Lecture Notes to Accompany

Scientific Computing

An Introductory Survey

Second Edition

by Michael T. Heath

Chapter 12

Fast Fourier Transform

Trigonometric Interpolation

In modeling periodic or cyclic phenomena, sines and cosines are more appropriate functions than polynomials or piecewise polynomials

Representation as linear combination of sines and cosines decomposes continuous function or discrete data into components of various frequencies

Representation in *frequency space* enables more efficient manipulations than in original time or space domain

Complex Exponential Notation

We will use complex exponential notation:

$$e^{i\theta} = \cos\theta + i\sin\theta,$$

where $i = \sqrt{-1}$ (Euler's identity)

Since

$$e^{-i\theta} = \cos(-\theta) + i\sin(-\theta) = \cos\theta - i\sin\theta,$$

$$\cos(2\pi kt) = \frac{e^{2\pi ikt} + e^{-2\pi ikt}}{2}$$

and

$$\sin(2\pi kt) = i\,\frac{e^{-2\pi ikt} - e^{2\pi ikt}}{2}$$

So pure cosine or sine wave of frequency k is equivalent to sum or difference of complex exponentials of half amplitude and frequencies k and $-k$

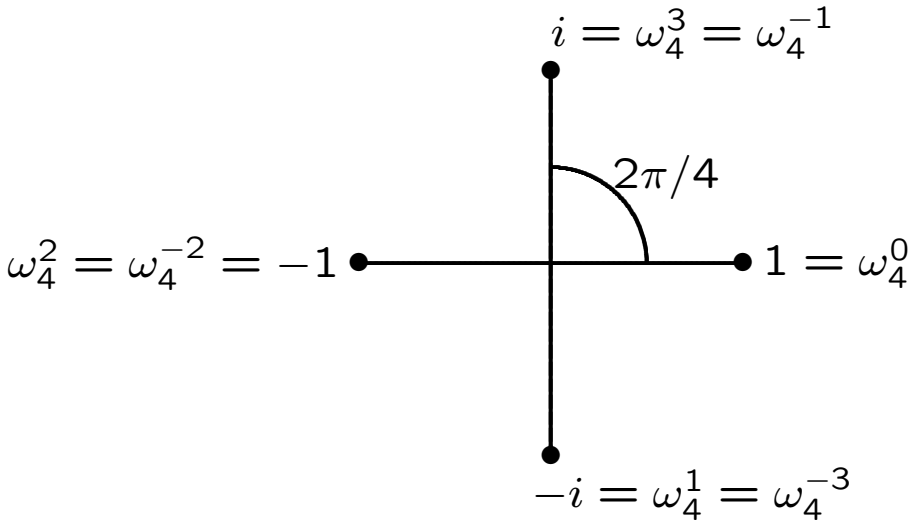
Roots of Unity

For given integer n, we use notation

$$\omega_n = \cos(2\pi/n) - i\sin(2\pi/n) = e^{-2\pi i/n}$$

for primitive nth root of unity

nth roots of unity, sometimes called *twiddle factors* in this context, are then given by ω_n^k or by ω_n^{-k}, $k = 0, \ldots, n-1$



Discrete Fourier Transform

Given sequence $\boldsymbol{x} = [x_0, \ldots, x_{n-1}]^T$, its *discrete Fourier transform*, or *DFT*, is sequence $\boldsymbol{y} = [y_0, \ldots, y_{n-1}]^T$ given by

$$y_m = \sum_{k=0}^{n-1} x_k \, \omega_n^{mk}, \quad m = 0, 1, \ldots, n-1,$$

or, written more compactly,

$$\boldsymbol{y} = \boldsymbol{F}_n \, \boldsymbol{x},$$

with entries of Fourier matrix $\boldsymbol{F}_n$ given by

$$\{\boldsymbol{F}_n\}_{mk} = \omega_n^{mk}$$

For example,

$$\boldsymbol{F}_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & \omega^1 & \omega^2 & \omega^3 \\ 1 & \omega^2 & \omega^4 & \omega^6 \\ 1 & \omega^3 & \omega^6 & \omega^9 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -i & -1 & i \\ 1 & -1 & 1 & -1 \\ 1 & i & -1 & -i \end{bmatrix}$$

