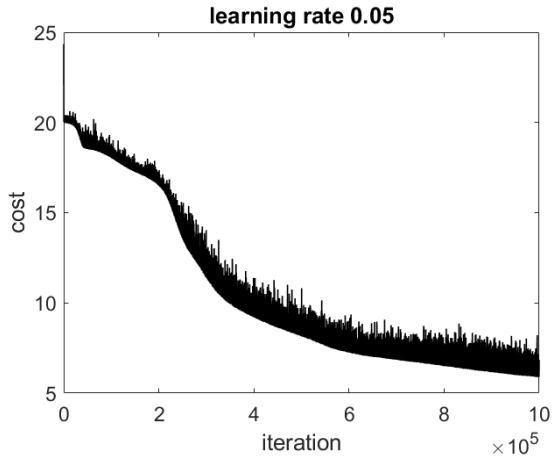
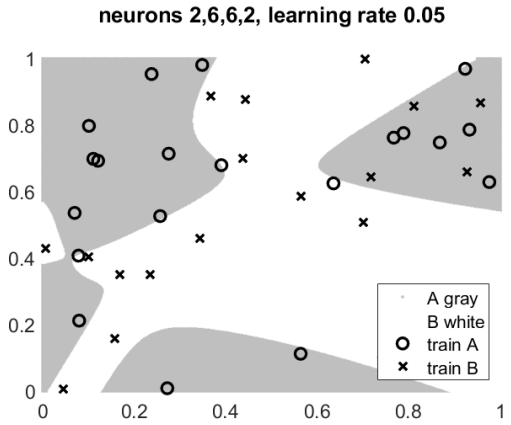
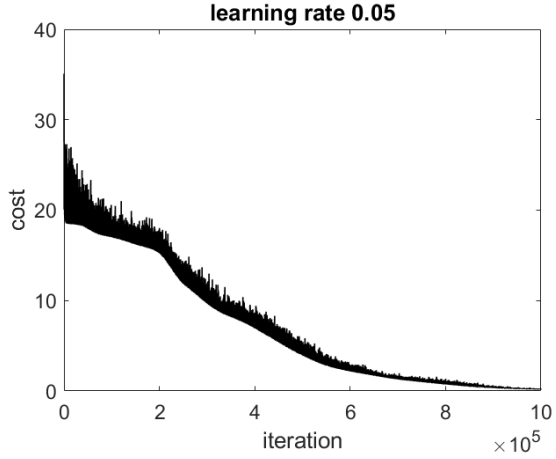
Q1

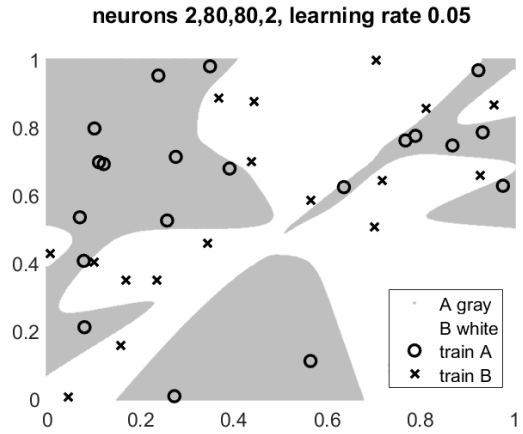
```
neurons=[6,6];
learning_rate=0.05;
niter=1e6;
```



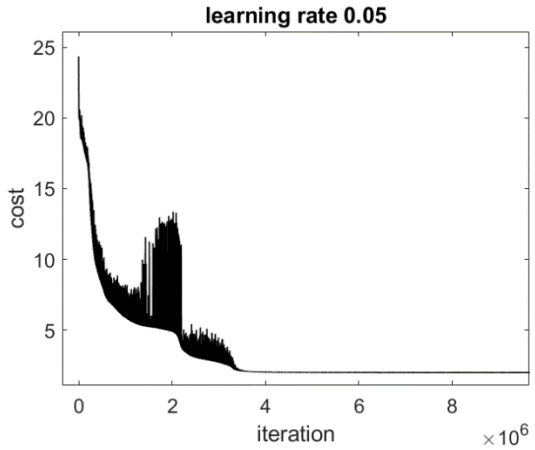


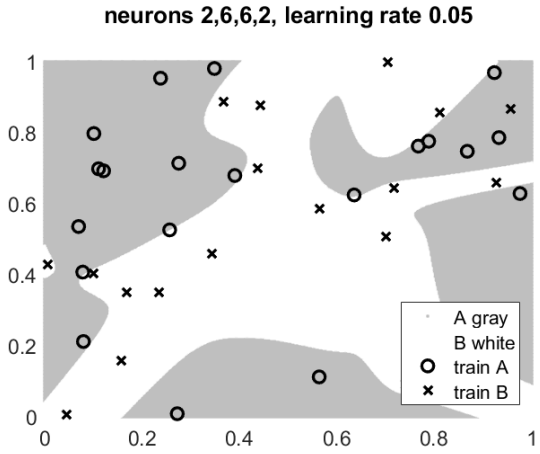

```
neurons=[80,80];
learning_rate=0.05;
niter=1e6;
```



