FFT

January 20, 2016

1 1.1. The DFT matrix.

The DFT matrix. By *definition*, the sequence $f(\tau)$ ($\tau = 0, 1, 2, \ldots, N-1$), posesses a discrete Fourier transform $F(\nu)$ ($\nu = 0, 1, 2, \ldots, N-1$), given by

$$F(\nu) = \frac{1}{N} \sum_{\tau=0}^{N-1} f(\tau) e^{-i2\pi(\nu/N)\tau}. \tag{1.1}$$

Of course, this definition can be immediately rewritten in the matrix form as follows

$$\begin{bmatrix} F(1) \\ F(2) \\ \vdots \\ F(N-1) \end{bmatrix} = \frac{1}{\sqrt{N}} \mathcal{F} \begin{bmatrix} f(1) \\ f(2) \\ \vdots \\ f(N-1) \end{bmatrix}, \tag{1.2}$$

where the *DFT* (i.e., the *discrete Fourier transform*) matrix is defined by

$$\mathcal{F} = \frac{1}{\sqrt{N}} \left[\; w^{(k-1)(j-1)} \; \right]_{1 \le k,j \le N} = \frac{1}{\sqrt{N}} \begin{bmatrix} 1 & 1 & 1 & 1 & \cdot & 1 \\ 1 & w & w^2 & w^3 & \cdots & w^{N-1} \\ 1 & w^2 & w^4 & w^6 & \cdots & w^{2(N-1)} \\ 1 & w^3 & w^6 & w^9 & \cdots & w^{3(N-1)} \\ \vdots & \vdots & & & & \vdots \\ 1 & w^{N-1} & w^{2(N-1)} & w^{3(N-1)} & \ldots & w^{(N-1)(N-1)} \end{bmatrix} \tag{1.3}$$

with $w = e^{\frac{2\pi i}{N}}$ being the primitive N-th root of unity.

1.2. The IDFT matrix. To recover N values of the function from its discrete Fourier transform we simply have to invert the DFT matrix to obtain

$$\begin{bmatrix} f(1) \\ f(2) \\ \vdots \\ f(N-1) \end{bmatrix} = \sqrt{N} \mathcal{F}^{-1} \begin{bmatrix} F(1) \\ F(2) \\ \vdots \\ F(N-1) \end{bmatrix}, \tag{1.4}$$

The DFT matrix $\mathcal{F}$ is nicely structured, and it is not quite unexpectable, that the entries of its inverse $\mathcal{F}$ also admit a similar description. It turns out that the matrix $\mathcal{F}$ is *unitary*, which by definition means that its inverse coincides with its conjugate transpose,

$$\mathcal{F}^{-1} = \mathcal{F}^*. \tag{1.5}$$

In other words, rows of $\mathcal{F}$ are orthonormal vectors, i.e.,

$$\sum_{k=0}^{N-1} (w^{\nu})^k \cdot (w^{-\nu'})^k = \begin{cases} N & \nu = \nu' \\ 0 & \text{otherwise} \end{cases} \tag{1.6}$$

Of course this is just a discrete analog of the well-known orthogonality of sinusoids of different frequences, and (1.6) also can be understood as a summation of equally spaced numbers

$$\{1, w^{(\nu-\nu')}, w^{2(\nu-\nu')}, w^{3(\nu-\nu')}, \ldots, w^{(N-1)(\nu-\nu')}\},$$

whose average is clearly zero, if $\nu \neq \nu'$.

A different, polynomial interpretation for (1.5) will be offered below, and next we address the question on how to compute the DFT and IDFT. Formulas (1.2) and (1.5) reduce these problems to just matrix-vector multiplication, which require n^2 multiplications and $n^2 - n$ additions. However, this is still too much for many applications. The only way to speed-up the computation is to exploit the structure of $\mathcal{F}$ and $\mathcal{F}^{-1}$. Before describing the corresponding *Fast Fourier Transform* algorithm it is instructive to highlight an idea allowing to speed-up a large class of the compuational methods, leading to what is called *superfast algorithms*. This is done in the next short section.

