arXiv:math/0302212v1 [math.HO] 18 Feb 2003

Graphical explanation for the speed of the Fast Fourier Transform

Randall D. Peters

Physics Department

Mercer University

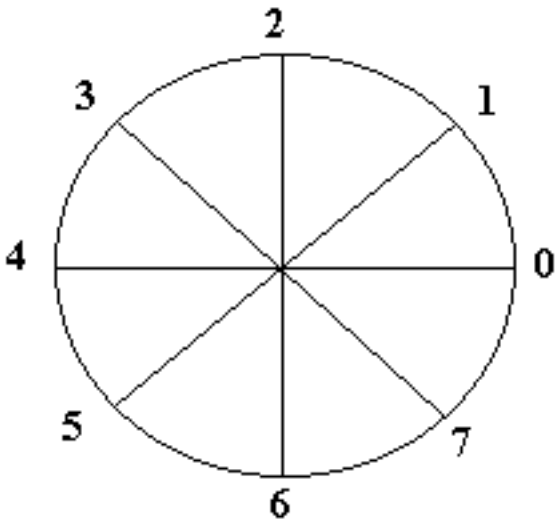
1400 Coleman Ave

Macon, Georgia 31207

Abstract

For a sample set of 1024 values, the FFT is 102.4 times faster than the discrete Fourier transform (DFT). The basis for this remarkable speed advantage is the `bit-reversal' scheme of the Cooley-Tukey algorithm. Eliminating the burden of `degeneracy' by this means is readily understood using vector graphics.

The key to the power of the fast Fourier transform (FFT), as compared to the discrete Fourier transform (DFT), is the bit reversal scheme of the Cooley-Tukey algorithm [1]. It is illustrated very simply as follows. Instead of a practically-sized number of samples in the record to be transformed, consider the pedagogically useful n = 8, distributed on the unit circle as shown in Fig. 1.



decimal (bit-reversed)	binary number	decimal (usual)
0	000	0
4	001	1
2	010	2
6	011	3
1	100	4
5	101	5
3	110	6
7	111	7

Figure 1. Means for understanding why the Cooley -Tukey FFT algorithm is so much faster than the outdated DFT.

Observe that the roots of unity in the complex plane, which have been numbered 0 through 7, `slice the pie' into 8 equal pieces. Such division requires that the algorithm be expressible as a power of 2; i.e., $n = 2^p$, where for the figure $p = 3$. For reasonable resolution, the number of sampled points must typically be greater than 512; i.e., $p > 9$.

The usual decimal counting scheme for the 8 'vectors' is as indicated, traversing the phasor diagram (circle on left) sequentially. In the Cooley-Tukey algorithm, the bits of the binary representation of the vector are reversed according to significance. Usually, the least significant bit is on the right and the most significant bit on the left; so that decimal counting is as shown on the right column of the table--from 0 to 7. With bit reversal, 'lsb' becomes the left most binary digit and the 'msb' is the right most digit. Thus, for example binary 110 (usually 6) becomes 3. With this bit reversal scheme, the phasor diagram is not traversed in the usual sequential, circulatory sense; rather there are 'flip-flops' across the circle. By this `book-keeping' means, there is no needless repetition in the calculation of vector components (real and imaginary values of a given term in the transform). For example, 5 is the simple negative of 1, because of the inversion symmetry. We see that the number of `independent' vectors has been immediately reduced by a factor of 2 through the use of bit-reversal.

There is enormous `degeneracy' (needless repetitive calculations) when one `blindly' calculates the DFT. With typically-sized sample sets, huge reduction in execution time is possible by recognizing (i) the inversion symmetries emphasized in Figure 1 above, and also (ii) properties of orthogonality. Because of the latter, there is further reduction realized by the Cooley-Tukey algorithm. Orthogonality of various pairs permits the determination of the components of one vector from another by simply interchanging

sine terms with cosine terms and vice-versa. Thus, for a large fraction of the vectors in the set, we can determine the components of a given vector from those of another vector by simply reversing algebraic sign and/or interchanging sine and cosine terms.

From the discussion above, it is seen (from a computation viewpoint) that there are only 3 `unique' vectors in the 8-vector set of Figure 1. For a 16-point sample set, the number of `unique' vectors is readily seen to be 4. Extending to the general case, it is seen that the number of `independent' vectors m, for which complete trigonometric calculation is necessary to obtain the FFT, is given by

$$m = p < n = 2^p \qquad (1)$$

In calculating each of the components (real and imaginary pair) of the Fourier transform--one must use all n vectors of the sampled set. With the DFT as described in [2], it is readily seen that the total number of operations is thus given by

$$N_{DFT} = n \times n = n^2 \qquad (2)$$

since all n vectors are treated as independent. With the FFT, on the other hand, since the number of independent vectors has been reduced from n to p; the total number of operations (involving time-consuming sine, cosine calculations) becomes

$$N_{FFT} = p \times n = n \log_2 n \qquad (3)$$

Since the CPU processing time required to calculate the Fourier transform is proportional to N, we see that the speed advantage of the FFT over the DFT is given by

$$\text{Speed advantage} = N_{DFT} / N_{FFT} = n / \log_2 n \qquad (4)$$

The savings to be realized with the FFT is dramatic with typical values of n. For example, with a 1K set of samples ($n = 2^{10} = 1024$), we see from Eq. (4) that the algorithm is 102.4 times faster.

References:

1. Barry Cipra, "The FFT: Making Technology Fly", SIAM News, Vol. 26, No. 3, May 1993--online at

http://www.siam.org/siamnews/mtc/mtc593.htm

2. R. Peters, "Fourier transform construction by vector graphics", Am. J. Phys. 60, 439-441 (1992).