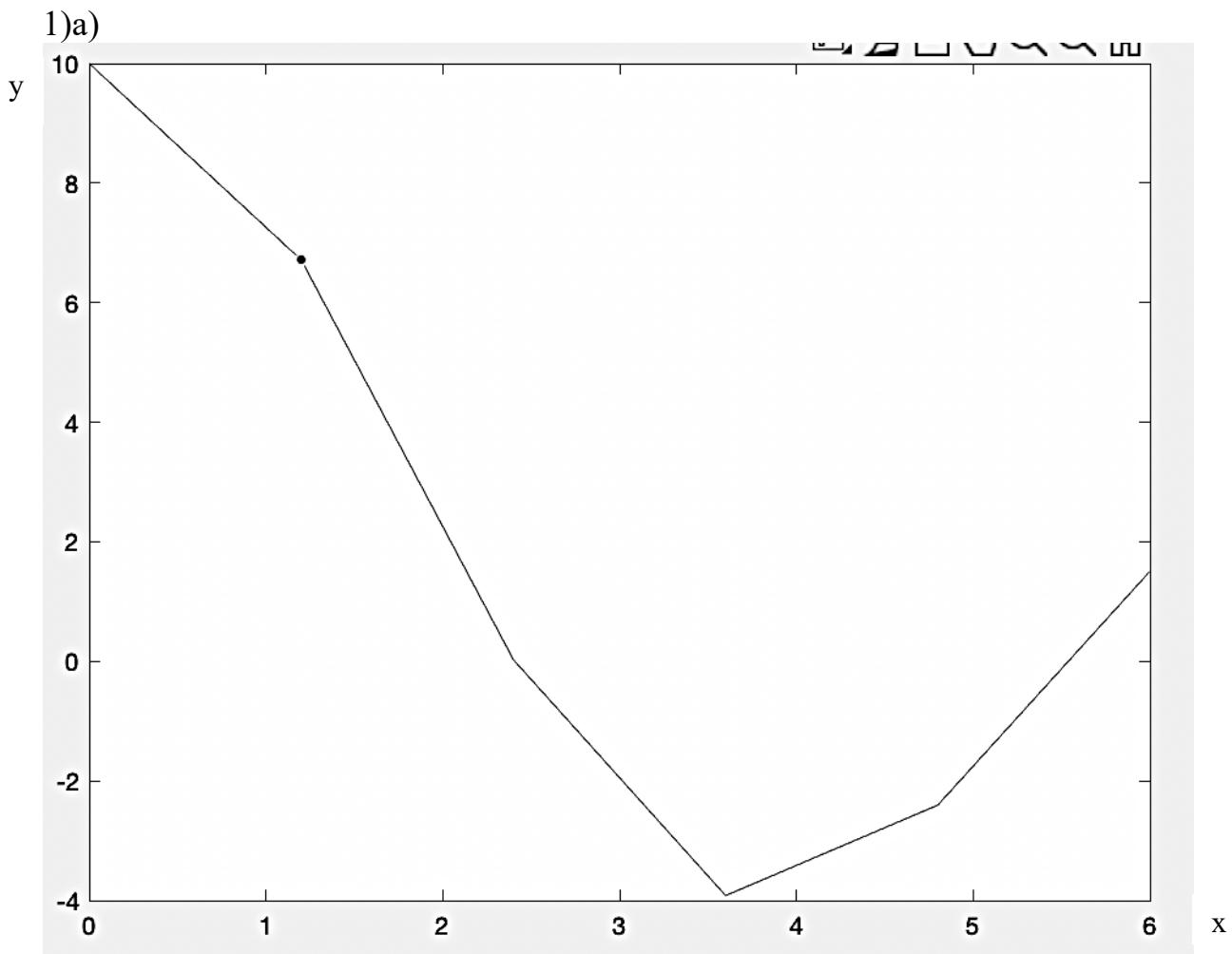


MAT 214

NÜMERİK YÖNTEMLER

ÖDEV 1



Kod:

```
Editor - /Users/fuad/Documents/MATLAB/proje1/a.m
+4 secant_method.m x calculate_lagrange_error.m x calculate_newton_raphson_error.m x minus.m x a.m x +
1 xvalues = [0 1.2 2.4 3.6 4.8 6];
2 yvalues=[10.0000 6.7113 0.0251 -3.9177 -2.4043 1.5065];
3 plot(xvalues, yvalues );
```

b) Lagrange interpolasyon polinomu şu şekilde bulunur:

$$L_{n,k} = \prod_{\substack{i=0 \\ i \neq k}}^n \frac{(x-x_i)}{(x_k-x_i)}$$
 olmak üzere

$$\text{Lagrange interpolasyon polinomu} = P_n(x) = \sum_{k=0}^n L_{n,k}(x) * f(x_k)$$

Soruda 5.dereceden Langrange interpolasyon polinomu istenilmektedir

O zaman Langrange İnterpolasyon polinomu aşağıdaki gibi olur:

$$\begin{aligned}P_5(x) = & L_{5,0}(x) * f(x_0) + L_{5,1}(x) * f(x_1) + L_{5,2}(x) * f(x_2) \\& + L_{5,3}(x) * f(x_3) + L_{5,4}(x) * f(x_4) \\& + L_{5,5}(x) * f(x_5)\end{aligned}$$

Matlaba implemente ederken $L_{n,k}$ fonksiyonunu x değerelerini alarak hesaplayan bir fonksiyon eklendi ve x ‘i symbol olarak aldım böylece x değişken gibi gözükmeyecektir. Böylece P interpolasyon polinomu hesaplanırken her toplamda L fonksiyonu n ve k ya göre hesaplanıp kullanılmaktadır.
Matlab symbol değerlerini çarpıp toplayıp denkelemi sadeleştirmesi için expand() fonksiyonunu kullandı (lagrangeip.m dosyasında)

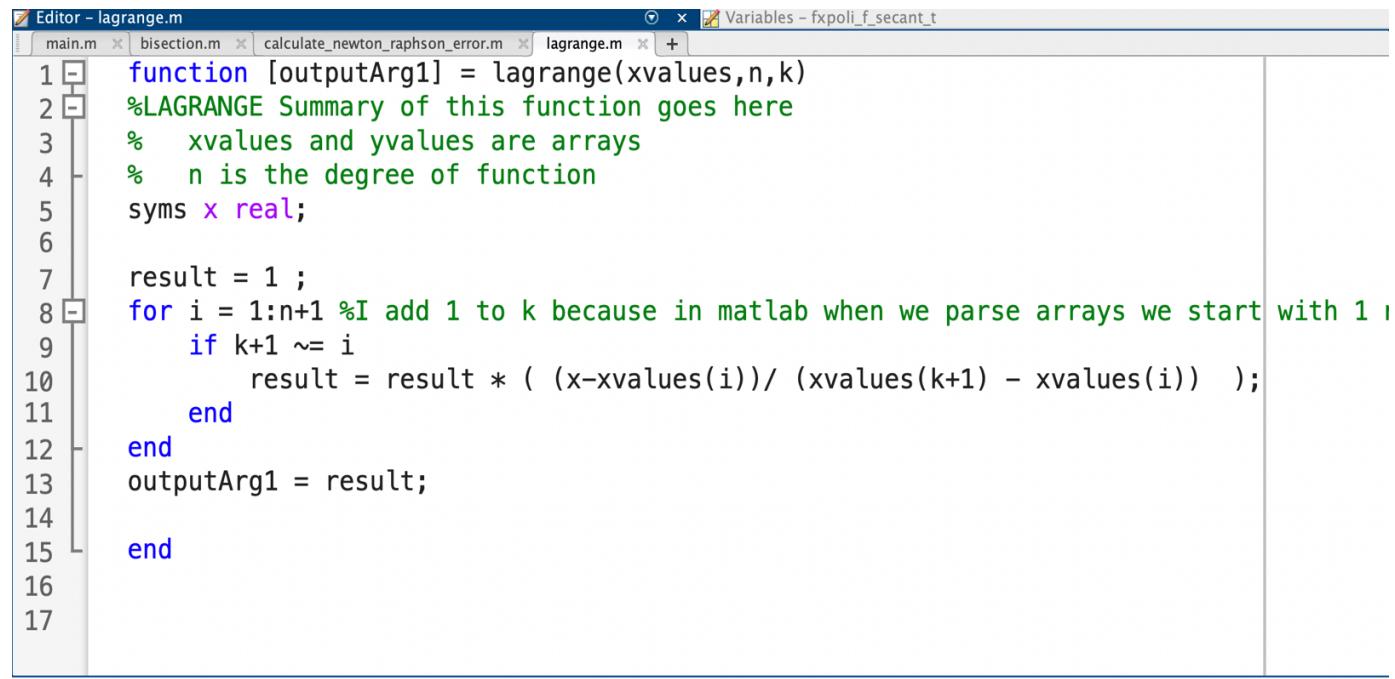
Sonra main.m içinde x ve y vektörlerini tanımladım ve
`lpoli = lagrangeip(xvalues, yvalues , 5);`
çağırdım böylece fonksiyon hesaplanıp lpoli değişkenine atanmaktadır.
lpoli yaani lagrange interpolasyon polinomu sonucu bu şekilde bulundu:

$$P(x) = - (23435*x^5)/2985984 + (12589*x^4)/497664 + (95759*x^3)/138240 - (1287349*x^2)/345600 + (101441*x)/144000 + 10$$

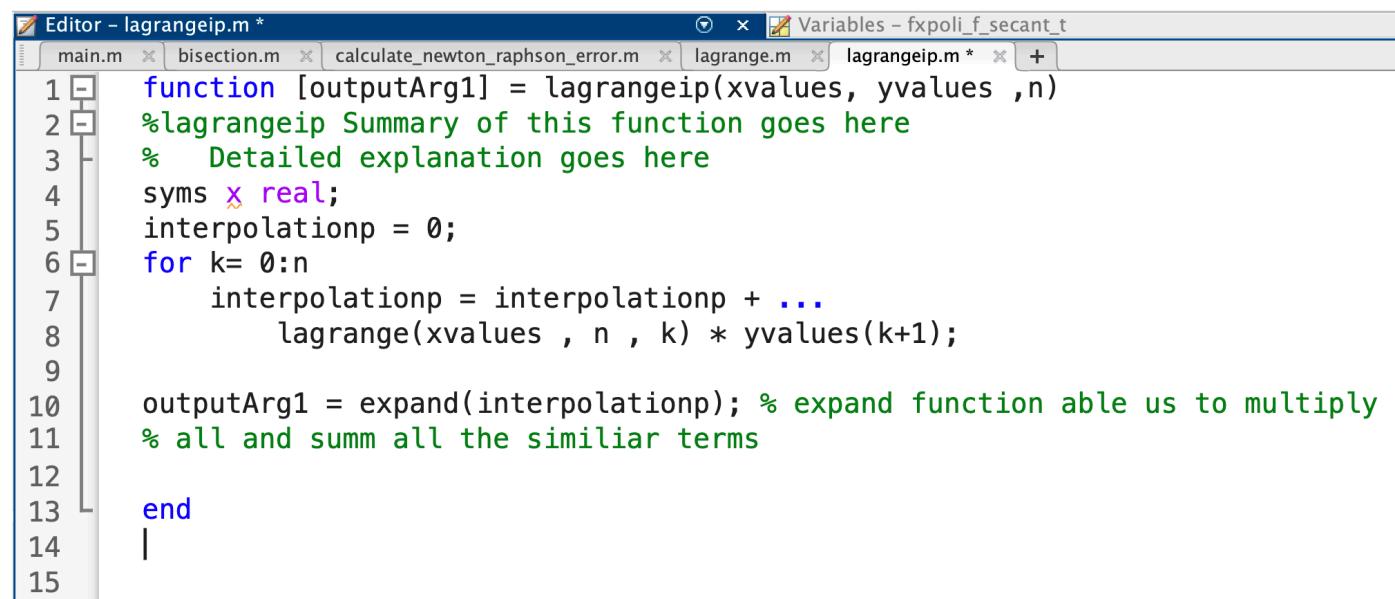
Sayıların bölümmediğini farkettim ve bunun için ne yapabileceğimi araştırdı
Sayıların kesirli değil de bölümnesi için vpa() fonksiyonu kullanılabilceği
bulundu. İlk parametre fonksiyon ve ikinci parametre virgülden sonra basamak
sayısı olmak üzere bu fonksiyon kullanabilir ama çözünürlük düşüşü olmaması
adına fonksiyon içinde kullanmamayı tercih edildi ve sadece main.m de
denemeler yapmak amaçlı bırakıldı.

