Fast Fourier Transform

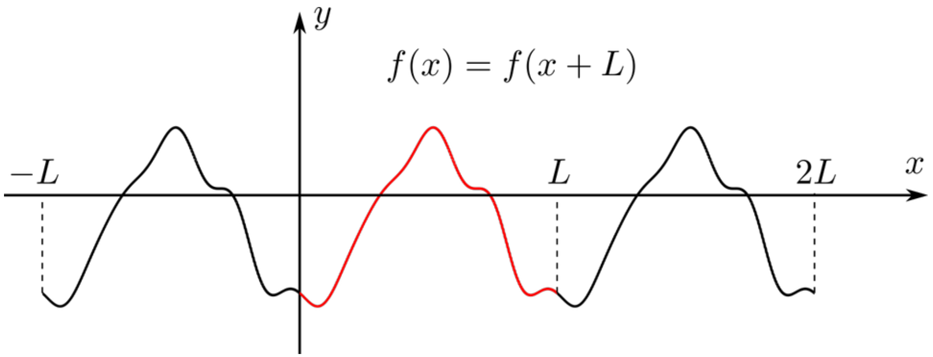
outline

- trigonometric series
- complex Fourier series
- Discrete Fourier Transformation (DFT)
    - Nyquist frequency
    - encoding positive and negative frequencies in DFT/FFT
- Fast Fourier Trasform (FFT)
    - Radix-2 (Cooley-Tukey)
    - inverse transformation
    - multidimensional FFT
- applications

Trigonometric series

Periodic function

$$f : \mathbb{R} \to \mathbb{R}, \quad f(x) \equiv f(x+L)$$



can be replaced by infinite series of sine and cosine basis functions – trigonometric interpolation

$$\boxed{f(x) = \frac{a_0}{2} + \sum_{k=1}^{\infty} a_k \cos(\gamma_k x) + b_k \sin(\gamma_k x)} \qquad \gamma_k \cdot L = 2\pi \cdot k \quad \to \quad \gamma_k = \frac{2\pi}{L} k$$

To calculate the coefficents a_k and b_k one must utilize the orthogonality property of basis function

$$\int_0^L dx \cos(\gamma_k x)\cos(\gamma_m x) = \frac{L}{2}\delta_{k,m} \qquad \int_0^L dx \sin(\gamma_k x)\sin(\gamma_m x) = \frac{L}{2}\delta_{k,m}$$

$$\int_0^L dx \cos(\gamma_m x) \quad \Longrightarrow \quad \int_0^L dx \cos(\gamma_m x) f(x) = a_k \int_0^L dx \cos(\gamma_m x)\cos(\gamma_k x) = a_k \frac{L}{2}\delta_{k,m}$$

$$\boxed{a_k = \frac{2}{L}\int_0^L dx \cos(\gamma_k x) f(x)}$$

and analogically we get the b_k

$$\boxed{b_k = \frac{2}{L}\int_0^L dx \sin(\gamma_k x) f(x)}$$

Fourier series

Euler formula for complex number of unit amplitude enables connecting two functions of the same argument

$$e^{i\theta} = \cos(\theta) + i\sin(\theta) \qquad \cos(\theta) = \frac{e^{i\theta} + e^{-i\theta}}{2} \qquad \sin(\theta) = \frac{e^{i\theta} - e^{-i\theta}}{2i}$$

$$a_k \cos(\gamma_k x) + b_k \sin(\gamma_k x) = \frac{1}{2}\left[(a_k - i\, b_k)e^{i\gamma_k x} + (a_k + i\, b_k)e^{-i\gamma_k x}\right]$$

substituting the rhs of above expression into series we get the complex Fourier seres

$$\boxed{f(x) = \sum_{k=-\infty}^{\infty} c_k e^{i\gamma_k x}, \qquad c_{k\in\mathbb{C}}} \qquad \gamma_k = \frac{2\pi}{L}k$$

Coefficents ck can be calculated directly in the same way as a_k and b_k – by projecting f(x) on each exponential function

$$\int_0^L dx\, e^{-i\gamma_m x} \cdot / \quad \rightarrow \quad \int_0^L dx\, e^{-i\gamma_m x} f(x) = \sum_{k=-\infty}^{\infty} c_k \int_0^L dx\, e^{-i(\gamma_m - \gamma_k)x} = c_k L \delta_{k,m}$$

$$\boxed{c_k = \int_0^L dx\, e^{-i\gamma_k x} f(x)}$$ ← coefficents of Fourier series

