Fitting a polynomial function to experimental data

Let's say we have some temperature measurements in column vector T and some heat capacity measurements in column vector C_p, e.g., in MATLAB,

```
>> T = [0 18 25 100]'; % an apostrophe is the transform operator
>> Cp = [29.062 29.075 29.075 29.142]';
```

We want to fit a cubic (3rd order) polynomial to these data. That means, we want to find the values of the unknown coefficients a, b, c, d. With this equation, we can estimate the value of C_p at some other T value.

$$C_p = a + bT + cT^2 + dT^3$$

Even though a polynomial or order 2 or greater is a nonlinear equation, our problem here is linear in the unknown coefficients.

$$\begin{bmatrix} 0^3 & 0^2 & 0 & 1 \\ 18^3 & 18^2 & 18 & 1 \\ 25^3 & 25^2 & 25 & 1 \\ 100^3 & 100^2 & 100 & 1 \end{bmatrix} \begin{bmatrix} d \\ c \\ b \\ a \end{bmatrix} = \begin{bmatrix} 29.062 \\ 29.075 \\ 29.075 \\ 29.142 \end{bmatrix}$$

Fitting a polynomial function to experimental data

With a cubic polynomial, we have 4 unknowns. Here, we also have 4 T,C_p pairs and, thus, 4 equations. Thus, we have a uniquely determined system and can find a polynomial function that goes through all 4 data points.

```
>> A = [T.^3 T.^2 T.^1 T.^0] % note dot operator .^ which operates element-by element
A =

           0           0           0           1
        5832         324          18           1
       15625         625          25           1
     1000000       10000         100           1

>> x = inv(A)*Cp
x =

   3.9783e-07
  -4.5996e-05
   1.4212e-03
   2.9062e+01
```

after previously setting the display format to short e. But what if we have more than 4 data points...

Fitting a polynomial function to experimental data

```
>> T = [0 18 25 100 200 300 400]';
>> Cp = [29.062 29.075 29.075 29.142 29.292 29.514 29.782]';
```

Here we have 7 data points. We don't want to fit a 6-th order polynomial to these data because it would fit the random errors and give us bad predictions except at the T's where we have C_p measurements.

We still want to fit a cubic polynomial to these data. But now we have 4 unknowns and 7 equations: an over-determined system. Our matrix A now has 7 rows and 4 columns. The A matrix is not square and so we cannot invert it:

$$\underset{A}{\begin{bmatrix} T_1^3 & T_1^2 & T_1^1 & T_1^0 \\ T_2^3 & T_2^2 & T_2^1 & T_2^0 \\ T_3^3 & T_3^2 & T_3^1 & T_3^0 \\ T_4^3 & T_4^2 & T_4^1 & T_4^0 \\ T_5^3 & T_5^2 & T_5^1 & T_5^0 \\ T_6^3 & T_6^2 & T_6^1 & T_6^0 \\ T_7^3 & T_7^2 & T_7^1 & T_7^0 \end{bmatrix}} \quad \underset{x}{\begin{bmatrix} d \\ c \\ b \\ a \end{bmatrix}} = \underset{y}{\begin{bmatrix} 29.062 \\ 29.075 \\ 29.075 \\ 29.142 \\ 29.292 \\ 29.514 \\ 29.782 \end{bmatrix}}$$

But there is a solution. Multiply from the left by the transform of A

$$[A^T A]x = A^T y$$

The matrix $[A^T A]$ is square and can be inverted

$$[A^T A]^{-1}[A^T A]x = [A^T A]^{-1} A^T y$$

$$x = [A^T A]^{-1} A^T y$$

Our solution x, the vector of our polynomial coefficient values, is the solution that minimizes the sum of the squared errors between the experimental values and the model values predicted by the polynomial.

$$min. \sum_i (C_{p,i}^{\exp} - C_{p,i}^{model})^2$$

That is, this solution provides the "least squares" fit of the polynomial to these data.

Fitting a polynomial function to experimental data

In Matlab, this is

```
>> y = Cp;
>> A = [T.^3 T.^2 T.^1 T.^0];
>> x = inv(A'*A)*A'*y
x =
  -1.9925e-09
   4.4195e-06
   3.4656e-04
   2.9064e+01
```

For larger data sets, the inv function may not work well. There are alternatives that accomplish the same result but use other numerical algorithms, e.g., the backslash operator (left matrix divide):

```
>> x = A\y
x =

  -1.9925e-09
   4.4195e-06
   3.4656e-04
   2.9064e+01
```

Fitting a polynomial function to experimental data

Fitting a polynomial to data points is such a common problem that Matlab and other tools have special functions to accomplish this task. See the examples at the course web site's Math Tools link. Here, use the standard polyfit function in Matlab to fit a 3rd-order polynomial to these data:

```
>> x = polyfit(T,Cp,3)
x =
  -1.9925e-09   4.4195e-06   3.4656e-04   2.9064e+01
```

Now generate a plot of the data points and the polynomial curve. The curve doesn't exactly go through each data point, even though it looks that way on this small snapshot of the plot.

```
>> Tm = T(1):1:T(7); % compute model at more points so can see if it "wiggles" between data points
>> Cpm = polyval(x,Tm); % compute model using standard polyval function and coefficient vector x
>> plot(T,Cp,'bo',Tm,Cpm,'k')
>> title('Cp vs. T & cubic polynomial fit')
```