The DFT as Convolution or Filtering*

C. Sidney Burrus

This work is produced by OpenStax-CNX and licensed under the
Creative Commons Attribution License 2.0†

A major application of the FFT is fast convolution or fast filtering where the DFT of the signal is multiplied term-by-term by the DFT of the impulse (helps to be doing finite impulse response (FIR) filtering) and the time-domain output is obtained by taking the inverse DFT of that product. What is less well-known is the DFT can be calculated by convolution. There are several different approaches to this, each with different application.

1 Rader's Conversion of the DFT into Convolution

In this section a method quite different from the index mapping or polynomial evaluation is developed. Rather than dealing with the DFT directly, it is converted into a cyclic convolution which must then be carried out by some efficient means. Those means will be covered later, but here the conversion will be explained. This method requires use of some number theory, which can be found in an accessible form in [12] or [14] and is easy enough to verify on one's own. A good general reference on number theory is [13].

The DFT and cyclic convolution are defined by

$$C(k) = \sum_{n=0}^{N-1} x(n) \; W^{nk} \tag{1}$$

$$y(k) = \sum_{n=0}^{N-1} x(n) \; h(k-n) \tag{2}$$

For both, the indices are evaluated modulo N. In order to convert the DFT in (1) into the cyclic convolution of (2), the nk product must be changed to the $k-n$ difference. With real numbers, this can be done with logarithms, but it is more complicated when working in a finite set of integers modulo N. From number theory [1], [12], [14], [13], it can be shown that if the modulus is a prime number, a base (called a primitive root) exists such that a form of integer logarithm can be defined. This is stated in the following way. If N is a prime number, a number r called a primitive roots exists such that the integer equation

$$n = \left(\left(r^m\right)\right)_N \tag{3}$$

creates a unique, one-to-one map of the $N-1$ member set $m = \{0, ..., N-2\}$ and the $N-1$ member set $n = \{1, ..., N-1\}$. This is because the multiplicative group of integers modulo a prime, p, is isomorphic to the additive group of integers modulo $(p-1)$ and is illustrated for $N = 5$ below.

*Version 1.9: Sep 21, 2009 4:28 pm +0000
†http://creativecommons.org/licenses/by/2.0/

r	m=	0	1	2	3	4	5	6	7
1		1	1	1	1	1	1	1	1
2		1	2	4	3	1	2	4	3
3		1	3	4	2	1	3	4	2
4		1	4	1	4	1	4	1	4
5		*	0	0	0	*	0	0	0
6		1	1	1	1	1	1	1	1

Table 1: Table of Integers $n = ((r^m))$ modulo 5, [* not defined]

Table 1 is an array of values of r^m modulo N and it is easy to see that there are two primitive roots, 2 and 3, and (3) defines a permutation of the integers n from the integers m (except for zero). (3) and a primitive root (usually chosen to be the smallest of those that exist) can be used to convert the DFT in (1) to the convolution in (2). Since (3) cannot give a zero, a new length-(N-1) data sequence is defined from $x(n)$ by removing the term with index zero. Let

$$n = r^{-m} \tag{4}$$

and

$$k = r^s \tag{5}$$

where the term with the negative exponent (the inverse) is defined as the integer that satisfies

$$\left(\left(r^{-m} r^m\right)\right)_N = 1 \tag{6}$$

If N is a prime number, r^{-m} always exists. For example, $\left(\left(2^{-1}\right)\right)_5 = 3$. (1) now becomes

$$C\left(r^s\right) = \sum_{m=0}^{N-2} x\left(r^{-m}\right)\ W^{r^{-m} r^s} +\ x(0), \tag{7}$$

for $s = 0, 1, .., N-2$, and

$$C(0) = \sum_{n=0}^{N-1} x(n) \tag{8}$$

New functions are defined, which are simply a permutation in the order of the original functions, as

$$x^{'}(m) = x\left(r^{-m}\right), \quad C^{'}(s) = C\left(r^s\right), \quad W^{'}(n) = W^{r^n} \tag{9}$$

