1. DFT Rules

In §6.6 we discuss the discrete Fourier transforms. Given N complex numbers, $\{f(n)\}_{n=0}^{N-1}$, their N-point discrete Fourier transform (DFT), denoted by $\{F(k)\}$, is given by (6.6.2), where F_k and f_j are now written as $F(k)$ and $f(n)$, respectively. In matrix notation, this relationship, which is represented by (6.6.4), can be written as $\mathbf{F} = \mathcal{F}_N\{\mathbf{f}\}$, where $\mathbf{F} = [F(0) \quad F(1) \quad \dots \quad F(N-1)]^T$ and $\mathbf{f} = [f(0) \quad f(1) \quad \dots \quad f(N-1)]^T$, and $\mathcal{F}_N$ represents the $N \times N$ matrix

$$\mathcal{F}_N = \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & W_N & W_n^2 & \dots & W_N^{N-1} \\ 1 & W_n^2 & W_N^4 & \dots & W_N^{2N-2} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & W_N^{N-1} & W_n^{2N-2} & \dots & W_N^{(N-1)(N-1)} \end{bmatrix},$$

where $W_N = e^{-2i\pi/N}$. Thus, $\mathcal{F}_{16}$ denotes the 16×16 matrix. We use the matrix $\mathcal{F}_N$ when we compute a DFT, but we want to avoid generating and storing the N complex numbers N^{-1}, $N^{-1}W_N$, $\dots$, $N^{-1}W_N^{N-1}$ complex numbers, which come to N^2 elements of $\mathcal{F}_N$. Since the function $N\,F(k) = f(0) + f(1)\,W_N^k + f(2)\,\left(W_N^k\right)^2 + \cdots + f(N-1)\,\left(W_N^k\right)^{N-1}$ is a polynomial with the coefficients $f(0), f(1), \dots , f(N-1)$, and the argument W_N^k, we can use Horner's algorithm for computing the DFT, which is as follows:

$x = 1$
$W = e^{-2i\pi/N}$
For $k = 0, 1, \dots , N-1$ do
    $S = f(N-1)$
    For $m = 2, 3, \dots , N$ do
        $S = f(N-m) + x \cdot S$
    $F(k) = S/N$
    $x = x \cdot W$

The DFT rules to compute FFT are as follows:

1. Linearity Rule: $g(n) = c_1\, f_1(n) + \cdots + c_m\, f_m(n)$
$$\text{has the FT } G(k) = c_1\, F_1(k) + \cdots + c_m\, F_m(k). \tag{1.1}$$
2. Reflection Rule: $g(n) = f(-n)$ has the FT $G(k) = F(-k)$; (1.2)
3. Conjugation Rule: $g(n) = f^*(n)$ has the FT $G(k) = F^*(-k)$; (1.3)
4. Translation Rule: $g(n) = f\,(n - n_0)$
$$\text{has the FT } G(k) = W_N^{kn_0}\, F(k), \quad n_0 = 0, \pm 1, \pm 2, \ldots; \tag{1.4}$$
5. Modulation Rule: $g(n) = W_N^{kn_0} f(n)$
$$\text{has the FT } G(k) = F\,(k - k_0)\,, \quad k_0 = 0, \pm 1, \pm 2, \ldots; \tag{1.5}$$
6. Convolution Rule: $g(n) = (f_1 * f_2)\,(n)$
$$\text{has the FT } G(k) = N\, F_1(k) \cdot F_2(k); \tag{1.6}$$
7. Multiplication Rule: $g(n) = f_1(n) \cdot f_2(n)$
$$\text{has the FT } G(k) = (F_1 * F_2)\,(k); \tag{1.7}$$
8. Inversion Rule: $g(n) = F(k)$ has the FT $G(k) = \dfrac{1}{N}\, f(-k)$; (1.8)
9. Zero Packing Rule: $g(n) = \begin{cases} f(n/m) & n = 0, \pm m, \pm 2m, \ldots, \\ 0 & \text{otherwise,} \end{cases}$
$$\text{has the FT } G(k) = \frac{1}{m}\, F(k),; \tag{1.9}$$
10. Summation Rule: $g(n) = \displaystyle\sum_{j=0}^{m-1} f(n - jN)$
$$\text{has the FT } G(k) = m\, F(mk), \tag{1.10}$$

with g on P_N in rules 9, and 11, f on $P_{N/m}$ in rule 9, and f on $P_{m\cdot N}$ in rule 10, where P_N denotes a discrete polygon with N equally spaced points lying on a circle.

