TECHNISCHE UNIVERSITÄT WIEN
Vienna University of Technology

DIPLOMARBEIT

Theory of a massively parallel coherent perfect absorber in a degenerate 4f-cavity

ausgeführt am

Institut für Theoretische Physik
der Technischen Universität Wien

Institute for Theoretical Physics
Vienna University of Technology

unter der Anleitung von

Univ.Prof. Dipl.-Ing. Dr.techn. Stefan Rotter
Univ.Ass. Dipl.-Ing. Kevin Pichler, BSc

durch

Helmut Hörner
Matrikelnummer 08850092
helmut.hoerner@tuwien.ac.at

Wien, 2. August 2021 .... [illegible]

Contents

Abstract

Currently existing implementations of so-called Coherent Perfect Absorbers (CPAs) are in general two-port devices, where two incident beams must be carefully controlled with regard to the exact shape of their transverse mode profile, and also with regard to the mutual phase relations. As a result, these experimental setups require very elaborate and delicate tuning, and even with perfect adjustment they are not able to absorb just any incoming coherent light-beam of matching wavelength, but only carefully engineered wavefronts.

In this thesis we present a new type of massively parallel, coherent perfect absorber in the form of a degenerate 4f-cavity. The according theory is derived and supported by extensive computer simulations. In these numerical simulations, the expected efficiency of a potential experimental implementation is determined. Moreover, the sensitivity to deviations from ideal parameters is explored.

This new type of 4f-cavity-CPA requires just one single port to absorb the incident light wave, and also the incident beam does not need any special wavefront-shaping to be absorbed. Rather, the 4f-cavity-CPA is capable of perfectly absorbing any superposition of a very large number of transverse modes, which is a significant advancement compared to currently existing implementations.

1. Introduction

1.1. Coherent Perfect Absorbers (CPAs)

A Coherent Perfect Absorber (or CPA for short) was first theoretically proposed in an article in *Physical Review Letters* by Chong et al. in 2010 [1]. It is a device that completely absorbs incoming light of a specific wavelength and a specific spatial pattern. In contrast to this, incoming light *not* matching the required conditions is always partly reflected.

In analogy to the operating principle of a laser, a CPA is often referred to as an "Anti-Laser". The meaning of this becomes clear when we recapitulate how a laser works: a laser, in essence, consists of a leaky optical cavity, an active medium acting as gain element, and a pump mechanism that provides the energy for the gain. If the pumping is strong enough such that the gain reaches a critical level, then light is emitted from the cavity in the form of an outgoing monochromatic wave, as sketched in image (a) of figure 1.1.

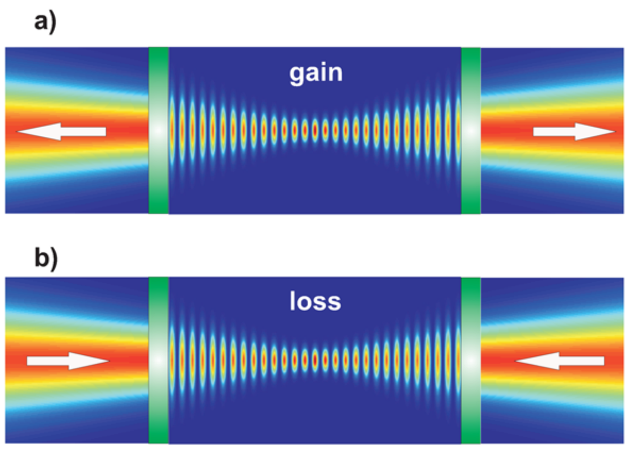

Figure 1.1.: (a) Schematic of a laser oscillator: a gain medium with complex refractive index $n = n' - in''$, embedded in an optical cavity, emits outgoing coherent waves with defined phase and spatial pattern at wavelength λ_0 when the amount of amplification n'' reaches a threshold value. (b) Schematic of the time-reversed laser oscillator that realizes a CPA: when the gain medium is replaced by an absorbing medium with complex refractive index $n^* = n' + in''$, incoming waves with definite phase and spatial pattern [the time-reversed waves of the lasing mode in (a)] are fully absorbed in the medium. Source: [2]

When this process is considered in a time-reversed manner, then the outgoing wavefront with a specific wavelength and specific spatial pattern becomes an incoming wavefront with specific wavelength and spatial pattern, and the gain element providing just the critical gain becomes a loss element providing just the critical loss. This is visualized in image (b) of figure 1.1.

1.2. Existing CPA implementations

The first experimental CPA implementation from 2011 is a two-channel device and is described in [3] as follows:

> “A laser beam [...] enters a beam splitter [...]. The two split beams are directed normally onto opposite sides of a silicon wafer of thickness 110 µm, using a MachZehnder geometry. A phase delay in one of the beam paths controls the relative phase of the two beams. An additional attenuator ensures that the input beams have equal intensities, compensating for imbalances and imperfections. The output beams are rerouted, via beam splitters into a spectrometer.”

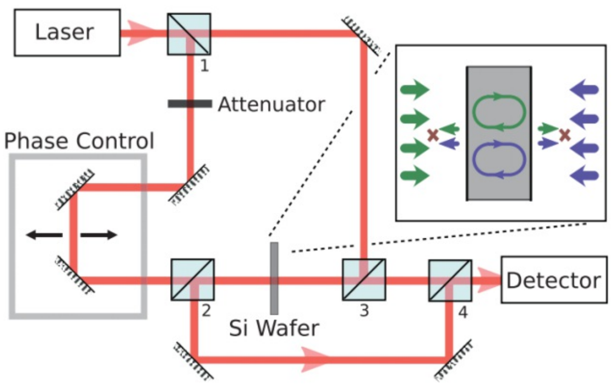

Figure 1.2.: First experimental CPA implementation from 2011. Two split beams are directed onto opposite sides of a silicon wafer with careful attenuation and phase control, such that the reflected and transmitted contributions are cancelled out and the incoming signal is perfectly absorbed. Source: [3]

Another experimentally realized CPA variant is described in [4]. It uses a pair of passive resonators coupled to a microwave transmission line in the background, which can completely absorb light in its parity-time (PT-) symmetric phase.

In recent years a special line of research in the CPA domain has developed around anti-lasing-effects created in disordered materials (“random anti-lasing”). Some examples of this interesting variant are presented in [5–8].

Whereas the requirement of having to carefully control the phase and wavelength of the two incoming light beams can be seen as an inconvenient limitation for non-random CPAs, the core of random anti-lasing experiments lies exactly in the art of carefully controlling and shaping the incident signal, so that perfect absorption occurs even if the absorber material has a random structure.

A proof-of-concept experimental setup is described in [8]: microwaves of a well-defined frequency are generated by a vector network analyser and equally distributed by a power splitter to eight in-phase/quadrature (IQ) modulators, where the amplitudes and relative phases of the signals are independently tuned. These signals are injected into an aluminium waveguide via eight external antennas (four on each side). The central scattering region of the waveguide contains a disordered medium consisting of a set of randomly placed Teflon scatterers. Localized absorption is introduced to the system by placing a monopole antenna with a 50 Ω resistance in the middle of the disordered region (central antenna).

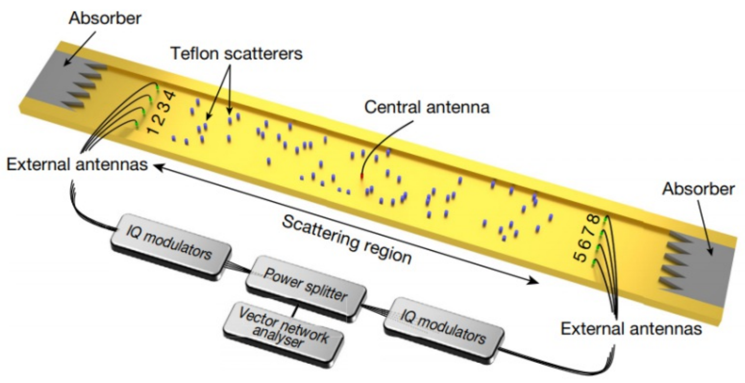

Figure 1.3.: A microwave random CPA as presented in [8].

1.3. The new concept of a mode-independent, universal 4f cavity CPA

The currently existing CPA implementations as presented in the previous section are characterized by the following properties:

- They are usually two-port devices, where the two incident beams must be carefully controlled not only in terms of their wavelength, but also with regard to the exact shape of their transverse mode profile and the corresponding phases.
- The current CPA implementations are not "universal" anti-lasers in the sense that they are not able to absorb any incoming coherent light-beam of matching wavelength, but only carefully engineered wavefronts.

This thesis aims to set the foundation for the implementation of a new type of "universal" CPA with the following characteristics:

- One single port to absorb the incident light wave, so that there is no need to create two specially correlated beams for the anti-lasing effect to occur.
- No need for wavefront shaping.
- Simple, "non-fiddly" hardware design.

How could this be accomplished? Let us, for a moment, consider the initial idea on how to create a CPA: a very simple laser in form of a plane-parallel cavity with a partially-transparent mirror on one side an a perfect mirror on the other side (figure 1.4) becomes a corresponding anti-laser just by replacing the gain-element by a loss-element (figure 1.5).

However, this simple design is obviously not yet the universal anti-laser we are looking for. If, for example, the incident wave is not exactly perpendicular to the xy-plane, the anti-lasing effect breaks down (see figure 1.6).

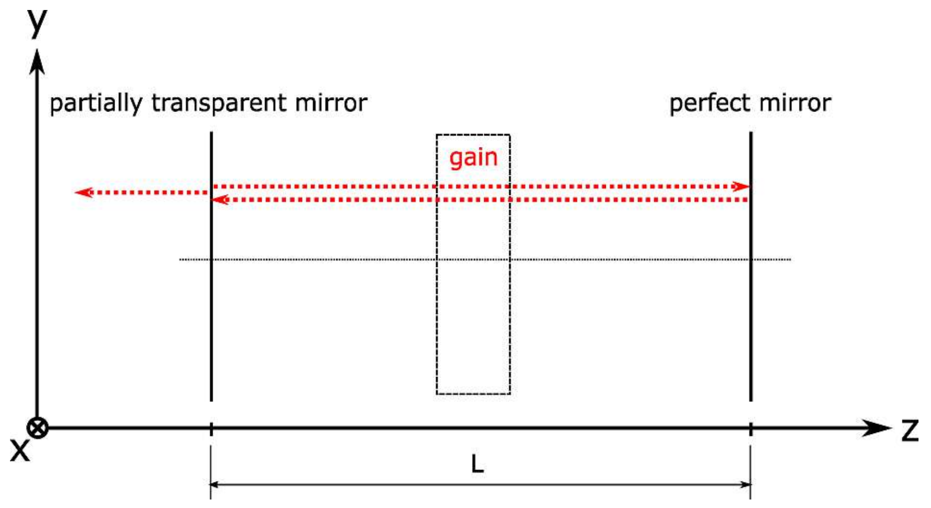

Figure 1.4.: A simple plane-parallel cavity with a gain element acting as a laser.

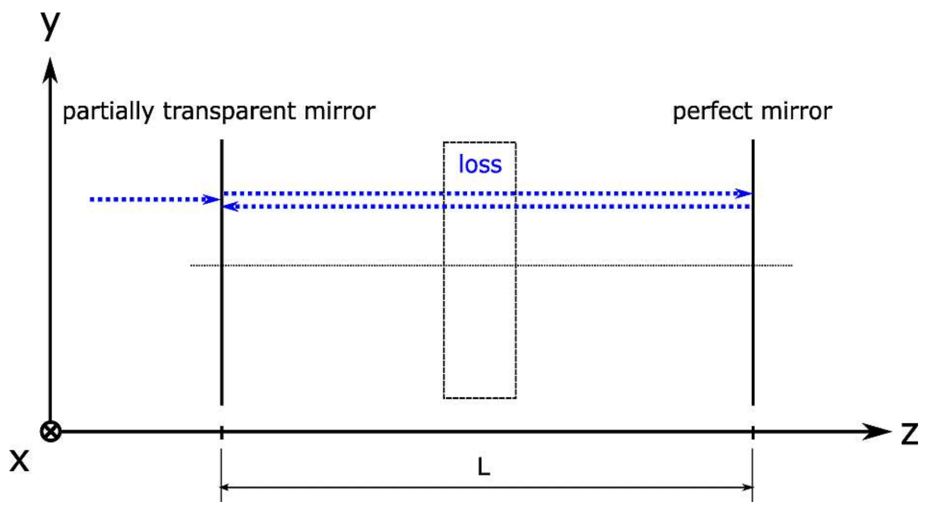

Figure 1.5.: The plane-parallel cavity becomes an anti-laser if the gain is replaced by loss.

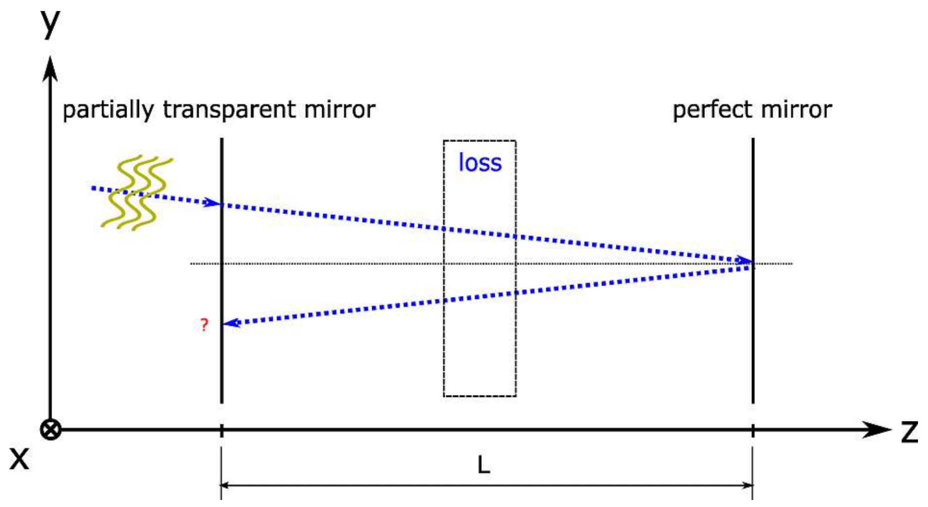

Figure 1.6.: The simple plane-parallel cavity does not act as anti-laser anymore if the incident wave is not exactly perpendicular to the xy-plane.

Thus, we would need a cavity where every ray of light is always reflected into itself, regardless of the incident angle. And — fortunately — there is such a cavity, namely a “4f-cavity”, which is depicted in figure 1.7.

What is new compared to the previous type of cavity are the two converging lenses. The first lens is positioned exactly one focal length f after the left mirror, and the second lens is positioned two focal lengths after the first lens. The right mirror is eventually positioned yet another focal length after the second lens, hence altogether at a distance of four focal lengths after the left mirror.

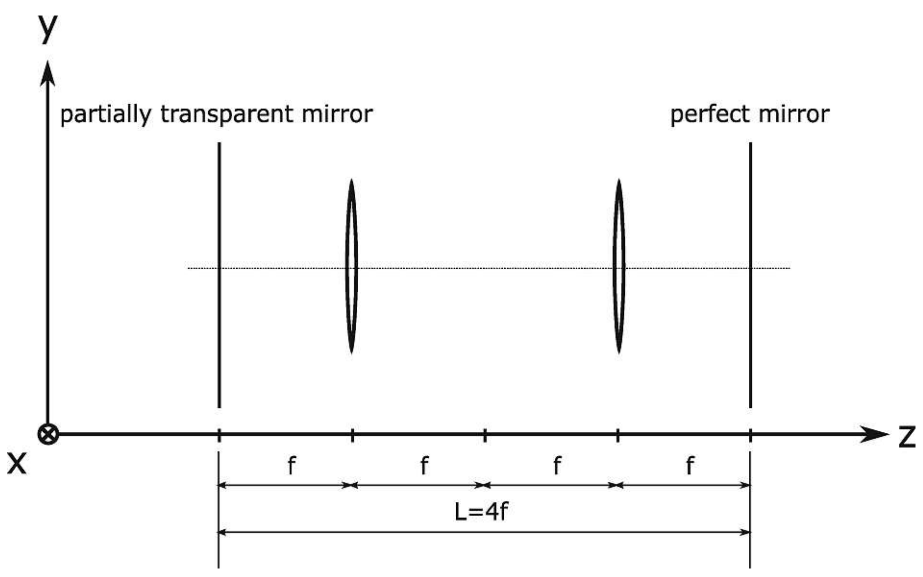

Figure 1.7.: A plane-parallel 4f cavity.

This arrangement ensures that (all lens-errors and other side-effects aside) each beam entering the cavity from the left side through the semi-transparent mirror is always reflected into itself (see figure 1.8).

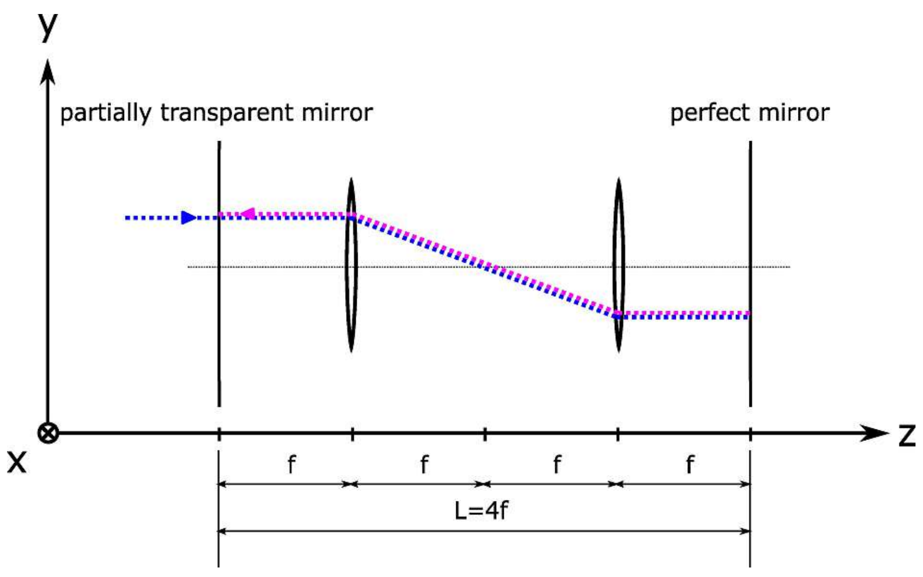

Figure 1.8.: In a 4f cavity, every light-ray is reflected into itself.

This is true not only for rays perpendicular to the xy-plane, but also for incident rays at any arbitrary angle (see figure 1.9).

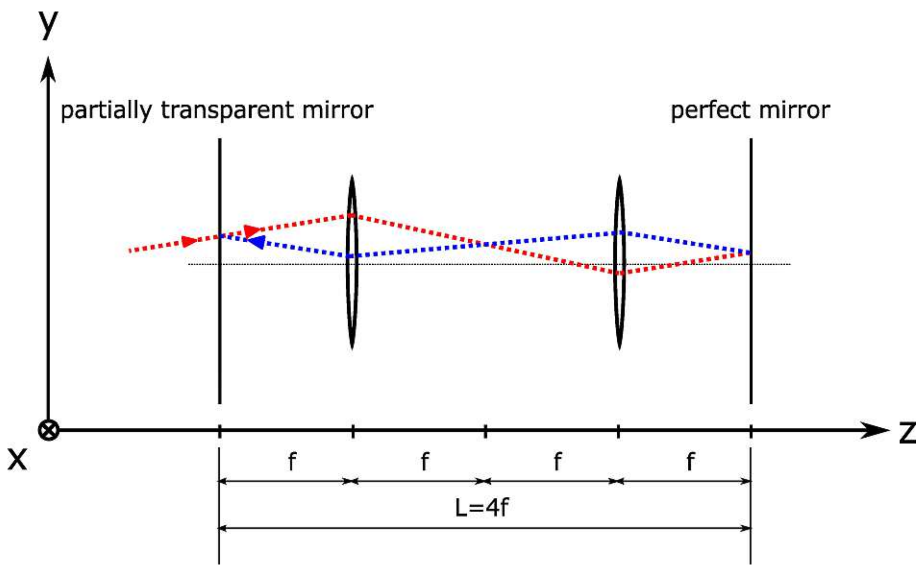

Figure 1.9.: In a 4f-cavity, not only rays perpendicular to the xy-plane, but also rays at any arbitrary angle are reflected into themselves.

By adding the loss element, we get the final, quite simple design of the universal 4f-cavity CPA (see figure 1.10).

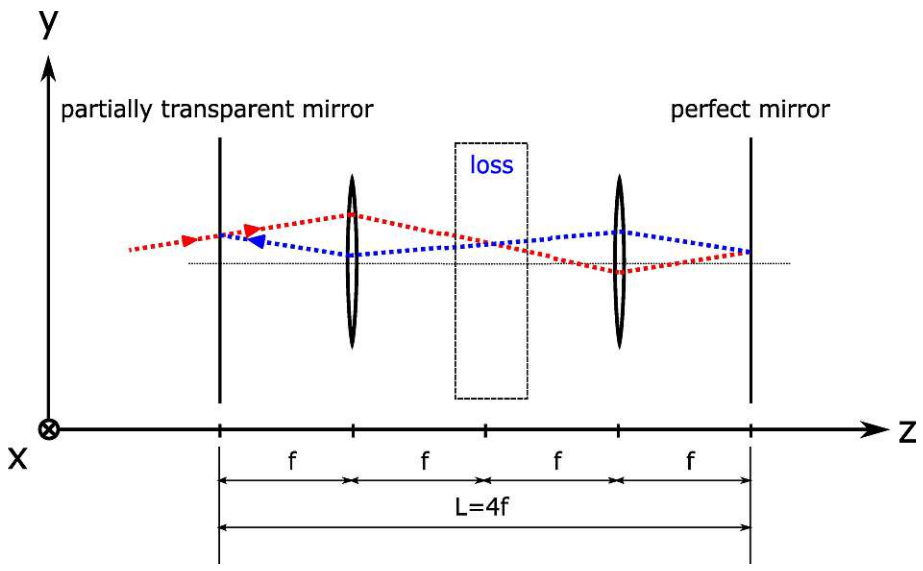

Figure 1.10.: Final design of the universal 4f-cavity CPA (anti-laser).

2. Numerical simulation of light propagation and lenses

2.1. Foundations of scalar wave optics and Fourier optics

2.1.1. The Helmholtz equation

Our starting point is the classical theory of electrodynamics, with Maxwell's equations [9] being the central framework (all calculations in SI units):

$$\vec{\nabla} \cdot \vec{E} = \frac{\rho}{\epsilon_r \epsilon_0} \tag{2.1}$$

$$\vec{\nabla} \cdot \vec{B} = 0 \tag{2.2}$$

$$\vec{\nabla} \times \vec{E} = -\frac{\partial \vec{B}}{\partial t} \tag{2.3}$$

$$\vec{\nabla} \times \vec{B} = \mu_r \mu_0 \vec{j} + \mu_r \mu_0 \epsilon_r \epsilon_0 \frac{\partial \vec{E}}{\partial t} \tag{2.4}$$

In these equations, $\vec{E}$ is the electric field, $\vec{B}$ the magnetic field, ρ the electric charge density, and $\vec{j}$ the electric current density. The symbols ϵ_0 and μ_0 represent the permittivity and the permeability of free space, and ϵ_r and μ_r are the relative permittivity and permeability as material constants to represent the behavior of electromagnetic fields in matter.

We deal with optical wave propagation through linear, isotropic, homogeneous, nondispersive, dielectric media in the absence of source charges and currents. Therefore, we can set $\epsilon_r = \text{const}$, scalar and $\mu_r = 1$, $\rho = 0$, and $j = 0$, and get the following, simplified set of equations:

$$\vec{\nabla} \cdot \vec{E} = 0 \tag{2.5}$$
$$\vec{\nabla} \cdot \vec{B} = 0 \tag{2.6}$$
$$\vec{\nabla} \times \vec{E} = -\frac{\partial \vec{B}}{\partial t} \tag{2.7}$$
$$\vec{\nabla} \times \vec{B} = \mu_0 \epsilon_r \epsilon_0 \frac{\partial \vec{E}}{\partial t} \tag{2.8}$$

As demonstrated in appendix A.1, one can derive from equations (2.5) to (2.8) a steady state differential equation for the electromagnetic field, which is the Helmholtz equation:

$$\vec{\nabla}^2 U(\vec{r}) + k^2 U(\vec{r}) = 0. \tag{2.9}$$

The scalar function $U(\vec{r}, t)$ in equation (2.9) stands for any of the E_i or B_i components of the electric or magnetic field.

2.1.2. The Rayleigh-Sommerfeld transfer function

The Helmholtz equation (2.9) allows us to reduce the vectorial calculation of light propagation in empty space (i.e., calculating the propagation of the vector-fields $\vec{E}$ and $\vec{B}$) to the more simple problem of calculating the propagation of the complex-valued scalar field $U(\vec{r})$ without much loss of generality.

This is justified because, in the problem at hand, effects arising from the vectorial nature of the $\vec{E}$ and $\vec{B}$ fields, like — for example — polarization effects, can be neglected. Also, as we are only interested in the electromagnetic field's stationary field intensity, it is no problem that we dumped the separated $e^{-i\omega t}$ part of the wave equation in the derivation of the Helmholtz equation.

We now suppose that a light wave from a monochromatic source is incident on the transverse xy-plane traveling in the positive z-direction towards the 4f cavity. The complex field across the $z = 0$ plane is represented by $U(x,y;0)$. Following the derivation in [10, p. 588], it is our goal to calculate the field $U(x,y;z)$ that appears at another parallel xy-plane at a distance z to the right of the first plane.

The field $U(x,y;z)$ has a two-dimensional Fourier transform given by

$$A(k_x, k_y; z) = \mathscr{F}\left(U(x, y; z)\right)$$
$$A(k_x, k_y; z) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} U(x, y; z) e^{-i(k_x x + k_y y)}\, \mathrm{d}x\, \mathrm{d}y \tag{2.10}$$

$A(k_x, k_y; z)$ is called the *angular spectrum* of $U(x, y; z)$. Accordingly, $U(x, y; z)$ can be represented as the Inverse Fourier Transform of its angular spectrum by

$$\begin{aligned} U(x, y; z) &= \mathscr{F}^{-1}\left(A(k_x, k_y; z)\right) \\ U(x, y; z) &= \frac{1}{(2\pi)^2} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} A(k_x, k_y; z) e^{i(k_x x + k_y y)} \, \mathrm{d}k_x \, \mathrm{d}k_y \end{aligned} \tag{2.11}$$

$U(x, y; z)$ has to fulfill the Helmholtz equation (2.9). Inserting equation (2.11) into equation (2.9) leads to (see appendix A.2):

$$\frac{\partial^2 A}{\partial z^2} + \left(k^2 - k_x^2 - k_y^2\right) A = 0 \tag{2.12}$$

Equation (2.12) is a differential equation of the form $A''(z) + \alpha A(z) = 0$ with $\alpha = k^2 - k_x^2 - k_y^2$. Its characteristic polynomial $\lambda^2 + \alpha = 0$ has the two solutions $\lambda = \pm i\sqrt{\alpha} = \pm i\sqrt{k^2 - k_x^2 - k_y^2}$. Therefore, the general solution of the differential equation (2.12) is given by a superposition of

$$A = A_+ \, e^{+iz\sqrt{k^2 - k_x^2 - k_y^2}} \tag{2.13}$$

$$A = A_- \, e^{-iz\sqrt{k^2 - k_x^2 - k_y^2}} \tag{2.14}$$

According to physics convention, equation (2.13) represents a wave propagating in positive z-direction, and equation (2.14) represents a wave propagating in negative z-direction. Our goal was to describe the field $U(x,y; z)$ that appears on a parallel xy-plane at a distance z to the *right* of the first plane with $U(x,y; 0)$. Therefore, we continue with equation (2.13) as the forward-propagating solution.

For $z = 0$ we demand $A|_{z=0} = A(k_x, k_y; 0)$. By replacing $A|_{z=0}$ with equation (2.13) while setting $z = 0$, this boundary condition simply becomes $A_+ = A(k_x, k_y; 0)$, and hence the final solution is

$$A(k_x, k_y; z) = A(k_x, k_y; 0) \, e^{+iz\sqrt{k^2 - k_x^2 - k_y^2}}. \tag{2.15}$$

This is known as the Rayleigh-Sommerfeld transfer function.

With equations (2.11) and (2.21) it is possible to write down the relation between the source image plane $U(x,y;0)$ and the observation plane $U(x,y;z)$ in position space (cf. figure 2.1):

$$U(x,y;z) = \mathscr{F}^{-1}\left(\mathscr{F}\left(U(x,y;0)\right)\; e^{iz\sqrt{k^2-k_x^2-k_y^2}}\right) \tag{2.16}$$

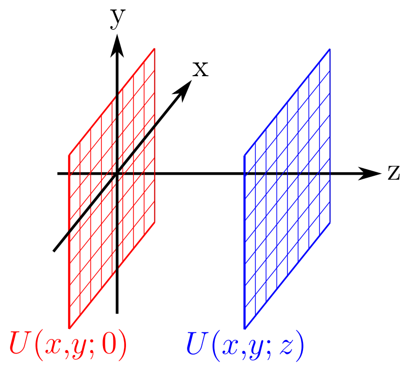

Figure 2.1.: Given a field on the source plane $U(x,y;0)$, we can calulate the field on the observation plane $U(x,y;z)$.

The numerical, algorithmic equivalent of equations (2.15) and (2.16) — using discrete (x,y)-coordinate values and the Fast-Fourier-Transform (FFT) method instead of the Fourier transformation — will become the foundation of our numerical light propagation simulation, as presented in section 2.3.

2.1.3. The Fresnel transfer function

The Fresnel transfer function is a paraxial approximation of the Rayleigh-Sommerfeld transfer function. The reason we are considering this approximation is mainly because it will help us to determine the optimal parameters for the discretised (x,y)-grid.

To derive the Fresnel transfer function, let us review the Rayleigh-Sommerfeld transfer function from equation (2.15):

$$A(k_x,k_y;z) = A(k_x,k_y;0)\; e^{+iz\sqrt{k^2-k_x^2-k_y^2}}$$

This can be expressed as

$$A(k_x,k_y;z) = A(k_x,k_y;0)\; e^{+i\phi(k_r)} \tag{2.17}$$

with

$$\phi(k_r) = z\sqrt{k^2 - k_r^2} \quad \text{and} \tag{2.18}$$
$$k_r^2 \overset{\text{def}}{=} k_x^2 - k_y^2 \tag{2.19}$$

Developing equation (2.18) into a series around $k_r = 0$ results in

$$\phi(k_r) = kz - \frac{zk_r^2}{2k} + O^4(k_r) \tag{2.20}$$

Inserting this expression into equation (2.17) gives

$$\begin{aligned}
A(k_x, k_y; z) &\approx A(k_x, k_y; 0)\ e^{+i\left(kz - \frac{zk_r^2}{2k}\right)} \\
A(k_x, k_y; z) &\approx A(k_x, k_y; 0)\ e^{ikz} e^{-i\frac{z}{2k}k_r^2} \overset{(2.19)}{\Longrightarrow} \\
A(k_x, k_y; z) &\approx A(k_x, k_y; 0)\ e^{ikz} e^{-i\frac{z}{2k}\left(k_x^2 + k_y^2\right)} \qquad \Big| k = \frac{2\pi}{\lambda} \\
A(k_x, k_y; z) &\approx A(k_x, k_y; 0)\ e^{ikz} e^{-i\frac{\lambda z}{4\pi}\left(k_x^2 + k_y^2\right)}
\end{aligned} \tag{2.21}$$

Equation (2.21) is a paraxial approximation of equation (2.15), known as the Fresnel transfer function. As mentioned in the introduction to this section, we will use this result in section 2.6 to derive the optimal parameters of the (x,y)-grid.

2.1.4. Thin spherical lenses

To numerically simulate a 4f-cavity as depicted in figure 1.10, it is not only necessary to simulate light propagation between the lenses, or through the absorber, but we also need to be able to simulate the optical behavior of the two lenses of the 4f-cavity.

A common method for simulating lenses in numerical simulations is the so-called "thin lens approximation". An optical lens is called a thin lens if a light ray entering at coordinates (x, y) on one face exits at approximately the same coordinates on the opposite face, i.e., if there is negligible translation of a ray within the lens. Following the derivation in [11, p. 96ff], we will derive in this section how to simulate a thin *spherical* lens.

To understand how to correctly model the behavior of a thin spherical lens mathematically, we first consider an actual spherical lens with a finite thickness Δ_0 along its optical axis. In case of a convergent lens, the thickness becomes smaller the farther away from the optical axis we are, and hence can be expressed by a *thickness function* $\Delta(x,y)$ (see figure 2.2).

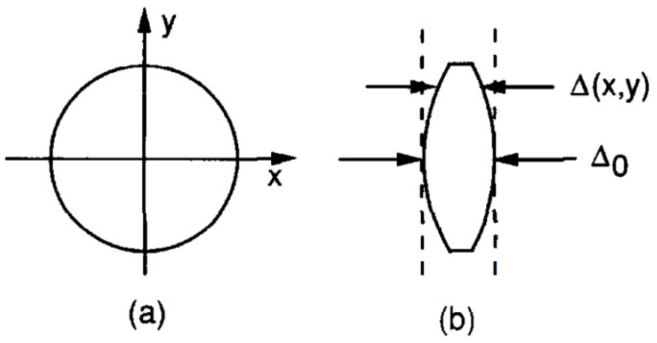

Figure 2.2.: The Thickness Function: (a) front view, (b) side view. Source: [11, p. 97]

Light passing the lens at the optical axis experiences a phase delay of $\phi(0,0) = kn\Delta_0$, with n being the refractive index of the lens material. On the other hand, a light-beam entering the lens at a certain (x,y)-position only passes a shorter distance $\Delta(x,y)$ through the lens, acquiring a phase delay $\phi_{lens} = kn\Delta(x,y)$, and in addition it has to cover the remaining distance $\Delta_0 - \Delta(x,y)$ through air, thereby acquiring a further phase delay $\phi_{air} = k(\Delta_0 - \Delta(x,y))$. Hence, the total phase delay can be expressed as

$$\begin{aligned}
\phi(x,y) &= \phi_{lens}(x,y) + \phi_{air}(x,y) \\
\phi(x,y) &= kn\Delta(x,y) + k(\Delta_0 - \Delta(x,y)) \\
\phi(x,y) &= kn\Delta(x,y) + k\Delta_0 - k\Delta(x,y) \\
\phi(x,y) &= k\Delta_0 + (kn-k)\Delta(x,y) \\
\phi(x,y) &= k\Delta_0 + k(n-1)\Delta(x,y) \qquad (2.22)
\end{aligned}$$

The complex field across the xy-plane $U'(x,y)$ directly behind the lens is then related to the incident complex field $U(x,y)$ directly in front of the lens by

$$\begin{aligned}
U'(x,y) &= U(x,y)e^{i\phi(x,y)} \overset{(2.22)}{\Longrightarrow} \\
U'(x,y) &= U(x,y)e^{ik\Delta_0 + ik(n-1)\Delta(x,y)} \\
U'(x,y) &= U(x,y)e^{ik\Delta_0}e^{ik(n-1)\Delta(x,y)} \qquad (2.23)
\end{aligned}$$

To determine the thickness function $\Delta(x{,}y)$, we split the whole lens into three parts as shown in figure 2.3, and write the total thickness function as the sum of the three individual thickness functions:

$$\Delta(x{,}y) = \Delta_1(x{,}y) + \Delta_2(x{,}y) + \Delta_3(x{,}y) \tag{2.24}$$

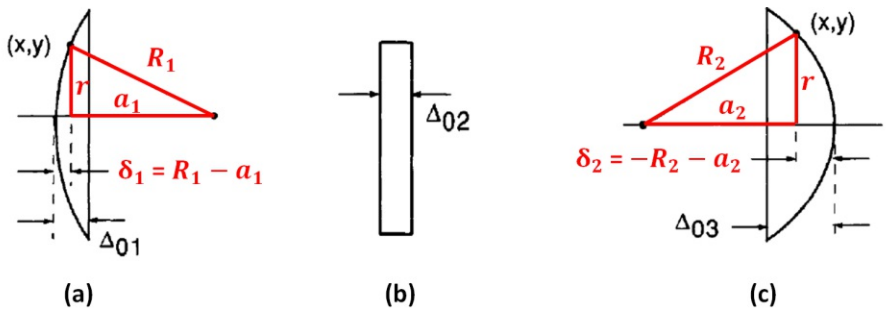

Figure 2.3.: (a) Geometry for Δ_1, (b) Geometry for Δ_2, (c) Geometry for Δ_3. From: [11, p. 98] with color edits

Looking at image (a) of figure 2.3, the thickness function of the first lens segment $\Delta_1(x{,}y)$ can be derived as (see appendix A.3):

$$\Delta_1(x{,}y) = \Delta_{01} - R_1\left(1 - \sqrt{1 - \frac{x^2+y^2}{R_1^2}}\right) \tag{2.25}$$

The middle lens segment as depicted in image (b) of figure 2.3 simply has a constant thickness Δ_{02}:

$$\Delta_2(x{,}y) = \Delta_{02} \tag{2.26}$$

The derivation of the thickness function of the third lens segment Δ_3 as depicted in image (c) of figure 2.3 is equivalent to the derivation of Δ_1, with the only difference that the radii of convex and concave curvature must have different signs.

In accordance with [11] we adopt the sign convention that as rays travel from left to right, each convex surface encountered is taken to have a positive radius of curvature, while each concave surface is taken to have a negative radius of curvature. Thus in figure 2.3 the radius of curvature of the left-hand surface of the lens is a *positive* number R_1, while the radius of curvature of the right-hand surface is a *negative* number R_2.

This leads to the following result (see appendix A.3):

$$\Delta_3(x,y) = \Delta_{02} + R_2 \left(1 - \sqrt{1 - \frac{x^2 + y^2}{R_2^2}}\right) \tag{2.27}$$

By inserting equations (2.25) to (2.27) into equation (2.24), and taking into account that $\Delta_0 = \Delta_{01} + \Delta_{02} + \Delta_{03}$, we can write the total thickness function to be:

$$\Delta(x,y) = \Delta_0 - R_1 \left(1 - \sqrt{1 - \frac{x^2 + y^2}{R_1^2}}\right) + R_2 \left(1 - \sqrt{1 - \frac{x^2 + y^2}{R_2^2}}\right) \tag{2.28}$$

If we assume that $x,y \ll R_1, R_2$ (paraxial approximation), then $\sqrt{1 - \frac{x^2+y^2}{R^2}} \approx 1 - \frac{x^2+y^2}{2R^2}$. Inserting this into equation (2.28) results in the simplified expression (see appendix A.4):

$$\Delta(x,y) = \Delta_0 - \frac{x^2 + y^2}{2} \left(\frac{1}{R_1} - \frac{1}{R_2}\right) \tag{2.29}$$

By simply re-arranging the *Lensmaker Equation* $\frac{1}{f} = (n-1)\left(\frac{1}{R_1} - \frac{1}{R_2}\right)$ for a thin lens in air [12, p. 14] we get:

$$\left(\frac{1}{R_1} - \frac{1}{R_2}\right) = \frac{1}{f\,(n-1)} \tag{2.30}$$

Inserting equation (2.30) into equation (2.29) and calculating the limit $\Delta_0 \to 0$ (thin lens approximation) finally results in

$$\begin{aligned}
&\Delta(x,y) = \Delta_0 - \frac{x^2 + y^2}{2} \frac{1}{f\,(n-1)} \overset{(2.23)}{\Longrightarrow} \\
&U'(x,y) = U(x,y)\ e^{ik\Delta_0}\ e^{ik(n-1)\left(\Delta_0 - \frac{x^2+y^2}{2} \frac{1}{f(n-1)}\right)} \\
&U'(x,y) = U(x,y)\ e^{ik\Delta_0}\ e^{ik(n-1)\Delta_0}\ e^{-ik\cancel{(n-1)}\frac{x^2+y^2}{2f\cancel{(n-1)}}} \qquad \qquad |\Delta_0 \to 0 \\
&U'(x,y) = U(x,y)\ e^{-i\frac{k}{2f}\left(x^2+y^2\right)} \qquad (2.31)
\end{aligned}$$

In equation (2.31) the complex scalar function $U(x,y)$ represents the light field immediately before it has passed the lens, and $U'(x,y)$ the field immediately after it has passed the lens.

2.1.5. Thin aspherical perfect lenses

Spherical lenses suffer from spherical aberration, i.e., parallel rays of incident light do not perfectly converge at the focal point. The farther away the light ray from the optical axis, the greater the deviation.

However, it is possible to shape an aspherical perfect lens so that all incident light rays parallel to the optical axis converge at the focal point. In this section we derive how to calculate such an aspherical, perfect lens in the thin lens approximation.

Figure 2.4 shows how an exemplary incident light ray enters the lens parallel to the optical axis from the left side. The corresponding wavefront is depicted in light blue. The ray then gets refracted so that it meets the optical axis exactly at the focal point $z = f$. The path length between the right face of the lens and the focal point for a ray entering the lens at position (x, y) is $\sqrt{x^2 + y^2 + f^2}$.

A ray entering the lens at $(x,y) = (0,0)$ has to cross the distance f between the right side of the lens and the focal point. In contrast, a ray entering the lens at an arbitrary position (x,y) has to cross a longer distance $\sqrt{x^2 + y^2 + f^2}$ between the right side of the lens and the focal point f.

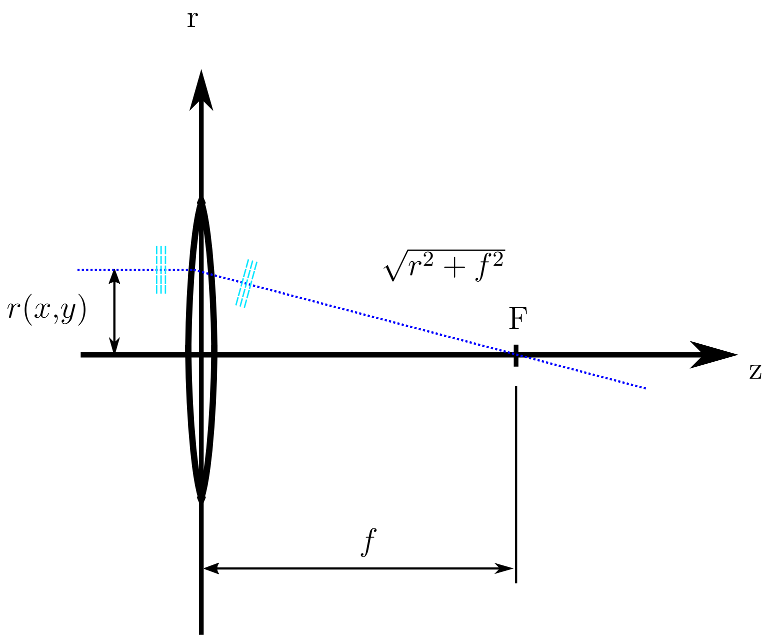

Figure 2.4.: A perfect, aspherical lens focuses every incident light ray parallel to the optical axis into the focal point F. The path length between the right face of the lens and the focal point for a ray passing the lens at position (x, y) is $\sqrt{x^2 + y^2 + f^2}$. This leads directly to the required phase shift.

