



The Interpolating Polynomial

Math 45 — Linear Algebra

David Arnold

David-Arnold@Eureka.redwoods.cc.ca.us

Abstract

A polynomial that passes through a given set of data points is called an *interpolating polynomial*. In this exercise you will use Matlab to find the interpolating polynomial (if one exists) for a variety of different data sets. The Vandermonde matrix is introduced as an efficient means of entering the augmented matrix. *Prerequisites: Solving linear systems with Matlab's rref command.*

next page

close

exit

Table of Contents

Polynomials in Matlab

Evaluating a Polynomial in Matlab

Plotting in Matlab

Plotting Polynomials in Matlab

The Interpolating Polynomial

The Vandermonde Matrix

Homework

The
Interpolating
Polynomial

title page

contents

previous page

next page

back

print doc

close

exit

Polynomials in Matlab

The equation

$$p(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0 \quad (1)$$

is called a *polynomial* in x . The terms of this polynomial are arranged in *descending* powers of x while the terms of the polynomial

$$p(x) = a_0 + a_1 x + \dots + a_{n-1} x^{n-1} + a_n x^n \quad (2)$$

are arranged in *ascending* powers of x . The a_i 's are called the *coefficients* of the polynomial and are usually real or complex numbers. The *degree* of the polynomial in (2) is n , the highest available power of x .

For example, $p(x) = x^6 - 2x^4 - x^3 + 2x^2 + 5x - 8$ is a polynomial that is arranged in descending powers of x . The coefficients of the polynomial are 1, 0, -2, -1, 2, 5, and -8. In this case, there is an understood term of $0x^5$. However, Matlab doesn't "understand" that a term is missing unless you explicitly tell it so. The degree of the polynomial is 6.

A polynomial in Matlab is represented by a vector of its coefficients. Matlab assumes that you have arranged the polynomial in descending powers of x . For example, the vector $p=[5 \ 2 \ -3 \ 1]$ represents the polynomial $p(x) = 5x^3 + 2x^2 - 3x + 1$, while the vector $q=[\ 2 \ 0 \ 0 \ -4 \ -3]$ represents the polynomial $q(x) = 2x^2 + 0x^3 + 0x^2 - 4x - 3$, or, more simply, $q(x) = 2x^2 - 4x - 3$. Note that you must use zeros to denote the missing terms of the polynomial.

Some polynomials and their Matlab representations are displayed in **Table 1**. Note well the use of zeros as placeholders.

Evaluating a Polynomial in Matlab

Matlab has a number of useful routines for working with polynomials.¹ One particularly useful

The
Interpolating
Polynomial

title page

contents

previous page

next page

back

print doc

close

exit

¹Use Matlab's help facility to examine the help files for roots and poly. Although these routines will not be used in this

$p(x) = x^3 - x^2 - 1$	$p=[1 \ -1 \ 0 \ -1]$
$q(x) = x^4 - x^2 - x$	$q=[1 \ 0 \ -1 \ -1 \ 0]$
$r(x) = x^4 - 2x^2$	$r=[1 \ 0 \ -2 \ 0 \ 0]$

Table 1 Matlab represents polynomials with vectors.

The Interpolating Polynomial

[title page](#)

[contents](#)

[previous page](#)

[next page](#)

[back](#)

[print doc](#)

[close](#)

[exit](#)

routine is `polyval`. Enter the following command at the Matlab prompt to generate the resulting help file.

```
>> help polyval
```

POLYVAL Polynomial evaluation.

$Y = \text{POLYVAL}(P,X)$, when P is a vector of length $N+1$ whose elements are the coefficients of a polynomial, is the value of the polynomial evaluated at X .

$$Y = P(1)*X^N + P(2)*X^{(N-1)} + \dots + P(N)*X + P(N+1)$$

If X is a matrix or vector, the polynomial is evaluated at all points in X . See also `POLYVALM` for evaluation in a matrix sense.

