Chapter 1

1.1

1. Compute the loudness of the Krakatoa explosion on the decibel scale, assuming that the variation in air pressure peaked at 100 000 Pa.
Solution: Setting p_{ref}=0.00002 Pa and p=100 000 Pa in the decibel expression we get

$$\begin{aligned}20\log_{10}\left(\frac{p}{p_{\text{ref}}}\right) &= 20\log_{10}\left(\frac{100000}{0.00002}\right) = 20\log_{10}\left(\frac{10^5}{2\times 10^{-5}}\right)\\ &= 20\log_{10}\left(\frac{10^{10}}{2}\right) = 20\left(10-\log_{10}2\right) \approx 194\text{db}.\end{aligned}$$

2. Consider a sum of two pure tones, $f(t) = A_1\sin(2\pi\nu_1 t)+A_2\sin(2\pi\nu_2 t)$. For which values of A_1, A_2, ν_1, ν_2 is f periodic? What is the period of f when it is periodic?
Solution: $\sin(2\pi\nu_1 t)$ has period $1/\nu_1$, while $\sin(2\pi\nu_2 t)$ has period $1/\nu_2$. The period is not unique, however. The first one also has period n/ν_1, and the second also n/ν_2, for any n. The sum is periodic if there exist n_1, n_2 so that $n_1/\nu_1 = n_2\nu_2$, i.e. so that there exists a common period between the two. This common period will also be a period of f. This amounts to that $\nu_1/\nu_2 = n_1/n_2$, i.e. that ν_1/ν_2 is a rational number.

1.2

1. Find a function f which is Riemann-integrable on $[0, T]$, and so that $\int_0^T f(t)^2 dt$ is infinite.

Solution: The function $f(t) = \frac{1}{\sqrt{t}} = t^{-1/2}$ can be used since it has the properties

$$\begin{aligned}\int_0^T f(t)dt &= \lim_{x\to 0+}\int_x^T t^{-1/2}dt = \lim_{x\to 0+}\left[2t^{1/2}\right]_x^T \\ &= \lim_{x\to 0+}(2T^{1/2} - 2x^{1/2}) = 2T^{1/2} \\ \int_0^T f(t)^2 dt &= \lim_{x\to 0+}\int_x^T t^{-1}dt = \lim_{x\to 0+}[\ln t]_x^T \\ &= \ln T - \lim_{x\to 0+}\ln x = \infty.\end{aligned}$$

2. Given the two Fourier spaces V_{N_1,T_1}, V_{N_2,T_2}. Find necessary and sufficient conditions in order for $V_{N_1,T_1} \subset V_{N_2,T_2}$.
Solution: The space V_{N_1,T_1} is spanned by pure tones with frequencies $1/T_1,\ldots,N_1/T_1$, while V_{N_2,T_2} is spanned by pure tones with frequencies $1/T_2,\ldots,N_2/T_2$. We must have that the first set of frequencies is contained in the second. This is achieved if and only if $1/T_1 = k/T_2$ for some integer k, and also $N_1/T_1 \le N_2/T_2$. In other words, T_2/T_1 must be an integer, and $T_2/T_1 \le N_2/N_1$.

3. Prove the second part of Theorem 1.21, i.e. show that if f is antisymmetric about 0 (i.e. $f(-t) = -f(t)$ for all t), then $a_n = 0$, i.e. the Fourier series is actually a sine-series.

4. Find the Fourier series coefficients of the periodic functions with period T defined by being $f(t) = t$, $f(t) = t^2$, and $f(t) = t^3$, on $[0, T]$.
Solution: For $f(t) = t$ we get that $a_0 = \frac{1}{T}\int_0^T t\,dt = \frac{T}{2}$. We also get

$$\begin{aligned}a_n &= \frac{2}{T}\int_0^T t\cos(2\pi nt/T)dt \\ &= \frac{2}{T}\left(\left[\frac{T}{2\pi n}t\sin(2\pi nt/T)\right]_0^T - \frac{T}{2\pi n}\int_0^T \sin(2\pi nt/T)dt\right) = 0 \\ b_n &= \frac{2}{T}\int_0^T t\sin(2\pi nt/T)dt \\ &= \frac{2}{T}\left(\left[-\frac{T}{2\pi n}t\cos(2\pi nt/T)\right]_0^T + \frac{T}{2\pi n}\int_0^T \cos(2\pi nt/T)dt\right) = -\frac{T}{\pi n}.\end{aligned}$$

The Fourier series is thus

$$\frac{T}{2} - \sum_{n\ge 1}\frac{T}{\pi n}\sin(2\pi nt/T).$$

Note that this is almost a sine series, since it has a constant term, but no other cosine terms. If we had subtracted $T/2$ we would have obtained a function which is antisymmetric, and thus a pure sine series.

For $f(t)=t^2$ we get that $a_0=\frac{1}{T}\int_0^T t^2dt=\frac{T^2}{3}$. We also get

$$\begin{aligned}
a_n &= \frac{2}{T}\int_0^T t^2\cos(2\pi nt/T)dt \\
&= \frac{2}{T}\left(\left[\frac{T}{2\pi n}t^2\sin(2\pi nt/T)\right]_0^T - \frac{T}{\pi n}\int_0^T t\sin(2\pi nt/T)dt\right) \\
&= \left(-\frac{T}{\pi n}\right)\left(-\frac{T}{\pi n}\right)=\frac{T^2}{\pi^2n^2} \\
b_n &= \frac{2}{T}\int_0^T t^2\sin(2\pi nt/T)dt \\
&= \frac{2}{T}\left(\left[-\frac{T}{2\pi n}t^2\cos(2\pi nt/T)\right]_0^T + \frac{T}{\pi n}\int_0^T t\cos(2\pi nt/T)dt\right) \\
&= -\frac{T^2}{\pi n}.
\end{aligned}$$

Here we see that we could use the expressions for the Fourier coefficients of $f(t)=t$ to save some work. The Fourier series is thus

$$\frac{T^2}{3}+\sum_{n\geq 1}\left(\frac{T^2}{\pi^2n^2}\cos(2\pi nt/T)-\frac{T^2}{\pi n}\sin(2\pi nt/T)\right).$$

For $f(t)=t^3$ we get that $a_0=\frac{1}{T}\int_0^T t^3dt=\frac{T^3}{4}$. We also get

$$\begin{aligned}
a_n &= \frac{2}{T}\int_0^T t^3\cos(2\pi nt/T)dt \\
&= \frac{2}{T}\left(\left[\frac{T}{2\pi n}t^3\sin(2\pi nt/T)\right]_0^T - \frac{3T}{2\pi n}\int_0^T t^2\sin(2\pi nt/T)dt\right) \\
&= \left(-\frac{3T}{2\pi n}\right)\left(-\frac{T^2}{\pi n}\right)=\frac{3T^3}{2\pi^2n^2} \\
b_n &= \frac{2}{T}\int_0^T t^3\sin(2\pi nt/T)dt \\
&= \frac{2}{T}\left(\left[-\frac{T}{2\pi n}t^3\cos(2\pi nt/T)\right]_0^T + \frac{3T}{2\pi n}\int_0^T t^2\cos(2\pi nt/T)dt\right) \\
&= -\frac{T^3}{\pi n}+\frac{3T}{2\pi n}\frac{T^2}{\pi^2n^2}=-\frac{T^3}{\pi n}+\frac{3T^3}{2\pi^3n^3}.
\end{aligned}$$

Also here we saved some work, by reusing the expressions for the Fourier coefficients of $f(t)=t^2$. The Fourier series is thus

$$\frac{T^3}{4}+\sum_{n\geq 1}\left(\frac{3T^3}{2\pi^2n^2}\cos(2\pi nt/T)+\left(-\frac{T^3}{\pi n}+\frac{3T^3}{2\pi^3n^3}\right)\sin(2\pi nt/T)\right).$$

We see that all three Fourier series converge slowly. This is connected to the fact that none of the functions are continuous at the borders of the periods.

5. Write down difference equations for finding the Fourier coefficients of $f(t) = t^{k+1}$ from those of $f(t) = t^k$, and write a program which uses this recursion. Use the program to verify what you computed in Exercise 4.
Solution: Let us define $a_{n,k}, b_{n,k}$ as the Fourier coefficients of t^k. When $k > 0$ and $n > 0$, integration by parts gives us the following difference equations:

$$\begin{aligned}
a_{n,k} &= \frac{2}{T}\int_0^T t^k \cos(2\pi nt/T)dt \\
&= \frac{2}{T}\left(\left[\frac{T}{2\pi n}t^k \sin(2\pi nt/T)\right]_0^T - \frac{kT}{2\pi n}\int_0^T t^{k-1}\sin(2\pi nt/T)dt\right) \\
&= -\frac{kT}{2\pi n}b_{n,k-1} \\
b_{n,k} &= \frac{2}{T}\int_0^T t^k \sin(2\pi nt/T)dt \\
&= \frac{2}{T}\left(\left[-\frac{T}{2\pi n}t^k \cos(2\pi nt/T)\right]_0^T + \frac{kT}{2\pi n}\int_0^T t^{k-1}\cos(2\pi nt/T)dt\right) \\
&= -\frac{T^k}{\pi n} + \frac{kT}{2\pi n}a_{n,k-1}.
\end{aligned}$$

When $n > 0$, these can be used to express $a_{n,k}, b_{n,k}$ in terms of $a_{n,0}, b_{n,0}$, for which we clearly have $a_{n,0} = b_{n,0} = 0$. For $n = 0$ we have that $a_{0,k} = \frac{T^k}{k+1}$ for all k. The following program computes $a_{n,k}, b_{n,k}$ recursively when $n > 0$.

```
def findfouriercoeffs(n, k, T):
    ank, bnk = 0, 0
    if k > 0:
        ankprev, bnkprev = findfouriercoeffs(n,k-1,T)
        ank = -k*T*bnkprev/(2*pi*n)
        bnk = -T**k/(pi*n) + k*T*ankprev/(2*pi*n)
    return ank, bnk
```

6. Use the previous exercise to find the Fourier series for $f(x) = -\frac{1}{3}x^3 + \frac{1}{2}x^2 - \frac{3}{16}x + 1$ on the interval $[0,1]$. Plot the 9th order Fourier series for this function. You should obtain the plots from Figure 1.5.

1.3

1. Show that the complex functions $e^{2\pi int/T}$ are orthonormal.
Solution: For $n_1 \neq n_2$ we have that

$$\begin{aligned}
\langle e^{2\pi in_1t/T}, e^{2\pi in_2t/T}\rangle &= \frac{1}{T}\int_0^T e^{2\pi in_1t/T}e^{-2\pi in_2t/T}dt = \frac{1}{T}\int_0^T e^{2\pi i(n_1-n_2)t/T}dt \\
&= \left[\frac{T}{2\pi i(n_1-n_2)}e^{2\pi i(n_1-n_2)t/T}\right]_0^T \\
&= \frac{T}{2\pi i(n_1-n_2)} - \frac{T}{2\pi i(n_1-n_2)} = 0.
\end{aligned}$$

When $n_1 = n_2$ the integrand computes to 1, so that $\|e^{2\pi i n t/T}\| = 1$.

2. Compute the complex Fourier series of the function $f(t) = \sin^2(2\pi t/T)$.
Solution: We have that

$$\begin{aligned} f(t) = \sin^2(2\pi t/T) &= \left(\frac{1}{2i}(e^{2\pi i t/T} - e^{-2\pi i t/T}\right)^2 \\ &= -\frac{1}{4}(e^{2\pi i 2t/T} - 2 + e^{-2\pi i 2t/T}) = -\frac{1}{4}e^{2\pi i 2t/T} + \frac{1}{2} - \frac{1}{4}e^{-2\pi i 2t/T}. \end{aligned}$$

This gives the Fourier series of the function (with $y_2 = y_{-2} = -1/4$, $y_0 = 1/2$). This could also have been shown by using the trigonometric identity $\sin^2 x = \frac{1}{2}(1-\cos(2x))$ first, or by computing the integral $\frac{1}{T}\int_0^T f(t)e^{-2\pi i n t/T}dt$ (but this is rather cumbersome).

3. Repeat Exercise 4 in Section 1.2, computing the complex Fourier series instead of the real Fourier series.

4. In this exercise we will find a connection with certain Fourier series and the rows in Pascal's triangle.

a. Show that both $\cos^n(t)$ and $\sin^n(t)$ are in $V_{N,2\pi}$ for $1 \leq n \leq N$.
Solution: We have that

$$\begin{aligned} \cos^n(t) &= \left(\frac{1}{2}(e^{it} + e^{-it})\right)^n \\ \sin^n(t) &= \left(\frac{1}{2i}(e^{it} - e^{-it})\right)^n \end{aligned}$$

If we multiply out here, we get a sum of terms of the form e^{ikt}, where $-n \leq k \leq n$. As long as $n \leq N$ it is clear that this is in $V_{N,2\pi}$.

b. Write down the N'th order complex Fourier series for $f_1(t) = \cos t$, $f_2(t) = \cos^2 t$, og $f_3(t) = \cos^3 t$.
Solution: We have that

$$\begin{aligned} \cos(t) &= \frac{1}{2}(e^{it} + e^{-it}) \\ \cos^2(t) &= \frac{1}{4}(e^{it} + e^{-it})^2 = \frac{1}{4}e^{2it} + \frac{1}{2} + \frac{1}{4}e^{-2it} \\ \cos^3(t) &= \frac{1}{8}(e^{it} + e^{-it})^3 = \frac{1}{8}e^{3it} + \frac{3}{8}e^{it} + \frac{3}{8}e^{-it} + \frac{1}{8}e^{-3it}. \end{aligned}$$

Therefore, for the first function the nonzero Fourier coefficients are $y_{-1} = 1/2$, $y_1 = 1/2$, for the second function $y_{-2} = 1/4$, $y_0 = 1/2$, $y_2 = 1/4$, for the third function $y_{-3} = 1/8$, $y_{-1} = 3/8$, $y_1 = 3/8$, $y_3 = 1/8$.

c. In (b) you should be able to see a connection between the Fourier coefficients and the three first rows in Pascal's triangle. Formulate and prove a

general relationship between row n in Pascal's triangle and the Fourier coefficients of $f_n(t) = \cos^n t$.
Solution: In order to find the Fourier coefficients of $\cos^n(t)$ we have to multiply out the expression $\frac{1}{2^n}(e^{it} + e^{-it})^n$. The coefficients we get after this can alos be obtained from Pascal's triangle.

5. Compute the complex Fourier coefficients of the square wave using Equation 1.23, i.e. repeat the calculations from Example 1.18 for the complex case. Use Theorem 1.27 to verify your result.
Solution: We obtain that

$$\begin{aligned} y_n &= \frac{1}{T}\int_0^{T/2} e^{-2\pi int/T}dt - \frac{1}{T}\int_{T/2}^{T} e^{-2\pi int/T}dt \\ &= -\frac{1}{T}\left[\frac{T}{2\pi in}e^{-2\pi int/T}\right]_0^{T/2} + \frac{1}{T}\left[\frac{T}{2\pi in}e^{-2\pi int/T}\right]_{T/2}^{T} \\ &= \frac{1}{2\pi in}\left(-e^{-\pi in} + 1 + 1 - e^{-\pi in}+\right) \\ &= \frac{1}{\pi in}\left(1 - e^{-\pi in}\right) = \begin{cases} 0, & \text{if } n \text{ is even;} \\ 2/(\pi in), & \text{if } n \text{ is odd.} \end{cases}. \end{aligned}$$

Instead using Theorem 1.27 together with the coefficients $b_n = \frac{2(1-\cos(n\pi))}{n\pi}$ we computed in Example 1.18, we obtain

$$y_n = \frac{1}{2}(a_n - ib_n) = -\frac{1}{2}i\begin{cases} 0, & \text{if } n \text{ is even;} \\ 4/(n\pi), & \text{if } n \text{ is odd.} \end{cases} = \begin{cases} 0, & \text{if } n \text{ is even;} \\ 2/(\pi in), & \text{if } n \text{ is odd.} \end{cases}$$

when $n > 0$. The case $n < 0$ follows similarly.