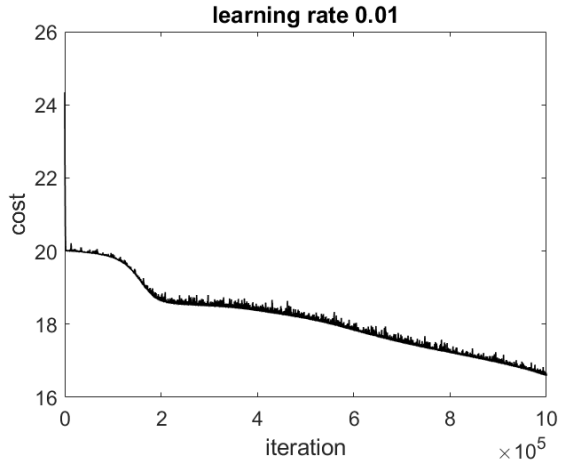


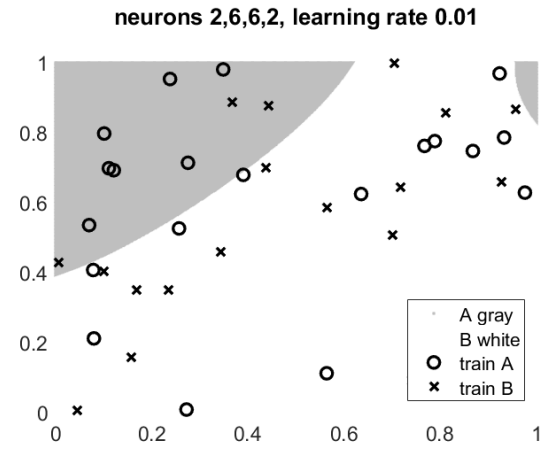

```
neurons=[6,6];
learning_rate=0.01;
niter=1e7;
```






```
neurons=[6,6];
learning_rate=0.01;
niter=1e6;
```





Discussion:

1. More neurons, points are classified more accurate.
2. More iteration, points are classified more accurate.
3. Smaller learning rate, points are classified less accurate and also less fast.
4. Smaller learning rate, less cost.
5. More neurons, more complex mapping.
6. Less learning rate, the cost is more stable.
7. When iteration is getting larger, the cost will constantly getting smaller.

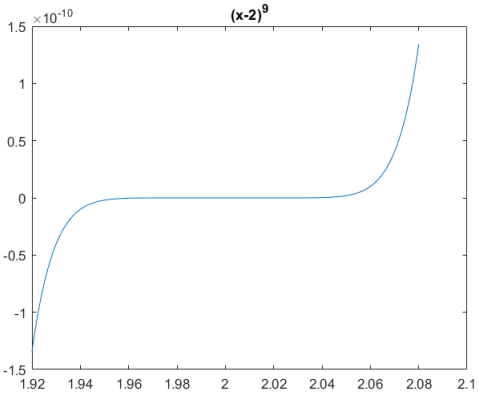
Matlab code

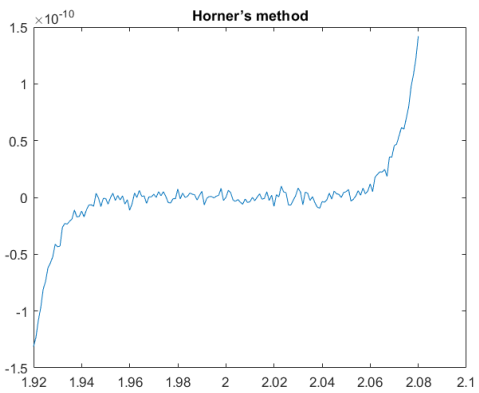
```
function category = classifypoints(file, points)
    load(file);
    for i = 1:length(points)
        x = points(:,i);
        a2 = activate(x,W2,b2);
```

```
        a3 = activate(a2,W3,b3);
        a4 = activate(a3,W4,b4);
        if a4(1,1) >= a4(2,1)
            category(i) = 1;
        else
            category(i) = 0;
        end
    end

end
function cost =netbp2(neurons, data, labels, niter, lr,
file)
%NETBP  Uses backpropagation to train a network
% Initialize weights and biases
rng(5000);
W2 = 0.5*randn(neurons(1),2); W3 =
0.5*randn(neurons(2),neurons(1)); W4 =
0.5*randn(2,neurons(2));
b2 = 0.5*randn(neurons(1),1); b3 = 0.5*randn(neurons(2),1);
b4 = 0.5*randn(2,1);
eta=lr;

savecost = zeros(niter,1); % value of cost function at each
iteration
for counter = 1:niter
    k = randi(length(data));          % choose a training
point at random
    % Forward pass
    x = data(:,k);
    a2 = activate(x,W2,b2);
    a3 = activate(a2,W3,b3);
    a4 = activate(a3,W4,b4);
    % Backward pass
    delta4 = a4.*(1-a4).*(a4-labels(:,k));
    delta3 = a3.*(1-a3).*(W4'*delta4);
    delta2 = a2.*(1-a2).*(W3'*delta3);
    % Gradient step
    W2 = W2 - eta*delta2*x';
    W3 = W3 - eta*delta3*a2';
    W4 = W4 - eta*delta4*a3';
    b2 = b2 - eta*delta2;
    b3 = b3 - eta*delta3;
    b4 = b4 - eta*delta4;
    % Monitor progress
```

```
    newcost = costs(W2,W3,W4,b2,b3,b4) ;  % display cost to
screen
    savecost(counter) = newcost;
    %fprintf("i=%d  %e\n", counter, newcost);
end

% Show decay of cost function
%save costvec;
%semilogy([1:1e4:niter],savecost(1:1e4:niter));

  function costval = costs(W2,W3,W4,b2,b3,b4)
        costvec = zeros(length(data),1);
            for i = 1:length(data)
                x = data(:,i);
                a2 = activate(x,W2,b2);
                a3 = activate(a2,W3,b3);
                a4 = activate(a3,W4,b4);
                costvec(i) = norm(labels(:,i) - a4,2);
            end
        costval = norm(costvec,2)^2;
    end % of nested function
  save(file,'W2','W3','W4','b2','b3','b4');
  cost=savecost;
end
```

