ENS409 Project 2 Report İlker Gül - 26352

Question 1

Data is generated using function $f(x) = 3xe^x - e^{2x}$. Compute the exact value of the function at $x = 1.03$ and find the absolute approximate error with respect to obtained approximations.

Solution

Now the function is given, and the generated data is given we have 6 points. Thus, we can come up with 5th degree interpolation with Newton and Lagrange. Furthermore, we can have 11th degree Hermite interpolation. Since it asks for absolute approximate error. I will find all possible degree functions and their scores. Furthermore, in the end I will display the absolute true error with the function input point 1.03 and the approximate value by using interpolating function.

```
function newtoninterpolation(input, xo, givenoutputs)
    myfunc = @(x)3*x*exp(x)- exp(2*x);
    secondfunc = @(x)1/(1+(x*x));
    thirdfunc = @(x)sqrt(1 + x^2);
    out = myfunc(input);
    fprintf('The real value found by given function is %.8f', out);
    fprintf('\n')
    degree = length(xo) -1;
    DD = zeros(length(xo),length(givenoutputs));
    DD(:,1) = givenoutputs;
    for k = 2: degree + 1
        for i = 1: degree + 2 - k
            DD(i, k) = (DD(i+1,k-1)-DD(i, k-1))/(xo(i + k -1) -
xo(i));
        end
    end
    f1 = @(x)DD(1,1) +DD(1,2)*(x- xo(1));
    f2 = @(x)DD(1,1) +DD(1,2)*(x- xo(1)) + DD(1,3)*(x- xo(1))*(x-
xo(2));
    f3 = @(x)DD(1,1) +DD(1,2)*(x- xo(1)) + DD(1,3)*(x- xo(1))*(x-
xo(2)) + DD(1,4)*(x- xo(1))*(x- xo(2))*(x - xo(3));
    f4 = @(x)DD(1,1) +DD(1,2)*(x- xo(1)) + DD(1,3)*(x- xo(1))*(x-
xo(2)) + DD(1,4)*(x- xo(1))*(x- xo(2))*(x - xo(3)) + DD(1,5)*(x-
xo(1))*(x- xo(2))*(x - xo(3))*(x-xo(4));
    interpol = @(x)DD(1,1) +DD(1,2)*(x- xo(1)) + DD(1,3)*(x-
xo(1))*(x- xo(2)) + DD(1,4)*(x- xo(1))*(x- xo(2))*(x - xo(3)) +
DD(1,5)*(x- xo(1))*(x- xo(2))*(x - xo(3))*(x-xo(4)) + DD(1,6)*(x-
xo(1))*(x- xo(2))*(x - xo(3))*(x-xo(4))*(x-xo(5));
    o1 = f1(input);
    o2 = f2(input);
    o3 = f3(input);
    o4 = f4(input);
    out2 = interpol(input);
```

```
    fprintf('The linear interpolation value is %.8f', o1);
    fprintf('\n')
    fprintf('The quadratic interpolation value is %.8f', o2);
    fprintf('\n')
    fprintf('The cubic interpolation value is %.8f', o3);
    fprintf('\n')
    fprintf('The fourth degree interpolation value is %.8f', o4);
    fprintf('\n')
    fprintf('The fifth degree interpolation value is %.8f', out2);
    fprintf('\n')
    aae = abs(o4 - out2);
    fprintf('The absolute approximate error is %.8f', aae);
    fprintf('\n')
    aae1 = abs(out-out2);
    fprintf('The absolute true error is %.8f', aae1);
    fprintf('\n')
end
```

In the function above, I used Divided difference technique to come up with the coefficients of the x-degrees. I calculated the divided difference with the given values that are generated and given.

	1	2	3	4	5	6
1	0.7658	1.3928	-3.1114	-5.2897	-2.6512	-10.9036
2	0.8354	1.1750	-3.5875	-5.8199	-5.1591	0
3	0.8589	1.0315	-4.4605	-6.7486	0	0
4	0.8796	0.4516	-5.5403	0	0	0
5	0.9292	-0.3240	0	0	0	0
6	0.9195	0	0	0	0	0
7						

These are the values, I found by in the Divided difference matrix. I used the algorithm that we have discussed in the lectures.

```
The real value found by given function is 0.80932362
The linear interpolation value is 0.80757400
The quadratic interpolation value is 0.80944086
The cubic interpolation value is 0.80931390
The fourth degree interpolation value is 0.80931772
The fifth degree interpolation value is 0.80931505
The absolute approximate error is 0.00000267
The absolute true error is 0.00000857
fx >>
```

These are the results that I have obtained

Then, I used the Lagrange interpolation

```
function Lagrangeinterpolation(input, x1, y1)
    myfunc = @(x)3*x*exp(x)- exp(2*x);
    Lag10 = @(x)((x-x1(2))/(x1(1)-x1(2)));
    Lag11 = @(x)((x-x1(1))/(x1(2)-x1(1)));
    lagf1 = @(x)y1(1)*Lag10(x) + y1(2)*Lag11(x);


    Lag20 = @(x)((x - x1(2))*(x - x1(3)))/((x1(1) - x1(2))*(x1(1) -
x1(3)));
    Lag21 = @(x)((x - x1(1))*(x - x1(3)))/((x1(2) - x1(1))*(x1(2) -
x1(3)));
    Lag22 = @(x)((x - x1(1))*(x - x1(2)))/((x1(3) - x1(1))*(x1(3) -
x1(2)));
    lagf2 = @(x)y1(1)*Lag20(x) + y1(2)*Lag21(x) + y1(3)*Lag22(x);


    Lag30 = @(x)((x - x1(2))*(x - x1(3))*(x - x1(4)))/((x1(1) -
x1(2))*(x1(1) - x1(3))*(x1(1) - x1(4)));
    Lag31 = @(x)((x - x1(1))*(x - x1(3))*(x - x1(4)))/((x1(2) -
x1(1))*(x1(2) - x1(3))*(x1(2) - x1(4)));
    Lag32 = @(x)((x - x1(1))*(x - x1(2))*(x - x1(4)))/((x1(3) -
x1(1))*(x1(3) - x1(2))*(x1(3) - x1(4)));
    Lag33 = @(x)((x - x1(1))*(x - x1(2))*(x - x1(3)))/((x1(4) -
x1(1))*(x1(4) - x1(2))*(x1(4) - x1(3)));
    lagf3 = @(x)y1(1)*Lag30(x) + y1(2)*Lag31(x) + y1(3)*Lag32(x) +
y1(4)*Lag33(x);


    Lag40 = @(x)((x - x1(2))*(x - x1(3))*(x - x1(4))*(x -
x1(5)))/((x1(1) - x1(2))*(x1(1) - x1(3))*(x1(1) - x1(4))*(x1(1) -
x1(5)));
    Lag41 = @(x)((x - x1(1))*(x - x1(3))*(x - x1(4))*(x -
x1(5)))/((x1(2) - x1(1))*(x1(2) - x1(3))*(x1(2) - x1(4))*(x1(2) -
x1(5)));
    Lag42 = @(x)((x - x1(1))*(x - x1(2))*(x - x1(4))*(x -
x1(5)))/((x1(3) - x1(1))*(x1(3) - x1(2))*(x1(3) - x1(4))*(x1(3) -
x1(5)));
    Lag43 = @(x)((x - x1(1))*(x - x1(2))*(x - x1(3))*(x -
x1(5)))/((x1(4) - x1(1))*(x1(4) - x1(2))*(x1(4) - x1(3))*(x1(4) -
x1(5)));
    Lag44 = @(x)((x - x1(1))*(x - x1(2))*(x - x1(3))*(x -
x1(4)))/((x1(5) - x1(1))*(x1(5) - x1(2))*(x1(5) - x1(3))*(x1(5) -
x1(4)));
    lagf4 = @(x)y1(1)*Lag40(x) + y1(2)*Lag41(x) + y1(3)*Lag42(x) +
y1(4)*Lag43(x) + y1(5)*Lag44(x);


    Lag50 = @(x)((x - x1(2))*(x - x1(3))*(x - x1(4))*(x - x1(5))*(x
- x1(6)))/((x1(1) - x1(2))*(x1(1) - x1(3))*(x1(1) - x1(4))*(x1(1) -
x1(5))*(x1(1) - x1(6)));
    Lag51 = @(x)((x - x1(1))*(x - x1(3))*(x - x1(4))*(x - x1(5))*(x
- x1(6)))/((x1(2) - x1(1))*(x1(2) - x1(3))*(x1(2) - x1(4))*(x1(2) -
x1(5))*(x1(2) - x1(6)));
    Lag52 = @(x)((x - x1(1))*(x - x1(2))*(x - x1(4))*(x - x1(5))*(x
- x1(6)))/((x1(3) - x1(1))*(x1(3) - x1(2))*(x1(3) - x1(4))*(x1(3) -
x1(5))*(x1(3) - x1(6)));
```

```
    Lag53 = @(x)((x - x1(1))*(x - x1(2))*(x - x1(3))*(x - x1(5))*(x
- x1(6)))/((x1(4) - x1(1))*(x1(4) - x1(2))*(x1(4) - x1(3))*(x1(4) -
x1(5))*(x1(4) - x1(6)));
    Lag54 = @(x)((x - x1(1))*(x - x1(2))*(x - x1(3))*(x - x1(4))*(x
- x1(6)))/((x1(5) - x1(1))*(x1(5) - x1(2))*(x1(5) - x1(3))*(x1(5) -
x1(4))*(x1(5) - x1(6)));
    Lag55 = @(x)((x - x1(1))*(x - x1(2))*(x - x1(3))*(x - x1(4))*(x
- x1(5)))/((x1(6) - x1(1))*(x1(6) - x1(2))*(x1(6) - x1(3))*(x1(6) -
x1(4))*(x1(6) - x1(5)));
    lagf5 = @(x)y1(1)*Lag50(x) + y1(2)*Lag51(x) + y1(3)*Lag52(x) +
y1(4)*Lag53(x) + y1(5)*Lag54(x) + y1(6)*Lag55(x);
    out = myfunc(input);
    fprintf('The value is %.8f', out);
    fprintf('\n')
    o1 = lagf1(input);
    o2 = lagf2(input);
    o3 = lagf3(input);
    o4 = lagf4(input);
    out2 = lagf5(input);
    fprintf('The Lagrange linear interpolation value is %.8f', o1);
    fprintf('\n')
    fprintf('The Lagrange quadratic interpolation value is %.8f',
o2);
    fprintf('\n')
    fprintf('The Lagrange cubic interpolation value is %.8f', o3);
    fprintf('\n')
    fprintf('The Lagrange fourth degree interpolation value is
%.8f', o4);
    fprintf('\n')
    fprintf('The Lagrange fifth degree interpolation value is %.8f',
out2);
    fprintf('\n')
    aae = abs(o4 - out2);
    fprintf('The absolute approximate error is %.8f', aae);
    fprintf('\n')
    aae1 = abs(out-out2);
    fprintf('The absolute true error is %.8f', aae1);
    fprintf('\n')
end
```

