Numerical Methods

EE1103

Indian Institute of Technology, Madras

February 9, 2021

References

- Numerical Methods for Engineers, Chapra and Canale, 6th edition
- onlinegdb.com
- www.geeksforgeeks.org/c-programming-language/
- Linux cheat sheet

Intro to Linux

1. Its an **Operating System**, like Windows or MacOS
2. It is **NOT** a programming language
3. It has different programs like Terminal, OpenOffice or Libre Office, Gnuplot,

There are many variants of Linux. The most popular ones being **Ubuntu, Redhat and Debian**

You can install **VirtualBox** and run multiple OS'es without having to reboot your computer.

Numerical Methods

EE1103

The Linux Terminal

- The Terminal runs a **Shell** (default is likely to be **bash**)
- You can write bash-scripts that do interesting things
- You need to type commands at the prompt within a Terminal e.g.
  - **clear** will clear the terminal screen
  - **mkdir, cd, pwd**, and **popd/pushd** help you navigate within the folders/directories,
  - **ls** will list the files within a folder, **mv** will move them
  - **history** will give you the list of commands
    - you could also use e.g. **!3** or **!m** to recall commands from the history

The Linux Terminal

- You must learn to work with processes
  - ps (lists the processes, fg (foreground) or bg (background) allows you to control their execution
  - Ctrl-C : kill a running program Ctrl-Z: suspend a running program, restart with fg or bg
- Use the **man** pages to learn more about a command e.g.
  - **man ls**
- Redirection: overwrite (>), or append (>>) output from a command to a file e.g.
  - ls -a > myfiles.txt
- Use **more, less, tail** to look inside a file

Working with data

- Generate your own data - try **ping** www.iitm.ac.in
- string editor sed 's/old/new/g' works on each line
  - the first **s** is to substitute
  - the last **g** stands for global
- awk -F, '{print $7}' filename.txt
  - uses the comma as a delimiter
  - prints the 7th column in the file
- Use | to pipe the output of a command to another command

Numerical Data Types

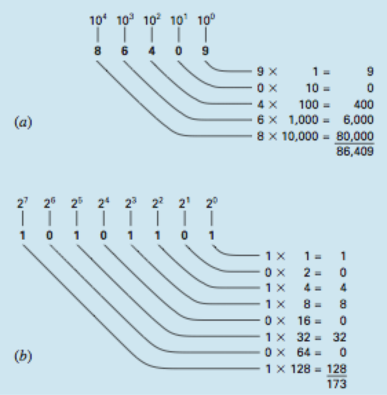
1. Single float (float)
2. Double float (double)
3. Long Double float. (long double)

Called **floats** because the decimal point is moving.

C does not support **fixed-points**

Integer Representation

- Binary
  - Powers of 2
- Decimal
  - Powers of 10
- Hexadecimal e.g. 0x1F
  - $A = 10$
  - $B = 11$
  - $\vdots$
  - $F = 15$



Numerical Data Types

15.875 can be converted to binary as

$$15 = 8 + 4 + 2 + 1 = \text{Binary}(1111)$$
$$.875 = 0.5 + 0.25 + 0.125 = 2^{-1} + 2^{-2} + 2^{-3}$$

So, the binary form of 15.875 is 1111.111 or represented as 01111.111

Space allocated

- float: 4 bytes or 32 bits
- double: 8 bytes or 64 bits
- long double: 10 bytes or 80 bits

Learn to use the data type that is relevant.

Don't misuse long double, or double

32bit Float

- 1 bit represent sign bit
- 8 bits represents exponent (bias is 127)
- 23 bits represents mantissa (recall characteristic and mantissa)
- 32 bit representation is 4 bytes

$$15.875 = \quad 01000001 \quad 01111110 \quad 00000000 \quad 00000000$$

32bit Float - Example

$$15.875 = 1111.111 = 1.111111 \times 2^3$$

- Number is +ve so sign bit is 0
- Mantissa (significand) is 111111 represented as 11111100000000000000000
- Exponent is 11 (or 3) represented as BinaryBinary$((127 + 3) = 130) = 10000010$
- 32 bit representation is $01000001011111100000000000000000$ (4 bytes)

$$01000001 \quad 01111110 \quad 00000000 \quad 00000000$$

64bit Double

- 1 bit represent sign bit
- 11 bits represent exponent (exponent bias is 1023)
- 52 bits represent mantissa.