Virgülden sonra 10 basamak çözünürlükle bölmeler yapılırsa:
 $P(x) = - 0.007848334084*x^5 + 0.02529618377*x^4 + 0.6927010995*x^3 - 3.724968171*x^2 + 0.7044513889*x + 10.0$
şeklinde bulunur.

Kodları :



```
Editor - lagrange.m
main.m bisection.m calculate_newton_raphson_error.m lagrange.m + Variables - fxpoli_f_secant_t
1 function [outputArg1] = lagrange(xvalues,n,k)
2 %LAGRANGE Summary of this function goes here
3 % xvalues and yvalues are arrays
4 % n is the degree of function
5 syms x real;
6
7 result = 1 ;
8 for i = 1:n+1 %I add 1 to k because in matlab when we parse arrays we start with 1 !
9     if k+1 ~= i
10         result = result * ( (x-xvalues(i))/ (xvalues(k+1) - xvalues(i)) );
11     end
12 end
13 outputArg1 = result;
14
15 end
16
17
```



```
Editor - lagrangeip.m *
main.m bisection.m calculate_newton_raphson_error.m lagrange.m lagrangeip.m * + Variables - fxpoli_f_secant_t
1 function [outputArg1] = lagrangeip(xvalues, yvalues ,n)
2 %lagrangeip Summary of this function goes here
3 % Detailed explanation goes here
4 syms x real;
5 interpolationp = 0;
6 for k= 0:n
7     interpolationp = interpolationp + ...
8         lagrange(xvalues , n , k) * yvalues(k+1);
9
10 outputArg1 = expand(interpolationp); % expand function able us to multiply
11 % all and summ all the similiar terms
12
13 end
14
15
```

Aşağıdaki kod main.m içindedir

```
main.m x] bisection.m x] minus.m x] a.m x] b.m x] c.m x] +]
1 xvalues = [0 1.2 2.4 3.6 4.8 6];
2 yvalues=[10.0000 6.7113 0.0251 -3.9177 -2.4043 1.5065];
3 lpoli = lagrangeip(xvalues, yvalues , 5);
4 disp(lpoli);|
5 lpoli = vpa(lpoli , 10);
6 disp(lpoli);
```

c)

$$n_k = \prod_{i=0}^{k-1} (x - x_i)$$

= newton çarpanı

Ve

$$a_n = f[x_0, x_1, \dots, x_n]$$

= bölünmüş farklar olmak üzere

(Köşeli parantezler bölünmüş farklara ait bir tanımdır)

Ve $f[x_i, x_{i+1}, \dots, x_{i+k}]$ bu şekilde tanımlanırsa

$$f[x_i] = f(x_i) \text{ ve}$$

$$f[x_i, x_{i+1}, \dots, x_{i+k}] = \frac{f[x_{i+1}, \dots, x_{i+k}] - f[x_{i+1}, \dots, x_{i+k-1}]}{x_{i+k} - x_i}$$

Newton interpolasyon polinomu şu şekilde bulunur:

$$N(x) = \sum_{i=0}^k a_k n_k(x)$$

a bölünmüş farklar katsayısını bulmak için dividedDifferences() fonksiyonunu yazıldı bu fonksiyon recursive bir şekilde a sayısını hesaplamaktadır.

Ve n çarpanını bulmak için newton adlı fonksiyonu yukarıdaki tanıma uyacak şekilde yazdım ve x'i symbol olarak tanımlaandı .

Sonra interpolasyon polinomu N(x) i bulmak için newtonip fonksiyonu yazıldı Bu fonksiyonun içinde Newton ve dirivedDifferences fonksiyonlarına çağrıarak genel fonksiyon hesaplandı ve exapand() fonksiyonuyla denklem sadeleştirildi. main.m de Newton interpolasyon polinomu bu şekilde hesaplatırıldı

`npoli =newtonip(xvalues,yvalues,5);`

ve sorunun cevabı ise şu şekilde bulundu:

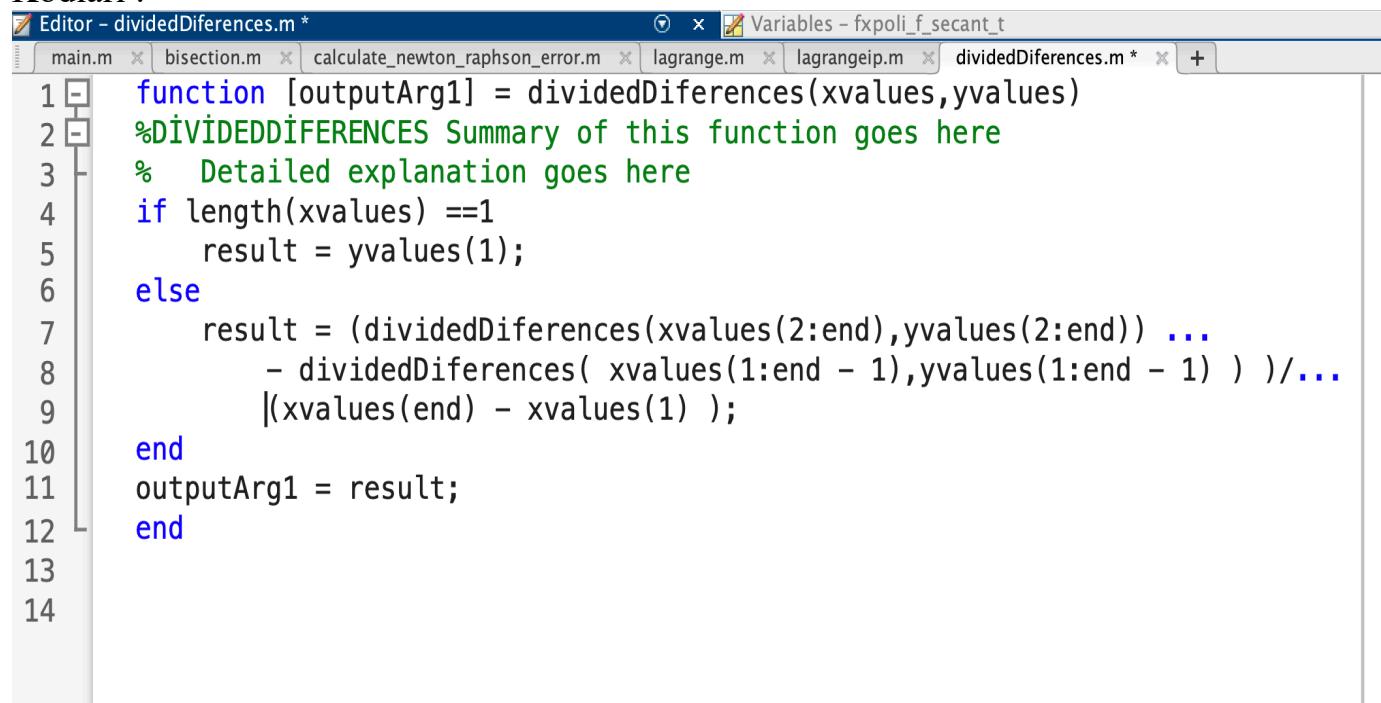
$$N(x) = - (1131064142526445*x^5)/144115188075855872 +$$

$$(24607558901461501*x^4)/972777519512027136 +$$

$$(374357809650495791*x^3)/540431955284459520 - \\(2516364790232543299*x^2)/675539944105574400 + \\(297428157421933753*x)/422212465065984000 + 10$$

Virgülden sonra 10 basamak çözünürlüğüyle bölmeler yapılrsa
 $N(x) = - 0.007848334084*x^5 + 0.02529618377*x^4 + 0.6927010995*x^3 - 3.724968171*x^2 + 0.7044513889*x + 10.0$
sonucu elde edilir.

Kodları :



The screenshot shows the MATLAB Editor window with the file 'dividedDifferences.m' open. The code implements the divided difference formula for polynomial interpolation. It starts with a function definition, includes a detailed explanation comment, and uses a recursive approach to calculate the divided differences for all pairs of points.

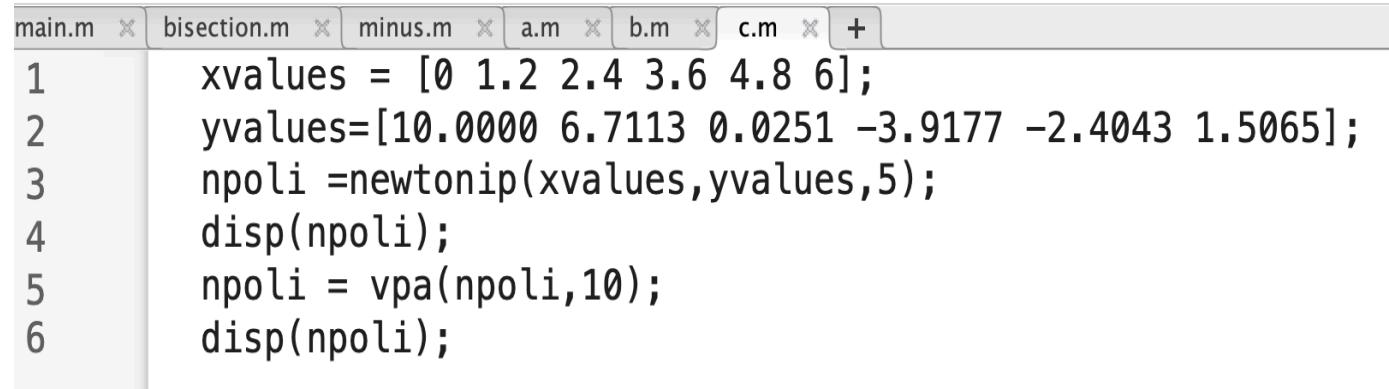
```
function [outputArg1] = dividedDifferences(xvalues,yvalues)
%DIVIDEDDIFFERENCES Summary of this function goes here
% Detailed explanation goes here
if length(xvalues) ==1
    result = yvalues(1);
else
    result = (dividedDifferences(xvalues(2:end),yvalues(2:end)) ...
        - dividedDifferences( xvalues(1:end - 1),yvalues(1:end - 1) ) )/...
        (xvalues(end) - xvalues(1));
end
outputArg1 = result;
end
```

Editor - newton.m

```
1 function [outputArg1] = newton(xvalues,k)
2 %newton Summary of this function goes here
3 % Detailed explanation goes here
4 syms x real;
5 result = 1;
6
7 for i=1:k-1
8     result = result * (x - xvalues(i));
9 end
10 outputArg1 = result;
11 end
12
13
```