2 Divide-and-conquer approach.

For the detailed treatment of the divide-and-conquer method we refer to standard computer science textbooks and here only briefly clarify the crucial points.

Suppose that we have to solve a certain computational problem of size N (for example, to multiply a $N \times N$ matrix $\mathcal{F}$ by a vector). Suppose that we have certain method of doing so, and denote by $C(N)$ its *complexity*, i.e. the number of operations to compute the answer. Now suppose that we succeded to reduce our large problem of size N to two smaller problems of the sizes $\frac{N}{2}$ each, and the process of such a reduction requires $O(N)$ operations. Then we could proceed analogous devision for each of the obtained subproblems, and so on. The question is what would be the total number of operations in the overall procedure. It turns out that this question has a nice answer, and that its derivation is trivial.

Lemma 2.1 *Let b be a certain fixed integer, and N be a power of 2. Then the solution of*

$$C(N) = \begin{cases} b & N = 1 \\ 2C(\frac{N}{2}) + bN & N > 1 \end{cases}$$

is given by $C(N) = bN \log_2 N$.

Proof. Of course,

$$T(N) = bN + 2(T\frac{N}{2}) = bN + bN + 4T(\frac{N}{4}) = \ldots = bN \cdot \log_2 N,$$

implying the statement of Lemma.

The proof is immediate, but it is worth to clarify the idea with the following picture.

In the next section we show that this simple approach is very powerful, and instead of usual multiplication (requiring $O(n^2)$ operations we describe how to divide-and conquer with each of the DFT matrix and its inverse, obtaining $O(n \log n)$ fast Fourier transform algorithm.

3 Fast Fourier transform : complexity

The idea would be to cut the he DFT matrix of size N, which we now may denote as $\mathcal{F}_N$ into four smaller $N/2$ parts, each related to $\mathcal{F}_{\frac{N}{2}}$. However, doing this in a straightforward manner does not seem to lead to anything useful, and a certain reordering of the rows is required. To motivate this, let us introduce a Vandermonde matrix,

$$V(x_1, \ldots, x_n) = \begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^{N-1} \\ 1 & x_2 & x_2^2 & \cdots & x_2^{N-1} \\ \vdots & \vdots & & & \vdots \\ 1 & x_N & x_N^2 & \cdots & x_N^{N-1} \end{bmatrix} \tag{3.7}$$

which is completely describes by its *nodes* $\{x_k\}_{k=1}^N$. Of course the DFT matrix $\mathcal{F}_N$ is the special case of the Vandermonde matrix with the nodes $\{1, w, w^2, \ldots, w^{N-1}\}$. A hint on how to permute the rows of $\mathcal{F}_N$ to develop a divide-and-conquer algorithm can now be inferred from the fact that the nodes of $\mathcal{F}_{\frac{N}{2}}$ are the subset of the nodes of $\mathcal{F}_N$. This gives us a recepie for the row permutation in $\mathcal{F}_N$: collect first all rows with odd numbers, then all rows with even numbers.

Taking into account $w^N = 1$, $w^{\frac{N}{2}} = -1$ we obtain the factorization

$$\mathcal{F}_N = \begin{bmatrix} \text{odd-even} \\ \text{permutation} \end{bmatrix} \begin{bmatrix} \mathcal{F}_{\frac{N}{2}} & \mathcal{F}_{\frac{N}{2}} \\ \mathcal{F}_{\frac{N}{2}}\mathcal{D}_{\frac{N}{2}} & -\mathcal{F}_{\frac{N}{2}}\mathcal{D}_{\frac{N}{2}} \end{bmatrix} =$$

$$\begin{bmatrix} \text{odd-even} \\ \text{permutation} \end{bmatrix} \begin{bmatrix} \mathcal{F}_{\frac{N}{2}} & 0 \\ 0 & \mathcal{F}_{\frac{N}{2}} \end{bmatrix} \begin{bmatrix} I & I \\ \mathcal{D}_{\frac{N}{2}} & -\mathcal{D}_{\frac{N}{2}} \end{bmatrix} \tag{3.8}$$

where

$$\mathcal{D}_{\frac{N}{2}} = \text{ diag } (1, w, w^2, \ldots, w^{\frac{N}{2}-1})$$

Clearly (3.8) redices one DFT of the order N to compution two DFT's of the order $\frac{N}{2}$, and the complexity of this reduction step is $2N$ operations. By lemma 2.1 the complexity of the overall procedure will be no more than $2N \log N$ operations, which compares favorably with the complexity $O(n^2)$ operations of matrix multiplication.

4 Fast Fourier transform : implementation details

Recursion. Of course the procedure of factoring $\mathcal{F}_N$ can be proceeded, resulting in

$$\mathcal{F}_N = \begin{bmatrix} \text{odd-even} \\ \text{permutation} \end{bmatrix} \begin{bmatrix} \begin{bmatrix} \text{odd-even} \\ \text{permutation} \end{bmatrix} & \\ & \begin{bmatrix} \text{odd-even} \\ \text{permutation} \end{bmatrix} \end{bmatrix} \cdots$$

$$\cdots \begin{bmatrix} \begin{bmatrix} I & I \\ \mathcal{D}_{\frac{N}{4}} & -\mathcal{D}_{\frac{N}{4}} \end{bmatrix} & \\ & \begin{bmatrix} I & I \\ \mathcal{D}_{\frac{N}{4}} & -\mathcal{D}_{\frac{N}{4}} \end{bmatrix} \end{bmatrix} \begin{bmatrix} I & I \\ \mathcal{D}_{\frac{N}{2}} & -\mathcal{D}_{\frac{N}{2}} \end{bmatrix} \tag{4.9}$$

This factorization reduces the FFT algorithm to the following two stages : (a) multiplying the input vector $\begin{bmatrix} f(1) & \cdots & f(N) \end{bmatrix}^T$ by $\log N$ matrices consisting of diagonal blocks; (b) an appropriate permutation. We briefly describe an implementation of these two steps.

Dual nodes. A close look at the matrix

$$\begin{bmatrix} I & I \\ \mathcal{D}_{\frac{N}{2}} & -\mathcal{D}_{\frac{N}{2}} \end{bmatrix} \tag{4.10}$$

reveals that each its row contains just two nonzero entres, and moreover, there are *dual nodes*, having these nonzeros entries in the same locations. For example, the nodes 1 and $\frac{N}{2}+1$ are dual, the nodes 2 and $\frac{N}{2}+2$ are dual, and so on. The same observation is valid for the other matrices in (4.9), thus allowing us to draw the following *signal flow graph* for the FFT algorithm

Bit-reverse permutation. A close look at the latter signal flow graph indicates that we obtained a *permuted version* of the DFT of $\{f(\tau)\}$, and the problem is how to unscramble the obtained data to recover a proper oredring. It can be shown that this is easy to do : just write the index ν of $F(\nu)$ in binary, and flip the order of bits. This will give a location of the obtained component $F(\nu)$. We next give an example of doing this.

Natural and inverse order formulations. In fact the algorithm descibed above is totally equivalent to the well-known Cooley-Tukey algorithm, which, however, can be traced back to the (unpublished at his time) work of Gauss.

We gave the so-called *natural order* formulation : one applies an algorithm to the data in a natural order and then unscramble the result.

However, $\mathcal{F}_N$ is a symmetrice matrix, $\mathcal{F}_N^T = \mathcal{F}$. Therefore, transposing (3.8) we obtain another factorization

$$\mathcal{F}_N = \begin{bmatrix} I & \mathcal{D}_{\frac{N}{2}} \\ I & -\mathcal{D}_{\frac{N}{2}} \end{bmatrix} \begin{bmatrix} \mathcal{F}_{\frac{N}{2}} & 0 \\ 0 & \mathcal{F}_{\frac{N}{2}} \end{bmatrix} \begin{bmatrix} \text{odd-even} \\ \text{permutation} \end{bmatrix}^T \tag{4.11}$$

which gives us the possibility first to scramble the input data and then to apply the algorithm. However, by drawing the corresponding signal flow graph, we realize that this althogh the formulas (3.8) and (4.11) are formally different, they lead to exactly the same computation! We show it in the next example.

