Computational photonics – Excercise 1

– Julien Kluge (564513) –

Date: April 15, 2019

Task 1

Table 1: Program file index - Task 1

A1/A1.m	Matlab	program with whole task in sections

a)

```
gauss1D = @(x) exp(-(x * x));
x = -xMax:dx:xMax;
y = arrayfun(gauss1D, x);
plot(x, y)
```

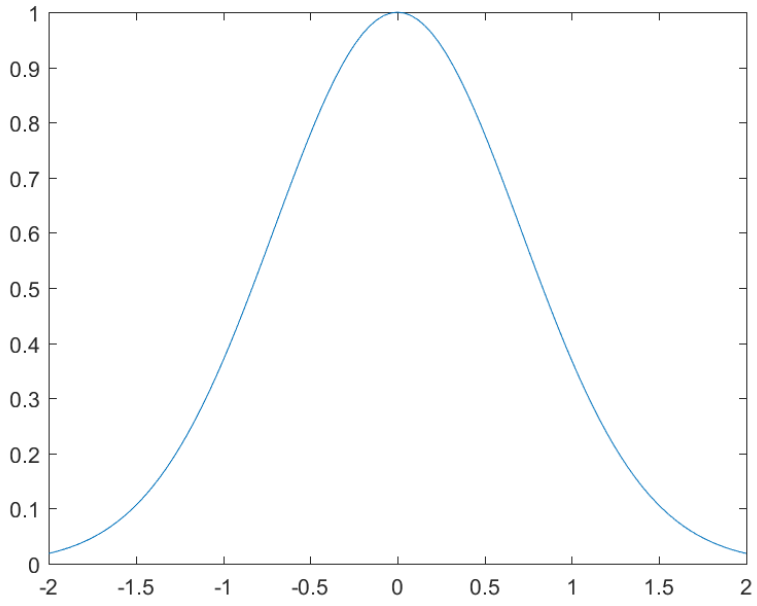


Figure 1: Plot of the function $f(x) = \exp\left(-x^2\right)$ in matlab.

b)

```
gauss2D = @(x, y) exp(-(x * x + y * y));
g = -gMax:dg:gMax;
[x, y] = meshgrid(g, g);
z = arrayfun(gauss2D, x, y);
mesh(x, y, z)
surface(x, y, z)
pcolor(x, y, z)
contour(x, y, z)
```

