

Problem 1:

a. Without pivoting

$$\begin{array}{cccccc} 3 & 4 & 3 & x_1 & 10 \\ [1 & 5 & -1] & [x_2] & = & [7] \\ 6 & 3 & 7 & x_3 & 15 \end{array}$$

$$\begin{array}{cccc} 3 & 4 & 3 & 10 \\ A|b = [1 & 5 & -1 & 7] \\ 6 & 3 & 7 & 15 \end{array}$$

Multiply first row by 2 and subtract from third row we get:

$$\begin{array}{cccc} 3 & 4 & 3 & 10 \\ A|b = [1 & 5 & -1 & 7] \\ 0 & -5 & 1 & -5 \end{array}$$

Multiply first row by 1/3 and subtract from second row we get:

$$\begin{array}{cccc} 3 & 4 & 3 & 10 \\ A|b = [0 & 3.6667 & -2 & 3.6667] \\ 0 & -5 & 1 & -5 \end{array}$$

Multiply second row by -5/3.6667 and subtract from third row we get:

$$\begin{array}{cccc} 3 & 4 & 3 & 10 \\ A|b = [0 & 3.6667 & -2 & 3.6667] \\ 0 & 0 & -1.7272 & 0 \end{array}$$

As a result we have:

$$x_3 = 0$$

$$x_2 = 3.6667/3.6667 = 1$$

$$x_1 = (10 - 4 \cdot x_2)/3 = 2$$

b. With pivoting

$$\begin{array}{cccccc} 3 & 4 & 3 & x_1 & 10 \\ [1 & 5 & -1] & [x_2] & = & [7] \\ 6 & 3 & 7 & x_3 & 15 \end{array}$$

$$\begin{array}{cccc} 3 & 4 & 3 & 10 \\ A|b = [1 & 5 & -1 & 7] \\ 6 & 3 & 7 & 15 \end{array}$$

Interchange row 1 and row 3 we have:

$$\begin{array}{cccc} 6 & 3 & 7 & 15 \\ A|b = [1 & 5 & -1 & 7] \\ 3 & 4 & 3 & 10 \end{array}$$

Multiply first row by 1/6 and subtract from second row we get:

$$A|b = \begin{bmatrix} 6 & 3 & 7 & 15 \\ 0 & 4.5 & -2.6667 & 4.5 \\ 3 & 4 & 3 & 10 \end{bmatrix}$$

Multiply first row by 3/6 and subtract from third row we get:

$$A|b = \begin{bmatrix} 6 & 3 & 7 & 15 \\ 0 & 4.5 & -2.6667 & 4.5 \\ 0 & 2.5 & -0.5 & 2.5 \end{bmatrix}$$

Multiply second row by 2.5/4.5 and subtract from third row we get:

$$A|b = \begin{bmatrix} 6 & 3 & 7 & 15 \\ 0 & 4.5 & -2.6667 & 4.5 \\ 0 & 0 & 9.8150 \cdot 10^{-1} & 0 \end{bmatrix}$$

As a result, we have

$$x_3 = 0$$

$$x_2 = (4.5 - 0)/4.5 = 1$$

$$x_1 = (15 - 0 - 3 \cdot 1)/6 = 2$$

Problem 2:

```
>> a2_2
ε      |x1 - l|      |x2 - ε|/ε      cond(A)
2      0      0      2.618034e+00
Warning: Matrix is close to singular or badly scaled. Results may be inaccurate. RCOND = 5.551115e-17.
> In a2_2 (line 6)

7.450581e-09      1      134217727      34230795238752088
1.490116e-08      0      0      23251426040965572
2.235174e-08      1.111111e-01      4.971027e+06      7415325565567722
>>
```

From this experiment we can draw the conclusion that when the value of epsilon is around square root of machine epsilon, we get extremely large relative errors and we lose around 15 digits in the calculations. When epsilon is relatively large, we get much less error.

Because the $\text{cond}(A)$ and $\|r\|/\|b\|$ are both quite large when the value of epsilon is around square root of machine epsilon. That is expected because machine epsilon is the smallest unit representable by the machine. When we do calculations with values like this, we will for sure get more errors. The value of $\text{cond}(A)$ basically suggests how sensitive the result of the calculation is to any errors or changes in the input. This makes sense because we are dealing with values around the limitation of the machine so a very small change can make a huge difference.