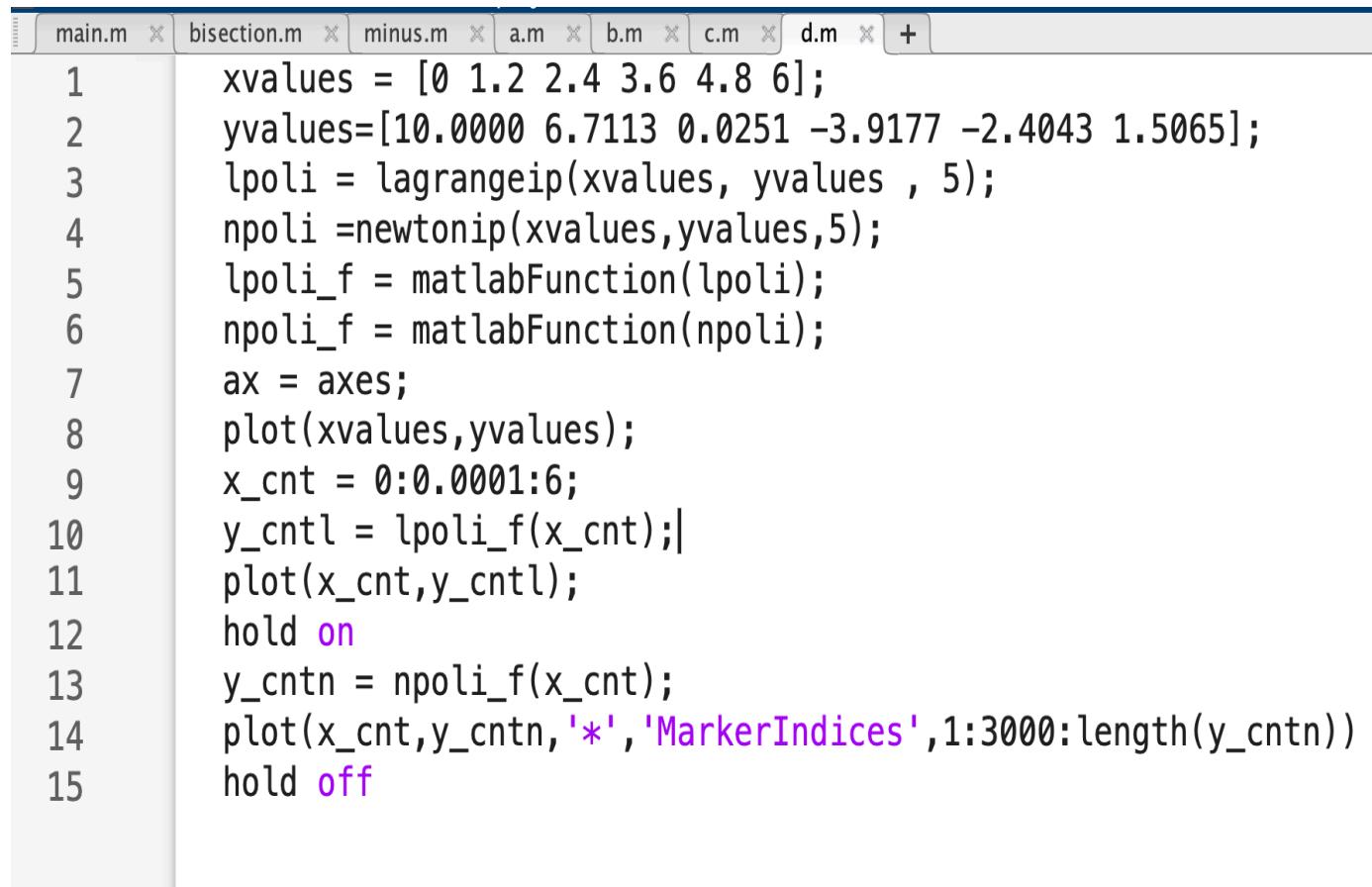
Editor - newtonip.m

```
1 function [outputArg1] = newtonip(xvalues,yvalues, n)
2 %NEWTONIP Summary of this function goes here
3 % Detailed explanation goes here
4 func = 0;
5 for i = 1:n+1
6     func = func + (dividedDifferences(xvalues(1:i) ,yvalues(1:i)) ...
7         * newton(xvalues,i));
8 end
9
10 outputArg1 = expand(func);
11
12 end
13
```



```
main.m x bisection.m x minus.m x a.m x b.m x c.m x +  
1 xvalues = [0 1.2 2.4 3.6 4.8 6];  
2 yvalues=[10.0000 6.7113 0.0251 -3.9177 -2.4043 1.5065];  
3 npoli =newtonip(xvalues,yvalues,5);  
4 disp(npoli);  
5 npoli = vpa(npoli,10);  
6 disp(npoli);
```

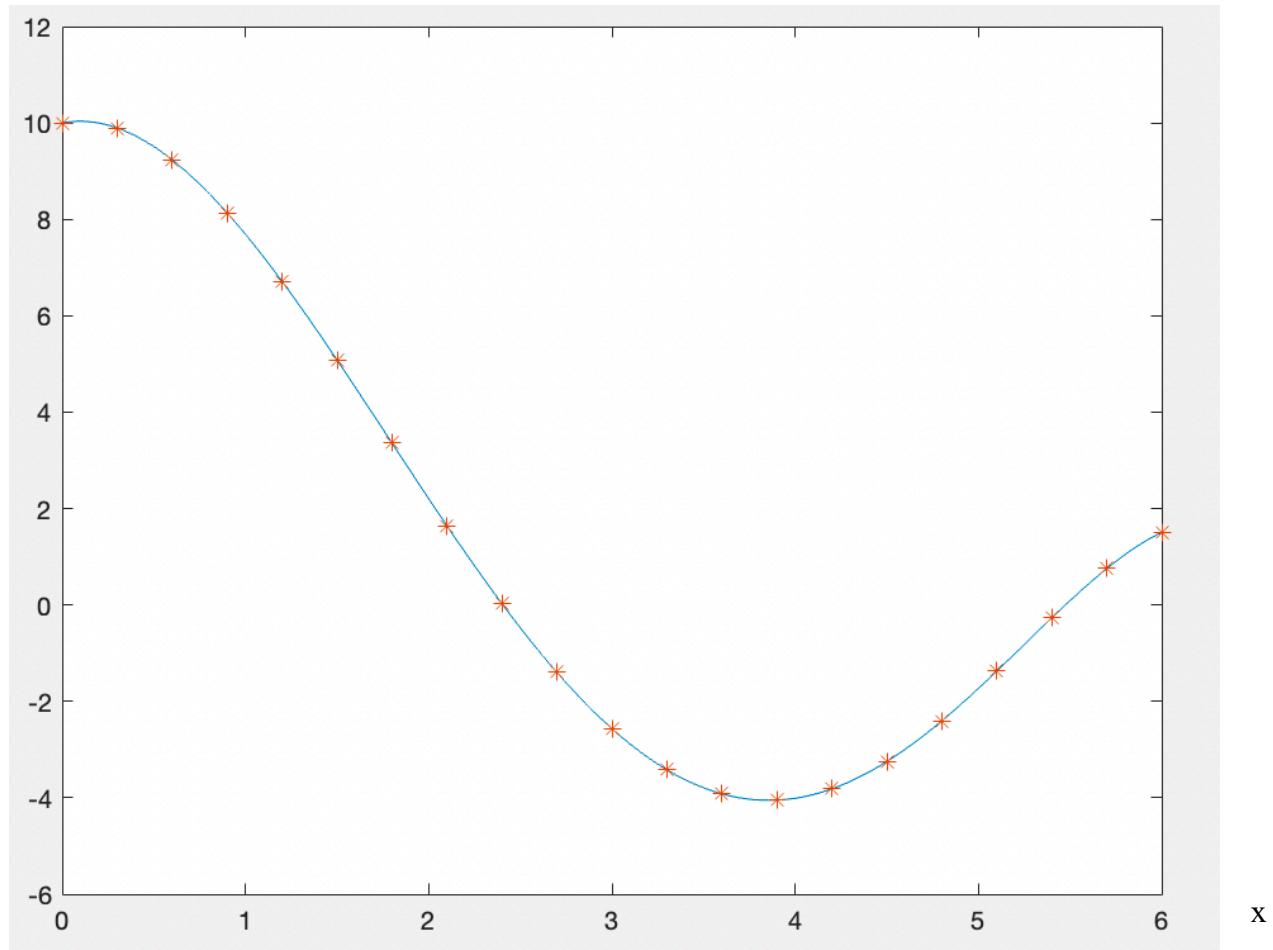
d)
kodu :



The screenshot shows a MATLAB editor window with several tabs at the top: main.m, bisection.m, minus.m, a.m, b.m, c.m, d.m, and a '+' tab. The 'main.m' tab is active. The code in the editor is as follows:

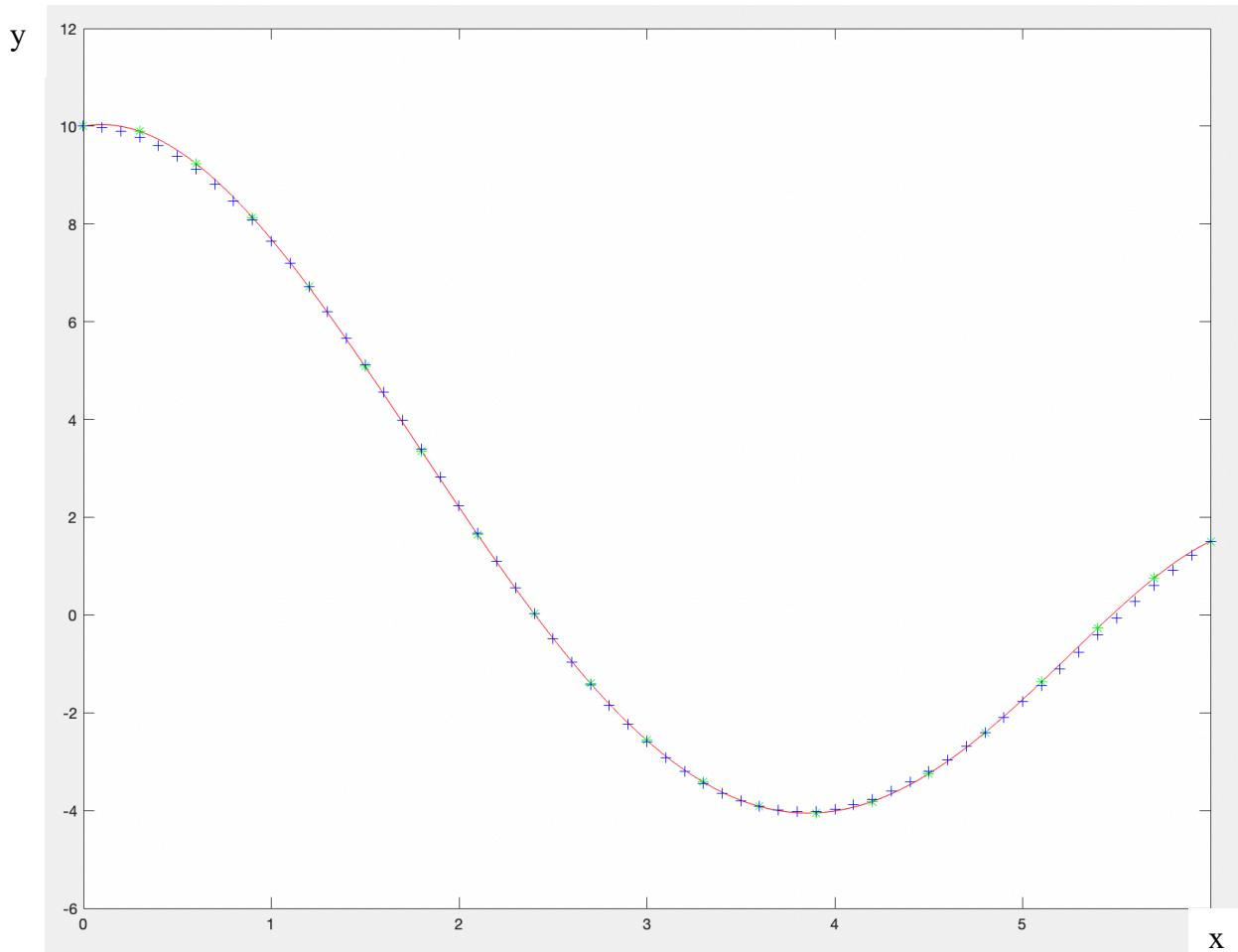
```
1 xvalues = [0 1.2 2.4 3.6 4.8 6];
2 yvalues=[10.0000 6.7113 0.0251 -3.9177 -2.4043 1.5065];
3 lpoli = lagrangeip(xvalues, yvalues , 5);
4 npoli =newtonip(xvalues,yvalues,5);
5 lpoli_f = matlabFunction(lpoli);
6 npoli_f = matlabFunction(npoli);
7 ax = axes;
8 plot(xvalues,yvalues);
9 x_cnt = 0:0.0001:6;
10 y_cntl = lpoli_f(x_cnt);
11 plot(x_cnt,y_cntl);
12 hold on
13 y_cntn = npoli_f(x_cnt);
14 plot(x_cnt,y_cntn,'*', 'MarkerIndices',1:3000:length(y_cntn))
15 hold off
```