5 Polynomial interpretation

Evaluation and interpolation. Clearly, an evaluation of the polynomial

$$a(x) = a_0 + a_1 x + \ldots + a_{N-1} x^{N-1} \tag{5.12}$$

at N points $\{x_1, \ldots, x_N\}$ is equivalent to the multiplication of a Vandermonde matrix, $V(x_1, \ldots, x_N)$ by the vector of coefficients.

$$\begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^{N-1} \\ 1 & x_2 & x_2^2 & \cdots & x_2^{N-1} \\ \vdots & \vdots & & & \vdots \\ 1 & x_N & x_N^2 & \cdots & x_N^{N-1} \end{bmatrix} \begin{bmatrix} a_0 \\ \vdots \\ a_{N-1} \end{bmatrix} = \begin{bmatrix} a(x_1) \\ \vdots \\ a(x_N) \end{bmatrix}.$$

Analogously, intepolation (i.e., recovering of the coefficients of a polynomial of degree $N-1$ from its values at N points) is equivalent to the multiplication of the inverse of a Vandermonde matrix,

$$\begin{bmatrix} a(x_1) \\ \vdots \\ a(x_N) \end{bmatrix} = V(x_1, \ldots, x_N)^{-1} \begin{bmatrix} a_0 \\ \vdots \\ a_{N-1} \end{bmatrix}.$$

In particular, the DFT and IDFT matrices correspond to the evaluation and interpolation at very special points - roots of unity. Therefore all basic operations on $\mathcal{F}$ and $\mathcal{F}^*$ admit a nice polynomial interpretation given next. Moreover, polynomial language is important and it was widely used by many authors to generalize and to develop new FFT algorithms.

IDFT matrix. As was obtained above, the matrix $\mathcal{F}$ is unitary, so its inverse is given by its conjugate transpose $\mathcal{F}^*$. In this section we offer one more explanation for this fact. Let us address the more general problem of inversion for a general Vandermonde matrix $V(x_1, \ldots, x_N)$. If we would like the vector $\begin{bmatrix} a_0 & a_1 & \cdots & a_{N-1} \end{bmatrix}^T$ in to be j-th column of the V^{-1} then of course the polynomial in (5.12), which we now may denote by $a_j(x)$ should satisfy

$$a_j(x_k) = 0 \qquad (k \neq j), \tag{5.13}$$

and

$$a(x_j) = 1 \tag{5.14}$$

Since $a_j(x)$ is of degree $N-1$, of course

$$a_j(x) = A_j \prod_{\substack{k=1 \\ k \neq j}}^{N} (x - x_k) \tag{5.15}$$

with some A_j to be determined later.

All the arguments given so far are valid for arbitrary Vandermonde matrices, but $\mathcal{F}$ has very special nodes $\{w^k\}_{k=0}^{N-1}$ which are all roots of $(x^N - 1)$. Hence we can rewrtite (5.15) as

$$a_j(x) = A_j \frac{x^N - 1}{x - w^{j-1}}.$$

This synthetic division can be trivially performed for each j, implying

$$\begin{bmatrix} a_0 \\ \vdots \\ a_{N-1} \end{bmatrix} = \begin{bmatrix} 1 \\ \bar{w}^j \\ \bar{w}^{2j} \\ \vdots \\ \bar{w}^{(N-1)j} \end{bmatrix} \cdot A_j, \tag{5.16}$$

where bar denotes complex conjugation. We have already mentioned that the latter vector of coefficients of (5.12) gives us the j-th column of the inverse, which by looking at (5.16) we now can recognize it as a scaled conjugate transpose of the j-th row of $\mathcal{F}$. Therefore we obtain