Therefore, the difference in path length is

$$\Delta s(x,y) = \sqrt{x^2 + y^2 + f^2} - f \tag{2.32}$$

This means that we need to add a *negative* phase shift at position (x,y) to compensate for the extra distance $\Delta s(x,y)$.

$$\begin{aligned} \Delta\phi(x,y) &= -k\ \Delta s(x,y) \overset{(2.32)}{\Longrightarrow} \\ \Delta\phi(x,y) &= -k\left(\sqrt{x^2+y^2+f^2} - f\right) \end{aligned} \tag{2.33}$$

Therefore we can write the final solution

$$\begin{aligned} U'(x,y) &= U(x,y)\ e^{i\Delta\phi(x,y)} \overset{(2.33)}{\Longrightarrow} \\ U'(x,y) &= U(x,y)\ e^{-ik(\sqrt{x^2+y^2+f^2}-f)} \end{aligned} \tag{2.34}$$

In equation (2.34) the complex scalar function $U(x,y)$ represents the light field immediately before it has passed the lens, and $U'(x,y)$ the light field immediately after it has passed the lens.

Equation (2.34) follows directly from equation (2.33). The latter can be verified by using the integral formula [13, eq. (13)]. It allows to calculate the necessary phase delay $\Delta\phi(r)$ for any given function $f(r)$ with $r = \sqrt{x^2+y^2}$:

$$\begin{aligned} \Delta\phi(r) &= \int_0^r \frac{-kr'}{\sqrt{f^2(r') + r'^2}}\,\mathrm{d}r' && \Big| f(r') = f \\ \Delta\phi(r) &= \int_0^r \frac{-kr'}{\sqrt{f^2 + r'^2}}\,\mathrm{d}r' \quad \overset{\text{(appendix A.5)}}{\Longrightarrow} \\ \Delta\phi(r) &= -k\left(\sqrt{r^2+f^2} - f\right) && \Big| r^2 = x^2 + y^2 \\ \Delta\phi(r) &= -k\left(\sqrt{x^2+y^2+f^2} - f\right) \end{aligned} \tag{2.35}$$

Equation (2.35) coincides with the independently derived equation (2.33).

2.1.6. Wavefront tilt

For an experimental implementation of the 4f-cavity it is interesting to know the effect of one of the mirrors not being perfectly aligned (see section 4.4.4). For this estimation, we need to be able to simulate a mirror whose orientation deviates by a small angle from the plane-parallel position. The basis for this is that we are able to apply a "tilt" to a beam wavefront; in our implementation specifically a tilt α relative to the z-axis in the yz-plane, as depicted in figure 2.5.

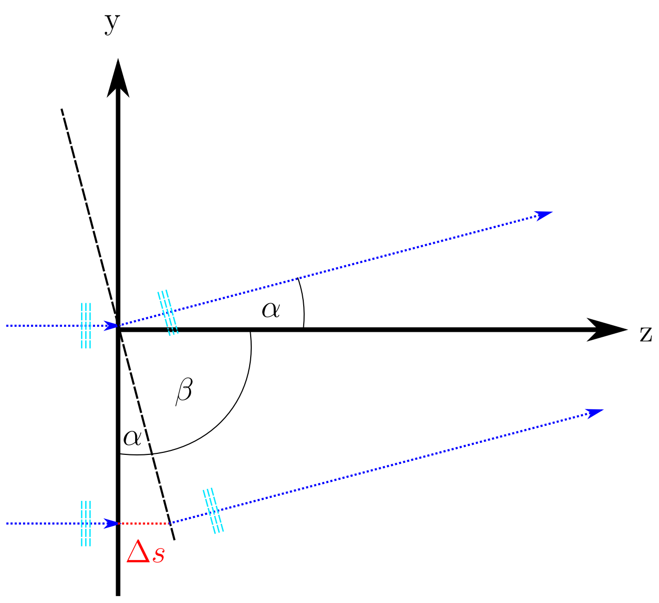

Figure 2.5.: The incident wavefront experiences a tilt α relative to the z-axis in the yz-plane.

The black, dashed line in figure 2.5 indicates the tilted wavefront, orthogonal to the tilted direction of propagation. The not-yet-tilted wavefront at negative y-positions has to pass the extra path-length Δs (indicated in red in figure 2.5). This extra path-length Δs is a linear function of y, and is determined by the black, dashed line expressed as $z = f(y)$, wich can be derived in the following way:

$$\begin{aligned}
y(z) &= -z\tan(\beta) && \Big|\beta = \frac{\pi}{2} - \alpha \\
y(z) &= -z\tan\left(\frac{\pi}{2} - \alpha\right) \\
y(z) &= -z\cot(\alpha) \\
z(y) &= -y\frac{1}{\cot(\alpha)} \\
z(y) &= -y\tan(\alpha) && |z(y) = \Delta s \\
\Delta s &= -y\tan(\alpha) && (2.36)
\end{aligned}$$

Next we need to replace Δs by a phase quantity. If Δs is *positive*, then this means that the light-front has to pass an extra path-length Δs, and the wave-function $e^{i(kz-\omega t)}$ needs to be corrected by a *negative* phase-shift. Analogously, if Δs is *negative*, then the corresponding phase-shift must be *positive*. This means we have to reverses the sign when switching from Δs to the phase representation:

$$\begin{aligned}&\Delta\phi = -k\Delta s \quad \overset{(2.36)}{\Longrightarrow}\\ &\Delta\phi = ky\tan(\alpha)\end{aligned} \tag{2.37}$$

Therefore we can write the final solution

$$\begin{aligned}&U'(x,y) = U(x,y)\ e^{i\Delta\phi(y)} \overset{(2.37)}{\Longrightarrow}\\ &U'(x,y) = U(x,y)\ e^{iky\tan(\alpha)}\end{aligned} \tag{2.38}$$

In equation (2.38) the complex scalar function $U(x,y)$ represents the light field immediately before the tilt, and $U'(y)$ the light field immediately after the tilt.

2.2. Software functionality around 2D FFT

Fourier optics provides an extremely efficient computational method for simulating wave optics problems, and the method is a widely used approach in the domain of simulating optical phenomena like wave diffraction, wave propagation, and many other effects.

The reason for the efficiency of the method (compared to, e.g., using numerical integration techniques) is the efficiency of the Fast-Fourier-Transform (FFT) algorithm, which substitutes the analytical Fourier transformations in the various equations derived in section 2.1. As FFT is at the core of Fourier optics, we have implemented a set of base functions dealing with certain subtleties around this topic.

2.2.1. Sign conventions

When it comes to Fourier Transformation there is generally considerable confusion with regard to the signs. This confusion is caused by the fact that there are two different sign conventions. In this thesis we will stick to the "Physics Positive Sign Convention" as follows:

Plane wave propagating in positive z-direction:

$$U = U_0 e^{+ikz} \tag{2.39}$$

Complex refractive index (with $\kappa > 0$ representing loss):

$$\tilde{n} = n + i\kappa \tag{2.40}$$

Spatial Fourier Transformation:

$$F(k) = \int_{-\infty}^{\infty} f(x) e^{-ikx}\, \mathrm{d}x \tag{2.41}$$

Inverse Spatial Fourier Transformation:

$$f(x) = \frac{1}{2\pi} \int_{-\infty}^{\infty} F(k) e^{+ikx}\, \mathrm{d}k \tag{2.42}$$

2.2.2. Position-space grid and xy-coordinates

As shown in figure 2.1, the field $U(x,y;z)$ extends over the xy-plane at a certain z-position. In a numerical computer simulation, this continuous light field $U(x,y;z)$

must be represented by an array of discrete complex values. Each entry in this array then represents the complex value of $U(x,y;z)$ at a certain, discrete $(x,y;z)$ coordinate point.

Instead of the analytic, non-periodic Fourier Transformation, a 2D Fast Fourier Transform (FFT) algorithm is to be used. For an $N \times N$ grid, the FFT produces N^2 complex Fourier coefficients, which are usually also stored in an array of equal size.

Although it is possible to choose an odd number for N, FFT works most efficiently if N is an even number. To visualize what this means for the grid position and the corresponding coordinates, let us assume an 8×8 grid which is to represent a quadratic area of 2 mm × 2 mm. Figure 2.6 visualizes how such a grid would be placed (almost) centered along the optical axis on the xy-plane.

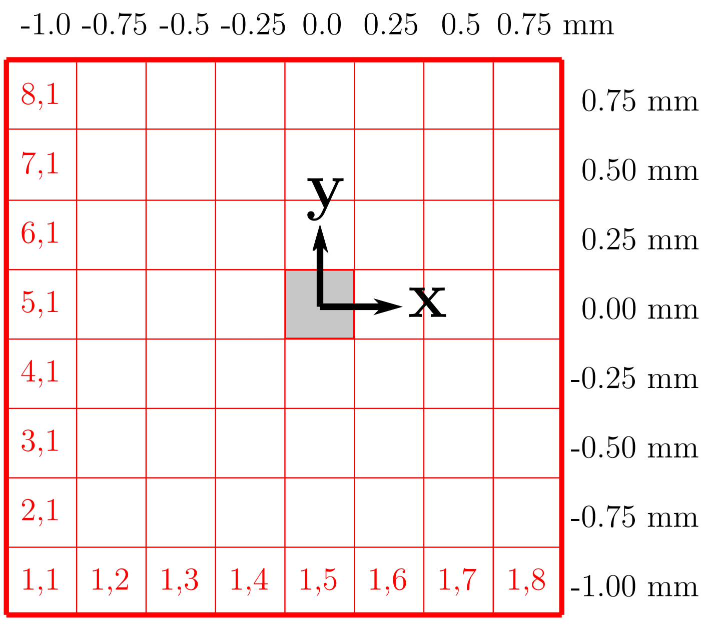

Figure 2.6.: An array of size 8×8 representing a quadratic area of 2 mm × 2 mm in the xy-plane. Each entry in this array represents the complex value of $U(x,y;z)$ at a certain, discrete $(x,y;z)$ coordinate point in position space. The grid is placed as centered as possible along the optical axis $x = y = 0$. The $(0,0)$ cell of the array is visualized in grey. The apparent lopsidedness is rooted in the FFT. The red numbers represent row and column indices of the array.

In figure 2.6 two facts immediately catch the eye: Firstly, the coordinate range on either axis stretches from −1 mm to +0.75 mm, and not from −1 mm to +1 mm as one would probably expect. This is due to the discrete and periodic nature of the FFT. Any component of a Fourier decomposition in either axis direction is a wave with a periodicity that is a multiple of the side length 2 mm. This means that (in our example) a period of a Fourier component wave starts at $-1\,\text{mm}$ and ends at $+1\,\text{mm}$. But the position $+1\,\text{mm}$ is at the same time already the beginning of a new period, and hence already part of the next "tile" in the periodic FFT. Therefore, this apparent lopsidedness in position space is rooted in the FFT.

Another effect of choosing an $N \times N$ grid with N being an even number is that the $(x,y) = (0,0)$ coordinate-point has an exact representation (depicted as the grey cell in figure 2.6).

It should be noted that the generic FFT of MatLab forces a grid where the coordinates on either side are lopsided towards the opposite axes directions, and would therefore (in our example) stretch from $-0.75\,\text{mm}$ to $+1\,\text{mm}$. However, this generic FFT of MatLab does not represent the "Physics Sign Convention" as explained in section 2.2.1, but rather the "Engineering Sign Convention". Therefore we have chosen to implement our own subroutines to match our sign convention.

It is further noteworthy that choosing a grid with an odd number of cells on either side (e.g. 9×9) does *not* make the lopsidedness in the coordinates go away. However, in such coordinates there would no longer be an explicit representation for the $U(0,0;z)$ value.

If L is the nominal length of the x and y axis (e.g. $2\,\text{mm}$ in figure 2.6), N the required number of discrete coordinate values along each axis (e.g. 8 in figure 2.6), and U(r, c) the position-space-array with r being the row index, and c being the column index, then the (x,y) coordinates can be calculated using the following formulas:

$$x(c) = -\frac{L}{2} + \frac{L}{N}(c-1) \quad \text{with} \quad c = 1 \ldots N \tag{2.43}$$

$$y(r) = -\frac{L}{2} + \frac{L}{N}(r-1) \quad \text{with} \quad r = 1 \ldots N \tag{2.44}$$

The following software function related to this section was implemented:

Creation of a vector containing FFT compatible axis coordinates (almost) symmetric around zero

- **Function:** `create_ax(gpu, N, L)`
- **Description:** Creates a vector with `N` entries containing the axis coordinates (almost) symmetric around zero so that the total length corresponds to target length `N` minus one step according to the requirements of FFT as explained above.
- **Documentation:** appendix C.1.

2.2.3. Spatial-frequency-space grid and k-space

As we deal with a 2D FFT on an xy-grid, two integer parameters n_x and n_y are required to identify a specific FFT basis function. Given the Fourier parameters n_x and n_y and the nominal side length L in meters, the values of such a basis function in each cell of the $N \times N$ position-space-array `U(r,c)` (figure 2.6) are calculated in our implementation as follows:

$$\mathrm{U(r,\, c)} = \frac{N}{L(N-1)}\; e^{i\frac{2\pi}{N}n_x(c-1)}\; e^{i\frac{2\pi}{N}n_y(r-1)} \tag{2.45}$$

In equation (2.45), `r` and `c` represent the row- and column-index of the corresponding MatLab array `U(r,c)` so that $1 \leq \mathrm{r} \leq N$ and $1 \leq \mathrm{c} \leq N$.

The normalization factor $\frac{N}{L(N-1)}$ ensures that integrating the squared absolute values over the whole surface area according to the axes scaling, calculated with `trapz(ax, trapz(ax, U .* conj(U), 2))`, always results in one (with `ax` being the coordinate vector generated by `create_ax(gpu, N, L)`). This means that the FFT Fourier basis functions are always normalized in position space in our implementation.

Given an $N \times N$ array, n_x and n_y can attain the following values if N is even:

$$-\frac{N}{2} < n_x \leq \frac{N}{2}, \quad n_x \in \mathbb{Z} \tag{2.46}$$

$$-\frac{N}{2} < n_y \leq \frac{N}{2}, \quad n_y \in \mathbb{Z} \tag{2.47}$$

For odd values of N this becomes

$$-\frac{N}{2} + \frac{1}{2} \leq n_x \leq \frac{N}{2} - \frac{1}{2}, \quad n_x \in \mathbb{Z} \tag{2.48}$$

$$-\frac{N}{2} + \frac{1}{2} \leq n_y \leq \frac{N}{2} - \frac{1}{2}, \quad n_y \in \mathbb{Z} \tag{2.49}$$

Fourier-transforming a position-space-array `U(r,c)` by using our MatLab function `fft2_phys_spatial(U, ax)` results in a spatial-frequency-space-array `A(r,c)` of equal size, where each array cell represents the complex Fourier-factor to be multiplied to the corresponding (n_x, n_y) basis function. Therefore, Fourier-transforming a normalized basis function according to equation (2.45) consequently results in an array where only one single cell contains the value 1, whereas all other cells contain 0 (plus-minus numerical inaccuracies).

The cell indices `r` and `c` of the spatial-frequency-space-array `A(r,c)` corresponding to n_x and n_y can be found by means of the following relations.

If N is even:

$$\mathrm{r} = \frac{N}{2} + 1 - n_y \tag{2.50}$$

$$\mathrm{c} = \frac{N}{2} + 1 - n_x \tag{2.51}$$

If N is odd:

$$\mathrm{r} = \frac{N}{2} + \frac{3}{2} - n_y \tag{2.52}$$

$$\mathrm{c} = \frac{N}{2} + \frac{3}{2} - n_x \tag{2.53}$$

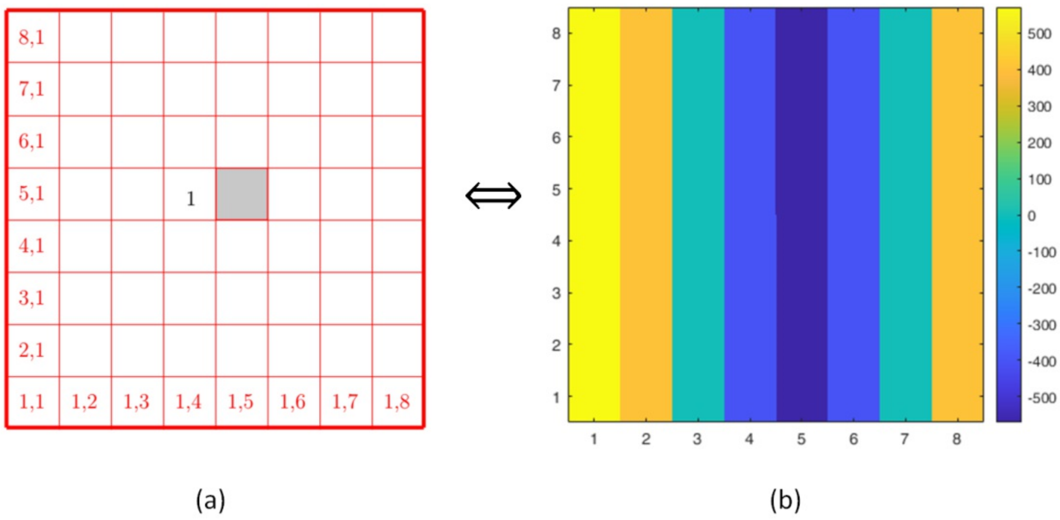

Figure 2.7.: (a) Spatial-frequency-space-array representing the the FFT basis function $n_x = 1, n_y = 0$. The grey field marks the $n_x = 0, n_y = 0$ position. (b) Position-space-array representing the FFT basis function $n_x = 1, n_y = 0$ on an area of $2\,\mathrm{mm} \times 2\,\mathrm{mm}$. The plot shows the real part of the complex function values. Because of the basis function being normalized over the area of just $4\,\mathrm{mm}^2$, the function values are relatively large.

The angular wavenumbers k_x and k_y are related to the transverse mode numbers n_x and n_y as follows (with L being the nominal side length of the position-space-array; e.g. $2\,\mathrm{mm}$ in figure 2.6):

$$k_x = \frac{2\pi n_x}{L} \tag{2.54}$$
$$k_y = \frac{2\pi n_y}{L} \tag{2.55}$$

The following software functions related to this section were implemented:

2D Fast Fourier Transform

- **Function:** `fft2_phys_spatial(X, ax)`
- **Description:** Performs a 2D Fast-Fourier-Transform (FFT) of the position-space-array `X` with xy-coordinates according to the axis coordinates provided in the `ax` vector. The output is the Fast Fourier Transform of `X` (according to physics sign convention).
- **Documentation:** appendix C.2.

Inverse 2D Fast Fourier Transform

- **Function:** `ifft2_phys_spatial(fourier_coeff, ax)`
- **Description:** Performs a 2D Inverse Fast-Fourier-Transform (IFFT) of the spatial-frequency Fourier coefficients array `fourier_coeff` and generates a position-space-array `Y` with xy-coordinates according to the axis coordinates provided in the `ax` vector.
- **Documentation:** appendix C.3.

Generation of normalized FFT basis functions

- **Function:** `fft2_basis_func(gpu, nx, ny, ax, f_space_out)`
- **Description:** Creates a normalized (`nx, ny`) Fast-Fourier basis function. The output is either in spatial-frequency space, or in position space.
- **Documentation:** appendix C.4.

Creation of a vector containing wave numbers matching the axis of the FFT-transformed position-space-array

- **Function:** `create_k_ax(gpu, ax)`
- **Description:** Creates a vector with the same number of entries as the `ax` vector, but containing wave numbers matching the axis of the FFT-transformed position-space-array (i.e., the spatial-frequency-space-array).
- **Documentation:** appendix C.5.

2.2.4. Tilt of the k-vector and mode-ordering

As an elementary wave propagates through the cavity, its $\vec{k}$-vector is generally tilted against the z-axis. This is manifested by non-zero values of the k_x and/or k_y wave numbers. The larger these numbers, the larger the tilt. Here we derive how much the total $\vec{k}$-vector of a propagating wave is tilted against the z-axis with given n_x and n_y mode numbers ($\vec{e}_z$ represents the unit vector in z-direction):

$$\begin{aligned}
||\vec{k}\cdot\vec{e}_z|| &= ||\vec{k}||\ \cancel{||\vec{e}_z||}\ \cos(\alpha) \\
k_z &= k\cos(\alpha) && \Big| k^2 = k_x^2 + k_y^2 + k_z^2 \implies k_z = \sqrt{k^2 - k_x^2 - k_y^2} \\
\sqrt{k^2 - k_x^2 - k_y^2} &= k\cos(\alpha) && \Big| k = \frac{2\pi}{\lambda} \qquad (2.56) \\
\sqrt{\left(\frac{2\pi}{\lambda}\right)^2 - k_x^2 - k_y^2} &= \frac{2\pi}{\lambda}\cos(\alpha) \\
\cos(\alpha) &= \frac{\lambda}{2\pi}\sqrt{\left(\frac{2\pi}{\lambda}\right)^2 - k_x^2 - k_y^2} \overset{(2.54)}{\implies} \overset{(2.55)}{\implies} \\
\cos(\alpha) &= \frac{\lambda}{2\pi}\sqrt{\left(\frac{2\pi}{\lambda}\right)^2 - \left(\frac{2\pi n_x}{L}\right)^2 - \left(\frac{2\pi n_y}{L}\right)^2} \\
\alpha &= \arccos\left(\frac{\lambda}{2\pi}\sqrt{\left(\frac{2\pi}{\lambda}\right)^2 - \left(\frac{2\pi n_x}{L}\right)^2 - \left(\frac{2\pi n_y}{L}\right)^2}\right) \qquad (2.57)
\end{aligned}$$

It should be noted that the expression under the root can become negative (evanescent modes). In this case there is no more propagation in z-direction, and we can assume $\alpha = \pi$.

As long as the expression under the root stays positive, equation (2.57) shows that the tilt of the $\vec{k}$-vector with respect to the z-axis increases strictly monotonically with $n_x^2 + n_y^2$. This fact can be used to define an order for all possible n_x, n_y pairs.

The following software functions related to this section were implemented:

Get angle between k-vector and z-axis

- **Function:** `k_vec_tilt(gpu, nx, ny, ax, lambda)`
- **Description:** Returns the angle with which the k-vector is inclined to the z-axis given n_x and n_y.
- **Documentation:** appendix C.6.

Get vector with all possible nx, ny values ordered by increasing angle of corresponding k-vectors with respect to the z-axis

- **Function:** `sorted_mode_numbers(gpu, N)`
- **Description:** Returns a vector with all combinations of n_x and n_y allowed for an array with side-length `N`. The entries are sorted according to the angle that the associated $\vec{k}$-vectors have with respect to the z-axis.
- **Documentation:** appendix C.7.

2.2.5. Fourier-coefficient vectors

In our calculations, we occasionally want to represent the behavior of an optical system by means of a transmission matrix $\underline{\underline{T}}$ (or a reflection matrix $\underline{\underline{R}}$, for that matter). The idea is that if we can represent the Fourier-coefficients of the system's input plane by a *vector* $\vec{a}_1$, we can easily perform a matrix-vector multiplication like $\vec{a}_2 = \underline{\underline{T}}\vec{a}_1$. The resulting vector $\vec{a}_2$ will then contain all Fourier-coefficients representing the light-field on the system's output-plane.

However, the Fourier-coefficients representing the modes on a transverse xy-plane are, by default, stored in a spatial-frequency-space *array*, and not in a *vector*. Therefore we have implemented two functions to convert such arrays into vectors and vice versa.

When converting an array into a Fourier-coefficient-vector, our implementation allows not only a spatial-frequency-space-array as input (which already contains Fourier coefficients), but optionally also a position-space-array may be provided. The required Fast-Fourier-Transformation will then be performed automatically before the matrix-vector transformation.

Equivalently, when back-converting a Fourier-coefficient-vector into an array, it can be chosen whether the output-array should be in spatial-frequency-space or position-space. In case of the latter, an Inverse-Fast-Fourier-Transformation will automatically be performed after the vector-matrix conversion.

The conversion and back-conversion functions both also expect an index vector, which determines the order of the entries in the Fourier-coefficient-vector. By the completeness of its entries it also determines whether or not all Fourier-coefficients are to be taken into account. By default, the vector created by `sorted_mode_numbers(gpu, N)` can be used as such an index vector. In that case, the Fourier-coefficient-vector will contain coefficients in order of modes representing increasingly tilted $\vec{k}$-vectors.

The following software functions related to this section were implemented:

Convert spatial-frequency-space-array or position-space-array into Fourier-coefficient-vector

- **Function:**
  `fft2_arr_to_vec(input_array, ax, mode_numbers, f_space_in)`
- **Description:** Converts `input_array`, which may be a position-space-array or a spatial-frequency-space-array, into a Fourier-coefficient-vector with the coefficients ordered according to the `mode_numbers` index vector. The boolean parameter `f_space_in` determines, whether `input_array` is a spatial-frequency-space-array, or a position-space-array.
- **Documentation:** appendix C.8.

Convert Fourier-coefficient-vector into spatial-frequency-space-array or position-space-array

- **Function:**
  `fft2_vec_to_arr(gpu, FFT_vec, ax, mode_numbers, f_space_out)`
- **Description:** Converts the Fourier-coefficient vector `FFT_vec` into a position-space-array or a spatial-frequency-space-array. The boolean parameter `f_space_out` determines, whether the output is a spatial-frequency-space-array, or a position-space-array.
- **Documentation:** appendix C.9.

2.3. Software implementation of light propagation

The following functions implement the Rayleigh-Sommerfeld and Fresnel transfer functions for propagation of the xy-light-field in z-direction through either empty space or a material.

With these functions, the propagation of the light field between the lenses as well as through an absorber can be simulated. The implementation of both propagation methods allows us to study the effect of the Fresnel approximation in comparison to the more accurate Rayleigh-Sommerfeld propagator.

Rayleigh-Sommerfeld transfer function propagator

- **Function:** `RSTF_prop(gpu, E_in, ax, z, lambda, n, f_space_in, f_space_out)`
- **Description:** Takes the xy-input field `E_in`, which can be either a position-space-array or a spatial-frequency-space-array, and propagates it in z-direction over the distance `z`, using the **Rayleigh Sommerfeld transfer function method** as described in section 2.1.2. For propagation in free space, the parameter `n` is to be set $n = 1$, otherwise `n` is the material-dependent (real or complex) refractive index. The returned output field is again optionally either a position-space-array or a spatial-frequency-space-array.
- **Documentation:** appendix C.10.

Fresnel transfer function propagator

- **Function:** `FRTF_prop(gpu, E_in, ax, z, lambda, n, f_space_in, f_space_out)`
- **Description:** Works like the function before, but uses the **Fresnel transfer function method** as described in section 2.1.3.
- **Documentation:** appendix C.11.

Universal propagator function

- **Function:** `prop(gpu, E_in, TF, ax, z, lambda, n, f_space_in, f_space_out)`
- **Description:** Depending on the value of the `TF` input parameter, **either** the Rayleigh Sommerfeld transfer function method, **or** the Fresnel transfer function method is used.
- **Documentation:** appendix C.12.

2.4. Software implementation of lenses

The following function implements the simulation of a thin spherical lens and a thin perfect, aspherical lens as described in sections 2.1.4 and 2.1.5.

Thin spherical lens and a thin perfect, aspherical lens

- **Function:** `lens(gpu, ax, lambda, pupil, NA, f, lens_type)`
- **Description:** Returns the phase-mask of a thin spherical lens or perfect aspherical lens in a position-space-array `lens_mask`. This array can be applied by simply multiplying it component-wisely to an input-field-array in position-space.
- **Documentation:** appendix C.13.

The theory behind the simulation of lenses is described in section 2.1.2.

2.5. A first, naive simulation approach for a one-way trip through a 4f cavity

The software functions presented so far (together with two helper functions for generating test-images and displaying input- and output fields, see appendices D.1 and D.2) allow a first, naive simulation attempt. The program on the next page simulates how a test-image (an image of the letter “T” in the top left corner) is transformed when going through the 4f-cavity in a single trip.

In lines 16-21 basic physical simulation parameters (focal length 75 mm, wavelength 800 nm, and lens-type “thin perfect, aspheric lens”) are defined. Line 24 defines that the (supposedly more accurate) Rayleigh Sommerfield transfer function is to be used for propagation. Line 25 just defines the test-image. In line 25 the side-length of the observed xy-plane is defined to be 2.5 mm. In line 27 the xy-grid-size is (arbitrarily) defined to be 800×800.

The simulation itself takes place in lines 33-49. The created test-image (line 37) is propagated one focal length (line 38), then the lens-mask is applied (line 41). Further propagation for two focal lengths takes place in line 43. Then the lens-mask is applied once more (line 45), and the final propagation to the end of the cavity (one more focal length) is simulated in line 47. The resulting image is displayed in line 50 (with the absolute value of input- and output-fields being displayed).

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Test: One-way trip through 4f-cavity with arbitrary grid size
% File: one_way_trip_test_simple.m
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all
close all
gpu = 1;               % use GPU
if gpu
    % initialize GPUs
    for ii = 1:gpuDeviceCount
        gpuDevice(ii);
    end
end

% Basic physical simulation parameters
lambda = 800e-9;       % Wavelength in meters
f = 75e-3;             % focal length of lenses 75 mm
pupil = false;         % no pupil
NA = 0.05;             % numerical aperture of pupil (if pupil == true)
lens_type = 2;         % aspherical, perfect lens

% technical simulation parameters
TF = 0;                % 0: Rayleigh Sommerfeld, 1: Fresnel
test_image = 4;        % Large off-center T
L = 2.5e-3;            % side length of input simulation grid
N = 800;               % number of steps alongside each ayis

ax = create_ax(gpu, N, L); % create xy-axes coordinates
% create lens phase mask
[lens_mask, lens_pupil] = lens(gpu, ax, lambda, pupil, NA, f, lens_type);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% SIMULATE SINGLE TRIP IN 4F CAVITY
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Create input image
E_in = create_test_image(gpu, ax, test_image, false);
% propagate to first lens
E = prop(gpu, E_in, TF, ax, f, lambda, 1, false, false);
% apply lens mask
E = E .* lens_mask;
% propagate 2f distance
E = prop(gpu, E, TF, ax, 2*f, lambda, 1, false, false);
% apply lens mask
E = E .* lens_mask;
% proagate f
E_out = prop(gpu, E, TF, ax, f, lambda, 1, false, false);

% Display result
plot_fields(gpu, 1, ax, E_in, 'Input Field', 3, E_out, 'Output Field', 3);
```

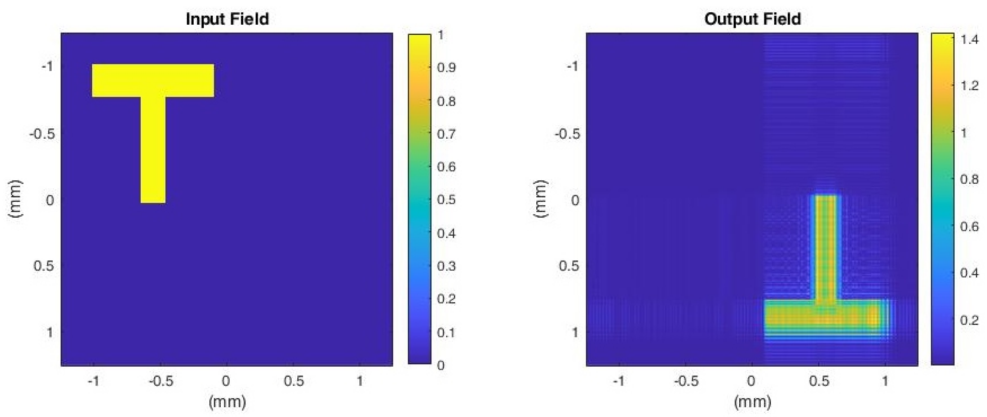

Figure 2.8.: A first, naive simulation attempt: A test-image is propagated one-way through a 4f-cavity of $4 \times 75\,\mathrm{mm}$ total length. The result should be an undistorted upside-down version of the input-image. However, the output image is subject to significant simulation artifacts.

Figure 2.8 shows that the simulated output image, which ideally should be just an upside-down version of the input image, is distorted by a significant amount of simulation artifacts. In the next chapter we will explore the reasons for these artifacts, and how to avoid them.

2.6. FFT vs. analytic FT: optimal grid and sampling parameters

In this chapter we demonstrate — following the line of argumentation in [14, p. 191] — how to determine the optimal grid spacing and sampling parameters.

The Fresnel Transfer Function approximation in equation (2.21) contains a phase function whose absolute value increases with the square of the wavenumber variables k_x and k_y. Such a function is referred to as “chirp” function. The sampling of chirp functions can become problematic because of the increasing slope of the phase of the frequency (cf. [15]).

Only the phase of the chirp term in equation (2.21) is a function of frequency. Extracting this phase gives:

$$\phi_c(k_x, k_y) = -\frac{\lambda z}{4\pi}\left(k_x^2 + k_y^2\right) \tag{2.58}$$

It is sufficient to examine only one transverse direction, as the sampling constraints for the two orthogonal variables can be evaluated separately. Let us choose k_x arbitrarily. To be able to unambiguously reconstruct the phase $\phi_c(k_x)$ from discrete sample values, each two adjacent sample values $\phi_c(k_{x1})$ and $\phi_c(k_{x1} + \Delta k_x)$ must fulfill the condition

$$|\phi_c(k_{x1} + \Delta k_x) - \phi_c(k_{x1})| \leq \pi \tag{2.59}$$

For small values of Δk_x this can be expressed as

$$\begin{aligned}
&\Delta k_x \left|\frac{\partial \phi_c}{\partial k_x}\right|_{\max} \leq \pi \overset{(2.58)}{\Longrightarrow} \\
&\Delta k_x \left|\frac{\partial}{\partial k_x}\left(-\frac{\lambda z}{4\pi}\left(k_x^2 + k_y^2\right)\right)\right|_{\max} \leq \pi \\
&\Delta k_x \left|-\frac{\lambda z}{2\pi} k_x\right|_{\max} \leq \pi \\
&\Delta k_x \frac{\lambda z}{2\pi} \left|k_x\right|_{\max} \leq \pi \\
&\Delta k_x \lambda z \left|k_x\right|_{\max} \leq 2\pi^2 \\
&\Delta k_x \leq \frac{2\pi^2}{\lambda z \left|k_x\right|_{\max}}
\end{aligned} \tag{2.60}$$

This is the sampling condition in k-space for the Fresnel transfer function, which, in a first approximation, also applies to the Rayleigh Sommerfeld transfer function. It is equally valid for Δk_y and $\left|k_y\right|_{\max}$. To translate this condition into position-space, one has to remember equation (2.54), from which we can deduce

$$\begin{aligned}
&\Delta k_x = \frac{2\pi}{L} \overset{(2.60)}{\Longrightarrow} \\
&\frac{\cancel{2\pi}}{L} \leq \frac{\cancel{2}\pi^{\cancel{2}}}{\lambda z \left|k_x\right|_{\max}} \overset{(2.54)}{\Longrightarrow} \\
&\frac{1}{L} \leq \frac{\cancel{\pi}}{\lambda z} \frac{L}{2\cancel{\pi} n_{\max}} \\
&\frac{1}{L} \leq \frac{L}{2\lambda z n_{\max}}
\end{aligned} \tag{2.61}$$

In equation (2.61) L represents the nominal side-length of the xy-grid, and $n_{\max}$ stands for the largest possible mode number in x or y direction, so that $\forall n_x, n_y : n_x \leq n_{\max}, n_y \leq n_{\max}$.

Let N be the number of discrete coordinate-points in either x and y-direction. Then we can write (see equations (2.46) and (2.48)):

$$\begin{aligned}
n_{\text{max}} &= \frac{N}{2} \overset{(2.61)}{\Longrightarrow} \\
\frac{1}{L} &\leq \frac{L}{\cancel{2}\lambda z}\frac{\cancel{2}}{N} && \left|\frac{L}{N} = \Delta x\right. \\
\frac{1}{L} &\leq \frac{\Delta x}{\lambda z} \\
\Delta x &\geq \frac{\lambda z}{L}
\end{aligned} \tag{2.62}$$

This is the sampling condition in position-space, where L represents the nominal side length of the grid in position-space, and Z is the propagation distance in z-direction. This sampling condition is equally valid for Δy.

Equation (2.62) can also be expressed in terms of N, which is the number of discrete coordinate points along the x and y-axis:

$$\begin{aligned}
\Delta x &= \frac{L}{N} \overset{(2.62)}{\Longrightarrow} \\
\frac{L}{N} &\geq \frac{\lambda z}{L} \\
\frac{1}{N} &\geq \frac{\lambda z}{L^2} \\
N &\leq \frac{L^2}{\lambda z}
\end{aligned} \tag{2.63}$$

2.6.1. Oversampled transfer function

As the transfer-function acts in spatial-frequency-space (or, in our formulation, more specifically in k-space), it makes most sense to look at equation (2.60) for distinguishing *oversampling* and *undersampling*. It is common to say that the transfer-function is *oversampled* if

$$\Delta k_x < \frac{2\pi^2}{\lambda z \left|k_x\right|_{\text{max}}} \tag{2.64}$$

Consequently (and a bit counter-intuitively) one also speaks of *oversampling* when the corresponding condition

$$\Delta x > \frac{\lambda z}{L} \tag{2.65}$$

is fulfilled. This condition is met for relatively "short" propagation distances z. The originally derived condition in equation (2.62) is certainly fulfilled, and so one would expect no problems with oversampling.

However, a look at figure 2.9 reveals that there is a problem with the *inverse* FFT that is required after the application of the transfer-function. Rather than a constant magnitude value, a window-like function with oscillations at the fringes appears. Also the phase follows the analytic phase only within this window, and then starts to deviate.

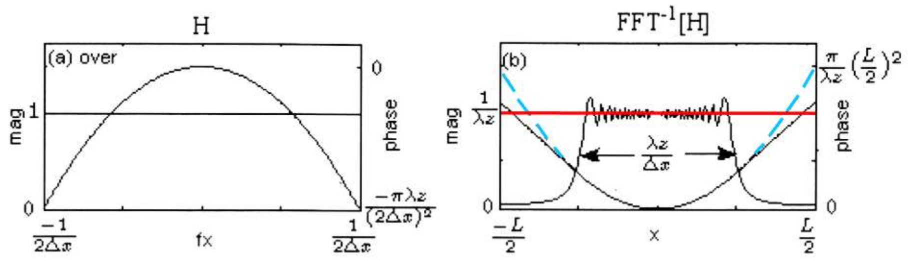

Figure 2.9.: Left: Magnitude (horizontal line) and unwrapped phase (curved line) of the oversampled transfer-function match the analytic result. Right: The magnitude and unwrapped phase of the inverse FFT of the oversampled transfer-function deviate from the analytic results. The magnitude shows a "window-like" quality with oscillations at the fringes (instead of the expected constant analytic result, displayed in red). The phase curve also deviates from the analytic result (blue dashed curve). From: [14, p. 193], with color edits.

Therefore, oversampling leads to the so-called "support-area", which is the center-window in the observation plane where the simulation agrees with the analytic result. This support area becomes more limited, the more oversampled the transfer-function is. The side-length of the support-area in the observation plane is (cf. [14, p. 198], [11, p. 15ff])

$$D = \frac{\lambda z}{\Delta x} \tag{2.66}$$

2.6.2. Critically sampled transfer function

The transfer function is *critically sampled* when

$$\Delta x = \frac{\lambda z}{L} \tag{2.67}$$

It is a property of the sampled chirp-function that when the critical sampling condition of equation (2.68) is fulfilled, both the sampled transfer-function and the inverse FFT of the transfer-function match exactly their analytic counterparts (see figure 2.10).

It should be noted that the chirp-function, for which this statement is exactly true, only describes the Fresnel approximation exactly. However, it is also the leading term in the series expansion of the phase term in the Rayleigh-Sommerfeld transfer function. This therefore means that, when the critical sampling condition is fulfilled, also the Rayleigh-Sommerfeld transfer function works best and creates the least simulation artifacts.

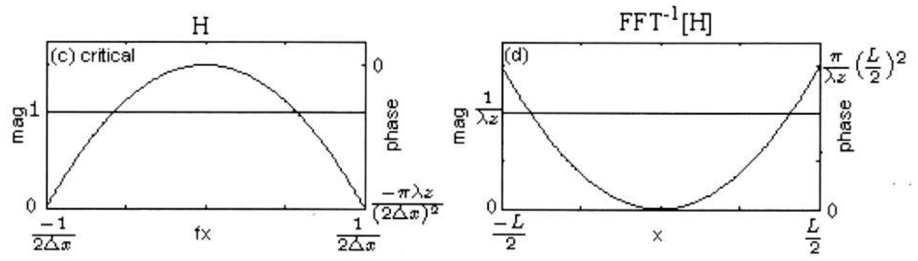

Figure 2.10.: Left: Magnitude (horizontal line) and unwrapped phase (curved line) of the critically sampled transfer-function match the analytic result. Right: The same is true for the inverse FFT of the critically sampled transfer-function. Note that this is only exactly true in the paraxial approximation of the Fresnel transfer function. For the exact Rayleigh-Sommerfeld transfer function deviations are still minimized when the critical sampling condition is met. From: [14, p. 193].

2.6.3. Undersampled transfer function

The transfer function is *undersampled* when

$$\Delta x < \frac{\lambda z}{L} \tag{2.68}$$

This condition is met for relatively "long" propagation distances z. The originally derived condition in equation (2.62) is no longer fulfilled, and therefore one rightly expects problems to appear not only in the inverse FFT, but already in the sampled transfer-function itself.

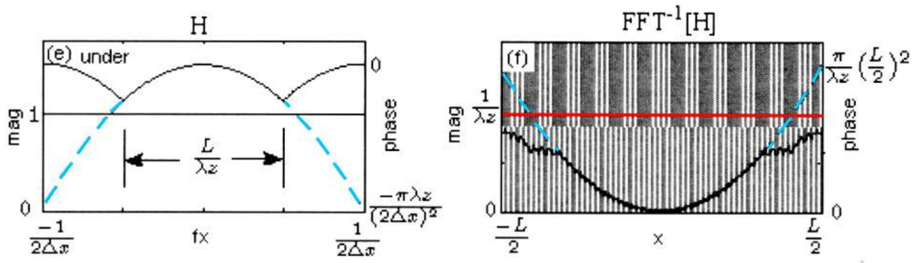

Figure 2.11.: Left: In the undersampled transfer-function, the phase curve significantly deviates from the analytic result (blue dashed curve) outside a certain bandwidth-window. This can lead to significant artifacts. Right: The magnitude oscillates every other sample, which causes the grey "blur" in the diagram. The analytic solution for the magnitude is shown as constant, red line. Also, the phase deviates on the fringes compared to the analytic results (blue, dashed curve). Source: [14, p. 193], with color edits.

As figure 2.11 shows, the phase curve significantly deviates from the analytic result outside a certain bandwidth-window. This can lead to strong artifacts which manifest themselves in the inverse FFT diagram on the right side of figure 2.11. There, the magnitude of the transfer function oscillates every other sample, which causes the grey "blur" in the diagram. Also, the phase deviates on the fringes compared to the analytic results.

The source bandwidth limitation in an undersampling situation as shown on the left side of figure 2.11 can be expressed as (cf. [15])

$$B_{\text{source}} \leq \frac{L}{2\lambda z} \tag{2.69}$$

2.6.4. Conclusion

The Fresnel transfer function, and also the Rayleigh-Sommerfeld transfer function (in first order) incorporate the Fresnel "chirp" term from equation (2.21), which is adequately sampled (oversampled) when the condition $\Delta x \geq \frac{\lambda z}{L}$ is fulfilled. However, it turns out that there is an equivalent chirp term in position space (namely $e^{i\frac{k}{2z}\left(x^2+y^2\right)}$), which becomes relevant with the inverse FFT. This chirp-term is adequately sampled (oversampled) when $\Delta x \leq \frac{\lambda z}{L}$ is fulfilled. Therefore, one should always attempt to fulfill the *critical sampling* condition

$$\Delta x_{opt} = \frac{\lambda z}{L} \tag{2.70}$$

Equivalently, the inequality (2.63) becomes an equation providing the optimal number of discrete coordinate points along the x- and y-axis.

$$N_{opt} = \frac{L^2}{\lambda z} \tag{2.71}$$

The effect of *oversampling* (short distance, $\Delta x > \frac{\lambda z}{L}$), is a "support-area", which is a square window in the center of the total simulated plane with side-length

$$D = \frac{\lambda z}{\Delta x} \tag{2.72}$$

Beyond this limit, both the Fresnel transfer function, as well as the Rayleigh-Sommerfeld transfer function primarily attenuate the field. In case of *critical sampling*, the guard-area D is exactly the nominal side-length L of the observation-grid.

Undersampling (long distance, $\Delta x < \frac{\lambda z}{L}$) leads to source bandwidth limitations with the condition $B_{\text{source}} \leq \frac{L}{2\lambda z}$, and quickly results in strong artifacts. Therefore, undersampling is to be avoided when using the Fresnel transfer function or the Rayleigh-Sommerfeld transfer function approach.

2.7. Guard-area and maximum mode number

It is good practice to surround the area of interest on the source plane by a so-called "guard-area", i.e., to embed the source-grid with side-length L_1 in a larger grid with side length L_2. Figure 2.12 shows this setting.

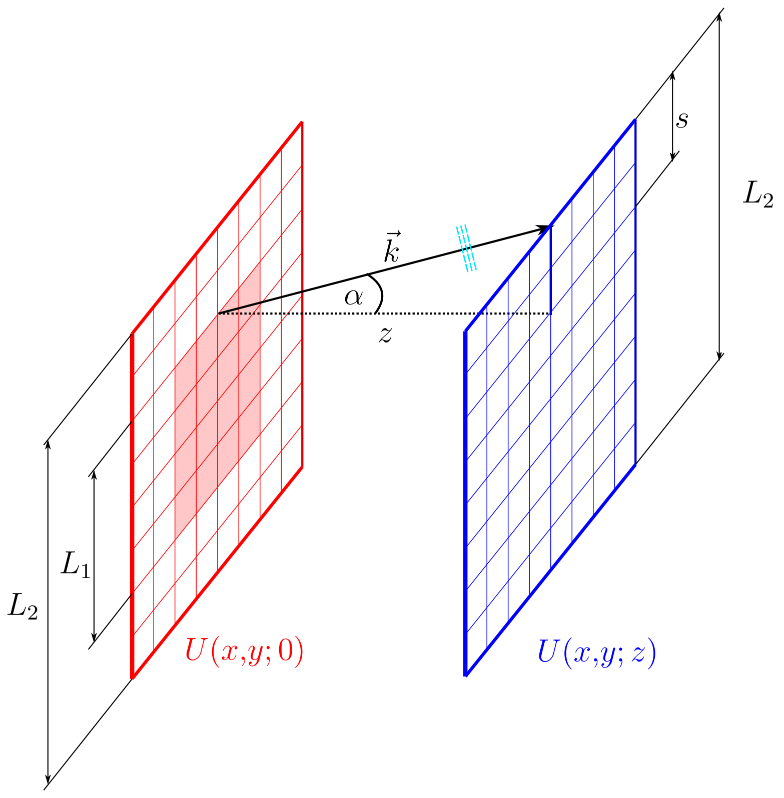

Figure 2.12.: The source field has the side-length L_1 (depicted with red shading) and is guarded by a larger grid with side-length L_2. This larger grid represents the whole source plane $U(x,y;0)$. A light wave corresponding to a certain n_y-mode (depicted in light blue) is represented by a $\vec{k}$-vector which is inclined by the angle α with respect to the z-axis. By simple triangular considerations it becomes clear that there is a maximum angle α, corresponding to a maximum mode number n_y, so that the propagated wave can still be registered at the observation plane $U(x,y;z)$.

There are two reasons to employ a guard-area: Firstly, as explained in the previous sections, it can happen (at least in case of oversampling) that not the whole area, but only a smaller support-area of the source-grid is actually usable. In addition, the considerations in the previous sections were only exact in the context of the paraxial Fresnel approximation. This means, that even when we use critical sampling, border-areas on the xy-plane are more prone to become subject to distortion by numerical artifacts in the context of the Rayleigh-Sommerfeld propagation.

But there is yet another important reason to use a guard-area in the source-plane. Figure 2.12 shows an elementary light-wave corresponding to a certain n_y-mode (depicted in light blue). This plane wave is represented by a $\vec{k}$-vector which is inclined by the angle α with respect to the z-axis. By simple triangular considerations it becomes clear that the observation plane must have a larger area than the source-image, so that also such tilted k-vectors are still fully observable on the observation plane $U(x{,}y;z)$.

The maximum allowed angle α_{max} can be calculated as follows:

$$\begin{aligned}
\tan(\alpha_{max}) &= \frac{s}{z} \\
\alpha_{max} &= \arctan\left(\frac{s}{z}\right)
\end{aligned} \tag{2.73}$$

For calculating the maximum mode-number $n_{max} = n_x^{max} = n_y^{max}$, we consider the $k_x = 0$ axis:

$$\begin{aligned}
k_x &= 0 \overset{(2.56)}{\Longrightarrow} \\
\sqrt{k^2 - k_y^2} &= k\cos(\alpha) \\
k^2 - k_y^2 &= k^2\cos^2(\alpha) \\
k_y^2 &= k^2 - k^2\cos^2(\alpha) \\
k_y^2 &= k^2\left(1 - \cos^2(\alpha)\right) \\
k_y^2 &= k^2\sin^2(\alpha) \\
k_y &= k\sin(\alpha) && \Big| k = \frac{2\pi}{\lambda} \\
k_y &= \frac{2\pi}{\lambda}\sin(\alpha) \overset{(2.55)}{\Longrightarrow} \\
\frac{\cancel{2\pi} n_y}{L_1} &= \frac{\cancel{2\pi}}{\lambda}\sin(\alpha) \\
n_y &= \frac{L_1}{\lambda}\sin(\alpha) && \Big| n_{max} = n_x^{max} = n_y^{max} \\
n_{max} &= \frac{L_1}{\lambda}\sin(\alpha_{max})
\end{aligned} \tag{2.74}$$

By inserting equation (2.73) into equation (2.74), we get:

$$
\begin{aligned}
n_{max} &= \frac{L_1}{\lambda} \sin\left(\arctan\left(\frac{s}{z}\right)\right) \\
n_{max} &= \frac{L_1}{\lambda} \frac{\frac{s}{z}}{\sqrt{1+\frac{s^2}{z^2}}} \\
n_{max} &= \frac{L_1}{\lambda} \frac{s}{\cancel{z}\sqrt{\frac{z^2+s^2}{\cancel{z^2}}}} \\
n_{max} &= \frac{L_1}{\lambda} \frac{s}{\sqrt{z^2+s^2}} \quad \text{with} \quad s = \frac{L_2 - L_1}{2} \qquad (2.75)
\end{aligned}
$$

2.8. A refined simulation approach for a one-way trip through a 4f cavity

Based on the insights of the previous sections it is now clear why the first simulation attempt in section 2.5 had such a dismal performance: Besides the fact that there was no guarding area, the resolution of the simulation grid was chosen arbitrarily, which happened to result in strong *undersampling* which notoriously produces heavy artifacts when using the Fresnel transfer function or Rayleigh-Sommerfeld transfer function method (see section 2.6.3).

In this section we present a refined software approach, that takes the insights of the previous sections into account.

2.8.1. Additional Subroutines

The following additional subroutines were implemented:

Calculation of optimal grid parameters

- **Function:** `opt_grid_params(L_des, factor, z, z_max, even, lambda)`
- **Description:** The main input parameters of this function are the desired side-length of the "guarded" input-grid (`L_des`, corresponding to L_1 in figure 2.12), a `factor` defining how much larger the "guarding" grid should be (side-length L_2 in figure 2.12), and the propagation distance `z`. As these input values generally do *not* result in an integer value for the optimal number of discrete coordinate points along the x- and y-axis according to equation (2.71), the best-matching integer value for N_1 is calculated, and returned together with the corresponding side-length L_1 (close to `L_des`). Equivalently, it is ensured that the larger side-length L_2 of the "guarding"

area exactly matches an integer value N_2, and these two values are also returned. The boolean parameter `even` allows to force N_1 and N_2 to be even, or odd. Further, a parameter `z_max` has to be passed to the function. It indicates the longest propagation distance in the simulation (in case that the propagation function is applied multiple times in a row). This input allows the function to additionally return the maximum tilt angle α_{max} and the maximum transverse mode number n_{max} according to equations (2.73) and (2.74).

- **Documentation:** appendix C.15.

Embed smaller position-space-grid in larger "guarding" grid

- **Function:** `embed_image(gpu, E_in_small, ax_large)`
- **Description:** Embeds smaller position-space-grid in larger "guarding" grid as shown in figure 2.12.
- **Documentation:** appendix C.16.

Extract smaller position-space-grid from larger "guarding" grid

- **Function:** `extract_center_image(E_in_large, ax_small)`
- **Description:** Extracts the smaller (position-space) grid from the center of a larger "guarding" grid (see figure 2.12).
- **Documentation:** appendix C.17.

2.8.2. Main program

In this section we present a refined version of the program from section 2.5, now using the additional subroutines from section 2.8.1. Again, the program simulates how a test-image (an image of the letter "T" in the top left corner) is transformed when being propagated through the 4f-cavity in a single left-to-right-trip.

In lines 18-23 of the source-code on the following page, basic physical simulation parameters (focal length 75 mm, wavelength 800 nm, and lens-type "thin perfect, aspheric lens") are defined. In line 26, the Rayleigh-Sommerfeld propagation method is chosen as a propagator. Line 27 just defines the test-image. In line 28 the desired side-length of the xy-plane is defined to be 2.5 mm. Line 29 defines that the larger "guarding" grid should have the double side-length of the smaller "guarded" grid.

