Interpolation and Curve Fitting

Scientific Computing 372

W. H. K. Bester

University of Stellenbosch

Last updated on Tuesday 31st July, 2018 at 11:39

Introduction

Interpolation

- Determining a function that exactly represents a collection of data points
- Why?
  - Determine values at intermediate points
  - Approximate integral or derivative of underlying function
  - Give a smooth or continuous representation of variables in a problem

Weierstrass Approximation Theorem

Suppose that f is defined and continuous on $[a, b]$. For each $\epsilon > 0$, there exists a polynomial $P(x)$ defined on $[a, b]$, with the property that

$$|f(x) - P(x)| < \epsilon, \qquad \text{for all } x \in [a, b].$$

Taylor polynomials

Taylor's Theorem

Suppose $f = C^n[a, b]$ and $f^{(n+1)}$ exists on $[a, b]$. Let x_0 be a number in $[a, b]$. For every x in $[a, b]$, there exists a number $\xi(x)$ between x_0 and x with

$$f(x) = P_n(x) + R_n(x),$$

where

$$P_n(x) = \sum_{k=0}^{n} \frac{f^{(k)}(x_0)}{k!}(x - x_0)^k \quad \text{and} \quad R_n(x) = \frac{f^{(n+1)}(\xi(x))}{(n+1)!}(x - x_0)^{n+1}.$$

Taylor polynomials

- Agree as closely as possible with function at a specific point
- But concentrate accuracy only near that point

Taylor polynomials

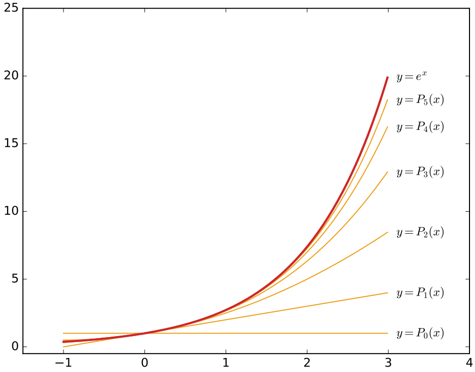


Figure: Python plot of e^x approximated by Taylor polynomials.

However, it is not always true that higher-degree Taylor polynomials lead to better approximations. . . .

Lagrange polynomials

The idea

- Determine an approximating polynomial by specifying the points through which it must pass
- Determining a first-degree polynomial that passes through distinct points (x_0, y_0) and (x_1, y_1) is the same as finding a function f such that $f(x_0) = y_0$ and $f(x_1) = y_1$
- Define the functions

$$\ell_0 = \frac{x - x_1}{x_0 - x_1} \quad \text{and} \quad \ell_1 = \frac{x - x_0}{x_1 - x_0},$$

  and note that $\ell_0(x_0) = 1$, $\ell_0(x_1) = 0$, $\ell_1(x_0) = 0$, and $\ell_1(x_1) = 1$
- Now, define

$$P(x) = \ell_0(x)f(x_0) + \ell_1(x)f(x_1),$$

  which gives

$$P(x_0) = y_0 \quad \text{and} \quad P(x_1) = y_1$$

- P is the unique linear function passing through (x_0, y_0) and (x_1, y_1)

Generalising Lagrange polynomials

Find the *unique* polynomial of degree n that passes through $n+1$ distinct points $(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)$:

$$P_n(x) = \sum_{i=0}^{n} y_i \ell_i(x), \tag{1}$$

where the cardinal functions are

$$\ell_i(x) = \frac{x - x_0}{x_i - x_0} \cdot \frac{x - x_1}{x_i - x_1} \cdots \frac{x - x_{i-1}}{x_i - x_{i-1}} \cdot \frac{x - x_{i+1}}{x_i - x_{i+1}} \cdots \frac{x - x_n}{x_i - x_n} = \prod_{j=0, j\neq i}^{n} \frac{x - x_j}{x_i - x_j} \tag{2}$$

for $i = 0, 1, \ldots, n$. Since

$$\ell_i(x_j) = \left\{ \begin{matrix} 0 & \text{if } i \neq j \\ 1 & \text{if } i = j \end{matrix} \right\} = \delta_{ij} \qquad \text{Kronecker delta} \tag{3}$$

we have

$$P_n(x_j) = \sum_{i=0}^{n} y_i \ell_i(x_j) = \sum_{i=0}^{n} y_i \delta_{ij} = y_j. \tag{4}$$

Newton polynomials

Problem with Lagrange's method

Langrange's method is conceptually simple, but does not lead to an efficient algorithm.