6. Repeat Exercise 5 for the triangle wave.

7. Use Equation (1.23) to compute the complex Fourier coefficients of the periodic functions with period T defined by, respectively, $f(t) = t$, $f(t) = t^2$, and $f(t) = t^3$, on $[0, T]$. Use Theorem 1.27 to verify your calculations from Exercise 4 in Section 1.2.
Solution: For $f(t) = t$ we get

$$\begin{aligned} y_n &= \frac{1}{T}\int_0^T te^{-2\pi int/T}dt = \frac{1}{T}\left(\left[-\frac{T}{2\pi in}te^{-2\pi int/T}\right]_0^T + \int_0^T \frac{T}{2\pi in}e^{-2\pi int/T}dt\right) \\ &= -\frac{T}{2\pi in} = \frac{T}{2\pi n}i. \end{aligned}$$

From Exercise 4 we had $b_n = -\frac{T}{\pi n}$, for which Theorem 1.27 gives $y_n = \frac{T}{2\pi n}i$ for $n > 0$, which coincides with the expression we obtained. The case $n < 0$ follows similarly.
For $f(t) = t^2$ we get

$$\begin{aligned} y_n &= \frac{1}{T}\int_0^T t^2e^{-2\pi int/T}dt = \frac{1}{T}\left(\left[-\frac{T}{2\pi in}t^2e^{-2\pi int/T}\right]_0^T + 2\int_0^T \frac{T}{2\pi in}te^{-2\pi int/T}dt\right) \\ &= -\frac{T^2}{2\pi in} + \frac{T^2}{2\pi^2n^2} = \frac{T^2}{2\pi^2n^2} + \frac{T^2}{2\pi n}i. \end{aligned}$$

From Exercise 4 we had $a_n = \frac{T^2}{\pi^2 n^2}$ and $b_n = -\frac{T^2}{\pi n}$, for which Theorem 1.27 gives $y_n = \frac{1}{2}\left(\frac{T^2}{\pi^2 n^2} + i\frac{T^2}{\pi n}\right)$ for $n > 0$, which also is seen to coincide with what we obtained. The case $n < 0$ follows similarly.
For $f(t) = t^3$ we get

$$
\begin{aligned}
y_n &= \frac{1}{T}\int_0^T t^3 e^{-2\pi int/T} dt = \frac{1}{T}\left(\left[-\frac{T}{2\pi in} t^3 e^{-2\pi int/T}\right]_0^T + 3\int_0^T \frac{T}{2\pi in} t^2 e^{-2\pi int/T} dt\right) \\
&= -\frac{T^3}{2\pi in} + 3\frac{T}{2\pi in}\left(\frac{T^2}{2\pi^2 n^2} + \frac{T^2}{2\pi n} i\right) = 3\frac{T^3}{4\pi^2 n^2} + \left(\frac{T^3}{2\pi n} - 3\frac{T^3}{4\pi^3 n^3}\right) i =
\end{aligned}
$$

From Exercise 4 we had $a_n = \frac{3T^3}{2\pi^2 n^2}$ and $b_n = -\frac{T^3}{\pi n} + \frac{3T^3}{2\pi^3 n^3}$ for which Theorem 1.27 gives

$$
y_n = \frac{1}{2}\left(\frac{3T^3}{2\pi^2 n^2} + i\left(\frac{T^3}{\pi n} - \frac{3T^3}{2\pi^3 n^3}\right)\right) = \frac{3T^3}{4\pi^2 n^2} + \left(\frac{T^3}{2\pi n} - \frac{3T^3}{4\pi^3 n^3}\right) i
$$

for $n > 0$, which also is seen to coincide with what we obtained. The case $n < 0$ follows similarly.

8. In this exercise we will prove a version of Theorem 1.21 for complex Fourier coefficients.

a. If f is symmetric about 0, show that y_n is real, and that $y_{-n} = y_n$.
Solution: If f is symmetric about 0 we have that $b_n = 0$. Theorem 1.27 then gives that $y_n = \frac{1}{2}a_n$, which is real. The same theorem gives that $y_{-n} = \frac{1}{2}a_n = y_n$.

b. If f is antisymmetric about 0, show that the y_n are purely imaginary, $y_0 = 0$, and that $y_{-n} = -y_n$.
Solution: If f is antisymmetric about 0 we have that $a_n = 0$. Theorem 1.27 then gives that $y_n = -\frac{1}{2}b_n$, which is purely imaginary. The same theorem gives that $y_{-n} = \frac{1}{2}b_n = -y_n$.

c. Show that $\sum_{n=-N}^{N} y_n e^{2\pi int/T}$ is symmetric when $y_{-n} = y_n$ for all n, and rewrite it as a cosine-series.
Solution: When $y_n = y_{-n}$ we can write

$$
y_{-n}e^{2\pi i(-n)t/T} + y_n e^{2\pi int/T} = y_n(e^{2\pi int/T} + e^{-2\pi int/T}) = 2y_n\cos(2\pi nt/T)
$$

This is clearly symmetric, but then also $\sum_{n=-N}^{N} y_n e^{2\pi int/T}$ is symmetric since it is a sum of symmetric functions.

d. Show that $\sum_{n=-N}^{N} y_n e^{2\pi int/T}$ is antisymmetric when $y_0 = 0$ and $y_{-n} = -y_n$ for all n, and rewrite it as a sine-series.
Solution: When $y_n = -y_{-n}$ we can write

$$
y_{-n}e^{2\pi i(-n)t/T} + y_n e^{2\pi int/T} = y_n(-e^{2\pi int/T} + e^{2\pi int/T}) = 2iy_n\sin(2\pi nt/T)
$$

This is clearly antisymmetric, but then also $\sum_{n=-N}^{N} y_n e^{2\pi int/T}$ is antisymmetric since it is a sum of antisymmetric functions, and since $y_0 = 0$.

1.4

1. Define the function f with period T on $[-T/2, T/2]$ by

$$f(t) = \begin{cases} 1, & \text{if } -T/4 \le t < T/4; \\ -1, & \text{if } |T/4| \le t < |T/2|. \end{cases}$$

f is just the square wave, shifted with $T/4$. Compute the Fourier coefficients of f directly, and use Property 3 in Theorem 1.29 to verify your result.
Solution: We obtain that

$$\begin{aligned} y_n &= \frac{1}{T}\int_{-T/4}^{T/4} e^{-2\pi int/T} dt - \frac{1}{T}\int_{-T/2}^{-T/4} e^{-2\pi int/T} dt - \frac{1}{T}\int_{T/4}^{T/2} e^{-2\pi int/T} dt \\ &= -\left[\frac{1}{2\pi in} e^{-2\pi int/T}\right]_{-T/4}^{T/4} + \left[\frac{1}{2\pi in} e^{-2\pi int/T}\right]_{-T/2}^{-T/4} + \left[\frac{1}{2\pi in} e^{-2\pi int/T}\right]_{T/4}^{T/2} \\ &= \frac{1}{2\pi in}\left(-e^{-\pi in/2} + e^{\pi in/2} + e^{\pi in/2} - e^{\pi in} + e^{-\pi in} - e^{-\pi in/2}\right) \\ &= \frac{1}{\pi n}\left(2\sin(\pi n/2) - \sin(\pi n)\right) = \frac{2}{\pi n}\sin(\pi n/2). \end{aligned}$$

The square wave defined in this exercise can be obtained by delaying our original square wave with $-T/4$. Using Property 3 in Theorem 1.29 with $d = -T/4$ on the complex Fourier coefficients

$$y_n = \begin{cases} 0, & \text{if } n \text{ is even;} \\ 2/(\pi in), & \text{if } n \text{ is odd.} \end{cases}$$

which we obtained for the square wave in Exercise 5 in Section 1.3, we obtain the Fourier coefficients

$$\begin{aligned} e^{2\pi in(T/4)/T}\begin{cases} 0, & \text{if } n \text{ is even;} \\ 2/(\pi in), & \text{if } n \text{ is odd.} \end{cases} &= \begin{cases} 0, & \text{if } n \text{ is even;} \\ \frac{2i\sin(\pi n/2)}{\pi in}, & \text{if } n \text{ is odd.} \end{cases} \\ &= \begin{cases} 0, & \text{if } n \text{ is even;} \\ \frac{2}{\pi n}\sin(\pi n/2), & \text{if } n \text{ is odd.} \end{cases}. \end{aligned}$$

This verifies the result.

2. Find a function f which has the complex Fourier series

$$\sum_{n \text{ odd}} \frac{4}{\pi(n+4)} e^{2\pi int/T}.$$

Hint: Attempt to use one of the properties in Theorem 1.29 on the Fourier series of the square wave.
Solution: Since the real Fourier series of the square wave is

$$\sum_{n\ge 1, n \text{ odd}} \frac{4}{\pi n}\sin(2\pi nt/T),$$

Theorem 1.27 gives us that the complex Fourier coefficients are $y_n = -\frac{1}{2}i\frac{4}{\pi n} = -\frac{2i}{\pi n}$, and $y_{-n} = \frac{1}{2}i\frac{4}{\pi n} = \frac{2i}{\pi n}$ for $n > 0$. This means that $y_n = -\frac{2i}{\pi n}$ for all n, so that the complex Fourier series of the square wave is

$$-\sum_{n\,\text{odd}} \frac{2i}{\pi n} e^{2\pi i n t/T}.$$

Using Property 4 in Theorem 1.29 we get that the $e^{-2\pi i 4t/T}$ (i.e. set $d = -4$) times the square wave has its n'th Fourier coefficient equal to $-\frac{2i}{\pi(n+4)}$. Using linearity, this means that $2ie^{-2\pi i4t/T}$ times the square wave has its n'th Fourier coefficient equal to $\frac{4}{\pi(n+4)}$. We thus have that the function

$$f(t) = \begin{cases} 2ie^{-2\pi i 4t/T} & ,0 \le t < T/2 \\ -2ie^{-2\pi i 4t/T} & ,T/2 \le t < T \end{cases}$$

has the desired Fourier series.

3. Show that the complex Fourier coefficients y_n of f, and the cosine-coefficients a_n of $\breve{f}$ are related by $a_{2n} = y_n + y_{-n}$. This result is not enough to obtain the entire Fourier series of $\breve{f}$, but at least it gives us half of it.
Solution: The $2n$th complex Fourier coefficient of $\breve{f}$ is

$$\begin{aligned} &\frac{1}{2T}\int_0^{2T} \breve{f}(t)e^{-2\pi i 2nt/(2T)}dt \\ &= \frac{1}{2T}\int_0^{T} f(t)e^{-2\pi i nt/T}dt + \frac{1}{2T}\int_T^{2T} f(2T-t)e^{-2\pi i nt/T}dt. \end{aligned}$$

Substituting $u = 2T - t$ in the second integral we see that this is

$$\begin{aligned} &= \frac{1}{2T}\int_0^{T} f(t)e^{-2\pi i nt/T}dt - \frac{1}{2T}\int_T^{0} f(u)e^{2\pi i nu/T}du \\ &= \frac{1}{2T}\int_0^{T} f(t)e^{-2\pi i nt/T}dt + \frac{1}{2T}\int_0^{T} f(t)e^{2\pi i nt/T}dt \\ &= \frac{1}{2}y_n + \frac{1}{2}y_{-n}. \end{aligned}$$

Therefore we have $a_{2n} = y_n - y_{-n}$.

Chapter 2

2.1

1. Define the following sound signal

$$f(t) = \begin{cases} 0 & 0 \le t \le 4/440 \\ 2\frac{440t-4}{8}\sin(2\pi 440t) & 4/440 \le t \le 12/440 \\ 2\sin(2\pi 440t) & 12/440 \le t \le 20/440 \end{cases}$$

This corresponds to the sound plotted in Figure 1.1(a), where the sound is unaudible in the beginning, and increases linearly in loudness over time with a given frequency until maximum loudness is avchieved. Write a function which generates this sound, and listen to it.
Solution: The code for playing the sound can look like this:

```
# Exercise 2.2.1
t1 = arange(0, 4/440.0, 1/float(fs))
t2 = arange(4/440.0, 12/440.0, 1/float(fs))
t3 = arange(12/440.0, 20/440.0, 1/float(fs))

f1 = 0*t1                              # The first part of f
f2 = 2*((440*t2-4)/8)*sin(2*pi*440*t2)   # The second part of f
f3 = 2*sin(2*pi*440*t3)                # The third part of f
x = hstack([f1, f2, f3])
x /= abs(x).max()
play(x, fs)
```

Note that the sound has duration less than 0.05s, so you should only hear a very short beep. You also need to scale the values to be within -1 and 1, since some of the listed values are outside this range.

2. Find two constant a and b so that the function $f(t) = a\sin(2\pi 440t)+b\sin(2\pi 4400t)$ resembles the plot from Figure 1.1(b) as closely as possible. Generate the samples of this sound, and listen to it.
Solution: The important thing to note here is that there are two oscillations present in Figure 1.1(b): One slow oscillation with a higher amplitude, and one faster oscillation, with a lower amplitude. We see that there are 10 periods of the smaller

oscillation within one period of the larger oscillation, so that we should be able to reconstruct the figure by using frequencies where one is 10 times the other, such as 440Hz and 4400Hz. Also, we see from the figure that the amplitude of the larger oscillation is close to 1, and close to 0.3 for the smaller oscillation. A good choice therefore seems to be $a = 1, b = 0.3$. The code can look this:

```
# Exercise 2.2.2
t = arange(0,3,1/float(fs))
x = sin(2*pi*440*t) + 0.3*sin(2*pi*4400*t)
x /= abs(x).max()
play(x, fs)
```

3. Let us write some code so that we can experiment with different pure sounds

a. Write a function `play_pure_sound(f)` which generates the samples over a period of 3 seconds for a pure tone with frequency f, with sampling frequency $f_s = 2.5f$ (we will explain this value later).
Solution: The code can look like this:

```
def play_pure_sound(f):
    """
    Play a pure sound with a given frequency over three seconds.
    """
    fs = 44100
    t = linspace(0, 3, 3*fs)
    x = sin(2*pi*f*t)
    play(x, fs)
```

b. Use the function `play_pure_sound` to listen to pure sounds of frequency 440Hz and 1500Hz, and verify that they are the same as the sounds you already have listened to in this section.

c. How high frequencies are you able to hear with the function `play_pure_sound`? How low frequencies are you able to hear?

4. Write functions `play_square` and `play_triangle` which take T as input, and which play the square wave of Example 1.11 and the triangle wave of Example 1.12, respectively. In your code, let the samples of the waves be taken at a frequency of 44100 samples per second. Verify that you generate the same sounds as you played in these examples when you set $T = \frac{1}{440}$.
Solution: The code can look like this:

```
def play_square(T):
    length=3
    fs = 44100
    samplesperperiod = fs*T
    oneperiod = hstack([ones((samplesperperiod/2),dtype=float),\
                        -ones((samplesperperiod/2),dtype=float)])
    x = tile(oneperiod,length/T)
    play(x, fs)
```

```
def play_triangle(T):
    length = 3
    fs = 44100
    samplesperperiod = fs*T
    oneperiod = hstack([linspace(-1, 1, samplesperperiod/2), \
                        linspace(1, -1, samplesperperiod/2)])
    x = tile(oneperiod, length/T)
    play(x, fs)
```

5. Let us write programs so that we can listen to the Fourier approximations of the square wave and the triangle wave.

a. Write functions `play_square_fourier` and `play_triangle_fourier` which take T and N as input, and which play the order N Fourier approximation of the square wave and the triangle wave, respectively, for three seconds. Verify that you can generate the sounds you played in examples 1.18 and 1.19.
Solution: The code can look like this:

```
def play_square_fourier(T, N):
    length = 3
    fs = 44100
    t = linspace(0, length, fs*length)
    x = zeros(t.size)
    for n in range(1,N+1,2):
        x += sin(2*pi*n*t/T)/n
    x *= 4/pi
    x /= abs(x).max()
    play(x, fs)
```

```
def play_triangle_fourier(T, N):
    length = 3
    fs = 44100
    t = linspace(0, length, fs*length)
    x = zeros(t.size)
    for n in range(1,N+1,2):
        x -= cos(2*pi*n*t/T)/n**2
    x *= 8/(pi**2)
    x /= abs(x).max()
    play(x,fs)
```

b. For these Fourier approximations, how high must you choose N for them to be indistuingishable from the square/triangle waves themselves? Also describe how the characteristics of the sound changes when n increases.

6. In this exercise we will experiment as in the first examples of this section.

a. Write a function `play_with_different_fs` which takes sound samples and a sampling rate as input, and plays the sound samples of with the same sample rate as the original file, then with twice the sample rate, and then half the sample rate. You should start with reading the file into a matrix (as explained in this section). When applied to the sample audio file, are the sounds the same as those you heard in Example 2.5?
Solution: The code can look like this:

```
def play_with_different_fs( x, fs):
    play(x, fs)
    raw_input('PRESS ENTER TO CONTINUE.')
    play(x, 2*fs)
    raw_input('PRESS ENTER TO CONTINUE.')
    play(x, fs/2)
```

b. Write a function `play_reverse` which takes sound data and a sample rate as input, and plays the sound samples backwards. When you run the code on our sample audio file, is the sound the same as the one you heard in Example 2.7?
Solution: The code can look like this:

```
def play_reverse(x, fs):
    """
    Play the sound backwards.
    """
    N = shape(x)[0]
    play(x[(N-1)::(-1), :], fs)
```

c. Write the new sound samples from b. to a new `wav`-file, as described above, and listen to it with your favourite mediaplayer.

7. In this exercise, we will experiment with adding noise to a signal.

a. Write a function `play_with_noise` which takes sound data, sampling rate, and the damping constant c as input, and plays the sound samples with noise added as described above. Your code should add noise to both channels of the sound, and scale the sound samples so that they are between -1 and 1.
Solution: The code can look like this:

```
def play_with_noise(x, fs, c=0.1):
    """
    Play the sound with noise added. c represents the noise level,
    a number between 0 and 1.
    """
    z = x + c*(2*random.rand(shape(x))-1)
    z /= abs(z).max()
    play(z, fs)
```

b. With your program, generate the two sounds played in Example 2.9, and verify that they are the same as those you heard.

c. Listen to the sound samples with noise added for different values of c. For which range of c is the noise audible?

