4X03 – Assignment 2

Jeff Suitor 4001386789

* 1. The accuracy of the solution is reduced as the conditional increases and the residual error increases as the conditional increases. This is because as the condition number increases the matrix becomes more susceptible and becomes more susceptible to errors such as those that occur when computing the inverse.
  2. My results were not as accurate as MATLAB’s for both the partial pivoting and scaled partial pivoting. It is likely that MATLAB is taking extra steps to ensure result accuracy.
  3. Large residuals are likely caused by subtraction of values that are very close numerically causing a loss-of-significance error which gives the large residual.
  4. Scaled partial pivoting can sometimes improve the accuracy of results such as when in a 2x2 matrix of the form



Scaled partial pivoting would pick row 2 as the pivot row which would be more accurate than row 1 as row one differs by orders of magnitude where row 2 does not

1. To test if MATLAB’s lu function does scaled or partial pivoting you would create a 2x2 matrix of the form



You would then perform lu decomposition on it and look at the permutation matrix. If the permutation is identical to the identity matrix for a 2x2 than it does not do scaled partial pivoting but if the permutation is of the form



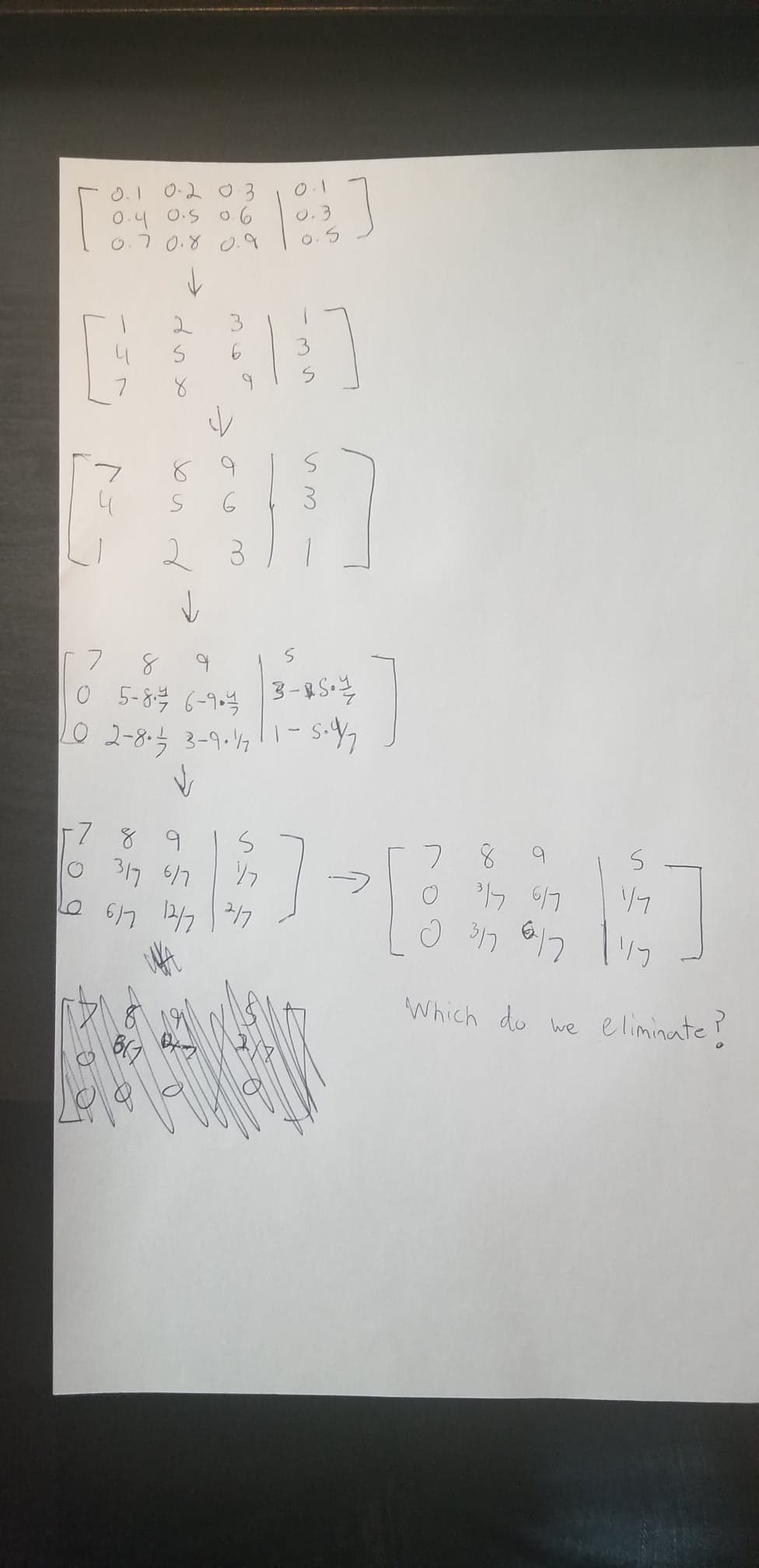
Then MATLAB does scaled partial pivoting.

