Exercise Sheet № 7 FFT and Numerical Differentiation & Integration

7.0 Theoretical Backgrounds

7.0.1 Fourier-Transform and the FFT

Since a large topic of this exercise is the ***Fast-Fourier Transform***[1], I wanted to introduce some definitions and notations used throughout this submission. Formally speaking, the fourier-transform[2] arises through observations in *functional analysis*. In the context of linear algebra, the FT represents a change of basis for signals.

Definition 7.1: Hilbert Space

Let $\mathcal{X}$ be a vector space over a field $\mathbb{F}$, either $\mathbb{R}$ or $\mathbb{C}$. If $\langle\cdot,\cdot\rangle : \mathcal{X}^2 \to \mathbb{F}$ is a semi-positive definite sesquilinear dot-product, and $\mathcal{X}$ is complete with the respect to the induced norm $\|\cdot\|\colon \mathcal{X} \to \mathbb{F}$, $\|\boldsymbol{x}\| = \sqrt{\langle \boldsymbol{x}, \boldsymbol{x}\rangle}$, then we call $\mathcal{X}$ a Hilbert-Space.

Definition 7.2: Signal

A signal $x\colon D \to R$ is a function, that is typically element of a Hilbert-space $\mathcal{X}$. The support, or carrier, of a signal x, denoted $\operatorname{supp} x$ is the following set:

$$\operatorname{supp} x = \{t \in D\colon x(t) \neq 0\}$$

where $0 \in \mathbb{F}$. We call a signal x time-discrete, if $|D| \leq \aleph_0$. Similarly, we call x value-discrete, if $|R| \leq \aleph_0$. If x is time- and value-discrete, we call the signal *discrete*.

While generally speaking, we define signals in a very abstract way, we want to focus on time-discrete signals $x\colon \mathbb{Z} \to \mathbb{C}$ with finite support. For simplicity's sake, we set $\operatorname{supp} x = \{0, \dots, N-1\}$ for $N \in \mathbb{N}$.

Definition 7.3: Discrete Fourier Transform

Let $x\colon \mathbb{Z} \to \mathbb{C}$ be a signal with finite support and $N \geq 2$. Then the „coordinates“ in the DFT[a]-basis are given by:

$$X[k] = \sum_{n=0}^{N-1} x[n] \exp\left(\frac{-\mathrm{i}2\pi kn}{N}\right) \tag{1}$$

Given the DFT of x, we can get the original signal via the *inverse DFT*[b]

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k] \exp\left(\frac{\mathrm{i}2\pi kn}{N}\right) \tag{2}$$

Given a signal x and it's DFT X, we write $x \overset{\text{DFT}}{\longleftrightarrow} X$ to denote their relation. We set

$$\exp\left(\frac{\mathrm{i}2\pi kn}{N}\right) = \zeta_N^{kn}$$

Notice that ζ_N^k represents the k-th of the N roots of unity, thus:

$$\zeta_N^{Nn} = \mathrm{e}^{\mathrm{i}2\pi n} = \cos(2\pi n) + \mathrm{i}\sin(2\pi n) = 1$$

Notice now, that both eqs. (1) and (2) require N *complex* multiplications and $N-1$ additions, per time-step

[1] abbr. FFT
[2] abbr. FT
[a] abbr. DFT = Discrete Fourier Transform
[b] abbr. iDFT

n. Thus to transform an entire signal x into it's DFT X, we need N^2 complex multiplications and $N^2 - N$ additions. The required time to perform the DFT lies in $\mathcal{O}(N^2)$, which is sub-optimal, given the DFT is often required to run on low-end devices[3]. The idea of the ***Fast-Fourier Transform***[4] now is to split x into two signals x_e and x_o, where

$$x_e[n] = \begin{cases} x[n] & n \equiv 0 \mod 2 \\ 0 & \text{else} \end{cases} \qquad x_o[n] = \begin{cases} x[n] & n \equiv 1 \mod 2 \\ 0 & \text{else} \end{cases}$$
$$\Rightarrow x[n] = x_e[n] + x_o[n]$$

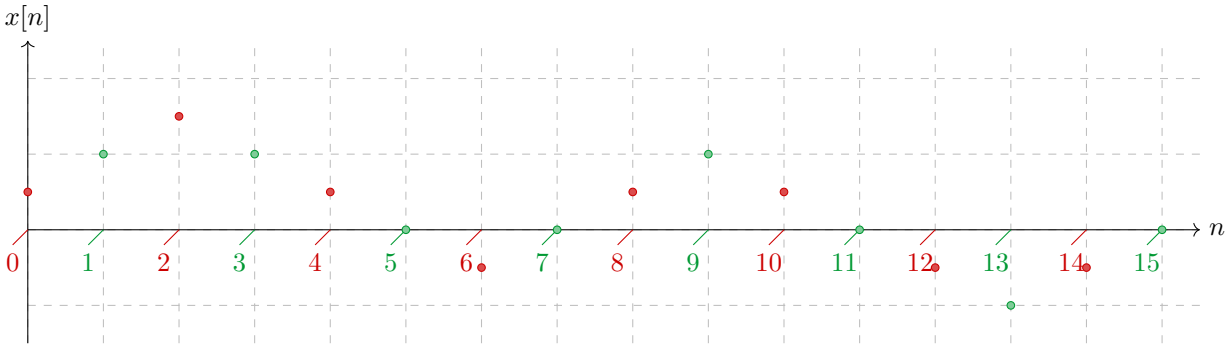

Figure 1: Splitting of x into x_e and x_o

Without loss of generality, let $N \equiv 1 \mod 2$, then computing $X[k]$ yields:

$$\begin{aligned} X[k] &= \sum_{n=0}^{N-1} x[n] \mathrm{e}^{-\mathrm{i}\frac{2\pi}{N}kn} = \sum_{n=0}^{N-1} x_e[n] \mathrm{e}^{-\mathrm{i}\frac{2\pi}{N}kn} + \sum_{n=0}^{N-1} x_o[n] \mathrm{e}^{-\mathrm{i}\frac{2\pi}{N}kn} \\ &= \sum_{m=0}^{\frac{N}{2}-1} x[2m] \mathrm{e}^{-\mathrm{i}\frac{2\pi}{N}k2m} + \sum_{m=0}^{\frac{N}{2}-1} x[2m+1] \mathrm{e}^{-\mathrm{i}\frac{2\pi}{N}k(2m+1)} \\ &= \sum_{m=0}^{\frac{N}{2}-1} x[2m] \mathrm{e}^{-\mathrm{i}\frac{2\pi}{\frac{N}{2}}km} + \mathrm{e}^{-\mathrm{i}\frac{2\pi}{N}k} \sum_{m=0}^{\frac{N}{2}-1} x[2m+1] \mathrm{e}^{-\mathrm{i}\frac{2\pi}{\frac{N}{2}}km} \end{aligned} \tag{3}$$

We see, that we split the DFT of x into two shorter sums of half the length. Equation (3) can be represented by the following signal flow-graph (for $N = 8$):

[3]The DFT is a critical operation in digital communications, as it allows far more efficient data transfer and compression. Using the classical definition of the DFT would result in bad operational speeds.
[4]abbr. FFT

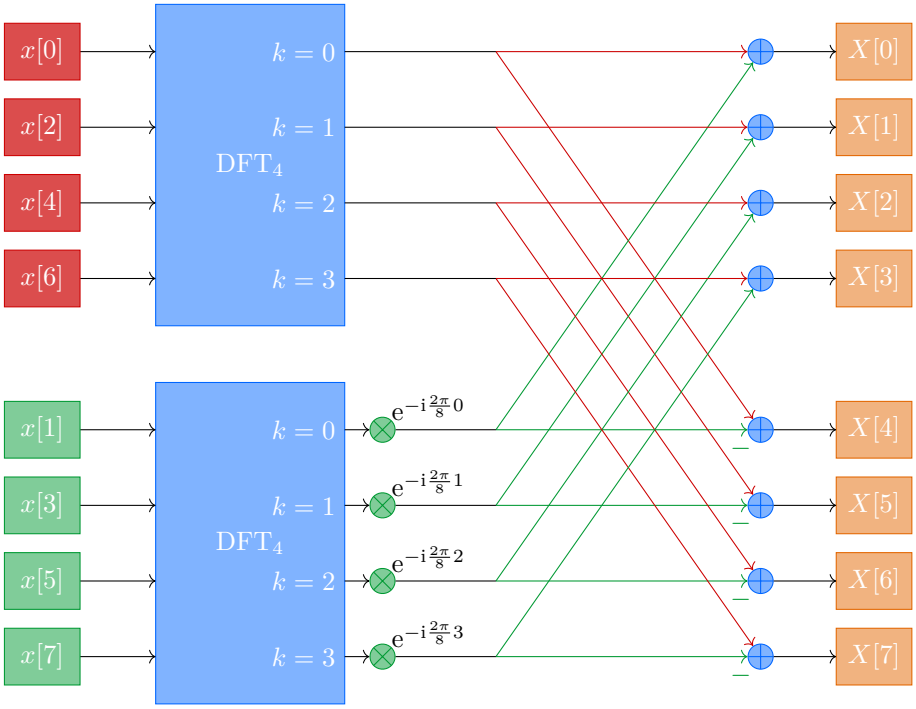

Figure 2: Recursive View of 8-point Radix-2 FFT as a signal-flow-graph

Equation (3) and fig. 2 lead us to the following recursive algorithm for the FFT:

Algorithm 1: Recursive Implementation of Radix-2 FFT

```
input: signal x of length N
output: DFT X of length N

FFT(x):
    X = 0_N
    if N == 1 then
        X[0] = x[0]
    else
        a = FFT(x[even indices])
        b = FFT(x[odd indices])
        φ_k = e^{-2iπk/N}
        m = N/2
        for k, a_k, b_k in enum(a, b) do
            X[k] = a_k + φ_k b_k
            X[k + m] = a_k − φ_k b_k
        end
    end
    return X
```

7.0.2 Numerical Integration : Newton Cotes Formulae

Definition 7.4: Integral Operator Notation

Let $I \subseteq \mathbb{R}$ be an interval $[a, b]$, then we set:

$$\mathcal{I}_I(f) = \int_I f(x)\,\mathrm{d}x \qquad \mathcal{I}(f) = \int_{-1}^{1} f(x)\,\mathrm{d}x$$

for $f \in \mathcal{C}(I)$.

