Queens College, CUNY, Department of Computer Science
Numerical Methods
CSCI 361 / 761
Spring 2018
Instructor: Dr. Sateesh Mane

April 14, 2018

30 Lecture 30

Fourier Series

- In this lecture, we continue our study of **Fourier series.**
- We shall treat functions of one variable only.
- We shall study algorithms to perform computations for Fourier series.
- We shall study the **Discrete Fourier Transform (DFT).**
- A particularly important concept is the **Fast Fourier Transform (FFT).**
- This lecture requires knowledge of **complex numbers.**

30.1 Fourier series using sines and cosines

- This is a review from a previous lecture, reproduced here for ease of reference.
- Let $f(\theta)$ be a periodic function of and angle θ with period 2π.
- We assume f is sufficiently well behaved to justify the relevant calculations or algorithms.
  1. For example, the function is absolutely integrable around a circle:
  $$\int_0^{2\pi} |f(\theta)| \, d\theta < \infty \,. \tag{30.1.1}$$
  2. The function also has at most a finite number of discontinuities in the interval $0 \le \theta < 2\pi$.
- Then f can be expressed (or expanded) in a **Fourier series** as follows:
$$\begin{aligned} f(\theta) &= \tfrac{1}{2}a_0 + a_1\cos(\theta) + a_2\cos(2\theta) + a_3\cos(3\theta) + \cdots \\ &\qquad + b_1\sin(\theta) + b_2\sin(2\theta) + b_3\sin(3\theta) + \cdots \\ &= \tfrac{1}{2}a_0 + \sum_{j=1}^{\infty} \big[\, a_j\cos(j\theta) + b_j\sin(j\theta) \,\big] \,. \end{aligned} \tag{30.1.2}$$
- The coefficients a_j and b_j are obtained from the function $f(\theta)$ via
$$\begin{aligned} a_j &= \frac{1}{\pi}\int_0^{2\pi} f(\theta)\cos(j\theta)\,d\theta \qquad (j \ge 0)\,, \\ b_j &= \frac{1}{\pi}\int_0^{2\pi} f(\theta)\sin(j\theta)\,d\theta \qquad (j > 0)\,. \end{aligned} \tag{30.1.3}$$

30.2 Representation using complex exponentials

- In all the examples we have studied, the function $f(\theta)$ is real, hence all the Fourier coefficients a_j amd bj are real.
- However, this is not the best formulation for efficient numerical computation.
- Let us reexpress eq. (30.1.2) using complex exponentials. Note that

$$\begin{aligned} a_j\cos(j\theta) + b_j\sin(j\theta) &= a_j\,\frac{e^{ij\theta}+e^{-ij\theta}}{2} + b_j\,\frac{e^{ij\theta}-e^{-ij\theta}}{2i} \\ &= \frac{1}{2}\Bigl[\,(a_j-ib_j)\,e^{ij\theta} + (a_j+ib_j)\,e^{-ij\theta}\Bigr]\,. \end{aligned} \tag{30.2.1}$$

- Let us therefore define complex Fourier harmonics via

$$F_0 = \frac{a_0}{2}\,, \qquad F_j = \frac{a_j-ib_j}{2}\,, \qquad F_{-j} = \frac{a_j+ib_j}{2} \qquad (j>0)\,. \tag{30.2.2}$$

- **The Fourier series using complex Fourier harmonics is given by**

$$f(\theta) = \sum_{j=-\infty}^{\infty} F_j e^{ij\theta}\,. \tag{30.2.3}$$

- This is a discrete sum, not an integral, but has the appearance of an inverse Fourier transform.
- Note the following orthogonality and normalization relations:

$$\frac{1}{2\pi}\int_0^{2\pi} e^{i(m-n)\theta} = \delta_{mn} \qquad\qquad (-\infty < m,n < \infty)\,. \tag{30.2.4}$$

- Then the value of F_j is obtained from the function $f(\theta)$ via

$$F_j = \frac{1}{2\pi}\int_0^{2\pi} f(\theta)\,e^{-ij\theta}\,d\theta \qquad (-\infty < j < \infty)\,. \tag{30.2.5}$$

- The integral is over a finite domain, not the entire real line, but has the appearance of a Fourier transform.

30.3 Example using complex Fourier harmonics

- Let us study the same example treated previously for a Fourier series.
- Let us examine the results when we employ complex Fourier harmonics.
- The example function is

$$f_{\rm ex2}(\theta) = \tfrac{1}{2}A_0 + A_1\cos(\theta) + B_1\sin(\theta) + A_2\cos(2\theta) + B_2\sin(2\theta)\,. \qquad (30.3.1)$$

- We employ $n = 4$ points. Hence $\theta_k = 2k\pi/4 = k\pi/2$, for $k = 0, 1, 2, 3$.
- Hence we evaluate the integral in eq. (30.2.5) using a discrete sum with n points:

$$F_j = \frac{1}{n}\sum_{k=0}^{n-1} f(\theta_k)\, e^{-ij\theta_k} \qquad (j = 0, \dots, n-1)\,. \qquad (30.3.2)$$

- For $n = 4$ and the example function in eq. (30.3.1), this yields

$$\begin{aligned} F_j &= \frac{1}{4}\Bigl[f_{\rm ex2}(0) + f_{\rm ex2}(\tfrac{1}{2}\pi)e^{-ij\pi/2} + f_{\rm ex2}(\pi)e^{-ij\pi} + f_{\rm ex2}(\tfrac{3}{2}\pi)e^{-i2j\pi/2} \Bigr] \\ &= \frac{1}{4}\Bigl[(\tfrac{1}{2}A_0 + A_1 + A_2) + (-1)^j(\tfrac{1}{2}A_0 + B_1 - A_2) \\ &\qquad + (-1)^j(\tfrac{1}{2}A_0 - A_1 + A_2) + i^j(\tfrac{1}{2}A_0 - B_1 - A_2) \Bigr]\,. \end{aligned} \qquad (30.3.3)$$