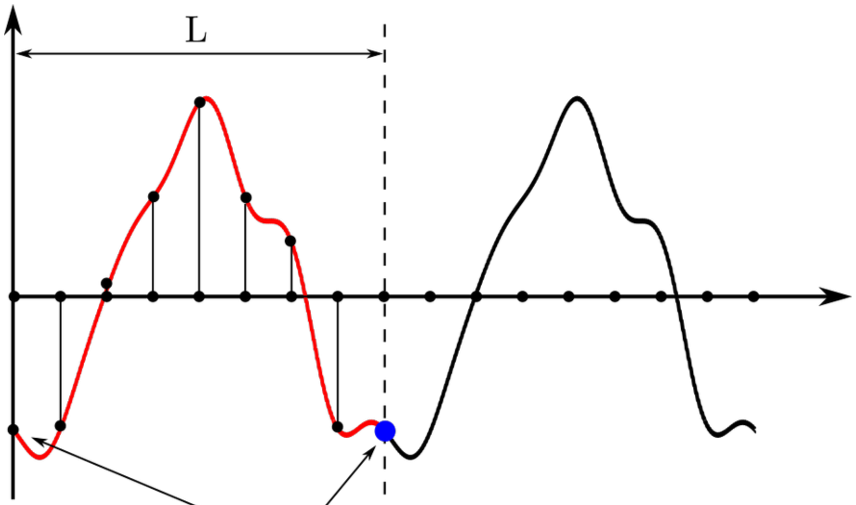
Discrete Fourier Transform (DFT)

Now let's narrow our considerations to the case when the function f(x) values are given only at discrete set of equidistant nodes. This finite number of nodes limits also the number of complex function needed to reconstruct the function values at these nodes – this corresponds to interpolation with complex exponential polynomials

We introduce nodes along x-axis

$$\Delta = \frac{L}{N}$$

$$x_j = \Delta \cdot j, \quad j = 0, 1, 2, \ldots, N-1$$



and replace the integral

$$c_k = \int_0^L dx^{-i\gamma_k x} f(x)$$

by summation over discrete values

$$c_k = \frac{1}{L} \sum_{j=0}^{N-1} e^{-i\gamma_k x_j} f(x_j) \Delta$$

$$\gamma_k x_j = \frac{2\pi}{L} k \Delta\, j = \frac{2\pi}{N} k\, j$$

$$f(x_j) = f_j$$

definition of DFT coefficients

$$\boxed{c_k = \frac{1}{N} \sum_{j=0}^{N-1} e^{-i\frac{2\pi}{N}k\,j} f_j}$$

Nyquist frequency and aliasing

By performing the DFT (FFT) transforms for equidistantly sampled data we encounter a barrier, namely, for the sampling interval Δ the number of sampling points for a sine function falls to only two for the **critical Nyquist frequency**

$$\gamma_{k_{crit}} = \frac{1}{2\Delta}$$

It means that periodic function of frequency bandwidth lower than this threshold would be completely defined by the set of function's samples.

Unfortunately, when the function's frequency exceeds this threshold, information from higher γ-s are shifted into the interval $[-\gamma_{Kcrit}, \gamma_{Kcrit}]$ disturbing these values. This effect is called **aliasing**.

To avoid this problem, function should be sampled with frequency larger than the function's frequency bandwidth, if it is possible.

Encoding positive and negative frequencies in DFT/FFT

The DFT transform should map set of N complex numbers (samples) into another set of N complex numbers to be unique. From the equation defining the DFT follows that

$$\gamma_k \in [-\gamma_{crit}, \gamma_{crit}]$$

$$c_k = \frac{1}{N} \sum_{j=0}^{N-1} e^{-i \frac{2\pi}{N} k\, j} f_j \qquad \Longrightarrow \qquad k = -\frac{N}{2}, \ldots, 0, \ldots, \frac{N}{2}$$

hence, we have N+1 data in reciprocal space. However, taking into account the periodicity of the phase factor

$$c_{-k} = c_{-k+N}$$

we reduce one excess point. Moreover, to avoid the use of negative indices for negative frequencies these are shifted by N, so the usual relations between coefficents c_k and the frequencies are following

$$k = 0, 1, 2, \ldots, N-1$$

$$\gamma_0 = 0$$

$$\gamma_k > 0 \quad \Longleftrightarrow \quad k = 1, 2, \ldots, \frac{N}{2} - 1$$

$$\gamma_k = \gamma_{crit} \quad \Longleftrightarrow \quad k = \frac{N}{2}$$

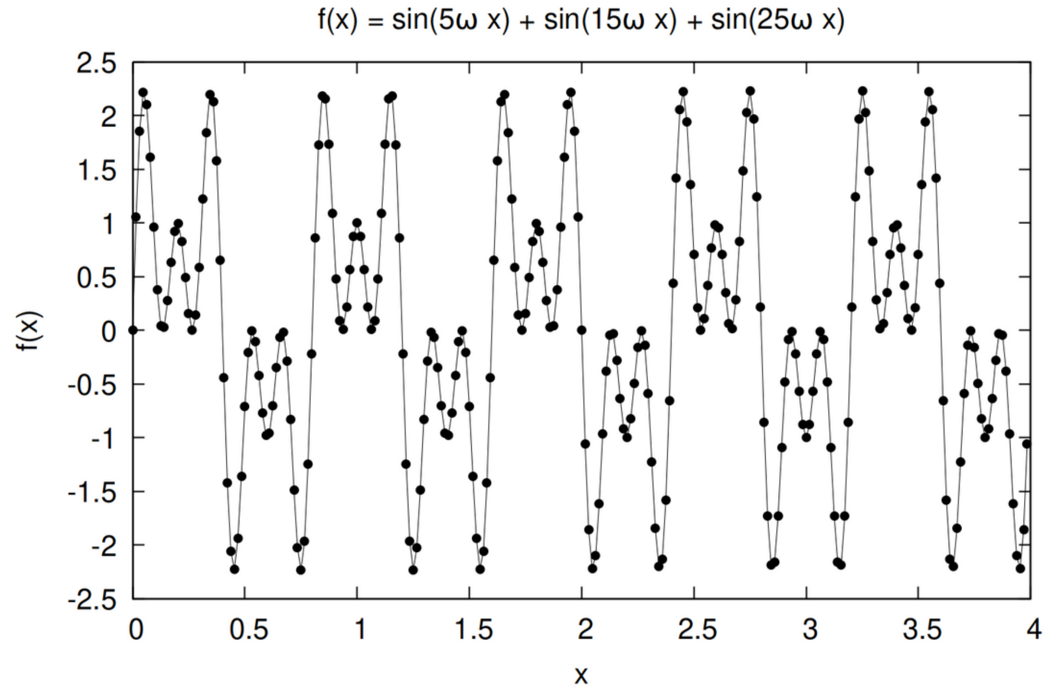
$$\gamma_k < 0 \quad \Longleftrightarrow \quad k = \frac{N}{2} + 1, \frac{N}{2} + 2, \ldots, N-1$$