128bit Long Double

- 1 bit represent sign bit
- 15 bits represent exponent (exponent bias is 16383)
- 64 bits represent mantissa.

Check your conversions here

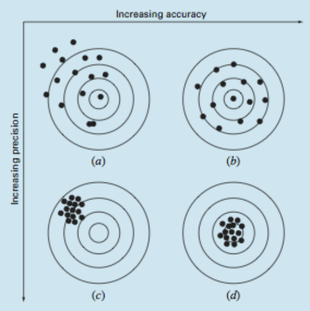
https://www.h-schmidt.net/FloatConverter/IEEE754.html

The Test

- Sample $\sin(x)$ for $x\epsilon[0, 2\pi]$
- Define x as
  - float
  - double
  - long double
- Compare the values generated

Accurate and Precise

- Accurate: average value (mean) is correct
- Precise: spread (standard deviation) is small



Errors

$$\text{Error } \mathcal{E} = \frac{\text{Calculated Value-True Value}}{\text{True Value}}$$

- Round-off: using finite precision
- Truncation: numerical methods use approximations

Iterative Methods

$$e^{-x} = 1 - x + \frac{x^2}{2!} - \frac{x^3}{3!} + \cdots$$

- Series must be truncated after some terms

$$\text{Normalized Error } \varepsilon = \frac{\text{approximate error}}{\text{approximation}} \times 100\%$$

$$\text{Relative Error } \varepsilon_a = \frac{\text{current approx -previous approx}}{\text{current approx}} \times 100\%$$

Taylor Series

Taylor Polynomial and remainder

$$f(x) = \sum_{n=0}^{N} \frac{1}{n!} f^{(n)}(c)(x-c)^n + \frac{1}{(N+1)!} f^{(N+1)}(\xi)(x-c)^{N+1}$$

Standard functions

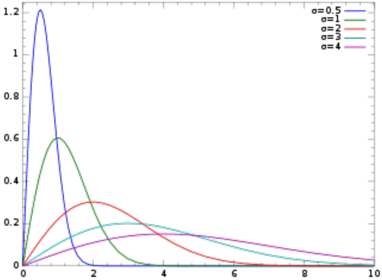
$$e^x = 1 - x + \frac{x^2}{2!} - \frac{x^3}{3!} + \cdots$$

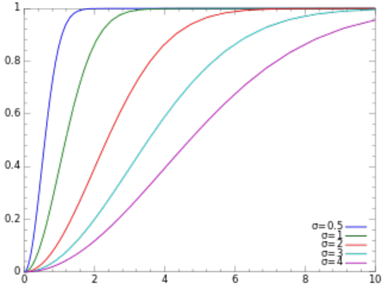
$$\sin x = x - \frac{1}{3!}x^3 + \frac{1}{5!}x^5 + \cdots$$