If p is a polynomial and x is a number, then `polyval(p,x)` evaluates the polynomial at the number x . For example, if $p(x) = x^3 + 2x - 8$, then $p(-2) = -20$.

```
>> p=[1 0 2 -8];
>> x=-2;
>> y=polyval(p,x)
```

```
y =  
-20
```

The Interpolating Polynomial

Matlab can evaluate a polynomial at each element of a vector or matrix.² For example, use your calculator to verify these calculations: $p(x) = x^3 + 2x - 8$, then $p(-2) = -20$, $p(-1) = -11$, $p(0) = -8$, $p(1) = -5$, and $p(2) = 4$. It is not difficult to evaluate the polynomial at each x -value; it's just tedious. Matlab saves you some effort by evaluating the polynomial at each value of x with one command.

```
>> p=[1 0 2 -8];  
>> x=[-2 -1 0 1 2];           % or try x=-2:2;  
>> y=polyval(p,x)  
y =  
-20   -11   -8   -5    4
```

[title page](#)

[contents](#)

[previous page](#)

[next page](#)

[back](#)

[print doc](#)

[close](#)

[exit](#)

The % is Matlab's comment delimiter. Anything appearing after the % sign is ignored by Matlab when executing a command. In this case, we provide this comment as an alternative command. You should try the commented command in place of the original and see what happens.

Plotting in Matlab

Plotting is done with Matlab's `plot` command. Enter the following command at the Matlab prompt and read the resulting help file.

```
>> help plot
```

The help file is rather extensive so we will not reproduce it here. For purposes of this activity, if x and y are vectors of equal length, then the command `plot(x,y)` plots the elements of vector

²You will only evaluate a polynomial at each point of a vector in this exercise.

The Interpolating Polynomial

[title page](#)

[contents](#)

[previous page](#)

[next page](#)

[back](#)

[print doc](#)

[close](#)

[exit](#)

y versus those of vector x . The following commands should produce a graph similar to that shown in **Figure 1**.

```
>> x=[-1 0 2 3 5];  
>> y=[2 -2 3 0 4];  
>> plot(x,y)
```

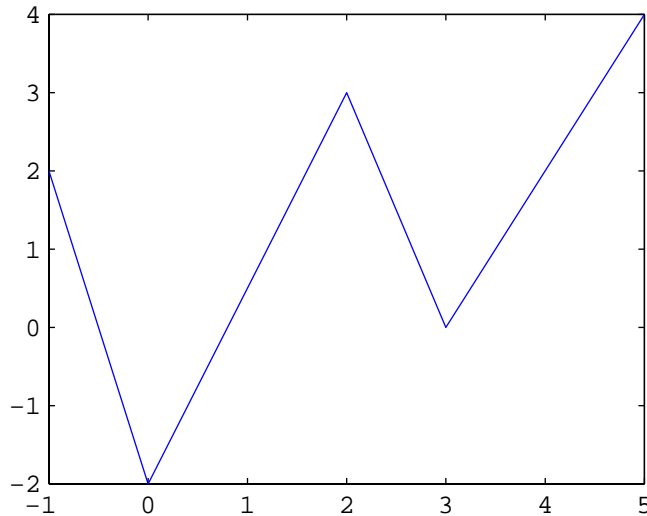


Figure 1 The data points are connected by line segments.

The idea behind Matlab's plotting routine is simple. When you execute the command `plot(x,y)`, Matlab first plots all possible points (x,y) , taking an x -coordinate from the vector x and a y -coordinate from the vector y . In this case, Matlab plots the points $(-1, 2)$, $(0, -2)$, $(2, 3)$, $(3, 0)$, and $(5, 4)$.

By default, Matlab connects consecutive data points with line segments. Of course, Matlab's plot command offers a number of alternatives to override this default style. You might want to

The Interpolating Polynomial

[title page](#)

[contents](#)

[previous page](#)

[next page](#)

[back](#)

[print doc](#)

[close](#)

[exit](#)

type `help plot` again, only this time pay particular attention to the wide variety of linestyle, markers, and color combinations that are available for your plot.

There are a number of predefined markers that you can use for your data points in a Matlab plot. Markers are especially valuable when you want to plot your data as discrete points. For example, the command

```
>> plot(x,y,'ro')      % or try plot(x,y,'g*')
```

will produce an image similar to that shown in **Figure 2**.

