

Problem 1.

Find a polynomial of degree ≤ 3 that fits the points $(2, 1), (1, 0), (3, -1), (0, 2)$ using the following methods. Convert the polynomial to the standard form $P(x) = ax^3 + bx^2 + cx + d$.

- Solving a system of equations
- Lagrange's formula.
- Newton's formula.

Solution

a) The system

$$\begin{cases} P(2) = 1, \\ P(1) = 0, \\ P(3) = -1, \\ P(0) = 2 \end{cases}$$

can be written in matrix form as

$$Yx = b \implies \begin{bmatrix} 1 & 2 & 4 & 8 \\ 1 & 1 & 1 & 1 \\ 1 & 3 & 9 & 27 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} d \\ c \\ b \\ a \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ -1 \\ 2 \end{bmatrix} \implies x = \begin{bmatrix} 2 \\ -\frac{11}{2} \\ \frac{9}{2} \\ -1 \end{bmatrix}$$

Then $P(x) = -x^3 - \frac{11}{2}x^2 + \frac{9}{2}x + 2$.

b) Use Lagrange's method. We construct the basis functions from

$$P(x) = L_1(x) + 0L_2(x) - L_3(x) + 2L_4(x) = L_1(x) - L_3(x) + 2L_4(x)$$

where

$$L_1(x) = \frac{(x-1)(x-3)(x-0)}{(2-1)(2-3)(2-0)}, \quad L_3(x) = \frac{(x-2)(x-1)(x-0)}{(3-2)(3-1)(3-0)}, \quad L_4(x) = \frac{(x-2)(x-1)(x-3)}{(-2)(-1)(-3)}$$

Then

$$P(x) = -\frac{(x-1)(x-3)(x-0)}{2} - \frac{(x-2)(x-1)(x)}{6} - \frac{(x-2)(x-1)(x-3)}{3}$$

We can simplify this and find $P(x) = -x^3 + \frac{9}{2}x^2 - \frac{11}{2}x + 2$.

c) Use Newton's method. We build a table recursively (this class of algorithm is known as dynamic programming)

Let i, j denote the row and column of the array respectively, set the vector of observations in the first column. Then proceeding left to right, $a_{i,j} = \frac{a_{i,j-1} - a_{i-1,j-1}}{x_j - x_{i+j}}$ for $i = 1, \dots, 4$ and $j = 1, \dots, i$ where x is the data vector $x = (2, 1, 3, 0)$.

1	1	-1.5	-1
0	-0.5	0.5	
-1	-1		
2			

We can read off the first row and obtain our coefficients,

$$P(x) = 1 + (1)(x-2) - \frac{3}{2}(x-2)(x-1) - (1)(x-2)(x-1)(x-0)$$

We can expand and simplify to obtain $P(x) = -x^3 - \frac{11}{2}x^2 + \frac{9}{2}x + 2$.

Problem 2.

Let f be a function such that $f(1) = 3$, $f(2) = 1$, and $f(3) = 0$. Compute the divided difference coefficient $f[1, 2, 3]$.

Solution

The table of divided difference coefficients is:

3	-2	0.5
1	-1	
0		

So

$$f[1, 2, 3] = 0.5$$

Problem 3.

In this problem, you can use the Matlab program posted on the course website and on Canvas that computes the interpolation polynomial. We want to see how well a given function can be approximated by interpolation polynomials. Let f be a function and divide $[-0.6, 0.6]$. Take $N = 61$ uniformly spaced points $-0.6 = x_1 < x_2 < \dots < x_{60} < x_{61} = 0.6$.

