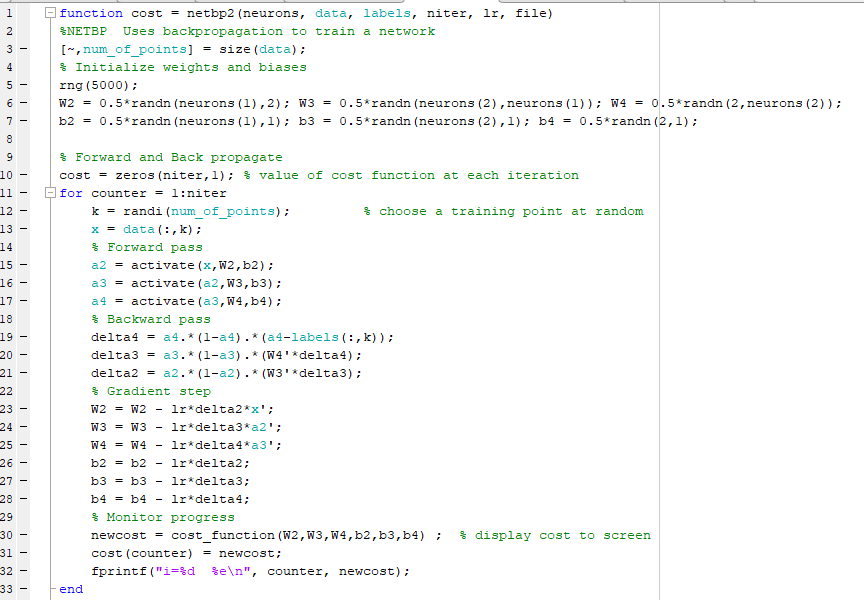
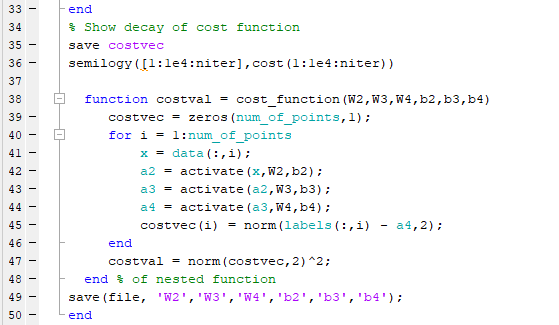
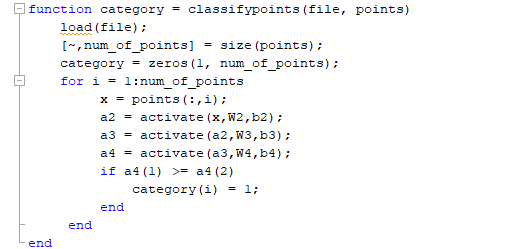
Problem 1:

Netbp2.m:

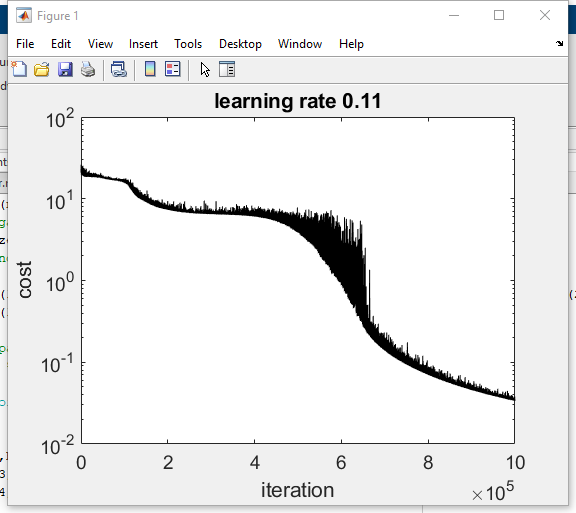


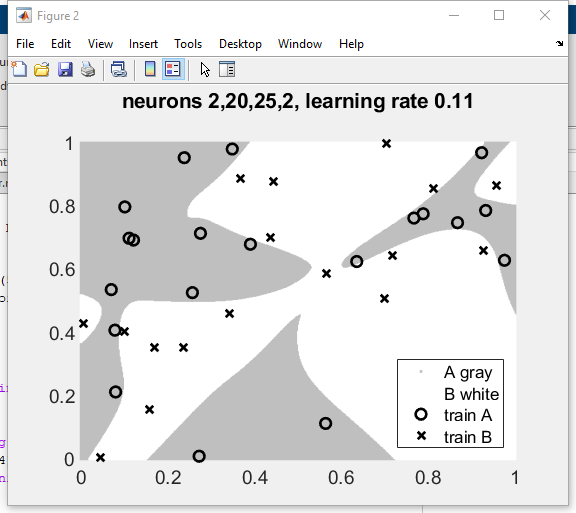


Classifypoints.m:



Two plots:





Params:

neurons = [20 25];

learning\_rate = 0.11;

niter = 1e6;