Problem 3:

a.

```
Week01.mlx GEPP.m backward.m GE.m main_ge.m +
1 function B = GE(A)
2     A_size = size(A);
3     n = A_size(1);
4     B = zeros(n,n);
5     for k = 1:n
6         %store U
7         for j = k:n
8             sum = 0;
9             for s = 1:k-1
10                sum = sum + B(k, s) * B(s, j);
11            end
12            B(k, j) = A(k, j) - sum;
13        end
14        %store L
15        for i = k+1:n
16            sum = 0;
17            for s = 1:k-1
18                sum = sum + B(i, s) * B(s, k);
19            end
20            B(i, k) = (A(i, k) - sum) / B(k, k);
21        end
22    end
23 end
```

```
Editor - C:\Users\lizhi\OneDrive\Documents\MATLAB\GEPP.m
a2_7_b.m a2_7_c.m newton.m hornerN.m main_interp.m a2_6_a.m a2_6_b.m GEPP.m test.m main_ge.m backward.m +
1 function [B, ipivot] = GEPP(A)
2     n = length(A);
3     ipivot = 1:n;
4     L = zeros(n,n);
5     for k = 1:n
6         max = 0;
7         q = 0;
8         for i=k:n
9             if abs(A(ipivot(i),k)) > max
10                max = abs(A(ipivot(i),k));
11            end
12            q = i;
13        end
14        ipivot([q k]) = ipivot([k q]);
15        for i = k+1:n
16            L(ipivot(i), k) = A(ipivot(i), k) / A(ipivot(k), k);
17            A(ipivot(i), :) = A(ipivot(i), :) - A(ipivot(k), :)*L(ipivot(i), k);
18        end
19    end
20    B = L+A;
21 end
```

```

Editor - C:\Users\lizhi\OneDrive\Documents\MATLAB\backward.m
a2_7_b.m a2_7_c.m newton.m hornerN.m main_interp.m a2_6_a.m a2_6_b.m GEPP.m test.m main_ge.m backward.m
1 function x = backward(B, b, ipivot)
2     b_size = size(b);
3     n = b_size(1);
4     x = zeros(n, 1);
5     y = zeros(n, 1);
6     %solve y
7     for k=1:n
8         sigma_akj_xj = 0;
9         for j = k-1:-1:1
10            sigma_akj_xj = sigma_akj_xj+B(ipivot(k),j)*y(ipivot(j));
11        end
12        y(ipivot(k)) = b(ipivot(k))-sigma_akj_xj;
13    end
14    %solve x
15    for k=n:-1:1
16        sigma_akj_xj = 0;
17        for j = k+1:n
18            sigma_akj_xj = sigma_akj_xj+B(ipivot(k),j)*x(ipivot(j));
19        end
20        x(ipivot(k)) = (y(ipivot(k))-sigma_akj_xj)/B(ipivot(k),k);
21    end
22 end

```

b.

```

newton.m hornerN.m main_interp.m Week05.mlx main_ge.m
1 n = 2000;
2 x = ones(n,1);
3
4 num_of_matrices = 5;
5 A = rand(n,n,num_of_matrices);
6 b = zeros(n,num_of_matrices);
7 for i=1:num_of_matrices
8     b(:,i) = A(:, :, i)*x;
9 end
10
11 fprintf('exp# \t A\b \t no pivoting \t pivoting \t cond(A) \n');
12 for i=1:num_of_matrices
13     B = GE(A(:, :, i));
14     x_GE=backward(B,b(:,i),1:n);
15     [B,ipivot] = GEPP(A(:, :, i));
16     x_GEPP=backward(B,b(:,i),ipivot);
17     fprintf('%i \t % .2e \t % .2e \t % .2e \t %i \n', i, norm(x-A(:, :, i)\b(:,i))/norm(x), norm(x-x_GE)/norm(x), norm(x-x_GEPP)/norm(x), cond(A(:, :, i)));
18 end

```

c.

```

>> main_ge
exp#      A\b      no pivoting      pivoting      cond(A)
1      9.53e-12      3.57e-10      3.46e-11      4.804432e+05
2      8.09e-13      1.17e-10      1.72e-11      1.229722e+05
3      8.41e-12      8.20e-10      4.11e-11      4.953358e+05
4      1.03e-12      3.36e-10      2.32e-11      1.916369e+05
5      5.23e-12      1.78e-09      1.46e-10      1.349748e+06
>>

```

d) Based on the experiment result, we can tell that the greater the $\text{cond}(A)$ value is for a matrix, the more likely the result will be more inaccurate. This is expected because the condition number of a matrix is suggesting how sensitive the result of the calculation is to any errors or changes in the input.