$$\ln(1+x) = x - \frac{1}{2}x^2 + \frac{1}{3}x^3 + \cdots$$

$$(1+x)^p = 1 + px + \frac{p(p-1)}{2!} + \frac{p(p-1)(p-2)}{3!} + \cdots$$

Probability Distributions

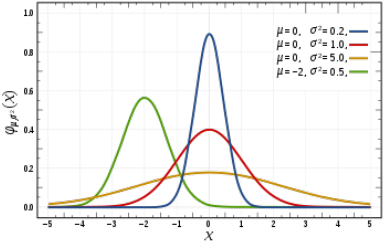


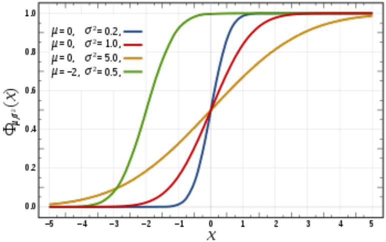


Probability Distribution: $Pr[a \leq X \leq b] = \int_a^b f_X dx$

Cumulative Distribution: $F_X = \int_{-\infty}^{x} f_X(u) du$

Normal Distribution





$$f(x|\mu,\sigma^2) = \frac{1}{\sqrt{(2\pi\sigma^2}} e^{-(s-\mu)^2/\sigma^2}$$

- Mean μ and variance σ^2
- Show that $\sigma^2 = \frac{1}{N}\sum_i \left(x - \langle x \rangle^2\right) = \langle x^2 \rangle - \langle x \rangle^2$

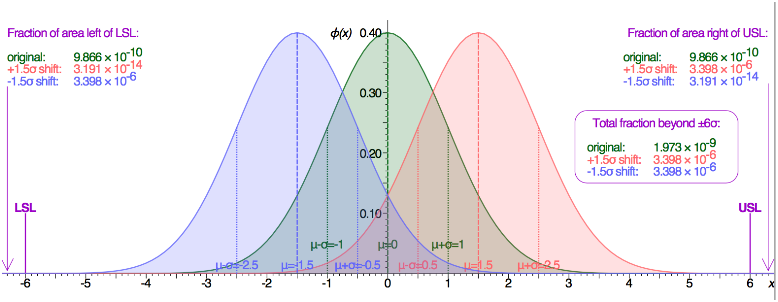
Normal Distribution

- Box-Mueller transform: generate a normal distribution from two uniform distributions $U_1$and U_2

$$Z_0 = R\cos\theta = \sqrt{-2\log U_1}\cos(2\pi U_2)$$
$$Z_1 = R\sin\theta = \sqrt{-2\log U_1}\sin(2\pi U_2)$$

- $Z_0$and $Z_1$are independent variables with standard normal distributions
- https://en.wikipedia.org/wiki/Box-Muller_transform

SixSigma



- Normal distribution characterized by Mean and Standard Deviation
  - 1 part in a billion fall outside 6σ
  - What fraction fall within 2σ and 4σ?

EE1103

Additive Noise

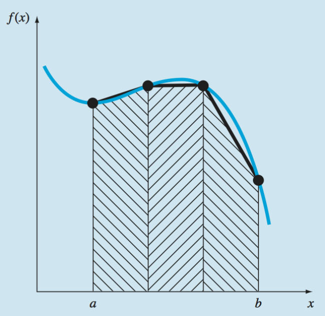
- Generate a sine wave x as an array of points $x_{i=1\ldots N}$
  - Inputs: amplitude $V_{\textsf{pp}}$, freq f, number of cycles M
  - Choose $3 < P < 10$ number of points per cycle such that $N = M \times P$
  - Add Gaussian noise e_i to create the noisy sine wave y_i
    - Choose $\mu = 0$, $\sigma = 0.1 V_{\textsf{pp}}$
    - Use the Box-Muller transform to generate e_i
  - Estimate the error $\varepsilon_i = y_i - x_i$ for all N points and determine μ and σ of $\varepsilon_{i=1\ldots N}$
  - Tabulate the statistics: what % of points lie within 2σ, 4σ, 6σ of the mean?
- Choose a random $\mu \neq 0$, $\sigma = 0.2 V_{\textsf{pp}}$ and repeat the exercise
  - Does your choice of P and μ affect the distribution of e?

Integral of a function

Find the value of $\int_a^b f(x)dx$ given a function $f(x) = 0$

- Rectangular bins
- Trapezoid rule
- Simpson's rule
- What are the errors?

