MATHEMATICAL MODELLING AND SIMULATION

MSMA- 213

2023- 2024

LAB FILE

Submitted to:

Dr. Nilam
Mr. Abhay Srivastava
Department of Applied Mathematics

Submitted by:

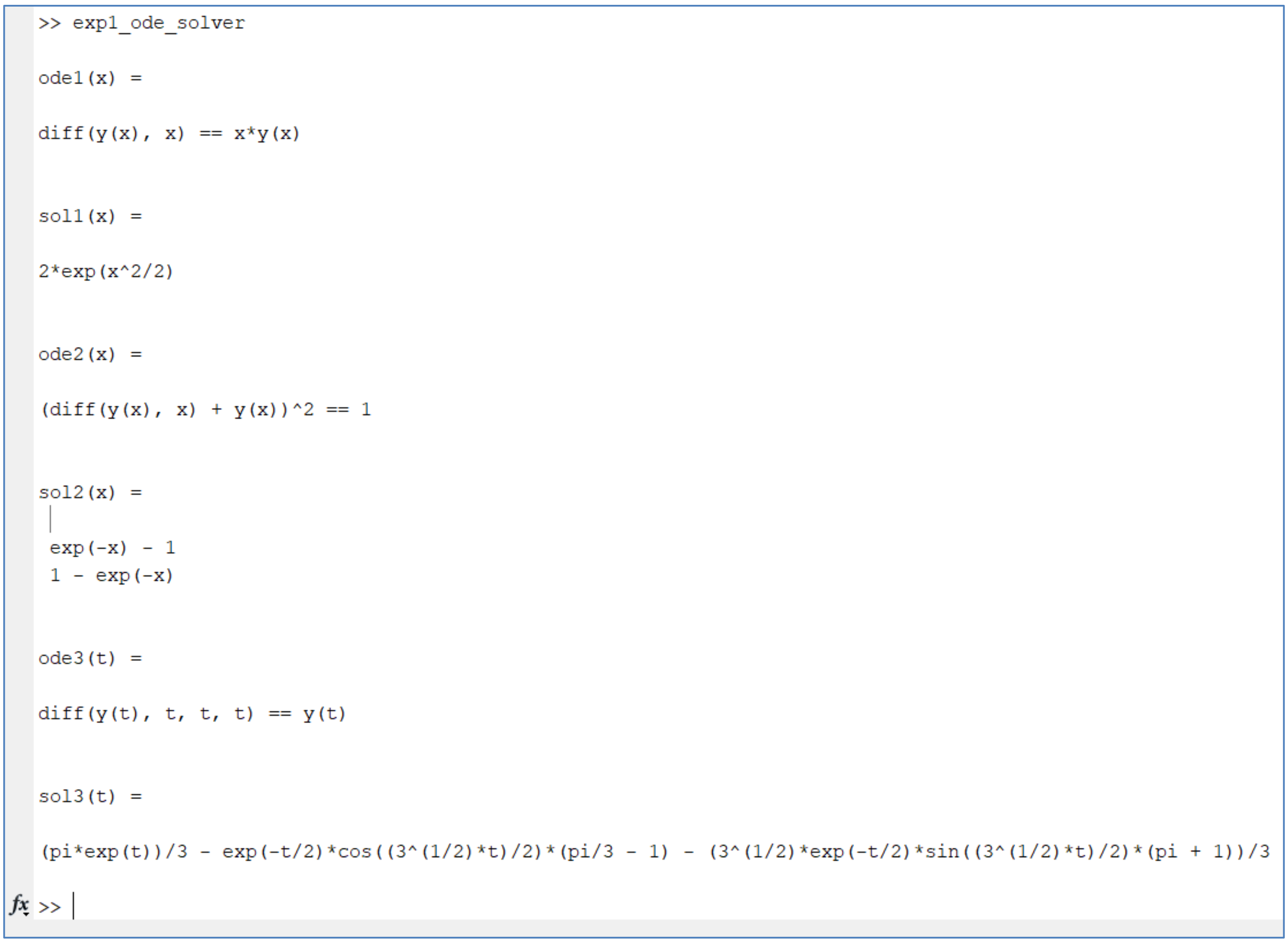
Ritika Gupta
2K22/MSCMAT/54

INDEX

PRACTICAL 1

Solving ODE with initial condition

```
% 1. First order linear ode with initial condition
% y' + P(x)y = Q(x)

% creating symbolic func y(x)
syms y(x)
% defining ode
ode1 = diff(y,x) == x*y
% defining initial condition
cond = y(0) == 2;
% solving ode
y(x) = dsolve(ode1,cond);
sol1 = simplify(y)

% 2. Non linear ODE with initial cond
syms y(x)
ode2 = (diff(y,x)+y)^2 == 1
cond = y(0) == 0;
y(x) = dsolve(ode2,cond);
sol2 = simplify(y)

% 3. Higher order ode with initial cond
syms y(t)
ode3 = diff(y,t,3) == y
% declaring 1st and 2nd derivatives to later define initial conditions
D1 = diff(y,t);
D2 = diff(y,t,2);
% defining initial conditions
cond0 = y(0) == 1;
cond1 = D1(0) == -1;
cond2 = D2(0) == pi;
% solving
y(t) = dsolve(ode3,[cond0,cond1,cond2]);
sol3 = simplify(y)
```

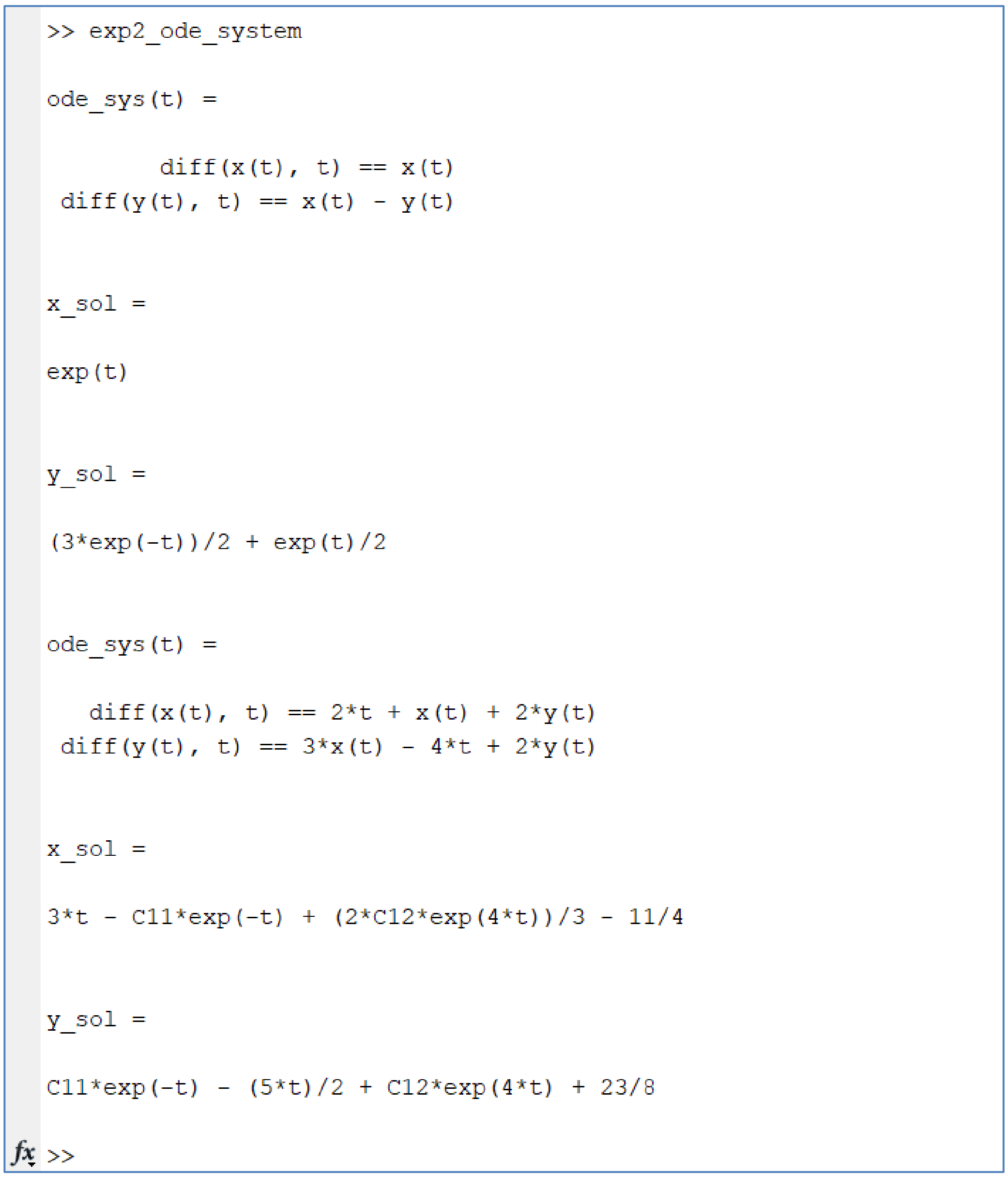


```
>> exp1_ode_solver

ode1(x) =

diff(y(x), x) == x*y(x)


sol1(x) =

2*exp(x^2/2)


ode2(x) =

(diff(y(x), x) + y(x))^2 == 1


sol2(x) =

 exp(-x) - 1
 1 - exp(-x)


ode3(t) =

diff(y(t), t, t, t) == y(t)


sol3(t) =

(pi*exp(t))/3 - exp(-t/2)*cos((3^(1/2)*t)/2)*(pi/3 - 1) - (3^(1/2)*exp(-t/2)*sin((3^(1/2)*t)/2)*(pi + 1))/3

>> 
```