Not : çizgi olan fonksiyon langrange interpolasyon fonksiyonu ve yıldızlı olan ise Newton interpolasyon fonksiyonu , ayırt edilebilmesi için bu şekilde gösterilmiştir.



e) Kod:

```
Editor - /Users/fuad/Documents/MATLAB/proje1/e.m
main.m bisection.m minus.m a.m b.m c.m d.m e.m +
1 xvalues = [0 1.2 2.4 3.6 4.8 6];
2 yvalues=[10.0000 6.7113 0.0251 -3.9177 -2.4043 1.5065];
3 lpoli = lagrangeip(xvalues, yvalues , 5);
4 npoli =newtonip(xvalues,yvalues,5);
5 lpoli_f = matlabFunction(lpoli);
6 npoli_f = matlabFunction(npoly);
7 ax = axes;
8 plot(xvalues,yvalues);
9 x_cnt = 0:0.0001:6;
10 y_CNTL = lpoli_f(x_CNT);
11 plot(x_CNT,y_CNTL);
12 hold on
13 y_CNTN = npoli_f(x_CNT);
14 plot(x_CNT,y_CNTN,'*', 'MarkerIndices',1:3000:length(y_CNTN));
15 y_exact = 10 * besselj(0, x_CNT);
16 plot(x_CNT,y_exact,'+', 'MarkerIndices',1:1000:length(y_CNTN));
17 ax.ColorOrder = [1 0 0; 0 1 0; 0 0 1]; %the first plot will be red , the
18 hold off
```



Not : burda çizgi olan fonksiyon langrange interpolasyon fonksiyonu, yıldızlı olan ise Newton interpolasyon fonksiyonu ve artı ile çizilen fonksiyon $10 * \text{besselj}$ fonksiyonudur, ayırt edilebilmesi için bu şekilde gösterilmiştir.

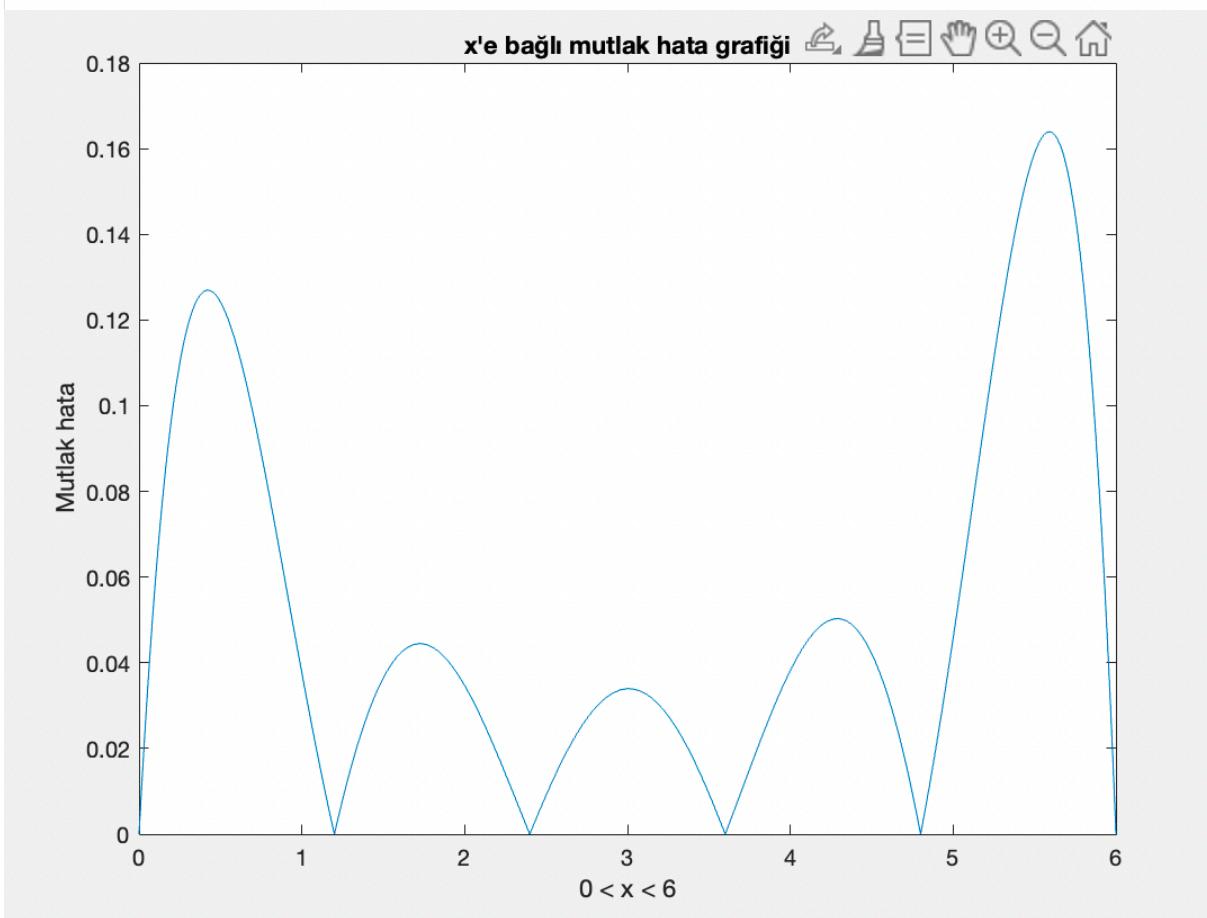
f) Eğer değerler $f(x) = 10 * J_0(x)$ uyması gerekiyorsa ve interpolasyon polinomumuza $P(x)$ denilirse o zaman mutlak hata

$$R(x) = |(P(x) - f(x))|$$

Kod:

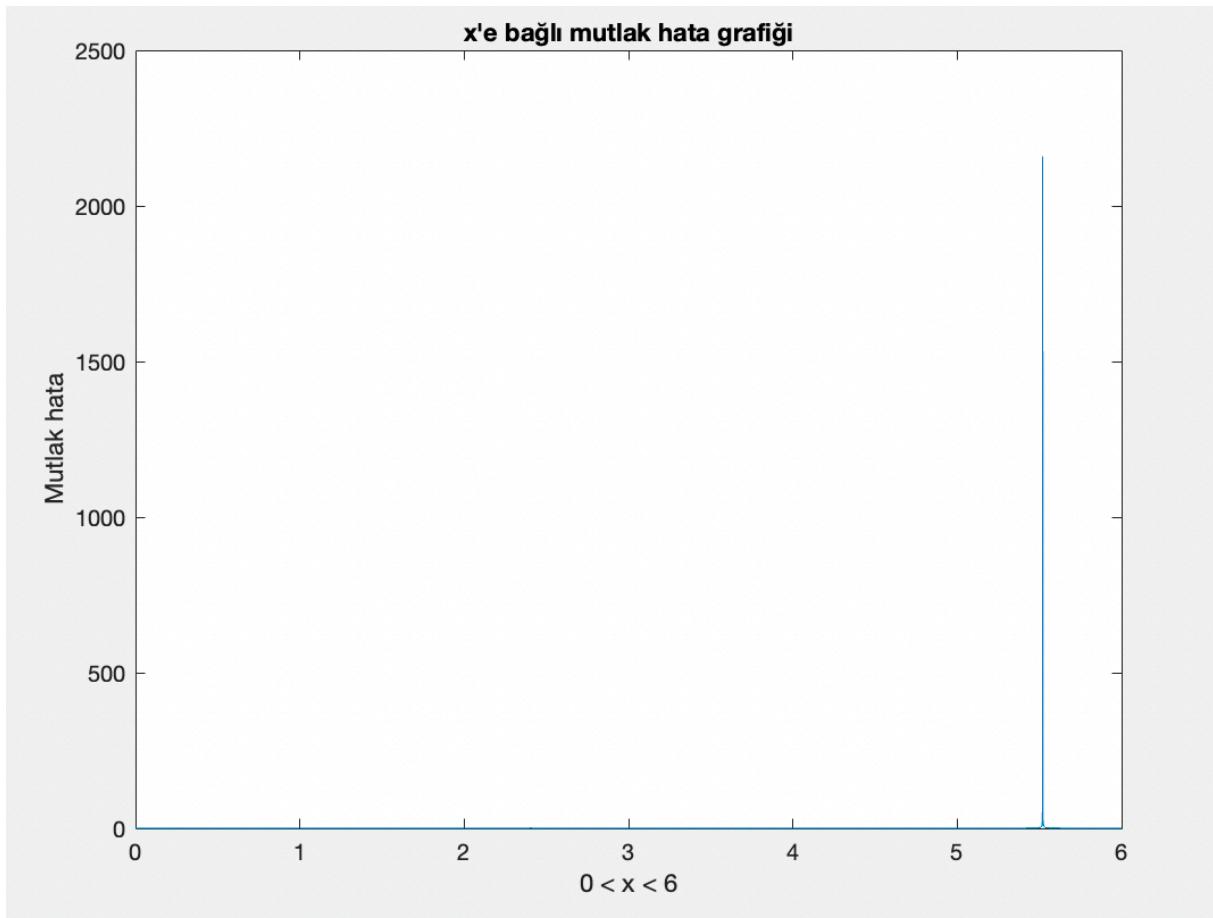
```
xvalues = [0 1.2 2.4 3.6 4.8 6];
yvalues=[10.0000 6.7113 0.0251 -3.9177 -2.4043 1.5065];
lpoly = lagrangeip(xvalues, yvalues , 5);
lpoly_f = matlabFunction(lpoly);
x_cnt = 0:0.0001:6;
y_CNTL = lpoly_f(x_CNT);
y_exact = 10 * besselj(0, x_CNT);
%hold on
%plot(x_CNT ,y_CNTL );
%plot(x_CNT,y_exact);
error = abs((y_CNTL - y_exact) ) ;
abs_error = [];
for i=1:1:length(error)
    abs_error(end+1) = abs(error(i)/y_exact(i));
end

plot(x_CNT , error )
title("x'e bağlı mutlak hata grafiği")
xlabel('0 < x < 6')
ylabel('Mutlak hata')
```



Veya

$$R(x) = \left| \frac{(P(x)-f(x))}{f(x)} \right| \text{ şeklinde tanımlarsak}$$



Ve kodu şu şekildedir:

```
main.m x bisection.m x minus.m x a.m x b.m x c.m x d.m x e.m x f.m x +  
1 xvalues = [0 1.2 2.4 3.6 4.8 6];  
2 yvalues=[10.0000 6.7113 0.0251 -3.9177 -2.4043 1.5065];  
3 lpoli = lagrangeip(xvalues, yvalues , 5);  
4 lpoli_f = matlabFunction(lpoli);  
5 x_cnt = 0:0.0001:6;  
6 y_ctrl = lpoli_f(x_cnt);  
7 y_exact = 10 * besselj(0, x_cnt);  
8 %hold on  
9 %plot(x_cnt ,y_ctrl );  
10 %plot(x_cnt,y_exact);  
11 error = abs((y_ctrl - y_exact) ) ;  
12 abs_error = [];  
13 for i=1:1:length(error)  
14 abs_error(end+1) = abs(error(i))/y_exact(i);  
15 end  
16 plot(x_cnt,abs_error);  
17 title("x'e bağlı mutlak hata grafiği")  
18 xlabel('0 < x < 6')  
19 ylabel('Mutlak hata')  
20 %plot(x_cnt , error )  
21 |  
22 %hold off
```

g) d şıkkında intepolasyon polinomunda ve gerçek fonksiyona bakılırsa
 $x = [2, 3]$ aralığında ve $x = [5,6]$ aralığında bir kök bulunduğu gözükmüyor .

