SFWRENG 4X03 Lab 4

1a) netbp.m code:

function netbp(points , labels , neurons , learning\_rate , niter , 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(neurons(3),neurons(2));

b2 = 0.5\*randn(neurons(1),1); b3 = 0.5\*randn(neurons(2),1); b4 = 0.5\*randn(neurons(3),1);

% Forward and Back propagate

savecost = zeros(niter,1); % value of cost function at each iteration

for counter = 1:niter

k = randi(length(points)); % choose a training point at random

x = points(:,k); %%for every point

% Forward pass

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 - learning\_rate\*delta2\*x';

W3 = W3 - learning\_rate\*delta3\*a2';

W4 = W4 - learning\_rate\*delta4\*a3';

b2 = b2 - learning\_rate\*delta2;

b3 = b3 - learning\_rate\*delta3;

b4 = b4 - learning\_rate\*delta4;

% Monitor progress

newcost = cost(W2,W3,W4,b2,b3,b4); % display cost to screen

savecost(counter) = newcost;

end

% Show decay of cost function

save costvec

semilogy((1:1e4:niter),savecost(1:1e4:niter));

function costval = cost(W2,W3,W4,b2,b3,b4)

costvec = zeros(length(points),1);

for i = 1:length(points) %%iterate for every point

x = points(:,i); %% for every point

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','savecost','learning\_rate');

end

1b) classifypoints.m code:

function category = classifypoints(file,points)

load(file); %%load all variables

category = zeros(1,length(points)); %% initialize return var

for i = 1:length(points)

x = points(:,i); %%from netbp

a2 = activate(x,W2,b2);

a3 = activate(a2,W3,b3);

output\_vec = activate(a3,W4,b4); %%past through sigmoid activate function layer times

if output\_vec(1,1) >= output\_vec(2,1)

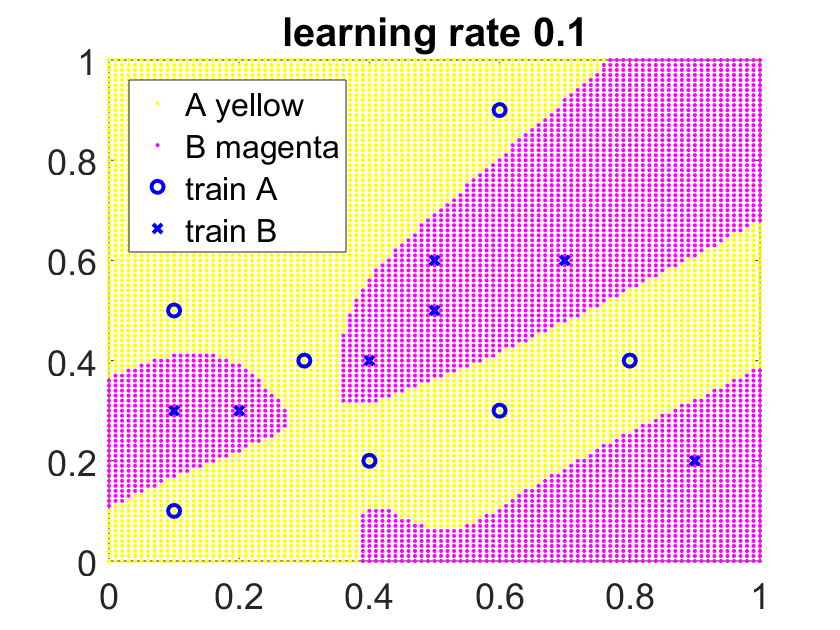
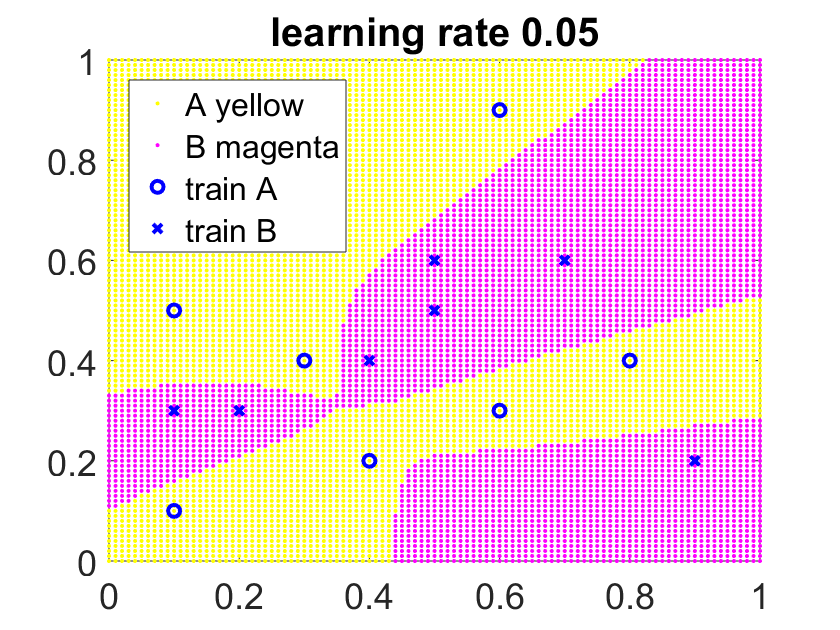
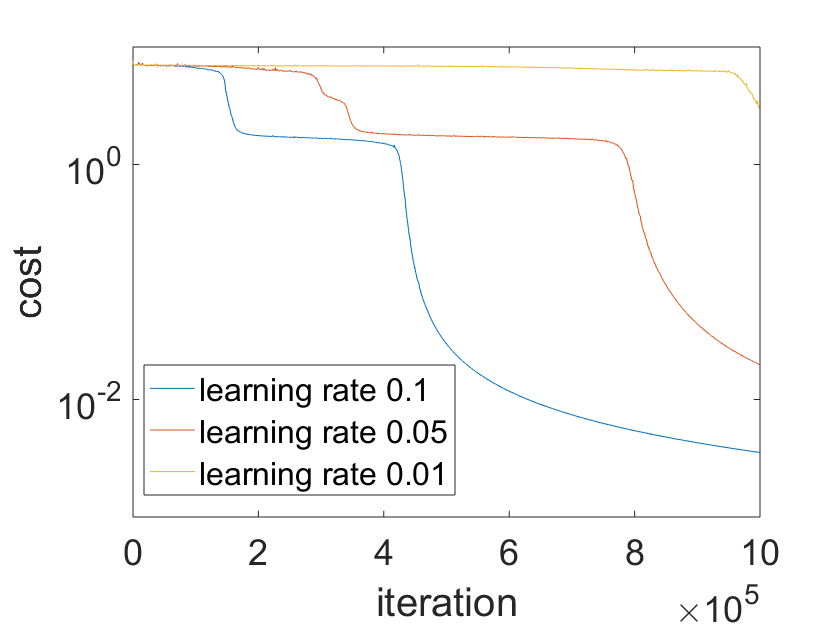
category(i) = 1;