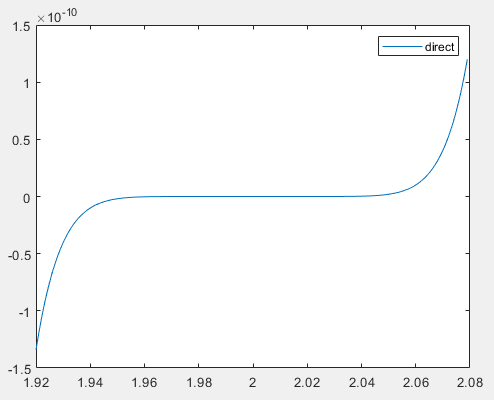
What I found:

1. More neurons will reduce the speed of training.
2. More iterations do not necessarily lead to a better learning and classification outcome.
3. Minimizing the cost will lead to a better learning outcome.
4. We need enough number of iterations to ensure the neural network can minimize the cost function.

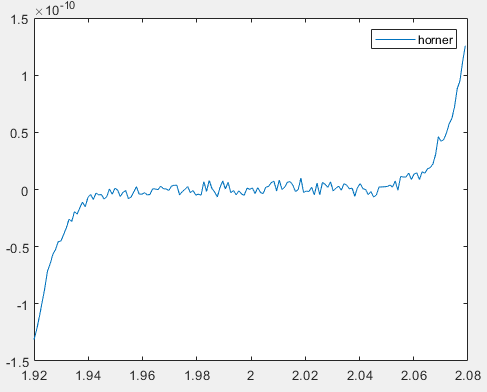
Problem 2:

a)

Direct evaluation:



Horner’s method:



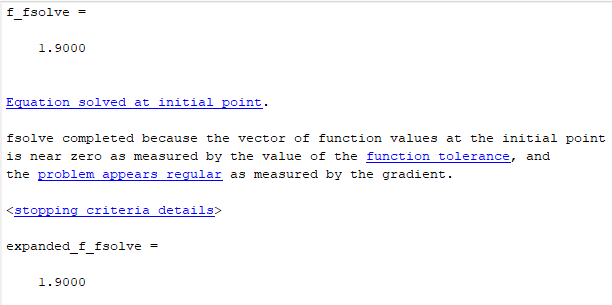
We can easily tell that Horner’s method appeared to have more visible errors in the plot. This is because we expanded our function and thus introduced more calculations in the evaluation process. Therefore, the errors were larger.

So, when the function is not transformable into a less complicated expression, Horner’s method has a better performance because it involves the least amount of calculation possible. However, if the whole expression can be transformed into a simpler one, the simpler expression outperforms Horner’s method.

b) Applying bisection to (x-2)^9 ended up returning the result 1.999999 which is within 1e-6 error with the correct result. Applying bisection to horner’s ended up returning the result 1.964358 which isn’t within the error tolerance.

c) Because the accuracy of horner’s method is limited. As you can see from the plot, horner’s evaluation’s plot was jagged, although the overall shape is very close. Because of this jagged nature of our data, bisection is determining the root based on the signs of two function evaluations, which is obviously very sensitive to the Jagged data that horner’s evaluation had.

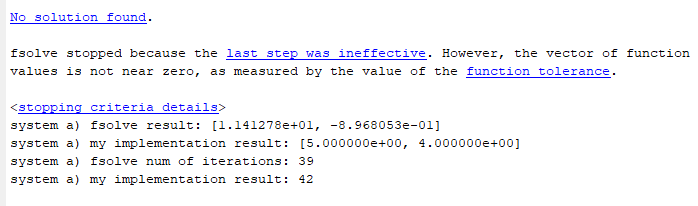
For example, f(a) and f(b) may actually have the same sign, but in horner’s plot because of the inaccuracy of our data, they might even have the same sign and lead the algorithm to proceed with another interval of a and b that doesn’t actually contain the correct root.

d) 

Both terminated at 1.9 as solution.

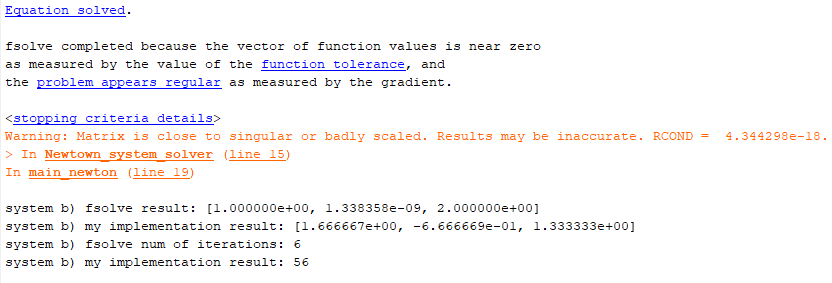
Problem 3:

a)



