# Week 9: Polynomial Interpolation and Least Squares Fitting

## 1 Interpolation

Given some $x$-$y$ data $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$, the goal of polynomial interpolation is to find a polynomial that passes through all of the data points. For example, suppose you have three data points: $(0,1), (0.5,-2)$, and $(1,3)$. With three points, you can find a quadratic polynomial that passes through all of them. First, assume the polynomial looks like

$$p(x) = c_1 + c_2 x + c_3 x^2.$$

Next, force the polynomial to match the data at each of the $x$-coordinates:

$$\begin{aligned} p(0) =& c_1 + c_2 \cdot 0 + c_3 \cdot 0^2 = 1, \\ p(0.5) =& c_1 + c_2(0.5) + c_3(0.5)^2 = -2, \\ p(1) =& c_1 + c_2 \cdot 1 + c_3 \cdot 1^2 = 3. \end{aligned}$$

This gives a system of three equations and three unknowns, $\mathbf{A}\vec{\mathbf{c}} = \vec{\mathbf{y}}$, which can be solved in Matlab using the backslash \ operator. In this example, the linear system ends up looking like this:

$$\begin{bmatrix} 1 & 0 & 0^2 \\ 1 & 0.5 & (0.5)^2 \\ 1 & 1 & 1^2 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} 1 \\ -2 \\ 3 \end{bmatrix}.$$

In general, whenever you have $n$ data points, you can always find a polynomial of degree $n-1$ which passes through all of them:

$$\begin{bmatrix} 1 & x_1 & x_1^2 & \ldots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \ldots & x_2^{n-1} \\ 1 & x_3 & x_3^2 & \ldots & x_3^{n-1} \\ \vdots & \vdots & \vdots & \ldots & \vdots \\ 1 & x_n & x_n^2 & \ldots & x_n^{n-1} \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{bmatrix}$$

On the other hand, polynomials of very high degree can have undesirable properties, so next we will consider an alternative which, in Matlab, is basically the same process.

## 2 Least Squares Fitting

**Details on the mathematical derivation of the least-squares method can be found in your book, on pages 162-163 and in Problem 45 of Section 3.4.**
If the number of points is larger than the degree of polynomial that you want to use, then the linear system for determining the coefficients will be over-determined (more rows than columns). To find the least-squares polynomial of a given degree, you carry out the same

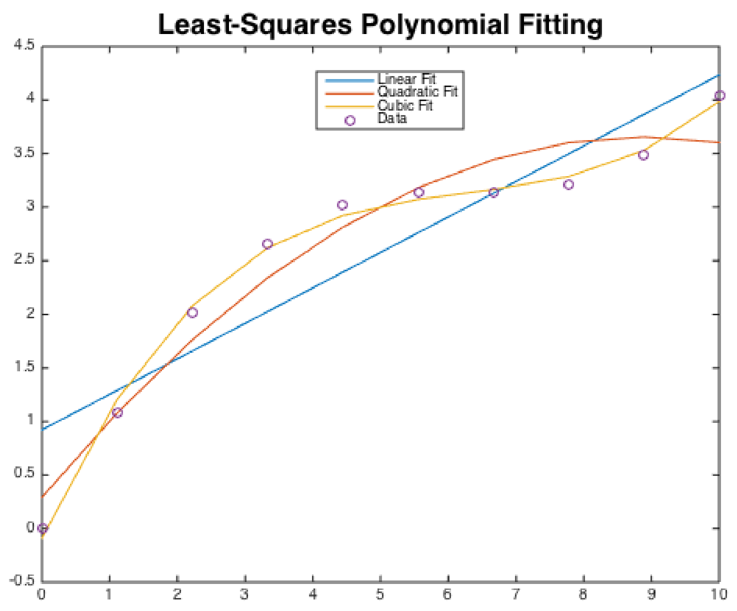

Figure 1: Example of least squares fitting with polynomials of degrees 1, 2, and 3.

