Mathematics III

Prof. Dr. Michael Möckel
WS 2022/23

TH Aschaffenburg
university of applied sciences

Fast Fourier Transformation (FFT)

Fast Fourier Transform
Foundation of our modern civilization

History of FFT algorithm:

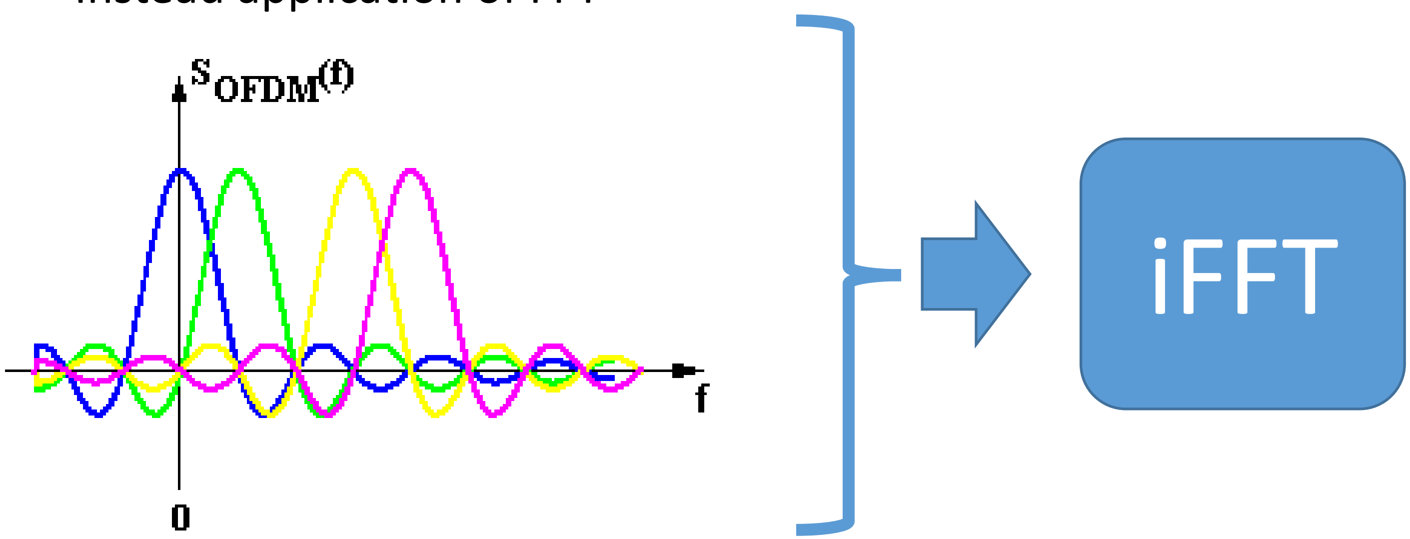
- Gauss (1805): calculation of astronomical predictions
- Cooley, Tukey (1965, IBM): efficient implementation on modern computers

Many applications of the Fast Fourier Transform:

- Digital signal processing: Feature selection in the frequency domain
- Reconstruction of images from nuclear magnetic resonance (NMR) tomography
- Digital compression algorithms (MP3 Standard, save jpeg)
- Data transmission (telecommunication)
- Basically all modern standards use FFT (OFCD for WLAN, DVBT, LTE (mobil))

Orthogonal Frequency-Division Multiplexing (OFCD)

- Decomposition of signal
- Simultaneous transmission of Fourier coefficients on multiple carrier frequencies (7000 frequencies, each carrying 1,2,4 or 8 bit (DVBT2))
- Reconstruction of the original signal:
  - use of 7000 frequency filters not possible!
  - Instead application of FFT



This would not work with DFT as done in lecture

Numerical complexity of Discrete Fourier Series (DFT):

O(N^2) multiplications:

- We need n coefficients A and n coefficients B, in total 2n coefficients
- Each coefficient is calculated by multiplying 2n function values with sin/cos
- In total this is N * N = 4^n^2 multiplications (+ some additions, less costly)

Numerical complexity of Fast Fourier Transform (FFT):

- O(N log(N))
- Much faster!!! Bsp N = 1e6, N^2 = 1e12,
  N log(N) = 13e6,
  i.e. about 100,000 times smaller!