$$\mathcal{F}^{-1} = \mathcal{F}^* \cdot \text{ diag } (A_1, A_2, \ldots, A_N). \tag{5.17}$$

where A_j are still to be determined. However, this is easy : just multply (5.17) by $\mathcal{F}$ from the left, obtaining that all of A_j are just ones, implying

$$\mathcal{F}^{-1} = \mathcal{F}^*. \tag{5.18}$$

The Cooley-Tukey FFT algorithm. In section 4 above we described in matrix terms the Cooley-Tukey FFT algorithm, and here we offer its interpretation. As was just mentioned the DFT is equivalent to the evaluation at N roots of unity of the order N :

$$\begin{bmatrix} F(1) \\ \vdots \\ F(N-1) \end{bmatrix} = \mathcal{F}_N \begin{bmatrix} f(0) \\ \vdots \\ f(N-1) \end{bmatrix} = \begin{bmatrix} \mathbf{f}(1) \\ \mathbf{f}(w) \\ \mathbf{f}(w^2) \\ \vdots \\ \mathbf{f}(w^{N-1}) \end{bmatrix}$$

where

$$\mathbf{f}(x) = f(0) + xf(1) + \ldots + x^{N-1}f(N-1)$$

Clearly the problem of evaluation of $f(x)$ at N roots of unity

$$\{1, w, w^2, \ldots, w^{N-1}\}, \tag{5.19}$$

can be divided into two smaller problems of evalaution of $f(x)$ at each of the two following subsets.

$$\{1, w^3, w^5, \ldots, w^{N-2}\} \tag{5.20}$$

$$\{w, w^2, w^4, \ldots, w^{N-1}\} \tag{5.21}$$

- **Evaluation of $\mathbf{f}(x)$ at (5.20).** Let us define two polynomials

$$\mathbf{f}_1(x) = f(0) + f(1)x + f(2)x^2 + \ldots f(\frac{N}{2} - 1)x^{\frac{N}{2}-1}$$

$$\mathbf{f}_2(x) = f(\frac{N}{2}) + f(\frac{N}{2} + 1)x + f(\frac{N}{2} + 2)x^2 + \ldots f(N-1)x^{\frac{N}{2}-1}$$

so that

$$\mathbf{f}(x) = \mathbf{f}_1(x) + x^{\frac{N}{2}}\mathbf{f}_2(x). \tag{5.22}$$

Therefore, the evaluatation of $\mathbf{f}(x)$ at (5.20) is equivalent to such an evaluation for the polynomial

$$\mathbf{f}_1(x) + \mathbf{f}_2(x) \tag{5.23}$$

of the degree $\frac{N}{2} - 1$ (because for the $\frac{N}{2}$-th power for the nodes in (5.20) is equal to one.

- **Evaluation of $\mathbf{f}(x)$ at (5.21).** Let us define another pair of polynomials

$$\mathbf{g}_1(x) = f(0) + wf(1)x + w^2(2)x^2 + \ldots w^{\frac{N}{2}-1}f(\frac{N}{2} - 1)x^{\frac{N}{2}-1}$$

$$\mathbf{g}_2(x) = f(\frac{N}{2}) + wf(\frac{N}{2} + 1)x + w^2 f(\frac{N}{2} + 2)x^2 + \ldots w^{\frac{N}{2}-1}f(N-1)x^{\frac{N}{2}-1}$$

Then of course

$$\mathbf{f}_1(w^{2k}) = g_1(w^{2k-1}), \qquad \mathbf{f}_2(w^{2k}) = g_2(w^{2k-1}).$$

Therefore to evaluate $\mathbf{f}(x)$ in (5.22) at (5.21) we have to evaluate

$$\mathbf{g}_1(x) - \mathbf{g}_2(x) \tag{5.24}$$

at (5.20).

- Summarizing, the evaluation of the polynomial $\mathbf{f}(x)$ at N roots of unity (5.19) is reduced to the two evaluations at $\frac{N}{2}$ roots of unity for the polynomials in (5.23) and (5.24). By lemma 2.1 we obtain the desired result.