Trapezoid Rule

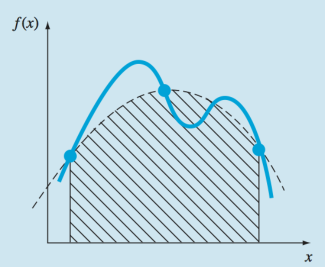


- Break the range $[x_L, x_U]$ into N equal bins
- Calculate

$$y_i = \int_{x_i}^{x_{i+1}} f(x_i)\,dx$$
$$= (\Delta x)\,\frac{f(x_{i+1}) - f(x_i)}{2}$$

- Calculate $I_N = \sum_{i=1}^{N} y_i$
- Plot I_N versus N and look for convergence

Simpsons 1/3 Rule



- Break the range $[x_L, x_U]$ into N equal bins
- Approximate $f(x) \approx f_2(x)$

$$y_i = \int_{x_i}^{x_{i+1}} f_2(x_i)\, dx$$

- Calculate $I_N = \sum_{i=1}^{N} y_i$

- $f_2(x)$ is a second order *Lagrange polynomial*

$$y_i = (\Delta x)\, \frac{f(x_{i-1}) + 4f(x_i) + f(x_{i+1})}{6}$$

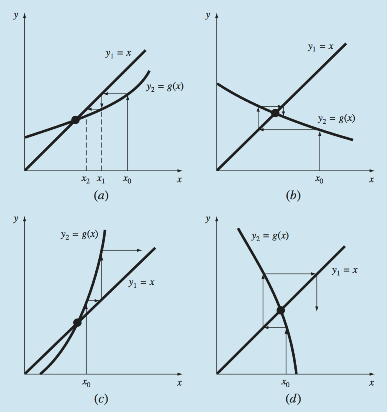
- Plot I_N versus N and look for convergence

Finding the Root

Find the value of x that satisfies a given function $f(x) = 0$

- Simple fixed point iteration
- Bracketing
  - Bisection
  - False position
- Newton-Raphson
- Secant

Fixed Point Iteration

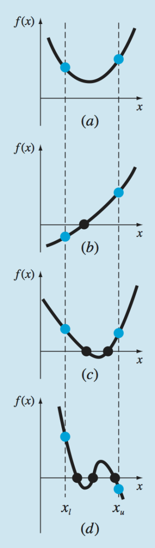


- Start at some value $x = x_0$
- Calculate $y_0 = f(x_0)$
- Set $x_1 = y_0$
- Iterate until
  $\varepsilon_i = \left|\frac{x_{i+1} - x_i}{x_{i+1}}\right| <$ tolerance

Try it for

- $f(x) = \sin(x) + x$
- $f(x) = e^{-x} - x$

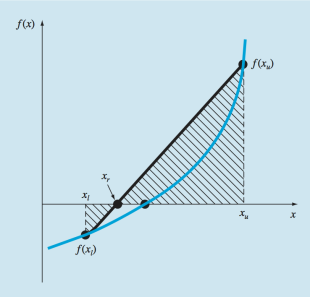
Bracketing



- Start with some values of x_L and x_U
- Calculate $y_L = f(x_L)$ and $y_U = f(x_U)$
- Check that $y_L \times y_U < 0$
- Estimate $x_i = 0.5(x_L + x_U)$
  - if $y_L f(x_i) < 0$ then $x_U = x_i$
  - else $x_L = x_i$
  - Define ε_i and iterate until $\varepsilon_i <$ tolerance
- Find all the roots of the equation

$$f(x) = \sin 10x + \cos 3x \quad 0 < x < 5$$

False Position

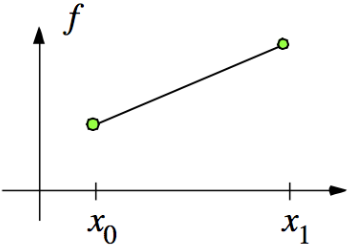


- Start with some values of x_L and x_U
- Calculate $f(x_L)$ and $f(x_U)$
- Check that $f(x_L)f(x_U) < 0$
- Use the slope to estimate x_i