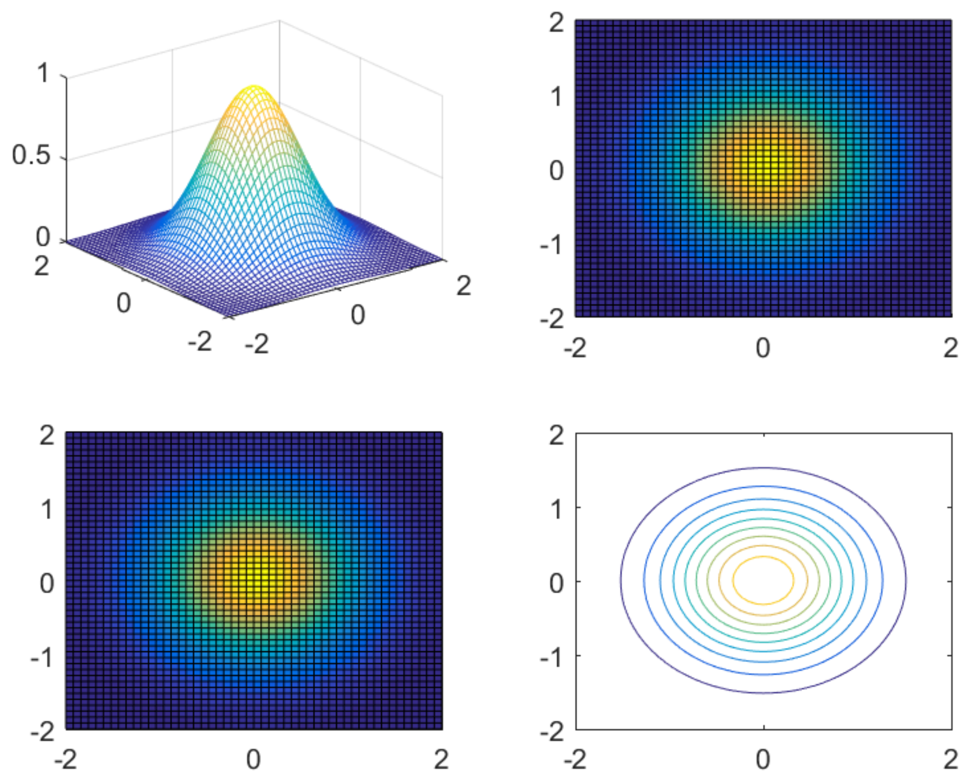


Figure 2: Plot of the function $f(x) = \exp\left(-\left(x^2 + y^2\right)\right)$ in matlab for different commands.

c)

Quiver code omitted for clarity in this document.

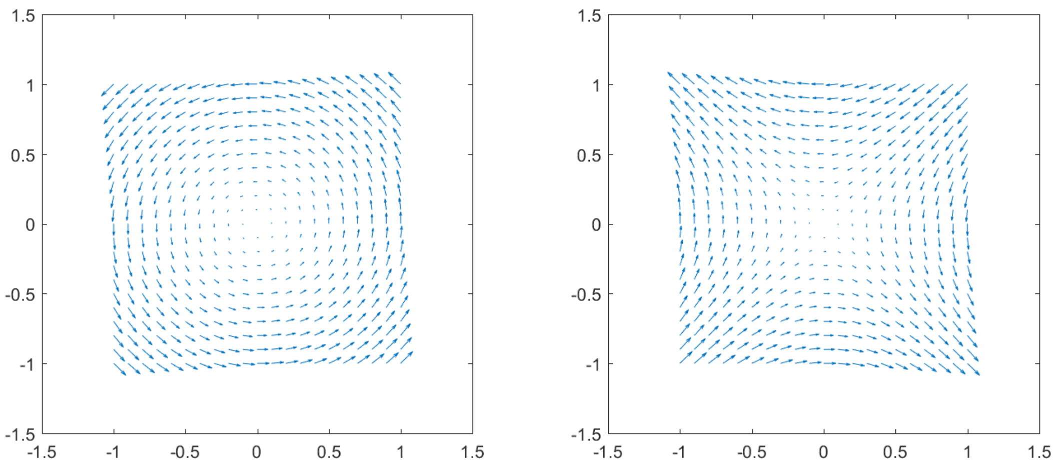


Figure 3: Vectorfields $\mathbf{E_1} = (-y, x)$ and $\mathbf{E_2} = -(y, x)$ plotted with quiver.

d)

```
h = @(x) exp(1 - 1 ./ x) ./ x;
int1 = integral(h, eps, 1);
```

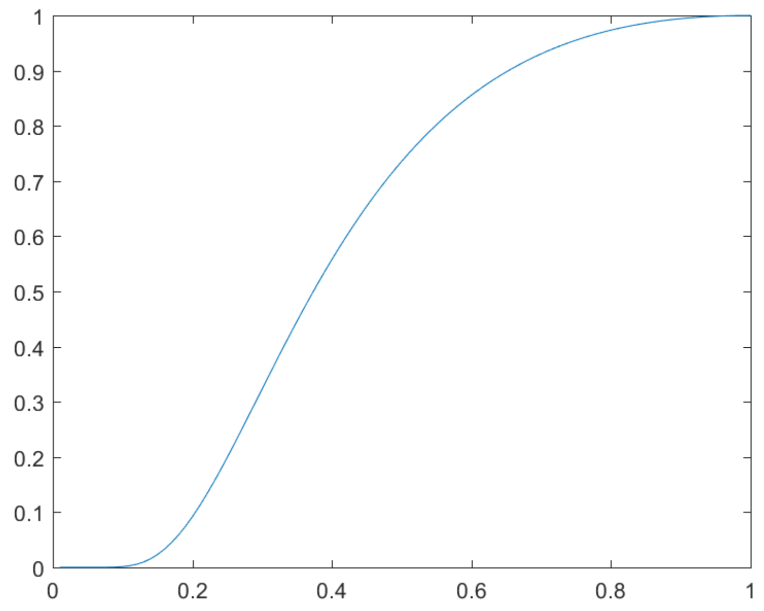


Figure 4: Function $h(x)$ plotted in matlab.