PRACTICAL 2

Solving system of ODE(s)

```
syms x(t) y(t)

% QUEST 1

% first ODE with IC
ode1 = diff(x,t) == x;
cond1 = x(0) == 1;
% second ODE with IC
ode2 = diff(y,t) == x - y;
cond2 = y(0) == 2;
% system of ODEs with ICs
ode_sys = [ode1 ; ode2]
cond_sys = [cond1,cond2];
% solved system
sol_sys = dsolve(ode_sys,cond_sys);
x_sol = sol_sys.x
y_sol = sol_sys.y

% QUEST 2 (matrix form)
syms x(t) y(t)
A = [1 2 ; 3 2];
B = [2*t ; -4*t];
X = [x ; y];
ode_sys = diff(X,t) == A*X + B
sol_sys = dsolve(ode_sys);
x_sol = simplify(sol_sys.x)
y_sol = simplify(sol_sys.y)
```

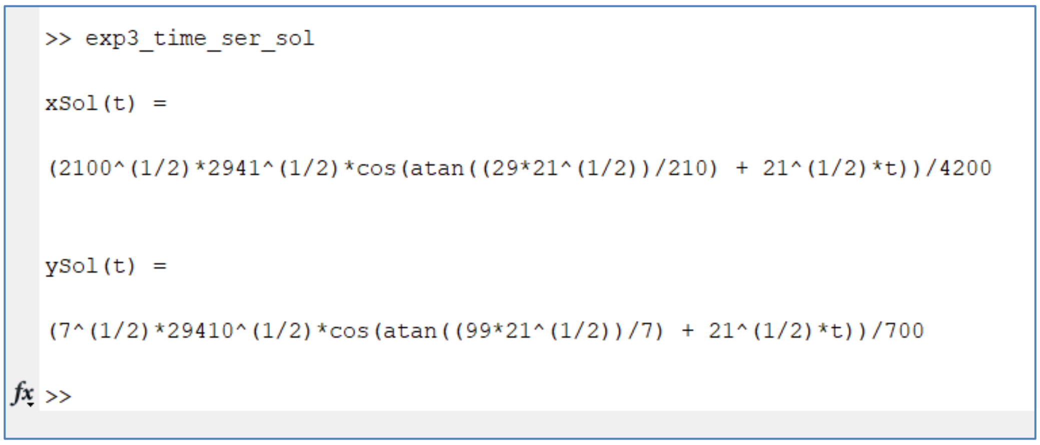


```
>> exp2_ode_system

ode_sys(t) =

        diff(x(t), t) == x(t)
 diff(y(t), t) == x(t) - y(t)


x_sol =

exp(t)


y_sol =

(3*exp(-t))/2 + exp(t)/2


ode_sys(t) =

   diff(x(t), t) == 2*t + x(t) + 2*y(t)
 diff(y(t), t) == 3*x(t) - 4*t + 2*y(t)


x_sol =

3*t - C11*exp(-t) + (2*C12*exp(4*t))/3 - 11/4


y_sol =

C11*exp(-t) - (5*t)/2 + C12*exp(4*t) + 23/8

>>
```

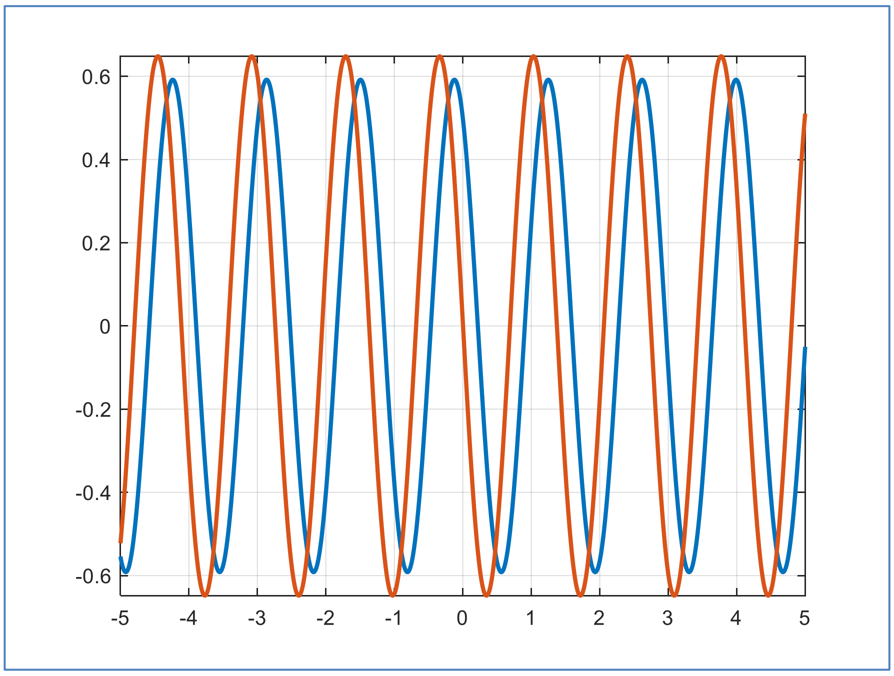
PRACTICAL 3

Plot the time series solution of a system of differential equations

```
% x'(t) = -3x + 5y
% y'(t) = -6x + 3y
% with initial conditions x(0)=0.5 , y(0)=0.01

syms x(t) y(t)
ode1 = diff(x) == -3*x + 5*y;
ode2 = diff(y) == -6*x + 3*y;

ode_sys = [ode1 ; ode2];
cond_sys = [x(0)==0.5,y(0)==0.01];

sol_sys = dsolve(ode_sys,cond_sys);
xSol(t) = sol_sys.x
ySol(t) = sol_sys.y

fplot(xSol,'Linewidth',2)
grid on
hold on
fplot(ySol,'Linewidth',2)
hold off
```