I used the technique we have seen in the class to calculate the Lagrange polynomials. Then, I have shown absolute approximate between 5^{th} and 4^{th} degree and absolute true error.

```
The value is 0.80932362
The Lagrange linear interpolation value is 0.80757400
The Lagrange quadratic interpolation value is 0.80944086
The Lagrange cubic interpolation value is 0.80931390
The Lagrange fourth degree interpolation value is 0.80931772
The Lagrange fifth degree interpolation value is 0.80931505
The absolute approximate error is 0.00000267
The absolute true error is 0.00000857
fx >>
```

As you can see the scores we obtained with lagrange are same as Newton since these two are techniques that are used to find the unique interpolation function.

Now, I have implemented the Hermite. So, I also used the derivate value of the points.

```
function Iout = Hermiteinterpolation(input, x1, y1, y2)
    firstfunc = @(x)3*x*exp(x)- exp(2*x);
    firstderivative = @(x)-exp(x)*(2*exp(x) - 3*x - 3);
    syms x
%     secondfunc = @(x)1/(1+(x*x));
%     secondderivative = @(x)-2*x/(x^2+1)^2;
%     thirdfunc = @(x)sqrt(1 + x^2);
%     thirdderivative = @(x)x/sqrt(1 + x^2);
    Lag10 = @(x)((x-x1(2))/(x1(1)-x1(2)));
    Lag11 = @(x)((x-x1(1))/(x1(2)-x1(1)));
    Hcap10 = @(x)(x - x1(1))*(Lag10(x)^2);
    Hcap11 = @(x)(x - x1(2))*(Lag11(x)^2);
    Lag10d = eval(['@(x)' char(diff(Lag10(x)))]);
    Lag11d = eval(['@(x)' char(diff(Lag11(x)))]);
    H10 = @(x)(1 - 2*(x - x1(1))*Lag10d(x1(1)))*(Lag10(x)^2);
    H11 = @(x)(1 - 2*(x - x1(2))*Lag11d(x1(2)))*(Lag11(x)^2);
    HermiteFinal1 = @(x)y1(1)*H10(x) + y1(2)*H11(x) +
y2(1)*Hcap10(x) + y2(2)*Hcap11(x);


    Lag20 = @(x)((x - x1(2))*(x - x1(3)))/((x1(1) - x1(2))*(x1(1) -
x1(3)));
    Lag21 = @(x)((x - x1(1))*(x - x1(3)))/((x1(2) - x1(1))*(x1(2) -
x1(3)));
    Lag22 = @(x)((x - x1(1))*(x - x1(2)))/((x1(3) - x1(1))*(x1(3) -
x1(2)));
    Hcap20 = @(x)(x - x1(1))*(Lag20(x)^2);
    Hcap21 = @(x)(x - x1(2))*(Lag21(x)^2);
    Hcap22 = @(x)(x - x1(3))*(Lag22(x)^2);
    Lag20d = eval(['@(x)' char(diff(Lag20(x)))]);
    Lag21d = eval(['@(x)' char(diff(Lag21(x)))]);
    Lag22d = eval(['@(x)' char(diff(Lag22(x)))]);
    H20 = @(x)(1 - 2*(x - x1(1))*Lag20d(x1(1)))*(Lag20(x)^2);
    H21 = @(x)(1 - 2*(x - x1(2))*Lag21d(x1(2)))*(Lag21(x)^2);
    H22 = @(x)(1 - 2*(x - x1(3))*Lag22d(x1(3)))*(Lag22(x)^2);
    HermiteFinal2 = @(x)y1(1)*H20(x) + y1(2)*H21(x) + y1(3)*H22(x)+
y2(1)*Hcap20(x) + y2(2)*Hcap21(x) + y2(3)*Hcap22(x);


    Lag30 = @(x)((x - x1(2))*(x - x1(3))*(x - x1(4)))/((x1(1) -
x1(2))*(x1(1) - x1(3))*(x1(1) - x1(4)));
    Lag31 = @(x)((x - x1(1))*(x - x1(3))*(x - x1(4)))/((x1(2) -
x1(1))*(x1(2) - x1(3))*(x1(2) - x1(4)));
    Lag32 = @(x)((x - x1(1))*(x - x1(2))*(x - x1(4)))/((x1(3) -
x1(1))*(x1(3) - x1(2))*(x1(3) - x1(4)));
    Lag33 = @(x)((x - x1(1))*(x - x1(2))*(x - x1(3)))/((x1(4) -
x1(1))*(x1(4) - x1(2))*(x1(4) - x1(3)));
    Hcap30 = @(x)(x - x1(1))*(Lag30(x)^2);
    Hcap31 = @(x)(x - x1(2))*(Lag31(x)^2);
    Hcap32 = @(x)(x - x1(3))*(Lag32(x)^2);
```

```
    Hcap33 = @(x)(x - x1(4))*(Lag33(x)^2);
    Lag30d = eval(['@(x)' char(diff(Lag30(x)))]);
    Lag31d = eval(['@(x)' char(diff(Lag31(x)))]);
    Lag32d = eval(['@(x)' char(diff(Lag32(x)))]);
    Lag33d = eval(['@(x)' char(diff(Lag33(x)))]);
    H30 = @(x)(1 - 2*(x - x1(1))*Lag30d(x1(1)))*(Lag30(x)^2);
    H31 = @(x)(1 - 2*(x - x1(2))*Lag31d(x1(2)))*(Lag31(x)^2);
    H32 = @(x)(1 - 2*(x - x1(3))*Lag32d(x1(3)))*(Lag32(x)^2);
    H33 = @(x)(1 - 2*(x - x1(4))*Lag33d(x1(4)))*(Lag33(x)^2);
    HermiteFinal3 = @(x)y1(1)*H30(x) + y1(2)*H31(x) + y1(3)*H32(x) +
y1(4)*H33(x) + y2(1)*Hcap30(x) + y2(2)*Hcap31(x) + y2(3)*Hcap32(x) +
y2(4)*Hcap33(x);


    Lag40 = @(x)((x - x1(2))*(x - x1(3))*(x - x1(4))*(x -
x1(5)))/((x1(1) - x1(2))*(x1(1) - x1(3))*(x1(1) - x1(4))*(x1(1) -
x1(5)));
    Lag41 = @(x)((x - x1(1))*(x - x1(3))*(x - x1(4))*(x -
x1(5)))/((x1(2) - x1(1))*(x1(2) - x1(3))*(x1(2) - x1(4))*(x1(2) -
x1(5)));
    Lag42 = @(x)((x - x1(1))*(x - x1(2))*(x - x1(4))*(x -
x1(5)))/((x1(3) - x1(1))*(x1(3) - x1(2))*(x1(3) - x1(4))*(x1(3) -
x1(5)));
    Lag43 = @(x)((x - x1(1))*(x - x1(2))*(x - x1(3))*(x -
x1(5)))/((x1(4) - x1(1))*(x1(4) - x1(2))*(x1(4) - x1(3))*(x1(4) -
x1(5)));
    Lag44 = @(x)((x - x1(1))*(x - x1(2))*(x - x1(3))*(x -
x1(4)))/((x1(5) - x1(1))*(x1(5) - x1(2))*(x1(5) - x1(3))*(x1(5) -
x1(4)));
    Hcap40 = @(x)(x - x1(1))*(Lag40(x)^2);
    Hcap41 = @(x)(x - x1(2))*(Lag41(x)^2);
    Hcap42 = @(x)(x - x1(3))*(Lag42(x)^2);
    Hcap43 = @(x)(x - x1(4))*(Lag43(x)^2);
    Hcap44 = @(x)(x - x1(5))*(Lag44(x)^2);
    Lag40d = eval(['@(x)' char(diff(Lag40(x)))]);