h) Genel kök bulma kodları:

```
1 function [outputArg1,outputArg2,outputArg3,outputArg4] = bisection(f,interval , iteration_number)
2 %UNTITLED Summary of this function goes here
3 % Detailed explanation goes here
4 a_all = [];
5 b_all = [];
6 p_all = [];
7 fp_all = [];
8 a = interval(1);
9 %a_all(end + 1) = a;
10 b = interval(2);
11 %b_all(end + 1) = b;
12 for i = 1:iteration_number
13     p = (a + b)/2;
14     a_all(end + 1) = a;
15     b_all(end + 1) = b;
16     p_all(end + 1) = p;
17     fp_all(end + 1) = f(p);
18     %xline(p);
19
20     if f(a) * f(p) < 0
21         b = p ;
22     elseif f(b) * f(p) < 0
23         a = p ;
24     end
25
26 end
27 a_all(end + 1) = a;
28 b_all(end + 1) = b;
29 p_all(end + 1) = p;
30 fp_all(end + 1) = f(p);
31
32 outputArg1 = a_all;
33 outputArg2 = b_all;
34 outputArg3 = p_all ;
35 outputArg4 = fp_all;
36 end
```

The screenshot shows a MATLAB code editor window with the following details:

- Tab bar: main.m, bisection.m, newton_raphson.m
- Code area:

```
1 function [outputArg1,outputArg2] = newton_raphson(f,p0,iteration_num)
2 %NEWTON_RAPHSON Summary of this function goes here
3 %   f is symbolic function
4 f_n = matlabFunction(f);%fn is numeric function
5 df = diff(f);
6 df_n = matlabFunction(df);

7
8 p = p0;
9 p_all = [p0];
10 fp_all = [f_n(p0)];
11 for i= 1:iteration_num
12     p = p - (f_n(p) / df_n(p));
13     p_all(end + 1) = p;
14     fp_all(end+1) = [f_n(p)];
15 end
16 outputArg1 = p_all;
17 outputArg2 = fp_all;
18 end
19
20 |
```

The screenshot shows a MATLAB code editor window with the following details:

- Tab bar: main.m, bisection.m, newton_raphson.m, fixed_point.m
- Code area:

```
1 function [outputArg1,outputArg2] = fixed_point(fx,p0,iteration_number)
2 %FIXED_POINT Summary of this function goes here
3 %   Detailed explanation goes here
4 p_all = [];
5 fp_all = [];
6 p = p0;
7 for i = 1:iteration_number
8     p_all(end+1) = p;
9     fp_all(end + 1) = fx(p);
10    p = fx(p);
11    %xline(p);
12 end
13 outputArg1 = p_all;
14 outputArg2 = fp_all;
15 end
16
17 |
```

The screenshot shows the MATLAB Editor window with the file `secant_method.m` open. The code implements the Secant Method for finding roots of a function. It starts by defining input arguments and a summary comment. It then initializes arrays for points and function values, and enters a loop to calculate successive approximations until the iteration number is reached. Finally, it returns the last two approximations.

```
Editor - secant_method.m
Variables - fxpoli_f_secant_t
main.m × bisection.m × newton_raphson.m × fixed_point.m × secant_method.m × + [x]
1 function [outputArg1,outputArg2] = secant_method(f, p0 ,p1 ,iteration_num)
2 %SECANT_METHOD Summary of this function goes here
3 %    f is matlab function (numeric)
4
5 p_all =[p0 p1];
6 fp_all = [f(p0) f(p1)];
7 for i=3:iteration_num
8     p = p_all(i-1) - ( ( f(p_all(i-1)) * ( p_all(i-1) - p_all(i-2) ) ) /...
9             (f(p_all(i-1)) - f(p_all(i-2)) ) );
10    p_all(end + 1) = p;
11    fp_all(end + 1) = f(p);
12 end
13 outputArg1 = p_all;
14 outputArg2 = fp_all;
15 end
16
17
```

The screenshot shows the MATLAB Editor window with the file `calculate_newton_raphson_error.m` open. This script calculates the error for the Newton-Raphson method. It first defines the function and its derivative, then initializes variables for the error calculation. A loop iterates over a range of values, calculating the function result at each point and updating the error value. Finally, it returns the maximum error found.

```
+2 Editor - calculate_newton_raphson_error.m
Variables - fxpoli_f_secant_t
newton_raphson.m × fixed_point.m × secant_method.m × calculate_lagrange_error.m × calculate_newton_raphson_error.m × + [x]
1 function [outputArg1,outputArg2] = calculate_newton_raphson_error(f,p0,p,inteval)
2 %CALCULATE_NEWTON_RAPHSON_ERROR Summary of this function goes here
3 %    f is symbolic function
4 result = (diff(diff(f))/2) * ((p - p0)^2);
5 delta = 0.0001;
6 func_result = matlabFunction(result);
7 value_of_result = [];
8 for i=inteval(1):delta:inteval(2)
9     value_of_result(end + 1) = func_result(i);
10 end
11 numeric_result = max(value_of_result);
12
13 outputArg1 = numeric_result ;
14 outputArg2 = result;
15
16 end
17
18 |
```

1.kök için kodlar:

```
xvalues = [0 1.2 2.4 3.6 4.8 6];
yvalues=[10.0000 6.7113 0.0251 -3.9177 -2.4043 1.5065];
lpoly = lagrangeip(xvalues, yvalues , 5);
npoly =newtonip(xvalues,yvalues,5);
%%%%%%%%%%%%%
lpoly_f = matlabFunction(lpoly);
[l_a_it_b,l_b_it_b,l_p_it_b,l_fp_it_b] = bisection(lpoly_f,[2,3],20);
lpoly_f_bis_t = table(l_a_it_b.',l_b_it_b.',l_p_it_b.',l_fp_it_b.');
l_poly_bis_err =( l_b_it_b(end) - l_a_it_b(end)) / 2;

[l_p_it_f , l_fp_it_f] = fixed_point(lpoly_f,2.5,20);
lpoly_f_fixed_t = table(l_p_it_f.',l_fp_it_f.');

[l_p_it_n , l_fp_it_n] = newton_raphson(lpoly,2.5,6);
lpoly_f_newtonr_t = table(l_p_it_n.',l_fp_it_n.');
[l_raph_error ,l_raph_error_func] = calculate_newton_raphson_error...
(lpoly , l_p_it_n(end-1) , l_p_it_n(end), [2.4 , 2.5]);

[l_p_it_s , l_fp_it_s] = secant_method(lpoly_f,2,3,20);
lpoly_f_secant_t = table(l_p_it_s.',l_fp_it_s.');
%%%%%%%%%%%%%
npoly_f = matlabFunction(npoly);
[n_a_it_b,n_b_it_b,n_p_it_b,n_fp_it_b] = bisection(npoly_f,[2,3],20);
npoly_f_bis_t = table(n_a_it_b.',n_b_it_b.',n_p_it_b.',n_fp_it_b.');
npoly_bis_err =( n_b_it_b(end) - n_a_it_b(end)) / 2;

[n_p_it_b , n_fp_it_b] = fixed_point(npoly_f,2.5,20);
npoly_f_fixed_t = table(n_p_it_b.',n_fp_it_b.');

[n_p_it_n , n_fp_it_n] = newton_raphson(npoly,2.5,4);
npoly_f_newtonr_t = table(n_p_it_n.',n_fp_it_n.');
[n_raph_error ,n_raph_error_func] = calculate_newton_raphson_error(lpoly ,
n_p_it_n(end-1) , n_p_it_n(end), [2.4 , 2.5]);

[n_p_it_s , n_fp_it_s] = secant_method(npoly_f,2,3,20);
npoly_f_secant_t = table(n_p_it_s.',n_fp_it_s.');
%%%%%%%%%%%%%
syms x real;
fx_poly = 10 * besselj(0,x);
fx_poly_f = matlabFunction(fx_poly);

[f_a_it_b,f_b_it_b,f_p_it_b,f_fp_it_b] = bisection(fx_poly_f,[2,3],20);
fxpoly_f_bis_t = table(f_a_it_b.',f_b_it_b.',f_p_it_b.',f_fp_it_b.');
fx_poly_bis_err =( f_b_it_b(end) - f_a_it_b(end)) / 2;

[p_it , fp_it] = fixed_point(fx_poly_f,2.5,20);
fxpoly_f_fixed_t = table(p_it.',fp_it.');

[p_it , fp_it] = newton_raphson(fx_poly,2.5,4);
fxpoly_f_newtonr_t = table(p_it.',fp_it.');
[fx_raph_error ,fx_raph_error_func] = calculate_newton_raphson_error(... 
lpoly , p_it(end-1) , p_it(end), [2.4 , 2.5]);

[p_it , fp_it] = secant_method(fx_poly_f,2,3,9);
fxpoly_f_secant_t = table(p_it.',fp_it.');
```

1.kök için

1)Lagrange interpolasyon polinomu için

1.1)Bisection yöntemi ile kök bulma :

n	a _n	b _n	p _n	f(p) _n
0	2	3	2.500000000000000	-0.474774614917559
1	2	2.500000000000000	2.250000000000000	0.813401750564577
2	2.250000000000000	2.500000000000000	2.375000000000000	0.153466957632814
3	2.375000000000000	2.500000000000000	2.437500000000000	-0.164971381487323
4	2.375000000000000	2.437500000000000	2.406250000000000	-0.00678809448387696
5	2.375000000000000	2.406250000000000	2.390625000000000	0.0730859864383273
6	2.390625000000000	2.406250000000000	2.398437500000000	0.0330848901527041
7	2.398437500000000	2.406250000000000	2.402343750000000	0.0131322973964743
8	2.402343750000000	2.406250000000000	2.404296875000000	0.00316806555764337
9	2.404296875000000	2.406250000000000	2.405273437500000	-0.00181102478510908
10	2.404296875000000	2.405273437500000	2.404785156250000	0.00067826797412843
11	2.404785156250000	2.405273437500000	2.405029296875000	- 0.000566441529572970
12	2.404785156250000	2.405029296875000	2.40490722656250	5.58974438877868e-05
13	2.40490722656250	2.40502929687500	2.40496826171875	- 0.000255275987768755
14	2.40490722656250	2.40496826171875	2.40493774414063	-9.96902581302805e-05
15	2.40490722656250	2.40493774414063	2.40492248535156	-2.18966536653653e-05
16	2.40490722656250	2.40492248535156	2.40491485595703	1.70003334751812e-05
17	2.40491485595703	2.40492248535156	2.40491867065430	-2.44817550409948e-06
18	2.40491485595703	2.40491867065430	2.40491676330566	7.27607513439921e-06
19	2.40491676330566	2.40491867065430	2.40491771697998	2.41394885058810e-06
20	2.40491771697998	2.40491867065430	2.40491771697998	2.41394885058810e-06

Bisection yönteminin mutlak hatası bu şekilde hesaplanır

$$|p_n - p| \leq \frac{b_n - a_n}{2}$$

O zaman

$$|p_n - p| \leq 4.768371582031250e - 07$$

1.2)Fixed point yöntemi ile ($p_0 = 2.5$ alındı tahmini aralığın yarısı(g şıklına bakınız))

n	p_n	$f(p)_n$
0	2.500000000000000	-0.474774614917559
1	-0.474774614917559	8.75323798526763
2	8.75323798526763	-59.4590178638101
3	-59.4590178638101	5990013.13280748
4	5990013.13280748	-6.05223970948188e+31
5	-6.05223970948188e+31	6.37320788780027e+156
6	6.37320788780027e+156	NaN

Bir noktaya yaklaşmak söz konusu olmadığından kök bulunamadı.