- We obtain the following results for $F_0, \dots, F_3$:

$$\begin{aligned} F_0 &= \frac{1}{4}\Bigl[(\tfrac{1}{2}A_0{+}A_1{+}A_2){+}(\tfrac{1}{2}A_0{+}B_1{-}A_2){+}(\tfrac{1}{2}A_0{-}A_1{+}A_2){+}(\tfrac{1}{2}A_0{-}B_1{-}A_2)\Bigr] &&= \tfrac{1}{2}A_0\,, && (30.3.4a) \\ F_1 &= \frac{1}{4}\Bigl[(\tfrac{1}{2}A_0{+}A_1{+}A_2){-}i(\tfrac{1}{2}A_0{+}B_1{-}A_2){-}(\tfrac{1}{2}A_0{-}A_1{+}A_2){+}i(\tfrac{1}{2}A_0{-}B_1{-}A_2)\Bigr] &&= \frac{A_1 - iB_1}{2}\,, && (30.3.4b) \\ F_2 &= \frac{1}{4}\Bigl[(\tfrac{1}{2}A_0{+}A_1{+}A_2){-}(\tfrac{1}{2}A_0{+}B_1{-}A_2){+}(\tfrac{1}{2}A_0{-}A_1{+}A_2){-}(\tfrac{1}{2}A_0{-}B_1{-}A_2)\Bigr] &&= A_2\,, && (30.3.4c) \\ F_3 &= \frac{1}{4}\Bigl[(\tfrac{1}{2}A_0{+}A_1{+}A_2){+}i(\tfrac{1}{2}A_0{+}B_1{-}A_2){-}(\tfrac{1}{2}A_0{-}A_1{+}A_2){-}i(\tfrac{1}{2}A_0{-}B_1{-}A_2)\Bigr] &&= \frac{A_1 + iB_1}{2}\,. && (30.3.4d) \end{aligned}$$

- The values of F_0 and F_1 are as expected. We calculated the Fourier coefficient F_3 as opposed to F_{-1}, but we obtained the value of F_{-1} anyway:

$$F_{-1} = F_3 = \frac{A_1 + iB_1}{2}\,. \qquad (30.3.5)$$

- **In general, for any value of n (at least if n is even), the Fourier coefficients 'wrap around' because of the periodicity of the function:**

$$F_{-j} = F_{n-j}\,. \qquad (30.3.6)$$

- The Fourier coefficient $F_{n/2}$, i.e. F_2 in our example, yields $F_{n/2} = A_{n/2}$ not $\frac{1}{2}A_{n/2}$.
- *Is this a problem? Not necessarily.*

30.4 Consequences of employing complex Fourier harmonics

- Let us sum the Fourier series obtained in the example in Sec. 30.3.
- We obtain the following sum, using eq. (30.2.3):

$$\begin{aligned} f_{\rm sum}(\theta) &= F_0 + F_1 e^{i\theta} + F_{-1}e^{-i\theta} + F_2 e^{i2\theta} \\ &= \tfrac{1}{2}A_0 + \frac{A_1 - iB_1}{2}\,e^{i\theta} + \frac{A_1 + iB_1}{2}\,e^{-i\theta} + A_2\,e^{i2\theta} \\ &= \tfrac{1}{2}A_0 + A_1\cos(\theta) + B_2\sin(\theta) + A_2\big[\cos(2\theta) \boldsymbol{+ i\sin(2\theta)}\,\big]\,. \end{aligned} \tag{30.4.1}$$

- We obtain an extra term $iA_2\sin(2\theta)$ which does not exist in the original function.
- As we know, we do not obtain the term $B_2\sin(2\theta)$, which *does* exist in the original function.
- Clearly the sum of the series is inaccurate for the $j = n/2$ harmonic, i.e. 2θ in this example.
- ***However, are we making unrealistic demands?***
- We evaluated the function $f_{\rm ex2}(\theta)$ at only four points. It is unreasonable to expect that the resulting Fourier series will match the original function exactly for all values of θ.
- Let us evaluate the sum of the series **only at the input points $\theta_k = 2k\pi/n$,** *i.e. only at the input values of θ used to derive the series.*
- Then $\sin(2\theta) = 0$ at all the points $\theta_k = 2k\pi/n$, hence the sum of the series equals the original function exactly:

$$f_{\rm sum}(0) = \tfrac{1}{2}A_0 + A_1 + A_2 \qquad = f_{\rm ex2}(0)\,, \tag{30.4.2a}$$

$$f_{\rm sum}(\tfrac{1}{2}\pi) = \tfrac{1}{2}A_0 + B_1 - A_2 \qquad = f_{\rm ex2}(\tfrac{1}{2}\pi)\,, \tag{30.4.2b}$$

$$f_{\rm sum}(\pi) = \tfrac{1}{2}A_0 - A_1 + A_2 \qquad = f_{\rm ex2}(\pi)\,, \tag{30.4.2c}$$

$$f_{\rm sum}(\tfrac{3}{2}\pi) = \tfrac{1}{2}A_0 - B_1 - A_2 \qquad = f_{\rm ex2}(\tfrac{3}{2}\pi)\,. \tag{30.4.2d}$$

- **We therefore deduce the following important fact:**
In general, for any value of n, the sum of the Fourier series, using complex Fourier harmonics, **exactly equals the original function at the n input points $\theta_k = 2k\pi/n$.**
- The claim 'exactly equals the original function' must however be interpreted with care.
  1. If a continuous signal is sampled using a finite number of values, the input data will in general contain aliased information.
  2. Recall the statements in previous lectures about the Nyquist frequency and aliasing.
- The statement 'the sum of the Fourier series, ..., exactly equals the original function at the n input points $\theta_k = 2k\pi/n$' is correct, *but the original data may contain aliased information.*
- Using complex Fourier coefficients does not solve the aliasing problem.

