Interpolation

Examples of Applications: Interpolation problems can arise

- in heat transfer estimation
- in approximating of function values (e.g., temperature) at points where there is no data reading
- when finding derivatives from experimental data values.

Read text pp. 221-3 Gerald and Wheatley or p. 150 Epperson.

If the data is precise, our interpolation methods should work well in many cases, but if the data is not precise, problems are likely to arise. If the data is expected to have errors, we can fit curves with few parameters that go near the true points in some "best" sense. Minimizing the error in a particular sense leads to the *Least Squares* fitting technique.

Interpolating can turn complicated functions into much simpler ones (like polynomials or trigonometric functions) that are easier to evaluate. This can improve efficiency if the function is to be called many times.

Straight lines - These are okay for connecting points but they do not have continuous derivatives. The derivative does not exist at a corner. Piecewise linear curves are not smooth, like most data ought to be.

If we know a function's value and its derivatives at a particular point, then we could use Taylor series:

$$P(x) \;=\; f(a) \;+\; f'(a)(x-a) \;+\; \frac{1}{2!}f''(a)(x-a)^2 \;+\; \ldots \;+\; \frac{1}{n!}f^{(n)}(a)(x-a)^n$$

with error term

$$\frac{(x-a)^{n+1}f^{(n+1)}(\xi)}{(n+1)!}$$

This works well if we have lots of the derivatives at a and if we want to know the function value near a. Sometimes the radius of convergence can be small and thus the Taylor series can be inaccurate far from a.

We want to use polynomials for interpolation computations, since they are easy to evaluate. However, high degree polynomials oscillate wildly so we don't use high degree polynomials. (See fig 3.1 p. 239 Gerald and Wheatley or Fig. 4.3 p. 155 of Epperson)

Example: fit the data points (3.2,22.0)(2.7,17.8)(1.0,14.2)(4.8,38.3) such that they all lie on a polynomial curve.

We can do this by brute force, noting that an n-th degree polynomial can fit $n+1$ points. Let $P(x) = a_0 + a_1x + a_2x^2 + a_3x^3$. We get:

$$\begin{pmatrix} 1 & 3.2 & (3.2)^2 & (3.2)^3 \\ 1 & 2.7 & (2.7)^2 & (2.7)^3 \\ 1 & 1.0 & (1.0)^2 & (1.0)^3 \\ 1 & 4.8 & (4.8)^2 & (4.8)^3 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{pmatrix} 22.0 \\ 17.8 \\ 14.2 \\ 38.3 \end{pmatrix}$$

Drawbacks:

- With lots of points, the matrix becomes big and has high condition number. It is ill-conditioned. So the computation is sensitive to small errors in the experimental data (or numerical error/rounding error).
- This method would give us a polynomial that passes through all the points, but adding a new data point would mean we need to start all over. We cannot use anything from the previous result.
- High degree polynomials tend to have wild oscillations and are often not appropriate for modeling smooth curves, especially those arising in physical situations.

This matrix arising from fitting polynomials through data points has a name the Vandermonde matrix

$$\begin{pmatrix} 1 & x_0 & (x_0)^2 & . & (x_0)^n \\ 1 & x_1 & (x_1)^2 & . & (x_1)^n \\ . & . & . & . & . \\ . & . & . & . & . \\ 1 & x_n & (x_n)^2 & . & (x_n)^n \end{pmatrix}$$

To add the point (5.6,51.7) to the above list of points we have to start over, obtaining:

$$\begin{pmatrix} 1 & 3.2 & (3.2)^2 & (3.2)^3 & (3.2)^4 \\ 1 & 2.7 & (2.7)^2 & (2.7)^3 & (2.7)^4 \\ 1 & 1.0 & (1.0)^2 & (1.0)^3 & (1.0)^4 \\ 1 & 4.8 & (4.8)^2 & (4.8)^3 & (4.8)^4 \\ 1 & 5.6 & (5.6)^2 & (5.6)^3 & (5.6)^4 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \end{pmatrix} = \begin{pmatrix} 22.0 \\ 17.8 \\ 14.2 \\ 38.3 \\ 51.7 \end{pmatrix}$$

Since the n-th degree polynomial through $n+1$ points is unique, any other method of finding it must give the same answer (i.e., the same polynomial).

Lagrange Polynomials

We write polynomials that have value 0 at all grid (x) values except for one and that have value 1 at the grid point of interest. If we write all $n+1$ of these and multiply each one by the value at the relevant grid point, we will have the correct polynomial (same as found above) but easier to set up.