Even though matlab properly integrates this function and acquires the right value of about

$$\int_0^1 \mathrm{d}x\, h(x) = e \cdot \Gamma\left(0,1\right) \approx 0.5963 \tag{1}$$

it could still pose a problem for similar functions since the function inhibits a pole on $x = 0$. It can easily through variable substitution be shown, that

$$\lim_{x\to 0} h(x) = 0 \tag{2}$$

but programs cannot do the symbolic limit implicitly and therefore acquire an error due to the division by zero. This can be cirumvented with two possibilities:

1. Since we know $h(x \to 0) \to 0$ we can explicitly spare the integral-function of the pole by adjusting integration limit to $[\epsilon, 1]$ where ϵ is the machine-epsilon of the according floating point type (*eps* in matlab).

2. We can do a variable substition $x = 1/y$ under the integral

$$\int_0^1 \mathrm{d}x\, \frac{\exp\left(1 - \frac{1}{x}\right)}{x} = \int_1^\infty \mathrm{d}y\, \frac{\exp\left(1 - y\right)}{y} \tag{3}$$

   This seems to pose the exchanged problem of integrating to infinity. However, after quick calculations in can be shown that $h(y) < \epsilon$ for $y \geq 34.31$ with a strong monotonic descrease $\left(h(y) \propto 1/\left(y \cdot \exp y\right)\right)$. So we can comfortably set the integration limits to $[1, 34.31]$ (or adapting to a specific precision-limit).

Task 2

Table 2: Program file index - Task 2

A2/A2.jl	Julia	program with tasks a, b and part of c
A2/naiveDFT.jl	Julia	implementation of the naive dft in matrix and iterative form
A2/simpleDFT.jl	Julia	implementation of the recursive dft
A2/benchmarkFunction.jl	Julia	benchmark function for fft-functions
A2/A2c_MatlabFFT.m	Matlab	programm to acquire matlab fft benchmarks
A2/A2d.nb	Mathematica	notebook to calculate aliasing graphs
A2/data/Visualize.nb	Mathematica	notebook to calculate graphs for c

a) and b)

According to the program file index the ffts where implemented in naive and recursive form and compared to the FFTW package. The comparison was made with an artificial created dataset of three sine waves with different amplitude, frequency and phase.