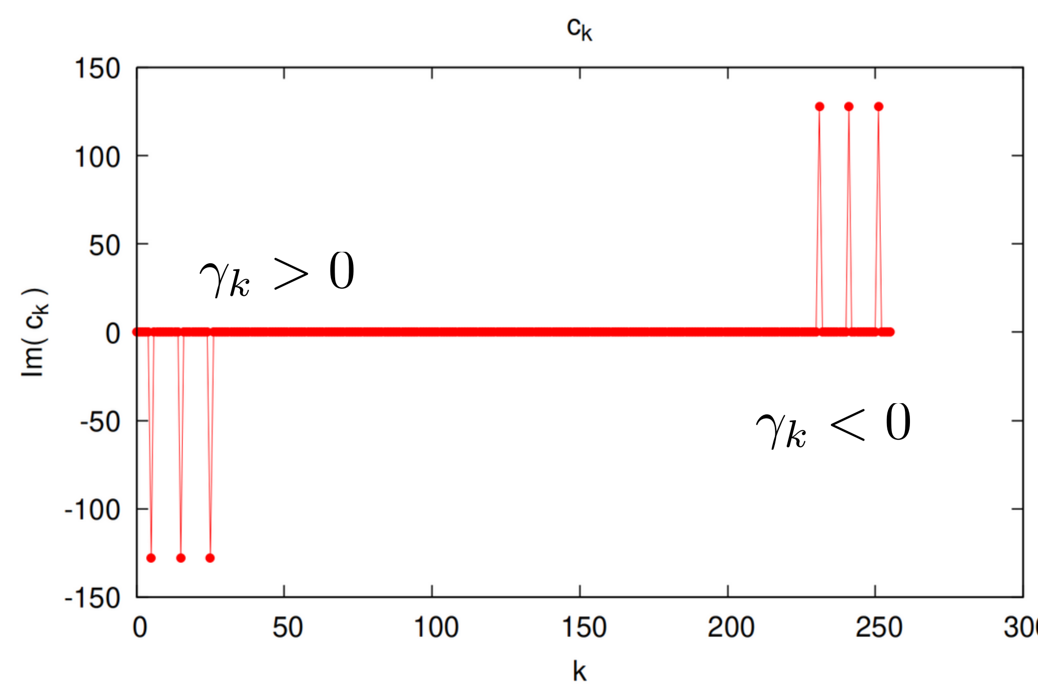
Example: FFT of f(x) given by expression

$$f(x) = \sin(5\omega\, x) + \sin(15\omega\, x) + \sin(25\omega\, x)$$

$$N = 2^8 = 256$$
$$L = 4$$
$$\Delta_x = \frac{L}{N}$$
$$\Delta_k = \frac{2\pi}{L}$$

$$Re(c_k) = 0$$
$$Im(c_k) \neq 0$$

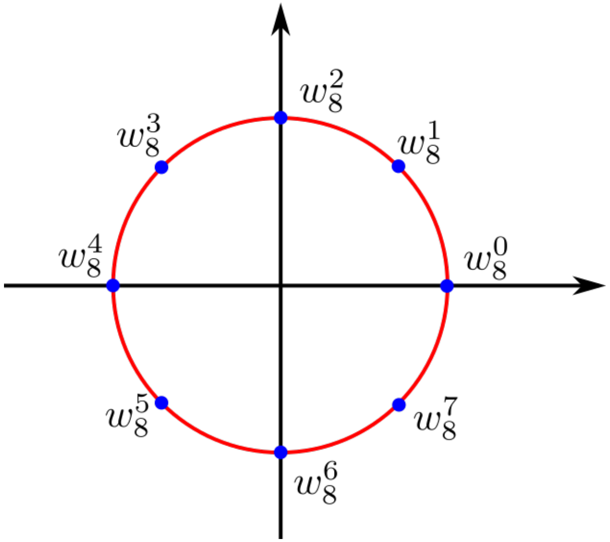




A more common notation of DFT contains twiddle factors defined as complex roots of unit