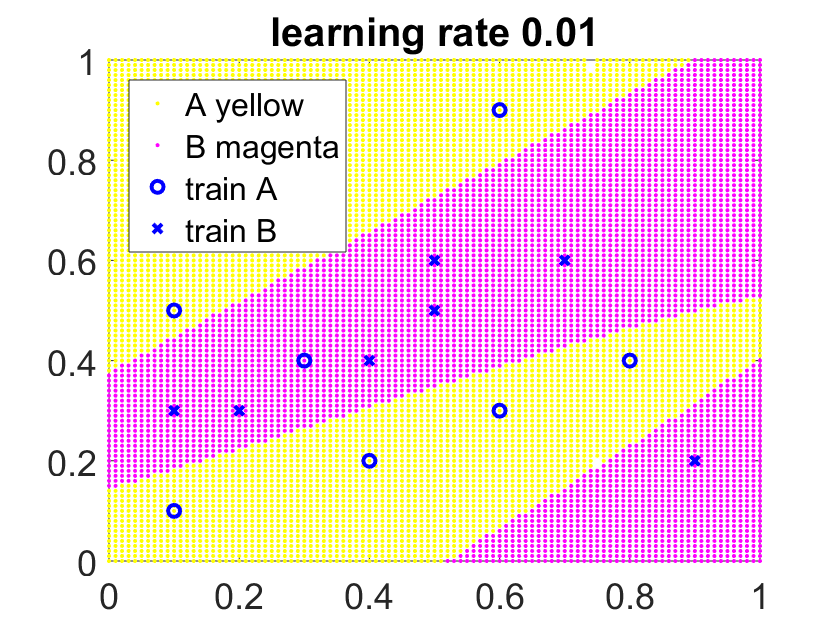
end

end

end

1c) best results at 4 layers,

neurons = [10, 16, 2], learning\_rates = [0.1, 0.05, 0.01]



Across multiple learning rates, this neuron configuration can produce similar plots.

