Table of Contents


```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%
% ASEN 3111 - CA1
%
% Created By: Johnathan Tucker
%
% Collaborators: N/A
%
% The purpose of the script is to contain all of the constants and
 basic
% computation that will be passed to the question functions where all
 of
% the necessary outputs for each question will come from
%
% Created Date: 1/21/2020
%
% Change Log:
%           - 1/25/2020: Code up trap and simpsons rule for the
 cylinder
%           - 1/28/2020: Code up trap rule and error logic for the
 airfoil
%           - 2/02/2020: Finalize formatting
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%
```

Housekeeping

```
clc;
clear all;
close all;
tic
```

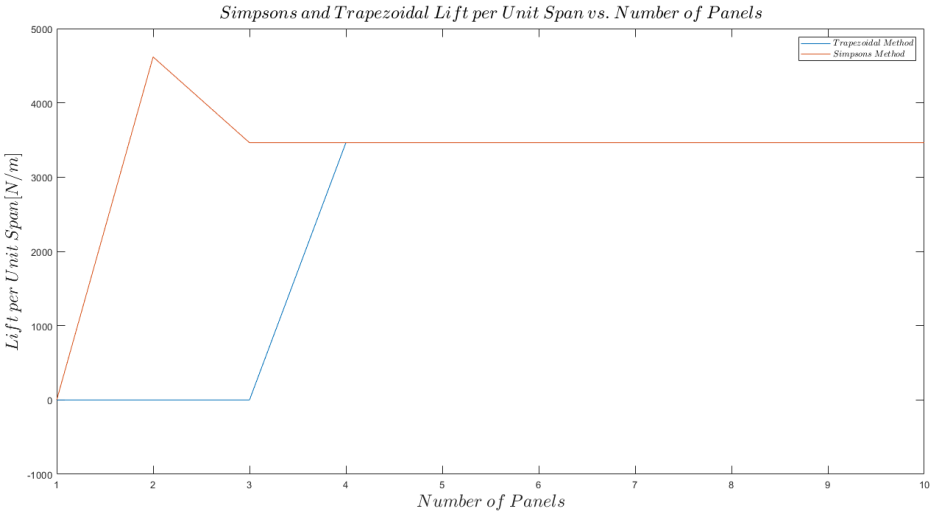
Create all necessary constants and put them in a struct

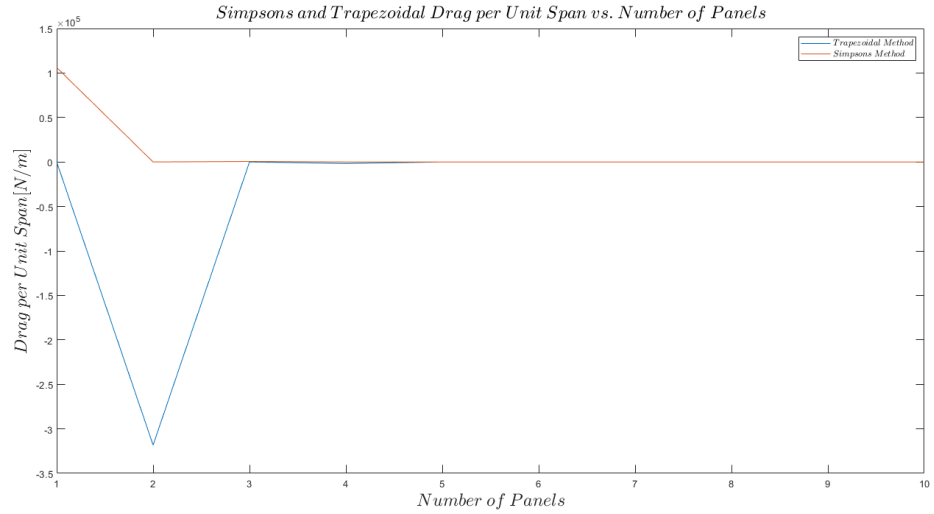
```
const.rho_inf = 1.225; % kg/m^3
const.p_inf = 101.3e3; % Pa
const.d = 1; % m
const.r = const.d/2; % m
const.v_inf = 30; % m/s
```

```
const.q_inf = 0.5*const.rho_inf*const.v_inf^2; % Pa
const.gamma = 2*pi*const.r*const.v_inf; % m^2/s^2
const.aoa = 9; % Degrees
const.chord = 2; % m
```

Begin Question 1

```
Question_1(const);
```





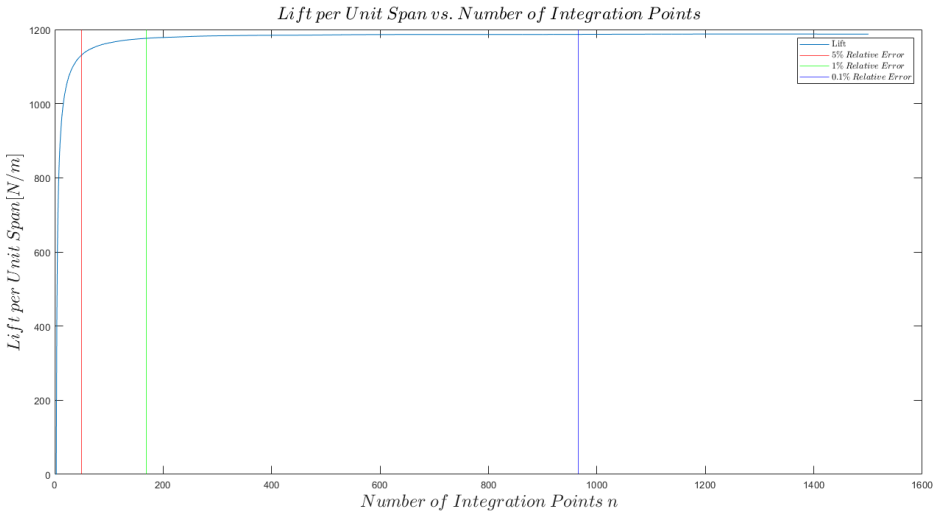
Begin Question 2

```
Question_2(const);
toc
```

```
%%%%%%%%BEGIN QUESTION TWO ANSWERS%%%%%%%%%%%%%%%
The Lift per unit span over the airfoil was caluclated to be 1.188110e
+03 [N/m] using the trapezoidal rule.
```

```
The Drag per unit span over the airfoil was caluclated to be 1.463459e
+00 [N/m] using the trapezoidal rule.

The number of panels required for a 5% relative error lift solution
 is: 48
The number of panels required for a 1% relative error lift solution
 is: 168
The number of panels required for a 0.1% relative error lift solution
 is: 964
Elapsed time is 18.921633 seconds.
```



Functions Used for CA 1 Below

```