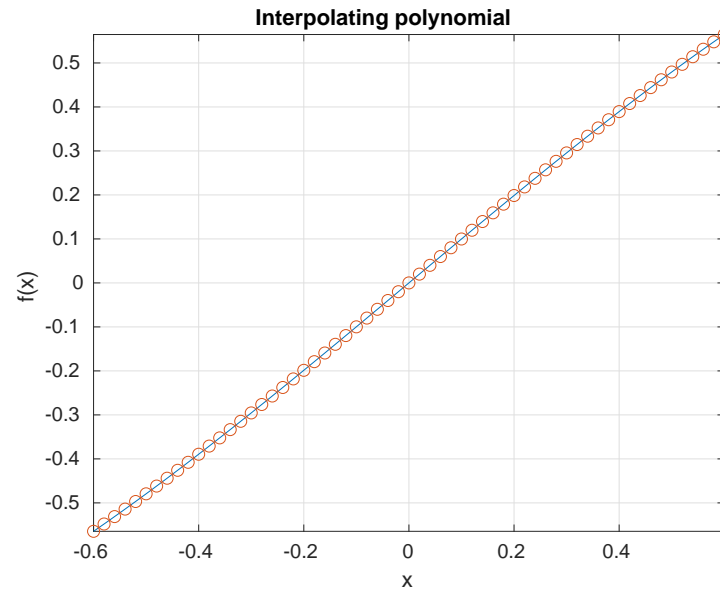
1. Define $f(x) = \sin(x)$. Plot the plot of f and its interpolation polynomial P using the points listed above. Does the interpolation polynomial P well approximate f ?
2. Repeat part a for $f(x) = (1 + x)^{-1}$.
3. The error bound for the interpolating polynomial is given by

$$|f(x) - P(x)| \leq \frac{1}{n} \left(\frac{b-a}{n-1} \right)^n \max_{x \in [a,b]} |f^{(n)}(x)|$$

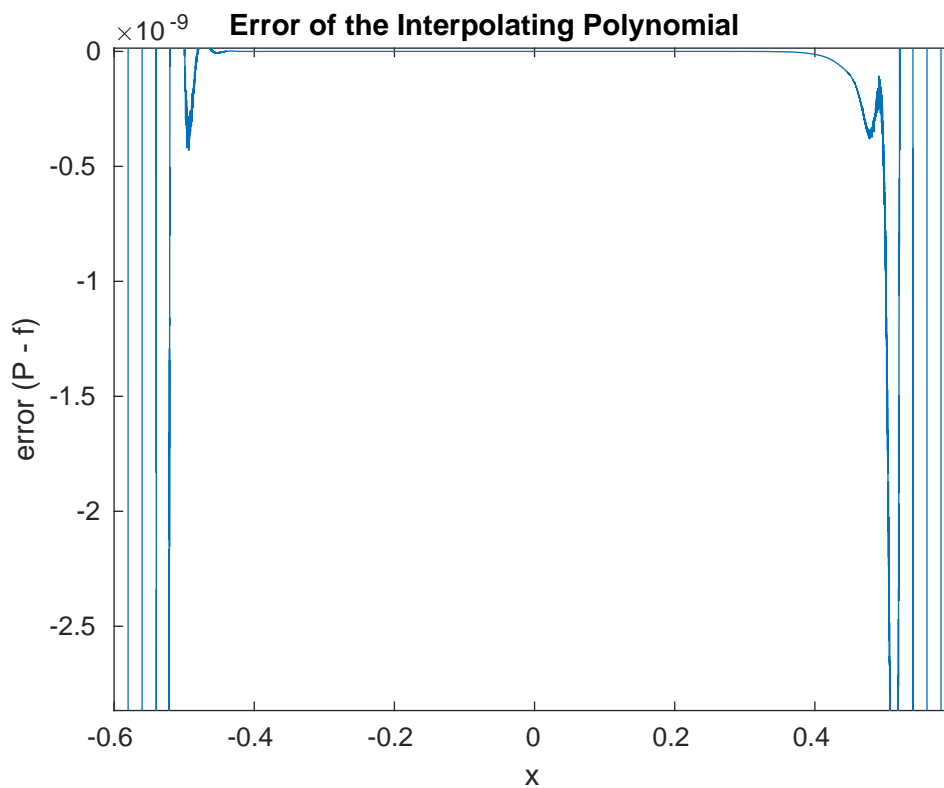
Let $f(x) = (1 + x)^{-1}$ as before and $[a, b] = [-0.5, 0.5]$. Use Stirling's approximation for large m , $\frac{\sqrt{m!}}{m} \approx \frac{1}{e}$, to show that the right hand side of the above equation goes to $+\infty$ as $n \rightarrow \infty$.

Solution

a) The plot appears to well interpolate the function. The interpolating polynomial is in blue, our data points are scattered in x, y space with red circles.



To see how good the interpolation is, we can check the error.