30.5 Convolution of discrete series

- In the continuous case, the **convolution** of two functions $f(x)$ and $g(x)$ is given by

$$(f * g)(x) = \int_{-\infty}^{\infty} f(u)g(x-u)\,du\,. \tag{30.5.1}$$

- For periodic functions, the convolution is given by a discrete sum.
- Let the periodic functions be $f(\theta)$ and $g(\theta)$.
- We calculate both functions at n points $\theta_k = 2k\pi/n$, where $k = 0, \dots, n-1$.
- Denote the resulting function values by $f_k = f(\theta_k)$ and $g_k = g(\theta_k)$, respectively.
- Note also the wrap around relations $f_{-k} = f_{n-k}$ and $h_{-k} = g_{n-k}$.
- Then for $k = 0, \dots, n-1$, the convolution of two discrete series $h = f * g$ is given by

$$h_k = (f * g)_k = \sum_{\ell=0}^{n-1} f_\ell g_\kappa \qquad \text{where} \qquad \kappa = \begin{cases} k-\ell & (k-\ell \geq 0) \\ n+k-\ell & (k-\ell < 0)\,. \end{cases} \tag{30.5.2}$$

- In the same way, the convolution $H = F * G$ of two Fourier series F_j and G_j is given by

$$H_j = (F * G)_j = \sum_{\ell=0}^{n-1} F_\ell G_\kappa \qquad \text{where} \qquad \kappa = \begin{cases} j-\ell & (j-\ell \geq 0) \\ n+j-\ell & (j-\ell < 0)\,. \end{cases} \tag{30.5.3}$$

30.6 Cross-correlation & autocorrelation of discrete series

- The same ideas in Sec. 30.5 also apply to cross–correlation and autocorrelation.
- In the continuous case, the **cross–correlation** of two functions $f(x)$ and $g(x)$ is given by

$$(f \star g)(x) = \int_{-\infty}^{\infty} f^*(u) g(x+u)\, du\,. \tag{30.6.1}$$

- Employing the same definitions as in Sec. 30.5, the cross–correlation of two discrete series $h = f \star g$ is given by

$$h_k = (f \star g)_k = \sum_{\ell=0}^{n-1} f^*_\ell g_\kappa \qquad \text{where} \qquad \kappa = \begin{cases} k+\ell & (k+\ell < n) \\ k+\ell-n & (k+\ell \geq n)\,. \end{cases} \tag{30.6.2}$$

- The cross–correlation $H = F \star G$ of two Fourier series F_j and G_j is given by

$$H_j = (F \star G)_j = \sum_{\ell=0}^{n-1} F^*_\ell G_\kappa \qquad \text{where} \qquad \kappa = \begin{cases} j+\ell & (j+\ell < n) \\ j+\ell-n & (j+\ell \geq n)\,. \end{cases} \tag{30.6.3}$$

- For autocorrelation we simply set $f = g$ or $F = G$.
- **Parseval's theorem** for discrete series states

$$\frac{1}{n} \sum_{k=0}^{n-1} |f_k|^2 \quad = \quad \sum_{j=0}^{n-1} |F_j|^2\,. \tag{30.6.4}$$

- Let us verify eq. (30.6.4) for the example function $f_{\rm ex2}$.

1. The sum of the absolute squares of the Fourier coefficients is

$$\sum_{j=0}^{3} |F_j|^2 = \frac{|A_0|^2}{4} + \frac{|A_1|^2 + |B_1|^2}{2} + |A_2|^2\,. \tag{30.6.5}$$

2. The sum of the absolute squares of the function values is

$$\begin{aligned} \sum_{k=0}^{3} |f_k|^2 = \quad & \left|\,\tfrac12 A_0 + A_1 + A_2\,\right|^2 + \left|\,\tfrac12 A_0 + B_1 - A_2\,\right|^2 \\ & + \left|\,\tfrac12 A_0 - A_1 + A_2\,\right|^2 + \left|\,\tfrac12 A_0 - B_1 - A_2\,\right|^2 \\ = & \; |A_0|^2 + 2|A_1|^2 + 2|B_1|^2 + 4|A_2|^2\,. \end{aligned} \tag{30.6.6}$$

3. Hence Parseval's theorem eq. (30.6.4) is satisfied for the example function $f_{\rm ex2}$:

$$\frac14 \sum_{k=0}^{3} |f_k|^2 = \sum_{j=0}^{3} |F_j|^2 = \frac{|A_0|^2}{4} + \frac{|A_1|^2 + |B_1|^2}{2} + |A_2|^2\,. \tag{30.6.7}$$

30.7 Change of notation

- Since we shall eventually want to write programming code, it is confusing to use the notation '`f[k]`' for the function array and '`F[j]`' for the Fourier coefficients array.
- **We change notation to $X(\theta)$ for the function.**
- Hence we write $X_k = X(\theta_k)$. Recall $\theta_k = 2k\pi/n$, where $k = 0, \ldots, n-1$.
- We continue to denote the Fourier coefficients by F_j, where $j = 0, \ldots, n-1$.

30.8 Discrete Fourier Transform (DFT)

- We now define the **Discrete Fourier Transform (DFT).**
- We employ n equally spaced data points around the circle $\theta_k = 2k\pi/n$, where $k = 0, \dots, n-1$.
- The function values are $X_k = X(\theta_k)$.
- Let ω be a primitive n^{th} root of unity, i.e. $\omega^n = 1$ and choose $\omega = e^{-i2\pi/n}$.
- The complex Fourier coefficients are denoted by $(F_0, \dots, F_{n-1})$.
- Both $(X_0, \dots, X_{n-1})$ and $(F_0, \dots, F_{n-1})$ are complex, in general.
- The **Discrete Fourier Transform (DFT)** is given by the following sum:

$$F_j = \sum_{k=0}^{n-1} X(\theta_k)\, e^{-ij\theta_k} \;=\; \sum_{k=0}^{n-1} \omega^{jk} X_k \qquad (j, k = 0, \dots, n-1)\,. \tag{30.8.1}$$

- The inverse Discrete Fourier Transform computes $(f_0, \dots, f_{n-1})$ from $(F_0, \dots, F_{n-1})$:

$$X_k = \frac{1}{n} \sum_{j=0}^{n-1} F_j\, e^{ik\theta_j} \;=\; \frac{1}{n} \sum_{j=0}^{n-1} (\omega^*)^{jk} F_j \qquad (j, k = 0, \dots, n-1)\,. \tag{30.8.2}$$

- **The inverse transform sums the series only at the input points $\theta = 2k\pi/n$, hence it returns the original data values X_k exactly.**
- The inverse transform is obtained by the same procedure as the discrete Fourier transform, replacing ω by ω^* and dividing the sum by n.
- To avoid the asymmetry of $1/n$ between eqs. (30.8.1) and (30.8.2), many authors prefer to divide by $1/\sqrt{n}$ in both formulas and write

$$F_j = \frac{1}{\sqrt{n}} \sum_{k=0}^{n-1} \omega^{jk} X_k\,, \qquad X_k = \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} (\omega^*)^{jk} F_j \qquad (j, k = 0, \dots, n-1)\,. \tag{30.8.3}$$