```
 1 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 2 % Simple test: one-way trip through 4f cavity
 3 % with optimal grid size and guarding area
 4 % File: one_way_trip_test_opt.m
 5 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 6 
 7 clear all
 8 close all
 9 
10 gpu = 1;              % use GPU
11 if gpu
12     % initialize GPUs
13     for ii = 1:gpuDeviceCount
14         gpuDevice(ii);
15     end
16 end
17 
18 % Basic physical simulation parameters
19 lambda = 800e-9;      % Wavelength in meters
20 f = 75e-3;            % focal length of lenses 75 mm
21 pupil = false;        % no pupil
22 NA = 0.05;            % numerical aperture of pupil (if pupil == true)
23 lens_type = 2;        % aspherical, perfect lens
24 
25 % technical simulation parameters
26 TF = 0;               % 0: Rayleigh Sommerfeld, 1: Fresnel
27 test_image = 4;       % Large off-center T
28 L_des = 2.5e-3;       % desired side length of input simulation grid
29 factor = 2;           % factor by which embedding grid should be larger
30 
31 % create optimal grid
32 [N1, L1, N2, L2, alpha_max, n_max] = ...
33 opt_grid_params(L_des, factor, f, f, true, lambda);
34 % display grid parameters
35 disp(['N1=', num2str(N1), ' N2=', num2str(N2), ' n_max=', num2str(n_max)])
36 
37 ax1 = create_ax(gpu, N1, L1); % create xy-axes coordinates "small" grid
38 ax2 = create_ax(gpu, N2, L2); % create xy-axes coordinates "large" grid
39 
40 % create lens phase mask
41 [lens_mask, lens_pupil] = lens(gpu, ax2, lambda, pupil, NA, f, lens_type);
42 
43 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
44 % SIMULATE SINGLE TRIP IN 4F CAVITY
45 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
46 % Create input image
47 E_in = create_test_image(gpu, ax1, test_image, false);
48 % embed in "large" grid
49 E = embed_image(gpu, E_in, ax2);
50 % propagate to first lens
51 E = prop(gpu, E, TF, ax2, f, lambda, 1, false, false);
52 % apply lens mask
53 E = E .* lens_mask;
54 % propagate 2f distance
55 E = prop(gpu, E, TF, ax2, f, lambda, 1, false, true);
56 E = prop(gpu, E, TF, ax2, f, lambda, 1, true, false);
57 % apply lens mask
58 E = E .* lens_mask;
59 % proagate f
60 E = prop(gpu, E, TF, ax2, f, lambda, 1, false, false);
61 % extract smaller center image
62 E_out = extract_center_image(E, ax1);
63 % Display result
64 plot_fields(gpu, 1, ax1, E_in, 'Input Field', 3, E_out, 'Output Field', 3);
```

The first lines of code that are really new compared to the initial implementation on page 32 are the lines 31-35, where the optimal sizes of the small and large grids are computed and printed out. Please note that the `z_max`-parameter of the `opt_grid_params`-function is set to one focal length `f`, even if there is a $2f$ propagation in the middle of the cavity. This is correct, because the $2f$ propagation takes place between the two converging lenses where there is (simply speaking) no danger of "light-rays leaving the observation plane".

The simulation itself takes place in lines 43-62. The created test-image (line 46) is embedded in the larger grid (line 49), and then the larger grid is propagated one focal length (line 51). After that, the lens-mask is applied (line 53).

Further propagation for two focal lengths takes place in lines 55-56. Please note that this is broken down into two propagation steps with one focal length each, so that no undersampling happens. It is also noteworthy that the first propagation function in line 55 takes a position-space array as input, but returns a spatial-frequency-space-array (this is defined by the last two boolean parameters). The second propagation step in line 56 then continues with the spatial-frequency-space-array as input, and returns a position-space array. This choice makes sense, as it saves an unnecessary IFFT conversion at the output-end of line 55, and an unnecessary FFT-conversion at the input-end of line 56.

Then the lens-mask is applied once more (line 58), and the final propagation to the end of the cavity (one more focal length) is simulated in line 60. After that, the resulting image is extracted from the center of the embedding grid in line 62, and subsequently is displayed in line 64 (with the absolute value of input- and output-fields being displayed).

2.8.3. Simulation result

The program displays the following calculated simulation parameters:

```
N1=210 N2=418 n_max=52
```

That means that the optimal “small” grid size $N_1 \times N_1$ for the given geometry was computed to be 210×210. This grid was then embedded in a $N_2 \times N_2 = 418 \times 418$ “guarding” grid. The maximum traverse mode that can be safely simulated is $(n_x, n_y) = (52{,}52)$.

The visual output is displayed in figure 2.13. As can be seen, the simulation works much better, and the artifacts from figure 2.8 have vanished.

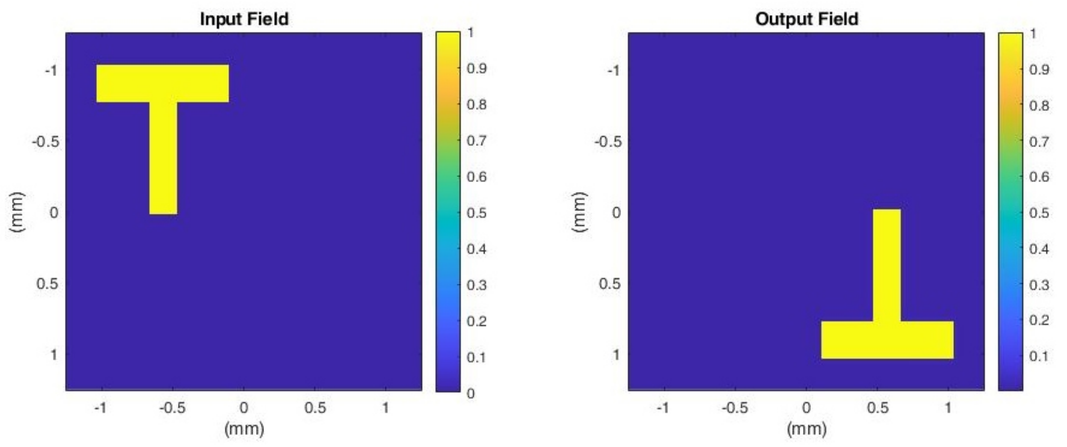

Figure 2.13.: A refined simulation of a test-image propagated one-way through a 4f-cavity of $4 \times 75\,\mathrm{mm}$ total length. This time, the input-image is embedded in a guarding area, and the grid resolution has been calculated so that there is no oversampling or undersampling. The artifacts from figure 2.8 have now vanished.

2.9. Simulating a round-trip through a 4f cavity

To simulate a round-trip through a 4f-cavity, we can simply take the one-way-trip-simulation as demonstrated in the previous section, apply a π phase-shift to the output (assuming an ideal mirror on the back-plane), and then feed the result into the very same one-way-trip-simulation once more.

2.9.1. Round-trip subroutine

The following subroutine was implemented:

Simulation of round-trip through 4f-cavity without attenuation

- **Function:** `round_trip_no_atten(gpu, E_in, ax_small, ax_large, TF, lens_mask, f, lambda, f_space_in, f_space_out)`
- **Description:** Takes the input field and sends it on a round-trip through a 4f-cavity with given focal length and perfect rear-mirror, assuming no attenuation. Coordinate-axes-vectors for the smaller input-grid and the larger guarding-grid must be provided together with a phase-mask for the lenses. Input and output can both be optionally either in position-space or in spatial-frequency-space. The propagation method can be chosen by means of the `TF` parameter.
- **Documentation:** appendix C.18.

2.9.2. Main program

Below is the source-code of the main program simulating a round-trip through a 4f-cavity. The structure is equivalent to the program in section 2.8.2, except that we now use the new round-trip-routine from section 2.9.1.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Simple test: round-trip through 4f cavity
% with optimal grid size and guarding area
% File: round_trip_test_opt.m
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all
close all

gpu = 1;                % use GPU
if gpu
    % initialize GPUs
    for ii = 1:gpuDeviceCount
        gpuDevice(ii);
    end
end

% Basic physical simulation parameters
lambda = 800e-9;        % Wavelength in meters
f = 75e-3;              % focal length of lenses 75 mm
pupil = false;          % no pupil
NA = 0.05;              % numerical aperture of pupil (if pupil == true)
lens_type = 1;          %0: spherical lens, 1: aspherical, perfect lens

% technical simulation parameters
TF = 0;                 % 0: Rayleigh Sommerfeld, 1: Fresnel
test_image = 4;         % Large off-center T
L_des = 2.5e-3;         % desired side length of input simulation grid
factor = 2;             % factor by which embedding grid should be larger

% create optimal grid
[N1, L1, N2, L2, alpha_max, n_max] = ...
opt_grid_params(L_des, factor, f, f, true, lambda);
% display grid parameters
disp(['N1=', num2str(N1), ' N2=', num2str(N2), ' n_max=', num2str(n_max)])

ax1 = create_ax(gpu, N1, L1); % create xy-axes coordinates "small" grid
ax2 = create_ax(gpu, N2, L2); % create xy-axes coordinates "large" grid

% create lens phase mask
[lens_mask, lens_pupil] = lens(gpu, ax2, lambda, pupil, NA, f, lens_type);
% Create input image
E_in = create_test_image(gpu, ax1, test_image, false);
% simulate round-trip
E_out = round_trip_no_atten( ...
    gpu, E_in, ax1, ax2, TF, lens_mask, f, lambda, false, false);
% Display result
plot_fields(gpu, 1, ax1, E_in, 'Input Field', 3, E_out, 'Output Field', 3);
```

2.9.3. Simulation result

The program displays the following calculated simulation parameters:

```
N1=210 N2=418 n_max=52
```

The visual output is displayed in figure 2.14. As expected with a 4f-cavity, the output image equals the input-image.

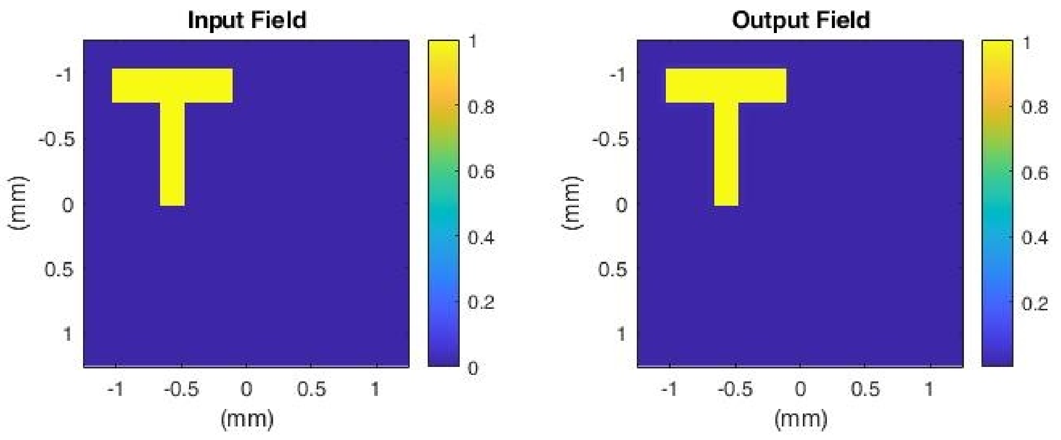

Figure 2.14.: A simulation of a test-image propagated on a single round-trip through a 4f-cavity of $4 \times 75\,\mathrm{mm}$ total length. The absolute value of the complex-valued fields is displayed.

At first glance, the output field in figure 2.14 does not seem to show any artifacts. However, when zooming in to the point where the horizontal and the vertical bar meet, residual artifacts become visible (see figure 2.15).

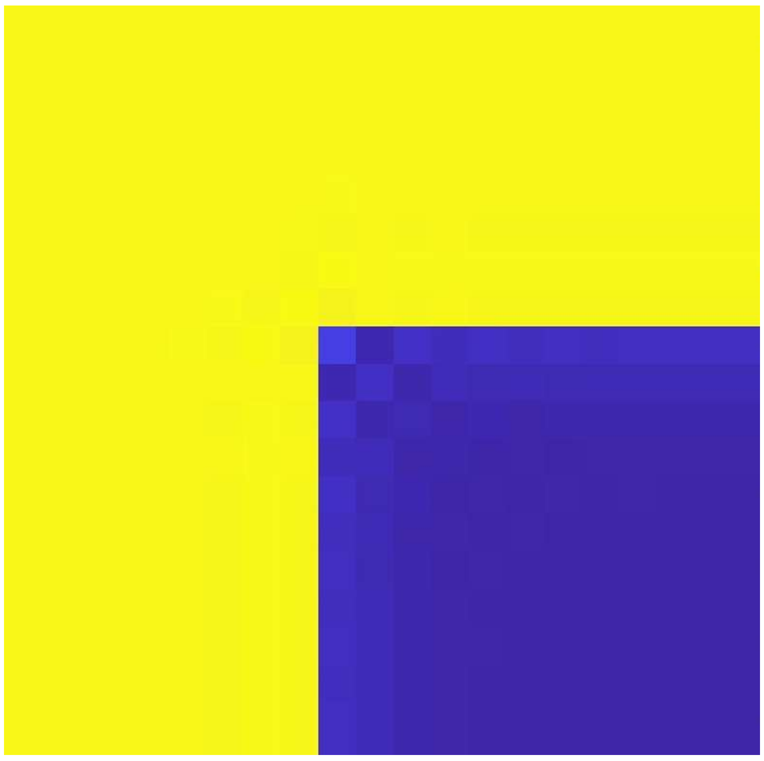

Figure 2.15.: Residual artifacts are visible when using the Rayleigh-Sommerfeld propagator, even when the optimal grid-size is used.

By changing line 26 to `TF = 1;`, the software is modified so that the (physically less accurate) Fresnel transfer function will be used instead of the Rayleigh-Sommerfeld propagator. This eliminates the artifacts (see figure 2.16).

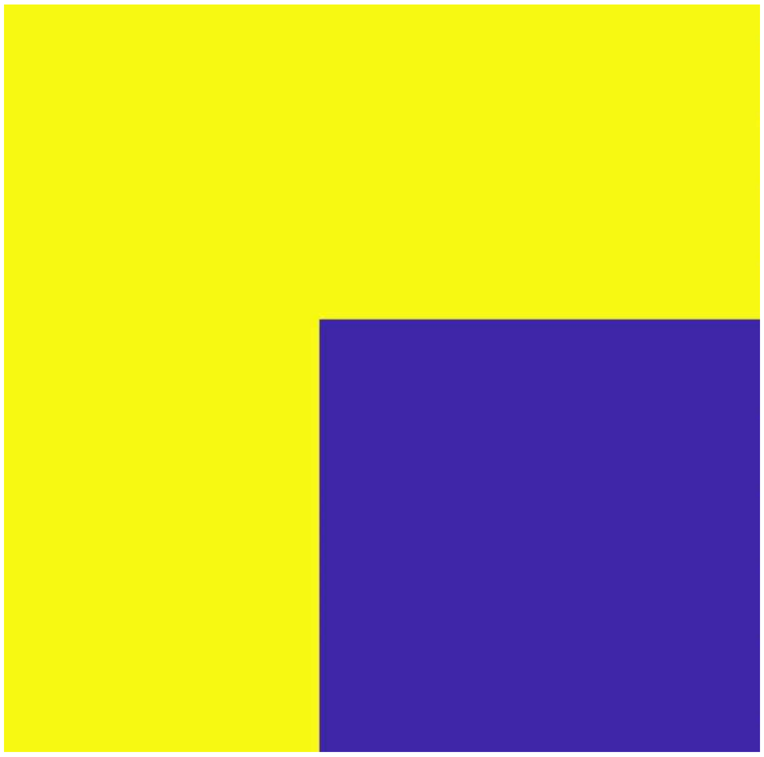

Figure 2.16.: The residual artifacts vanish when the Fresnel propagator with optimal grid-size is used.

Indeed, this behavior does not come as a surprise. The sampling condition derived in section 2.6 is only exact for the Fresnel transfer function, but only an approximation for the Rayleigh-Sommerfeld method. Still, it seems better to generally use the physically more accurate Rayleigh-Sommerfeld method in our simulations, but it must be kept in mind that residual artifacts can creep in and might sometimes reduce the accuracy of a simulation.

As a last check for the method, we make the following two code modifications in line 27 and 48.

```
test_image = 3;      % Large centered T with a phase gradient
```

```
plot_fields(gpu, 1, ax1, E_in, 'Input Field', 1, E_out, 'Output Field', 1);
```

With these code modifications, a large centered “T” with a phase gradient is used as input image, and the real part of the complex input- and output-fields is displayed instead of the absolute value.

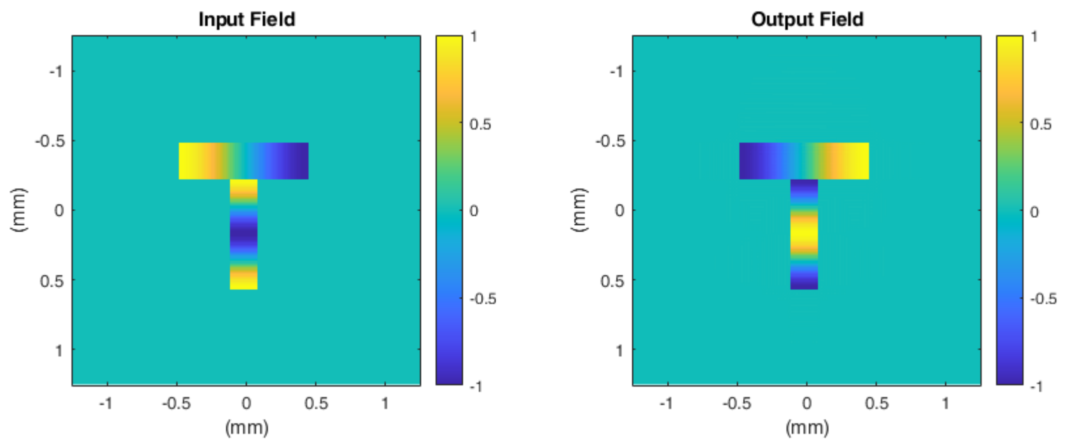

Figure 2.17.: Simulation of a test-image with a phase gradient propagated on a round-trip through a 4f-cavity of $4 \times 75\,\mathrm{mm}$ total length. The real part of the complex-valued fields is displayed.

Again, the output image equals the input image, except for a visible phase-shift of π. This meets the expectations, as the total distance of a single round-trip in this simulation is $2L = 8f = 8 \times 75\,\mathrm{mm} = 600\,\mathrm{mm}$, and the assumed wavelength $\lambda = 800\,\mathrm{nm}$ fits exactly 750,000 times into this length. Therefore, the only remaining phase-shift stems from the reflection at the total reflective mirror.

Further we notice that there are still not visible artifacts or distortions, although the phase-gradients in the input image correspond to tilted light-rays in the corresponding ray-model.

3. A fast and efficient 4f-cavity-CPA simulation

In this chapter we develop a fast and efficient method for using the unattenuated round-trip-simulation of the previous chapter to simulate the effect of the infinite number of round-trips in an attenuated, plane-parallel 4f-cavity with one perfect mirror and one partially reflecting mirror as shown in figure 1.10.

3.1. Properties of an ideal partially reflecting mirror

Correctly modeling the exact amplitude and phase relations of the partially reflecting mirror of the 4f cavity is important for the simulation of the 4f-cavity-CPA. Interestingly, standard optics textbooks are quite vague when it comes to describing the exact amplitude and phase relations of the transmitted and reflected light waves going through a partially reflecting mirror, let alone a partially reflecting mirror with arbitrary reflection coefficient. What's more, ever so often it is hinted that a certain phase is assigned to, e.g., the reflected wave just *by convention*, and that all other phase relations are then just following this convention (see, e.g., [16, p. 450] or [17, p. 1265]).

Or, as Salik puts it in [18, p. 223]:

> Students do not receive a consistent description of phase shifts upon reflection when they consult optical textbooks. [...] The analytical treatment of phase shifts also takes slightly different forms.

Salik then provides the following explanations for the confusion:

> The inconsistency of the phase shifts upon reflection is due in part to the conventions used in the derivation of the Fresnel equations. [...] The reflection phase shifts are not to be understood as the phase change relative to the incident beam; rather, they are in reference to the assumed directions of the electric field vectors at the point of incidence. [...] Even though conventions for the assumed field directions resolve most of the discrepancy, some authors who use the same conventions calculate these phase shifts differently. [...] What

explains the difference between textbooks that use the same conventions is yet another convention in the expression of polarized states of the electromagnetic waves. In Eq. (1), we expressed the fields as $Acos(kz - \omega t + \phi_1)$. However, the same fields can also be expressed as $Acos(\omega t - kz + \phi_2)$.

That these *conventions for the assumed field directions* are necessary becomes clear when one visualizes that the electric and magnetic field-vectors for the incident light field, the reflected light field, and the refracted light field need three different coordinate systems, which allow for 8 different coordinate-system conventions, as depicted in figure 3.1

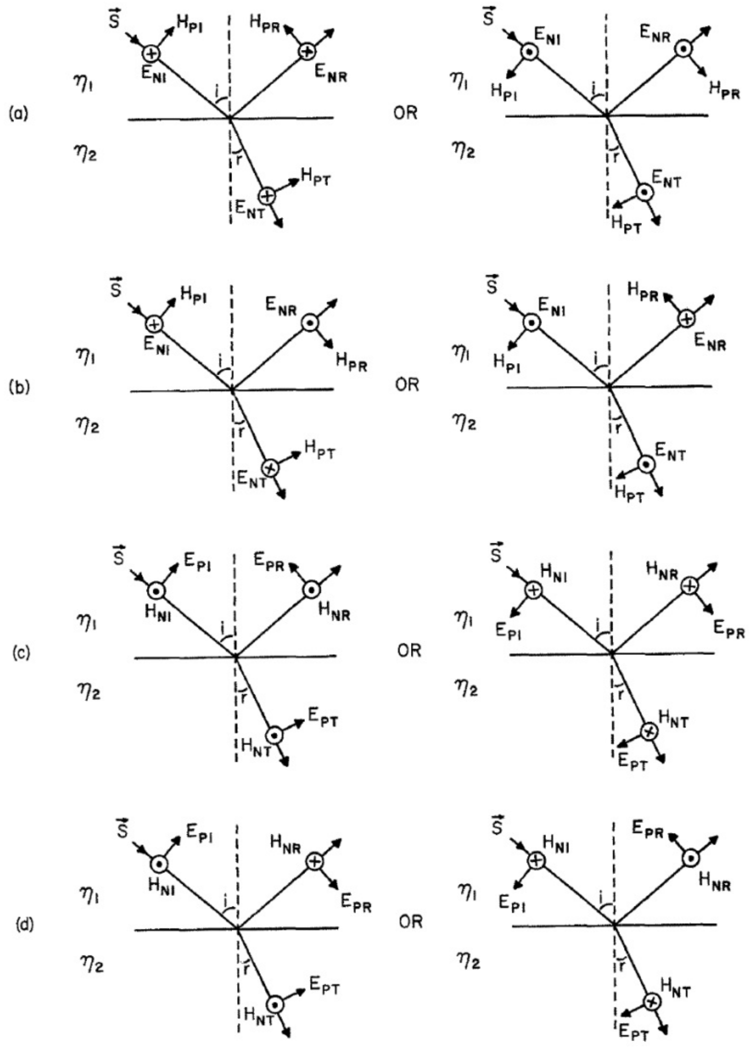

Figure 3.1.: Eight different conventions for modeling reflection and refraction of linearly polarized light at a boundary. E and H are the electric and magnetic vectors, n_1 and n_2 the refractive indices of the two media. From: [19, p. 136]

Switching between the eight conventions depicted in figure 3.1 can make a phase-difference of π. To complicate things even further, we additionally have to face the challenge of the scalar optics simplification. In our simulation we do not implicitly distinguish between different directions of the $\vec{E}$ and $\vec{B}$ vectors.

3.1.1. Creating a model suitable for scalar optics

Since we are designing a 4f-cavity-CPA, the exact amplitude and phase relations of the transmitted and reflected light waves at the partially reflecting mirror are important, and must be accommodated in our scalar-optics model. To deal with the challenges explained in the previous section, we derive the complex phasor relations of incoming, reflected, and transmitted light of a partially reflecting mirror by employing the simple 1D model shown in figure 3.2.

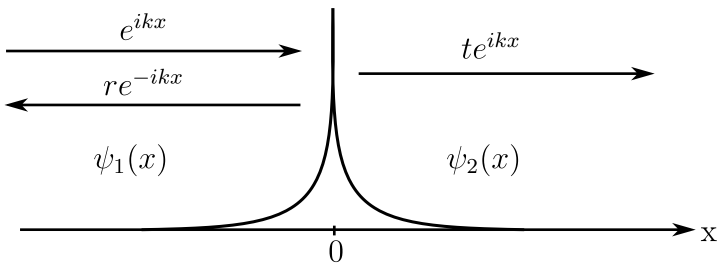

Figure 3.2.: Deriving phasor relations of incoming, reflected, and transmitted light for a partially reflecting mirror with a simple 1D model.

In this model, a plane wave hits a delta potential of a given height at position $x = 0$, where it is partially reflected and transmitted. The superposition of the incoming and the reflected wave is represented by $\psi_1(x)$, and the transmitted wave is represented by $\psi_2(x)$:

$$\psi_1(x) = e^{ikx} + re^{-ikx} \tag{3.1}$$

$$\psi_2(x) = te^{ikx} \tag{3.2}$$

The whole system is governed by a one-dimensional version of the Helmholtz equation, which we can get by taking equation (2.9) and substituting $U(\vec{r}) \rightarrow \psi(x)$, $\vec{\nabla}^2 \rightarrow \frac{\partial^2}{\partial x^2}$, and $k^2 \rightarrow n^2(x)k^2$:

$$\frac{\mathrm{d}^2}{\mathrm{d}x^2}\psi(x) + n^2(x)k^2\psi(x) = 0 \qquad \Big|\cdot\frac{1}{k^2}$$

$$\frac{1}{k^2}\frac{\mathrm{d}^2}{\mathrm{d}x^2}\psi(x) + n^2(x)\psi(x) = 0 \tag{3.3}$$

with

$$n^2(x) = \begin{cases} 1 & x \neq 0 \\ V_0 \delta(x) & x = 0. \end{cases} \tag{3.4}$$

Solving equation (3.3) under consideration of equation (3.4) with the adequate matching conditions results in (see appendix A.6):

$$r = -\frac{kV_0}{kV_0 + 2i} \tag{3.5}$$

$$t = \frac{2i}{kV_0 + 2i} \tag{3.6}$$

Now, let r_0 be the absolute value of the reflection coefficient of a partially reflecting mirror, so that

$$r_0 \overset{\text{def}}{=} |r| \tag{3.7}$$

By inserting (3.5) into (3.7), and solving the resulting equation for V_0, we get (see appendix A.7):

$$V_0(r_0) = \pm \frac{2r_0}{k\sqrt{1 - r_0^2}} \tag{3.8}$$

Inserting this into equations (3.5) and (3.6) allows to express $r(r_0)$ and $t(r_0)$, i.e., the complex-valued reflection coefficient r and transmission coefficient t based on a given absolute value of the reflection coefficient $r_0 \overset{\text{def}}{=} |r|$ (see appendix A.8).

$$r(r_0) = -r_0^2 \pm i r_0 \sqrt{1 - r_0^2} \tag{3.9}$$

$$t(r_0) = 1 - r_0^2 \pm i r_0 \sqrt{1 - r_0^2} \tag{3.10}$$

Both equations (3.9) and (3.10) are ambiguous because of the ± sign. However, as long as we choose the sign consistently in both equations, our calculations will represent the phase relation between *transmitted* and *reflected* light wave at the partially reflective mirror correctly so that all energy conditions are fulfilled. Of course, depending on our choice of sign, we will get different results regarding the phase difference between the *incident* and the *reflected* wave. However — as r_0 is constant in a given setup — any deviation between our calculation and an actual experiment with respect to that can be instantly compensated by the experimenter by adjusting the total length of the 4f-cavity by an according fraction of a wavelength[1].

Having said that, let's visualize the two alternative formulas for the reflection coefficient $r(r_0)$ in figure 3.3 for a reasonable choice.

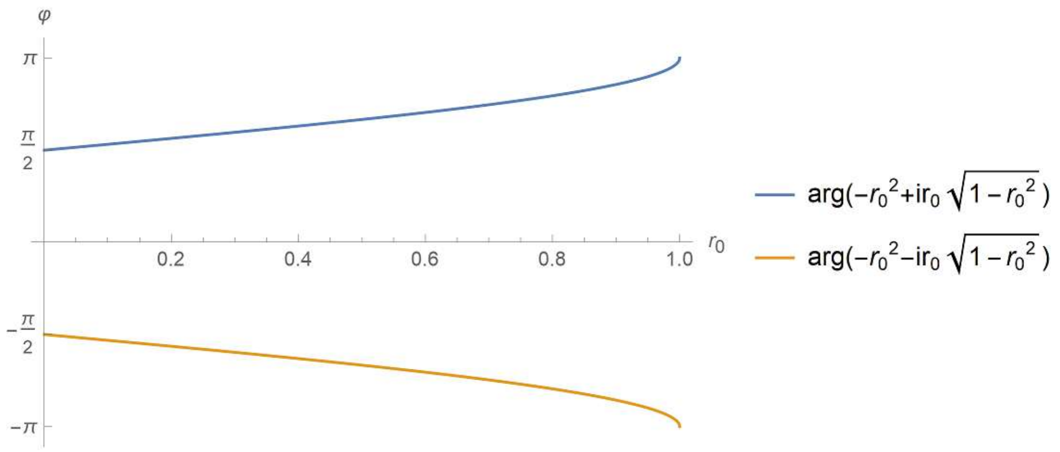

Figure 3.3.: Two possible formulas to model the phase-difference between the reflected and the incident beam of a partially reflective mirror as a function of the absolute value of the reflection coefficient.

As we expect an $e^{i\pi} = -1$ phase jump at total reflection, one could be tempted to select the upper curve where $r(1) = \pi$. However, at second thought, also $e^{-i\pi} = -1$, so that is not really a criterion. Therefore, we take the fact that, for metals, at normal incidence, the component of the electric field vector normal to the plane of incidence suffers a phase change of $\phi = -\pi$, (see [20, p. 271]), and therefore we pick the lower curve from figure 3.3.

Hence, our model for calculating the reflection phasor r and transmission phasor t given the absolute value of the reflection coefficient r_0 from now on is:

$$r(r_0) = -r_0^2 - ir_0\sqrt{1-r_0^2} \tag{3.11}$$

$$t(r_0) = 1 - r_0^2 - ir_0\sqrt{1-r_0^2} \tag{3.12}$$

[1] At the end of section 3.2.2, we will put the hypothesis that the choice of sign in equations (3.9) and (3.10) is irrelevant to the test.

Figure 3.4 shows how the arguments of the $r(r_0)$ and $t(r_0)$ functions develop. One can clearly see that they maintain a phase distance of exactly $\frac{\pi}{2}$. Also, figure 3.5, showing the plot of the absolute value of the transmission phasor as a function of r_0, makes sense: the larger the absolute value of the reflection, the smaller the absolute value of the transmission.

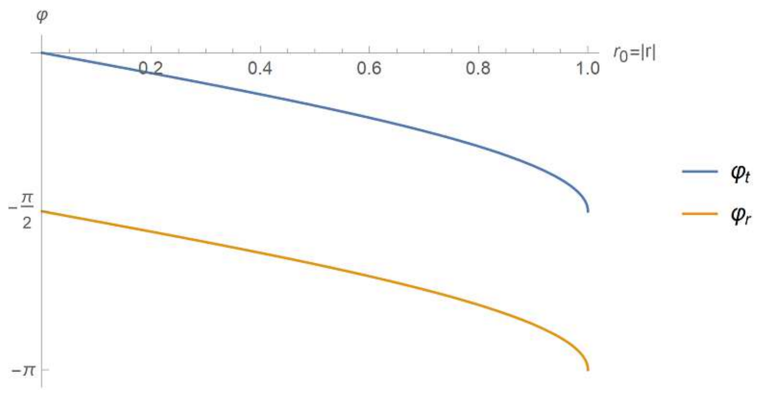

Figure 3.4.: Phase shifts of reflected and transmitted light in relation to the incident light beam as a function of the absolute value of the reflection coefficient of a partially reflecting mirror

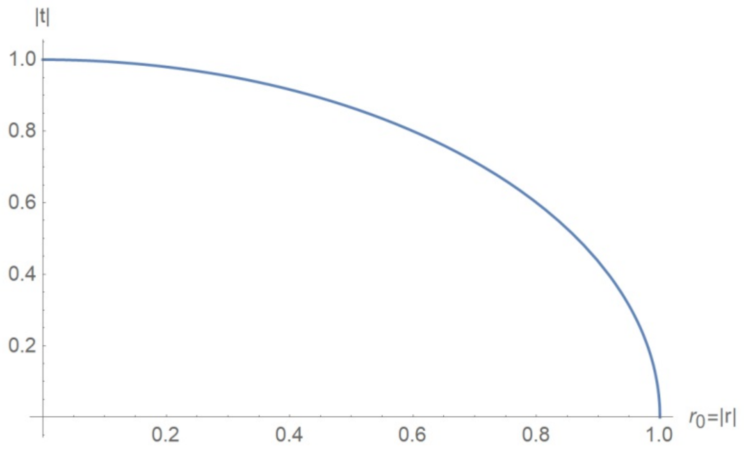

Figure 3.5.: Absolute value of the transmission coefficient of a partially reflecting mirror as a function of the absolute value of the reflection coefficient.