Definition 7.5: Quadrature of order n

Let $x_0, \dots, x_n \in [a,b]$ and $w_0, \dots, w_n \in \mathbb{R}$ be weights. We call

$$\mathcal{I}_n(f) = \sum_{j=0}^{n} w_j f(x_j)$$

the quadrature of order n. We call $\mathcal{I}_n$ exakt to degree m, iff

$$\forall p \in \mathbb{R}_m[x] \colon \mathcal{I}_n(p) = \int_a^b p(x)\,\mathrm{d}x$$

Let $h \in \mathbb{R}$. If $\forall i = 1, \dots, n \colon x_i - x_{i-1} = h$, i.e. the x_i are equidistant, we get the so called *Newton-Cotes* formulae:

$$\mathcal{I}_N(f) = (b-a)\sum_{j=0}^{n} w_j f(x_j) \qquad x_j = a + \frac{j}{n}(b-a) \tag{4}$$

The weights w_j are given by the integrals of the Lagrange-Basis polynomials ℓ_j:

$$w_j = \int_a^b \ell_j(x)\,\mathrm{d}x$$

n	Name	w_j	m	Error
1	Trapezoidal	$\frac{1}{2}, \frac{1}{2}$	1	$\lVert f'' \rVert_\infty \mathcal{O}(h^3)$
2	Simpson	$\frac{1}{6}, \frac{4}{6}, \frac{1}{6}$	3	$\lVert f^{(4)} \rVert_\infty \mathcal{O}(h^5)$
3	Newtons $\frac{3}{8}$	$\frac{1}{8}, \frac{3}{8}, \frac{3}{8}, \frac{1}{8}$	3	$\lVert f^{(4)} \rVert_\infty \mathcal{O}(h^5)$
4	Milne	$\frac{7}{90}, \frac{32}{90}, \frac{12}{90}, \frac{32}{90}, \frac{7}{92}$	5	$\lVert f^{(6)} \rVert_\infty \mathcal{O}(h^7)$

Table 1: Weights for the first four newton cotes formulae

Similarly to interpolation, increasing n also increases the error of $\mathcal{I}_n$. Thus, alike to spline-interpolation, we want to split the integral over a „big“ Interval $I = [a,b]$ into N sub-intervals $[x_j, x_{j+1}]$ and compute $\mathcal{I}_n$ over $[x_j, x_{j+1}]$. This gives the following formula for summed quadratures

$$\mathcal{I}_n^N(f) = \sum_{j=0}^{N-1} \mathcal{I}_n(f, [x_j, x_{j+1}]) \tag{5}$$

Given table 1, we can express trapezoidal numerical integration as a closed expression. Note that $x_1 - x_0 = h$, i.e. $x_1 = h + x_0$, such that $x_1, x_0 \in [a,b]$. If $h = b - a$, then $x_0 = a$ and $x_1 = b$.

$$\ell_0(x) = \frac{x - x_1}{x_0 - x_1} = \frac{x_1 - x}{h} \Rightarrow w_0 = \int_a^b \ell_0(x)\,\mathrm{d}x = \frac{1}{h}\left(x_1 x - \frac{x^2}{2}\Big|_a^b\right) = \frac{1}{h}\left(x_1 h - \frac{1}{2}(b^2 - a^2)\right)$$
$$= \frac{1}{2h}(2hx_1 - (b-a)(b+a)) = \frac{1}{2}(2x_1 - b - a)$$
$$\ell_1(x) = \frac{x - x_0}{x_1 - x_0} = \frac{x - x_0}{h} \Rightarrow w_1 = \int_a^b \ell_1(x)\,\mathrm{d}x = \frac{1}{h}\left(\frac{x^2}{2} - x_0 x\Big|_a^b\right) = \frac{1}{h}\left(\frac{1}{2}h(b+a) - x_0 h\right)$$
$$= \frac{1}{2}\left(b + a - 2x_0\right)$$

For the trapezoidal rule, we set $h = b - a$, thus the integration weights become:

$$w_0 = \frac{1}{2}(b-a) = \frac{h}{2}$$

$$w_1 = \frac{1}{2}(b-a) = \frac{h}{2}$$

With these weights, we get the following closed expression for $\mathcal{I}_1(f)$:

$$\mathcal{I}_1(f) = \frac{h}{2}(f(a) + f(b))$$

7.0.3 Numerical Integration : Gaussian Quadrature

While the summed Newton-Cotes quadratures may have a relatively small error, in order to allow for high-order exactness, we need large n. However, using equidistant nodes x_i, we get negative weights for $n > 8$. This poses cancellation problems, thus we need to adapt parts of our quadratures. Our goal is to find a quadrature $\mathcal{I}_n$ that is exact with order $2n+1$. In order to achieve this degree of exactness, we need to find the zeros $x_0, \dots, x_n \in I$, with $I = [x_0, x_n]$ of the polynomial $\omega_{n+1} \in \mathbb{R}_n[x]$, which fulfills:

$$\forall p \in \mathbb{R}_n[x]\colon \int_I \omega_n(x)p(x)\,\mathrm{d}x = 0$$

Given that $\langle \cdot, \cdot \rangle_I : (R_{n+1}[x])^2 \to \mathbb{R}$ with:

$$\langle p, q \rangle_I = \int_I p(x)q(x)\,\mathrm{d}x$$

is bilinear by the properties of the integral, we see that $\mathrm{span}(\omega_n) \oplus \mathbb{R}_n[x] = \mathbb{R}_{n+1}[X]$, i.e. $\omega_n \perp \mathbb{R}_n[x]$. If $I = [-1, 1]$, we can use the Legendre-polynomials.

Definition 7.6: Legendre-Polynomials

The Legendre-polynomial L_n is the solution of the Legendre differential equation with $\alpha = n$:

$$(x^2-1)L_n''(x) + 2xL_n'(x) - n(n+1)L_n(x) = 0 \qquad x \in (-1, 1) \tag{6}$$

Solving eq. (6) yields, among other ones, the following representation for $L_n(x)$:

$$L_n(x) = \frac{1}{2^n}\sum_{k=0}^{n}\binom{n}{k}^2 (x-1)^{n-k}(x+1)^k$$

k	$L_k(x)$	x_i	$L_k'(x)$
0	1		0
1	x	0	1
2	$\frac{1}{2}(3x^2-1)$	$-\frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}$	$3x$
3	$\frac{1}{2}(5x^3-3x)$	$-\sqrt{\frac{3}{5}}, 0, \sqrt{\frac{3}{5}}$	$\frac{3}{2}(5x^2-1)$

Table 2: The first four Legendre-polynomials, their zeros and derivatives

The weights for the Gauss-Legendre quadrature of order n are given by:

$$w_i = \frac{2}{(1-x_i)(L_n'(x_i))^2} \tag{7}$$

An alternative representation to eq. (7) is:

$$w_i = \int_{-1}^{1} \frac{L_n(x)}{(x-x_i)L_n'(x_i)}\,\mathrm{d}x = \int_{-1}^{1} \ell_{in}(x)\,\mathrm{d}x \tag{8}$$

n	weights
1	2
2	$1, 1$
3	$\frac{5}{9}, \frac{8}{9}, \frac{5}{9}$

Table 3: Weights for Gauss-Legendre quadrature

Definition 7.7: Gauss-Legendre quadrature

Let $x_0, \ldots, x_n$ be the zeros of $L_n(x)$ and w_i be given by eq. (7), we define the Gauss-Legendre quadrature:

$$\mathcal{G}_n(f) = \sum_{j=0}^{n} w_j f(x_j) \approx \int_{-1}^{1} f(x)\,\mathrm{d}x$$

7.1 Numerical Differentiation

Task 7.1: Numerical Differentiation via Interpolation

1. Prove the following second order forward finite difference approximation for the derivative of a function f at x:

$$f'(x) = \frac{1}{2h}\big(-f(x+2h) + 4f(x+h) - 3f(x)\big) + \mathcal{O}(h^2) \tag{9}$$

   Derive a formula for a second order forward finite difference approximation for $f''(x)$. What are the corresponding approximations for $f'(x)$ and $f''(x)$ using Newtons's interpolation method?
2. Given the evenly space data points from table 4

$\mathcal{X}$	0	0.1	0.2	0.3	0.4
$\boldsymbol{y}$	0	0.0819	0.1341	0.1646	0.1797

Table 4: Data for Subtask 2

   approximate $f'(x)$ and $f''(x)$ at $x = 0$ and $x = 0.2$ using the second order forward finite difference formulae from Subtask 1. When the given values of x are not equidistant, how are you going to approximate the values of $f'(x)$ and $f''(x)$?
3. The data points from table 5 lie on the graph of $f(x) = 4.8\cos\left(\frac{\pi x}{20}\right)$

$\mathcal{X}$	0.15	2.30	3.15	4.85	6.25	7.95
$\boldsymbol{y}$	4.79867	4.49013	4.2243	3.47313	2.66674	1.51909

Table 5: Data for Subtask 3

   Approximate the value of $f'(x)$ and $f''(x)$ at $x = 2.50$ via Newton Interpolation and compare your results with the exact values. Revise your python program from Exercise 6.2 to plot the graph f the first derivatives of the interpolant and the exact function in one window (and do the same for the second derivative).

Subtask 1: For a second order forward finite difference, we consider the taylor-expansions of both $f(x+h)$ and $f(x+2h)$:

$$\begin{aligned} f(x+h) &= f(x) + hf'(x) + \frac{h^2}{2}f''(x) + \frac{1}{6}h^3 f'''(x) + \mathcal{O}(h^4) \\ f(x+2h) &= f(x) + 2hf'(x) + 2h^2 f''(x) + \frac{4h^3}{3}f'''(x) + \mathcal{O}(h^4) \end{aligned}$$

Eliminating f'' yields:

$$\begin{aligned} & f(x+2h) - 4f(x+h) = -3f(x) - 2hf'(x) + \mathcal{O}(h^3) \\ \Leftrightarrow & 2hf'(x) = -3f(x) + 4f(x+h) - f(x+2h) + \mathcal{O}(h^3) \\ \Leftrightarrow & f'(x) = \frac{1}{2h}\left(4f(x+h) - 3f(x) - f(x+2h)\right) + \mathcal{O}(h^2) \end{aligned} \tag{10}$$

Eliminating f' yields:

$$\begin{aligned} & f(x+2h) - 2f(x+h) = -f(x) + h^2 f''(x) + \mathcal{O}(h^3) \\ \Leftrightarrow & h^2 f''(x) = f(x+2h) - 2f(x+h) + f(x) + \mathcal{O}(h^3) \\ \Leftrightarrow & f''(x) = \frac{1}{h^2}\left(f(x+2h) - 2f(x+h) + f(x)\right) + \mathcal{O}(h) \end{aligned} \tag{11}$$

Given that Newton's method produces interpolating polynomials $p \in \mathbb{R}_n[x]$, we can use the fact, that $\mathbb{R}_n[x] \subset \mathcal{C}^\infty(\mathbb{R})$. Let c_k be the coefficients of p, such that $\forall i = 0, \ldots, n\colon p(x_i) = f(x_i)$, where f is the unknown function

p the interpolant and x_i form an ordered set of nodes $\mathcal{X} = (x_0, \cdots, x_n)$. Let

$$\omega_k(x) = \prod_{i=0}^{k} x - x_i$$

By convention we set $\omega_0 = 1$, then the newton interpolant becomes:

$$p(x) = \sum_{k=0}^{n} c_k \omega_k(x)$$

By theorem 7.1, we get:

$$p'(x) = \sum_{k=0}^{n-1} c_{k+1}\omega_k'(x) = \sum_{k=0}^{n-1} c_{k+1}\omega_k(x) \sum_{i=0}^{k} \frac{1}{x - x_i}$$
$$p''(x) = \sum_{k=0}^{n-2} c_{k+2}\omega_k''(x) = \sum_{k=0}^{n-2} c_{k+2}\omega_k(x) \left(\left(\sum_{i=0}^{k} \frac{1}{x - x_i} \right)^2 - \sum_{i=0}^{k} \frac{1}{(x - x_i)^2} \right)$$

Subtask 2: With eqs. (10) and (11) we get:

x	$f'(x)$	$f''(x)$	$p'(x)$	$p''(x)$
0	0.9675	−2.97	0.9675	0.382
0.2	0.382	−1.54	−2.97	−1.54

Table 6: Values of f' and f'' in 0 and 0.2 based on data from table 4

If the given x-values are not equidistant, we can interpolate f with a fitting method and introduce new x_i, until the x-values are equidistant and use the interpolant for the missing function-values.