1.3)Newton raphson metodu ile ($p_0 = 2.5$ alındı tahmini aralığın yarısı)

n	p_n	$f(p)_n$
0	2.500000000000000	-0.474774614917559
1	2.40279199206967	0.0108447942490102
2	2.40491725418038	4.77344394766988e-06
3	2.40491819046024	9.30810983845731e-13
4	2.404918190460423	-3.55271367880050e-15
5	2.404918190460422	1.77635683940025e-15
6	2.404918190460423	0

Newton metodunun hata payı ,
 epsilon tahmini kök aralığında([2.4,2.5])(fonksiyon aralığı almamıza gerek yoktur çünkü $p_0 = 2.5$ verdik ve p_n 2.4 ve 2.5 aralığında olduğunu görüyoruz) bir sayı olmak üzere şu şekilde bulunur

$$R(p_n) = \frac{f''(\varepsilon)}{2} * (p_n - p_{n-1})^2$$

Bu hatayı bulmak için calculate_newton_raphson_error adında bir fonksiyon yazıldı içinde $R(p)$ hata polinomunu bulundu sonra aralık sınırları içinde $\Delta = 0.0001$ olmak üzere epsilon örneklenip $R(x)$ 'in maximum olduğu nokta bulundu.

$$\begin{aligned} R(p) = & -\frac{117175 * x^3}{7570368819743778682397463648002899968} \\ & + \frac{12589 * x^2}{420576045541321037910970202666827776} \\ & + \frac{95759 * x}{233653358634067243283872334814904320} \\ & - \frac{1287349}{175240018975550432462904251111782400} \end{aligned}$$

$$\max(R(p)) = 2.3520e - 31$$

1.4) Sekant metodu ile ($p_0 = 2$, $p_1 = 3$ alındı, tahmini aralığın sınırları)

n	p_n	$f(p)_n$
0	2	2.20423113854595
1	3	-2.56658398437500
2	2.46202401094017	-0.287600849383132
3	2.39413305157952	0.0551085111331737
4	2.40505008543040	-0.000672422615746626
5	2.40491848382453	-1.49566014684410e-06
6	2.40491819045239	4.09716705007668e-11
7	2.404918190460422	8.88178419700125e-15
8	2.404918190460424	-7.10542735760100e-15
9	2.404918190460423	0

Secant yönteminin hatası ε hata olmak üzeri bu şekilde hesaplanabilir

$$|r_{n+1} - r_n| \leq \varepsilon$$

O zaman $\varepsilon = \pm 7.1e - 15$

Newton interpolasyon polinomu için kod :

bu kodları sırasıyla çalıştırarak Newton interpolasyon polinomunun değerlerini hesapladım ve tablolara aldım (a_it, b_it, p_it, fp_it aynı olduğundan kullanırken diğer satırları yorumla aldım)

2)Newton interpolasyon polinomu için

2.1) Bisection yöntemi ile kök bulma :

n	a _n	b _n	p _n	f(p) _n
0	2	3	2.500000000000000	-0.474774614917560
1	2	2.500000000000000	2.250000000000000	0.813401750564578
2	2.250000000000000	2.500000000000000	2.375000000000000	0.153466957632816
3	2.375000000000000	2.500000000000000	2.437500000000000	-0.164971381487327
4	2.375000000000000	2.437500000000000	2.406250000000000	-0.00678809448387696
5	2.375000000000000	2.406250000000000	2.390625000000000	0.0730859864383273
6	2.390625000000000	2.406250000000000	2.398437500000000	0.0330848901527023
7	2.398437500000000	2.406250000000000	2.402343750000000	0.0131322973964760
8	2.402343750000000	2.406250000000000	2.404296875000000	0.00316806555764337
9	2.404296875000000	2.406250000000000	2.405273437500000	-0.00181102478511264
10	2.404296875000000	2.405273437500000	2.404785156250000	0.000678267974128843
11	2.404785156250000	2.405273437500000	2.405029296875000	-0.000566441529572970
12	2.404785156250000	2.405029296875000	2.40490722656250	5.58974438877868e-05
13	2.40490722656250	2.405029296875000	2.40496826171875	-0.000255275987768755
14	2.40490722656250	2.40496826171875	2.40493774414063	-9.96902581320569e-05
15	2.40490722656250	2.40493774414063	2.40492248535156	-2.18966536653653e-05
16	2.40490722656250	2.40492248535156	2.40491485595703	1.70003334716284e-05
17	2.40491485595703	2.40492248535156	2.40491867065430	-2.44817550232312e-06
18	2.40491485595703	2.40491867065430	2.40491676330566	7.27607513439921e-06
19	2.40491676330566	2.40491867065430	2.40491771697998	2.41394885058810e-06
20	2.40491771697998	2.40491867065430	2.40491771697998	2.41394885058810e-06

Bisection yönteminin mutlak hatası bu şekilde hesaplanır

$$|p_n - p| \leq \frac{b_n - a_n}{2}$$

O zaman

$$|p_n - p| \leq 4.768371582031250e - 07$$

2.2)Fixed point yöntemi ile (p0 = 2.5 alındı tahmini aralığın yarısı(g şıklına bakınız))

n	p _n	f(p) _n
0	2.500000000000000	-0.474774614917560
1	-0.474774614917560	8.75323798526762
2	8.75323798526762	-59.4590178638096
3	-59.4590178638096	5990013.13280723
4	5990013.13280723	-6.05223970948061e+31
5	-6.05223970948061e+31	6.37320788779356e+156
6	6.37320788779356e+156	NaN

Bir noktaya yaklaşmak söz konusu olmadığından kök bulunamadı.

2.3) Newton raphson metodu ile ($p_0 = 2.5$ alındı tahmini aralığın yarısı)

n	p_n	$f(p)_n$
0	2.500000000000000	-0.474774614917560
1	2.40279199206967	0.0108447942490084
2	2.40491725418038	4.77344395122259e-06
3	2.40491819046024	9.25481913327531e-13
4	2.40491819046042	0

$$R(p_n) = \frac{f''(\varepsilon)}{2} * (p_n - p_{n-1})^2$$

$$R(p) = -\frac{19601151175 * x^3}{7570368819743778682397463648002899968} \\ + \frac{2105900509 * x^2}{420576045541321037910970202666827776} \\ + \frac{16018661279 * x}{233653358634067243283872334814904320} \\ - \frac{215349028069}{175240018975550432462904251111782400}$$

Ve

$$\max(R(p)) = 3.934413099098205e - 26$$

2.4) Sekant metodu ile ($p_0 = 2$, $p_1 = 3$ alındı, tahmini aralığın sınırları)

n	p_n	$f(p)_n$
0	2	2.20423113854595
1	3	-2.56658398437500
2	2.46202401094017	-0.287600849383130
3	2.39413305157952	0.0551085111331684
4	2.40505008543039	-0.000672422615739521
5	2.40491848382453	-1.49566015217317e-06
6	2.40491819045239	4.09734468576062e-11
7	2.40491819046042243357	3.55271367880050e-15
8	2.40491819046042332175	-5.32907051820075e-15
9	2.40491819046042287766	0

Secant yönteminin hatası ε hata olmak üzeri bu şekilde hesaplanabilir

$$|r_{n+1} - r_n| \leq \varepsilon$$

O zaman $\varepsilon = \pm 5.32907051820075e - 15$

fx interpolasyon polinomu için kod :

bu kodları sırasıyla çalıştırarak fx interpolasyon polinomunun değerlerini hesapladım ve tablolara aldım ($a_{it}, b_{it}, p_{it}, fp_{it}$ aynı olduğundan kullanırken diğer satırları yorumla aldım)

3) $f(x)$ polinomu için

3.1) Bisection yöntemi ile kök bulma :

n	a_n	b_n	p_n	$f(p)_n$
0	2	3	2.500000000000000	-0.483837764681979
1	2	2.500000000000000	2.250000000000000	0.827498512887340
2	2.250000000000000	2.500000000000000	2.375000000000000	0.155783734620732
3	2.375000000000000	2.500000000000000	2.437500000000000	-0.168456529803729
4	2.375000000000000	2.437500000000000	2.406250000000000	-0.00739276482216946
5	2.375000000000000	2.406250000000000	2.390625000000000	0.0739378802025111
6	2.390625000000000	2.406250000000000	2.398437500000000	0.0332073408094519
7	2.398437500000000	2.406250000000000	2.402343750000000	0.0128908822279104
8	2.402343750000000	2.406250000000000	2.404296875000000	0.00274494459455180
9	2.404296875000000	2.406250000000000	2.405273437500000	-0.00232494022259895
10	2.404296875000000	2.405273437500000	2.40478515625000	0.000209744856419509
11	2.40478515625000	2.40527343750000	2.40502929687500	-0.00105766204018874
12	2.40478515625000	2.40502929687500	2.40490722656250	- 0.000423974678071470
13	2.40478515625000	2.40490722656250	2.40484619140625	- 0.000107118931985526
14	2.40478515625000	2.40484619140625	2.40481567382813	5.13119569749008e-05
15	2.40481567382813	2.40484619140625	2.40483093261719	-2.79037388226405e-05
16	2.40481567382813	2.40483093261719	2.40482330322266	1.17040462478698e-05
17	2.40482330322266	2.40483093261719	2.40482711791992	-8.09986199381796e-06
18	2.40482330322266	2.40482711791992	2.40482521057129	1.80208819959307e-06
19	2.40482521057129	2.40482711791992	2.40482616424561	-3.14888787994886e-06
20	2.40482521057129	2.40482616424561	2.40482616424561	-3.14888787994886e-06

Bisection yönteminin mutlak hatası bu şekilde hesaplanır

$$|p_n - p| \leq \frac{b_n - a_n}{2}$$

O zaman

$$|p_n - p| \leq 4.768371582031250e - 07$$

3.2) Fixed point yöntemi ile ($p_0 = 2.5$ alındı tahmini aralığın yarısı(g şıklına bakınız))

n	p_n	$f(p)_n$
0	2.500000000000000	-0.483837764681979
1	- 0.483837764681979	9.42325992933161
2	9.42325992933161	-1.80942948791555
3	-1.80942948791555	3.34502309172189
4	3.34502309172189	-3.53813386670414
5	-3.53813386670414	-3.85061519836204
6	-3.85061519836204	-4.02687510748892
7	-4.02687510748892	-3.95237940579397
8	-3.95237940579397	-3.99860477518174
9	-3.99860477518174	-3.97241584597260
10	-3.97241584597260	-3.98826205546148
11	-3.98826205546148	-3.97898758587437
12	-3.97898758587437	-3.98453268999997
13	-3.98453268999997	-3.98125694733214
14	-3.98125694733214	-3.98320632965977
15	-3.98320632965977	-3.98205121429512
16	-3.98205121429512	-3.98273744191241
17	-3.98273744191241	-3.98233038646896
18	-3.98233038646896	-3.98257206081466
19	-3.98257206081466	-3.98242865207945
20	2.500000000000000	-0.483837764681979

İstediğimiz aralığın dışına değerler çıktı ve kök bulunamadı

3.3) Newton raphson metodu ile ($p_0 = 2.5$ alındı tahmini aralığın yarısı)

n	p_n	$f(p)_n$
0	2.500000000000000	-0.483837764681979
1	2.40266676625544	0.0112123363919162
2	2.40482459179194	5.01446659545133e-06
3	2.40482555769558	1.00708169328851e-12
4	2.40482555769577	0

$$R(p_n) = \frac{f''(\varepsilon)}{2} * (p_n - p_{n-1})^2$$

$$R(p_n) = -\frac{22376792575 * x^3}{7570368819743778682397463648002899968} \\ + \frac{2404108741 * x^2}{420576045541321037910970202666827776} \\ + \frac{18287000471 * x}{233653358634067243283872334814904320} \\ - \frac{245843751181}{175240018975550432462904251111782400}$$