3.1.2. Plausibility checks

Energy conservation

For the sake of energy conservation, we expect for a partially transparent, lossless mirror:

$$|r(r_0)|^2 + |t(r_0)|^2 = 1 \tag{3.13}$$

This condition is satisfied as proven in appendix A.9.

Phase difference between transmitted and reflected light

Degiorgio shows in [21] that in a semireflecting lossless mirror – also because of energy conservation – the phase difference between the transmitted and reflected optical field always has to be $\frac{\pi}{2}$:

$$\arg\left(t(r_0)\right) - \arg\left(r(r_0)\right) = \frac{\pi}{2} \tag{3.14}$$

Figure 3.4 shows visually that this condition is fulfilled. The mathematical proof can be found in appendix A.10.

Unitarity of the scattering matrix

Finally, energy conservation also dictates that the scattering matrix of a partially reflective, lossless mirror must be unitary (see [16, p. 405]):

$$\underline{\underline{S}}\,\underline{\underline{S}}^\dagger = \begin{bmatrix} r & t \\ t & r \end{bmatrix} \begin{bmatrix} r^* & t^* \\ t^* & r^* \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{3.15}$$

The mathematical proof that the unitarity-condition is fulfilled can be found in appendix A.11.

3.2. A simple, one-dimensional CPA as a toy model

3.2.1. Reflection coefficient and critical coupling

We now consider a simple one-dimensional optical cavity of length l enclosed by two plane-parallel mirrors as depicted in figure 3.6. The right mirror M_2 is a perfect totally reflective mirror with a reflection coefficient $r_2 = e^{i\pi} = -1$ and reflectance $R_2 = |r_2|^2 = 1$. The left mirror M_1 is a symmetrical partially reflecting mirror characterized by the complex reflection coefficient r_1 and reflectance $R_1 = |r_1|^2 \overset{\text{def}}{=} r_0^2$, such that $0 < R < 1$. Light propagation within the cavity is characterized by a complex wavenumber $\tilde{k} = k + i\kappa$, with the attenuation constant κ effective over the whole length l.

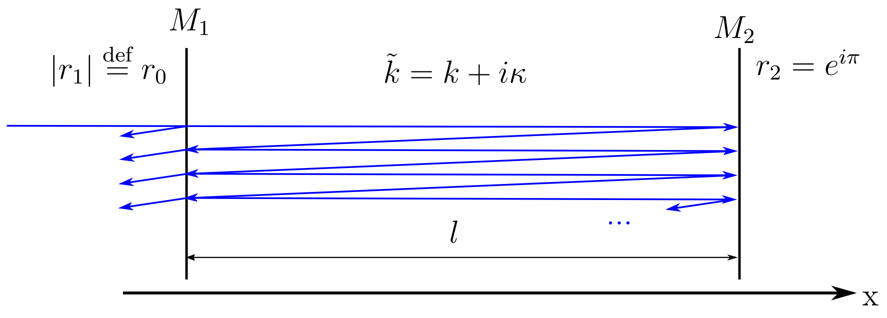
M_1
$|r_1| \overset{\text{def}}{=} r_0$
$\tilde{k} = k + i\kappa$
M_2
$r_2 = e^{i\pi}$
l
...
x

Figure 3.6.: A simple one-dimensional plane-parallel optical cavity with a fully reflective mirror M_2 on the right side and a partially reflective mirror M_1 on the left side.

To calculate the reflection coefficient of the total cavity r_{cav} for an incident beam from the left side, we just have to follow each individual light-path and sum up all contributions:

- The fraction of light that gets directly reflected at the left mirror: r_1
- The fraction of light that gets transmitted through the left mirror, travels from left to right, gets reflected at the right mirror, travels back from right to left to left, and then leaves the cavity to the left side after being once more transmitted through the left mirror: $t_1 e^{i\tilde{k}l} r_2 e^{i\tilde{k}l} t_1$
- The fraction of light that gets transmitted through the left mirror, takes a round-trip, gets reflected at the left mirror, takes another round-trip, and *then* leaves the cavity to the left side after being once more transmitted through the left mirror: $t_1 e^{i\tilde{k}l} r_2 e^{i\tilde{k}l} r_1 e^{i\tilde{k}l} r_2 e^{i\tilde{k}l} t_1$
- etc.

$$\begin{aligned}
r_{cav} &= r_1 + \underline{t_1 \cdot e^{i\tilde{k}l} r_2 e^{i\tilde{k}l} \cdot t1} \\
&\quad + \underline{t_1 \cdot e^{i\tilde{k}l} r_2 e^{i\tilde{k}l}} \cdot \quad \underline{r_1 e^{i\tilde{k}l} r_2 e^{i\tilde{k}l}} \quad \cdot \underline{t1} \\
&\quad + \underline{t_1 \cdot e^{i\tilde{k}l} r_2 e^{i\tilde{k}l}} \cdot \left(\underline{r_1 e^{i\tilde{k}l} r_2 e^{i\tilde{k}l}} \right)^2 \cdot \underline{t1} \\
&\quad + \dots \\
r_{cav} &= r_1 + \underline{t_1^2 r_2 e^{2i\tilde{k}l}} \sum_{n=0}^{\infty} \left(\underline{r_1 r_2 e^{2i\tilde{k}l}} \right)^n & \left| \sum_{n=0}^{\infty} q^n = \frac{1}{1-q} \quad \text{(geometric series)} \right. \\
r_{cav} &= r_1 + t_1^2 r_2 e^{2i\tilde{k}l} \frac{1}{1 - r_1 r_2 e^{2i\tilde{k}l}} & \left| r_2 = e^{i\pi} = -1 \right.
\end{aligned} \tag{3.16}$$

$$\begin{aligned}
r_{cav} &= r_1 - \frac{t_1^2 e^{2i\tilde{k}l}}{1 + r_1 e^{2i\tilde{k}l}} \\
r_{cav} &= r_1 - \frac{t_1^2}{e^{-2i\tilde{k}l} + r_1} \overset{(3.11)}{\Longrightarrow} \overset{(3.12)}{\Longrightarrow} \\
r_{cav} &= -r_0^2 - ir_0\sqrt{1 - r_0^2} - \frac{\left(1 - r_0^2 - ir_0\sqrt{1 - r_0^2}\right)^2}{e^{-2i\tilde{k}l} - r_0^2 - ir_0\sqrt{1 - r_0^2}}
\end{aligned} \tag{3.17}$$

Equation (3.16) is consistent with the result presented in [16, p. 422], except for a $\frac{\pi}{2}$ phase-convention in t. In order to realize a coherent perfect absorber, the reflection coefficient r_{cav} must vanish. Therefore, we have to set the right part of equation (3.17) equal to zero. This enables us — after a two-page calculation in appendix A.12 — to express the condition that has to be fulfilled by the complex wavenumber $\tilde{k}$:

$$\tilde{k}_c = \frac{1}{2l}\left(2\pi n - \arctan\left(\frac{\sqrt{1 - r_0^2}}{r_0}\right)\right) - i\frac{1}{2l}\ln(r_0) \quad \text{with} \quad n \in \mathbb{Z} \tag{3.18}$$

Because we have defined $\tilde{k} = k + i\kappa$, the conditions, which $k_c = \mathrm{Re}(\tilde{k}_c)$ and $\kappa_c = \mathrm{Im}(\tilde{k}_c)$ have to fulfill for critical coupling, can be read off directly from equation (3.18):

$$\begin{aligned}
k_c &= \mathrm{Re}(\tilde{k}_c) \overset{(3.18)}{\Longrightarrow} \\
k_c &= \frac{1}{2l}\left(2\pi n - \arctan\left(\frac{\sqrt{1 - r_0^2}}{r_0}\right)\right) \quad \text{with} \quad n \in \mathbb{Z}
\end{aligned} \tag{3.19}$$

$$\kappa_c = \mathrm{Im}\,(\tilde{k}_c) \overset{(3.18)}{\Longrightarrow}$$
$$\kappa_c = -\frac{1}{2l}\ln(r_0) \tag{3.20}$$

The total attenuation for one round-trip from left to right and back is obviously given by $e^{-2\kappa l}$. Inserting equation (3.20) into this expression, we get:

$$e^{-2\kappa_c l} = e^{\cancel{2l}\frac{1}{\cancel{2l}}\ln(r_0)}$$
$$e^{-2\kappa_c l} = r_0 \tag{3.21}$$

This is a remarkably simple and pleasing result. The cavity's reflection coefficient r_{cav} vanishes when the absolute value of the left mirror's reflection coefficient exactly equals the loss term $e^{-2\kappa_c l}$.

This is often called the *impedance-matched* situation, because our setup — as it consists of an optical resonator driven from the exterior via a certain coupling constant determined by the partially reflective mirror — corresponds to a resonant circuit, consisting of a "load" with a certain input impedance, driven by a signal source with an equal output impedance. When the input impedance of the load matches the output impedance of the source, reflections are minimized (cf. [16, p. 423]).

3.2.2. Simulation results

By substituting $\tilde{k}$ with $\frac{2\pi}{\lambda} + i\kappa_c$ in equation (3.17) and further substituting κ_c with the result from equation (3.20), we are now in the position to calculate the total reflectance $|r_{cav}|^2$ of the cavity.

Figures 3.7 and 3.8 show the result of such one-dimensional simulations under the assumption of $L = 300\,\mathrm{mm}$, $\lambda = 800\,\mathrm{nm}$ and $r_0 = 0.7$ for critical attenuation and various over- and under-critical attenuations. As expected, the reflectance drops to zero at resonance conditions if the attenuation matches the critical attenuation. These plots coincide very well with figure 11.16 in [16, p. 423].

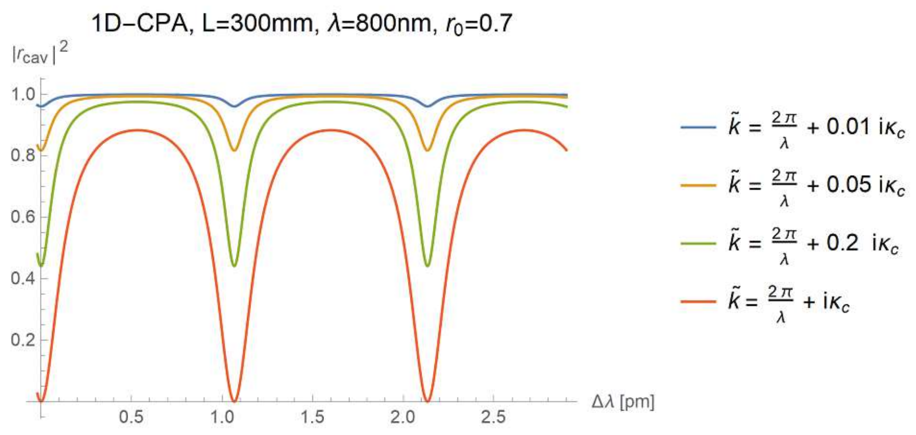

Figure 3.7.: Reflectance of a 1D-cavity with an $|r| = r_0 = 0.7$ mirror assuming a coherent incident light beam with a wavelength of $\lambda = 800\,\mathrm{nm}$. The red curve shows the results with critical attenuation, the other curves the result with various under-critical attenuations.

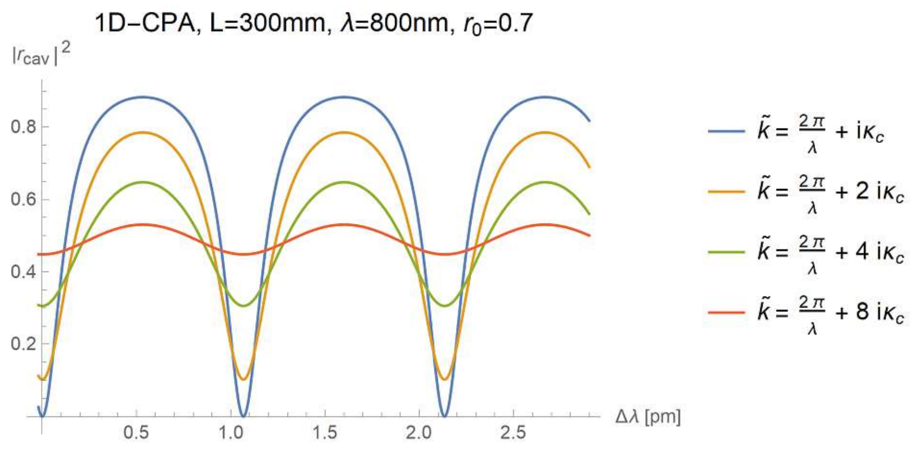

Figure 3.8.: Reflectance of a 1D-cavity with an $|r| = r_0 = 0.7$ mirror assuming a coherent incident light beam with a wavelength of $\lambda = 800\,\mathrm{nm}$. The blue curve shows the result with critical attenuation, the other curves the results with various over-critical attenuations.

Finally, we can check whether our choice of sign in equations (3.9) and (3.10) really has no relevant effect to our simulations by calculating the critically attenuated 1D-CPA with either choice of sign.

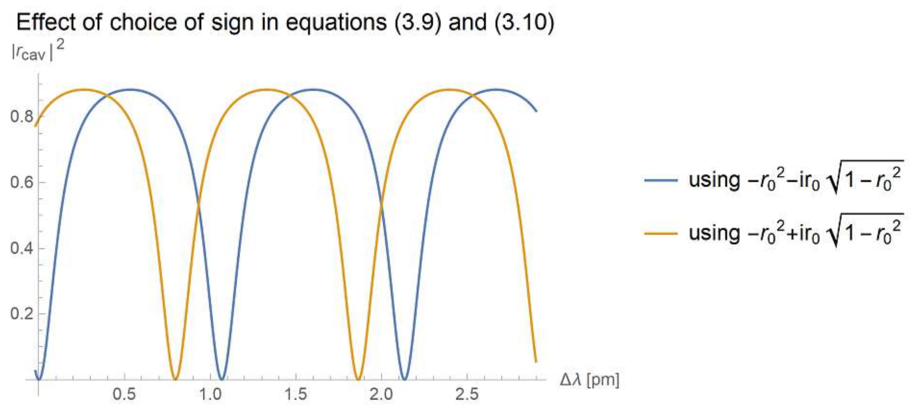

Figure 3.9.: Effect of choice of sign in equations (3.9) and (3.10) to the 1D-CPA-simulation.

As can be seen in figure 3.9, the CPA-effect appears with either choice of sign. The only difference (assuming an unchanged cavity length) is a minimal shift in the resonance wavelength.

3.2.3. Plausibility check

We now assume a 1D-cavity without attenuation, i.e., we assume that $\kappa = 0$, and therefore $\tilde{k} = k$. Replacing $\tilde{k}$ with k in equation (3.17), and doing a lengthy transformation (see appendix A.13), we can convert equation (3.17) into equation (3.22), which explicitly shows the real part and the imaginary part of the cavity's total (outside) reflection coefficient r_{cav}.

$$
\begin{aligned}
r_{cav} = & \frac{(3r_0^2 - 1)\cos(2kl) - 2r_0\sqrt{1-r_0^2}\sin(2kl) - 2r_0^2}{1 + r_0^2 - 2r_0^2\cos(2kl) + 2r_0\sqrt{(1-r_0^2)}\sin(2kl)} \\
& + i\frac{2r_0\sqrt{1-r_0^2}\cos(2kl) + (r_0^2 - 1)\sin(2kl) - 2r_0\sqrt{(1-r_0^2)}}{1 + r_0^2 - 2r_0^2\cos(2kl) + 2r_0\sqrt{(1-r_0^2)}\sin(2kl)}
\end{aligned}
\tag{3.22}
$$

Plotting the real part and the imaginary part of r_{cav} reveals a well-known behavior around the resonance point (see figure 3.10).

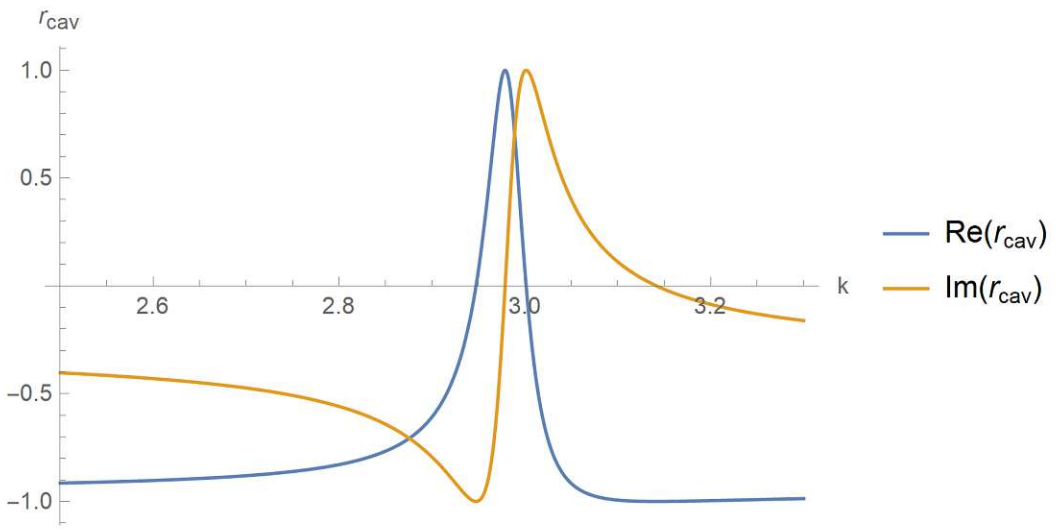

Figure 3.10.: The real and imaginary parts of r_{cav} reveal the well-known behavior of resonant systems around $k = k_c$. (Parameters of the cavity: $l = 1\,\mathrm{m}, r_0 = 0.95, \kappa = 0$)

The representation in equation (3.22) with separated real part and imaginary part allows the calculation of $\varphi_{cav} = \arg(r_{cav})$:

$$\begin{aligned}
\varphi_{cav} &= \arg(r_{cav}) \\
\varphi_{cav} &= \operatorname{arctan2}(\mathrm{Re}(r_{cav}),\mathrm{Im}(r_{cav})) \overset{(3.22)}{\Longrightarrow} \\
\varphi_{cav} &= \operatorname{arctan2}\Big(\Big(3r_0^2 - 1\Big)\cos(2kl) - 2r_0\sqrt{1-r_0^2}\sin(2kl) - 2r_0^2, \\
&\qquad 2r_0\sqrt{1-r_0^2}\cos(2kl) + (r_0^2-1)\sin(2kl) - 2r_0\sqrt{(1-r_0^2)}\Big)
\end{aligned} \tag{3.23}$$

Figure 3.11 below visualizes the phase behavior of φ_{cav} around a resonance point.

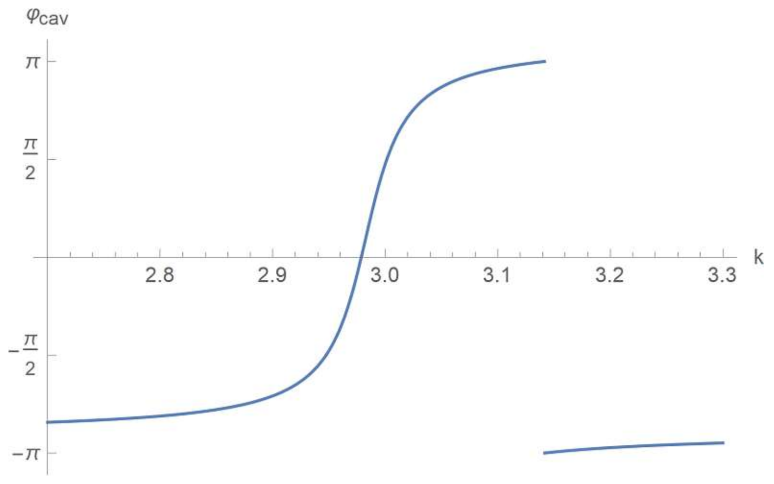

Figure 3.11.: The phase behavior of the cavity's outside reflection coefficient $\varphi_{cav} = \arg(r_{cav})$ around the first resonance point. $(l = 1\,\mathrm{m}, r_0 = 0.95, \kappa = 0)$

Figure 3.11 shows the behavior of an undamped 4f-cavity's outside reflection coefficient at the partially reflective mirror around the first resonance point, assuming a cavity length of $l = 1\,\mathrm{m}$, and the absolute value of the reflection coefficient being $r_0 = 0.95$. One can see a 2π phase shift around $k = k_c = 2.98\,\mathrm{m}^{-1}$, which can be calculated using equation (3.19) by setting $n = 1$, $l = 1\,\mathrm{m}$, and $r_0 = 0.95$.

For a plausibility check, we want to refer to the input-output theory of a damped quantum system, where a single mode in a cavity like our system is governed by the stochastic quantum Langevin equation as described in [22, p. 3763, eq. (2.13)]:

$$\dot{a}(t) = -i\omega_0 a(t) - \frac{\gamma}{2} a(t) - \sqrt{\gamma} b_{in}(t) \tag{3.24}$$

In equation (3.24), $b_{in}(t)$ represents the incoming field, $a(t)$ the harmonic oscillator cavity field, and γ the damping coefficient of the Markovian damping term, which stems from the fact that — although we do not assume explicit damping in the cavity (we chose $\kappa = 0$) — there is still energy loss by means of light leaving the cavity through the partially reflective mirror.

The relation between the incoming field $b_{in}(t)$ and the outgoing field $b_{out}(t)$ is given by [22, p. 3764, eq. (2.22)]:

$$b_{out}(t) = b_{in}(t) + \sqrt{\gamma} a(t). \tag{3.25}$$

By Fourier-transforming equations (3.24) and (3.25), the according equations in frequency-space can be obtained:

$$-i\omega a(\omega) = -i\omega_0 a(\omega) - \frac{\gamma}{2}a(\omega) - \sqrt{\gamma}b_{in}(\omega) \tag{3.26}$$

$$b_{out}(\omega) = b_{in}(\omega) + \sqrt{\gamma}a(\omega) \tag{3.27}$$

Solving this system of equations leads to the input-output relation

$$b_{out}(\omega) = \frac{i\left(\omega_0 - \omega\right) - \frac{\gamma}{2}}{i\left(\omega_0 - \omega\right) + \frac{\gamma}{2}} b_{in}(\omega) \tag{3.28}$$

which can be expressed as

$$b_{out}(\omega) = e^{i\varphi_{QLE}(\omega)} b_{in}(\omega) \tag{3.29}$$

with the phase-difference $\varphi_{QLE}(\omega)$ being

$$\varphi_{QLE}(\omega) = \pi + 2\arctan\left(\frac{2\left(\omega - \omega_0\right)}{\gamma}\right). \tag{3.30}$$

In an equivalent representation, we can re-write equation (3.30) from frequency-space to k-space:

$$\varphi_{QLE}(k) = \pi + 2\arctan\left(\frac{2\left(k - k_0\right)}{K}\right) \tag{3.31}$$

Equation (3.31) shows the same 2π phase shift at $k = k_0$ as we can also observe in figure 3.11 at $k = k_c$, which is a first hint that equation (3.23) is correct.

However, comparing the first derivative $\frac{\mathrm{d}\varphi_{cav}}{\mathrm{d}k}$ (by calculating the derivative of equation (3.23)) with the first derivative $\frac{\mathrm{d}\varphi_{QLE}}{\mathrm{d}k}$ (by calculating the derivative of equation (3.30)) will prove to become a much deeper plausibility check, as it will reveal that the full-width-at-half-maximum (FWHM) value of $\frac{\mathrm{d}\varphi}{\mathrm{d}k}$ encodes a very good estimation of the cavity's critical damping κ_c, as derived in equation (3.20).

Please note that the relation between FWHM and critical damping is usually drawn from the amplitude as a function of frequency or wavenumber. However, as the cavity that we are currently considering has no explicit damping (we chose $\kappa = 0$), the amplitude of the reflected field is constant and does not carry any information.

Now let us first consider the first derivative $\frac{\mathrm{d}\varphi_{cav}}{\mathrm{d}k}$, which follows from equation (3.23), and can be expressed as:

$$\begin{aligned}\frac{\mathrm{d}\varphi_{cav}}{\mathrm{d}k} = &\left\{2l\left(r_0^2-1\right)\left(2r_0^2\cos(2kl) - r_0^2 - 2r_0\sqrt{1-r_0^2}\sin(2kl) - 1\right)\right\}\cdot \\ &\left\{2r_0\left[2\left(r_0+r_0^3\right)\cos(2kl) + \left(r_0 - 2r_0^3\right)\cos(4kl) + \right.\right. \\ &\left.\left. 2\sqrt{1-r_0^2}\left(2r_0^2\cos(2kl) - r_0^2 - 1\right)\sin(2kl)\right] - r_0^2(4+r_0^2) - 1\right\}^{-1}\end{aligned} \tag{3.32}$$

Figure 3.12 shows the behavior of equation (3.32) around the first resonance point, assuming a cavity length of $l = 1\,\mathrm{m}$, and the absolute value of the reflection coefficient being $r_0 = 0.95$.

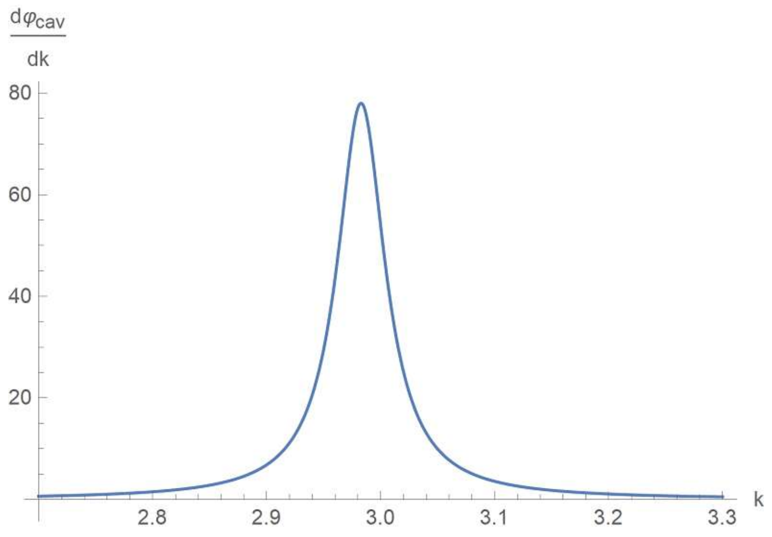

Figure 3.12.: The first derivative of the cavity's outside reflection coefficient phase $\frac{\mathrm{d}\varphi_{cav}}{\mathrm{d}k} = \frac{\mathrm{d}}{\mathrm{d}k}\arg(r_{cav})$ around a resonance point ($l = 1\,\mathrm{m}, r_0 = 0.95, \kappa = 0$).

In comparison, calculating the first derivative of equation (3.31) yields

$$\frac{\mathrm{d}\varphi_{QLE}}{\mathrm{d}k} = \frac{4K}{4\left(k-k_0\right)^2 + K^2}. \tag{3.33}$$

By substituting $k_0 \rightarrow k_c$ and $K \rightarrow 2\kappa_c$, and calculating k_c and κ_c according to equations (3.19) and (3.20), we get a virtually perfect match between $\frac{\mathrm{d}\varphi_{cav}}{\mathrm{d}k}$ (figure 3.12) and $\frac{\mathrm{d}\varphi_{QLE}}{\mathrm{d}k}$ (figure 3.13) around the resonance point.

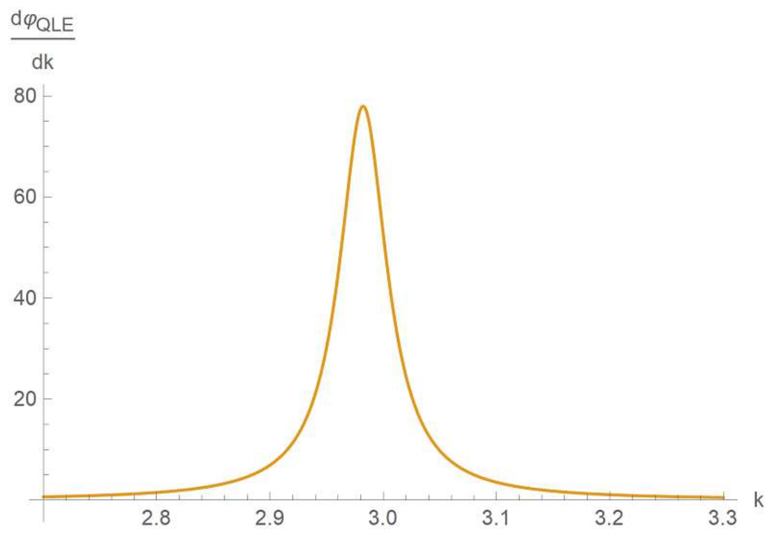

Figure 3.13.: The first derivative of the cavity's outside reflection coefficient phase $\frac{\mathrm{d}\varphi_{QLE}}{\mathrm{d}k}$ around the first resonance point, calculated based on the stochastic quantum Langevin equation (QLE) (parameters: $l = 1\,\mathrm{m}, r_0 = 0.95, \kappa = 0$).

To see that the heuristically determined substitution $K \rightarrow 2\kappa_c$ makes sense, we need to express K as a function of the partially reflective mirror's transmissivity $|t_1|^2$. The larger $|t_1|^2$ gets, the more energy can leave the cavity, and the larger we therefore expect the QLE damping coefficient K in equation (3.33) to become.

$$\begin{aligned}
&K = 2\kappa_c \overset{(3.20)}{\Longrightarrow} \\
&K = -\cancel{2}\frac{1}{\cancel{2}l}\ln\left(r_0\right) && |r_0 = |r_1| \\
&K = -\frac{1}{l}\ln\left(|r_1|\right) && \Big|\, |r_1|^2 + |t_1|^2 = 1 \implies |r_1| = \sqrt{1-|t_1|^2} \\
&K = -\frac{1}{l}\ln\left(\sqrt{1-|t_1|^2}\right) \\
&K = -\frac{1}{2l}\ln\left(1-|t_1|^2\right)
\end{aligned} \tag{3.34}$$

By developing $\ln\left(1-|t_1|^2\right)$ around small values of $|t_1|^2$, we get

$$\ln\left(1-|t_1|^2\right)\approx -|t_1|^2 \tag{3.35}$$

Inserting equation (3.35) into equation (3.34) yields the final result

$$K\approx\frac{|t_1|^2}{2l} \tag{3.36}$$

As expected, there is an almost linear relation between K and $|t_1|^2$ for small values of $|t_1|^2$. Also, the constant factor $\frac{1}{2l}$ makes sense from the standpoint of units as well as $2l$ being the length of a full round-trip.

In the next step of our plausibility check, we want to find out how FWHM $\left(\frac{\mathrm{d}\varphi_{QLE}}{\mathrm{d}k}\right)$ is related to κ_c. To do so, we explicitly perform both substitutions $k_0\to k_c$ and $K\to 2\kappa_c$ in equation (3.33), and get:

$$\frac{\mathrm{d}\varphi_{QLE}}{\mathrm{d}k}=\frac{2\kappa_c}{(k-k_c)^2+\kappa_c^2} \tag{3.37}$$

After a brief derivation (see appendix A.14) it turns out that

$$\kappa_c=\frac{1}{2}\,\mathrm{FWHM}\left(\frac{\mathrm{d}\varphi_{QLE}}{\mathrm{d}k}\right) \tag{3.38}$$

Equipped with this insight, we can now — as a plausibility check for equation (3.17) — numerically evaluate the FWHM-value of equation (3.32) around the first resonance-point, to see if the relation from equation (3.38) also holds for $\frac{\mathrm{d}\varphi_{cav}}{\mathrm{d}k}$.

Figure 3.14 shows the result of this calculation. It turns out that in a numerical evaluation of the chosen example ($l = 1\,\mathrm{m}, r_0 = 0.95$) the anticipated relation $\frac{1}{2}\,\mathrm{FWHM}\left(\frac{\mathrm{d}\varphi_{cav}}{\mathrm{d}k}\right) = \kappa_c$ holds with an accuracy of three significant digits: $\kappa_c = 0.025647\ \mathrm{m}^{-1}$, and $\frac{1}{2}\,\mathrm{FWHM}\left(\frac{\mathrm{d}\varphi_{cav}}{\mathrm{d}k}\right) = 0.025652\ \mathrm{m}^{-1}$.

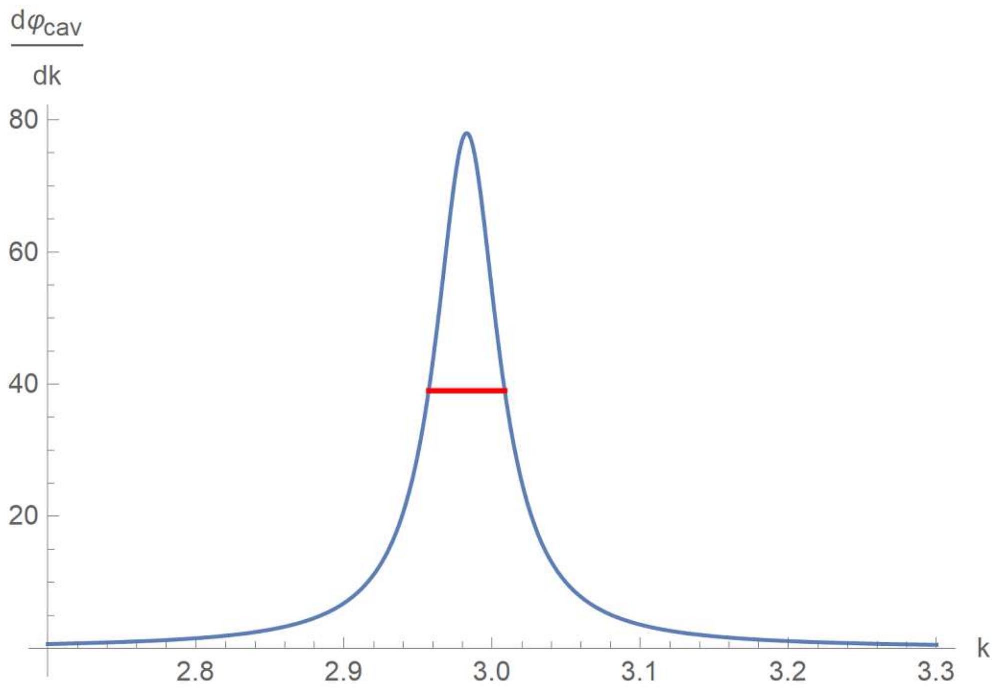

Figure 3.14.: The first derivative of the cavity's outside reflection coefficient phase $\frac{\mathrm{d}\varphi_{cav}}{\mathrm{d}k} = \frac{\mathrm{d}}{\mathrm{d}k}\arg\left(r_{cav}\right)$ around a resonance point ($l = 1\,\mathrm{m}, r_0 = 0.95, \kappa = 0$). The red line marks the full width at half maximum (FWHM). As expected: $\frac{1}{2}\,\mathrm{FWHM}\left(\frac{\mathrm{d}\varphi_{cav}}{\mathrm{d}k}\right) = \kappa_c$.

3.3. Extending the simple 1D approach to 3D: Calculating a 4f-Cavity-CPA

In this section, we consider a 4f-cavity as depicted in Figure 3.15.

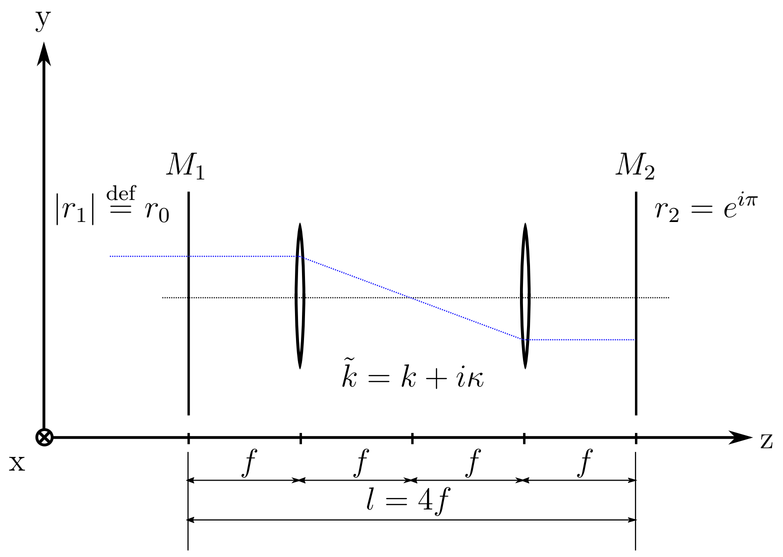
y
M_1
$|r_1| \overset{\text{def}}{=} r_0$
M_2
$r_2 = e^{i\pi}$
$\tilde{k} = k + i\kappa$
x
z
f
f
f
f
$l = 4f$

Figure 3.15.: An optical 4f-cavity with a fully reflective mirror M_2 on the right side and a partially reflective mirror M_1 on the left side.

Again, the right mirror M_2 is a perfect total reflective mirror with a reflection coefficient $r_2 = e^{i\pi} = -1$ and reflectance $R_2 = |r_2|^2 = 1$, and the left mirror M_1 is a symmetrical, partially reflecting mirror, characterized by the complex reflection coefficient r_1 and reflectance $R_1 = |r_1|^2 \overset{\text{def}}{=} r_0^2$, so that $0 < R < 1$. Similar as in the previous section, light propagation within the cavity is characterized by a complex wavenumber $\tilde{k} = k + i\kappa$, with the attenuation constant κ effective over the whole length l.

In contrast to figure 3.6 there are now two converging lenses inside the cavity. The first lens is positioned exactly one focal length f after the left mirror, and the second lens is positioned two focal lengths after the first lens. The right mirror is eventually positioned yet another focal length after the second lens, hence altogether at a distance of four focal lengths after the left mirror.

As already explained in section 1.3, this arrangement ensures that (all lens-errors and other side-effects aside) each beam entering the cavity from the left side through mirror M_1 is always reflected into itself and then bounces back and forth the optical cavity following the very same path. Therefore, we can expect that the results from the previous section can be, in first approximation, transferred to this advanced cavity with satisfying accuracy.

Using the scalar optics computer simulation developed in chapter 2, we can express any incident wave-front coming from the left side as a two-dimensional complex function in the xy-plane. Observing a limited square-shaped xy-area centered with respect to the optical axis, we can decompose the incoming wavefront into eigenstates by Fourier Transformation.

Because our computer simulation necessarily decomposes the field on the xy-plane into a countable number of complex values at the discrete coordinate points of the digitizing grid (see chapter 2), Fourier Transformation becomes FFT, and the number of the transverse xy-eigenstates becomes limited and countable. Therefore, we can express the behavior of the cavity with reflection and transmission matrices.

3.3.1. Calculating the 4f-cavity reflection matrix from the single-round-trip transmission matrix

Let $\underline{\underline{\tilde{T}}}$ be the transmission matrix describing a single round-trip in the cavity, including attenuation. This means that $\underline{\underline{\tilde{T}}}$ describes how an xy-light-field is transformed on its way from the inside surface of the left mirror when it takes a single round-trip through the whole cavity, i.e., when it propagates from left to right through both lenses and a potential absorber, gets reflected at the right mirror, and then propagates from right to left through both lenses and the absorber again until it hits the inside surface of the left mirror.

To derive this equation we can take the concept from the derivation of equation (3.17); only this time we use matrix notation. The scalar reflection coefficient r_{cav} becomes a reflection matrix $\underline{\underline{R}}_{cav}$, and instead of just propagating with e^{ikz} the whole (more complex) single-round-trip propagation (including the propagation through two lenses and the attenuation) is represented by the transmission matrix $\underline{\underline{\tilde{T}}}$, which we shall eventually determine by computer simulation.

The total reflection matrix $\underline{\underline{R}}_{cav}$ is obviously the sum of the following components:

- The fraction of light that gets directly reflected at the left mirror: $r_1\mathbb{1}$
- The fraction of light that gets transmitted through the left mirror, takes a single round-trip, and then leaves the cavity to the left side after being once more transmitted through the left mirror: $t_1\underline{\underline{\tilde{T}}}t_1$
- The fraction of light that gets transmitted through the left mirror, takes a round-trip, gets reflected at the left mirror, takes another round-trip, and *then* leaves the cavity to the left side after being once more transmitted through the left mirror: $t_1\underline{\underline{\tilde{T}}}r_1\underline{\underline{\tilde{T}}}t_1$
- etc.