Subtask 3:

$f'(2.5)$	−0.28854
$f''(2.5)$	−0.10942
$p'(2.5)$	−0.29672
$p''(2.5)$	−0.09468

Table 7: Values of f' and f'' and the approximated values in $x = 2.5$

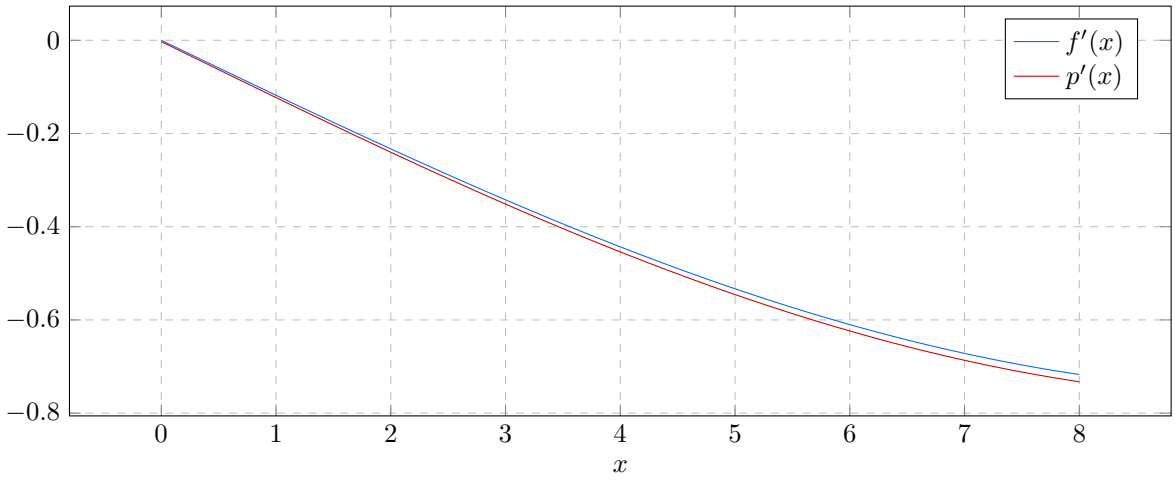

Figure 3: Plot of f' and p' over $[0, 8]$

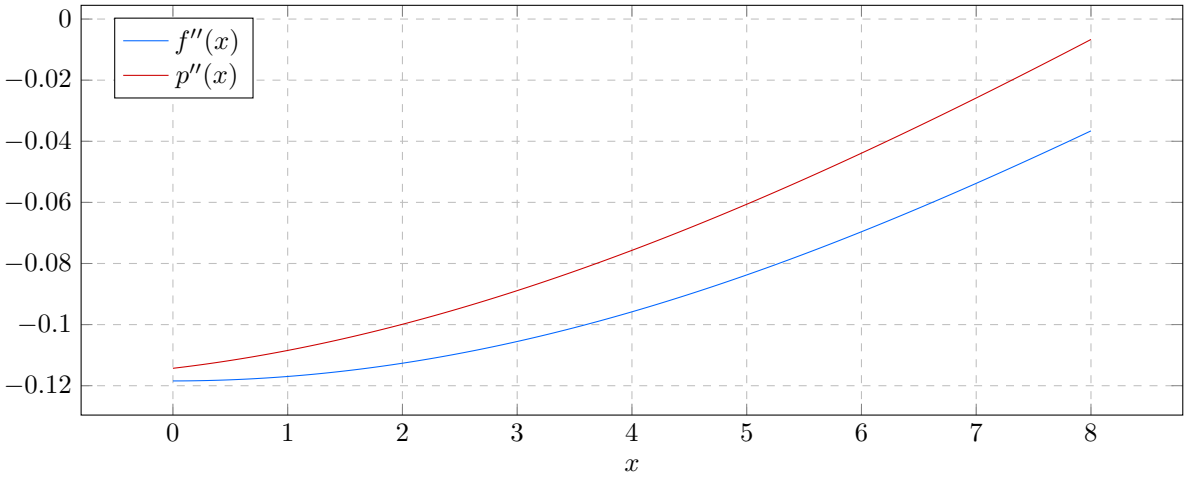

Figure 4: Plot of f'' and p'' over $[0, 8]$

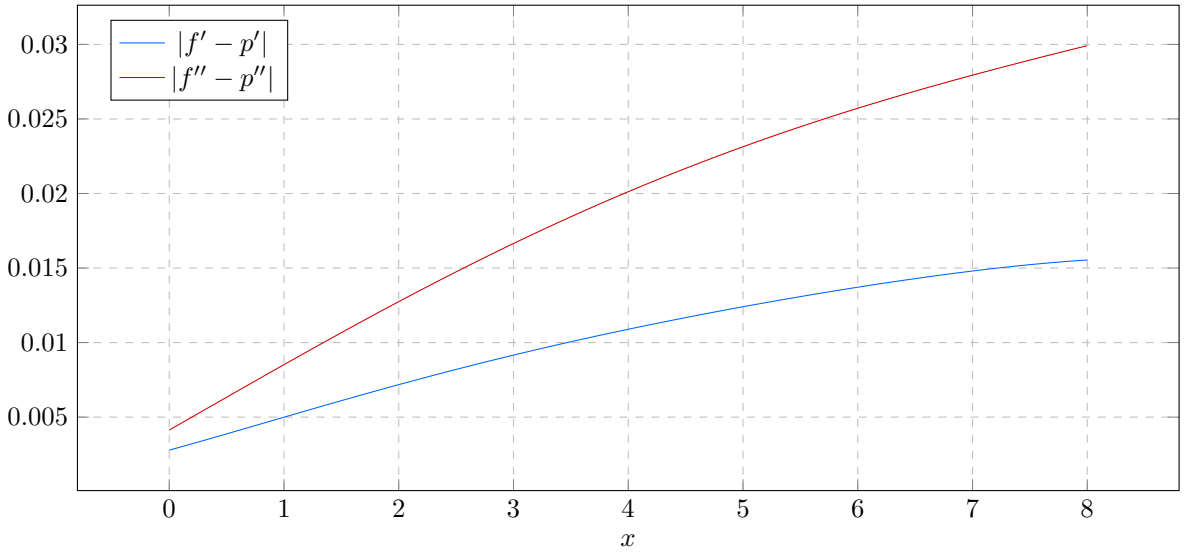

Figure 5: Approximation errors

The values for figs. 3 to 5 as well as table 7 are computed in the submitted jupyter-notebook `7_1_3.ipynb`, while the data from table 6 is generated in `forwdiff.ipynb`.

7.2 Fast Fourier Transform I

Task 7.2: Fast Fourier Transform I

1. Given a signal x, write a python function `FFT(x)` which implements algorithm 1. Provide an algorithm for the iFFT and write another function `iFFT(x)` which implements the iFFT.
2. Consider the following signals and DFTs:

$$x_1 = \begin{bmatrix}1 & 1 & 1 & 1\end{bmatrix} \qquad X_1 = \begin{bmatrix}1 & 1 & 1 & 1\end{bmatrix}$$
$$x_2 = \begin{bmatrix}1 & 1 & 1 & 1 & 0 & 0 & 0 & 0\end{bmatrix} \quad X_2 = \begin{bmatrix}1 & 1 & 1 & 1 & 0 & 0 & 0 & 0\end{bmatrix}$$
$$x_3 = \begin{bmatrix}1 & 0 & 1 & 0 & 1 & 0 & 1 & 0\end{bmatrix} \quad X_3 = \begin{bmatrix}1 & 0 & 1 & 0 & 1 & 0 & 1 & 0\end{bmatrix}$$

Table 8: Data for Subtask 2

   Compute the DFT of x_1 and iDFT of X_1 by hand. Test your functions `FFT` and `iFFT` using the data from table 8. Discuss what happens if you pad the signal x_1 or the DFT X_1 with trailing zeros, e.g. x_2 and X_2, or interlace them with zeros, i.e. x_3 and X_3.
3. The file `fftdata.dat` contains the data points $(t, x(t))$ of fig. 6. Read the data in a python program and use your function `FFT` to determine the frequencies and corresponding amplitudes present in the signal. Visualize your results and do the same for Subtask 2

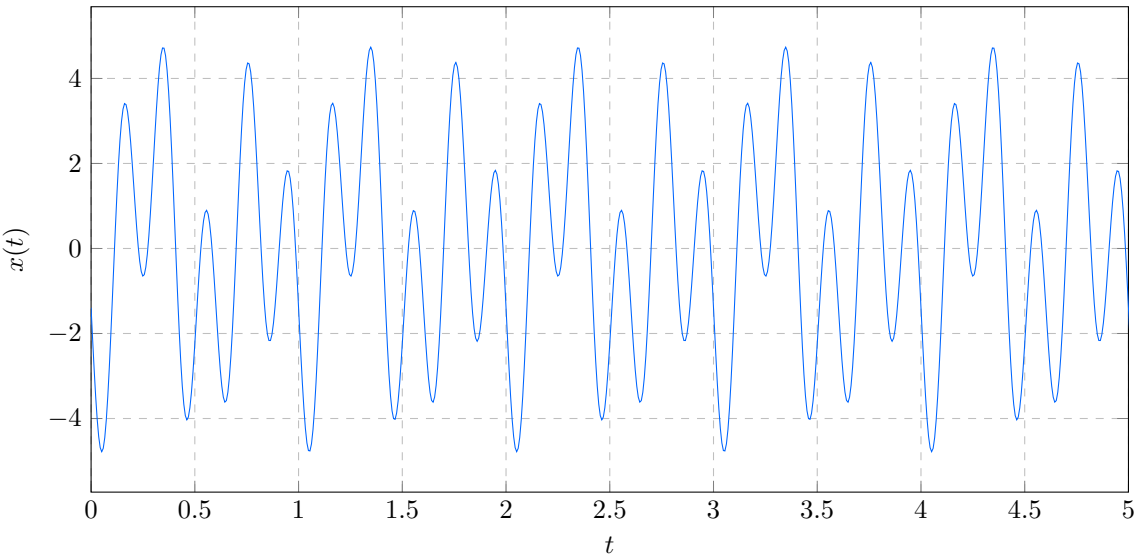

Figure 6: Part of the signal for Subtask 3

Subtask 1: As stated in section 7.6.2, I opted to implement algorithm 2 in favor of algorithm 1.