<u>Example:</u>
Audio signal
sampled at 44.1 kHz (CD)
10 sec 4.4e5 values
20 sec 1.0e6 values

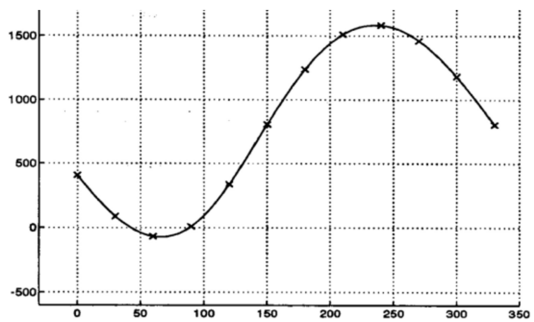
Gauss: calculation of orbit of Pallas

- 12 measurement points of a periodic orbit (two angles)

θ	0	30	60	90	120	150	180	210	240	270	300	330
X	408	89	−66	10	338	807	1238	1511	1583	1462	1183	804

- Numerical complexity: N^2 = 144 vs N log(N) = 30 operations
- Expectation for the motion, but 12 parameters had to be determined

$$X = f(\theta) = a_0 + \sum_{k=1}^{5} \left[a_k \cos\left(\frac{2\pi k\theta}{360}\right) + b_k \sin\left(\frac{2\pi k\theta}{360}\right)\right] + a_6 \cos\left(\frac{12\pi\theta}{360}\right)$$



<u>Modern solution:</u>

k	0	1	2	3	4	5	6
a_k	780.6	−411.0	43.4	−4.3	−1.1	0.3	0.1
b_k	–	−720.2	−2.2	5.5	−1.0	−0.3	–

Prof. Osgood, Lecture Notes

Complex representation of Discrete Fourier Transform

$$\mathbf{F}[m] = \sum_{k=0}^{N-1} \mathbf{f}[k]\omega^{-km} = \sum_{k=0}^{N-1} \mathbf{f}[k]e^{-2\pi i km/N}$$

Vector of Fourier coefficients

$$\omega = e^{2\pi i/N}$$

Fourier basis functions

$$\underline{\mathcal{F}} = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega^{-1} & \omega^{-2} & \cdots & \omega^{-(N-1)} \\ 1 & \omega^{-2} & \omega^{-4} & \cdots & \omega^{-2(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{-(N-1)} & \omega^{-2(N-1)} & \cdots & \omega^{-(N-1)^2} \end{pmatrix}$$

Discrete Fourier Transform As matrix multipl.

Inverse Discrete Fourier Transform

- Inverse DFT = re-composition of original function from Fourier coefficients

$$\frac{1}{N}\begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega^1 & \omega^2 & \cdots & \omega^{(N-1)} \\ 1 & \omega^2 & \omega^4 & \cdots & \omega^{2(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{(N-1)} & \omega^{2(N-1)} & \cdots & \omega^{(N-1)^2} \end{pmatrix} \begin{pmatrix} F[0] \\ F[1] \\ F[2] \\ \vdots \\ F[N-1] \end{pmatrix} = \begin{pmatrix} f[0] \\ f[1] \\ f[2] \\ \vdots \\ f[N-1] \end{pmatrix}$$

Inverse DFT: sign of exponent, prefactor differ!

- Compact representation of DFT and iDFT as matrices is very clear!

$$(\underline{\mathcal{F}})_{mn} = \omega^{-mn}, \quad m, n = 0, 1, \ldots, N-1$$

Fast Fourier Transform Algorithm (FFT)

- Example: N=4. $\omega_4 = e^{2\pi i/4}$

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & \omega_4^{-1} & \omega_4^{-2} & \omega_4^{-3} \\ 1 & \omega_4^{-2} & \omega_4^{-4} & \omega_4^{-6} \\ 1 & \omega_4^{-3} & \omega_4^{-6} & \omega_4^{-9} \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & \omega_4^{-1} & -1 & -\omega_4^{-1} \\ 1 & -1 & 1 & -1 \\ 1 & -\omega_4^{-1} & -1 & \omega_4^{-1} \end{pmatrix}$$

