# ME 594 – Numerical Methods – HW09

Viral Panchal | Due Date: 11/30

"I pledge my honor that I have abided by the Stevens Honor System"

**1*.** An important problem in fluid dynamics concerns the motion of a layer of fluid, such as the earth's atmosphere, that is warmer at the bottom than at the top. If the vertical temperature difference $\Delta T$ is small, then there is a linear variation of temperature with altitude but no significant motion of the fluid layer. However, if $\Delta T$ is large enough, then the warmer air rises, displacing the cooler air above it, and a steady convective motion results. If the temperature difference increases further, then eventually the steady convective flow breaks up and a more complex turbulent motion ensues.

The nonlinear system of equations governing the phenomena is written

$$\begin{aligned}\frac{\mathrm{d}x}{\mathrm{d}t} &= \sigma\left(-x+y\right)\\ \frac{\mathrm{d}y}{\mathrm{d}t} &= rx-y-xz\\ \frac{\mathrm{d}z}{\mathrm{d}t} &= -bz+xy,\end{aligned}$$

and is commonly referred to at the Lorentz equations. The variable $x$ is related to the intensity of the fluid motion, while the variables $y$ and $z$ are related to the temperature variations in the horizontal and vertical directions. The parameters $\sigma$ and $b$ depend on the material and geometrical properties of the fluid layer. For the earth's atmosphere, reasonable values of these parameters are $\sigma = 10$ and $b = 8/3$. The parameter $r$ is proportional to the temperature difference $\Delta T$, and is the critical parameter in the analysis; take $r = 21$ for this problem.

Start with the initial point $(x_0, y_0, z_0) = (5, 5, 10)$, and use any (self-written) numerical method you like to evolve the system numerically. To receive full credit, you must find the equilibrium point of the system, (7.302967, 7.302967, 20). Plot the projected trajectory of the system in the $xy-$plane and the $xz-$plane, respectively. Also plot the evolution of $x$, $y$, and $z$ as a function of time on separate plots. Include the code that was used to evolve the system. For best results, use RK4 with $\Delta t = 0.01$ up to $t_{\max} = 200$.

## MATLAB code:

- ***RK4 function:***

```
% Function to imple Runge-Kutta method

function [x,y,z] = RK4(ODE,a,b,h,x0,y0,z0)

n = (b-a)/h;
t = zeros(1,n+1);
x = zeros(1,n+1);
y = zeros(1,n+1);
z = zeros(1,n+1);

K = zeros(3,4);

t(1) = a;
x(1) = x0;
y(1) = y0;
z(1) = z0;

for i = 1:n
    t(i+1) = t(i) + h;
    [K(:,1)] = feval(ODE,x(i),y(i),z(i));
    [K(:,2)] =
feval(ODE,x(i)+0.5*K(1,1)*h,y(i)+0.5*K(2,1)*h,z(i)+0.5*K(3,1)*h);
    [K(:,3)] =
feval(ODE,x(i)+0.5*K(1,2)*h,y(i)+0.5*K(2,2)*h,z(i)+0.5*K(3,2)*h);
    [K(:,4)] = feval(ODE,x(i)+K(1,3)*h,y(i)+K(2,3)*h,z(i)+K(3,3)*h);
    x(i+1) = x(i) + 1/6*(K(1,1)+2*K(1,2)+2*K(1,3)+K(1,4))*h;
    y(i+1) = y(i) + 1/6*(K(2,1)+2*K(2,2)+2*K(2,3)+K(2,4))*h;
    z(i+1) = z(i) + 1/6*(K(3,1)+2*K(3,2)+2*K(3,3)+K(3,4))*h;
end

plot(t,x);
xlabel('t');
ylabel('x');
axis padded
grid on

figure
plot(t,y)
xlabel('t')
ylabel('y')
axis padded
grid on

figure
plot(t,z)
xlabel('t')
ylabel('z')
axis padded
grid on

figure
plot(x,y)
xlabel('x')
```

```
ylabel('y')
axis padded
grid on

figure
plot(x,z)
xlabel('x')
ylabel('z')
axis padded
grid on

fprintf('Equilibrium point: \n\n')
disp([x(n+1),y(n+1),z(n+1)])
```