Subtask 2: With eq. (1) we get the following values for X_1:

$$X_1[0] = \sum_{n=0}^{3} 1 = 4$$
$$X_1[1] = \sum_{n=0}^{3} \mathrm{e}^{-\frac{\mathrm{i}\pi n}{2}} = 1 - \mathrm{i} - 1 + \mathrm{i} = 0$$
$$X_1[2] = \sum_{n=0}^{3} \mathrm{e}^{-\mathrm{i}\pi n} = \sum_{n=0}^{3} \cos(\pi n) = 1 - 1 + 1 - 1 = 0$$
$$X_1[3] = \sum_{n=0}^{3} \mathrm{e}^{-\frac{3\pi i n}{2}} = 1 + \mathrm{i} - 1 - \mathrm{i} = 0$$

With eq. (2) we get the following values for x_1:

$$x_1[0] = \frac{1}{4}\sum_{k=0}^{3} 1 = 1$$
$$x_1[1] = \frac{1}{4}\sum_{k=0}^{3} \mathrm{e}^{\frac{\pi \mathrm{i} k}{2}} = \frac{1}{4}(1 + \mathrm{i} - 1 - \mathrm{i}) = 0$$
$$x_1[2] = \frac{1}{4}\sum_{k=0}^{3} \mathrm{e}^{\mathrm{i}\pi k} = 0$$
$$x_1[3] = \frac{1}{4}\sum_{k=0}^{3} \mathrm{e}^{\frac{3\mathrm{i}\pi k}{2}} = \frac{1}{4}(1 - \mathrm{i} - 1 + \mathrm{i}) = 0$$

Using algorithm 2 in the submitted jupyter-notebook `fft_short.ipynb`, I get the following values

input	operation	output
x_2	DFT	$\begin{bmatrix}4 & 1+2.414\mathrm{i} & 0 & 1+0.414\mathrm{i} & 0 & 1-0.414\mathrm{i} & 0 & 1-2.414\mathrm{i}\end{bmatrix}$
X_2	iDFT	$\begin{bmatrix}0.5 & 0.125-0.301\mathrm{i} & 0 & 0.125-0.051\mathrm{i} & 0 & 0.125+0.051\mathrm{i} & 0 & 0.125+0.301\mathrm{i}\end{bmatrix}$
x_3	DFT	$\begin{bmatrix}4 & 0 & 0 & 0 & 4 & 0 & 0 & 0\end{bmatrix}$
X_3	iDFT	$\begin{bmatrix}0.5 & 0 & 0 & 0 & 0.5 & 0 & 0 & 0\end{bmatrix}$

Table 9: DFTs and iDFTs from x_2, x_3 and X_2, X_3 from table 4

Subtask 3: Performing the FFT on the data from `fftdata.csv` yields the following spectrum:

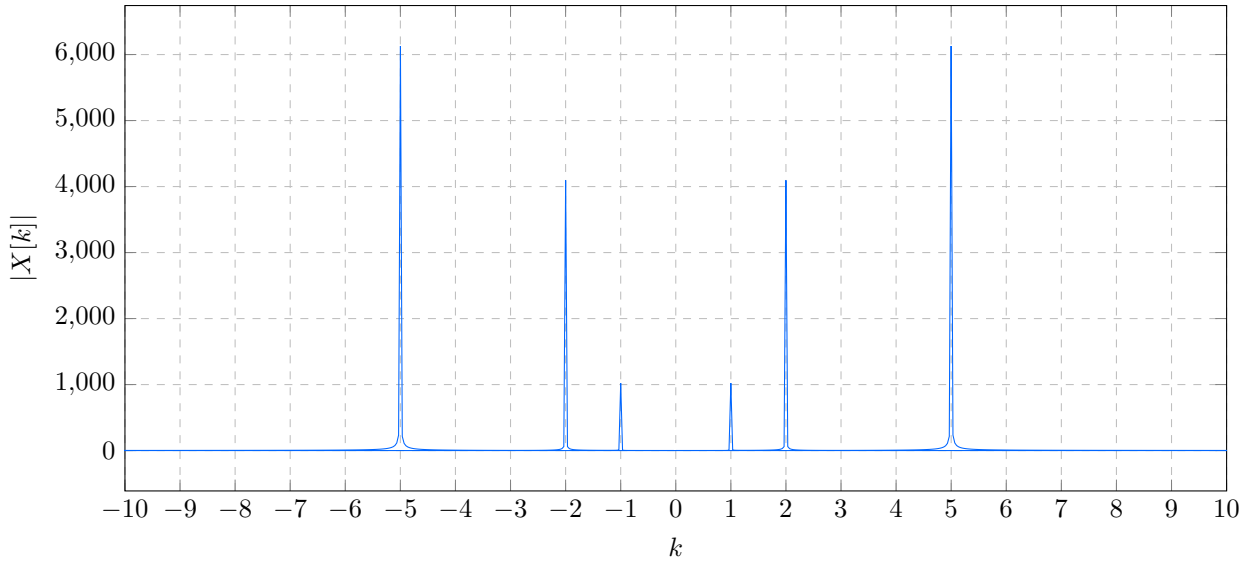

Figure 7: Frequency Spectrum from the signal of fig. 6

We see in fig. 7, that the spectrum is symmetric. This is not surprising, as the given input signal is real-valued. If the signal were complex-value, an asymmetric spectrum could have occurred. Notice that there is no peak at 0, as the signal has no vertical offset, i.e. $\mathrm{avg}_T\, x = 0$, where T denotes one period of x. Reading the peaks from fig. 7 yields the following representation for a possible reconstructed signal:

$$x_r(t) = \sum_{k\in\{1,2,5\}} |X[k]|\cos(2k\pi t + \arg(X[k])) + |X[-k]|\cos(-2k\pi t + \arg(X[-k]))$$

Finding the peaks is possible in python as well, using the function `find_peaks` from the `scipy.signal` module. We can specify a height-threshold H for a minimum peak-height and `find_peaks` will return the indices of all peaks above the given threshold. Notice that we need a separate frequency axis of the same dimension as X. The frequency axis $\boldsymbol{f}$ is symmetric, by design, around 0, and can be obtained via the function `fftfreq`[5] from the `scipy.fftpack` module. Since we know the original time-steps of the sampling of x, we can give `fftfreq` a value for the optional time-step `d`. Let $\boldsymbol{t}$ denote the time-instances, then we set $d = \boldsymbol{t}[1] - \boldsymbol{t}[0]$. Using the indices from `find_peaks`, we can read the corresponding frequencies from $\boldsymbol{f}$. The following snippet first computes the FFT X and then reconstructs x as x_r using the found peaks.

```
data = genfromtxt('fftdata.dat')
k = data[:,0]
x = data[:,1]
X = FFT(x)
N = len(x)
f = fftfreq(N, d=k[1]-k[0])
peaks, props = find_peaks(abs(X), height=100)
t = np.linspace(x[0], 5, 5000)
xr = 1/N * sum([abs(X[k])*np.cos(2*pi*f[k]*t + angle(X[k])) for k in peaks],axis=0)
```

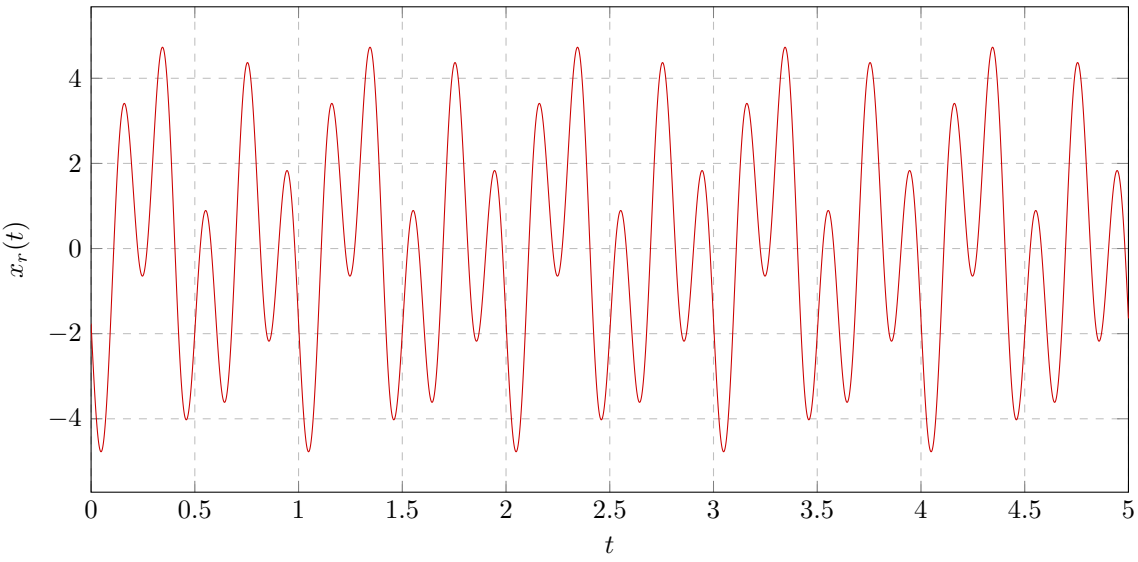

Figure 8: Reconstructed signal from Subtask 3

[5] see the documentation

7.3 Fast Fourier Transform II

Task 7.3: Trigonometric Interpolation

1. Let $h = f + \mathrm{i}g$, where $f, g\colon \mathbb{R} \to \mathbb{R}$. Let $h \overset{\text{DFT}}{\longleftrightarrow} H$. Show that the DFTs of f and g are given by
$$F[k] = \frac{1}{2}(H[k] + \overline{H[N-k]}) = \mathfrak{Re}(H[k])$$
$$G[k] = \frac{\mathrm{i}}{2}(\overline{H[n-k]} - H[k]) = \mathfrak{Im}(H[k])$$
2. Compute the trigonometric interpolating polynomial of degree 4 on the interval $[0, 2]$ for the data defined by:
$$\left\{\left(\frac{j}{4}, f\left(\frac{j}{4}\right)\right)\right\}_{j=0}^{7} \qquad f(x) = x^4 - 3x^3 + 2x^2 - \tan\left(x^2 - 2x\right)$$
3. Write a python program that uses the function `FFT` to compute the trigonometric interpolant of degrees 8,16,32 and 64 for the function $f(x) = x^2 \cos x$ on the interval $[-\pi, \pi]$. Compute the interpolation error $\|f - p_n\|_2$ and discuss what happens to the error as the interpolant degree increases. Provide a visualization of your results.