Q2





a. Difference between 2 plots: around the real root x=2, for plot of (x-2)^9 is constantly increasing and have a root of 2. But for horners method, the plot is unstable around 0. Because we divide the interval into 161 points, for horner's methods, the value of the function may change between positive and negative in those inconsistent points. But for (x-2)^9 the sign of the function will be unchanged in each side of 2.

b. We can directly get x0=2 in the first division since (1.92+2.08)/2=2

But when we change the range to[1.92,2.07], the matlab code will end at x= 2.0 2.013673529767470, which is 0. 013673529767470 >1e-6, (the max level of iteration in my bisection method is 1000 and it reached there), we can't get a root-2<1e-6 here.

c. for Horner's method, the value of the function may change between positive and negative in different points, that doesn't mean the original function have many roots, so we may find a root using our bisection function due to any sign-change phenomenon when using Horner's method. And when we find the first root in this way. The root may be far away from the true root.

d. For function f= (x.^9-18.*x.^8 + 144.*x.^7-672.*x.^6 + 2016.*x.^5-4032.*x.^4 + 5376.*x.^3-4608.*x.^2 + 2304.*x-512) and we set tolerance in fsolve function to 1e-30
Result is x0=1.900000209876868
Matlab showed:
"Equation solved, solver stalled.

fsolve stopped because the relative size of the current step is less than the value of the step size tolerance squared and the vector of function values is near zero as measured by the value of the function tolerance."
For function f= ((x-2).^9)
Result is x0= 1.920987678169728
Matlab showed
"Equation solved, inaccuracy possible.

The vector of function values is near zero, as measured by the value of the function tolerance. However, the last step was ineffective."

we set tolerance in fsolve function to 1e-6
Result is x0=1.900000000000000
For function f= ((x-2).^9)
Result is x0= 1.900000000000000
and matlab showed
"Equation solved at initial point.

fsolve completed because the vector of function values at the initial point is near zero as measured by the value of the function tolerance, and the problem appears regular as measured by the gradient."

Matlab code is provided both on avenue and the last page of the pdf document

Q3

	a	b	c	d
Newton x	4.999999999999	0.999999999996842 0.0000000000003157933	-0.000000894069671630859375	NaN NaN

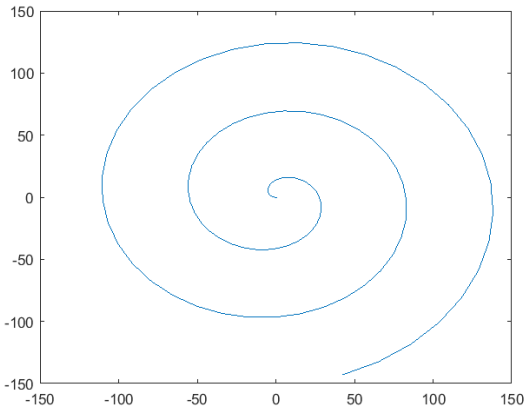
	4.000000000000 0	2.000000000000	0.000000089406967163085937 5 0.000000059604644775390625 0.000000059604644775390625	
Newton iteration	44	58	26	Non
Folve x	11.4128 -0.8968	1.0000 0.0000 2.0000	-0.0027 0.0003 0.0004 0.0004	0.0100 0.0100
Folve feval	4.9490 -4.9490	2.67899658012993e-09 0 5.70210545447480e-13	0 0 1.96108072167687e-08 3.00049276159312e-05	0.0097 0.9799
Fsolve exit reason	-2 No solution found. fsolve stopped because the last step was ineffective. However, the vector of function values is not near zero, as measured by the value of the function tolerance.	1 Equation solved. fsolve completed because the vector of function values is near zero as measured by the value of the function tolerance, and the problem appears regular as measured by the gradient.	1 Equation solved. fsolve completed because the vector of function values is near zero as measured by the value of the function tolerance, and the problem appears regular as measured by the gradient.	-2 No solution found. fsolve stopped because the problem appears regular as measured by the gradient, but the vector of function values is not near zero as measured by the value of the function tolerance.

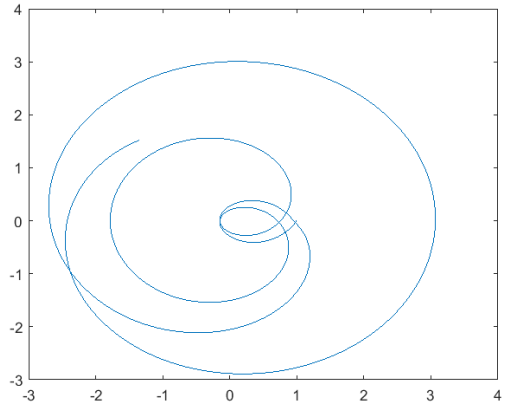
Disscussion	New ton method give the true root, but fsolve stopped because the last step was ineffective	Both newton and fsolve solve it successfully	Both newton and fsolve solve it successfully(the true solution needs to be near 0)	Both newton and fsolve failed since the original equation system do not have a root.