Example 1.1. Let (a, b, c, d) have the DFT (A, B, C, D). Then by using the translation rule (1.4) we get

$$(d, a, b, c) \text{ has the DFT } \left(A, W_4 B, W_4^2 C, W_4^3 D\right),$$

where $W_4 = e^{-2i\pi/4}$, and the zero packing rule (1.9) gives

$$(a, 0, b, 0, c, 0, d, 0) \text{ had the DFT } \tfrac{1}{2}\,(A, B, C, D, A, B, C, D). \quad \blacksquare$$

Let $\mathcal{T}$, $\mathcal{E}$, $\mathcal{Z}$ and $\mathcal{R}$ denote the translation, exponential modulation, zero packing and repeat operator, respectively. Then, a 4-component initial vector (a, b, c, d) of Example 1.1 has the following assembly:

$$\begin{array}{ccc} (a,b,c,d) & \xrightarrow{\mathcal{F}} & (A,B,C,D) \\ \mathcal{T}\downarrow & & \downarrow\mathcal{E} \\ (d,a,b,c) & \xrightarrow{\mathcal{F}} & \left(A,W_4B,W_4^2C,W_4^3D\right) \end{array}$$

$$\begin{array}{ccc} (a,b,c,d) & \xrightarrow{\mathcal{F}} & (A,B,C,D) \\ \mathcal{Z}\downarrow & & \downarrow\frac{1}{2}\mathcal{R} \\ (a,0,b,0,c,0,d,0) & \xrightarrow{\mathcal{F}} & \frac{1}{2}\,(A,B,C,D,A,B,C,D) \end{array}$$

EXAMPLE 1.2. Consider an 8-component initial vector (a,b,c,d,e,f,g,h). A decimation-in-frequency scheme is as follows:

When (a,b,c,d,e,f,g,h) has the DFT (A,B,C,D,E,F,G,H), then by using the modulation rule (1.5) we find that

$$\left(a,Wb,W^2c,W^3d,W^4e,W^5f,W^6g,W^7h\right) \text{ has the DFT } (B,C,D,E,F,G,H,A),$$

and by using the summation rule (1.10) we find that

$$\tfrac{1}{2}\,(a+e,b+f,c+g,d+h) \text{ has the DFT } (A,C,E,G).$$

The commutative diagrams for the modulation and summation rules are given below.

$$\begin{array}{ccc} (a,b,c,d,e,f,g,h) & \xrightarrow{\mathcal{F}} & (A,B,C,D,E,F,G,H) \\ \mathcal{E}\downarrow & & \downarrow\mathcal{T} \\ (a,Wb,W^2c,W^3d,W^4e,W^5f,W^6g,W^7h) & \xrightarrow{\mathcal{F}} & (B,C,D,E,F,G,H,A) \end{array}$$

$$\begin{array}{ccc} (a,b,c,d,e,f,g,h) & \xrightarrow{\mathcal{F}} & (A,B,C,D,E,F,G,H) \\ \Sigma\downarrow & & \downarrow\Xi \\ \frac{1}{2}\,(a+c,b+f,c+g,d+h) & \xrightarrow{\mathcal{F}} & (A,C,E,G) \end{array}$$

where Σ and Ξ represents the summation and decimation operator, respectively. ∎

2. Bit Reversal Permutation

Let $N = 8$, and consider the permutation

$$(f_0, f_1, f_2, f_3, f_4, f_5, f_6, f_7) \longrightarrow (f_0, f_4, f_2, f_6, f_1, f_5, f_3, f_7).$$

For the bit reversal permutation consider the radix 2 representation of the integers 1 through 7:

$$\begin{aligned}
0 = (000)_2 &\longrightarrow (000)_2 = 0 \\
1 = (001)_2 &\longrightarrow (100)_2 = 4 \\
2 = (010)_2 &\longrightarrow (010)_2 = 2 \\
3 = (011)_2 &\longrightarrow (110)_2 = 6 \\
4 = (100)_2 &\longrightarrow (001)_2 = 1 \\
5 = (101)_2 &\longrightarrow (101)_2 = 5 \\
6 = (110)_2 &\longrightarrow (011)_2 = 3 \\
7 = (111)_2 &\longrightarrow (111)_2 = 7
\end{aligned}$$

This shows how we get the above permutation.