1. 1. .1× (.5×.9 − .8×.6) − .2×(.4×.9 − .7×.6) + .3×(.4×.8 − .7×.5)

.1 x (-0.03) - .2 x (-0.06) + .3 x (-0.03)

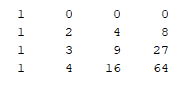
-0.12 + 0.12 = 0

det(A) = 0, therefore singular



Simple Gaussian elimination fails at this point because the we will end up with a zero row.

* 1. MATLAB when using the backslash operator will issue a warning if rcond is between 0 and eps but will proceed with the calculation according to the MATLAB website. This solution is an approximation and can highly vary in accuracy depending on the conditions of the matrix. However, the accuracy can vary wildly as on the MATLAB they give an example with a low residual and a 0 rcond which results in an inf as an output



=

* 1. L0(x) = [(x-2)(x-3)(x-4)] / [(0-2)(0-3)(0-4)] = (x-2)(x-3)(x-4)/-24

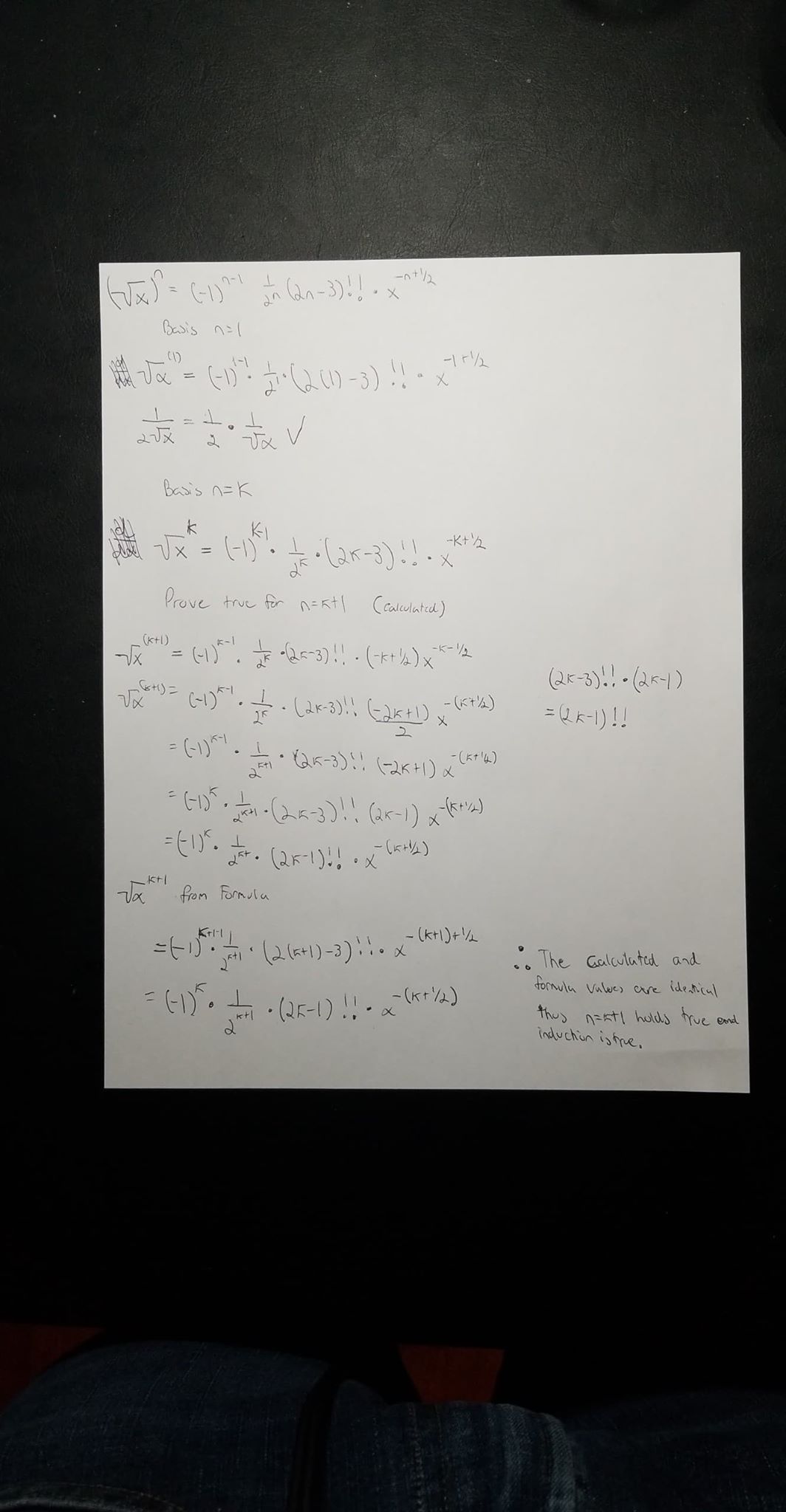
…

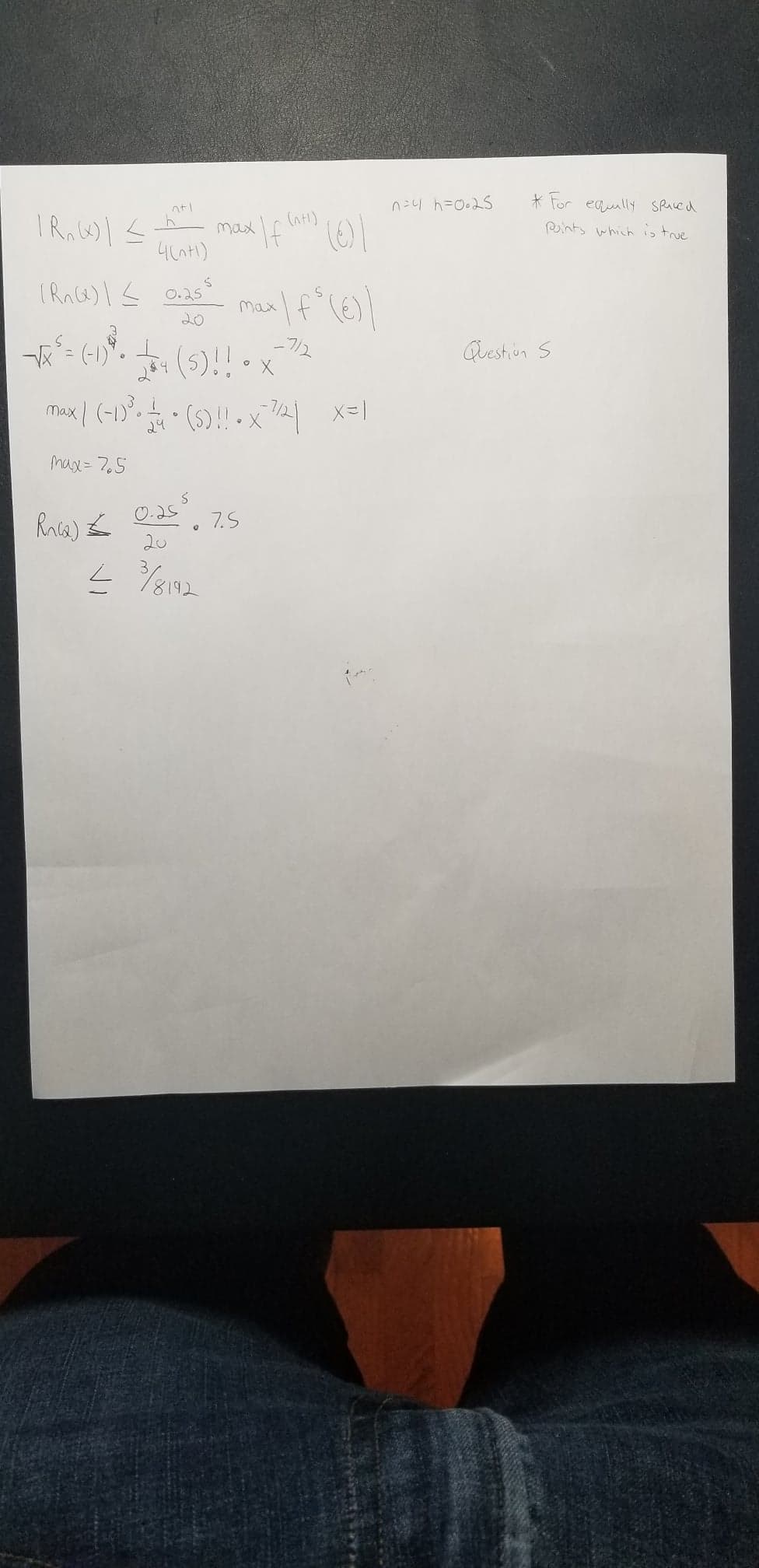
p(x) = 7\*L0 + 11 \*L2 + 28\*L3 + 63\*L4