function [] = Question_1(const)
%Q1_TRAPEZOIDAL Performs all calculations and outputs for question 1
%
% Author: Johnathan Tucker
% Collaborators: N/A
% This function takes in the struct of constants and outputs all of
 the
% required command line statements and plots for the first question
%
% Last Revised: 2/2/2020
% First obtain the exact solution using MATLABs symbolic solver
% Notice that since there is no AoA N=L and A=D
syms P(theta)
% Define pressure as a function of theta
P(theta) = const.q_inf*(-4*(sin(theta))^2 -4*sin(theta)) +
 const.p_inf;
% Define L and D and Solve them symbolically
L_exact = double(-const.r*int(P*sin(theta),theta,0,2*pi));
D_exact = double(-const.r*int(P*cos(theta),theta,0,2*pi));
fprintf("%%%%%%%%%%%%%%%%%%BEGIN QUESTION ONE ANSWERS%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%\n")
fprintf("Exact Lift Solution is: %d [N/m]\n",L_exact)
```

```
fprintf("Exact Drag Solution is: %d [N/m]\n\n",D_exact)

% Apply trapezoid rule to get lift and drag
% Pre-allocate memory
D_trap_vec = zeros(10,1);
L_trap_vec = zeros(10,1); %
% Loop through the trapezoid rule 9 times
for k = 2:10
    % Increment through the number of panels
    num_panels = linspace(0,2*pi,k);
    % Reset sum variables each loop
    sum_var_L = 0;
    sum_var_D = 0;
    for i = 1:k-1
        % Perform next trapezoidal iteration for lift
        sum_var_L_1 = 0.5*abs(num_panels(i+1) - num_panels(i))*...
            double((P(num_panels(i+1))*sin(num_panels(i+1)) +...
            P(num_panels(i))*sin(num_panels(i))));
        % Add it to the ongoing sum
        sum_var_L = sum_var_L + sum_var_L_1;
        % Update the lift solution
        L_trap = -const.r*sum_var_L;

        % Perform next trapezoidal iteration for drag
        sum_var_D_1 = 0.5*abs(num_panels(i+1) - num_panels(i))*...
            double((P(num_panels(i+1))*cos(num_panels(i+1)) +...