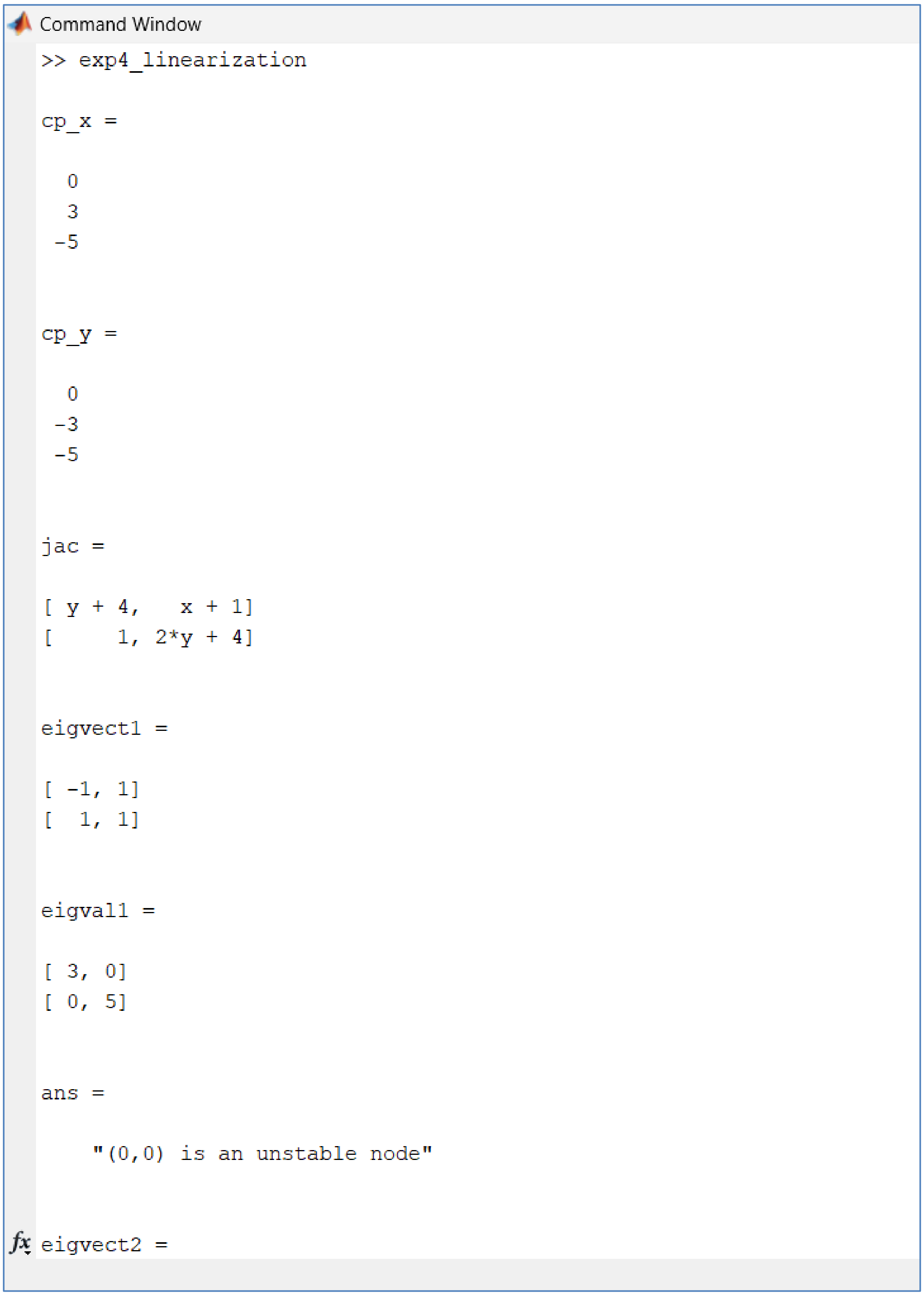
```
>> exp3_time_ser_sol

xSol(t) =

(2100^(1/2)*2941^(1/2)*cos(atan((29*21^(1/2))/210) + 21^(1/2)*t))/4200


ySol(t) =

(7^(1/2)*29410^(1/2)*cos(atan((99*21^(1/2))/7) + 21^(1/2)*t))/700

>>
```