2.2

1. Compute $F_4\boldsymbol{x}$ when $\boldsymbol{x} = (2,3,4,5)$.
Solution: As in Example 2.19 we get

$$F_4\begin{pmatrix} 2\\3\\4\\5 \end{pmatrix} = \frac{1}{2}\begin{pmatrix} 1 & 1 & 1 & 1\\ 1 & -i & -1 & i\\ 1 & -1 & 1 & -1\\ 1 & i & -1 & -i \end{pmatrix}\begin{pmatrix} 2\\3\\4\\5 \end{pmatrix}$$
$$= \frac{1}{2}\begin{pmatrix} 2+3+4+5\\ 2-3i-4+5i\\ 2-3+4-5\\ 2+3i-4-5i \end{pmatrix} = \begin{pmatrix} 7\\ -1+i\\ -1\\ -1-i \end{pmatrix}.$$

2. As in Example 2.19, state the exact cartesian form of the Fourier matrix for the cases $N = 6$, $N = 8$, and $N = 12$.
Solution: For $N = 6$ the entries are on the form $\frac{1}{\sqrt{6}}e^{-2\pi ink/6} = \frac{1}{\sqrt{6}}e^{-\pi ink/3}$. This means that the entries in the Fourier matrix are the numbers $\frac{1}{\sqrt{6}}e^{-\pi i/3} = \frac{1}{\sqrt{6}}(1/2 - i\sqrt{3}/2)$, $\frac{1}{\sqrt{6}}e^{-2\pi i/3} = \frac{1}{\sqrt{6}}(-1/2 - i\sqrt{3}/2)$, and so on. The matrix is thus

$$F_6 = \frac{1}{\sqrt{6}}\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1\\ 1 & 1/2-i\sqrt{3}/2 & -1/2-i\sqrt{3}/2 & -1 & -1/2+i\sqrt{3}/2 & 1/2+i\sqrt{2}/2\\ 1 & -1/2-i\sqrt{3}/2 & -1/2+i\sqrt{3}/2 & 1 & -1/2-i\sqrt{3}/2 & +1/2-i\sqrt{3}/2\\ 1 & -1 & 1 & -1 & 1 & -1\\ 1 & -1/2+i\sqrt{3}/2 & -1/2-i\sqrt{3}/2 & 1 & -1/2+i\sqrt{3}/2 & -1/2-i\sqrt{3}/2\\ 1 & 1/2+i\sqrt{2}/2 & -1/2+i\sqrt{3}/2 & -1 & -1/2-i\sqrt{3}/2 & 1/2-i\sqrt{3}/2 \end{pmatrix}$$

The cases $N = 8$ and $N = 12$ follow similarly, but are even more tedious. For $N = 8$ the entries are $\frac{1}{\sqrt{8}}e^{\pi ink/4}$, which can be expressed exactly since we can express exactly any sines and cosines of a multiple of $\pi/4$. For $N = 12$ we get the base angle $\pi/6$, for which we also have exact values for sines and cosines for all multiples.

3. We have a real vector $\boldsymbol{x}$ with length N, and define the vector $\boldsymbol{z}$ by delaying all elements in $\boldsymbol{x}$ with 5 cyclically, i.e. $z_5 = x_0$, $z_6 = x_1,\ldots,z_{N-1} = x_{N-6}$, and $z_0 = x_{N-5},\ldots,z_4 = x_{N-1}$. For a given n, if $|(F_N\boldsymbol{x})_n| = 2$, what is then $|(F_N\boldsymbol{z})_n|$? Justify the answer.
Solution: $\boldsymbol{z}$ is the vector $\boldsymbol{x}$ delayed with $d = 5$ samples, and then Property 3 of Theorem 2.21 gives us that $(F_N\boldsymbol{z})_n = e^{-2\pi i5k/N}(F_N\boldsymbol{x})_n$. In particular $|(F_N\boldsymbol{z})_n| = |(F_N\boldsymbol{x})_n| = 2$, since $|e^{-2\pi i5k/N}| = 1$.

4. Given a real vector $\boldsymbol{x}$ of length 8 where $(F_8(\boldsymbol{x}))_2 = 2 - i$, what is $(F_8(\boldsymbol{x}))_6$?
Solution: By Theorem 2.21 we know that $(F_N(\boldsymbol{x}))_{N-n} = \overline{(F_N(\boldsymbol{x}))_n}$ when $\boldsymbol{x}$ is a real vector. If we set $N = 8$ and $n = 2$ we get that $(F_8(\boldsymbol{x}))_6 = \overline{(F_8(\boldsymbol{x}))_2} = \overline{2 - i} = 2 + i$.

5. Let $\boldsymbol{x}$ be the vector of length N where $x_k = \cos^2(2\pi k/N)$. What is then $F_N\boldsymbol{x}$?
Solution: The idea is to express $\boldsymbol{x}$ as a linear combination of the Fourier basis vectors ϕ_n, and use that $F_N\phi_n = \boldsymbol{e}_n$. We have that

$$\begin{aligned}
\cos^2(2\pi k/N) &= \left(\frac{1}{2}\left(e^{2\pi ik/N} + e^{-2\pi ikn/N}\right)\right)^2 \\
&= \frac{1}{4}e^{2\pi i2k/N} + \frac{1}{2} + \frac{1}{4}e^{-2\pi i2k/N} = \frac{1}{4}e^{2\pi i2k/N} + \frac{1}{2} + \frac{1}{4}e^{2\pi i(N-2)k/N} \\
&= \sqrt{N}\left(\frac{1}{4}\phi_2 + \frac{1}{2}\phi_0 + \frac{1}{4}\phi_{N-2}\right).
\end{aligned}$$

We here used the periodicity of $e^{2\pi ikn/N}$, i.e. that $e^{-2\pi i2k/N} = e^{2\pi i(N-2)k/N}$. Since F_N is linear and $F_N(\phi_n) = \boldsymbol{e}_n$, we have that

$$F_N(\boldsymbol{x}) = \sqrt{N}\left(\frac{1}{4}\boldsymbol{e}_2 + \frac{1}{2}\boldsymbol{e}_0 + \frac{1}{4}\boldsymbol{e}_{N-2}\right) = \sqrt{N}(1/2, 0, 1/4, 0, \ldots, 0, 1/4, 0).$$

6. Let $\boldsymbol{x}$ be the vector with entries $x_k = c^k$. Show that the DFT of $\boldsymbol{x}$ is given by the vector with components

$$y_n = \frac{1 - c^N}{1 - ce^{-2\pi in/N}}$$

for $n = 0, \ldots, N-1$.
Solution: We get

$$\begin{aligned}
y_n &= \sum_{k=0}^{N-1} c^k e^{-2\pi ink/N} = \sum_{k=0}^{N-1} (ce^{-2\pi in/N})^k \\
&= \frac{1 - (ce^{-2\pi in/N})^N}{1 - ce^{-2\pi in/N}} = \frac{1 - c^N}{1 - ce^{-2\pi in/N}}.
\end{aligned}$$

7. If $\boldsymbol{x}$ is complex, Write the DFT in terms of the DFT on real sequences. Hint: Split into real and imaginary parts, and use linearity of the DFT.

8. Extend the code for the function `DFTImpl` in Example 2.20 so that

1. The function also takes a second parameter called `forward`. If this is true the DFT is applied. If it is false, the IDFT is applied. If this parameter is not present, then the forward transform should be assumed.

2. If the input `x` is two-dimensional (i.e. a matrix), the DFT/IDFT should be applied to each column of `x`. This ensures that, in the case of sound, the FFT is applied to each channel in the sound when the enrire sound is used as input, as we are used to when applying different operations to sound.

Also, write documentation for the code.
Solution: The code can look like this:

```
def DFTImpl(x, forward=True):
    """
    Compute the DFT of the vector x using standard matrix
    multiplication. To avoid out of memory situations, we do not
    allocate the entire DFT matrix, only one row of it at a time.
    Note that this function differs from the FFT in that it includes
    the normalizing factor 1/sqrt(N). The DFT is computed along axis
    0. If there is another axis, the DFT is computed for each element
    in this as well.

    x: a vector
    forward: Whether or not this is forward (i.e. DFT)
    or reverse (i.e. IDFT)
    """
    y = zeros_like(x).astype(complex)
    N = len(x)
    sign = -(2*forward - 1)
    if ndim(x) == 1:
        for n in xrange(N):
            D = exp(sign*2*pi*n*1j*arange(float(N))/N)
            y[n] = dot(D, x)
    else:
        for n in range(N):
            D = exp(sign*2*pi*n*1j*arange(float(N))/N)
            for s2 in xrange(shape(x)[1]):
                y[n,s2] = dot(D,x[:, s2])
    if sign == 1:
        y /= float(N)
    return y
```

9. Assume that N is even.

a. Show that, if $x_{k+N/2} = x_k$ for all $0 \le k < N/2$, then $y_n = 0$ when n is odd.
Solution: We have that

$$\begin{aligned}
y_n &= \frac{1}{\sqrt{N}}\left(\sum_{k=0}^{N/2-1} x_k e^{-2\pi ikn/N} + \sum_{k=N/2}^{N-1} x_k e^{-2\pi ikn/N}\right) \\
&= \frac{1}{\sqrt{N}}\left(\sum_{k=0}^{N/2-1} x_k e^{-2\pi ikn/N} + \sum_{k=0}^{N/2-1} x_k e^{-2\pi i(k+N/2)n/N}\right) \\
&= \frac{1}{\sqrt{N}}\sum_{k=0}^{N/2-1} x_k (e^{-2\pi ikn/N} + (-1)^n e^{-2\pi ikn/N}) \\
&= (1+(-1)^n)\frac{1}{\sqrt{N}}\sum_{k=0}^{N/2-1} x_k e^{-2\pi ikn/N}
\end{aligned}$$

If n is odd, we see that $y_n = 0$.

b. Show that, if $x_{k+N/2} = -x_k$ for all $0 \le k < N/2$, then $y_n = 0$ when n is even.
Solution: The proof is the same as in a., except for a sign change.

c. Show also the converse statements in a. and b..
Solution: Clearly the set of vectors which satisfies $x_{k+N/2} = \pm x_k$ is a vector

space V of dimension $N/2$. The set of vectors where every second component is zero is also a vector space of dimension $N/2$, let us denote this by W. We have shown that $F_N(V) \subset W$, but since F_N is unitary, $F_N(V)$ also has dimension $N/2$, so that $F_N(V) = W$. This shows that when every second y_n is 0, we must have that $x_{k+N/2} = \pm x_k$, and the proof is done.

d. Also show the following:
Solution: In the proofs above, compute the IDFT instead.

1. $x_n = 0$ for all odd n if and only if $y_{k+N/2} = y_k$ for all $0 \le k < N/2$.
2. $x_n = 0$ for all even n if and only if $y_{k+N/2} = -y_k$ for all $0 \le k < N/2$.

10. Let $\boldsymbol{x}_1, \boldsymbol{x}_2$ be real vectors, and set $\boldsymbol{x} = \boldsymbol{x}_1 + i\boldsymbol{x}_2$. Use Theorem 2.21 to show that

$$(F_N(\boldsymbol{x}_1))_k = \frac{1}{2}\left((F_N(\boldsymbol{x}))_k + \overline{(F_N(\boldsymbol{x}))_{N-k}}\right)$$
$$(F_N(\boldsymbol{x}_2))_k = \frac{1}{2i}\left((F_N(\boldsymbol{x}))_k - \overline{(F_N(\boldsymbol{x}))_{N-k}}\right)$$

This shows that we can compute two DFT's on real data from one DFT on complex data, and $2N$ extra additions.
Solution: We have that

$$(F_N(\boldsymbol{x}))_k = (F_N(\boldsymbol{x}_1 + i\boldsymbol{x}_2))_k = (F_N(\boldsymbol{x}_1))_k + i(F_N(\boldsymbol{x}_2))_k$$
$$(F_N(\boldsymbol{x}))_{N-k} = (F_N(\boldsymbol{x}_1))_{N-k} + i(F_N(\boldsymbol{x}_2))_{N-k} = \overline{(F_N(\boldsymbol{x}_1))_k} + i\overline{(F_N(\boldsymbol{x}_2))_k},$$

where we have used Property 1 of Theorem 2.21. If we take the complex conjugate in the last equation, we are left with the two equations

$$(F_N(\boldsymbol{x}))_k = (F_N(\boldsymbol{x}_1))_k + i(F_N(\boldsymbol{x}_2))_k$$
$$\overline{(F_N(\boldsymbol{x}))_{N-k}} = (F_N(\boldsymbol{x}_1))_k - i(F_N(\boldsymbol{x}_2))_k.$$

If we add these we get

$$(F_N(\boldsymbol{x}_1))_k = \frac{1}{2}\left((F_N(\boldsymbol{x}))_k + \overline{(F_N(\boldsymbol{x}))_{N-k}}\right),$$

which is the first equation. If we instead subtract the equations we get

$$(F_N(\boldsymbol{x}_2))_k = \frac{1}{2i}\left((F_N(\boldsymbol{x}))_k - \overline{(F_N(\boldsymbol{x}))_{N-k}}\right),$$

which is the second equation

2.3

1. Explain what the code below does, line by line:

```
# Exercise 2.3.1
x = x[0:2**17]
y = fft.fft(x)
y[(2**17/4):(3*2**17/4)] = 0
newx = abs(fft.ifft(y))
newx /= abs(newx).max()
play(newx, fs)
```

Comment in particular why we adjust the sound samples by dividing with the maximum value of the sound samples. What changes in the sound do you expect to hear?
Solution: First a sound file is read. We then restrict to the first 2^{12} sound samples, perform a DFT, zero out the frequencies which correspond to DFT-indices between 2^{10} and $2^{12}-2^{10}-1$, and perform an IDFT. Finally we scale the sound samples so that these lie between -1 and 1, which is the range we demand for the sound samples, and play the new sound.

2. In the code from the previous exercise it turns out that $f_s = 44100$Hz. Which frequencies in the sound file will be changed on the line where we zero out some of the DFT coefficients?
Solution: As we have seen, DFT index n corresponds to frequency $\nu = nf_s/N$. Above $N = 2^{17}$, so that we get the connection $\nu = nf_s/N = n\times 44100/2^{17}$. We zeroed the DFT indices above $n = 2^{15}$, so that frequencies above $\nu = 2^{15}\times 44100/2^{17} = 11025Hz$ are affected.

3. Implement code where you do the following:

1. at the top you define the function $f(x) = \cos^6(x)$, and $M = 3$,
2. compute the unique interpolant from $V_{M,T}$ (i.e. by taking $N = 2M+1$ samples over one period), as guaranteed by Theorem 2.24,
3. plot the interpolant againts f over one period.

Finally run the code also for $M = 4$, $M = 5$, and $M = 6$. Explain why the plots coincide for $M = 6$, but not for $M < 6$. Does increasing M above $M = 6$ have any effect on the plots?
Solution: The code can look as follows.

```
from scitools.std import *

f = lambda t:cos(t)**6
M = 5
T = 2*pi
N = 2*M + 1
t = linspace(0, T, 100)
x = f(linspace(0, T - T/float(N), N))
y = fft.fft(x)/N
s = real(y[0])*ones(len(t))
for k in range(1,(N+1)/2):
    s += 2*real(y[k]*exp(2*pi*1j*k*t/float(T)))
plot(t, s, 'r', t, f(t), 'g')
legend('Interpolant from V_{M,T}','f')
raw_input('Press any key to terminate')
```

2.4

1. Recall that, in exercise 8 in section 2.2, we extended the direct DFT implementation so that it accepted a second parameter telling us if the forward or reverse transform should be applied. Extend the general function and the standard kernel in the same way. Again, the forward transform should be used if the `forward` parameter is not present. Assume also that the kernel accepts only one-dimensional data, and that the general function applies the kernel to each column in the input if the input is two-dimensional (so that the FFT can be applied to all channels in a sound with only one call). The signatures for our methods should thus be changed as follows:

```
def FFTImpl(x, FFTKernel, forward = True):
def FFTKernelStandard(x, forward):
```

It should be straightforward to make the modifications for the reverse transform by consulting the second part of Theorem 2.36. For simplicity, let `FFTImpl` take care of the additional division with N we need to do in case of the IDFT. In the following we will assume these signatures for the FFT implementation and the corresponding kernels.
Solution: The functions can be implemented as follows:

```
def FFTImpl(x, FFTKernel, forward = True):
    """
    Compute the FFT or IFFT of the vector x. Note that this function
    differs from the DFT in that the normalizing factor 1/sqrt(N) is
    not included. The FFT is computed along axis 0. If there is
    another axis, the FFT is computed for each element in this as
    well. This function calls a kernel for computing the FFT. The
    kernel assumes that the input has been bit-reversed, and contains
    only one axis. This function is where the actual bit reversal and
    the splitting of the axes take place.

    x: a vector
    FFTKernel: can be any of FFTKernelStandard, FFTKernelNonrec, and
    FFTKernelSplitradix. The kernel assumes that the input has been
    bit-reversed, and contains only one axis.
    forward: Whether the FFT or the IFFT is applied
    """
    if ndim(x) == 1:
        bitreverse(x)
        FFTKernel(x, forward)
    else:
        bitreversearr(x)
        for s2 in xrange(shape(x)[1]):
            FFTKernel(x[:, s2], forward)
    if not forward:
        x /= len(x)
```

```
def FFTKernelStandard(x, forward):
    """
    Compute the FFT of x, using a standard FFT algorithm.
    x: a bit-reversed version of the input. Should have only one axis
    forward: Whether the FFT or the IFFT is applied
    """
    N = len(x)
    sign = -1
    if not forward:
        sign = 1
    if N > 1:
        xe, xo = x[0:(N/2)], x[(N/2):]
        FFTKernelStandard(xe, forward)
        FFTKernelStandard(xo, forward)
        D = exp(sign*2*pi*1j*arange(float(N/2))/N)
        xo *= D
        x[:] = concatenate([xe + xo, xe - xo])
```

2. In this exercise we will compare execution times for the different methods for computing the DFT.

a. Write code which compares the execution times for an N-point DFT for the following three cases: Direct implementation of the DFT (as in Example 2.20), the FFT implementation used in this chapter, and the built-in `fft`-function. Your code should use the sample audio file `castanets.wav`, apply the different DFT implementations to the first $N = 2^r$ samples of the file for $r = 3$ to $r = 15$, store the execution times in a vector, and plot these. You can use the function `time()` in the `time` module to measure the execution time.

b. A problem for large N is that there is such a big difference in the execution times between the two implementations. We can address this by using a loglog-plot instead. Plot N against execution times using the function `loglog`. How should the fact that the number of arithmetic operations are $8N^2$ and $5N\log_2 N$ be reflected in the plot?
Solution: The two different curves you see should have a derivative approximately equal to one and two, respectively.

c. It seems that the built-in FFT is much faster than our own FFT implementation, even though they may use similar algorithms. Try to explain what can be the cause of this.
Solution: There may be several reasons for this. One is that Python code runs slowly when compared to native code, which is much used in the built-in FFT. Also, the built-in `fft` has been subject to much more optimization than we have covered here.