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & \omega_4^{-1} & -1 & -\omega_4^{-1} \\ 1 & -1 & 1 & -1 \\ 1 & -\omega_4^{-1} & -1 & \omega_4^{-1} \end{pmatrix} \begin{pmatrix} \mathbf{f}[0] \\ \mathbf{f}[1] \\ \mathbf{f}[2] \\ \mathbf{f}[3] \end{pmatrix} = \begin{pmatrix} \mathbf{f}[0] + \mathbf{f}[1] + \mathbf{f}[2] + \mathbf{f}[3] \\ \mathbf{f}[0] + \mathbf{f}[1]\omega_4^{-1} - \mathbf{f}[2] - \mathbf{f}[3]\omega_4^{-1} \\ \mathbf{f}[0] - \mathbf{f}[1] + \mathbf{f}[2] - \mathbf{f}[3] \\ \mathbf{f}[0] - \mathbf{f}[1]\omega_4^{-1} - \mathbf{f}[2] + \mathbf{f}[3]\omega_4^{-1} \end{pmatrix}$$

FFT – clever sorting!

- Trick: Smart arrangement of products and re-use of intermediate results
- Sort even and odd terms on each step

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & \omega_4^{-1} & -1 & -\omega_4^{-1} \\ 1 & -1 & 1 & -1 \\ 1 & -\omega_4^{-1} & -1 & \omega_4^{-1} \end{pmatrix} \begin{pmatrix} \mathbf{f}[0] \\ \mathbf{f}[1] \\ \mathbf{f}[2] \\ \mathbf{f}[3] \end{pmatrix} = \begin{pmatrix} \mathbf{f}[0] + \mathbf{f}[1] + \mathbf{f}[2] + \mathbf{f}[3] \\ \mathbf{f}[0] + \mathbf{f}[1]\omega_4^{-1} - \mathbf{f}[2] - \mathbf{f}[3]\omega_4^{-1} \\ \mathbf{f}[0] - \mathbf{f}[1] + \mathbf{f}[2] - \mathbf{f}[3] \\ \mathbf{f}[0] - \mathbf{f}[1]\omega_4^{-1} - \mathbf{f}[2] + \mathbf{f}[3]\omega_4^{-1} \end{pmatrix}$$

- Equivalent representation, but new interpretation as a simpler DFT

$$\begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & \omega_4^{-1} \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -\omega_4^{-1} \end{pmatrix} \begin{pmatrix} \mathbf{f}[0] + \mathbf{f}[2] \\ \mathbf{f}[0] + \mathbf{f}[2]\omega_2^{-1} \\ \mathbf{f}[1] + \mathbf{f}[3] \\ \mathbf{f}[1] + \mathbf{f}[3]\omega_2^{-1} \end{pmatrix} = \begin{pmatrix} \mathbf{f}[0] + \mathbf{f}[2] + \mathbf{f}[1] + \mathbf{f}[3] \\ \mathbf{f}[0] + \mathbf{f}[2]\omega_2^{-1} + \mathbf{f}[1]\omega_4^{-1} + \mathbf{f}[3]\omega_4^{-1}\omega_2^{-1} \\ \mathbf{f}[0] + \mathbf{f}[2] - \mathbf{f}[1] - \mathbf{f}[3] \\ \mathbf{f}[0] + \mathbf{f}[2]\omega_2^{-1} - \mathbf{f}[1]\omega_4^{-1} - \mathbf{f}[3]\omega_4^{-1}\omega_2^{-1} \end{pmatrix}$$

DFT with N‘ = N/2 for even and odd points!

Decomposition into DFTs of smaller N

- Block structure:

$$\begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & \omega_4^{-1} \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -\omega_4^{-1} \end{pmatrix} \begin{pmatrix} \mathbf{f}[0] + \mathbf{f}[2] \\ \mathbf{f}[0] + \mathbf{f}[2]\omega_2^{-1} \\ \mathbf{f}[1] + \mathbf{f}[3] \\ \mathbf{f}[1] + \mathbf{f}[3]\omega_2^{-1} \end{pmatrix} = \begin{pmatrix} \mathbf{f}[0] + \mathbf{f}[2] + \mathbf{f}[1] + \mathbf{f}[3] \\ \mathbf{f}[0] + \mathbf{f}[2]\omega_2^{-1} + \mathbf{f}[1]\omega_4^{-1} + \mathbf{f}[3]\omega_4^{-1}\omega_2^{-1} \\ \mathbf{f}[0] + \mathbf{f}[2] - \mathbf{f}[1] - \mathbf{f}[3] \\ \mathbf{f}[0] + \mathbf{f}[2]\omega_2^{-1} - \mathbf{f}[1]\omega_4^{-1} - \mathbf{f}[3]\omega_4^{-1}\omega_2^{-1} \end{pmatrix}$$