Problem 4:

For degree five, $n = 5$.

Then we have $M = \max_{a \leq t \leq b} |f^{(n+1)}(t)| = e$ because $f^{(n+1)}(t)$ is the largest when $t = 1$.

Therefore, I will expect an error of at most $M/(4(n+1)) * h^{(n+1)} = e/(4*6) * ((1-0)/5)^6 = 7.248751543e-6$ if I use this polynomial.

If I want a max error of $1e-8$, M is still equal to e because in the $[0, 1]$ interval e^x is the largest when $t = 1$ no matter how many points we take. Then we have $e/(4(n+1)) * (1/n)^{(n+1)} \leq 1e-8$. Substituting $n = 7$ we get the error of approximately $1.47353e-8$. So we need more precision. Substituting $n = 8$ we get the error of approximately $5.62577e-10$ which is less than $1e-8$. So we need at least 8 degree polynomial to achieve a maximum error of $1e-8$.

Problem 5:

- Using matlab's polyfit function, I got $y = 1.0247$ and error = $4.9234e-06$ when $x = 0.05$, I got $y = 1.0723$ and error = $3.0529e-05$ when $x = 0.15$.
- For the above polyfit, $n = 3$, $a = 0$, $b = 0.3$. calculating $f^{(n+1)}(0) * 1/(4*(n+1)) * h^{(n+1)}$ gives us an error bound of $5.8594e-06$.
- The error bound is larger than one of the actual errors but is smaller than the other one. Because we were given only the data points that are not accurate enough (only 4 digits after the decimal point). If we directly calculate the values of the y_i points by substituting the x_i values into the original function $y = \sqrt{x+1}$ in matlab, we would get more accurate results and the

actual errors will fall within the error bound.



The image shows a MATLAB Editor window with a script named 'a2_5.m'. The script defines a polynomial fit and calculates the error at two points. The Command Window shows the results of the script execution.