p(x) = x3 – 2x + 7

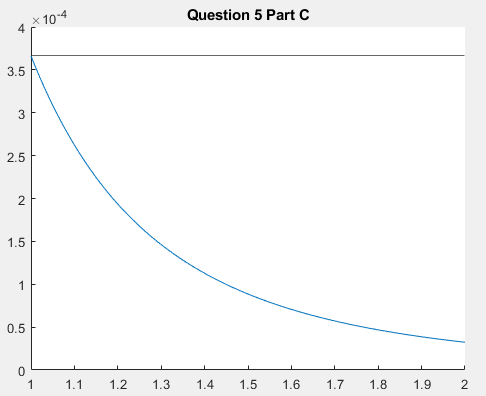
* 1. p(x) = 7 + 2(x-0) + 5(x-0)(x-2) + 1(x-0)(x-2)(x-3)

p(x) = x3 – 2x + 7

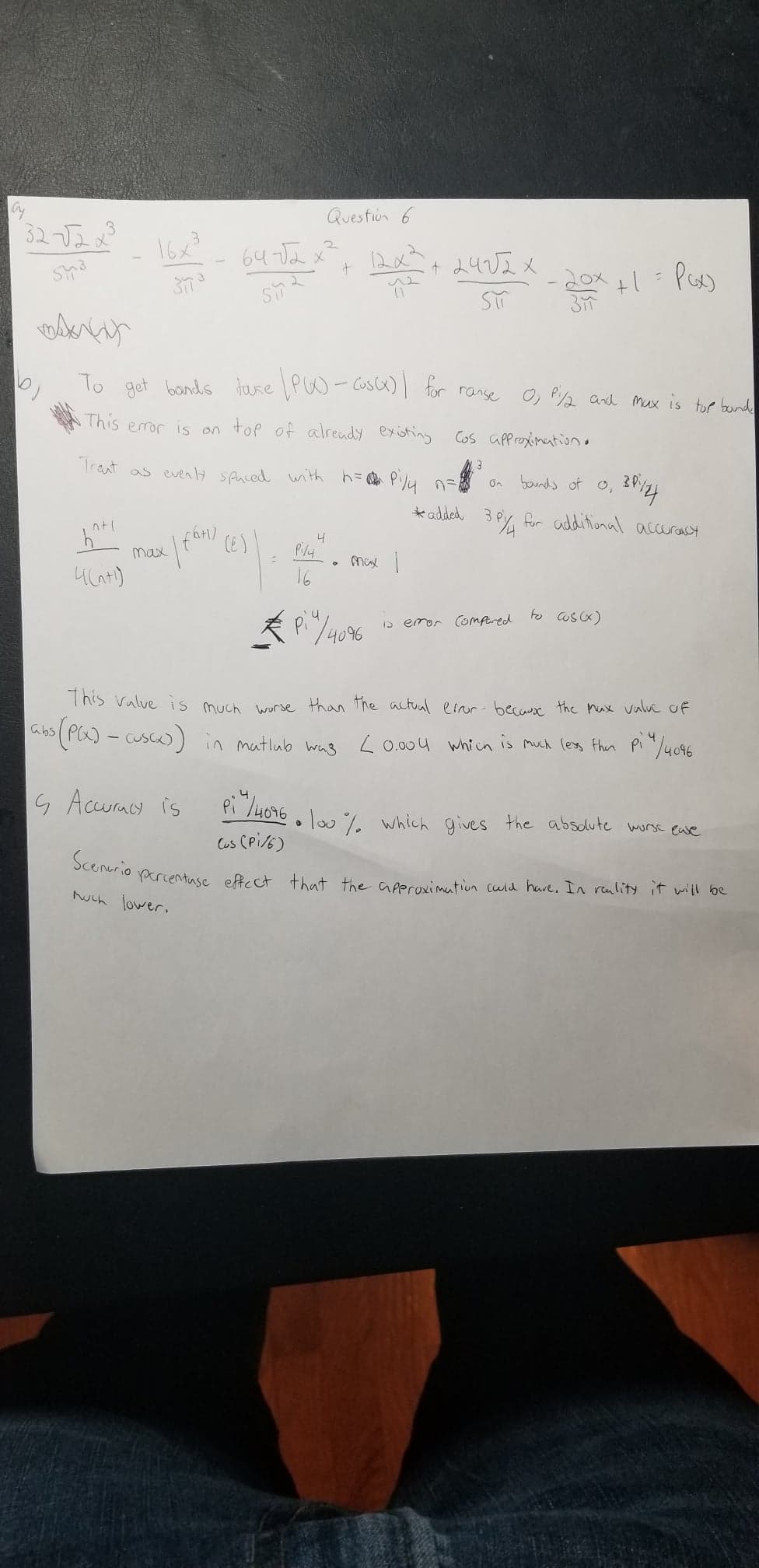




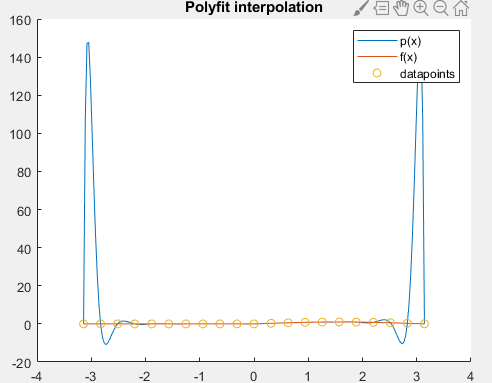
c.