This just a polynomial formulation for the Cooley-Tukey algorithm of Sec. 4.

6 Discrete convolution

An immediate application of the FFT algorithm is to the rapid computing of the coefficient of the product of two polynomials

$$c(x) = a(x)\cdot b(x) = (a_0 + a_1x + a_2x^2 + \ldots + a_{N-1}x^{N-1})\cdot(b_0 + b_1x + b_2x^2 + \ldots + b_{N-1}x^{N-1}) = \tag{6.25}$$
$$= a_0b_0 + (a_1b_0 + a_0b_1)x + (a_2b_0 + a_1b_1 + a_0b_2)x^2 + \ldots + a_{N-1}b_{N-1}x^{2N-2}$$

The idea is to evaluate each of the $a(x)$, $b(x)$ at $2N$ roots of unity, then to pairwise multiply the obtained valies, and finally to recover the coefficients of the product via interpolation. Since we are working at roots of unity, evaluation and interpolation are just DFT and IDFT, implying

$$\begin{bmatrix} c_0 \\ \vdots \\ c_{2N-2} \end{bmatrix} = \mathcal{F}^*_{2N}\cdot((\mathcal{F}_{2N}\cdot\begin{bmatrix} a_0 \\ \vdots \\ a_{N-1} \\ 0 \\ \vdots \\ 0 \end{bmatrix})\odot(\mathcal{F}_{2N}\cdot\begin{bmatrix} a_0 \\ \vdots \\ a_{N-1} \\ 0 \\ \vdots \\ 0 \end{bmatrix})) \tag{6.26}$$

where $\odot$ denotes the componentwise product of two vectors. The complexity of the computation suggested by (6.26) is 3 FFT's of the order $2N$, which compares favorably with the complexity $O(N^2)$ operations, corresponding to the straigtforward computation of the coefficients by (6.25).

Recall the sequence $\{c_k\}_{k=0}^{2N-1}$ is called the *discrete convolution* of $\{a_k\}_{k=0}^{N-1}$ and $\{b_k\}_{k=0}^{N-1}$, and that (6.25) admits the following equivalent formulation in terms of multiplication of triangular Toeplitz matrices.

$$\begin{bmatrix} c_0 \\ \vdots \\ c_{2N-1} \end{bmatrix} = \begin{bmatrix} a_0b_0 \\ a_1b_0 + a_0b_1 \\ \vdots \\ a_{N-1}b_{N-1} \end{bmatrix} = \begin{bmatrix} a_0 & & & & & & & \\ a_1 & a_0 & & & & & & \\ a_2 & a_1 & a_0 & & & & & \\ \vdots & a_2 & a_1 & \ddots & & & & \\ a_{N-1} & & a_2 & \ddots & \ddots & & & \\ 0 & a_{N-1} & & \ddots & \ddots & a_0 & & \\ \vdots & \ddots & \ddots & & \ddots & a_1 & a_0 & \\ 0 & \cdots & 0 & a_{N-1} & \cdots & a_2 & a_1 & a_0 \end{bmatrix} \cdot \begin{bmatrix} b_0 \\ \vdots \\ b_{N-1} \\ 0 \\ \vdots \\ 0 \end{bmatrix}. \tag{6.27}$$

7 Cyclic convolution

Although polynomial argument suggest a simple explanation for the possibility co compute the product of two polynomials via 3 FFT's, a close examination of (6.27) does not reveal why this matrix-vector product can be so efficiently evaluated. In fact, to give an appropriate matrix argument we have to bring the *cyclic convolution*,

$$\{c_k\}_{k=0}^{N-1} = \{a_k\}_{k=0}^{N-1} * \{b_k\}_{k=0}^{N-1} = \{\sum_{k=0}^{N-1} a_k \cdot b_{N+s-k}\}_{s=0}^{N-1} \tag{7.28}$$

into our discussion. The difference between the usual convolution and the cyclic convolution is in that the matrix interpretation for the latter is given in terms of multiplication of a circulant matrix by a vector, namely

