



BRAC UNIVERSITY

CSE 330: Numerical Methods (LAB)

Lab 5: Interpolation

Polynomial

A polynomial $P(x) = a_0 + a_1x + a_2x^2 + \dots + a_{n-1}x^{n-1} + a_nx^n$ of degree n in MATLAB is stored as a row vector of length $(n+1)$. The components of the vector represent the coefficients of polynomial and are given in the descending order of the power of x ,

$$P = [a_n \ a_{n-1} \ a_{n-2} \ \dots \ a_1 \ a_0]$$

Some MATLAB Built-in Functions for Polynomial Operations

polyval(): $Y = \text{polyval}(p, X)$ returns the predicted value of a polynomial given its coefficients, p , at the values in X .

poly(): $p = \text{poly}(r)$, where r is a vector, returns a row vector p , whose elements are the coefficients of the polynomial whose roots are the elements of r .

roots(): $r = \text{roots}(c)$ returns a column vector whose elements are the roots of the polynomial c . c is a row vector containing the coefficients [in descending order] of the polynomial.

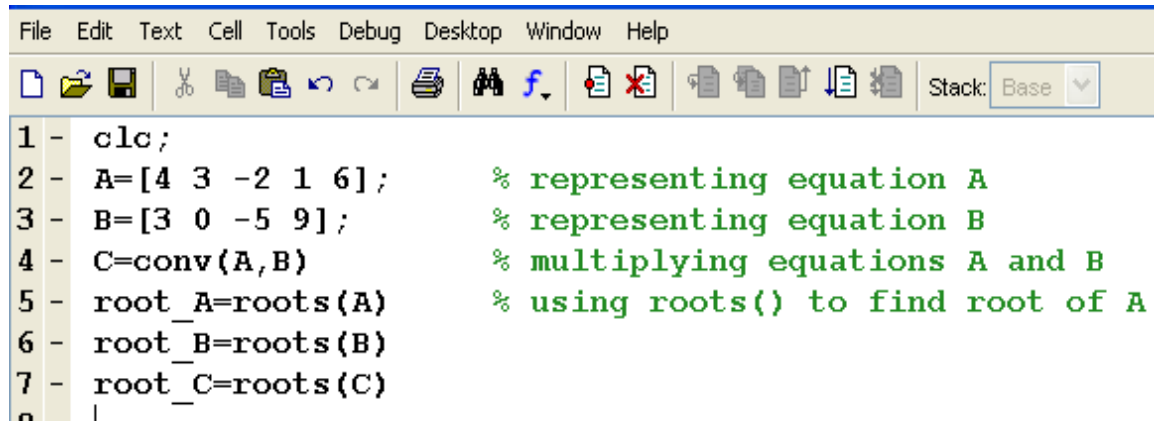
conv(): $w = \text{conv}(u, v)$ convolves vectors u and v . Algebraically, convolution is the same operation as multiplying the polynomials whose coefficients [in descending order] are the elements of u and v .

Example 1: Construct the polynomial $C(x) = A(x) * B(x)$

Where, $A(x) = 4x^4 + 3x^3 - 2x^2 + x + 6$ and $B(x) = 3x^3 - 5x + 9$

Also find the roots of $A(x)$, $B(x)$ and $C(x)$

Solution:



```
File Edit Text Cell Tools Debug Desktop Window Help
[Icons]
1 - clc;
2 - A=[4 3 -2 1 6]; % representing equation A
3 - B=[3 0 -5 9]; % representing equation B
4 - C=conv(A,B) % multiplying equations A and B
5 - root_A=roots(A) % using roots() to find root of A
6 - root_B=roots(B)
7 - root_C=roots(C)
8 -
```

Figure 1: Use of conv and roots function

Lab Task 1: For a 5th degree polynomial three roots are given as 5-2i, 2+3i and 5. Find the coefficients of the polynomial. After you find the polynomial predict the value of polynomial at x=2. Ans: @x=2, y = -351

Interpolation

Frequently we need to estimate intermediate values between precise data points. The most common method used for this purpose is polynomial interpolation. Interpolation is a method of constructing new data points from a discrete set of known data points. The general formula for an nth order polynomial can be written as

$$P(x) = a_0 + a_1x + a_2x^2 + \dots + a_{n-1}x^{n-1} + a_nx^n$$

For (n+1) data points there is one and only one polynomial of order n that passes through all the points. Polynomial interpolation consists of determining the unique nth order polynomial that fits (n+1) data points. This polynomial then provides a formula to compute intermediate values.

In engineering and science one often has a number of data points, as obtained by sampling or some experiment, and tries to construct a function which closely fits those data points. This is called curve fitting. Interpolation is a specific case of curve fitting, in which the function must go exactly through the data points.