With this approach it is easy to see that we get the following geometric series:

$$\begin{aligned}
\underline{\underline{\mathrm{R}}}_{cav} &= r_1\mathbb{1} + t_1\underline{\underline{\tilde{\mathrm{T}}}}t_1 + t_1\underline{\underline{\tilde{\mathrm{T}}}}r_1\underline{\underline{\tilde{\mathrm{T}}}}t_1 + t_1\underline{\underline{\tilde{\mathrm{T}}}}r_1\underline{\underline{\tilde{\mathrm{T}}}}r_1\underline{\underline{\tilde{\mathrm{T}}}}t_1 + \dots \\
\underline{\underline{\mathrm{R}}}_{cav} &= r_1\mathbb{1} + t_1^2\underline{\underline{\tilde{\mathrm{T}}}} + t_1^2r_1\underline{\underline{\tilde{\mathrm{T}}}}^2 + t_1^2r_1^2\underline{\underline{\tilde{\mathrm{T}}}}^3 + \dots \\
\underline{\underline{\mathrm{R}}}_{cav} &= r_1\mathbb{1} + t_1^2\underline{\underline{\tilde{T}}}\left(\mathbb{1} + r_1\underline{\underline{\tilde{\mathrm{T}}}} + r_1^2\underline{\underline{\tilde{\mathrm{T}}}}^2 + \dots\right) \\
\underline{\underline{\mathrm{R}}}_{cav} &= r_1\mathbb{1} + t_1^2\underline{\underline{\tilde{\mathrm{T}}}}\sum_{n=0}^{\infty}\left(r_1\underline{\underline{\tilde{\mathrm{T}}}}\right)^n \\
\underline{\underline{\mathrm{R}}}_{cav} &= r_1\mathbb{1} + t_1^2\underline{\underline{\tilde{\mathrm{T}}}}\left(\mathbb{1} - r_1\underline{\underline{\tilde{\mathrm{T}}}}\right)^{-1}
\end{aligned}$$

Using the relation $t_1 = 1 + r_1$ (see equation (9) in appendix A.6) this finally becomes

$$\underline{\underline{\mathrm{R}}}_{cav} = r_1\mathbb{1} + (1 + r_1)^2\,\underline{\underline{\tilde{\mathrm{T}}}}\left(\mathbb{1} - r_1\underline{\underline{\tilde{\mathrm{T}}}}\right)^{-1} \tag{3.39}$$

Considering relation (3.11), $\underline{\underline{\mathrm{R}}}_{cav}$ can be expressed as a function of r_0, the absolute value of the left mirror's reflection coefficient:

$$\begin{aligned}
\underline{\underline{\mathrm{R}}}_{cav} = &-\left(r_0^2 + ir_0\sqrt{1-r_0^2}\right)\mathbb{1} \\
&+\left(1 - r_1r_0^2 - ir_0\sqrt{1-r_0^2}\right)^2\underline{\underline{\tilde{\mathrm{T}}}}\left(\mathbb{1} - \left(r_0^2 + ir_0\sqrt{1-r_0^2}\right)\underline{\underline{\tilde{\mathrm{T}}}}\right)^{-1}
\end{aligned} \tag{3.40}$$

3.3.2. Estimating the single-round-trip transmission matrix for critical absorption

We will eventually calculate the single-round-trip transmission matrix $\underline{\underline{\tilde{T}}}$ with a computer simulation, but given the structure of the 4f-cavity we can estimate that $\underline{\underline{\tilde{T}}}$ has diagonal shape, and the matrix $\underline{\underline{\tilde{T}_c}}$ at critical coupling can be approximated by

$$
\begin{aligned}
\underline{\underline{\tilde{T}_c}} &\approx \mathbb{1} r_2 e^{2i\tilde{k}_c l} && \Big| r_2 = e^{i\pi} = -1 \\
\underline{\underline{\tilde{T}_c}} &\approx -\mathbb{1} e^{2i\tilde{k}_c l} && \Big| \tilde{k}_c = k_c + i\kappa_c \\
\underline{\underline{\tilde{T}_c}} &\approx -\mathbb{1} e^{2i(k_c + i\kappa_c)l} \\
\underline{\underline{\tilde{T}_c}} &\approx -\mathbb{1} e^{2ik_c l} e^{-2\kappa_c l} \overset{(3.21)}{\Longrightarrow} \\
\underline{\underline{\tilde{T}_c}} &\approx -\mathbb{1} e^{2ik_c l} r_0 \\
\underline{\underline{\tilde{T}_c}} &\approx r_0 \underline{\underline{T_c}} \quad \text{with } \underline{\underline{T_c}} \overset{\text{def}}{=} -\mathbb{1} e^{2ik_c l}
\end{aligned}
\tag{3.41}
$$

The transmission matrix $\underline{\underline{T_c}}$ in expression (3.41) describes a single round-trip in the cavity at critical coupling without any attenuation. Attenuation can then be added by simply multiplying the scalar value r_0 to the matrix.

With the help of equation (3.19), it is possible to (approximately) express the matrix $\underline{\underline{T_c}}$ from equation (3.41) as a function of r_0. The whole derivation can be found in appendix A.15.

$$
\begin{aligned}
\underline{\underline{T_c}}(r_0) &\approx -\mathbb{1} e^{2ik_c l} \overset{(3.19)}{\Longrightarrow} \\
\underline{\underline{T_c}}(r_0) &\approx \mathbb{1} \left(-r_0 + i\sqrt{1 - r_0^2} \right)
\end{aligned}
\tag{3.42}
$$

3.3.3. Software implementation

The following subroutines have been implemented:

Create transmission matrix for a single round-trip through a 4f-cavity without attenuation

- **Function:** The function `transmission_matrix_round_trip_no_atten(gpu, ax_small, ax_large, TF, modes, lens_mask, f, lambda)` creates all transverse (x,y)-modes encoded in the `modes` input vector and sends these modes on a single round-trip through an unattenuated 4f-cavity with the given parameters. Based on this, a transmission matrix – corresponding to $\underline{\underline{T}}_c$ in equation (3.41) – is created and returned. The n-th column of the returned transmission matrix represents the spatial frequency response to the n-th mode according to the `modes` input-vector.
- **Documentation:** appendix C.21.

Main program for calculating the reflection matrix

- **Description:** The main program listed in appendix E.1 first calculates a critical wavenumber k_c based on equation (3.19), so that the corresponding critical wavelength λ_c is as close as possible to the given reference-wavelength λ_0. Then it uses the function described above to create a single-round-trip transmission matrix $\underline{\underline{T}}_c$ for a 4f-cavity without attenuation based on equation (3.41). Using equation (3.41), the transmission matrix $\underline{\underline{\tilde{T}}}_c$ is calculated. Eventually, the cavity's total reflection matrix $\underline{\underline{R}}_{cav}$ is calculated based on equations (3.11) and (3.39). In the remaining lines of the code, the unitarity of $\underline{\underline{T}}_c$ is checked by calculating $\underline{\underline{T_c T_c}^\dagger}$, and displaying the largest off-diagonal absolute value of $\underline{\underline{T_c T_c}^\dagger}$. Also, the top-left diagonal entry of $\underline{\underline{T}}_c$ is compared with the expected value based on equation (3.42). Finally, $\underline{\underline{R}}_{cav}$ (with element-wisely squared absolute values) and $\underline{\underline{T}}_c$ are visualized graphically .
- **Documentation:** appendix E.1.

3.3.4. Simulation results and plausibility check

The main program described in the previous section gives the following text output:

```
N1=210 N2=418 n_max=52
max T*T' off-diagonal value:    0.0001988
estimated diagonal value of T: -0.89443+0.44721i
actual T(1,1) value:           -0.89458+0.44678i
```

The first line shows the used optimal grid size 210×210, and the guarding grid size 418×418 allows a maximum transverse mode number $n_x = n_y = 52$. Therefore, the chosen maximum mode number of `16` (see line 50 in the listing in appendix E.1) is well within borders.

The second line of the output shows the maximal non-diagonal absolute value of $\underline{\underline{\mathrm{T}}}\,\underline{\underline{\mathrm{T}}}^\dagger$. If the matrix $\underline{\underline{\mathrm{T}}}$ was exactly unitary, this value would be zero. That it is close to zero indicates that $\underline{\underline{\mathrm{T}}}$ is at least close to unitarity.

The last two lines reveal good agreement between the estimated diagonal value of the $\underline{\underline{\mathrm{T}}}$-matrix, calculated with equation (3.42), and the simulation result (at least in the topmost left array entry).

The generated images of the transmission matrix and the element-wisely squared reflection matrix can be seen on the next two pages in figures 3.16 and 3.17. The transmission matrix is in very good approximation a diagonal matrix, as expected. The (element-wisely squared) reflection matrix at critical coupling shows that most of the reflection almost vanishes completely for all modes up to $n_x = n_y = 16$ (with higher modes showing a bit more reflection than lower modes).

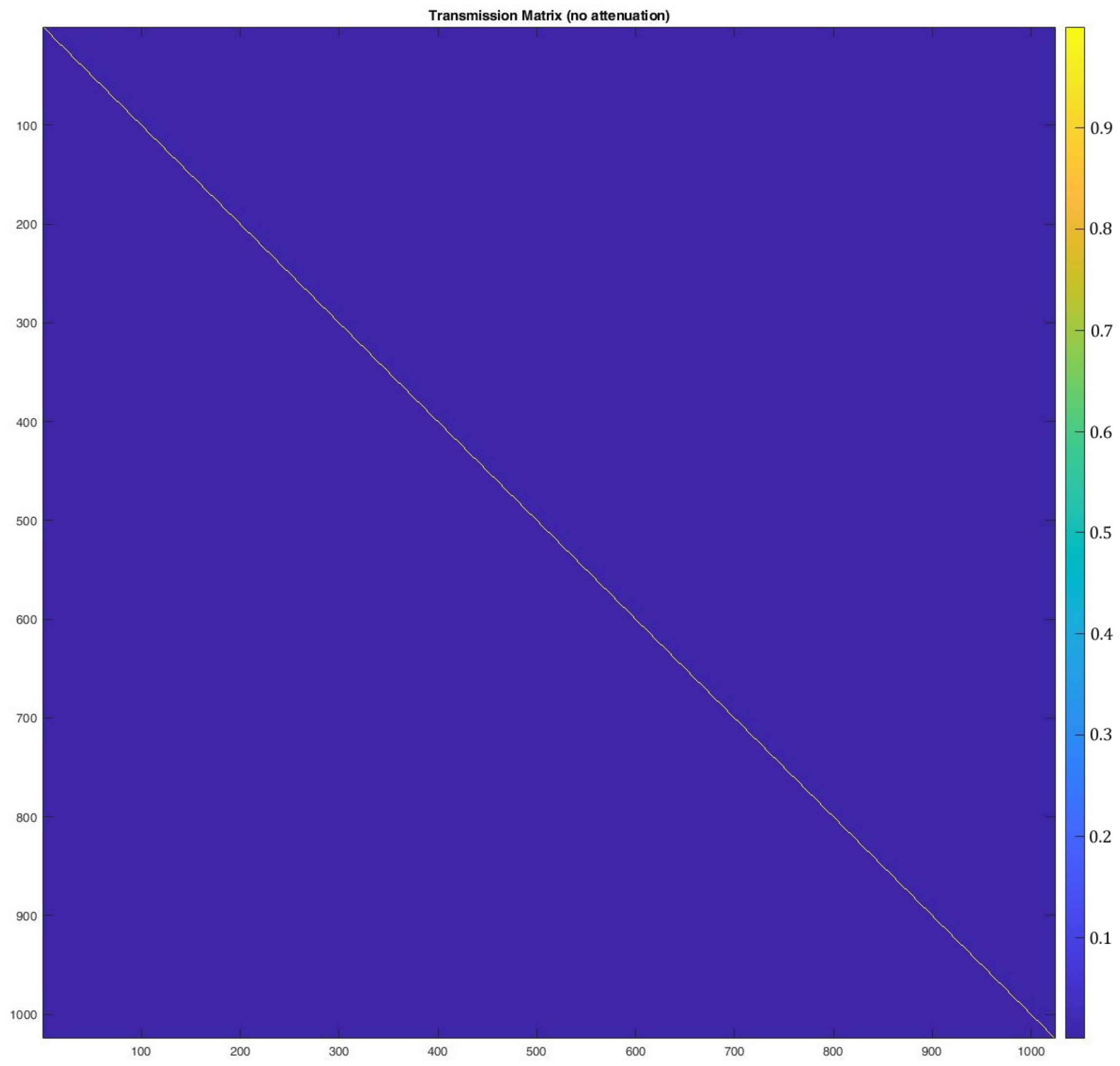

Figure 3.16.: The transmission matrix of a round-trip through a 4f-cavity without any attenuation (the plot shows absolute values). Parameters: $f = 75\,\mathrm{mm}$, $\lambda_0 = 800\,\mathrm{nm}$. Observation plane $2.5\,\mathrm{mm} \times 2.5\,\mathrm{mm}$ covered by a 210×210 grid. Maximum modes: $n_y = n_y = 16$. Lenses: aspherical perfect lenses. Propagation simulation: Rayleigh-Sommerfeld.

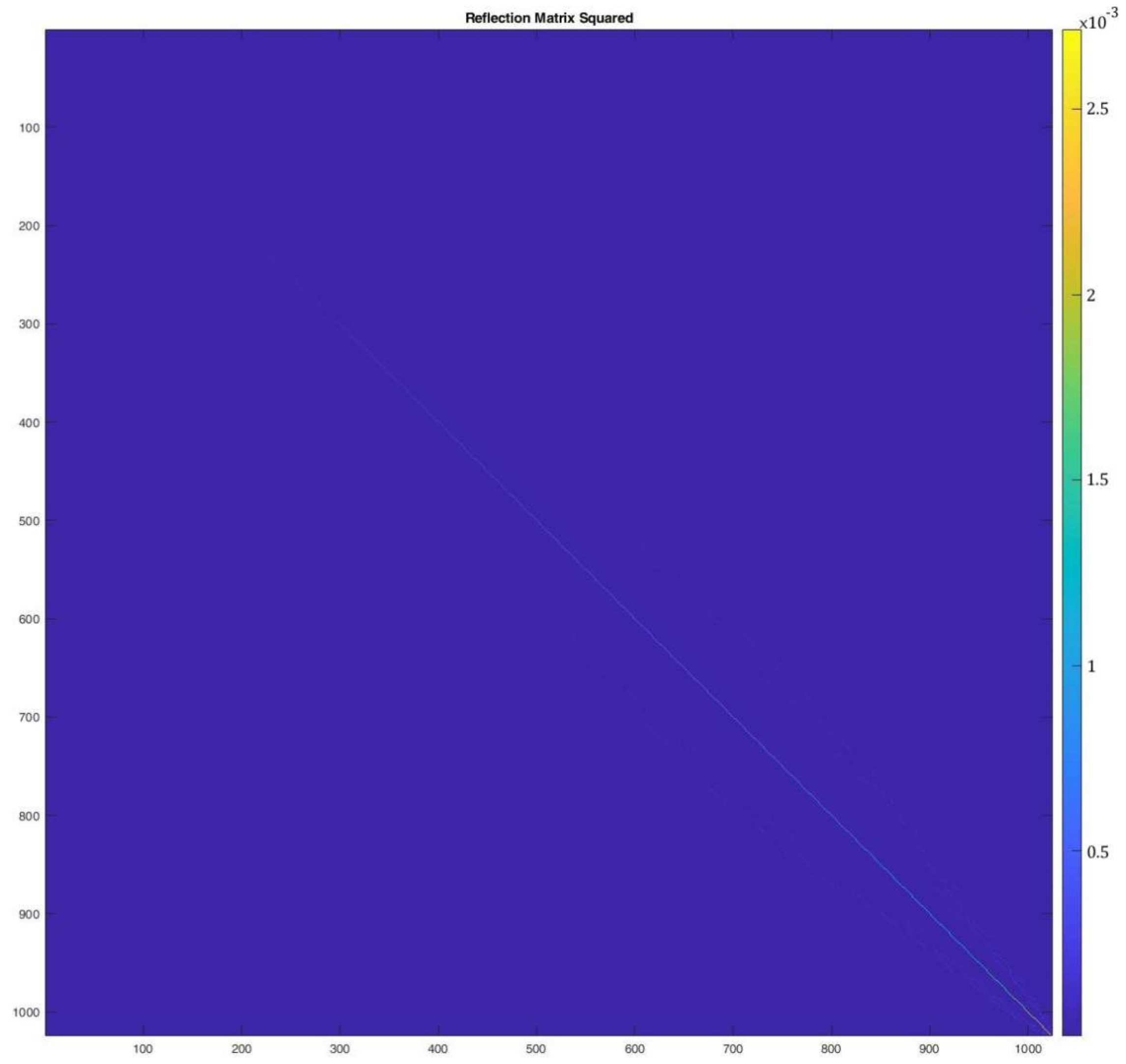

Figure 3.17.: The reflection matrix of a 4f-cavity-CPA at critical wavelength and attenuation (the plot shows squared absolute values). Parameters: $f = 75\,\mathrm{mm}$, $\lambda_0 = 800\,\mathrm{nm}$. Observation plane $2.5\,\mathrm{mm} \times 2.5\,\mathrm{mm}$. Maximum modes: $n_x = n_y = 16$. Lenses: aspherical perfect lenses. Propagation simulation: Rayleigh-Sommerfeld.

Changing the propagation method to the Fresnel transfer function by modifying line 29 to `TF = 1` makes the reflection matrix noisier, as can be seen in figure 3.18.

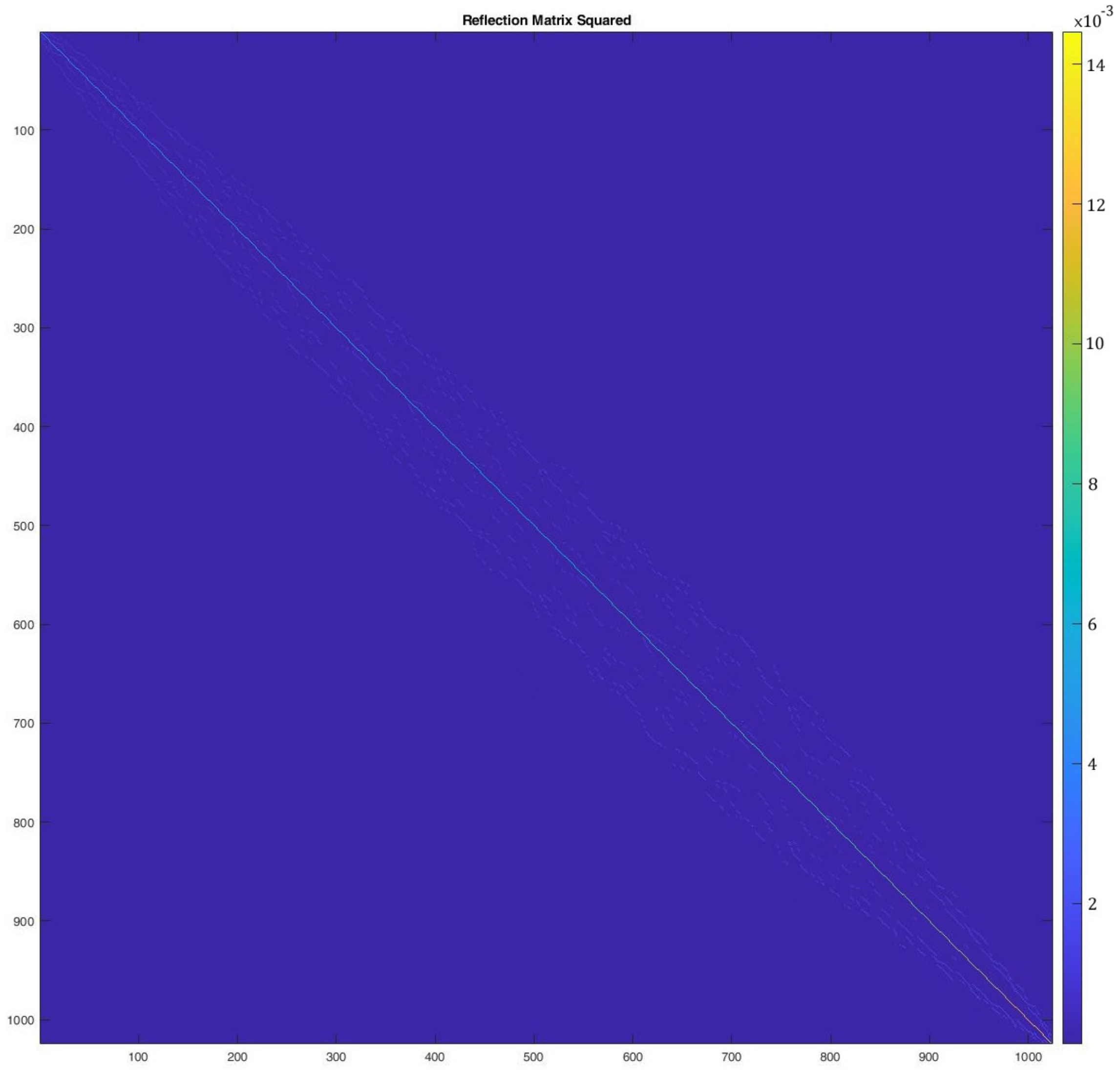

Figure 3.18.: The reflection matrix of a 4f-cavity-CPA at critical wavelength and attenuation (the plot shows squared absolute values). Parameters: $f = 75\,\mathrm{mm}$, $\lambda_0 = 800\,\mathrm{nm}$. Observation plane $2.5\,\mathrm{mm} \times 2.5\,\mathrm{mm}$. Maximum modes: $n_x = n_y = 16$. Lenses: aspherical perfect lenses. Propagation simulation: Fresnel.

However, using the Fresnel transfer function and also switching to spherical lenses by modifying line 24 to `lens_type == 1` produces a strictly diagonal reflection matrix with a maximal reflectance that is 5 orders of magnitude smaller, and also does not show significant differences between “lower” and “higher” modes, as can be seen in figure 3.19 on the next page.

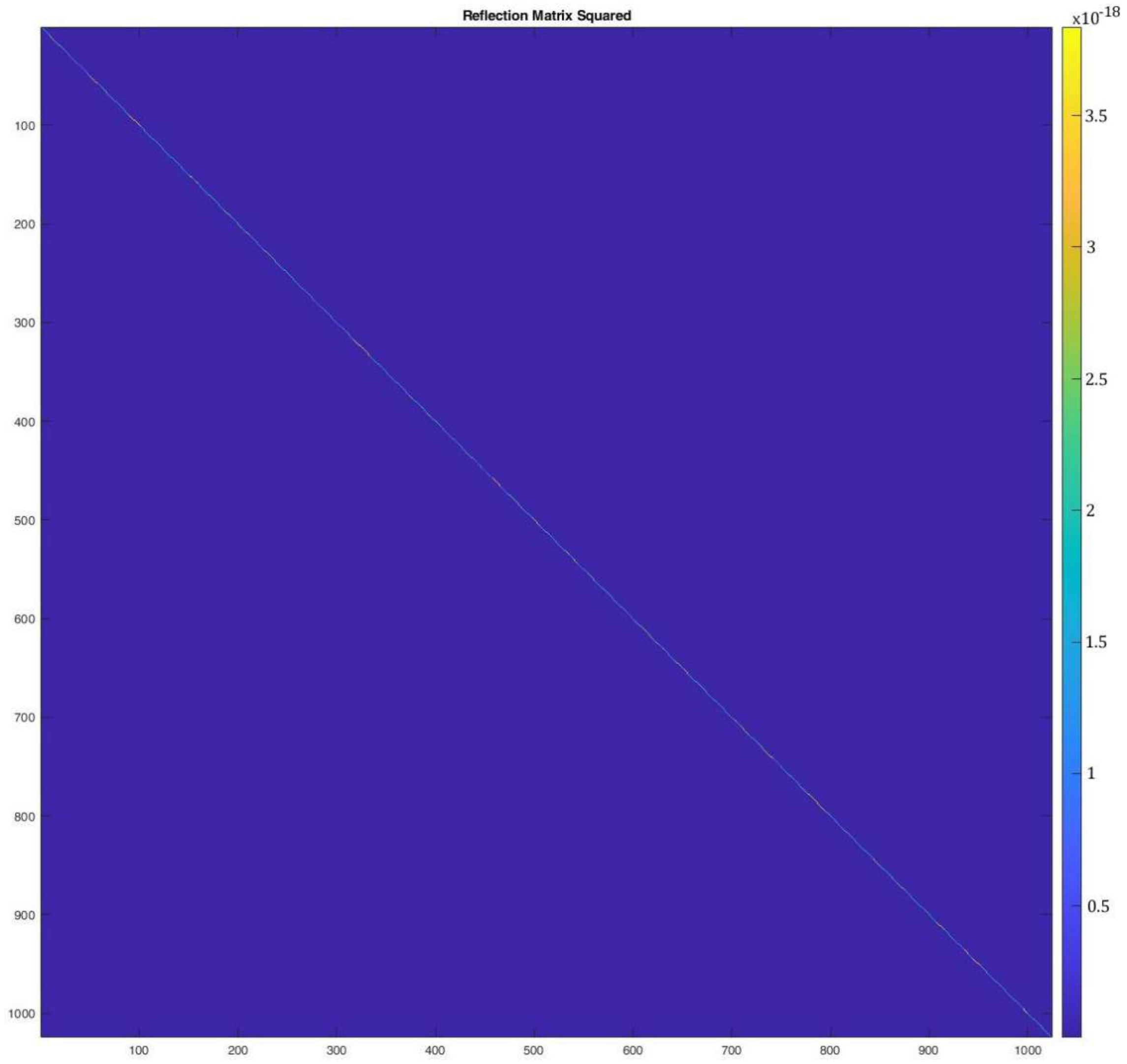

Figure 3.19.: The reflection matrix of a 4f-cavity-CPA at critical wavelength and attenuation (the plot shows squared absolute values). Parameters: $f = 75\,\mathrm{mm}$, $\lambda_0 = 800\,\mathrm{nm}$. Observation plane $2.5\,\mathrm{mm} \times 2.5\,\mathrm{mm}$. Maximum modes: $n_x = n_y = 16$. Lenses: spherical. Propagation simulation: Fresnel.

We understand this behavior to be an effect of the paraxial Fresnel-simplification. The Fresnel propagation simulation with its paraxial approximation is “blind” to the spherical aberration of the spherical lenses, and also seemingly simplifies the simulation so that it rather resembles a light-ray-simulation.

4. Implementation and simulation results

4.1. Effects of propagation simulation method and lens geometry

4.1.1. Software Implementation

The main program `CPA_sim_002_r_curve`, which is documented in appendix E.2, iterates through various wavelengths around a critical wavelength and generates a reflection matrix $\underline{\underline{R}}_{cav}$ for each of these wavelengths. The minimum, average, and maximum eigenvalue of each of these reflection matrices is stored into a file (together with the corresponding wavenumber and wavelength).

Besides other parameters, the program allows to choose the following:

- **Iterator:** Rayleigh-Sommerfeld or Fresnel
- **Lenses:** spherical lenses or perfect, aspherical lenses
- **Attenuation factor:** factor determining how much attenuation is simulated. If this factor is 1, then critical attenuation is simulated; for values smaller or larger than one, under-critical or over-critical attenuation is simulated.

4.1.2. Simulation Results

Rayleigh-Sommerfeld propagator

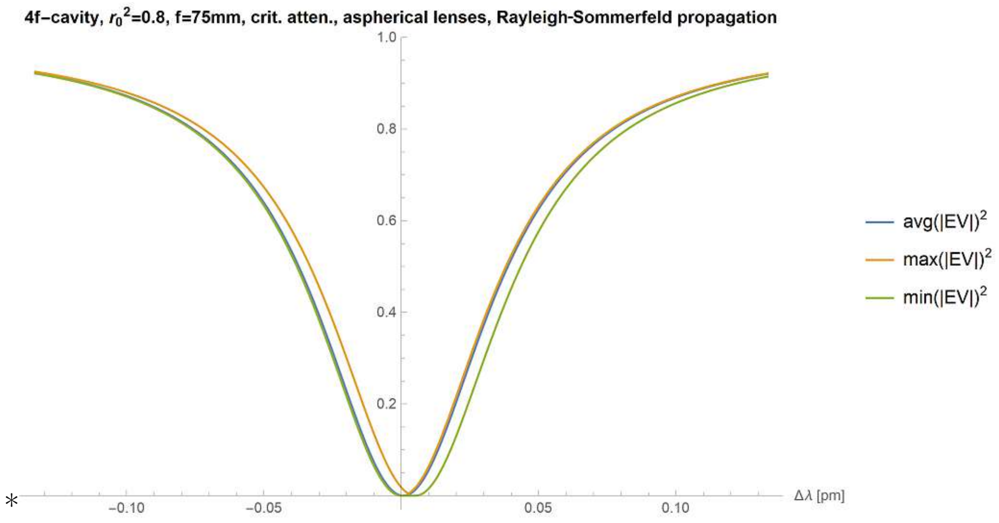

Figure 4.1.: The average, maximum, and minimum eigenvalues of the reflection matrix of a 4f-cavity-CPA at critical attenuation around the critical wavelength (the plot shows squared absolute values). Parameters: $r_0^2 = 0.8, f = 75\,\mathrm{mm},\ \lambda_0 = 800\,\mathrm{nm}$. Maximum modes: $n_x = n_y = 16$. **Lenses: aspherical.** Propagation simulation: Rayleigh-Sommerfeld.

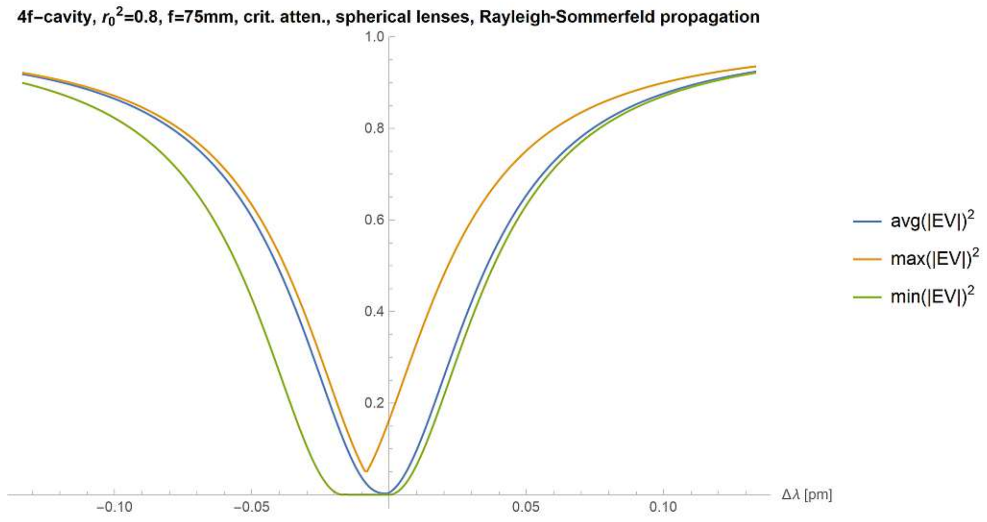

Figure 4.2.: The average, maximum, and minimum eigenvalues of the reflection matrix of a 4f-cavity-CPA at critical attenuation around the critical wavelength (the plot shows squared absolute values). Parameters: $r_0^2 = 0.8, f = 75\,\mathrm{mm},\ \lambda_0 = 800\,\mathrm{nm}$. Maximum modes: $n_x = n_y = 16$. **Lenses: spherical.** Propagation simulation: Rayleigh-Sommerfeld.

Fresnel propagator

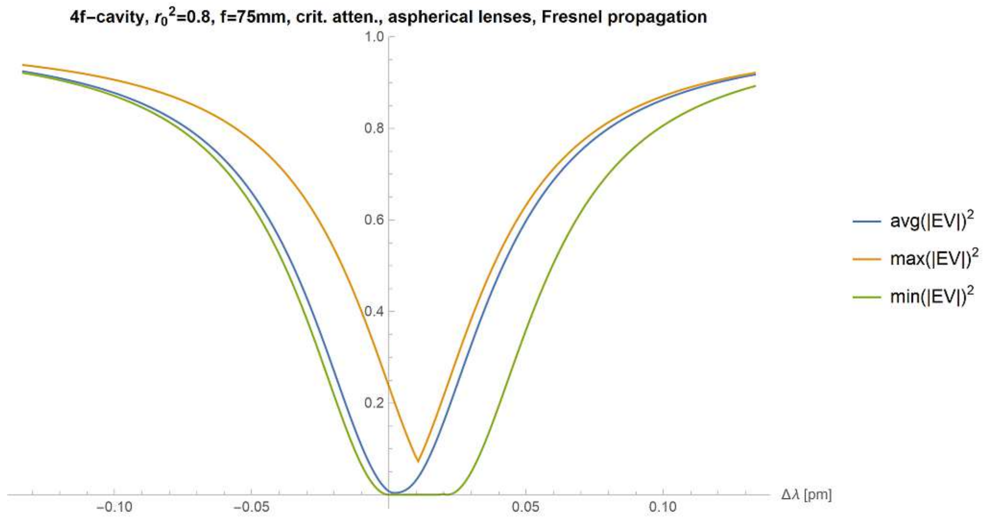

Figure 4.3.: The average, maximum, and minimum eigenvalues of the reflection matrix of a 4f-cavity-CPA at critical attenuation around the critical wavelength (the plot shows squared absolute values). Parameters: $r_0^2 = 0.8, f = 75\,\text{mm}, \lambda_0 = 800\,\text{nm}$. Maximum modes: $n_x = n_y = 16$. **Lenses: aspherical.** Propagation simulation: Fresnel.

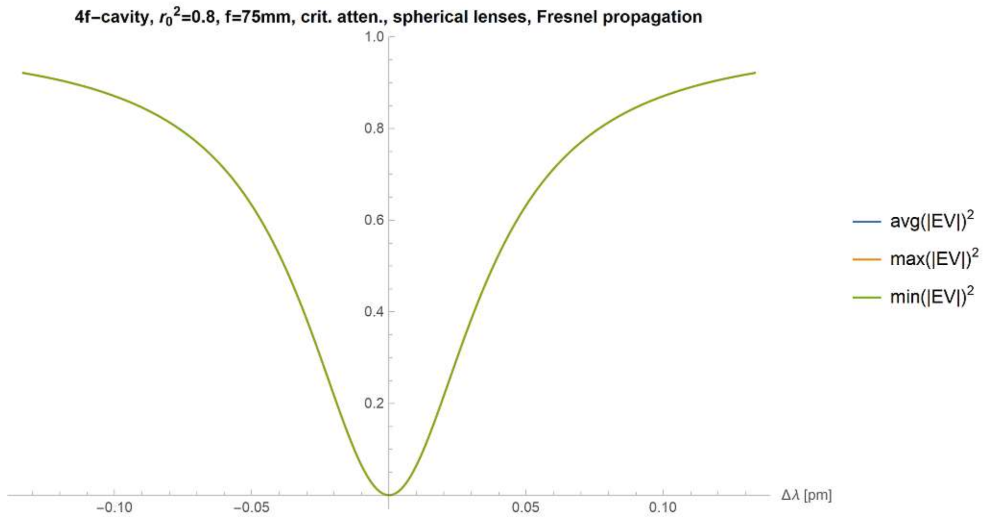

Figure 4.4.: The average, maximum, and minimum eigenvalues of the reflection matrix of a 4f-cavity-CPA at critical attenuation around the critical wavelength (the plot shows squared absolute values). Parameters: $r_0^2 = 0.8, f = 75\,\text{mm}, \lambda_0 = 800\,\text{nm}$. Maximum modes: $n_x = n_y = 16$. **Lenses: spherical.** Propagation simulation: Fresnel.

4.1.3. Discussion

Using the **Rayleigh-Sommerfeld propagator** in the simulation, a 4×75 mm, $r_0^2 = 0.8$ cavity with **aspherical, perfect lenses** becomes a CPA at critical attenuation with very good fidelity. The average reflectance over all eigenmodes practically vanishes at the resonance point, and also the maximal reflective eigenmode dips to almost zero, although at a slightly shifted wavelength (see figure 4.1).

Replacing the aspherical by **spherical lenses** leads to an inferior result (see figure 4.2). This makes sense because it is to be expected that the spherical aberration reduces the 4f-effect for the part of the field that is more distant to the optical axis.

On the other hand, using the **Fresnel propagator** to simulate an attenuated 4f-cavity with **aspherical, perfect lenses** also leads to inferior results (see figure 4.3), whereas using **spherical lenses** seemingly results in an almost perfect CPA where average, minimum, and maximum squared eigenvalues of the reflection matrix are in perfect agreement (see figure 4.4). This result is to be taken with caution, though, as it seems to be an effect of the paraxial Fresnel approximation canceling out the lenses' spherical aberration.

4.2. Eigenmode decomposition of reflection matrix

4.2.1. Software implementation

The results presented in this section were produced with the program from appendix E.3. It generates the reflection-matrix of an attenuated 4f-cavity, then calculates the corresponding eigenvectors and eigenvalues, and finally writes the eigenvalues in ascending order into an `.xlsx`-file. Optionally, the corresponding eigenmodes (encoded in the eigenvectors) can also be saved as images, and/or it creates a video where the eigenmodes in ascending order are visualized in an animation.

4.2.2. Simulation Results

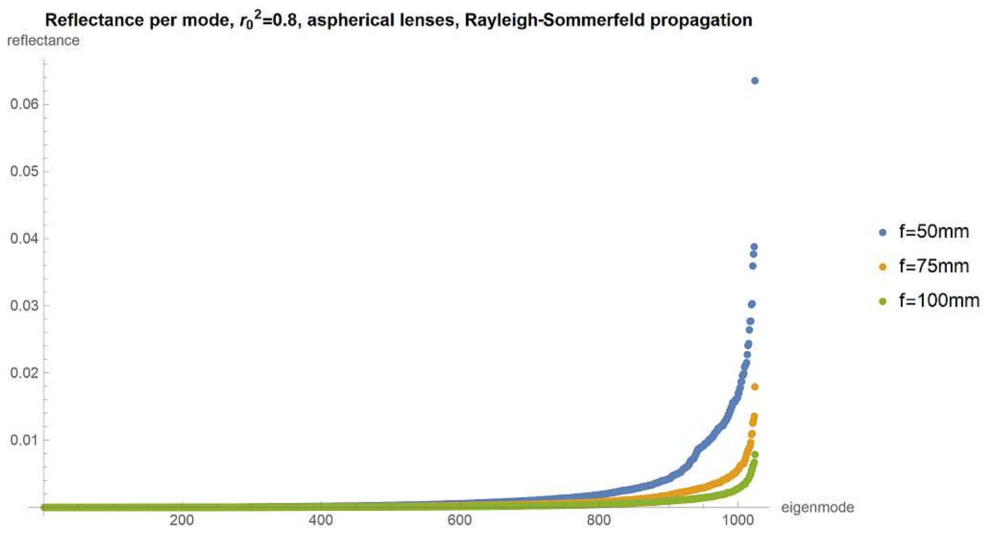

Figure 4.5.: 4f-cavity-CPA at critical attenuation: squared absolute eigenvalues of the reflection matrix plotted in ascending order. Parameters: $r_0^2 = 0.8$, $\lambda_0 = 800\,\mathrm{nm}$. Observation plane $2.5\,\mathrm{mm} \times 2.5\,\mathrm{mm}$. Maximum modes: $n_x = n_y = 16$. Lenses: **aspherical**. Propagation simulation: **Rayleigh-Sommerfeld**.

Figure 4.5 shows the squared absolute eigenvalues of the reflection matrix of attenuated 4f-cavities with different focal lengths in ascending order. The transverse n_x, n_y mode numbers have been limited with 16, which results in a total of 1024 transverse modes. The simulation shows that for all chosen cavity lengths, a majority of the eigenmodes of the reflection matrix show a strong CPA effect. However, the shorter the cavity gets, the less pronounced the CPA effect becomes for some of the modes. While with a $4 \times 100\,\mathrm{mm}$ cavity up to around 1000 of the 1024 modes show a reflectance below 0.002, this is only the case for approximately 800 modes with a $4 \times 50\,\mathrm{mm}$ cavity.

This effect does *not* appear when the Fresnel propagator and spherical lenses are used instead of the Rayleigh-Sommerfeld propagator and aspherical lenses. In this case, the reflectance of all eigenmodes is computed to be below 10^{-17} when using double precision GPU arithmetic. However, even that turns out to be a numerical artifact. When running the same simulation with, e.g., 34 decimal digits precision[2], this value drops to below 10^{-55}.

[2] This is accomplished by initializing `gpu = 2` in the source-code, which re-routes all calculations to the Advanpix Multiprecision Computing Toolbox from `www.advanpix.com`.

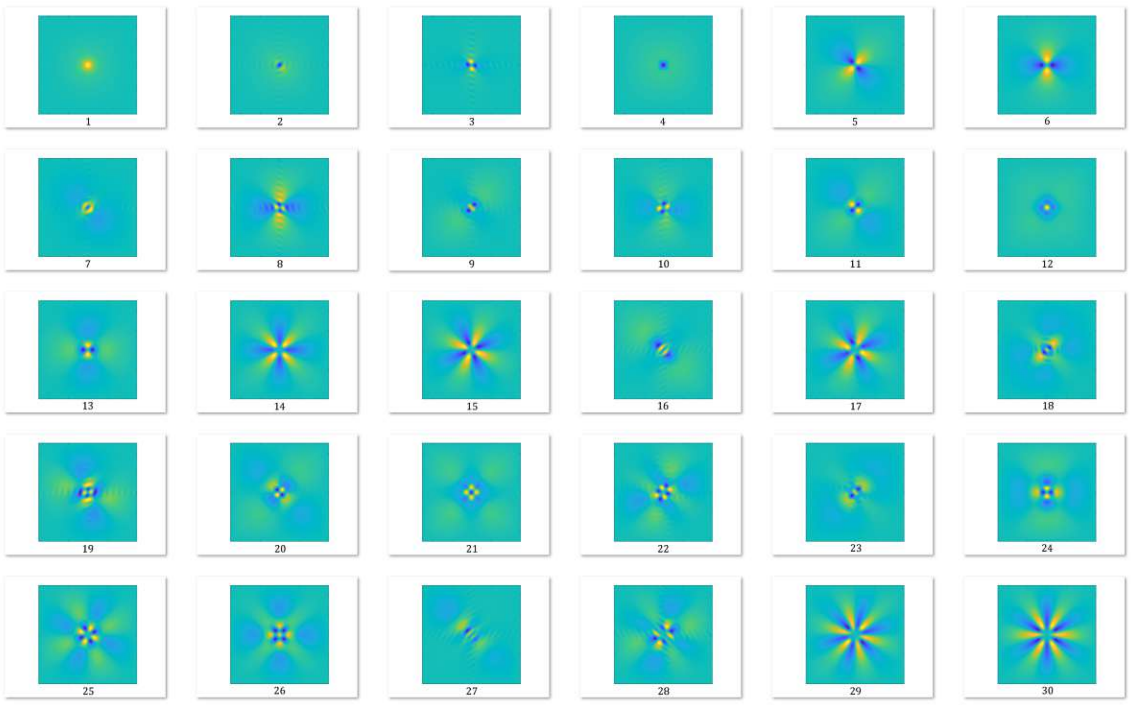

Figure 4.6.: The 30 least reflective eigenmodes of the reflection matrix of a $4 \times 75\,\mathrm{mm}$ 4f-cavity-CPA at critical attenuation, ordered by increasing reflectivity. Parameters: $r_0^2 = 0.8$, $\lambda_0 = 800\,\mathrm{nm}$. Observation plane $2.5\,\mathrm{mm} \times 2.5\,\mathrm{mm}$. Maximum modes: $n_x = n_y = 16$. Lenses: **aspherical**. Propagation simulation: **Rayleigh-Sommerfeld**.

Figure 4.6 shows the 30 least reflective ("best absorbed") eigenmodes of the reflection matrix of a $4 \times 75\,\mathrm{mm}$ 4f-cavity-CPA. They correspond to the leftmost modes in figure 4.5. It can be seen that the CPA-effect is strongest for centrally symmetrical eigenmodes that are concentrated in or around the center region of the xy-plane.

In comparison, figure 4.7 on the following page shows the 30 most reflective ("worst absorbed") eigenmodes of the 4f-cavity, which correspond to the rightmost modes in figure 4.5. These modes are preferably concentrated on the outer border of the observation plane.