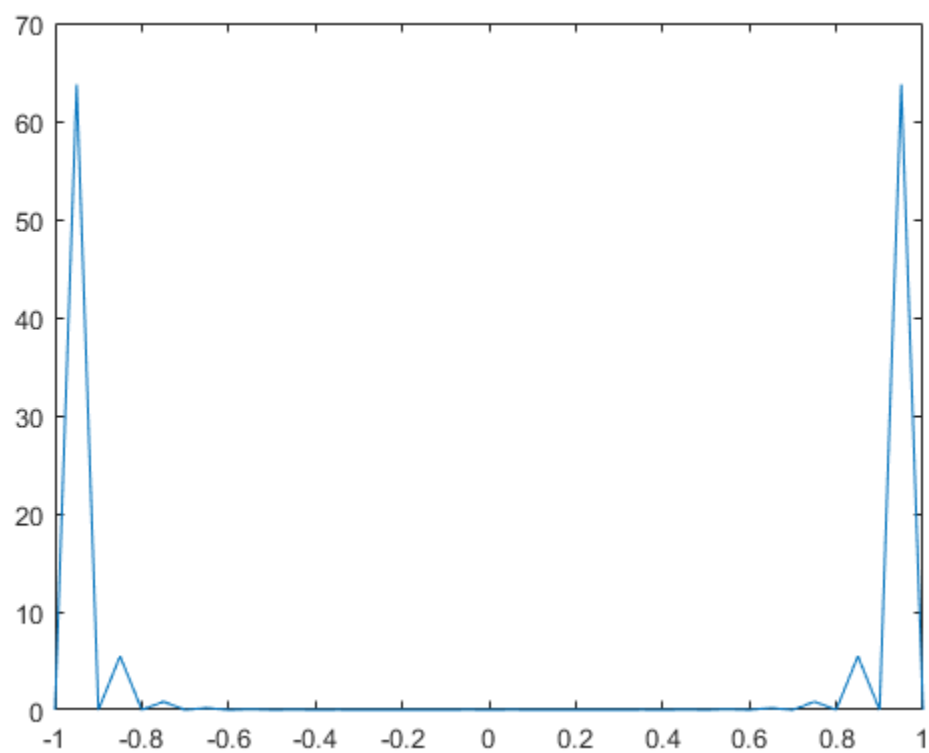
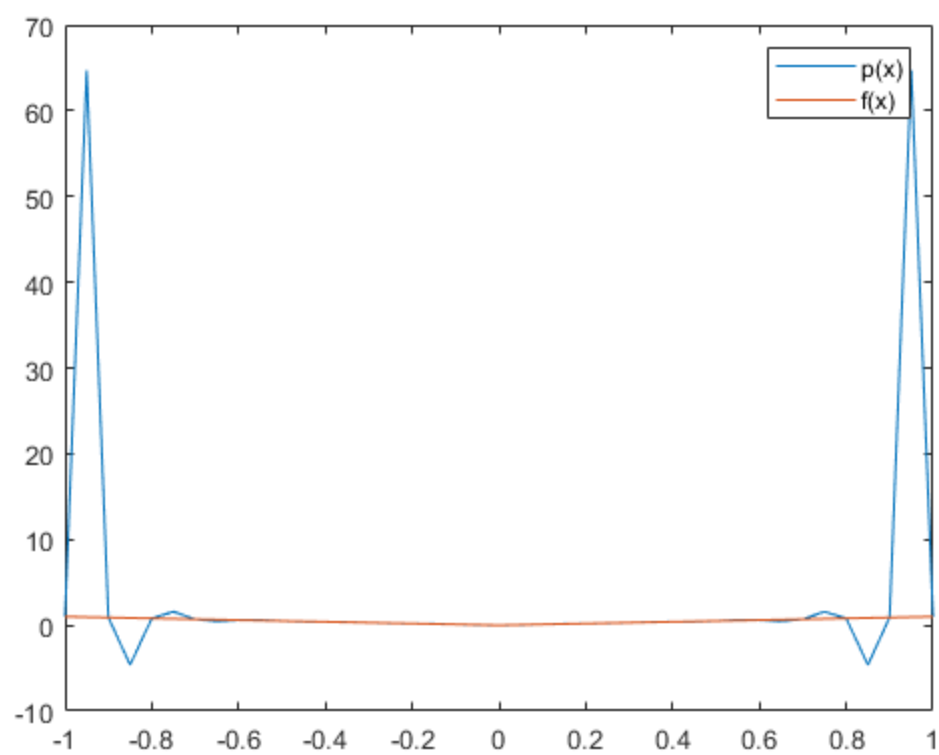
```
1 b = 0.3;  
2 a = 0;  
3 n = 3;  
4 x = linspace(a,b,n+1);  
5 y = [1.0000 1.0488 1.0954 1.1402];  
6 p = polyfit(x, y, n);  
7 x1 = 0.05; x2 = 0.15;  
8 f = @(x) sqrt(x+1);  
9 y1 = polyval(p,x1);  
10 y2 = polyval(p,x2);  
11 error_x1 = abs(f(x1)-y1);  
12 error_x2 = abs(f(x2)-y2);  
13 h = (b-a)/n;  
14 error_bound = (15/16/(0+1)^(7/2)) * (1/(4*(n+1)) * h^(n+1))
```