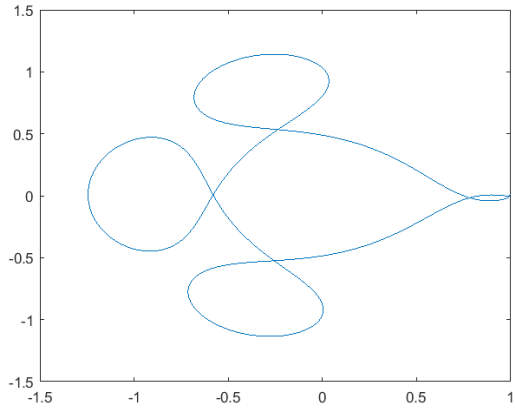
Matlab code is provided both on avenue and the last page of the pdf document

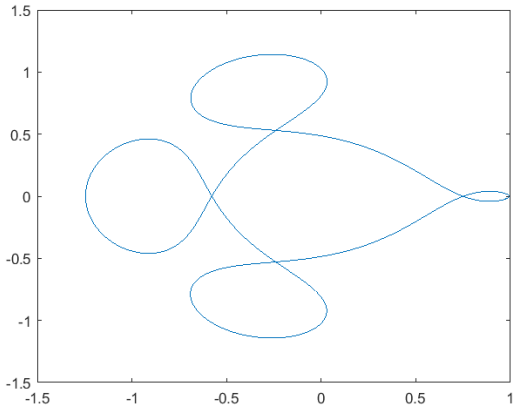
Q4

10000 steps are needed before the orbit appears to be qualitatively correct.









Matlab code is provided both on avenue and the last page of the pdf document

Q5

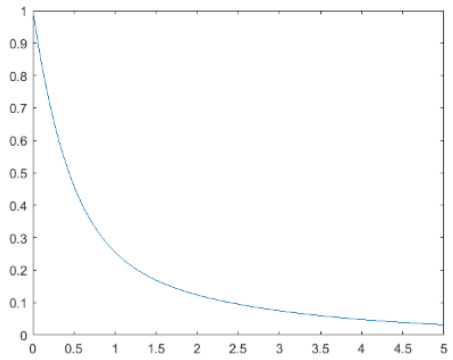
	CPU time	steps	failed steps	function evaluations
ode23	8.437500	1203425	0	3610276
ode45	0.421875	48793	0	292759
ode23s	10.203125	153143	630	1073269
ode15s	0.625000	50099	833	58611
ode113	0.375000	25906	413	52226

Discussion: ode45, 23, 113 are used to solve nonstiff problems and ode15s and ode23s are used to solve stiff problem. Ode45 usually is the recommended matlab function and it cost the second least cpu time and also have good accuracy. Ode113 cost least cpu time, but it is less accurate than ode23 and ode45. For stiff problem, obivious ode15s is more efficient than ode23s.

Matlab code is provided both on avenue and the last page of the pdf document

Q6

Since t[0 321.8122] is too large to represent fig1-4 and 7-8, I changed the range for these figure and 2 version of plots will be provided below:



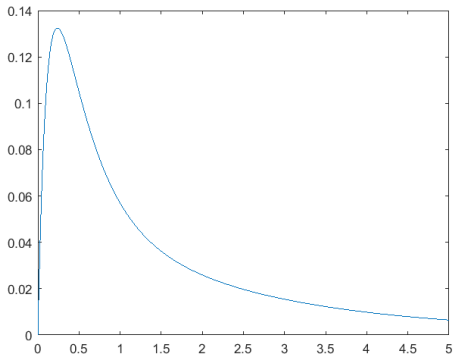
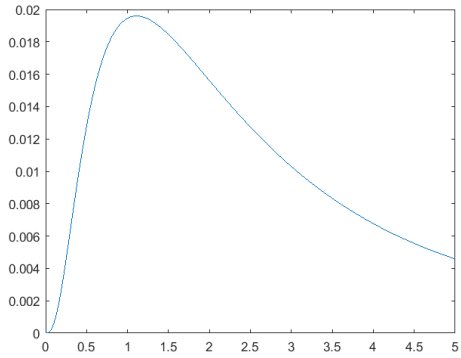


Fig 1&2



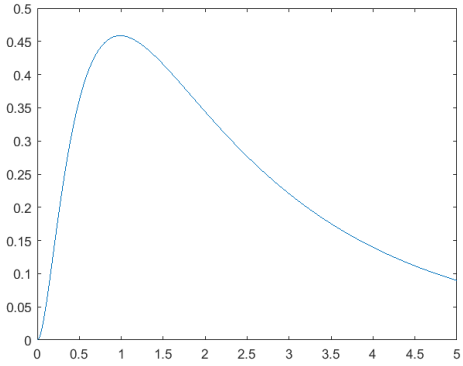
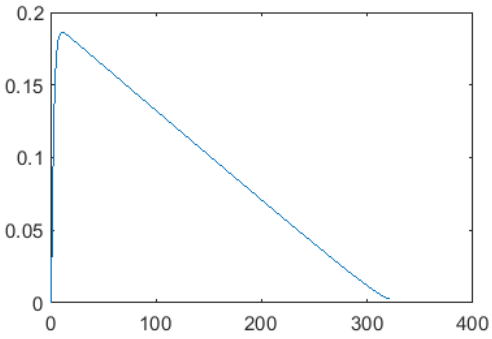