    Lag41d = eval(['@(x)' char(diff(Lag41(x)))]);
    Lag42d = eval(['@(x)' char(diff(Lag42(x)))]);
    Lag43d = eval(['@(x)' char(diff(Lag43(x)))]);
    Lag44d = eval(['@(x)' char(diff(Lag44(x)))]);
    H40 = @(x)(1 - 2*(x - x1(1))*Lag40d(x1(1)))*(Lag40(x)^2);
    H41 = @(x)(1 - 2*(x - x1(2))*Lag41d(x1(2)))*(Lag41(x)^2);
    H42 = @(x)(1 - 2*(x - x1(3))*Lag42d(x1(3)))*(Lag42(x)^2);
    H43 = @(x)(1 - 2*(x - x1(4))*Lag43d(x1(4)))*(Lag43(x)^2);
    H44 = @(x)(1 - 2*(x - x1(5))*Lag44d(x1(5)))*(Lag44(x)^2);
    HermiteFinal4 = @(x)y1(1)*H40(x) + y1(2)*H41(x) + y1(3)*H42(x) +
y1(4)*H43(x) + y1(5)*H44(x) + y2(1)*Hcap40(x) + y2(2)*Hcap41(x) +
y2(3)*Hcap42(x) + y2(4)*Hcap43(x) + y2(5)*Hcap44(x);


    Lag50 = @(x)((x - x1(2))*(x - x1(3))*(x - x1(4))*(x - x1(5))*(x
- x1(6)))/((x1(1) - x1(2))*(x1(1) - x1(3))*(x1(1) - x1(4))*(x1(1) -
x1(5))*(x1(1) - x1(6)));
    Lag51 = @(x)((x - x1(1))*(x - x1(3))*(x - x1(4))*(x - x1(5))*(x
- x1(6)))/((x1(2) - x1(1))*(x1(2) - x1(3))*(x1(2) - x1(4))*(x1(2) -
x1(5))*(x1(2) - x1(6)));
    Lag52 = @(x)((x - x1(1))*(x - x1(2))*(x - x1(4))*(x - x1(5))*(x
- x1(6)))/((x1(3) - x1(1))*(x1(3) - x1(2))*(x1(3) - x1(4))*(x1(3) -
x1(5))*(x1(3) - x1(6)));
```

```
    Lag53 = @(x)((x - x1(1))*(x - x1(2))*(x - x1(3))*(x - x1(5))*(x
- x1(6)))/((x1(4) - x1(1))*(x1(4) - x1(2))*(x1(4) - x1(3))*(x1(4) -
x1(5))*(x1(4) - x1(6)));
    Lag54 = @(x)((x - x1(1))*(x - x1(2))*(x - x1(3))*(x - x1(4))*(x
- x1(6)))/((x1(5) - x1(1))*(x1(5) - x1(2))*(x1(5) - x1(3))*(x1(5) -
x1(4))*(x1(5) - x1(6)));
    Lag55 = @(x)((x - x1(1))*(x - x1(2))*(x - x1(3))*(x - x1(4))*(x
- x1(5)))/((x1(6) - x1(1))*(x1(6) - x1(2))*(x1(6) - x1(3))*(x1(6) -
x1(4))*(x1(6) - x1(5)));
    Hcap50 = @(x)(x - x1(1))*(Lag50(x)^2);
    Hcap51 = @(x)(x - x1(2))*(Lag51(x)^2);
    Hcap52 = @(x)(x - x1(3))*(Lag52(x)^2);
    Hcap53 = @(x)(x - x1(4))*(Lag53(x)^2);
    Hcap54 = @(x)(x - x1(5))*(Lag54(x)^2);
    Hcap55 = @(x)(x - x1(6))*(Lag55(x)^2);


    Lag50d = eval(['@(x)' char(diff(Lag50(x)))]);
    Lag51d = eval(['@(x)' char(diff(Lag51(x)))]);
    Lag52d = eval(['@(x)' char(diff(Lag52(x)))]);
    Lag53d = eval(['@(x)' char(diff(Lag53(x)))]);
    Lag54d = eval(['@(x)' char(diff(Lag54(x)))]);
    Lag55d = eval(['@(x)' char(diff(Lag55(x)))]);
    H50 = @(x)(1 - 2*(x - x1(1))*Lag50d(x1(1)))*(Lag50(x)^2);
    H51 = @(x)(1 - 2*(x - x1(2))*Lag51d(x1(2)))*(Lag51(x)^2);
    H52 = @(x)(1 - 2*(x - x1(3))*Lag52d(x1(3)))*(Lag52(x)^2);
    H53 = @(x)(1 - 2*(x - x1(4))*Lag53d(x1(4)))*(Lag53(x)^2);
    H54 = @(x)(1 - 2*(x - x1(5))*Lag54d(x1(5)))*(Lag54(x)^2);
    H55 = @(x)(1 - 2*(x - x1(6))*Lag55d(x1(6)))*(Lag55(x)^2);
    HermiteFinal = @(x)y1(1)*H50(x) + y1(2)*H51(x) + y1(3)*H52(x) +
y1(4)*H53(x) + y1(5)*H54(x) + y1(6)*H55(x) + y2(1)*Hcap50(x) +
y2(2)*Hcap51(x) + y2(3)*Hcap52(x) + y2(4)*Hcap53(x) +
y2(5)*Hcap54(x) + y2(6)*Hcap55(x);


    out = firstfunc(input);
    fprintf('The value is %.8f', out);
    fprintf('\n')
    o1 = HermiteFinal1(input);
    o2 = HermiteFinal2(input);
    o3 = HermiteFinal3(input);
    o4 = HermiteFinal4(input);
    out2 = HermiteFinal(input);
    fprintf('The hermite 3rd degree interpolation value is %.8f',
o1);
    fprintf('\n')
    fprintf('The hermite 5h degree interpolation value is %.8f',
o2);
    fprintf('\n')
    fprintf('The hermite 7th degree interpolation value is %.8f',
o3);
    fprintf('\n')
    fprintf('The hermite 8th degree interpolation value is %.8f',
o4);
    fprintf('\n')
    fprintf('The hermite 11th degree interpolation value is %.8f',
out2);
    fprintf('\n')
```

```
    aae = abs(o4 - out2);
    fprintf('The absolute approximate error is %.8f', aae);
    fprintf('\n')
    aae1 = abs(out-out2);
    fprintf('The absolute true error is %.8f', aae1);
    fprintf('\n')
end
```

As you can see I have used the Lagrange interpolation polynomials to find the Hermite interpolation as we have discussed in the lectures.

```
The value is 0.80932362
The hermite 3rd degree interpolation value is 0.80932456
The hermite 5h degree interpolation value is 0.80932408
The hermite 7th degree interpolation value is 0.80931652
The hermite 8th degree interpolation value is 0.80930579
The hermite 11th degree interpolation value is 0.80928733
The absolute approximate error is 0.00001846
The absolute true error is 0.00003629
fx >>
```

These are the scores I have obtained.

Question 2

Interpolate $f(x) = \#$ at evenly spaced points on the interval $[-5, 5]$ with interpolating #$!! polynomials of degree $n = 1, 5, 10$. Plot the data points, interpolating polynomials and $f(x)$ on the same graph. Does interpolation accuracy increase with n?

