Problem 1:

a. Without pivoting

$$\begin{bmatrix} 3 & 4 & 3 \\ 1 & 5 & -1 \\ 6 & 3 & 7 \end{bmatrix}\begin{bmatrix} x1 \\ x2 \\ x3 \end{bmatrix} = \begin{bmatrix} 10 \\ 7 \\ 15 \end{bmatrix}$$

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

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

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

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

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

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

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

As a result we have:

X3 = 0

X2 = 3.6667/3.6667 = 1

X1 = (10-4*X2)/3 = 2

b. With pivoting

$$\begin{bmatrix} 3 & 4 & 3 \\ 1 & 5 & -1 \\ 6 & 3 & 7 \end{bmatrix}\begin{bmatrix} x1 \\ x2 \\ x3 \end{bmatrix} = \begin{bmatrix} 10 \\ 7 \\ 15 \end{bmatrix}$$

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

Interchange row 1 and row 3 we have:

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

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

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

Mutiply 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:

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

As a result, we have

x3 = 0

X2 = (4.5-0)/4.5 = 1

X1 = (15-0-3*1)/6 = 2

Problem 2:

```
>> a2_2
ε    |x1 − 1|    |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 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. The value of cond(A) basically suggests how sensitive the result of the calculation is to any errors or changes in the input.

Problem 3:

a.

Week01.mlx | GEPP.m | backward.m | GE.m | main_ge.m

```matlab
function B = GE(A)
    A_size = size(A);
    n = A_size(1);
    B = zeros(n,n);
    for k = 1:n
        %store U
        for j = k:n
            sum = 0;
            for s = 1:k-1
                sum = sum + B(k, s) * B(s, j);
            end
            B(k, j) = A(k, j) - sum;
        end
        %store L
        for i = k+1:n
            sum = 0;
            for s = 1:k-1
                sum = sum + B(i, s) * B(s, k);
            end
            B(i, k) = (A(i, k) - sum) / B(k, k);
        end
    end
end
```

Editor - C:\Users\lizhi\OneDrive\Documents\MATLAB\GEPP.m

+1 | 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

```matlab
function [B, ipivot] = GEPP(A)
    n = length(A);
    ipivot = 1:n;
    L = zeros(n,n);
    for k = 1:n
        max = 0;
        q = 0;
        for i=k:n
            if abs(A(ipivot(i),k)) > max
                max = abs(A(ipivot(i),k));
                q = i;
            end
        end
        ipivot([q k]) = ipivot([k q]);
        for i = k+1:n
            L(ipivot(i), k) = A(ipivot(i), k) / A(ipivot(k), k);
            A(ipivot(i), :) = A(ipivot(i), :) - A(ipivot(k), :)*L(ipivot(i), k);
        end
    end
    B = L+A;
end
```

Editor - C:\Users\lizhi\OneDrive\Documents\MATLAB\backward.m

```matlab
function x = backward(B, b, ipivot)
    b_size = size(b);
    n = b_size(1);
    x = zeros(n, 1);
    y = zeros(n, 1);
    %solve y
    for k=1:n
        sigma_akj_xj = 0;
        for j = k-1:-1:1
            sigma_akj_xj = sigma_akj_xj+B(ipivot(k),j)*y(ipivot(j));
        end
        y(ipivot(k)) = b(ipivot(k))-sigma_akj_xj;
    end
    %solve x
    for k=n:-1:1
        sigma_akj_xj = 0;
        for j = k+1:n
            sigma_akj_xj = sigma_akj_xj+B(ipivot(k),j)*x(ipivot(j));
        end
        x(ipivot(k)) = (y(ipivot(k))-sigma_akj_xj)/B(ipivot(k),k);
    end
end
```

b.

```matlab
n = 2000;
x = ones(n,1);

num_of_matrices = 5;
A = rand(n,n,num_of_matrices);
b = zeros(n,num_of_matrices);
for i=1:num_of_matrices
    b(:,i) = A(:,:,i)*x;
end

fprintf("exp# \t A\\b \t no pivoting \t pivoting \t cond(A) \n");
for i=1:num_of_matrices
    B = GE(A(:,:,i));
    x_GE=backward(B,b(:,i),1:n);
    [B,ipivot] = GEPP(A(:,:,i));
    x_GEPP=backward(B,b(:,i),ipivot);
    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)/no
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 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)<=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:

a. 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.
b. For the above polyfit, n = 3, a = 0, b = 0.3. calculating f(0.3)*1/(4*(n+1))*h^(n+1) gives us an error bound of 5.8594e-06.
c. 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 yi points by substituting the xi 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.