3. Let $\boldsymbol{x}_1 = (1,3,5,7)$ and $\boldsymbol{x}_2 = (2,4,6,8)$. Compute $\mathrm{DFT}_4\boldsymbol{x}_1$ and $\mathrm{DFT}_4\boldsymbol{x}_2$. Explain how you can compute $\mathrm{DFT}_8(1,2,3,4,5,6,7,8)$ based on these computations (you don't need to perform the actual computation). What are the benefits of this approach?

Solution: We get

$$\mathrm{DFT}_4\boldsymbol{x}_1 = \begin{pmatrix}1 & 1 & 1 & 1\\ 1 & -i & -1 & i\\ 1 & -1 & 1 & -1\\ 1 & i & -1 & -i\end{pmatrix}\begin{pmatrix}1\\3\\5\\7\end{pmatrix} = \begin{pmatrix}16\\ -4+4i\\ -4\\ -4-4i\end{pmatrix}$$

$$\mathrm{DFT}_4\boldsymbol{x}_2 = \begin{pmatrix}1 & 1 & 1 & 1\\ 1 & -i & -1 & i\\ 1 & -1 & 1 & -1\\ 1 & i & -1 & -i\end{pmatrix}\begin{pmatrix}2\\4\\6\\8\end{pmatrix} = \begin{pmatrix}20\\ -4+4i\\ -4\\ -4-4i\end{pmatrix}$$

In the FFT-algorithm we split the computation of $\mathrm{DFT}_4(\boldsymbol{x})$ into the computation of $\mathrm{DFT}_2(\boldsymbol{x}^{(e)})$ and $\mathrm{DFT}_2(\boldsymbol{x}^{(o)})$, where $x^{(e)}$ and $x^{(o)}$ are vectors of length 4 with even-indexed and odd-indexed components, respectively. In this case we have $\boldsymbol{x}^{(e)} = (1,3,5,7)$ and $\boldsymbol{x}^{(o)} = (2,4,6,8)$. In other words, the FFT-algorithm uses the FFT-computations we first made, so that we can save computation. The benefit of using the FFT-algorithm is that we save computations, so that we end up with $O(5N\log_2 N)$ real arithmetic operations.

4. When N is composite, there are a couple of results we can state regarding polyphase components.

> **a.** Assume that $N = N_1N_2$, and that $\boldsymbol{x} \in \mathbb{R}^N$ satisfies $x_{k+rN_1} = x_k$ for all k, r, i.e. $\boldsymbol{x}$ has period N_1. Show that $y_n = 0$ for all n which are not multiplums of N_2.
> **Solution**: We have that $\boldsymbol{x}^{(p)}$ is a constant vector of length N_2 for $0 \le p < N_1$. But then the DFT of all the $\boldsymbol{x}^{(p)}$ has zero outside entry zero. Multiplying with $e^{-2\pi ikn/N}$ does not affect this. The last $N_2 - 1$ rows are thus zero before the final DFT is applied, so that these rows are zero also after this final DFT. After assembling the polyphase components again we have that yrN_2 are the only nonzero DFT-coefficients.
>
> **b.** Assume that $N = N_1N_2$, and that $\boldsymbol{x}^{(p)} = \boldsymbol{0}$ for $p \neq 0$. Show that the polyphase components $\boldsymbol{y}^{(p)}$ of $\boldsymbol{y} = \mathrm{DFT}_N\boldsymbol{x}$ are constant vectors for all p.

5. When we wrote down the difference equation for the number of multiplications in the FFT algorithm, you could argue that some multiplications were not counted. Which multiplications in the FFT algorithm were not counted when writing down this difference equation? Do you have a suggestion to why these multiplications were not counted?
Solution: When we compute $e^{-2\pi in/N}$, we do some multiplications/divisions in the exponent. These are not counted because they do not depend on $\boldsymbol{x}$, and may therefore be precomputed.

6. (Adapting the FFT algorithm to real data, first approach). There was an error in this exercise.

7. (Adapting the FFT algorithm to real data, second approach). In this exercise we will look at a second approach to how we can adapt an FFT algorithm to real input $\boldsymbol{x}$. We will now instead rewrite Equation (2.14) for indices n and $N/2-n$ as

$$\begin{aligned} y_n &= (\mathrm{DFT}_{N/2}\boldsymbol{x}^{(e)})_n + e^{-2\pi i n/N}(\mathrm{DFT}_{N/2}\boldsymbol{x}^{(o)})_n \\ y_{N/2-n} &= (\mathrm{DFT}_{N/2}\boldsymbol{x}^{(e)})_{N/2-n} + e^{-2\pi i(N/2-n)/N}(\mathrm{DFT}_{N/2}\boldsymbol{x}^{(o)})_{N/2-n} \\ &= (\mathrm{DFT}_{N/2}\boldsymbol{x}^{(e)})_{N/2-n} - e^{2\pi i n/N}\overline{(\mathrm{DFT}_{N/2}\boldsymbol{x}^{(o)})_n} \\ &= \overline{(\mathrm{DFT}_{N/2}\boldsymbol{x}^{(e)})_n} - \overline{e^{-2\pi i n/N}(\mathrm{DFT}_{N/2}\boldsymbol{x}^{(o)})_n}. \end{aligned}$$

We see here that, if we have computed the terms in y_n (which needs an additional 4 real multiplications, since $e^{-2\pi i n/N}$ and $(\mathrm{DFT}_{N/2}\boldsymbol{x}^{(o)})_n$ are complex), no further multiplications are needed in order to compute $y_{N/2-n}$, since its compression simply conjugates these terms before adding them. Again $y_{N/2}$ must be handled explicity with this approach. For this we can use the formula

$$y_{N/2} = (\mathrm{DFT}_{N/2}\boldsymbol{x}^{(e)})_0 - (D_{N/2}\mathrm{DFT}_{N/2}\boldsymbol{x}^{(o)})_0$$

instead.

a. Conclude from this that an FFT algorithm adapted to real data at each step requires $N/4$ complex additions and $N/2$ additions. Conclude from this as before that an algorithm based on real data requires $M_N = O(N\log_2 N)$ multiplications and $A_N = O\left(\frac{3}{2}N\log_2 N\right)$ additions (i.e. again we obtain half the operation count of complext input).

b. Find an IFFT algorithm adapted to vectors $\boldsymbol{y}$ which have conjugate symmetry, which has the same operation count we found above.
Hint: Consider the vectors $y_n + \overline{y_{N/2-n}}$ and $e^{2\pi i n/N}(y_n - \overline{y_{N/2-n}})$. From the equations above, how can these be used in an IFFT?

8 (Non-recursive FFT algorithm). Use factorization (2.19) to write a kernel function `FFTKernelNonrec` for a non-recursive FFT implementation. In your code, perform the matrix multiplications in factorization (2.19) from right to left in an (outer) for-loop. For each matrix loop through the different blocks on the diagonal in an (inner) for-loop. Make sure you have the right number of blocks on the diagonal, each block being on the form $\begin{pmatrix} I & D_{N/2^k} \\ I & -D_{N/2^k} \end{pmatrix}$. It may be a good idea to start by implementing multiplication with such a simple matrix first as these are the building blocks in the algorithm (aslo attempt to do this so that everything is computed in-place). Also compare the execution times with our original FFT algorithm, as we did in Exercise 2, and try to explain what you see in this comparison.
Solution: The algorithm for the nonecursive FFT can look as follows

```
def FFTKernelNonrec(x, forward):
    """
    Compute the FFT of x, using a non-recursive FFT algorithm.
    x: a bit-reversed version of the input. Should have only one axis
    forward: Whether the FFT or the IFFT is applied
    """
    N = len(x)
    sign = -1
    if not forward:
        sign = 1
    D = exp(sign*2*pi*1j*arange(float(N/2))/N)
    nextN = 1
    while nextN < N:
        k = 0
        while k < N:
            xe, xo = x[k:(k + nextN)], x[(k + nextN):(k + 2*nextN)]
            xo *= D[0::(N/(2*nextN))]
            x[k:(k+2*nextN)] = concatenate([xe + xo, xe - xo])
            k += 2*nextN
        nextN *= 2
```

If you add the nonrecursive algorithm to the code from Exercise 2, you will see that the non-recursive algorithm performs much better. There may be several reasons for this. First of all, there are no recursive function calls. Secondly, the values in the matrices $D_{N/2}$ are constructed once and for all with the non-recursive algorithm. Code which compares execution times for the original FFT algorithm, our nonrecursive implementation, and the split-radix algorithm of the next exercise, can look as follows:

```
# Exercise 2.4.8
x0, fs = audioread('castanets.wav')

kvals = arange(3,16)
slowtime = zeros(len(kvals))
fasttime = zeros(len(kvals))
fastesttime = zeros(len(kvals))
N = 2**kvals
for k in kvals:
    x = x0[0:2**k].astype(complex)

    start = time()
    FFTImpl(x, FFTKernelStandard)
    slowtime[k - kvals[0]] = time() - start

    start = time()
    FFTImpl(x, FFTKernelNonrec)
    fasttime[k - kvals[0]] = time() - start

    start = time()
    FFTImpl(x, FFTKernelSplitradix)
    fastesttime[k - kvals[0]] = time() - start

plot(kvals, slowtime, 'ro-', \
     kvals, fasttime, 'bo-', \
     kvals, fastesttime, 'go-')
grid('on')
title('time usage of the DFT methods')
legend('Standard FFT algorithm', \
       'Non-recursive FFT', \
       'Split radix FFT')
xlabel('log_2 N')
```

```
ylabel('time used [s]')
```

9 (The Split-radix FFT algorithm). In this exercise we will develop a variant of the FFT algorithm called the *split-radix FFT algorithm*, which until recently held the record for the lowest operation count for any FFT algorithm.

We start by splitting the rightmost $\mathrm{DFT}_{N/2}$ in Equation (2.18) by using Equation (2.18) again, to obtain

$$\mathrm{DFT}_N \boldsymbol{x} = \begin{pmatrix} \mathrm{DFT}_{N/2} & D_{N/2}\begin{pmatrix} \mathrm{DFT}_{N/4} & D_{N/4}\mathrm{DFT}_{N/4} \\ \mathrm{DFT}_{N/4} & -D_{N/4}\mathrm{DFT}_{N/4} \end{pmatrix} \\ \mathrm{DFT}_{N/2} & -D_{N/2}\begin{pmatrix} \mathrm{DFT}_{N/4} & D_{N/4}\mathrm{DFT}_{N/4} \\ \mathrm{DFT}_{N/4} & -D_{N/4}\mathrm{DFT}_{N/4} \end{pmatrix} \end{pmatrix} \begin{pmatrix} \boldsymbol{x}^{(e)} \\ \boldsymbol{x}^{(oe)} \\ \boldsymbol{x}^{(oo)} \end{pmatrix}. \tag{2.1}$$

The term radix describes how an FFT is split into FFT's of smaller sizes, i.e. how the sum in an FFT is split into smaller sums. The FFT algorithm we started this section with is called a radix 2 algorithm, since it splits an FFT of length N into FFT's of length $N/2$. If an algorithm instead splits into FFT's of length $N/4$, it is called a radix 4 FFT algorithm. The algorithm we go through here is called the split radix algorithm, since it uses FFT's of both length $N/2$ and $N/4$.
Solution: The code for the split-radix algorithm can look as follows

```
def FFTKernelSplitradix(x, forward):
    """
    Compute the FFT of x, using the split-radix FFT algorithm.

    x: a bit-reversed version of the input. Should have only one axis
    forward: Whether the FFT or the IFFT is applied
    """
    N = len(x)
    sign = -1
    if not forward:
        sign = 1
    if N == 2:
        x[:] = [x[0] + x[1], x[0] - x[1]]
    elif N > 2:
        xe, xo1, xo2 = x[0:(N/2)], x[(N/2):(3*N/4)], x[(3*N/4):N]
        FFTKernelSplitradix(xe, forward)
        FFTKernelSplitradix(xo1, forward)
        FFTKernelSplitradix(xo2, forward)
        G = exp(sign*2*pi*1j*arange(float(N/4))/N)
        H = G*exp(sign*2*pi*1j*arange(float(N/4))/(N/2))
        xo1 *= G
        xo2 *= H
        xo = concatenate( [xo1 + xo2, -sign*1j*(xo2 - xo1)] )
        x[:] = concatenate([xe + xo, xe - xo])
```

a. Let $G_{N/4}$ be the $(N/4)\times(N/4)$ diagonal matrix with $e^{-2\pi i n/N}$ on the diagonal. Show that $D_{N/2} = \begin{pmatrix} G_{N/4} & \mathbf{0} \\ \mathbf{0} & -iG_{N/4} \end{pmatrix}$.

b. Let $H_{N/4}$ be the $(N/4)\times(N/4)$ diagonal matrix $G_{D/4}D_{N/4}$. Verify the following rewriting of Equation (2.1):

$$\begin{aligned}
\text{DFT}_N\boldsymbol{x} &= \begin{pmatrix} \text{DFT}_{N/2} & \begin{pmatrix} G_{N/4}\text{DFT}_{N/4} & H_{N/4}\text{DFT}_{N/4} \\ -iG_{N/4}\text{DFT}_{N/4} & iH_{N/4}\text{DFT}_{N/4} \end{pmatrix} \\ \text{DFT}_{N/2} & \begin{pmatrix} -G_{N/4}\text{DFT}_{N/4} & -H_{N/4}\text{DFT}_{N/4} \\ iG_{N/4}\text{DFT}_{N/4} & -iH_{N/4}\text{DFT}_{N/4} \end{pmatrix} \end{pmatrix} \begin{pmatrix} \boldsymbol{x}^{(e)} \\ \boldsymbol{x}^{(oe)} \\ \boldsymbol{x}^{(oo)} \end{pmatrix} \\
&= \begin{pmatrix} I & \mathbf{0} & G_{N/4} & H_{N/4} \\ \mathbf{0} & I & -iG_{N/4} & iH_{N/4} \\ I & \mathbf{0} & -G_{N/4} & -H_{N/4} \\ \mathbf{0} & I & iG_{N/4} & -iH_{N/4} \end{pmatrix} \begin{pmatrix} \text{DFT}_{N/2} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \text{DFT}_{N/4} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \text{DFT}_{N/4} \end{pmatrix} \begin{pmatrix} \boldsymbol{x}^{(e)} \\ \boldsymbol{x}^{(oe)} \\ \boldsymbol{x}^{(oo)} \end{pmatrix} \\
&= \begin{pmatrix} \text{DFT}_{N/2}\boldsymbol{x}^{(e)} + \begin{pmatrix} G_{N/4}\text{DFT}_{N/4}\boldsymbol{x}^{(oe)} + H_{N/4}\text{DFT}_{N/4}\boldsymbol{x}^{(oo)} \\ i\left(H_{N/4}\text{DFT}_{N/4}\boldsymbol{x}^{(oe)} - G_{N/4}\text{DFT}_{N/4}\boldsymbol{x}^{(oo)}\right) \end{pmatrix} \\ \text{DFT}_{N/2}\boldsymbol{x}^{(e)} - \begin{pmatrix} G_{N/4}\text{DFT}_{N/4}\boldsymbol{x}^{(oe)} + H_{N/4}\text{DFT}_{N/4}\boldsymbol{x}^{(oo)} \\ i\left(H_{N/4}\text{DFT}_{N/4}\boldsymbol{x}^{(oe)} - G_{N/4}\text{DFT}_{N/4}\boldsymbol{x}^{(oo)}\right) \end{pmatrix} \end{pmatrix}
\end{aligned}$$

c. Explain from the above expression why, once the three FFT's above have been computed, the rest can be computed with $N/2$ complex multiplications, and $2\times N/4+N=3N/2$ complex additions. This is equivalent to $2N$ real multiplications and $N+3N=4N$ real additions.
Hint: It is important that $G_{N/4}\text{DFT}_{N/4}\boldsymbol{x}^{(oe)}$ and $H_{N/4}\text{DFT}_{N/4}\boldsymbol{x}^{(oo)}$ are computed first, and the sum and difference of these two afterwards.

d. Due to what we just showed, our new algorithm leads to real multiplication and addition counts which satisfy

$$M_N = M_{N/2}+2M_{N/4}+2N \qquad\qquad A_N = A_{N/2}+2A_{N/4}+4N$$

Find the general solutions to these difference equations and conclude from these that $M_N = O\left(\frac{4}{3}N\log_2 N\right)$, and $A_N = O\left(\frac{8}{3}N\log_2 N\right)$. The operation count is thus $O\left(4N\log_2 N\right)$, which is a reduction of $N\log_2 N$ from the FFT algorithm.

e. Write an FFT kernel function `FFTKernelSplitradix` for the split-radix algorithm (again this should handle both the forward and reverse transforms). Are there more or less recursive function calls in this function than in the original FFT algorithm? Also compare the execution times with our original FFT algorithm, as we did in Exercise 2. Try to explain what you see in this comparison.
Solution: If you add the split-radix FFT algorithm also to the code from Exercise 2, you will see that it performs better than the FFT algorithm, but worse than the non-recursive algorithm. That it performs better than the FFT algorithm is as expected, since it has a reduced number of arithmetic operations, and also a smaller number of recursive calls. It is not surprising that the non-recursive function performs better, since only that function omits recursive calls, and computes the values in the diagonal matrices once and for all.

By carefully examining the algorithm we have developed, one can reduce the operation count to $4N\log_2 N - 6N + 8$. This does not reduce the order of the algorithm, but for small N (which often is the case in applications) this reduces the number of operations considerably, since $6N$ is large compared to $4N\log_2 N$ for small N. In addition to having a lower number of operations than the FFT algorithm of Theorem 2.34, a bigger percentage of the operations are additions for our new algorithm: there are now twice as many additions than multiplications. Since multiplications may be more time-consuming than additions (depending on how the CPU computes floating-point arithmetic), this can be a big advantage.