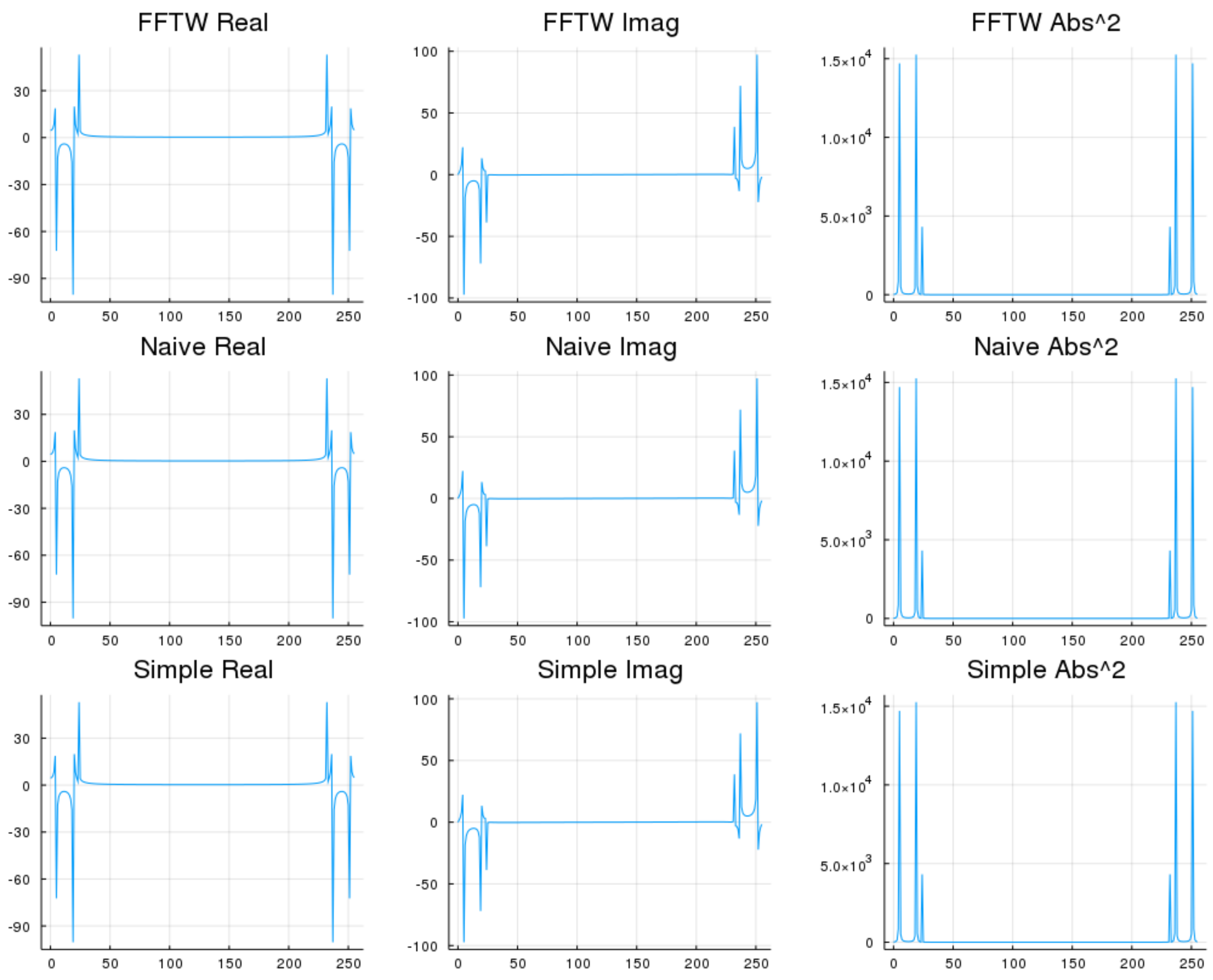


Figure 5: Comparison of real, imaginary and absolute values of FFTW, the naive and recursive implementation

The summed differences where all calculated to be below 10^{-10}.

c)

The Julia functions and a Matlab benchmark where made with different problem sizes all with a power of two. The stopping criterion was set, when an implementation reached calculation times over a second. Each test was repeated five times and was averaged out.
Following results where acquired:

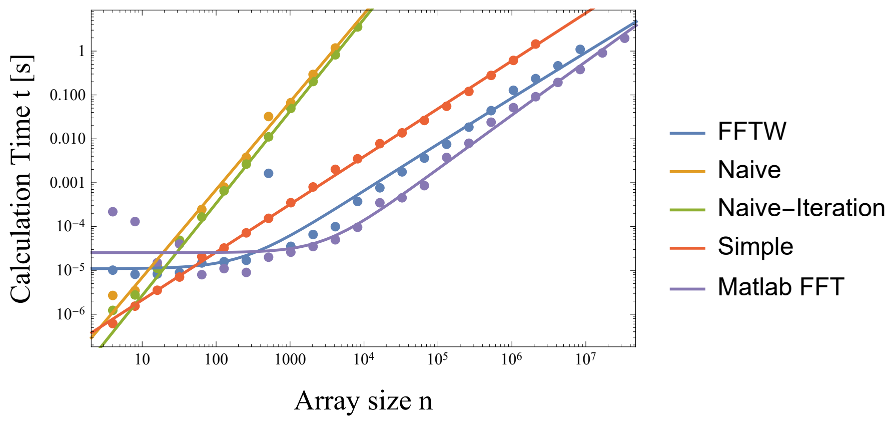
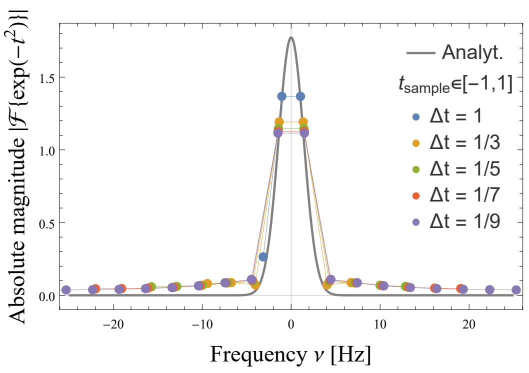


Figure 6: Comparisons between different dft implementations. The simple, recursive Julia implementation is notably only 5 times slower than FFTW.