- ***Script for Lorentz function:***

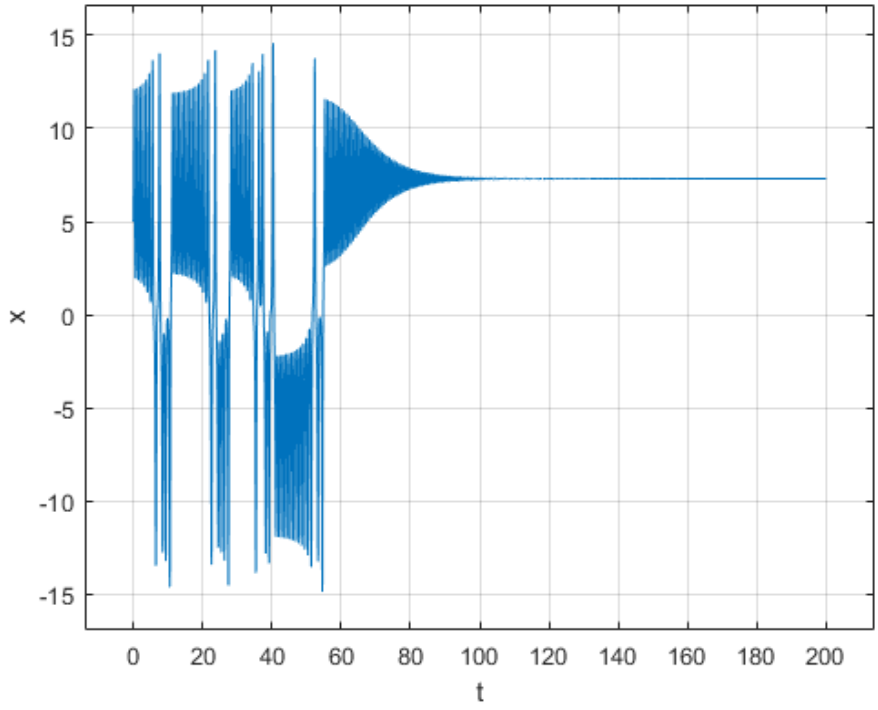
```
% Function to implement Lorentz method
function dVdt = lorentz(x,y,z)

r = 21;
b = 8/3;
sigma = 10;

dVdt(1) = sigma*(-x+y);
dVdt(2) = r*x-y-x*z;
dVdt(3) = -b*z + x*y;
```

