Zach Swain 4/2/18
MEEG 332
Coding Challenge 2

Part 1

Define a function in a very similar manner to Dr. Feser's example provided in lecture making relevant definitions, but making dyvect_dt a three-column vector.

Use ode45( ) to evaluate the ODE, using the given n span and initial conditions. Isolate each column as f, f', f''. Evaluate where y1=0, y2=0, y3=0.5 as prompted. Plot solutions y1, y2, y3 vs. n for given initial conditions.

Evaluate y2 at 100, where it is horizontal, equivalent to the value at infinity. Run the script again after making a small alteration the third column entry in the y0 definition. Note the change in y2(100). If it got closer to 1, continue changing the initial condition in the same direction; if it got further from 1, begin changing the initial condition in the opposite direction as the original change. Record a range of values that will output y2(100) = 1.0000.

Use an initial guess of .5 as used before. Make necessary definitions such as n, y0, and y2(via ode45) to begin setting up a while loop. Find the difference between the y2(infinity) obtained using your initial guess and 1. If that difference is greater than zero, enter the appropriate while loop. Originally, the adjustments made to approximate which alpha will give y2(infinity)=1 were done in very tiny increments to provide precise results, but iterating ~1.68 million times proved to be fairly time-consuming at around 5-10 min. This needed to be optimized. Instead of making all adjustments from your initial guess at the same step size, much courser adjustments were made while possible – and incrementally so. So as the step size was gradually being refined, the difference between the current alpha value being looped and 1 was also decreasing as it honed in on a precise alpha result. This provided a run time of a mater of seconds to achieve a resulting alpha. If the difference between the y2(infinity) obtained using your initial guess and 1 is less than zero, then copy & paste the above-zero code and make the relevant necessary +,-,<,> adjustments.

Part 2

Make relevant if statements to reflect the piecewise functions defining f' as provided for u/U. Overlay that function with the y2=f' found as the ODE solution on a plot vs. n.

Part 3

As there are no fluid properties provided, the value of ν cannot be determined, and the displacement thickness cannot be determined as instructed by using the given expression involving A. Use trapz( ) to evaluate the A integral using the y2 profile calculated.

Part 4

Since ν cannot be determined, the momentum thickness cannot be determined as instructed by using the given expression involving B. Use trapz( ) to evaluate the B integral using the y2 profile calculated.

Coding Challenge 2 - Part 1

Zach Swain, 4/2/18, All files available at https://www.github.com/zswain/MEEG332

```
function dyvect_dt = lamBoundLayerVeloODE(n,y)

f = 0;              %let initial guess = 0

y1 = f;             %substitution/definition as given in part a
y2 = diff(y1);      %y2 = f' as given in part a
y3 = diff(y2);      %y3 = f" as given in part a

dyvect_dt(1,1) = y(2); %first row of column vector
dyvect_dt(2,1) = y(3); %second row of column vector
dyvect_dt(3,1) = -.5*y(1)*y(3); %f''' as given in simplified x
 momentum eq, third row
end
```