$$w_N = e^{-i\frac{2\pi}{N}}$$

$$c_k = \frac{1}{N}\sum_{j=0}^{N-1} w_N^{k\,j} f_j$$



To accomplish the DFT we need all coefficents, based on the definition equation these can be calculated as simple matrix-vector multiplication

$$\begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_{N-2} \\ c_{N-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & \mathrm{w}_N^1 & \mathrm{w}_N^2 & \dots & \dots & \mathrm{w}_N^{N-1} \\ 1 & \mathrm{w}_N^2 & \mathrm{w}_N^4 & \dots & \dots & \mathrm{w}_N^{2(N-1)} \\ 1 & \dots & \dots & \dots & \dots & \dots \\ 1 & \dots & \dots & \dots & \dots & \dots \\ 1 & \mathrm{w}_N^{N-1} & \mathrm{w}_N^{2(N-1)} & \dots & \dots & \mathrm{w}_N^{(N-1)^2} \end{bmatrix} \cdot \begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ \vdots \\ f_{N-2} \\ f_{N-1} \end{bmatrix}$$

The problem with such formulated DFT is that it requires O(N^2) operations, while the efficent FFT algorithm can reduce it to O[N*$\log_2$(N)] utilizing periodicity of twiddle factors

N	N^2	$N \cdot log_2(N)$
1024	1 048 576	10 240
4096	16 777 216	49 152
16384	268 435 456	229 375

FFT - algorithm Radix-2 (Cooley-Tukey)

The first publicly presented FFT algorithm (1965) to perform on a computer.
It assumes the number of samples to be processed is a power of 2

$$N = 2^m, \quad m \in \mathbb{N}$$

We start considering the general formula for DFT

$$c_k = \frac{1}{N} \sum_{j=0}^{N-1} e^{-i \frac{2\pi}{N} \cdot k\, j} f_j, \qquad k = 0, 1, \ldots, N-1$$

and divide the summation operation into two blocks, each contining only one type of elements, even and odd parity

$$c_k = \sum_{m=0}^{\frac{N}{2}-1} f_{2m} \exp\left(-I \frac{2\pi}{N}(2m)k\right) + \sum_{m=0}^{\frac{N}{2}-1} f_{2m+1} \exp\left(-I \frac{2\pi}{N}(2m+1)k\right)$$

In the second term we may pull out the phase factor from the series

$$c_k = \underbrace{\sum_{m=0}^{\frac{N}{2}-1} f_{2m} \exp\left(-I \frac{2\pi}{N/2} mk\right)}_{p_k} + \underbrace{\exp\left(-I \frac{2\pi}{N} k\right)}_{\varphi_k} \underbrace{\sum_{m=0}^{\frac{N}{2}-1} f_{2m+1} \exp\left(-I \frac{2\pi}{N/2} mk\right)}_{q_k}$$

Using these abbreviations we get more compact formula

$$\boxed{c_k = p_k + \varphi_k q_k}$$

Now let's check what happens when the index k is shifted by N/2 $\boxed{k \quad \rightarrow \quad k + \frac{N}{2}}$

$$p_{k+\frac{N}{2}} = \sum_{m=0}^{\frac{N}{2}-1} f_{2m} \exp\left(-I\frac{2\pi}{\frac{N}{2}} m \left(k + \frac{N}{2}\right)\right) = \sum_{m=0}^{\frac{N}{2}-1} f_{2m} \exp\left(-I\frac{2\pi}{\frac{N}{2}} mk\right) \exp\left(-I\frac{2\pi}{\frac{N}{2}}\frac{N}{2}\right) = p_k$$

$$q_{k+\frac{N}{2}} = \sum_{m=0}^{\frac{N}{2}-1} f_{2m+1} \exp\left(-I\frac{2\pi}{\frac{N}{2}} m \left(k + \frac{N}{2}\right)\right) = \sum_{m=0}^{\frac{N}{2}-1} f_{2m+1} \exp\left(-I\frac{2\pi}{\frac{N}{2}} mk\right) \exp\left(-I\frac{2\pi}{\frac{N}{2}}\frac{N}{2}\right) = q_k$$

$$\varphi_{k+\frac{N}{2}} = \exp\left(-I\frac{2\pi}{N}\left(k\frac{N}{2}\right)\right) = \exp\left(-I\frac{2\pi}{N}k\right) \underbrace{\exp\left(-I\frac{2\pi}{N}\frac{N}{2}\right)}_{-1} = -\varphi_k$$

$$\boxed{\begin{aligned} p_{k+\frac{N}{2}} &= p_k \\ q_{k+\frac{N}{2}} &= q_k \\ \varphi_{k+\frac{N}{2}} &= -\varphi_k \end{aligned}}$$

- shifting of k by half of number of data gives at once the coefficents forr the second half without computing DFT, hence the number of computations is reduced by 2
- in next step, each sum is again divided into even and odd parity elements which can be processed separately, one more time we reduce the number of operations by 2
- this consequtive reduction of number of data taken into partial DFT stops when there is left only two elements
- How to assemble all these partial DFTs?
  The best way is to follow example for small number of data, i.e. N=8

$$\boxed{\begin{aligned} p_{k+\frac{N}{2}} &= p_k \\ q_{k+\frac{N}{2}} &= q_k \\ \varphi_{k+\frac{N}{2}} &= -\varphi_k \end{aligned}}$$