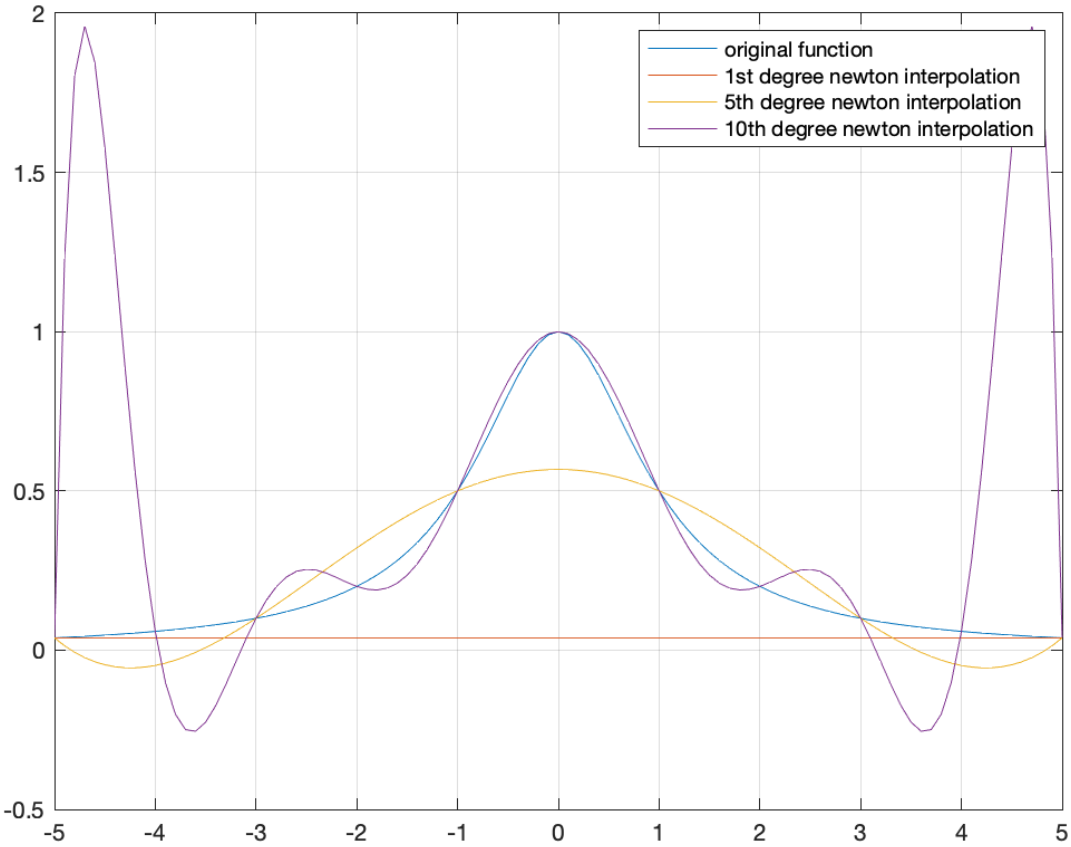
Solution

Now the function is given and polynomial degrees are stated. Now, I am going to implement Newton and Lagrange with n degrees and Hermite with 2n + 1 degree. Then, I am going to plot the polynomials pf different degrees for each technique.

```
%% Newton Interpolation
myf = @(x)1/(1+(x*x));
x1 = [-5,5];
y1 = [];
for i = 1: length(x1)
    y1(i) = myf(x1(i));
end
x2 = [-5, -3, -1, 1, 3, 5];
y2 = [];
for i = 1: length(x2)
    y2(i) = myf(x2(i));
```

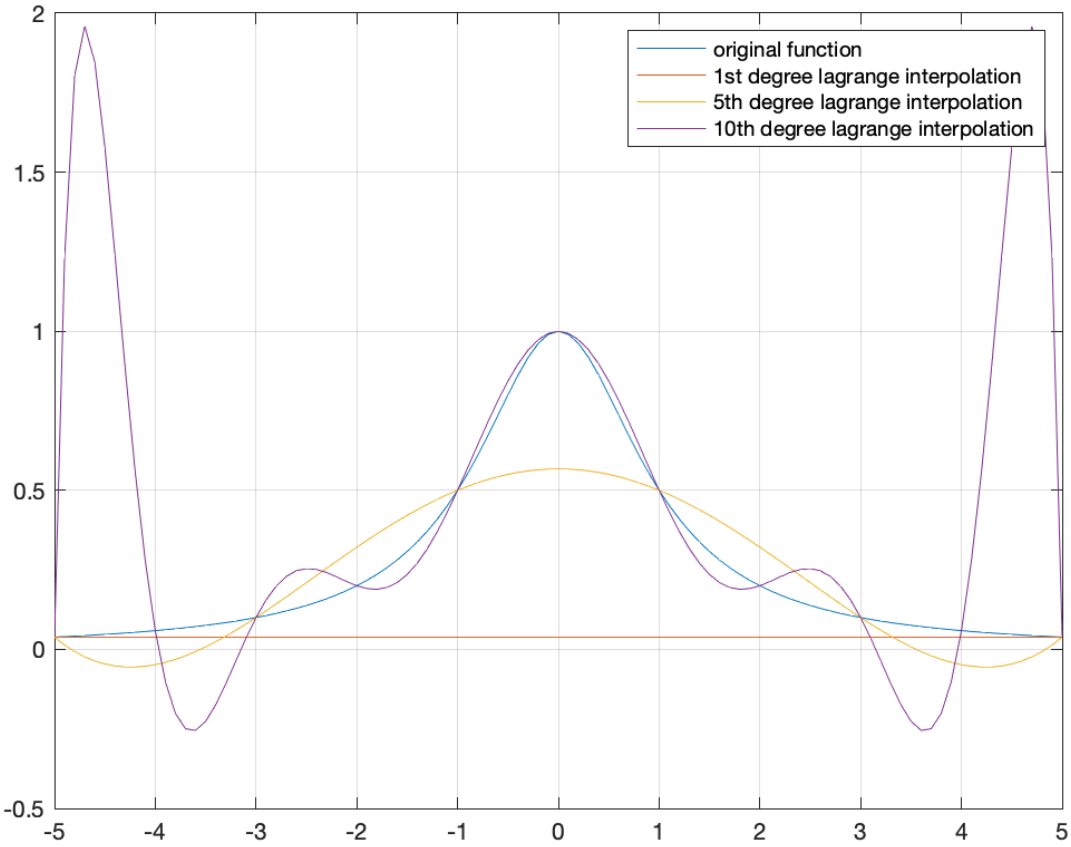
```
end
x3 = [-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5];
y3 = [];
for i = 1: length(x3)
    y3(i) = myf(x3(i));
end
q2newton(x1, y1, x2, y2, x3, y3);
%% Lagrange Interpolation
myf = @(x)1/(1+(x*x));
x1 = [-5,5];
y1 = [];
for i = 1: length(x1)
    y1(i) = myf(x1(i));
end
x2 = [-5, -3, -1, 1, 3, 5];
y2 = [];
for i = 1: length(x2)
    y2(i) = myf(x2(i));
end
x3 = [-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5];
y3 = [];
for i = 1: length(x3)
    y3(i) = myf(x3(i));
end
q2lagrange(x1, y1, x2, y2, x3, y3);
%% Hermite Interpolation
myf = @(x)1/(1+(x*x));
fd = @(x)-2*x/(x^2+1)^2;
x1 = [-5,5];
y1 = [];
y1dev = [];
for i = 1: length(x1)
    y1(i) = myf(x1(i));
    y1dev(i) = fd(x1(i));
end
x2 = [-5, -3, -1, 1, 3, 5];
y2 = [];
y2dev = [];
for i = 1: length(x2)
    y2(i) = myf(x2(i));
    y2dev(i) = fd(x2(i));
end
x3 = [-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5];
y3 = [];
y3dev = [];
for i = 1: length(x3)
    y3(i) = myf(x3(i));
    y3dev(i) = fd(x3(i));
end
q2hermite(x1, y1,y1dev, x2, y2,y2dev, x3, y3, y3dev);
```

This is the q2main.m where I am generating the data points for the given functions. Now, let's check the graphs.