(7) then becomes

$$C^{'}(s) = \sum_{m=0}^{N-2} x^{'}(m)\ W^{'}(s-m)\ +\ x(0) \tag{10}$$

which is cyclic convolution of length N-1 (plus $x(0)$) and is denoted as

$$C^{'}(k) = x^{'}(k) * W^{'}(k) + x(0) \tag{11}$$

Applying this change of variables (use of logarithms) to the DFT can best be illustrated from the matrix formulation of the DFT. (1) is written for a length-5 DFT as

$$\begin{bmatrix} C(0) \\ C(1) \\ C(2) \\ C(3) \\ C(4) \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 3 & 4 \\ 0 & 2 & 4 & 1 & 3 \\ 0 & 3 & 1 & 4 & 2 \\ 0 & 4 & 3 & 2 & 1 \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \\ x(4) \end{bmatrix} \tag{12}$$

where the square matrix should contain the terms of W^{nk} but for clarity, only the exponents nk are shown. Separating the $x(0)$ term, applying the mapping of (9), and using the primitive roots $r = 2$ (and $r^{-1} = 3$) gives

$$\begin{bmatrix} C(1) \\ C(2) \\ C(4) \\ C(3) \end{bmatrix} = \begin{bmatrix} 1 & 3 & 4 & 2 \\ 2 & 1 & 3 & 4 \\ 4 & 2 & 1 & 3 \\ 3 & 4 & 2 & 1 \end{bmatrix} \begin{bmatrix} x(1) \\ x(3) \\ x(4) \\ x(2) \end{bmatrix} + \begin{bmatrix} x(0) \\ x(0) \\ x(0) \\ x(0) \end{bmatrix} \tag{13}$$

and

$$C(0) = x(0) + x(1) + x(2) + x(3) + x(4) \tag{14}$$

which can be seen to be a reordering of the structure in (12). This is in the form of cyclic convolution as indicated in (10). Rader first showed this in 1968 [12], stating that a prime length-N DFT could be converted into a length-(N-1) cyclic convolution of a permutation of the data with a permutation of the W's. He also stated that a slightly more complicated version of the same idea would work for a DFT with a length equal to an odd prime to a power. The details of that theory can be found in [12], [10].

Until 1976, this conversion approach received little attention since it seemed to offer few advantages. It has specialized applications in calculating the DFT if the cyclic convolution is done by distributed arithmetic table look-up [5] or by use of number theoretic transforms [1], [12], [14]. It and the Goertzel algorithm [16], [4] are efficient when only a few DFT values need to be calculated. It may also have advantages when used with pipelined or vector hardware designed for fast inner products. One example is the TMS320 signal processing microprocessor which is pipelined for inner products. The general use of this scheme emerged when new fast cyclic convolution algorithms were developed by Winograd [21].

2 The Chirp Z-Transform (or Bluestein's Algorithm)

The DFT of $x(n)$ evaluates the Z-transform of $x(n)$ on N equally spaced points on the unit circle in the z plane. Using a nonlinear change of variables, one can create a structure which is equivalent to modulation and filtering $x(n)$ by a "chirp" signal. [2], [20], [18], [16], [19], [4].

The mathematical identity $(k-n)^2 = k^2 - 2kn + n^2$ gives

$$nk = \left(n^2 - (k-n)^2 + k^2\right)/2 \tag{15}$$

which substituted into the definition of the DFT in Multidimensional Index Mapping: Equation 1[1] gives

$$C(k) = \{\sum_{n=0}^{N-1} \left[x(n)\ W^{n^2/2}\right]\ W^{-(k-n)^2/2}\}\ W^{k^2/2} \tag{16}$$

[1] "Multidimensional Index Mapping", (1) <http://cnx.org/content/m16326/latest/#uid1>

This equation can be interpreted as first multiplying (modulating) the data $x(n)$ by a chirp sequence ($W^{n^2/2}$, then convolving (filtering) it, then finally multiplying the filter output by the chirp sequence to give the DFT.

Define the chirp sequence or signal as $h(n) = W^{n^2/2}$ which is called a chirp because the squared exponent gives a sinusoid with changing frequency. Using this definition, (16) becomes

$$C(n) = \{[x(n) \;\; h(n)] \;\; * \;\; h^{-1}\} \; h(n) \tag{17}$$

We know that convolution can be carried out by multiplying the DFTs of the signals, here we see that evaluation of the DFT can be carried out by convolution. Indeed, the convolution represented by $*$ in (17) can be carried out by DFTs (actually FFTs) of a larger length. This allows a prime length DFT to be calculated by a very efficient length-2^M FFT. This becomes practical for large N when a particular non-composite (or N with few factors) length is required.

As developed here, the chirp z-transform evaluates the z-transform at equally spaced points on the unit circle. A slight modification allows evaluation on a spiral and in segments [18], [16] and allows savings with only some input values are nonzero or when only some output values are needed. The story of the development of this transform is given in [19].