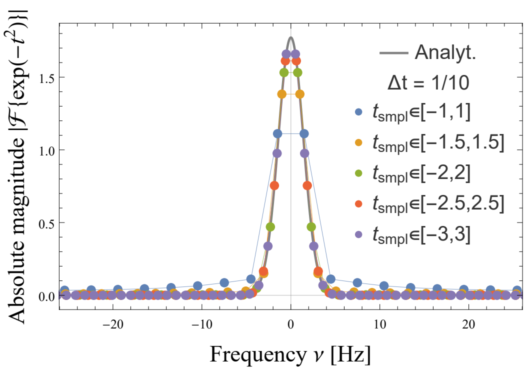
The fastest implementation seems to be matlabs fft. However, it is known that matlab utilizes FFTW-plan implementations. The exact same speed could be therefore in Julia be acquired with a single FFTW-plan allocation.
The naive implementations are fitted with a runtime slightly above $\mathcal{O}\left(N^2\right)$. All other implemn-tations are approximately $\approx \mathcal{O}\left(N^{1.1}\right)$ for the simple, recursive implementation and $\mathcal{O}\left(N \log N\right)$ in the given uncertainty intervals.

d)

$$\hat{\mathcal{F}}_t^\omega \left\{\exp\left(-t^2\right)\right\} = \sqrt{\pi} \exp\left(\frac{-\omega^2}{4}\right) \tag{4}$$



(a) Comparison between different sampling distances Δt



(b) Comparison between different sampling intervals $t_{smpl} \in [-t, t]$

Figure 7: Comparison of aliasing in different sample intervals and sampling distances.

It can quickly be seen how the different intervals and distances distort the resulting fourier points. The sampling interval seems to have a stronger effect.

To confirm that, a density plot of the absolute differences to the analytical solution can be made.

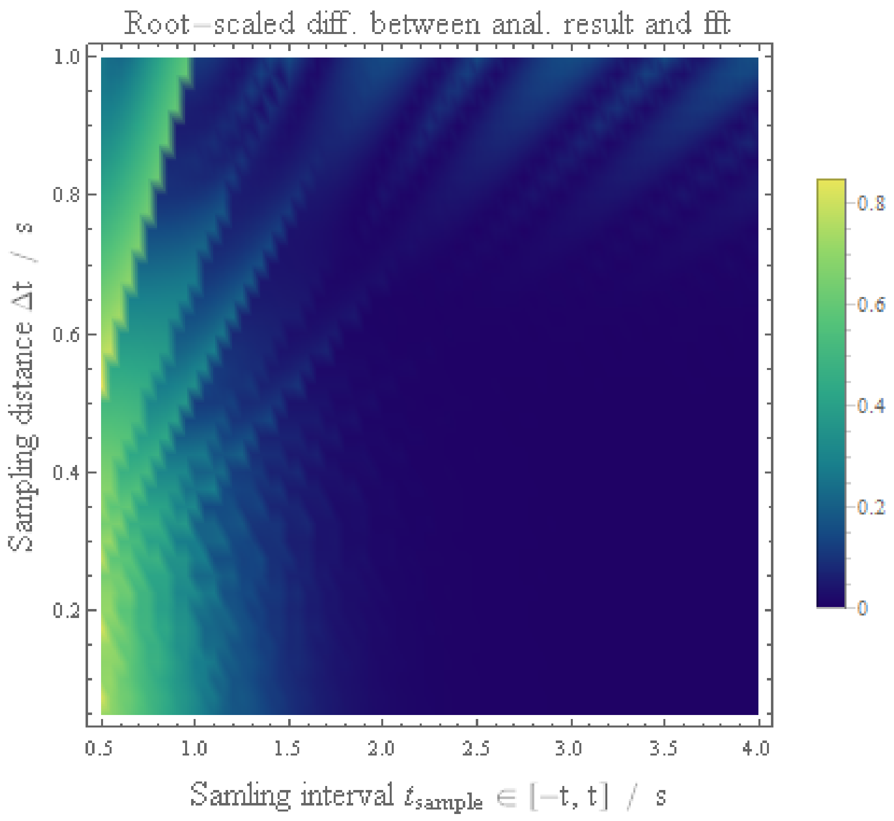


Figure 8: Absolute differences of fft to analytical solution, scaled by the squareroot (for better visibility) in respect to the sampling interval and distances.