- Each block of the second matrix represents a DFT with N' = N/2 (here: 4/2=2)
- Such a DFT is simpler to calculate
- The first matrix represents a mixing term

Even-odd Order by Bit Inversion of Index

f[0]	f[0]	f[0]	f[0]
f[1]	f[2]	f[4]	f[4]
f[2]	f[4]	f[2]	f[2]
f[3]	f[6]	f[6]	f[6]
f[4]	f[1]	f[1]	f[1]
f[5]	f[3]	f[5]	f[5]
f[6]	f[5]	f[3]	f[3]
f[7]	f[7]	f[7]	f[7]

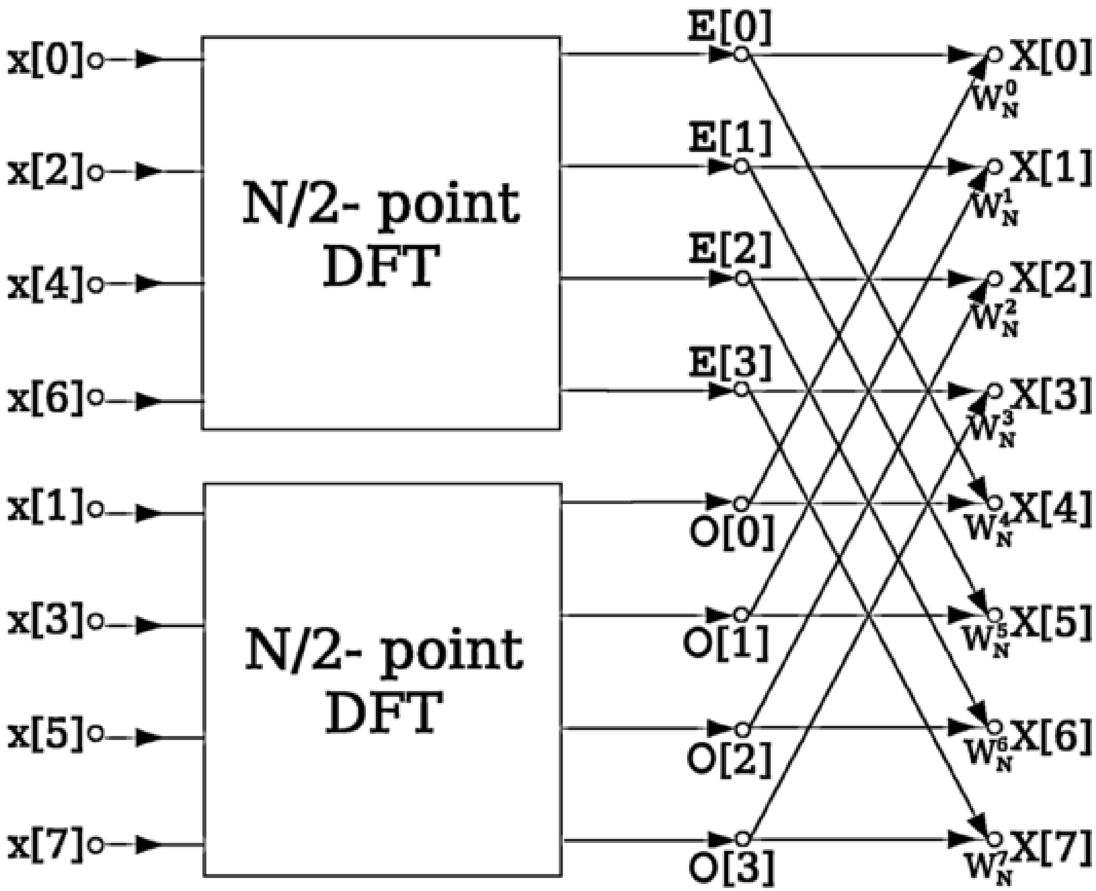
f[0]	000
f[1]	001
f[2]	010
f[3]	011
f[4]	100
f[5]	101
f[6]	110
f[7]	111

→

f[0]	000
f[4]	100
f[2]	010
f[6]	110
f[1]	001
f[5]	101
f[3]	011
f[7]	111

New order can be reached by simple bit inversion!