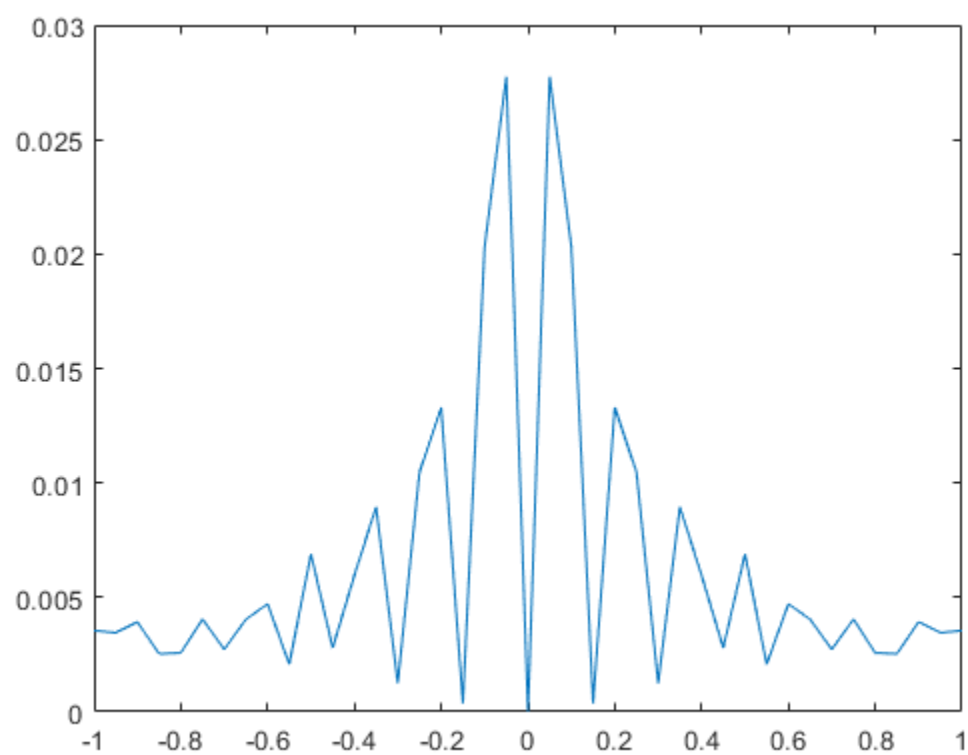
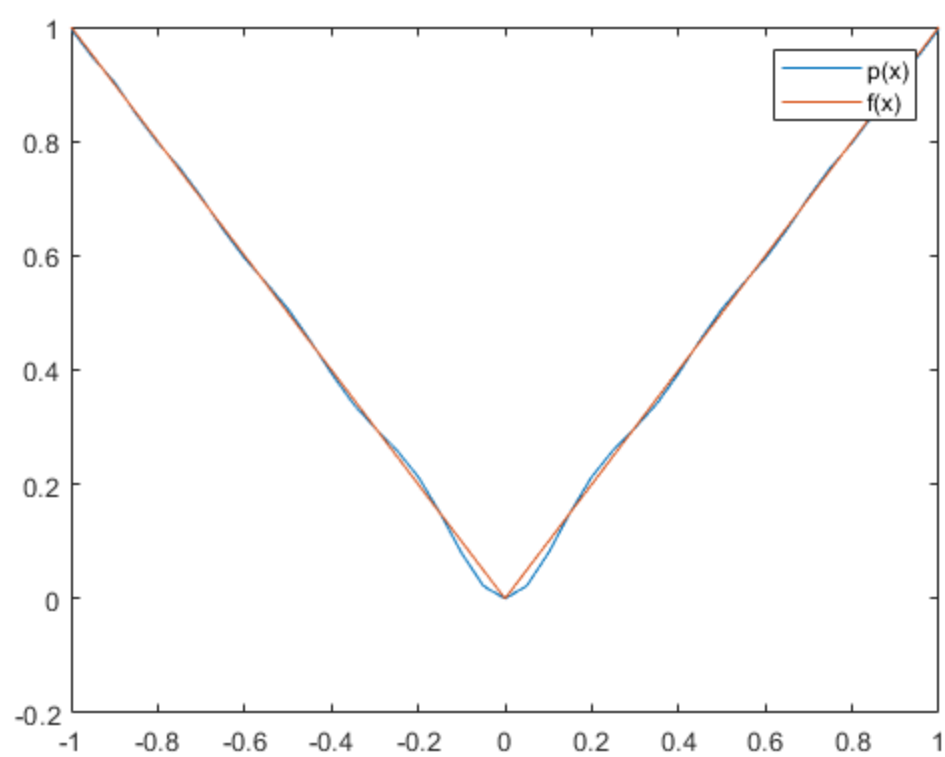
Command Window

New to MATLAB? See resources for [Getting Started](#).