The entire process can be described as follows: At the first stage we map

$$(f_0, f_1, f_2, f_3, f_4, f_5, f_6, f_7) \longrightarrow (f_0, f_2, f_4, f_6, f_1, f_3, f_5, f_7)$$

by cyclically permuting the three base 2 index bits, placing the lower-order bit (0 for an even index, 1 for an odd index) in the high-order position, i.e, 0 for the first half, 1 for the second half; thus, $f\left[(b_3b_2b_1)\right]$ is placed in the position $(b_1b_3b_2)_2$.

At the second stage we map

$$(f_0, f_2, f_4, f_6, f_1, f_3, f_5, f_7) \longrightarrow (f_0, f_4, f_2, f_6, f_1, f_5, f_3, f_7)$$

Thus, $f\left[(b_3b_2b_1)\right]$ that was in the position $(b_1b_3b_2)_2$ after the first stage is now placed in the position $(b_1b_2b_3)_2$.

For $N = 16$, the bit reversal permutation yields

$$\begin{aligned}
0 = (0000)_2 &\longrightarrow (0000)_2 = 0 & \qquad 8 = (1000)_2 &\longrightarrow (0001)_2 = 1 \\
1 = (0001)_2 &\longrightarrow (1000)_2 = 8 & \qquad 9 = (1001)_2 &\longrightarrow (1001)_2 = 9 \\
2 = (0010)_2 &\longrightarrow (0100)_2 = 4 & \qquad 10 = (1010)_2 &\longrightarrow (0101)_2 = 5 \\
3 = (0011)_2 &\longrightarrow (1100)_2 = 12 & \qquad 11 = (1011)_2 &\longrightarrow (1101)_2 = 13 \\
4 = (0100)_2 &\longrightarrow (0010)_2 = 2 & \qquad 12 = (1100)_2 &\longrightarrow (0011)_2 = 3 \\
5 = (0101)_2 &\longrightarrow (1010)_2 = 10 & \qquad 13 = (1101)_2 &\longrightarrow (1011)_2 = 11 \\
6 = (0110)_2 &\longrightarrow (0110)_2 = 6 & \qquad 14 = (1110)_2 &\longrightarrow (0111)_2 = 7 \\
7 = (0111)_2 &\longrightarrow (1110)_2 = 14 & \qquad 15 = (1111)_2 &\longrightarrow (1111)_2 = 15
\end{aligned}$$

This gives the following bit reversal:

$$(f_0, f_1, f_2, f_3, f_4, f_5, f_6, f_7, f_8, f_9, f_{10}, f_{11}, f_{12}, f_{13}, f_{14}, f_{15})$$
$$\longrightarrow (f_0, f_8, f_4, f_{12}, f_2, f_{10}, f_6, f_{14}, f_1, f_9, f_5, f_{13}, f_3, f_{11}, f_7, f_{15}).$$

In this bit reversal structure, we move the component $f\left[(b_4b_3b_2b_1)_2\right]$ from its original position $(b_4b_3b_2b_1)_2$ successively to the positions $(b_1b_4b_3b_2)_2$, $(b_1b_2b_4b_3)_2$ and $(b_1b_2b_3b_4)_2$. In general, for $N = 2^j$, $j = 1, 2, \ldots$, we move $f\left[(b_jb_{j-1}\ldots b_2b_1)_2\right]$ from the position $(b_1b_2\ldots b_{j-1}b_j)_2$ for every choice of $b_1, b_2, \ldots, b_{j-1}, b_j = 0, 1$. In the case when $s = (b_1b_2\ldots b_{j-1}b_j)_2$ is bit-reversed from $n = (b_jb_{j-1}\ldots b_2b_1)_2$, then n is the bit-reversed form of s, and we interchange the values of $f[s]$ and $f[n]$ to carry out the permutation. The positions of $f[0] = f[(0\,0\,\ldots\,0)_2]$ and $f(N-1) = f[(1\,1\,\ldots,\,1)_2]$ remain unchanged during this process. A simple algorithm for applying the bit reversal permutation to an N-vector function $\mathbf{f}$ for $N = 2^j$ is as follows:

For $n - 1, 2, \ldots, N - 2$ do
  Find the integer $s = (b_1b_2\ldots b_{j-1}b_j)_2$ that corresponds to $n = (b_jb_{j-1}\ldots b_2b_1)_2$
If $s > n$, then interchange $f[s]$ and $f[n]$

EXAMPLE 1.3. Let $j = 5$ and $n = 13 = (01101)_2$. Then

$$\begin{aligned}
13/2 = 6 \text{ with remainder } b_1 = 1, &\qquad s_1 = 1,\\
6/2 = 3 \text{ with remainder } b_2 = 0, &\qquad s_2 = 2s_1 + b_2 = 2,\\
3/2 = 1 \text{ with remainder } b_3 = 1, &\qquad s_3 = 2s_2 + b_3 = 5,\\
1/2 = 0 \text{ with remainder } b_4 = 1, &\qquad s_4 = 2s_3 + b_4 = 11,\\
0/2 = 0 \text{ with remainder } b_5 = 0, &\qquad s_5 = 2s_4 + b_5 = 22,
\end{aligned}$$

which shows that $s = (10110)_2 = 22$. ∎

Given an index $n = (b_jb_{j-1}\ldots b_2b_1)_2 = b_1 + b_2 \cdot 2 + b_3 \cdot 2^2 + \cdots + b_j \cdot 2^{j-1}$, we can generate in tern the bits b_1, b_2, ... and compute the corresponding Horner's sequence $b_1, 2b_1 + b_2, 2(2b_1 + b_2) + b_3, \ldots$ for

$$s = (b_1b_2\ldots b_{j-1}b_j)_2 = b_j + b_{j-1} \cdot 2 + b_{j-2} \cdot 2^2 + \cdots + b_1 \cdot 2^{j-1}.$$

Thus, we use the following algorithm for the bit reversal permutation:

For $n = 1, 2, \ldots, N - 2$ do
  $s = 0$
  $d = n$
  For $k = 1, 2, \ldots, j$ do
    $q = \lfloor d/2 \rfloor$
    $b = d - 2q$
    $s = 2s + b$
    $d = q$
  If $s > n$, then interchange $f(s)$ and $f(n)$

While applying the above algorithm, each time we compute the bit-reversed index $s = s(n)$ from the index n, which computes $s(1), s(2), \ldots, s(N-1)$ in turn, we can improve upon the computation if we use a recursive scheme which uses a known value of $s(n)$ to obtain the value of $s(n+1)$. If n is even, a simple addition computes the bit-reversed index, as the following example shows.

EXAMPLE 1.4. To compute $s(23) = 29$ from $s(22) = 13$ for $N = 2^5$, we find that

	Mirror	
$n = (10110)_2 = 22$	⋮	$(01101)_2 = s(n) = 13$
$+1 = (00001)_2 = 1$	⋮	$+(10000)_2 = N/2 = 16$
	⋮	
$n+1 = (10111)_2 = 23$	⋮	$+(11101)_2 = s(n+1) = 29$

Note that $s(n+1) = s(n) + N/2$ when $s(n) < N/2$. ∎

In the case when n is odd, i.e., when $s(n) \geq N/2$, we can still generate $s(n+1)$ from $s(n)$, but we must mirror the carry associated with base 2 addition, as shown in the following example:

EXAMPLE 1.5. Let $N = 2^5$. We compute $s(12) = 6$ from $s(11) = 26$ bt reverse carry method.