10. In this exercise we will make some considerations which will help us explain the code for bit-reversal. This is perhaps not a mathematically challenging exercise, but neverthesless a good exercise in how to think when developing an efficient algorithm. We will use the notation i for an index, and j for its bit-reverse. If we bit-reverse k bits, we will write $N = 2^k$ for the number of possible indices.

a. Consider the following code

```
j = 0
for i in range(N-1):
    print j
    m = N/2
    while (m >= 1 and j >= m):
        j -= m
        m /= 2
    j += m
```

Explain that the code prints all numbers in $[0, N-1]$ in bit-reversed order (i.e. j). Verify this by running the program, and writing down the bits for all numbers for, say $N = 16$. In particular explain the decrements and increments made to the variable j. The code above thus produces pairs of numbers (i, j), where j is the bit-reverse of i. As can be seen, `bitreverse` applies similar code, and then swaps the values x_i and x_j in $\boldsymbol{x}$, as it should.

Solution: Note that, if the bit representation of i ends with $0\underbrace{1\dots1}_{n}$, then $i+1$ has a bit representation which ends with $1\underbrace{0\dots0}_{n}$, with the remaining first bits unaltered. Clearly the bit-reverse of i then starts with $\underbrace{1\dots1}_{n}0$ and the bit-reverse of $i+1$ starts with $1\underbrace{0\dots0}_{n}$. We see that the bit reverse of $i+1$ can be obtained from the bit-reverse of i by replacing the first consecutive set of ones by zeros, and the following zero by one. This is performed by the line above where j is decreased by m: Decreasing j by $N/2$ when $j \geq N/2$ changes the first bit from 1 to 0, and similarly for the next n bits. The line where j is increased with m changes bit number $n+1$ from 0 to 1.

Since bit-reverse is its own inverse (i.e. $P^2 = I$), it can be performed by swapping elements i and j. One way to secure that bit-reverse is done only once, is to perform it only when $j > i$. You see that `bitreverse` includes this check.

b. Explain that $N-j-1$ is the bit-reverse of $N-i-1$. Due to this, when $i, j < N/2$, we have that $N-i-1, N-j-l \geq N/2$, and that `bitreversal` can swap them. Moreover, all swaps where $i, j \geq N/2$ can be performed immdiately when pairs where $i, j < N/2$ are encountered. Explain also that $j < N/2$ if and only if i is even. In the code you can see that the swaps (i, j) and $(N-i-1, N-j-1)$ are performed together when i is even, due to this.
Solution: Clearly $N-i-1$ has a bit representation obtained by changing every bit in i. That $N-j-1$ is the bit-reverse of $N-i-1$ follows immediately from this. If i is even, the least significant bit is 0. After bit-reversal, this becomes the most significant bit, and the most significant bit of j is 0 which is the case if and only if $j < N/2$.

c. Assume that $i < N/2$ is odd. Explain that $j \geq N/2$, so that $j > i$. This says that when $i < N/2$ is odd, we can always swap i and j (this is the last swap performed in the code). All swaps where $0 \leq j < N/2$ and $N/2 \leq j < N$ can be performed in this way.
Solution: If $i < N/2$ is odd, then the least significant bit is 1. This means that the most significant bit of j is 1, so that $j \geq N/2$, so that $j > i$.

In `bitreversal`, you can see that the bit-reversal of $2r$ and $2r+1$ are handled together (i.e. i is increased with 2 in the `for`-loop). The effect of this is that the number of `if`-tests can be reduced, due to the observations from b) and c).

Chapter 3

3.1

1. Assume that the filter S is defined by the formula

$$z_n = \frac{1}{4}x_{n+1} + \frac{1}{4}x_n + \frac{1}{4}x_{n-1} + \frac{1}{4}x_{n-2}.$$

Write down the filter coefficients t_k, and the matrix for S when $N = 8$.
Solution: Here we have that $t_{-1} = 1/4$, $t_0 = 1/4$, $t_1 = 1/4$, and $t_2 = 1/4$. We now get that $s_0 = t_0 = 1/4$, $s_1 = t_1 = 1/4$, and $s_2 = t_2 = 1/4$ (first formula), and $s_{N-1} = s_7 = t_{-1} = 1/4$ (second formula). This means that the matrix of S is

$$S = \frac{1}{4}\begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix}.$$

2. Given the circulant Toeplitz matrix

$$S = \begin{pmatrix} 1 & 2 & 0 & 0 \\ 0 & 1 & 2 & 0 \\ 0 & 0 & 1 & 2 \\ 2 & 0 & 0 & 1 \end{pmatrix},$$

write down the filter coefficients t_k.

3. Assume that S is a circulant Toeplitz matrix so that only

$$S_{0,0}, \ldots, S_{0,F} \text{ and } S_{0,N-E}, \ldots, S_{0,N-1}$$

are nonzero on the first row, where E, F are given numbers. When implementing this filter on a computer we need to make sure that the vector indices lie in $[0, N-1]$.

Show that $z_n = (S\boldsymbol{x})_n$ can be split into the following different formulas, depending on n, to achieve this:

a. $0 \le n < E$:

$$z_n = \sum_{k=0}^{n-1} S_{0,N+k-n} x_k + \sum_{k=n}^{F+n} S_{0,k-n} x_k + \sum_{k=N-1-E+n}^{N-1} S_{0,k-n} x_k. \tag{3.1}$$

b. $E \le n < N-F$:

$$z_n = \sum_{k=n-E}^{n+F} S_{0,k-n} x_k. \tag{3.2}$$

c. $N-F \le n < N$:

$$z_n = \sum_{k=0}^{n-(N-F)} S_{0,k-n} x_k + \sum_{k=n-E}^{n-1} S_{0,N+k-n} x_k + \sum_{k=n}^{N-1} S_{0,k-n} x_k. \tag{3.3}$$

From these three formulas we can write down a full implementation of the filter. This implementation is often more useful than writing down the entire matrix S, since we save computation when many of the matrix entries are zero.

3.2

1. In Example 2.7 we looked at time reversal as an operation on digital sound. In $\mathbb{R}^N$ this can be defined as the linear mapping which sends the vector $\boldsymbol{e}_k$ to $\boldsymbol{e}_{N-1-k}$ for all $0 \le k \le N-1$.

a. Write down the matrix for the time reversal linear mapping, and explain from this why time reversal is not a digital filter.
Solution: The matrix for time reversal is the matrix

$$\begin{pmatrix} 0 & 0 & \cdots & 0 & 1 \\ 0 & 0 & \cdots & 1 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 1 & \cdots & 0 & 0 \\ 1 & 0 & \cdots & 0 & 0 \end{pmatrix}$$

This is not a circulant Toeplitz matrix, since all diagonals assume the values 0 and 1, so that they are not constant on each diagonal. Time reversal is thus not a digital filter.

b. Prove directly that time reversal is not a time-invariant operation.
Solution: Let S denote time reversal. Clearly $S\boldsymbol{e}_1 = \boldsymbol{e}_{N-2}$. If S was time-invariant we would have that $S\boldsymbol{e}_0 = \boldsymbol{e}_{N-3}$, where we have delayed the input and output. But this clearly is not the case, since by definition $S\boldsymbol{e}_0 = \boldsymbol{e}_{N-1}$.

2. Let S be a digital filter. Show that S is symmetric if and only if the frequency response satisfies $s_k = s_{N-k}$ for all k.

3. Consider the matrix

$$S = \begin{pmatrix} 4 & 1 & 3 & 1 \\ 1 & 4 & 1 & 3 \\ 3 & 1 & 4 & 1 \\ 1 & 3 & 1 & 4 \end{pmatrix}.$$

a. Compute the eigenvalues and eigenvectors of S using the results of this section. You should only need to perform one DFT in order to achieve this.
Solution: The eigenvalues of S are $1,5,9$, and are found by computing a DFT of the first column (and multiplying by $\sqrt{N} = 2$). The eigenvectors are the Fourier basis vectors. 1 has multiplicity 2.

b. Verify the result from a. by computing the eigenvectors and eigenvalues the way you taught in your first course in linear algebra. This should be a much more tedious task.

c. Use a computer to compute the eigenvectors and eigenvalues of S also. For some reason some of the eigenvectors seem to be different from the Fourier basis vectors, which you would expect from the theory in this section. Try to find an explanation for this.
Solution: The computer uses some numeric algorithm to find the eigenvectors. However, eigenvectors may not be unique, so you have no control on which eigenvectors it actually selects. In particular, here the eigenspace for $\lambda = 1$ has dimension 2, so that any linear combination of the two eigenvectors from this eigenspace also is an eigenvector. Here it seems that a linear combination is chosen which is different from a Fourier basis vector.

4. Assume that S_1 and S_2 are two circulant Toeplitz matrices.

a. How can you express the eigenvalues of $S_1 + S_2$ in terms of the eigenvalues of S_1 and S_2?
Solution: If we write $S_1 = F_N^H D_1 F_N$ and $S_2 = F_N^H D_2 F_N$ we get

$$S_1 + S_2 = F_N^H (D_1 + D_2) F_N \qquad S_1 S_2 = F_N^H D_1 F_N F_N^H D_2 F_N = F_N^H D_1 D_2 F_N$$

This means that the eigenvalues of $S_1 + S_2$ are the sum of the eigenvalues of S_1 and S_2. The actual eigenvalues which are added are dictated by the index of the frequency response, i.e. $\lambda_{S_1+S_2,n} = \lambda_{S_1,n} + \lambda_{S_2,n}$.

b. How can you express the eigenvalues of $S_1 S_2$ in terms of the eigenvalues of S_1 and S_2?
Solution: As above we have that $S_1 S_2 = F_N^H D_1 F_N F_N^H D_2 F_N = F_N^H D_1 D_2 F_N$, and the same reasoning gives that the eigenvalues of $S_1 S_2$ are the product of the eigenvalues of S_1 and S_2. The actual eigenvalues which are multiplied are dictated by the index of the frequency response, i.e. $\lambda_{S_1 S_2,n} = \lambda_{S_1,n} \lambda_{S_2,n}$.

c. If A and B are general matrices, can you find a formula which expresses the eigenvalues of $A + B$ and AB in terms of those of A and B? If not, can you

find a counterexample to what you found in a. and b.?
Solution: In general there is no reason to believe that there is a formula for the eigenvalues for the sum or product of two matrices, based on eigenvalues of the individual matrices. However, the same type of argument as for filters can be used in all cases where the eigenvectors are equal.

5. Consider the linear mapping S which keeps every second component in $\mathbb{R}^N$, i.e. $S(\boldsymbol{e}_{2k}) = \boldsymbol{e}_{2k}$, and $S(\boldsymbol{e}_{2k-1}) = \boldsymbol{0}$. Is S a digital filter?
Solution: The matrix for the operation which keeps every second component is

$$\begin{pmatrix} 1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \\ 0 & 0 & \cdots & 0 & 0 \end{pmatrix},$$

where 1 and 0 are repeated in alternating order along the main diagonal. Since the matrix is not constant on the main diagonal, it is not a circulant Toeplitz matrix, and hence not a filter.

3.3

1. Let again S be the filter defined by the equation

$$z_n = \frac{1}{4}x_{n+1} + \frac{1}{4}x_n + \frac{1}{4}x_{n-1} + \frac{1}{4}x_{n-2},$$

as in Exercise 1 in Section 3.1. Compute and plot (the magnitude of) $\lambda_S(\omega)$.
Solution: The frequency response is

$$\lambda_S(\omega) = \frac{1}{4}(e^{i\omega} + 1 + e^{-i\omega} + e^{-2i\omega}) = \frac{e^{i\omega}(1 - e^{-4i\omega})}{4(1 - e^{-i\omega})} = \frac{1}{4}e^{-i\omega/2}\frac{\sin(2\omega)}{\sin(\omega/2)}.$$

2. A filter S is defined by the equation

$$z_n = \frac{1}{3}(x_n + 3x_{n-1} + 3x_{n-2} + x_{n-3}).$$

a. Compute and plot the (magnitude of the continuous) frequency response of the filter, i.e. $|\lambda_S(\omega)|$. Is the filter a lowpass filter or a highpass filter?
Solution: The filter coefficients are $t_0 = t_3 = 1/3$, $t_1 = t_2 = 1$. We have that

$$\begin{aligned}\lambda_S(\omega) &= \sum_k t_k e^{-ik\omega} = \frac{1}{3}(1 + 3e^{-i\omega} + 3e^{-2i\omega} + e^{-3i\omega}) \\ &= \frac{2}{3}e^{-3i\omega/2}\frac{1}{2}(e^{3i\omega/2} + 3e^{i\omega/2} + 3e^{-i\omega/2} + e^{-3i\omega/2}) \\ &= \frac{2}{3}e^{-3i\omega/2}(\cos(3\omega/2) + 3\cos(\omega/2)).\end{aligned}$$

From this expression it is easy to plot the frequency response, but since this is complex, we have to plot the magnitude, i.e. $|\lambda_S(\omega)| = \frac{2}{3}|\cos(3\omega/2)+3\cos(\omega/2)|$. We also see that $\lambda_S(0) = \frac{2}{3}$, and that $\lambda_S(\pi) = 0$, so that the filter is a lowpass filter.

b. Find an expression for the vector frequency response $\lambda_{S,2}$. What is $S\boldsymbol{x}$ when $\boldsymbol{x}$ is the vector of length N with components $e^{2\pi i 2k/N}$?
Solution: If we use the connection between the vector frequency response and the continuous frequency response we get

$$\lambda_{S,2} = \lambda_S(2\pi 2/N) = \frac{2}{3}e^{-6\pi i/N}(\cos(6\pi/N) + 3\cos(2\pi/N)).$$

Alternatively you can here compute that the first column in the circulant Toeplitz matrix for S is given by $s_0 = t_1$, $s_2 = t_2$, $s_3 = t_3$, and $s_4 = t_4$, and insert this in the definition of the vector frequency response, $\lambda_{S,2} = \sum_{k=0}^{N-1} s_k e^{-2\pi i 2k/N}$. We know that $e^{2\pi i 2k/N}$ is an eigenvector for S since S is a filter, and that $\lambda_{S,2}$ is the corresponding eigenvalue. We therefore get that

$$S\boldsymbol{x} = \lambda_{S,2}\boldsymbol{x} = \frac{2}{3}e^{-6\pi i/N}(\cos(6\pi/N) + 3\cos(2\pi/N))\boldsymbol{x}.$$

3. A filter S_1 is defined by the equation

$$z_n = \frac{1}{16}(x_{n+2} + 4x_{n+1} + 6x_n + 4x_{n-1} + x_{n-2}).$$

a. Write down an 8×8 circulant Toeplitz matrix which corresponds to applying S_1 on a periodic signal with period $N = 8$.
Solution: Since clearly $t_{-2} = t_2 = 1/16$, $t_{-1} = t_1 = 1/4$, and $t_0 = 6/16$, the first column $\boldsymbol{s}$ in the circulant Toeplitz matrix is given by $s_0 = t_0 = 6/16$, $s_1 = t_1 = 4/16$, $s_2 = t_2 = 1/16$, $s_{N-2} = t_{-2} = 1/16$, $s_{N-1} = t_{-1} = 4/16$. An 8×8 circulant Toeplitz matrix which corresponds to applying S_1 to a periodic signal of length $N = 8$ is therefore

$$\frac{1}{16}\begin{pmatrix} 6 & 4 & 1 & 0 & 0 & 0 & 1 & 4 \\ 4 & 6 & 4 & 1 & 0 & 0 & 0 & 1 \\ 1 & 4 & 6 & 4 & 1 & 0 & 0 & 0 \\ 0 & 1 & 4 & 6 & 4 & 1 & 0 & 0 \\ 0 & 0 & 1 & 4 & 6 & 4 & 1 & 0 \\ 0 & 0 & 0 & 1 & 4 & 6 & 4 & 1 \\ 1 & 0 & 0 & 0 & 1 & 4 & 6 & 4 \\ 4 & 1 & 0 & 0 & 0 & 1 & 4 & 6 \end{pmatrix}.$$

b. Compute and plot (the continuous) frequency response of the filter. Is the filter a lowpass filter or a highpass filter?
Solution: The frequency response is

$$\lambda_{S_1}(\omega) = \frac{1}{16}(e^{2i\omega} + 4e^{i\omega} + 6 + 4e^{-i\omega} + e^{-2i\omega}) = \left(\frac{1}{2}(e^{i\omega/2} + e^{-i\omega/2})\right)^4 = \cos^4(\omega/2),$$

where we recognized $(1,4,6,4,1)$ as a row in Pascal's triangle, so that we could write the expression as a power. From this expression it is easy to plot the frequency response, and it is clear that the filter is a lowpass filter, since $\lambda_{S_1}(0) = 1$, $\lambda_{S_1}(\pi) = 0$.

c. Another filter S_2 has (continuous) frequency response $\lambda_{S_2}(\omega) = (e^{i\omega} + 2 + e^{-i\omega})/4$. Write down the filter coefficients for the filter S_1S_2.
Solution: We have that

$$\lambda_{S_2}(\omega) = (e^{i\omega} + 2 + e^{-i\omega})/4 = \left(\frac{1}{2}(e^{i\omega/2} + e^{-i\omega/2})\right)^2 = \cos^2(\omega/2).$$

We then get that

$$\begin{aligned}\lambda_{S_1S_2}(\omega) &= \lambda_{S_1}(\omega)\lambda_{S_2}(\omega) = \cos^4(\omega/2)\cos^2(\omega/2) = \cos^6(\omega/2)\\ &= \left(\frac{1}{2}(e^{i\omega/2} + e^{-i\omega/2})\right)^6\\ &= \frac{1}{64}(e^{3i\omega} + 6e^{2i\omega} + 15e^{i\omega} + 20 + 15e^{-i\omega} + 6e^{-2i\omega} + e^{-3i\omega}),\end{aligned}$$

where we have used that, since we have a sixth power, the values can be obtained from fra a row in Pascal's triangle also here. It is now clear that

$$S_1S_2 = \frac{1}{64}\{1, 6, 15, \underline{20}, 15, 6, 1\}.$$

You could also have argumented here by taking the convolution of $\frac{1}{16}(1,4,6,4,1)$ with $\frac{1}{4}(1,2,1)$.