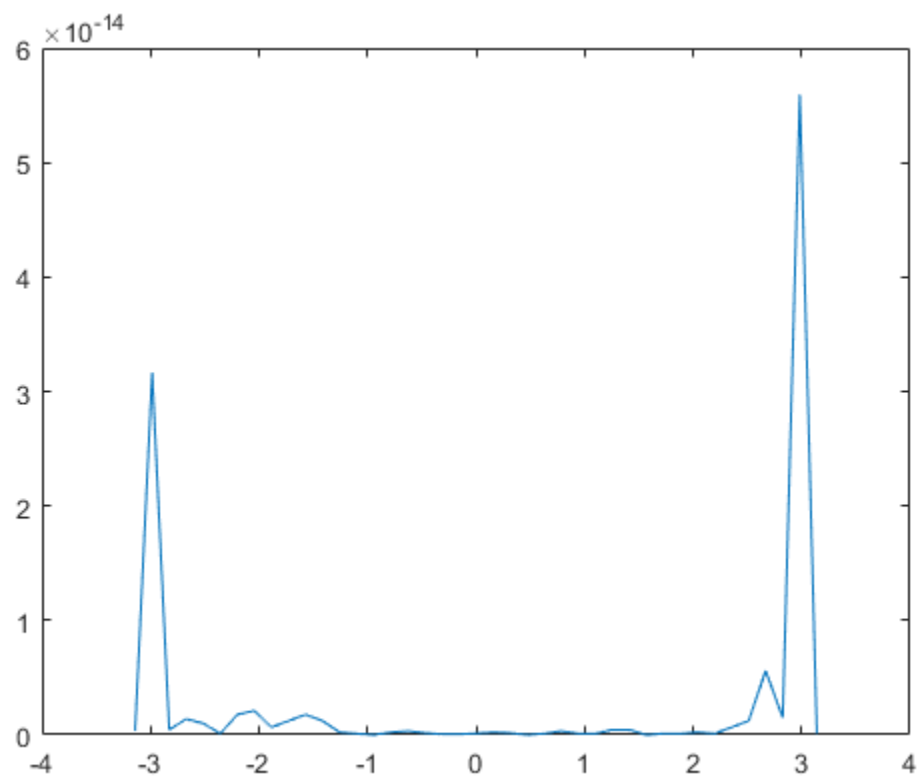
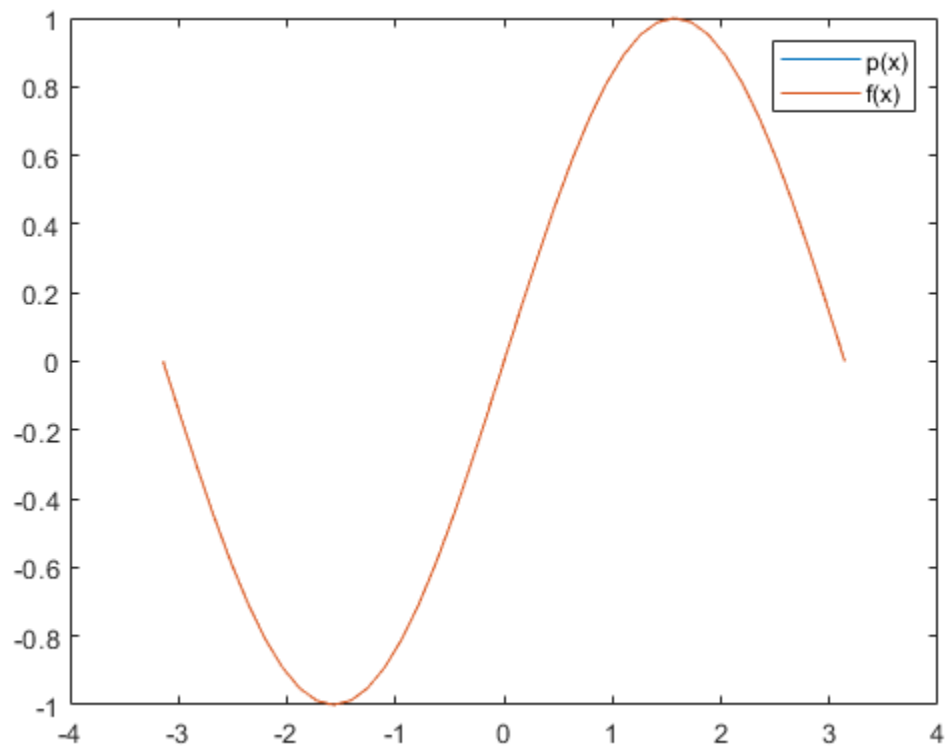
```
>> a2_5  
  
y1 =  
  
    1.0247  
  
y2 =  
  
    1.0723  
  
error_x1 =  
  
    4.9234e-06  
  
error_x2 =  
  
    3.0529e-05  
  
error_bound =  
  
    5.8594e-06
```

