Chung-Ang University

Linear Algebra Spring 2014

Computer Project #1

Assigned: March 18, 2014
Due Date: March 27, 2014

Read Carefully:

These projects are going to be a **major part of your final grade**, and all work must be done alone. No group efforts and no copying of the results from someone else is allowed. Infractions will result in an automatic score of zero.

You are to use MATLAB or Octave, and in your computer project write-up, it should be a formal report with plots generated by MATLAB or Octave, short listings or screen outputs of what you did, and details that describe everything that you did. Your *Reports* must be well done, as you would do in submitting a technical report to your boss at work. They should be well-organized, neat, and fully documented with all results explained. Explanations of what you did and interpretation of your result is expected. **Do not** submit more information than is necessary to illustrate or present your results. A large part of your grade will be based on **how well-organized and complete your report is**.

In this computer project, you will look at the problem of polynomial interpolation using ideas introduced in class, and finding these interpolating polynomials using MATLAB or Octave. This project begins with some examples of what the problem is, and provides a description of some useful tools and techniques in MATLAB . And then you are asked to some some specific polynomial interpolation problems. It should be pointed out this this process is very important in many areas of science in engineering, and can generally be classified as one of *fitting a curve to some data.* Once the "best curve" has been found, it can then be used as a model to predict values of data that have not yet been observed or recorded. For example, given ten daily prices of Samsung stock, fitting a tenth-order polynomial to this data, the polynomial may then be used to predict future values of Samsung stock.

Interpolating Polynomials

The equation

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

is an nth order polynomial in x. The a_is are called the *coefficients* of the polynomial. A polynomial in MATLAB is represented by a vector that contains the polynomial coefficients. MATLAB assumes that the polynomial is arranged in descending powers of x. For example, the vector `p=[5 2 -3 1]` represents the polynomial $p(x) = 5x^3+2x^2-3x+1$ whereas the vector `q=[ 2 0 0 -4 -3]` represents the polynomial $q(x) = 2x^4 - 4x - 3$. Note that zeros must be entered to denote any missing terms in the polynomial.

MATLAB has a number of useful routines for working with polynomials[1] One particularly useful routine is `polyval`. The help file in MATLAB for this function says

```
>> help polyval

   POLYVAL Polynomial evaluation.
```

[1] Use the `help` function in MATLAB to look at the to routines `roots` and `poly`. Although these routines will not be used here, they will be useful later.

```
    Y = 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)*XN + P(2)*X(N-1) + ... + 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.
```

Thus, if `p` is a vector representing a polynomial and if `x` is a number, then `polyval(p,x)` evaluates the polynomial at `x`. For example, if $p(x) = x^3 + 2x - 8$, then $p(-2) = -20$ as shown in the following sequence of MATLAB commands,

```
>> p=[1 0 2 -8];
>> x=-2;
>> y=polyval(p,x)
y =
  -20
```

MATLAB can also evaluate a polynomial for every value in a vector or matrix. For example, if $p(x) = x^3 + x^2 - 8$, then we may evaluate this polynomial at $x = -2, -1, 0, 1, 2, 5$ in MATLAB as follows,

```
>> p=[1 0 2 -8];
>> x=[-2 -1 0 1 2 3 4 5];             % or try x=-2:5;
>> y=polyval(p,x)
y =
  -20   -11   -8   -5   25   64   127
```

Plotting is done in MATLAB using the `plot` command. You may read the documentation on `plot` by typing the command

```
>> help plot
```