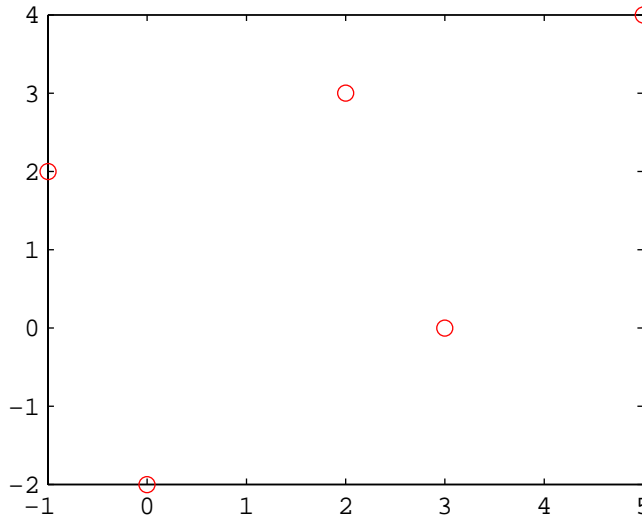


Figure 2 Using markers to display discrete data points.

Plotting Polynomials in Matlab

If Matlab's `plot` command connects consecutive points with line segments, it seems at first hopeless that one can plot smooth, continuous curves. However, one can provide a fair approx-

imation of the graph of a curve by sampling the curve at a lot of points. If enough points are used, the curve drawn with the `plot` command will take on a smooth appearance.

To produce the smooth shape of a polynomial in Matlab, you first have to generate a sufficient number of data points that satisfy the equation of the polynomial. The following commands produce the graph of the polynomial $p(x) = x^3 - 6x^2 + 3x + 10$ over the interval $[-2, 6]$, as shown in **Figure 3**.

```
>> p=[1 -6 3 10];  
>> x=linspace(-2,6); %You need a lot of points to draw a smooth curve.  
>> y=polyval(p,x);  
>> plot(x,y)
```

Matlab's `linspace` command is an easy way to generate a vector of equally spaced points. By default, the command `linspace(a,b)` generates 100 equally spaced points, starting at a and ending at b . You can generate more points with the syntax is `linspace(a,b,N)`. For example, the command `linspace(0,1,1000)` generates 1000 equally spaced points, starting at 0 and ending at 1000.

The Interpolating Polynomial

Given a set of data points, scientists and mathematicians often need to find a polynomial of specified degree whose graph passes through each of the given data points. Such a polynomial is called an *interpolating polynomial*.³

In this exercise, we will find a third degree polynomial that passes through the points $(-3, 0)$, $(-2, -3)$, $(0, 3)$, and $(2, -15)$. A third degree polynomial has the form $p(x) = ax^3 + bx^2 + cx + d$.

³ In later activities you will fit polynomials to data sets in the *least squares sense*, where the objective is to find a polynomial that passes near each data point but not necessarily through them.

The Interpolating Polynomial

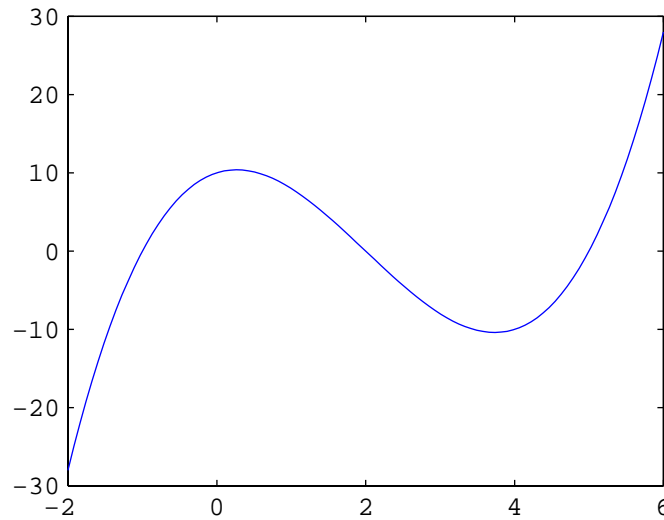


Figure 3 The graph of a polynomial is a smooth curve.

