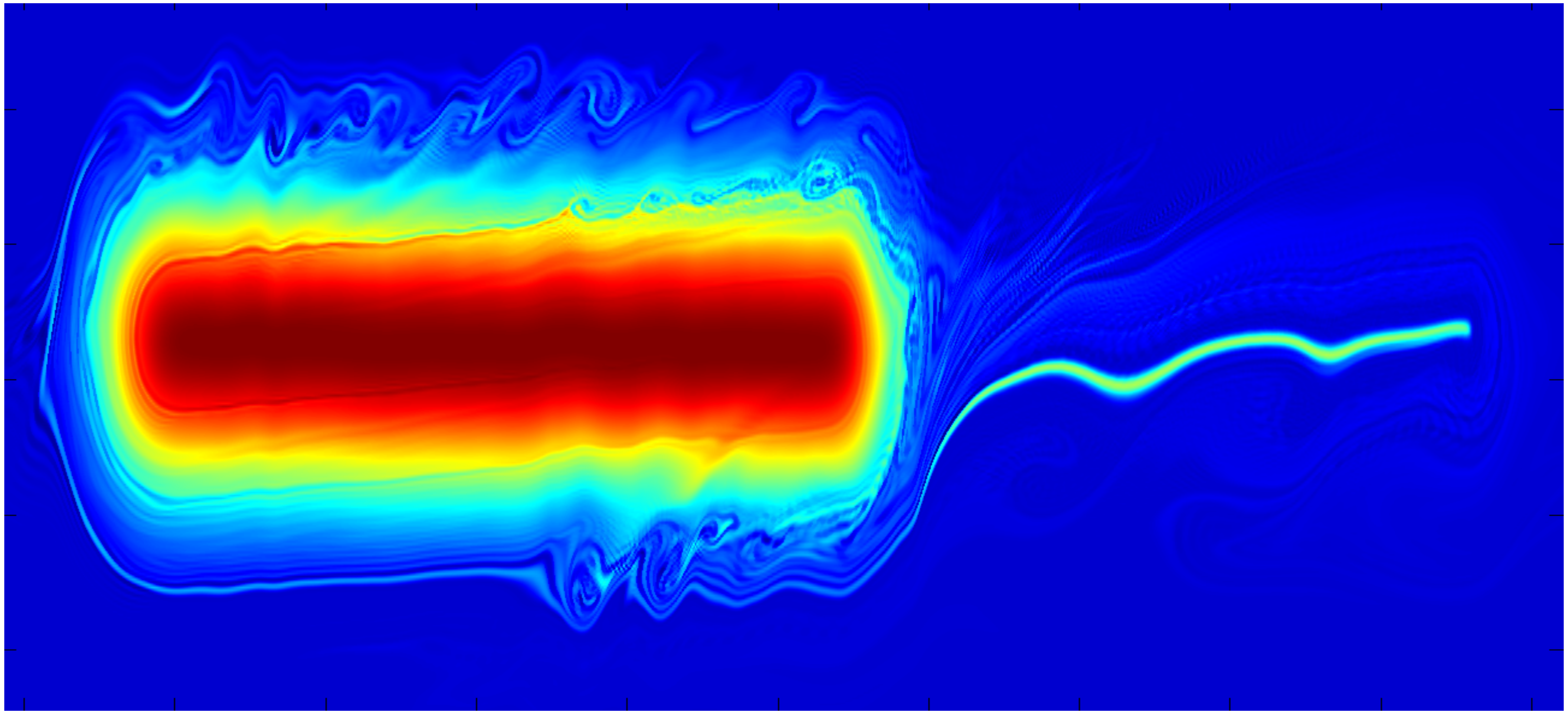
HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF



Chapter 7 - Fast Fourier Transformation

DFT in a nutshell

- Standard FT (without normalization):
$$f(k) = \int_{-\infty}^{\infty} e^{-ikx} f(x) dx$$

- FT of periodic functions of period *L*:
(In numerical representation we are limitied to an interval, *f* has to be quasi-periodic)
$$f(k_n) = \int_0^L e^{-ik_n x} f(x) dx$$

- Sampling of *f* on a (discrete) grid yields:
$$f(k_n) = \Delta x \sum_{j=0}^{N-1} e^{-ik_n j \Delta x} f(x_j)$$

- *N* grid points to cover interval of length *L* results in two limits:
  - Due to the finite length L there is a lowest wave-number $k_{\mathrm{min}} = \Delta k = 2\pi/L$
  - *Sampling Theorem*: To uniquely identify an oscillation we need two sampling points per period, i.e. $\lambda_{min}=2\Delta x$, thus $k_{max} = \pi/\Delta x$