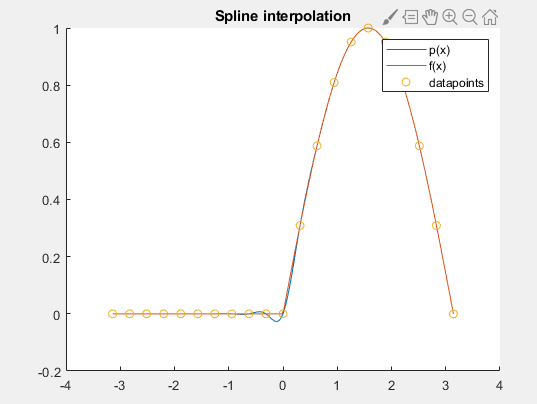


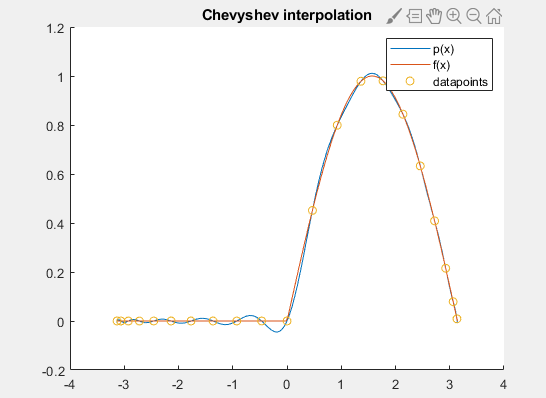
The thin black line is the maximum error bounds.











* 1. The difference between part a and part c is that the Chebyshev points remove the error at the bounds of the function reducing the overall error.

Appendix A: Code for Question 1

solve\_all.m

function [xm, xpp, xspp] = solve\_all(A,b)

xm = A\b;

[n, m] = size(A);

if n ~= m

disp("ERROR: Matrix not N x N")

return

end

[L,U,P] = lu\_pp(A);

inv\_P = inv(P);

B = b;

for i=1:n-1

max\_value=max(abs(inv\_P(i:n,i))); % Find the pivot row

for j=i:n

if(abs(inv\_P(j,i))== max\_value)

swap\_index=j; % Get the swap index

break;

end

end

inv\_P([i,swap\_index],:) = inv\_P([swap\_index,i],:);

B([i,swap\_index],:) = B([swap\_index,i],:);

end

[LS,US,PS] = lu\_spp(A);

inv\_PS = inv(PS);

BS = b;

for i=1:n-1

max\_value=max(abs(inv\_PS(i:n,i))); % Find the pivot row

for j=i:n

if(abs(inv\_PS(j,i))== max\_value)

swap\_index=j; % Get the swap index

break;

end

end

inv\_PS([i,swap\_index],:) = inv\_PS([swap\_index,i],:);

BS([i,swap\_index],:) = BS([swap\_index,i],:);

end

xpp = lu\_solve(L,U,B);

xspp = lu\_solve(LS,US,BS);

end

lu\_spp.m

function [L, U, P] = lu\_spp(A)

[m, n] = size(A);

if n ~= m

disp("ERROR: Matrix not N x N")

return

end

% Create matrices

L=eye(n);

P=eye(n);

U=A;

ratio\_vector = max(abs(A')); % Create the ratio vector

for i=1:n-1

temp = U;

div\_temp = temp ./ ratio\_vector'; % Divide the matrix by the transpose of this vector

max\_value=max(abs(div\_temp(i:n,i))); % Find the mmax column value

for j=i:n

if(abs(div\_temp(j,i))== max\_value)

swap\_index=j; % Get the swap index

break;

end

end

% Swap rows in matrix

U([i,swap\_index],i:n) = U([swap\_index,i],i:n);

L([i,swap\_index],1:i-1) = L([swap\_index,i],1:i-1);

P([i,swap\_index],:) = P([swap\_index,i],:);

% Elimination

for j=i+1:n

L(j,i)=U(j,i)/U(i,i);

U(j,i:n)=U(j,i:n)-L(j,i)\*U(i,i:n);

end

end

end

lu\_solve.m

function x = lu\_solve(L,U,b)

y = inv(L) \* b;

x = inv(U) \* y;

end

lu\_pp.m

function [L, U, P] = lu\_pp(A)

[m, n] = size(A);

if n ~= m

disp("ERROR: Matrix not N x N")

return

end

% Create matrices

L=eye(n);

P=eye(n);

U=A;

for i=1:n-1

max\_value=max(abs(U(i:n,i))); % Find the pivot row

for j=i:n

if(abs(U(j,i))== max\_value)

swap\_index=j; % Get the swap index

break;

end

end

% Swap rows in matrix

U([i,swap\_index],i:n) = U([swap\_index,i],i:n);

L([i,swap\_index],1:i-1) = L([swap\_index,i],1:i-1);

P([i,swap\_index],:) = P([swap\_index,i],:);