The overall error is quite good, consistent with our plot of the interpolating polynomial. Our matlab code is:

```
f = @(x) sin(x);
x = linspace(-0.6, 0.6, 61);

poly = newton_builder(x,f);
f1 = figure();
fplot(poly, [x(1), x(end)])% x(1) <0
```

```

grid on
hold on
scatter(x, f(x))
title('Interpolating polynomial')
xlabel 'x'
ylabel 'f(x)'
hold off
polynomial = matlabFunction(poly);
error_P = @(x) polynomial(x) - f(x);

f2 = figure();
fplot(error_P, [x(1), x(end)] ) % x(1) < 0
hold on
title('Error of the Interpolating Polynomial')
xlabel 'x'
ylabel 'error (P - f)'
hold off

function lastval = last_divdif(Xpts, f)
    coef_array = divdif(Xpts, f(Xpts));
    lastval = coef_array(1,1:end);
end

function coef_array = divdif(Xpts,Ypts)
    % Xpts and Ypts are data vectors of the same length
    % Xpts = [x1, x2, x3, ... xN]
    % Ypts = [y1, y2, y3, ... yN]
    % coef_array is a table of intermediate divided difference
    % coefficients
    datalength = length(Xpts);
    coef_array = zeros(datalength);
    coef_array(:,1) = Ypts'; % Write the data values to the first column
    for col = 2:datalength
        for row = 1 : (datalength - col + 1)
            %and now our magic step
            coef_array(row, col) = (coef_array(row+1, col-1) - coef_array(
                row, col - 1) )/(Xpts(row + col -1) - Xpts(row));
        end
    end
end

function polynomial = newton_builder(xpts, f)
    data_length = length(xpts);
    % Find div-dif coefficients
    coef = last_divdif(xpts, f);

    % Find the basis polynomials
    basis = ones(1,data_length, 'sym'); % To store our basis polynomials
    syms t % Our symbolic variable
    for basis_index = 2:length(basis) % Loop over each basis
        for x_index = 1:basis_index-1 % Loop over the first basis_index

```

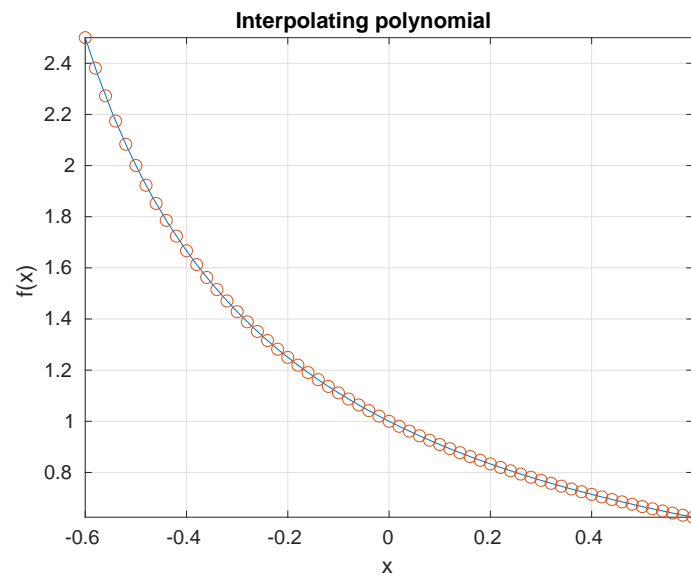
```

        data points we want
        basis(basis_index) = basis(basis_index) * (t - xpts(x_index));
    end
end

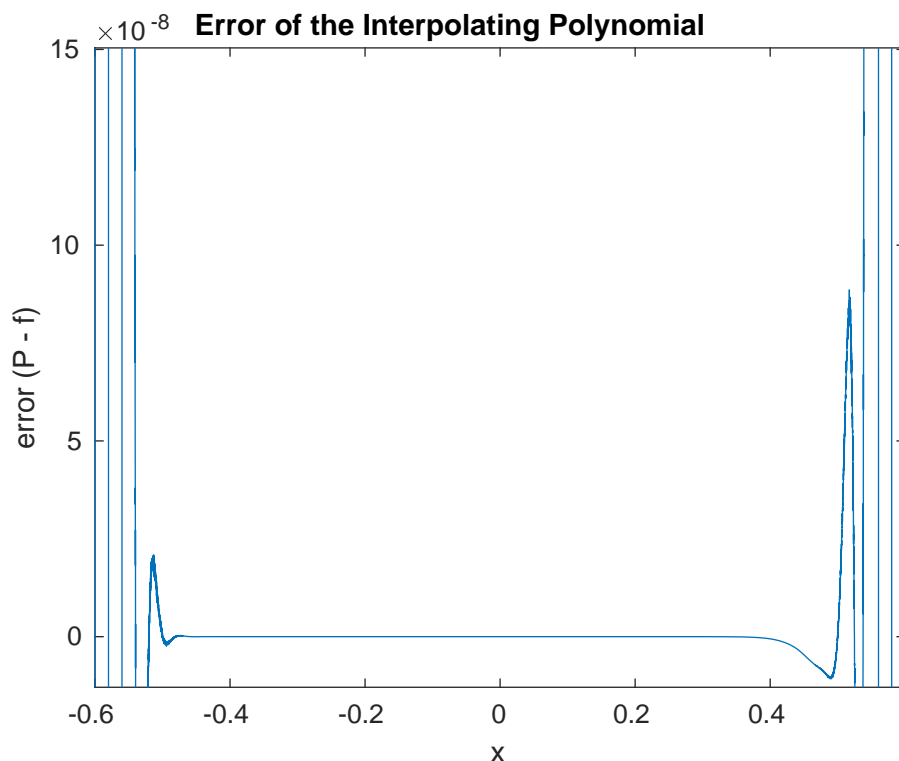
% Construct the interpolating polynomial
P = basis*coef';
polynomial = simplify(P);
end