Example: for 4 points (x_0, f_0) (x_1, f_1) (x_2, f_2) (x_3, f_3), we have:

$$\begin{aligned} P(x) \quad = \quad & \tfrac{(x-x_1)(x-x_2)(x-x_3)}{(x_0-x_1)(x_0-x_2)(x_0-x_3)} f_0 \; + \; \tfrac{(x-x_0)(x-x_2)(x-x_3)}{(x_1-x_0)(x_1-x_2)(x_1-x_3)} f_1 \\ & + \; \tfrac{(x-x_0)(x-x_1)(x-x_3)}{(x_2-x_0)(x_2-x_1)(x_2-x_3)} f_2 + \tfrac{(x-x_0)(x-x_1)(x-x_2)}{(x_3-x_0)(x_3-x_1)(x_3-x_2)} f_3 \end{aligned}$$

Again, if we add a new point, we have to start all over.

An error term can be derived (based on a Taylor series-like error derivation) and it yields:

$$E(x) \; = \; \frac{(x-x_0)(x-x_1)\ldots(x-x_n)}{(n+1)!} f^{(n+1)}(\xi)$$

for x and ξ in the interval containing all the x_is.

It is difficult to apply this formula since we usually don't know the function $f(x)$, just its values at a bunch of points. If we know the function, we estimate the error by using the largest possible value of $f^{(n+1)}(\xi)$ on the interval.

Divided Differences

Here's another way to compute the same polynomial. The advantage here is that if we add a new point, we can still use all the old information. We write the solution in the case of four points (x_0, f_0) (x_1, f_1) (x_2, f_2) and (x_3, f_3):

$$\begin{aligned} P_n(x) \quad = \quad & f_0 \; + \; (x-x_0) f[x_0, x_1] \; + \; (x-x_0)(x-x_1) f[x_0, x_1, x_2] \\ & + \; (x-x_0)(x-x_1)(x-x_2) f[x_0, x_1, x_2, x_3] \end{aligned}$$

$$\text{where } f[x_0, x_1, ..., x_n] \; = \; \frac{f[x_1, x_2, ..., x_n] \; - \; f[x_0, x_1, ..., x_{n-1}]}{x_n - x_0}$$

$$\text{and } f[x_0, x_1] \; = \; \frac{f(x_1) - f(x_0)}{x_1 \; - \; x_0}.$$

We can start anywhere in the table, with certain rules, but we usually use the top or bottom value of each column. Here's an example for the data above:

x_i	f_i	$f[x_i, x_{i+1}]$	$f[x_i, x_{i+1}, x_{i+2}]$	$f[x_i, .., x_{i+3}]$	$f[x_i, .., x_{i+4}]$
3.2	22.0	8.400	2.856	−0.528	0.256
2.7	17.8	2.118	2.012	0.087	
1.0	14.2	6.342	2.263		
4.8	38.3	16.750			
5.6	51.7				

Then, the polynomial through the first 4 points is:
$22.0 + 8.4(x - 3.2) + 2.856(x - 3.2)(x - 2.7) - 0.528(x - 3.2)(x - 2.7)(x - 1.0)$
The polynomial through the last four points is:
$51.7 + 16.75(x - 5.6) + 2.263(x - 5.6)(x - 4.8) + 0.087(x - 5.6)(x - 4.8)(x - 1.0)$
The polynomial through all five points can be written: (from the top on down)
$22.0 + 8.4(x - 3.2) + 2.856(x - 3.2)(x - 2.7) - 0.528(x - 3.2)(x - 2.7)(x - 1.0)$
$+0.256(x - 3.2)(x - 2.7)(x - 1.0)(x - 4.8)$
or (from the bottom up)
$51.7 + 16.75(x - 5.6) + 2.263(x - 5.6)(x - 4.8) + 0.087(x - 5.6)(x - 4.8)(x - 1.0)$
$+0.256(x - 5.6)(x - 4.8)(x - 1.0)(x - 2.7)$

Interpolation with equispaced points:

For equispaced points with $x_{i+1} - x_i = h$, we note that $f[x_0, x_1, ..., x_n] = \dfrac{\Delta^n f_0}{n! h^n}$ where Δ denotes forward differences $\Delta f_0 = f(x_1) - f(x_0)$, $\Delta^2 f_0 = f(x_2) - 2f(x_1) + f(x_0)$, etc.

The Newton-Gregory formula for equispaced interpolation is:

$$\begin{aligned} P_n(x) \quad = \quad & f(x_0) + \tfrac{\Delta f_0}{h}(x - x_0) + \tfrac{\Delta^2 f_0}{2! h^2}(x - x_0)(x - x_1) + \ldots \\ & + \tfrac{\Delta^n f_0}{n! h^n}(x - x_0)(x - x_1)\ldots(x - x_{n-1}) \; + \; R_n(x) \end{aligned}$$

where

$$R_n(x) \; = \; \frac{f^{(n+1)}(\xi(x))}{(n+1)!} h^{n+1} \alpha(\alpha - 1)\ldots(\alpha - n) \quad \text{and} \quad \alpha \; = \; \frac{x - x_0}{h}$$

Cubic Splines

High degree polynomials oscillate alot so it's not so good to approximate using them. (E.g. $f(x) = \frac{1}{1+x^2}$ with 11 equidistant points on [-1,1], see Dahlquist p. 101).