% Gaussian elimination

for j=i+1:n

L(j,i)=U(j,i)/U(i,i);

U(j,i:n)=U(j,i:n)-L(j,i)\*U(i,i:n);

end

end

end

linearsolve.dat

n cond(A) Matlab lu\_pp lu\_spp

error residual error residual error residual

Vandermonde matrices (flipped with fliplr)

5 2.6e+04 0.0e+00 0.0e+00 7.1e-15 2.6e-13 0.0e+00 0.0e+00

6 7.3e+05 4.1e-11 7.8e-13 4.9e-11 2.4e-12 3.4e-11 1.9e-12

7 2.4e+07 3.7e-11 2.3e-13 8.9e-10 2.0e-10 2.0e-09 2.4e-10

8 9.5e+08 3.5e-09 1.7e-10 6.0e-08 1.1e-08 5.1e-08 1.5e-09

9 4.2e+10 1.1e-07 3.6e-09 1.2e-06 2.0e-08 1.2e-06 3.4e-07

10 2.1e+12 1.7e-05 2.2e-07 1.1e-04 4.6e-05 2.6e-05 5.7e-06

11 1.2e+14 2.2e-03 1.9e-06 3.8e-03 1.4e-03 2.2e-03 1.8e-03

12 7.0e+15 3.8e-01 1.9e-04 3.1e-01 1.6e-02 4.6e-02 5.3e-01

13 1.1e+18 1.5e+00 2.5e-03 2.3e+01 7.2e-01 6.6e+01 2.8e+01

14 2.1e+18 2.1e+02 9.5e-02 1.6e+03 1.2e+02 9.7e+02 5.2e+03

15 9.9e+19 9.4e+04 8.7e+00 1.1e+05 3.2e+04 1.8e+05 1.6e+05

Hilbert matices

2 1.9e+01 9.0e-16 0.0e+00 6.0e-16 0.0e+00 6.0e-16 0.0e+00

3 5.2e+02 1.8e-14 2.2e-16 1.2e-15 2.2e-16 1.2e-15 2.2e-16

4 1.6e+04 5.4e-13 2.5e-16 1.5e-12 2.2e-16 7.8e-13 5.6e-16

5 4.8e+05 6.3e-12 2.5e-16 8.4e-13 6.0e-16 6.2e-13 7.5e-16

6 1.5e+07 6.8e-10 3.1e-16 9.8e-11 1.6e-16 1.3e-09 1.1e-16

7 4.8e+08 3.0e-08 5.0e-16 1.6e-08 9.6e-16 5.3e-09 9.4e-16

8 1.5e+10 6.7e-07 5.2e-16 9.8e-07 2.4e-15 7.3e-07 4.7e-16

9 4.9e+11 2.8e-05 5.6e-16 2.5e-05 2.8e-15 5.9e-05 7.9e-15

10 1.6e+13 5.2e-04 5.2e-16 1.7e-03 1.8e-15 9.0e-04 2.4e-14

11 5.2e+14 1.5e-02 4.2e-16 3.2e-02 8.6e-15 2.2e-02 6.0e-15

12 1.7e+16 9.0e-01 7.3e-16 4.8e-01 2.9e-14 1.3e+00 4.0e-14

13 6.9e+17 9.4e+00 7.9e-16 7.2e+00 2.6e-14 8.5e+00 9.4e-14

14 4.3e+17 9.5e+00 6.9e-16 1.5e+02 2.7e-13 1.4e+01 4.9e-14

15 4.5e+17 1.5e+01 1.3e-15 1.4e+01 6.5e-13 1.8e+01 3.7e-14

Random matrices

10 6.1e+01 5.0e-15 2.0e-15 1.3e-14 5.3e-15 1.3e-14 4.8e-15

100 2.8e+03 9.1e-13 1.8e-13 6.6e-13 9.5e-13 1.4e-12 1.5e-12

1000 2.9e+04 3.5e-11 2.2e-11 1.9e-10 6.0e-10 2.3e-10 1.2e-09

2000 3.0e+05 5.9e-10 1.2e-10 8.4e-10 4.0e-09 1.0e-09 4.5e-09