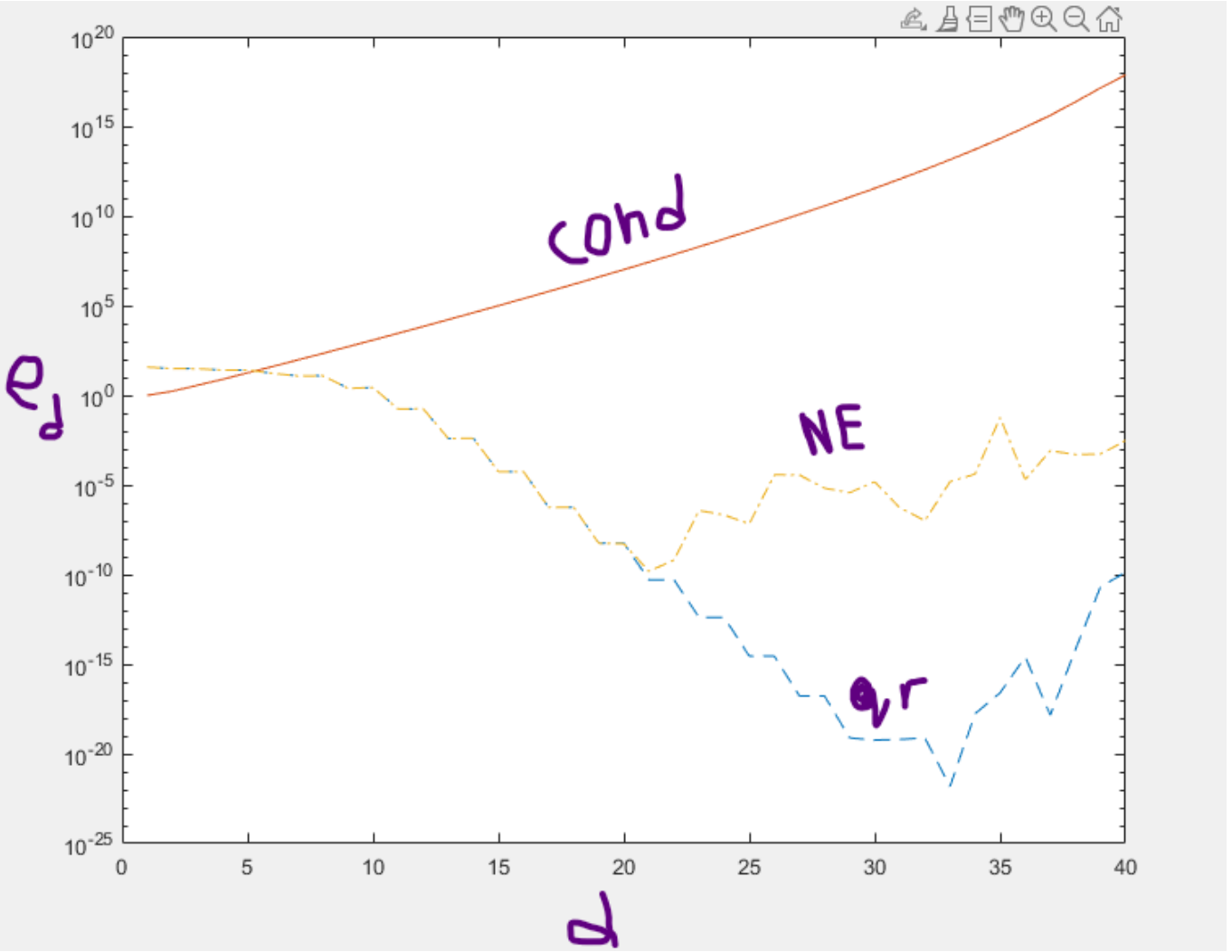
NumComp - Fall 2022
Project #9

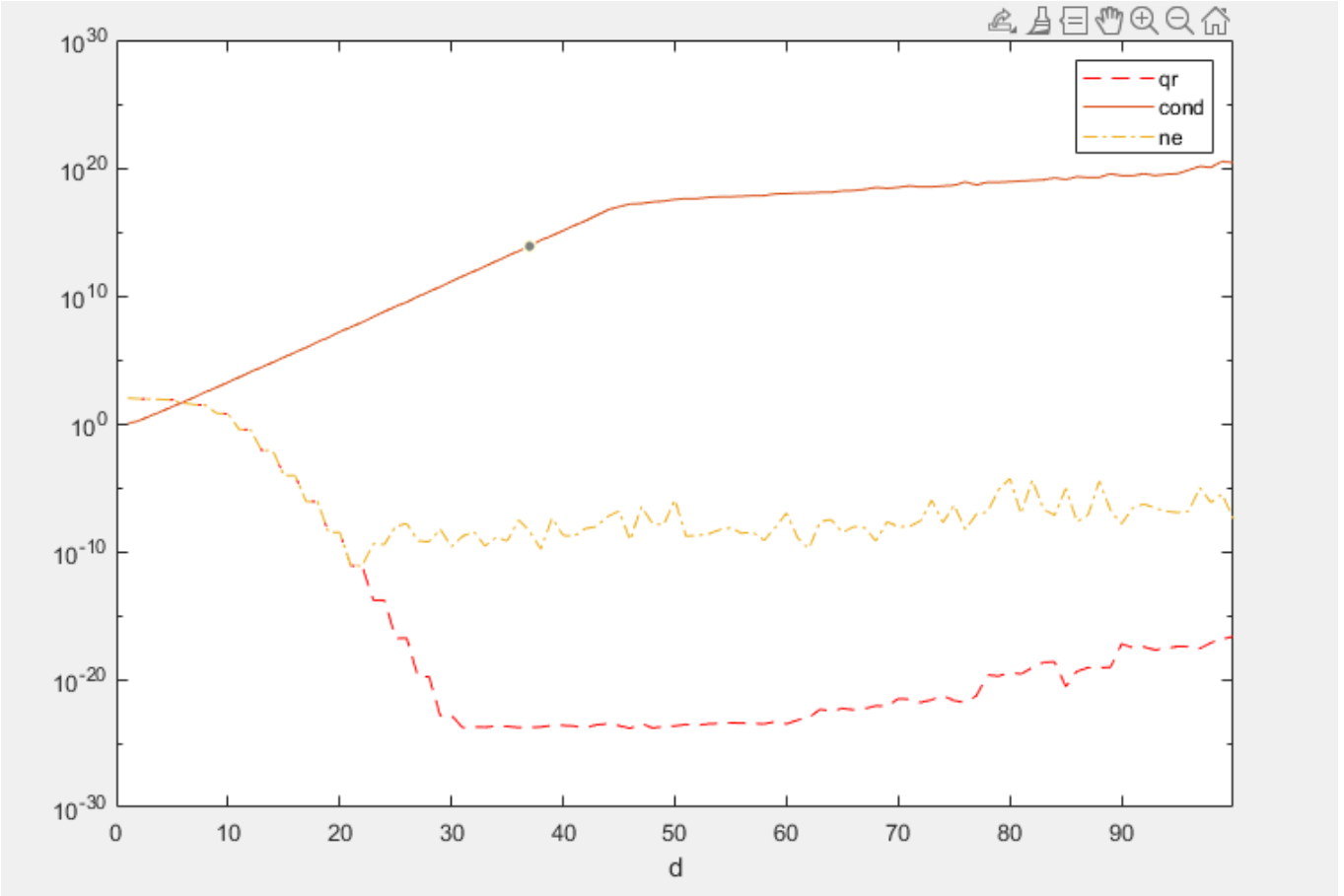