Example: factorization of 8-element data vector for DFT: top-to-bottom process

top-bottom process

$$N = 2^3, \qquad \{f_0, f_1, f_2, \ldots, f_7\}$$

1-st step

$$c_8^{(k)}(f_0, f_1, \ldots, f_7) = c_4^{(k)}(f_0, f_2, f_4, f_6) + \boldsymbol{w_8^k} \cdot c_4^{(k)}(f_1, f_3, f_5, f_7)$$

2-st step

$$c_4^{(k)}(f_0, f_2, f_4, f_6) = c_2^{(k)}(f_0, f_4) + \boldsymbol{w_4^k} \cdot c_2^{(k)}(f_2, f_6)$$

$$c_4^{(k)}(f_1, f_3, f_5, f_7) = c_2^{(k)}(f_1, f_5) + \boldsymbol{w_4^k} \cdot c_2^{(k)}(f_3, f_7)$$

3-st step

$$c_2^{(k)}(f_0, f_4) = f_0 + \boldsymbol{w_2^k} \cdot f_4$$

$$c_2^{(k)}(f_2, f_6) = f_2 + \boldsymbol{w_2^k} \cdot f_6$$

$$c_2^{(k)}(f_1, f_5) = f_1 + \boldsymbol{w_2^k} \cdot f_5$$

$$c_2^{(k)}(f_3, f_7) = f_3 + \boldsymbol{w_2^k} \cdot f_7$$

notice how the data are gathered in pairs at the lowest level, **we must set elements in this order** to start the next bottom-up process

$$\{f_0, f_4, \;\; f_0, f_6, \;\; f_1, f_5, \;\; f_3, f_7\}$$

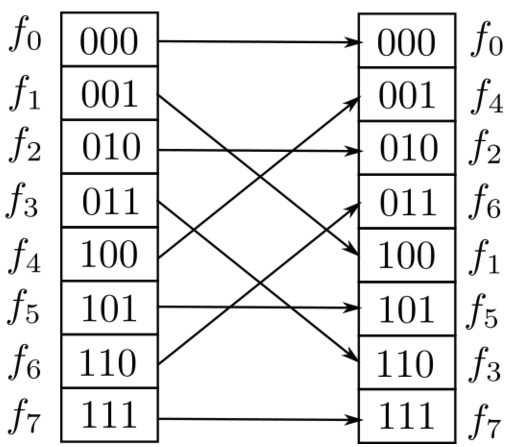
each division is made with twiddle factor for the half of data of the previous upper step

botom-up process to assemble all coefficients (next two pages)

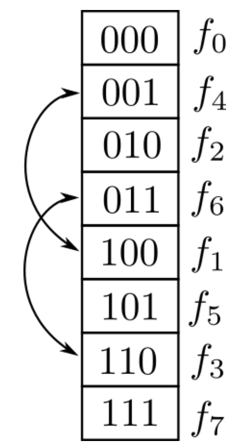
Bit-reversal ordering

Before we proceed calculations of coefficients c_k further we must reorder the elements in the input vector.
It is an easy task if we utilize the bit-arithmetic operation: bit-reversal