Ve

$$\max(R(p)) = 4.491549758320938e - 26$$

3.4) Sekant metodu ile ($p_0 = 2$, $p_1 = 3$ alındı, tahmini aralığın sınırları)

n	p_n	$f(p)_n$
0	2	2.23890779141236
1	3	-2.60051954901934
2	2.46263899298727	-0.296420873174667
3	2.39350783095040	0.0588931328735816
4	2.40496628922179	-0.000730582815975592
5	2.40482588615207	-1.70517254195778e-06
6	2.40482555768616	4.98985125661385e-11
7	2.404825557695772	1.57274600722102e-15
8	2.404825557695773	0

Secant yönteminin hatası ε hata olmak üzeri bu şekilde hesaplanabilir

$$|r_{n+1} - r_n| \leq \varepsilon$$

O zaman $\varepsilon = \pm 1.57274600722102e - 15$

2.kök için kod:

```
xvalues = [0 1.2 2.4 3.6 4.8 6];
yvalues=[10.0000 6.7113 0.0251 -3.9177 -2.4043 1.5065];
lpoly = lagrangeip(xvalues, yvalues , 5);
npoly =newtonip(xvalues,yvalues,5);
%%%%%%%%%%%%%
lpoly_f = matlabFunction(lpoly);
[l_a_it_b,l_b_it_b,l_p_it_b,l_fp_it_b] = bisection(lpoly_f,[5,6],20);
lpoly_f_bis_t = table(l_a_it_b.',l_b_it_b.',l_p_it_b.',l_fp_it_b.');
l_poly_bis_err =( l_b_it_b(end) - l_a_it_b(end)) / 2;

[l_p_it_f , l_fp_it_f] = fixed_point(lpoly_f,5.5,20);
lpoly_f_fixed_t = table(l_p_it_f.',l_fp_it_f.');

[l_p_it_n , l_fp_it_n] = newton_raphson(lpoly,5.5,6);
lpoly_f_newtonr_t = table(l_p_it_n.',l_fp_it_n.');
[l_raph_error ,l_raph_error_func] = calculate_newton_raphson_error...
(lpoly , l_p_it_n(end-1) , l_p_it_n(end), [5,6]);

[l_p_it_s , l_fp_it_s] = secant_method(lpoly_f,5,6,20);
lpoly_f_secant_t = table(l_p_it_s.',l_fp_it_s.');
%%%%%%%%%%%%%
npoly_f = matlabFunction(npoly);
[n_a_it_b,n_b_it_b,n_p_it_b,n_fp_it_b] = bisection(npoly_f,[5,6],20);
npoly_f_bis_t = table(n_a_it_b.',n_b_it_b.',n_p_it_b.',n_fp_it_b.');
npoly_bis_err =( n_b_it_b(end) - n_a_it_b(end)) / 2;

[n_p_it_b , n_fp_it_b] = fixed_point(npoly_f,5.5,20);
npoly_f_fixed_t = table(n_p_it_b.',n_fp_it_b.');

[n_p_it_n , n_fp_it_n] = newton_raphson(npoly,5.5,4);
npoly_f_newtonr_t = table(n_p_it_n.',n_fp_it_n.');
[n_raph_error ,n_raph_error_func] = calculate_newton_raphson_error...
(lpoly , n_p_it_n(end-1) , n_p_it_n(end), [5,6]);

[n_p_it_s , n_fp_it_s] = secant_method(npoly_f,5,6,20);
npoly_f_secant_t = table(n_p_it_s.',n_fp_it_s.');
%%%%%%%%%%%%%
syms x real;
fx_poly = 10 * besselj(0,x);
fx_poly_f = matlabFunction(fx_poly);

[f_a_it_b,f_b_it_b,f_p_it_b,f_fp_it_b] = bisection(fx_poly_f,[5,6],20);
fxpoly_f_bis_t = table(f_a_it_b.',f_b_it_b.',f_p_it_b.',f_fp_it_b.');
fx_poly_bis_err =( f_b_it_b(end) - f_a_it_b(end)) / 2;

[p_it , fp_it] = fixed_point(fx_poly_f,5.5,20);
fxpoly_f_fixed_t = table(p_it.',fp_it.');

[p_it , fp_it] = newton_raphson(fx_poly,5.5,3);
fxpoly_f_newtonr_t = table(p_it.',fp_it.');
[fx_raph_error ,fx_raph_error_func] = calculate_newton_raphson_error...
(lpoly , p_it(end-1) , p_it(end), [5,6]);

[p_it , fp_it] = secant_method(fx_poly_f,5,6,9);
fxpoly_f_secant_t = table(p_it.',fp_it.');
```

2.kök için

1)Lagrange interpolasyon polinomu için

1.1)Bisection yöntemi ile kök bulma :

n	a _n	b _n	p _n	f(p) _n
0	5	6	5.500000000000000	0.0904909141341719
1	5	5.500000000000000	5.250000000000000	-0.819895986557000
2	5.250000000000000	5.500000000000000	5.375000000000000	-0.359068486954541
3	5.375000000000000	5.500000000000000	5.437500000000000	-0.132116637321182
4	5.437500000000000	5.500000000000000	5.468750000000000	-0.0201706275674027
5	5.468750000000000	5.500000000000000	5.484375000000000	0.0353333375714016
6	5.468750000000000	5.484375000000000	5.476562500000000	0.00762306351271036
7	5.468750000000000	5.476562500000000	5.472656250000000	-0.00626355272949297
8	5.472656250000000	5.476562500000000	5.474609375000000	0.000682337403429045
9	5.472656250000000	5.474609375000000	5.473632812500000	-0.00278996524857789
10	5.473632812500000	5.474609375000000	5.474121093750000	-0.00105365293307358
11	5.474121093750000	5.474609375000000	5.474365234375000	-0.000185617469192323
12	5.474365234375000	5.474609375000000	5.474487304687500	0.000248370047067681
13	5.474365234375000	5.474487304687500	5.47442626953125	3.13788081598432e-05
14	5.474365234375000	5.47442626953125	5.47439575195313	-7.71187008048457e-05
15	5.47439575195313	5.47442626953125	5.47441101074219	-2.28697888573493e-05
16	5.47441101074219	5.47442626953125	5.47441864013672	4.25454900820910e-06
17	5.47441101074219	5.47441864013672	5.47441482543945	-9.30761010309311e-06
18	5.47441482543945	5.47441864013672	5.47441673278809	-2.52652807830600e-06
19	5.47441673278809	5.47441864013672	5.47441768646240	8.64011077794658e-07
20	5.47441673278809	5.47441768646240	5.47441768646240	8.64011077794658e-07

Bisection yönteminin mutlak hatası bu şekilde hesaplanır

$$|p_n - p| \leq \frac{b_n - a_n}{2}$$

O zaman

$$|p_n - p| \leq 4.768371582031250e - 07$$

1.2)Fixed point yöntemi ile ($p_0 = 2.5$ alındı tahmini aralığın yarısı(g şıkkına bakınız))

n	p_n	$f(p)_n$
0	5.500000000000000	0.0904909141341719
1	0.0904909141341719	10.0337590913431
2	10.0337590913431	-199.983208650812
3	-199.983208650812	2545183796.57242
4	2545183796.57242	-8.38249358265348e+44
5	-8.38249358265348e+44	3.24820648941753e+222
6	3.24820648941753e+222	NaN

Bir noktaya yaklaşmak söz konusu olmadığından kök bulunamadı.

1.3)Newton raphson metodu ile ($p_0 = 2.5$ alındı tahmini aralığın yarısı)

n	p_n	$f(p)_n$
0	5.500000000000000	0.0904909141341719
1	5.47428106939667	-0.000484854701408466
2	5.47441743990191	-1.25700125863659e-08
3	5.47441744343755	-2.84217094304040e-14
4	5.47441744343755	2.13162820728030e-14
5	5.47441744343755	-7.10542735760100e-15
6	5.47441744343755	3.55271367880050e-15

Newton metodunun hata payı ,
 epsilon tahmini kök aralığında([2.4,2.5])(fonksiyon aralığı almamıza gerek yoktur çünkü $p_0 = 2.5$ verdik ve p_n 2.4 ve 2.5 aralığında olduğunu görüyoruz) bir sayı olmak üzere şu şekilde bulunur

$$R(p_n) = \frac{f''(\varepsilon)}{2} * (p_n - p_{n-1})^2$$

Bu hatayı bulmak için calculate_newton_raphson_error adında bir fonksiyon yazıldı içinde $R(p)$ hata polinomunu bulundu sonra aralık sınırları içinde $\Delta = 0.0001$ olmak üzere epsilon örneklenip $R(x)$ 'in maximum olduğu nokta bulundu.

$$R(p) = -\frac{117175 * x^3}{473148051233986167649841478000181248} + \frac{12589 * x^2}{26286002846332564869435637666676736} + \frac{95759 * x}{14603334914629202705242020925931520} - \frac{1287349}{109525011859719020289315156944486400}$$

$$\max(R(p)) = 2.049644540373287e - 30$$

1.4) Sekant metodu ile ($p_0 = 2$, $p_1 = 3$ alındı, tahmini aralığın sınırları)

n	p_n	$f(p)_n$
0	5	-1.73023905017574
1	6	1.506500000000001
2	5.53456241709748	0.211140499852434
3	5.45869720447276	-0.0560520146994605
4	5.47461231354814	0.000692783871489411
5	5.47441800973249	2.01331292615237e-06
6	5.47441744341654	-7.46780415283865e-11
7	5.47441744343755	3.55271367880050e-15
8	5.47441744343755	1.06581410364015e-14
9	5.47441744343755	3.55271367880050e-15
10	5.47441744343755	3.55271367880050e-15

Secant yönteminin hatası ε hata olmak üzeri bu şekilde hesaplanabilir

$$|r_{n+1} - r_n| \leq \varepsilon$$

O zaman $\varepsilon = \pm e - 16$

2)Newton interpolasyon polinomu için

2.1) Bisection yöntemi ile kök bulma :

n	a _n	b _n	p _n	f(p) _n
0	5	6	5.500000000000000	0.0904909141341683
1	5	5.500000000000000	5.250000000000000	-0.819895986556990
2	5.250000000000000	5.500000000000000	5.375000000000000	-0.359068486954527
3	5.375000000000000	5.500000000000000	5.437500000000000	-0.132116637321186
4	5.437500000000000	5.500000000000000	5.468750000000000	-0.0201706275673850
5	5.468750000000000	5.500000000000000	5.484375000000000	0.0353333375714051
6	5.468750000000000	5.484375000000000	5.476562500000000	0.00762306351271391
7	5.468750000000000	5.476562500000000	5.472656250000000	-0.00626355272947521
8	5.472656250000000	5.476562500000000	5.474609375000000	0.000682337403432598
9	5.472656250000000	5.474609375000000	5.473632812500000	-0.00278996524856723
10	5.473632812500000	5.474609375000000	5.474121093750000	-0.00105365293305582
11	5.474121093750000	5.474609375000000	5.474365234375000	-0.000185617469174559
12	5.474365234375000	5.474609375000000	5.474487304687500	0.000248370047078339
13	5.474365234375000	5.474487304687500	5.47442626953125	3.13788081705013e-05
14	5.474365234375000	5.47442626953125	5.47439575195313	-7.71187007941876e-05
15	5.47439575195313	5.47442626953125	5.47441101074219	-2.28697888751128e-05
16	5.47441101074219	5.47442626953125	5.47441864013672	4.25454902597267e-06
17	5.47441101074219	5.47441864013672	5.47441482543945	-9.30761009243497e-06
18	5.47441482543945	5.47441864013672	5.47441673278809	-2.52652808185871e-06
19	5.47441673278809	5.47441864013672	5.47441768646240	8.64011095558226e-07
20	5.47441673278809	5.47441768646240	5.47441768646240	8.64011095558226e-07

Bisection yönteminin mutlak hatası bu şekilde hesaplanır

$$|p_n - p| \leq \frac{b_n - a_n}{2}$$

O zaman

$$|p_n - p| \leq 4.768371582031250e - 07$$

2.2)Fixed point yöntemi ile (p0 = 5 .5 alındı tahmini aralığın yarısı(g sıkkına bakınız))

n	p _n	f(p) _n
0	5.500000000000000	0.0904909141341683
1	0.0904909141341683	10.0337590913431
2	10.0337590913431	-199.983208650812
3	-199.983208650812	2545183796.57240
4	2545183796.57240	-8.38249358265327e+44
5	-8.38249358265327e+44	3.24820648941713e+222
6	3.24820648941713e+222	NaN

Bir noktaya yaklaşmak söz konusu olmadığından kök bulunamadı.