$$x_r = x_u - \frac{f(x_u)(x_l - x_u)}{f(x_l) - f(x_u)}$$

- Define ε_i and iterate until $\varepsilon_i <$ tolerance

Linear Interpolation

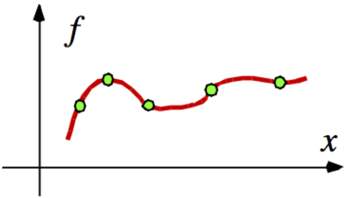


Fit $n+1$ points to a polynomial of order n

- Obtain the value $y = f(x)$ by interpolating between $y_0 = f(x_0)$ and $y_1 = f(x_1)$

$$y - y_0 = m(x - x_0) = \frac{(y_1 - y_0)}{(x_1 - x_0)}(x - x_0)$$

Polynomial Interpolation



- Fit $n + 1$ points to a polynomial of order n
- Set of equations for points $(x_0, y_0) \ldots (x_n, y_n)$

Vandermode Polynomial

$$P_n(x) = c_0 x^n + c_1 x^{n-1} + \ldots + c_n$$

- System of equations to be solved

$$\begin{pmatrix} x_0^n & x_0^{n-1} & \ldots & x_0 & 1 \\ x^{n}1 & x_1^{n-1} & \ldots & x_1 & 1 \\ \vdots & & & & \vdots \\ x_n^n & x_n^{n-1} & \ldots & x_n & 1 \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{pmatrix}$$

- Easiest to do using matrix inversion methods

Lagrange Polynomial

$$\begin{aligned} P_n(x) = & \frac{(x-x_1)(x-x_2)\ldots(x-x_n)y_0}{(x_0-x_1)(x_0-x_2)\ldots(x_0-x_n)} \\ & + \frac{(x-x_0)(x-x_2)\ldots(x-x_n)y_1}{(x_1-x_0)(x_1-x_2)\ldots(x_1-x_n)} \\ & + \frac{(x-x_0)(x-x_1)\ldots(x-x_n)y_2}{(x_2-x_9)(x_2-x_1)\ldots(x_2-x_n)} \\ & + \cdots + \frac{(x-x_0)(x-x_1)\ldots(x-x_{n-1})y_n}{(x_2-x_9)(x_2-x_1)\ldots(x_n-x_{n-1})} \end{aligned}$$

Note: First term does not have $(x - x_0)$ in the numerator, and uses $x_0$in the denominator. Similar for each subsequent term.

Newton Polynomial

$$P_n(x) = c_0 + c_1(x - x_0) + c_2(x - x_0)(x - x_1) + \ldots + c_n(x - x_0)\cdots(x - x_{n-1})$$

Substitutions at each given point (x_i, y_i) results in a lower diagonal matrix that is easy to invert

$$\begin{aligned}
y_0 &= c_0 \\
y_1 &= c_0 + c_1(x - x_0) \\
y_2 &= c_0 + c_1(x - x_0) + c_2(x - x_0)(x - x_1) \\
&\vdots \\
y_n &= c_0 + c_1(x - x_0) + \ldots + c_n(x - x_0)\cdots(x - x_{n-1})
\end{aligned}$$

Adaptive Runge Kutta

- Step halving: Compare the solution of one large step with two half steps
- Embedded RK methods: Compare solutions from two different RK methods to detect sudden changes

$$
\begin{aligned}
y_{i+1} &= y_i + \frac{1}{9}\left(2k_1 + 3k_2 + 4k_3\right) h \\
k_1 &= f(t_i, y_i) \\
k_2 &= f\left(t_i + \frac{1}{2}h, y_i + \frac{1}{2}k_1 h\right) \\
k_3 &= f\left(t_i + \frac{3}{4}h, y_i + \frac{3}{4}k_2 h\right)
\end{aligned}
$$

Local truncation error $E_{i+1} = \frac{1}{72}\left(-5k_1 + 6k_2 + 8k_3 - 9k_4\right) h$ where $k_4 = f(t_{i+1}, y_{i+1})$