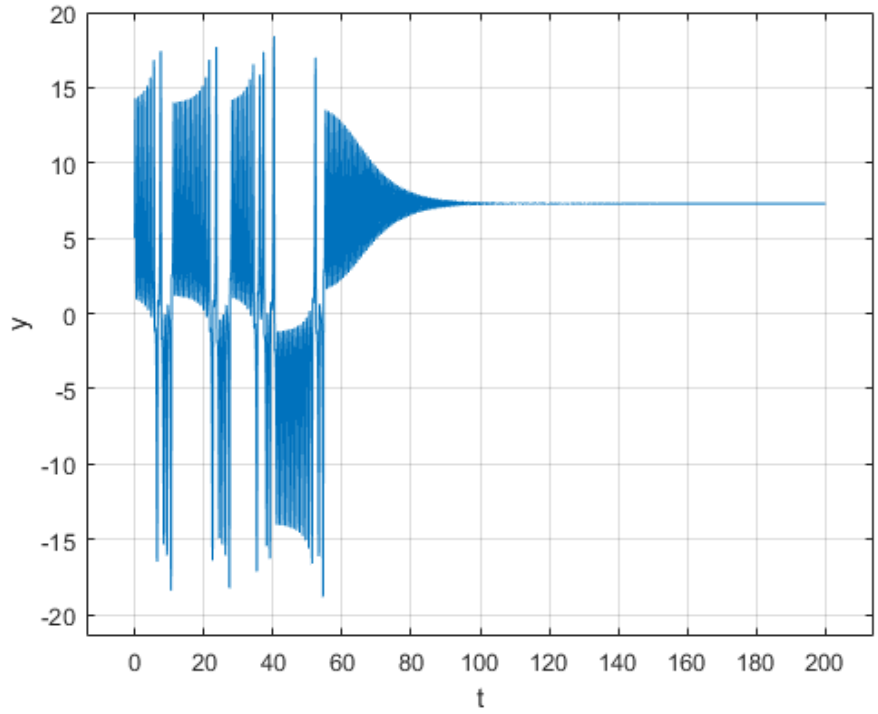
- ***Driver for Q1:***

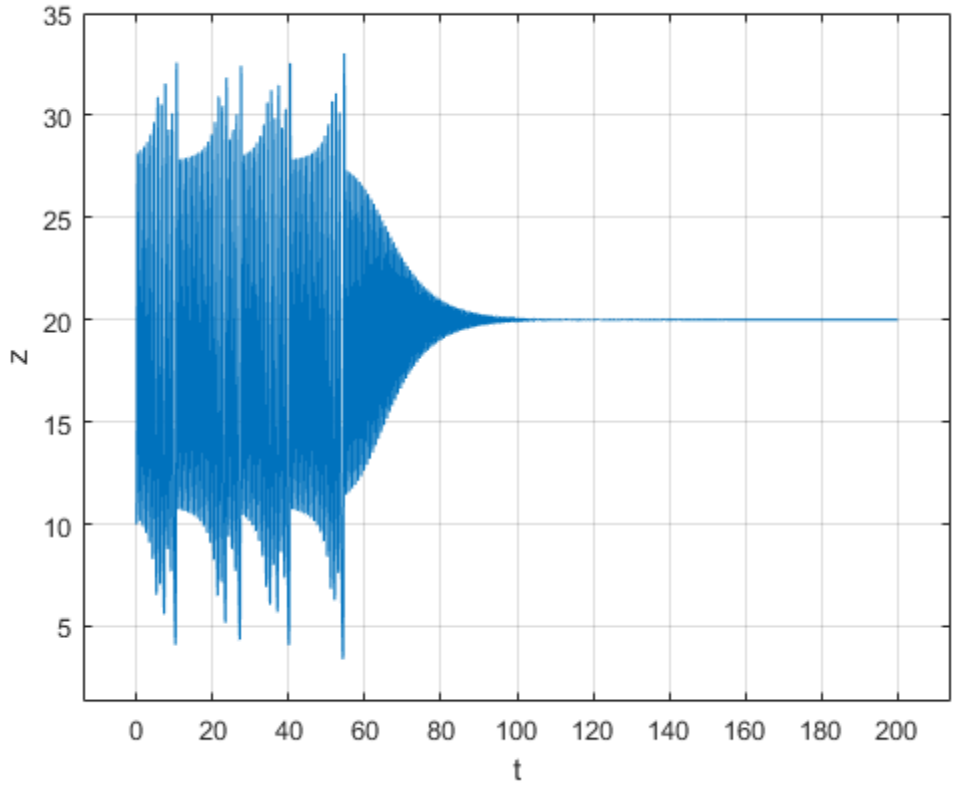
```
% Driver for Q1

close all
clear all
clc

a = 0;
b = 200;
h = 0.01;

x0 = 5;
y0 = 5;
z0 = 10;

[x,y,z] = RK4('lorentz',a,b,h,x0,y0,z0);
```

