Lecture 19
Polynomial and Spline Interpolation

A Chemical Reaction

In a chemical reaction the concentration level y of the product at time t was measured every half hour. The following results were found:

t	0	.5	1.0	1.5	2.0
y	0	.19	.26	.29	.31

We can input this data into MATLAB as

```
> t1 = 0:.5:2
> y1 = [ 0 .19 .26 .29 .31 ]
```

and plot the data with

```
> plot(t1,y1)
```

MATLAB automatically connects the data with line segments. This is the simplest form of *interpolation*, meaning fitting a graph (of a function) between data points. What if we want a smoother graph? Try

```
> plot(t1,y1,'*')
```

which will produce just asterisks at the data points. Next click on `Tools`, then click on the `Basic Fitting` option. This should produce a small window with several fitting options. Begin clicking them one at a time, clicking them off before clicking the next. Which ones produce a good-looking fit? You should note that the spline, the shape-preserving interpolant and the 4th degree polynomial produce very good curves. The others do not. We will discuss polynomial interpolation and spline interpolation in this lecture.

Polynomial Fitting

The most general degree n polynomial is

$$p_n(x) = a_n x^n + a_{n-1}x^{n-1} + \ldots + a_1 x + a_0 \,.$$

If we have exactly $n+1$ data points, that is enough to exactly determine the $n+1$ coefficients of $p_n(x)$ (as long as the data does not have repeated x values). If there are more data points, that would give us an overdetermined system (more equations than unknowns) and if there is less data the system would be undetermined.

Conversely, if we have n data points, then an $n-1$ degree polynomial has exactly enough coefficients to fit the data. This is the case in the example above; there are 5 data points so there is exactly one 4th degree polynomial that fits the data.

When we tried to use a 5th or higher degree polynomial MATLAB returned a warning that the polynomial is not unique since "degree >= number of data points".

When we tried a lower degree polynomial, we did not get a warning, but we did notice that the resulting curve does not hit all the data points. If there was an overdetermined system how did MATLAB come up with a pretty good approximation (for cubic)? The answer is the Least Squares Method, which we will study later.

Predicting the future?

Suppose we want to use the data to extrapolate into the future. Set the plot to the 4th degree polynomial. Then click the `Edit` button and select the `Axes Properties` option. A box should appear that allows you to adjust the domain of the x axes. Change the upper limit of the x-axis from 2 to 4. Based on the 4th degree polynomial, what will the chemical reaction do in the future? Is this reasonable?

Next change from 4th degree polynomial to spline interpolant. According to the spline, what will the chemical reaction do in the future? Try the shape-preserving interpolant also.

From our (limited) knowledge of chemical reactions, what should be the behavior as time goes on? It should reach a limiting value (chemical equilibrium). Could we use the data to predict this equilibrium value? Yes, we could and it is done all the time in many different contexts, but to do so we need to know that there is an equilibrium to predict. This requires that we understand the chemistry of the problem. Thus we have the following principle: To *extrapolate* beyond the data, one must have some knowledge of the process.

More data

Generally one would think that more data is better. Input the following data vectors:

```
> t2 = [ 0  .1  .4  .5  .6  1.0  1.4 1.5 1.6 1.9 2.0]
> y2 = [ 0  .06 .17 .19 .21 .26  .29 .29 .30 .31 .31]
```

There are 11 data points, so a 10-th degree polynomial will fit the data. However, this does not give a good graph. Thus: **A polynomial fit is better for small data sets.**

Finally, we note that each data point (x_i, y_i) gives an equation

$$p_n(x_i) = a_0 + a_1 x_i + a_2 x_i^2 + \cdots + a_n x_i^n = y_i.$$

The unknowns are the coefficients $a_0, \ldots, a_n$. This equation is linear in these unknowns, so determining a polynomial fit requires solving a linear system of equations. This makes polynomial fits quick to calculate since *solving linear systems is what computers do best.*

A challenging data set

Input the following data set:

```
> x = -4:1:5
> y = [ 0 0 0 1 1 1 0 0 0 0]
```

and plot it:

```
> plot(x,y,'*')
```

There are 10 data points, so there is a unique 9 degree polynomial that fits the data. Under `Tools` and `Basic Fitting` select the 9th degree polynomial fit. How does it look? De-select the 9th degree polynomial and select the spline interpolant. This should produce a much more satisfactory graph and the shape-preserving spline should be even better. In this section we learn what a spline is.