Stiff Equations

- Solutions have both a rapidly changing component and a slowly changing component
- Often the rapidly changing one's die as initial transients and the solution is dominated by the slow solution
- Example

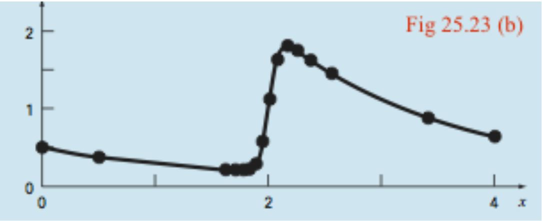
$$\frac{dy}{dt} = -1000y + 3000 - 2000e^{-t}$$

- for $y(0) = 0$ we get the analytic solution

$$y(t) = 3 - 0.998e^{-1000t} - 2.002e^{-t}$$

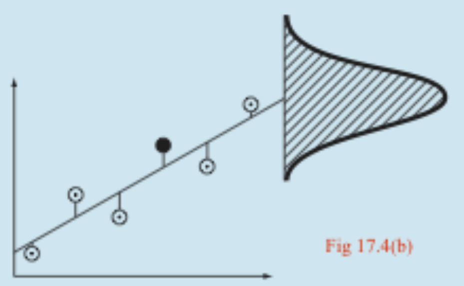
Stiff Equations

- $y = 3 - 0.998e^{-1000t} - 2.002e^{-t}$
- Fast transient from $y = 0$ to $y = 1$ that occurs in $0 < t < 0.005$



Linear Regression

- Values of x are known without error
- Values of y are independent random variables with the same variance
- Values of y for a given x must be normally distributed



Linear Regression

- Given a data set $y(x)$
- Best fit is $y = a_0 + a_1 x + e$
- Residual error $e = y - a_0 + a_1 x$
- Minimize the sum of the squares of the residuals

$$S_r = \sum_{i=1}^{n} e_i^2 = \sum_{i=1}^{n} (y_{i,\text{data}} - y_{i,\text{fit}})^2 = \sum_{i=1}^{n} (y_i - a_0 - a_1 x_i)^2$$

Linear Regression

- Sum of the squares of the residuals

$$S_r = \sum_{i=1}^{n} e_i^2 = \sum_{i=1}^{n} (y_{i,\text{data}} - y_{i,\text{fit}})^2 = \sum_{i=1}^{n} (y_i - a_0 - a_1 x_i)^2$$

- Differentiate with respect to $a_0$and $a_1$and set to zero to minimize S

$$na_0 + \left(\sum x_i\right) a_1 = \sum y_i$$
$$\left(\sum x_i\right) a_0 + \left(\sum x_i^2\right) a_1 = \sum x_i y_i$$

- Solve for a_0 and a_1

Goodness of Fit

- How good is a straight line fit compared to just using the mean of the data?
- Normalize the residual error against the variance of the data

$$S_r = \sum_{i=1}^{n} e_i^2 = \sum_{i=1}^{n} (y_{i,\text{data}} - y_{i,\text{fit}})^2 = \sum_{i=1}^{n} (y_i - a_0 - a_1 x_i)^2$$

$$S_t = \sum_{i=1}^{n} (y_{i,\text{data}} - y_{\text{mean}})^2 = \sum_{i=1}^{n} y_i^2 - n y_{\text{mean}}^2$$

- Coefficient of determination

$$r^2 = \frac{S_t - S_r}{S_t}$$

- Correlation coefficient $0 < r < 1$
- Unwise to claim e.g. $r^2 = 0.93$ is better than $r^2 = 0.92$

Linearization

- Linearize before using regression

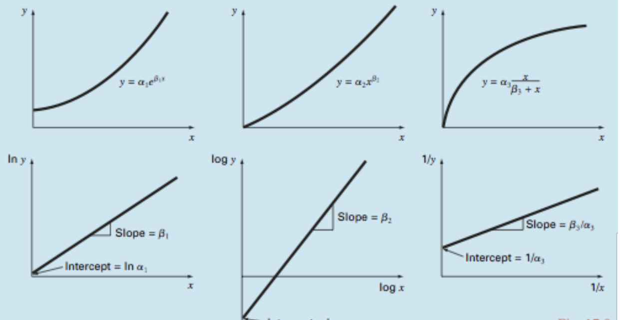
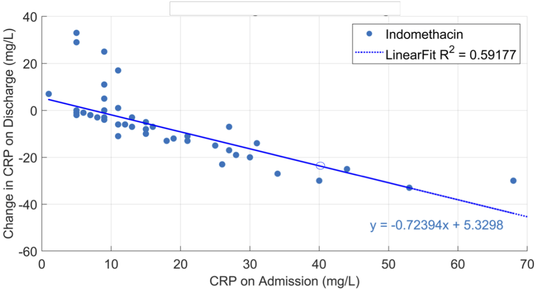


Fig. 17.9

Poor Fits

- Errors must be normally distributed about the fit
- Many data sets will have outliers
  - more than 2σ away
  - decide whether to keep or remove them before you fit



Polynomial Regression

- Fit to $y_{\text{fit}} = a_0 + a_1 x + a_2 x^2 + e$
- Sum of the squares of the residuals

$$S_r = \sum_{i=1}^{n} e_i^2 = \sum_{i=1}^{n} (y_i - a_0 - a_1 x_i - a_2 x_i^2)^2$$

- Differentiate (paritials) with respect to a_0, a_1 and a_2 and set them to zero to minimize S

$$\begin{aligned} n a_0 + \left(\sum x_i\right) a_1 + \left(\sum x_i^2\right) a_2 &= \sum y_i \\ \left(\sum x_i\right) a_0 + \left(\sum x_i^2\right) a_1 + \left(\sum x_i^3\right) a_2 &= \sum x_i y_i \\ \left(\sum x_i^2\right) a_0 + \left(\sum x_i^3\right) a_1 + \left(\sum x_i^4\right) a_2 &= \sum x_i^2 y_i \end{aligned}$$

- Solve for a_0, a_1 and a_2

Multiple Linear Regression

- Fit to $y_{\text{fit}} = a_0 + a_1 x_1 + a_2 x_2 + e$
- Sum of the squares of the residuals

$$S_r = \sum_{i=1}^{n} e_i^2 = \sum_{i=1}^{n} (y_i - a_0 - a_1 x_{1i} - a_2 x_{2i})^2$$

- Differentiate (paritials) with respect to a_0, a_1 and a_2 and set them to zero to minimize S

$$\begin{bmatrix} n & \sum x_{1i} & \sum x_{2i} \\ \sum x_{1i} & \sum x_{1i}^2 & \sum x_{1i} x_{2i} \\ \sum x_{2i} & \sum x_{1i} x_{2i} & \sum x_{2i}^2 \end{bmatrix} \begin{Bmatrix} a_0 \\ a_1 \\ a_2 \end{Bmatrix} = \begin{Bmatrix} y_i \\ \sum x_{1i} y_{1i} \\ \sum x_{2i} y_i \end{Bmatrix}$$

- Invert the coefficient matrix to solve for a_0, a_1 and a_2

Standard Error

- Lose 2 degrees of freedom when we estimate a_0 and a_1
- Standard error of the estimate for a linear fit

$$s_{y/x} = \sqrt{\frac{S_r}{n-2}} \qquad \text{for a fit to n-points}$$

- Polynomial fit, of order m, with $m+1$ DOF

$$s_{y/x} = \sqrt{\frac{S_r}{n-(m+1)}} \qquad \text{for a fit to n-points}$$

- Also, if you fit to $y = a_0 + a_1 x_1 + a_2 x_2 + \cdots + a_m x_m + e$

$$s_{y/x} = \sqrt{\frac{S_r}{n-(m+1)}} \qquad \text{for a fit to n-points}$$