LARS INFORMATION NOTE 061367

A NOTE ON THE USE OF THE FAST FOURIER TRANSFORM

by H. Merrill

The Cooley-Tukey algorithm for computation of the Fourier Transform first appeared in Mathematics of Computation[1] in April, 1965, and has had a tremendous impact on data processing wherein the Fourier transform is a useful instrument. Not only does the algorithm produce a fifty-fold reduction in computation time (the time is proportional to $N \log_2 N$ rather than the normal dependence on N^2), it also produces a more accurate transformation by reducing the number of additions required, and the resulting round off error is proportional to $\log_2 N$ rather than N.

The essence of the algorithm was apparently first developed in the Physics Department of Purdue University in 1942[2] for real valued functions and designed for use on hand calculators, but was never extended to complex functions and digital machines until Cooley-Tukey, unaware of the previous work, developed the algorithm for complex functions of three variables. In the fall of 1966, Cooley himself created a FORTRAN subroutine for implementing the one-dimensional algorithm, and this program is documented under the SHARE Library, Number 3465[3]. A copy of this document is also filed in the LARS Program Library, DH 0004, and it is this program which is discussed herein. The only significant limitation of the algorithm is the requirement that the number of points to be transformed be equal to some power of two, but this limitation can be eliminated.

Three points are to be mentioned herein: (a) the technique when the number of points M is not equal to a power of two; (b) resolution of the resulting transform; and (c) phase angle correction. As will be seen in the third part, the difference between the forward and inverse transforms is essentially the change of a sign in the phase angle calculation, and an appropriate amplitude constant.

The notation used herein is that f(t) is the function (assumed in the time domain for convenience) to be transformed and $F(\omega)$ is the resulting transform, which may be either the forward or inverse transform. The forward transform is defined by

$$F(\omega) = \int f(t) \exp(-j\omega t)\, dt$$

where f(t) may be complex.

a. Number of Points. The basic algorithm requires the number of data points to be a power of two. It is possible to avoid this limitation, if one is careful. If M points are to be transformed, where M is not a power of two, compute the average value of the M points, and subtract this value from each point (i.e., remove the dc value). After removing the dc value, pad the upper end of the data array with zeroes until the number of points is N, the next highest power of two. The desired transform can be then taken on the N points. It is improper to simply pad zeroes to increase the number of points to a power of two, because the algorithm essentially removes the dc value of the waveform prior to transformation, and the zeroes padded on the upper end of the data array would appear as a negative step function with value equal to the dc value of the first M points, resulting in a transform modulated by a $\frac{\text{Sin } X}{X}$ function.

b. Resolution in the Transform Domain. The algorithm essentially assumes the time-bandwidth product is unity, from which the scaling of the axis in the transform domain is such that the frequency at the i^{th} point is

$$\omega_i = \frac{i}{N}\, \omega_s$$

where i = point number in transform domain (=1, 2, ..., $\frac{N}{2}$)

N = number of points transformed (power of two)

ω_s = sampling rate of time-domain waveform, or

$\omega_s = \frac{2\pi}{t_s}$ where t_s is the sampling period time.

Since the transform is symmetric about the $\frac{N^{th}}{2}$ point in the transform domain, the maximum frequency occurs at this point, and is seen to be $\frac{1}{2}\omega_s$, with which Nyquist would certainly agree. Thus the resolution between points in the transform domain is $\frac{\omega_s}{N}$ and a simple method for increasing the resolution is to pad more zeroes to the data array (after removal of the dc value, as in part a.), thus increasing N and the resolution. If the original waveform sample points did not honestly represent the waveform, increasing the number of points in this manner would incorrectly fill in points in the transform domain, but one generally must assume good representation of the sampled waveform as a starting point.

