>>g=-9.80;                    % gravitational Constant
>>initial_velocity=100;              %variables can be up to 19 characters long
>>initial_angle=60;           %use semicolon after entry to suppress printing
>>Vox=initial_velocity*cosd(initial_angle);
                              %cosd() is one of the many built-in functions
 >> Voy=initial_velocity*sind(initial_angle);
                              %sin() is one of the many built-in functions
>>time_to_top=-Voy/g                 %
time_to_top =
8.8370
>>max_height=0.5*g*time_to_top^2 + Voy*time_to_top     %
                               % note exponentiation operator ^
max_height =
382.6531
                              %Solve quadratic equation to find x2-x1 at y=0, 150, 300 and 400 m
>>height=[0 150 300 400];     %define an array of 4 heights
                              %(what will happen when we solve for the time at y=400?)
>>A=g/2;                             %
>>B=Voy;                            %
>>C=-height;                        %
>>time_1=(-B+sqrt(B^2-4*A*C))/(2*A) %solving for time 1 using quadratic eqn
                              %solve for all values of the height at once!
time_1 =
0 1.9464 4.7299 8.8370- 1.8815i
                              %last value is complex (why?)
>>                                  %solving for time 2 using quadratic eqn
                              %try using up arrow to get this step
time_2 = (-B-sqrt(B^2-4*A*C))/(2*A)
17.6740 15.7276 12.9441 8.8370+ 1.8815i
>>time_1(1:3)                 %1:3 corresponds to elements 1 2 3
ans =
0 1.9464 4.7299
>>rtime_1=time_1(1:3);        %define real time array not including complex answers
>>rtime_2=time_2(1:3);        %
>>x2_minus_x1=(rtime_2-rtime_1)*Vox
                              %subtracting arrays subtracts corresponding elements
x2_minus_x1 =
883.6994  689.0589  410.7064
                              %Find and plot the (x,y) points at y=0, 150, 300 m
>>time=[rtime_1 rtime_2]      %combine two time arrays into larger array
time =
0 1.9464 4.7299 17.6740 15.7276 12.9441
```

```
>>time.*time                    %like this
ans =
0 3.7885 22.3722 312.3698 247.3569 167.5486
>>y=0.5*g*time.*time+Voy*time          %
y =
0 150.0000 300.0000 0.0000 150.0000 300.0000
>>x = Vox*time
x =
0 97.3202 236.4965 883.6994 786.3791 647.2029
>>figure(1)                    %begin a new figure (Figure No. 1)
>>plot(x,y,'o')                %plot x-y pairs we found as points
>>xlabel('Distance(m)'); ylabel('Height(m)')      %
                               %Print out the graph now using File>Print in the Figure menu*
                                    %Plot out the full trajectory along with the points
>>N=0:100;                     %define a 101-point array N=0, 1, 2, 3, ... 100
                               %Don't forget the semicolon (;) for long arrays!
>>max_time=max(time)           %
max_time =
17.6740
                               %("max" is a built-in MATLAB function)
>>TIME=N*max_time/100;         %make TIME 101 values from 0 to max_time
>>XX=TIME*Vox;                 %
                               %always use semicolon for long arrays!
>>YY=0.5*g*TIME.^2+Voy*TIME;          %can also use .^ to do array exponentiation
>>figure(2)                    %begin a new figure instead of overwriting figure 1
>>plot(TIME,XX, TIME,YY)       %
>>xlabel('Time(s)');ylabel('Height and Distance(m)')      %
>>figure(3)                     %
>>plot(XX,YY,'-',x,y,'o')            %plot YY vs. XX as line; y vs. x as points
>>xlabel('Distance(m)');ylabel('Height(m)')       %label the axes
>>title('Your Name -  Trajectory for Vo=100 m/s, angle=60 degrees')      %title the graph
                               %Print out Figure 2 and 3 now using File>Print in the Figure menu
```