The idea of a spline

The general idea of a spline is this: on each interval between data points, represent the graph with a simple function. The simplest spline is something very familiar to you; it is obtained by connecting the data with lines. Since linear is the most simple function of all, linear interpolation is the simplest form of spline. The next simplest function is quadratic. If we put a quadratic function on each interval then we should be able to make the graph a lot smoother. If we were really careful then we should be able to make the curve smooth at the data points themselves by matching up the derivatives. This can be done and the result is called a

quadratic spline. Using cubic functions or 4th degree functions should be smoother still. So, where should we stop? There is an almost universal consensus that *cubic* is the optimal degree for splines and so we focus the rest of the lecture on cubic splines.

Cubic spline

Again, the basic idea of the cubic spline is that we represent the function by a different cubic function on each interval between data points. That is, if there are n data points, then the spline $S(x)$ is the function

$$S(x) = \begin{cases} C_1(x), & x_0 \le x \le x_1 \\ C_i(x), & x_{i-1} \le x \le x_i \\ C_n(x), & x_{n-1} \le x \le x_n \end{cases} \tag{19.1}$$

where each C_i is a cubic function. The most general cubic function has the form

$$C_i(x) = a_i + b_i x + c_i x^2 + d_i x^3.$$

To determine the spline we must determine the coefficients, a_i, b_i, c_i, and d_i for each i. Since there are n intervals, there are $4n$ coefficients to determine. First we require that the spline be exact at the data:

$$C_i(x_{i-1}) = y_{i-1} \quad \text{and} \quad C_i(x_i) = y_i, \tag{19.2}$$

at every data point. In other words,

$$a_i + b_i x_{i-1} + c_i x_{i-1}^2 + d_i x_{i-1}^3 = y_{i-1} \quad \text{and} \quad a_i + b_i x_i + c_i x_i^2 + d_i x_i^3 = y_i.$$

Notice that there are $2n$ of these conditions. Then to make $S(x)$ as smooth as possible we require

$$\begin{aligned} C_i'(x_i) &= C_{i+1}'(x_i) \quad \text{and} \\ C_i''(x_i) &= C_{i+1}''(x_i) \end{aligned} \tag{19.3}$$

at all the internal points, i.e. $x_1, x_2, x_3, \ldots, x_{n-1}$. In terms of the points these conditions can be written as

$$\begin{aligned} b_i + 2c_i x_i + 3d_i x_i^2 &= b_{i+1} + 2c_{i+1} x_i + 3d_{i+1} x_i^2 \quad \text{and} \\ 2c_i + 6d_i x_i &= 2c_{i+1} + 6d_{i+1} x_i. \end{aligned} \tag{19.4}$$

There are $2(n-1)$ of these conditions. Since each C_i is cubic, there are a total of $4n$ coefficients in the formula for $S(x)$. So far we have $4n-2$ equations, so we are 2 equations short of being able to determine all the coefficients. At this point we have to make a choice. The usual choice is to require

$$C_1''(x_0) = C_n''(x_n) = 0. \tag{19.5}$$

These are called *natural* or *simple* boundary conditions. The other common option is called *clamped* boundary conditions:

$$C_1'(x_0) = C_n'(x_n) = 0. \tag{19.6}$$

The terminology used here is obviously parallel to that used for beams. That is not the only parallel between beams and cubic splines. It is an interesting fact that a cubic spline is exactly the shape of a (linear) beam restrained to match the data by simple supports.

Note that the equations above are all linear equations with respect to the unknowns (coefficients). This feature makes splines easy to calculate since *solving linear systems is what computers do best.*

Exercises

19.1 You are given the following data:

```
> t = [ 0  .1  .499  .5  .6  1.0  1.4 1.5 1.899 1.9 2.0]
> y = [ 0  .06 .17 .19 .21 .26  .29 .29 .30 .31 .31]
```

(a) Plot the data, using "*" at the data points, then try a polynomial fit of the correct degree to interpolate this number of data points: What do you observe. Give an explanation of this error, in particular why is the term *badly conditioned* used?

(b) Plot the data along with a spline interpolant. How does this compare with the plot above? What is a way to make the plot better?