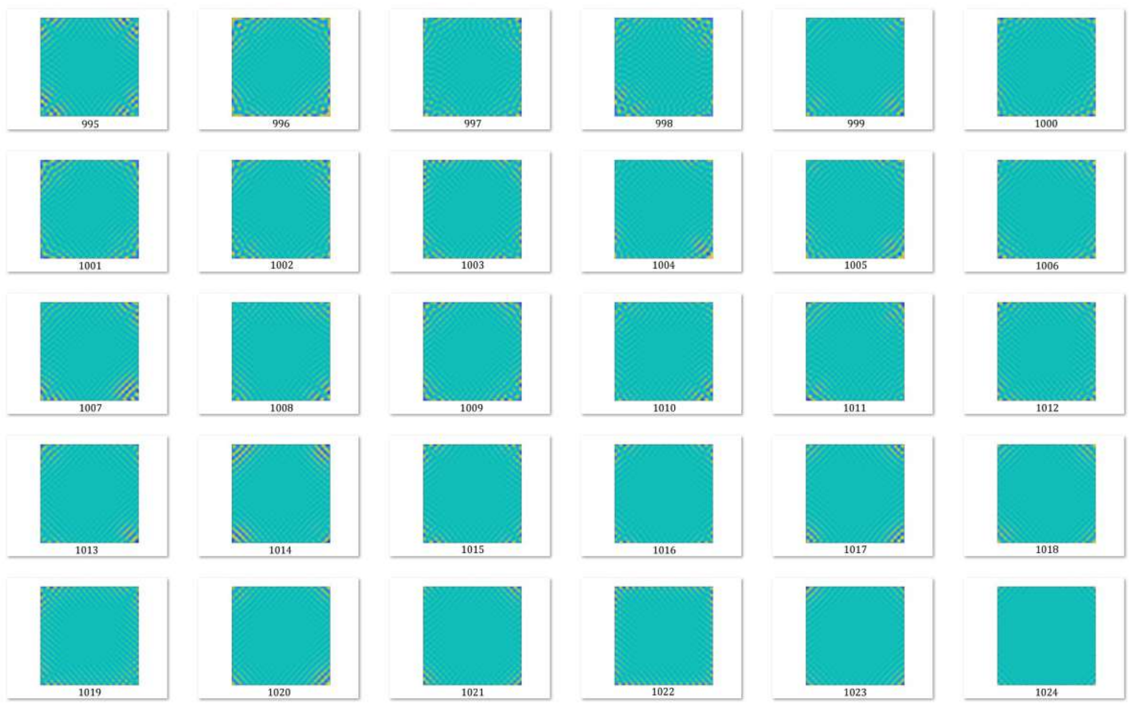

Figure 4.7.: The 30 most reflective eigenmodes of the reflection matrix of a $4 \times 75\,\mathrm{mm}$ 4f-cavity-CPA at critical attenuation, ordered by increasing reflectivity. Parameters: $r_0^2 = 0.8$, $\lambda_0 = 800\,\mathrm{nm}$. Observation plane $2.5\,\mathrm{mm} \times 2.5\,\mathrm{mm}$. Maximum modes: $n_x = n_y = 16$. Lenses: **aspherical**. Propagation simulation: **Rayleigh-Sommerfeld**.

4.3. Exploring the parameter-space for optimization potential

In this chapter we explore the potential for improving the CPA-behavior beyond the performance depicted in figure 4.5. To do so, we assume for the time being that it would be possible to adjust certain parameters like, for example, attenuation or cavity length individually for each of the eigenmodes in figure 4.5. These considerations do not yet have a specific technical implementation in mind, but could possibly lead to an improved design.

4.3.1. Software implementation

The following subroutines have been additionally implemented to support some of the simulations in this section:

Simulation of a single round-trip through a 4f-cavity without attenuation where the positions of the lenses and the back-mirror can deviate from the optimal f-2f-f-positions

- **Function:** `round_trip_no_atten2(gpu, E_in, ax_small, ax_large, TF, lens_mask, d1, d2, d3, lambda, f_space_in, f_space_out)`
- **Description:** This function is an enhanced version of the function `round_trip_no_atten` as described in appendix C.18. It allows to separately define the distance between the partially reflective mirror and the first lens, between the first lens and the second lens, and between the second lens and the total reflective, perfect back-mirror.
- **Documentation:** appendix C.19.

Create transmission matrix for a single round-trip through a 4f-cavity without attenuation where the positions of the lenses and the back-mirror can deviate from the optimal f-2f-f-positions

- **Function:** `transmision_matrix_round_trip_no_atten2(gpu, ax_small, ax_large, TF, modes, lens_mask, d1, d2, d3, lambda)`
- **Description:** This function is an enhanced version of the function `transmision_matrix_round_trip_no_atten` as described in appendix C.21. It allows to separately define the distances between the partially reflective mirror and the first lens, between the first lens and the second lens, and between the second lens and the total reflective, perfect back-mirror.
- **Documentation:** appendix C.22.

4.3.2. Mode-dependent attenuation

We used the program documented in appendix E.4 to study the effect of slightly overcritical and undercritical attenuation on the various eigenmodes of the critically attenuated $4 \times 50\,\text{mm}$ cavity depicted in figure 4.5. It turned out that mode-dependent variation of the attenuation can not significantly improve the CPA performance for any of the 1024 modes.

4.3.3. Mode-dependent focal length (and total length)

Using the program documented in appendix E.5, we slightly varied both the focal length, and consequently also the total length of the 4f-cavity, and researched the effect on the individual eigenmodes of the critically attenuated $4 \times 50\,\mathrm{mm}$ cavity. The results are shown in figure 4.8.

For a 4f-cavity with a mode-independent, fixed focal length $f_0 = 50\,\mathrm{mm}$, eigenmodes of the reflection matrix above mode number $m = 800$ quickly become more and more reflective. However, assuming an *individual* (optimal) focal length $f(m) = f_0 + c(m)$ for each eigenmode, the reflectance can be brought down to almost zero for all modes. The mode-dependent additive correction factor $c(m)$ for minimal reflectance is in the range between $0\,\mathrm{pm}$ (for eigenmode $m = 1 \dots 10$) and $-959\,\mathrm{pm}$ (for eigenmode $m = 1024$).

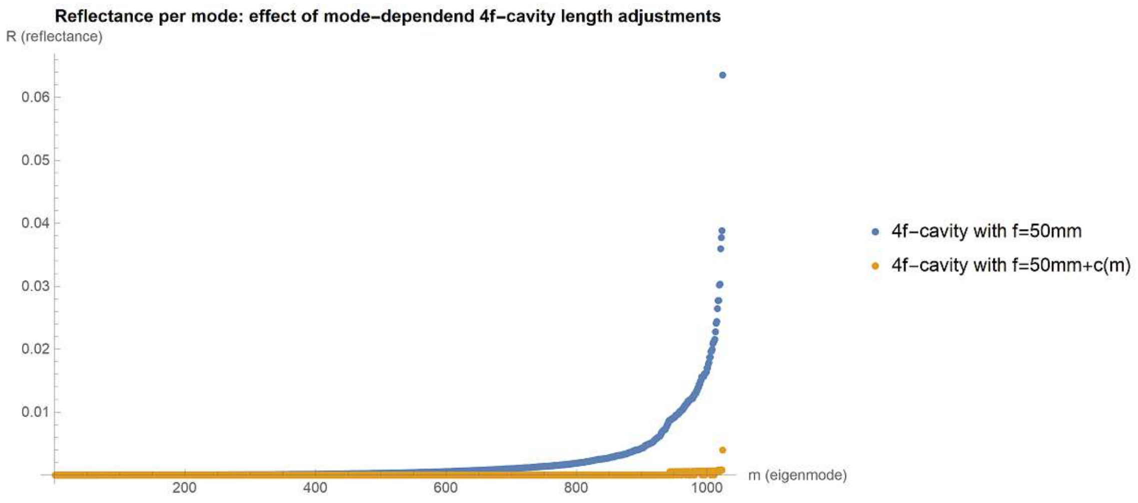

Figure 4.8.: Effect of mode-dependent adjustments of the critically attenuated 4f-cavity's focal length (and consequently also of the total length). Parameters: $f_0 = 50\,\mathrm{mm}, r_0^2 = 0.8$, $\lambda_0 = 800\,\mathrm{nm}$. Observation plane $2.5\,\mathrm{mm} \times 2.5\,\mathrm{mm}$. Maximum modes: $n_x = n_y = 16$. Lenses: **aspherical**. Propagation simulation: **Rayleigh-Sommerfeld**.

Figure 4.9 depicts the mode-dependent, additive compensation factor $c(m)$ required to obtain the optimized result in figure 4.8. That this factor is in the $0 \cdots -1000\,\mathrm{pm}$ range is an indicator to expect the 4f-cavity-CPA to be very sensitive against lens displacements and deviations in the total length.

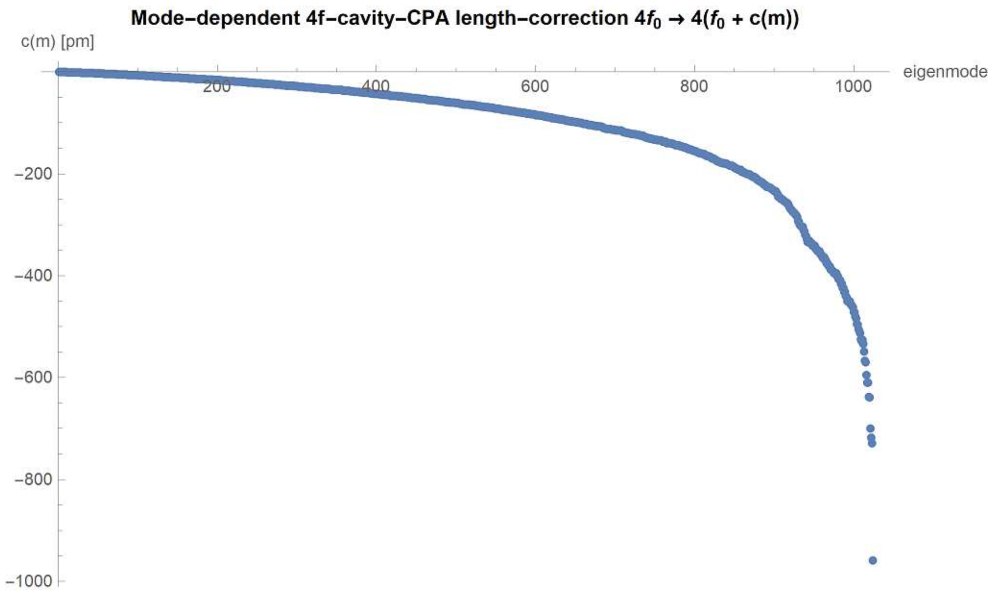

Figure 4.9.: Mode-dependent, additive compensation factor $c(m)$ required to obtain the optimized result in figure 4.8.

4.3.4. Mode-dependent distance between second lens and perfect back-mirror

Using the program documented in appendix E.5, we slightly varied the distance between the second lens and the total reflective, perfect back-mirror, and investigated the effect on the various eigenmodes of the critically attenuated $4 \times 50\,\mathrm{mm}$ cavity.

For a 4f-cavity with a mode-independent, fixed distance $d_3 = f$ between the second lens and the total reflective mirror, the eigenmodes of the reflection matrix above mode number $m = 800$ quickly become more and more reflective (see figure 4.10). However, assuming an *individual* (optimal) distance $d_3(m) = f + c(m)$ for each eigenmode, the reflectance can be brought down to almost zero for all modes. The mode-dependent additive correction factor $c(m)$ for minimal reflection is in the range between $0\,\mathrm{pm}$ (for eigenmode $m = 1 \ldots 3$) and $-3687\,\mathrm{pm}$ (for eigenmode $m = 1024$).

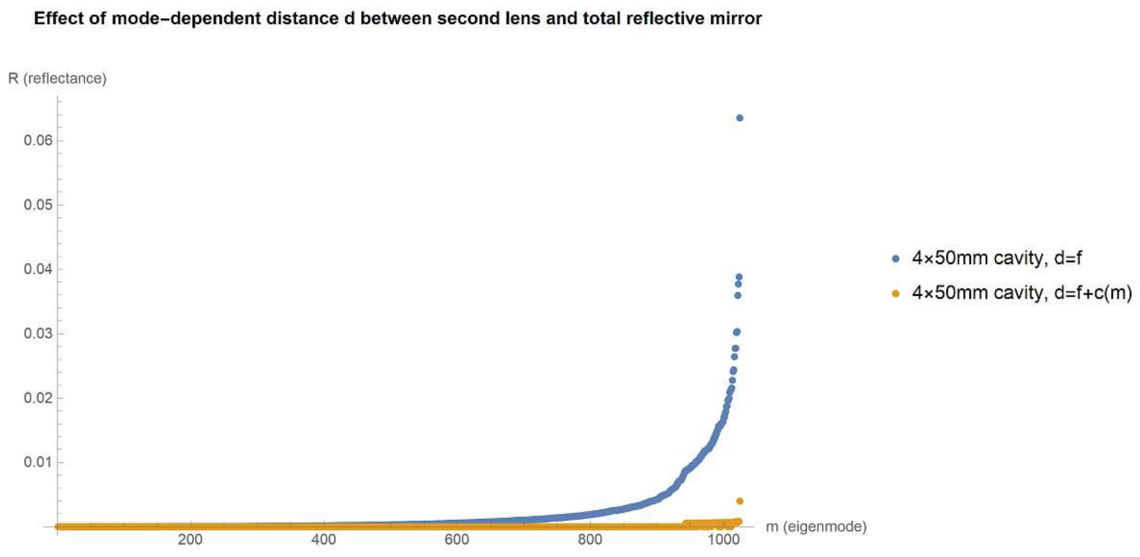

Figure 4.10.: Effect of mode-dependent adjustments of the distance d between the second lens and the total reflective, perfect mirror. Parameters: $f = 50\,\mathrm{mm}, r_0^2 = 0.8$, $\lambda_0 = 800\,\mathrm{nm}$. Observation plane $2.5\,\mathrm{mm} \times 2.5\,\mathrm{mm}$. Maximum modes: $n_x = n_y = 16$. Lenses: **aspherical**. Propagation simulation: **Rayleigh-Sommerfeld**.

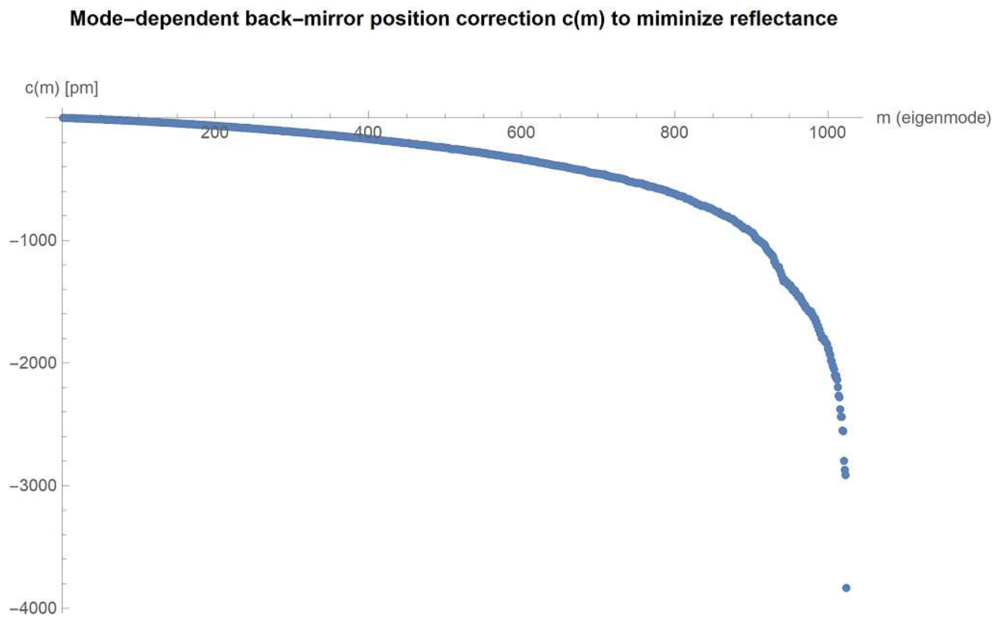

Figure 4.11.: Mode-dependent, additive compensation factor $c(m)$ required to obtain the optimized result in figure 4.10.

4.4. Sensitivity against deviations from optimal parameters

This section evaluates how sensitive an actual experimental implementation of a 4f-cavity-CPA might be against deviations from optimal parameters.

All simulations in this section share the following set of parameter values:

- **base wavelength:** $\lambda_0 = 785\,\mathrm{nm}$
- **focal length:**$f = 50\,\mathrm{mm}$
- **lens type:** perfect, aspherical lens
- **reflectance of partially reflective mirror:** $r_0^2 = 0.8$
- **grid size:** $2.5\,\mathrm{mm} \times 2.5\,\mathrm{mm}$
- **propagation simulation:** Rayleigh-Sommerfeld
- **maximum mode numbers:** $n_x = n_y = 16$
- **number of calculated points:** 900
- **area of interest:** 1.5 periods around resonance point closest to λ_0 (assuming an optimally configured 4f-cavity)

4.4.1. Deviations in the position of the total reflective mirror

Parameters

The simulations in this subsection have been carried out using the program described in appendix E.7 with the following parameter vectors:

```
% parameter variations
rho=[1, 1, 1, 1, 1];
d1 = [f, f, f, f, f];
d2 = [f*two, f*two, f*two, f*two, f*two];
d3 = [f, f+lambda_c/four, f+pval(gpu,'1e-3'), f+pval(gpu,'2e-3'), f+pval(gpu,'5e-3')];
```

These parameter vectors define five simulation rounds, each with critical attenuation and the first and second lens being in the optimal position, but with the total reflective back mirror (mirror M2 in figure 3.15) in a slightly different position in each round, namely:

- exactly one focal length after the second lens
- one focal length plus $\frac{\lambda}{4}$ after the second lens
- one focal length plus $1\,\mathrm{mm}$ after the second lens
- one focal length plus $2\,\mathrm{mm}$ after the second lens
- one focal length plus $5\,\mathrm{mm}$ after the second lens

Simulation Results

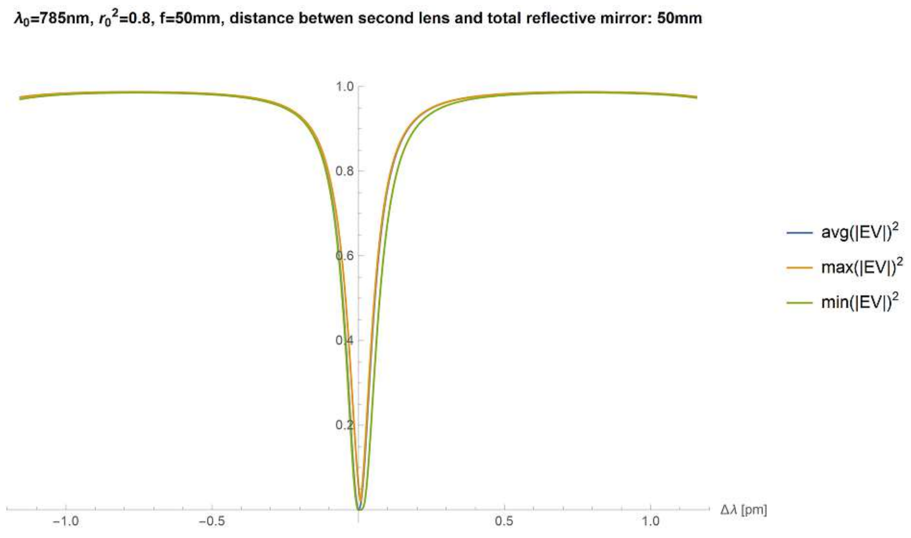

Figure 4.12.: Average over all squared absolute reflection-matrix eigenvalues, compared to largest and smallest squared absolute reflection-matrix eigenvalues of an optimally configured 4f-cavity-CPA, plotted as a function of the wavelength around the resonance point. Parameters: $f = 50\,\text{mm}, r_0^2 = 0.8,\ \lambda_0 = 785\,\text{nm}$.

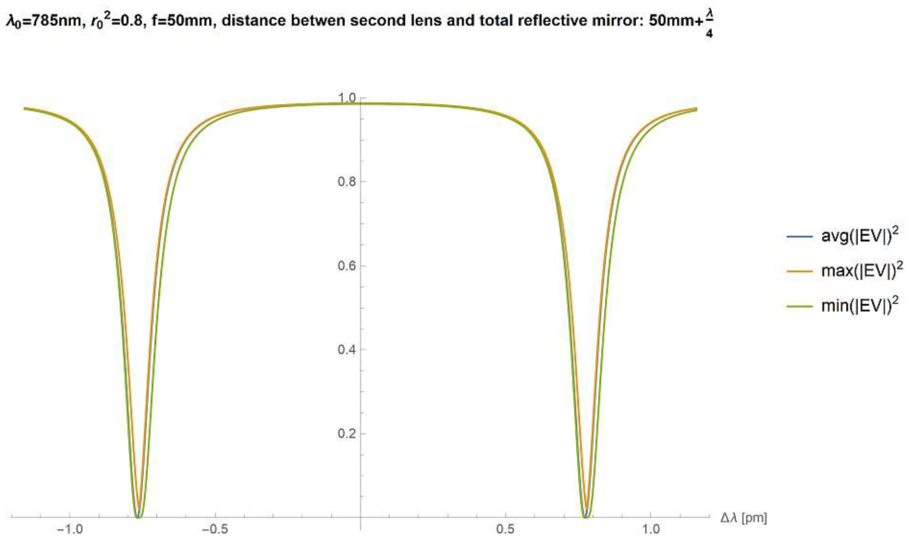

Figure 4.13.: Average over all squared absolute reflection-matrix eigenvalues, compared to the largest and smallest squared absolute reflection-matrix eigenvalues of a 4f-cavity-CPA with the total reflective mirror moved away from the optimal position by $\frac{\lambda_0}{4}$, plotted as a function of the wavelength around the resonance point of an optimally configured 4f-cavity. Parameters: $f = 50\,\text{mm}, r_0^2 = 0.8,\ \lambda_0 = 785\,\text{nm}$.

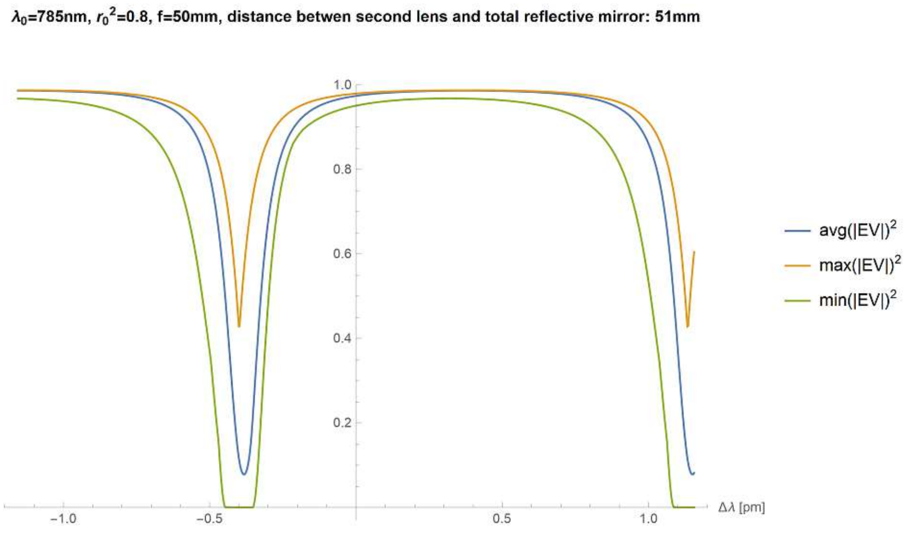

Figure 4.14.: Average over all squared absolute reflection-matrix eigenvalues, compared to the largest and smallest squared absolute reflection-matrix eigenvalues of an 4f-cavity-CPA with the the total reflective mirror moved away from the optimal position by $1\,\mathrm{mm}$, plotted as a function of the wavelength around the resonance point of an optimally configured 4f-cavity. Parameters: $f = 50\,\mathrm{mm}, r_0^2 = 0.8,\ \lambda_0 = 785\,\mathrm{nm}$.

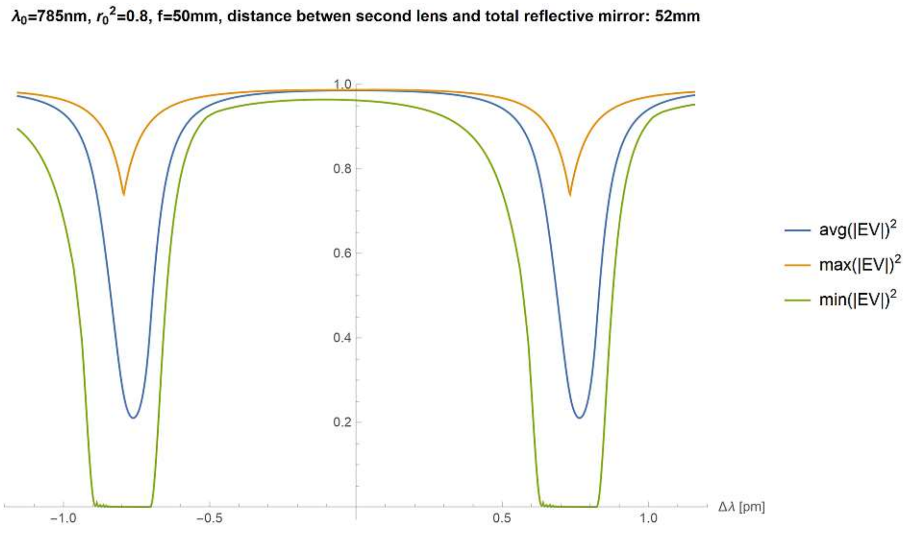

Figure 4.15.: Average over all squared absolute reflection-matrix eigenvalues, compared to the largest and smallest squared absolute reflection-matrix eigenvalues of a 4f-cavity-CPA with the total reflective mirror moved away from the optimal position by $2\,\mathrm{mm}$, plotted as a function of the wavelength around the resonance point of an optimally configured 4f-cavity. Parameters: $f = 50\,\mathrm{mm}, r_0^2 = 0.8,\ \lambda_0 = 785\,\mathrm{nm}$.

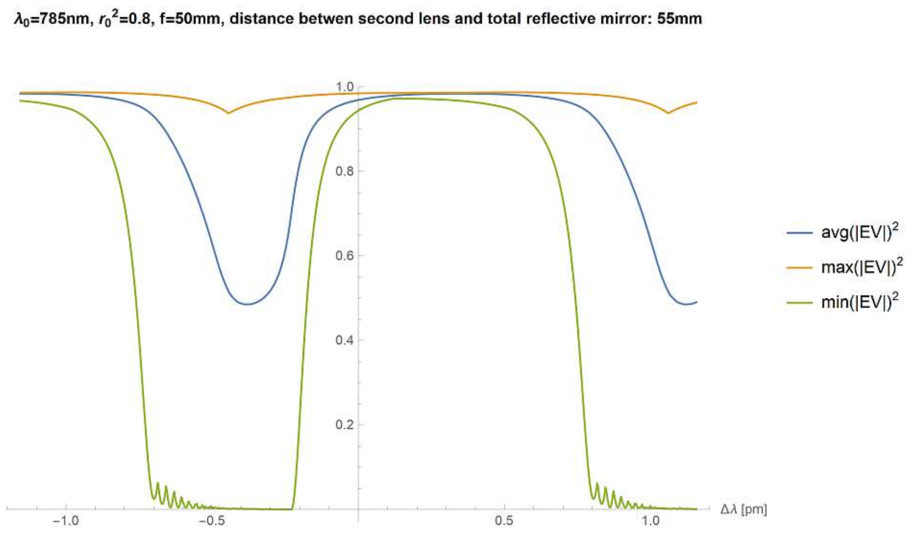

Figure 4.16.: Average over all squared absolute reflection-matrix eigenvalues, compared to the largest and smallest squared absolute reflection-matrix eigenvalues of a 4f-cavity-CPA with the total reflective mirror moved away from the optimal position by $5\,\mathrm{mm}$, plotted as a function of the wavelength around the resonance point of an optimally configured 4f-cavity. Parameters: $f = 50\,\mathrm{mm}, r_0^2 = 0.8,\ \lambda_0 = 785\,\mathrm{nm}$.

Discussion

An $f = 50\,\mathrm{mm}$ 4f-cavity with the total reflective back-mirror M2 being displaced by $1\,\mathrm{mm}$ still shows a CPA-behavior where the average over all squared absolute reflection-matrix eigenvalues goes down below 0.1 at the resonance point (see figure 4.14). With larger deviations the CPA-effect deteriorates (see figures 4.15 and 4.16).

At first glance it seems that large deviations would lead to some sort of “broad-band” effect for the least reflective eigenmodes. For example, in figure 4.16, depicting the situation where the total reflective mirror is moved away from the optimal position by $5\,\mathrm{mm}$, there is a whole range between $\Delta\lambda = -0.5\,\mathrm{pm}$ and $\Delta\lambda = -0.22\,\mathrm{pm}$ where the least reflective eigenmodes are practically zero.

However, as figure 4.17 reveals, the minimal reflective mode at each wavelength is not always the same mode. Therefore, this effect is not really an interesting broadband-effect.

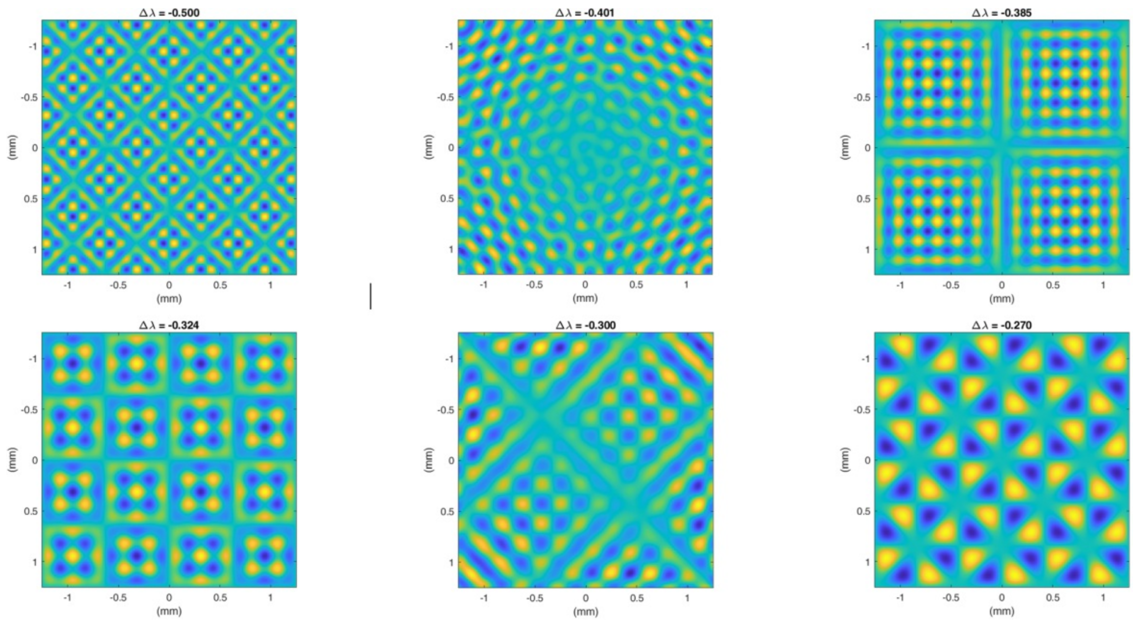

Figure 4.17.: Some of the almost perfectly absorbed eigenmodes of a 4f-cavity-CPA with the total reflective mirror moved away from the optimal position by $5\,\mathrm{mm}$ in the range between $\Delta\lambda = -0.5\,\mathrm{pm}$ and $\Delta\lambda = -0.22\,\mathrm{pm}$.

4.4.2. Deviations in the position of the second lens

Parameters

The simulations in this subsection have been carried out using the program described in appendix E.7 with the following parameter vectors:

```
% parameter variations
rho=[1, 1, 1, 1, 1];
d1 = [f, f, f, f, f];
d2 = [f*two+pval(gpu,'10e-6'), f*two+pval(gpu,'50e-6'), f*two+pval(gpu,'100e-6'), ...
      f*two+pval(gpu,'200e-6', f*two+pval(gpu,'500e-6')];
d3 = [f-pval(gpu,'10e-6'), f-pval(gpu,'50e-6'), f-pval(gpu,'100e-6'), ...
      f-pval(gpu,'200e-6'), f-pval(gpu,'500e-6')];
```

These parameter vectors define five simulation rounds, each with critical attenuation and the first lens and the total reflective back mirror M2 being in optimal position, but with the second lens in a slightly different position in each round, namely:

- two focal lengths after the first lens
- two focal lengths plus $0.05\,\mathrm{mm}$ after the first lens
- two focal lengths plus $0.1\,\mathrm{mm}$ after the first lens
- two focal lengths plus $0.2\,\mathrm{mm}$ after the first lens
- two focal lengths plus $0.3\,\mathrm{mm}$ after the first lens
- two focal lengths plus $0.5\,\mathrm{mm}$ after the first lens

Simulation Results

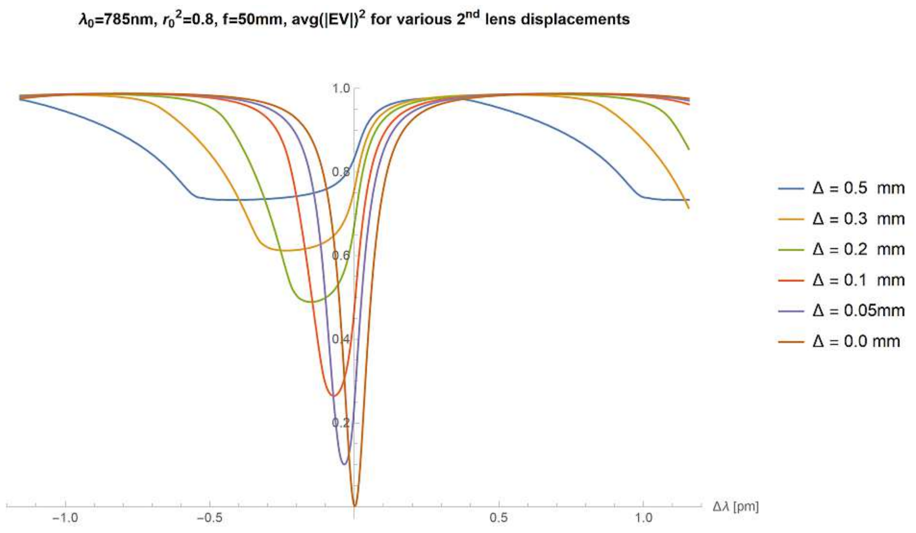

Figure 4.18.: Average over all squared absolute reflection-matrix eigenvalues of a 4f-cavity-CPA with the second lens moved away from the optimal position, plotted as a function of the wavelength around the resonance point of an optimally configured 4f-cavity. Parameters: $f = 50\,\text{mm}, r_0^2 = 0.8,\ \lambda_0 = 785\,\text{nm}$.

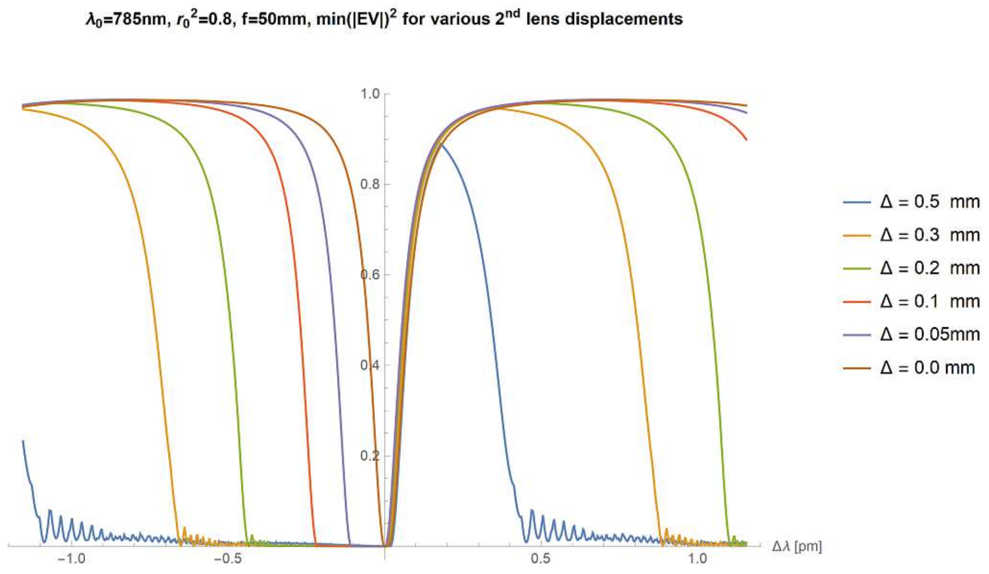

Figure 4.19.: Minimum squared absolute reflection-matrix eigenvalues of a 4f-cavity-CPA with the the second lens moved away from the optimal position, plotted as a function of the wavelength around the resonance point of an optimally configured 4f-cavity. Parameters: $f = 50\,\text{mm}, r_0^2 = 0.8,\ \lambda_0 = 785\,\text{nm}$.

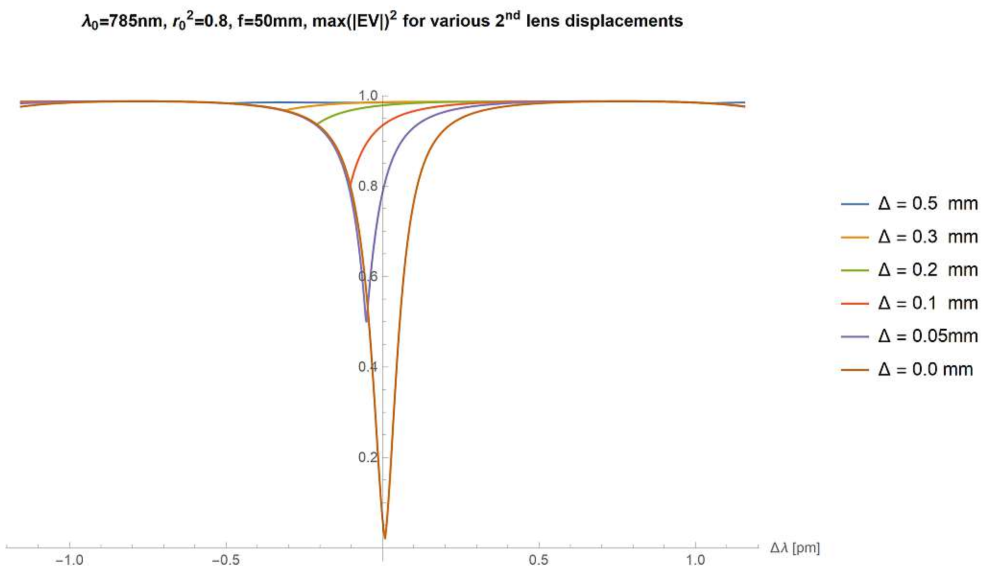

Figure 4.20.: Maximum squared absolute reflection-matrix eigenvalues of a 4f-cavity-CPA with the second lens moved away from the optimal position, plotted as a function of the wavelength around the resonance point of an optimally configured 4f-cavity. Parameters: $f = 50\,\mathrm{mm}, r_0^2 = 0.8,\ \lambda_0 = 785\,\mathrm{nm}$.

Discussion

An $f = 50\,\mathrm{mm}$ 4f-cavity with the second lens displaced by 0.05 mm still shows a CPA-behavior where the average over all squared absolute reflection-matrix eigenvalues goes down to 0.1 at the resonance point. With larger deviations the CPA-effect deteriorates (see figure 4.18).

Like in the previous subsection we observe a pseudo-broadband-effect for the least reflective eigenmodes (see figure 4.19).

4.4.3. Deviations from critical absorption

In this section we evaluate how the performance of a 4f-CPA is affected when the absorption deviates from the critical value.

Definition of absorption

Before we can evaluate how a 4f-CPA is affected by an absorption that is a certain fraction of the critical absorption, we have to clarify how to define “absorption”.

First of all, in our simulations so far we have just multiplied a "critical factor" $e^{-2\kappa_c l} = r_0$ to the single-round-trip transmission-matrix of an unattenuated 4f-cavity (see equations (3.21) and (3.41)). This is equivalent to assuming uniform absorption distributed across the whole length of the 4f-cavity.

In an actual experimental setup, however, we would insert an absorbing element with a certain thickness, as sketched in figure 4.21.

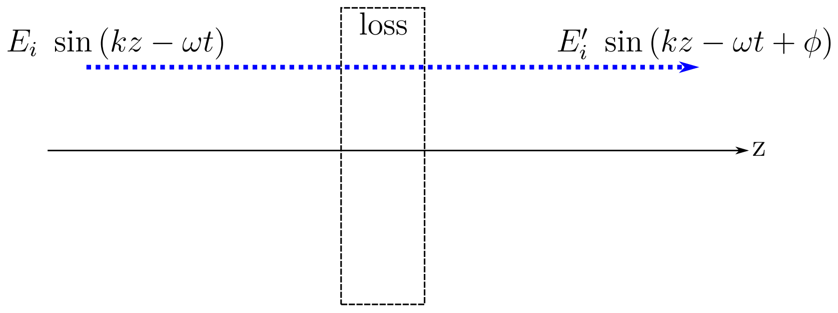
loss
$E_i \ \sin(kz - \omega t)$
$E_i' \ \sin(kz - \omega t + \phi)$
z

Figure 4.21.: Single-pass-absorption (passing the absorber once).

To define absorption, we assume a coherent electromagnetic field with a fixed wavelength. Let E_i be the amplitude of the electric field vector in any of the two directions before having passed the absorber, and E_i' the amplitude of this electric field vector after having passed the absorber. Then the absorption a shall be defined as

$$\begin{aligned} E_i'^2 &= E_i^2 - aE_i^2 \\ E_i'^2 &= E_i^2 (1 - a) \end{aligned} \tag{4.1}$$

As the electromagnetic field intensity is proportional to $\left|\vec{E}\right|^2$, and we choose the representation $E_i(z,t) = E_i \sin(kz - \omega t + \Phi)$, where E_i is is just a real number representing the amplitude of the electric field vector in the given transverse direction i, E_i^2 is also proportional to the intensity (assuming no rotating polarization and a isotropic, homogeneous medium). Therefore, a is the absorption of intensity.

The factor $e^{-2\kappa_c l} = r_0$ that we have used so far in equations (3.21) and (3.41) expresses the absorption of a *single round-trip* through the cavity. If we want to achieve the critical *round-trip* absorption by means of such an absorber with finite thickness somewhere in the 4f-cavity, we must consider that the electromagnetic wave passes the absorber twice, as depicted in figure 4.22.

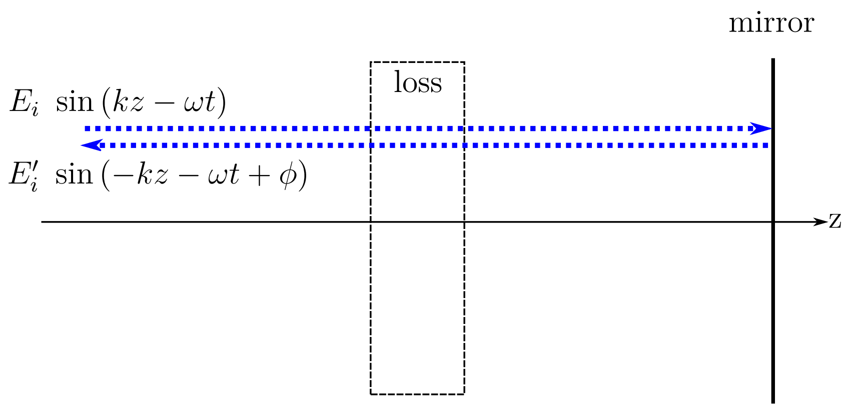

Figure 4.22.: Single-round-trip-absorption (em-wave is passing the absorber twice).