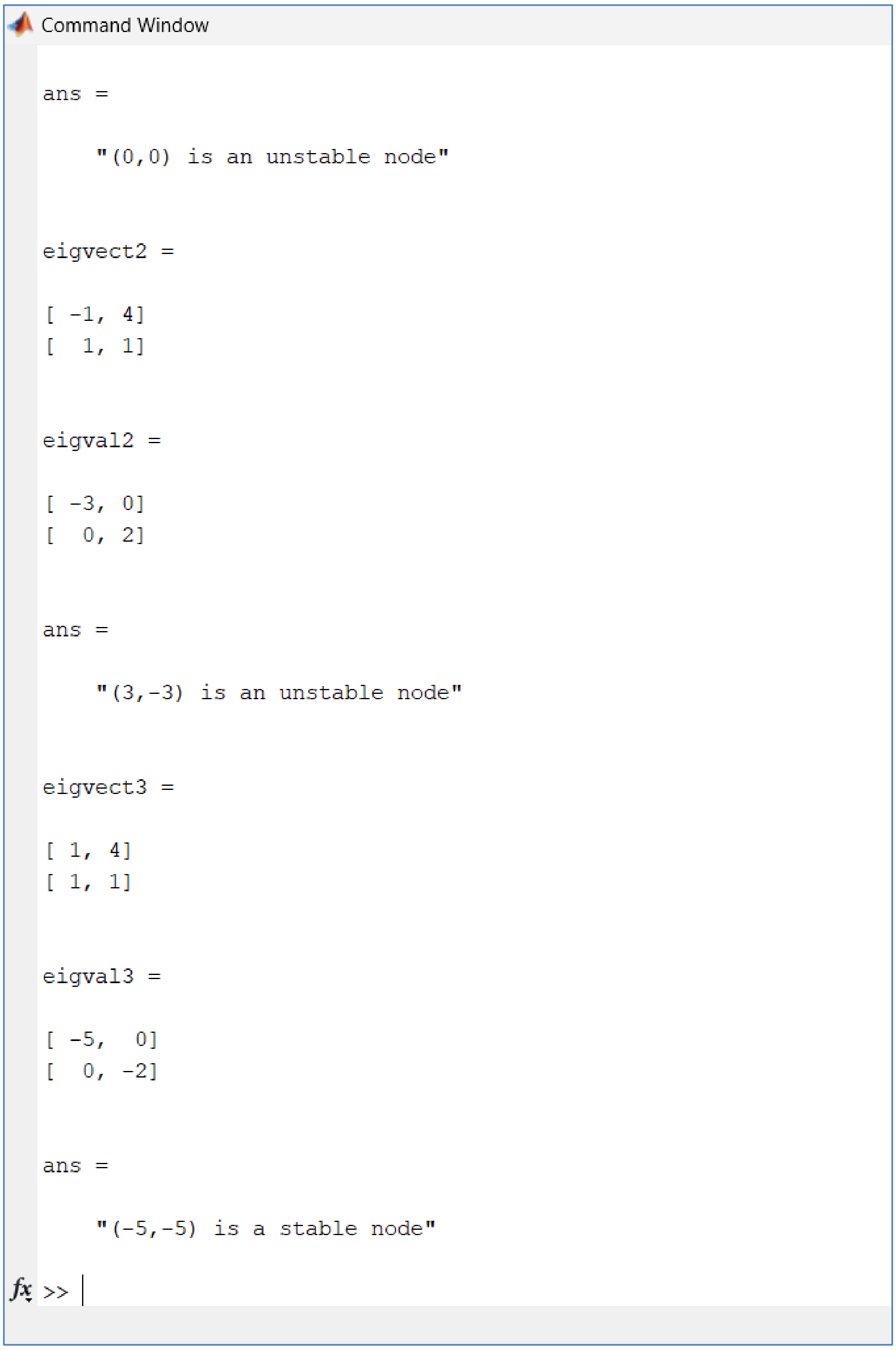
PRACTICAL 4

Find critical points, Jacobian matrix, eigenvalues and eigenvectors of a system of differential equations for stability analysis

```
%x'(t) = 4x + y + xy
%y'(t) = x + 4y + y^2

syms x y t

dydt = @(t,s) [(4*s(1) + s(2) + s(1)*s(2));
    (s(1) + 4*s(2) + s(2).^2)];
sys = dydt(t,[x;y]);

% critical points (x_cp,y_cp)
[cp_x,cp_y] = solve(sys(1)==0,sys(2)==0,x,y)

% jacobian matrix
jac = jacobian(sys,[x;y])

% eigenvalues of jacobian wrt first critical point
jac1 = subs(jac,{x,y},{cp_x(1),cp_y(1)});
[eigvect1,eigval1] = eig(jac1)
stability( eigval1(1,1),eigval1(2,2),cp_x(1),cp_y(1))

% eigenvalues of jacobian wrt second critical point
jac2 = subs(jac,{x,y},{cp_x(2),cp_y(2)});
[eigvect2,eigval2] = eig(jac2)
stability( eigval2(1,1),eigval2(2,2),cp_x(2),cp_y(2))

% eigenvalues of jacobian wrt third critical point
jac3 = subs(jac,{x,y},{cp_x(3),cp_y(3)});
[eigvect3,eigval3] = eig(jac3)
stability( eigval3(1,1),eigval3(2,2),cp_x(3),cp_y(3))

% function to decide stability from eigenvalues of equilibrium
% if both eigenvalues are negative, the node is stable else unstable
function node = stability(eig1,eig2,cpx,cpy)
    if eig1 < 0 & eig2 < 0
        node = strcat('(',string(cpx),',',string(cpy),')',' is a stable node');
    else
        node = strcat('(',string(cpx),',',string(cpy),')',' is an unstable node');
    end
end
```



```
>> exp4_linearization

cp_x =

   0
   3
  -5


cp_y =

   0
  -3
  -5


jac =

[ y + 4,    x + 1]
[     1, 2*y + 4]


eigvect1 =

[ -1, 1]
[  1, 1]


eigval1 =

[ 3, 0]
[ 0, 5]


ans =

    "(0,0) is an unstable node"


eigvect2 =
```

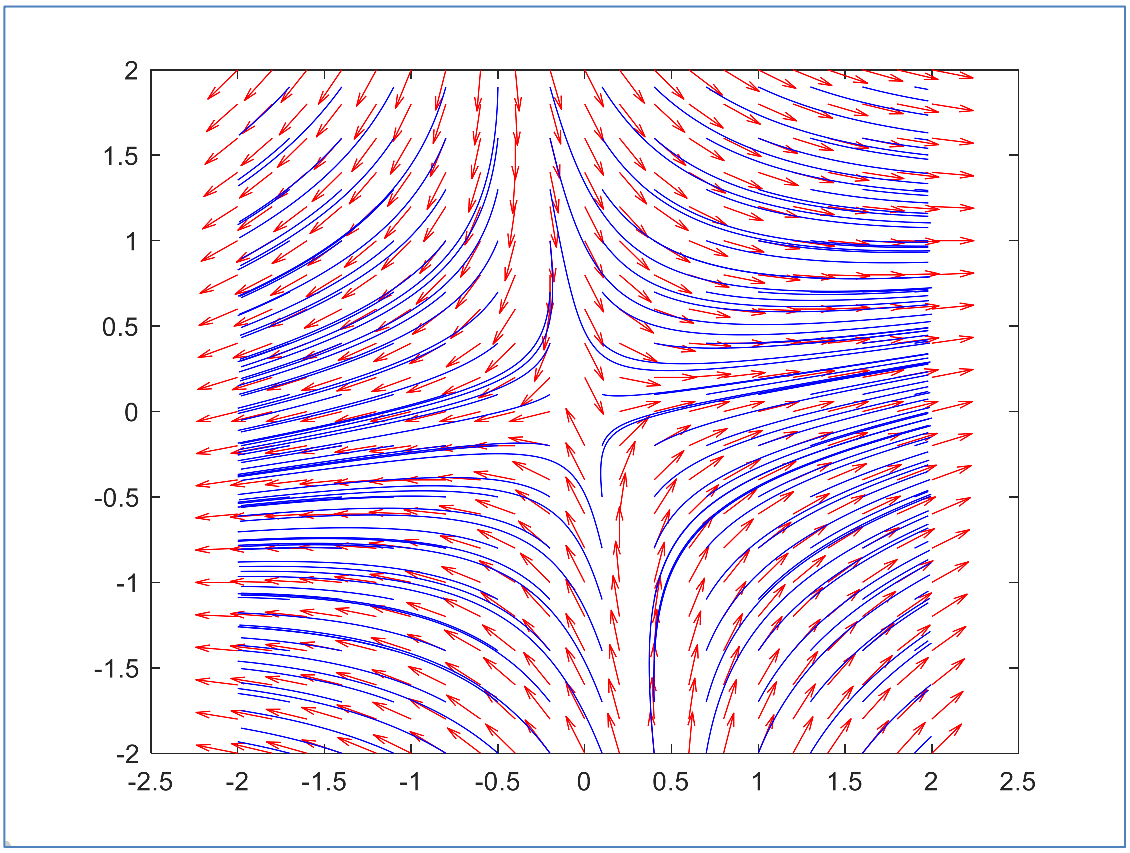

Command Window

```
ans =

    "(0,0) is an unstable node"


eigvect2 =

[ -1, 4]
[  1, 1]


eigval2 =

[ -3, 0]
[  0, 2]


ans =

    "(3,-3) is an unstable node"


eigvect3 =

[ 1, 4]
[ 1, 1]


eigval3 =

[ -5,  0]
[  0, -2]


ans =

    "(-5,-5) is a stable node"

>> 
```

PRACTICAL 5

WAP to draw direction fields for a system of differential equations

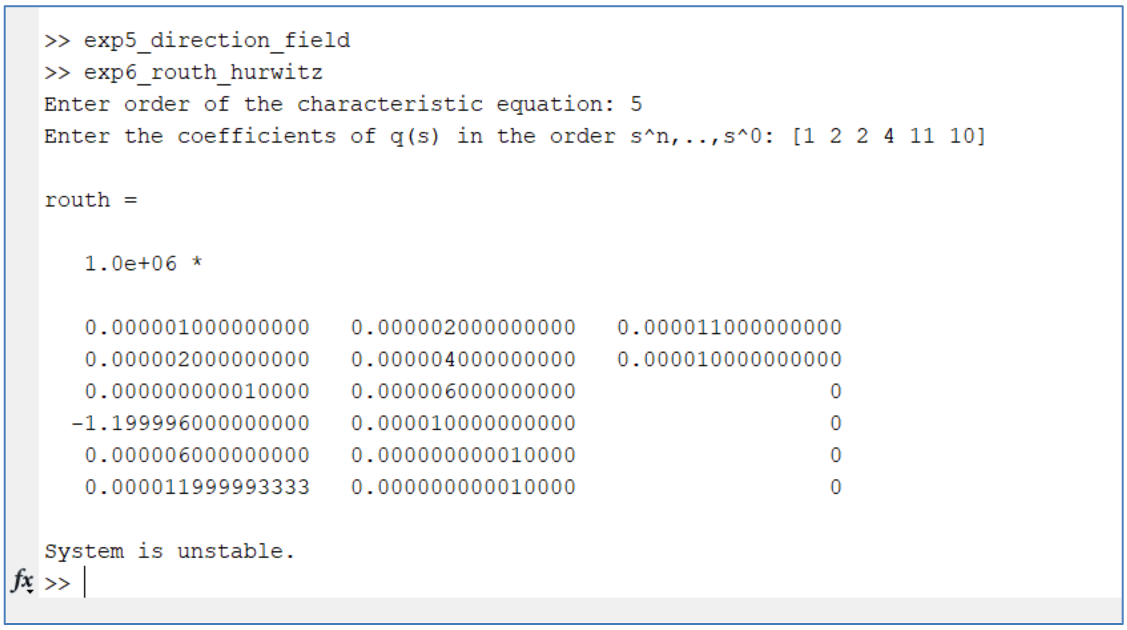
```
% x'(t) = 4x + y
% y'(t) = x - 2y

[x, y] = meshgrid(-2:0.2:2);

% define ODEs
dx = 4*x + y;
dy = x - 2*y;

% normalize vectors
norm_factor = sqrt(dx.^2 + dy.^2);
dy_norm = dy./norm_factor;
dx_norm = dx./norm_factor;

% plotting the direction field
quiver(x, y, dx_norm, dy_norm,'r');

hold on
[X,Y] = meshgrid(-2:0.3:2);
streamline(x,y,dx,dy,X,Y)
hold off
```