4. Assume that the filters S_1 and S_2 have the frequency responses $\lambda_{S_1}(\omega) = 2 + 4\cos(\omega)$, $\lambda_{S_2}(\omega) = 3\sin(2\omega)$.

a. Compute and plot the frequency response of the filter S_1S_2.

b. Write down the filter coefficients t_k and the impulse response $\boldsymbol{s}$ for the filter S_1S_2.

5. The Hanning window is defined by $w_n = 1 - \cos(2\pi n/(N-1))$. Compute and plot the window coefficients and the continuous frequency response of this window for $N = 32$, and compare with the window coefficients and the frequency responses for the rectangular- and the Hamming window.

3.4

1. Compute and plot the continuous frequency response of the filter $S = \{1/4, \underline{1/2}, 1/4\}$. Where does the frequency response achieve its maximum and minimum value, and what are these values?
Solution: We have that $\lambda_S(\omega) = \frac{1}{2}(1 + \cos\omega)$. This clearly has the maximum point $(0,1)$, and the minimum point $(\pi, 0)$.

2. Plot the continuous frequency response of the filter $T = \{1/4, \underline{-1/2}, 1/4\}$. Where does the frequency response achieve its maximum and minimum value, and what are these values? Can you write down a connection between this frequency response and that from Exercise 1?
Solution: We have that $|\lambda_T(\omega)| = \frac{1}{2}(1-\cos\omega)$. This clearly has the maximum point $(\pi, 1)$, and the minimum point $(0,0)$. The connection between the frequency responses is that $\lambda_T(\omega) = \lambda_S(\omega+\pi)$.

3. Define the filter S by $S = \{1, 2, \underline{3}, 4, 5, 6\}$. Write down the matrix for S when $N = 8$. Plot (the magnitude of) $\lambda_S(\omega)$, and indicate the values $\lambda_{S,n}$ for $N = 8$ in this plot.
Solution: Here we have that $s_0 = t_0 = 3$, $s_1 = t_1 = 4$, $s_2 = t_2 = 5$, and $s_3 = t_3 = 6$ (first formula), and $s_{N-2} = t_{-2} = 1$, $s_{N-1} = t_{-1} = 2$ (second formula). This means that the matrix of S is

$$S = \begin{pmatrix} 3 & 2 & 1 & 0 & 0 & 6 & 5 & 4 \\ 4 & 3 & 2 & 1 & 0 & 0 & 6 & 5 \\ 5 & 4 & 3 & 2 & 1 & 0 & 0 & 6 \\ 6 & 5 & 4 & 3 & 2 & 1 & 0 & 0 \\ 0 & 6 & 5 & 4 & 3 & 2 & 1 & 0 \\ 0 & 0 & 6 & 5 & 4 & 3 & 2 & 1 \\ 1 & 0 & 0 & 6 & 5 & 4 & 3 & 2 \\ 2 & 1 & 0 & 0 & 6 & 5 & 4 & 3 \end{pmatrix}$$

The frequency response is

$$\lambda_S(\omega) = e^{2i\omega} + 2e^{i\omega} + 3 + 4e^{-i\omega} + 5e^{-2i\omega} + 6e^{-3i\omega}.$$

4. Given the circulant Toeplitz matrix

$$S = \frac{1}{5}\begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & 1 & 1 & \cdots & 0 \\ 0 & 1 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ 1 & 0 & 0 & \cdots & 1 \\ 1 & 1 & 0 & \cdots & 1 \\ 1 & 1 & 1 & \cdots & 1 \end{pmatrix}$$

Write down the compact notation for this filter. Compute and plot (the magnitude) of $\lambda_S(\omega)$.
Solution: The filter coefficients are $t_0 = s_0 = 1/5$, $t_1 = s_1 = 1/5$ (first formula), and $t_{-1} = s_{N-1} = 1/5$, $t_{-2} = s_{N-2} = 1/5$, $t_{-3} = s_{N-3} = 1/5$ (second formula). All other t_k are zero. This means that the filter can be written as $\frac{1}{5}\{1, 1, 1, \underline{1}, 1\}$, using our compact notation.

5. Assume that $S = \{\underline{1}, c, c^2, \ldots, c^k\}$. Compute and plot $\lambda_S(\omega)$ when $k = 4$ and $k = 8$. How does the choice of k influence the frequency response? How does the choice of

c influence the frequency response?
Solution: The frequency response is

$$\sum_{s=0}^{k} c^s e^{-is\omega} = \frac{1-c^{k+1}e^{-i(k+1)\omega}}{1-ce^{-i\omega}}.$$

It is straightforward to compute the limit as $\omega \to 0$ as $c^k(k+1)$. This means that as we increase k or c, this limit also increases. The value of k also dictates oscillations in the frequency response, since the numerator oscillates fastest. When $c = 1$, k dictates how often the frequency response hits 0.

6. Compute the convolution of $\{\underline{1}, 2, 1\}$ with itself. interpret the result in terms of two polynomials.

7. Assume that $\boldsymbol{t}$ consists of the $2L+1$ nonzero filter coefficients of a symmetric filter (i.e. $t_{L+k} = t_{L-k}$ for all k).
$\boldsymbol{t} * \boldsymbol{x}$. In this exercise we will find out how to keep to track of the length and the start and end indices when we convolve two sequences.

a. Let $\boldsymbol{x}$ be zero outside $x_a, \ldots, x_{a+N-1}$, and $\boldsymbol{y}$ be zero outside $y_b, \ldots, y_{b+M-1}$. Show that $\boldsymbol{z} = \boldsymbol{x} * \boldsymbol{y}$ is zero outside $z_{a+b}, \ldots, z_{a+b+M+N-2}$. Explain why this means that $l(\boldsymbol{x} * \boldsymbol{y}) = l(\boldsymbol{x}) + l(\boldsymbol{y})$ for general vectors.

b. Find expressions for the start- and end indices k_{min}, k_{max} for $\boldsymbol{x} * \boldsymbol{y}$, in terms of those of $\boldsymbol{x}$ and $\boldsymbol{y}$.

8. Implement a function `convimpl(x,y)` which from input vectors of dimension N and M, respectively, returns an output vector of dimension $N+M-1$, as dictated by Equation (**??**). Compare your function together with the built-in `conv`-function to verify that they give the same results.

9. Show that if $S = \{\underline{t_0}, \ldots, t_F\}$ and $\boldsymbol{x} \in \mathbb{R}^N$, then $S\begin{pmatrix} \boldsymbol{x} \\ \boldsymbol{0}_F \end{pmatrix} = \boldsymbol{t} * \boldsymbol{x}$. Thus if we add zeros in a vector, filtering and convolution are the same.

3.5

1. Let E_{d_1} and E_{d_2} be two time delay filters. Show that $E_{d_1}E_{d_2} = E_{d_1+d_2}$ (i.e. that the composition of two time delays again is a time delay) in two different ways

a. Give a direct argument which uses no computations.

b. By using Property 3 in Theorem 2.21, i.e. by using a property for the Discrete Fourier Transform.

2. In this exercise, we will experiment with adding echo to a signal.

a. Write a function `play_with_echo` which takes the sound samples, the sample rate, a damping constant `c`, and a delay `d` as input, and plays the sound samples with an echo added, as described in Example 3.31. Recall that you have to ensure that the sound samples lie in $[-1,1]$.
Solution: The code can look like this:

```
def play_with_echo(x, fs, c, d):
    """
    Play the sound with an echo added.

    x: the vector of sound samples
    fs: the sample rate
    c:the strength of the echo
    d: the delay of the echo in samples.
    """
    N,nchannels = shape(x)
    z = zeros((N,nchannels))
    z[0:d] = x[0:d]
    z[d:N] = x[d:N] + c*x[0:(N-d)]
    z /= abs(z).max()
    play(z, fs)
```

b. Generate the sound from Example 3.31, and verify that it is the same as the one you heard there.

c. Listen to the sound samples for different values of d and c. For which range of d is the echo distinguisible from the sound itself? How low can you choose c in order to still hear the echo?

3. Consider the two filters $S_1 = \{\underline{1}, 0, \ldots, 0, c\}$ and $S_2 = \{\underline{1}, 0, \ldots, 0, -c\}$. Both of these can be interpreted as filters which add an echo. Show that $\frac{1}{2}(S_1 + S_2) = I$. What is the interpretation of this relation in terms of echos?
Solution: The sum of two digital filters is again a digital filter, and the first column in the sum can be obtained by summing the first columns in the two matrices. This means that the filter coefficients in $\frac{1}{2}(S_1 + S_2)$ can be obtained by summing the filter coefficients of S_1 and S_2, and we obtain

$$\frac{1}{2}\left(\{\underline{1}, 0, \ldots, 0, c\} + \{\underline{1}, 0, \ldots, 0, -c\}\right) = \{\underline{1}\}.$$

This means that $\frac{1}{2}(S_1 + S_2) = I$, since I is the unique filter with $\boldsymbol{e}_0$ as first column. The interpretation in terms of echos is that the echo from S_2 cancels that from S_1.

4. In this exercise, we will experiment with increasing and reducing the treble and bass in a signal as in examples 3.38 and 3.41.

a. Write functions `play_with_reduced_treble` and `play_with_reduced_bass` which take a data vector, sampling rate, and k as input, and which reduce bass and treble, respectively, in the ways described above, and plays the result, when row number $2k$ in Pascal' triangle is used to construct the filters. Use the function `conv` to help you to find the values in Pascal's triangle.
Solution: The code can look like this:

```
def play_with_reduced_bass( x, fs, k):
    t = [1.]
    for kval in range(k):
        t = convolve(t, [1/2., -1/2.])
    N,nchannels=shape(x)
    z = convolve(t, x[:, 0])
    play(z,fs)
```

```
def play_with_reduced_treble( x, fs, k):
    t = [1.]
    for kval in range(k):
        t = convolve(t, [1/2., 1/2.])
    N,nchannels=shape(x)
    z = convolve(t, x[:, 0])
    play(z,fs)
```

b. Generate the sounds you heard in examples 3.38 and 3.41, and verify that they are the same.

c. In your code, it will not be necessary to scale the values after reducing the treble, i.e. the values are already between -1 and 1. Explain why this is the case.

d. How high must k be in order for you to hear difference from the actual sound? How high can you choose k and still recognize the sound at all?

5. Consider again Example 3.35. Find an expression for a filter so that only frequencies so that $|\omega-\pi| < \omega_c$ are kept, i.e. the filter should only keep angular frequencies close to π (i.e. here we construct a highpass filter).

6. In this exercise we will investigate how we can combine lowpass and highpass filters to produce other filters

a. Assume that S_1 and $S2$ are lowpass filters. What kind of filter is S_1S_2? What if both S_1 and S_2 are highpass filters?

b. Assume that one of S_1, S_2 is a highpass filter, and that the other is a lowpass filter. What kind of filter S_1S_2 in this case?

7. A filter S_1 has the frequency response $\frac{1}{2}(1+\cos\omega)$, and another filter has the frequency response $\frac{1}{2}(1+\cos(2\omega))$.

a. Is S_1S_2 a lowpass filter, or a highpass filter?

b. What does the filter S_1S_2 do with angular frequencies close to $\omega = \pi/2$.

c. Find the filter coefficients of S_1S_2.
Hint: Use Theorem 3.21 to compute the frequency response of S_1S_2 first.

d. Write down the matrix of the filter $S_1 S_2$ for $N = 8$.

8. An operation describing some transfer of data in a system is defined as the composition of the following three filters:

- First a time delay filter with delay $d_1 = 2$, due to internal transfer of data in the system,
- then the treble-reducing filter $T = \{1/4, \underline{1/2}, 1/4\}$,
- finally a time delay filter with delay $d_2 = 4$ due to internal transfer of the filtered data.

We denote by $T_2 = E_{d_2} T E_{d_1} = E_4 T E_2$ the operation which applies these filters in succession.

a. Explain why T_2 also is a digital filter. What is (the magnitude of) the frequency response of E_{d_1}? What is the connection between (the magnitude of) the frequency response of T and T_2?

b. Show that $T_2 = \{\underline{0}, 0, 0, 0, 0, 1/4, 1/2, 1/4\}$.
Hint: Use the expressions $(E_{d_1}\boldsymbol{x})_n = x_{n-d_1}$, $(T\boldsymbol{x})_n = \frac{1}{4}x_{n+1} + \frac{1}{2}x_n + \frac{1}{4}x_{n-1}$, $(E_{d_2}\boldsymbol{x})_n = x_{n-d_2}$, and compute first $(E_{d_1}\boldsymbol{x})_n$, then $(TE_{d_1}\boldsymbol{x})_n$, and finally $(T_2\boldsymbol{x})_n = (E_{d_2} T E_{d_1}\boldsymbol{x})_n$. From the last expression you should be able to read out the filter coefficients.

c. Assume that $N = 8$. Write down the 8×8-circulant Toeplitz matrix for the filter T_2.

9. In Example 3.37, we mentioned that the filters used in the MP3-standard were constructed from a lowpass prototype filter by multiplying the filter coefficients with a complex exponential. Clearly this means that the new frequency response is a shift of the old one. The disadvantage is, however, that the new filter coefficients are complex. It is possible to address this problem as follows. Assume that t_k are the filter coefficients of a filter S_1, and that S_2 is the filter with filter coefficients $\cos(2\pi kn/N)t_k$, where $n \in \mathbb{N}$. Show that

$$\lambda_{S_2}(\omega) = \frac{1}{2}(\lambda_{S_1}(\omega - 2\pi n/N) + \lambda_{S_1}(\omega + 2\pi n/N)).$$

In other words, when we multiply (modulate) the filter coefficients with a cosine, the new frequency response can be obtained by shifting the old frequency response with $2\pi n/N$ in both directions, and taking the average of the two.
Solution: We have that

$$\begin{aligned}
\lambda_{S_2}(\omega) &= \sum_k \cos(2\pi kn/N) t_k e^{-ik\omega} = \frac{1}{2}\sum_k (e^{2\pi ikn/N} + e^{-2\pi ikn/N}) t_k e^{-ik\omega} \\
&= \frac{1}{2}\left(\sum_k t_k e^{-ik(\omega - 2\pi n/N)} + \sum_k t_k e^{-ik(\omega + 2\pi n/N)}\right) \\
&= \frac{1}{2}(\lambda_{S_1}(\omega - 2\pi n/N) + \lambda_{S_1}(\omega + 2\pi n/N)).
\end{aligned}$$

10. **a.** Explain what the code below does, line by line.

```
x, fs = audioread('castanets.wav')
N, nchannels = shape(x)
z = zeros((N, nchannels))
for n in range(1,N-1):
    z[n] = 2*x[n+1] + 4*x[n] + 2*x[n-1]
z[0] = 2*x[1] + 4*x[0] + 2*x[N-1]
z[N-1] = 2*x[0] + 4*x[N-1] + 2*x[N-2]
z = z/abs(z).max()
play(z, fs)
```

Comment in particular on what happens in the three lines directly after the `for`-loop, and why we do this. What kind of changes in the sound do you expect to hear?

Solution: In the code a filter is run on the sound samples from the file `castanets.wav`. Finally the new sound is played. In the first two lines after the `for`-loop, the first and the last sound samples in the filtered sound are computed, under the assumption that the sound has been extended to a periodic sound with period `N`. After this, the sound is normalized so that the sound samples lie in the range between -1 and 1. In this case the filter is a lowpass-filter (as we show in b.), so that we can expect that that the treble in the sound is reduced.

b. Write down the compact filter notation for the filter which is used in the code, and write down a 5×5 circulant Toeplitz matrix which corresponds to this filter. Plot the (continuous) frequency response. Is the filter a lowpass- or a highpass filter?

Solution: Compact filter notation for the filter which is run is $\{2,\underline{4},2\}$. A 5×5 circulant Toeplitz matrix becomes

$$\begin{pmatrix} 4 & 2 & 0 & 0 & 2 \\ 2 & 4 & 2 & 0 & 0 \\ 0 & 2 & 4 & 2 & 0 \\ 0 & 0 & 2 & 4 & 2 \\ 2 & 0 & 0 & 2 & 4 \end{pmatrix}.$$

The frequency response is $\lambda_S(\omega) = 2e^{i\omega} + 4 + 2e^{-i\omega} = 4 + 4\cos\omega$. It is clear that this gives a lowpass filter.

c. Another filter is given by the circulant Toeplitz matrix

$$\begin{pmatrix} 4 & -2 & 0 & 0 & -2 \\ -2 & 4 & -2 & 0 & 0 \\ 0 & -2 & 4 & -2 & 0 \\ 0 & 0 & -2 & 4 & -2 \\ -2 & 0 & 0 & -2 & 4 \end{pmatrix}.$$

Express a connection between the frequency responses of this filter and the filter from b. Is the new filter a lowpass- or a highpass filter?

Solution: The frequency response for the new filter is

$$-2e^{i\omega}+4-2e^{-i\omega}=4-4\cos\omega=4+4\cos(\omega+\pi)=\lambda_S(\omega+\pi),$$

where S is the filter from the first part of the exercise. The new filter therefore becomes a highpass filter, since to add π to ω corresponds to swapping the frequencies 0 and π. We could also here refered to Observation 3.40, where we stated that adding an alternating sign in the filter coefficients turns a lowpass filter into a highpass filter and vice versa.