Due –
Isaiah Thomas

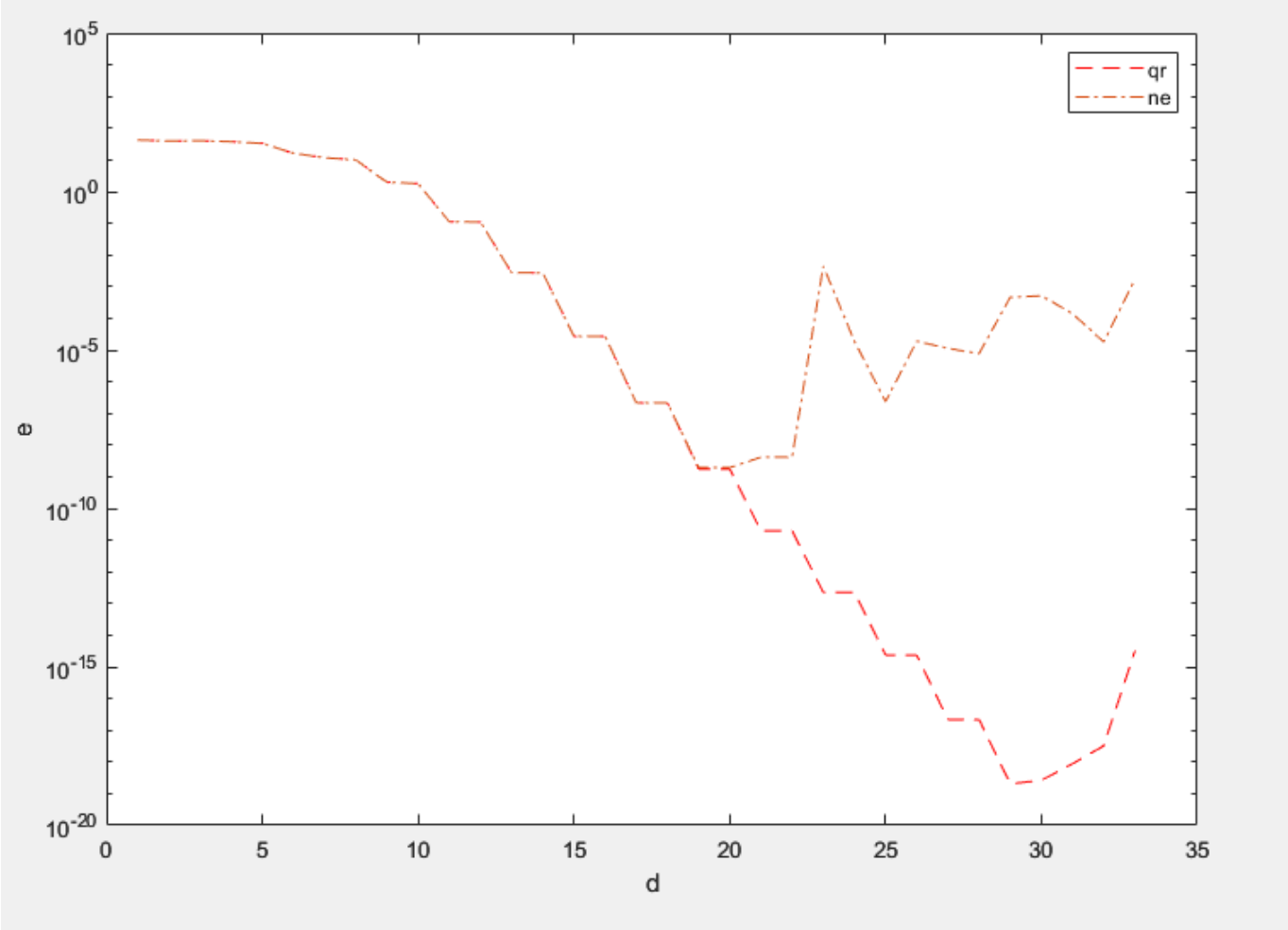
Preface– I made a mistake on error calculation on my last submission. I calculated the norm of the residual, rather than whatever the heck formula you had on the board–