Inverse DFT

Note that

$$\frac{1}{n}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & \omega^{-1} & \omega^{-2} & \omega^{-3} \\ 1 & \omega^{-2} & \omega^{-4} & \omega^{-6} \\ 1 & \omega^{-3} & \omega^{-6} & \omega^{-9} \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & \omega^{1} & \omega^{2} & \omega^{3} \\ 1 & \omega^{2} & \omega^{4} & \omega^{6} \\ 1 & \omega^{3} & \omega^{6} & \omega^{9} \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

In general, $\boldsymbol{F}_n^{-1} = (1/n)\boldsymbol{F}_n^H$

Inverse DFT is therefore given by

$$x_k = \frac{1}{n}\sum_{m=0}^{n-1} y_m\, \omega_n^{-mk}, \quad k = 0, 1, \ldots, n-1$$

DFT gives trigonometric interpolant using only matrix-vector multiplication, which costs only $\mathcal{O}(n^2)$

DFT, continued

DFT of sequence, even purely real sequence, is in general complex

Components of DFT $\boldsymbol{y}$ of real sequence $\boldsymbol{x}$ of length n are *conjugate symmetric*: y_k and y_{n-k} are complex conjugates for $k = 1, \ldots, (n/2) - 1$

Two components of special interest are:

- y_0, whose value is sum of components of $\boldsymbol{x}$, is sometimes called DC component, corresponding to zero frequency (i.e., constant function)
- $y_{n/2}$, corresponding to *Nyquist frequency*, which is highest frequency representable at given sampling rate

Components of $\boldsymbol{y}$ beyond Nyquist frequency correspond to frequencies that are negatives of those below Nyquist frequency

Example: DFT

For randomly chosen sequence $\boldsymbol{x}$,

$$\boldsymbol{F}_8\,\boldsymbol{x} = \boldsymbol{F}_8 \begin{bmatrix} 4 \\ 0 \\ 3 \\ 6 \\ 2 \\ 9 \\ 6 \\ 5 \end{bmatrix} = \begin{bmatrix} 35 \\ -5.07 + 8.66i \\ -3 + 2i \\ 9.07 + 2.66i \\ -5 \\ 9.07 - 2.66i \\ -3 - 2i \\ -5.07 - 8.66i \end{bmatrix} = \boldsymbol{y}$$

Transformed sequence is complex, but y_0 and y_4 are real, while y_5, y_6, and y_7 are complex conjugates of y_3, y_2, and y_1, respectively

There appears to be no discernible pattern to frequencies present, and y_0 is indeed equal to sum of elements of $\boldsymbol{x}$

Example: DFT

For cyclic sequence $\boldsymbol{x}$,

$$\boldsymbol{F}_8\,\boldsymbol{x} = \boldsymbol{F}_8 \begin{bmatrix} 1 \\ -1 \\ 1 \\ -1 \\ 1 \\ -1 \\ 1 \\ -1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 8 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \boldsymbol{y}$$

Sequence has highest possible rate of oscillation (between 1 and -1) for this sampling rate

In transformed sequence, only nonzero component is at Nyquist frequency (in this case y_4)

Computing DFT

By taking advantage of symmetries and redundancies in definition of DFT, shortcut algorithm can be developed for evaluating DFT very efficiently