The help file is quite long, but some of the more important and useful information will be summarized here. First, if `x` and `y` are vectors of equal length, then the command `plot(x,y)` will plot the elements in the vector `y` versus those in the vector `x`. For example, the following commands will 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)
```

When you execute the command `plot(x,y)`, MATLAB first plots each pair of 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 then connects consecutive data points with line segments. However, MATLAB offers a variety of alternatives to override this default style including a variety of line styles, markers, and color combinations. Markers are especially valuable if you 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 plot similar to that shown in Figure 2. Note that the *string 'ro'* tells MATLAB to place a "red o" at each point.

Although the MATLAB `plot` command connects consecutive points with line segments, it is possible to create a plot that appears to be continuous and smooth. To do this, you must generate a sufficient number of data points for the plot so that the line segments between the points appear to be continuous. The following commands will plot the polynomial $p(x) = x^3 - 6x^2 + 3x + 10$ over the interval $[-2, 6]$ as shown in Figure 3,

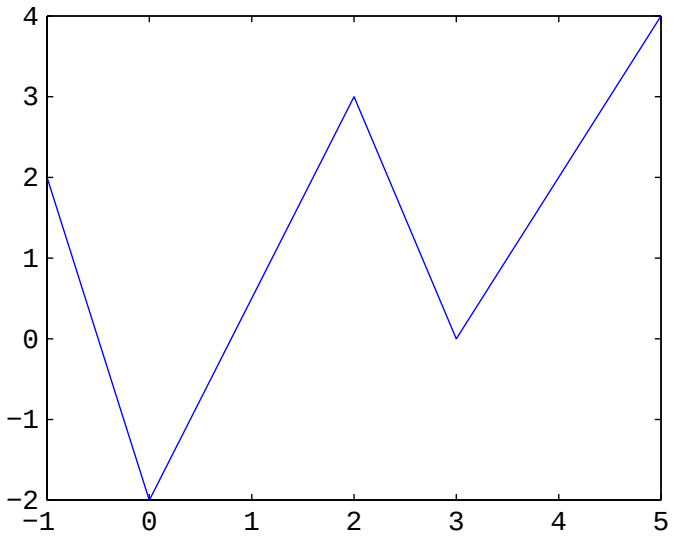

Figure 1: The data points are connected by line segments.

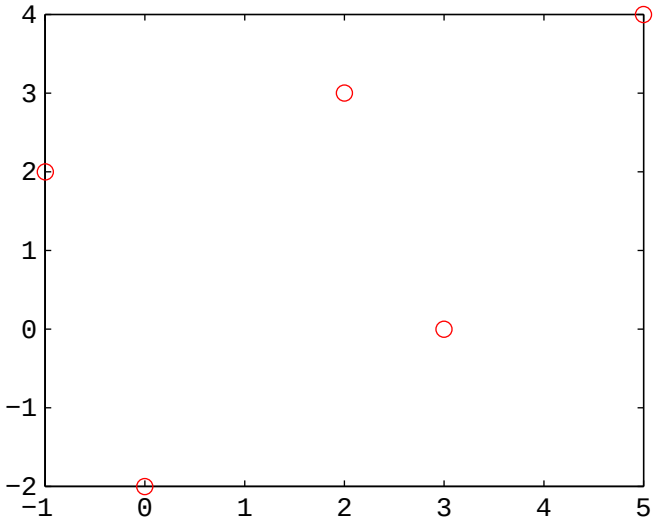

Figure 2: Using markers to display discrete data points.

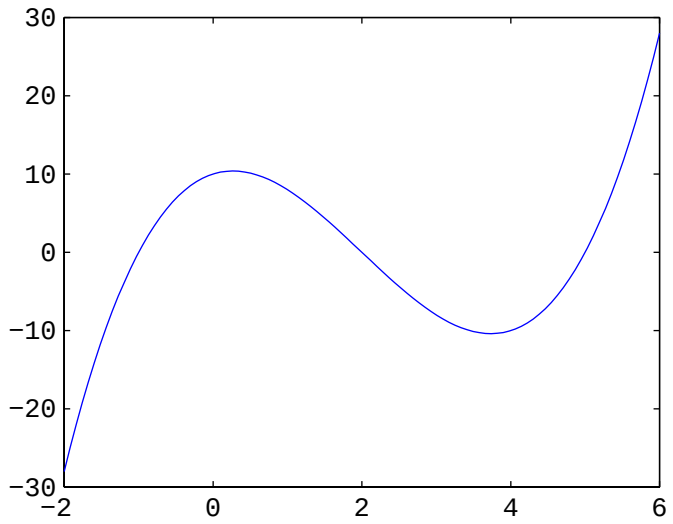

Figure 3: The graph of a polynomial as a smooth curve.