Newton's method

Use an interpolating polynomial of the form

$$\begin{aligned} P_n(x) &= a_0 + \underbrace{(x - x_0)}_{p_1(x)} a_1 + \underbrace{(x - x_0)(x - x_1)}_{p_2(x)} a_2 + \cdots \\ &\quad + \underbrace{(x - x_0)(x - x_1)\cdots(x - x_{n-1})}_{p_n(x)} a_n \\ &= \sum_{i=0}^{n} a_i p_i(x), \end{aligned}$$

a *linear combination* of the Newton basis polynomials,

$$p_i(x) = \begin{cases} \prod_{j=0}^{i-1}(x - x_j) & \text{if } i > 0, \\ 1 & \text{if } i = 0. \end{cases}$$

Newton polynomials

Example

For $n = 3$, we have

$$
\begin{aligned}
P_3(x) &= a_0 + (x - x_0)a_1 + (x - x_0)(x - x_1)a_2 + (x - x_0)(x - x_1)(x - x_2)a_3 \\
&= \underbrace{a_0 + (x - x_0)\underbrace{\left(a_1 + (x - x_1)\underbrace{\left(a_2 + (x - x_2)\underbrace{a_3}_{P_0(x)}\right)}_{P_1(x)}\right)}_{P_2(x)}}_{P_3(x)},
\end{aligned}
$$

which can be evaluated with with the following recurrence equations:

$$
\begin{aligned}
P_0(x) &= a_3 \\
P_1(x) &= a_2 + (x - x_2)P_0(x) \\
P_2(x) &= a_1 + (x - x_1)P_1(x) \\
P_3(x) &= a_0 + (x - x_0)P_2(x)
\end{aligned}
$$

Newton polynomials

General recurrence equations

For arbitrary n, we have

$$P_k(x) = \begin{cases} a_n & \text{for } k = 0, \\ a_{n-k} + (x - x_{n-k})P_{k-1}(x) & \text{for } k = 1, 2, \ldots, n. \end{cases} \tag{5}$$

Coefficients of P_n

Determine coefficients by forcing the polynomial through each data point $y_i = P_n(x_i)$, for $i = 0, 1, \ldots, n$, yields the simultaneous equations

$$\begin{aligned} y_0 &= a_0, \\ y_0 &= a_0 + (x_1 - x_0)a_1, \\ y_0 &= a_0 + (x_2 - x_0)a_1 + (x_2 - x_0)(x_2 - x_1)a_2, \\ &\vdots \\ y_n &= a_0 + (x_n - x_0)a_1 + \ldots + (x_n - x_0)(x_n - x_1)\cdots(x_n - x_{n-1})a_n, \end{aligned} \tag{6}$$

Newton polynomials

Divided differences

$$
\begin{aligned}
\nabla y_i &= \frac{y_i - y_0}{x_i - x_0}, \quad i = 1, 2, \ldots, n \\
\nabla^2 y_i &= \frac{\nabla y_i - \nabla y_1}{x_i - x_1}, \quad i = 2, 3, \ldots, n \\
\nabla^3 y_i &= \frac{\nabla^2 y_i - \nabla^2}{x_i - x_2}, \quad i = 3, 4, \ldots, n \\
&\vdots \\
\nabla^n y_n &= \frac{\nabla^{n-1} y_n - \nabla^{n-1} y_{n-1}}{x_n - x_{n-1}}
\end{aligned}
$$