PRACTICAL 6

WAP to determine the stability using the Routh Hurwitz criterion

```
n = input( 'Enter order of the characteristic equation: ' );
c = input( 'Enter the coefficients of q(s) in the order s^n,..,s^0: ' );

% checking if the first row is positive
% numel gives number of elements
% size(c,2) gives number of columns in c
if( numel(find(c>0)) == size(c,2) )
    routh = zeros (n+1, ceil((n+1)/2));
    % if number of coeffs is odd, add a 0 coeff
    if (mod(numel(c),2) == 1)
        c = [c 0];
    end
    % parting c into 2 columns for odd and even indexed coeffs
    c2 = reshape(c,2,[]);
    routh(1:2,:) = c2;
    for i = 3:size(routh,1)
        for j = 1:size(routh,2)-1
            routh(i,j) = (routh(i-1,1)*routh(i-2,j+1) - routh(i-2,1)*routh(i-1,j+1)) ↙
/ routh(i-1,1);
            % element can not be 0
            if (routh(i,j) == 0)
                routh(i,j) = 0.00001;
            end
        end
    end

    routh

    if (numel(find(routh(:,1)>0)) == n+1)
        disp('System is stable.')
    else
        disp('System is unstable.')
    end

else
    disp('Order and number of coeff(s) entered mismatch.')
end
```

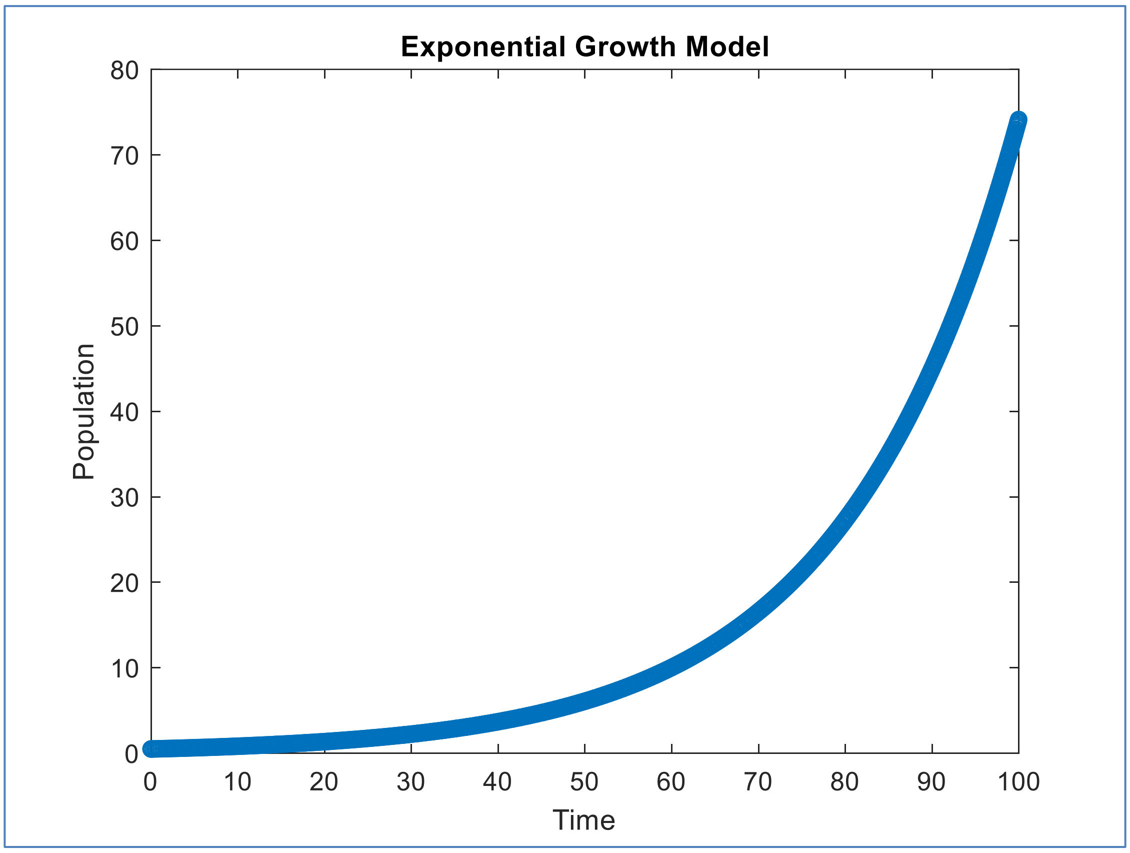


```
>> exp5_direction_field
>> exp6_routh_hurwitz
Enter order of the characteristic equation: 5
Enter the coefficients of q(s) in the order s^n,..,s^0: [1 2 2 4 11 10]

routh =

   1.0e+06 *

   0.000001000000000   0.000002000000000   0.000011000000000
   0.000002000000000   0.000004000000000   0.000010000000000
   0.000000000010000   0.000006000000000                   0
  -1.199996000000000   0.000010000000000                   0
   0.000006000000000   0.000000000010000                   0
   0.000011999993333   0.000000000010000                   0

System is unstable.
>>
```

PRACTICAL 7

WAP to simulate the exponential growth population model

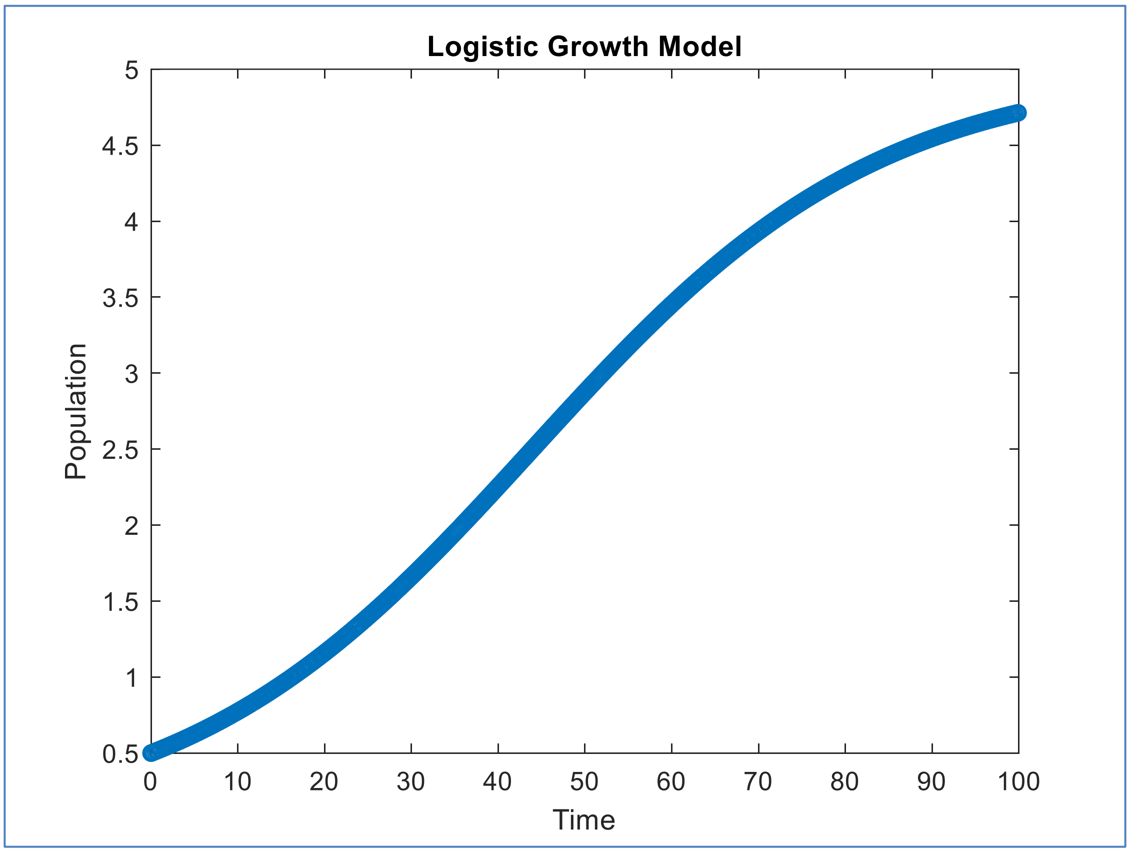
```
r = 0.05 ; N0 = 0.5;
tspan = 0:0.01:100;

ode45(@(t,N) [(r*N)] , tspan , N0);

title('Exponential Growth Model')
xlabel('Time')
ylabel('Population')
```