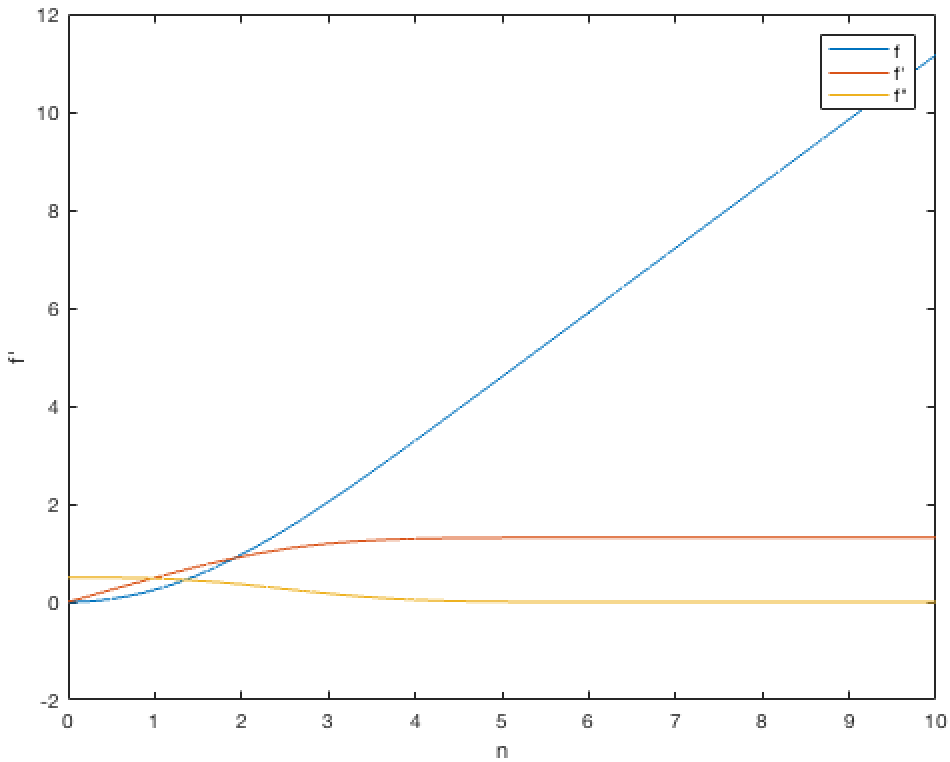
Published with MATLAB® R2017b

Coding Challenge 2 - Part 1

Zach Swain, 4/2/18, All files available at https://www.github.com/zswain/MEEG332

```
clear all

n = 0:.1:10;     %let n span from 0 to 10 as given, steps of .1 for
 decent precision
y0 = [0 0 0.5]; %define given initial conditions for y1,y2,y3

[nSol,ySol] = ode45(@(n,y) lamBoundLayerVeloODE(n,y),n,y0); %evaluate
 the ODE and give result ySol
y1 = ySol(:,1); %define y1 as all rows in column 1 of ySol, f
y2 = ySol(:,2); %define y2 as all rows in column 2 of ySol, f'
y3 = ySol(:,3); %define y3 as all rows in column 3 of ySol, f"

y1(1)           %evaluate where y1 = 0
y2(1)           %evaluate where y2 = 0
y3(1)           %evaluate where y3 = .5

figure(1)       %plot f,f',f" vs. n
plot(n,y1,n,y2,n,y3)
xlabel('n')
ylabel("f'")
legend('f',"f'",'f"')

y2(100);        %play with initial guess in y0 definition to find
 y2@infinity = 1
                %let acceptable range be whatever values produce
 1.0000 in format short
                %for 0.332019 < y3(0)< .332068 matlab outputs y2 @ n =
 10 as 1.0000
```

```
ans =

     0


ans =

     0


ans =

    0.5000
```



Published with MATLAB® R2017b

Coding Challenge 2 - Part 1

Zach Swain, 4/2/18, All files available at https://www.github.com/zswain/MEEG332

```
clear all

initGuess = .5;           %let initial guess be .5 as in c

n = 0:.1:10;              %same n range and step
y0 = [0 0 initGuess];     %same y0 definition as in c
[nSol,ySol] = ode45(@(n,y) lamBoundLayerVeloODE(n,y),n,y0); %evaluate
 the ODE as in c
y2 = ySol(:,2);           %define y2 as all rows in column 2 of ySol,
 f'

diff = y2(100)-1;         %evaluate the difference between y2@infinity
 and 1
if diff > 0               %if initial guess gives y2@infinity greater
 than 1
    guess = initGuess;    %for cohesion in while
    while diff > 0        %until y2@infinity gets just barely below 1
        y0 = [0 0 guess]; %redefine y0 with current guess value
        [nSol,ySol] = ode45(@(n,y)
 lamBoundLayerVeloODE(n,y),n,y0); %update evaluation of ODE
        y2 = ySol(:,2);   %update y2 def as current y2 column 2
        diff = y2(100)-1; %update new difference from 1
        if diff > .002    %originally had just guess-=.0000001 but
 runtimes were absurd
            guess = guess-.001; %blunt if statements are a poor man's
 optimization
        end
        if diff <= .002 && diff > .00014
            guess = guess-.0001; %more precise guess steps
        end
        if diff <= .00014 && diff > .000035
            guess = guess-.00001; %more precise guess steps
        end
        if diff <= .000035 && diff > .00001
            guess = guess-.000001; %more precise guess steps
        end
        if diff <= .00001
            guess = guess-.0000001; %actual guess step precision
 wanted
        end
    end
    alpha = guess         %define alpha as latest guess value
end

y0 = [0 0 initGuess];     %redefine y0 to have original guess not
 latest guess value from while
[nSol,ySol] = ode45(@(n,y) lamBoundLayerVeloODE(n,y),n,y0); %redefine
 using original guess
```

```
y2 = ySol(:,2);            %redefine using original guess
                           %redefs to ensure cant go through both
 whiles
diff = y2(100)-1;          %redefine diff to original
if diff < 0                %if inital guess gives y2 less than 1
    guess = initGuess;     %for cohesion in while
    while diff < 0         %until y2@infinity gets just barely above 1
        y0 = [0 0 guess]; %redefine y0 with current guess value
        [nSol,ySol] = ode45(@(n,y)
 lamBoundLayerVeloODE(n,y),n,y0); %update evaluation of ODE
        y2 = ySol(:,2);    %update y2 def as current y2 column 2
        diff = y2(100)-1; %update new difference from 1
        if diff < -.002    %ifs to avoid long runtimes as previous
 while
            guess = guess+.001; %makeshift optimizaiton as previous
 while
        end
        if diff >= -.002 && diff < -.00014
            guess = guess+.0001; %more precise guess steps
        end
        if diff >= -.00014 && diff < -.000035
            guess = guess+.00001; %more precise guess steps
        end
        if diff >= -.000035 && diff < -.00001
            guess = guess+.000001; %more precise guess steps
        end
        if diff >= -.00001
            guess = guess+.0000001; %more precise guess steps
        end
    end
    alpha = guess          %define alpha as latest guess value
end
```