            P(num_panels(i))*cos(num_panels(i))));
        % Add it to the ongoing sum
        sum_var_D = sum_var_D + sum_var_D_1;
        % Update the drag solution
        D_trap = -const.r*sum_var_D;
    end
    % Save obtained L and D into their respective vectors
    L_trap_vec(k,:) = L_trap;
    D_trap_vec(k,:) = D_trap;
end

% Get the absolute error between the exact and estimated values
trap_L_error = abs(L_trap_vec(end) - L_exact);
trap_D_error = abs(D_trap_vec(end) - D_exact);

% These panel values were obtained from plot inspection
L_trap_panels = 4;
D_trap_panels = 3;

% Print out necessary information for the trapezoidal solutions
fprintf("Trapezoidal Lift solution is: %d [N/m]\n",L_trap_vec(end))
fprintf("Error between trapezoidal lift solution and exact lift
 solution is %d\n",trap_L_error)
fprintf("It took %d panels to get trapezoidal Lift within 0.001 N of
 the exact Lift\n\n",L_trap_panels)

fprintf("Trapezoidal Drag solution is: %d [N/m]\n",D_trap_vec(end))
```

```
fprintf("Error between trapezoidal drag solution and exact drag
 solution is %d\n",trap_D_error)
fprintf("It took %d panels to get trapezoidal Drag within 0.001 N of
 the exact Drag\n\n",D_trap_panels)

% Apply Simpsons rule to get lift and drag
% Loop through the simpsons rule process 10 times
for i = 1:10
    % Recalculate and reassign necessary variables each loop
    h = 2*pi/(2*i);
    sum_var = 0;
    % Loop through the simpsons process for drag
    for k = 1:i
        % Calculate the next increment in the summation
        sum_var_1 = double(P((2*k-2)*h)*cos((2*k-2)*h) +...
        4*P((2*k-1)*h)*cos((2*k-1)*h) + P((2*k)*h)*cos((2*k)*h));
        % Add it to all previous increments
        sum_var = sum_var + sum_var_1;
        % Update drag solution
        D_simps = -(h/3)*const.r*sum_var;

    end
    % Re-zero summation variable
    sum_var = 0;
    % Loop through the simpsons process for lift
    for k = 1:i
        % Calculate the next increment in the summation
        sum_var_1 = double(P((2*k-2)*h)*sin((2*k-2)*h) +...
        4*P((2*k-1)*h)*sin((2*k-1)*h) + P((2*k)*h)*sin((2*k)*h));
        % Add it to all previous increments
        sum_var = sum_var + sum_var_1;
        % Update drag solution
        L_simps = -(h/3)*const.r*sum_var;

    end
    % Save each increment to a vector for plotting
    L_simps_vec(i,:) = L_simps;
    D_simps_vec(i,:) = D_simps;
end
% Calculate absolute error between simpsons solution and exact
 solution
simp_L_error = abs(L_simps_vec(end) - L_exact);
simp_D_error = abs(D_simps_vec(end) - D_exact);

% These panel values were obtained from plot inspection
L_simp_panels = 3;
D_simp_panels = 2;
% Print out necessary information for the simpsons solutions
fprintf("Simpsons Lift solution is: %d [N/m]\n",L_simps_vec(end))
fprintf("Error between simpsons lift solution and exact lift solution
 is: %d\n",simp_L_error)
fprintf("It took %d panels to get simpson Lift within 0.001 N of the
 exact Lift\n\n",L_simp_panels)
```

```
fprintf("Simpsons Drag solution is: %d [N/m]\n",D_simps_vec(end))
fprintf("Error between simpsons drag solution and exact drag solution
 is: %d\n",simp_D_error)
fprintf("It took %d panels to get simpson Drag within 0.001 N of the
 exact Drag\n\n",D_simp_panels)

% Create Plots for question 1
figure(1)
set(gcf, 'Units', 'Normalized', 'OuterPosition', [0, 0.04, 1, 0.96]);
plot(1:1:10, L_trap_vec)
hold on
plot(1:1:10, L_simps_vec)
xlabel("$Number\:of\:Panels$",'Interpreter','latex','FontSize',18)
ylabel("$Lift\:per\:Unit\:Span[N/
m]$",'Interpreter','latex','FontSize',18)
title("$Simpsons\:and\:Trapezoidal\:Lift\:per\:Unit\:Span\:vs.\:Number
\:of\:Panels$",...
