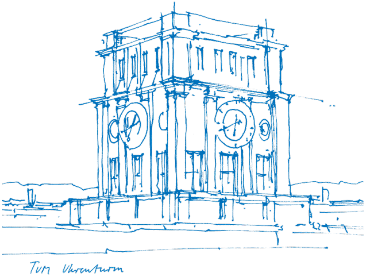
Algorithms of Scientific Computing

Discrete Fourier Transform (DFT)

Michael Bader
Technical University of Munich

Summer 2022



Fast Fourier Transform – Outline

- Discrete Fourier transform
- Fast Fourier transform
- Special Fourier transform:
  - real-valued FFT
  - sine/cosine transform
- Applications:
  - Fast Poisson solver (FST)
  - Computergraphics (FCT)
- Efficient Implementation

Discrete Fourier Transform (DFT)

Definition:
For a vector of N complex numbers $(f_0, \dots, f_{N-1})^T$, the **discrete Fourier transform** (DFT) is given by the vector $(F_0, \dots, F_{N-1})^T$, where

$$F_k = \frac{1}{N} \sum_{n=0}^{N-1} f_n e^{-i2\pi nk/N}.$$

Interpretation:

The DFT can be derived as (to be discussed ...):

- trigonometric interpolation/approximation
- approximation of the coefficients of the Fourier series

Applications:

- JPEG, MPEG, MP3, etc.
- signal processing in general, solvers in science and engineering

DFT as the Solution of an Interpolation Problem

Interpolation problem:

- N ansatz functions: $g_k(x) := e^{ikx}$ in the interval $[0, 2\pi]$, $k = 0, \ldots, N-1$
- N supporting points: $x_n := 2\pi n/N$, $n = 0, \ldots, N-1$
- N interpolation value f_n, $n = 0, \ldots, N-1$
- find N weights F_k such that at all supporting points

$$f_n = \sum_{k=0}^{N-1} F_k g_k(x_n) \qquad \Leftrightarrow \qquad f_n = \sum_{k=0}^{N-1} F_k e^{i2\pi nk/N}.$$

"trigonometric interpolation"

DFT as Interpolation – Formulation via Complex Polynomials

Interpolation problem:

- N ansatz functions: $\widetilde{g}_k(z) := z^k$ (complex unit polynomials), $k = 0, \ldots, N-1$
- N supporting points: $z_n := e^{i2\pi n/N} = \omega_N^n$, where $\omega_N := e^{i2\pi/N}$
- N interpolation values f_n, $n = 0, \ldots, N-1$, respectively.
- find the N weights F_k such that at all supporting points

$$f_n = \sum_{k=0}^{N-1} F_k \widetilde{g}_k(z_n) \qquad \Leftrightarrow \qquad f_n = \sum_{k=0}^{N-1} F_k e^{i2\pi nk/N}.$$

Polynomial interpolation at the **"complex unit roots"** ω_N^n

Interpretation of the Interpolation Problem

Starting from the first formulation,

$$f_n = \sum_{k=0}^{N-1} F_k g_k(x_n), \qquad g_k(x_n) = e^{i2\pi nk/N},$$

we look for a representation of the signal f_n – or of a function $f(x)$ – of the form

$$f(x) = \sum_{k=0}^{N-1} F_k g_k(x), \qquad g_k(x) = e^{i2\pi kx}.$$

The ansatz functions are sine or cosine oscillations:

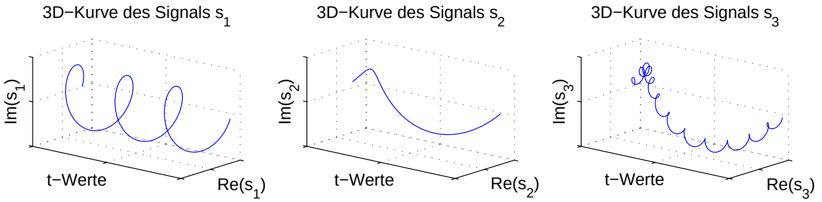
$$e^{ikx} = \cos(kx) + i\sin(kx)$$

Interpretation of the Interpolation Problem (2)