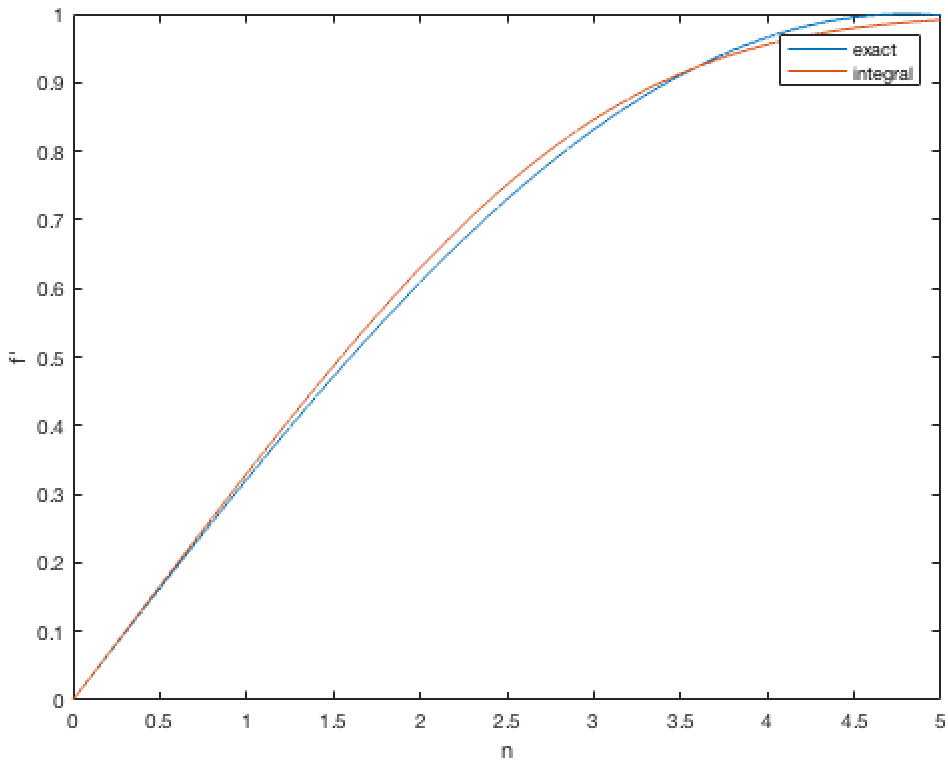
```
alpha =

    0.3320
```

Published with MATLAB® R2017b

Coding Challenge 2 - Parts 2,3,4

Zach Swain, 4/2/18, All files available at https://www.github.com/zswain/MEEG332

```
clear all

n = 0:.05:5;          %now using n from 0-5, decide to keep use amount
 of steps
y0 = [0 0 .332043]; %define y0 as before but use found alpha value
[nSol,ySol] = ode45(@(n,y) lamBoundLayerVeloODE(n,y),n,y0); %evaluate
 ODE

y1 = ySol(:,1);       %define y1 as all rows in column 1 of ySol, f
y2 = ySol(:,2);       %define y1 as all rows in column 1 of ySol, f'
y3 = ySol(:,3);       %define y1 as all rows in column 1 of ySol, f"

if n>4.8 & n<5        %if n between 4.8-5
    y = 1;            %let y = 1
else                  %if y between 0-4.8
    y = sin((pi/2)*(n/4.8)); %let y as defined
end

figure(2)             %plot f' from ODE and sin func given, both vs. n
plot(n,y,n,y2)
xlabel('n')
ylabel("f'")
legend('exact','integral')

A = trapz(n,1-(y2/y2(100))) %evaluate integral given as A using trapz,
 A found as 1.6860, A_book = 1.7210

B = trapz(n,((y2/y2(100)).*(1-(y2/y2(100))))) %evaluate integral given
 as B using trapz, B found as .6414, B_book = .6640


A =

    1.6860


B =

    0.6414
```



Published with MATLAB® R2017b