Solutions to simultaneous Eqs. (6)

$$
a_0 = y_0, \quad a_1 = \nabla y_1, \quad a_2 = \nabla^2 y_2, \quad \ldots \quad a_n = \nabla^n y_n
$$

Newton polynomials

Table: Tableau for Newton's method for $n = 4$

x_0	y_0				
x_1	y_1	∇y_1			
x_2	y_2	∇y_2	$\nabla^2 y_2$		
x_3	y_3	∇y_3	$\nabla^2 y_3$	$\nabla^3 y_3$	
x_4	y_4	∇y_4	$\nabla^2 y_4$	$\nabla^3 y_4$	$\nabla^4 y_4$

Use one-dimensional array in Python

```
as = ys.copy()
for k in range(1, m):
    for i in range(k, m):
        as[i] = (a[i] - a[k-1])/(xs[i] - xs[k-1])
```

Cubic splines

Problem

- Polynomial interpolation is “smooth”
- But they can oscillate “wildly”

Piecewise polynomial approximation

- Construct a different approximating polynomial on each subinterval
- Involve the derivatives:
  - Imagine a thin, elastic beam is passed through the points
  - The slope (and hence first derivative) is continuous
  - The bending moment (and hence second derivative) is continuous
- No bending moment at endpoints; second derivative zero
- Resulting curve is known as the natural cubic spline
- The data points are called the knots

Cubic splines

Notation

- Use $f_{i,i+1}(x)$ for the cubic polynomial between knots i and $i+1$
- Use k_i for the second derivative at knot i.

Second derivatives

- Continuity of second derivatives, so

$$f''_{i-1,i}(x_i) = f''_{i,i+1}(x_i) = k_i$$

- Each k is still unknown, except for $k_0 = k_n = 0$
- We know the second derivative is linear, so

$$f''_{i,i+1}(x) = k_i \ell_i(x) + k_{i+1} \ell_{i+1}(x)$$

where, using Lagrange's two-point interpolation,

$$\ell_i(x) = \frac{x - x_{i+1}}{x_i - x_{i+1}}$$

Cubic splines

- So, we have

$$f''_{i,i+1}(x) = \frac{k_i(x - x_{i+1}) - k_{i+1}(x - x_i)}{x_i - x_{i+1}}$$

- Integrate twice with respect to x:

$$f_{i,i+1}(x) = \frac{k_i(x - x_{i+1})^3 - k_{i+1}(x - x_i)^3}{6(x_i - x_{i+1})} + A(x - x_{i+1}) - B(x - x_i) \qquad (7)$$

  where A and B are constants of integration, but instead of writing $Cx + D$, we let $C = A - B$ and $D = -Ax_{i+1} + Bx_i$
- Imposing the condition $f_{i,i+1}(x_i) = y_i$, from Eq. (7):

$$\frac{k_i(x_i - x_{i+1})^3}{6(x_i - x_{i+1})} + A(x_i - x_{i+1}) = y_i$$

- Therefore,

$$A = \frac{y_i}{x_i - x_{i+1}} - \frac{k_i}{6}(x_i - x_{i+1}) \qquad (8)$$

- Similarly, $f_{i,i+1}(x_{i+1}) = y_{i+1}$ yields

$$B = \frac{y_{i+1}}{x_i - x_{i+1}} - \frac{k_{i+1}}{6}(x_i - x_{i+1}) \qquad (9)$$

Cubic splines

- Substituting Eqs. (8) and (9) into Eq. (7):

$$
\begin{aligned}
f_{i,i+1}(x) = \frac{k_i}{6}\left(\frac{(x-x_{i+1})^3}{x_i-x_{i+1}} - (x-x_{i+1})(x_i-x_{i+1})\right) \\
- \frac{k_i}{6}\left(\frac{(x-x_i)^3}{x_i-x_{i+1}} - (x-x_i)(x_i-x_{i+1})\right) \\
+ \frac{y_i(x-x_{i+1}) - y_{i+1}(x-x_i)}{x_i-x_{i+1}}
\end{aligned}
\qquad (10)
$$