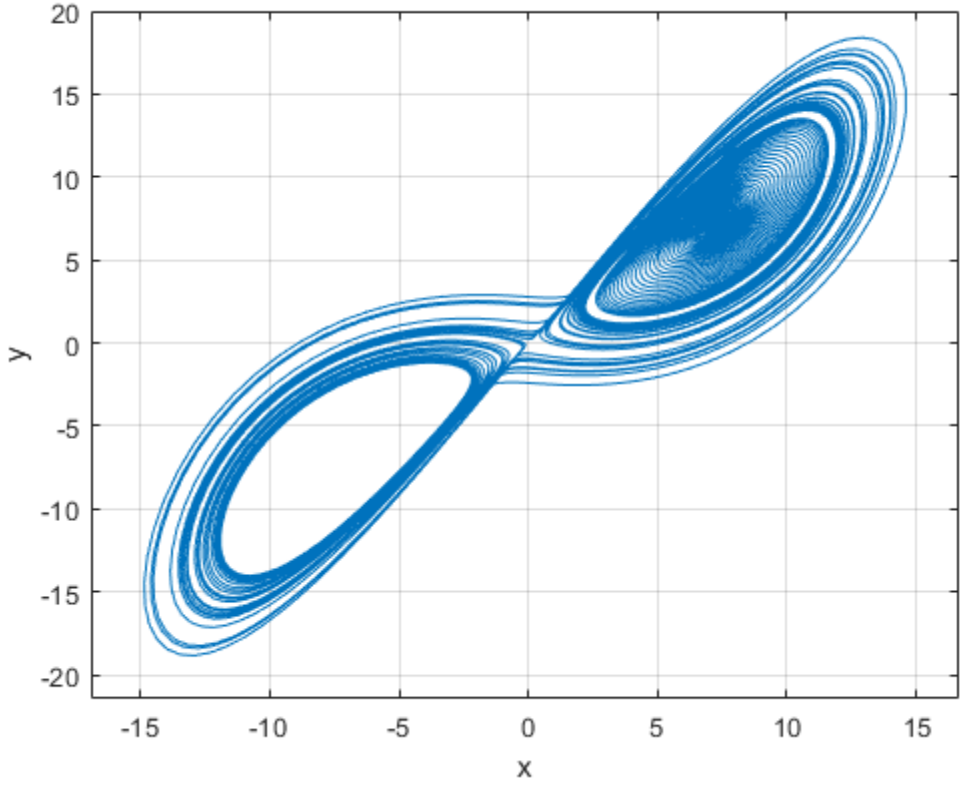
- ***MATLAB output:***

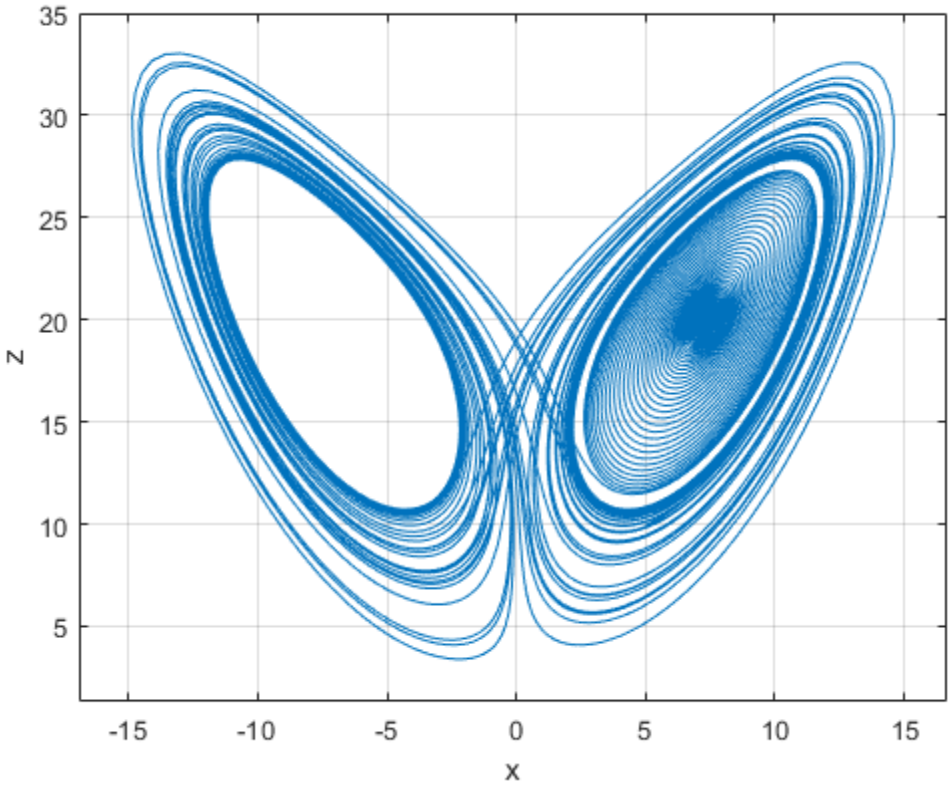
```
Equilibrium point:

   7.302967393526780   7.302967675198756  19.999999541614901
```




