Interpolation & Approximation
Lecture Notes

BM 531
Numerical Methods and C/C++ Programming

Ahmet Ademoglu, *PhD*
Bogazici University
Institute of Biomedical Engineering

Linear Regression

$y_i = ax_i + b_i$ for $i = 1, \ldots, n$

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} y_1 & 1 \\ y_2 & 1 \\ \vdots & \\ y_n & 1 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} + \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_n \end{bmatrix}, \qquad \mathbf{y} = \mathbf{A}\alpha + \epsilon$$

Least Squares Solution

$$\frac{\partial |\epsilon|^2}{\partial a} = \frac{\partial |\epsilon|^2}{\partial b} = 0$$

$$\frac{\partial |\epsilon|^2}{\partial a} = \mathbf{x}^T\mathbf{y} - \mathbf{y}^T\mathbf{x} + \mathbf{x}^T(\mathbf{A}\alpha) + (\mathbf{A}\alpha)^T\mathbf{x} = 0 \rightarrow \mathbf{x}^T(\mathbf{A}\alpha) = \mathbf{x}^T\mathbf{y}$$

$$\frac{\partial |\epsilon|^2}{\partial b} = \mathbf{1}^T\mathbf{y} - \mathbf{y}^T\mathbf{1} + \mathbf{1}^T(\mathbf{A}\alpha) + (\mathbf{A}\alpha)^T\mathbf{1} = 0 \rightarrow \mathbf{1}^T(\mathbf{A}\alpha) = \mathbf{1}^T\mathbf{y}$$

$$\mathbf{A}^T\mathbf{A}\alpha = \mathbf{A}^T\mathbf{y}$$

Multivariable Linear Regression

$$\begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_n \end{bmatrix} = \begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ \vdots & \vdots & \vdots \\ x_n & y_n & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} + \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_n \end{bmatrix}, \qquad \mathbf{z} = \mathbf{A}\alpha + \epsilon$$

Polynomial Regression

$$y_i = a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n$$

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} 1 & x_1 & x_1^2 \ldots x_1^n \\ 1 & x_2 & x_2^2 \ldots x_2^n \\ & \vdots & \vdots \quad \vdots \\ 1 & x_n & x_n^2 \ldots x_n^n \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix} + \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_n \end{bmatrix},$$

$$\mathbf{y} = \mathbf{A}\alpha + \epsilon$$

Nonlinear Regression

$$y = f(x_i; a_0, a_1, \ldots, a_m)$$

$$f(x_i)_{j+1} = f(x_i)_j + \partial f(x_i)_j/\partial a_0 \Delta a_0 + \partial f(x_i)_j/\partial a_1 \Delta a_1 + \ldots$$

$$\begin{bmatrix} y_1 - f(x_1)_j \\ y_2 - f(x_2)_j \\ \vdots \\ y_n - f(x_n)_j \end{bmatrix} =$$

$$\begin{bmatrix} \partial f(x_1)_j/\partial a_0 & \partial f(x_1)_j/\partial a_1 \\ \vdots & \vdots \\ \partial f(x_n)_j/\partial a_0 & \partial f(x_n)_j/\partial a_1 \end{bmatrix} \begin{bmatrix} \Delta a_0 \\ \\ \Delta a_1 \end{bmatrix} + \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_n \end{bmatrix},$$

$$\mathbf{y} = \mathbf{A}\alpha + \epsilon$$

Example : $f(x) = a_0(1 - e^{-a_1 x})$

Initial Conditions

$a_0 = a_1 = 0.1$

$\partial f(x)/\partial a_0 = (1 - e^{-a_1 x})$

$\partial f(x)/\partial a_1 = a_0 a_1 (e^{-a_1 x})$

1. Start with j=0 and inital a_0 and a_1
2. Apply least squares to find Δa_0 and Δa_1
3. Update $(a_0)_{j+1} = (a_0)_j + (\Delta a_0)_j$
4. Iterate until convergence