- The inverse transform in eq. (30.8.3) is given by simply replacing ω by ω^*.
- The standard convention in physics is to employ eqs. (30.8.1) and (30.8.2).
- We shall employ eqs. (30.8.1) and (30.8.2) in this course.
- Naïvely, the calculations in eqs. (30.8.1) and (30.8.2) require $O(n^2)$ computations.
- The **Fast Fourier Transform (FFT)** can calculate the sums in eqs. (30.8.1) and (30.8.2) using **$O(n \log n)$ computations.**
- The breakthrough was published by Cooley and Tukey in 1965. With hindsight, it is now known that others had published $O(n \log n)$ algorithms for discrete Fourier transforms before, going back to Gauss in 1805, but the significance of those earlier algorithms was not recognized until after Cooley and Tukey's publication.
- We shall study the FFT algorithm later in this lecture.

30.9 FFT: warmup exercise $n = 4$

- Let us consider a Discrete Fourier Transform (DFT) with $n = 4$ points.
- Set $\omega = e^{-i2\pi/n} = e^{-i2\pi/4}$.
- The initial data values are $(X_0, \dots, X_{n-1}) = (X_0, X_1, X_2, X_3)$.
- The transformed data values are $(F_0, \dots, F_{n-1}) = (F_0, F_1, F_2, F_3)$.
- We write out the equations in eq. (30.8.1) explicitly to obtain

$$
\begin{aligned}
F_0 &= X_0 + X_1 + X_2 + X_3 \,, \\
F_1 &= X_0 + \omega X_1 + \omega^2 X_2 + \omega^3 X_3 \,, \\
F_2 &= X_0 + \omega^2 X_1 + \omega^4 X_2 + \omega^6 X_3 \,, \\
F_3 &= X_0 + \omega^3 X_1 + \omega^6 X_2 + \omega^9 X_3 \,.
\end{aligned}
\tag{30.9.1}
$$

- Collect terms on the right hand side in odd/even pairs. Also use $\omega^n = \omega^4 = 1$.

$$
\begin{aligned}
F_0 &= X_0 + X_2 + X_1 + X_3 \,, \\
F_1 &= X_0 + \omega^2 X_2 + \omega(X_1 + \omega^2 X_3) \,, \\
F_2 &= X_0 + X_2 + \omega^2(X_1 + X_3) \,, \\
F_3 &= X_0 + \omega^2 X_2 + \omega^3(X_1 + \omega^2 X_3) \,.
\end{aligned}
\tag{30.9.2}
$$

- Notice that the X_k come in only four combinations. Use 'E' and 'O' for even and odd:

$$
\begin{aligned}
E_0 &= X_0 + X_2 \ , & O_0 &= X_1 + X_3 \ , \\
E_1 &= X_0 + \omega^2 X_2 = X_0 - X_2 \ , & O_1 &= X_1 + \omega^2 X_3 = X_1 - X_3 \ .
\end{aligned}
\tag{30.9.3}
$$

- This can clearly be formalized, so as to generalize to larger values of n. Define $\boldsymbol{\Omega}$ via

$$
\boldsymbol{\Omega} = \begin{pmatrix} 1 & 0 \\ 0 & \omega \end{pmatrix} .
\tag{30.9.4}
$$

- Then note that, using $\omega^{n/2} = \omega^2 = -1$,

$$
\begin{aligned}
\begin{pmatrix} F_0 \\ F_1 \end{pmatrix} &= \begin{pmatrix} E_0 \\ E_1 \end{pmatrix} + \begin{pmatrix} 1 & 0 \\ 0 & \omega \end{pmatrix} \begin{pmatrix} O_0 \\ O_1 \end{pmatrix} = \begin{pmatrix} E_0 \\ E_1 \end{pmatrix} + \begin{pmatrix} O_0 \\ \omega O_1 \end{pmatrix} , \\
\begin{pmatrix} F_2 \\ F_3 \end{pmatrix} &= \begin{pmatrix} E_0 \\ E_1 \end{pmatrix} - \begin{pmatrix} 1 & 0 \\ 0 & \omega \end{pmatrix} \begin{pmatrix} O_0 \\ O_1 \end{pmatrix} = \begin{pmatrix} E_0 \\ E_1 \end{pmatrix} - \begin{pmatrix} O_0 \\ \omega O_1 \end{pmatrix} .
\end{aligned}
\tag{30.9.5}
$$

- We define $\boldsymbol{F}_+$ and $\boldsymbol{F}_-$ via

$$
\boldsymbol{F}_+ = \begin{pmatrix} F_0 \\ F_1 \end{pmatrix} , \qquad \boldsymbol{F}_- = \begin{pmatrix} F_2 \\ F_3 \end{pmatrix} .
\tag{30.9.6}
$$

- Then we see from above that

$$
\begin{aligned}
\boldsymbol{F}_+ &= \boldsymbol{E} + \boldsymbol{\Omega O} \,, \\
\boldsymbol{F}_- &= \boldsymbol{E} - \boldsymbol{\Omega O} \,.
\end{aligned}
\tag{30.9.7}
$$

30.10 FFT: second exercise $n = 8$

- Next let us see how this works for for $n = 8$ points.
- Set $\omega = e^{-i2\pi/n} = e^{-i2\pi/8}$.
- The initial data values are $(X_0, \dots, X_{n-1}) = (X_0, \dots, X_7)$.
- The transformed data values are $(F_0, \dots, F_{n-1}) = (F_0, \dots, F_7)$.
- Write this out explicitly (in even/odd blocks) to obtain