Editor - C:\Users\lizhi\OneDrive\Documents\MATLAB\a2_5.m

newton.m | hornerN.m | main_interp.m | Week05.mlx | main_ge.m | a2_2.m | a2_5.m

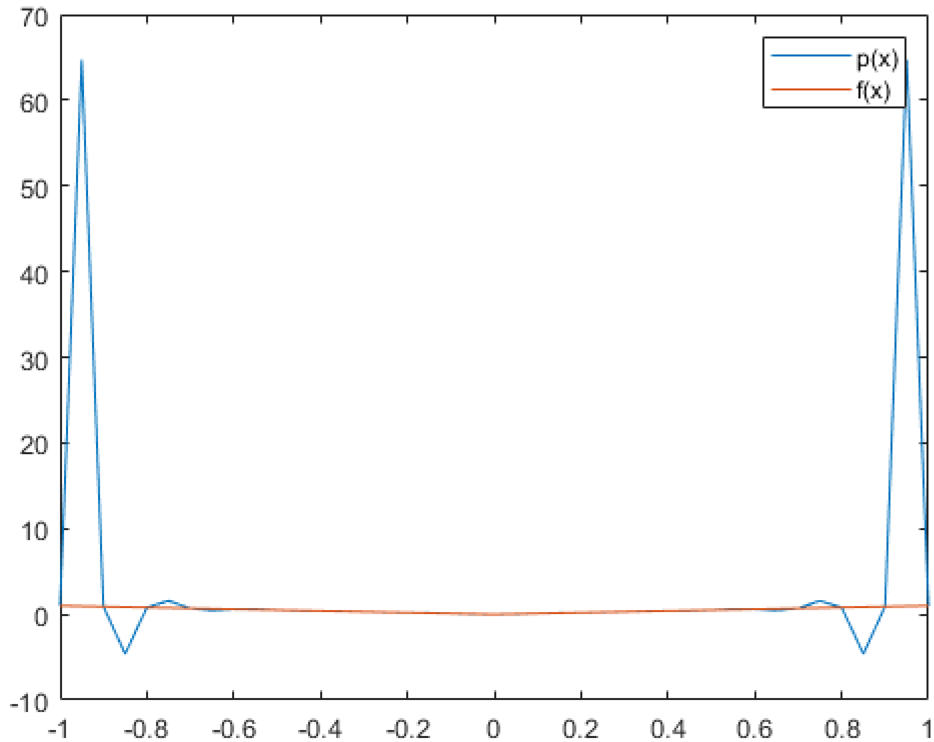
```
b = 0.3;
a = 0;
n = 3;
x = linspace(a,b,n+1);
y = [1.0000 1.0488 1.0954 1.1402];
p = polyfit(x, y, n);
x1 = 0.05; x2 = 0.15;
f = @(x) sqrt(x+1);
y1 = polyval(p,x1)
y2 = polyval(p,x2)
error_x1 = abs(f(x1)-y1)
error_x2 = abs(f(x2)-y2)
h  = (b-a)/n;
error_bound =  (15/16/(0+1)^(7/2))*(1/(4*(n+1))*h^(n+1))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

Command Window

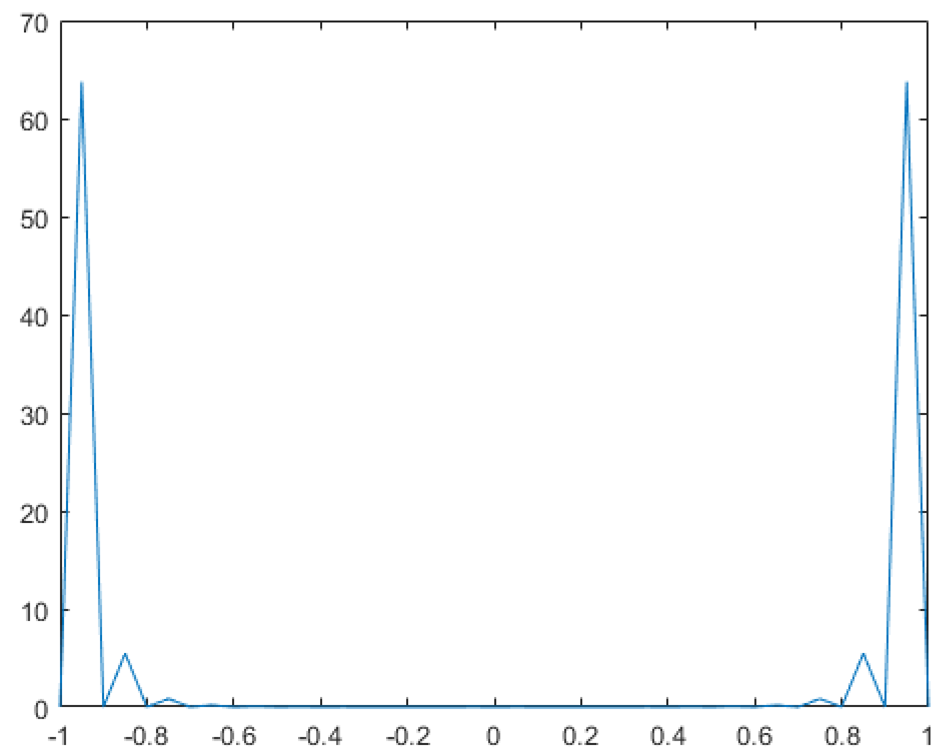
New to MATLAB? See resources for Getting Started.