DFT in a nutshell

$$f(k_n) = \Delta x \sum_{j=0}^{N-1} e^{-ik_n j\Delta x} f(x_j) \longrightarrow f(k_n) = \Delta x \sum_{j=0}^{N-1} e^{-i\frac{2\pi}{N\Delta x}nj\Delta x} f_j$$

$$k_n = \frac{2\pi}{N\Delta x}n, \quad n = 0, \ldots, N-1$$

- Forward DFT: $f(k_n) = \Delta x \sum_{j=0}^{N-1} e^{-2\pi inj/N} f_j$
- This is nothing but a projection of *f(x_i)* on a new basis:

$$\hat{\mathbf{f}} = \begin{pmatrix} 1 & 1 & 1 & \ldots & 1 \\ 1 & \omega & \omega^2 & \ldots & \omega^{N-1} \\ 1 & \omega^2 & \omega^4 & \ldots & \omega^{2(N-1)} \\ \vdots & & & & \\ 1 & \omega^{N-1} & \omega^{2(N-1)} & \ldots & \omega^{(N-1)^2} \end{pmatrix} \begin{pmatrix} f(x_0) \\ f(x_1) \\ f(x_2) \\ \vdots \\ f(x_{N-1}) \end{pmatrix} \qquad \omega = e^{-2\pi i/N}$$

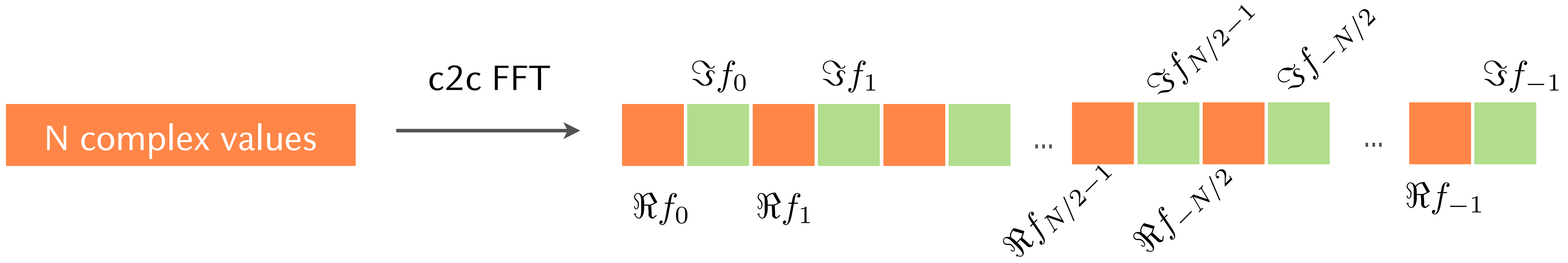
- Fast Fourier Transformation is a clever way to compute this matrix-vector multiplication in $\mathcal{O}(\,N\log(N)\,)$ instead of $\mathcal{O}(N^2)$

DFT in a nutshell

$$f(k_n) = \Delta x \sum_{j=0}^{N-1} e^{-2\pi i n j/N} f_j$$ is periodic with period N: $f(k_{N-n}) = f(k_{-n})$

- There are two ways in which we can identify the frequencies:
  - All frequencies are positive, ranging from $k_{min}=0$ to $k_{max}=k_{N-1}$
  - There are negative and positive frequencies, symmetric about 0: $k_{min} = k_{-N/2}$ to $k_{max} = k_{N/2-1}$