Obviously, in that case (where the wave-front passes the absorber twice) we have to write:

$$E_i'^2 = E_i^2 (1-a)(1-a)$$
$$E_i'^2 = E_i^2 (1-a)^2 \tag{4.2}$$

From that we can derive:

$$\frac{E_i'^2}{E_i^2} = (1-a)^2$$
$$\frac{E_i'}{E_i} = 1-a \qquad |a \to a_c$$
$$\left(\frac{E_i'}{E_i}\right)_c = 1-a_c \qquad \left|\left(\frac{E_i'}{E_i}\right)_c = e^{-2\kappa l}\right.$$
$$e^{-2\kappa l} = 1-a_c \overset{(3.21)}{\Longrightarrow}$$
$$r_0 = 1-a_c \tag{4.3}$$
$$a_c = 1-r_0 \tag{4.4}$$

We now replace r_0 by ρr_0 (see line 85 in the program listed in appendix E.7), and a_c by αa_c. Both ρ and α are real numbers so that $\rho \in [0,1]$ and $\alpha \geq 0$. For example, $\alpha = 0.5$ would mean to have an absorption that is 50% of the critical absorption.

The question to be solved is: given a value α, how can the corresponding value ρ (as required in line 85 of the program listed in appendix E.7) be calculated? The following simple derivation provides the answer. We start with equation (4.3):

$$
\begin{aligned}
r_0 &= 1 - a_c && |r_0 \to \rho r_0,\ a \to \alpha a_c \\
\rho r_0 &= 1 - \alpha a_c \overset{(4.4)}{\Longrightarrow} \\
\rho r_0 &= 1 - \alpha\,(1 - r_0) \\
\rho &= \frac{1 - \alpha\,(1 - r_0)}{r_0} && (4.5)
\end{aligned}
$$

Parameters

Using the program described in appendix E.7, we run five simulations, each with both lenses and both mirrors in optimal position, but each with different absorption, namely:

- 25% of the critical absorption
- 50% of the critical absorption
- 100% of the critical absorption
- 200% of the critical absorption
- 400% of the critical absorption

Using equation (4.5), this can be translated into the following parameter vectors for the program described in appendix E.7:

```
% parameter variations
rho=(1-[0.25, 0.5, 1, 1.5, 2, 4] * (1-r0))/r0;
d1 = [f, f, f, f, f, f];
d2 = [f*two, f*two, f*two, f*two, f*two, f*two];
d3 = [f, f, f, f, f, f];
```

Simulation Results

The following figures show the behavior of an $f = 50\,\mathrm{mm}$ 4f-cavity for undercritical attenuation (average squared absolute eigenvalues, minimum squared absolute eigenvalues, and maximum squared absolute eigenvalues of the reflection matrix):

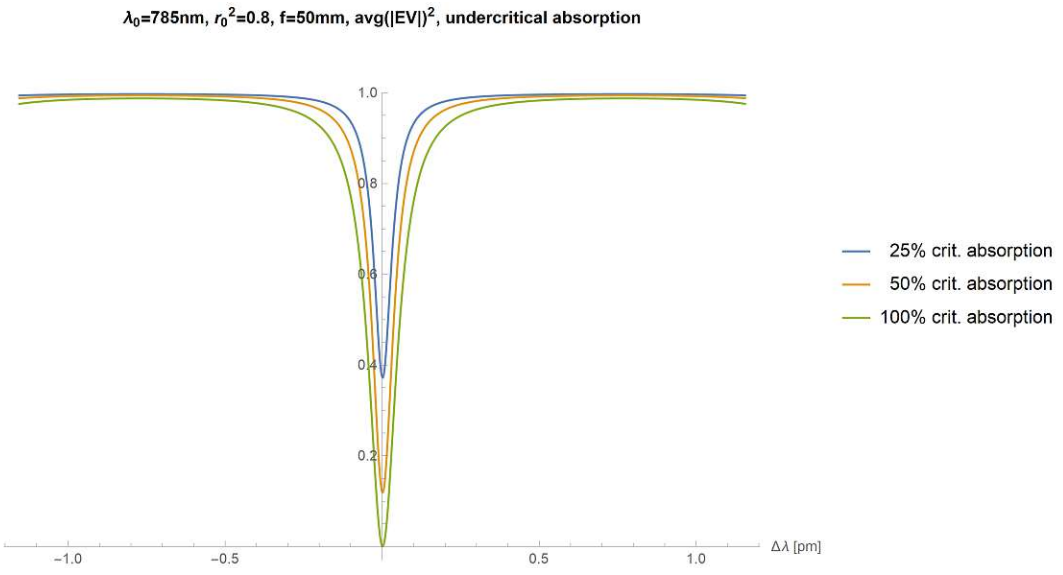

Figure 4.23.: Average over all squared absolute reflection-matrix eigenvalues of a 4f-cavity-CPA with various undercritical absorption values, plotted as a function of the wavelength around the resonance point of an optimally configured 4f-cavity. Parameters: $f = 50\,\text{mm}, r_0^2 = 0.8,\ \lambda_0 = 785\,\text{nm}$.

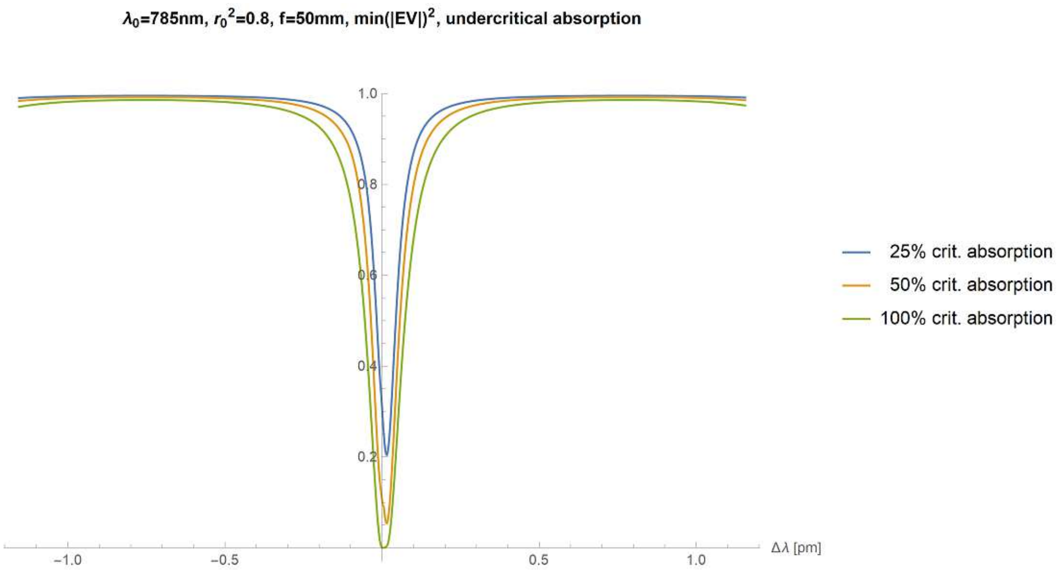

Figure 4.24.: Minimum squared absolute reflection-matrix eigenvalues of a 4f-cavity-CPA with various undercritical absorption values, plotted as a function of the wavelength around the resonance point of an optimally configured 4f-cavity. Parameters: $f = 50\,\text{mm}, r_0^2 = 0.8$, $\lambda_0 = 785\,\text{nm}$.

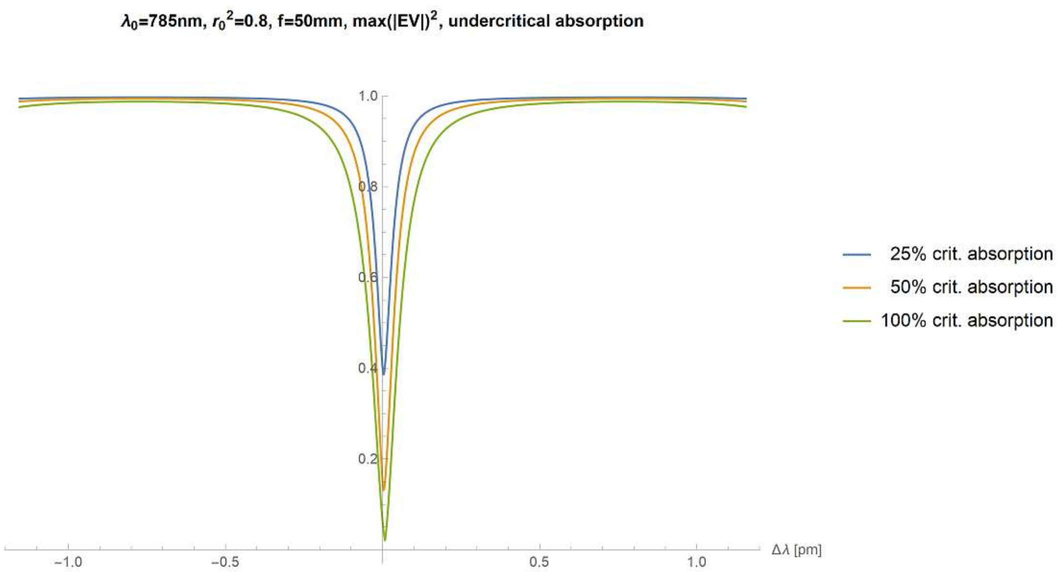

Figure 4.25.: Maximum squared absolute reflection-matrix eigenvalues of a 4f-cavity-CPA with various undercritical absorption values, plotted as a function of the wavelength around the resonance point of an optimally configured 4f-cavity. Parameters: $f = 50\,\mathrm{mm}, r_0^2 = 0.8$, $\lambda_0 = 785\,\mathrm{nm}$.

The following three figures depict the simulation results for overcritical absorption:

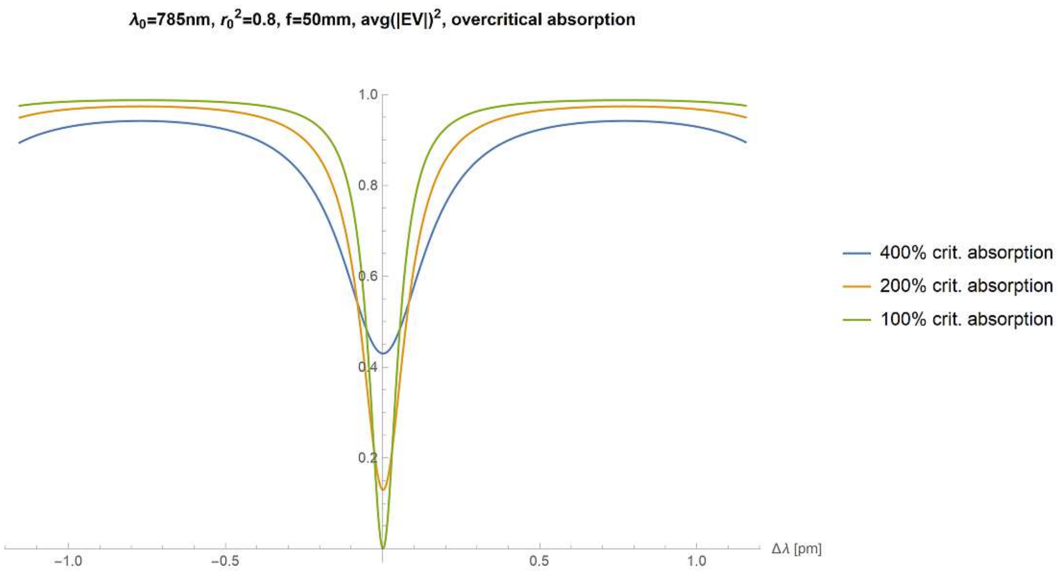

Figure 4.26.: Average over all squared absolute reflection-matrix eigenvalues of a 4f-cavity-CPA with various overcritical absorption values, plotted as a function of the wavelength around the resonance point of an optimally configured 4f-cavity. Parameters: $f = 50\,\mathrm{mm}, r_0^2 = 0.8$, $\lambda_0 = 785\,\mathrm{nm}$.

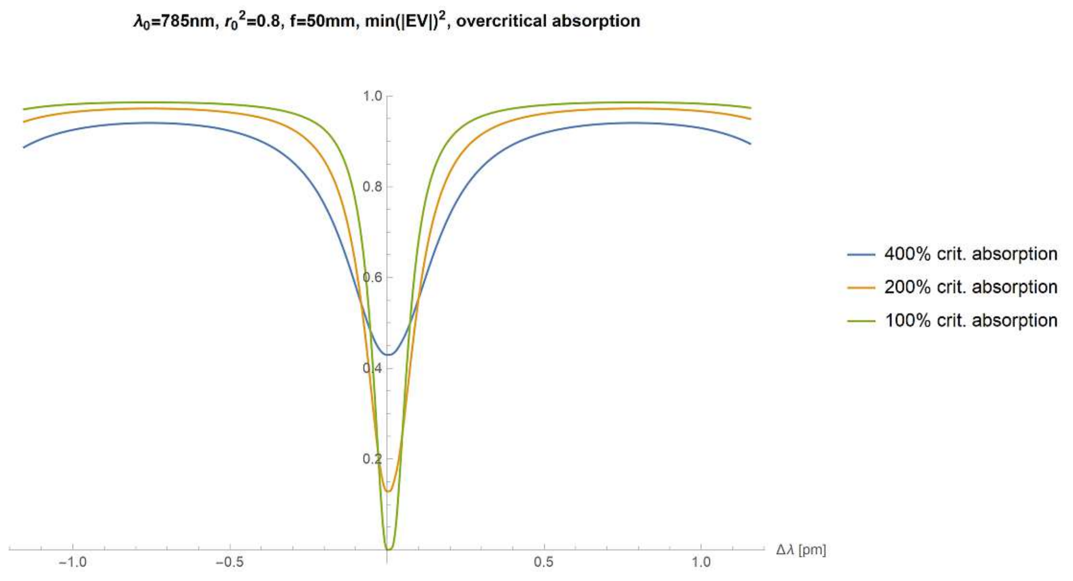

Figure 4.27.: Minimum squared absolute reflection-matrix eigenvalues of a 4f-cavity-CPA with various overcritical absorption values, plotted as a function of the wavelength around the resonance point of an optimally configured 4f-cavity. Parameters: $f = 50\,\text{mm}, r_0^2 = 0.8$, $\lambda_0 = 785\,\text{nm}$.

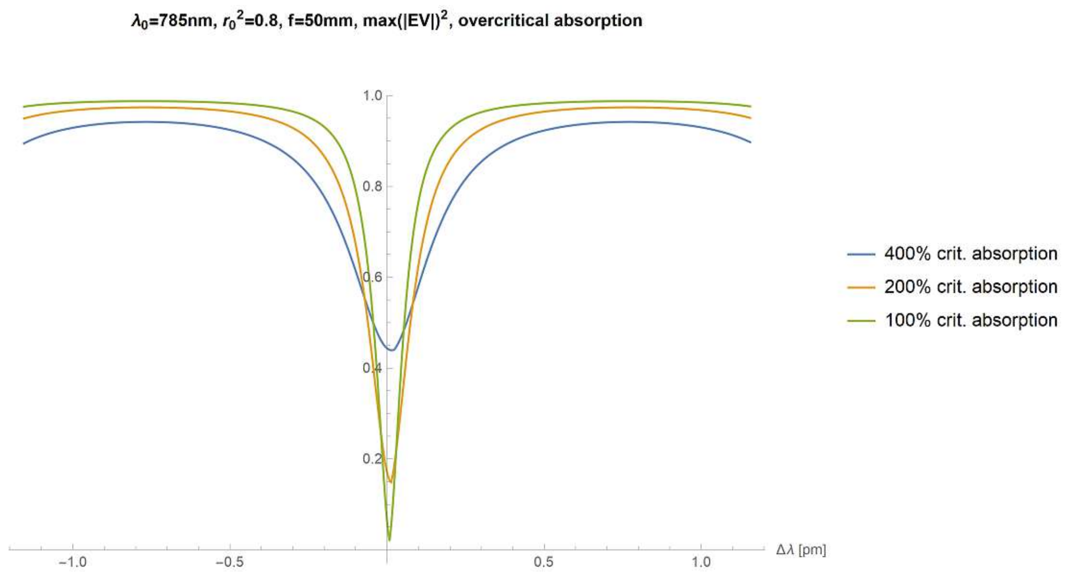

Figure 4.28.: Maximum squared absolute reflection-matrix eigenvalues of a 4f-cavity-CPA with various overcritical absorption values, plotted as a function of the wavelength around the resonance point of an optimally configured 4f-cavity. Parameters: $f = 50\,\text{mm}, r_0^2 = 0.8$, $\lambda_0 = 785\,\text{nm}$.

Discussion

The 4f-cavity-CPA turns out to be quite insensitive to deviations from the critical absorption. Even with just 50% or as much as 200% of the critical absorption, the average, minimum, and maximum squared absolute eigenvalues of the reflection matrix stay around or below 0.1 at the resonance point.

4.4.4. Sensitivity against mirror tilt

Software Implementation

For this simulation, we have implemented the function `round_trip_no_atten3` (see appendix C.20). It is an enhanced version of the function `round_trip_no_atten2` as described in appendix C.19, which sends an input field on a round-trip through a 4f-cavity, assuming no attenuation. By employing the `tilt`-function – documented in appendix C.14 – the new function allows to define the angle in degrees by which the total reflective back-mirror is tilted relative to the z-axis in the yz-plane. The theory behind the `tilt`-function is explained in section 4.4.4.

Based on this, another new function `transmision_matrix_round_trip_no_atten3` (see appendix C.23), which is an enhanced version of the function `transmision_matrix_round_trip_no_atten2` (see appendix C.22), creates the transmission matrix of a 4f-cavity with a slightly tilted back-mirror.

Parameters

The simulations in this subsection have been carried out using the program described in appendix E.8 with the following parameter vectors:

```
% parameter variations
rho= [1, 1, 1, 1, 1, 1];
d1 = [f, f, f, f, f, f];
d2 = [f*two, f*two, f*two, f*two, f*two, f*two];
d3 = [f, f, f, f, f, f];
mirror_tilt = [0.005, 0.003, 0.001, 0.0005, 0.0003, 0.0001]; % degrees
```

These parameter vectors define six simulation rounds, each with critical attenuation and all lenses and mirrors being in the optimal position, but with the total reflective back mirror (mirror M2 in figure 3.15) tilted relative to the z-axis by an angle of 0.005°, 0.003°, 0.001°, 0.0005°, 0.0003°, and 0.0001°.

Simulation Results

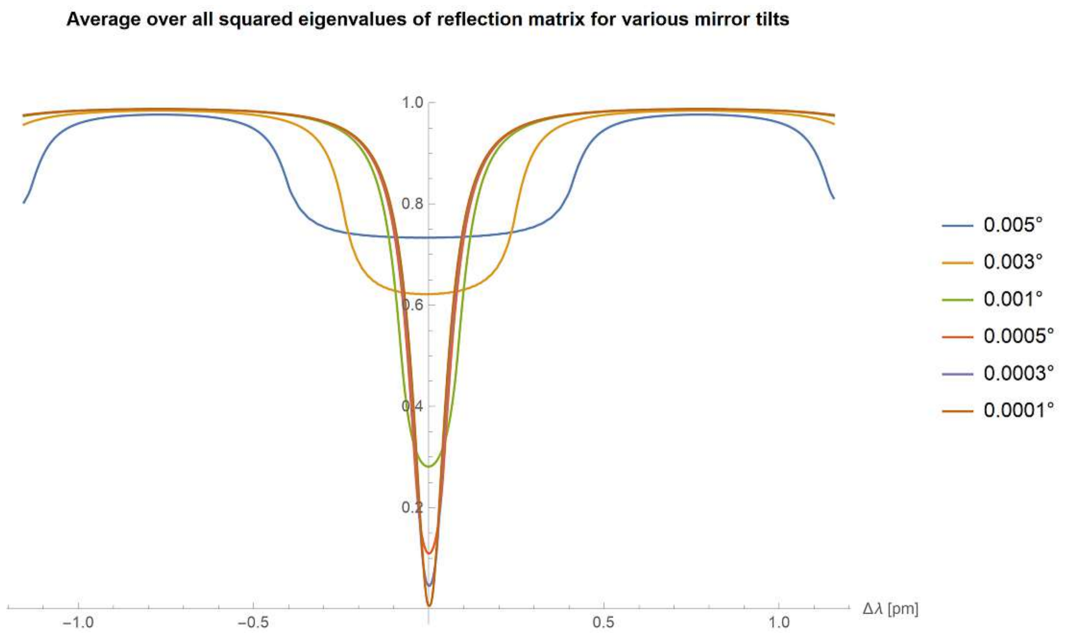

Figure 4.29.: Average over all squared absolute reflection-matrix eigenvalues of a 4f-cavity-CPA with various tilt angles of the total reflective mirror, plotted as a function of the wavelength around the resonance point of an optimally configured 4f-cavity. Parameters: $f = 50\,\mathrm{mm}, r_0^2 = 0.8,\ \lambda_0 = 785\,\mathrm{nm}$, respectively.

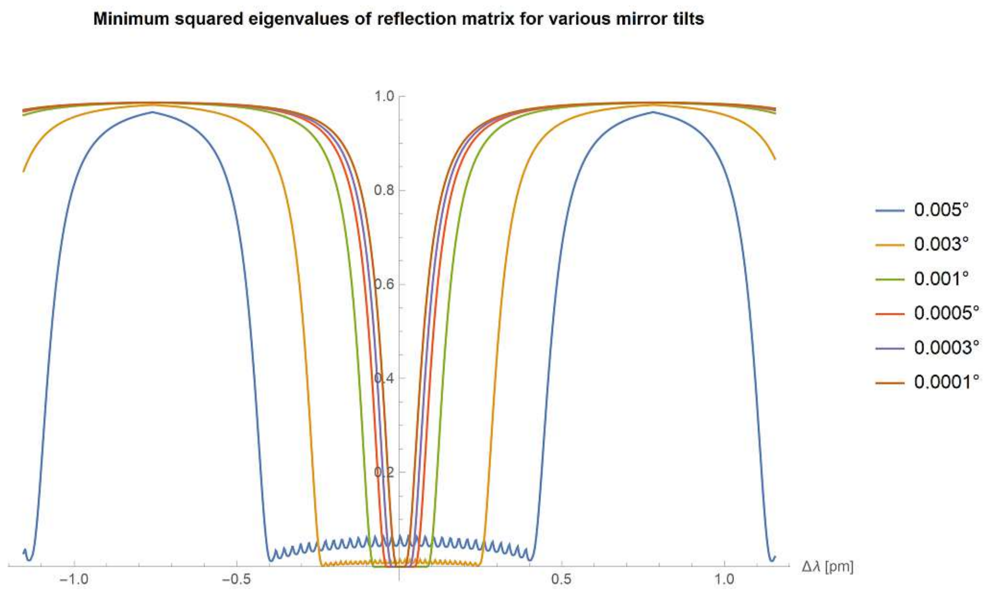

Figure 4.30.: Minimum squared absolute reflection-matrix eigenvalues of a 4f-cavity-CPA with various tilt angles of the total reflective mirror, plotted as a function of the wavelength around the resonance point of an optimally configured 4f-cavity. Parameters: $f = 50\,\mathrm{mm}, r_0^2 = 0.8$, $\lambda_0 = 785\,\mathrm{nm}$.

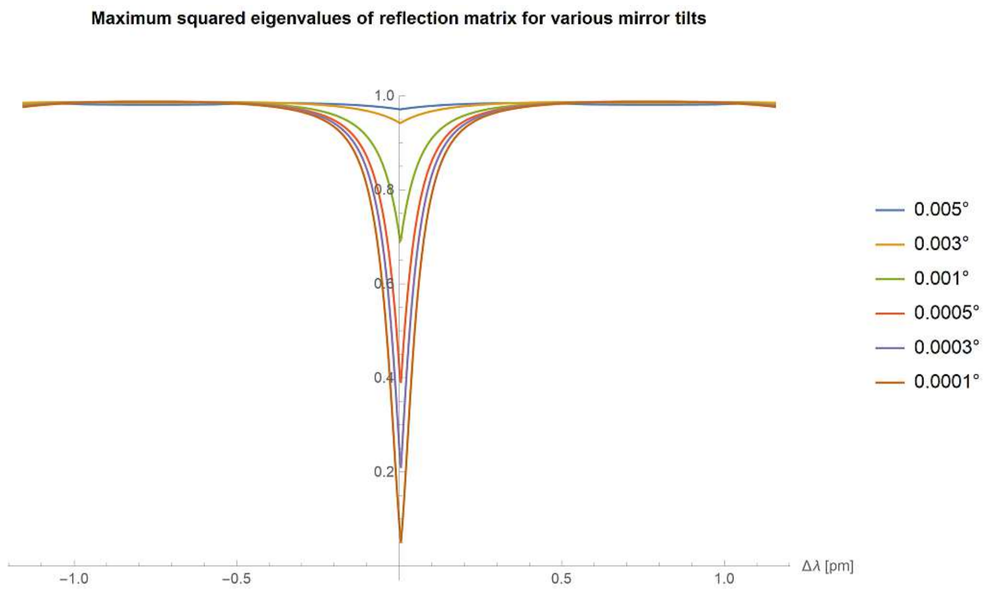

Figure 4.31.: Maximum squared absolute reflection-matrix eigenvalues of a 4f-cavity-CPA with various tilt angles of the total reflective mirror, plotted as a function of the wavelength around the resonance point of an optimally configured 4f-cavity. Parameters: $f = 50\,\mathrm{mm}, r_0^2 = 0.8,\ \lambda_0 = 785\,\mathrm{nm}$.

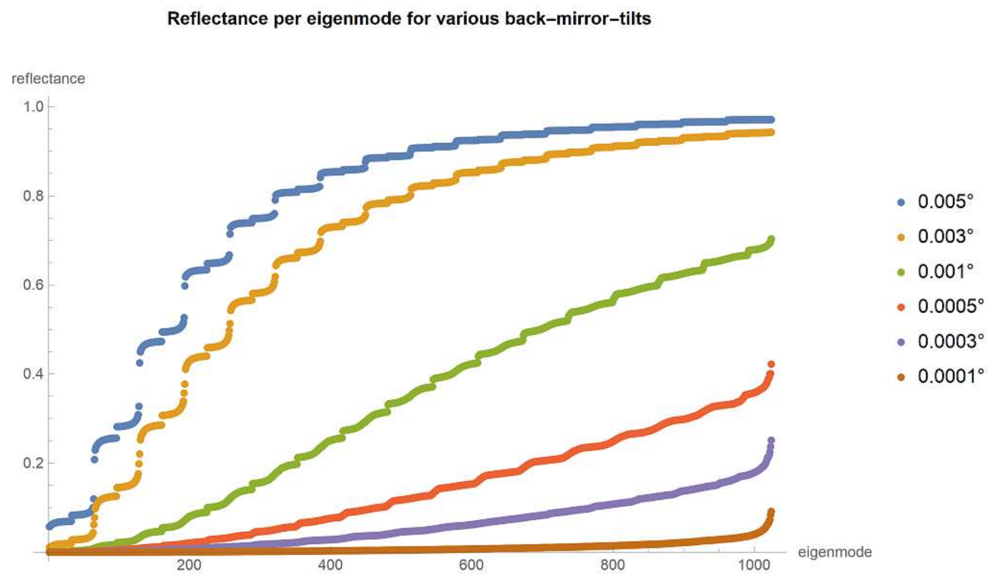

Figure 4.32.: Squared absolute eigenvalues of the 4f-cavity-CPA reflection-matrix plotted in ascending order for various tilt angles of the total reflective mirror. Parameters: $f = 50\,\mathrm{mm}, r_0^2 = 0.8,\ \lambda_0 = 785\,\mathrm{nm}$.

Discussion

As to be expected for a cavity with plane-parallel mirrors, the 4f-cavity-CPA is very sensitive to mirror tilts. Tilting the total reflective mirror by more than 0.0001° already leads to a noticeable deterioration of the CPA-effect. In an experimental setup, the effective tilt of both mirrors together should not exceed 0.0005°. At this value, the average over all squared absolute eigenvalues still reaches 0.1 at the resonance point (see figure 4.29).

5. A refined simulation method using scattering matrices and transfer matrices

5.1. Introduction

Although the model for simulating a 4f-cavity, as derived in sections 3.2 and 3.3, and as used in the computer simulations so far, has proven to be quite useful and efficient, it nevertheless has certain limitations.

Most noticeably, the computer simulation itself does *not* directly produce the interesting CPA-effects arising from the infinite number of round-trips in the cavity. Rather, the computer only simulates a *single* round-trip through the cavity by creating the corresponding transmission matrix, and then this result needs to be fed into the explicitly derived problem-specific geometric-series-formula (3.39).

Also, with this approach we are, e.g., not able to simulate the effect that residual reflections on the facets of the lenses or the absorber would have. To overcome these limitations, we will derive a more refined and more flexible simulation method in this chapter, using scattering matrices and transfer matrices which (unlike the simple transmission matrix) are able to deal with incoming and outgoing fields in *both* directions at each optical component.

5.2. Simulating the 1D toy-model with transfer matrices and scattering matrices

5.2.1. One-dimensional cavity without attenuation

In this section we again consider the simple one-dimensional toy-model from section 3.2 as depicted in figure 3.6. Generally, the scattering matrix of any optical component in our simple, one-dimensional setting can be expressed as

$$\underline{\underline{S}} = \begin{bmatrix} r & t' \\ t & r' \end{bmatrix} \tag{5.1}$$

In equation (5.1), r and t represent the complex reflection coefficient and transmission coefficient for light waves passing from left to right, whereas r' and t' represent the complex reflection coefficient and transmission coefficient for light waves passing from right to left.

Let r_1 and t_1 be the reflection coefficient and transmission coefficient of the partially reflective, left mirror M_1. If we assume the mirror to be symmetric, then we can substitute in equation (5.1) $r = r' = r_1$, $t = t' = t_1$, and the left mirror's scattering matrix $\underline{\underline{S}}_1$ becomes:

$$\underline{\underline{S}}_1 = \begin{bmatrix} r_1 & t_1 \\ t_1 & r_1 \end{bmatrix} \tag{5.2}$$

Similarly, the scattering matrix of the total reflective, perfect mirror M_2 can be expressed as

$$\underline{\underline{S}}_2 = \begin{bmatrix} r_2 & t_2 \\ t_2 & r_2 \end{bmatrix} \tag{5.3}$$

Please note that, for reasons to become obvious soon, we are not substituting t_2 with the specific value $t_2 = 0$.

Also the propagation of the light field in either direction between the two mirrors can be expressed by means of a scattering matrix $\underline{\underline{S}}_p$ with the transmission coefficients $t = t' = e^{ikl}$, and the reflection coefficients set to $r = r' = 0$. This results in the following scattering matrix for propagation between mirrors M_1 and M_2:

$$\underline{\underline{S}}_p = \begin{bmatrix} 0 & e^{ikl} \\ e^{ikl} & 0 \end{bmatrix} \tag{5.4}$$

To combine the effects of all individual optical components, the scattering matrices need to be converted into transfer matrices. Generally, a scattering matrix

$$\underline{\underline{S}} = \begin{bmatrix} S_{11} & S_{12} \\ S_{21} & S_{22} \end{bmatrix} \tag{5.5}$$

can be converted into a corresponding transfer matrix $\underline{\underline{M}}$ by means of the following conversion formula:

$$\underline{\underline{M}}\left(\underline{\underline{S}}\right) = \begin{bmatrix} S_{12} - \frac{S_{11}S_{22}}{S_{21}} & \frac{S_{11}}{S_{21}} \\ -\frac{S_{22}}{S_{21}} & \frac{1}{S_{21}} \end{bmatrix} \tag{5.6}$$

Applying the conversion formula (5.6) to equations (5.2) and (5.3) results in the following transfer matrices $\underline{\underline{M}}_1$ and $\underline{\underline{M}}_2$ for the left and right mirror:

$$\underline{\underline{M}}_1 = \begin{bmatrix} t_1 - \frac{r_1^2}{t_1} & \frac{r_1}{t_1} \\ -\frac{r_1}{t_1} & \frac{1}{t_1} \end{bmatrix} \tag{5.7}$$

$$\underline{\underline{M}}_2 = \begin{bmatrix} t_2 - \frac{r_2^2}{t_2} & \frac{r_2}{t_2} \\ -\frac{r_2}{t_2} & \frac{1}{t_2} \end{bmatrix} \tag{5.8}$$

Note that each expression in the $\underline{\underline{M}}_2$ conversion-matrix of equation (5.8) has at least one term with t_2 being the denominator. This is why we have not substituted t_2 with the specific value $t_2 = 0$ in equation (5.3). Although this seems like a problem at this point, it will eventually turn out not to be, at least for calculating the cavity's reflection matrix.

To complete this step of the calculation, we also need to convert the propagation scattering matrix $\underline{\underline{S}}_p$ into a propagation transfer matrix $\underline{\underline{M}}_p$ by applying equation (5.6) to equation (5.4), which results in

$$\underline{\underline{M}}_p = \begin{bmatrix} e^{ikl} & 0 \\ 0 & e^{-ikl} \end{bmatrix} \tag{5.9}$$

With all three transfer matrices in place, we can now calculate the total transfer matrix $\underline{\underline{M}}_{tot}$ for the entire cavity:

$$\begin{aligned}
\underline{\underline{M}}_{tot} &= \underline{\underline{M}}_1 \underline{\underline{M}}_p \underline{\underline{M}}_2 \xRightarrow{(5.7)(5.9)(5.8)} \\
\underline{\underline{M}}_{tot} &= \begin{bmatrix} t_1 - \frac{r_1^2}{t_1} & \frac{r_1}{t_1} \\ -\frac{r_1}{t_1} & \frac{1}{t_1} \end{bmatrix} \begin{bmatrix} e^{ikl} & 0 \\ 0 & e^{-ikl} \end{bmatrix} \begin{bmatrix} t_2 - \frac{r_2^2}{t_2} & \frac{r_2}{t_2} \\ -\frac{r_2}{t_2} & \frac{1}{t_2} \end{bmatrix} \\
\underline{\underline{M}}_{tot} &= \begin{bmatrix} \left(t_1 - \frac{r_1^2}{t_1}\right)\left(t_2 - \frac{r_2^2}{t_2}\right) e^{ikl} - \frac{r_1 r_2}{t_1 t_2} e^{-ikl} & \frac{r_2}{t_2}\left(t_1 - \frac{r_1^2}{t_1}\right) e^{ikl} + \frac{r_1}{t_1 t_2} e^{-ikl} \\ -\frac{r_1}{t_1}\left(t_2 - \frac{r_2^2}{t_2}\right) e^{ikl} - \frac{r_2}{t_1 t_2} e^{-ikl} & -\frac{r_1 r_2}{t_1 t_2} e^{ikl} + \frac{1}{t_1 t_2} e^{-ikl} \end{bmatrix}
\end{aligned} \tag{5.10}$$

The goal of our calculation is to determine the reflection-coefficient r_{tot} of the entire cavity with respect to an incident wave traveling in the positive x-direction towards the cavity from the left side. But — as shown in equation (5.1) — this is nothing else than the top-left entry of the entire cavity's scattering matrix. Therefore, to determine the reflection coefficient r_{tot}, we just need to back-convert the transfer matrix $\underline{\underline{M}}_{tot}$ into the corresponding scattering matrix $\underline{\underline{S}}_{tot}$.

Analogously to the opposite operation presented above, a transfer matrix

$$\underline{\underline{M}} = \begin{bmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{bmatrix} \tag{5.11}$$

can be converted into a corresponding scattering matrix $\underline{\underline{S}}$ by means of the following conversion formula:

$$\underline{\underline{S}}\left(\underline{\underline{M}}\right) = \begin{bmatrix} \frac{M_{12}}{M_{22}} & M_{11} - \frac{M_{12}M_{21}}{M_{22}} \\ \frac{1}{M_{22}} & -\frac{M_{21}}{M_{22}} \end{bmatrix} \tag{5.12}$$

Applying the conversion formula (5.12) to equation (5.10), and simplifying the expressions, results in the following scattering matrix for the whole unattenuated 4f-cavity:

$$\underline{\underline{S}}_{tot} = \begin{bmatrix} r_1 + \frac{r_2 t_1^2 e^{2ikl}}{1-r_1 r_2 e^{2ikl}} & \frac{t_1 t_2 e^{ikl}}{1-r_1 r_2 e^{2ikl}} \\ \frac{t_1 t_2 e^{ikl}}{1-r_1 r_2 e^{2ikl}} & r_2 + \frac{r_1 t_2^2 e^{2ikl}}{1-r_1 r_2 e^{2ikl}} \end{bmatrix} \tag{5.13}$$

We are interested in the reflection-coefficient r_{tot}, which is the top-left entry in the scattering matrix $\underline{\underline{S}}_{tot}$:

$$r_{tot} = r_1 + \frac{r_2 t_1^2 e^{2ikl}}{1 - r_1 r_2 e^{2ikl}} \tag{5.14}$$

After renaming k to $\tilde{k}$, the result in equation (5.14) is identical with equation (3.16). Therefore, we have now proven that this method of using scattering matrices and transfer matrices can directly produce the reflection-coefficient of a one-dimensional cavity. Further, it has turned out that the total reflective mirror's transmission coefficient t_2 does not appear in the final formula. Therefore, the apparent problem with $t_2 = 0$ in equation (5.8) has disappeared.

5.2.2. One-dimensional cavity with absorber

We are now considering a one-dimensional plane-parallel optical cavity of total length l with a fully reflective mirror M_2 on the right side and a partially reflective mirror M_1 on the left side, as well as an attenuating element of thickness d in the middle, as depicted in figure 5.1.

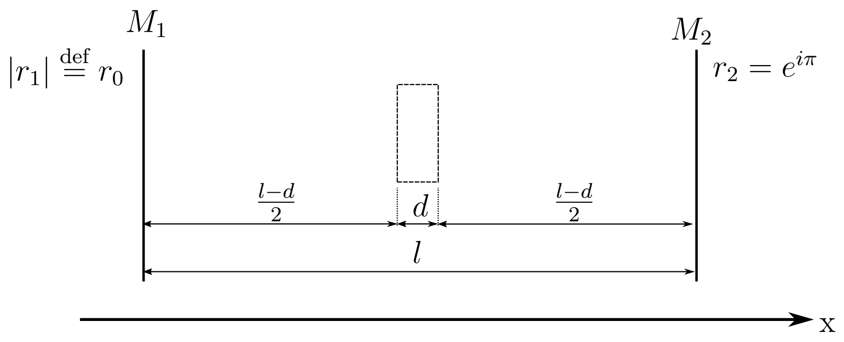

Figure 5.1.: A simple one-dimensional plane-parallel optical cavity of total length l with a fully reflective mirror M_2 on the right side, as well as a partially reflective mirror M_1 on the left side, and an attenuating element of thickness d in the center.

The scattering matrix of the attenuation element can be expressed as

$$\underline{\underline{S}}_a = \begin{bmatrix} 0 & e^{id(k+i\kappa)} \\ e^{id(k+i\kappa)} & 0 \end{bmatrix} \tag{5.15}$$

Applying the conversion formula (5.6) to equation (5.15) gives the corresponding transfer matrix :

$$\underline{\underline{M}}_a = \begin{bmatrix} e^{id(k+i\kappa)} & 0 \\ 0 & e^{-id(k+i\kappa)} \end{bmatrix} \tag{5.16}$$

The propagation in either direction between mirror M_1 and the left side of the attenuation element, as well as the propagation in either direction between the right side of the attenuation element and mirror M_2 can be described by the following scattering matrix:

$$\underline{\underline{S}}_p = \begin{bmatrix} 0 & e^{ik\frac{l-d}{2}} \\ e^{ik\frac{l-d}{2}} & 0 \end{bmatrix} \tag{5.17}$$

Again, applying the conversion formula (5.6) to equation (5.17) yields the corresponding transfer matrix:

$$\underline{\underline{\mathrm{M}}}_p = \begin{bmatrix} e^{ik\frac{l-d}{2}} & 0 \\ 0 & e^{-ik\frac{l-d}{2}} \end{bmatrix} \tag{5.18}$$

Hence, the total transfer matrix $\underline{\underline{\mathrm{M}}}_{tot}$ for the entire cavity can be calculated as:

$$\begin{aligned}
\underline{\underline{\mathrm{M}}}_{tot} &= \underline{\underline{\mathrm{M}}}_1 \underline{\underline{\mathrm{M}}}_p \underline{\underline{\mathrm{M}}}_a \underline{\underline{\mathrm{M}}}_p \underline{\underline{\mathrm{M}}}_2 \overset{(5.7)(5.8)(5.16)(5.20)}{\Longrightarrow} \\
\underline{\underline{\mathrm{M}}}_{tot} &= \begin{bmatrix} t_1 - \frac{r_1^2}{t_1} & \frac{r_1}{t_1} \\ -\frac{r_1}{t_1} & \frac{1}{t_1} \end{bmatrix} \begin{bmatrix} e^{ik\frac{l-d}{2}} & 0 \\ 0 & e^{-ik\frac{l-d}{2}} \end{bmatrix} \begin{bmatrix} e^{id(k+i\kappa)} & 0 \\ 0 & e^{-id(k+i\kappa)} \end{bmatrix} \cdot \\
&\quad \begin{bmatrix} e^{ik\frac{l-d}{2}} & 0 \\ 0 & e^{-ik\frac{l-d}{2}} \end{bmatrix} \begin{bmatrix} t_2 - \frac{r_2^2}{t_2} & \frac{r_2}{t_2} \\ -\frac{r_2}{t_2} & \frac{1}{t_2} \end{bmatrix} \\
\underline{\underline{\mathrm{M}}}_{tot} &= \begin{bmatrix} \frac{(r_1-t_1)(r_1+t_1)(r_2-t_2)(r_2+t_2)e^{2ikl}-r_1r_2e^{2d\kappa}}{t_1t_2e^{ikl}e^{d\kappa}} & \frac{-r_2(r_1-t_1)(r_1+t_1)e^{2ikl}+r_1e^{2d\kappa}}{t_1t_2e^{ikl}e^{d\kappa}} \\ \frac{r_1(r_2-t_2)(r_2+t_2)e^{2ikl}-r_2e^{2d\kappa}}{t_1t_2e^{d\kappa}e^{ikl}} & \frac{e^{-ikl}e^{d\kappa}-r_1r_2e^{ikl}e^{-d\kappa}}{t_1t_2} \end{bmatrix}
\end{aligned} \tag{5.19}$$

Applying the conversion formula (5.12) to equation (5.19), and simplifying the expressions, results in the following scattering matrix for the whole attenuated cavity:

$$\underline{\underline{\mathrm{S}}}_{tot} = \begin{bmatrix} \frac{r_2(r_1-t_1)(r_1+t_1)e^{2ikl}-r_1e^{2d\kappa}}{r_1r_2e^{2ikl}-e^{2d\kappa}} & -\frac{t_1t_2e^{ikl}e^{d\kappa}}{r_1r_2e^{2ikl}-e^{2d\kappa}} \\ -\frac{t_1t_2e^{ikl}e^{d\kappa}}{r_1r_2e^{2ikl}-e^{2d\kappa}} & \frac{r_1(r_2-t_2)(r_2+t_2)e^{2ikl}-r_2e^{2d\kappa}}{r_1r_2e^{2ikl}-e^{2d\kappa}} \end{bmatrix} \tag{5.20}$$

We are interested in the reflection-coefficient r_{tot}, which is the top-left entry in the scattering matrix $\underline{\underline{\mathrm{S}}}_{tot}$:

$$\begin{aligned}
r_{tot} &= \frac{r_2(r_1-t_1)(r_1+t_1)e^{2ikl}-r_1e^{2d\kappa}}{r_1r_2e^{2ikl}-e^{2d\kappa}} && \Big| r_2 = e^{i\pi} = -1 \\
r_{tot} &= \frac{-(r_1-t_1)(r_1+t_1)e^{2ikl}-r_1e^{2d\kappa}}{-r_1e^{2ikl}-e^{2d\kappa}} \\
r_{tot} &= \frac{(r_1-t_1)(r_1+t_1)e^{2ikl}+r_1e^{2d\kappa}}{r_1e^{2ikl}+e^{2d\kappa}}
\end{aligned} \tag{5.21}$$

Because $t_1 = 1 + r_1$ (see equation (9) in appendix A.6), this can be further simplified:

$$r_{tot} = \frac{(\cancel{r_1} - 1 - \cancel{r_1})(r_1 + 1 + r_1)e^{2ikl} + r_1e^{2d\kappa}}{r_1e^{2ikl} + e^{2d\kappa}}$$

$$r_{tot} = \frac{-(2r_1+1)e^{2ikl} + r_1 e^{2d\kappa}}{r_1 e^{2ikl} + e^{2d\kappa}} \xRightarrow{(3.11)}$$

$$r_{tot} = \frac{-\left(2\left(-r_0^2 - ir_0\sqrt{1-r_0^2}\right)+1\right)e^{2ikl} + \left(-r_0^2 - ir_0\sqrt{1-r_0^2}\right)e^{2d\kappa}}{\left(-r_0^2 - ir_0\sqrt{1-r_0^2}\right)e^{2ikl} + e^{2d\kappa}} \tag{5.22}$$

By setting $d = 0$ (no attenuation), equation (5.22) reduces to equation (3.17). On the other hand, taking equation (3.20), and replacing the total cavity length l with the absorber thickness d, gives the value that κ must take on for critical attenuation:

$$\kappa_c = -\frac{1}{2d}\ln(r_0) \tag{5.23}$$

By inserting this value for κ into equation (5.22), and by further substituting $k = \frac{2\pi}{\lambda}$, we can plot the total reflectance $|r_{cav}|^2$ of the cavity as a function of the wavelength and are expecting to see CPA behavior at certain resonance wavelengths.