Subtask 1: Let $h \overset{\text{DFT}}{\longleftrightarrow} H$ and $h = f + \mathrm{i}g$, then:

$$\begin{aligned}
H[k] + \overline{H[N-k]} &= \sum_{n=0}^{N-1} h[n]\zeta_N^{-kn} + \overline{\sum_{n=0}^{N-1} h[n]\zeta_N^{-(N-k)n}} = \sum_{n=0}^{N-1} h[n]\zeta_N^{-kn} + \sum_{n=0}^{N-1} \overline{h[n]}\,\overline{\zeta_N^{-(N-k)n}} \\
&= \sum_{n=0}^{N-1} h[n]\zeta_N^{-kn} + \sum_{n=0}^{N-1} \overline{h[n]}\zeta_N^{(N-k)n} = \sum_{n=0}^{N-1} h[n]\zeta_N^{-kn} + \sum_{n=0}^{N-1} \overline{h[n]}\zeta_N^{Nn}\zeta_N^{-kn} \\
&= \sum_{n=0}^{N-1} h[n]\zeta_N^{-kn} + \sum_{n=0}^{N-1} \overline{h[n]}\zeta_N^{-kn} = \sum_{n=0}^{N-1} \Big(\underbrace{h[n] + \overline{h[n]}}_{=f[n]+\mathrm{i}g[n]+f[n]-\mathrm{i}g[n]} \Big)\zeta_N^{-kn} \\
&= \sum_{n=0}^{N-1} 2f[n]\zeta_N^{-kn} = 2F[k] \Rightarrow \frac{1}{2}(H[k] + \overline{H[N-k]}) = F[k]
\end{aligned}$$

Similarly for $G[k]$:

$$\begin{aligned}
H[k] - \overline{H[N-k]} &= \sum_{n=0}^{N-1} h[n]\zeta_N^{-kn} - \overline{\sum_{n=0}^{N-1} h[n]\zeta_N^{-(N-k)n}} = \sum_{n=0}^{N-1} h[n]\zeta_N^{-kn} - \sum_{n=0}^{N-1} \overline{h[n]}\,\overline{\zeta_N^{-(N-k)n}} \\
&= \sum_{n=0}^{N-1} h[n]\zeta_N^{-kn} - \sum_{n=0}^{N-1} \overline{h[n]}\zeta_N^{(N-k)n} = \sum_{n=0}^{N-1} h[n]\zeta_N^{-kn} - \sum_{n=0}^{N-1} \overline{h[n]}\zeta_N^{Nn}\zeta_N^{-kn} \\
&= \sum_{n=0}^{N-1} h[n]\zeta_N^{-kn} - \sum_{n=0}^{N-1} \overline{h[n]}\zeta_N^{-kn} = \sum_{n=0}^{N-1} \Big(\underbrace{h[n] - \overline{h[n]}}_{=f[n]+\mathrm{i}g[n]-f[n]+\mathrm{i}g[n]} \Big)\zeta_N^{-kn} \\
&= \sum_{n=0}^{N-1} 2\mathrm{i}g[n]\zeta_N^{-kn} = 2\mathrm{i}G[k] \Rightarrow \frac{1}{2\mathrm{i}}(H[k] - \overline{H[N-k]}) = G[k]
\end{aligned}$$

Subtask 2:

The trigonometric interpolant of degree p over a given Interval $[a, b]$ is defined as

$$p(x) = X[0] + 2\sum_{k=1}^{p} \mathfrak{Re}(X[k]) \cos\left(2\pi \boldsymbol{f}_k(x-a)\right) - \mathfrak{Im}(X[k]) \sin\left(2\pi \boldsymbol{f}_k(x-a)\right)$$

Where X is DFT of $x = f(x_i)_{i=0}^{2p+1}$, where $x_i = a + i\frac{(b-a)}{2p+1}$, and $\boldsymbol{f}$ is the correct frequency axis-vector for X.

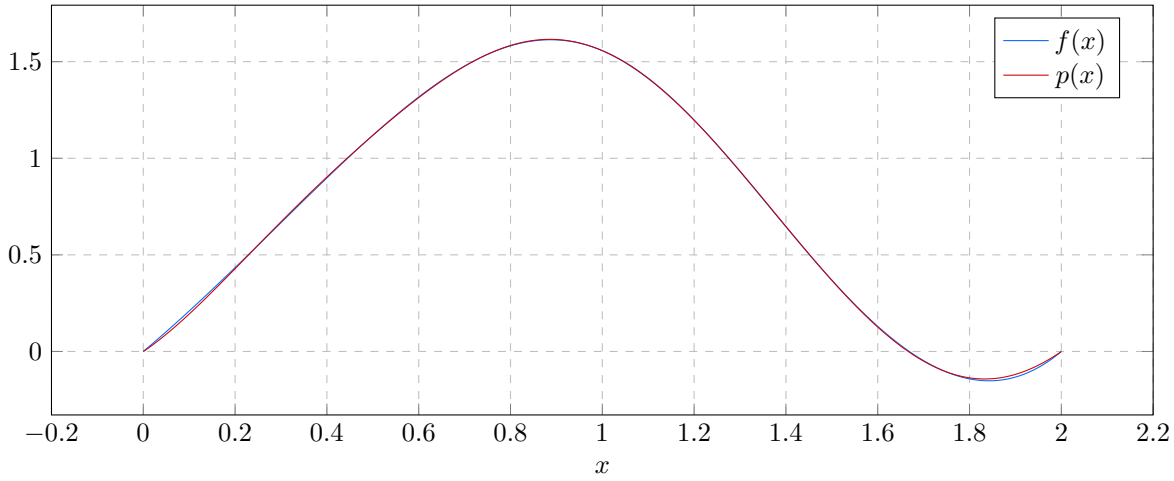

Figure 9: Trigonometric interpolant of degree 4

The data-generation of fig. 9 can be found in the submitted jupyter-notebook `ex_7_3_2.ipynb`.

Subtask 3:

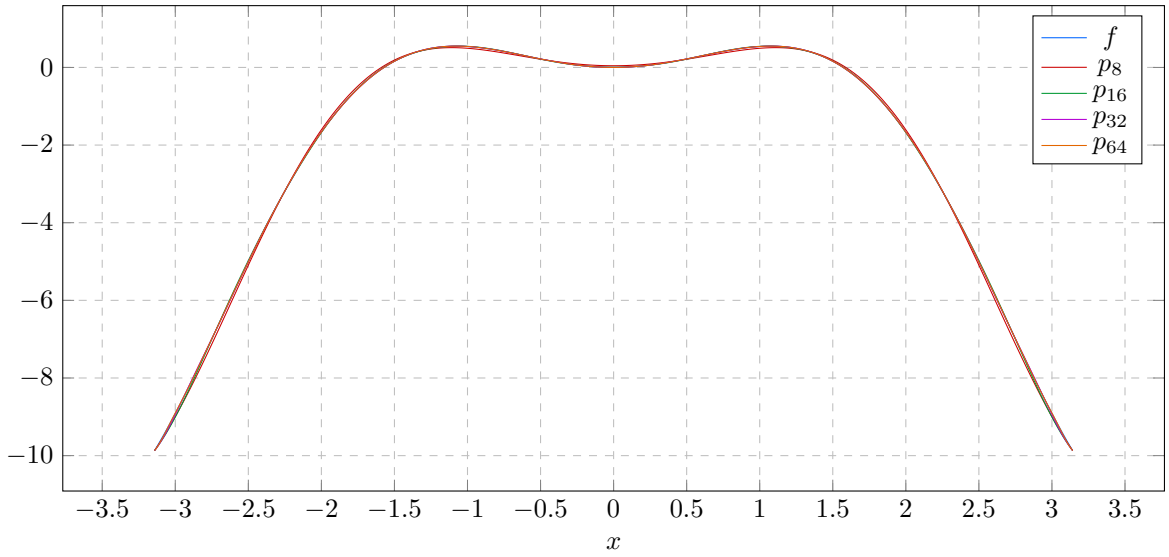

Figure 10: Plot of f corresponding trigonometric interpolants

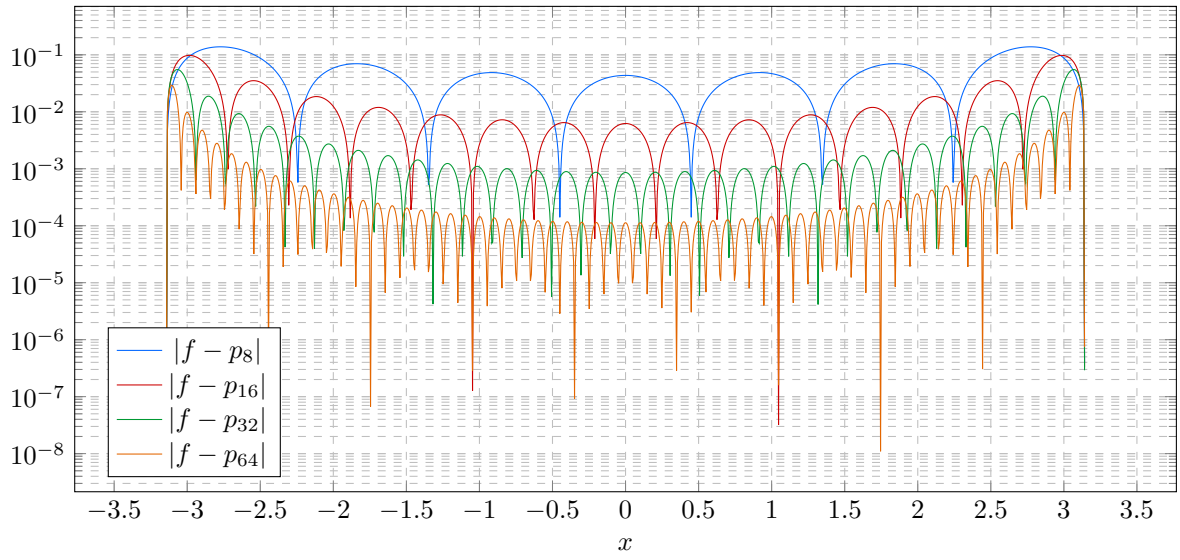

Figure 11: Absolute error $|f - p_k|$

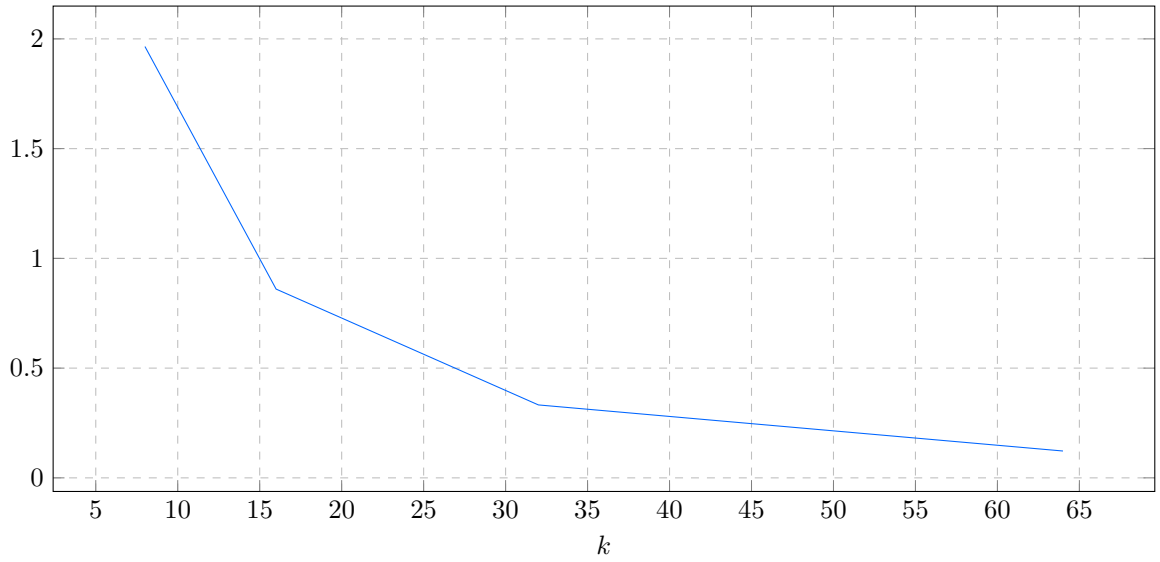

Figure 12: Evolution of $\|f - p_k\|_2$

The data-generation of figs. 10 to 12 can be found in the submitted jupyter-notebook `ex_7_3_3.ipynb`. The implementation of the real trigonometric interpolation can be found in the submitted file `trig_inter.py`.