c. Phase angle considerations. The algorithm returns the real and imaginary parts of the transform points, and a phase angle can be computed directly from these values, but very frequently a more meaningful phase angle spectrum can be constructed. Computing the phase angle based on the real and imaginary points (referred to herein as the uncorrected phase angle) essentially assumes the original waveform had its epoch at zero. For the general case, a waveform with epoch at t_o would introduce a linear phase shift ωt_o in its transform, since if

$$f(t) \longrightarrow F(\omega)$$

then

$$f(t-t_o) \longrightarrow F(\omega)\ e^{j\omega t_o}$$

Removal of the linear phase shift tends to make the dispersion of the phase spectrum less, and variations are more easily seen. By applying the above to a shifted pulse, the magnitude of the necessary correction is found to be $\frac{2\pi n_c}{N}$ where n_c is the epoch point of the time waveform. The sign of the correction is dependent on whether the transform being taken is the forward or inverse. For the forward transform the linear phase shift has negative slope, hence the correction term must have positive sign, and vice-versa. A simple

test is to observe the slope of the uncorrected phase shift and choose the opposite sign for the correction term. Since digitally implemented arctangent rountines are normally module π, it has been found advantageous to displace the corrected phase angle by π to avoid jumps from the top to bottom of the phase spectrum plot, and thus center the spectrum about π radians.

The phase angle computation is given by

$$\theta_i = \mathrm{Tan}^{-1}\left[\frac{\mathrm{Im}\{F(\omega_i)\}}{\mathrm{Re}\{F(\omega_i)\}}\right] \pm 2\pi \frac{n_c}{N} i \begin{matrix} + \pi \text{ if } \mathrm{Re}\{F(\omega)\} \text{is negative} \\ + 0 \text{ if } \mathrm{Re}\{F(\omega)\} \text{is positive} \end{matrix}$$

where ω_i = radian frequency of i^{th} point

n_c = epoch point of time waveform

N = number of points transformed

Sign determined as mentioned above.

The change of sign compensates for the direction (forward or inverse) of the transformation, and the only other difference the direction introduces is an amplitude constant, which can be sorted out by either direct calibration, or by observing mean square values of both waveform and transform, and using Parseval's theorem to set the constant.

Cooley's SHARE abstract points out methods of simultaneously transforming two real waveforms in one pass, and methods for transformation data sets which exceed core; in both cases the above remarks apply. Gentleman and Sande[4] discuss methods of convolution by Fourier transformation rather than by normal lagged product methods. Cooley has also written an Assembly Level Program for the three dimensional transform[5]. Substantial work in other areas of application is widespread.

The main advantage of the Fast Fourier Transform is its speed of computation, which has brought the transform, always an excellent theoretical tool, into the domain of nearly real-time data processing techniques. Comparison of transform times have shown for a 500 point transform by a reasonably fast

direct integration algorithm required 180 seconds, as compared with three seconds under the Cooley-Tukey algorithm.

A convenient subroutine for computing transforms using the Cooley-Tukey algorithm with the modifications discussed herein has been written and documented by the author under IARS Program Library DH0003.

REFERENCES

1. Cooley, J. W. and J. W. Tukey, "An Algorithm for the Machine Calculation of Complex Fourier Series", Math. of Comp., vol. 19, no. 90, (April 1965), pp. 297-301.

2. Danielson, G. C. and C. Lanczos, "Some Improvements in Practical Fourier Analysis and Their Application to X-ray Scattering from Liquids", J. Franklin Inst., vol. 233, (April 1942), pp. 365-380 and pp. 435-452.

3. FORT - "Complex Finite Fourier Transform Subroutine", SDA no. 3465, SHARE Distribution Agency, Program Information Dept., IBM Corp., 40 Saw Mill River Rd., Hawthorne, New York 10532.

4. Gentleman, W. M., and G. Sande, "Fast Fourier Transforms--For Fun and Profit", Proceedings of the Fall Joint Computer Conference, San Francisco, California, Nov. 8-10, 1966.

5. HARM - "Harmonic Analysis Subroutine for IBM 7094, SDA no. 3425". SHARE Distribution Agency. (see ref. III)