Two Matlab programs to calculate an arbitrary length DFT using the chirp z-transform is shown in p. ??.

```
        function y = chirpc(x);
% function y = chirpc(x)
% computes an arbitrary-length DFT with the
% chirp z-transform algorithm.  csb.  6/12/91
%
N  = length(x);  n = 0:N-1;     %Sequence length
W  = exp(-j*pi*n.*n/N);         %Chirp signal
xw = x.*W;                      %Modulate with chirp
WW = [conj(W(N:-1:2)),conj(W)]; %Construct filter
y  = conv(WW,xw);               %Convolve w filter
y  = y(N:2*N-1).*W;             %Demodulate w chirp


function y = chirp(x);
% function y = chirp(x)
% computes an arbitrary-length Discrete Fourier Transform (DFT)
% with the chirp z transform algorithm. The linear convolution
% then required is done with FFTs.
% 1988: L. Arevalo; 11.06.91 K. Schwarz, LNT Erlangen; 6/12/91 csb.
%
N   = length(x);                %Sequence length
L   = 2^ceil(log((2*N-1))/log(2)); %FFT length
n   = 0:N-1;
W   = exp(-j*pi*n.*n/N);        %Chirp signal
FW  = fft([conj(W), zeros(1,L-2*N+1), conj(W(N:-1:2))],L);
y   = ifft(FW.*fft(x.'.*W,L));  %Convolve using FFT
y   = y(1:N).*W;                %Demodulate
```

Figure 1

3 Goertzel's Algorithm (or A Better DFT Algorithm)

Goertzel's algorithm [7], [4], [15] is another methods that calculates the DFT by converting it into a digital filtering problem. The method looks at the calculation of the DFT as the evaluation of a polynomial on the unit circle in the complex plane. This evaluation is done by Horner's method which is implemented recursively by an IIR filter.

3.1 The First-Order Goertzel Algorithm

The polynomial whose values on the unit circle are the DFT is a slightly modified z-transform of x(n) given by

$$X(z) = \sum_{n=0}^{N-1} x(n)\, z^{-n} \tag{18}$$

which for clarity in this development uses a positive exponent . This is illustrated for a length-4 sequence as a third-order polynomial by

$$X(z) = x(3)\,z^3 + x(2)\,z^2 + x(1)\,z + x(0) \tag{19}$$

The DFT is found by evaluating (18) at $z = W^k$, which can be written as

$$C(k) = X(z)\,|_{z=W^k} = DFT\{x(n)\} \tag{20}$$

where

$$W = e^{-j2\pi/N} \tag{21}$$

The most efficient way of evaluating a general polynomial without any pre-processing is by "Horner's rule" [11] which is a nested evaluation. This is illustrated for the polynomial in (19) by

$$X(z) = \{[x(3)\,z + x(2)]\,z + x(1)\}z + x(0) \tag{22}$$

This nested sequence of operations can be written as a linear difference equation in the form of

$$y(m) = z\,y(m-1) + x(N-m) \tag{23}$$

with initial condition $y(0) = 0$, and the desired result being the solution at $m = N$. The value of the polynomial is given by

$$X(z) = y(N)\,. \tag{24}$$

(23) can be viewed as a first-order IIR filter with the input being the data sequence in reverse order and the value of the polynomial at z being the filter output sampled at $m = N$. Applying this to the DFT gives the Goertzel algorithm [17], [15] which is

$$y(m) = W^k y(m-1) + x(N-m) \tag{25}$$

with $y(0) = 0$ and

$$C(k) = y(N) \tag{26}$$

where

$$C(k) = \sum_{n=0}^{N-1} x(n)\;W^{nk}. \tag{27}$$

The flowgraph of the algorithm can be found in [4], [15] and a simple FORTRAN program is given in the appendix.

When comparing this program with the direct calculation of (27), it is seen that the number of floating-point multiplications and additions are the same. In fact, the structures of the two algorithms look similar, but close examination shows that the way the sines and cosines enter the calculations is different. In (27), new sine and cosine values are calculated for each frequency and for each data value, while for the Goertzel algorithm in (25), they are calculated only for each frequency in the outer loop. Because of the recursive or feedback nature of the algorithm, the sine and cosine values are "updated" each loop rather than recalculated. This results in $2N$ trigonometric evaluations rather than $2N^2$. It also results in an increase in accumulated quantization error.

It is possible to modify this algorithm to allow entering the data in forward order rather than reverse order. The difference (23) becomes

$$y(m) = z^{-1} y(m-1) + x(m-1) \tag{28}$$

if (24) becomes

$$C(k) = z^{N-1} y(N) \tag{29}$$

for $y(0) = 0$. This is the algorithm programmed later.