	Mirror	
$n = (11011)_2 = 11$	⋮	$(11010)_2 = s(n) = 26$
$+1 = (00001)_2 = 1$	⋮	$+(10000)_2 = N/2 = 16$
	⋮	
carry ↑	⋮	↑ reverse carry
	⋮	
$n-1 = (01010)_2 = 10$	⋮	$(01010)_2 = s(n) - N/2 = 10$
$+2 = (00010)_2 = 2$	⋮	$+(01000)_2 = N/2 = 8$
	⋮	
carry ↑	⋮	↑ reverse carry
	⋮	
$n-1-2 = (01000)_2 = 8$	⋮	$(00010)_2 = s(n) - N/2 - N/4 = 2$
$+4 = (00100)_2 = 4$	⋮	$+(00100)_2 = N/8 = 4$
	⋮	
$n+1 = (01100)_2 = 12$	⋮	$(00110)_2 = s(n+1) = 6$ ∎

This leads to the REVERSE CARRY ALGORITHM:

$s = 0$
For $n = 1, 2, \ldots, N-2$ do
  $k = N/2$
  While $s \geq k$ do
    $s = s - k$
    $k = k/2$
  $s = s + k$
  If $s > k$, then interchange $f(s)$ and $f(n)$

The above algorithm generates a complete set of n, s pairs for a given $N = 2^j$ by generating $s(n+1)$ from $s(n)$. A better algorithm which is more efficient in storage and speed is the Bracewell-Buneman algorithm, which is explained below.

For each $n = 0, 1, \ldots, 2^j - 1$, $j = 1, 2, \ldots$, the functions $s_j(n)$ are formed by reversing the bits of n such that

$$s_j\left[(b_j b_{j-1} \ldots b_2 b_1)_2\right] = (b_1 b_2 \ldots b_{j-1} b_j)_2.$$

For example $s_2(3) = (11)_2 = 3$, $s_3(3) = (110)_2 = 6$, $s_4(3) = (1100)_2 = 12$, and so on. Since

$$\begin{aligned} s_{j+1}\left[(b_{j+1} b_j \ldots b_2 b_1)_2\right] &= (b_1 b_2 \ldots b_j b_{j+1})_2 \\ &= 2 \cdot (b_1 b_2 \ldots b_{j-1} b_j)_2 + b_{j+1} \\ &= 2 s_j\left[(b_j b_{j-1} \ldots b_2 b_1)_2\right] + b_{j+1}, \end{aligned}$$

we obtain

$$s_{j+1}(n) = \begin{cases} 2 s_j(n) & \text{if } n = 0, 1, \ldots, 2^j - 1, \\ 2 s_j\left(n - 2^j\right) + 1 & \text{if } n = 2^j, 2^{j+1}, \ldots, 2^{j+1} - 1. \end{cases}$$

Thus, we obtain the left half of the $(m+1)$-st row from the following table:

n	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$s_1(n)$	0	1														
$s_2(n)$	0	2	1	3												
$s_3(n)$	0	4	2	6	1	5	3	7								
$s_4(n)$	0	8	4	12	2	10	6	14	1	9	5	13	3	11	7	15

by doubling the m-th row. We get the right half of the $(m+1)$-st row by adding 1 to each component of the left half. This leads to the following algorithm for generating the bit-reversed indices $s_j(0), s_j(1), \ldots, s_j(N-1)$, for $N = 2^j$:

$s(0) = 0$
$M = 1$
While $M < N$ do
  For $k = 0, 1, \ldots, M-1$ do
    $T = 2s(k)$
    $s(k) = T$
    $s(k+M) = T+1$
  $M = 2M$

After initializing the array s, the bit reversal permutation can be executed by using

For $n = 1, 2, \ldots, N-2$ do
  If $s(n) > n$, then interchange $s(n)$ and $f(s(n))$.

Bracewell (1986) and Buneman (1986) gave a more efficient algorithm that uses only $\sqrt{2N}$ components of storage. It uses a left-right decomposition of the radix 2 representations of n and s. For example, for $N = 2^6$, a 6-bit index n is represented by

$$n = (b_6 b_5 b_4 b_3 b_2 b_1) = 8p + q,$$

where $p = (b_6 b_5 b_4)_2$ and $q = (b_3 b_2 b_1)_2$. Then $s_6(n) = (b_1 b_2 b_3 b_4 b_5 b_6)_2 = 8s_3(q) + s_3(p)$. All pairs of the 6 bit reversed pairs n, s, with $s > n$, can be obtained from the 3-bit integers p, q, where $8s_3(q) + s_3(p) > 8p + q$, or $s_3(q) > p$. In the case when $s_3(q) = p$ we also have $s_3(p) = q$, thus $s_6(n) = n$. Then every bit-reversed pair n, s with $s > n$ occurs precisely once in the list

$$n = 8p + q, \quad s = 8s_3(q) + s_3(p), \quad q = 1, 2, \ldots, 7, \text{ and } p = 0, 1, \ldots, s_3(q) - 1.$$