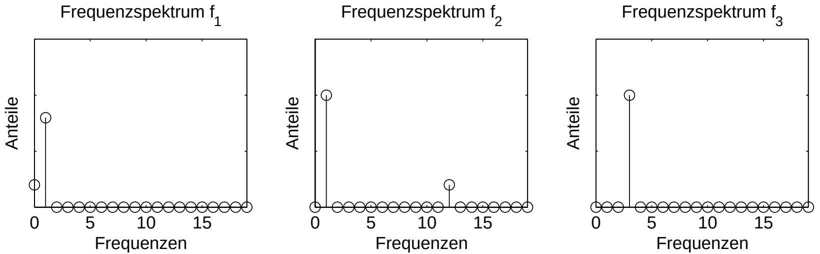
Conclusions:

- we look for the **representation of a periodic function** as a sum of sine and cosine modes
- the F_k are, thus, called **Fourier coefficients**:
  - k represents the wave number
  - the value of F_k represents the amplitude of the corresponding frequency
- the Fourier transform leads to a **frequency spectrum**
- useful when a problem is easier to solve in the **frequency domain** than in the **spatial domain**.

Example: Spatial vs. Frequency Domain



$$s_1 = e^{3it} \qquad s_2 = 0.2\,i + 0.8\,e^{it} \qquad s_3 = e^{it} + 0.2\,e^{12it}$$



Solution of the Interpolation Problem

Both interpolation problems lead to the identical linear systems of equations:

$$f_n = \sum_{k=0}^{N-1} F_k \omega_N^{nk}, \quad \text{for all } n = 0, \ldots, N-1;$$

where $\omega_N := e^{i2\pi/N}$, i.e. $\omega_N^{nk} := e^{i2\pi nk/N}$ (**"unit roots": $\omega_N^N = 1$**)

If we write the vectors of the f_n and F_k as $\mathbf{f} := (f_0, \ldots, f_{N-1})$ and $\mathbf{F} := (F_0, \ldots, F_{N-1})$, the linear system of equations can be formulated in matrix-vector notation

$$\mathbf{WF} = \mathbf{f},$$

where the entries of the **Fourier matrix** **W** are given by $W_{nk} := \omega_N^{nk}$.

Next Step(s): Show that system $\mathbf{WF} = \mathbf{f}$ can be solved with effort $\mathcal{O}(N^2)$ – and using FFT even only $\mathcal{O}(N \log N)$ – instead of $\mathcal{O}(N^3)$.

Properties of the Fourier Matrix W

- **W** is symmetric: $\mathbf{W} = \mathbf{W}^T$, and has the form

$$\mathbf{W} = \begin{pmatrix} \omega_N^0 & \omega_N^0 & \omega_N^0 & \dots & \omega_N^0 \\ \omega_N^0 & \omega_N^1 & \omega_N^2 & \dots & \omega_N^{(N-1)} \\ \omega_N^0 & \omega_N^2 & \omega_N^4 & \dots & \omega_N^{2(N-1)} \\ \vdots & \vdots & \vdots & & \vdots \\ \omega_N^0 & \omega_N^{(N-1)} & \omega_N^{2(N-1)} & \dots & \omega_N^{(N-1)(N-1)} \end{pmatrix}$$

- $\mathbf{W}\left(\mathbf{W}^T\right)^* = \mathbf{W}\mathbf{W}^H = N\mathbf{I}$, since

$$\left[\mathbf{W}\mathbf{W}^H\right]_{kl} = \sum_{j=0}^{N-1} \omega_N^{kj}\left(\omega_N^{lj}\right)^* = \sum_{j=0}^{N-1} \omega_N^{(k-l)j} = \begin{cases} N & \text{if} \quad k = l \\ 0 & \text{if} \quad k \neq l. \end{cases}$$

$\rightsquigarrow$ inverse of **W** easily available!