To avoid these oscillations, it is common to divide the interval into subintervals and approximate the function using low degree polynomials on each subinterval.

Linear Splines
Piecewise linear is easy. We have a continuous approximation, but we have discontinuous derivatives.

Quadratic Splines
Quadratic splines are piecewise quadratic. We need 3 pieces of information for each quadratic piece. We can match the function values at the endpoints of each subinterval. This gives two conditions. We can also have the first derivative match where two quadratics join. (This is basically half a condition at each end of the quadratic). However, then we can only have one derivative boundary condition (i.e., at only one end of the curve) but not at both ends. Often $f''(x_{left}) = 0$ is used.

Cubic Splines
Cubic splines allow for continuous first and second derivatives, while matching the function values.

Definition of a cubic spline:

Given a function f on [a,b] and nodes $a = x_0 < x_1 < ... < x_n = b$, a cubic spline interpolant S satisfies the following conditions:

1. S is a cubic polynomial on each subinterval $[x_j, x_{j+1}]$
2. $S(x_j) = f(x_j)$ for $j = 0, 1, 2, ..., n$ (The spline matches function values)
3. $S_{j+1}(x_{j+1}) = S_j(x_{j+1})$ for $j = 0, 1, ..., n-2$ (The spline is continuous)
4. $S'_{j+1}(x_{j+1}) = S'_j(x_{j+1})$ for $j = 0, 1, ..., n-2$ (The spline $\in C^1$).
5. $S''_{j+1}(x_{j+1}) = S''_j(x_{j+1})$ for $j = 0, 1, ..., n-2$ (The spline $\in C^2$).
6. 2 Boundary conditions – examples include free boundary $S''(x_0) = S''(x_n) = 0$, or clamped boundary $S'(x_0) = f'(x_0)$ and $S'(x_n) = f'(x_n)$.

We need 4 conditions to determine a cubic. Matching function values at 2 points gives 2; matching the first derivative on the left and right sides of the spline piece is 1/2 each and doing the same for the 2nd derivative on left and right is 1/2 each. Because on the left and right boundaries, we cannot match 1st and 2nd derivatives with the next spline, we have one boundary condition available for each end.

Derivation of the equations for fitting a cubic spline through 4 points: (Gerald and Wheatley)

Let $y_0 = f(x_0)$, $y_1 = f(x_1)$, $y_2 = f(x_2)$, and $y_3 = f(x_3)$,
with boundary conditions $f'(x_0) = z_0$ and $f'(x_3) = z_3$
Let $S_0(x) = a_0 + b_0(x - x_0) + c_0(x - x_0)^2 + d_0(x - x_0)^3$

0th spline: Match left point: $S_0(x_0) = a_0 = y_0$
0th spline: Match right point: $S_0(x_1) = a_0 + b_0(x_1 - x_0) + c_0(x_1 - x_0)^2 + d_0(x_1 - x_0)^3 = y_1$
1st spline: Match left point: $S_1(x_1) = a_1 = y_1$
1st spline: Match right point: $S_1(x_2) = a_1 + b_1(x_2 - x_1) + c_1(x_2 - x_1)^2 + d_1(x_2 - x_1)^3 = y_2$
2nd spline: Match left point: $S_2(x_2) = a_2 = y_2$
2nd spline: Match right point: $S_2(x_3) = a_2 + b_2(x_3 - x_2) + c_2(x_3 - x_2)^2 + d_2(x_3 - x_2)^3 = y_3$
Match left BC: $S_0'(x_0) = b_0 = z_0$
Match right BC: $S_2'(x_3) = b_2 + 2c_2(x_3 - x_2) + 3d_2(x_3 - x_2)^2 = z_3$
Match first derivatives where splines meet: $S_0'(x_1) = S_1'(x_1)$ and $S_1'(x_2) = S_2'(x_2) \Rightarrow$
$b_0 + 2c_0(x_1 - x_0) + 3d_0(x_1 - x_0)^2 = b_1$ and $b_1 + 2c_1(x_2 - x_1) + 3d_1(x_2 - x_1)^2 = b_2$
Match second derivatives where splines meet: $S_0''(x_1) = S_1''(x_1)$ and $S_1''(x_2) = S_2''(x_2) \Rightarrow$
$2c_0 + 6d_0(x_1 - x_0) = 2c_1$ and $2c_1 + 6d_1(x_2 - x_1) = 2c_2$

This gives 12 equations in 12 unknowns, 4 are trivial (for a_0 , a_1, a_2 and b_0). In general, there are $4n$ equations (for $n+1$ points $x_0, \ldots, x_n$ there are n splines) minus $n+1$ trivial ones (matching n left endpoints of splines plus the left boundary condition).