PRACTICAL 8

WAP to simulate the logistic growth population model

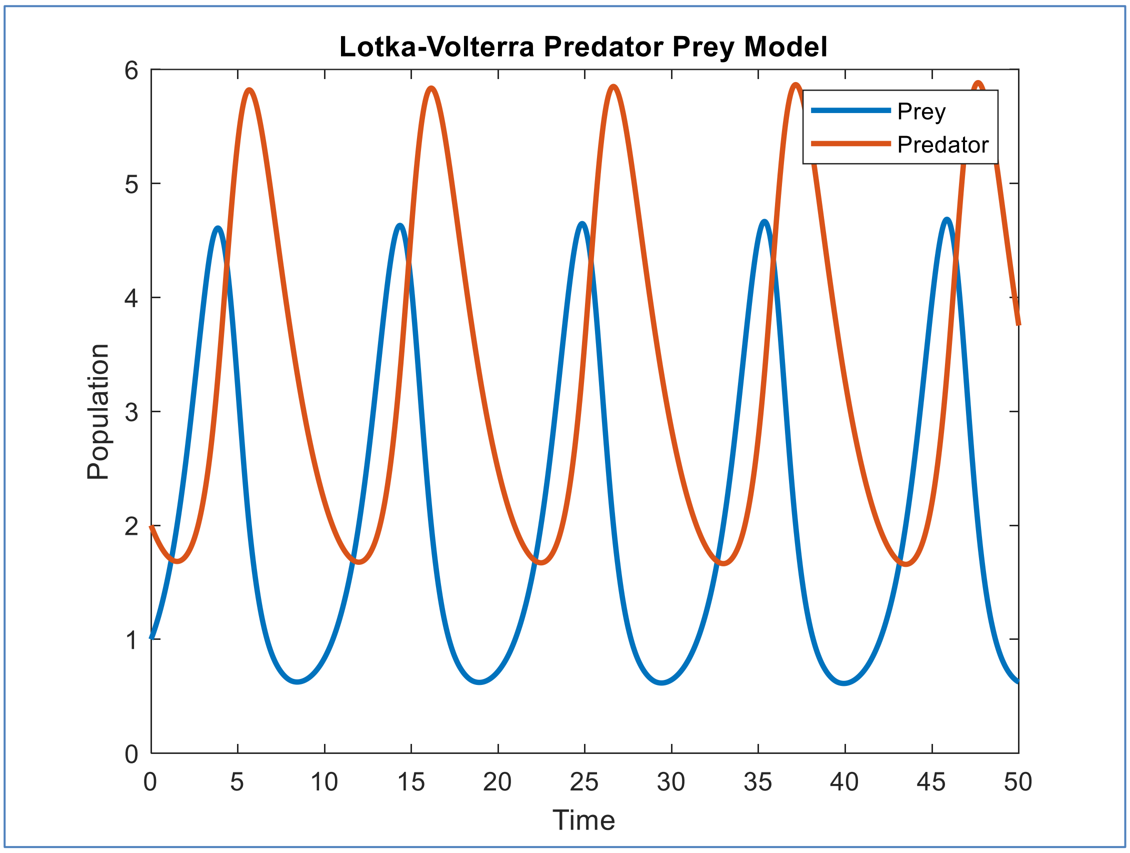
```
r = 0.05 ; K = 5 ; N0 = 0.5;
tspan = 0:0.01:100;

ode45(@(t,N) r*N*(1 - N/K) , tspan , N0);

title('Logistic Growth Model')
xlabel('Time')
ylabel('Population')
```



PRACTICAL 9

WAP to draw the trajectories of a predator prey model

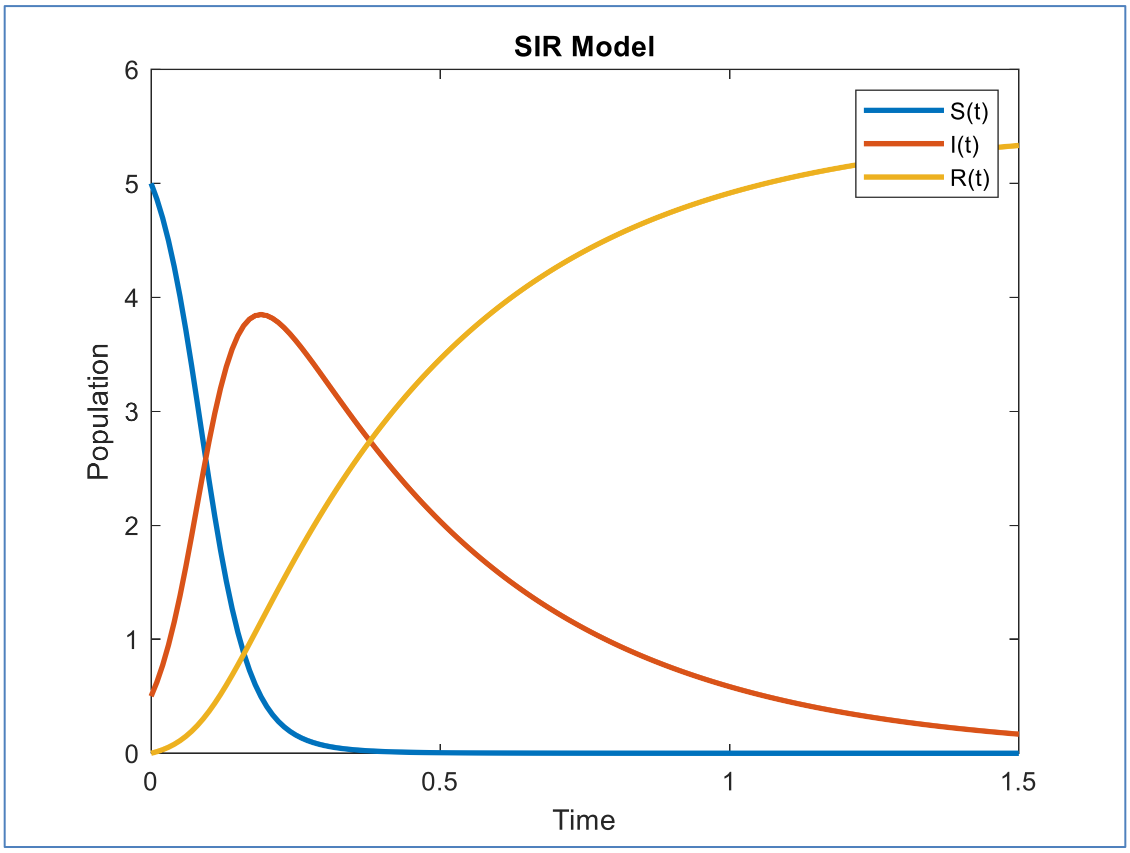
```
b = 1; c1 = 0.3; c2 = 0.2; a = 0.4;
Y0 = [1;2]

tspan = 0:0.01:50;

dydt = @(t,y) [(b*y(1) - c1*y(1)*y(2));
     (c2*y(1)*y(2) - a*y(2))];

[y,t] = ode45(dydt,tspan,Y0);
plot(y,t,'linewidth',2)

legend('Prey','Predator')
title('Lotka-Volterra Predator Prey Model')
xlabel('Time')
ylabel('Population')
```