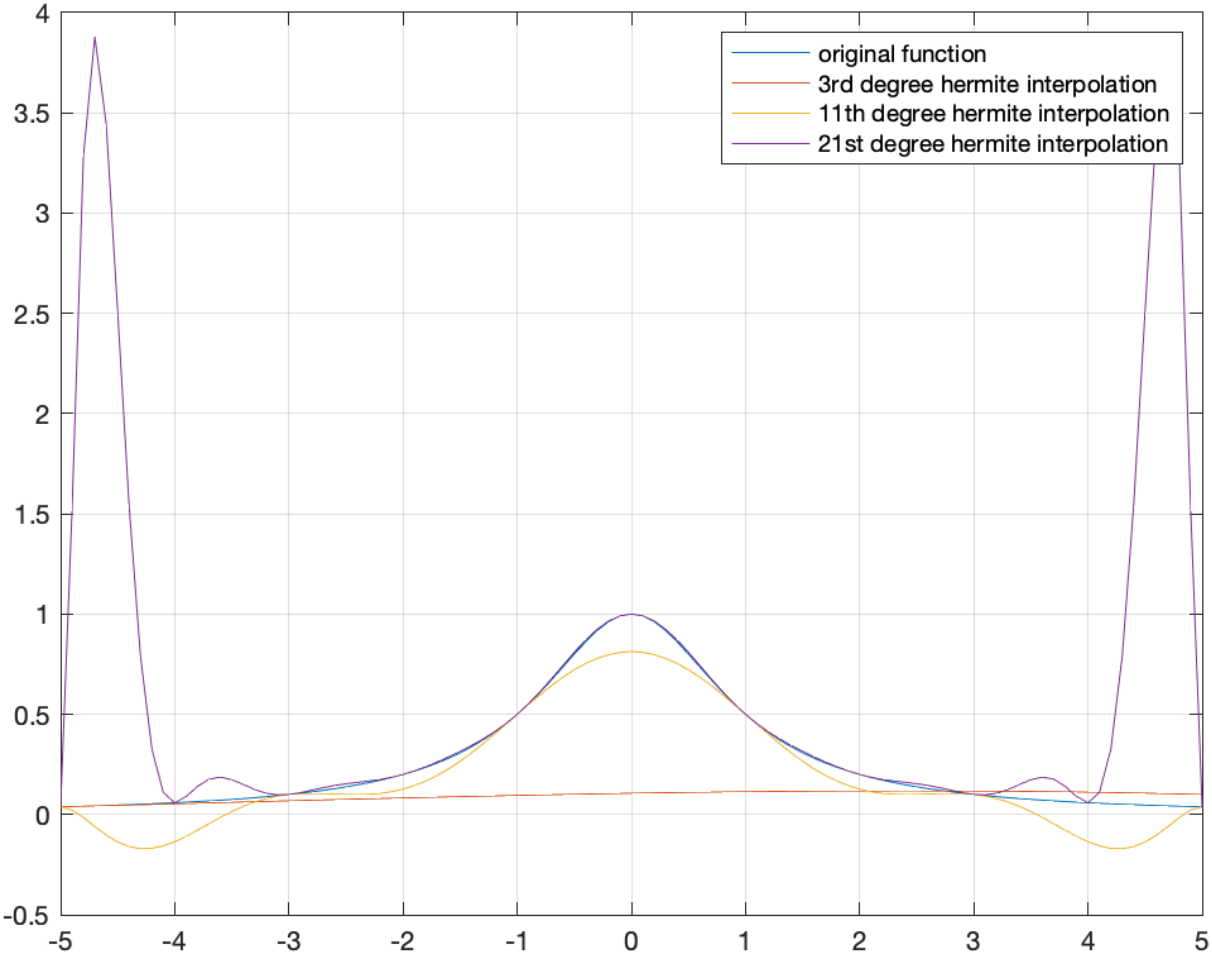
Newton interpolation polynomial

When we increase the degree from 1 to 5 the accuracy increased significantly. However, when we increase the degree from 5 to 10, we get some awkward results for some points in the data. When it is between -2 and 2 the score is very close to real function. However, between -5 and-4 also 4 and 5. It diverges from the real function' behaviour. Thus, one should be careful about selecting the appropriate degree of interpolation. In some intervals, the behaviour can be unexpected.



Lagrange interpolation polynomial

When we increase the degree from 1 to 5 the accuracy increased significantly. However, when we increase the degree from 5 to 10, we get some awkward results for some points in the data. When it is between -2 and 2 the score is very close to real function. However, between -5 and-4 also 4 and 5. It diverges from the real function' behaviour. Thus, one should be careful about selecting the appropriate degree of interpolation. In some intervals, the behaviour can be unexpected.



Hermite interpolation polynomial

When we increase the degree from 3 to 11 the accuracy increased significantly. However, when we increase the degree from 11 to 21, we get some awkward results for some points in the data. When it is between -2 and 2 the score is very close to real function. However, between -5 and-4 also 4 and 5. It diverges from the real function' behaviour. Thus, one should be careful about selecting the appropriate degree of interpolation. In some intervals, the behaviour can be unexpected. In this case I would most probably prefer 11^{th} degree hermite polynomial since it behaves very close to the real function for the whole given range. Although 21^{st} degree polynomial gives very very close result to the real function in some parts of the interval, there are some unexpected results that are very different from the real function. For generality, it may cause unwanted results.

For plotting purposes, I added the following part to the code. The calculation technique for polynomials are same as the previous one.

Newton

```
funcx = -5:0.1:5;
funcy1 = [];
```

```
funcy2 = [];
funcy3 = [];
funcy4 = [];
for i = 1: length(funcx)
    funcy1(i) = myfunc(funcx(i));
    funcy2(i) = interpol(funcx(i));
    funcy3(i) = interpol2(funcx(i));
    funcy4(i) = interpol3(funcx(i));
end
plot(funcx,funcy1,'DisplayName','original function');
hold on
plot(funcx,funcy2,'DisplayName','1st degree newton interpolation')
hold on
plot(funcx,funcy3,'DisplayName','5th degree newton interpolation')
hold on
plot(funcx,funcy4, 'DisplayName','10th degree newton interpolation')
grid on
legend
```

Lagrange

```
funcx = -5:0.1:5;
funcy5 = [];
funcy6 = [];
funcy7 = [];
funcy8 = [];
for i = 1: length(funcx)
    funcy5(i) = myfunc(funcx(i));
    funcy6(i) = lagf1(funcx(i));
    funcy7(i) = lagf2(funcx(i));
    funcy8(i) = lagf3(funcx(i));
end


plot(funcx,funcy5,'DisplayName','original function');
hold on
plot(funcx,funcy6,'DisplayName','1st degree lagrange
interpolation')
hold on
plot(funcx,funcy7,'DisplayName','5th degree lagrange
interpolation')
hold on
plot(funcx,funcy8, 'DisplayName','10th degree lagrange
interpolation')
grid on
legend
```

Hermite

```
funcx = -5:0.1:5;
funcy5 = [];
funcy6 = [];
funcy7 = [];
funcy8 = [];
for i = 1: length(funcx)
    funcy5(i) = myfunc(funcx(i));
    funcy6(i) = HermiteFinal1(funcx(i));
    funcy7(i) = HermiteFinal2(funcx(i));
    funcy8(i) = HermiteFinal3(funcx(i));
```

```
    end

    plot(funcx,funcy5,'DisplayName','original function');
    hold on
    plot(funcx,funcy6,'DisplayName','3rd degree hermite
interpolation')
    hold on
    plot(funcx,funcy7,'DisplayName','11th degree hermite
interpolation')
    hold on
    plot(funcx,funcy8, 'DisplayName','21st degree hermite
interpolation')
    grid on
    legend
```

To answer the question "Does interpolation accuracy increase with n?". We need to examine the plotting figures.

Question 3

Interpolate $f(x) = \sqrt{1+x^2}$ using a set of $n+1$ regularly spaced nodes $x_k = -1 + \frac{2(k-1)}{n}$, $k = 1,2,\dots,n+1$. Take $n = 5, 10$ compute the interpolation polynomials $P_n(x)$ and the error $f(x) - P_n(x)$ at 41 regularly spaced points. Plot the error. Repeat computation by replacing the regularly spaced nodes with nodes $x_k = \cos\left(\frac{2(k-1)}{n}\right)$, $k = 1,2,\dots,n+1$. Compare the accuracies.

```
%% Newton Interpolation
myf = @(x)sqrt(1 + x^2);
x1 = [];
y1 = [];
x2 = [];
y2 = [];
n = 5;
for i = 1: n+1
    x1(i) = -1 + (2*(i-1)/n);
end
for j = 1: length(x1)
    y1(j) = myf(x1(j));
end
n = 10;
for i = 1: n+1
    x2(i) = -1 + (2*(i-1)/n);
end
for j = 1: length(x2)
    y2(j) = myf(x2(j));
end

x3 = [];
```

```
y3 = [];
x4 = [];
y4 = [];
n = 5;
for i = 1: n+1
    x3(i) = cos((2*(i-1)/n));
end
for j = 1: length(x3)
    y3(j) = myf(x3(j));
end
n = 10;
for i = 1: n+1
    x4(i) = cos((2*(i-1)/n));
end
for j = 1: length(x4)
    y4(j) = myf(x4(j));
end
q3newton(x1, y1, x2, y2);

%% Lagrange Interpolation
myf = @(x)sqrt(1 + x^2);
x1 = [];
y1 = [];
x2 = [];
y2 = [];
n = 5;
for i = 1: n+1
    x1(i) = -1 + (2*(i-1)/n);
end
for j = 1: length(x1)
    y1(j) = myf(x1(j));
end
n = 10;
for i = 1: n+1
    x2(i) = -1 + (2*(i-1)/n);
end
for j = 1: length(x2)
    y2(j) = myf(x2(j));
end

x3 = [];
y3 = [];
x4 = [];
y4 = [];
n = 5;
for i = 1: n+1
    x3(i) = cos((2*(i-1)/n));
end
for j = 1: length(x3)
    y3(j) = myf(x3(j));
end
n = 10;
for i = 1: n+1
    x4(i) = cos((2*(i-1)/n));
end
for j = 1: length(x4)
    y4(j) = myf(x4(j));
```