[title page](#)

[contents](#)

[previous page](#)

[next page](#)

[back](#)

[print doc](#)

[close](#)

[exit](#)

If you substitute each of the given points into the equation $p(x) = ax^3 + bx^2 + cx + d$ and you will create a system of four equations in four unknowns a , b , c , and d .

$$a(-3)^3 + b(-3)^2 + c(-3) + d = 0$$

$$a(-2)^3 + b(-2)^2 + c(-2) + d = -3$$

$$a(0)^3 + b(0)^2 + c(0) + d = 3$$

$$a(2)^3 + b(2)^2 + c(2) + d = -15$$

(3)

As you recall, the first step in solving **system 3** requires that you set up an augmented matrix for the system.

$$\begin{pmatrix} (-3)^3 & (-3)^2 & -3 & 1 & 0 \\ (-2)^3 & (-2)^2 & -2 & 1 & -3 \\ (0)^3 & (0)^2 & 0 & 1 & 3 \\ (2)^3 & (2)^2 & 2 & 1 & -15 \end{pmatrix}$$

Simplify the augmented matrix.

$$\begin{pmatrix} -27 & 9 & -3 & 1 & 0 \\ -8 & 4 & -2 & 1 & -3 \\ 0 & 0 & 0 & 1 & 3 \\ 8 & 4 & 2 & 1 & -15 \end{pmatrix}$$

Next, enter the augmented matrix and use Matlab's `rref` command to place the augmented matrix in reduced row echelon form.

```
>> M=[-27 9 -3 1 0;-8 4 -2 1 -3;0 0 0 1 3;8 4 2 1 -15]
```

```
M =
   -27     9    -3     1     0
    -8     4    -2     1    -3
     0     0     0     1     3
     8     4     2     1    -15
```

```
>> R=rref(M)
```

```
R =
     1     0     0     0    -1
     0     1     0     0    -3
     0     0     1     0     1
     0     0     0     1     3
```

It is clear from this last result that the solution of **system 3** is $a = -1$, $b = -3$, $c = 1$, and $d = 3$. If you substitute these values in the general third degree polynomial, $p(x) = ax^3 + bx^2 + cx + d$, then the interpolating polynomial is

The Interpolating Polynomial

[title page](#)

[contents](#)

[previous page](#)

[next page](#)

[back](#)

[print doc](#)

[close](#)

[exit](#)

$$p(x) = -x^3 - 3x^2 + x + 3. \quad (4)$$

The Interpolating Polynomial

[title page](#)

[contents](#)

[previous page](#)

[next page](#)

[back](#)

[print doc](#)

[close](#)

[exit](#)

It is essential that we check our solution. Remember, the interpolating polynomial must pass through each of the data points. If it doesn't, then we do not have the correct answer. So, first we will plot the original data as discrete points. Then we will plot the polynomial. If the polynomial passes through each data point, then we are confident that we have the correct answer.

Enter the x - and y -values of the data points $(-3, 0)$, $(-2, -3)$, $(0, 3)$, and $(2, -15)$ in the vectors x and y .

```
>> x=[-3 -2 0 2];  
>> y=[0 -3 3 -15];
```

The minimum and maximum x -values of the set of data points are -3 and 2 , respectively. Let's sketch the polynomial $p(x) = -x^3 - 3x^2 + x + 3$ over the interval $[-4, 3]$. This will ensure that each of the given data points is visible in our final plot. The following commands should produce a graph similar to that in **Figure 4**.

```
>> p=[-1 -3 1 3];  
>> xp=linspace(-4,3); % 100 equally spaced points from -4 to 3  
>> yp=polyval(p,xp);  
>> plot(x,y,'ro',xp,yp,'-')
```

Matlab's `plot` command is an extremely flexible tool. In this case, note that we are creating two plots with one command. In general, the command `plot(x1,y1,s1,x2,y2,s2,...,xN,yN,sN)` will draw N plots: the plot of y_1 versus x_1 , the plot of y_2 versus x_2 , and finally, the plot of y_N versus x_N . Line styles, markers, and colors are defined for each plot in the strings⁴ s_1, s_2, \dots, s_N .

⁴In Matlab, 'ro' is an example of a string. Note that strings are delimited by single apostrophes.