- The second derivatives k_i at the interior knots are obtained from the slope continuity conditions $f'_{i-1,i}(x_i) = f'_{i,i+1}(x_i)$ for $i = 1, 2, \ldots, n-1$
- After some algebra, this results in the simultaneous equations:

$$
\begin{aligned}
&k_{i-1}(x_{i-1}-x_i) + 2k_i(x_{i-1}-x_{i+1}) + k_{i+1}(x_i-x_{i+1}) \\
&=6\left(\frac{y_{i-1}-y_i}{x_{i-1}-x_i} - \frac{y_i-y_{i+1}}{x_i-x_{i+1}}\right), \qquad i = 1, 2, \ldots, n-1
\end{aligned}
$$

- These equations have a tridiagonal matrix, and we can solve them with a bit of linear algebra

Interlude: LU decomposition

Basic idea

- For the matrix equation $A\mathbf{x} = \mathbf{b}$, decompose (or factorise) A into the product of a lower triangular matrix L and an upper triangular matrix U
- We then rewrite $A\mathbf{x} = \mathbf{b}$ as $LU\mathbf{x} = \mathbf{b}$
- Let $U\mathbf{x} = \mathbf{y}$
- Solve $L\mathbf{y} = \mathbf{b}$ for $\mathbf{y}$ by forward substitution
- Solve $U\mathbf{x} = \mathbf{y}$ for $\mathbf{x}$ by back substitution

Better than Gaussian elimination?

The substitution process is much less time consuming than the decomposition process. So, once A is decomposed, we can solve $A\mathbf{x} = \mathbf{b}$ for as many constant vectors $\mathbf{b}$ as we want.

Name	**Constraints**
Doolittle	$\ell_{ii} = 1$ for $i = 1, 2, \ldots, n$
Crout	$u_{ii} = 1$ for $i = 1, 2, \ldots, n$
Choleski	$L = U^T$

Interlude: LU decomposition

Example (Doolittle)

- Consider $A = LU$, where

$$L = \begin{bmatrix} 1 & 0 & 0 \\ \ell_{21} & 1 & 0 \\ \ell_{31} & \ell_{32} & 1 \end{bmatrix} \quad U = \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix}$$

- Complete the multiplication

$$A = \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ u_{11}\ell_{21} & u_{12}\ell_{21} + u_{22} & u_{13}\ell_{21} + u_{23} \\ u_{11}\ell_{31} & u_{12}\ell_{31} + u_{22}\ell_{32} & u_{13}\ell_{31} + u_{23}\ell_{32} + u_{33} \end{bmatrix} \tag{11}$$

- Apply Gauss elimination to Eq. (11):
  row 2 ← row 2 − ℓ_{21} × row 1 (Eliminates a_{21})
  row 3 ← row 3 − ℓ_{31} × row 1 (Eliminates a_{31})

$$A' = \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & u_{22}\ell_{32} & u_{23}\ell_{32} + u_{33} \end{bmatrix}$$

Interlude: LU decomposition

Example (Doolittle—continued)

- Apply row 3 ← row 3 − ℓ_{32} × row 2 to eliminate a_{32}

$$A'' = U = \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix}$$

- The matrix U is identical to the upper triangular matrix that results from Gaussian elimination
- The off-diagonal elements of L are the pivot equation multipliers used during Gaussian elimination: ℓ_{ij} is the multiplier that eliminated a_{ij}
- The usual practice is to store the multipliers in the lower triangular portion of the coefficient matrix, replacing a_{ij} by ℓ_{ij}

$$[L\backslash U] = \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ \ell_{21} & u_{22} & u_{23} \\ \ell_{31} & \ell_{32} & u_{33} \end{bmatrix} \tag{12}$$

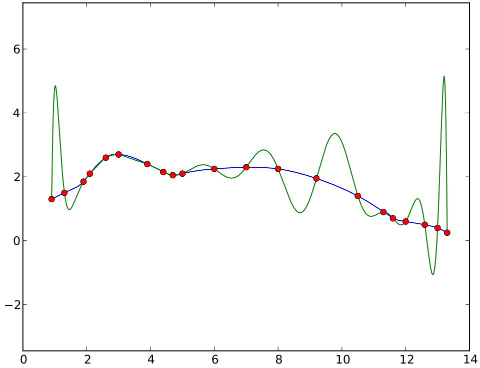
Interlude: Tridiagonal coefficient matrix