7.4 Newton-Cotes Quadrature

Task 7.4: Closed Newton-Cotes Quadratures

1. Using the trapezoidal and Simpson's $\frac{1}{3}$ rule, compute the integral $\int_0^2 f(x)\,\mathrm{d}x$ by hand for
   a $f(x) = x^2$
   b $f(x) = \frac{1}{1+x}$
2. Prove that for any quadrature rule that is exact of at least order 1, the quadrature weights w_j satisfy:
   a $\sum_{i=0}^{n} w_i = 1$
   b $w_{n-i} = w_i$ for $j = 0, \dots, n$
3. Derive the summed versions of the Newton-Cotes rules for $n = 1, 2, 3$ based on Hermite-Interpolation and their corresponding asymptotic accuracy.
4. Write a python function that implements the Newton-Cotes formulae for $n = 1, 2, 3, 4$ to approximate the integral $\int_a^b f(x)\,\mathrm{d}x$ of a given function f on the closed interval $[a, b]$. Write another function that implements the composite versions of these Newton-Cotes formulae. Test your script by approximating the integral
$$\int_0^1 \frac{1}{1+x^2}\,\mathrm{d}x = \frac{\pi}{4}$$
using the four composite Newton-Cotes rules.
5. In this sub-task, we compare the performance of Simpson's and Newton's $\frac{3}{8}$ rules. Note that both have the same exactness and the same asymptotic accuracy. Using Lagrange-Interpolation, derive the asymptotic accuracy of both rules and compare their error constants. Use your python script from Subtask 4 to approximate the integral
$$\int_0^1 \frac{1}{1+x^2}\,\mathrm{d}x = \frac{\pi}{4}$$
using composite Simpson's and Newton's $\frac{3}{8}$ rules, and provide a visualization of the errors for $h = 2^{-k}$, where $k = 1, 2, 3, 4, 5, 6$. Discuss your observations.

Subtask 1:

Since we integrate over the same integral, we can determine h_1 and h_2 beforehand.

$$h_1 = 2 \qquad h_2 = 1 \qquad x_1 = 1$$

Sub-Subtask a:

$$\mathcal{I}_1(f) = \frac{2}{2}\left(0^2 + 2^2\right) = 4 \qquad \mathcal{I}_2(f) = \frac{1}{3}\left(0^2 + 4 \cdot 1^2 + 2^2\right) = \frac{8}{3}$$

Sub-Subtask b:

$$\mathcal{I}_1(f) = \frac{2}{2}\left(\frac{1}{1+0} + \frac{1}{1+2}\right) = \frac{4}{3} \qquad \mathcal{I}_2(f) = \frac{1}{3}\left(1 + \frac{4}{2} + \frac{1}{3}\right) = \frac{10}{9}$$

Subtask 2, Sub-Subtask a:

$$\sum_{i=0}^{n} w_i = \sum_{i=0}^{n} \frac{1}{b-a}\int_a^b \ell_i(x)\,\mathrm{d}x = \frac{1}{b-a}\int_a^b \sum_{i=0}^{n} \ell_i(x)\,\mathrm{d}x = \frac{1}{b-a}\int_a^b \mathrm{d}x = 1$$

Sub-Subtask b: The lagrange polynomials are symmetric with respect to $\chi = \frac{a+b}{2} = \frac{x_i + x_{n-i}}{2}$, i.e. $\ell_i(x) = \ell_{n-i}(2\chi - x)$. We introduce $u = 2\chi - x$, thus $\mathrm{d}u = -\mathrm{d}x$:

$$\int_a^b \ell_{n-i}(x)\,\mathrm{d}x = -\int_{2\chi-a}^{2\chi-b} \ell_{n-i}(2\chi - x)\,\mathrm{d}x = -\int_{2\chi-a}^{2\chi-b} \ell_i(x)\,\mathrm{d}x = -\int_b^a \ell_i(x)\,\mathrm{d}x = \int_a^b \ell_i(x)\,\mathrm{d}x$$

Subtask 3

Given eq. (5), we get:

$$\begin{aligned}\mathcal{I}_1^N(f) &= \sum_{j=0}^{N-1} \mathcal{I}_n(f,[x_j,x_{j+1}]) = \sum_{j=0}^{N-1} \frac{h}{2}(f(x_j)+f(x_{j+1})) \\ &= \frac{h}{2}\left(f(a)+f(b)+2\sum_{j=1}^{N-1} f(x_j)\right)\end{aligned}$$

Using the known error-bounds of $\mathcal{I}_1$, we get:

$$|\mathcal{I}(f)-\mathcal{I}_1^N(f)| \leq \sum_{j=0}^{N-1} \frac{h^3}{12}\|f''\|_\infty = \frac{h^3}{12}\|f''\|_\infty N = \frac{h^2}{12}\|f''\|_\infty (b-a)$$

From eq. (13):

$$\begin{aligned}\mathcal{I}_2^N(f) &= \sum_{j=0}^{\frac{N}{2}-1} \frac{h}{3}(f(x_{2j})+4f(x_{2j+1})+f(x_{2j+2})) \\ &= \frac{h}{3}\left(f(a)+f(b)+2\sum_{j=1}^{\frac{N}{2}-1} f(x_{2j})+4\sum_{j=1}^{\frac{N}{2}} f(x_{2j-1})\right)\end{aligned}$$

With the bound from Subtask 5, we get:

$$\begin{aligned}|\mathcal{I}(f)-\mathcal{I}_2^N(f)| &\leq \sum_{j=0}^{\frac{N}{2}-1} |\mathcal{I}_2(f,[x_{2j},x_{2j+2}]) - \mathcal{I}(f,[x_{2j},x_{2j+1}])| \\ &\leq \sum_{j=0}^{\frac{N}{2}-1} \left(\frac{b-a}{\frac{N}{2}}\right)^5 \frac{1}{2880}\|f^{(4)}\|_\infty \\ &= \frac{N}{5760}\left(\frac{2(b-a)}{N}\right)^5 = \frac{32N\|f^{(4)}\|_\infty}{5760}h^5 = \frac{h^4\|f^{(4)}\|_\infty}{180}(b-a)\end{aligned}$$

From eq. (14):

$$\begin{aligned}\mathcal{I}_3^N(f) &= \sum_{j=0}^{\frac{N}{3}-1} \frac{3h}{8}(f(x_{3j})+3f(x_{3j+1})+3f(x_{3j+2})+f(x_{3j+3})) \\ &= \frac{3h}{8}\left(f(a)+f(b)+2\sum_{j=1}^{\frac{N}{3}-1} f(x_{3j})+3\sum_{j=0}^{\frac{N}{3}} f(x_{3j-1})+f(x_{3j-2})\right)\end{aligned}$$

Again, using the bound from Subtask 5:

$$\begin{aligned}|\mathcal{I}(f)-\mathcal{I}_3^N(f)| &\leq \sum_{j=0}^{\frac{N}{3}-1} |\mathcal{I}_2(f,[x_{3j},x_{3j+3}]) - \mathcal{I}(f,[x_{3j},x_{3j+3}])| \\ &\leq \sum_{j=0}^{\frac{N}{3}-1} \left(\frac{b-a}{\frac{N}{3}}\right)^5 \frac{\|f^{(4)}\|_\infty}{6480} = \frac{N}{19440}\left(\frac{3(b-a)}{N}\right)^5 \|f^{(4)}\|_\infty \\ &= N\frac{h^5\|f^{(4)}\|_\infty}{80} = (b-a)h^4\frac{\|f^{(4)}\|_\infty}{80}\end{aligned}$$

Subtask 4: See the submitted jupyter-notebook `7_4_4.ipynb` for the data-generation and `ncotes.py` for the implementation of the quadratures.

Subtask 5

We simplify our observations to the interval $[0, h]$:

$$\begin{aligned}
|\mathcal{I}(f) - \mathcal{I}_2(f)| = |\mathcal{I}(f) - \mathcal{I}(p_2)| &= \left| \int_0^h f(x) - p_2(x)\, \mathrm{d}x \right| \\
&= \left| \int_0^h \frac{f'''(\xi)}{6} x(x-h)\left(x - \frac{h}{2}\right)\, \mathrm{d}x \right| \leq \frac{\|f'''\|_\infty}{6} \left| \int_0^h (x^2 - xh)\left(x - \frac{h}{2}\right)\, \mathrm{d}x \right| \\
&= \frac{\|f'''\|_\infty}{6} \left| \int_0^h x^3 - \frac{3x^2 h}{2} + x\frac{h^2}{2}\, \mathrm{d}x \right| = \frac{\|f'''\|_\infty}{6} \left| \frac{h^4}{4} - \frac{h^4}{2} + \frac{h^4}{4} \right| = 0
\end{aligned}$$

Notice that using Lagrange-Interpolation for $\mathcal{I}_2$ produces 0 when searching for a bound of the error. Thus we use Hermite-Interpolation instead and get the following result:

$$\begin{aligned}
|\mathcal{I}(f) - \mathcal{I}_2(f)| = |\mathcal{I}(f) - \mathcal{I}(H_2)| &= \left| \int_0^h f(x) - H_2(x)\, \mathrm{d}x \right| \\
&= \left| \int_0^h x(x-h)\left(x - \frac{h}{2}\right)^2 \frac{f^{(4)}(\xi)}{24}\, \mathrm{d}x \right| \leq \frac{\|f^{(4)}\|_\infty}{24} \left| \int_0^h x^4 - 2hx^3 + \frac{5h^2x^2}{4} - \frac{h^3 x}{4}\, \mathrm{d}x \right| \\
&= \frac{\|f^{(4)}\|_\infty}{24} \cdot \frac{h^5}{120} = \frac{h^5 \|f^{(4)}\|_\infty}{2880}
\end{aligned}$$

For Newton's $\frac{3}{8}$ rule we get:

$$\begin{aligned}
|\mathcal{I}(f) - \mathcal{I}_3(f)| = |\mathcal{I}(f) - \mathcal{I}(p_3)| &= \left| \int_0^h f(x) - p_3(x)\, \mathrm{d}x \right| \\
&= \left| \int_0^h \frac{f^{(4)}(\xi)}{24} x(x-h)\left(x - \frac{h}{3}\right)\left(x - \frac{2h}{3}\right)\, \mathrm{d}x \right| \\
&\leq \frac{\|f^{(4)}\|_\infty}{24} \left| \frac{x^5}{5} - \frac{hx^4}{2} + \frac{11h^2x^3}{27} - \frac{h^3x^2}{9} \Big|_0^h \right| = \frac{\|f^{(4)}\|_\infty}{24} \cdot \frac{h^5}{270} \\
&= \frac{h^5 \|f^{(4)}\|_\infty}{6480}
\end{aligned}$$