The Interpolating Polynomial

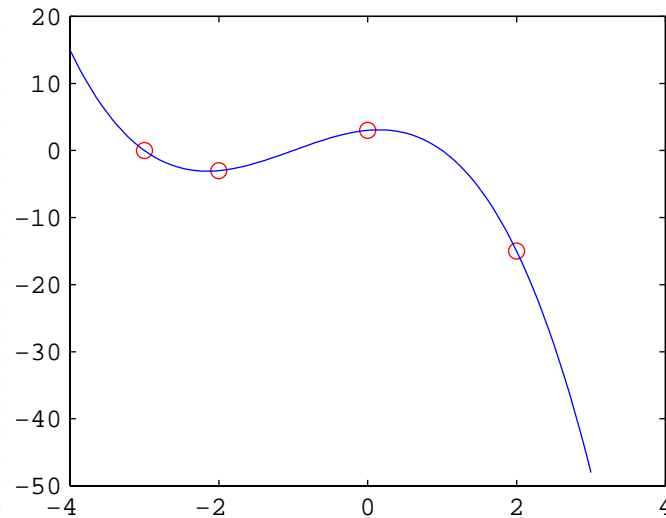


Figure 4 The interpolating polynomial must pass through each data point.

[title page](#)

[contents](#)

[previous page](#)

[next page](#)

[back](#)

[print doc](#)

[close](#)

[exit](#)

The Vandermonde Matrix

Suppose that you wish to find a fourth degree interpolating polynomial that passes through the points (x_1, y_1) , (x_2, y_2) , (x_3, y_3) , (x_4, y_4) , and (x_5, y_5) . If you substitute each of these points in the polynomial $y = ax^4 + bx^3 + cx^2 + dx + e$, you will arrive at the following system of linear equations.

$$\begin{aligned} ax_1^4 + bx_1^3 + cx_1^2 + dx_1 + e &= y_1 \\ ax_2^4 + bx_2^3 + cx_2^2 + dx_2 + e &= y_2 \\ ax_3^4 + bx_3^3 + cx_3^2 + dx_3 + e &= y_3 \\ ax_4^4 + bx_4^3 + cx_4^2 + dx_4 + e &= y_4 \\ ax_5^4 + bx_5^3 + cx_5^2 + dx_5 + e &= y_5 \end{aligned} \tag{5}$$

The augmented matrix for **system 5** is

$$\begin{pmatrix} x_1^4 & x_1^3 & x_1^2 & x_1 & 1 & y_1 \\ x_2^4 & x_2^3 & x_2^2 & x_2 & 1 & y_2 \\ x_3^4 & x_3^3 & x_3^2 & x_3 & 1 & y_3 \\ x_4^4 & x_4^3 & x_4^2 & x_4 & 1 & y_4 \\ x_5^4 & x_5^3 & x_5^2 & x_5 & 1 & y_5 \end{pmatrix}. \quad (6)$$

If

$$\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} \quad \text{and} \quad \mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{pmatrix},$$

note that the columns of the augmented matrix (6) are $x.^4$, $x.^3$, $x.^2$, x , $\text{ones}(\text{size}(x))$, and y . The coefficient matrix of **system 5**,

$$\begin{pmatrix} x_2^4 & x_2^3 & x_2^2 & x_2 & 1 \\ x_3^4 & x_3^3 & x_3^2 & x_3 & 1 \\ x_4^4 & x_4^3 & x_4^2 & x_4 & 1 \\ x_5^4 & x_5^3 & x_5^2 & x_5 & 1 \end{pmatrix}, \quad (7)$$

is called a *Vandermonde matrix*. The Vandermonde matrix is important in a number of applications, but it is particularly useful when computing the interpolating polynomial.

For our next example, we will find a fourth degree interpolating polynomial that passes through the points $(-2, 26)$, $(-1, -2)$, $(1, -4)$, $(2, -2)$, and $(4, 128)$. First, substitute each data point into the equation for a general fourth degree polynomial, $y = ax^4 + bx^3 + cx^2 + dx + e$.