Properties of the Fourier Matrix W (details)

- important property #1: $(\omega_N^k)^N = e^{N \cdot i2\pi k/N} = e^{i2\pi k} = 1$
- important property #2: $(\omega_N^k)^* = \left(e^{i2\pi k/N}\right)^* = (\cos(2\pi k/N) + i \sin(2\pi k/N))^* = \cos(2\pi k/N) - i \sin(2\pi k/N) = \cos(-2\pi k/N) + i \sin(-2\pi k/N) = e^{-i2\pi k/N} = \omega_N^{-k}$
- $\mathbf{W}\left(\mathbf{W}^T\right)^* = \mathbf{W}\mathbf{W}^H = N\,\mathbf{I}$, since

$$\left[\mathbf{W}\mathbf{W}^H\right]_{kl} = \sum_{j=0}^{N-1} \omega_N^{kj} \left(\omega_N^{lj}\right)^* = \sum_{j=0}^{N-1} \omega_N^{kj} \omega_N^{-lj} = \sum_{j=0}^{N-1} \omega_N^{(k-l)j}$$

- if $k = l$ then $\sum_{j=0}^{N-1} \omega_N^{(k-l)j} = \sum_{j=0}^{N-1} \omega_N^0 = N$
- if $k \neq l$ then $\sum_{j=0}^{N-1} \omega_N^{(k-l)j} = \sum_{j=0}^{N-1} \xi^j$ where $\xi = \omega_N^{(k-l)}$ and then:

$$(1-\xi) \sum_{j=0}^{N-1} \xi^j = \sum_{j=0}^{N-1} \xi^j - \sum_{j=0}^{N-1} \xi^{(j+1)} = 1 - \xi^N, \text{thus} \quad \sum_{j=0}^{N-1} \xi^j = \frac{1-\xi^N}{1-\xi}$$

Remember that $(\omega_N^k)^N = 1$ for any k (“unit roots”!), thus $\sum_{j=0}^{N-1} \omega_N^{(k-l)j} = 0$

Computation of the Fourier Coefficients F_k

- Since $\mathbf{W}\mathbf{W}^H = N\,\mathbf{I}$, the inverse of $\mathbf{W}$ is $\mathbf{W}^{-1} = \frac{1}{N}\mathbf{W}^H$:

$$\mathbf{W}^{-1} = \frac{1}{N}\begin{pmatrix} \omega_N^0 & \omega_N^0 & \omega_N^0 & \dots & \omega_N^0 \\ \omega_N^0 & \omega_N^{-1} & \omega_N^{-2} & \dots & \omega_N^{-(N-1)} \\ \omega_N^0 & \omega_N^{-2} & \omega_N^{-4} & \dots & \omega_N^{-2(N-1)} \\ \vdots & \vdots & \vdots & & \vdots \\ \omega_N^0 & \omega_N^{-(N-1)} & \omega_N^{-2(N-1)} & \dots & \omega_N^{-(N-1)(N-1)} \end{pmatrix}$$

$\Rightarrow$ the vector $\mathbf{F}$ of the Fourier coefficients can be computed **easily** as a matrix-vector product – with computational effort $\mathcal{O}(N^2)$:

$$\mathbf{F} = \frac{1}{N}\mathbf{W}^H\mathbf{f} \qquad \text{or} \qquad F_k = \frac{1}{N}\sum_{n=0}^{N-1} f_n \omega_N^{-nk}.$$

Inverse Discrete Fourier Transform (IDFT)

Definition:
The inverse Discrete Fourier Transform (IDFT) of the vector $(F_0, \ldots, F_{N-1})$ is given by the vector $(f_0, \ldots, f_{N-1})$, where

$$f_n = \sum_{k=0}^{N-1} F_k e^{i2\pi nk/N}.$$

Observation:
DFT and IDFT are inverse operations:

$$F_k = \frac{1}{N} \sum_{n=0}^{N-1} f_n e^{-i2\pi nk/N}, \qquad f_n = \sum_{k=0}^{N-1} F_k e^{i2\pi nk/N}.$$

$$\mathbf{F} = \mathrm{DFT}(\mathrm{IDFT}(\mathbf{F})) \qquad \text{or} \qquad \mathbf{f} = \mathrm{IDFT}(\mathrm{DFT}(\mathbf{f})).$$