Derivation of the equations for fitting a cubic spline in general:
We work out how we can reduce the number of equations to just n for the case of equidistant points, $x_{j+1} - x_j = h$.

Let $S_j(x) = a_j + b_j(x - x_j) + c_j(x - x_j)^2 + d_j(x - x_j)^3$
Left endpoint of each spline:

$$S_j(x_j) = a_j = f(x_j)$$

So all the as are known.
Match function at the right end of each spline:

$$S_{j+1}(x_{j+1}) = S_j(x_{j+1}) = f(x_{j+1}) \Rightarrow a_j + b_j h + c_j h^2 + d_j h^3 = a_{j+1} \tag{1}$$

Match first derivatives on right:

$$S_{j+1}'(x_{j+1}) = S_j'(x_{j+1}) \Rightarrow b_{j+1} = b_j + c_j h + 3d_j h^2 \tag{2}$$

Match second derivatives on right:

$$S_{j+1}''(x_{j+1}) = S_j''(x_{j+1}) \Rightarrow 2c_{j+1} = 2c_j h + 6d_j h \tag{3}$$

Solve (3) for $d_j \Rightarrow d_j = \frac{1}{3h}[c_{j+1} - c_j]$
Eliminate d_j in (1) $\Rightarrow a_{j+1} = a_j + b_j h + c_j h^2 + \frac{1}{3h}[c_{j+1} - c_j]h^3$ or

$$a_{j+1} = a_j + b_j h + \frac{h^2}{3}[c_{j+1} + 2c_j] \tag{4}$$

Eliminating d_j in (2) $\Rightarrow$ $\begin{array}{rcl} b_{j+1} & = & b_j + 2c_jh + 3\frac{1}{3h}[c_{j+1} - c_j]h^2 \\ & = & b_j + 2c_jh + hc_{j+1} - hc_j. \end{array}$

So, $b_{j+1} = b_j + h(c_j + c_{j+1})$ or reducing the index $b_j = b_{j-1}h + h(c_{j-1} + c_j)$ (5)

Solve (4) for b_j, giving

$$b_j = \frac{a_{j+1} - a_j}{h} - \frac{h}{3}[c_{j+1} + 2c_j] \text{ or reducing index } b_{j-1} = \frac{a_j - a_{j-1}}{h} - \frac{h}{3}[c_j + 2c_{j-1}] \quad (6)$$

Using (6) in (5) gives

$$\frac{a_{j+1} - a_j}{h} - \frac{h}{3}[c_{j+1} + 2c_j] = \frac{a_j - a_{j-1}}{h} - \frac{h}{3}[c_j + 2c_{j-1}] + h(c_{j-1} + c_j)$$

Bringing the cs together gives:

$$hc_{j-1} + 4hc_j + hc_{j+1} = \frac{3}{h}(a_{j+1} - 2a_j + a_{j-1}) \Rightarrow$$

$$c_{j-1} + 4c_j + c_{j+1} = \frac{3}{h^2}(a_{j+1} - 2a_j + a_{j-1}) \text{ where } j = 1, 2, ..., n-1$$

We use a pretend S_n and the right endpoint boundary condition to deal with it.

Thus we have a linear system for the cs (since the as are known. After finding the cs, we can plug into (3) or the simpler equation just after it to compute the ds and into (6) to get the bs. Thus, we have derived the following tridiagonal system for the cs.

$$\begin{array}{rrrrrrrrrrr}
c_0 & +4c_1 & + c_2 & & & & & & = & \frac{3}{h^2}(a_2 - 2a_1 + a_0) \\
 & c_1 & +4c_2 & + c_3 & & & & & = & \frac{3}{h^2}(a_3 - 2a_2 + a_1) \\
 & & c_2 & +4c_3 & +c_4 & & & & = & \frac{3}{h^2}(a_4 - 2a_3 + a_2) \\
 & & & . & . & . & & & = & . \\
 & & & & . & . & . & & = & . \\
 & & & & & . & . & . & = & . \\
 & & & & & c_{n-3} & +4c_{n-2} & + c_{n-1} & = & \frac{3}{h^2}(a_{n-1} - 2a_{n-2} + a_{n-3}) \\
 & & & & & & c_{n-2} & + 4c_{n-1} + c_n & = & \frac{3}{h^2}(a_n - 2a_{n-1} + a_{n-2})
\end{array}$$

This is $n-1$ equations for the n unknowns $c_0 - c_{n-1}$ plus c_n which is fake. We use the boundary conditions to deal with c_0 and c_n.

Free boundaries:
$S_0''(x_0) = 0 \Rightarrow 2c_0 = 0 \Rightarrow c_0 = 0$.
Similarly, $S_n''(x_n) = 0 \Rightarrow 2c_n = 0 \Rightarrow c_n = 0$.