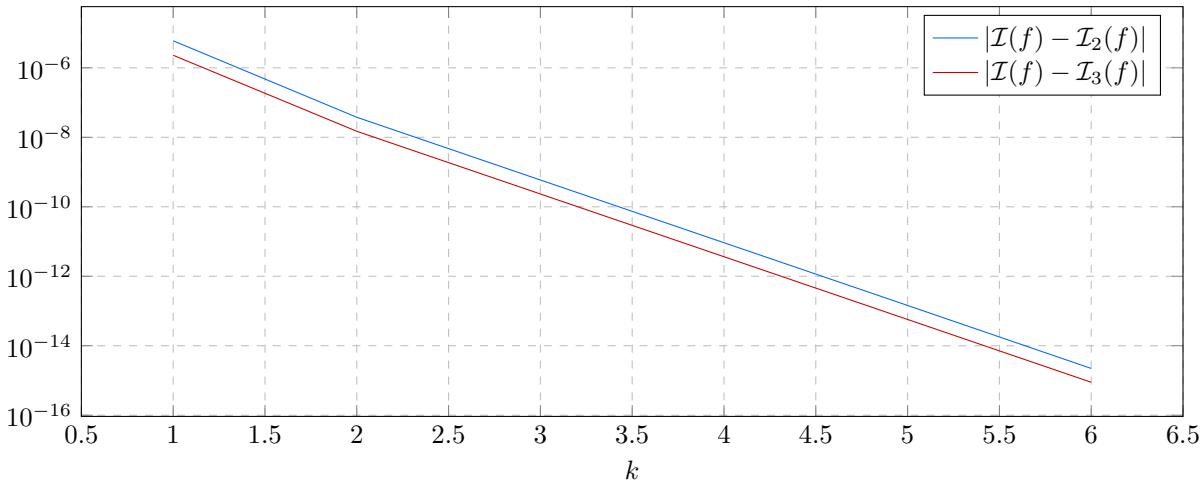

Figure 13: Error evolution of $\mathcal{I}_2$ and $\mathcal{I}_3$

Since both $\mathcal{I}_2$ and $\mathcal{I}_3$ have the same asymptotic error, the only notable difference is the constant. From our computations above, we get that Newton's $\frac{3}{8}$ is „a tiny bit“ better than Simpson's. See `7_4_5.ipynb` for data-generation.

7.5 Gauss-Legendre Quadrature

Task 7.5: Gauss-Legendre Quadrature

1. Derive the following Gaussian Quadrature formula for $n = 2$:
$$\int_{-1}^{1} f(x)\,\mathrm{d}x = \frac{5}{9} f\left(-\sqrt{\frac{3}{5}}\right) + \frac{8}{9} f(0) + \frac{5}{9} f\left(\sqrt{\frac{3}{5}}\right)$$
and show that it is exact for any polynomial up to degree five.
2. Using the three-point Gaussian quadrature formula, evaluate
$$\int_0^1 \frac{1}{1+x}\,\mathrm{d}x$$
3. Write a python program that approximates $\mathcal{I}_I(f)$ given the function $f\colon [a,b] \to \mathbb{R}$, where $I = [a,b]$, using Gauss-Legendre quadrature of order 2, $\widetilde{\mathcal{G}}_{2,I}(f)$. Test your script by approximating
$$\int_1^3 x^6 - x^2 \sin(2x)\,\mathrm{d}x \approx 317.3442466$$

Subtask 1: We already derived the corresponding nodes and weights in section 7.0.3. We prove exactness for general $\mathcal{G}_n$. Since we are given $n+1$ nodes, we get $2n+2$ conditions to find the Hermite-interpolant $H \in \mathbb{R}_{2n+1}[X]$:

$$H(x_i) = f(x_i) \qquad H'(x_i) = f'(x_i)$$

Notice that H is of the following form:

$$H(x) = \sum_{i=0}^{n} f(x_i)\eta_i(x) + \sum_{i=0}^{n} f'(x_i)\kappa_i(x)$$

where:

$$\begin{aligned} \eta_i(x_j) &= \delta_{ij} & \eta_i'(x_j) &= 0 \\ \kappa_i(x_j) &= 0 & \kappa_i'(x_j) &= \delta_{ij} \end{aligned}$$

Both η_i and κ_i can be expressed using the Lagrange-polynomials ℓ_{jn}:

$$\eta_i(x) = \ell_{in}^2(x)(1 - 2\ell_{in}'(x_i)(x - x_i)) \qquad \kappa_i(x) = \ell_{in}^2(x)(x - x_i)$$

Let $\omega_n(x) = \prod_{i=0}^{n}(x - x_i)$, then:

$$\int_{-1}^{1} \eta_i(x)\,\mathrm{d}x = \int_{-1}^{1} \ell_{in}^2(x)\,\mathrm{d}x \qquad \int_a^b \kappa_i(x)\,\mathrm{d}x = 0$$

Let $q(x) = \ell_{in}^2(x) - \ell_{in}(x) \in \mathbb{R}_{2n}[x]$. Note that $q(x_i) = 0$, thus $\frac{q}{\omega_n} = r \in \mathbb{R}_{n-1}[x]$. Notice $\langle \omega_n, r \rangle = 0$ by the orthogonality of polynomials of L_k, thus:

$$\begin{aligned} 0 &= \int_{-1}^{1} \omega_n(x) r(x)\,\mathrm{d}x = \int_{-1}^{1} \ell_{in}^2(x) - \ell_{in}(x)\,\mathrm{d}x \\ \Rightarrow \int_{-1}^{1} \ell_{in}^2(x)\,\mathrm{d}x &= \int_{-1}^{1} \ell_{in}(x)\,\mathrm{d}x = w_i \end{aligned}$$

Since the Lagrange-Polynomials L_k form a basis of $\mathbb{R}_n[x]$, we can represent r as a a linear combination of $L_0, \ldots, L_{n-1}$, thus:

$$\int_{-1}^{1} \omega_n(x) r(x)\,\mathrm{d}x = \int_{-1}^{1} \omega_n(x) \sum_{i=0}^{n-1} \alpha_i L_i(x)\,\mathrm{d}x = \sum_{i=0}^{n-1} \alpha_i \int_{-1}^{1} \omega_n(x) L_i(x)\,\mathrm{d}x$$

Notice that $\omega_n(x) = KL_{n+1}(x)$ for some $K \in \mathbb{R}$. By the orthogonality $\langle T_n, T_m \rangle = \frac{2}{2n+1}\delta_{mn}$, we get that $\langle \omega_n, r \rangle = 0$. Therefore:

$$\int_{-1}^{1} H(x)\,\mathrm{d}x = \sum_{i=0}^{n} f(x_i)w_i = \mathcal{G}_n(f)$$

Computing the error $\mathcal{E}_n(f) = \mathcal{I}(f) - \mathcal{G}_n(f)$ and using the interpolation error[6] from Hermite-interpolation yields:

$$\mathcal{E}_n(f) = \int_{-1}^{1} f(x) - H(x)\,\mathrm{d}x = \int_{-1}^{1} \omega_n^2(x)\frac{f^{(2n+2)}(\xi)}{(2n+2)!}\,\mathrm{d}x$$

For $f = p \in \mathbb{R}_{2n+1}$ we get $f^{(2n+2)} = 0$, therefore $\mathcal{E}_n(p) = 0$, i.e. $\mathcal{I}(p) = \mathcal{G}_n(p)$. However for $p \in \mathbb{R}_{2n+2}$, we get $p^{(2n+2)}(x) = (2n+2)! \cdot a_{2n+2}$. Thus for $n = 2$, $\mathcal{G}_n$ is exact for polynomials up to degree 5.

Subtask 2:

Since we do not integrate over $[-1, 1]$, we need to use the transformed Gauss-Legendre quadrature $\widetilde{\mathcal{G}}_{2,I}(f)$, where $I = [0, 1]$. Thus:

$$\widetilde{\mathcal{G}}_{2,I}(f) = \frac{5}{18}\left(f\left(-\sqrt{\frac{3}{20}}+1\right) + f\left(\sqrt{\frac{3}{20}}+1\right)\right) + \frac{4}{9}f(1) \approx 0.6931216$$

Notice that:

$$\int_0^1 \frac{1}{1+x}\,\mathrm{d}x = \ln(1+x)\Big|_0^1 = \ln(2) \approx 0.693147180$$

Subtask 3:

$$\mathcal{G}_{2,I}(f) = 306.81992$$
$$\mathcal{G}_{3,I}(f) = 317.26414$$

[6] $f(x) - H_n(x) = \omega_n^2(x)\frac{f^{(n+1)}(\xi)}{(n+1)!}$

7.6 Appendix

7.6.1 Proof of the General Product Rule

Theorem 7.1: General Product Rule

Let f_i for $i = 0, \dots, k$ be a family of differentiable functions, then

$$\frac{\mathrm{d}}{\mathrm{d}x} \underbrace{\prod_{i=0}^{k} f_i(x)}_{=f(x)} = \sum_{i=0}^{k} f_i'(x) \prod_{j=0}^{k} \frac{f_j(x)}{f_i(x)} = \sum_{i=0}^{k} f_i'(x) \prod_{\substack{j=0 \\ j\neq i}}^{k} f_j(x) = f(x) \sum_{i=0}^{k} \frac{f_i'(x)}{f_i(x)} \tag{12}$$

Proof for Theorem 7.1. We want to use induction. Let $k = 1$, then we set $f = f_0 f_1$. By the product rule, we get $f' = f_0' f_1 + f_0 f_1'$, which is our desired result. Thus for $k - 1 \to k$ we get:

$$\frac{\mathrm{d}}{\mathrm{d}x} \prod_{i=0}^{k} f_i(x) = \frac{\mathrm{d}}{\mathrm{d}x} f_k(x) \underbrace{\prod_{i=0}^{k-1} f_i(x)}_{=f(x)} = f_k'(x) f(x) + f_k(x) f'(x)$$

$$= f_k'(x) f(x) + f_k(x) \sum_{i=0}^{k-1} f_i'(x) \prod_{\substack{j=0 \\ j\neq i}}^{k-1} f_j(x) = f_k'(x) \prod_{\substack{j=0 \\ j\neq k}}^{k} f_j(x) + \sum_{i=0}^{k-1} f_k(x) f_i'(x) \prod_{\substack{j=0 \\ j\neq i}}^{k-1} f_j(x)$$

$$= f_k'(x) \prod_{\substack{j=0 \\ j\neq k}}^{k} f_j(x) + \sum_{i=0}^{k-1} f_i'(x) \prod_{\substack{j=0 \\ j\neq i}}^{k} f_j(x) = \sum_{i=0}^{k} f_i'(x) \prod_{\substack{j=0 \\ j\neq i}}^{k} f_j(x)$$