    'Interpreter','latex','FontSize',18)
legend("$Trapezoidal\:Method$","$Simpsons\:Method
$",'Interpreter','latex')

figure(2)
set(gcf, 'Units', 'Normalized', 'OuterPosition', [0, 0.04, 1, 0.96]);
plot(1:1:10, D_trap_vec)
hold on
plot(1:1:10, D_simps_vec)
xlabel("$Number\:of\:Panels$",'Interpreter','latex','FontSize',18)
ylabel("$Drag\:per\:Unit\:Span[N/
m]$",'Interpreter','latex','FontSize',18)
title("$Simpsons\:and\:Trapezoidal\:Drag\:per\:Unit\:Span\:vs.\:Number
\:of\:Panels$",...
    'Interpreter','latex','FontSize',18)
legend("$Trapezoidal\:Method$","$Simpsons\:Method
$",'Interpreter','latex')

end

function [] = Question_2(const)
%QUESTION_2 Performs all calculations and outputs for question 2
%
% Author: Johnathan Tucker
% Collaborators: N/A
% This function takes in the struct of constants and outputs all of
 the
% required command line statements and plots for the second question
%
% Last Revised: 2/2/2020
% Begin the solution process for question 2
% Load the Cp data
load('Cp.mat')

% Solve for lift using the trapezoid method. This will be used for the
% relative error when determining the number of panels for 1,5, and .1
% percent error.
```

```
%NOTE: The code commented below was run through the trapezoidal rule
 with
%150,000 panels to get an approximate lift value that will be used as
 the
%"exact" value. Uncomment the code below to confirm the L_exact_2 and
%D_exact_2 values below it.
% num_panels = 150000;
% [L_exact_2, D_exact_2] =
 airfoil_LD(const,num_panels,Cp_upper,Cp_lower);
L_exact_2 = 1.188109731345318e+03; %N/m
D_exact_2 = 1.46345878973628; %N/m
fprintf("%%%%%%%%%%%%%%%%%%BEGIN QUESTION TWO ANSWERS%%%%%%%%%%%%%%%%%%
%%%%%%%%%%\n")
fprintf("The Lift per unit span over the airfoil was caluclated to be
 %d [N/m] using the trapezoidal rule.\n",L_exact_2)
fprintf("The Drag per unit span over the airfoil was caluclated to be
 %d [N/m] using the trapezoidal rule.\n\n",D_exact_2)

% Create constants necessary to begin while loops
L_error = 0;
num_panels = 1;
% Loop until the relative error is equal to or less than 5 percent
while abs(L_exact_2 - L_error)/L_exact_2 > 0.05
    % Get the updated Lift
    [L_error,~] = airfoil_LD(const,num_panels,Cp_upper,Cp_lower);
    % Update the 5 percent number of panels solution
    num_panels_5_percent = num_panels;
    % Increment number of panels
    num_panels = num_panels + 1;
end
% Loop until the relative error is equal to or less than 1 percent
while abs(L_exact_2 - L_error)/L_exact_2 > 0.01
    % Get the updated Lift
    [L_error,~] = airfoil_LD(const,num_panels,Cp_upper,Cp_lower);
    % Update the 1 percent number of panels solution
    num_panels_1_percent = num_panels;
    % Increment number of panels
    num_panels = num_panels + 1;
end
% Seed the number of panels to 600 to reduce runtime. This value is
 less
% than the actual value but determined by letting it run completely
 without
% the seed a few times.
num_panels = 960;
% Loop until the relative error is equal to or less than .1 percent
while abs(L_exact_2 - L_error)/L_exact_2 > 0.001
    % Get the updated Lift
    [L_error,~] = airfoil_LD(const,num_panels,Cp_upper,Cp_lower);
    % Update the .1 percent number of panels solution
    num_panels_tenth_percent = num_panels;
    % Increment number of panels
    num_panels = num_panels + 1;
```

```
end
% Begin Required Outputs Section
% Print all required statements
fprintf("The number of panels required for a 5%% relative error lift
 solution is: %d\n",num_panels_5_percent);
fprintf("The number of panels required for a 1%% relative error lift
 solution is: %d\n",num_panels_1_percent);
fprintf("The number of panels required for a 0.1%% relative error lift
 solution is: %d\n",num_panels_tenth_percent);

% The chunk of code below will compute the lift for a scaling number
 of
% panels up to 1500. The lift value at each panel was put into a
 vector and
% that vector was then saved to be used later.