3.6

1. A filter is defined by demanding that $z_{n+2}-z_{n+1}+z_n=x_{n+1}-x_n$.

a. Compute and plot the frequency response of the filter.

b. Use a computer to compute the output when the input vector is $\boldsymbol{x}=(1,2,\dots,10)$. In order to do this you should write down two 10×10-circulant Toeplitz matrices.

3.7

1. Write a function `filterdftimpl`, which takes the filter coefficients $\boldsymbol{t}$ and the value k_0 from this section, computes the optimal M, and implements the filter as here.

2. Factor the filter $S=\{\underline{1},5,10,6\}$ into a product of two filters, one with two filter coefficients, and one with three filter coefficients.

Chapter 4

4.1

1. Consider the matrix

$$S = \frac{1}{3}\begin{pmatrix} 2 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 2 \end{pmatrix}$$

a. Compute the eigenvalues and eigenvectors of S using the results of this section. You should only need to perform one DFT or one DCT in order to achieve this.

b. Use a computer to compute the eigenvectors and eigenvalues of S also. What are the differences from what you found in a.?

c. Find a filter T so that $S = T_r$. What kind of filter is T?

2. Consider the averaging filter $S = \{\frac{1}{4}, \underline{\frac{1}{2}}, \frac{1}{4}\}$. Write down the matrix S_r for the case when $N = 4$.
Solution: First we obtain the matrix S as

$$\left(\begin{array}{cccc|cccc} \frac{1}{2} & \frac{1}{4} & 0 & 0 & 0 & 0 & 0 & \frac{1}{4} \\ \frac{1}{4} & \frac{1}{2} & \frac{1}{4} & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{4} & \frac{1}{2} & \frac{1}{4} & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{4} & \frac{1}{2} & \frac{1}{4} & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & \frac{1}{4} & \frac{1}{2} & \frac{1}{4} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{2} & \frac{1}{4} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \\ \frac{1}{4} & 0 & 0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{2} \end{array}\right)$$

where we have drawn the boundaries between the blocks S_1, S_2, S_3, S_4. From this we see that

$$S_1 = \begin{pmatrix} \frac{1}{2} & \frac{1}{4} & 0 & 0 \\ \frac{1}{4} & \frac{1}{2} & \frac{1}{4} & 0 \\ 0 & \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \\ 0 & 0 & \frac{1}{4} & \frac{1}{2} \end{pmatrix} \quad S_2 = \begin{pmatrix} 0 & 0 & 0 & \frac{1}{4} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \frac{1}{4} & 0 & 0 & 0 \end{pmatrix} \quad (S_2)^f = \begin{pmatrix} \frac{1}{4} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{4} \end{pmatrix}.$$

From this we get

$$S_r = S_1 + (S_2)^f = \begin{pmatrix} \frac{3}{4} & \frac{1}{4} & 0 & 0 \\ \frac{1}{4} & \frac{1}{2} & \frac{1}{4} & 0 \\ 0 & \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \\ 0 & 0 & \frac{1}{4} & \frac{3}{4} \end{pmatrix}.$$

3. As in Example 4.15, state the exact cartesian form of the DCT matrix for the case $N = 3$.

Solution: We first see that $d_{0,3} = \sqrt{\frac{1}{3}}$ and $d_{k,3} = \sqrt{\frac{2}{3}}$ for $k = 1,2$. We also have that

$$\cos\left(2\pi\frac{n}{2N}\left(k+\frac{1}{2}\right)\right) = \cos\left(\pi\frac{n}{3}\left(k+\frac{1}{2}\right)\right),$$

so that the DCT matrix can be written as

$$\begin{aligned}
\mathrm{DCT}_3 &= \begin{pmatrix} \sqrt{\frac{1}{3}} & \sqrt{\frac{1}{3}} & \sqrt{\frac{1}{3}} \\ \sqrt{\frac{2}{3}}\cos\left(\frac{\pi}{3}\frac{1}{2}\right) & \sqrt{\frac{2}{3}}\cos\left(\frac{\pi}{3}\frac{3}{2}\right) & \sqrt{\frac{2}{3}}\cos\left(\frac{\pi}{3}\frac{5}{2}\right) \\ \sqrt{\frac{2}{3}}\cos\left(\frac{2\pi}{3}\frac{1}{2}\right) & \sqrt{\frac{2}{3}}\cos\left(\frac{2\pi}{3}\frac{3}{2}\right) & \sqrt{\frac{2}{3}}\cos\left(\frac{2\pi}{3}\frac{5}{2}\right) \end{pmatrix} \\
&= \begin{pmatrix} \sqrt{\frac{1}{3}} & \sqrt{\frac{1}{3}} & \sqrt{\frac{1}{3}} \\ \sqrt{\frac{2}{3}}\cos(\pi/6) & \sqrt{\frac{2}{3}}\cos(\pi/2) & \sqrt{\frac{2}{3}}\cos(5\pi/6) \\ \sqrt{\frac{2}{3}}\cos(\pi/3) & \sqrt{\frac{2}{3}}\cos(\pi) & \sqrt{\frac{2}{3}}\cos(5\pi/3) \end{pmatrix} \\
&= \begin{pmatrix} \sqrt{\frac{1}{3}} & \sqrt{\frac{1}{3}} & \sqrt{\frac{1}{3}} \\ \sqrt{\frac{2}{3}}(\sqrt{3}/2+i/2) & 0 & \sqrt{\frac{2}{3}}(-\sqrt{3}/2+i/2) \\ \sqrt{\frac{2}{3}}(1/2+\sqrt{3}i/2) & -\sqrt{\frac{2}{3}} & \sqrt{\frac{2}{3}}(1/2-\sqrt{3}i/2) \end{pmatrix}
\end{aligned}$$

4. Show that the vectors $\left\{\cos\left(2\pi\frac{n+\frac{1}{2}}{2N}\left(k+\frac{1}{2}\right)\right)\right\}_{n=0}^{N-1}$ in $\mathbb{R}^N$ are orthogonal, with lengths $\sqrt{N/2}$. This means that the matrix with entries $\sqrt{\frac{2}{N}}\cos\left(2\pi\frac{n+\frac{1}{2}}{2N}\left(k+\frac{1}{2}\right)\right)$ is orthogonal. Since this matrix also is symmetric, it is its own inverse. This is the DCT-IV, which we denote by $\mathrm{DCT}_N^{(\mathrm{IV})}$. Although we will not consider this, the DCT-IV also has an efficient implementation. Hint: Compare with the orthogonal vectors $\boldsymbol{d}_n$, used in the DCT.

Solution: We can write

$$\cos\left(2\pi\frac{n+\frac{1}{2}}{2N}\left(k+\frac{1}{2}\right)\right) = \cos\left(2\pi\frac{2n+1}{4N}\left(k+\frac{1}{2}\right)\right).$$

If we consider these as vectors of length $2N$, we recognize these as the unit vectors $\boldsymbol{d}_{2n+1}$ in the $2N$-dimensional DCT, divided by the factor $d_n = \sqrt{2/(2N)} = \sqrt{1/N}$, so that these vectors have length $\sqrt{N}$. To see that these vectors are orthogonal when we restrict to the first N elements we use property 6.35 as follows:

$$\begin{aligned}
N\delta_{n_1,n_2} &= \sum_{k=0}^{2N-1} \cos\left(2\pi\frac{n_1+\frac{1}{2}}{2N}\left(k+\frac{1}{2}\right)\right)\cos\left(2\pi\frac{n_2+\frac{1}{2}}{2N}\left(k+\frac{1}{2}\right)\right)\\
&= \sum_{k=0}^{N-1} \cos\left(2\pi\frac{n_1+\frac{1}{2}}{2N}\left(k+\frac{1}{2}\right)\right)\cos\left(2\pi\frac{n_2+\frac{1}{2}}{2N}\left(k+\frac{1}{2}\right)\right)\\
&\quad+ \sum_{k=N}^{2N-1} \cos\left(2\pi\frac{n_1+\frac{1}{2}}{2N}\left(k+\frac{1}{2}\right)\right)\cos\left(2\pi\frac{n_2+\frac{1}{2}}{2N}\left(k+\frac{1}{2}\right)\right)\\
&= \sum_{k=0}^{N-1} \cos\left(2\pi\frac{n_1+\frac{1}{2}}{2N}\left(k+\frac{1}{2}\right)\right)\cos\left(2\pi\frac{n_2+\frac{1}{2}}{2N}\left(k+\frac{1}{2}\right)\right)\\
&\quad+ \sum_{k=0}^{N-1} \cos\left(2\pi\frac{n_1+\frac{1}{2}}{2N}\left(k+\frac{1}{2}\right)\right)\cos\left(2\pi\frac{n_2+\frac{1}{2}}{2N}\left(k+\frac{1}{2}\right)\right)\\
&=2 \sum_{k=0}^{N-1} \cos\left(2\pi\frac{n_1+\frac{1}{2}}{2N}\left(k+\frac{1}{2}\right)\right)\cos\left(2\pi\frac{n_2+\frac{1}{2}}{2N}\left(k+\frac{1}{2}\right)\right).
\end{aligned}$$

This shows that

$$\sum_{k=0}^{N-1} \cos\left(2\pi\frac{n_1+\frac{1}{2}}{2N}\left(k+\frac{1}{2}\right)\right)\cos\left(2\pi\frac{n_2+\frac{1}{2}}{2N}\left(k+\frac{1}{2}\right)\right) = \frac{N}{2}\delta_{n_1,n_2},$$

so that the vectors are orthogonal with lengths $\sqrt{N/2}$.

5. The MDCT is defined as the $N\times(2N)$-matrix M with elements $M_{n,k} = \cos(2\pi(n+1/2)(k+1/2+N/2)/(2N))$. This exercise will take you through the details of the transformation which corresponds to multiplication with this matrix. The MDCT is very useful, and is also used in the MP3 standard and in more recent standards.

a. Show that

$$M = \sqrt{\frac{N}{2}}\mathrm{DCT}_N^{(\mathrm{IV})}\begin{pmatrix}\mathbf{0} & A\\ B & \mathbf{0}\end{pmatrix}$$

where A and B are the $(N/2)\times N$-matrices

$$A = \begin{pmatrix}\cdots & \cdots & 0 & -1 & -1 & 0 & \cdots & \cdots\\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots\\ 0 & -1 & \cdots & \cdots & \cdots & \cdots & -1 & 0\\ -1 & 0 & \cdots & \cdots & \cdots & \cdots & 0 & -1\end{pmatrix} = \begin{pmatrix}-I_{N/2}^f & -I_{N/2}\end{pmatrix}$$

$$B = \begin{pmatrix}1 & 0 & \cdots & \cdots & \cdots & \cdots & 0 & -1\\ 0 & 1 & \cdots & \cdots & \cdots & \cdots & -1 & 0\\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots\\ \cdots & \cdots & 0 & 1 & -1 & 0 & \cdots & \cdots\end{pmatrix} = \begin{pmatrix}I_{N/2} & -I_{N/2}^f\end{pmatrix}.$$

Due to this expression, any algorihtm for the DCT-IV can be used to compute the MDCT.

Solution: Clearly, columns $0,\dots,N/2-1$ of the MDCT are columns $N/2,\dots,N-1$ of the DCT-IV. For the remaining columns, note first that, for $0 \le k < N$, the properties

$$\begin{aligned}\cos(2\pi(n+1/2)((2N-1-k)+1/2)/(2N)) &= -\cos(2\pi(n+1/2)(k+1/2)/(2N))\\ \cos(2\pi(n+1/2)((k+2N)+1/2)/(2N)) &= -\cos(2\pi(n+1/2)(k+1/2)/(2N))\end{aligned}$$

are easy to verify. From the first property it follows that columns $N/2,\dots,3N/2-1$ of the MDCT are columns $N-1, N-2,\dots,0$ of the DCT-IV, with a sign change (they occur in opposite order). From the second property, it follows that columns $3N/2,\dots,2N-1$ of the MDCT are columns $0,\dots,N/2-1$ of the DCT-IV, with a sign change. This means that, if $\boldsymbol{y}$ is a vector of length $2N$, the MDCT of $\boldsymbol{y}$ can be written as

$$\sqrt{\frac{N}{2}}\mathrm{DCT}_N^{(\mathrm{IV})}\begin{pmatrix} -y_{3N/2}-y_{3N/2-1}\\ -y_{3N/2+1}-y_{3N/2-2}\\ \vdots\\ -y_{2N-1}-y_N\\ \hline y_0-y_{N-1}\\ y_1-y_{N/2+1}\\ \vdots\\ y_{N/2-1}-y_{N/2}\end{pmatrix}.$$

The factor $\sqrt{\frac{N}{2}}$ was added since $\sqrt{\frac{2}{N}}$ was added in front of the cosine-matrix in order to make $\mathrm{DCT}_N^{(\mathrm{IV})}$ orthogonal. The result now follows by noting that we can write $\begin{pmatrix}\mathbf{0} & A\\ B & \mathbf{0}\end{pmatrix}\boldsymbol{y}$ for the vector on the right hand side, with A and B as defined in the text of the exercise.

b. The MDCT is not invertible, since it is not a square matrix. We will show here that it still can be used in connection with invertible transformations. We first define the IMDCT as the matrix M^T/N. Transposing the matrix expression we obtained in a. gives

$$\frac{1}{\sqrt{2N}}\begin{pmatrix}\mathbf{0} & B^T\\ A^T & \mathbf{0}\end{pmatrix}\mathrm{DCT}_N^{(\mathrm{IV})}$$

for the IMDCT, which thus also has an efficient implementation. Show that if

$$\boldsymbol{x}_0 = (x_0,\dots,x_{N-1}) \qquad \boldsymbol{x}_1 = (x_N,\dots,x_{2N-1}) \qquad \boldsymbol{x}_2 = (x_{2N},\dots,x_{3N-1})$$

and

$$\boldsymbol{y}_{\mathbf{0,1}} = M\begin{pmatrix}\boldsymbol{x}_0\\ \boldsymbol{x}_1\end{pmatrix} \qquad\qquad \boldsymbol{y}_{\mathbf{1,2}} = M\begin{pmatrix}\boldsymbol{x}_1\\ \boldsymbol{x}_2\end{pmatrix}$$

(i.e. we compute two MDCT's where half of the data overlap), then

$$\boldsymbol{x}_1 = \{\text{IMDCT}(\boldsymbol{y_{0,1}})\}_{k=N}^{2N-1} + \{\text{IMDCT}(\boldsymbol{y_{1,2}})\}_{k=0}^{N-1}.$$

Even though the MDCT itself is not invertible, the input can still be recovered from overlapping MDCT's.
Solution: Applying the MDCT first, and then the IMDCT, gives us the matrix

$$\frac{1}{2}\begin{pmatrix} \mathbf{0} & B^T \\ A^T & \mathbf{0} \end{pmatrix}\begin{pmatrix} \mathbf{0} & A \\ B & \mathbf{0} \end{pmatrix} = \frac{1}{2}\begin{pmatrix} B^T B & \mathbf{0} \\ \mathbf{0} & A^T A \end{pmatrix}$$

Note that

$$A^T A = \begin{pmatrix} I_{N/2} & I^f_{N/2} \\ I^f_{N/2} & I_{N/2} \end{pmatrix} \qquad B^T B = \begin{pmatrix} I_{N/2} & -I^f_{N/2} \\ -I^f_{N/2} & I_{N/2} \end{pmatrix}.$$

Inserting this in the above gives

$$\frac{1}{2}\begin{pmatrix} I_{N/2} & -I^f_{N/2} & \mathbf{0} & \mathbf{0} \\ -I^f_{N/2} & I_{N/2} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & I_{N/2} & I^f_{N/2} \\ \mathbf{0} & \mathbf{0} & I^f_{N/2} & I_{N/2} \end{pmatrix} = \frac{1}{2}\begin{pmatrix} I_N - I^f_N & \mathbf{0} \\ \mathbf{0} & I_N + I^f_N \end{pmatrix}.$$

Assume now that we have computed the MDCT of $\begin{pmatrix} \boldsymbol{x}_0 \\ \boldsymbol{x}_1 \end{pmatrix} = (x_0, \ldots, x_{2N-1})$, and of $\begin{pmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \end{pmatrix} = (x_N, \ldots, x_{3N-1})$. Performing the IMDCT on these two thus gives

$$\frac{1}{2}\begin{pmatrix} \boldsymbol{x}_0 - \boldsymbol{x}_0^{\text{rev}} \\ \boldsymbol{x}_1 + \boldsymbol{x}_1^{\text{rev}} \end{pmatrix} \text{ and } \frac{1}{2}\begin{pmatrix} \boldsymbol{x}_1 - \boldsymbol{x}_1^{\text{rev}} \\ \boldsymbol{x}_2 + \boldsymbol{x}_2^{\text{rev}} \end{pmatrix}$$

Adding the second component of the first and the first component of the second gives $\boldsymbol{x}_1$, which proves the result.