- Consider the solution of $A\mathbf{x} = \mathbf{b}$ by Doolittle's composition where A is $n \times n$ tridiagonal matrix

$$A = \begin{bmatrix} d_1 & e_1 & 0 & 0 & \cdots & 0 \\ c_1 & d_2 & e_2 & 0 & \cdots & 0 \\ 0 & c_2 & d_3 & e_3 & \cdots & 0 \\ 0 & 0 & c_3 & d_4 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & 0 & c_{n-1} & d_n \end{bmatrix}, c = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_{n-1} \end{bmatrix}, d = \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_{n-1} \\ d_n \end{bmatrix}, e = \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_{n-1} \end{bmatrix}$$

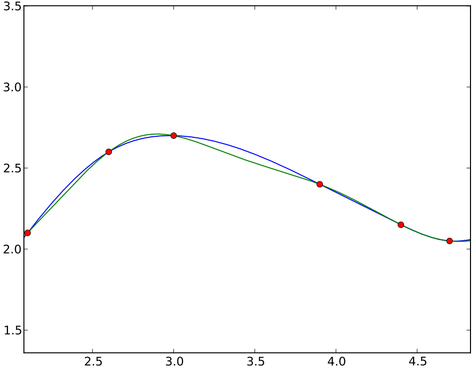
- Apply LU decomposition to the coefficient matrix:
  row k ← row k $-$ $(c_{k-1}/d_{k-1}) \times$ row $(k-1)$ for $k = 2, 3, \ldots, n$
- The corresponding change in d_k is (e_k is not affected):
  $d_k \leftarrow d_k - (c_{k-1}/d_{k-1})e_{k-1}$
- Store the multiplier $\lambda = c_{k-1}/d_{k-1}$ in the location previously occupied by c_{k-1}

Polynomial interpolation v. spline



Plot of a cubic spline (in blue) and a degree 20 polynomial (in green) for 21 data points (in red)

Polynomial interpolation v. spline



Zoomed into the previous comparative plot

- For both plots, the slopes are continuous at each point
- The spline follows the curvature "suggested" by the data

Least-squares fit

- Data obtained from experiments typically contain a significant amount of random noise due to measurement errors
- Curve fitting finds a smooth curve that fits the data points "on average"
- The curve should not reproduce the noise, so it should have a simple form, e.g., a low-order polynomial
- Let

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

  be the function to be fitted to the $n + 1$ data points (x_i, y_i) for $i = 0, 1, \ldots, n$
- We have a function of x that contains $m + 1$ variable parameters $a_0, a_1, \ldots, a_m$, where $m < n$
- Curve fitting consists of two steps:
  - First, choose the form of $f(x)$, usually from theory associated with the experiment
  - Then, compute that parameters that best fit the data

Least-squares fit

- Minimise the function S with respect to each a_j

$$S(a_0, a_1, \ldots, a_m) = \sum_{i=0}^{n} [y_i - f(x_i)]^2 \tag{13}$$

The terms $r_i = y_i - f(x_i)$ are called the residuals: the discrepancies between the data points and the fitting function at x_i

- The optimal parameter values are given by solutions to

$$\frac{\partial S}{\partial a_k} = 0 \quad \text{for } k = 0, 1, \ldots, m \tag{14}$$

Generally, Eqs. (14) are nonlinear in a_j, and thus, difficult to solve—so, often, the fitting function is chosen to be a linear combination of specified functions $f_j(x)$, called the basis functions:

$$f(x) = a_0 f_0(x) + a_1 f_1(x) + \cdots + a_m f_m(x) = \sum_{j=0}^{m} a_j f_j(x) \tag{15}$$