out of place bit-reversal
needs two arrays

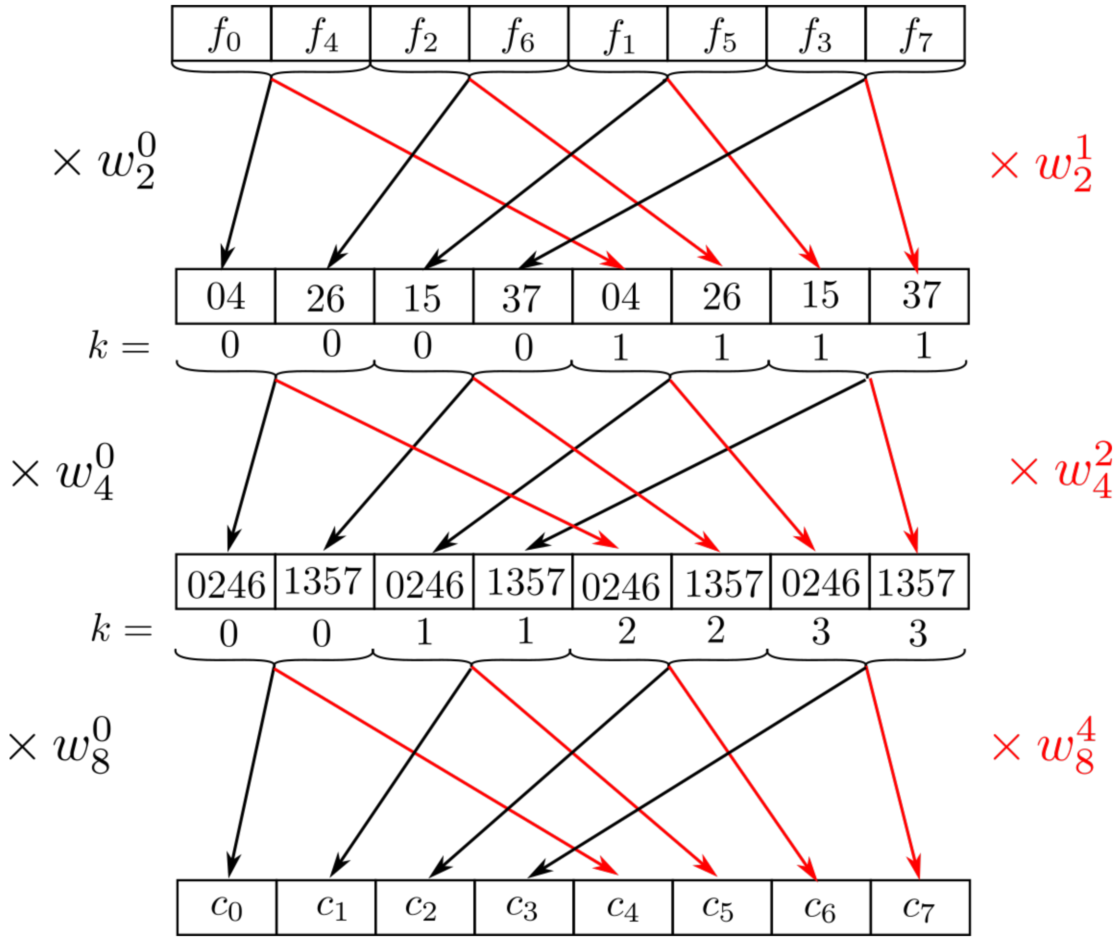


in-place bit-reversal
we only operate on input data array



Remark: positions are exchanged only once,
between elements of one pair,
it's enough to scan half of the array

Computing the DFT coefficients in fast way for 8-element array → FFT



How much operations we do?

- each step requires N multiplications and N addittion
- we make p steps $N=2^p$

$$m = N \cdot p = N\, log_2 N$$

Remarks:

- algorithm Radix-2 works for the number of data equal 2^p, if we short of some data e.g. we have 2^p-k then we may add these k empty elements (values=0) and performed FFT

- the poblem occurs when k is comparible with 2^p , e.g. we have N=1025 and need empty 1023 array cells, in such case is better to use another FFT algorithm like e.g. **PFA**

- **PFA** – **P**rime **F**actor **A**lgorithm, is based on factorization of N into a product of prime numbers, for each prime number it calculates the DFT which are then assembled into coeffcients

- there are other efficient algorithm which exploit computer architecure like split-Radix: Radix-4, Radix-8, transformations are then conducted on a bundle of data that can be simultaneously processed with vectorized instructions

- if it is needed the real-valued function to be transform we may use Discrete Sine Transfrom or Discrete Cosine Transfrom, these trigonometric functions have the same periodicity properties as complex twiddle factors, but we operate on the real values not the complex once so the number of arithmetic operations is reduced at least twice

Inverse FFT

choose $l \in \{0, 1, 2, \ldots, N-1\}$

$$c_k = \frac{1}{N} \sum_{j=0}^{N-1} f_j e^{-i\frac{2\pi}{N} \cdot k\, j} \qquad \sum_{k=0}^{N-1} e^{i\frac{2\pi}{N} l\, k} \cdot \;/$$

$$\sum_{k=0}^{N-1} e^{i\frac{2\pi}{N} l\, k} c_k = \frac{1}{N} \sum_{j=0}^{N-1} f_j \sum_{k=0}^{N-1} e^{-i\frac{2\pi}{N} \cdot k\,(j-l)}$$

$$\sum_{k=0}^{N-1} \left[e^{-i\frac{2\pi}{N}(j-l)} \right]^k = \sum_{k=0}^{N-1} \lambda^k = \begin{cases} \frac{\lambda^N - 1}{\lambda - 1} = \frac{e^{-i\,2\pi\,(j-l)} - 1}{e^{-i\frac{2\pi}{N}(j-l)} - 1} = 0 & \iff \quad l \neq j \\ N & \iff \quad l = j \end{cases}$$

$$\sum_{k=0}^{N-1} \left[e^{-i\frac{2\pi}{N}(j-l)} \right]^k = \sum_{k=0}^{N-1} \lambda^k = N \delta_{j,l}$$

$$\sum_{k=0}^{N-1} e^{i\frac{2\pi}{N} l\, k} c_k = \frac{1}{N} \sum_{j=0}^{N-1} f_j \sum_{k=0}^{N-1} e^{-i\frac{2\pi}{N} \cdot k\,(j-l)} = \frac{1}{N} \sum_{j=0}^{N-1} f_j N \delta_{j,l} = f_l$$

- to perform the inverse transformation we need to sum the coefficents with conjugated twiddle factors
- **note that the normalization factor now equals 1**

$$\boxed{f_l = \sum_{k=0}^{N-1} c_k e^{i\frac{2\pi}{N} l\, k}}$$

Multivariate FFT

Another great advantage of Fourier transform directly results from its definition, it is a linear transformation. Thanks to this property the multidimensional FFT can be processed for each dimension separately from the other. For the d-dimensional problem the basic DFT equation has the following form

$$N = N_1 N_2 \dots N_d$$

$$c_{k_1,k_2,\dots,k_d} = \frac{1}{N} \sum_{j_1=0}^{N_1} \sum_{j_2=0}^{N_2} \dots \sum_{j_d=0}^{N_d} f_{j_1,j_2,\dots,j_d} \exp\left(-i\,2\pi\left(\frac{j_1k_1}{N_1} + \frac{j_2k_2}{N_2} + \dots + \frac{j_dk_d}{N_d}\right)\right)$$

The data for each dimension can have different number of elements, and the FFT can be performed with different FFT algorithms.