□

7.6.2 Iterative FFT

While algorithm 1 gives a nice view of how the algorithm works, it may not be the best choice on devices with little memory available, as for large N, the call stack can get rather large through recursion. Thus an iterative solution would reduce the required space for recursive function calls. While the algorithm used in my submission is indeed this iterative one, I won't provide a full derivation for the pseudocode. We begin by expanding on fig. 2 and adding the recursive steps:

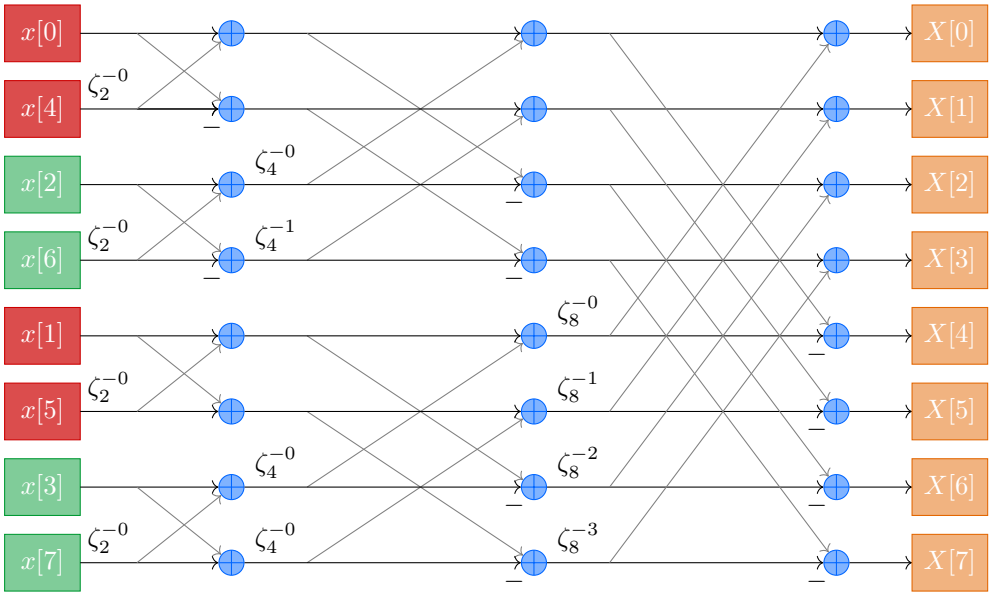

Figure 14: Radix-2 8-point FFT as a signal-flow-graph

The order of the input-values can be determined via bit-reversal, where we read the bits of n from LSB[7] to MSB[8]. See table 10 below for an example.

n	$[n]_2$	$[\widetilde{n}]_2$	$\widetilde{n}$
0	000	000	0
1	001	100	4
2	010	010	2
3	011	110	6
4	100	001	1
5	101	101	5
6	110	011	3
7	111	111	7

Table 10: Bit-reversals of indices $0, \dots, 7$

Let, without loss of generality be $N = 2^p$, then one iterative version of the FFT is given by:

Algorithm 2: Iterative Version of the FFT

```
input: signal x of length N
output: DFT X of length N

FFT(x):
    X = 0_N

    for n = 0, ..., N − 1 do
        X[n] = x[ñ]
    end

    for s = 1, ..., ld(N) do
        m_1 = 2^s
        m_2 = m_1/2 − 1
        μ = 1
        ζ = e^(iπ/m_2)
        for j = 0, ..., m_2 do
            for k = j, j + m, j + 2, ..., N do
                t = X[k + m_2]
                u = μX[k]

                X[k] = u + t
                X[k + m_2] = u − t
            end
            μ = ζμ
        end
    end
    return X
```

7.6.3 Closed Expressions for the Newton-Cotes Formulae

In section 7.0.2, I already computed the closed expression $\mathcal{I}_1(f)$. Here I want to find closed expressions for $\mathcal{I}_2$, $\mathcal{I}_3$ and $\mathcal{I}_4$. To simplify computations, we assume that

$$\mathcal{I}_n(f) = (b-a)\sum_{i=0}^{n} \widetilde{w}_i f(x_i)$$

thus the computations of the weights become:

$$\widetilde{w}_i = \frac{1}{b-a}\int_a^b \ell_i(x)\,\mathrm{d}x$$

As a shortcut, we introduce $I_j = \int_a^b \ell_j(x)\,\mathrm{d}x$.

We start with $\mathcal{I}_2$, thus we have three equidistant x_i, with $x_0 = a$, $x_1 = \frac{a+b}{2}$ and $x_2 = b$. Furthermore $h = \frac{b-a}{2}$. Since the Newton-Cotes formulae are exact, we can exploit $w_i = w_{n-i}$.

[7] abbr. LSB = least significant bit
[8] abbr. MSB = most significant bit

$$\ell_0(x) = \frac{(x-x_1)(x-b)}{(a-x_1)(a-b)} = \frac{x^2 - x(x_1+b) + x_1 b}{(a-x_1)(a-b)}$$
$$I_0 = \frac{1}{(a-x_1)(a-b)}\left(\frac{x^3}{3} - \frac{x^2}{2}(x_1+b) + x_1 bx\Big|_a^b\right)$$
$$= \frac{(b-a)}{6} \Rightarrow w_0 = w_2 = \frac{1}{6}$$
$$w_1 = 1 - 2w_0 = \frac{4}{6}$$

Thus:

$$\mathcal{I}_2(f) = \frac{b-a}{6}\left(f(a) + 4f(x_1) + f(b)\right) = \frac{h}{3}\big(f(a) + 4f(x_1) + f(b)\big) \tag{13}$$

For $\mathcal{I}_3$ we have four equidistant x_i, with $x_0 = a$ and $x_3 = b$. Furthermore, we get $h = \frac{b-a}{3}$, therefore $x_1 = \frac{2a+b}{3}$ and $x_2 = \frac{2b+a}{3}$. We need again only need to compute I_0, since $w_3 = w_0$, $w_1 = w_2$ and $2w_0 + 2w_1 = 1$:

$$\ell_0(x) = \frac{(x-x_1)(x-x_2)(x-b)}{(a-x_1)(a-x_2)(a-b)} = \frac{(x^2 - xx_2 - xx_1 + x_1x_2)(x-b)}{(a-x_1)(a-x_2)(a-b)}$$
$$= \frac{x^3 - x^2(x_1+x_2) + xx_1x_2 - bx^2 + xbx_1 + xbx_2 - bx_1x_2}{(a-x_1)(a-x_2)(a-b)}$$
$$= \frac{x^3 - x^2(x_1+x_2+b) + x(x_1x_2 + bx_1 + bx_2) - bx_1x_2}{(a-x_1)(a-x_2)(a-b)}$$

Integrating $\ell_0(x)$ over $[a,b]$ yields:

$$I_0 = \frac{1}{(a-x_1)(x_2-a)(b-a)}\left(\frac{x^4}{4} - \frac{x^3}{3}(x_1+x_2+b) + \frac{x^2}{2}(x_1x_2 + bx_1 + bx_2) - xbx_1x_2\Big|_a^b\right)$$
$$= \frac{(b-a)}{8} \Rightarrow w_0 = w_3 = \frac{1}{8}$$
$$2w_1 = 1 - w_0 = \frac{6}{8} \Rightarrow w_1 = w_2 = \frac{3}{8}$$

Therefore:

$$\mathcal{I}_3(f) = \frac{b-a}{8}\big(f(a) + 3f(x_1) + 3f(x_2) + f(b)\big) = \frac{3h}{8}\big(f(a) + 3f(x_1) + 3f(x_2) + f(b)\big) \tag{14}$$

For $\mathcal{I}_4$ I opted to use the following python-program, which utilizes the `sympy` package. A symbolic library for python.

```python
import numpy as np
import sympy as sp

a, b, x = sp.symbols('a,b,x')

def ljn(x_data, j):
    xj = x_data[j]
    xtmp = np.delete(x_data, j)
    return sp.simplify(np.prod((x-xtmp) / (xj - xtmp)))


def wj(x0, xn, j, n):
    h = (xn - x0) / n
    x_data = [x0 + k*h for k in range(n+1)]
    Ljn = sp.integrate(ljn(x_data, j), x)
    H = x_data[-1] - x_data[0]
    return sp.simplify((Ljn.subs({x: x_data[-1]}) - Ljn.subs({x: x_data[0]})) / H )
```

Notice that the program above is not very efficient, as the computation of the five weights already took almost 2 seconds. Given the weights w_0, w_1, w_2, w_3, w_4 from table 1, we get:

$$\begin{aligned}\mathcal{I}_4(f) &= \frac{(b-a)}{90}\big(7f(a) + 32f(x_1) + 12f(x_2) + 32f(x_3) + 7f(b)\big) \\ &= \frac{2h}{45}\big(7f(a) + 32f(x_1) + 12f(x_2) + 32f(x_3) + 7f(b)\big)\end{aligned} \tag{15}$$

7.6.4 Transformation for Gauss-Legendre quadrature

Given the Gauss-Legendre quadrature is only defined for integrals over $[-1, 1]$, we need to transform integrals over $[a, b]$ with the substitution rule. We need a function $u\colon [a, b] \to [-1, 1]$, such that:

$$\int_{-1}^{1} f(x)\,\mathrm{d}x = \int_{u^{-1}(-1)}^{u^{-1}(1)} f(u)\,\mathrm{d}u$$

Choosing $u = \operatorname{lerp}(a, b)$ we get

$$u(x) = \frac{b-a}{2}x + \frac{b+a}{2}$$

Definition 7.8: Transformed Gauss-Legendre Quadrature

Let f be the function whose integral over $I = [a, b]$ we want to approximate, then we define the transformed Gauss-Legendre quadrature $\widetilde{\mathcal{G}}_{n,I}(f)$ as:

$$\widetilde{\mathcal{G}}_{n,I}(f) = \frac{b-a}{2}\sum_{i=0}^{n} w_i f\left(\frac{b-a}{2}x_i + \frac{b+a}{2}\right)$$

7.6.5 Alternative Gauss-Legendre Weights

Let $L_n(x)$ be the n-th Legendre-polynomial with zeros $x_0, \ldots, x_n$, then:

$$L_n(x) = K\prod_{i=0}^{n}(x - x_i)$$

By theorem 7.1 we get:

$$\begin{aligned}L_n'(x) &= K\sum_{i=0}^{n}\prod_{\substack{j=0\\ j\neq i}}^{n}(x - x_j) \\ \Rightarrow L_n'(x_l) &= K\prod_{\substack{j=0\\ j\neq l}}^{n}(x_l - x_j)\end{aligned}$$

Furthermore:

$$\frac{L_n(x)}{x - x_i} = K\prod_{\substack{j=0\\ j\neq i}}^{n}(x - x_j)$$

Therefore we get from eq. (8):

$$\frac{L_n(x)}{(x - x_i)L_n'(x_i)} = \frac{K\prod_{\substack{j=0\\ j\neq i}}^{n}(x - x_j)}{K\prod_{\substack{j=0\\ j\neq i}}^{n}(x_i - x_j)} = \prod_{\substack{j=0\\ j\neq i}}^{n}\frac{x - x_j}{x_i - x_j} = \ell_{in}(x)$$