PRACTICAL 10

WAP to simulate the SIR model

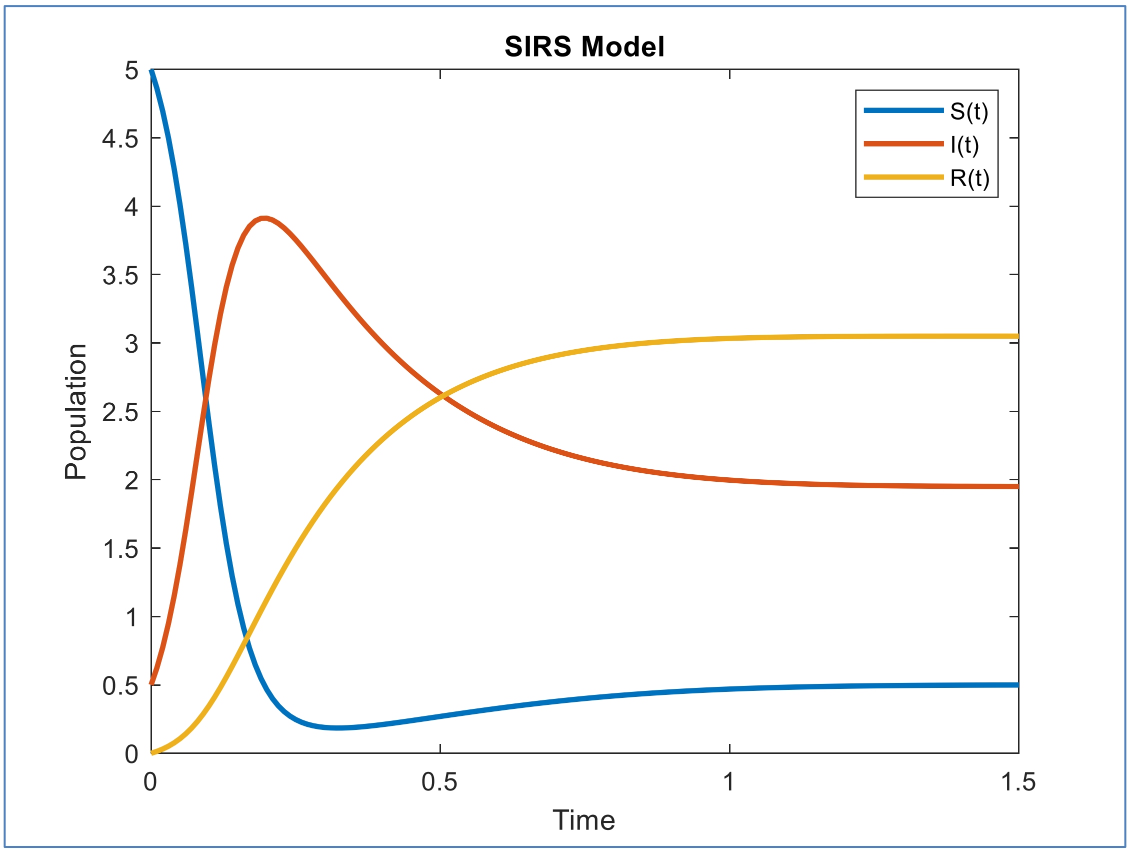
```
b = 5; a = 2.5;
S0=5;I0=0.5;R0=0;

tspan = 0:0.01:1.5;

dydt = @(t,y) [(-b*y(1)*y(2));
     (b*y(1)*y(2) - a*y(2));
     (a*y(2))];

[y,t] = ode45(dydt,tspan,[S0 I0 R0]);
plot(y,t,'linewidth',2)

legend('S(t)','I(t)','R(t)')
title('SIR Model')
xlabel('Time')
ylabel('Population')
```



PRACTICAL 11

WAP to simulate the SIRS model

```
b = 5; a = 2.5; c = 1.6
S0=5;I0=0.5;R0=0;

tspan = 0:0.01:1.5;

dydt = @(t,y) [(-b*y(1)*y(2) + c*y(3));
     (b*y(1)*y(2) - a*y(2));
     (a*y(2) - c*y(3))];

[y,t] = ode45(dydt,tspan,[S0 I0 R0]);
plot(y,t,'linewidth',2)

legend('S(t)','I(t)','R(t)')
title('SIRS Model')
xlabel('Time')
ylabel('Population')
```