The
Interpolating
Polynomial

title page

contents

previous page

next page

back

print doc

close

exit

$$a(-2)^4 + b(-2)^3 + c(-2)^2 + d(-2) + e = 26$$

$$a(-1)^4 + b(-1)^3 + c(-1)^2 + d(-1) + e = -2$$

$$a(1)^4 + b(1)^3 + c(1)^2 + d(1) + e = -4$$

$$a(2)^4 + b(2)^3 + c(2)^2 + d(2) + e = -2$$

$$a(4)^4 + b(4)^3 + c(4)^2 + d(4) + e = 128$$

(8)

The augmented matrix for **system 8** is

$$\left(\begin{array}{cccccc} (-2)^4 & (-2)^3 & (-2)^2 & -2 & 1 & 26 \\ (-1)^4 & (-1)^3 & (-1)^2 & -1 & 1 & -2 \\ 1^4 & 1^3 & 1^2 & 1 & 1 & -4 \\ 2^4 & 2^3 & 2^2 & 2 & 1 & -2 \\ 4^4 & 4^3 & 4^2 & 4 & 1 & 128 \end{array} \right).$$

(9)

Note that the coefficient matrix of **system 5** is

$$\left(\begin{array}{ccccc} (-2)^4 & (-2)^3 & (-2)^2 & -2 & 1 \\ (-1)^4 & (-1)^3 & (-1)^2 & -1 & 1 \\ 1^4 & 1^3 & 1^2 & 1 & 1 \\ 2^4 & 2^3 & 2^2 & 2 & 1 \\ 4^4 & 4^3 & 4^2 & 4 & 1 \end{array} \right).$$

(10)

Note that the coefficient matrix **(10)** is a Vandermonde matrix, where

$$\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} = \begin{pmatrix} -2 \\ -1 \\ 1 \\ 2 \\ 4 \end{pmatrix}.$$

Matlab's matrix building capability makes it especially easy to create the augmented matrix **(9)**.

First, enter the data points in two column vectors.

The
Interpolating
Polynomial

title page

contents

previous page

next page

back

print doc

close

exit

The Interpolating Polynomial

```
>> x=[-2 -1 1 2 4]'  
x =  
    -2  
    -1  
     1  
     2  
     4  
>> y=[26 -2 -4 -2 128]'  
y =  
    26  
    -2  
    -4  
    -2  
   128
```

[title page](#)

[contents](#)

[previous page](#)

[next page](#)

[back](#)

[print doc](#)

[close](#)

[exit](#)

Note that the first column of the augmented matrix (9) is crafted by raising every element of the vector x to the fourth power. The second column is built by raising each element of the vector x to the third power, and so on. The last column of the augmented matrix (9) is a vector of ones that has the same size as the vector x . Matlab's elementwise operators make it extremely easy to build the augmented matrix.

```
>> M=[x.^4,x.^3,x.^2,x,ones(size(x)),y]  
M =  
    16    -8     4    -2     1    26  
     1    -1     1    -1     1    -2  
     1     1     1     1     1    -4  
    16     8     4     2     1    -2  
   256    64    16     4     1   128
```

Next, place the augmented matrix in reduced row echelon form.

```
>> R=rref(M)
```

```
R =
```

```
  1   0   0   0   0   1
  0   1   0   0   0  -2
  0   0   1   0   0   0
  0   0   0   1   0   1
  0   0   0   0   1  -4
```

Hence, $a = 1$, $b = -2$, $c = 0$, $d = 1$, and $e = -4$. Substitute these values in the general form $y = ax^4 + bx^3 + cx^2 + dx + e$ and the interpolating polynomial is $y = 1x^4 - 2x^3 + 0x^2 + 1x - 4$, or $p(x) = x^4 - 2x^3 + x - 4$. Enter this polynomial at the Matlab prompt as follows:

```
>> p=[1 -2 0 1 -4]'
```

```
p =
```

```
  1
 -2
  0
  1
 -4
```

Again, if the interpolating polynomial does not pass through each of the original data points, then our answer is wrong. So, let's plot the data points, then determine whether our solution passes through each data point.

Recall that the original data are still stored in the vectors x and y . We need only create a set of data points satisfying our polynomial. The minimum x -value in our data set is -2 and the maximum x -value is 4 . Let's plot the polynomial on the interval $[-3, 5]$. This will insure that each of our data points is visible in the final plot.

You can use column vectors with the `plot` command as easily as row vectors. The following commands produce an image similar to that in **Figure 5**. Because the polynomial passes through each data point, it is highly likely that we have found the correct interpolating polynomial.