4.2

1. Recall that in Exercise 3 in Section 3.1 we wrote down formulas for the output of a filter. Using the results of this section these formulas can be be written in a way which reduces the number of arithmetic operations. Assume that $S = t_{-E}, \ldots, \underline{t_0}, \ldots, t_E$ is a symmetric filter. Use Equation (4.7) to show that $z_n = (S\boldsymbol{x})_n$ in this case can be split into the following different formulas, depending on n:

a. $0 \le n < E$:

$$z_n = t_0 x_n + \sum_{k=1}^{n} t_k(x_{n+k} + x_{n-k}) + \sum_{k=n+1}^{E} t_k(x_{n+k} + x_{n-k+N}). \qquad (4.1)$$

b. $E \leq n < N - E$:

$$z_n = t_0 x_n + \sum_{k=1}^{E} t_k (x_{n+k} + x_{n-k}). \tag{4.2}$$

c. $N - E \leq n < N$:

$$z_n = t_0 x_n + \sum_{k=1}^{N-1-n} t_k (x_{n+k} + x_{n-k}) + \sum_{k=N-1-n+1}^{E} t_k (x_{n+k-N} + x_{n-k}). \tag{4.3}$$

2. Assume that $S = t_{-E}, \ldots, \underline{t_0}, \ldots, t_E$ is a symmetric filter. Write a function `filterS` which takes a symmetric vector `t` and a vector `x` as input, and returns $\boldsymbol{z} = S\boldsymbol{x}$, using the formulas from Exercise 1.
Solution: The code can look like this:

```
def filterS(t,x):
    N = len(x)
    y = zeros(N)
    E = (len(t)+1)/2
    t = t[(E-1):]

    n = 0
    while n<E:
        y[n]= t[0]*x[n]
        for k in range(1, n + 1):
            y[n] = y[n] + t[k]*(x[n + k]+x[n - k])
        for k in range(n + 1, E + 1):
            y[n] = y[n] + t[k]*(x[n + k]+x[n - k + N])
        n = n + 1
    while n < (N-E):
        y[n]= t[0]*x[n]
        for k in range(1, E+1):
            y[n] = y[n] + t[k]*(x[n + k]+x[n - k])
        n = n + 1
    while n<N:
        y[n] = t[0]*x[n]
        for k in range(1,N - n):
            y[n] = y[n] + t[k]*(x[n+k]+x[n-k])
        for k in range(N - n, E + 1):
            y[n] = y[n] + t[k]*(x[n+k-N]+x[n-k])
        n = n + 1
    return y
```

3. Reimplement the function `filterS` from the previous exercise so that it instead performs the filtering using the function `numpy.convolve`. This gives a much nicer implementation, where there is no need to split the implementation into the different formulas stated in Exercise 1.
Solution: The code can look like this:

```
def filterS(t, x):
    tlen = len(t)
    N0 = (tlen - 1)/2
    N = len(x)
    y = concatenate([ x[(N - N0):], x, x[:N0]])
    y = convolve(t, y)
    y = y[(2*N0):(length(y)-2*N0)]
```

4. Repeat Exercise 4 in Section 2.1 by reimplementing the functions `play_with_reduced_treble` and `play_with_reduced_bass` using the function `filterS` from the previous exercise. The resulting sound files should sound the same, since the only difference is that we have modified the way we handle the beginning and end portion of the sound samples.

4.3

1. In this exercise we will take a look at a small trick which reduces the number of additional multiplications we need for DCT algorithm from Theorem 4.23. This exercise does not reduce the order of the DCT algorithms, but we will see in Exercise 2 how the result can be used to achieve this.

a. Assume that $\boldsymbol{x}$ is a real signal. Equation (4.9), which said that

$$\begin{aligned} y_n &= \cos\left(\pi\frac{n}{2N}\right)\Re((\mathrm{DFT}_N\boldsymbol{x}^{(1)})_n) + \sin\left(\pi\frac{n}{2N}\right)\Im((\mathrm{DFT}_N\boldsymbol{x}^{(1)})_n) \\ y_{N-n} &= \sin\left(\pi\frac{n}{2N}\right)\Re((\mathrm{DFT}_N\boldsymbol{x}^{(1)})_n) - \cos\left(\pi\frac{n}{2N}\right)\Im((\mathrm{DFT}_N\boldsymbol{x}^{(1)})_n) \end{aligned}$$

for the n'th and $N-n$'th coefficient of the DCT. This can also be rewritten as

$$\begin{aligned} y_n &= \left(\Re((\mathrm{DFT}_N\boldsymbol{x}^{(1)})_n) + \Im((\mathrm{DFT}_N\boldsymbol{x}^{(1)})_n)\right)\cos\left(\pi\frac{n}{2N}\right) \\ &\quad - \Im((\mathrm{DFT}_N\boldsymbol{x}^{(1)})_n)(\cos\left(\pi\frac{n}{2N}\right) - \sin\left(\pi\frac{n}{2N}\right)) \\ y_{N-n} &= -\left(\Re((\mathrm{DFT}_N\boldsymbol{x}^{(1)})_n) + \Im((\mathrm{DFT}_N\boldsymbol{x}^{(1)})_n)\right)\cos\left(\pi\frac{n}{2N}\right) \\ &\quad + \Re((\mathrm{DFT}_N\boldsymbol{x}^{(1)})_n)(\sin\left(\pi\frac{n}{2N}\right) + \cos\left(\pi\frac{n}{2N}\right)). \end{aligned}$$

Explain that the first two equations require 4 multiplications to compute y_n and y_{N-n}, and that the last two equations require 3 multiplications to compute y_n and y_{N-n}.

b. Explain why the trick in a. reduces the number of additional multiplications in a DCT, from $2N$ to $3N/2$.

c. Explain why the trick in a. can be used to reduce the number of additional multiplications in an IDCT with the same number.
Hint: match the expression $e^{\pi i n/(2N)}(y_n - i y_{N-n})$ you encountered in the IDCT with the rewriting you did in b.

d. Show that the penalty of the trick we here have used to reduce the number of multiplications, is an increase in the number of additional additions from N to $3N/2$. Why can this trick still be useful?

2. (An efficient joint implementation of the DCT and the FFT). In this exercise we will explain another joint implementation of the DFT and the DCT, which has the benefit of a low multiplication count, at the expense of a higher addition count. It also has the benefit that it is specialized to real vectors, with a very structured implementation (this is not always the case for the quickest FFT implementations. Not surprisingly, one often sacrifices clarity of code when one pursues higher computational speed). a. of this exercise can be skipped, as it is difficult and quite technical. For further details of the algorithm the reader is refered to [**?**].

a. Let $\boldsymbol{y} = \mathrm{DFT}_N \boldsymbol{x}$ be the N-point DFT of the real vector $\boldsymbol{x}$. Show that

$$\Re(y_n) = \begin{cases} \Re((\mathrm{DFT}_{N/2}\boldsymbol{x}^{(e)})_n) + (C_{N/4}\boldsymbol{z})_n & 0 \le n \le N/4-1 \\ \Re((\mathrm{DFT}_{N/2}\boldsymbol{x}^{(e)})_n) & n = N/4 \\ \Re((\mathrm{DFT}_{N/2}\boldsymbol{x}^{(e)})_n) - (C_{N/4}\boldsymbol{z})_{N/2-n} & N/4+1 \le n \le N/2-1 \end{cases} \tag{4.4}$$

$$\Im(y_n) = \begin{cases} \Im((\mathrm{DFT}_{N/2}\boldsymbol{x}^{(e)})_n) & n = 0 \\ \Im((\mathrm{DFT}_{N/2}\boldsymbol{x}^{(e)})_n) + (C_{N/4}\boldsymbol{w})_{N/4-n} & 1 \le n \le N/4-1 \\ \Im((\mathrm{DFT}_{N/2}\boldsymbol{x}^{(e)})_n) + (C_{N/4}\boldsymbol{w})_{n-N/4} & N/4 \le n \le N/2-1 \end{cases} \tag{4.5}$$

where $\boldsymbol{x}^{(e)}$ is as defined in Theorem 2.34, where $\boldsymbol{z}, \boldsymbol{w} \in \mathbb{R}^{N/4}$ defined by

$$\begin{aligned} z_k &= x_{2k+1} + x_{N-2k-1} & 0 \le k \le N/4-1, \\ w_k &= (-1)^k (x_{N-2k-1} - x_{2k+1}) & 0 \le k \le N/4-1, \end{aligned}$$

Explain from this how you can make an algorithm which reduces an FFT of length N to an FFT of length $N/2$ (on $\boldsymbol{x}^{(e)}$), and two DCT's of length $N/4$ (on $\boldsymbol{z}$ and $\boldsymbol{w}$). We will call this algorithm the revised FFT algorithm.
Solution: Taking real and imaginary parts in Equation (2.14) for the FFT algorithm we obtain

$$\begin{aligned} \Re(y_n) &= \Re\left((\mathrm{DFT}_{N/2}\boldsymbol{x}^{(e)})_n + \Re((D_{N/2}\mathrm{DFT}_{N/2}\boldsymbol{x}^{(o)})_n\right) \\ \Im(y_n) &= \Im\left((\mathrm{DFT}_{N/2}\boldsymbol{x}^{(e)})_n + \Im((D_{N/2}\mathrm{DFT}_{N/2}\boldsymbol{x}^{(o)})_n\right), \end{aligned}$$

These equations explain the first terms $\Re((\mathrm{DFT}_{N/2}\boldsymbol{x}^{(e)})_n)$ and $\Im((\mathrm{DFT}_{N/2}\boldsymbol{x}^{(e)})_n)$ on the right hand sides in equations (4.4) and (4.5). It remains to rewrite $\Re\left((D_{N/2}\mathrm{DFT}_{N/2}\boldsymbol{x}^{(o)})_n\right)$ and $\Im\left((D_{N/2}\mathrm{DFT}_{N/2}\boldsymbol{x}^{(o)})_n\right)$ so that the remaining terms on the right hand sides can be seen. Let us first consider the equation for the real part with $0 \le n \le N/4-1$. In this case we can write

$$\begin{aligned}
&\Re((D_{N/2}\mathrm{DFT}_{N/2}\boldsymbol{x}^{(o)})_n) \\
&= \Re\left(e^{-2\pi i n/N}\sum_{k=0}^{N/2-1}(\boldsymbol{x}^{(o)})_k e^{-2\pi i nk/(N/2)}\right) = \Re\left(\sum_{k=0}^{N/2-1}(\boldsymbol{x}^{(o)})_k e^{-2\pi i n(k+\frac{1}{2})/(N/2)}\right) \\
&= \sum_{k=0}^{N/2-1}(\boldsymbol{x}^{(o)})_k \cos\left(2\pi\frac{n(k+\frac{1}{2})}{N/2}\right) \\
&= \sum_{k=0}^{N/4-1}(\boldsymbol{x}^{(o)})_k \cos\left(2\pi\frac{n(k+\frac{1}{2})}{N/2}\right) \\
&\quad + \sum_{k=0}^{N/4-1}(\boldsymbol{x}^{(o)})_{N/2-1-k} \cos\left(2\pi\frac{n(N/2-1-k+\frac{1}{2})}{N/2}\right) \\
&= \sum_{k=0}^{N/4-1}((\boldsymbol{x}^{(o)})_k + (\boldsymbol{x}^{(o)})_{N/2-1-k}) \cos\left(2\pi\frac{n\left(k+\frac{1}{2}\right)}{N/2}\right) \\
&= \sum_{k=0}^{N/4-1} z_k \cos\left(2\pi\frac{n\left(k+\frac{1}{2}\right)}{N/2}\right),
\end{aligned}$$

where we have used that cos is periodic with period 2π, that cos is symmetric, and where z is the vector defined in the text of the theorem. When $0 \le n \le N/4-1$ this can also be written as

$$\sum_{k=0}^{N/4-1}(C_{N/4})_{n,k} z_k = (C_{N/4}\boldsymbol{z})_n,$$

This proves the first formula in Equation 4.4.

For $N/4+1 \le n \le N/2-1$, everything above is valid, except for that $\cos(2\pi n(k+1/2)/(N/2))$ are not entries in the matrix $C_{N/4}$, since n is outside the legal range of the indices. However, $N/2-n$ is now a legal index in $C_{N/4}$, and using that

$$\cos\left(2\pi\frac{n(k+\frac{1}{2})}{N/2}\right) = -\cos\left(2\pi\frac{\left(\frac{N}{2}-n\right)\left(k+\frac{1}{2}\right)}{N/2}\right),$$

we arrive at $-(C_{N/4}\boldsymbol{z})_{N/2-n}$ instead, and this proves the third formula in Equation 4.4. For the case $n = \frac{N}{4}$ all the cosine entries in the sum are zero, and this completes the proof of Equation (4.4).

For the imaginary part, using that sin is periodic with period 2π, and that sin is anti-symmetric, analogous calculations as above give

$$\Im((D_{N/2}\mathrm{DFT}_{N/2}\boldsymbol{x}^{(o)})_n) = \sum_{k=0}^{N/4-1}((\boldsymbol{x}^{(o)})_{N/2-1-k} - (\boldsymbol{x}^{(o)})_k)\sin\left(2\pi\frac{n\left(k+\frac{1}{2}\right)}{N/2}\right). \tag{4.6}$$

Using that

$$\sin\left(2\pi\frac{n(k+\frac{1}{2})}{N/2}\right) = \cos\left(\frac{\pi}{2}-2\pi\frac{n(k+\frac{1}{2})}{N/2}\right) = \cos\left(2\pi\frac{(N/4-n)(k+\frac{1}{2})}{N/2}-k\pi\right)$$
$$= (-1)^k\cos\left(2\pi\frac{(N/4-n)(k+\frac{1}{2})}{N/2}\right),$$

Equation 4.6 can be rewritten as

$$\sum_{k=0}^{N/4-1}((\boldsymbol{x}^{(o)})_{N/2-1-k}-(\boldsymbol{x}^{(o)})_k)(-1)^k\cos\left(2\pi\frac{(N/4-n)(k+\frac{1}{2})}{N/2}\right)$$
$$=\sum_{k=0}^{N/4-1}w_k\cos\left(2\pi\frac{(N/4-n)(k+\frac{1}{2})}{N/2}\right),$$

where $\boldsymbol{w}$ is the vector defined as in the text of the theorem. When $n=0$ this is 0 since all the cosines entries are zero. When $1\le n\le N/4$ this is $(C_{N/4}\boldsymbol{w})_{N/4-n}$, since $\cos(2\pi(N/4-n)(k+1/2)/(N/2))$ are entries in the matrix $C_{N/4}$. This proves the second formula in Equation 4.5.

For $N/4\le n\le N/2-1$ we can use that $\cos(2\pi(N/4-n)(k+1/2)/(N/2)) = \cos(2\pi(n-N/4)(k+1/2)/(N/2))$, which is an entry in the matrix $\mathrm{DCT}_{N/4}$ as well, so that we get $(C_{N/4}\boldsymbol{z})_{n-N/4}$. This also proves the third formula in Equation (4.5), and the proof is done.

a. says nothing about the coefficients y_n for $n>\frac{N}{2}$. These are obtained in the same way as before through symmetry. a. also says nothing about $y_{N/2}$. This can be obtained with the same formula as in Theorem 2.34.
Let us now compute the number of arithmetic operations our revised algorithm needs. Denote by the number of real multiplications needed by the revised N-point FFT algorithm

b. Explain from the algorithm in a. that

$$M_N = 2(M_{N/4}+3N/8)+M_{N/2} \qquad A_N = 2(A_{N/4}+3N/8)+A_{N/2}+3N/2 \quad (4.7)$$

HINT: $3N/8$ should come from the extra additions/multiplications (see Exercise 1) you need to compute when you run the algorithm from Theorem 4.23 for $C_{N/4}$. Note also that the equations in a. require no extra multiplications, but that there are xix equations involved, each needing $N/4$ additions, so that we need $6N/4=3N/2$ extra additions.

c. Explain why $x_r = M_{2^r}$ is the solution to the difference equation

$$x_{r+2}-x_{r+1}-2x_r = 3\times 2^r,$$

and that $x_r = A_{2^r}$ is the solution to

$$x_{r+2}-x_{r+1}-2x_r = 9\times 2^r.$$

and show that the general solution to these are $x_r=\frac{1}{2}r2^r+C2^r+D(-1)^r$ for multiplications, and $x_r=\frac{3}{2}r2^r+C2^r+D(-1)^r$ for additions.

d. Explain why, regardless of initial conditions to the difference equations, $M_N = O\left(\frac{1}{2}N\log_2 N\right)$ and $A_N = O\left(\frac{3}{2}N\log_2 N\right)$ both for the revised FFT and the revised DCT. The total number of operations is thus $O(2N\log_2 N)$, i.e. half the operation count of the split-radix algorithm. The orders of these algorithms are thus the same, since we here have adapted to read data.

e. Explain that, if you had not employed the trick from Exercise 1, we would instead have obtained $M_N = O\left(\frac{2}{3}\log_2 N\right)$, and $A_N = O\left(\frac{4}{3}\log_2 N\right)$, which equal the orders for the number of multiplications/additions for the split-radix algorithm. In particular, the order of the operation count remains the same, but the trick from Exercise 1 turned a bigger percentage of the arithmetic operations into additions.

The algorithm we here have developed thus is constructed from the beginning to apply for real data only. Another advantage of the new algorithm is that it can be used to compute both the DCT and the DFT.

3. We did not write down corresponding algorithms for the revised IFFT and IDCT algorithms. We will consider this in this exercise.

a. Using equations (4.4)-(4.5), show that

$$\begin{aligned}\Re(y_n) - \Re(y_{N/2-n}) &= 2(C_{N/4}\boldsymbol{z})_n \\ \Im(y_n) + \Im(y_{N/2-n}) &= 2(C_{N/4}\boldsymbol{w})_{N/4-n}\end{aligned}$$

for $1 \le n \le N/4-1$. Explain how one can compute $\boldsymbol{z}$ and $\boldsymbol{w}$ from this using two IDCT's of length $N/4$.

b. Using equations (4.4)-(4.5), show that

$$\begin{aligned}\Re(y_n) + \Re(y_{N/2-n}) &= \Re((\mathrm{DFT}_{N/2}\boldsymbol{x}^{(e)})_n) \\ \Im(y_n) - \Im(y_{N/2-n}) &= \Im((\mathrm{DFT}_{N/2}\boldsymbol{x}^{(e)})_n),\end{aligned}$$

and explain how one can compute $\boldsymbol{x}^{(e)}$ from this using an IFFT of length $N/2$.