```
>> 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)
```

The `linspace` command generates a vector of equally spaced points. By default, the command `linspace(a,b)` generates 100 equally spaced points between `a` and `b`. More points may be generated with the command `linspace(a,b,N)`. For example, the command `linspace(0,1,1000)` generates 1000 equally spaced points between 0 and 1000.

Given a set of data points, an important problem in many applications is to find a polynomial of a specified degree that pases through each of the points.[2] Such a polynomial is called an interpolating polynomial. Suppose that we want to 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

$$p(x) = ax^3 + bx^2 + cx + d$$

and if we substitute each of the given points into this equation we will have a system of four equations in four unknowns a, b, c, and d,

$$\begin{aligned} -27a + 9b - 3c + d &= 0 \\ -8a + 4b - 2c + d &= -3 \\ d &= 3 \\ 8a + 4b + 2c + d &= -15 \end{aligned}$$

The first step to solve this linear system is to set up the augmented matrix.

$$\left[\begin{array}{cccc|c} -27 & 9 & -3 & 1 & 0 \\ 9 & 4 & -2 & 1 & -3 \\ 1 & 1 & 0 & 1 & 3 \\ 8 & 4 & 2 & 1 & 15 \end{array}\right]$$

Entering this matrix in MATLAB and using the `rref` command we obtain the reduced row echelon form, which is

[2] Later, we will look at how to 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.

```
>> 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 the linear system is $a = -1$, $b = -3$, $c = 1$, and $d = 3$. Substituting these values into the general third degree polynomial, $p(x) = ax^3 + bx^2 + cx + d$, we find that the interpolating polynomial is

$$p(x) = -x^3 - 3x^2 + x + 3$$

Note: It is important that you check the solution. Remember that the interpolating polynomial must pass through each of the data points, and if it does not, then you do not have the correct answer. So, first we will plot the original data as discrete points, and then plot the polynomial. If the polynomial passes through each data point, then we are confident that we have the correct answer. So, first 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 values of x in this set of points are -3 and 2, respectively. Therefore, we will plot the polynomial $p(x) = -x^3 - 3x^2 + x + 3$ over the interval $[-4, 3]$ so that we are sure that each of the given data points will be visible in the final plot. The following commands will produce a graph similar to that shown 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,'-')
```

Note that the MATLAB plot command is an extremely flexible tool. In the commands given above, two plots were created with one command. In general, the command