The Interpolating Polynomial

[title page](#)

[contents](#)

[previous page](#)

[next page](#)

[back](#)

[print doc](#)

[close](#)

[exit](#)

```
>> xp=(linspace(-3,5))';           % or try xp=(-3:.1:5)';  
>> yp=polyval(p,xp);  
>> plot(x,y,'ro',xp,yp,'-')
```

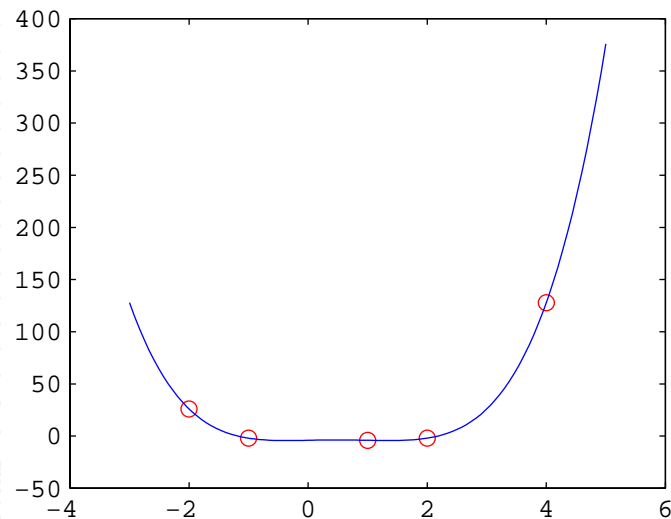


Figure 5 The interpolating polynomial must pass through each data point.

Homework

In the past, students in linear algebra usually begin using the computer way too soon on this assignment. They start typing this and that at the Matlab prompt, and it isn't long before they are totally confused. Why does this happen? What can you do to avoid being frustrated?

First, write down the appropriate general form of the interpolating polynomial on notebook paper. Then, substitute each data point into the general form. Write down the system of equations generated by each data point on your notebook paper. Now you can go to the computer.

The Interpolating Polynomial

[title page](#)

[contents](#)

[previous page](#)

[next page](#)

[back](#)

[print doc](#)

[close](#)

[exit](#)

The Interpolating Polynomial

[title page](#)

[contents](#)

[previous page](#)

[next page](#)

[back](#)

[print doc](#)

[close](#)

[exit](#)

Enter the augmented matrix, place the result into reduced row echelon form, then return to your notebook paper and write down the reduced row echelon form of the augmented matrix. Write down the values of the unknown coefficients, then substitute them into your general form and write the equation of the interpolating polynomial on your notebook paper.

You are now ready to return to the computer. Enter the data points in the vectors x and y . Enter the interpolating polynomial in the vector p . Finally, follow the examples in the narrative to craft a plot of the interpolating polynomial that passes through each of the given data points. Good luck!

1. Find a tenth degree interpolating polynomial that passes through the points $(-5, -10)$, $(-4, -5)$, $(-3, 2)$, $(-2, -5)$, $(-1, 6)$, $(0, 8)$, $(1, 1)$, $(2, -9)$, $(3, 1)$, $(4, 2)$, and $(5, -1)$. Obtain a plot of the data points that includes the graph of the interpolating polynomial.

Note: This problem has caused searing headaches for linear algebra students at the College over the last few years. It turns out that the interpolating polynomial is extremely sensitive to small changes in some of its coefficients. That is, the interpolating polynomial can do some pretty wild things, including missing most of its data points, if the reader truncates the coefficients of the interpolating polynomial beyond a certain number of significant figures. One way to avoid this problem is to type

```
format long
```

at the Matlab prompt. This will provide about 14 significant figures. This bears both good news and bad news for the user. The good news is the interpolating polynomial will do what is expected if the reader uses all significant figures of the coefficients provided by Matlab. The bad news is that it takes forever to type these in at the Matlab prompt. It is also extremely difficult to type them correctly. In the past, more often than not, most students have made typing errors when entering these coefficients at the Matlab prompt.

One workaround is to create a *script file* with your Matlab commands. Script files are like batch files. You simply list the commands that you want executed, save the file with a

name, then enter the name of the file at the Matlab prompt. Every command in the file is executed in sequence.