process as we did for interpolation, but the resulting polynomial will not interpolate the data, it will just be "close". Here are some examples of what the linear system will look like for determining the least-squares polynomial coefficients:

$$\text{Linear:}\quad \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ 1 & x_3 \\ \vdots & \vdots \\ 1 & x_n \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{bmatrix}$$

$$\text{Quadratic:}\quad \begin{bmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ 1 & x_3 & x_3^2 \\ \vdots & \vdots & \vdots \\ 1 & x_n & x_n^2 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{bmatrix}$$

$$\text{Cubic:}\quad \begin{bmatrix} 1 & x_1 & x_1^2 & x_1^3 \\ 1 & x_2 & x_2^2 & x_2^3 \\ 1 & x_3 & x_3^2 & x_3^3 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_n & x_n^2 & x_n^3 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{bmatrix}$$

If you were solving one of these over-determined linear systems by hand, you would multiply both sides by $\mathbf{A}^T$, which would give you a square linear system:

$$\mathbf{A}^T\mathbf{A}\vec{\mathbf{c}} = \mathbf{A}^T\vec{\mathbf{y}}.$$

This method will still work in Matlab. You can set

```
c = (A.'*A)\(A.'*b)
```

However, this is unnecessary, because Matlab can recognize an overdetermined system and find the least-squares solution automatically. In other words, you can type `c=A\b` and it will give you the same solution.

## 3 Homework #8

Make a function called `leastSquares.m`, which, given some data points defined by the vectors `x` and `y`, will find and evaluate the least-squares polynomial. The user should be able to choose the degree `d` of polynomial, and the vector `X` of evaluation points. *Hints:*

1. The first line should declare that the m-file is a function with four inputs and one output:

   ```
   function Y = leastSquares( x, y, d, X )
   ```

   - The column-vectors `x` and `y` specify the data that you know.
   - The number `d` is the degree of the polynomial that you want to use.
   - The column-vector `X` has all of the $x$-coordinates where you want to evaluate the least-squares polynomial.
   - The output `Y` will be a column-vector which contains the values of the least-squares polynomial at the $x$-coordinates given in `X`.

2. Here are some guidelines for what your function should do:

   (a) Initialize the matrix **A** to be all zeros with `length(x)` rows and `d+1` columns.

   (b) Initialize the matrix **B** to be all zeros with `length(X)` rows and `d+1` columns.

   (c) Define the entries of the matrix **A** and the matrix **B** using a for-loop over the columns:

   ```
   for j = 1 : d+1
       -set the jth column of A equal to x.^(j-1)
       -set the jth column of B equal to X.^(j-1)
   end
   ```

   (d) Find the least-squares solution to the over-determined linear system `A*c=y`. Even though the linear system may not be square, you can still use the backslash operator to solve for `c`. Matlab will automatically find the least-squares solution if you type `c=A\y`. `c` contains the coefficients for the least-squares polynomial.

   (e) Evaluate the least-squares polynomial at the coordinates given in `X`, by setting `Y=B*c`.

Once your function `leastSquares.m` is complete, test it by picking a function, sampling it at a few points, and using these as your `x` and `y` vectors. For example, you could sample the function $f(x) = e^{-x^2}$ at six equally spaced points from 0 to 1 by typing these lines:

```
f = @(x) exp(-x.^2);
x = (0:.2:1).';
y = f(x);
```

Then, you can choose a polynomial degree, pick which $x$-coordinates you want to evaluate at, and call `leastSquares.m`:

```
d = 2;
X = (0:.01:1).';
Y = leastSquares( x, y, d, X );
```

Finally, view the results of your labor by plotting the original function $e^{-x^2}$ and your least squares fit (represented in your program by the $x$-values $X$ and the $y$-values $Y$) on the same plot. Place a legend on the plot (try `help legend` in the command line if you don't know how to use this MATLAB function) to label the original function and the least squares fit. What happens if you change $d$ from 2 to 3? Does it look like the cubic least-squares polynomial provides a better fit?