Standard deviation

- Quantify the spread of the data about the fitting curve by the standard deviation

$$\sigma = \sqrt{\frac{S}{n-m}} \tag{16}$$

- If $n = m$, we have interpolation, not curve fitting
- In this case, both the numerator and the denominator in Eq. (16) are zero, so that σ is indeterminate

Linear regression

- Fitting to a straight line

$$f(x) = a + bx$$

- The function to be minimised is

$$S(a, b) = \sum_{i=1}^{n} [y_i - f(x_i)]^2 = \sum_{i=0}^{n} (y_i - a - bx_i)^2$$

- So, Eqs. (14) become

$$\frac{\partial S}{\partial a} = \sum_{i=0}^{n} -2(y_i - a - bx_i) = 2\left[a(n+1) + b\sum_{i=0}^{n} x_i - \sum_{i=0}^{n} y_i\right] = 0$$

$$\frac{\partial S}{\partial b} = \sum_{i=0}^{n} -2(y_i - a - bx_i)x_i = 2\left(a\sum_{i=0}^{n} x_i + b\sum_{i=0}^{n} x_i^2 - \sum_{i=0}^{n} x_i y_i\right) = 0$$

Linear regression

- Divide both by $2(n+1)$ and rearrange terms to get

$$a + \bar{x}b = \bar{y} \quad \text{and} \quad \bar{x}a + \left(\frac{1}{n+1}\sum_{i=0}^{n} x_i^2\right)b = \frac{1}{n+1}\sum_{i=0}^{n} x_i y_i$$

  where

$$\bar{x} = \frac{1}{n+1}\sum_{i=0}^{n} x_i \quad \text{and} \quad \bar{y} = \frac{1}{n+1}\sum_{i=0}^{n} y_i$$

  are the mean values of the x and y data, respectively
- The parameter solutions are

$$a = \frac{\bar{y}\sum x_i^2 - \bar{x}\sum x_i y_i}{\sum x_i^2 - n\bar{x}^2} \quad \text{and} \quad b = \frac{\sum x_i y_i - \bar{x}\sum y_i}{\sum x_i^2 - n\bar{x}^2}$$

- But these are susceptible to round-off errors, so we use

$$b = \frac{\sum y_i(x_i - \bar{x})}{\sum x_i(x_i - \bar{x})} \quad \text{and} \quad a = \bar{y} - \bar{x}b \qquad (17)$$

Fitting linear forms

- Substitute Eq. (15) into Eqs. (13) to yield:

$$S = \sum_{i=0}^{n} \left[y_i - \sum_{j=0}^{m} a_j f_j(x_i) \right]^2$$

- Thus, Eqs. (14) are

$$\frac{\partial S}{\partial a_k} - 2 \left\{ \sum_{i=0}^{n} \left[y_i - \sum_{j=0}^{m} a_i f_j(x_i) \right] f_k(x_i) \right\} = 0 \quad \text{for } k = 0, 1, \ldots, m$$

- Dropping the constant and interchanging the order of summation:

$$\sum_{j=0}^{m} \left[\sum_{i=0}^{n} f_j(x_i) f_k(x_i) \right] a_j = \sum_{i=0}^{n} f_k(x_i) y_i \quad \text{for } k = 0, 1, \ldots, m \tag{18}$$

- In matrix notation, Eqs. (18) are

$$A\mathbf{a} = \mathbf{b} \tag{19}$$

where the entries of A and $\mathbf{b}$ are

$$A_{kj} = \sum_{i=0}^{n} f_j(x_i) f_k(x_i) \quad \text{and} \quad b_k = \sum_{i=0}^{n} f_k(x_i) y_i \tag{20}$$