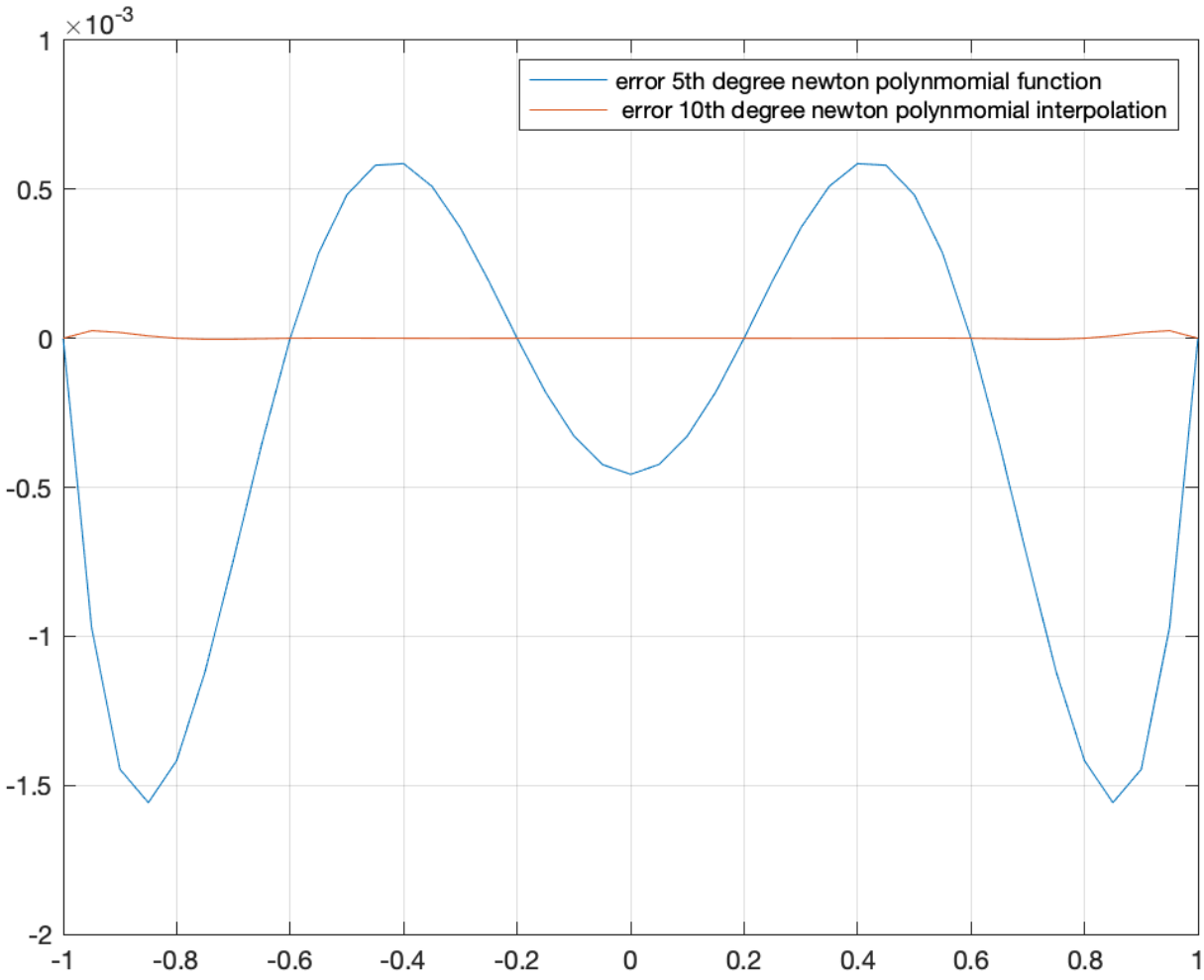
```
end
q3lagrange(x3, y3, x4, y4);

%% Hermite Interpolation
myf = @(x)sqrt(1 + x^2);
hd = @(x)x/sqrt(1 + x^2);
x1 = [];
y1 = [];
y1dev = [];
x2 = [];
y2 = [];
y2dev = [];
n = 5;
for i = 1: n+1
    x1(i) = -1 + (2*(i-1)/n);
end
for j = 1: length(x1)
    y1(j) = myf(x1(j));
    y1dev(j) = hd(x1(j));
end
n = 10;
for i = 1: n+1
    x2(i) = -1 + (2*(i-1)/n);
end
for j = 1: length(x2)
    y2(j) = myf(x2(j));
    y2dev(j) = hd(x2(j));
end

x3 = [];
y3 = [];
y3dev = [];
x4 = [];
y4 = [];
y4dev = [];
n = 5;
for i = 1: n+1
    x3(i) = cos((2*(i-1)/n));
end
for j = 1: length(x3)
    y3(j) = myf(x3(j));
    y3dev(j) = hd(x3(j));
end
n = 10;
for i = 1: n+1
    x4(i) = cos((2*(i-1)/n));
end
for j = 1: length(x4)
    y4(j) = myf(x4(j));
    y4dev(j) = hd(x4(j));
end
q3hermite(x3, y3, y3dev, x4, y4, y4dev);
```

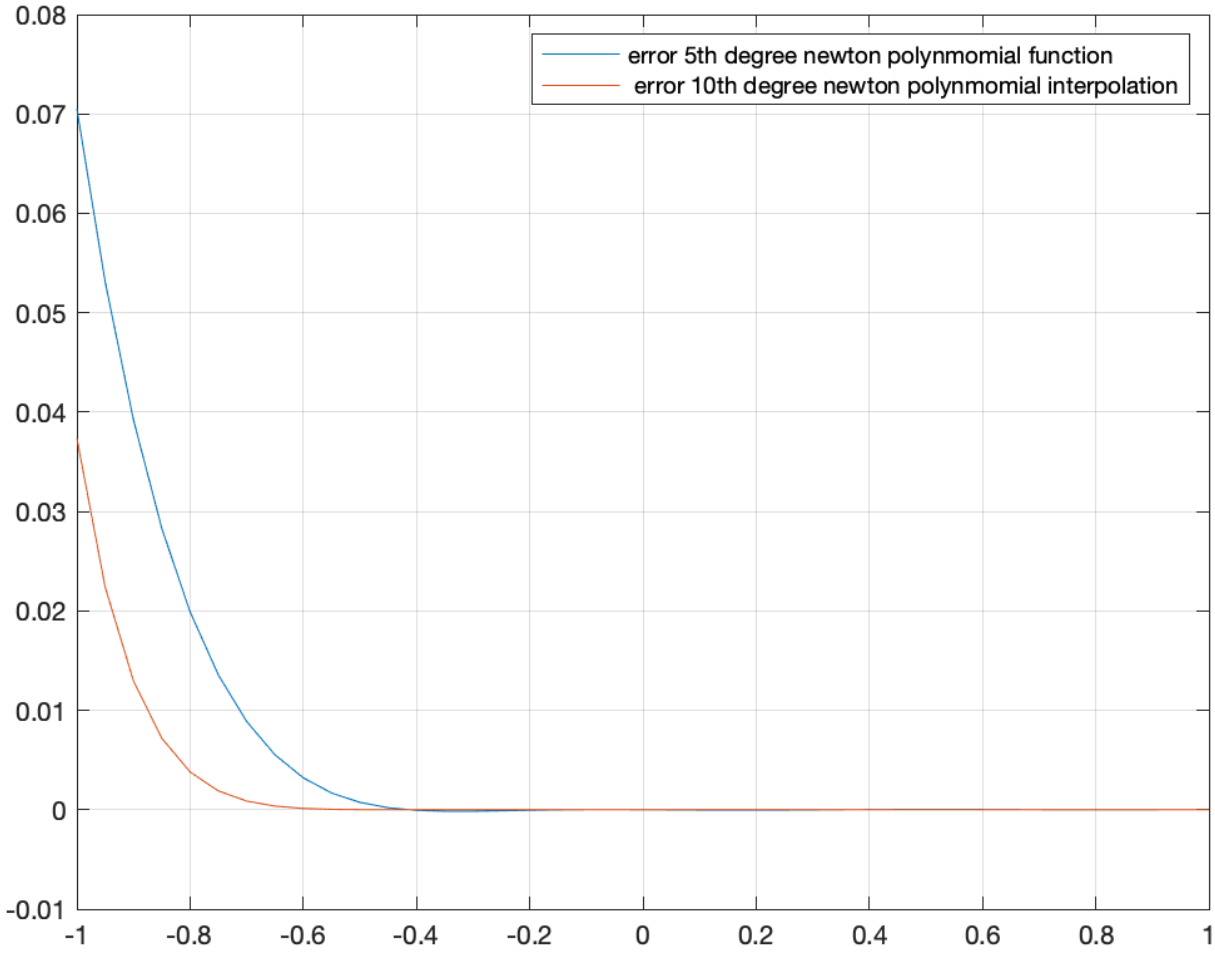
Here, I generated the function input points its values. Furthermore, I have utilized the two different techniques to generate the data points. I will use them to sketch the graph for each of

the function. I am going to need 41 requlary spaced points since the x values will be between -1 and 1, I am going to generate those 41 requlary spaced points between -1 and 1.



