Physik auf dem Computer **SS 2013**

Worksheet 4: Fourier Transform

May 8, 2013

General Remarks

- Deadline is **Tuesday, 14th May 2013, 10:00**
- On this worksheet, you can achieve a maximum of 10 points.
- To hand in your solutions, send an email to
  - Olaf (`olenz@icp.uni-stuttgart.de`; Wednesday, 14:00–15:30)
  - Elena (`minina@icp.uni-stuttgart.de`; Wednesday, 15:45–17:15)
  - Tobias (`richter@icp.uni-stuttgart.de`; Friday, 15:45–17:15)
- Attach all required files to the mailing. If asked to write a program, attach the *source code* of the program. If asked for a text, send it as PDF or in the text format. We will *not* accept MS Word files!
- The worksheets are to be solved in groups of two or three people.
- The tutorials take place in the CIP-Pool of the ICP in Allmandring 3.

Task 4.1 (3 points): Fourier Transform in NumPy

In the following, we will use the periodic Runge function $f(x) = \frac{1}{1+\cos^2(x)}$ on the domain $[0, 2\pi]$ and the Lennard-Jones-Function $g(x) = x^{-12} - x^{-6}$ on the domain $[1, 5]$.

- 4.1.1 (2 points) Write a Python program that does the following for the functions:
  1. It creates a data series of 2048 equidistant points on the respective domains and their function values.
  2. It uses the Python function `numpy.fft.rfft()`, to compute the Fourier coefficients of the data series.
  3. It truncates the Fourier series after the first 5, 10 and 50 coefficients.
  4. It back transforms the truncated Fourier series to real space with help of the Python function `numpy.fft.irfft()`. The length of the output vector should be again 2048. The output vector is the *reconstructed function.*
  5. It creates a plot of the function and the different reconstructed functions.

- 4.1.3 (1 point) Why does the reconstruction of the function $g(x)$ show artefacts at the right boundary even when 50 Fourier coefficients are used, while this is not the case for function $f(x)$ even when only 10 Fourier coefficients are used?

Task 4.2 (3 points): Discrete Fourier Transform

- 4.2.1 (2 points) Implement a Python function `dft_forw()` that computes the discrete Fourier transform of a data series and a Python function `dft_back()` for the back transformation. The results should be identical to the results of `numpy.fft.fft` and `numpy.fft.ifft`.
- 4.2.2 (1 point) Implement two further Python functions `rdft_forw()` and `rdft_back()` that handle data series with purely real data and require only half of the coefficients. `rdft_back(a,n)` requires an optional second parameter `n` that you can use to define the length of the output vector.

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.
- Put the Python functions in this and later tasks into a separate Python file (*e.g.* `myfourier.py`). Then you can import this file from another Python program (`import myfourier`) and reuse the functions in later tasks.
- The Python functions from task 4.2.2 shall do the same as the Python functions `numpy.fft.rfft()` and `numpy.fft.irfft()`. In particular it should be possible to use these functions in the script of task 4.1.

Task 4.3 (2 points): Fast Fourier Transform

Implement the Python functions `fft_forw()` and `fft_back()`, that do the same as `dft_forw()` and `dft_back()` but use the Fast Fourier transform algorithm.

Task 4.4 (2 points): Timing the Fourier Transform

- 4.3.2 (1 point) Measure the run time of the Python DFT functions from task 4.2, the FFT functions from task 4.3 and the NumPy FFT functions for $N \in \{4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048\}$ via the Python function `timeit.timeit`. Create a loglog plot of the run times versus N.
- 4.3.3 (1 point) What do you learn from this plot?

Hint The tar-file `timing.tar.gz` on the lecture's home page contains two Python files that demonstrate how to measure the timings for the NumPy FFT.