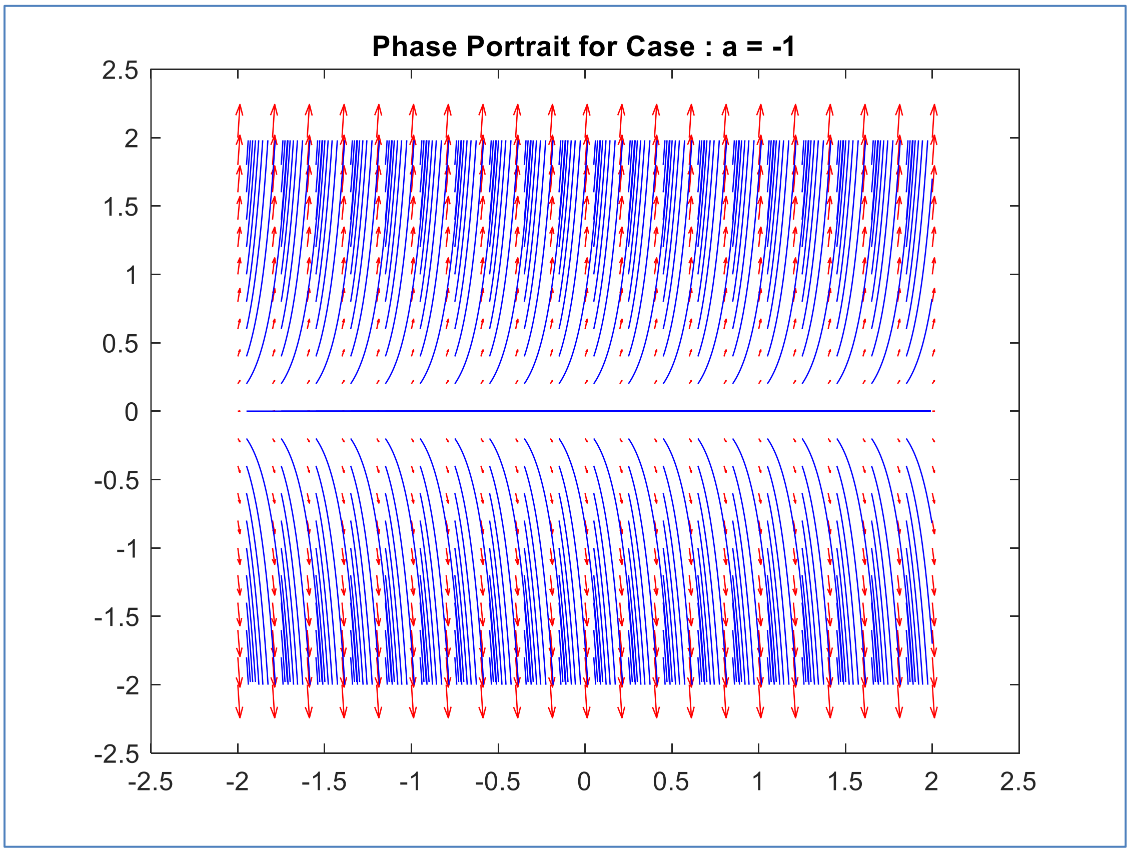


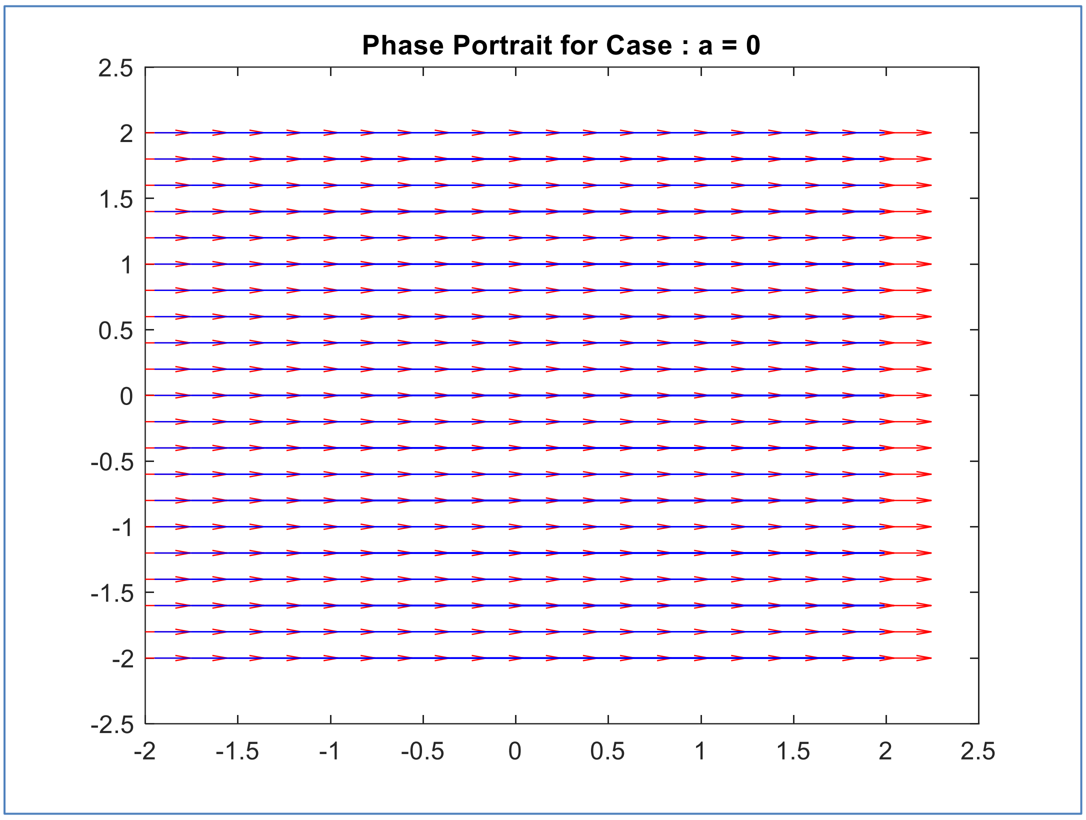
PRACTICAL 12

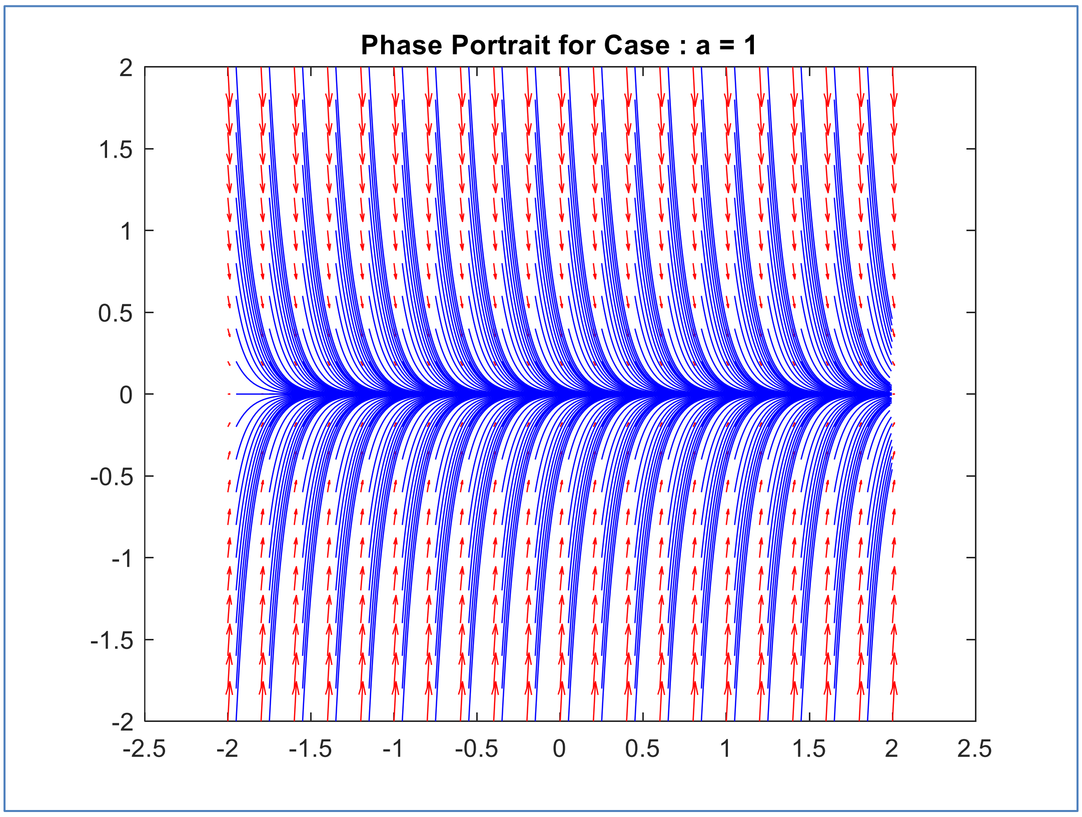
Plot a phase portrait and classify the fixed points of the linear system dx/dt = ax(1-x^2) .

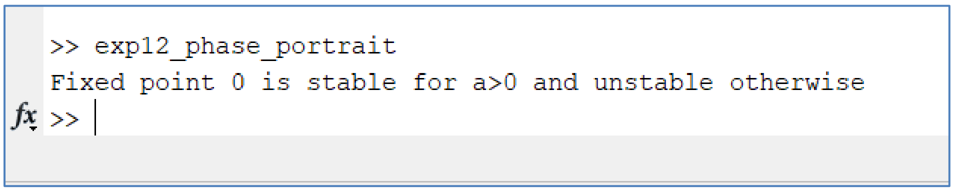
Where, 'a' is positive, negative or zero. Discuss all cases.

```
[t, x] = meshgrid(-2:0.2:2);

for a = -1:1:1
    dx  = a*x*(1-x.^2);
    dt  = ones(size(dx));
    %creates a new figure window
    figure
    %plots direction components
    quiver(t,x,dt,dx,'r')
    title(sprintf('Phase Portrait for Case : a = %d',a))
    hold on
    % returns plotted streamlines
    streamline(t,x,dt,dx,t+0.05,x);
    hold off
end

disp ('Fixed point 0 is stable for a>0 and unstable otherwise')
```









```
>> exp12_phase_portrait
Fixed point 0 is stable for a>0 and unstable otherwise
>>
```