Clamped boundaries:
Left boundary: $S_0'(x_0) = z_0$ and $S_n'(x_n) = z_n$.
Thus, $b_0 = z_0$, which, using (6) becomes $z_0 = \frac{a_1 - a_0}{h} - \frac{h}{3}[c_1 + 2c_0]$ or
$2c_0 + c_1 = \frac{3}{h}\left[\frac{a_1 - a_0}{h} - z_0\right]$, and we can eliminate c_0.

Right boundary: We have $b_n = z_n = b_{n-1} + (c_{n-1} + c_n)h$ from (5) and using (6) we get
$z_n = \frac{a_n - a_{n-1}}{h} - \frac{h}{3}[c_n + 2c_{n-1}] + (c_{n-1} + c_n)h$.
Bringing the cs to one side gives: $c_{n-1} + 2c_n = \frac{3}{h}\left[z_n - \frac{a_n - a_{n-1}}{h}\right]$, and we can eliminate c_n.

For non-uniform grid spacing we use the general equation:
$h_{i-1}S_{i-1} + (2h_{i-1} + 2h_i)S_i + h_i S_{i+1} = 6\left(\frac{y_{i+1} - y_i}{h_i} - \frac{y_i - y_{i-1}}{h_{i-1}}\right)$

Linearity (free) end conditions ($y''(x_0) = y''(x_n) = 0$) $\Rightarrow c_0 = c_n = 0$.
Clamped boundary conditions $f'(x_0) = A$ and $f'(x_n) = B$ give
$2h_0c_0 + h_0c_1 = \frac{3[y_1 - y_0]}{h_o} - 3A$ and $h_{n-1} + c_n h_{n-1} = \frac{3[y_n - y_{n-1}]}{h_{n-1}} - 3B$ and we can eliminate c_0 and c_n.

Another Spline Derivation (Epperson)

Consider the function

$$B(x) = \begin{cases} 0 & x \le -2 \\ (x+2)^3 & -2 \le x \le -1 \\ 1 + 3(x+1) + 3(x+1)^2 - 3(x+1)^3 & -1 \le x \le 0 \\ 1 + 3(1-x) + 3(1-x)^2 - 3(1-x)^3 & 0 \le x \le 1 \\ (2-x)^3 & 1 \le x \le 2 \\ 0 & x \ge 2 \end{cases}$$

Check that $B(x)$ is a cubic spline.

Notice that:

$B(0) = 4$	$B(\pm 1) = 1$	$B(\pm 2) = 0$
$B'(0) = 0$	$B'(\pm 1) = \mp 3$	$B'(\pm 2) = 0$
$B''(0) = 12$	$B''(\pm 1) = 6$	$B''(\pm 2) = 0$

$B(x_i)$ is a spline with non-zero grid values only at x_i and $x_{i\pm 1}$, with

$B(x_i) = 4$	$B(x_{i\pm 1}) = 1$	$B(x_{i\pm 2}) = 0$
$B'(x_i) = 0$	$B'(x_{i\pm 1}) = \mp \frac{3}{h}$	$B'(x_{i\pm 2}) = 0$
$B''(x_i) = \frac{12}{h^2}$	$B''(x_{i\pm 1}) = \frac{6}{h^2}$	$B''(x_{i\pm 2}) = 0$

Now, we just want to combine

$$\sum_{i=-1}^{n+1} c_i B_i(x) \quad \text{for } n = 0, 1, 2, ..., n$$

such that this sum of the Bs passes through all the (x_i, y_i), i.e.,
$f(x_0) = c_{-1}B_{-1}(x_0) + c_0B_0(x_0) + c_1B_1(x_0)$

$$
\begin{array}{rccccccccccc}
 & c_{-1} & +4c_0 & + c_1 & & & & & & & = & f(x_0) \\
 & & c_0 & +4c_1 & + c_2 & & & & & & = & f(x_1) \\
 & & & c_1 & +4c_2 & + c_3 & & & & & = & f(x_2) \\
 & & & & c_2 & +4c_3 & +c_4 & & & & = & f(x_3) \\
 & & & & & . & . & . & & & = & . \\
\Rightarrow & & & & & & . & . & . & & = & . \\
 & & & & & & . & . & . & & = & . \\
 & & & & & & & c_{n-3} & +4c_{n-2} & + c_{n-1} & = & f(x_{n-2}) \\
 & & & & & & & & c_{n-2} & + 4c_{n-1} + c_n & = & f(x_{n-1}) \\
 & & & & & & & & & c_{n-1} + 4c_n + c_{n+1} & = & f(x_n)
\end{array}
$$