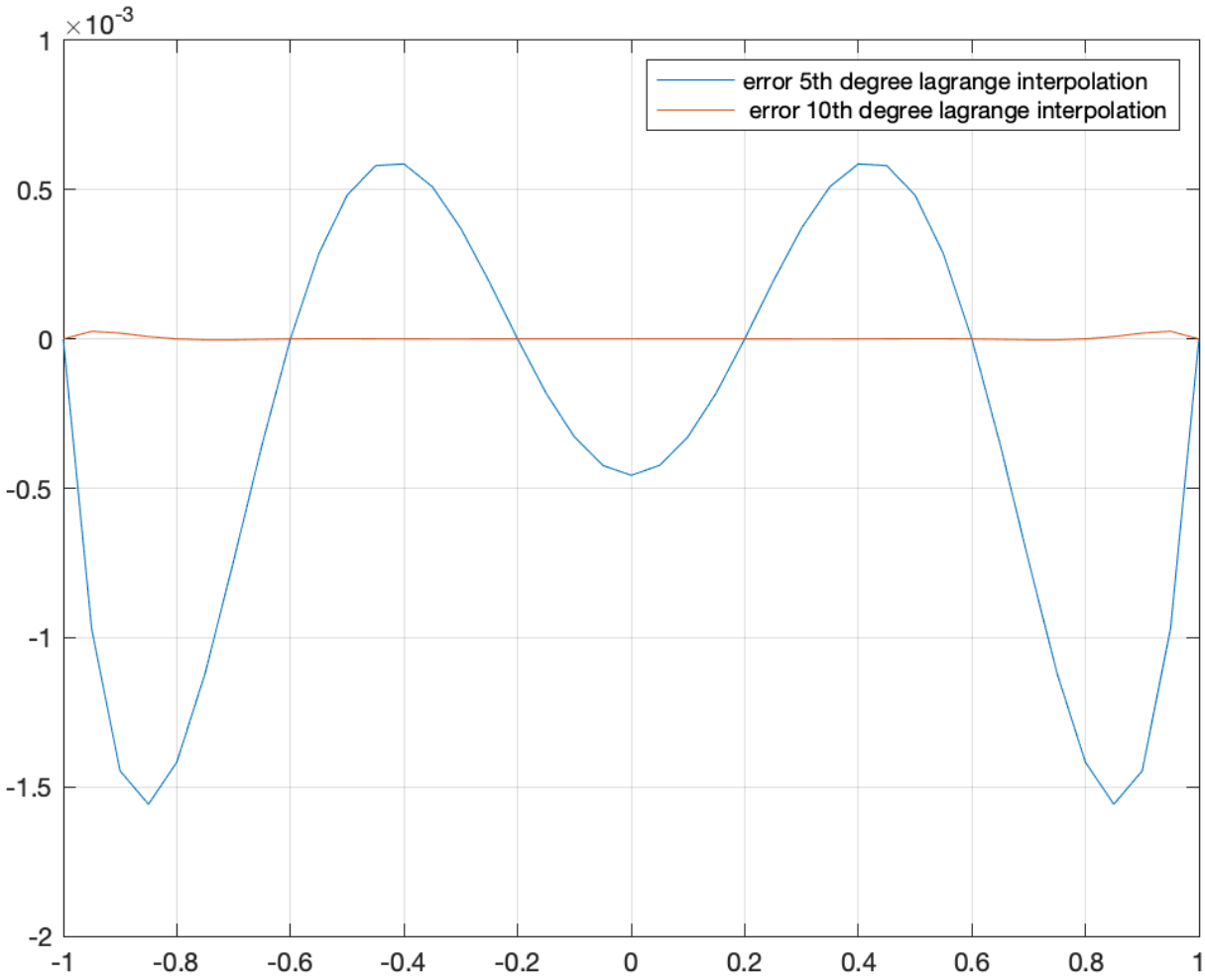
As you can see the error we get from the 10^{th} degree newton polynomial is very close to 0. Here notice that error from 5^{th} degree is also small error value ranges in 10^{-3}.

Then, I have used the latter equation for generating the data points. Here is the error result.



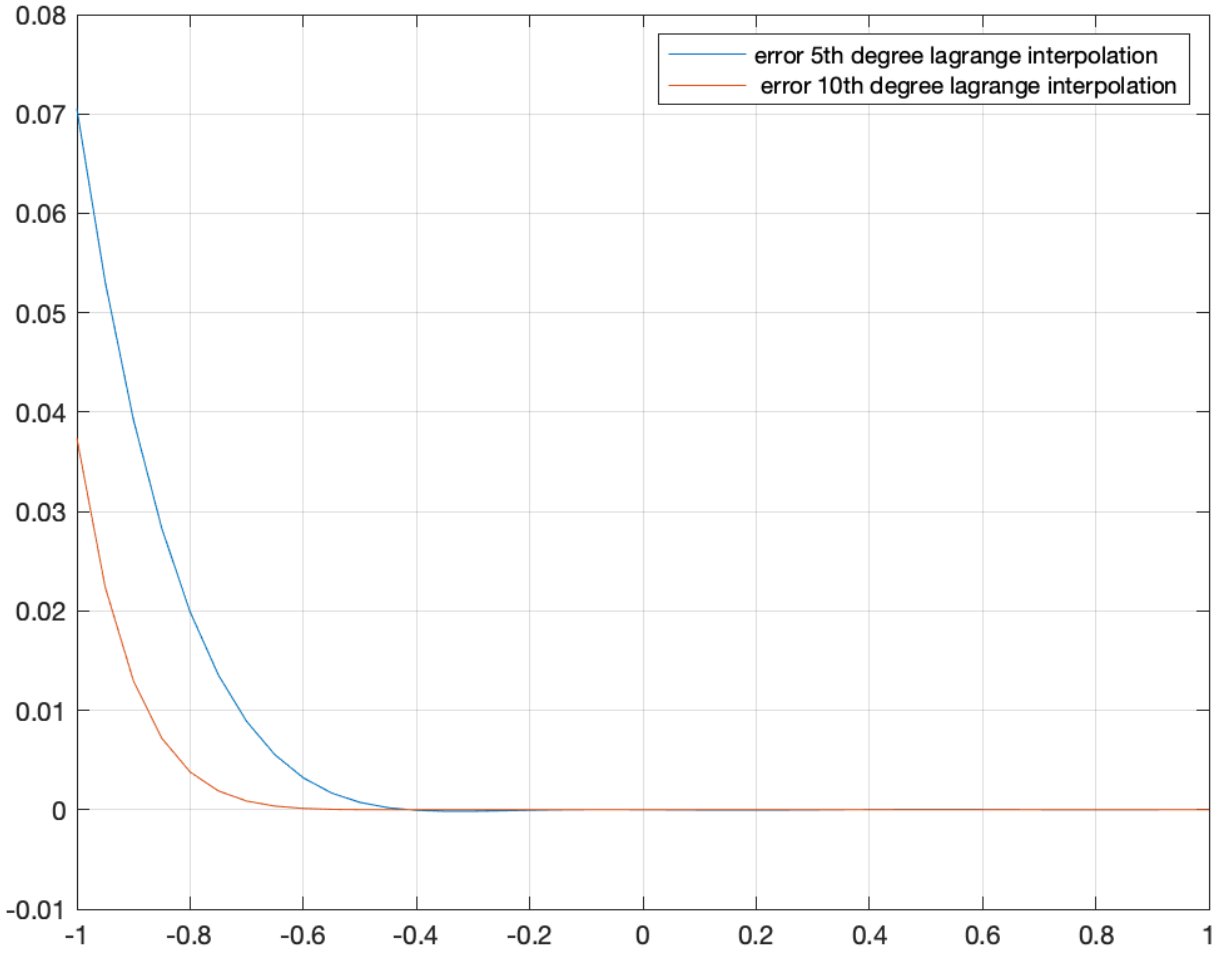
As you can see error behaviour is different than the previous one. However, the 10^{th} degree polynomial seems to be more successful for approximating the value for the data point. When we compare between these two functions, we get less error in the previous one. Thus, the equation used for generating data points influenced the results.

Now, let's check lagrange.