Applications of FFT

- processing of digital signals – analysis of:
    - frequency/power spectra,
    - correlation, autocorrelation
    - convolution, deconvolution
    - digital filtering, noise removal
- data compression: the MP3 format is based on Modified Discrete Cosine Transform
- in Physics:
    - solving partial differential equations, e.g. the Poisson equation (Sine/Cosine transform) e.g. Fast Poisson Solver in Math Kernel Library (Intel)
    - calculations of Coulomb integrals for many-body quantum problems

Example (simple): filtering a noised signal

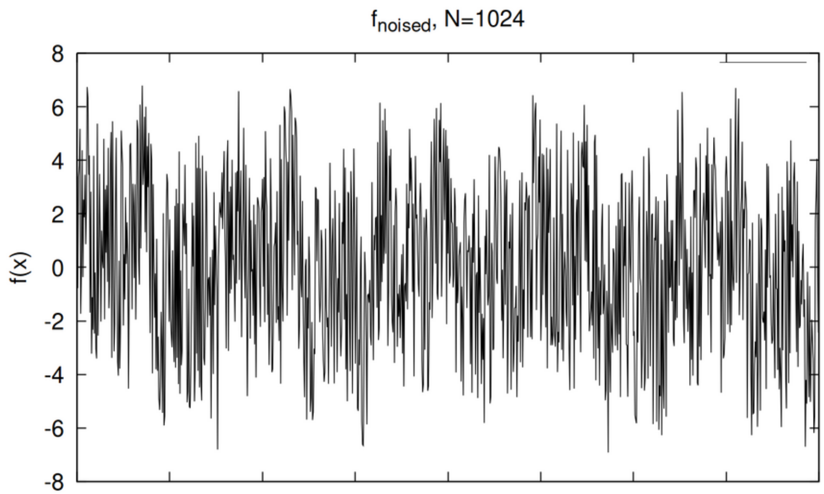
$$f_{pure}(x) = \sin(5\,\omega\,t) + \sin(15\,\omega\,t) + \sin(25\,\omega\,t)$$

$$f(x) = f_{pure}(x) + r \qquad\qquad r \sim U[-5,5]$$

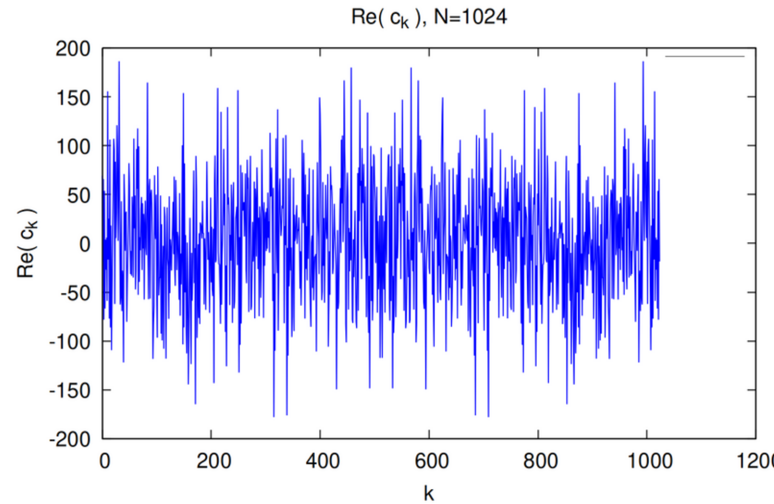
$$N = 2^{10} = 1024 \qquad \Delta_x = \frac{T}{N}$$

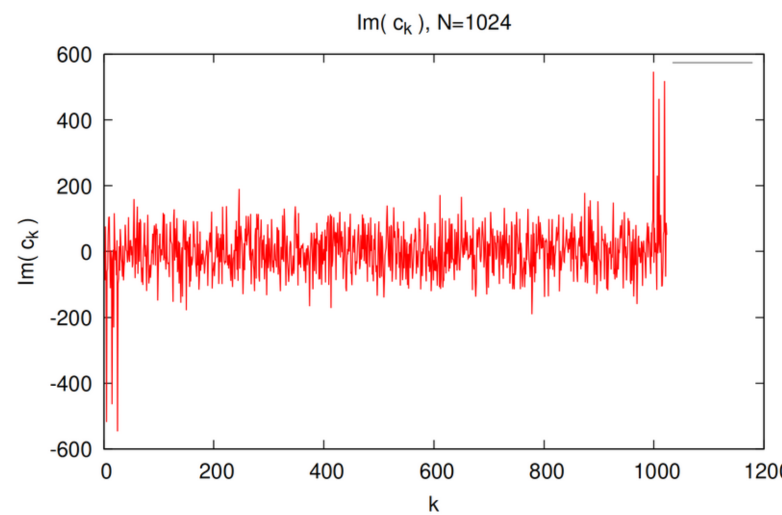
$$T = 4 \qquad \Delta_k = \frac{2\pi}{N}$$

$$F(k) = FFT\{f(x)\}, \qquad |F(k)|^2 < \frac{1}{2}\max|F(k)|^2 \implies F(k) = 0$$