For illustration, consider case $n = 4$

From definition of DFT,

$$y_m = \sum_{k=0}^{3} x_k \, \omega_n^{mk}, \quad m = 0, \ldots, 3$$

Writing out four equations in full,

$$\begin{aligned} y_0 &= x_0\omega_n^0 + x_1\omega_n^0 + x_2\omega_n^0 + x_3\omega_n^0 \\ y_1 &= x_0\omega_n^0 + x_1\omega_n^1 + x_2\omega_n^2 + x_3\omega_n^3 \\ y_2 &= x_0\omega_n^0 + x_1\omega_n^2 + x_2\omega_n^4 + x_3\omega_n^6 \\ y_3 &= x_0\omega_n^0 + x_1\omega_n^3 + x_2\omega_n^6 + x_3\omega_n^9 \end{aligned}$$

Computing DFT, continued

Noting that

$$\omega_n^0 = \omega_n^4 = 1, \quad \omega_n^2 = \omega_n^6 = -1, \quad \omega_n^9 = \omega_n^1$$

and regrouping, we obtain four equations

$$\begin{aligned} y_0 &= (x_0 + \omega_n^0 x_2) + \omega_n^0 (x_1 + \omega_n^0 x_3) \\ y_1 &= (x_0 - \omega_n^0 x_2) + \omega_n^1 (x_1 - \omega_n^0 x_3) \\ y_2 &= (x_0 + \omega_n^0 x_2) + \omega_n^2 (x_1 + \omega_n^0 x_3) \\ y_3 &= (x_0 - \omega_n^0 x_2) + \omega_n^3 (x_1 - \omega_n^0 x_3) \end{aligned}$$

DFT can now be computed with only 8 additions/subtractions and 6 multiplications, instead of expected $(4-1)*4 = 12$ additions and $4^2 = 16$ multiplications

Actually, even fewer multiplications are required for this small case, since $\omega_n^0 = 1$, but we have tried to illustrate how algorithm works in general

Computing DFT, continued

Main point is that computing DFT of original 4-point sequence has been reduced to computing DFT of its two 2-point even and odd subsequences

This property holds in general: DFT of n-point sequence can be computed by breaking it into two DFTs of half length, provided n is even

General pattern becomes clearer when viewed in terms of first few Fourier matrices

$$\boldsymbol{F}_1 = 1, \qquad \boldsymbol{F}_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix},$$

$$\boldsymbol{F}_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -i & -1 & i \\ 1 & -1 & 1 & -1 \\ 1 & i & -1 & -i \end{bmatrix}$$

Computing DFT, continued

Let $\boldsymbol{P}_4$ be permutation matrix

$$\boldsymbol{P}_4 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

and $\boldsymbol{D}_2$ diagonal matrix

$$\boldsymbol{D}_2 = \mathrm{diag}(1, \omega_4) = \begin{bmatrix} 1 & 0 \\ 0 & -i \end{bmatrix}$$

Then

$$\boldsymbol{F}_4\boldsymbol{P}_4 = \left[\begin{array}{cc|cc} 1 & 1 & 1 & 1 \\ 1 & -1 & -i & i \\ \hline 1 & 1 & -1 & -1 \\ 1 & -1 & i & -i \end{array}\right] = \begin{bmatrix} \boldsymbol{F}_2 & \boldsymbol{D}_2\boldsymbol{F}_2 \\ \boldsymbol{F}_2 & -\boldsymbol{D}_2\boldsymbol{F}_2 \end{bmatrix},$$

Thus, $\boldsymbol{F}_4$ can be rearranged so that each block is diagonally scaled version of $\boldsymbol{F}_2$

Computing DFT, continued

Such hierarchical splitting can be carried out at each level, provided number of points is even

In general, $\boldsymbol{P}_n$ is permutation that groups even-numbered columns of $\boldsymbol{F}_n$ before odd-numbered columns, and

$$\boldsymbol{D}_{n/2} = \text{diag}\left(1, \omega_n, \ldots, \omega_n^{(n/2)-1}\right)$$

To apply $\boldsymbol{F}_n$ to sequence of length n, we need merely apply $\boldsymbol{F}_{n/2}$ to its even and odd subsequences and scale results, where necessary, by $\pm\boldsymbol{D}_{n/2}$