```
plot(x1,y1,s1,x2,y2,s2,...,xN,yN,sN)
```

will draw `N` plots: the plot of `y1` versus `x1`, the plot of `y2` versus `x2`, and so on, up to the last plot of `yN` versus `xN`. Different line styles, markers, and colors are used for each plot using the strings `s1, s2,. . . ,sN`.

Now let's consider the problem of finding a fourth-order interpolating polynomial that passes through the following set of five points

$$(x_1, y_1), \quad (x_2, y_2), \quad (x_3, y_3), \quad (x_4, y_4), \quad (x_5, y_5)$$

Substituting each of these points into the polynomial

$$y = ax^4 + bx^3 + cx^2 + dx + e$$

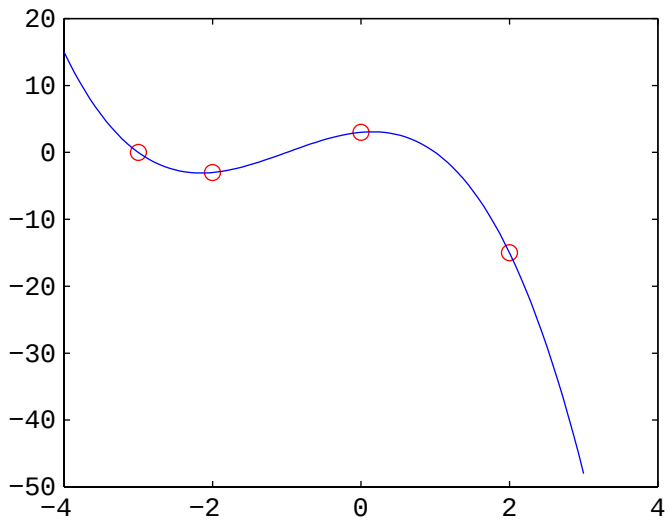

Figure 4: The interpolating polynomial must pass through each data point.

we will have a linear system of five equations in the five unknowns, a, b, c, d, e,

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

The augmented matrix for this linear system is

$$\left[\begin{array}{ccccc|c}
x_1^4 & x_1^3 & x_1^2 & x_1 & 1 & y_1 \\
x_2^4 & x_2^3 & x_2^2 & x_2 & 1 & y_2 \\
x_3^4 & x_3^3 & x_3^2 & x_3 & 1 & y_3 \\
x_4^4 & x_4^3 & x_4^2 & x_4 & 1 & y_4 \\
x_5^4 & x_5^3 & x_5^2 & x_5 & 1 & y_5
\end{array}\right]$$

If we define x and y to be the vectors

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} \quad ; \quad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{bmatrix}$$

then note that the columns of the augmented matrix may be created in MATLAB with the commands `x.^4`, `x.^3`, `x.^2`, `x`, `ones(size(x))`, and `y`. The coefficient matrix of this system 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.

As a specific example, let's find a fourth-order interpolating polynomial that passes through the points $(-2, 26)$, $(-1, -2)$, $(1, -4)$, $(2, -2)$, and $(4, 128)$. First, create the two vectors x and y that contain the x and y coordinates of the given set of points,

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

Next, we create the augmented matrix in MATLAB as follows,

```
>> 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
```

The next step is to find the reduced row echelon form of the augmented matrix,

```
>> 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
```

Therefore, the coefficients of our interpolation polynomial, which are given in the last column of the reduced echelon form of `N` are

$$a = 1, \quad b = -2, \quad c = 0, \quad d = 1, \quad e = -4$$

These coefficients may be extracted from `M` using the `colon` operator,

```
>> p=R(:,6)
ans =
      1
     -2
      0
      1
     -4
```

Before continuing, note that the complete set of commands used to find this solution are

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

The last step will be to check to make sure that the polynomial passes through the given points. The interpolation polynomial that we have found is given by

$$y = x^4 - 2x^3 + x - 4$$

Now, recall that the original data is still stored in the vectors x and y, so all we need to do is plot the polynomial to see if it passes through these points. Since the minimum value of x is -2 and the maximum value is 4, let's plot the polynomial over the interval $[-3, 5]$. The following commands produce a plot similar to that in Figure 5.

```
>> 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,'-')
```

Because the polynomial passes through each data point, it is highly likely that we have found the correct interpolating polynomial. Now, it is time to work a real problem.

Problem 1.1

(a) Find a tenth-order 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 $(5,-1)$.

(b) Plot the data points and the interpolating polynomial to see if the polynomial passes through the given points.

Note: This problem may cause headaches, because it turns out that the interpolating polynomial is extremely sensitive to small changes in some of its coefficients. In other words, the interpolating polynomial can do some pretty wild things, including missing most of its data points if the coefficients are truncated 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 a screen output of about 14 significant digits for your coefficient. The problem with this is that typing 14-digit numbers will be tedious and prone to typing errors.

So, another thing that will help is to create a *script file* that contains 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 (or use notepad++ in Octave), and then enter the following sequence of commands,

```
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 a name, such as `MyPoly` and return to the MATLAB prompt and enter the filename.

```
>> MyPoly
```

This should produce the image shown in Figure 5 without having to type any 14-digit numbers.

If you decide to use a script file on this first exercise, then you will find that it will be 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 what researchers do when working with MATLAB in their projects.