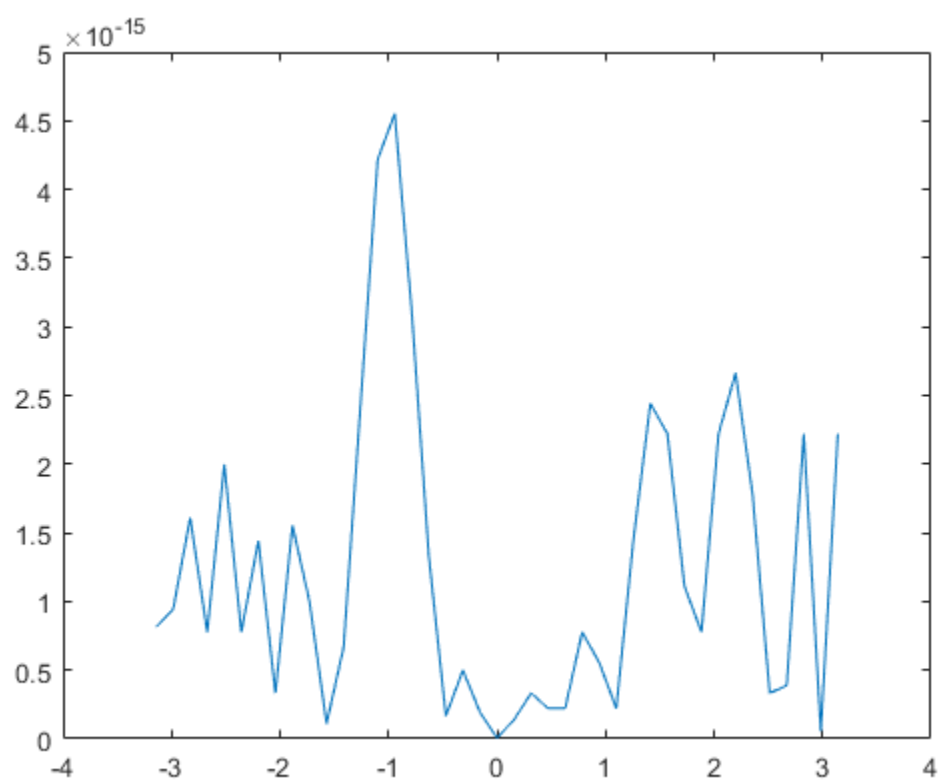
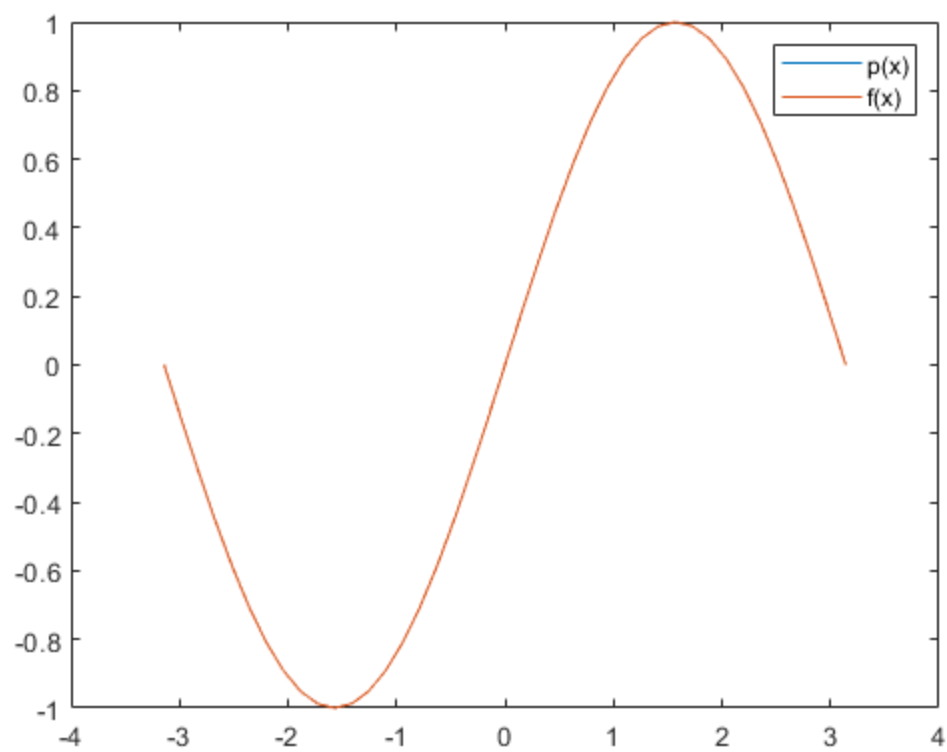
Problem 6:





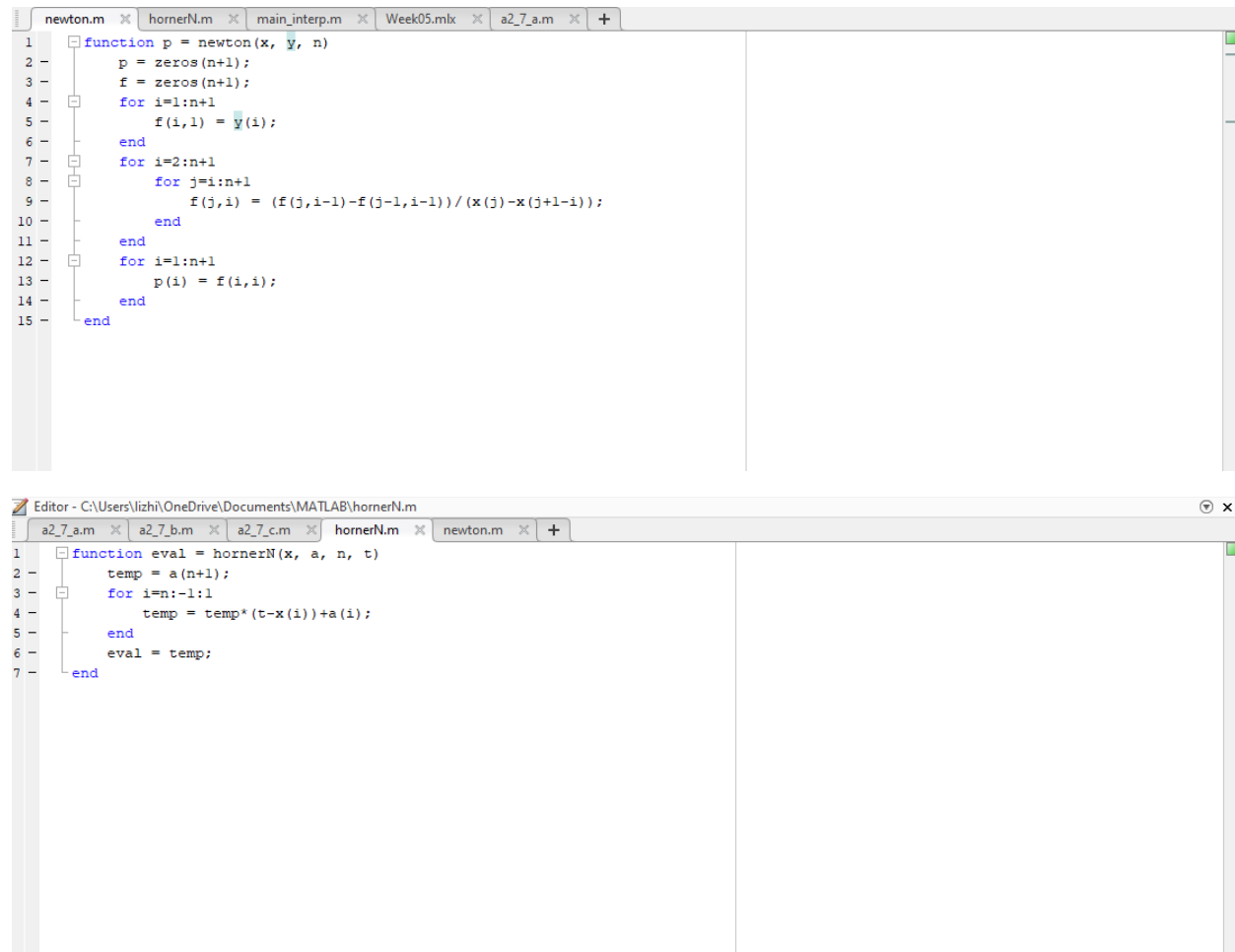
Problem 7:





The difference is that interpolated $|x|$'s errors are larger than that of $\sin(x)$. The reason is that $|x|$'s graph is more of a straight-line graph, and the degree of our interpolation is 20 which is relatively high. The nature of the high degree polynomials is that a polynomial of degree n has n zeros, if they are all real, the curves must cross the x -axis many times, making the resulting graph have a lot of oscillations. If the graph has a lot of oscillations, it is apparently not beneficial for interpolating some straight-line functions like $|x|$. But it will make interpolating more accurate for curvier or smoother functions like $\sin(x)$.

Problem 8:



```
1 function p = newton(x, y, n)
2     p = zeros(n+1);
3     f = zeros(n+1);
4     for i=1:n+1
5         f(i,1) = y(i);
6     end
7     for i=2:n+1
8         for j=i:n+1
9             f(j,i) = (f(j,i-1)-f(j-1,i-1))/(x(j)-x(j+1-i));
10        end
11    end
12    for i=1:n+1
13        p(i) = f(i,i);
14    end
15 end
```

```
1 function eval = hornerN(x, a, n, t)
2     temp = a(n+1);
3     for i=n:-1:1
4         temp = temp*(t-x(i))+a(i);
5     end
6     eval = temp;
7 end
```