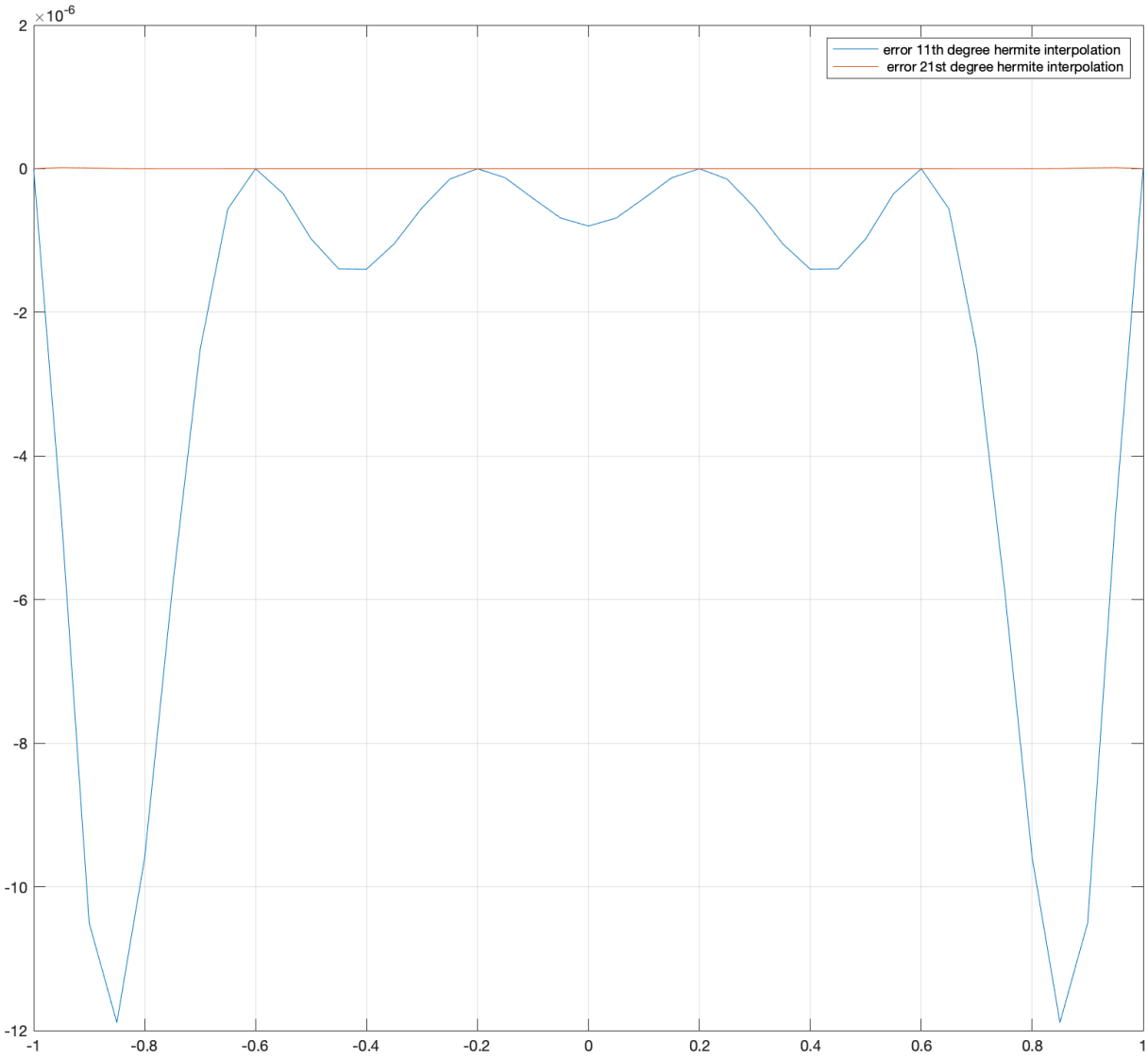
As you can see the error we get from the 10^{th} degree lagrange polynomial is very close to 0. Here notice that error from 5^{th} degree is also small error value ranges in 10^{-3}.

Then, I have used the latter equation for generating the data points. Here is the error result.



As you can see error behaviour is different than the previous one. However, the 10^{th} degree polynomial seems to be more successful for approximating the value for the data point. When we compare between these two functions, we get less error in the previous one. Thus, the equation used for generating data points influenced the results.

Now, let's check the hermite interpolation.

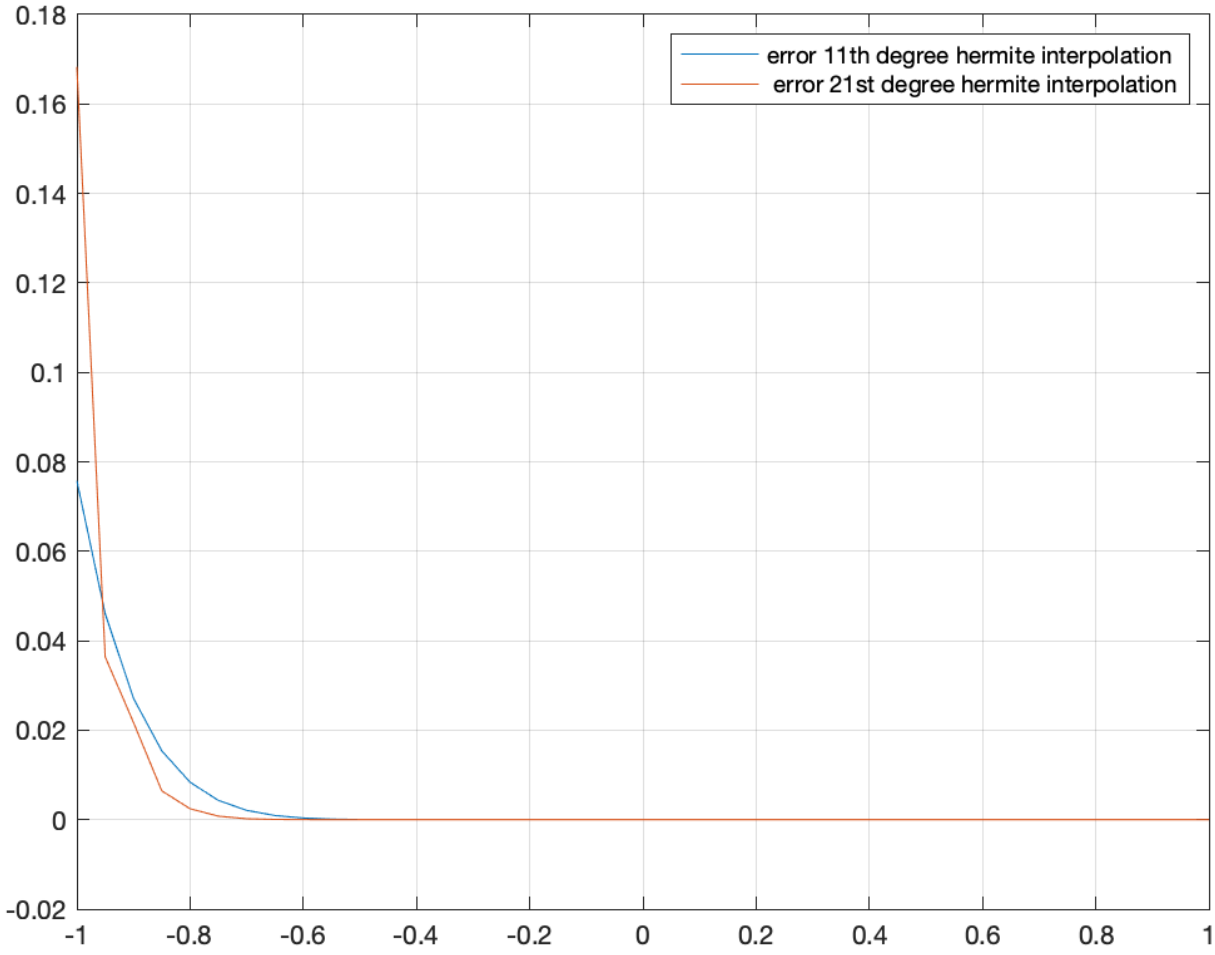


As you can see the error we get from the 21st degree lagrange polynomial is very close to 0.

Here notice that error from 11th degree is actually small error value ranges in 10^{-6}.

Here, 21st degree polynomial gives us a very close result to the actual function.

Then, I have used the latter equation for generating the data points. Here is the error result.



As you can see error behaviour is different than the previous one. The 21st degree polynomial seems to be more successful for approximating the value for the data point. However, -1 and very close point -0.955 21st degree interpolating gave us a worse error than the 11th degree polynomial.

When we compare between these two functions, we get less error in the previous one. Thus, the equation used for generating data points influenced the results. When we compare the techniques. Hermite interpolation function give us a more accurate result. We can say the degree of Hermite and usage of derivative influenced the result.