Interpolation

Taylor Approximation : A polynomial approximation

$f(x-x_0) = f(x_0) + f'(x_0)(x-x_0)/1! + f''(x_0)(x-x_0)^2/2! + \ldots =$
$a_0 + a_1(x-x_0) + a_2(x-x_0)^2 + \ldots$
n^{th} order polynomial fits to $n+1$ points

Newton's Interpolating Polynomials

- **Linear Interpolation**

$f(x) = a_0 + a_1 x$
$f(x_1) = a_0 + a_1 x_1$ and $f(x_2) = a_0 + a_1 x_2$
$a_0 = f(x_1) - x_1(f(x_2) - f(x_1))/(x_2 - x_1)$ and
$a_1 = (f(x_2) - f(x_1))/(x_2 - x_1)$
$f(x) = f(x_1) + (x - x_1)(f(x_2) - f(x_1))/(x_2 - x_1)$

- **Quadratic Polynomial**

$f(x) = a_0 + a_1 x^2 + a_2 x^2 = b_0 + b_1(x - x_0) + b_2(x - x_0)(x - x_1)$

$b_0 = f(x_0)$ $b_1 = (f(x_1) - f(x_0))/(x_1 - x_0)$ and

$b_2 = \frac{(f(x_2) - f(x_0))/(x_2 - x_0) - (f(x_1) - f(x_0))/(x_1 - x_0)}{(x_2 - x_1)}$

$f(x) = b_0 + b_1(x - x_0) + b_2(x - x_0)(x - x_1) + \ldots + b_n(x - x_0)(x - x_1)\ldots(x - x_{n-1})$

$b_0 = f(x_0)$

$b_1 = f[x_1, x_0] = (f(x_1) - f(x_0))/(x_1 - x_0)$

$b_2 = f[x_2, x_1, x_0] = f[x_2, x_1] - f[x_1, x_0]/(x_2 - x_0)$

$\vdots$

$b_n = f[x_n, x_{n-1}, \ldots x_1, x_0] =$

$f[x_n, x_{n-1}, \ldots x_1] - f[x_{n-1}, x_{n-2}, \ldots x_0]/(x_n - x_0)$

- **Lagrange Interpolating Polynomials**

  Linear Case

  $$f(x) = \frac{(x-x_1)}{x_0-x_1} f(x_0) + \frac{(x-x_0)}{x_1-x_0} f(x_1)$$

  Quadratic Case

  $$f(x) = \frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)} f(x_0) + \frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)} f(x_1) + \frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)} f(x_2)$$

- **Interpolating with Cubic Splines :**

  $i = 0, 1 \ldots n$: $n+1$ data points with n intervals

  $f_i(x) = a_i x^3 + b_i x^2 + c_i x + d_i$: $4n$ unknowns

  1. $f(x_i) = f_i(x_i)$ at all the knots. $(n+1)$ conditions.
  2. The $f_i(x_i) = f_{i+1}(x_i)$ at the internal knots. $(n-1)$ conditions.
  3. $f_i'(x_i) = f_{i+1}'(x_i)$ at the internal knots. $(n-1)$ conditions.
  4. $f_i''(x_i) = f_{i+1}''(x_i)$ at the internal knots. $(n-1)$ conditions.
  5. The $f_i''(x_0) = f''(x_n) = 0$ at the end nodes. 2 conditions.