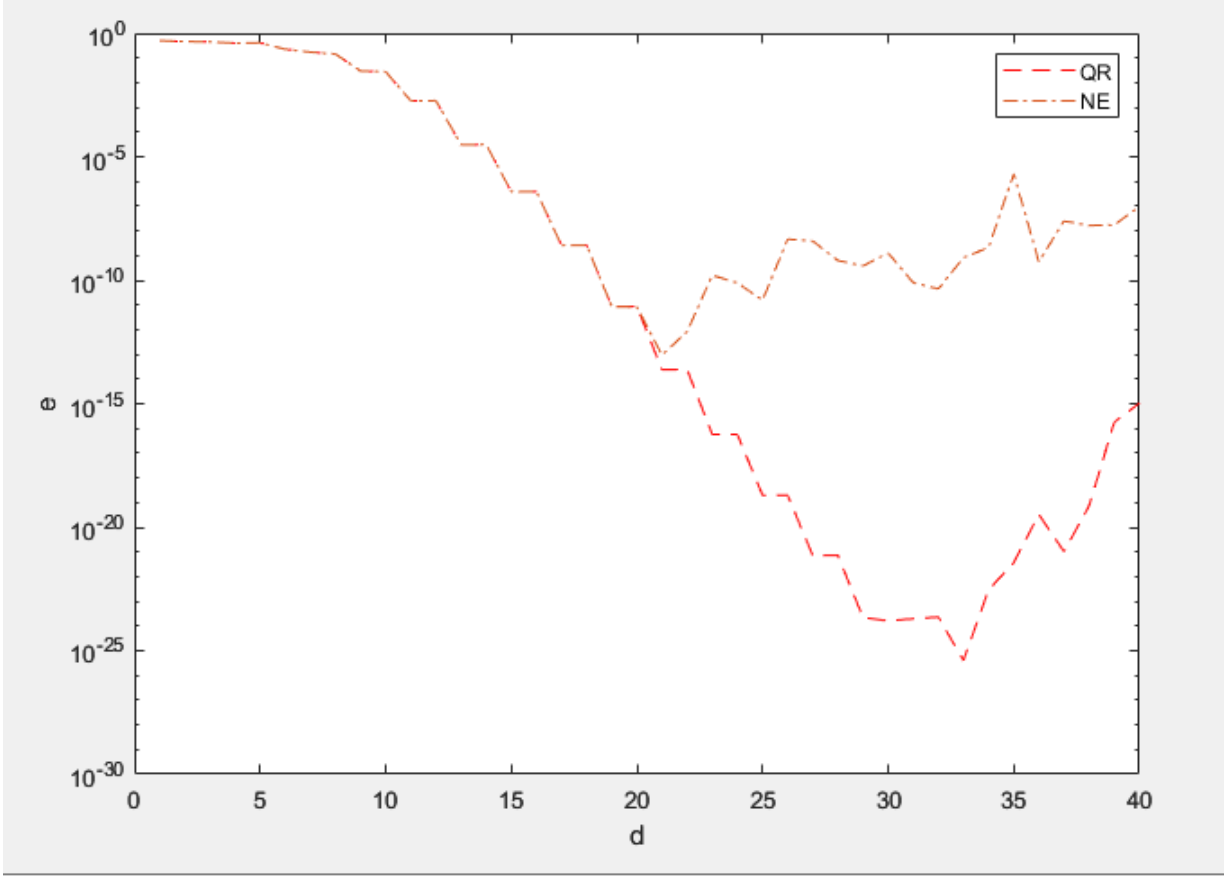
$$\left(\frac{\sum_{i=1}^{m}(y_i' - \sum_{k=0}^{d} c_k^*(x_i^k)^2}{\sum_{i=1}^{m}(y_i')}\right)^{1/2}$$