Resulting recursive divide-and-conquer algorithm for computing DFT is called *fast Fourier transform*, or *FFT*

FFT Algorithm

```
procedure fft(x, y, n, ω)
    if n = 1 then
        y[0] = x[0]
    else
        for k = 0 to (n/2) − 1
            p[k] = x[2k]
            s[k] = x[2k + 1]
        end
        fft(p, q, n/2, ω^2)
        fft(s, t, n/2, ω^2)
        for k = 0 to n − 1
            y[k] = q[k mod (n/2)] +
                   ω^k t[k mod (n/2)]
        end
    end
```

FFT Algorithm, continued

There are $\log_2 n$ levels of recursion, each of which involves $\mathcal{O}(n)$ arithmetic operations, so total cost is $\mathcal{O}(n \log_2 n)$

For clarity, separate arrays were used for subsequences, but in fact transform can be computed in place using no additional storage

Input sequence is assumed complex; if input sequence is real, then additional symmetries in DFT can be exploited to reduce storage and operation count by half

Output sequence is not produced in natural order, but either input or output sequence can be rearranged at cost of $\mathcal{O}(n \log_2 n)$ (analogous to sorting)

FFT Algorithm, continued

FFT algorithm can be formulated using iteration rather than recursion, which is often desirable for greater efficiency or when using programming language that does not support recursion

Despite its name, fast Fourier transform is an algorithm, not a transform

It is a particular way of computing DFT of sequence in efficient manner

Complexity of FFT

DFT is defined in terms of matrix-vector product, whose straightforward evaluation would appear to require $\mathcal{O}(n^2)$ arithmetic operations

Use of FFT algorithm reduces work to only $\mathcal{O}(n \log_2 n)$, which makes enormous practical difference in time required to transform large sequences

n	$n \log_2 n$	n^2
64	384	4096
128	896	16384
256	2048	65536
512	4608	262144
1024	10240	1048576

Inverse Transform

Due to similar form of DFT and its inverse, FFT algorithm can also be used to compute inverse DFT efficiently

Ability to transform back and forth quickly between time and frequency domains makes it practical to perform any computations or analysis that may be required in whichever domain is more convenient and efficient

Limitations of FFT

FFT algorithm is not always applicable or maximally efficient

Input sequence assumed to be:

- Equally spaced
- Periodic
- Power of two in length

First two of these follow from definition of DFT, while third is required for maximal efficiency of FFT algorithm

Care must be taken in applying FFT algorithm to produce most meaningful results as efficiently as possible

For example, transforming sequence that is not really periodic or padding sequence to make its length power of two may introduce spurious noise and complicate interpretation of results

Mixed-Radix FFT

It is possible to define "mixed-radix" FFT algorithm that does not require number of points n to be power of two

More general algorithm is still based on divide-and-conquer; sequence is not necessarily split exactly in half at each level, but by smallest prime factor of remaining sequence length

Efficiency depends on whether n is product of small primes (ideally power of two)

If not, then much of computational advantage of FFT may be lost

For example, if n itself is prime, then sequence cannot be split at all, and "fast" algorithm becomes standard $\mathcal{O}(n^2)$ matrix-vector multiplication

Applications of DFT

DFT is often of direct interest itself and is also useful as computational tool that provides efficient means for computing other quantities

DFT is of direct interest in detecting periodicities or cycles in discrete data

DFT can be used to *remove* unwanted periodicities

For example, to remove high-frequency noise from sequence, one can compute its DFT, set high-frequency components of transformed sequence to zero, then compute inverse DFT of modified sequence to get back into original domain

Applications of DFT, continued

As another example, weather data often contain two distinct cycles, diurnal and annual, and one might want to remove one to study the other in isolation

Economic data are also often "seasonally adjusted," removing unwanted periodicities to reveal "secular" trends

Because of such uses, DFT is of vital importance in many aspects of signal processing, such as digital filtering