Fig 3&4



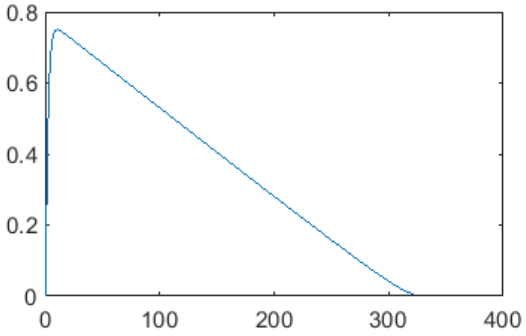
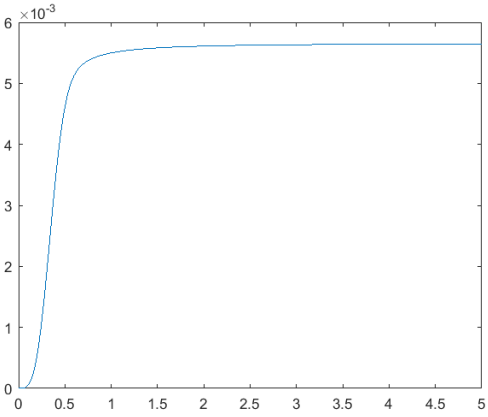


Fig 5&6



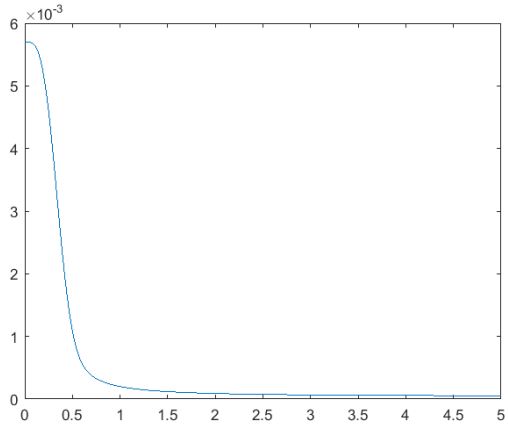
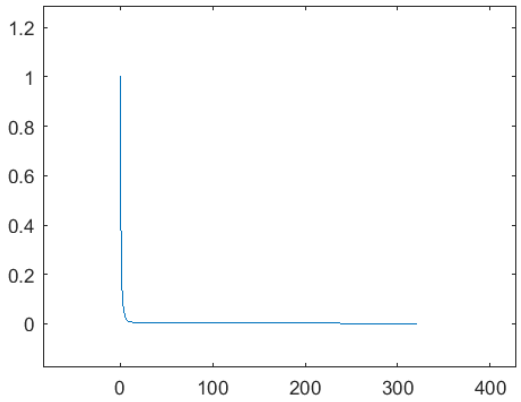


Fig 7&8



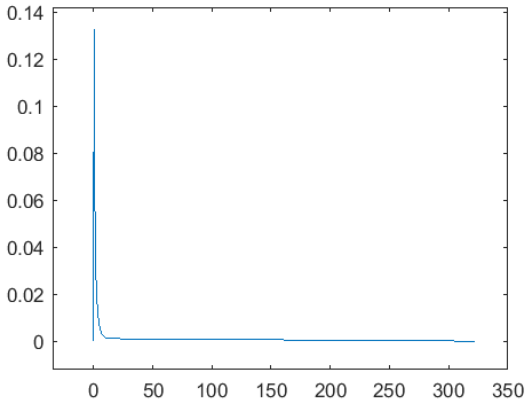
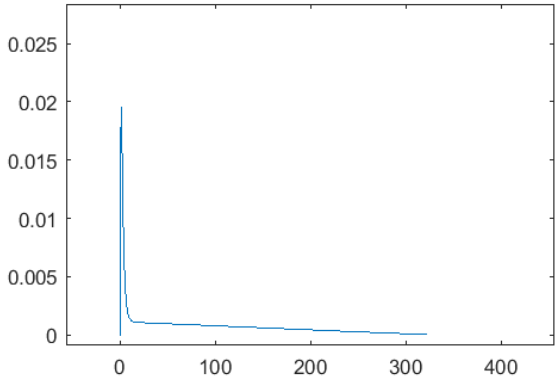


Fig1&2



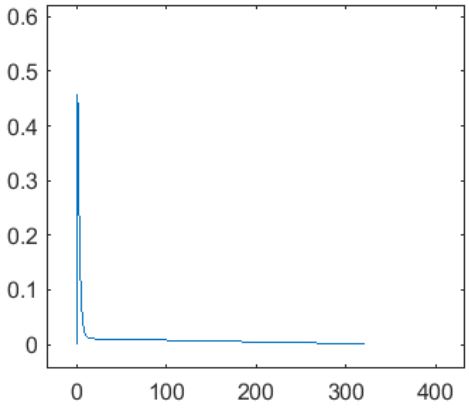
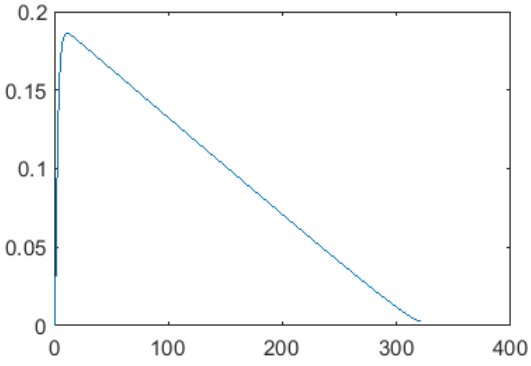