2.3) Newton raphson metodu ile ($p_0 = 5.5$ alındı tahmini aralığın yarısı)

n	p_n	$f(p)_n$
0	5.500000000000000	0.0904909141341683
1	5.47428106939667	-0.000484854701383597
2	5.47441743990191	-1.25700267972206e-08
3	5.47441744343754	-1.42108547152020e-14
4	5.47441744343755	1.77635683940025e-14

$$R(p_n) = \frac{f''(\varepsilon)}{2} * (p_n - p_{n-1})^2$$

$$R(p) = -\frac{2929375 * x^3}{1892592204935944670599365912000724992} \\ + \frac{314725 * x^2}{105144011385330259477742550666706944} \\ + \frac{478795 * x}{11682667931703362164193616740745216} \\ - \frac{1287349}{17524001897555043246290425111117824}$$

Ve

$$\max(R(p)) = 1.281027837733303e - 29$$

2.4) Sekant metodu ile ($p_0 = 5$, $p_1 = 6$ alındı, tahmini aralığın sınırları)

n	p_n	$f(p)_n$
0	5	-1.73023905017575
1	6	1.50650000000002
2	5.53456241709748	0.211140499852448
3	5.45869720447275	-0.0560520146994676
4	5.47461231354813	0.000692783871485858
5	5.47441800973249	2.01331289417794e-06
6	5.47441744341655	-7.46638306736713e-11
7	5.47441744343755	0

Secant yönteminin hatası ε hata olmak üzeri bu şekilde hesaplanabilir

$$|r_{n+1} - r_n| \leq \varepsilon$$

O zaman $\varepsilon = 3.54 e - 12$

3) $f(x)$ polinomu için

3.1) Bisection yöntemi ile kök bulma :

n	a_n	b_n	p_n	$f(p)_n$
0	5	6	5.500000000000000	-0.0684386941781923
1	5.500000000000000	6	5.750000000000000	0.759753320169011
2	5.500000000000000	5.750000000000000	5.625000000000000	0.353013198107559
3	5.500000000000000	5.625000000000000	5.562500000000000	0.143751807320986
4	5.500000000000000	5.562500000000000	5.531250000000000	0.0379748031886749
5	5.500000000000000	5.531250000000000	5.515625000000000	-0.0151584320854015
6	5.515625000000000	5.531250000000000	5.523437500000000	0.0114273225631227
7	5.515625000000000	5.523437500000000	5.519531250000000	-0.00186086517975801
8	5.519531250000000	5.523437500000000	5.521484375000000	0.00478441293112698
9	5.519531250000000	5.521484375000000	5.520507812500000	0.00146206845566525
10	5.519531250000000	5.520507812500000	5.520019531250000	- 0.000199324902092810
11	5.520019531250000	5.520507812500000	5.520263671875000	0.000631390164909089
12	5.520019531250000	5.520263671875000	5.52014160156250	0.000216037225546853
13	5.520019531250000	5.52014160156250	5.52008056640625	8.35730990114335e-06
14	5.520019531250000	5.52008056640625	5.52005004882813	-9.54835090959215e-05
15	5.52005004882813	5.52008056640625	5.52006530761719	-4.35630278419299e-05
16	5.52006530761719	5.52008056640625	5.52007293701172	-1.76028410302648e-05
17	5.52007293701172	5.52008056640625	5.52007675170898	-4.62276108188307e-06
18	5.52007675170898	5.52008056640625	5.52007865905762	1.86727553214875e-06
19	5.52007675170898	5.52007865905762	5.5200770538330	-1.37774249599332e-06
20	5.5200770538330	5.52007865905762	5.5200770538330	-1.37774249599332e-06

Bisection yönteminin mutlak hatası bu şekilde hesaplanır

$$|p_n - p| \leq \frac{b_n - a_n}{2}$$

O zaman

$$|p_n - p| \leq 4.768371582031250e - 07$$

3.2) Fixed point yöntemi ile ($p_0 = 2.5$ alındı tahmini aralığın yarısı(g sıkkına bakınız))

n	p_n	$f(p)_n$
0	5.500000000000000	-0.0684386941781923
1	- 0.0684386941781923	9.98829379029206
2	9.98829379029206	-2.45409719565079
3	-2.45409719565079	-0.253104865143048
4	-0.253104865143048	9.84048492020945
5	9.84048492020945	-2.35835818098580
6	-2.35835818098580	0.243507664944676
7	0.243507664944676	9.85230851585885
8	9.85230851585885	-2.36800092697162
9	-2.36800092697162	0.192609432787469
10	0.192609432787469	9.90746883996365
11	9.90746883996365	-2.40844849773075
12	-2.40844849773075	-0.0187942079143780
13	- 0.0187942079143780	9.99911696386667
14	9.99911696386667	-2.45897278868287
15	-2.45897278868287	-0.277850211253402
16	-0.277850211253402	9.80792739850549
17	9.80792739850549	-2.33004406228526
18	-2.33004406228526	0.394023124176986
19	0.394023124176986	9.61561447131022
20	5.500000000000000	-0.0684386941781923

İstediğimiz aralığın dışına değerler çıktı ve kök bulunamadı

3.3) Newton raphson metodu ile ($p_0 = 2.5$ alındı tahmini aralığın yarısı)

n	p_n	$f(p)_n$
0	5.500000000000000	-0.0684386941781923
1	5.52004423965613	-0.000115250187844122
2	5.52007811018241	-3.53534479359273e-10
3	5.52007811028631	7.22802933433496e-16

$$R(p_n) = \frac{f''(\varepsilon)}{2} * (p_n - p_{n-1})^2$$

$$R(p_n) = -\frac{1603487657250175 * x^3}{1892592204935944670599365912000724992} \\ + \frac{172274854850629 * x^2}{105144011385330259477742550666706944} \\ + \frac{1310419241054999 * x}{58413339658516810820968083703726080} \\ - \frac{17616797372078989}{438100047438876081157260627777945600}$$

Ve

$$\max(R(p)) = 7.012118033366270e - 21$$

3.4) Sekant metodu ile ($p_0 = 2$, $p_1 = 3$ alındı, tahmini aralığın sınırları)

n	p_n	$f(p)_n$
0	5	-1.77596771314338
1	6	1.50645257250997
2	5.54105433143516	0.0712342053162384
3	5.51827547719887	-0.00613472439632757
4	5.52008165434677	1.20591865834734e-05
5	5.52007811086310	1.96261177793863e-09
6	5.52007811028631	7.22802933433496e-16
7	5.52007811028631	7.22802933433496e-16

Secant yönteminin hatası ε hata olmak üzeri bu şekilde hesaplanabilir

$$|r_{n+1} - r_n| \leq \varepsilon$$

O zaman $\varepsilon = \pm e - 16$

i) 1.kök için ([2,3] aralığında):

Fonksiyon	Bisection metodu	Fixed point metodu	Newton-raphson metodu	Secant metodu
Lagrange interpolasyon polinomu	p20 = 2.40491771697998 tolerans = 4.76837158203125 0e-07	Doğru kök buluna madı (h' a bakınız)	p6 = 2.404918190460423 tolerans = 2.3520e-31	p8= 2.40483875970032 0 tolerans = ± 7.1e-15
Newton interpolasyon polinomu	p20 =2.40491771697998 tolerans = 4.768371582031250 e-07	Doğru kök buluna madı	p4 = 2.40491819046042 tolerans = 3.934413099098205e-26	p9 =2.4049181904604 2287766 tolerans = ±5.3290705182007 5e-15
f(x) polinomu	p20 = 2.40482521057129 tolerans =4.76837158203125 0e-07	Doğru kök buluna madı	p4=2.40482555769577 tolerans = 4.491549758320938e-26	p8=2.40482555769 5773 tolerans = 1.57274600722102 e-15

2.kök için ([5,6] aralığında):

Fonksiyon	Bisection metodu	Fixed point metodu	Newton-raphson metodu	Secant metodu
Lagrange interpolasyon polinomu	p20 = 5.47441673278809 tolerans = 4.76837158203125 0e-07	Doğru kök buluna madı (h' a bakınız)	p6 = 5.47441744343755 tolerans =2.049644540373287e-30	p10= 5.47441744343755 tolerans = ± e-16
Newton interpolasyon polinomu	p20 =5.47441673278809 tolerans = 4.768371582031250 e-07	Doğru kök buluna madı	p4 = 5.47441744343755 tolerans = 1.281027837733303e-29	p7 =5.4744174434375 5 tolerans = ± 3.54 e-12
f(x) polinomu	p20 = 5.5200770538330 tolerans =4.76837158203125 0e-07	Doğru kök buluna madı	p3=5.52007811028631 tolerans = 7.012118033366270e-21	p7=5.52007811028 631 tolerans = ± e-16

Yukardaki tablodan Newton metodunun kökü en hızlı şekilde bulduğunu görebiliriz , mesela newtonun dördüncü iterasyonunda bulunan kök secant metodu ile 8.inci iterasyonda bulunmaktadır, bisection metodu ise daha fazla iterasyonda bulmaktadır. Ama unutulmaması gereken bir şey var Newton metodun da p_0 'a 2.5 değeri verildi kökten daha uzak bir değer verilmiş olasayıda daha fazla iterasyona ihtiyaç duyardı, ve bisectionda eğer fonksiyon aralığı değil de daha fazla köke yakın aralıkta bir aralık verilseydi daha hızlı bulabilirdi, yine de Newton-raphsonun yaklaşma hızı sekant yönteminden büyüktür, sekant yönteminin yaklaşma hızı ise bisectiondan büyüktür.

Sabit nokta metoduna gelince fonksiyonda bir sabit noktaya yaklaşma olmadı onun için kök bulunamadı, böylece sabit nokta iterasyonun sadece spesifik problemlerde uygulanabileceğini görürüz ve bizim fonksiyonumuzda kullanılmaya uygun değil.

j) Lagrange interpolasyonu ve Newton interpolasyonu ile fonksiyon oturturmayı öğrendim ve bu iki metodun aynı sonucu verdieneni gözlemledim.

Ayrıca kök bulma metodlarının avantajları ve dezavantajlarını öğrendim: Newton metodunun avantajı köke hızlı yaklaşmasıdır buna karşın isterleri bir başlangıç noktası ve fonksiyonun türevi , her zaman fonksiyonun türevini bulmak mümkün olmayabilir bunun için nümerik türev kullanılabilir , bu ihtiyacı kalmak için sekant yöntemi geliştirilmiş ve fonksiyonun türevini kullanmadan bir birine yakın iki nokta seçerek (p_0 , p_1) kökün bulunması sağlanmaktadır buradaki ihtiyacımız ise ikinci bir noktadır ve genellikle bu ihtiyacı sağlamak mümkün değildir. Bisection'da ise tahmini bir aralık vermekteyiz Newton ve sekant metodundan daha yavaş ama bir türeve ihtiyaç duyulmamaktadır, sekant yöntemiyle ile çözünmeyen problemler daha fazla işlem yüküne katlanarak bisection ile çözülebilir. Ayrıca MATLAB kullanmayı , fonksiyon , vektör ve tablo oluşturmayı ve onlarla işlem yapmayı öğrendim.