$$\begin{bmatrix} c_0 \\ \vdots \\ c_{N-1} \end{bmatrix} = \begin{bmatrix} a_0b_0 + a_1b_{N-1} + \ldots + a_{N-2}b_2 + a_{N-1}b_1 \\ a_0b_1 + a_1b_0 + \ldots + a_{N-2}b_3 + a_{N-1}b_2 \\ \vdots \\ a_0b_{N-2} + a_1b_{N-3} + \ldots + a_{N-2}b_0 + a_{N-1}b_{N-1} \\ a_0b_{N-1} + a_1b_{N-2} + \ldots + a_{N-2}b_1 + a_{N-1}b_0 \end{bmatrix} =$$

$$= \begin{bmatrix} a_0 & a_{N-1} & a_{N-2} & \cdots & a_2 & a_1 \\ a_1 & a_0 & a_{N-1} & \cdots & a_3 & a_2 \\ a_2 & a_1 & a_0 & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & a_{N-1} & a_{N-2} \\ a_{N-2} & & & \ddots & a_0 & a_{N-1} \\ a_{N-1} & a_{N-2} & \cdots & a_2 & a_1 & a_0 \end{bmatrix} \cdot \begin{bmatrix} b_0 \\ \vdots \\ b_{N-1} \end{bmatrix} \tag{7.29}$$

A circulant matrix in (7.29) can be seen as a sum

$$circ(a_0, \ldots, a_{N-1}) = \sum_{k=0}^{N-1} a_k Z_1^k, \tag{7.30}$$

where

$$Z_1 = \begin{bmatrix} 0 & 0 & \cdots & 0 & 1 \\ 1 & \ddots & & \vdots & 0 \\ 0 & \ddots & 0 & \vdots & \vdots \\ \vdots & & 1 & 0 & 0 \\ 0 & \cdots & 0 & 1 & 0 \end{bmatrix}. \tag{7.31}$$

We shall show that the circulant matrix argument naturally leads to the use of DFT, however we first present an an auxiliary material on connection between polynomials and their so-called *companion* matrices.

8 Polynomials, Vandermonde and companion matrices

Let us associate with a polynomial

$$a(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_{N-1} x^{N-1}$$

its *companion* matrix,

$$C_a = \begin{bmatrix} 0 & 0 & \cdots & 0 & -a_0/a_{N-1} \\ 1 & \ddots & & \vdots & -a_1/a_{N-1} \\ 0 & \ddots & 0 & \vdots & \vdots \\ \vdots & & 1 & 0 & -a_{N-3}/a_{N-1} \\ 0 & \cdots & 0 & 1 & -a_{N-2}/a_{N-1} \end{bmatrix}$$

The nice fact about C_a is that it is diagonalized by the Vandermonde matrix whose N nodes are N roots of the polynomial $a(x)$ (for simplicity we assume them to be N different numbers). This fact can be checked by straightforward matrix multiplication

$$\text{diag } (x_1, \ldots, x_N) \cdot V(x_1, \ldots, x_n) = V(x_1, \ldots, x_N) \cdot C_a$$

Example. Consider $a(x) = x^N - 1$. Then clearly

$$C_a = Z_1 = \begin{bmatrix} 0 & 0 & \cdots & 0 & 1 \\ 1 & \ddots & & \vdots & 0 \\ 0 & \ddots & 0 & \vdots & \vdots \\ \vdots & & 1 & 0 & 0 \\ 0 & \cdots & 0 & 1 & 0 \end{bmatrix}$$

and since in this case $\mathcal{F}_N = V(x_1, \ldots, x_N)$, we have

$$\mathcal{F}_N Z_1 \mathcal{F}_N^* = \text{ diag } (1, w, w^2, \ldots, w^{N-1}) \tag{8.32}$$

This simple identity yields the following remarkable fact.