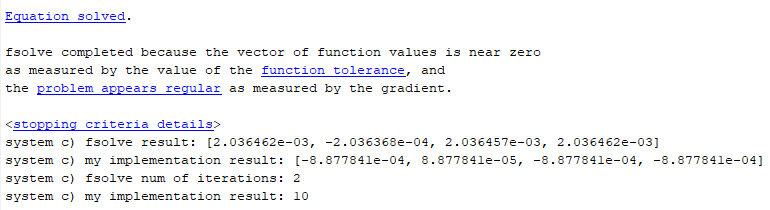
Because the system is in newton’s form and involve more calculations and potential errors.

b)



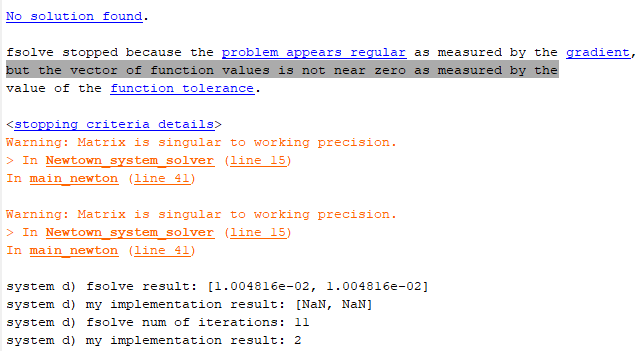
Results are different because the system might have multiple solutions or the fsolve only ends at a root that is close enough to 0 but not actually zero mathematically.

c)



Results are different because the system might have multiple solutions or the fsolve only ends at a root that is close enough to 0 but not actually zero mathematically.

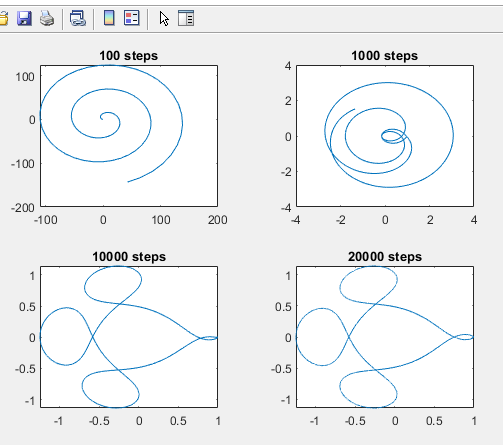
d)



Reason that it is not working is the matrix itself is singular to working precision.

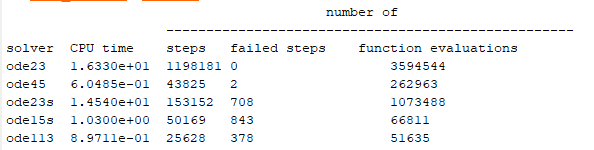
Problem 4:

About 10000 steps and more are needed.



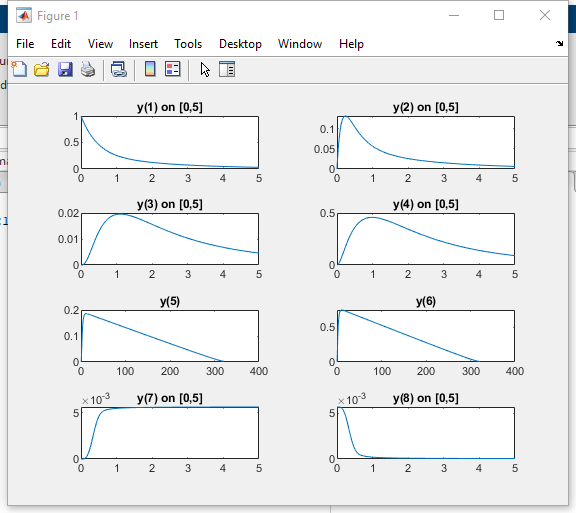
Problem 5:

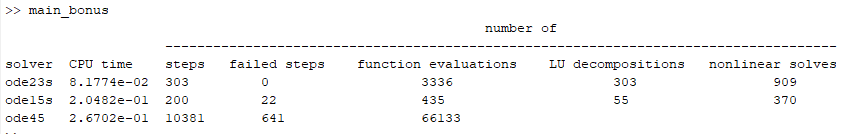
Table



Ode45 most efficient.

Problem 6:





Conclusion: When solving stiff problems, stiff solvers like ode23s and ode15s are much faster than solvers like ode45. We should always choose the most suitable ode solvers depending on the problem that we are dealing with.

Problem 7:

1. Words: Check along the dataset, starting from some point away from the first xyz. See if at any point of t(i) the norm of the difference between [x(i) y(i) z(i)] and [x(1) y(1) z(1)] is smaller than tolerance. I used 0.028 as tolerance.

Pesudocode:

function T = findPeriod(t, x, y, z)

vector0 = [x(1) y(1) z(1)];

x\_size = size(x);

n = x\_size(1);

tol = 0.028;

for i=100:n

if abs(norm([x(i) y(i) z(i)]-vector0))<tol && sign(x(i))==sign(x(1)) && sign(y(i))==sign(y(1)) && sign(z(i))==sign(z(1))

T = t(i)\*100/365;

return;

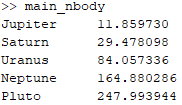
end

end

T = -1;

End





Problem 8:

