Section 5 – Curve Fitting

Quite often we are presented with data in the form of discrete values, and the need arises to find appropriate values between these discrete points Two general approaches exist, depending on the source of the data. If the discrete data are assumed to posses error (such as the results of physical experiments), functions are fit to the data to minimize the overall error but not necessarily pass through any of the points. A popular approach that we will explore is *least squares regression*. If the data is assumed to be precise, such as that obtained from tables of physical properties or mathematical functions, we fit a curve to pass through each point, a process called *interpolation.*

5.1 Least-Squares Regression

Consider the case fitting a polynomial to a set of imperfect data, the curve to have the form

$$y = a_0 + a_1 x + a_2 x^2 + \ldots + a_m x^m$$

The error associated with each data point (x_i, y_i) is

$$e_i = y_i - (a_0 + a_1 x_i + a_2 x_i^2 + \ldots + a_m x_i^m) \quad i = 1, 2, \ldots n = \#\text{ of data points}$$

To minimize the error of the entire problem, minimize the sum of the squares of the errors, or residuals. To do this, take the derivative with respect to each coefficient

$$\begin{aligned}
&\sum e_i^2 = S_r = \sum (y_i - a_0 - a_1 x_i - a_2 x_i^2 - \ldots - a_m x_i^m)^2 \\
&\frac{\partial S_r}{\partial a_0} = -2\sum (y_i - a_0 - a_1 x_i - a_2 x_i^2 - \ldots - a_m x_i^m) = 0 \\
&\frac{\partial S_r}{\partial a_1} = -2\sum x_i\,(y_i - a_0 - a_1 x_i - a_2 x_i^2 - \ldots - a_m x_i^m) = 0 \\
&\frac{\partial S_r}{\partial a_2} = -2\sum x_i^2 (y_i - a_0 - a_1 x_i - a_2 x_i^2 - \ldots - a_m x_i^m) = 0 \\
&\vdots
\end{aligned}$$

which can be rearranged to give

$$\begin{aligned}
&n a_0 + \sum x_i a_1 + \sum x_i^2 a_2 + \ldots + \sum x_i^m a_m = \sum y_i \\
&\sum x_i a_0 + \sum x_i^2 a_1 + \sum x_i^3 a_2 + \ldots + \sum x_i^{m+1} a_m = \sum x_i y_i \\
&\sum x_i^2 a_0 + \sum x_i^3 a_1 + \sum x_i^4 a_2 + \ldots + \sum x_i^{m+2} a_m = \sum x_i^2 y_i \\
&\vdots \\
&\sum x_i^m a_0 + \sum x_i^{m+1} a_1 + \sum x_i^{m+2} a_2 + \ldots + \sum x_i^{2m} a_m = \sum x_i^m y_i
\end{aligned}$$

The *standard error of the estimate*, $S_{y/x}$ (the error in a predicted value of y given a value of x) is

$$S_{y/x} = \sqrt{\frac{S_r}{n-(m+1)}}$$

where n is the number of data points and m is the order of the polynomial fit. The *sum of squares around the mean* for the dependent variable y is

$$S_t = \sum (y_i - \bar{y})^2$$

The *coefficient of determination* is a measure of how much of the variation in the data is explained by the model, that is, how well the data fits the statistical model:

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

The *correlation coefficient* is r and measures the strength of the association between the variables.

Pseudocode – Polynomial Regression, assembly of normal equations

```
‘ Order of polynomial to be fit: m
‘ Number of data points: n
‘ If n<m+1, regression not possible, if n>=m+1 regression possible

DOFOR i = 1, m + 1
      DOFOR j = 1, i
            k = i + j  -2
            sum = 0
            DOFOR L = 1, n
                  Sum = sum +x(L) ^ k
            END DO
            a(i,j) = sum
            a(j,i) = sum
      END DO
      sum = 0
      DOFOR L = 1, n
            sum = sum + y(L) * x(L) ^ (i-1)
      END DO
      a(i, m+2) = sum
END DO
```

5.1.1 Linear Regression (m=1)

For the case when the order of the fit m = 1, the above equations reduce to

$$na_0 + \sum x_i a_1 = \sum y_i$$
$$\sum x_i a_0 + \sum x_i^2 a_1 = \sum x_i y_i$$

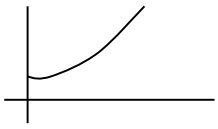
Solving for the regression coefficients gives

$$a_1 = \frac{n\sum x_i y_i - \sum x_i \sum y_i}{n\sum x_i^2 - (\sum x_i)^2}$$
$$a_0 = \bar{y} - a_1 \bar{x}$$

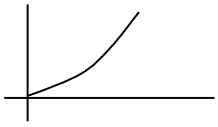
It is assumed here that (1) *x* has a fixed value, is not random and has no error, (2) *y* values are random and all have the same variance, and (3) the *y* values for a given *x* are normally distributed.

5.1.2 Linearizations of Non-Linear Relationships

Exponential model $y = \alpha_1 e^{\beta_1 x}$
Take the natural logarithm of both sides of the equations to yield the linear equation: $\ln y = \ln \alpha_1 + \beta_1 x$

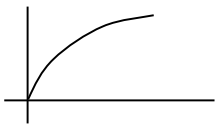


Power model $y = \alpha_2 x^{\beta_2}$
Take the base-10 logarithm of both sides of the equations to yield the linear equation: $\log y = \log \alpha_2 + \beta_2 \log x$



Saturation Growth Rate model $y = \alpha_3 \frac{x}{\beta_3 + x}$

Invert the model to yield the linear equation: $\frac{1}{y} = \frac{\beta_3}{\alpha_3 x} + \frac{1}{\alpha_3}$



In all of the models above, use linear regression to evaluate the regression constants, solve for the model parameters and use in the original model for predictive purposes.

5.1.3 Multiple Linear Regression

Consider the case where *y* is a function is *m* independent variables:

$$y = a_0 + a_1 x_1 + a_2 x_2 + \ldots + a_m x_m + e$$

Again, the "best" fit is determined by squaring the residuals

$$S_r = \sum (y_i - a_0 - a_1 x_{1i} - a_2 x_{2i} - \ldots - a_m x_{mi})^2 \quad i = 1, 2 \ldots n$$

and differentiating with respect to the unknown coefficients

$$\frac{\partial S_r}{\partial a_0} = -2 \sum (y_i - a_0 - a_1 x_{1i} - a_2 x_{2i} - \ldots - a_m x_{mi}) = 0$$
$$\frac{\partial S_r}{\partial a_1} = -2 \sum x_{1i} (y_i - a_0 - a_1 x_{1i} - a_2 x_{2i} - \ldots - a_m x_{mi}) = 0$$
$$\vdots$$

with standard error

$$S_{y/x} = \sqrt{\frac{S_r}{n - (m+1)}}$$

5.1.4 Power Equation of the General Form

When modeling systems of the type

$$y = a_0 x_1^{a_1} x_2^{a_2} x_3^{a_3} \ldots x_m^{a_m}$$

take the natural logarithm of both sides to yield

$$\log y = \log a_0 + a_1 \log x_1 + a_2 \log x_2 + a_3 \log x_3 + \ldots + a_m \log x_m$$

which can be fit with multiple linear regression.

5.1.5 Solution Techniques

The least squares equations are symmetric and are often not well conditioned. They can be solved by Gauss Elimination, by Crout or Cholesky decomposition methods, or by matrix inversion. If the best order of polynomial fit is sought, Cholesky's method can be used efficiently to build and solve the equations.

All solutions must be examined to observe the actual fit to the data. Relying solely on correlation coefficients to determine the appropriateness of the curve fit is not recommended.

5.1.6 Non-Linear Regression – *Gauss-Newton Method*

Consider a model of the form

$$y_i = f(x_i) + e_i$$

where x_i is a non-linear function of parameters $a_0, a_1, \ldots a_m$. Expand this function in a Taylor series around the parameters. For example, for 2 parameters a_0 and a_1

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

which linearizes the model with respect to the parameters. Therefore,

$$y_i - f(x_i)_j = \frac{\partial f(x_i)_j}{\partial a_0}\Delta a_0 + \frac{\partial f(x_i)_j}{\partial a_1}\Delta a_1 + e_i$$

or in matrix form

$$\vec{D} = \vec{Z}_j \Delta\vec{A} + \vec{E}$$

where n is the number of data points and

$$\vec{Z} = \begin{bmatrix} \frac{\partial f_1}{\partial a_0} & \frac{\partial f_1}{\partial a_1} \\ \frac{\partial f_2}{\partial a_0} & \frac{\partial f_2}{\partial a_1} \\ \vdots & \vdots \\ \frac{\partial f_n}{\partial a_0} & \frac{\partial f_n}{\partial a_1} \end{bmatrix} \qquad \vec{D} = \begin{bmatrix} y - f(x_1) \\ y - f(x_2) \\ \vdots \\ y - f(x_n) \end{bmatrix} \qquad \Delta\vec{A} = \begin{bmatrix} \Delta a_0 \\ \Delta a_1 \\ \vdots \\ \Delta a_m \end{bmatrix}$$

Apply the least squares theory to result in the normal equations

$$\vec{Z}_j^T \vec{Z}_j \Delta\vec{A} = \vec{Z}_j^T \vec{D}$$

These equations are solved for $\Delta\vec{A}$, the change in the parameters, and

$$a_{0,j+1} = a_{0,j} + \Delta a_0$$
$$a_{1,j+1} = a_{1,j} + \Delta a_1$$

The procedure is repeated until the relative error

$$\left|\varepsilon_a\right|_k = \left|\frac{a_{k,j+1} - a_{k,j}}{a_{k,j+1}}\right|$$

falls below an acceptable criterion. Shortcomings of this method are that it may converge slowly, it may oscillate (sometimes wildly) or it may not converge at all.

5.2 Interpolation

Interpolation is the estimation of intermediate values between precisely-known data points. The most common methods are polynomial interpolation and splines.

5.2.1 Newton's Divided-Difference Interpolating Polynomials

Consider an *n*th-order polynomial to be fit to *n*+1 data points, of the form

$$f_n(x) = b_0 + b_1(x - x_0) + b_2(x - x_0)(x - x_1) + b_3(x - x_0)(x - x_1)(x - x_2) + \cdots$$
$$\cdots + b_n(x - x_0)(x - x_1)\ldots(x - x_{n-1})$$

To evaluate the coefficients *b*, begin by setting $x = x_0$, which gives

$$b_0 = f(x_0)$$

Setting $x = x_1$ gives

$$b_1 = \frac{f(x_1) - b_0}{x_1 - x_0} = \frac{f(x_1) - f(x_0)}{x_1 - x_0} = f[x_1, x_0]$$

which is the *first divided difference*. Similarly for $x = x_2$, etc.:

$$b_2 = \frac{\dfrac{f(x_2) - f(x_1)}{x_2 - x_1} - \dfrac{f(x_1) - f(x_0)}{x_1 - x_0}}{x_2 - x_0} = f[x_2, x_1, x_0]$$
$$\vdots$$
$$b_n = f\left[x_{n,x_{n-1}}, \ldots, x_2, x_1, x_0\right]$$

These calculations can be efficiently organized in a table, for example, for 3 data points:

i	x_i	$f(x_i)$	First divided differences	Second divided differences	Third divided differences
0	x_0	$f(x_0)$	$f[x_1, x_0]$	$f[x_2, x_1, x_0]$	$f[x_3, x_2, x_1, x_0]$
1	x_1	$f(x_1)$	$f[x_2, x_1]$	$f[x_3, x_2, x_2]$	
2	x_2	$f(x_2)$	$f[x_3, x_2]$		

3	x_3	$f(x_3)$			

The first entries in the divided difference columns are the coefficients b_0, b_1, b_2, etc.

The data points need not be equally spaced, nor do they have to be in any particular order. If one has a choice of data points, it is always best to choose them to bracket and lie close to the region where the intermediate estimates will be required.

The error associated with Newton's divided-difference interpolating polynomial can be estimated, with an additional data point, as

$$R_n = f[x_{n+1}, x_n, x_{n-1}, \ldots, x_0](x - x_0)(x - x_1)\ldots(x - x_n)$$

where

$$f_{n+1}(x) = f_n(x) + R_n$$

that is, the error of the prediction can be estimated by examining the effect of adding another data point to the interpolation.

5.2.2 Lagrange Interpolating Polynomials

Newton's divided-difference interpolating polynomials can be reformulated to avoid the computation of the divided differences:

$$f_n(x) = \sum_{i=0}^{n} L_i(x) f(x_i)$$

where

$$L_i(x) = \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{x - x_j}{x_i - x_j}$$

so that for n=1 and n=2,

$$f_1(x) = \frac{x - x_1}{x_0 - x_1} f(x_0) + \frac{x - x_0}{x_1 - x_0} f(x_1)$$

$$f_2(x) = \frac{x - x_1}{x_0 - x_1}\frac{x - x_2}{x_0 - x_2} f(x_0) + \frac{x - x_0}{x_1 - x_0}\frac{x - x_2}{x_1 - x_2} f(x_1) + \frac{x - x_0}{x_2 - x_0}\frac{x - x_1}{x_2 - x_1} f(x_2)$$

In practice, Newton's polynomial is useful because it offers insight to the behavior of formulas of different orders, and the error estimate can be incorporated fairly easily. Therefore, Newton's method is good for exploratory computations. For single-use applications, the Newton and Lagrange methods offer similar performance, although Lagrange is easier to program and does not require the computation and storage of the divided differences. Lagrange is the preferred method if the order of the interpolating polynomial is known beforehand.

Inverse interpolation – This is the problem of finding a value x for a given value of $f(x)$. It is simply a root-finding problem, solved by the methods of Section 2.

Extrapolation – This is the process of estimating function values outside of the range of the given data. It is always risky and prone to large errors.

5.2.3 Spline Interpolation

Instead of fitting high-order interpolating polynomials to larges sets of data, we can fit lower-order polynomials, or *spline functions*, to subsets of the data points. Generally, spline functions avoid the wild swings associated with high-order polynomials while still passing through each of the precisely-known data points.

Linear Splines

Connect the data points with straight lines, as

$$f(x) = f(x_i) + m_i(x - x_i) \quad \text{for } x_i \le x \le x_{i+1}$$

where

$$m_i = \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i} \quad i = 0,1,\ldots,n$$

Note that the derivatives of linear splines are not continuous at the data points.

Quadratic Splines

We would like the spline functions to have continuous derivatives at the points where the spline functions meet, called *knots*. To insure that m derivatives are continuous, splines of at least m+1 order are required. For the first derivative to be continuous, fit quadratic functions of the form

$$f_i(x) = a_i x^2 + b_i x + c_i$$

to n+1 data points having n intervals, resulting in a system of 3n unknowns to determine. The requirements on the spline functions are

1. The function values of adjacent spline functions must be equal at the interior knots:

$$f(x_{i-1}) = a_{i-1}x_{i-1}^2 + b_{i-1}x_{i-1} + c_{i-1}$$
$$f(x_{i-1}) = a_i x_{i-1}^2 + b_i x_{i-1} + c_i$$

   for i = 2, 3, … n. This results in 2n-2 equations.

2. The first and last splines must pass through the end points.

$$f(x_0) = a_1 x_0^2 + b_1 x_0 + c_1$$
$$f(x_n) = a_n x_n^2 + b_n x_n + c_n$$

   This condition results in 2 equations.

3. The first derivatives at the interior knots must be equal for adjacent splines.

$$2a_{i-1}x_{i-1} + b_{i-1} = 2a_i x_{i-1} + b_i$$

   for $i = 2, 3, \ldots n$. This is another n-1 conditions.

4. Assume that the second derivative at x_0 is zero, so that $a_1 = 0$

The resulting system of 3n equations in 3n unknowns can be solved for the unknown spline function coefficients.

Cubic Splines

The most useful of the spline interpolations is the cubic spline, where we fit cubic equations of the form

$$f_i(x) = a_i x^3 + b_i x^2 + c_i x + d_i$$

between knots. For n+1 data points there are n intervals and 4n constraints:

1. The function values must be equal at the interior knots (2n-2 conditions)
2. The first and last spline functions must pass through the end points (2 conditions)
3. The first derivatives must be equal at the interior knots (n-1 conditions)
4. The second derivatives must be equal at the interior knots (n-1 conditions)
5. Assume that the second derivatives at the end points are zero (2 conditions)

Applying these conditions results in, for each interval:

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

The unknown second derivatives can be solved by applying

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

to each of the interior knots, resulting in *n*-1 equations for the *n*-1 unknowns (recall that the second derivatives at the ends are assumed to be zero). Note that this is a tridiagonal system and can be solved very efficiently through Thomas' Algorithm.

5.3 Fourier Approximation

A *periodic function* is one for which

$$f(t) = f(t+T)$$

where the constant T is called the *period* and is the smallest value for which the relationship holds. Examples include square and sawtooth waves; the simplest are sinusoidal functions. Sinusoids can be expressed as sine or cosine functions, with the general form

$$f(t) = A_0 + C_1 \cos(\omega_0 t + \theta)$$

where A_0 is the mean value, C_1 is the amplitude, ω_0 is the angular frequency or how often the cycles occur and θ is the phase shift. Recall that $\omega_0 = 2\pi f$, where $f = 1/T$ is the frequency of the periodic function. This form can also be written as

$$f(t) = A_0 + A_1 \cos \omega_0 t + B_1 \sin \omega_0 t$$

where $A_1 = C_1 \cos\theta$, $B_1 = -C_1 \sin\theta$ so that $\theta = \tan^{-1}(-B_1/A_1)$ and $C_1 = \sqrt{A_1^2 + B_1^2}$. The relationship can also be written in the form

$$f(t) = A_0 + C_1 \sin(\omega_0 t + \delta)$$

where $\delta = \theta + \pi/2$.

5.3.1 Fourier Approximation

Consider the case of fitting data to a linear least-squares model of the form

$$y(t) = A_0 + A_1 \cos \omega_0 t + B_1 \sin \omega_0 t + e$$

To determine the constants A_0, A_1 and B_1 that minimize the square of the residuals

$$S_r = \sum_{i=1}^{N} \left[y_i - \left(A_0 + A_1 \cos \omega_0 t + B_1 \sin \omega_0 t \right) \right]^2$$

One can write the normal equations for this system, which for N equispaced intervals Δt reduce to

$$\begin{bmatrix} N & 0 & 0 \\ 0 & N/2 & 0 \\ 0 & 0 & N/2 \end{bmatrix} \begin{Bmatrix} A_0 \\ A_1 \\ B_1 \end{Bmatrix} = \begin{Bmatrix} \sum y \\ \sum y \cos \omega_0 t \\ \sum y \sin \omega_0 t \end{Bmatrix}$$

So that

$$\begin{aligned} A_0 &= \sum y / N \\ A_1 &= (2/N) \sum y \cos \omega_0 t \\ B_1 &= (2/N) \sum y \sin \omega_0 t \end{aligned}$$

Extending this result to the general model

$$\begin{aligned} y(t) = A_0 + A_1 \cos \omega_0 t + B_1 \sin \omega_0 t + A_2 \cos 2\omega_0 t + B_2 \sin 2\omega_0 t + \ldots \\ \ldots + + A_m \cos m\omega_0 t + B_m \sin m\omega_0 t \end{aligned}$$

gives

$$\begin{aligned} A_0 &= \sum y / N \\ A_1 &= (2/N) \sum y \cos j\omega_0 t \\ B_1 &= (2/N) \sum y \sin j\omega_0 t \end{aligned} \quad j = 1, 2, \ldots m$$

This is seldom done for regression ($N > 2m + 1$), but rather for interpolation or collocation ($N = 2m + 1$)

5.3.2 Continuous Fourier Series

Fourier showed that an arbitrary periodic function can be represented by an infinite series of sinusoids of harmonically related frequencies. For a function with period T,

$$f(t) = a_0 + a_1 \cos(\omega_0 t) + b_1 \sin(\omega_0 t) + a_2 \cos(2\omega_0 t) + b_2 \sin(2\omega_0 t) + \ldots$$

or

$$f(t) = a_0 + \sum_{k=1}^{\infty} \left[a_k \cos(k\omega_0 t) + b_k \sin(k\omega_0 t) \right]$$

where $\omega_0 = 2\pi/T$ is the *fundamental frequency* and the integer multiples of ω_0 are *harmonics*. The coefficients above can be computed as

$$a_0 = (1/T)\int_0^T f(\tau)d\tau$$

$$\left.\begin{aligned} a_k &= (2/T)\int_0^T f(\tau)\cos(k\omega_0 t)d\tau \\ b_k &= (2/T)\int_0^T f(\tau)\sin(k\omega_0 t)d\tau \end{aligned}\right\} \quad k = 1, 2, \ldots$$

This series can also be expressed in terms of exponential functions as

$$f(t) = \sum_{-\infty}^{\infty} C_k e^{ik\omega_0 t}$$

where

$$C_k = (1/T)\int_{-T/2}^{T/2} f(\tau)e^{-ik\omega_0 t}d\tau \quad \text{and } i = \sqrt{-1}$$

Time and Frequency Domains
In the *time domain* all of the frequency components are superimposed on the same amplitude/time scale. In the *frequency domain* the component sinusoids are decomposed and represented as amplitudes and phase angles as functions of frequency.

5.3.3 Fourier Integral and Transform

Non-periodic or non-recurring signals exhibit a continuous rather than discrete frequency spectrum. Allow the exponential form of the Fourier Series to have a period that approaches infinity, then

$$f(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} F(i\omega_0)e^{i\omega_0 t}d\omega_0 \quad \text{(inverse Fourier Transform)}$$

and

$$F(i\omega_0) = \int_{-\infty}^{\infty} f(\tau)e^{-i\omega_0 \tau}d\tau \quad \text{(Fourier Transform)}$$

Fourier Series is applied to continuous, periodic time functions and yields frequency domain magnitudes at discrete frequencies, while the Fourier Transform is applied to continuous (not necessarily periodic) time domain functions to yield a continuous frequency domain function.

5.3.4 Discrete Fourier Transform (DFT)

Most often we obtain functions by finite sets of discrete values. Consider a function known on an interval from 0 to t in N equally spaced subintervals with width $\Delta t = T/N$. Subscript n will designate times at which samples are taken, so that f_n is a sample of $f(t)$ taken at time t_n. The data points are taken at n =0, 1, 2, … N-1. For such a sample a discrete Fourier Transform can be written as

$$F_k = \frac{1}{N}\sum_{n=0}^{N-1} f_n e^{-ik\omega_0 n} \quad k = 0, 1 \ldots N\text{-}1, \omega_0 = 2\pi / N$$

$$f_n = \sum_{k=0}^{N-1} F_k e^{ik\omega_0 n} \qquad n = 0, 1, \ldots N\text{-}1$$

To perform the calculations, use Euler's Identity $e^{\pm ia} = \cos a \pm i \sin a$

$$F_k = \frac{1}{N}\sum_{n=0}^{N} f_n \left[\cos(k\omega_0 n) - i\sin(k\omega_0 n)\right]$$

$$f_n = \sum_{k=0}^{N-1} F_k \left[\cos(k\omega_0 n) + i\sin(k\omega_0 n)\right]$$

The computational effort of this transform pair is proportional to N^2, which increases rapidly as the number of samples increases.

Fast Fourier Transform - The calculations of the DFT can be separated into even and odd expressions, and can be reduced to calculations of $N/2$ length sequences. This leads to a halving of the computational effort each time it is done. If the data consists of N intervals, where N is a power of 2, then this reduction can be carried to the individual terms of the series, resulting in a computational effort of the entire transform of $N\log_2 N$, a huge savings.

When performing an FFT, the amount of data that is included and the rate at which it was sampled determine both the frequency resolution and the maximum frequency at which reliable results can be calculated.

The maximum frequency is determined by the sampling rate, Δt. Nyquist said that functions need to be sampled at least twice the highest frequency of interest, or else the resulting FFT is subject to *aliasing*. Alternatively, the highest frequency components that can be relied on correspond to the half the sampling rate. So if the sampling rate is Δt, the highest frequency in Hz that can be relied on is $f_{max} = 1/(2\Delta t) = (N/2)/(N\Delta t)$, where N is the number of data points taken at the sampling rate and $N\Delta t$ is the total time of the input data. A simple way to avoid aliasing is to report only the first half of the FFT spectrum and to double the values of the magnitudes of the frequency components.

The frequency resolution is directly related to the total time data, $f_o = 1/N\Delta t$. The FFT returns frequencies in terms of an index. Dividing the index integer by $N\Delta t$ gives the corresponding frequency directly in Hz.

Pseudocode – Fast Fourier Transform

```
' The actual FFT itself, real-valued data stored in x
m = LOG(N)/LOG(2)
N2 = N
DOFOR k=1, m
      N1 = N2
      N2 = N2/2
      angle = 0
      arg = 2*PI/N1
      DOFOR j = 0, N2-1
            c = cos(angle)
            s = -sin(angle)
            DOFOR i = j, N-1, N1
                  kk = i + N2
                  xt = x(i) - x(kk)
                  x(i) = x(i) - y(kk)
                  yt = y(i) - y(kk)
                  y(i) = y(i) + y(kk)
                  x(kk) = xt*c - yt*s
                  y(kk) = yt*c + xt*s
            END DO
            angle = (j+1)*arg
      END DO
END DO
' Bit-reversal to unscramble the coefficients
j = 0
DOFOR i = 0, N-2
      IF (i<J) THEN
            xt = x(j)
            x(j) = x(i)
            x(i) = xt
            yt = y(j)
            y(j) = y(i)
            y(i) = yt
      END IF
      k = N/2
      DO
            IF (k >= j+1) EXIT
            j = j-k
            k = k/2
      END DO
      j = j+k
END DO
DOFOR i = 0, N-1
      x(i) = x(i)/N
      y(i) = y(i)/N
END DO
```