% for i = 1:1500
%     [L,D] = airfoil_LD(const,i,Cp_upper,Cp_lower);
%     L_vec(i,:) = L;
% end
% save('LiftVector','L_vec')


% Convert number of panels to number of integration points
N = 1:1:1500;
n = N + 1;
% Use the above saved lift vector to plot the lift vs number of
 integration
% steps n
load('LiftVector.mat')
figure(3)
set(gcf, 'Units', 'Normalized', 'OuterPosition', [0, 0.04, 1, 0.96]);
plot(n,L_vec)
hold on
xline(num_panels_5_percent+1,'r');
hold on
xline(num_panels_1_percent+1, 'g');
hold on
xline(num_panels_tenth_percent+1, 'b');
xlabel("$Number\:of\:Integration\:Points\:n
$",'Interpreter','latex','FontSize',18)
ylabel("$Lift\:per\:Unit\:Span[N/
m]$",'Interpreter','latex','FontSize',18)
title("$Lift\:per\:Unit\:Span\:vs.\:Number\:of\:Integration\:Points
$",...
    'Interpreter','latex','FontSize',18)
legend("Lift","$5\%\:Relative\:Error$","$1\%\:Relative\:Error$","$0.1\
%\:Relative\:Error$",'Interpreter','latex')
end

function [L,D] = airfoil_LD(const, num_panels, Cp_upper, Cp_lower)
%AIRFOIL_LD Output lift and drag of the airfoil using trapezoidal rule
%
% Author: Johnathan Tucker
% Collaborators: N/A
```

```
% This function takes in the struct of constants, number of panels to
 use,
% and the upper/lower Cp data. It outputs the lift and drag for the
 airfoil
% after using the trapezoidal rule to find the normal and axial
 forces,
% which are then converted to L and D.
%
% Last Revised: 2/2/2020
% Begin the trapezoidal rule process for the airfoil
% Create thickness as a fraction of chord
t = 12/100;
% Create vector for percentage of chord length
x = linspace(0,const.chord,num_panels);
% Create a vector of half thicknesses
y = (t/0.2)*const.chord*(0.2969.*sqrt(x./const.chord) - ...
    0.1260.*(x./const.chord) - 0.3516.*(x./const.chord).^2 +...
    0.2843.*(x./const.chord).^3 - 0.1036.*(x./const.chord).^4);

% Create summation variables
N_sum = 0;
A_sum = 0;
% Loop through the number of panels given minus one so that I can
 fully
% increment through the x vector
for i = 1:(num_panels -1)
    % Calculate the upper and lower pressures for the even and odd
 increments
    p_u_even = fnval(Cp_upper,x(i+1)/const.chord)*const.q_inf +
 const.p_inf;
    p_u_odd = fnval(Cp_upper,x(i)/const.chord)*const.q_inf +
 const.p_inf;
    p_l_even = fnval(Cp_lower,x(i+1)/const.chord)*const.q_inf +
 const.p_inf;
    p_l_odd = fnval(Cp_lower,x(i)/const.chord)*const.q_inf +
 const.p_inf;

    % Calculate the normal force
    N  = 0.5*(-(p_u_even + p_u_odd) + (p_l_even + p_l_odd))*(x(i+1) -
 x(i));
    % Add the normal force to the ongoing summation
    N_sum = N_sum + N;
    % Calculate the axial force
    A = 0.5*((p_u_even + p_u_odd) + (p_l_even + p_l_odd))*(y(i+1) -
 y(i));
    % Add the axial force to the ongoing summation
    A_sum = A_sum + A;
end

% Convert from N and A to L and D
L = N_sum*cosd(const.aoa) - A_sum*sind(const.aoa);
D = A_sum*cosd(const.aoa) + N_sum*sind(const.aoa);
end
```

```
%%%%%%%%BEGIN QUESTION ONE ANSWERS%%%%%%%%%%%%%%%
Exact Lift Solution is: 3.463606e+03 [N/m]
Exact Drag Solution is: 0 [N/m]

Trapezoidal Lift solution is: 3.463606e+03 [N/m]
Error between trapezoidal lift solution and exact lift solution is
 2.137313e-11
It took 4 panels to get trapezoidal Lift within 0.001 N of the exact
 Lift

Trapezoidal Drag solution is: 1.455192e-11 [N/m]
Error between trapezoidal drag solution and exact drag solution is
 1.455192e-11
It took 3 panels to get trapezoidal Drag within 0.001 N of the exact
 Drag

Simpsons Lift solution is: 3.463606e+03 [N/m]
Error between simpsons lift solution and exact lift solution is:
 1.227818e-11
It took 3 panels to get simpson Lift within 0.001 N of the exact Lift

Simpsons Drag solution is: 1.828648e-11 [N/m]
Error between simpsons drag solution and exact drag solution is:
 1.828648e-11
It took 2 panels to get simpson Drag within 0.001 N of the exact Drag
```

Published with MATLAB® R2019a