Proposition 8.1 *An arbitrary circulant matrix, circ* $(a_0, \ldots, a_{N-1})$ *is digonalized by the DFT matrix :*

$$diag\ (\lambda_1, \ldots, \lambda_N) = \mathcal{F}_N\ circ\ (a_0, \ldots, a_{N-1})\mathcal{F}_N^*$$

Moreover,

$$\begin{bmatrix} \lambda_1 \\ \vdots \\ \lambda_N \end{bmatrix} = \mathcal{F}_N \begin{bmatrix} a_0 \\ \vdots \\ a_{N-1} \end{bmatrix}$$

Proof. It is crucial to observe that the diagonal entries of the diagonal matrix on the right-hand side of (8.32) satisfy

$$\begin{bmatrix} 1 \\ w \\ w^2 \\ \vdots \\ w^{N-1} \end{bmatrix} = \mathcal{F} \cdot \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

i.e., one computes the DFT for the first column in (8.32). Therefore we have

$$\mathcal{F}_N Z_1^k \mathcal{F}_N^* = \text{ diag } (1, (w^k), (w^k)^2, \ldots, (w^k)^{N-1}).$$

Combining such identities for $k = 0, 1, \ldots, N-1$, we obtain

$$\mathcal{F}_N \text{ circ } (a_0, \ldots, a_{N-1})\mathcal{F}_N = \text{ diag } (a(1), a(w), a(w^2), \ldots, a(w^k)).$$

which is nothing else as evaluation of the polynomial $a(x) = a_0 + a_1 x + \ldots + a_{N-1}x^{N-1}$ at roots of unity, i.e., DFT. The proposition is now proved.

9 Cyclic convolution again

The results of the previous section shed light on how to rapidly multiply a circulant matrix by a vector. Indeed,

$$\text{circ } (a_0, \ldots, a_{N-1}) \cdot \begin{bmatrix} b_0 \\ \vdots \\ b_{N-1} \end{bmatrix} = \mathcal{F}_N^* \cdot ((\mathcal{F}_N \begin{bmatrix} a_0 \\ \vdots \\ a_{N-1} \end{bmatrix}) \odot (\mathcal{F}_N \begin{bmatrix} b_0 \\ \vdots \\ b_{N-1} \end{bmatrix}))$$

where again $\odot$ denotes a componentwise vector product.

Therefore the cyclic convolution can be computed in just 3 FFT's of the size N, and the explanation is in the fact that circulant matrices are all diagonalized by the DFT matrix.

But what about the usual convolution (recall that we did not find an immediate matrix argument for the possibilty to rapidly compute (6.27)). Now with the new argumants this is easy to do : a triangular $N \times N$ Toeplitz matrix can be embedded into a $2N \times 2N$ circulant matrix (just padd the first column with zeros). Then (6.27) can be replaced by a more clear formula.

$$\begin{bmatrix} c_0 \\ \vdots \\ c_{2N-1} \end{bmatrix} = \begin{bmatrix} a_0 b_0 \\ a_1 b_0 + a_0 b_1 \\ \vdots \\ a_{N-1} b_{N-1} \end{bmatrix} = \begin{bmatrix} a_0 & 0 & \cdots & 0 & a_{N-1} & \cdots & a_2 & a_1 \\ a_1 & a_0 & & & 0 & \ddots & & a_2 \\ a_2 & a_1 & a_0 & & & \ddots & \ddots & \vdots \\ \vdots & a_2 & a_1 & \ddots & & & \ddots & a_{N-1} \\ a_{N-1} & & a_2 & \ddots & \ddots & & & 0 \\ 0 & a_{N-1} & & \ddots & \ddots & a_0 & & \vdots \\ \vdots & \ddots & \ddots & & \ddots & a_1 & a_0 & 0 \\ 0 & \cdots & 0 & a_{N-1} & \cdots & a_2 & a_1 & a_0 \end{bmatrix} \cdot \begin{bmatrix} b_0 \\ \vdots \\ b_{N-1} \\ 0 \\ \vdots \\ 0 \end{bmatrix}.$$

One sees that the explanation is the same : the usual convolution of the size N can be interpreted as a cyclic convolution of the size $2N$.