35
30
25
20
15
10
5
z
0
20
40
60
80
100
120
140
160
180
200
t


20
15
10
5
0
-5
-10
-15
-20
y
-15
-10
-5
0
5
10
15
x




*Published with MATLAB® R2021a*

**2*.** For this problem, I will ask you to solve the laminar flat plate boundary layer problem and numerically obtain the Blasius solution. The dependent variable $f$ represents the dimensionless stream function, and $\eta = y\sqrt{U/(\nu x)}$ is the similarity variable indicating a dimensionless distance from the wall at a particular $x$ location, given the free stream velocity $U$ and the fluid kinematic viscosity $\nu$.

The nonlinear boundary value problem is defined by

$$\frac{\mathrm{d}^3 f}{\mathrm{d}\eta^3} + \frac{1}{2} f \frac{\mathrm{d}^2 f}{\mathrm{d}\eta^2} = 0,$$

with the boundary conditions

$$f(0) = 0, \qquad \frac{\mathrm{d}f}{\mathrm{d}\eta}(0) = 0, \qquad \frac{\mathrm{d}f}{\mathrm{d}\eta} \to 1 \quad \text{as} \quad \eta \to \infty.$$

The last boundary condition cannot be implemented by a computer so a simplification must be made. It has been found that $\eta = 10$ is sufficient to represent infinity. Specifically, if you double the size of the domain by setting the boundary condition to be at $\eta = 20$, the results are essentially unchanged. Therefore, the boundary condition you will employ in the problem is $f'(10) = 1$.

Follow the steps in the notes to rewrite the equation as a system of equations. Solve the system using the Shooting Method. This involves the use of a nonlinear equation solver to iterate the guess for $f''(0)$, and the secant method is recommended for this problem.

Plot $f'$, the normalized streamwise velocity, as a function of $\eta$ up to $\eta = 10$.

Numerically evaluate $f''(0) = 0.332057$ to six significant digits. This quantity is an important coefficient that becomes part of the equation for the skin friction coefficient, which is used to calculate the drag force in external flow problems.

## MATLAB code:

- ***Script for Runge-Kutta Method:***

```
% Runge Kutta function
function [x,f2] = RK4(ODE,a,b,h,f1_0,f2_0,f3_0)

n = (b-a)/h;
x = zeros(1,n+1);
f1 = zeros(1,n+1);
f2 = zeros(1,n+1);
f3 = zeros(1,n+1);
K = zeros(3,4);

x(1) = a;
```

```
f1(1) = f1_0;
f2(1) = f2_0;
f3(1) = f3_0;

for i = 1:n
    x(i+1) = x(i) + h;
    [K(:,1)] = feval(ODE,x(i),f1(i),f2(i),f3(i));
    [K(:,2)] =
feval(ODE,x(i)+0.5*h,f1(i)+0.5*K(1,1)*h,f2(i)+0.5*K(2,1)*h,f3(i)+0.5*K(3,1)*h
);
    [K(:,3)] =
feval(ODE,x(i)+0.5*h,f1(i)+0.5*K(1,2)*h,f2(i)+0.5*K(2,2)*h,f3(i)+0.5*K(3,2)*h
);
    [K(:,4)] =
feval(ODE,x(i)+h,f1(i)+K(1,3)*h,f2(i)+K(2,3)*h,f3(i)+K(3,3)*h);

    f1(i+1) = f1(i) + 1/6*(K(1,1)+2*K(1,2)+2*K(1,3)+K(1,4))*h;
    f2(i+1) = f2(i) + 1/6*(K(2,1)+2*K(2,2)+2*K(2,3)+K(2,4))*h;
    f3(i+1) = f3(i) + 1/6*(K(3,1)+2*K(3,2)+2*K(3,3)+K(3,4))*h;
end
```