Frequency mapping



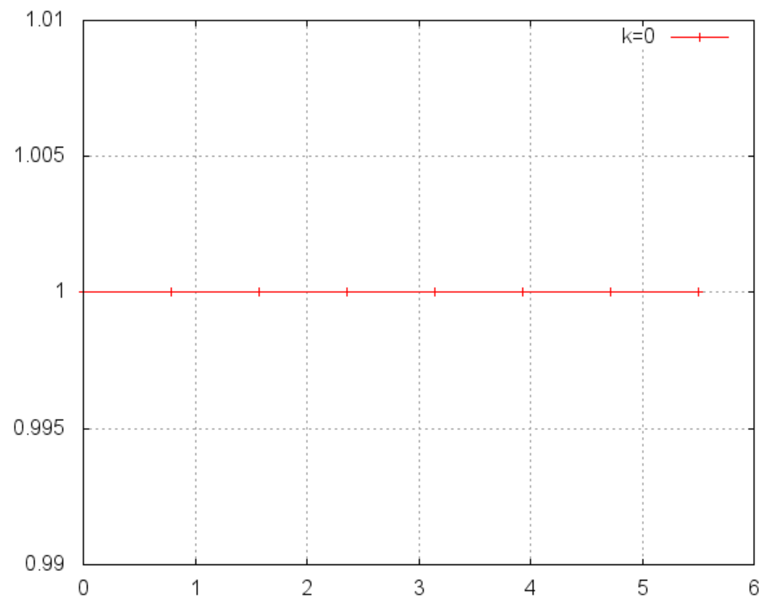
N complex values = 2N doubles

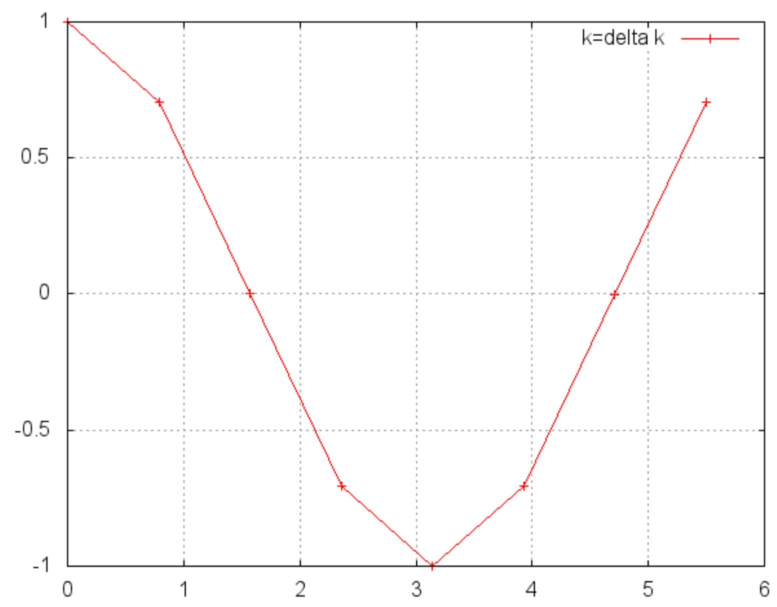
Example: N = 8

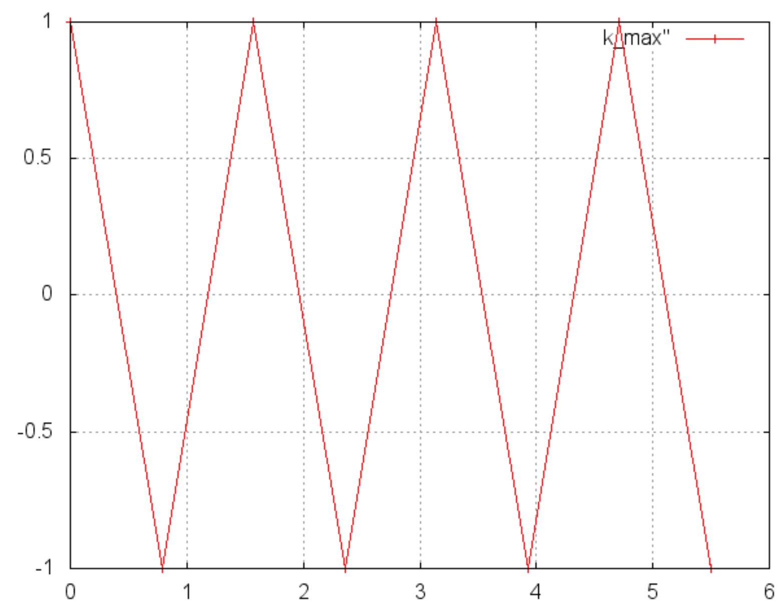
$$I = [0, 2\pi], \quad L = 2\pi, \quad N = 8, \quad \Delta k = \frac{2\pi}{L} = 1, \quad k_{max} = \frac{\pi}{\Delta x}$$

$$\Delta x = \frac{L}{N} = \frac{2\pi}{8} \longrightarrow \quad x_{max} = 7\Delta x = \frac{7}{8}2\pi \neq 2\pi$$

periodic grid: we do not store the redundant point at f(x=L)!







	Re	Im
f_0	8	0
	0	0
	0	0
	0	0
	0	0
	0	0
	0	0
	0	0

	Re	Im
	0	0
f_1	4	0
	0	0
	0	0
	0	0
	0	0
	0	0
f_{-1}	4	0

	Re	Im
	0	0
	0	0
	0	0
	0	0
$f_{-N/2}$	8	0
	0	0
	0	0
	0	0

Using FFTW to caclulate the DFT

- FFTW provides three interfaces:
  - Basic interface (we will use this one)
  - Advanced interface
  - Guru interface
- Results from FFTW are usually without normalization, i.e. one might need to normalize results by 1/N to obtain agreement with analytical results

C2C transformation with FFTW

```
#include <fftw3.h>
#include <cmath>
#include <iostream>
//--------------------
using namespace std;
//--------------------
int main()
{
     const int N=8;

    fftw_plan fw,fw2;

    fftw_complex* inC = (fftw_complex*) fftw_malloc(sizeof(fftw_complex) * N);
    fftw_complex* outC = (fftw_complex*) fftw_malloc(sizeof(fftw_complex) * N);
    const double L=2*M_PI;
    const double dx=L/N;
    double x;

    // inC = exp(ii*x)
    for(int i=0; i<N; i++){
        x=i*dx;
     inC[i][0]= cos(x); // Re
     inC[i][1]= sin(x); // Im
    }
    fw = fftw_plan_dft_1d(N, inC, outC, FFTW_FORWARD, FFTW_ESTIMATE);



    fftw_execute(fw);
    (...)
}
```