Fig 3&4



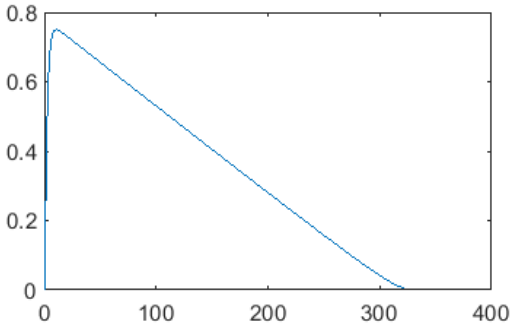
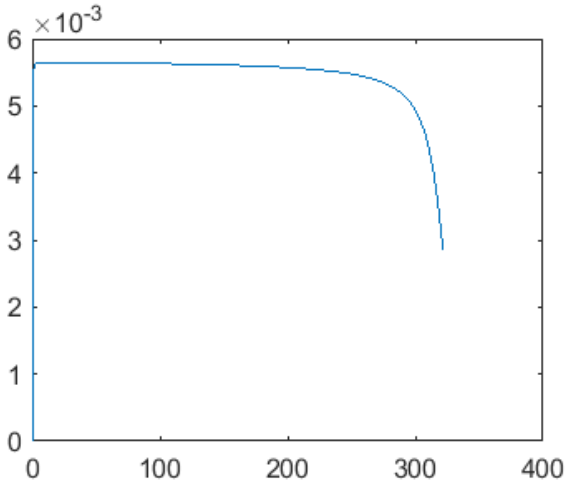


Fig 5&6



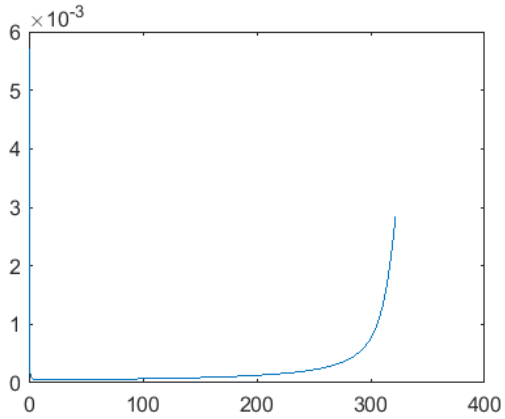


Fig7&8

b.	CPU time	steps	failed steps	function evaluations	LU decompositions	nonlinear solves
ode23s	0.078125	303	0	3336	303	909
ode15s	0.015625	200	22	435	55	370
ode45	0.156250	10381	641	66133	0	0

c. conclusion: ode23s and ode15s are used in stiff problem. Hire problem is a stiff system of 8 non-linear Ordinary Differential Equations, which means all equations are react on each other but the change speed of them may have huge difference. So here we can see ode23s and ode15s perform much better than ode45. For ode15 and ode23s, for this problem ode15s perform faster but less accurate than ode23s.

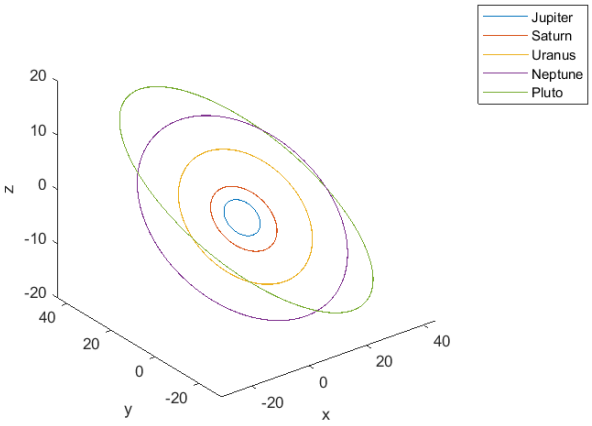
Matlab code is provided both on avenue and the last page of the pdf document

Q7

```
1. function T = findPeriod(t, x, y, z)
    l=length(t);
    T=-1;
    for i= 10:l
        if abs(x(i)-x(1))<2e-2&&abs(y(i)-y(1))<2e-
2&&abs(z(i)-z(1))<2e-2
            T=(t(i)-t(1))*100/365;
            break;
        end
    end
```

```
end
```

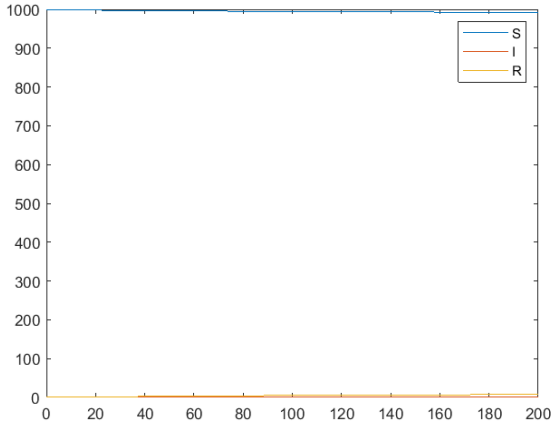
first I calculated the size of nbody data and than for each data set at time t, I compare them with the initial data at t(0) if all the 3 coordinates are all have difference smaller than 2e-2, than that is the initial point. Then I translated t to years. If no such a point could be found, return T=-1.

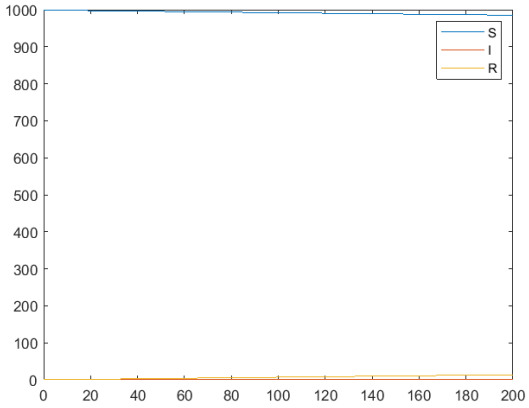


2.