In the general case when $N = 2^j$ and $j = 2m$ is even, every bit-reversed pair $n.s$ with $s > n$ occurs precisely once in the list

$$n = 2^m p + q, \quad s = 2^m s_m(q) + s_m(p), \; q = 1, 2, \ldots, 2^m - 1, \text{ and } p = 0, 1, \ldots, s_m(q) - 1.$$

In the case of an odd number of index bits, the above left-right representation of n, s is modified; for example, for $N = 2^7$ we have

$$n = (b_7 b_6 b_5 b_4 b_3 b_2 b_1)_2 = 8p + q,$$

where $p = (b_7 b_6 b_5 b_4)_2$ and $q = (b_3 b_2 b_1)_2 = (0 b_3 b_2 b_1)_2$. Then

$$s_7(n) = (b_1 b_2 b_3 b_4 b_5 b_6 b_7)_2 = 16 s_3(q) + s_4(p) = 8 s_4(q) + s_4(p).$$

This means that every bit-reversed pair n, s with $s > n$ occurs precisely once in the list

$$n = 8p + q, \quad s = 8s_4(q) + s_4(p) \text{ for } q = 1, 2, \ldots, 7, \text{ and } p = 0, 1, \ldots, s_4(q) - 1.$$

Thus, in general, for $N = 2^j$, $j = 2m + 1$ (odd), we find that every bit-reversed pair n, s with $s > n$ occurs precisely once in the list

$$n = 2^m p + q, \quad s = 2^m s_{m+1}(q) + s_{m+1}(p),$$

for $q = 1, 2, \ldots, 2^m - 1$, and $p = 0, 1, \ldots, s_{m+1}(q) - 1$.

These even and odd j representations for the pairs n, s provide the following algorithm for the bit reversal permutation when $N = 2^j$ with $j = 2m + l$ for $m = 1, 2, \dots$ and $l = 0, 1$:

For $q = 1, 2, \dots, 2^m - 1$ do
  For $p = 0, 1, \dots, s_{m+l}(q) - 1$ do
    $n' = 2^m p + q$
    $s' = 2^m s_m(q) + s_{m+l}(p)$
    Interchange $f(n')$ and $f(s')$

However, to avoid the repeated computation of $s_{m+l}(0), s_{m+l}(1), \dots, s_{m+l}(2^m - 1)$, these indices can be generated with a double-add one algorithm and stored in an auxiliary array (which has $2^{m+l} \le \sqrt{2N}$ components) as part of the initialization. Then part of the computation of n', s' is done outside the inner loop, which improves efficiency. The resulting Bracewell-Buneman algorithm for applying the bit reversal permutation to $f(0), f(1), \dots, f(N-1)$, $N = 2^j$, is as follows:

$m_+ = \lfloor (j+1)/2 \rfloor$ (i.e., $m_+ = m + l$)
$M = 1$
$s(0) = 0$
For $i = 1, 2, \dots, m_+$ do
  For $k = 0, 1, \dots, M - 1$ do
    $T = 2s(k)$
    $s(k) = T$
    $s(k + M) = T + 1$
  $M = M + 1$
If j is odd, then $M = M/2$ (i.e., $M = 2^m$)
For $q = 1, 2, \dots, M - 1$ do
  $n' = q - M$
  $s'' = s(q) \cdot M$
  For $p = 0, 1, \dots, s(q) - 1$ do
    $n' = n' + M$ (i.e., $s' = M(p + q)$)
    $s' = s'' + s(p)$ (i.e., $s' = Ms(q) + s(p)$)
    Interchange $f(n')$ and $f(s')$

A Fortran code based on this algorithm is given as `fft.f90` with `ffttest.f90` (on the CD-R); see also `BrucewellBuneman.nb` on the CD-R.