The Pair DFT/IDFT as Matrix-Vector Product

With the notation $\omega_N := e^{i2\pi/N}$, i.e. $\omega_N^{-nk} := e^{-i2\pi nk/N}$, we formulate the DFT/IDFT as

$$F_k = \frac{1}{N}\sum_{n=0}^{N-1} f_n \omega_N^{-nk} \qquad f_n = \sum_{k=0}^{N-1} F_k \omega_N^{nk}$$

With the vectors $\mathbf{f} := (f_0, \ldots, f_{N-1})^T$ and $\mathbf{F} := (F_0, \ldots, F_{N-1})^T$, we denote (and compute) the DFT und IDFT as matrix-vector products

$$\mathbf{F} = \frac{1}{N}\mathbf{W}^H\mathbf{f}\,, \qquad \mathbf{f} = \mathbf{W}\mathbf{F}\,,$$

where the elements of the Fourier matrix **W** are $W_{nk} := \omega_N^{nk}$.

Properties of the DFT

- DFT and IDFT are (as a matrix-vector product) **linear**:

$$\begin{aligned} \mathrm{DFT}(\alpha f + \beta g) &= \alpha\ \mathrm{DFT}(f) + \beta\ \mathrm{DFT}(g) \\ \mathrm{IDFT}(\alpha f + \beta g) &= \alpha\ \mathrm{IDFT}(f) + \beta\ \mathrm{IDFT}(g) \end{aligned}$$

- since $\omega_N^{nk} = \omega_N^{n(k+N)} = \omega_N^{(n+N)k}$, the f_n and the F_k are **periodic**:

$$f_{n+N} = f_n \qquad F_{k+N} = F_k \qquad \text{for all } k, n \in \mathbb{Z}$$

Alternative Forms of the DFT

Possible variants (in all imaginable combinations):

- Scaling factor $\frac{1}{N}$ in the IDFT instead of the DFT;
  alternatively a factor $\frac{1}{\sqrt{N}}$ in DFT and IDFT.
- switched signs in the exponent of the exponential function in DFT and IDFT
- use j for the imaginary unit (electrical engineering)

Shift of indices:

- periodic data: $F_k = F_{k+N}$
- aliasing of frequencies: $e^{-i2\pi nk/N} = e^{-i2\pi n(k \pm N)/N}$

DFT with Shifted Indices

Data and frequencies "symmetric":

$$F_k = \frac{1}{N} \sum_{n=-\frac{N}{2}+1}^{\frac{N}{2}} f_n e^{-i2\pi nk/N}\,, \qquad f_n = \sum_{k=-\frac{N}{2}+1}^{\frac{N}{2}} F_k e^{i2\pi nk/N}$$

In general:

$$F_k = \frac{1}{N} \sum_{n=P+1}^{P+N} f_n e^{-i2\pi nk/N}\,, \qquad f_n = \sum_{k=Q+1}^{Q+N} F_k e^{i2\pi nk/N}$$

DFT in Program Libraries

Different conventions for sign factors in exponent, normalization factors etc.

- Python/NumPy: `numpy.fft`; normalization factor $\frac{1}{N}$ in *inverse* transform by default
- Matlab, IMSL (Int. Math. and Stat. Library):

$$F_{k+1} = \sum_{n=0}^{N-1} f_{n+1} e^{-i2\pi nk/N} \qquad k = 0, \ldots, N-1$$

$$f_{n+1} = \frac{1}{N} \sum_{k=0}^{N-1} F_{k+1} e^{i2\pi nk/N} \qquad n = 0, \ldots, N-1$$

- Maple: $\frac{1}{\sqrt{N}}$ as factor for DFT and IDFT.

Index shift by $+1$, since:

- Data/coefficients start at index 0
- Arrays to store the numbers start at index 1