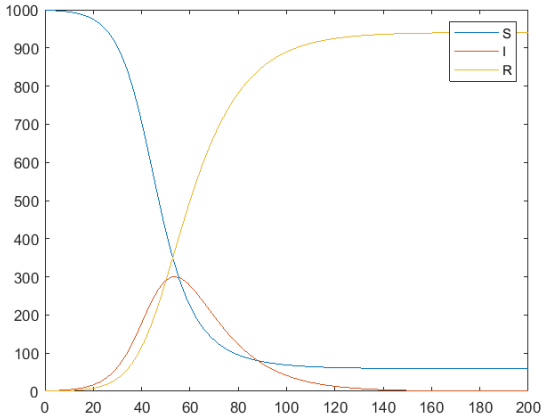
```
Jupiter      11.859730
Saturn       29.478098
Uranus       84.057336
Neptune      164.886759
Pluto        248.000088
fx >>
```

Q8

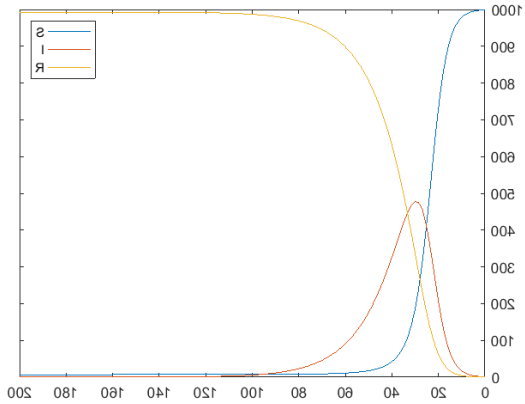




R0S0=0.9



R0S0=1



ROS0=3

R0S0=5

Matlab code is provided both on avenue and the last page of the pdf document

MATLABCODE

Q2

```
function x=bisection(f,a,b)
    c=8888;
    n=0;
    while abs(2-c)>1e-6 &&n<=1000
        n=n+1
        c=(a+b)/2
        if f(c)*f(b)==0
            x=c;
            break;
        elseif f(c)*f(b)<0
            a=c;
        else
            b=c;
        end
    end
    x=c;
end
f = @(x)((x-2).^9);
fff=@(x)(x.^9-18.*x.^8 + 144.*x.^7-672.*x.^6 + 2016.*x.^5-
4032.*x.^4 + 5376.*x.^3-4608.*x.^2 + 2304.*x-512);
ff=@(x)(((((((((x-18).*x+144).*x-672).*x+2016).*x-
4032).*x+5376).*x-4608).*x+2304).*x-512);
```

```
x=linspace(1.92,2.08,161);
y1=f(x);
y2=ff(x);
%plot(x,y1);
%title('(x-2)^9')
%plot(x,y2);
%title('Horner's method')
x=bisection(ff,1.92,2.07)
options=optimset('TolFun',1e-30);
ans1=fsolve(f,1.9,options)
ans2=fsolve(fff,1.9,options)
```

Q3

```
[xlist1, outputx1,iteration1]=newton(f1,[15;-2])
[xlist2, outputx2,iteration2]=newton(f2,[(1+sqrt(3))/2;(1-
sqrt(3))/2;sqrt(3)])
[xlist3, outputx3,iteration3]=newton(f3,[1;2;1;1])
[xlist4, outputx4,iteration4]=newton(f4,[0;0])
f11=@(x)[x(1)-13+x(2)*(x(2)*(5-x(2))-2);x(1)-
29+x(2)*((x(2)*(1+x(2))-14))];
f22=@(x)[x(1)^2+x(2)^2+x(3)^2-5;x(1)+x(2)-1;x(1)+x(3)-3];
f33=@(x)[x(1)+10*x(2);sqrt(5)*(x(3)-x(4));(x(2)-
x(3))^2;sqrt(10)*(x(1)-x(4))^2];
f44=@(x)[10^4*x(1)*x(2)-1;exp(-x(1))+exp(-x(2))-1.0001];
options=optimset('TolFun',1e-6);
[fsolve1,feval1,exit1]=fsolve(f11,[15;-2],options)
[fsolve2,feval2,exit2]=fsolve(f22,[(1+sqrt(3))/2;(1-
sqrt(3))/2;sqrt(3)],options)
[fsolve3,feval3,exit3]=fsolve(f33,[1;2;1;1],options)
[fsolve4,feval4,exit4]=fsolve(f44,[0;0],options)

function [xlist, outputx,iteration]=newton(f,x0)
    eps=1e-6;
    iteration=1;
    tol=8888;
    maxlevel=1000;
    x=transpose(symvar(f));
    diff = jacobian(f,x);
    while tol>eps && iteration<=maxlevel
        fx = subs(f,x,x0);
        dfx = subs(diff,x,x0);
        outputx=vpa(x0-dfx\fx');
        tol=norm(outputx-x0);
```

```
        x0=outputx;
        xlist(:,iteration)=x0;
        iteration=iteration+1;
    end
end
```