```
>> a2_5

y1 =

    1.0247


y2 =

    1.0723


error_x1 =

   4.9234e-06


error_x2 =

   3.0529e-05


error_bound =

   5.8594e-06

fx >>
```

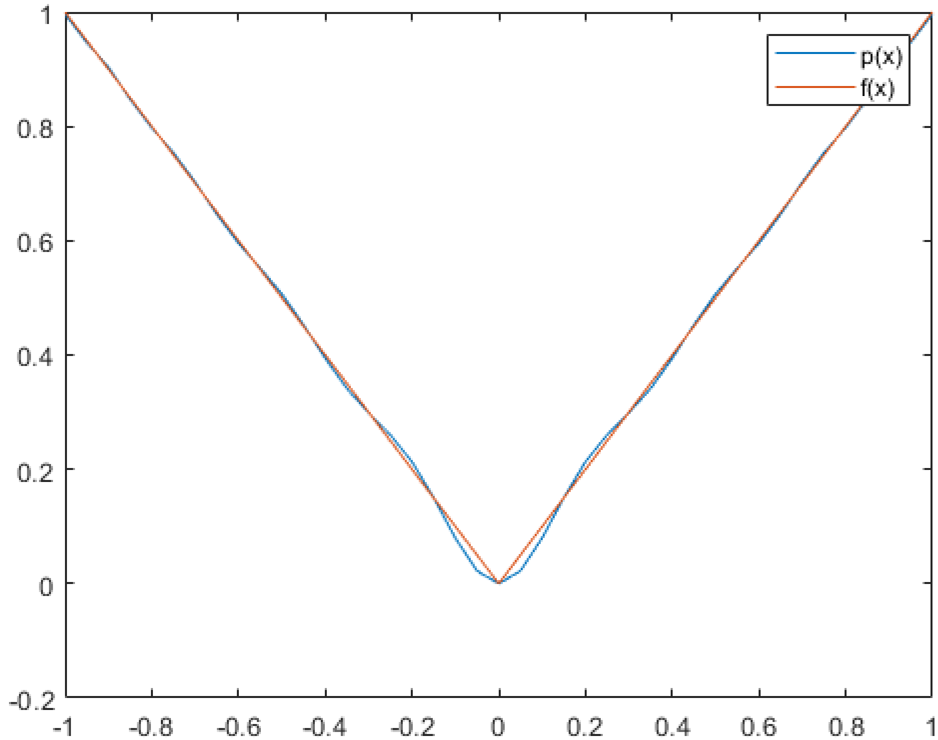
Problem 6:



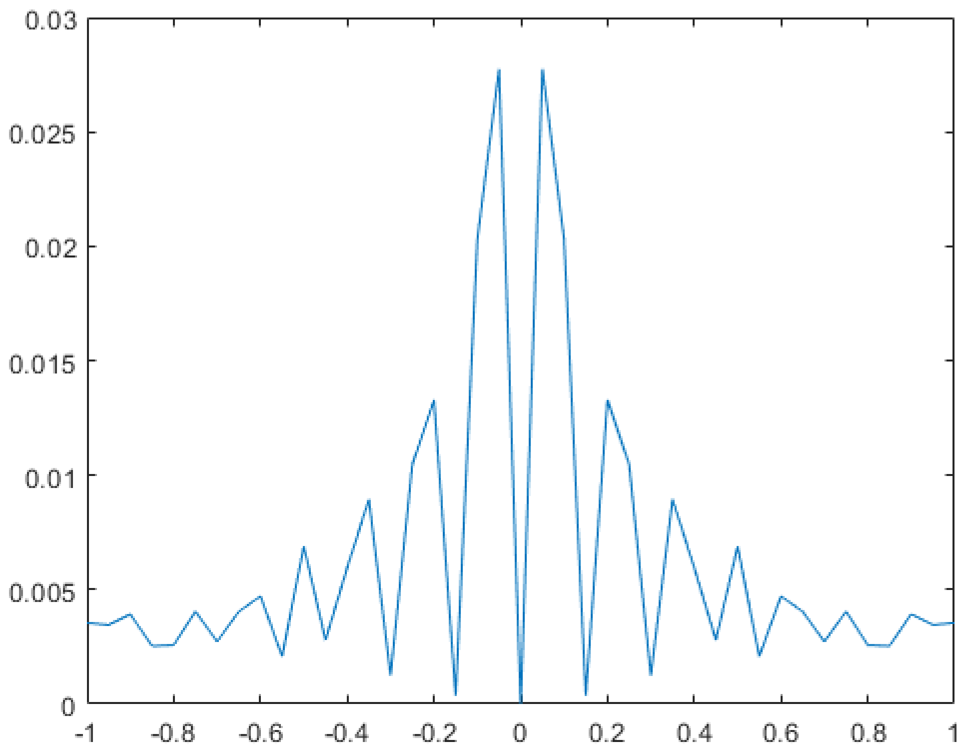
p(x)
f(x)
70
60
50
40
30
20
10
0
-10
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1



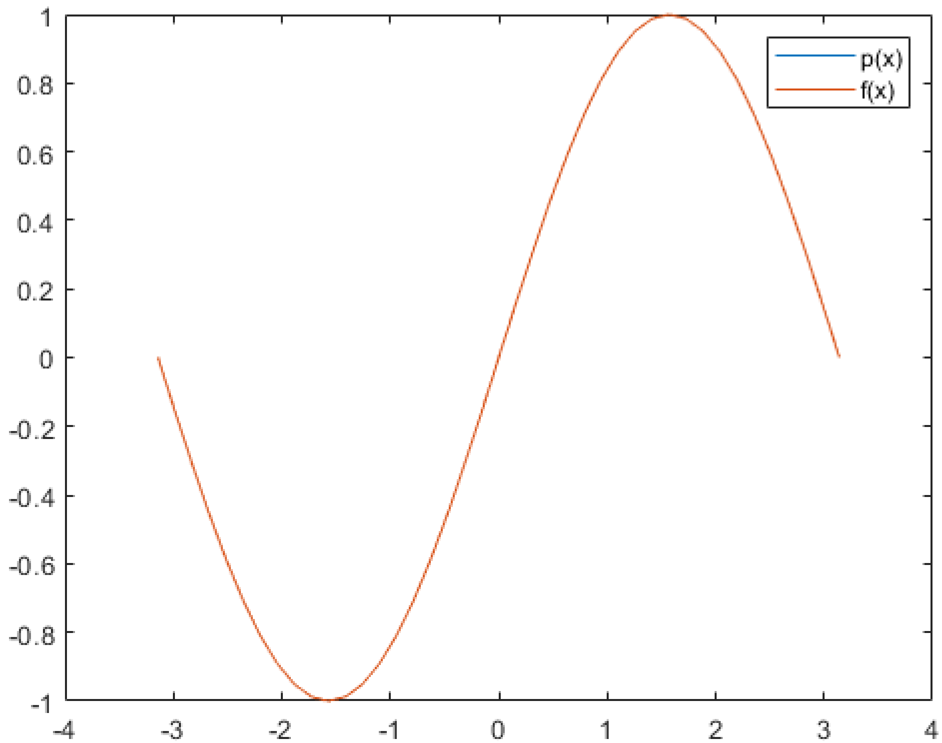
70
60
50
40
30
20
10
0