$$\begin{pmatrix} F_0 \\ F_1 \\ F_2 \\ F_3 \\ F_4 \\ F_5 \\ F_6 \\ F_7 \end{pmatrix} = \begin{pmatrix} X_0 & +X_2 & +X_4 & +X_6 & +(X_1 & +X_3 & +X_5 & +X_7) \\ X_0 & +\omega^2 X_2 & +\omega^4 X_4 & +\omega^6 X_6 & +\omega(X_1 & +\omega^2 X_3 & +\omega^4 X_5 & +\omega^6 X_7) \\ X_0 & +\omega^4 X_2 & +\omega^8 X_4 & +\omega^{12} X_6 & +\omega^2(X_1 & +\omega^4 X_3 & +\omega^8 X_5 & +\omega^{12} X_7) \\ X_0 & +\omega^6 X_2 & +\omega^{12} X_4 & +\omega^{18} X_6 & +\omega^3(X_1 & +\omega^6 X_3 & +\omega^{12} X_5 & +\omega^{18} X_7) \\ X_0 & +\omega^8 X_2 & +\omega^{16} X_4 & +\omega^{24} X_6 & +\omega^4(X_1 & +\omega^8 X_3 & +\omega^{16} X_5 & +\omega^{24} X_7) \\ X_0 & +\omega^{10} X_2 & +\omega^{20} X_4 & +\omega^{30} X_6 & +\omega^5(X_1 & +\omega^{10} X_3 & +\omega^{20} X_5 & +\omega^{30} X_7) \\ X_0 & +\omega^{12} X_2 & +\omega^{24} X_4 & +\omega^{36} X_6 & +\omega^6(X_1 & +\omega^{12} X_3 & +\omega^{24} X_5 & +\omega^{36} X_7) \\ X_0 & +\omega^{14} X_2 & +\omega^{28} X_4 & +\omega^{42} X_6 & +\omega^7(X_1 & +\omega^{14} X_3 & +\omega^{28} X_5 & +\omega^{42} X_7) \end{pmatrix}. \tag{30.10.1}$$

- Use $\omega^n = \omega^8 = 1$ and $\omega^{n/2} = \omega^4 = -1$ to simplify some exponents *(but maintain a pattern)*:

$$\begin{pmatrix} F_0 \\ F_1 \\ F_2 \\ F_3 \\ F_4 \\ F_5 \\ F_6 \\ F_7 \end{pmatrix} = \begin{pmatrix} \begin{pmatrix} X_0 & +X_2 & +X_4 & +X_6 \\ X_0 & +\omega^2 X_2 & +\omega^4 X_4 & +\omega^6 X_6 \\ X_0 & +\omega^4 X_2 & +\omega^8 X_4 & +\omega^{12} X_6 \\ X_0 & +\omega^6 X_2 & +\omega^{12} X_4 & +\omega^{18} X_6 \end{pmatrix} + \begin{pmatrix} X_1 & +X_3 & +X_5 & +X_7 \\ \omega(X_1 & +\omega^2 X_3 & +\omega^4 X_5 & +\omega^6 X_7) \\ \omega^2(X_1 & +\omega^4 X_3 & +\omega^8 X_5 & +\omega^{12} X_7) \\ \omega^3(X_1 & +\omega^6 X_3 & +\omega^{12} X_5 & +\omega^{18} X_7) \end{pmatrix} \\ \begin{pmatrix} X_0 & +X_2 & +X_4 & +X_6 \\ X_0 & +\omega^2 X_2 & +\omega^4 X_4 & +\omega^6 X_6 \\ X_0 & +\omega^4 X_2 & +\omega^8 X_4 & +\omega^{12} X_6 \\ X_0 & +\omega^6 X_2 & +\omega^{12} X_4 & +\omega^{18} X_6 \end{pmatrix} - \begin{pmatrix} X_1 & +X_3 & +X_5 & +X_7 \\ \omega(X_1 & +\omega^2 X_3 & +\omega^4 X_5 & +\omega^6 X_7) \\ \omega^2(X_1 & +\omega^4 X_3 & +\omega^8 X_5 & +\omega^{12} X_7) \\ \omega^3(X_1 & +\omega^6 X_3 & +\omega^{12} X_5 & +\omega^{18} X_7) \end{pmatrix} \end{pmatrix}. \tag{30.10.2}$$

- As before, we observe there are only two independent sets of terms that we need to compute.
- We define two blocks $\boldsymbol{E}$ and $\boldsymbol{O}$ as follows:

$$\begin{aligned} \boldsymbol{E} &= \begin{pmatrix} X_0 & +X_2 & +X_4 & +X_6 \\ X_0 & +\omega^2 X_2 & +\omega^4 X_4 & +\omega^6 X_6 \\ X_0 & +\omega^4 X_2 & +\omega^8 X_4 & +\omega^{12} X_6 \\ X_0 & +\omega^6 X_2 & +\omega^{12} X_4 & +\omega^{18} X_6 \end{pmatrix}, \\ \boldsymbol{O} &= \begin{pmatrix} X_1 & +X_3 & +X_5 & +X_7 \\ X_1 & +\omega^2 X_3 & +\omega^4 X_5 & +\omega^6 X_7 \\ X_1 & +\omega^4 X_3 & +\omega^8 X_5 & +\omega^{12} X_7 \\ X_1 & +\omega^6 X_3 & +\omega^{12} X_5 & +\omega^{18} X_7 \end{pmatrix}. \end{aligned} \tag{30.10.3}$$

- **Notice that each block is the same as in eq. (30.9.1), with $\omega^2 = e^{-i2\pi/4}$, which is exactly the primitive root of unity in eq. (30.9.1).**

- **Therefore we can calculate each block $\boldsymbol{E}$ and $\boldsymbol{O}$ usng the algorithm for $n = 4$.**
- To complete the procedure, we define $\boldsymbol{\Omega}$ as follows:

$$\boldsymbol{\Omega} = \begin{pmatrix} 1 & & & \\ & \omega & & \\ & & \omega^2 & \\ & & & \omega^3 \end{pmatrix} . \tag{30.10.4}$$

- We define $\boldsymbol{F}_+$ and $\boldsymbol{F}_-$ (with four components) via

$$\boldsymbol{F}_+ = \begin{pmatrix} F_0 \\ F_1 \\ F_2 \\ F_3 \end{pmatrix} , \qquad \boldsymbol{F}_- = \begin{pmatrix} F_4 \\ F_5 \\ F_6 \\ F_7 \end{pmatrix} . \tag{30.10.5}$$

- Then we see from above that once again

$$\begin{aligned} \boldsymbol{F}_+ &= \boldsymbol{E} + \boldsymbol{\Omega}\boldsymbol{O} , \\ \boldsymbol{F}_- &= \boldsymbol{E} - \boldsymbol{\Omega}\boldsymbol{O} . \end{aligned} \tag{30.10.6}$$