The multiplication by $N\,\Delta t\,(-1)^n$ is necessary to normalize the fourier coefficients and the last term is for shifting the resulting coefficients to the zero-frequency.

Task 3

$$\frac{\partial \mathbf{H}(\mathbf{r},\mathbf{t})}{\partial t} = -\frac{1}{\mu_0\,\mu(\mathbf{r})}\nabla \times \mathbf{E}(\mathbf{r},t) \tag{5}$$
$$\frac{\partial \mathbf{E}(\mathbf{r},\mathbf{t})}{\partial t} = \frac{1}{\epsilon_0\,\epsilon(\mathbf{r})}\nabla \times \mathbf{H}(\mathbf{r},t) \tag{6}$$

First we introduce to arbitrary field strengths E_0, H_0 such that

$$\frac{\mathbf{E}}{E_0} = \widetilde{\mathbf{E}} \tag{7}$$
$$\frac{\mathbf{H}}{H_0} = \widetilde{\mathbf{H}} \tag{8}$$

Which then renders the upper equations to

$$\frac{\partial \widetilde{\mathbf{H}}(\mathbf{r},\mathbf{t})}{\partial t} = -\frac{E_0}{H_0\,\mu_0}\frac{1}{\mu(\mathbf{r})}\nabla \times \widetilde{\mathbf{E}}(\mathbf{r},t) \tag{9}$$
$$\frac{\partial \widetilde{\mathbf{E}}(\mathbf{r},\mathbf{t})}{\partial t} = \frac{H_0}{E_0\,\epsilon_0}\frac{1}{\epsilon(\mathbf{r})}\nabla \times \widetilde{\mathbf{H}}(\mathbf{r},t) \tag{10}$$

We can equate the terms with $E_0, H_0, \epsilon_0, \mu_0$ to a constant. Since the field strength modifier are arbitrary, the constants are also arbitrary. Therefore we set it equal to one:

$$1 = \frac{E_0}{H_0\,\mu_0} \tag{11}$$
$$1 = \frac{H_0}{E_0\,\epsilon_0} \tag{12}$$

This system is underdetermined. We can therefore only solve for one solution. Multiplication and division yields following equations respectively

$$1 = \frac{1}{\epsilon_0\,\mu_0} = c^2 \tag{13}$$
$$1 = \frac{E_0^2\,\epsilon_0}{H_0^2\,\mu_0} \tag{14}$$

The first gives the natural units where $c = 1$. This enables further manipulation for time or space such that $\mathbf{r}/r_0 = \widetilde{\mathbf{r}}$ and $t \cdot c/r_0 = \widetilde{t}$. The second equation gives a solution for either variable. For the arbitrary decision to solve H_0 we get

$$H_0 = E_0\sqrt{\frac{\epsilon_0}{\mu_0}} \tag{15}$$

Thus the final result in natural units ($c = 1$) can be archived:

$$\frac{\partial \widetilde{\mathbf{H}}(r_0\,\widetilde{\mathbf{r}}, r_0\,\widetilde{t})}{\partial \widetilde{t}} = -\frac{1}{\mu(r_0\,\widetilde{\mathbf{r}})}\nabla_{\widetilde{r}} \times \widetilde{\mathbf{E}}(r_0\,\widetilde{\mathbf{r}}, r_0\,\widetilde{t}) \tag{16}$$
$$\frac{\partial \widetilde{\mathbf{E}}(r_0\,\widetilde{\mathbf{r}}, r_0\,\widetilde{t})}{\partial \widetilde{t}} = \frac{1}{\epsilon(r_0\,\widetilde{\mathbf{r}})}\nabla_{\widetilde{r}} \times \widetilde{\mathbf{H}}(r_0\,\widetilde{\mathbf{r}}, r_0\,\widetilde{t}) \tag{17}$$

Which also gives us the spatial discretization as r_0.