We need 2 end conditions:
Free boundaries (or Natural Spline):
$f''(x_0) = 0 \Rightarrow \frac{6}{h^2}c_{-1} - \frac{12}{h^2}c_0 + \frac{6}{h^2}c_1 = 0 \Rightarrow c_{-1} = 2c_0 - c_1$.
Similarly, $f''(x_n) = = 0 \Rightarrow \frac{6}{h^2}c_{n-1} - \frac{12}{h^2}c_n + \frac{6}{h^2}c_{n+1} = 0 \Rightarrow c_{n+1} = 2c_n - c_{n-1}$.
The linear system becomes:

$$
\begin{array}{rccccccccccc}
 & 6c_0 & & & & & & & & & = & f(x_0) \\
 & c_0 & +4c_1 & + c_2 & & & & & & & = & f(x_1) \\
 & & c_1 & +4c_2 & + c_3 & & & & & & = & f(x_2) \\
 & & & c_2 & +4c_3 & +c_4 & & & & & = & f(x_3) \\
 & & & & . & . & . & & & & = & . \\
 & & & & & . & . & . & & & = & . \\
 & & & & & & . & . & . & & = & . \\
 & & & & & & c_{n-3} & +4c_{n-2} & + c_{n-1} & & = & f(x_{n-2}) \\
 & & & & & & & c_{n-2} & + 4c_{n-1} & + c_n & = & f(x_{n-1}) \\
 & & & & & & & & & 6c_n & = & f(x_n)
\end{array}
$$

Clamped boundaries:
Left boundary: $f'(x_0) = z_0$ and $f'(x_n) = z_n$.
Taking the derivative of $c_{-1}B_{-1}(x_0) + c_0B_0(x_0) + c_1B_1(x_0)$ gives $-c_{-1}\frac{3}{h} + 0c_0 + \frac{3}{h}c_1 = z_0 \Rightarrow c_{-1} = c_1 - \frac{h}{3}z_0$. Similarly, $-c_{n-1}\frac{3}{h} + 0c_n + \frac{3}{h}c_{n+1} = z_n \Rightarrow c_{n+1} = c_{n-1} - \frac{h}{3}z_n$ and

The linear system becomes:

$$\begin{array}{rrrrrrrrrrrrr}
4c_0 & +2c_1 & & & & & & & & = & f(x_0) & + & \frac{h}{3}z_0 \\
c_0 & +4c_1 & + c_2 & & & & & & & = & & f(x_1) & \\
& c_1 & +4c_2 & + c_3 & & & & & & = & & f(x_2) & \\
& & c_2 & +4c_3 & +c_4 & & & & & = & & f(x_3) & \\
& & & . & . & . & & & & = & & . & \\
& & & & . & . & . & & & = & & . & \\
& & & & . & . & . & & = & . & & & \\
& & & & & c_{n-3} & +4c_{n-2} & + c_{n-1} & & = & & f(x_{n-2}) & \\
& & & & & & c_{n-2} & + 4c_{n-1} & + c_n & = & & f(x_{n-1}) & \\
& & & & & & & 2c_{n-1} & + 4c_n & = & f(x_n) & - & \frac{h}{3}z_n
\end{array}$$

Set up the (tridiagonal) system for $f(x) = \sin \pi x$ using 4 intervals on [0,1] with natural boundaries. We get,

$$\begin{array}{rrrrrrr}
6c_0 & & & & = & 0 \\
c_0 & +4c_1 & + c_2 & & = & \frac{\sqrt{2}}{2} \\
& c_1 & +4c_2 & + c_3 & = & 0 \\
& & c_2 & +4c_3 & +c_4 = & \frac{\sqrt{2}}{2} \\
& & & & +6c_4 = & 0
\end{array}$$

Error in Spline Approximation: If $f \in C^4$ then the error is $\leq \frac{5}{384}h^4 M$ where M is the maximum of $f^{(4)}$. There are other types of splines: Bezier splines and B-splines are two of them. These fit cubics, but they do not pass through every data point.

Inverse Interpolation Suppose we have a set of points or a function that is nonlinear in x and we want to know for what value of x the function will be 0. We can perform *inverse interpolation*. That is, we swap the meanings of x and y, perform interpolation on these values and let the value of y be 0.

Example: Find a root of $y = f(x) = x - e^{-x}$.

- List several values of x_i and y_i bracketing the root (0 and 1 bracket the root). E.g., $(0, -1)$, $(\frac{1}{2}, \frac{1}{2} - \frac{1}{\sqrt{e}})$, $(1, 1 - \frac{1}{e})$
- Let $X_i = y_i = x_i - e^{-x_i}$ and $Y_i = x_i$. E.g., $(-1, 0)$, $(\frac{1}{2} - \frac{1}{\sqrt{e}}, \frac{1}{2})$, $(1 - \frac{1}{e}, 1)$
- Perform interpolation (e.g., cubic spline or Lagrange polynomial) to obtain an approximating function for $Y(X)$.
- Let $X = 0$. Then $Y(0)$ is an approximation to the desired root.

