Fitting: Interpolation

“data fitting” in some sense
interpolate table of data
later: “best” fit to data

Rubin H Landau
With
Sally Haerer

Computational Physics for Undergraduates
BS Degree Program: Oregon State University

“Engaging People in Cyber Infrastructure”
Support by EPICS/NSF & OSU
**Problem: Interpolate Data**

<table>
<thead>
<tr>
<th>$E$</th>
<th>$f(E)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>10.6</td>
</tr>
<tr>
<td>25</td>
<td>16.0</td>
</tr>
<tr>
<td>50</td>
<td>45.0</td>
</tr>
<tr>
<td>75</td>
<td>83.5</td>
</tr>
<tr>
<td>100</td>
<td>52.8</td>
</tr>
<tr>
<td>125</td>
<td>19.9</td>
</tr>
<tr>
<td>150</td>
<td>10.8</td>
</tr>
<tr>
<td>175</td>
<td>8.25</td>
</tr>
<tr>
<td>200</td>
<td>4.7</td>
</tr>
</tbody>
</table>

- Determine $f(E)$ for $E$’s between entries
- Make table of numbers into a function
- Determine full width at half maximum
- Tables = hard
- Look at data
Method 1: Fit polynomial to Small Region

\[ f(x) \approx a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1}, \quad (x \approx x_i) \quad (1) \]

- Local: different poly each interval
- Low \( O \) polynomial (small \( n \)), but many: (b) (c)
- Full region: global fit higher poly (large \( n \))
  - probably bad representation (a)
  - even if perfect fit!
- Do not extrapolate
Point Lagrange Interpolation = \( n-1 \)th O Poly

\[
f(x) \approx f_1 \lambda_1(x) + f_2 \lambda_2(x) + \cdots + f_n \lambda_n(x) \tag{1}
\]

\[
\lambda_i(x) = \prod_{j(\neq i)=1}^{n} \frac{x-x_j}{x_i-x_j} = \frac{x-x_1}{x_i-x_1} \frac{x-x_2}{x_i-x_2} \cdots \frac{x-x_n}{x_i-x_n} \tag{2}
\]

- E.g. 4-points, 3rd degree polynomial to table:

<table>
<thead>
<tr>
<th>( i )</th>
<th>( f_i )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-12</td>
</tr>
<tr>
<td>2</td>
<td>-12</td>
</tr>
<tr>
<td>3</td>
<td>-24</td>
</tr>
<tr>
<td>4</td>
<td>-60</td>
</tr>
</tbody>
</table>

\[
f(x) = (-12) \frac{(x-1)(x-2)(x-4)}{(0-1)(0-2)(0-4)} + (-12) \frac{x(x-2)(x-4)}{(1-0)(1-2)(1-4)} + (-24) \frac{x(x-1)(x-4)}{(2-0)(2-1)(2-4)} + (-60) \frac{x(x-1)(x-2)}{(4-0)(4-1)(4-2)}
\]

\[
\Rightarrow f(x) = x^3 - 9x^2 + 8x - 12 \tag{4}
\]

- Check: \( f(4) = 4^3 - 94^2 + 32 - 12 = -60 \tag{5} \)
Method 2: Cubic Splines

\[ f(x) \approx f_i(x) = f_i + f_i^{(1)}(x - x_i) + \frac{1}{2} f_i^{(2)}(x - x_i)^2 + \frac{1}{6} f_i^{(3)}(x - x_i)^3 \]

- 3rd degree polynomial in each interval: most eye-pleasing
- Continuous 1st, 2nd derivatives
- Spline: flexible drafting tool
- Guaranteed: integrable, differentiable \((F = -dV/dx)\)
- Complex procedure: easy for computers
- Popular for graphics

Try out Applet: [Spline.html](Spline.html)
Cubic Splines Implementation

\[ f(x) \approx f_i(x) = f_i + f_i^{(1)}(x - x_i) + \frac{1}{2} f_i^{(2)}(x - x_i)^2 + \frac{1}{6} f_i^{(3)}(x - x_i)^3 \]  \hspace{1cm} (1)

1. Match \( f_i \) adjoining intervals \ Eq(2)
2. Match \( f^{(1)}, f^{(2)} \) adjoining intervals \ Eq(3)
3. Match \( f^{(3)} \) (forward difference) \ Eq(4)
4. Boundary Conditions: input \( f^{(1)} \)
   - natural spline: \( f^{(2)}(a) = f^{(2)}(b) = 0 \)

\[ f_i(x_{i+1}) = f_{i+1}(x_{i+1}), \quad i = 1, N - 1 \]  \hspace{1cm} (2)

\[ f_i^{(1)}(x_i) = f_i^{(1)}(x_i), \quad f_i^{(2)}(x_i) = f_i^{(2)}(x_i) \]  \hspace{1cm} (3)

\[ f_i^{(3)} \approx \frac{f_{i+1}^{(2)} - f_i^{(2)}}{x_{i+1} - x_i} \]  \hspace{1cm} (4)
public class SplineAppl {
    public static void main( {
        double x[] = { 0., 1.2, 2.5, 3.7, 5., 6.2, 7.5, 8.7, 9.9 }; // input
        double y[] = {0., 0.93,. 6, -0.53, -0.96, -0.08, 0.94,0.66,-0.46};
        + (y[2] - y[0]) / (x[2]-x[0]);
        ypn = (y[n-1] - y[n-2]) / (x[n-1]-x[n-2]) - (y[n-2]-y[n-3])
             / (x[n-2] - x[n-3]) + (y[n-1] - y[n-3]) / (x[n-1] - x[n-3]);
        y2[0] = u[0] = 0. ;                                       // Natural
        for ( i=1;  i <= n-2;  i++ )  {      …}                 // Decomposition loop
        qn = un = 0. ;                                                    // Natural
        y2[n-1] = (un-qn*u[n-2]) / (qn*y2[n-2] + 1.);
        for ( k = n-2;  k>= 0; k--)  y2[k] = y2[k] * y2[k + 1] + u[k];
        for ( i=1;  i <= Nfit;  i++ ) {                           }     // initialization
        xout = x[0] + ( x[n-1] - x[0] ) * (i - 1) / (Nfit);
        ...  
        yout = ( a*y[klo] + b*y[khi] + ( (a*a*a-a)*y2[klo]
               + ( b*b*b - b ) * y2[khi]) * (h*h)/6. );
    } } } }
Go try these out!