MATLAB Built-in Functions for Interpolation

interp1():

yi = interp1(X,Y,xi) interpolates to find yi, the values of the underlying function Y at the points in the vector or array xi. X must be a vector. (xi,yi) are generated based on the relation between X and Y

yi = interp1(X,Y,xi,method) interpolates using alternative methods:

'nearest' Nearest neighbor interpolation

'linear' Linear interpolation (default)

'spline' Cubic spline interpolation

Lab task 2: The table shows density of Nitrogen gas at various temperatures. Estimate the density at 220, 279 and 370 Kelvin. Use *interp1()* function.

Temperature (K)	200	250	300	350	400	450
Density (kg/m ³)	1.708	1.367	1.139	0.967	0.854	0.759

Lagrange Interpolating Polynomial

Suppose we formulate a linear interpolating polynomial as the weighted average of the two values that are connecting by a straight line:

$$f(x) = L_1 f(x_1) + L_2 f(x_2) \dots\dots\dots (1)$$

where L 's are the weighting coefficients. It is logical that the first weighting coefficient is the straight line that is equal to 1 at x_1 and 0 at x_2

$$L_1 = \frac{x - x_2}{x_1 - x_2}$$

Similarly, the second co-efficient is the straight line that is equal to 1 at x_2 and 0 at x_1

$$L_2 = \frac{x - x_1}{x_2 - x_1}$$

Substituting L_1 and L_2 in (1)

$$f_1(x) = \frac{x - x_2}{x_1 - x_2} f(x_1) + \frac{x - x_1}{x_2 - x_1} f(x_2) \dots\dots\dots (2)$$

Where the nomenclature $f_1(x)$ designates that this is a first-order polynomial. Equation (2) is referred to as the linear Lagrange interpolation polynomial.

The same strategy can be employed to fit a parabola through three points. For this case three parabolas would be used with each one passing through one of the points and equaling zero at other two. Their sum would then represent the unique parabola that connects the three points. Such a second order Lagrange interpolating polynomial can be written as

$$f_2(x) = \frac{(x - x_2)(x - x_3)}{(x_1 - x_2)(x_1 - x_3)} f(x_1) + \frac{(x - x_1)(x - x_3)}{(x_2 - x_1)(x_2 - x_3)} f(x_2) + \frac{(x - x_1)(x - x_2)}{(x_3 - x_1)(x_3 - x_2)} f(x_3) \dots [3]$$

Both the first and second order versions as well as higher order Lagrange polynomials can be represented concisely as,

$$f_{n-1}(x) = \sum_{i=1}^n L_i(x) f(x_i)$$

$$\text{where } , L_i(x) = \prod_{\substack{j=1 \\ j \neq i}}^n \frac{x - x_j}{x_i - x_j}$$

here, n = number of data points and \prod designates the “product of”

Example 2: Write a function to perform Lagrange interpolation. The function will take three arrays as input. The first two arrays will consist the values of independent variable and their corresponding dependent variable values. The third array will contain the independent variable values at which you wish to interpolate.

```

1  function yy = Lagrange_int(x,y,xx)
2  % This function uses an (n-1)order Lagrange
3  %interpolating polynomial based on n data points
4  %to determine a value of the dependent yy at
5  % a given value of the independent variable xx.
6  %x=independent variable
7  %y=dependent variable
8  %xx=values of given independent variable
9  %at which the interpolation is calculated
10 %yy = interpolated values
11 n=length(x);
12 if length(y)~=n , error('x and y must have same lengths');
13 end
14 for p=1:length(xx)
15     s=0;
16     for i=1:n
17         product=y(i);
18         for j=1:n
19             if i~=j
20                 product=product*(xx(p)-x(j))/(x(i)-x(j));
21             end
22         end
23         s=s+product;
24     end
25     yy(p)=s;
26 end

```

Figure 2: Matlab function for Lagrange interpolation

Newton Polynomial

General form of Newton polynomial is given by,

$$f(x) = b_1 + b_2(x - x_1) + b_3(x - x_1)(x - x_2) + \dots + b_n(x - x_1)(x - x_2)\dots(x - x_{n-1}) \text{ ----- (1)}$$

Data points can be used to evaluate the coefficients $b_1, b_2, b_3, \dots, b_n$. For an $(n-1)^{\text{th}}$ order polynomial n data points are required: $(x_1, f(x_1))$, $(x_2, f(x_2))$, $(x_3, f(x_3))$, $\dots, (x_n, f(x_n))$