Least Squares Approximation

Sometimes we have data, that theoretically should fit on a line but does not fit exactly. (For example, if there are small measurement errors or if the model does not represent the physics exactly). The equations are overdetermined if we have more than 2 points. We want to answer the question: What is the "best" line to fit through the points? One answer could be to minimize the sum of the absolute values of the perpendicular distances between the line and the given points, but perpendicular distances are more complicated than we desire. We can consider vertical differences between the line and the given points $\sum |y - y_i|$ but this is hard to minimize since it does not have a continuous derivative. Instead we minimize the sum of the squares of vertical differences.

$$\text{minimize} \quad \sum_{i=1}^{N} (y_i - (a + bx_i))^2$$

Taking partial derivatives with respect to a and b gives the equations:

$$\begin{array}{rcrcr} a\sum 1 & + & b\sum x_i & = & \sum y_i \\ a\sum x_i & + & b\sum x_i^2 & = & \sum x_i y_i \end{array}$$

Example: Find the "best" line through (2,2) (4,11) (6,28) (8,40). Our method gives:

$$\begin{array}{rcrcr} 4a & + & 20b & = & 81 \\ 20a & + & 120b & = & 536 \end{array}$$

So the "best" line is $y = -12.5 + 6.55x$.

We can also match a polynomial of a different degree, e.g. cubic $a_3x^3 + a_2x^2 + a_1x + a_0$. Here,

$$\text{minimize} \quad \sum_{i=1}^{N} (y_i - (a_3x_i^3 + a_2x_i^2 + a_1x_i + a_0))^2$$

We find (after taking partial derivatives with respect to the as and setting to zero)

$$\begin{pmatrix} \sum 1 & \sum x_i & \sum x_i^2 & \sum x_i^3 \\ \sum x_i & \sum x_i^2 & \sum x_i^3 & \sum x_i^4 \\ \sum x_i^2 & \sum x_i^3 & \sum x_i^4 & \sum x_i^5 \\ \sum x_i^3 & \sum x_i^4 & \sum x_i^5 & \sum x_i^6 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{pmatrix} \sum y_i \\ \sum x_i y_i \\ \sum x_i^2 y_i \\ \sum x_i^3 y_i \end{pmatrix}$$

Some other functional forms can also be put into a least squares format. E.g., $y = be^{ax}$ and $y = bx^a$. Take logs of both sides and fit e.g., $\ln y = \ln b + ax$ or $\ln y = \ln b + a \ln x$.

The results are often good, but these are *not* exactly the same as minimizing the squares of the vertical errors of the original function. To do that would be too difficult as nonlinear equations would arise.

Another functional form:
Suppose we have reason to believe that the points should fit on a curve of the form $y = \frac{ax}{b+x}$. Note that $\frac{1}{y} = \frac{b+x}{ax} = \frac{b}{a}\left(\frac{1}{x}\right) + \frac{1}{a}$. We can then apply least squares with $Y = \frac{1}{y}$ and $X = \frac{1}{x}$. Now $Y = A + BX$ where $A = \frac{1}{a}$ and $B = \frac{b}{a}$.

Multiple Linear Regression:
Suppose our data comes as a set of numbers $x_1, x_2, \ldots, x_n$ leading to an output value y and that we expect y to be linear in the values of the xs. I.e.,

$$y = a_0 + a_1x_1 + a_2x_2 + \ldots + a_nx_n$$

Suppose we have m data points – experimental readings, for example – with $m > n$. We would like to have:

$$\mathbf{A}a = y, \quad \mathbf{A} = \begin{pmatrix} 1 & x_{11} & x_{12} & \ldots & a_{1n} \\ 1 & x_{21} & x_{22} & \ldots & a_{2n} \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ 1 & x_{m1} & x_{m2} & \ldots & a_{mn} \end{pmatrix}, \quad a = \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \ldots \\ a_n \end{pmatrix} \quad y = \begin{pmatrix} y_1 \\ y_2 \\ \ldots \\ \ldots \\ \ldots \\ y_m \end{pmatrix}$$

Since this system is overdetermined, (there are more rows than columns) there is typically no exact solution. Performing the analysis for least squares (partial derivatives set to zero) leads to the result that we need only solve $\mathbf{A}^T\mathbf{A}a = \mathbf{A}^Ty$. Notice that $\mathbf{A}^T\mathbf{A}$ is a square matrix ($m \times m$) and we can use Gaussian elimination.