2.

bisection:

f = @(x) x-exp(2-sqrt(x)): (1 root)

my root = 1.877321666106582e+00

fzero(f,1) = 1.877321666687556e+00

absolute error = 5.809739356266164e-10

My method is accurate to the 10th significant digit

f = @(x) x\*sin(x^2) -1: (multiple roots in region, make sure both methods refer to same root)

my root = 4.368127214409469e+00

fzero(f,4.5) = 4.368127214401134e+00

absolute error = 8.335554468885675e-12

My method is accurate to the 12th significant digit

Newtons:

f = @(x) x^3 -2\*x -5: (1 root)

my root = 2.094551481543604e+00

fzero(f,4.5) = 2.094551481542327e+00

absolute error = 1.277644656738630e-12

My method is accurate to the 12th significant digit

f = @(x) x\*sin(x^2) -1: (multiple roots, make sure both methods refer to same root)

my root = 3.930741781510152e+00

fzero(f,4) = 3.930741781510152e+00

absolute error =0

My method is as accurate as the fzero method.

Bisection.m code:

function root = bisection(f,a,b)

if f(a)\*f(b)<0 %%guess is proper

mid = (a+b)/2;

if abs(f(mid)) < 1e-9 %% close to 0 with error tolerance

root = mid;

elseif f(a)\*f(mid)<0 %%root in a-mid interval

root = bisection(a,mid,f);

else %% root in mid-b interval

root = bisection(mid,b,f);

end

else

disp('cannot compute with this guess')

end

end

Newtons.m code:

function root = newtons(f,x0)

%%using syms package

%---------------initialize--------%

df = matlabFunction(diff(sym(f))) ; %%compute derivative

y0 = f(x0);

root = lineRoot(x0,y0,df); %%calculate root of line

yRoot = f(root); %%value at root

%-------------loop---------%

while abs(yRoot) > 1e-9 %%while not precise enough

x0 = root; %update x0

y0 = f(x0);

root = lineRoot(x0,y0,df);

yRoot = f(root);

end

%%--------------------functions---------------%%

function root = lineRoot(x0,y0,df) %%computes next root

slope = df(x0);

root = x0 - y0/slope;

end

end

3a)

The method does not finish executing under these circumstances. The line created at x0 = 1 is y = -2x -2 so the line’s root is x = -1. This new guess then creates a line y = -2x+2, which has a root at x = 1. This brings us back to our initial guess, so the loop will never terminate.

b)

The method computes the root to be x = 1.600485180440241e+00. The outcome with the guess is different then the first outcome because the lines created do not form an endless loop of repeating roots.

4)

When x0 = 0 the method returns root = Inf. This happens because at x = 0 the slope of the function is 0 so the line created does not have a root. The calculation of the root divides by 0 so this causes the root to be x = Inf. The method finishes because it evaluates the value of the function at x = inf. The value y = NaN is then compared to the tolerance where it is “smaller” so the loop finishes.

The result I obtain when computing fsolve(f,1) = 3.129879311117518e+00, and the result I obtain when computing my method is x = 9.424744704881126e+00. This does not mean my method is incorrect, both methods are just referring to different roots of the function.

5a)

Replacing the derivative with a constant d, will result in the following behavior:

Guess will only move forward if: y0 and d have **different** signs

Guess will only move backward if: y0 and d have the **same** signs

This means that the condition for local convergence is that the guess must be moving in the direction of the root for every iteration.

b)

if and the rate of convergence, where L is the value we converge to, the rate of convergence should be:

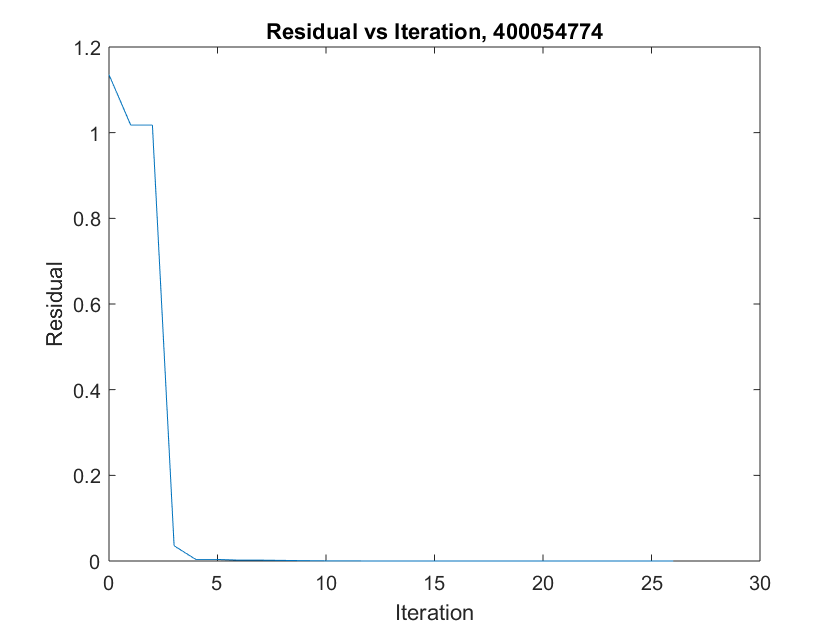
c)