Q4

```
f=@(x,u) [u(2);u(1)+2*u(4)-
0.987722529*((u(1)+0.012277471)/((((u(1)+0.012277471)^2)+u(
3)^2)^(3/2)))-0.012277471*((u(1)-0.987722529)/((((u(1)-
0.987722529)^2)+u(3)^2)^(3/2))); u(4); u(3)-2*u(2)-
0.987722529*(u(3)/((((u(1)+0.012277471)^2)+u(3)^2)^(3/2)))-
0.012277471*(u(3)/((((u(1)-
0.987722529)^2)+u(3)^2)^(3/2)))];
steps =[100,1000,10000,20000];
for i=1:4
    h=(17.1)/steps(i);
    x=linspace(0,17.1,steps(i));
    u=ones(4,steps(i));
    u(:,1)=[0.994,0,0,-2.00158510637908252242053786 22]';
    for i=1:steps(i)
        K1=h.*f(x(i),u(:,i));
        K2=h.*f(x(i)+h/2,u(:,i)+K1./2);
        K3=h.*f(x(i)+h/2,u(:,i)+K2./2);
        K4=h.*f(x(i)+h,u(:,i)+K3);
        u(:,i+1) = u(:,i)+(K1+2.*K2+2.*K3+K4)/6;
    end
    figure(i);
    plot(u(1,:),u(3,:));
end
```

Q5

```
f=@(x,u) [u(2);u(1)+2*u(4)-
0.987722529*((u(1)+0.012277471)/((((u(1)+0.012277471)^2)+u(
3)^2)^(3/2)))-0.012277471*((u(1)-0.987722529)/((((u(1)-
0.987722529)^2)+u(3)^2)^(3/2))); u(4); u(3)-2*u(2)-
0.987722529*(u(3)/((((u(1)+0.012277471)^2)+u(3)^2)^(3/2)))-
0.012277471*(u(3)/((((u(1)-
0.987722529)^2)+u(3)^2)^(3/2)))];
y0=[0.994,0,0,-2.00158510637908252242053786 22]';
opts = odeset('RelTol',1e-10,'AbsTol',1e-10);
t023=cputime;
[t, y, stats1] = ode23(f,[0 1000],y0,opts);
t123=cputime-t023
t045=cputime;
```

```
[t, y, stats2] = ode45(f,[0 1000],y0,opts);
t145=cputime-t045
t023s=cputime;
[t, y, stats3] = ode23s(f,[0 1000],y0,opts);
t123s=cputime-t023s
t015s=cputime;
[t, y, stats4] = ode15s(f,[0 1000],y0,opts);
t115s=cputime-t015s
t0113=cputime;
[t, y, stats5] = ode113(f,[0 1000],y0,opts);
t1113=cputime-t0113
fprintf('       CPU time      steps       failed steps
function evaluations\n')
fprintf('ode23  %f       %d      %d         %d
\node45  %f       %d      %d         %d
\node23s  %f       %d      %d         %d
\node15s  %f       %d      %d         %d
\node113  %f       %d      %d         %d
\n',t123,stats1(1),stats1(2),stats1(3),t145,stats2(1),stats
2(2),stats2(3),t123s,stats3(1),stats3(2),stats3(3),t115s,st
ats4(1),stats4(2),stats4(3),t1113,stats5(1),stats5(2),stats
5(3));
Q6
f=@(x,y) [-
1.71.*y(1)+0.43.*y(2)+8.32.*y(3)+0.0007;1.71.*y(1)-
8.75.*y(2);-
10.03.*y(3)+0.43.*y(4)+0.035.*y(5);8.32.*y(2)+1.71.*y(3)-
1.12.*y(4);-1.745.*y(5)+0.43.*y(6)+0.43.*y(7);-
280.*y(6)*y(8)+0.69.*y(4)+1.71.*y(5)-
0.43.*y(6)+0.69.*y(7);280.*y(6)*y(8)-1.81.*y(7);-
280.*y(6)*y(8)+1.81.*y(7)];
y0=[1; 0; 0; 0; 0; 0;0;0.0057 ]';
opts = odeset('RelTol',1e-6,'AbsTol',1e-6);
t023s=cputime;
[t, y, stats1] = ode23s(f,[0 321.8122],y0,opts);
t123s=cputime-t023s
stats1
for i=1:8
    figure(i)
    plot(t,y(:,i))
end
t015s=cputime;
[t, y, stats2] = ode15s(f,[0 321.8122],y0,opts);
```

```
t015s=cputime-t015s
stats2

t045=cputime;
[t, y, stats3] = ode45(f,[0 321.8122],y0,opts);
t145=cputime-t045
stats3
fprintf('       CPU time       steps       failed steps
function evaluations       LU decompositions
nonlinear solves\n')
fprintf('ode23s  %f       %d         %d             %d
%d             %d
\node15s  %f       %d         %d             %d
%d             %d
\node45  %f       %d         %d             %d
%d             %d
\n',t123s,stats1(1),stats1(2),stats1(3),stats1(5),stats1(6)
,t015s,stats2(1),stats2(2),stats2(3),stats2(5),stats2(6),
t145,stats3(1),stats3(2),stats3(3),stats3(5),stats3(6));
Q8

f=@(x,y) [-1.71.*y(1)+0.43.*y(2)+8.32.*y(3)+0.0007]
y0=[999,1,0];
RS=[0.9,1,3,5];
for i =1:4
   beta=RS(i)/999/14;
   f=@(x,y) [-beta*y(1)*y(2);beta*y(1)*y(2)-
y(2)/14;y(2)/14];
   [t,y]=ode45(f,[0 200],y0);
   figure(i);
   plot(t,y)
   legend('S','I','R')
end
```