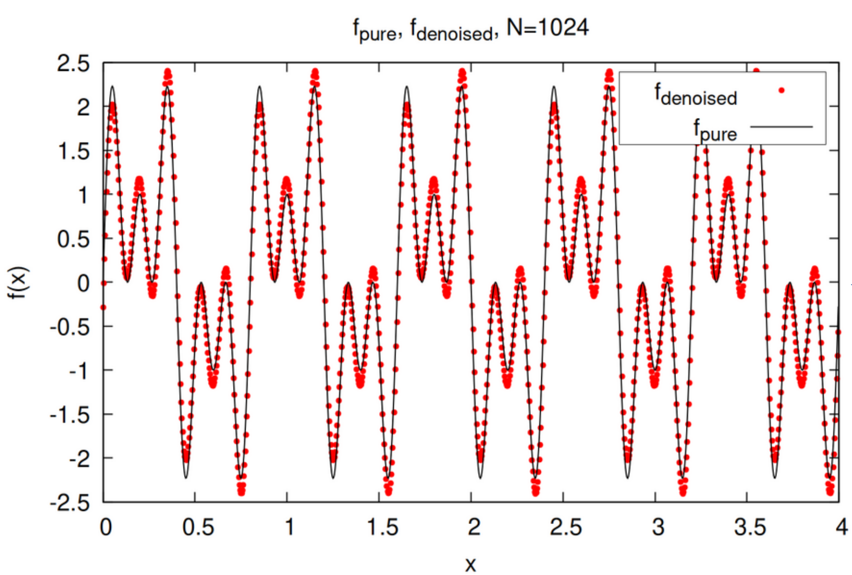


FFT





denoise
+
FFT^{-1}



Example (advanced): solving Poisson equation in 2D/3D with FFT solver

The Poisson equation

$$\nabla^2 V_r = \rho_r$$

is Fourier transformed to reciprocal space (wave vector k)

$$\boxed{\begin{aligned} FFT\{V(\vec{r})\} &= V(\vec{k}) \\ FFT\{\rho(\vec{r})\} &= \rho(\vec{k}) \\ FFT\{\nabla^2\} &= \vec{k}^2 = k^2 \end{aligned}}$$

$$k^2 V_k = \rho_k$$

Math Kernel Library (Intel) contains a routine named

Fast Poisson Solver

which use FFT for solving Poisson equation on 2D/3D mesh of nodes for Dirichlet and Neumann boundary conditions

after dividing both sides by k^2 we only need to perform the inverse transformation to get the solution

$$V_r = FFT^{-1}\{V_k\}$$

To solve partial differential equation we need to specify the boundary conditions, for Poisson equation we define two types of them

- for Dirichlet boundary conditions we use **discrete sine transform**

$$V_r|_{boundary} = V_b \neq 0$$

- for Neumann boundary condition we use **discrete cosine transform**

$$\frac{\partial V_r}{\partial \vec{n}}|_{boundary} = 0$$

Example (advanced): using convolution theorem to calculate integrals

In some quantum physics problems we need to calculate an electrostatic interaction between two charge densities (Coulomb integrals – 2 particles x 3 position variables = 6D problem)

$$C = \int_\Omega d\vec{r}_1 \int_\Omega d\vec{r}_2 \frac{\rho_1(\vec{r}_1)\rho_2(\vec{r}_2)}{|\vec{r}_1 - \vec{r}_2|} \quad \Rightarrow \quad f(\vec{r}_1 - \vec{r}_2) = \frac{1}{\vec{r}_1 - \vec{r}_2}$$

We may rewrite this integral to more palatable form, which can be fast integrated numerically with standard methods (it is reduced 3D problem)

$$C = \int_\Omega d\vec{r}_1 \rho_1(\vec{r}_1) V(\vec{r}_1) \quad \Leftarrow \quad V(\vec{r}_1) = \int_\Omega d\vec{r}_2 \rho_2(\vec{r}_2) f(\vec{r}_1 - \vec{r}_2)$$

before we make an integration we must calculate $V(r_1)$, this is done utilizing **convolution theorem**

$$h : \mathbb{R} \to \mathbb{R}, \quad g : \mathbb{R} \to \mathbb{R} \qquad \Longrightarrow \qquad (g * h)(t) \equiv \int_{-\infty}^{\infty} g(\tau) h(t - \tau) d\tau$$

$$FFT\{g(t)\} = G(\omega), \quad FFT\{h(t)\} = H(\omega) \qquad \Longrightarrow \qquad FFT\{h * g\} = G(\omega)\, H(\omega)$$

$$h * g = FFT^{-1}\{G(\omega) H(\omega)\}$$

finally we get simple recipe to calculate the needed function $V(r_1)$

$$V = \rho_2 * f = FFT^{-1}\{FFT\{\rho_2\} \cdot FFT\{f\}\}$$