My fourth plot holds the error results for the correct formula, which I could not really deduce any new results or ideas from. Plots!




qr
cond
ne
10^{30}
10^{20}
10^{10}
10^{0}
10^{-10}
10^{-20}
10^{-30}
0
10
20
30
40
50
60
70
80
90
d


qr
ne
e
d
10^{5}
10^{0}
10^{-5}
10^{-10}
10^{-15}
10^{-20}
0
5
10
15
20
25
30
35


QR
NE
e
d
10^{0}
10^{-5}
10^{-10}
10^{-15}
10^{-20}
10^{-25}
10^{-30}
0
5
10
15
20
25
30
35
40

1. Interpret the error behavior:
   Per your recommendation, I put the condition number of each d matrix some plots alongside error. This obviously shows a link between testing error and the condition of the matrix. For normal equations method, $k(A) \longrightarrow k(A^TA) \longrightarrow k(A)^2$ is apparent. NE's min error only gets to about the square root of QR's min error in the shown plots.

   One strange thing is seeing QR begin to regain error at about two times the d at which the min error bottoms out. I wish my linear algebra was as quick as my basic algebra so I could investigate this a little bit, but I still got a lot of time til that point.

   When thinking about wanting lower degrees of a polynomial to fit more data points, it seems NE is best at that. However, if we want a little more precision in sacrifice of gaining polynomial degrees, we can opt for QR.

   Also I kind of wanted you to make us plot testing error against training error, or the ratio between them for each method, but I don't know if there is any valuable insight behind that.

Code!

```
function y = fm(x)
%FM Summary of this function goes here
%   Detailed explanation goes here
    y = cos(3*pi*x) + sin(2*pi*x);
end

function [a1,a2,a3] = main(argv)


    frange = [-1,1];
    interval = argv;

    %creating equal points of testing and training data
    %   for easier error calculation alegbra
    xstrain = transpose(linspace(frange(1),frange(2),interval));
    fxstrain = fm( xstrain );

    rxstest = transpose(frange(1)+ (frange(2)-frange(1)).*rand(interval,1));
    fxstest = transpose(fm( rxstest ));


    %vandermonde for both training and testing data
    vpA = fliplr(vander(xstrain));
    vpAtest = fliplr(vander(rxstest));

    %polydsnuts is a cell holding all coeffiecent solutions
    %   for each d
    polydsnuts= cell(interval,2);

    %going to hold all error values for each d
    dterror= zeros(interval,2);

    %holding condition number for each d
    condd = zeros(1,interval);

    for d = 1:1:interval
        disp("degree poly: "); disp(d-1);

        %get vandermonde of up to d columns / coeffiencts
        vAtrain = vpA(:,1:d);
```

```
        vAtest = vpAtest(:,1:d);
        condd(d) = cond(vAtest);


        %get coeffiecients of polynomial degree d-1 for both
        %   methods and store in cell
        polyqr = thinqr(vAtrain, fxstrain);
        polyne = normeq(vAtrain, fxstrain);
        polydsnuts{d,1} = polyqr;
        polydsnuts{d,2} = polyne;

        %residual
        %(true values - (respectivevandermon * our coef of polyapprox))^2
        qrteste = (fxstest - (vAtest * polyqr)).^2;
        neteste = (fxstest - (vAtest * polyne)).^2;

        %frfr error
        pqrterror = sum(qrteste) / sum(fxstest.^2);
        pneterror = sum(neteste) / sum(fxstest.^2);

        %normin and squarin resid to get true error
        %   then store in matrix
        dterror(d,1) = pqrterror^1/2;
        dterror(d,2) = pneterror^1/2;
        %polyplotne = newpoly(polyne);
        %polyplotqr = newpoly(polyqr);

    end

    %chosen outputs
    a1 = polydsnuts;
    a2 = condd;
    a3 = dterror;
end
```

2pc I then plotted error vs d and condition using the outputs
I also looked at the pretty graphs from all the polynomials generated