3.2 The Second-Order Goertzel Algorithm

One of the reasons the first-order Goertzel algorithm does not improve efficiency is that the constant in the feedback or recursive path is complex and, therefore, requires four real multiplications and two real additions. A modification of the scheme to make it second-order removes the complex multiplications and reduces the number of required multiplications by two.

Define the variable $q(m)$ so that

$$y(m) = q(m) - z^{-1} q(m-1). \tag{30}$$

This substituted into the right-hand side of (23) gives

$$y(m) = z\, q(m-1) - q(m-2) + x(N-m). \tag{31}$$

Combining (30) and (31) gives the second order difference equation

$$q(m) = \left(z + z^{-1}\right) q(m-1) - q(m-2) + x(N-m) \tag{32}$$

which together with the output (30), comprise the second-order Goertzel algorithm where

$$X(z) = y(N) \tag{33}$$

for initial conditions $q(0) = q(-1) = 0$.

A similar development starting with (28) gives a second-order algorithm with forward ordered input as

$$q(m) = \left(z + z^{-1}\right) q(m-1) - q(m-2) + x(m-1) \tag{34}$$

$$y(m) = q(m) - z\, q(-1) \tag{35}$$

with

$$X(z) = z^{N-1} y(N) \tag{36}$$

and for $q(0) = q(-1) = 0$.

Note that both difference (32) and (34) are not changed if z is replaced with z^{-1}, only the output (30) and (35) are different. This means that the polynomial $X(z)$ may be evaluated at a particular z and its inverse z^{-1} from one solution of the difference (32) or (34) using the output equations

$$X(z) = q(N) - z^{-1} q(N-1) \tag{37}$$

and

$$X(1/z) = z^{N-1} \left(q(N) - z\, q(N-1)\right). \tag{38}$$

Clearly, this allows the DFT of a sequence to be calculated with half the arithmetic since the outputs are calculated two at a time. The second-order DE actually produces a solution $q(m)$ that contains two first-order components. The output equations are, in effect, zeros that cancel one or the other pole of the second-order solution to give the desired first-order solution. In addition to allowing the calculating of two outputs at a time, the second-order DE requires half the number of real multiplications as the first-order form. This is because the coefficient of the $q(m-2)$ is unity and the coefficient of the $q(m-1)$ is real if z and z^{-1} are complex conjugates of each other which is true for the DFT.

3.3 Analysis of Arithmetic Complexity and Timings

Analysis of the various forms of the Goertzel algorithm from their programs gives the following operation count for real multiplications and real additions assuming real data.

Algorithm	Real Mults.	Real Adds	Trig Eval.
Direct DFT	$4\,N^2$	$4\,N^2$	$2\,N^2$
First-Order	$4\,N^2$	$4\,N^2 - 2N$	$2\,N$
Second-Order	$2\,N^2 + 2N$	$4\,N^2$	$2\,N$
Second-Order 2	$N^2 + N$	$2\,N^2 + N$	N

Table 2

Timings of the algorithms on a PC in milliseconds are given in the following table.

Algorithm	$N = 125$	$N = 257$
Direct DFT	4.90	19.83
First-Order	4.01	16.70
Second-Order	2.64	11.04
Second-Order 2	1.32	5.55

Table 3

These timings track the floating point operation counts fairly well.

3.4 Conclusions

Goertzel's algorithm in its first-order form is not particularly interesting, but the two-at-a-time second-order form is significantly faster than a direct DFT. It can also be used for any polynomial evaluation or for the DTFT at unequally spaced values or for evaluating a few DFT terms. A very interesting observation is that the inner-most loop of the Glassman-Ferguson FFT [6] is a first-order Goertzel algorithm even though that FFT is developed in a very different framework.

In addition to floating-point arithmetic counts, the number of trigonometric function evaluations that must be made or the size of a table to store precomputed values should be considered. Since the value of the W^{nk} terms in (23) are iteratively calculate in the IIR filter structure, there is round-off error accumulation that should be analyzed in any application.

It may be possible to further improve the efficiency of the second-order Goertzel algorithm for calculating all of the DFT of a number sequence. Perhaps a fourth order DE could calculate four output values at a time and they could be separated by a numerator that would cancel three of the zeros. Perhaps the algorithm could be arranged in stages to give an $N\,log\,(N)$ operation count. The current algorithm does not take into account any of the symmetries of the input index. Perhaps some of the ideas used in developing the QFT [3], [8], [9] could be used here.