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

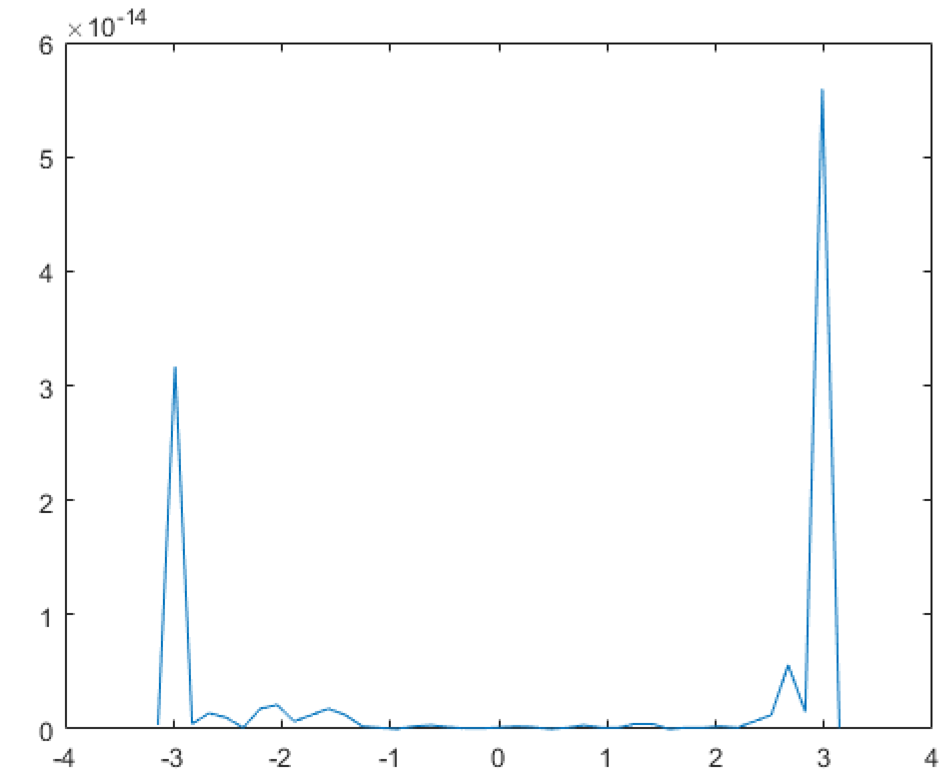


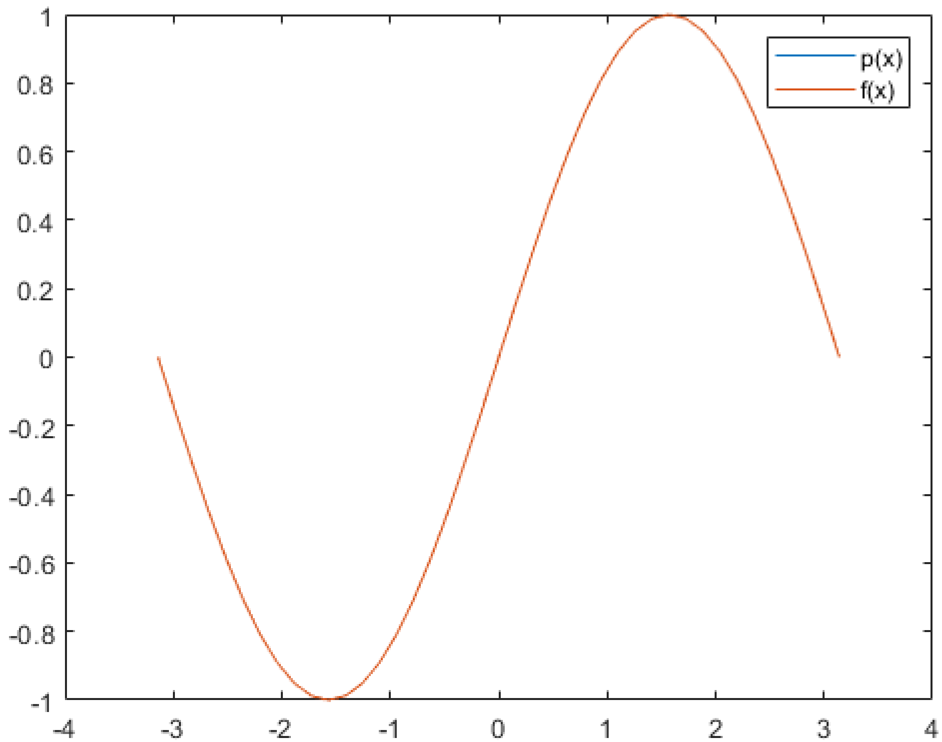
p(x)
f(x)



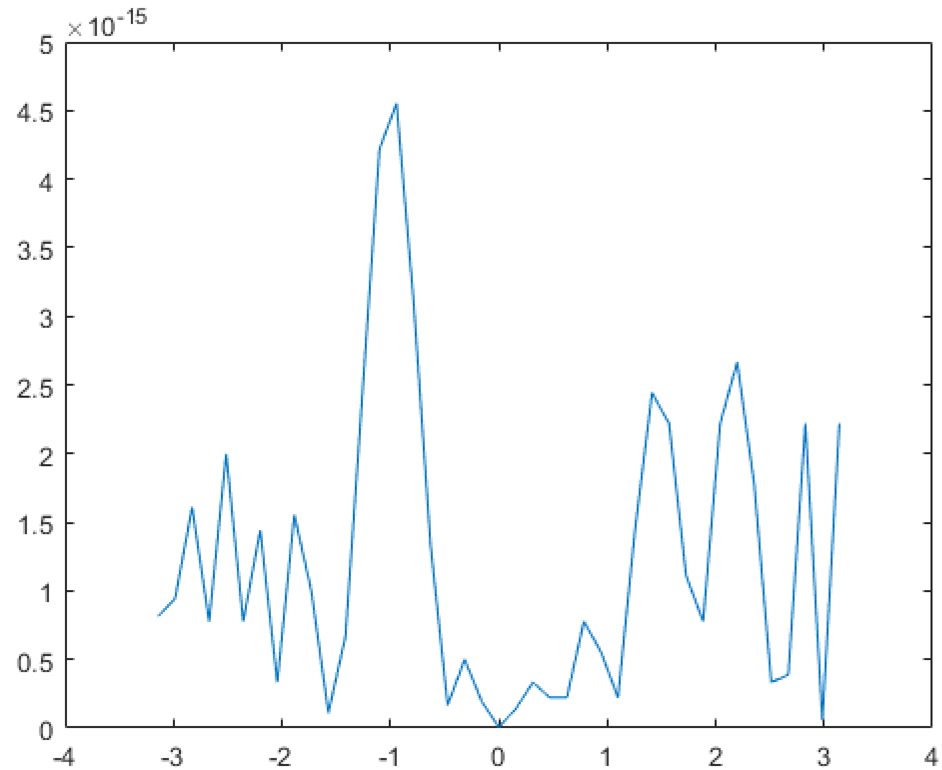
Problem 7:


p(x)
f(x)
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
-4
-3
-2
-1
0
1
2
3
4


$\times 10^{-14}$
6
5
4
3
2
1
0
-4
-3
-2
-1
0
1
2
3
4



p(x)
f(x)



×10^-15

The difference is that interpolated |x|'s errors are larger than that of sin(x). The reason is that |x|'s graph is sharper and straighter, so interpolation is harder. But sin(x) is sleeker and has more arcs so better for interpolation.

Problem 8:

newton.m | hornerN.m | main_interp.m | Week05.mlx | a2_7_a.m

```matlab
function p = newton(x, y, n)
    p = zeros(n+1);
    f = zeros(n+1);
    for i=1:n+1
        f(i,1) = y(i);
    end
    for i=2:n+1
        for j=i:n+1
            f(j,i) = (f(j,i-1)-f(j-1,i-1))/(x(j)-x(j+1-i));
        end
    end
    for i=1:n+1
        p(i) = f(i,i);
    end
end
```

Editor - C:\Users\lizhi\OneDrive\Documents\MATLAB\hornerN.m

a2_7_a.m | a2_7_b.m | a2_7_c.m | hornerN.m | newton.m

```matlab
function eval = hornerN(x, a, n, t)
    temp = a(n+1);
    for i=n:-1:1
        temp = temp*(t-x(i))+a(i);
    end
    eval = temp;
end
```