Physik auf dem Computer **SS 2017**

Worksheet 6: Discrete Fourier Transforms

May 29, 2017

General Remarks

- The deadline for handing in the worksheets is **Tuesday, June 6th, 2017, 10:00**.
- For this worksheet, you can achieve a maximum of 10 points.
- To hand in your solutions, send an email to your tutor:
  - Johannes Zeman `zeman@icp.uni-stuttgart.de` (Tue 15:45–17:15)
  - Michael Kuron `mkuron@icp.uni-stuttgart.de` (Wed 15:45–17:15)
  - Kai Szuttor `kai@icp.uni-stuttgart.de` (Thu 14:00–15:30)
- Please try to only hand in a single file that contains your program code for all tasks. If you are asked to answer questions, you should do so in a comment in your code file. If you are asked for graphs or figures, it is sufficient if your code generates them. You may as well hand in a separate PDF document with all your answers, graphs and equations.
- The worksheets are to be solved in groups of two or three people.

Task 6.1: Discrete Fourier Transform (6 points)

6.1.1 (4 points) Implement a Python function `dft_forw(xdata)` which computes the discrete Fourier transform of a data series `xdata`, and a Python function `dft_back(kdata)` for the back transformation. The results should be identical to the results of `numpy.fft.fft` and `numpy.fft.ifft`.

Hints

- In the Python functions, you can use the data type `numpy.complex`. The complex number $x = 1 + 2i$ can be written as `x=1.0+2.0j` in Python. All mathematical functions (in particular `numpy.exp()`) can also use such complex numbers.
- The Python script `ws6.py` defines a function `verify_fourier(forward, back)`, where `forward` and `back` are Python functions that do a Fourier forward and back transform, respectvely. `verify_fourier` checks whether the Fourier transforms work correctly by transforming and backtransforming a random data series and gives some hints on what might be wrong if they don't. This function should be useful while developing the transform.

6.1.2 (2 points) Show your success by forward- and back-transforming the Runge as well as the Lennard-Jones function (defined on the previous worksheet) sampled on 2048 equidistant points on the corresponding intervals given in the previous worksheet. Create plots with the data of the original functions, their Fourier transforms, and the back-transformed data.

Hints Be aware of the fact that Fourier transforms of real data are usually complex.

- A NumPy array containing `N` complex zeros can be created with `numpy.zeros(N, dtype=numpy.complex128)`.
- The real and imaginary parts of a complex NumPy array `A` can accessed via `A.real` and `A.imag`.

Task 6.2: Comparison with Fast Fourier Transforms (4 points)

6.2.1 (3 points) Using `timeit.timeit`, measure the run time of the forward and back Python DFT functions from task 6.1, and the forward and back NumPy FFT functions for $N \in \{4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096\}$. As input data for the DFT/FFT, you may use random numbers or the values of the Runge or Lennard-Jones functions from the previous worksheet. Create a double-logarithmic plot of the run times versus N.

Hints

- Pass the argument `number=10` to `timeit.timeit` in order to time 10 executions for each value of N. Note that you have to divide the measured runtimes by this number to obtain the runtime of a single execution!
- If the measurements take too long, either try to seed up your code by replacing as many Python loops as possible by NumPy operations (preferred solution!), reduce the number of executions per measurement, or conduct your measurements only up to $N = 2048$.

6.2.2 (1 point) What do you learn from this plot? In particular, how do the runtimes of the DFT and FFT transforms scale with N?