- ***Script for Blasius method:***

```
% Implementing Blasius method
function fp = blasius(x,f1,f2,f3)

fp(1) = f2;
fp(2) = f3;
fp(3) = -0.5*f1*f3;
```

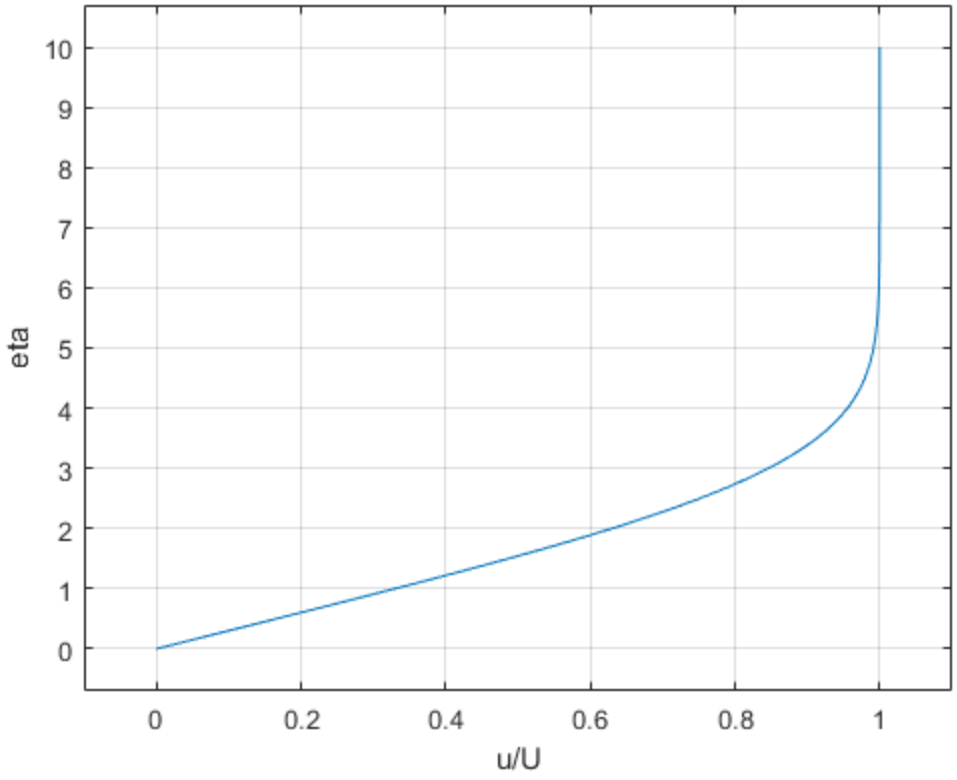
- ***Driver to run the above functions:***

```
% Driver for Q2

close all
clear all
clc

format long

a = 0;
b = 10;
h = 0.1;
n = (b-a)/h;

f1_0 = 0;
f2_0 = 0;
B = 1;
```

```
tolerance = 10^(-6);
xf30(1) = 0;
xf30(2) = 2;

[x,f2] = RK4('blasius',a,b,h,f1_0,f2_0,xf30(1));
yf210(1) = f2(n+1);

[x,f2] = RK4('blasius',a,b,h,f1_0,f2_0,xf30(2));
yf210(2) = f2(n+1);

error = abs(yf210(2) - B);
i = 3;

while(error>tolerance)
    xf30(i) = xf30(i-1) + (1-yf210(i-1))*(xf30(i-1)-xf30(i-2))/(yf210(i-1)-
yf210(i-2));
    [x,f2]=RK4('blasius',a,b,h,f1_0,f2_0,xf30(i));
    yf210(i) = f2(n+1);

    error = abs(yf210(i)-B);
    i = i+1;
end

fprintf('Error:\n')
disp(error)

fpp_0 = xf30(i-1);
fprintf('fpp_0: \n')
disp(fpp_0)

plot(f2,x)
xlabel('u/U')
ylabel('eta')
axis padded
grid on
```

- ***MATLAB output:***

```
Error:
    3.604962106606990e-09

fpp_0:
   0.332057347406128
```




*Published with MATLAB® R2021a*