The method will converge quadratically when where

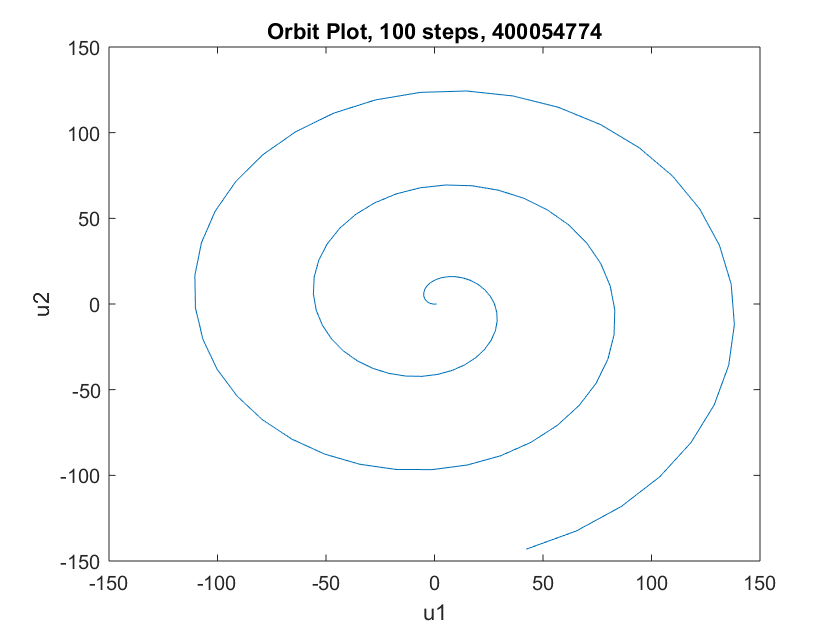
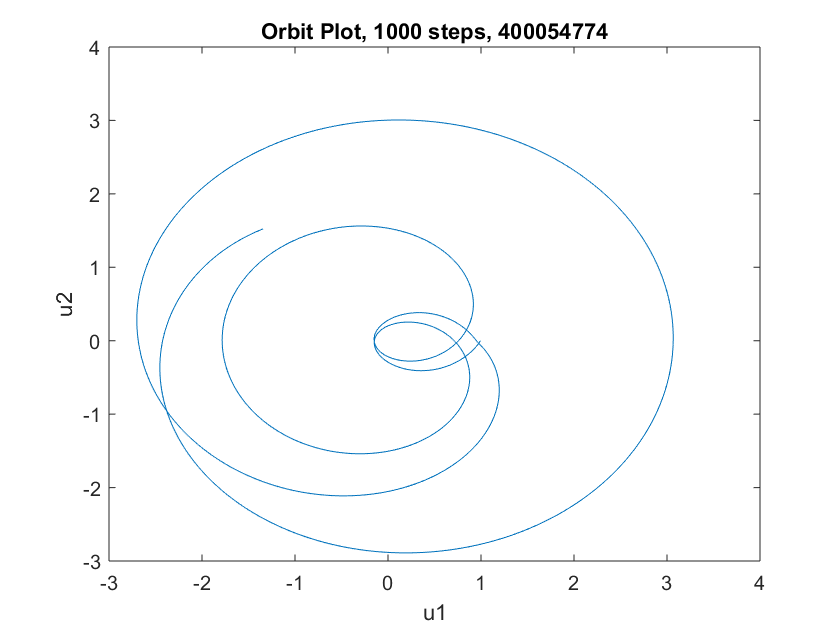
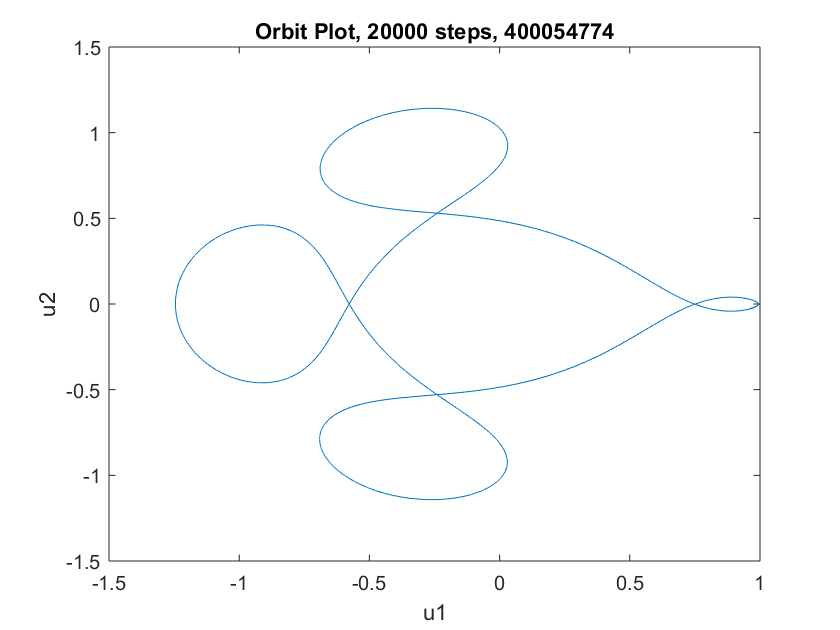
Therefore, it will converge quadratically when:

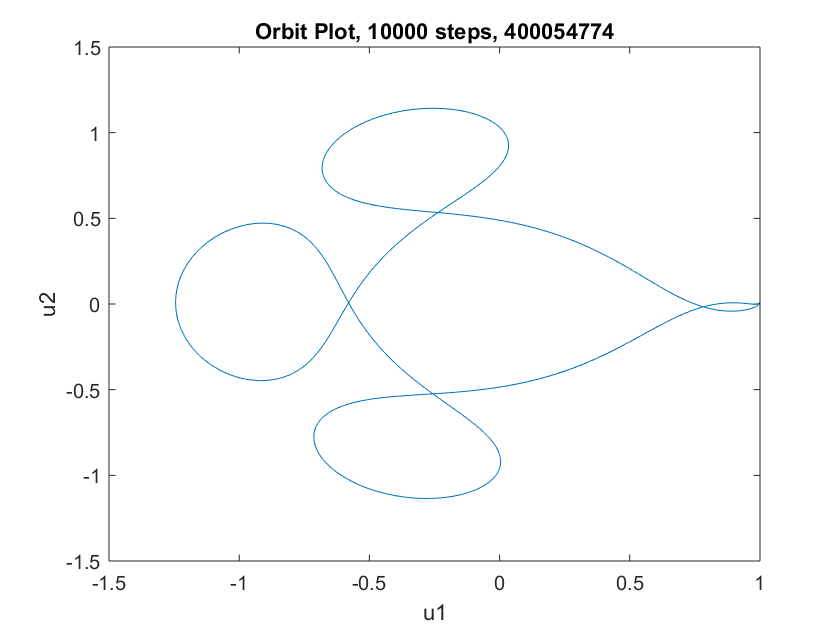
6

Yes, Newtons method will have difficulty solving this. The curves both have asymptotes so they never actually cross each other or the x-axis, therefore they do not have roots.