30.11 FFT: general case $n = 2^s$

- The pattern established above for $n = 4$ and 8 clearly generalizes to $n = 2^s$ for any $k \geq 1$.
- We define the primitive root of unity $\omega_s = e^{-i2\pi/n} = e^{-i2\pi/2^s}$.
- We group the X_k into even/odd combinations with $n/2$ components each:

$$
\begin{aligned}
\boldsymbol{E}_k &= \begin{pmatrix}
X_0 & +X_2 & +X_4 & +\cdots & +X_{n-2} \\
X_0 & +\omega_s^2 X_2 & +\omega_s^4 X_4 & +\cdots & +\omega_s^{n-2} X_{n-2} \\
X_0 & +\omega_s^4 X_2 & +\omega_s^8 X_4 & +\cdots & +\omega_s^{2(n-2)} X_{n-2} \\
 & \vdots & & & \\
X_0 & +\omega_s^{n-2} X_2 & +\omega_s^{2(n-2)} X_4 & +\cdots & +\omega_s^{(n-2)^2/2} X_{n-2}
\end{pmatrix}, \\
\boldsymbol{O}_k &= \begin{pmatrix}
X_1 & +X_3 & +X_5 & +\cdots & +X_{n-1} \\
X_1 & +\omega_s^2 X_3 & +\omega_s^4 X_5 & +\cdots & +\omega_s^{n-2} X_{n-1} \\
X_1 & +\omega_s^4 X_3 & +\omega_s^8 X_5 & +\cdots & +\omega_s^{2(n-2)} X_{n-1} \\
 & \vdots & & & \\
X_1 & +\omega_s^{n-2} X_3 & +\omega_s^{2(n-2)} X_5 & +\cdots & +\omega_s^{(n-2)^2/2} X_{n-1}
\end{pmatrix}.
\end{aligned}
\tag{30.11.1}
$$

- Each block is a Discrete Fourier Transform of $n/2$ points sampled at equal intervals, with $\omega_s^2 = e^{i2\pi/(n/2)} = e^{i2\pi/2^{k-1}}$, which is exactly the root of unity required for $n/2 = 2^{k-1}$.
- Therefore we calculate each block $\boldsymbol{E}_k$ and $\boldsymbol{O}_k$ recursively using the algorithm with $n/2$.
- To complete the procedure, we define $\boldsymbol{\Omega}_k$ as follows:

$$
\boldsymbol{\Omega}_k = \begin{pmatrix}
1 & & & \\
 & \omega_s & & \\
 & & \ddots & \\
 & & & \omega_s^{(n/2)-1}
\end{pmatrix}.
\tag{30.11.2}
$$

- We define $\boldsymbol{F}_{k+}$ and $\boldsymbol{F}_{k-}$ (with $n/2$ components each) via

$$
\boldsymbol{F}_{k+} = \begin{pmatrix} F_{k,0} \\ F_{k,1} \\ \vdots \\ F_{k,(n/2)-1} \end{pmatrix}, \qquad
\boldsymbol{F}_{k-} = \begin{pmatrix} F_{k,n/2} \\ F_{k,(n/2)+1} \\ \vdots \\ F_{k,n-1} \end{pmatrix}.
\tag{30.11.3}
$$

- Then as before we have

$$
\begin{aligned}
\boldsymbol{F}_{k+} &= \boldsymbol{E}_k + \boldsymbol{\Omega}_k \boldsymbol{O}_k\,, \\
\boldsymbol{F}_{k-} &= \boldsymbol{E}_k - \boldsymbol{\Omega}_k \boldsymbol{O}_k\,.
\end{aligned}
\tag{30.11.4}
$$

30.12 Bit reversal

- The FFT implementation in Sec. 30.11 is a bit simplistic.
- At every step of the recursion, we have to collect the points into even/odd sets.
- Let us see how this works for $n = 8$, hence we have eight data points $(X_0, \dots, X_7)$.
- At each stage of the recursion, we want to group the points into sets as follows:

$$
\begin{pmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \\ X_4 \\ X_5 \\ X_6 \\ X_7 \end{pmatrix}
\longleftarrow
\left\{
\begin{array}{l}
\begin{pmatrix} X_0 \\ X_2 \\ X_4 \\ X_6 \end{pmatrix}
\longleftarrow
\left\{
\begin{array}{l}
\begin{pmatrix} X_0 \\ X_4 \end{pmatrix} \longleftarrow \left\{ \begin{array}{l} \begin{pmatrix} X_0 \end{pmatrix} \\ \begin{pmatrix} X_4 \end{pmatrix} \end{array} \right. \\
\begin{pmatrix} X_2 \\ X_6 \end{pmatrix} \longleftarrow \left\{ \begin{array}{l} \begin{pmatrix} X_2 \end{pmatrix} \\ \begin{pmatrix} X_6 \end{pmatrix} \end{array} \right.
\end{array}
\right. \\
\begin{pmatrix} X_1 \\ X_3 \\ X_5 \\ X_7 \end{pmatrix}
\longleftarrow
\left\{
\begin{array}{l}
\begin{pmatrix} X_1 \\ X_5 \end{pmatrix} \longleftarrow \left\{ \begin{array}{l} \begin{pmatrix} X_1 \end{pmatrix} \\ \begin{pmatrix} X_5 \end{pmatrix} \end{array} \right. \\
\begin{pmatrix} X_3 \\ X_7 \end{pmatrix} \longleftarrow \left\{ \begin{array}{l} \begin{pmatrix} X_3 \end{pmatrix} \\ \begin{pmatrix} X_7 \end{pmatrix} \end{array} \right.
\end{array}
\right.
\end{array}
\right.
\tag{30.12.1}
$$

- It would therefore be helpful if we sorted the points in the following order:

$$
\begin{pmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \\ X_4 \\ X_5 \\ X_6 \\ X_7 \end{pmatrix}
\quad \underset{\text{sort}}{\longrightarrow} \quad
\begin{pmatrix} X_0 \\ X_4 \\ X_2 \\ X_6 \\ X_1 \\ X_5 \\ X_3 \\ X_7 \end{pmatrix} .
\tag{30.12.2}
$$