Combination of Even and Odd Contributions (Radix 2 algorithm)

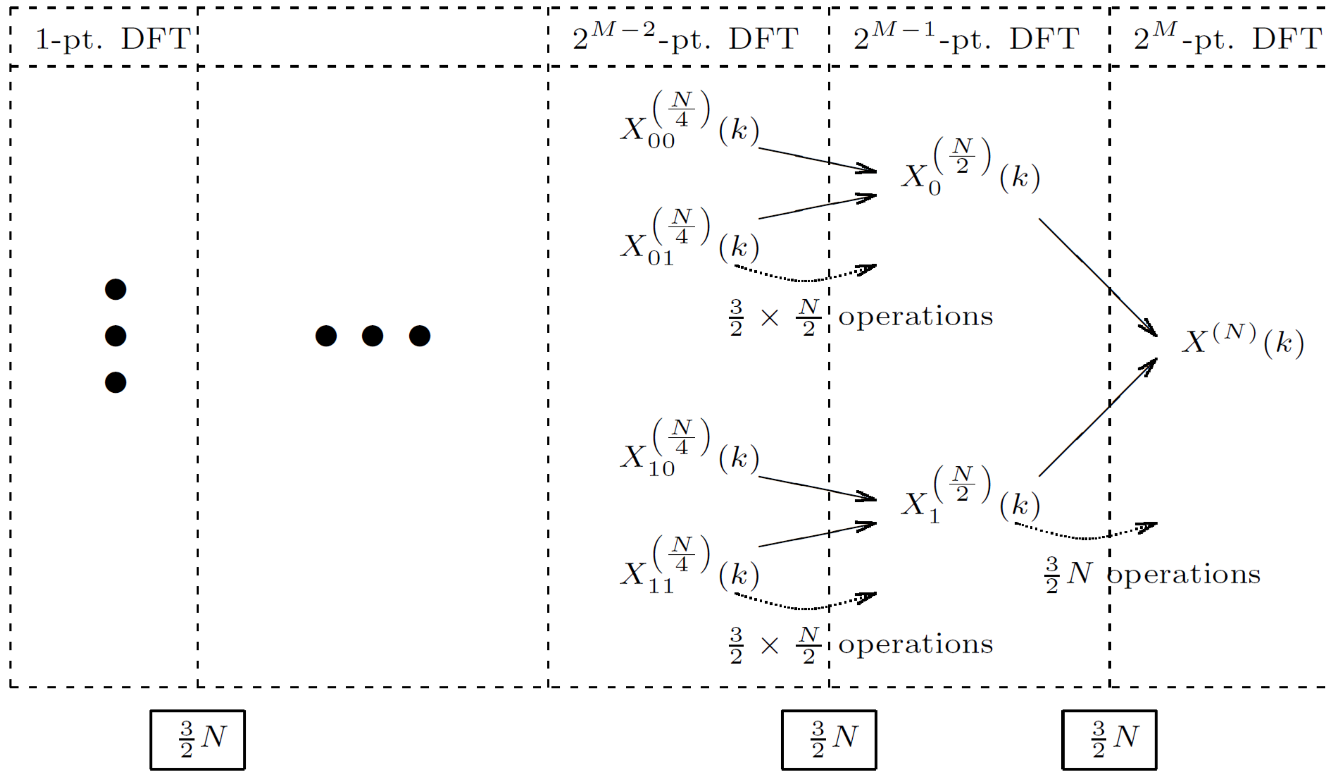


$$X[k] = \sum_{r=0}^{N/2-1} x(2r)\omega_{(N/2)}^{kr} + \omega_{(N)}^{k} \sum_{r=0}^{N/2-1} x(2r+1)\omega_{(N/2)}^{kr}$$

Wikiversity, FFT

Iterative Procedure

- In every step N coefficients need to be computed using N complex multiplications and N additions
- Because of
  $\omega_N^{k+N/2} = -W_N^k$
  half of them are the same up to a sign
- i.e. **in every step** only **N/2** complex multiplications are needed



If **N = 2^M** the FFT is particularly efficient.
Then **M = ln(N)/ln(2)** iterations are needed

Numerical complexity
$N \ln(N)$