4 The Quick Fourier Transform (QFT)

One stage of the QFT can use the symmetries of the sines and cosines to calculate a DFT more efficiently than directly implementing the definition Multidimensional Index Mapping: Equation 1[2]. Similar to the Goertzel

[2] "Multidimensional Index Mapping", (1) <http://cnx.org/content/m16326/latest/#uid1>

algorithm, the one-stage QFT is a better N^2 DFT algorithm for arbitrary lengths. See The Cooley-Tukey Fast Fourier Transform Algorithm: The Quick Fourier Transform, An FFT based on Symmetries[3].

References

[1] Richard E. Blahut. *Fast Algorithms for Digital Signal Processing.* Addison-Wesley, Reading, Mass., 1985.

[2] L. I. Bluestein. A linear filtering approach to the computation of discrete fourier transform. *IEEE Transactions on Audio Electroacoustics*, AU-18:4518211;455, December 1970.

[3] C. S. Burrus. The quick fourier transform. Unpublished notes, ECE Dept., Rice University, 1992.

[4] C. S. Burrus and T. W. Parks. *DFT/FFT and Convolution Algorithms.* John Wiley & Sons, New York, 1985.

[5] Shuni Chu and C. S. Burrus. A prime factor fft algorithm using distributed arithmetic. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 30(2):2178211;227, April 1982.

[6] W. E. Ferguson, Jr. A simple derivation of glassman general-n fast fourier transform. *Comput. and Math. with Appls.*, 8(6):4018211;411, 1982. Also, in Report AD-A083811, NTIS, Dec. 1979.

[7] G. Goertzel. An algorithm for the evaluation of finite trigonometric series. *Amer. Math. Monthly*, 65:34–35, January 1958.

[8] H. Guo, G. A. Sitton, and C. S. Burrus. The quick discrete fourier transform. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, page III:4458211;448, IEEE ICASSP-94, Adelaide, Australia, April 198211;22 1994.

[9] H. Guo, G. A. Sitton, and C. S. Burrus. The quick fourier transform: an fft based on symmetries. *IEEE Transactions on Signal Processing*, 46(2):3358211;341, February 1998.

[10] M. T. Heideman. *Fast Algorithms for the DFT and Convolution with Constrained Inputs and Restricted Outputs.* Ph. d. thesis, Rice University, May 1986.

[11] Donald E. Knuth. *The Art of Computer Programming, Vol. 2, Seminumerical Algorithms.* Addison-Wesley, Reading, MA, third edition, 1997.

[12] J. H. McClellan and C. M. Rader. *Number Theory in Digital Signal Processing.* Prentice-Hall, Englewood Cliffs, NJ, 1979.

[13] Ivan Niven and H. S. Zuckerman. *An Introduction to the Theory of Numbers.* John Wiley & Sons, New York, fourth edition, 1980.

[14] H. J. Nussbaumer. *Fast Fourier Transform and Convolution Algorithms.* Springer-Verlag, Heidelberg, Germany, second edition, 1981, 1982.

[15] A. V. Oppenheim and R. W. Schafer. *Discrete-Time Signal Processing.* Prentice-Hall, Englewood Cliffs, NJ, 1989.

[16] A. V. Oppenheim and R. W. Schafer. *Discrete-Time Signal Processing.* Prentice-Hall, Englewood Cliffs, NJ, second edition, 1999. Earlier editions in 1975 and 1989.

[17] T. W. Parks and C. S. Burrus. *Digital Filter Design.* John Wiley & Sons, New York, 1987.

[3] "The Cooley-Tukey Fast Fourier Transform Algorithm": Section The Quick Fourier Transform, An FFT based on Symmetries <http://cnx.org/content/m16334/latest/#uid26>

[18] L. R. Rabiner and B. Gold. *Theory and Application of Digital Signal Processing*. Prentice-Hall, Englewood Cliffs, NJ, 1975.

[19] Lawrence Rabiner. The chirp z-transform algorithm: a lesson in serendipity. *IEEE Signal Processing Magazine*, 24:1188211;119, March 2004.

[20] L.R. Rabiner, R.W. Schafer, and C.M. Rader. The chirp z-transform algorithm. *IEEE Transactions on Audio Electroacoustics*, AU-17:868211;92, June 1969.

[21] S. Winograd. On computing the discrete fourier transform. *Proc. National Academy of Sciences, USA*, 73:10068211;1006, April 1976.