- The sorting procedure is given by **bit reversal.**
- Write the indices $(0, 1, \dots, 7)$ in binary and **reverse the binary digits.**
- The result is the desired sort.

j	binary	bit reverse	sorted
0	000	000	0
1	001	100	4
2	010	010	2
3	011	110	6
4	100	001	1
5	101	101	5
6	110	011	3
7	111	111	7

- What are the sums we obtain?

$$
\begin{pmatrix} F_0 \\ F_1 \\ F_2 \\ F_3 \\ F_4 \\ F_5 \\ F_6 \\ F_7 \end{pmatrix}
\longleftarrow
\left\{
\begin{array}{l}
\begin{pmatrix} X_0 + X_4 + (X_2 + X_6) \\ X_0 - X_4 + (X_2 - X_6)\omega_2 \\ X_0 + X_4 - (X_2 + X_6) \\ X_0 - X_4 - (X_2 - X_6)\omega_2 \end{pmatrix}
\longleftarrow
\left\{
\begin{array}{l}
\begin{pmatrix} X_0 + X_4 \\ X_0 - X_4 \end{pmatrix} \longleftarrow \left\{ \begin{array}{l} (X_0) \\ (X_4) \end{array} \right. \\
\begin{pmatrix} X_2 + X_6 \\ X_2 - X_6 \end{pmatrix} \longleftarrow \left\{ \begin{array}{l} (X_2) \\ (X_6) \end{array} \right.
\end{array}
\right. \\
\begin{pmatrix} X_1 + X_5 + (X_3 + X_7) \\ X_1 - X_5 + (X_3 - X_7)\omega_2 \\ X_1 + X_5 - (X_3 + X_7) \\ X_1 - X_5 - (X_3 - X_7)\omega_2 \end{pmatrix}
\longleftarrow
\left\{
\begin{array}{l}
\begin{pmatrix} X_1 + X_5 \\ X_1 - X_5 \end{pmatrix} \longleftarrow \left\{ \begin{array}{l} (X_1) \\ (X_5) \end{array} \right. \\
\begin{pmatrix} X_3 + X_7 \\ X_3 - X_7 \end{pmatrix} \longleftarrow \left\{ \begin{array}{l} (X_3) \\ (X_7) \end{array} \right.
\end{array}
\right.
\end{array}
\right.
\tag{30.12.3}
$$

- Do we obtain the correct answer? Let us check. Note that $\omega_2 = \omega_3^2$ and $\omega_3^4 = -1$.

$$
\begin{aligned}
F_0 &= X_0 + X_4 + (X_2 + X_6) + [X_1 + X_5 + (X_3 + X_7)] \\
&= X_0 + X_1 + X_2 + X_3 + X_4 + X_5 + X_3 + X_7\,, && \text{(30.12.4a)} \\
F_1 &= X_0 - X_4 + \omega_2(X_2 - X_6) + \omega_3[X_1 - X_5 + \omega_2(X_3 - X_7)] \\
&= X_0 + \omega_3 X_1 + \omega_3^2 X_2 + \omega_3^3 X_3 + \omega_3^4 X_4 + \omega_3^5 X_5 + \omega_3^6 X_6 + \omega_3^7 X_7\,, && \text{(30.12.4b)} \\
F_2 &= X_0 + X_4 - (X_2 + X_6) + \omega_3^2[X_1 + X_5 - (X_3 + X_7)] \\
&= X_0 + \omega_3^2 X_1 + \omega_3^4 X_2 + \omega_3^6 X_3 + \omega_3^8 X_4 + \omega_3^{10} X_5 + \omega_3^{12} X_6 + \omega_3^{14} X_7\,, && \text{(30.12.4c)} \\
F_3 &= X_0 - X_4 - \omega_2(X_2 - X_6) + \omega_3^3[X_1 - X_5 - \omega_2(X_3 - X_7)] \\
&= X_0 + \omega_3^3 X_1 + \omega_3^6 X_2 + \omega_3^9 X_3 + \omega_3^{12} X_4 + \omega_3^{15} X_5 + \omega_3^{18} X_6 + \omega_3^{21} X_7\,, && \text{(30.12.4d)} \\
F_4 &= X_0 + X_4 + (X_2 + X_6) - [X_1 + X_5 + (X_3 + X_7)] \\
&= X_0 + \omega_3^4 X_1 + \omega_3^8 X_2 + \omega_3^{12} X_3 + \omega_3^{16} X_4 + \omega_3^{20} X_5 + \omega_3^{24} X_6 + \omega_3^{28} X_7\,, && \text{(30.12.4e)} \\
F_5 &= X_0 - X_4 + \omega_2(X_2 - X_6) - \omega_3[X_1 - X_5 + \omega_2(X_3 - X_7)] \\
&= X_0 + \omega_3^5 X_1 + \omega_3^{10} X_2 + \omega_3^{15} X_3 + \omega_3^{20} X_4 + \omega_3^{25} X_5 + \omega_3^{30} X_6 + \omega_3^{35} X_7\,, && \text{(30.12.4f)} \\
F_6 &= X_0 + X_4 - (X_2 + X_6) - \omega_3^2[X_1 + X_5 - (X_3 + X_7)] \\
&= X_0 + \omega_3^6 X_1 + \omega_3^{12} X_2 + \omega_3^{18} X_3 + \omega_3^{24} X_4 + \omega_3^{30} X_5 + \omega_3^{36} X_6 + \omega_3^{42} X_7\,, && \text{(30.12.4g)} \\
F_7 &= X_0 - X_4 - \omega_2(X_2 - X_6) - \omega_3^3[X_1 - X_5 - \omega_2(X_3 - X_7)] \\
&= X_0 + \omega_3^7 X_1 + \omega_3^{14} X_2 + \omega_3^{21} X_3 + \omega_3^{28} X_4 + \omega_3^{35} X_5 + \omega_3^{42} X_6 + \omega_3^{49} X_7\,. && \text{(30.12.4h)}
\end{aligned}
$$

- **Hence if we sort (or rearrange) the input data $(X_0, \ldots, X_{n-1})$ using bit-reversal, the algorithm in Sec. 30.11 can be implemented straightforwardly.**

- **Computational complexity.**
  Observe that the number of computations (multiplications) is about
  $8 + 8 + 8 = 8 \times 3 = 8 \log_2 8 = O(n \log_2 n)$.