There is a second work-around 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
```

By default, MATLAB displays floating point numbers with four decimal places on your screen. For example, if you enter `x=1/7` at the prompt, MATLAB responds with `0.1429`. However, this is not what MATLAB stores internally. MATLAB stores the number in double precision. Therefore, assuming that the interpolating polynomial is unique, a simple way to build the polynomial without entering the coefficients individually is to simply strip off the last column

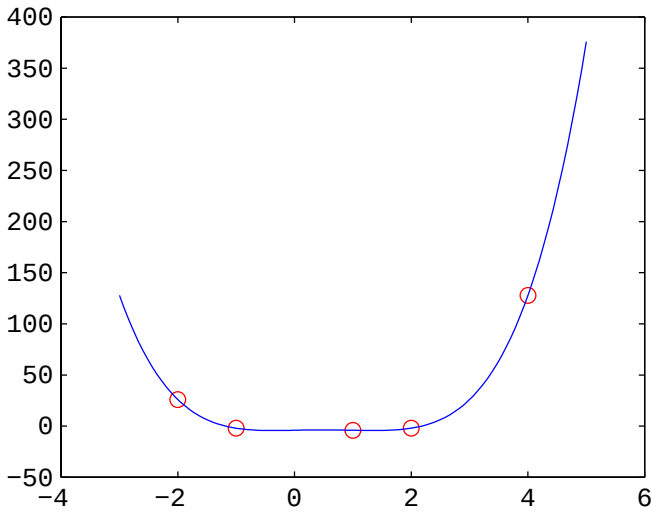

Figure 5: The interpolating polynomial must pass through each data point.

of the augmented matrix after it has been placed in reduced row echelon form. If the reduced row echelon form of the augmented matrix is

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 4 \\ 0 & 1 & 0 & 0 & 0 & 7 \\ 0 & 0 & 1 & 0 & 0 & 3 \\ 0 & 0 & 0 & 1 & 0 & 6 \\ 0 & 0 & 0 & 0 & 1 & 2 \end{bmatrix}$$

then the command

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

will store the last column of the matrix in the array `p`. Since the last column contains the coefficients of the interpolating polynomial, this is an effective way to build the interpolating polynomial. So, remember that even though only 4 decimal places are displayed, internally MATLAB keeps as many decimal places for each coefficient as possible. You can easily see this by entering the commands

```
>> x = sqrt(2);
>> format
>> x
>> format long
>> x
```

There is one last problem that you may encounter in working this problem. Polynomials can grow so quickly (not exponentially, but still pretty fast) that the scale is 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 the local behavior of the interpolating polynomial. Type `help axis` at the MATLAB prompt and read the help file to learn how to use this important command. You can also use the zoom tools in the figure windows toolbar to highlight regions of interest.

(c) 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, make a

plot that contains both the data points and the interpolating polynomial. If the interpolating polynomial does not exist, make a plot of the data points and explain why there can be no second degree polynomial that passes through the points.

(d) 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, make a plot that contains both the data points and the interpolating polynomial. If the interpolating polynomial does not exist, make a plot of the data points and explain why there can be no second degree polynomial that passes through the points.

(e) Find two different interpolating polynomials of degree three that pass through the points $(-3, -24)$, $(-1, 8)$, and $(2, -4)$. Make a plot that contains the data points and both of the interpolating polynomials.

(f) Now let's find the polynomial that produces the *best fit* to the function

$$y = 2^x$$

Choose five points between $x = -1$ and $x = 3$ that lie on this curve, and find a fourth-order polynomial that passes through these points.

(g) Plot the interpolating polynomial along with the function $y = 2^x$ and see how close the values of the polynomial are to the function between $x = -1$ and $x = 3$. Describe what happens when the polynomial is compared to the function for values of x that are outside the interval $[-1, 3]$. This problem illustrates the important issue of "goodness of fit." In fitting a polynomial through a set of points, the underlying assumption is that there is a polynomial relationship between the variables, or one that is approximately polynomial. If this is not the case, then the interpolating polynomials may not be a good solution for representing (modeling) the data.