From equation (1) for $x = x_1$,

$$b_1 = f(x_1)$$

for $x = x_2$,

$$f(x_2) = b_1 + b_2(x_2 - x_1)$$

$$\Rightarrow b_2 = \frac{f(x_2) - f(x_1)}{x_2 - x_1} = f[x_2, x_1]$$

Similarly,

$$b_3 = f[x_3, x_2, x_1]$$

.

$$b_n = f[x_n, x_{n-1}, \dots, x_2, x_1]$$

where the braced function $[]$ evaluations are finite divided differences. For example the first finite divided difference is represented generally as,

$$f[x_i, x_j] = \frac{f(x_i) - f(x_j)}{x_i - x_j}$$

The second finite divided difference, which represents the difference of two first divided differences, is expressed as

$$f[x_i, x_j, x_k] = \frac{f[x_i, x_j] - f[x_j, x_k]}{x_i - x_k}$$

Similarly, the n th finite divided difference is,

$$f[x_n, x_{n-1}, x_{n-2}, \dots, x_2, x_1] = \frac{f[x_n, x_{n-1}, x_{n-2}, \dots, x_2] - f[x_{n-1}, x_{n-2}, \dots, x_1]}{x_n - x_1}$$

X_i	$f(x_i)$	First divided difference	Second divided difference	Third divided difference
X_1	$f(x_1)$	$f[x_2, x_1]$	$f[x_3, x_2, x_1]$	$f[x_4, x_3, x_2, x_1]$
X_2	$f(x_2)$	$f[x_3, x_2]$	$f[x_4, x_3, x_2]$	
X_3	$f(x_3)$	$f[x_4, x_3]$		
X_4	$f(x_4)$			

Figure 3: Graphical depiction of the recursive nature of the finite divided differences.

Example 3: Write a function to perform Newton interpolation.

```
1 function yy = Newton_int(x,y,xx)
2 %This function uses an (n-1)order Newton
3 % interpolating polynomial based on n data points
4 % to determine a value of the dependent variable yy
5 % at a given value of the independent variable xx.
6 % x = independent variable
7 % y=dependent variable
8 % xx= values of independent variable at which
9 % the interpolation is calculated
10 % yy= interpolated values
11 n=length(x);
12 if length(y)~=n, error('x and y must be same length')
13 end
14 b=zeros(n,n);
15 b(:,1)=y';
16 for p=1:length(xx) % This loop selects each element of xx in turns
17     for j=2:n
18         for i=1:n - j + 1
19             b(i,j)=(b(i+1,j-1)-b(i,j-1))/(x(i+j-1)-x(i));
20         end
21     end
22     xt=1;
23     yy(p)=b(1,1);
24     for j=1:n-1
25         xt=xt*(xx(p)-x(j));
26         yy(p)=yy(p)+b(1,j+1)*xt;
27     end
28 end
```

Figure 4: Matlab function for Newton interpolation

Home work:

[1] Suppose $X = [0, 20, 40]$ and $Y = [7, 0.8, 0.212]$;

First, manually calculate [using equation [3] of page 3] and show that the derived second degree Lagrange polynomial is given by $f(x) = 0.007x^2 - 0.4503x + 7$. [In your report show detail calculations]

Now write an M file to compute the coefficients of the second degree Lagrange polynomial. So, your M file should return $C = [0.007 \ -0.4503 \ 7]$.

Submit your derivation and M file print out including the displayed solution in the command window

Hints: You can take the aid of figure 2, but note that here you only need to find the coefficients.

Algorithm For solution:

- Initialize Matrix $L = \text{zeros}(\text{length}(X), \text{length}(X))$ for keeping the coefficients.
- Do for $i=1: \text{length}(x)$
- initialize $v=1$
- Do for $k=1: \text{length}(x)$
- If $i \sim k$ then $v = \text{multiply } v \text{ with } (x - x_k)/(x_i - x_k)$
For multiplication you may need the function `poly()` and `conv()`. For example if you wish to perform $(x-3)*(x-4)$, you can write `conv(poly(3), poly(4))]`
- End if
- End Do for k
- Assign v to the i^{th} row of L . [$L(i,:) = v$]
- End Do for i
- Now its time to add the all “like terms” i.e. add all the x^2 terms together and similarly add the x terms and constant terms together respectively. This can be easily accomplished by multiplying the L matrix with the Y vector. The order of multiplication is important for perfect result.

[2]. Use `interp1()` function to find the value of $f(x)$ at $x=3.5$.

x	0	1	2	3	4	5
f(x)	0	0.5	0.8	0.90	0.941	0.962

References:

[1]. Steven C Chapara, “*Applied Numerical Method with MATLAB*”, Second edition Chapter 15

[2]. John H. Mathews, “*Numerical Method Using MATLAB*”, Second edition Chapter 4