Applications of DFT, continued

Some computations are simpler or more efficient in frequency domain than in time domain

Examples include discrete convolution of two sequences $\boldsymbol{u}$ and $\boldsymbol{v}$ of length n,

$$\{\boldsymbol{u} \star \boldsymbol{v}\}_m = \sum_{k=0}^{n-1} v_k u_{m-k}, \quad m = 0, 1, \ldots, n-1,$$

and related quantities such as cross correlation of two sequences or autocorrelation of a sequence with itself

Equivalent operation in frequency domain is simply pointwise multiplication (in some cases with complex conjugation)

Applications of DFT, continued

If DFT and its inverse can be computed efficiently, then it may be advantageous to transform to frequency domain to compute such convolutions, then transform back to time domain

For example, to compute convolution or correlation of two sequences, it is often advantageous to use FFT algorithm to compute DFT of sequences, take pointwise product in frequency domain, then inverse DFT back to time domain, again via FFT algorithm

FFT algorithm also forms basis for exceptionally efficient methods for solving certain periodic boundary value problems, such as Poisson's equation on regular domain with periodic boundary conditions

Fast Polynomial Multiplication

FFT algorithm also provides fast method for some computations that might not seem related to it

For example, complexity of straightforward multiplication of two polynomials is proportional to product of their degrees

However, polynomial of degree $n-1$ is uniquely determined by its values at n distinct points

Thus, product polynomial can be determined by interpolation from pointwise product of factor polynomials at n points

Both polynomial evaluation and interpolation using n points would normally require $\mathcal{O}(n^2)$ operations, but by choosing points to be nth roots of unity, FFT algorithm can be used to reduce complexity to $\mathcal{O}(n \log_2 n)$

Wavelets

Sine and cosine functions used in Fourier analysis are very smooth (infinitely differentiable), and very broad (nonzero almost everywhere on real line)

They are not very effective for representing functions that change abruptly or have highly localized support

Gibbs phenomenon in Fourier representation of square wave ("ringing" at corners) is one manifestation of this

In response to this shortcoming, there has been intense interest in recent years in new type of basis functions called *wavelets*

Wavelets, continued

Wavelet basis is generated from single function $\phi(x)$, called *mother wavelet* or *scaling function*, by dilation and translation, $\phi((x-b)/a)$, where a and b are real numbers with $a \neq 0$

There are many choices for mother wavelet

Main issue is tradeoff between smoothness and compactness

A member of one of most commonly used families of wavelets, due to Daubechies, is shown below

Wavelets, continued

Typical choices for dilation and translation parameters are $a = 2^{-j}$ and $b = k2^j$, where j and k are integers, so that $\phi_{jk}(x) = \phi(2^j x - k)$

If mother wavelet $\phi(x)$ has sufficiently localized support, then

$$\int \phi_{jk}\phi_{mn} = 0$$

whenever indices do not both match, so doubly-indexed basis functions $\phi_{jk}(x)$ are orthogonal

By replicating mother wavelet at many different scales, it is possible to mimic behavior of any function at many different scales; this property of wavelets is called *multiresolution*

Wavelets, continued

Fourier basis functions are localized in frequency but not in time: small changes in frequency produce changes everywhere in time domain

Wavelets are localized in both frequency (by dilation) and time (by translation)

This localization tends to make wavelet representation of function very sparse

Discrete Wavelet Transform

As with Fourier transform, there is analogous discrete wavelet transform, or DWT

DWT and its inverse can be computed very efficiently by pyramidal, hierarchical algorithm

Sparsity of wavelet basis makes computation of DWT even faster than FFT

DWT requires only $\mathcal{O}(n)$ work for sequence of length n, instead of $\mathcal{O}(n \log n)$

Because of their efficiency, both in computation and in compactness of representation, wavelets are playing an increasingly important role in many areas of signal and image processing, such as data compression, noise removal, and computer vision