Polynomial fit

- Fit a polynomial of degree m, so $f(x) = \sum_{j=0}^{m} a_j x^j$ and the basis functions are

$$f_j(x) = x^j \quad \text{for } j = 0, 1, \ldots, m$$

- Eqs. (20) become

$$A_{kj} = \sum_{i=0}^{n} x_i^{j+k} \quad \text{and} \quad b_k = \sum_{i=0}^{n} x_i^k y_i$$

or

$$A = \begin{bmatrix} n & \sum x_i & \sum x_i^2 & \cdots & \sum x_i^m \\ \sum x_i & \sum x_i^2 & \sum x_i^3 & \cdots & \sum x_i^{m+1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \sum x_i^{m-1} & \sum x_i^m & \sum x_i^{m+1} & \ldots & \sum x_i^{2m} \end{bmatrix}, \mathbf{b} = \begin{bmatrix} \sum y_i \\ \sum x_i y_i \\ \ldots \\ \sum x_i^m y_i \end{bmatrix} \tag{21}$$

- Eqs. (19) are known as the normal equations of the least-squares fit
- They can be solved with linear algebra, e.g., Gaussian elimination

Polynomial fit

Important

- The normal equations become progressively ill conditioned with increasing m
- Small changes in the argument have a progressively larger effect on the function value
- Fortunately, only low-order polynomials are useful in curve-fitting
- High-order polynomials tend to reproduce the noise in the data

Implementation in Python

- Store the terms that make up the coefficient matrix in Eqs. (21) in the vector $\mathbf{s}$
- Then insert them into A
- Solve the normal equations

Weighted linear regression

- Confidence in the accuracy of the data may vary from point to point
- E.g., the measuring device may be more sensitive in certain ranges
- So, we assign a confidence factor, the weight, to each data point
- Minimise the sum of the squares of the weighted residuals
  $r_i = W_i[y_i - f(x_i)]$, where W_i are the weights
- For a straight line $f(x) = a + bx$, minimise

$$S(a, b) = \sum_{i=0}^{n} W_i^2 (y_i - a - bx_i)^2$$

- The solutions for the parameters are

$$a = \hat{y} - b\hat{x} \quad \text{and} \quad b = \frac{\sum W_i^2 y_i (x_i - \hat{x})}{\sum W_i^2 x_i (x_i - \hat{x})} \qquad (22)$$

with the weighted averages

$$\hat{x} = \frac{\sum W_i^2 x_i}{\sum W_i^2} \quad \text{and} \quad \hat{y} = \frac{\sum W_i^2 y_i}{\sum W_i^2}$$

Fitting exponential functions

- For $f(x) = ae^{bx}$, least-squares fit would normally lead to equations nonlinear in a and b
- Transform to linear regression, by fitting the function

$$F(x) = \ln f(x) = \ln a + bx$$

  to the data points $(x_i, \ln y_i)$
- It's not quite same, however; compare the residuals of the logarithmic fit

$$R_i = \ln y_i - F(x_i) = \ln y_i - (\ln a + bx_i) \tag{23}$$

  to those of the oAriginal expression

$$r_i = y_i - f(x_i) = y_i - ae^{bx_i} \tag{24}$$

- From Eq. (24), we have $\ln(r_i - y_i) = \ln(ae^{bx_i}) = \ln a + bx_i$, so Eq. (23) can be written

$$R_i = \ln y_i - \ln(r_i - y_i) = \ln\left(1 - \frac{r_i}{y_i}\right)$$

Fitting exponential functions

- If $r_i \ll y_i$, use the approximation $\ln(1 - r_i/y_i) \approx r_i/y_i$, so that $R_i \approx r_i/y_i$
- By minimising $\sum R_i^2$, we have inadvertantly introduced the weights $1/y_i$
- Negate this effect by applying weights $W_i = y_i$, so minimising

$$S = \sum_{i=0}^{n} y_i^2 R_i^2$$

is a good approximation to minimising $\sum r_i^2$

Table: Fittings that benefit from using weights $W_i = y_i$

$f(x)$	$F(x)$	Data to be fitted by $F(x)$
ae^{bx}	$\ln f(x) = \ln a + bx$	$(x_i, \ln y_i)$
axe^{bx}	$\ln[f(x)/x] = \ln a + bx$	$(x_i, \ln(y_i/x_i))$
ax^b	$\ln f(x) = \ln a + b \ln x$	$(\ln x_i, \ln y_i)$