  Differentiate f twice and express it with 1st order Lagrange Interpolating Polynomials in the interval i:

  $$f_i''(x) = 6a_i x + 2b_i = f''(x_{i-1})\frac{(x-x_i)}{x_{i-1}-x_i} + f''(x_i)\frac{(x-x_{i-1})}{x_i-x_{i-1}}$$

Integrate it twice

$$f_i(x) = f''(x_{i-1})\frac{(x-x_i)^3}{6(x_{i-1}-x_i)} + f''(x_i)\frac{(x-x_{i-1})^3}{6(x_i-x_{i-1})} + c_1 x + c_2$$

The spline functions are evaluated at the interval knot points i.e. $f_i(x_{i-1})$ and $f_i(x_i)$ by which

c_1 and c_2 are solved in terms of $f''(x_i)$ and $f''(x_{i-1})$ using

$$f_i(x_{i-1}) = f''(x_{i-1})\frac{(x_{i-1}-x_i)^2}{6} + c_1 x_{i-1} + c_2$$

$$f_i(x_i) = f''(x_i)\frac{(x_i-x_{i-1})^2}{6} + c_1 x_i + c_2$$

Finally

$$f_i(x) = f''(x_{i-1})\frac{(x-x_i)^3}{6(x_{i-1}-x_i)} + f''(x_i)\frac{(x-x_{i-1})^3}{6(x_i-x_{i-1})} +$$
$$[\frac{f(x_{i-1})}{x_i-x_{i-1}} - \frac{f''(x_{i-1})(x_i-x_{i-1})}{6}](x_i - x) +$$
$$[\frac{f(x_i)}{x_i-x_{i-1}} - \frac{f''(x_i)(x_i-x_{i-1})}{6}](x - x_{i-1})$$

To determine the $f''(x_i)$ and $f''(x_{i-1})$ $f_i(x)$ is differentiated once and the condition that $f'_{i-1}(x_i) = f'_i(x_i)$ is used ($n-1$ equations):

$$(x_i - x_{i-1})f''(x_{i-1}) + 2(x_{i+1} - x_{i-1})f''(x_i) + (x_{i+1} - x_i)f''(x_{i+1}) = \frac{6}{(x_{i+1}-x_i)}[f(x_{i+1} - f(x_i))] + \frac{6}{(x_i-x_{i-1})}[f(x_{i-1} - f(x_i))]$$

The f'' is assumed to be 0 at the end knot points, therefore $n+1$ unknown $f''(x_i)$s and $n+1$ equations.

Example

$x_0 = 3, f(x_0) = 2.5$

$x_1 = 4.5, f(x_1) = 1$

$x_2 = 7, f(x_2) = 2.5$

$x_3 = 9, f(x_3) = 0.5$

$$(4.5-3)f''(3)+2(7-3)f''(4.5)+(7-4.5)f''(7) = \frac{6(2.5-1)}{(7-4.5)} + \frac{6(2.5-1)}{4.5-3}$$

$$(7-4.5)f''(4.5)+2(9-4.5)f''(7)+(9-7)f''(9) = \frac{6(0.5-2.5)}{(9-7)} + \frac{6(1-2.5)}{(7-4.5)}$$

$f''(3) = 0$

$f''(9) = 0$

Numerical Recipes in C

Construct a cubic spline

```
void spline(x,y,n,yp1,ypn,y2)
float x[],y[],yp1,ypn,y2[];
int n;
```

Input :

$x[1..n]$ and $y[1..n]$ i.e., $y_i = f(x_i)$ with $x_1 < x_2 < \ldots < x_N$,
$yp1$ and ypn : the 1^{st} derivative of the interpolating function at points 1 and n

Output :

$y2[1..n]$ the second derivatives of the interpolating function x_i
If $yp1$ and/or ypn are equal to 1×10^{30} or larger, a natural spline with zero second derivative on that boundary is assumed.

Cubic spline interpolation `void splint(xa,ya,y2a,n,x,y)`

```
float xa[],ya[],y2a[],x,*y;
int n;
```

Input:

$xa[1..n]$ and $ya[1..n]$, i.e., $y_i = f(x_i)$ with $x_1 < x_2 < \ldots < x_N$,

$y2a[1..n]$ the output from spline routine

x : an input value for interpolating $f(x)$

Output :

y: a cubic-spline interpolated value

`splie2` : construct two-dimensional spline

`splin2`: two-dimensional spline interpolation