For example, open the Matlab editor with the `edit` command, then enter this sequence of commands from the narrative.

```
x=[-2 -1 1 2 4]'  
y=[26 -2 -4 -2 128]'  
M=[x.^4,x.^3,x.^2,x,ones(size(x)),y]  
R=rref(M)  
p=[1 -2 0 1 -4]'  
xp=linspace(-3,5))';           % or try xp=(-3:.1:5)';  
yp=polyval(p,xp);  
plot(x,y,'ro',xp,yp,'-')
```

Save the file with the name `figure5.m`. Return to the Matlab prompt and enter the filename.

```
>> figure5
```

This should produce the image shown in **Figure 5**.

If you decide to use a script file on this first exercise, then you will find it much easier to eliminate typing mistakes. Just return to the editor, make your changes, save the file, then execute the filename at the Matlab prompt. This is the way that most Matlab gurus work. Script files are indeed a blessing.

There is a second workaround to the problem of significant digits. What Matlab outputs to the computer screen and what it stores in its internal memory are two entirely different things. Return to the default output precision by typing the following command at the Matlab prompt.

```
>> format
```

The Interpolating Polynomial

[title page](#)

[contents](#)

[previous page](#)

[next page](#)

[back](#)

[print doc](#)

[close](#)

[exit](#)

The Interpolating Polynomial

By default, Matlab displays floating point numbers with four decimal places. For example, if you enter $x=1/7$ at the prompt, Matlab responds with 0.1429. However, this is not what Matlab stores internally. Rather, Matlab stores the number in double precision. This strategy stores a lot more significant figures in memory than what is displayed on the screen.

In the case where the interpolating polynomial is unique, there is a simple way to build the polynomial without entering the coefficients individually. Simply strip off the last column of the augmented matrix after it has been placed in reduced row echelon form. If the reduced row echelon form of the augmented matrix

$$R = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & -2 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & -4 \end{pmatrix},$$

then the command

```
>> p=R(:,6)
```

will strip off the last column and store the result in the variable p. Of course, the last column contains the coefficients of the interpolating polynomial, so this is an extremely effective way to build the interpolating polynomial. Remember, even if only 4 decimal places are displayed, internally, Matlab keeps as many decimal places for each coefficient as possible. You can easily see this when you work this exercise if you enter the commands

```
>> format
>> p
>> format long
>> p
```

There is one final problem students have encountered in the past when working this exercise. Polynomials can grow so quickly (not exponentially, but still pretty fast) that the scale is

[title page](#)

[contents](#)

[previous page](#)

[next page](#)

[back](#)

[print doc](#)

[close](#)

[exit](#)

The Interpolating Polynomial

just too large to see the oscillating behavior of the interpolating polynomial as it wiggles through its data points. The `axis` command will help you construct a viewing window that highlights this local behavior of the interpolating polynomial. Type `help axis` at the Matlab prompt and read the resulting helpfile to learn how to use this important command. You can also use the zoom tools in the figure window's toolbar to highlight regions of interest.

If a system of linear equations has more equations than unknown variables, then the system is *overdetermined*. An overdetermined system may or may not be consistent.

2. Find a second degree polynomial (if one exists) that passes through the points $(-2, -7.6)$, $(-1, -7.8)$, $(1, -1.0)$, $(2, 6.0)$, and $(3, 15.4)$. If the interpolating polynomial exists, obtain a plot that contains both the data points and the interpolating polynomial. If the interpolating polynomial doesn't exist, obtain a plot of the data points and explain why there can be no second degree polynomial that passes through the points.
3. Find a second degree polynomial (if one exists) that passes through the points $(-3, 1)$, $(-1, 4)$, $(1, 2)$, and $(2, 6)$. If the interpolating polynomial exists, obtain a plot that contains both the data points and the interpolating polynomial. If the interpolating polynomial doesn't exist, obtain a plot of the data points and explain why there can be no second degree polynomial that passes through the points.

If a system of linear equations has fewer equations than unknown variables, then the system is *underdetermined*. An underdetermined system of equations always has an infinite number of solutions.

4. Find at least two third degree interpolating polynomials that pass through the points $(-3, -24)$, $(-1, 8)$, and $(2, -4)$. Obtain a plot that contains the data points and each of the interpolating polynomials.

[title page](#)

[contents](#)

[previous page](#)

[next page](#)

[back](#)

[print doc](#)

[close](#)

[exit](#)