30.13 *Is bit reversal necessary? Recursion? Extra memory storage?*

- Neither bit reversal, recursion nor extra memory storage are necessary.
- There exist FFT implementations which do not require bit reversal.
- Recursion: the FFT algorithm can be formulated using a set of nested loops.
- Extra memory storage: study the calculations in Sec. 30.12.
- At every step of the recursion (three steps because $n = 2^3 = 8$), we have n computed values.
- For example in the first calculation we have ($X_0 + X_4$, $X_0 - X_4$, $X_2 + X_6$, $X_2 - X_6$, $X_1 + X_5$, $X_1 - X_5$, $X_3 + X_7$, $X_3 - X_7$).
- The values can be held in an array of length n, which is $(F_0, \ldots, F_{n-1})$ itself.
- All the results of computations can be stored 'in place' in the array $(F_0, \ldots, F_{n-1})$.
- The even/odd arrays $\boldsymbol{E}$ and $\boldsymbol{O}$ can be held in $(F_0, \ldots, F_{n-1})$ itself.
- A few temporary variables may be required, but no extra memory of length n or $n/2$, etc.

30.14 FFT review

30.14.1 Main routine

- Let us review the the steps of the FFT algorithm.
- We consider only inputs where the number of points n is a power of 2.
- There are FFT algorithms for other values of n but we do not consider them.
- We begin with a bit reversal of the input data $(X_0, \dots, X_{n-1})$.
  1. Some authors perform the bit reversal 'in place' and overwrite the input data.
  2. We store the bit reversed data in a separate array and do not overwrite the input data.
  3. Let us denote the bit reversed array by $\boldsymbol{R} = (R_0, \dots, R_{n-1})$.
- We compute the primitive root of unity $\omega = e^{-i2\pi/n}$. For the inverse we compute $\omega = e^{i2\pi/n}$.
- We call a lower level function to compute the FFT recursively, with inputs n, ω and $\boldsymbol{R}$.
- For the inverse FFT, after the recursion is over, we divide $(F_0, \dots, F_{n-1})$ by n.

30.14.2 Recursion

- If $n = 1$, we set $F_0 = R_0$ and return.
- If $n = 2$, we set $F_0 = R_0 + R_1$ and $F_1 = R_0 - R_1$ and return.
- **Note in this context that 'F' and 'R' are arrays which have been passed to a recursive function call. They are *not* the array indices `F[0]`, `F[1]`, `R[0]`, `R[1]` of the original arrays in the top level FFT routine.**
- If $n > 2$, we proceed as follows.
- First define a temporary variable $w_2 = \omega^2$.
- Declare temporary arrays `Even` and `Odd` of length $n/2$.
- The arrays `Even` and `Odd` do *not* require additional memory.
- They can be implemented as **pointers to sections of the array F.**
- **Call the FFT routine recursively twice, once to compute `Even` and once for `Odd`.**
  1. The inputs to the recursive calls are $n/2$, w_2 and suitable sections of the array $\boldsymbol{R}$.
  2. Note that in a recursive function call, the parameters 'n' and 'ω' do not necessarily have the same values as in the top level routine.
  3. The compiler will take care of the details.
- After both recursive calls have completed, compute $\boldsymbol{F}_+$ and $\boldsymbol{F}_-$ from `Even` and `Odd`.
- Both $\boldsymbol{F}_+$ and $\boldsymbol{F}_-$ can be stored 'in place' in the array F.

30.15 FFT code

30.15.1 Bit reversal

There are multiple ways to code the bit reversal.

```
int bit_reverse(int num_bits, int n)
{
  int r = 0;
  for (int i=0; i < num_bits; ++i) {
    int rem = (n & 1);
    r = r*2 + rem;
    n /= 2;
  }
  return r;
}

int bit_reverse2(int num_bits, int n)
{
  const int nmax = 1 << (num_bits-1); // 2^(num_bits-1)
  int r = 0;
  for (int i = 0; i < num_bits; ++i) {
    int j = (1 << i);
    if (n & j) {
      int k = nmax >> i;
      r |= k;
    }
  }
  return r;
}
```

30.15.2 FFT recursion

```
static void FFT_recursion(int npts,
                          const std::complex<double> & omega,
                          const std::complex<double> * R,
                          std::complex<double> * F)
{
  if (npts == 1) {
    F[0] = R[0];
    return;
  }
  if (npts == 2) {
    F[0] = R[0] + R[1];
    F[1] = R[0] - R[1];
    return;
  }

  const std::complex<double> w2 = omega*omega;
  const int nhalf = npts/2;
  std::complex<double> * Even = F;
  std::complex<double> * Odd = F + nhalf;
  FFT_recursion(nhalf, w2, R, Even);       // use array f, no extra storage required
  FFT_recursion(nhalf, w2, R+nhalf, Odd);  // use array f, no extra storage required

  std::complex<double> wtmp(1.0, 0.0);
  for (int i = 0; i < nhalf; ++i) {
    std::complex<double> Fplus  = Even[i] + wtmp*Odd[i]; // Even + Omega*Odd
    std::complex<double> Fminus = Even[i] - wtmp*Odd[i]; // Even + Omega*Odd
    F[i] = Fplus;
    F[i+nhalf] = Fminus;
    wtmp *= omega;
  }
}
```

30.15.3 FFT top level function

```
static void FFT_top(bool inverse,
                    int num_bits,
                    int npts,
                    const std::complex<double> * X,
                    std::complex<double> * F)
{
  // bit reversal
  std::complex<double> R[npts];
  for (int i = 0; i < npts; ++i) {
    int i_rev = bit_reverse(num_bits, i);
    R[i_rev] = X[i];
  }

  // primitive root of unity
  const double pi = 4.0*atan2(1.0,1.0);
  double ang = 2.0*pi/npts;
  if (inverse) ang = -ang;
  std::complex<double> omega(cos(ang), -sin(ang));  // note the minus sign

  // FFT recursion
  FFT_recursion(npts, omega, R, F);

  if (inverse) {
    double ndbl = double(npts);
    for (int i = 0; i < npts; ++i) {
      F[i] /= ndbl;
    }
  }
}
```