Least squares for simple periodic data on one period:
Suppose $y = a_0 + a_1\cos(\omega_0 t) + b_1\sin(\omega_0 t)$. Least squares yields (with N data points):

$$\begin{pmatrix} N & \sum\cos(\omega_0 t) & \sum\sin(\omega_0 t) \\ \sum\cos(\omega_0 t) & \sum\cos^2(\omega_0 t) & \sum\cos(\omega_0 t)\sin(\omega_0 t) \\ \sum\sin(\omega_0 t) & \sum\cos(\omega_0 t)\sin(\omega_0 t) & \sum\sin^2(\omega_0 t) \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ b_1 \end{pmatrix} = \begin{pmatrix} \sum y \\ \sum y\cos(\omega_0 t) \\ \sum y\sin(\omega_0 t) \end{pmatrix}$$

Using the identities for sums of sines and cosines over one period:

$$\sum\frac{\cos(\omega_0 t)}{N} = \sum\frac{\sin(\omega_0 t)}{N} = \sum\frac{\cos(\omega_0 t)\sin(\omega_0 t)}{N} = 0$$

$$\text{and } \sum \frac{\cos^2(\omega_0 t)}{N} \;=\; \sum \frac{\sin^2(\omega_0 t)}{N} \;=\; \frac{1}{2}$$

we have

$$\begin{pmatrix} N & 0 & 0 \\ 0 & \frac{N}{2} & 0 \\ 0 & 0 & \frac{N}{2} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ b_1 \end{pmatrix} \;=\; \begin{pmatrix} \sum y \\ \sum y\cos(\omega_0 t) \\ \sum y\sin(\omega_0 t) \end{pmatrix}$$

yielding $a_0 \;=\; \dfrac{\sum y}{N}$, $a_1 \;=\; \dfrac{2\sum y\cos(\omega_0 t)}{N}$, and $b_j \;=\; \dfrac{2\sum y\sin(\omega_0 t)}{N}$ for $j = 1, 2, \ldots, m$.

This result can be generalized for

$$f(t) \;=\; a_0 \;+\; a_1\cos(\omega_0 t) \;+\; b_1\sin(\omega_0 t) \;+\; \ldots \;+\; a_m\cos(m\omega_0 t) \;+\; b_m\sin(m\omega_0 t)$$

yielding $a_0 \;=\; \dfrac{\sum y}{N}$, $a_j \;=\; \dfrac{2\sum y\cos(j\omega_0 t)}{N}$, and $b_j \;=\; \dfrac{2\sum y\sin(j\omega_0 t)}{N}$.

Nonlinear Regression:
Suppose we expect the data to lie on (or near) a messy function that we cannot transform into a linear form that we can use one of the previous least squares techniques on. For example, suppose $y \;=\; f(x, a_0, a_1) = a_0(1 - e^{-a_1 x})$ where the as are unknown. The method presented here is called the Gauss-Newton method.

If we guess values of a_0 and a_1, then we can approximate $f(x_i)$ for nearby values of a_0 and a_1 by $f(x_i)_{j+1}$ where $j+1$ indicates the $j+1$ iterate of a_0 and a_1.

$$f(x_i)_{j+1} \approx f(x_i)_j \;+\; \frac{\partial f(x_i)_j}{\partial a_0}\Delta a_0 \;+\; \frac{\partial f(x_i)_j}{\partial a_1}\Delta a_1$$

This leads to the system of linear equations (noting that we would like $f(x_i)_{j+1}$ to be equal to y_i.

$$\begin{pmatrix} \frac{\partial f(x_1)_j}{\partial a_0} & \frac{\partial f(x_1)_j}{\partial a_1} \\ \frac{\partial f(x_2)_j}{\partial a_0} & \frac{\partial f(x_2)_j}{\partial a_1} \\ \vdots & \vdots \\ \frac{\partial f(x_n)_j}{\partial a_0} & \frac{\partial f(x_n)_j}{\partial a_1} \end{pmatrix} \begin{pmatrix} \Delta a_0 \\ \Delta a_1 \end{pmatrix} \;=\; \begin{pmatrix} y_1 - f(x_1, a_{0,j}, a_{1,j}) \\ y_2 - f(x_2, a_{0,j}, a_{1,j}) \\ \vdots \\ y_n - f(x_n, a_{0,j+1}, a_{1,j+1}) \end{pmatrix}$$

where $\dfrac{\partial f(x_i)_j}{\partial a_0} \;=\; 1 - e^{-a_{1,j}x_i}$ and $\dfrac{\partial f(x_i)_j}{\partial a_1} \;=\; -a_{0,j}x_i e^{-a_{1,j}x_i}$ for the example function above.

Perform least squares ($\mathbf{A}^T\mathbf{A}x \;=\; \mathbf{A}^T y$) to find Δa_0 and Δa_1. Then $a_{0,j+1} = a_{0,j} + \Delta a_0$ and $a_{1,j+1} = a_{1,j} + \Delta a_1$ and repeat until the sequence of a_0s and a_1s converge.

Some issues to note concerning the Gauss-Newton method.

- If the partial derivatives are difficult to compute, numerical derivatives may be used.
- The Gauss-Newton method may converge slowly, oscillate wildly or may not converge at all.
- Another option is to guess a_0 and a_1 and use optimization techniques, such as steepest descent or conjugate gradient, on $\sum(f(x_i) - y_i)^2$.