```

b) We can reuse our algorithms developed above by changing our inline function f . The plot appears to well interpolate the function. The interpolating polynomial is in blue, our data points are scattered in x, y space with red circles.



To see how good the interpolation is, we can check the error.



The overall error is quite good, consistent with our plot of the interpolating polynomial.

With 61 interpolating points, we have sufficient data to well interpolate the function.

c) Let $f(x) = (x+1)^{-1}$ as before. Then with a bit of calculus,

$$f^{(n)}(x) = \frac{(-1)^n n!}{(1+x)^{n+1}}$$

The maximum is attained when $1+x$ is minimized (as it is in the denominator), we can write

$$\max_{x \in [a,b]} |f(x)| \leq \frac{n!}{(1+(-0.6))^{n+1}} = \frac{n!}{0.4^{n+1}}$$

Then

$$\begin{aligned} |f(x) - P(x)| &\leq \frac{1}{n} \left(\frac{0.6 - (-0.6)}{n-1} \right)^n \frac{n!}{0.4^n} = \frac{1}{n} \left(\frac{3}{n-1} \right)^n n! = \left(\frac{3}{n-1} \right)^n (n-1)! \\ &= \frac{3}{n-1} \left(\frac{3}{n-1} \right)^{n-1} (n-1)! = \frac{3}{n-1} \left(3 \frac{\sqrt[n-1]{(n-1)!}}{n-1} \right)^{n-1} \\ &\lesssim \frac{3}{n-1} \left(\frac{3}{e} \right)^{n-1} \end{aligned}$$

which goes to infinity as $n \rightarrow \infty$.

Problem 4.

Write a function in Matlab that does the following:

- Input: a function f and array of observations $x = (x_1, x_2, \dots, x_n)$,
- Output: the divided difference $f[x_1, x_2, \dots, x_n]$.

Solution

Note you can pass in a function (either an anonymous function or a local function defined in your script .m file) as an argument to your function. You can use additional helper function (local functions, in Matlab terminology)

```
function lastval = last_divdif(Xpts, f)
    % A function that computes the divided difference f[Xpts]
    coef_array = divdif(Xpts, f(Xpts));
    disp(coef_array)
    lastval = coef_array(1,end);
end

function coef_array = divdif(Xpts,Ypts)
    % Xpts and Ypts are data vectors of the same length
    % Xpts = [x1, x2, x3, ... xN]
    % Ypts = [y1, y2, y3, ... yN]
    % coef_array is a table of intermediate divided difference
    % coefficients
    datalength = length(Xpts);
    coef_array = zeros(datalength);
    coef_array(:,1) = Ypts'; % Write the data values to the first
    % column
    for col = 2:datalength
        for row = 1 : (datalength - col + 1)
            %and now our magic step
            coef_array(row, col) = (coef_array(row+1, col-1) -
                coef_array(row, col - 1) )/(Xpts(row + col -1)
                - Xpts(row));
        end
    end
end

end
```