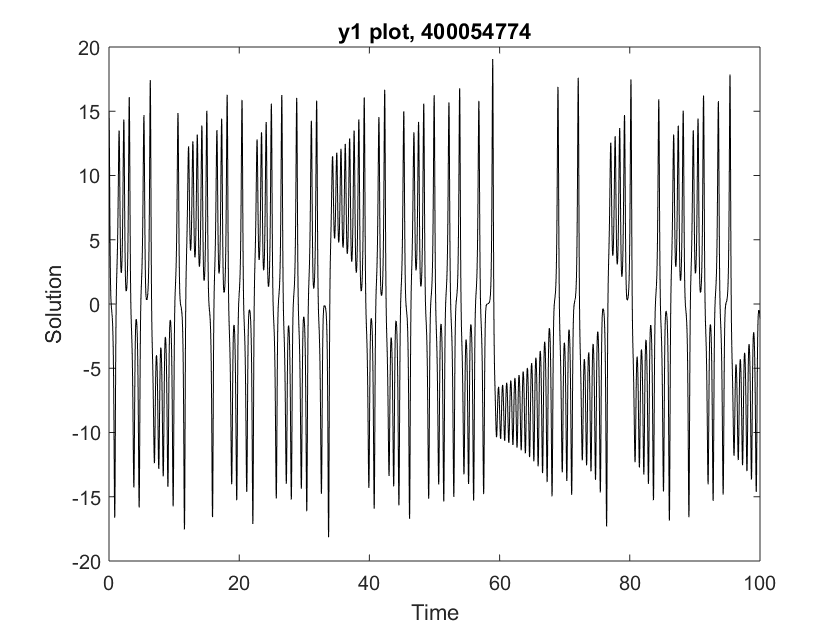
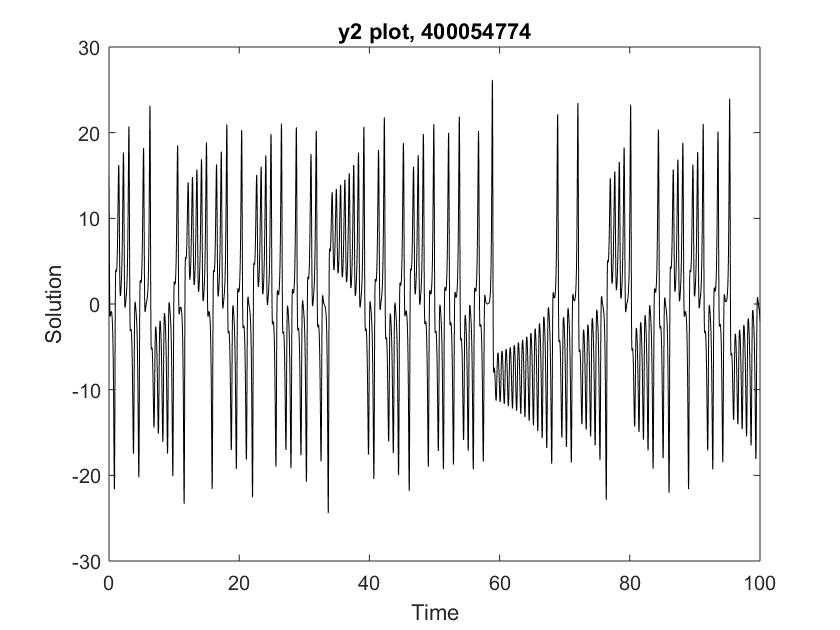
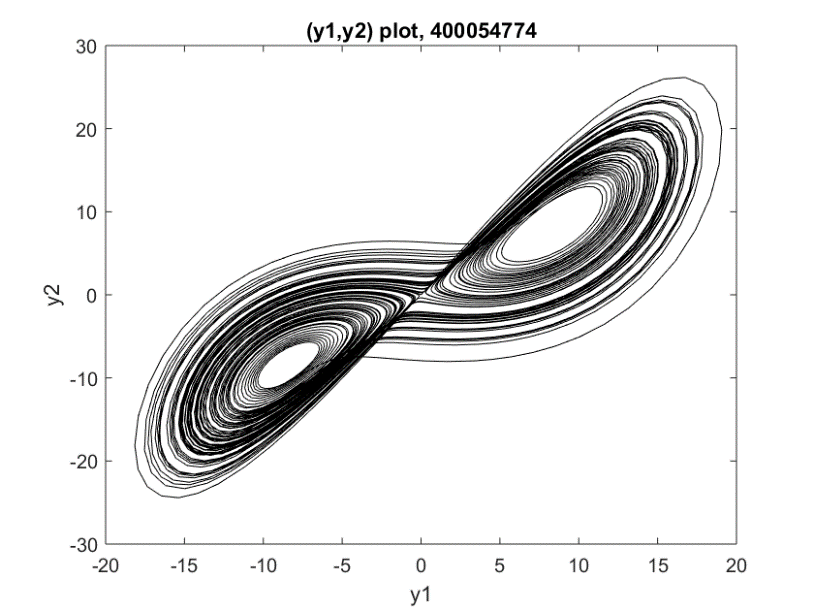
The output of fsolve states that the equation is solved however this does not mean it converges to a root. The residual becomes small but NOT zero.

I can conclude that this plot matches up with my statement about the residual.

7.



The data from the plots determine that it takes around 10000 uniform steps for the orbit to appear qualitatively correct.

8. A minor change of 10^-10 is enough to make a difference on each plot.