Figure 5.2 shows the total reflectance of the cavity as a function of the wavelength under the assumption of $L = 300\,\text{mm}$, $\lambda_0 = 800\,\text{nm}$ and $r_0 = 0.7$ for critical attenuation $\kappa = \kappa_c$. As expected, the reflectance drops to zero at resonance conditions. This plot perfectly coincides with figure 3.7.

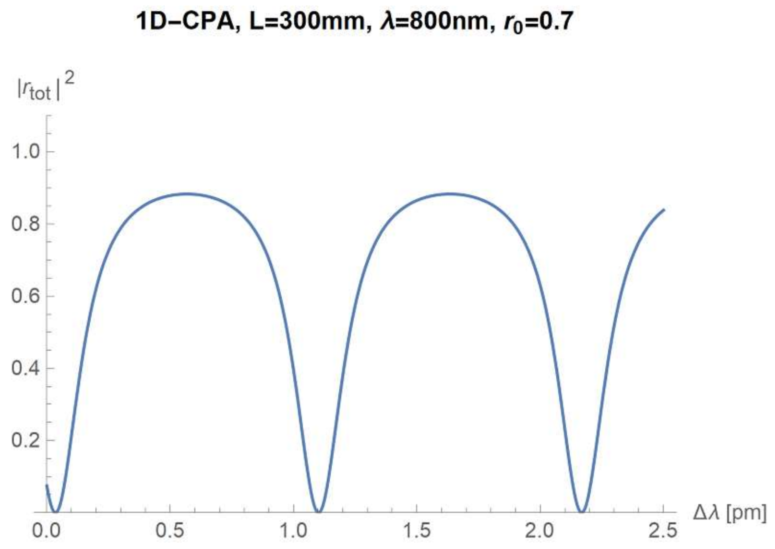

Figure 5.2.: Reflectance of a 1D-cavity with an $|r| = r_0 = 0.7$ partially reflective mirror, assuming a coherent incident light beam with a wavelength of $\lambda = 800\,\text{nm}$, and an absorber of thickness $d < L$ in the middle position providing critical attenuation.

5.2.3. One-dimensional cavity with an absorber having partially reflecting facets

In this subsection we are considering a setup just like the one in figure 5.1, with the only difference that the absorber now has partially reflective facets. These facets are modeled exactly like the partially reflective, left mirror (however, with their own reflection coefficient). Therefore, in analogy to equation (5.2), the scattering matrix describing either of the two facets can be expressed as

$$\underline{\underline{S}}_3 = \begin{bmatrix} r_3 & t_3 \\ t_3 & r_3 \end{bmatrix} \tag{5.24}$$

And, in analogy to equation (5.7), the transfer matrix describing each of the two facet, can be written as:

$$\underline{\underline{M}}_3 = \begin{bmatrix} t_3 - \frac{r_3^2}{t_3} & \frac{r_3}{t_3} \\ -\frac{r_3}{t_3} & \frac{1}{t_3} \end{bmatrix} \tag{5.25}$$

By multiplying all five transfer matrices in the according order, we obtain the total transfer matrix $\underline{\underline{M}}_{tot}$ for the whole cavity:

$$\begin{aligned} \underline{\underline{M}}_{tot} &= \underline{\underline{M}}_1 \underline{\underline{M}}_p \underline{\underline{M}}_3 \underline{\underline{M}}_a \underline{\underline{M}}_3 \underline{\underline{M}}_p \underline{\underline{M}}_2 \overset{(5.7)(5.8)(5.16)(5.20)(5.25)}{\Longrightarrow} \\ \underline{\underline{M}}_{tot} &= \begin{bmatrix} t_1 - \frac{r_1^2}{t_1} & \frac{r_1}{t_1} \\ -\frac{r_1}{t_1} & \frac{1}{t_1} \end{bmatrix} \begin{bmatrix} e^{ik\frac{l-d}{2}} & 0 \\ 0 & e^{-ik\frac{l-d}{2}} \end{bmatrix} \begin{bmatrix} t_3 - \frac{r_3^2}{t_3} & \frac{r_3}{t_3} \\ -\frac{r_3}{t_3} & \frac{1}{t_3} \end{bmatrix} \begin{bmatrix} e^{id(k+i\kappa)} & 0 \\ 0 & e^{-id(k+i\kappa)} \end{bmatrix} \cdot \\ &\quad \begin{bmatrix} t_3 - \frac{r_3^2}{t_3} & \frac{r_3}{t_3} \\ -\frac{r_3}{t_3} & \frac{1}{t_3} \end{bmatrix} \begin{bmatrix} e^{ik\frac{l-d}{2}} & 0 \\ 0 & e^{-ik\frac{l-d}{2}} \end{bmatrix} \begin{bmatrix} t_2 - \frac{r_2^2}{t_2} & \frac{r_2}{t_2} \\ -\frac{r_2}{t_2} & \frac{1}{t_2} \end{bmatrix} \end{aligned} \tag{5.26}$$

The calculation proceeds as in the previous two subsections (although now with much more complex expressions): The scattering matrix $\underline{\underline{S}}_{tot}$ for the whole cavity with a partially reflective absorber can be obtained by explicitly calculating $\underline{\underline{M}}_{tot}$ and then applying the conversion formula (5.12). The top-left entry of $\underline{\underline{S}}_{tot}$ is the reflection coefficient. After substituting $t_1 = 1 + r_1$ (see equation (9) in appendix A.6), and analogously also substituting $t_3 = 1 + r_3$, one gets the following expression for the cavity's reflection coefficient:

$$\begin{aligned} r_{tot} =& e^{4idk} \Big((2r_1+1) r_3^2 e^{2d\kappa+2ik(l-2d)} + (3r_1+1) r_3 e^{2d\kappa-3idk+ikl} + r_1 e^{2d(\kappa-ik)} + \\ & (3r_1+1)(2r_3+1) r_3 e^{ik(l-d)} - (2r_1+1)(2r_3+1)^2 e^{2ik(l-d)} - r_1 r_3^2 \Big) \cdot \\ & \Big(e^{2d(\kappa+ik)} - r_1 r_3^2 e^{2d\kappa+2ikl} - (r_1-1) r_3 e^{2d\kappa+ik(d+l)} - r_3^2 e^{4idk} - \\ & (r_1-1)(2r_3+1) r_3 e^{ik(3d+l)} + r_1 (2r_3+1)^2 e^{2ik(d+l)} \Big)^{-1} \end{aligned} \tag{5.27}$$

By substituting $k = \frac{2\pi}{\lambda}$, $r_1 = -|r_1|^2 - i\,|r_1|\,\sqrt{1-|r_1|^2}$ (see equation (3.11)), and analogously $r_3 = -|r_3|^2 - i\,|r_3|\,\sqrt{1-|r_3|^2}$, and by further inserting the critical value for κ (see equation (5.23)), we can plot the total reflectance $|r_{cav}|^2$ of the cavity as a function of the wavelength.

Figure 5.3 shows the total reflectance of a critically attenuated cavity ($\kappa = \kappa_c$) as a function of the wavelength under the assumption of $L = 300\,\text{mm}$, $\lambda_0 = 800\,\text{nm}$, $|r_1| = 0.7$, and a reflection coefficient $|r_3| = 0.1$ for both of the absorber's facets (chosen to be small, but not too small, so that some effect becomes visible). Figure 5.4 depicts the total reflectance of the cavity for the same parameters, except the absorber-facets having a unrealistically large reflectance of $|r_3| = 0.5$.

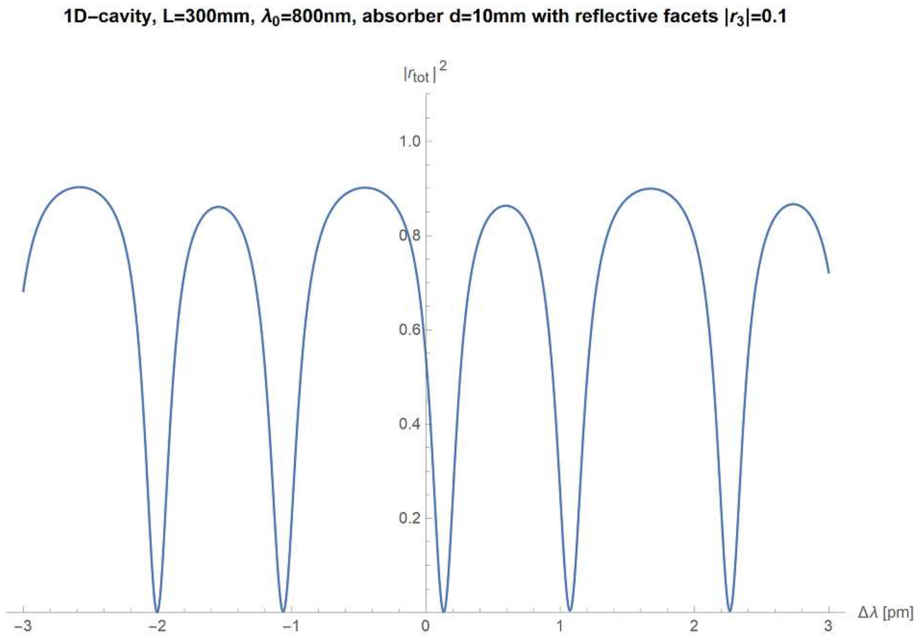

Figure 5.3.: Reflectance of a 1D-cavity with an $|r_1| = 0.7$ partially reflective mirror, $\lambda = 800\,\text{nm}$, and an absorber of thickness $d = 10\,\text{mm}$ in the center position with partially reflective facets ($|r_3| = 0.1$), tuned to critical attenuation.

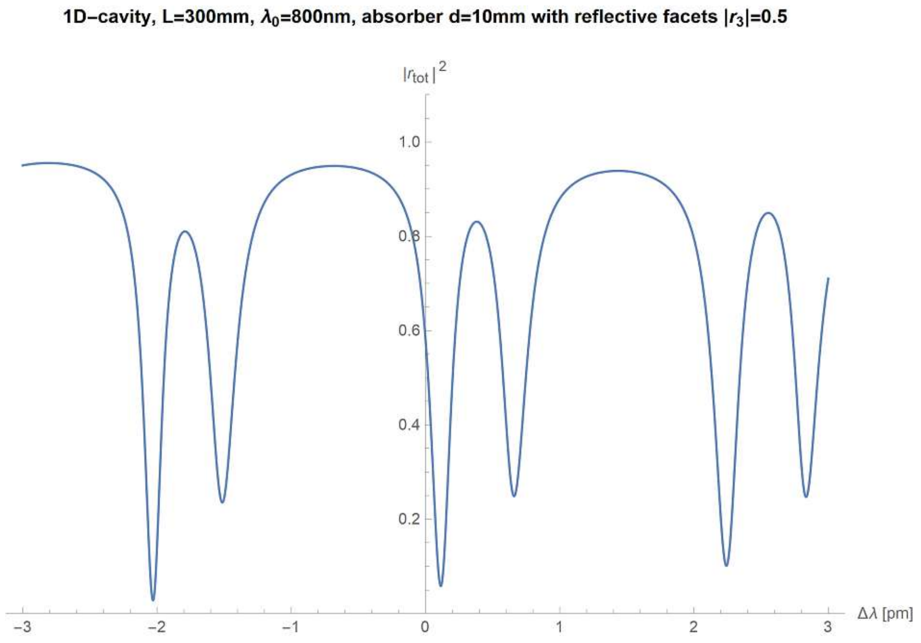

Figure 5.4.: Reflectance of a 1D-cavity with an $|r_1| = 0.7$ partially reflective mirror, $\lambda = 800\,\text{nm}$, and an absorber of thickness $d = 10\,\text{mm}$ in the middle position with partially reflective facets ($|r_3| = 0.5$), tuned to critical attenuation.

5.3. Simulating a 4f-cavity with transfer matrices and scattering matrices

5.3.1. Multiport scattering matrices

To be able to use the method described in section 5.2 for describing a realistic 4f-cavity, one needs to upgrade the 2×2 scattering matrices from section 5.2 to multiport scattering matrices, where each quadrant is not just a scalar reflection coefficient or transmission coefficient, but rather a whole reflection matrix or transmission matrix.

Hence, the scattering matrix of any optical component in the 4f-cavity has to be expressed as

$$\underline{\underline{\mathrm{S}}} = \begin{bmatrix} \underline{\underline{\mathrm{R}}} & \underline{\underline{\mathrm{T}}}' \\ \underline{\underline{\mathrm{T}}} & \underline{\underline{\mathrm{R}}}' \end{bmatrix} \tag{5.28}$$

In equation (5.28), $\underline{\underline{\mathrm{R}}}$ and $\underline{\underline{\mathrm{T}}}$ represent the reflection matrix and transmission matrix for wave fronts passing from left to right, whereas $\underline{\underline{\mathrm{R}}}'$ and $\underline{\underline{\mathrm{T}}}'$ represent the reflection matrix and transmission matrix for wave fronts passing from right to left.

In analogy to equation (5.2), the multiport scattering matrix for the left, partially reflective mirror can be written as

$$\underline{\underline{S}}_1 = \begin{bmatrix} \underline{\underline{R}}_1 & \underline{\underline{T}}_1 \\ \underline{\underline{T}}_1 & \underline{\underline{R}}_1 \end{bmatrix} \tag{5.29}$$

Given r_0 being the absolute value of the left mirror's reflection coefficient, so that $r_0 = |r_1|$; then based on equations (3.11) and (3.12) the sub-matrices $\underline{\underline{R}}_1$ and $\underline{\underline{T}}_1$ are given by the following expressions

$$\underline{\underline{R}}_1 = \mathbb{1}\left(-r_0^2 - ir_0\sqrt{1-r_0^2}\right) \tag{5.30}$$
$$\underline{\underline{T}}_1 = \mathbb{1}\left(1 - r_0^2 - ir_0\sqrt{1-r_0^2}\right) \tag{5.31}$$

The symbol $\mathbb{1}$ in equations (5.30) and (5.31) represents the unit-matrix. Similarly, we can write the multiport scattering matrix of the total reflective, perfect mirror M_2 as

$$\underline{\underline{S}}_2 = \begin{bmatrix} \underline{\underline{R}}_2 & \underline{\underline{T}}_2 \\ \underline{\underline{T}}_2 & \underline{\underline{R}}_2 \end{bmatrix} \tag{5.32}$$

For the total reflective, perfect mirror we can define

$$\underline{\underline{R}}_2 = \mathbb{1}e^{i\pi} = -\mathbb{1} \tag{5.33}$$
$$\underline{\underline{T}}_2 = \mathbb{O} \tag{5.34}$$

The symbol $\mathbb{O}$ in equation (5.34) represents the zero-matrix. However, as already explained in section 5.2.1, converting the scattering matrix from equation (5.32) with the substitutions 5.33 and 5.33 into the corresponding transfer matrix $\underline{\underline{S}}_2$ would directly lead to a singularity-problem. Again (as in section 5.2.1), it turns out that if we are only interested in the cavity's total reflection matrix, then we can safely replace the zero-value $\underline{\underline{T}}_2$ sub-matrix with the following diagonal $\underline{\underline{T}}_2$ sub-matrix:

$$\underline{\underline{T}}_2 = \mathbb{1}\varepsilon \tag{5.35}$$

In equation (5.35), ε can actually be any scalar value, because $\underline{\underline{T}}_2$ eventually cancels itself out in the final result we are interested in (i.e., in the cavity's reflection matrix). It seems sensible, though, to replace ε with a small, positive value in the numerical simulation as this then represents a mirror with *almost* no transmission.

To create a scattering matrix $\underline{\underline{S}}_p(z)$ representing any free-space propagation over a distance z between the lenses, or between the lenses and the absorber, we can just calculate the according transmission matrix $\underline{\underline{T}}_p(z)$, using the already established numerical simulation tools, conveniently encapsulated in the new function `transmision_matrix_prop` (see appendix C.27), and then build the following propagation scattering matrix:

$$\underline{\underline{S}}_p(z) = \begin{bmatrix} \mathbb{0} & \underline{\underline{T}}_p(z) \\ \underline{\underline{T}}_p(z) & \mathbb{0} \end{bmatrix} \tag{5.36}$$

The same method works for propagation within the absorber, with the only caveat that the propagation distance z corresponds to the (larger) optical distance $z_{opt} = z\ \mathrm{Re}(n)$ (with n being the complex refractive index).

Equivalently, to create a scattering-matrix $\underline{\underline{S}}_L$ representing a thin lens without reflection, the according transmission matrix $\underline{\underline{T}}_L$ can also be easily calculated by using already established numerical simulation tools, again encapsulated in a new convenience-function `transmision_matrix_lens` (see appendix C.28). The corresponding scattering matrix can then be constructed from the transmission matrix as:

$$\underline{\underline{S}}_L = \begin{bmatrix} \mathbb{0} & \underline{\underline{T}}_L \\ \underline{\underline{T}}_L & \mathbb{0} \end{bmatrix} \tag{5.37}$$

Finally, to simulate reflective facets, a scattering matrix just like the $\underline{\underline{S}}_1$-partially-reflective-mirror-matrix from equation (5.38) can be used:

$$\underline{\underline{S}}_3 = \begin{bmatrix} \underline{\underline{R}}_3 & \underline{\underline{T}}_3 \\ \underline{\underline{T}}_3 & \underline{\underline{R}}_3 \end{bmatrix} \tag{5.38}$$

Let $|r_3|$ be the absolute value of the facet's reflection coefficient, then the sub-matrices $\underline{\underline{R}}_3$ and $\underline{\underline{T}}_3$ can be written in analogy to equations (5.30) and (5.31) as

$$\underline{\underline{R}}_3 = \mathbb{1} \left(- |r_3|^2 - i\,|r_3| \sqrt{1 - |r_3|^2} \right) \tag{5.39}$$

$$\underline{\underline{T}}_3 = \mathbb{1} \left(1 - |r_3|^2 - i\,|r_3| \sqrt{1 - |r_3|^2} \right) \tag{5.40}$$

5.3.2. Converting multiport scattering matrices to transfer matrices

Frei et al. describe in [23] how multiport scattering matrices and multiport transfer matrices can be converted into each other. For converting a multiport scattering matrix into the corresponding transfer matrix, we assume the scattering matrix $\underline{\underline{S}}$ to consist of four sub-matrices $\underline{\underline{S}}_{11}$, $\underline{\underline{S}}_{12}$, $\underline{\underline{S}}_{21}$, and $\underline{\underline{S}}_{22}$ as shown in equation (5.41):

$$\underline{\underline{S}} = \begin{bmatrix} \underline{\underline{S}}_{11} & \underline{\underline{S}}_{12} \\ \underline{\underline{S}}_{21} & \underline{\underline{S}}_{22} \end{bmatrix} \tag{5.41}$$

According to [23, p. 2495], this scattering matrix $\underline{\underline{S}}$ can be converted into the corresponding transfer matrix $\underline{\underline{M}}\left(\underline{\underline{S}}\right)$ by means of the following conversion formula, which is encapsulated in function `convert_S_to_M` (see appendix C.25):

$$\underline{\underline{M}}\left(\underline{\underline{S}}\right) = \begin{bmatrix} \underline{\underline{S}}_{12} - \underline{\underline{S}}_{11}\underline{\underline{S}}_{21}^{-1}\underline{\underline{S}}_{22} & \underline{\underline{S}}_{11}\underline{\underline{S}}_{21}^{-1} \\ -\underline{\underline{S}}_{21}^{-1}\underline{\underline{S}}_{22} & \underline{\underline{S}}_{21}^{-1} \end{bmatrix} \tag{5.42}$$

Please note how equation (5.42) resembles equation (5.6), with the only difference that the scalar multiplication and division operations in equation (5.6) are converted into the equivalent matrix operations in equation (5.42).

Equation (5.42) allows us to convert all the scattering matrices from section 5.3.1 into their corresponding transfer matrices. Analogous to section 5.2.2, we can then, for example, express the total transfer matrix of a 4f-cavity with an absorber element of thickness d and complex refractive index n in the center position as follows:

$$\begin{aligned} \underline{\underline{M}}_{tot} = {} & \underline{\underline{M}}\left(\underline{\underline{S}}_1\right) \cdot \underline{\underline{M}}\left(\underline{\underline{S}}_p(f)\right) \cdot \underline{\underline{M}}\left(\underline{\underline{S}}_L\right) \cdot \underline{\underline{M}}\left(\underline{\underline{S}}_p\left(f - \frac{d}{2}\right)\right) \cdot \underline{\underline{M}}\left(\underline{\underline{S}}_p\left(\frac{d}{\mathrm{Re}(n)}; n\right)\right) \cdot \\ & \underline{\underline{M}}\left(\underline{\underline{S}}_p\left(f - \frac{d}{2}\right)\right) \cdot \underline{\underline{M}}\left(\underline{\underline{S}}_L\right) \cdot \underline{\underline{M}}\left(\underline{\underline{S}}_p(f)\right) \cdot \underline{\underline{M}}\left(\underline{\underline{S}}_2\right) \end{aligned} \tag{5.43}$$

The meaning of all scattering matrices in equation (5.43) is explained in section 5.3.1.

To additionally simulate reflective facets of the absorber, we can enhance equation (5.43) with two $\underline{\underline{M}}\left(\underline{\underline{S}}_3\right)$ terms, and get equation (5.44):

$$\begin{aligned}\underline{\underline{M}}_{tot} = {} & \underline{\underline{M}}\left(\underline{\underline{S}}_1\right) \cdot \underline{\underline{M}}\left(\underline{\underline{S}}_p(f)\right) \cdot \underline{\underline{M}}\left(\underline{\underline{S}}_L\right) \cdot \underline{\underline{M}}\left(\underline{\underline{S}}_p\left(f - \frac{d\,\mathrm{Re}(n)}{2}\right)\right) \cdot \underline{\underline{M}}\left(\underline{\underline{S}}_3\right) \cdot \underline{\underline{M}}\left(\underline{\underline{S}}_p\left(d;n\right)\right) \cdot \\ & \underline{\underline{M}}\left(\underline{\underline{S}}_3\right) \cdot \underline{\underline{M}}\left(\underline{\underline{S}}_p\left(f - \frac{d\,\mathrm{Re}(n)}{2}\right)\right) \cdot \underline{\underline{M}}\left(\underline{\underline{S}}_L\right) \cdot \underline{\underline{M}}\left(\underline{\underline{S}}_p(f)\right) \cdot \underline{\underline{M}}\left(\underline{\underline{S}}_2\right)\end{aligned} \tag{5.44}$$

5.3.3. Back-conversion of the cavity's total transfer matrix into the corresponding scattering matrix

For the final conversion of $\underline{\underline{M}}_{tot}$ into $\underline{\underline{S}}(\underline{\underline{M}}_{tot})$, we define the following four sub-matrices $\underline{\underline{M}}_{11}$, $\underline{\underline{M}}_{12}$, $\underline{\underline{M}}_{21}$, and $\underline{\underline{M}}_{22}$ as shown in equation (5.45):

$$\underline{\underline{M}}_{tot} = \begin{bmatrix} \underline{\underline{M}}_{11} & \underline{\underline{M}}_{12} \\ \underline{\underline{M}}_{21} & \underline{\underline{M}}_{22} \end{bmatrix} \tag{5.45}$$

According to [23, p. 2495], this transfer matrix $\underline{\underline{M}}_{tot}$ can be converted into the corresponding scattering matrix $\underline{\underline{S}}\left(\underline{\underline{M}}_{tot}\right)$ by means of the following conversion formula:

$$\underline{\underline{S}}\left(\underline{\underline{M}}_{tot}\right) = \begin{bmatrix} \underline{\underline{M}}_{12}\underline{\underline{M}}_{22}^{-1} & \underline{\underline{M}}_{11} - \underline{\underline{M}}_{12}\underline{\underline{M}}_{22}^{-1}\underline{\underline{M}}_{21} \\ \underline{\underline{M}}_{22}^{-1} & -\underline{\underline{M}}_{22}^{-1}\underline{\underline{M}}_{21} \end{bmatrix} \tag{5.46}$$

Having a look at equation (5.28), we see that the cavity's total reflection matrix $\underline{\underline{R}}$ is just the top-left sub-matrix in equation (5.46). Therefore, to just calculate the final reflection matrix, we can reduce equation (5.46) for our purpose to

$$\underline{\underline{R}}\left(\underline{\underline{M}}_{tot}\right) = \underline{\underline{M}}_{12}\underline{\underline{M}}_{22}^{-1} \tag{5.47}$$

The conversion formula in equation (5.47) is encapsulated in function `convert_M_to_R` (see appendix C.26).

5.3.4. Software implementation

The following subroutines have been additionally implemented to support the simulations in this chapter:

Creating a multiport scattering matrix

- **Function:** `create_S_matrix(gpu, R, T)`
- **Description:** Creates a scattering matrix based on the input matrices `R` and `T`, which represent the reflection-sub-matrix and the transmission-sub-matrix, respectively. The `R`-matrix goes in the top-left and the bottom-right quadrants of the resulting scattering matrix; and the `T`-matrix goes into the top-right and bottom-left quadrants of the resulting scattering matrix.
- **Documentation:** appendix C.24.

Converting a scattering matrix into the corresponding transfer matrix

- **Function:** `covert_S_to_M(S)`
- **Description:** Converts the input scattering matrix `S` into the corresponding transfer matrix `M`, using the conversion method described in section 5.3.2.
- **Documentation:** appendix C.25.

Converting a transfer matrix into a reflection matrix

- **Function:** `covert_M_to_R(M)`
- **Description:** Converts the input transfer matrix `M` into the top-left quadrant of the corresponding scattering matrix, which is the reflection matrix `R`, based on the theory presented in section 5.3.3.
- **Documentation:** appendix C.26.

Generating transmission matrix representing propagation through free space or material

- **Function:** `transmision_matrix_prop(gpu, TF, ax, z, lambda, n, modes)`
- **Description:** Creates a transmission matrix `T` representing propagation through free space or material (with refractive index n) over a distance `z`.
- **Documentation:** appendix C.27.

Generating a thin lens transmission matrix

- **Function:** `transmission_matrix_lens(gpu, ax, lambda, pupil, NA, f, lens_type, modes)`
- **Description:** Creates a transmission matrix `T` representing a thin lens.
- **Documentation:** appendix C.28.

Downscale transmission or reflection matrix from lager grid size to smaller grid size

- **Function:** `downscale_TR_matrix(gpu, TR, ax_small, ax_large, modes_large`
- **Description:** Downscales a transmission or reflection matrix `TR` which corresponds to the larger axis coordinates `ax_large` of the guarding grid to a smaller transmission or reflection matrix corresponding to the smaller observation grid coordinates `ax_small`.
- **Documentation:** appendix C.29.

Proof-of-concept main program CPA_sim_009

The main program `CPA_sim_009`, documented in appendix E.9, demonstrates how the concept of simulating a cavity-CPA by means of scattering matrices and transfer matrices can be extended from a one-dimensional toy-model (section 5.2) to a realistic model of a 4f-cavity by calculating the reflection matrix of a 4f-cavity-CPA with a non-reflecting absorber in the center position.

Simulation main program CPA_sim_010

The main program `CPA_sim_010`, documented in appendix E.10, uses the method of scattering and transfer matrices to simulate a 4f-cavity with an absorber in the middle position. In contrast to the previous proof-of-concept program `CPA_sim_009`, it allows to run several simulation rounds with different simulation parameters (namely, absorber thickness and reflectivity of the absorber facets).

The maximum, average, and minimum eigenvalues of the 4f cavity's reflection-matrix are calculated for various wavelengths around a critical wavelength, and stored into a single file per simulation round. In addition to that, separate files are generated at each calculated wavelength, containing a full eigenmode decomposition at the given wavelength with the simulation parameters of the respective round.

5.3.5. Simulation results

Attenuated 4f-cavity without reflections (proof of concept)

Using the main program `CPA_sim009` (documented in appendix E.9), we first simulate a simple 4f-cavity with a reflection-free absorber of thickness $d = 10\,\mathrm{mm}$ in the middle position to proof the concept of the scattering/transfer-matrix approach. The program creates all required scattering matrices as described in section 5.3.1, converts them into the corresponding transfer matrices using the method described in section 5.3.2, and then calculates the total transfer matrix according to equation (5.43).

Finally, the resulting total transfer matrix is back-converted into the top-left quadrant of the corresponding scattering matrix (see section 5.3.3), which happens to be the total cavity's reflection matrix. This total reflection matrix is then displayed graphically with element-wisely squared absolute values. In addition, to visualize the optical properties of the simulated 4f-cavity, all transmission matrices (for the lenses and the propagation distances), which are created to build the scattering matrices, are used to create a total transmission matrix simulating a single (one-way) trip through the 4f-cavity, and the effect of this total transmission matrix on a simple test-image (an off-center letter T) is visualized.

These are the simulation parameters:

- **lenses:** $f = 100\,\mathrm{mm}$, perfect aspherical
- **propagation simulation:** Rayleigh-Sommerfeld
- **wavelength:** $\lambda_0 = 785\,\mathrm{nm}$
- **absorber:** $d = 10\,\mathrm{mm}$ in center position, critical absorption
- **left mirror reflectivity:** $r_0^2 = 0.8$
- **observation plane:** $1.4\,\mathrm{mm} \times 1.4\,\mathrm{mm}$

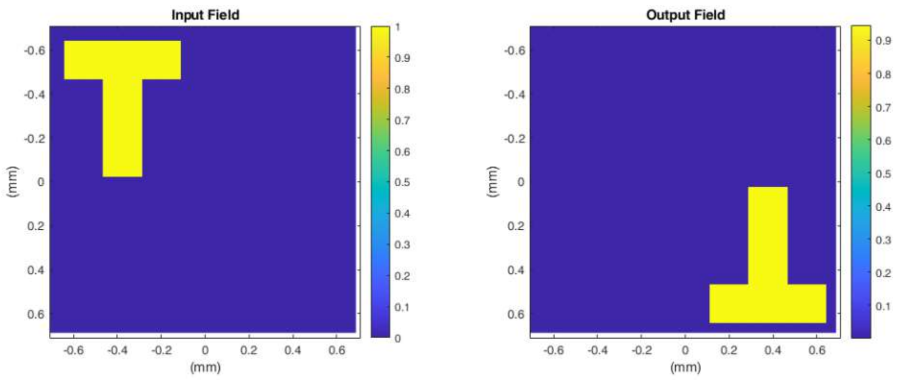

Figure 5.5.: All generated transmission matrices (for the lenses and the propagation distances) multiplied together in the right order simulate a one-way trip through the 4f-cavity.

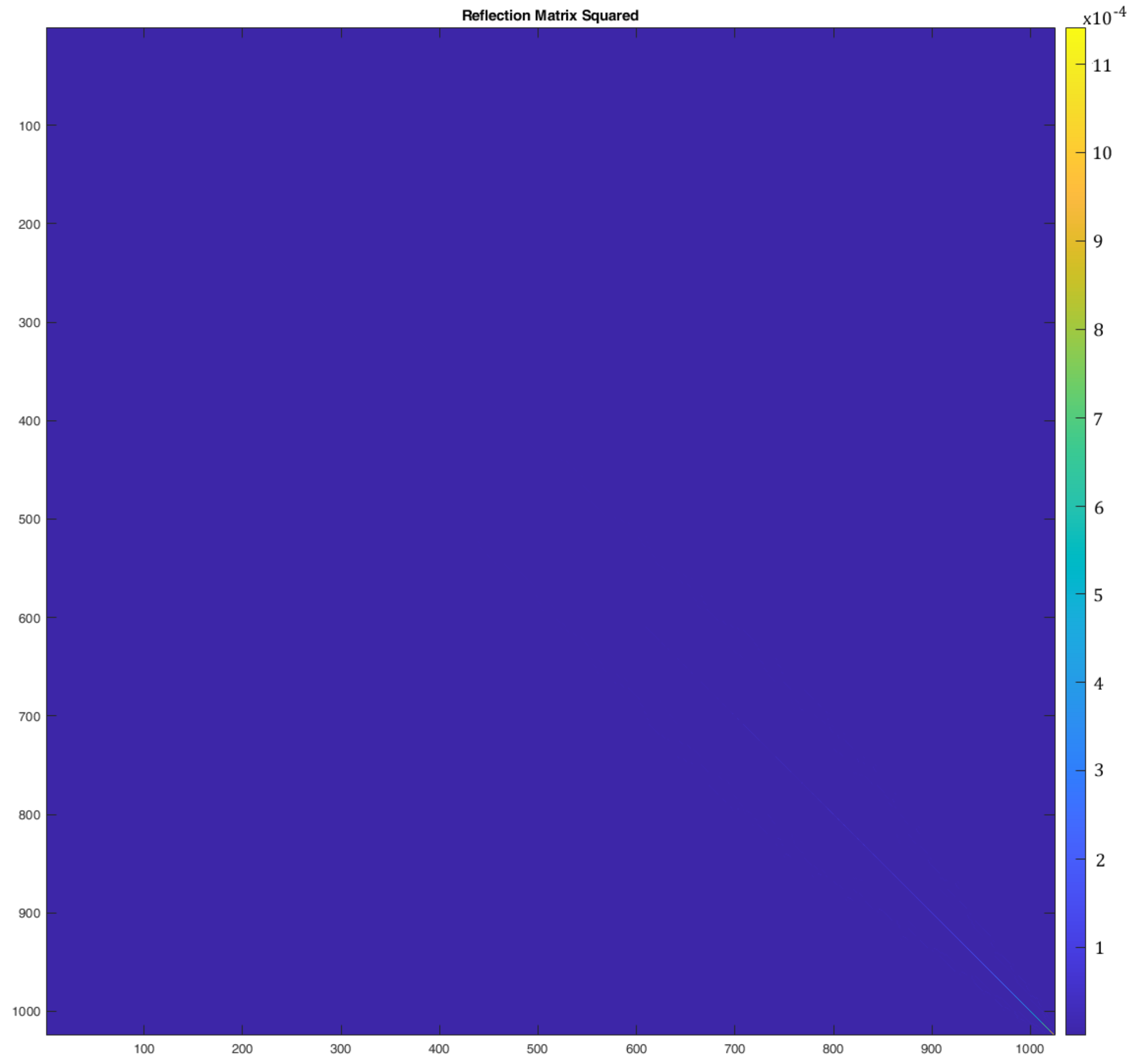

Figure 5.6.: The reflection matrix of a 4f-cavity-CPA with an absorber element in center (tuned to critical absorption), simulated using the scattering/transfer-matrix-method (the plot shows squared absolute values on a **linear color scale**). Parameters: $f = 100\,\mathrm{mm}$, $\lambda_0 = 785\,\mathrm{nm}$, absorber $d = 10\,\mathrm{mm}$. Observation plane $1.4\,\mathrm{mm} \times 1.4\,\mathrm{mm}$. Maximum modes: $n_x = n_y = 16$. Lenses: aspherical perfect lenses. Propagation simulation: Rayleigh-Sommerfeld.

Figure 5.5 shows the image-flip expected for a single-trip through a 4f-cavity; and also the reflection matrix depicted in figure 5.6 has the well-known form, similar to the reflection matrices calculated previously (i.e., uniformly low reflectance for most of the modes, and a slight increase of reflectance over the most reflective 100-or-so modes).

Figure 5.7 shows the same reflection matrix as depicted in figure 5.6, but this time with a logarithmic color scale. This representation shows that the diagonal values of the matrix go down to almost 10^{-7} for the least reflective eigenmodes, and it also reveals some "side-band"-noise around the diagonal entries.

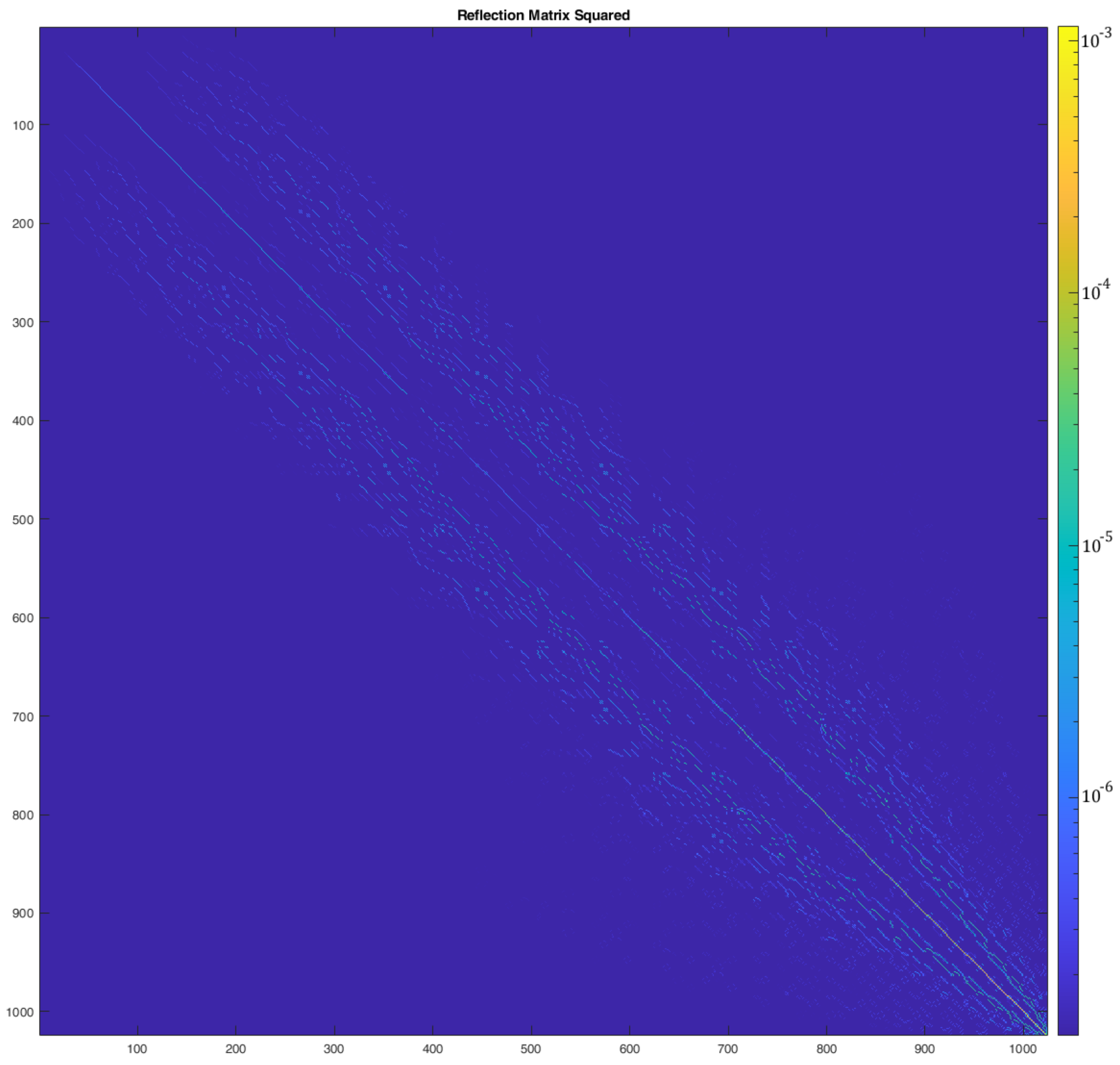

Figure 5.7.: The reflection matrix of a 4f-cavity-CPA with an absorber element in center (tuned to critical absorption), simulated using the scattering/transfer-matrix-method (the plot shows squared absolute values on a **logarithmic color scale**). Parameters: $f = 100\,\mathrm{mm}$, $\lambda_0 = 785\,\mathrm{nm}$, absorber $d = 10\,\mathrm{mm}$. Observation plane $1.4\,\mathrm{mm} \times 1.4\,\mathrm{mm}$. Maximum modes: $n_x = n_y = 16$. Lenses: aspherical perfect lenses. Propagation simulation: Rayleigh-Sommerfeld.

Attenuated 4f-cavity with reflections

Using the main program `CPA_sim010` (documented in appendix E.10), we simulate a 4f-cavity with an absorber with reflective facets of thickness $d = 10\,\text{mm}$, or $d = 1\,\text{mm}$, respectively.

Simulation parameters:

- **lenses:** $f = 100\,\text{mm}$, perfect aspherical
- **propagation simulation:** Rayleigh-Sommerfeld
- **wavelength:** $\lambda_0 = 785\,\text{nm}$
- **absorber:** center position, critical absorption, thickness $d = 10\,\text{mm}$, or $1\,\text{mm}$
- **absorber facets' reflectance:** $r_3^2 = 10^{-4}$, 10^{-3}, 10^{-2}, 0.5.
- **left mirror reflectivity:** $r_0^2 = 0.8$
- **observation plane:** $1.4\,\text{mm} \times 1.4\,\text{mm}$

Simulation results:

In figures 5.8 and 5.9, the squared absolute reflection-matrix eigenvalues are plotted in ascending order for different reflectances for a $d = 1\,\text{mm}$ and a $d = 10\,\text{mm}$ absorber, respectively. Each curve in the two diagrams is plotted for the wavelength where the average reflectance over all eigenmodes becomes minimal. The following figures 5.10 to 5.13 show the average and minimal squared reflection-matrix eigenvalues as a function of the wavelength for a $d = 1\,\text{mm}$ and a $d = 10\,\text{mm}$ absorber, respectively.

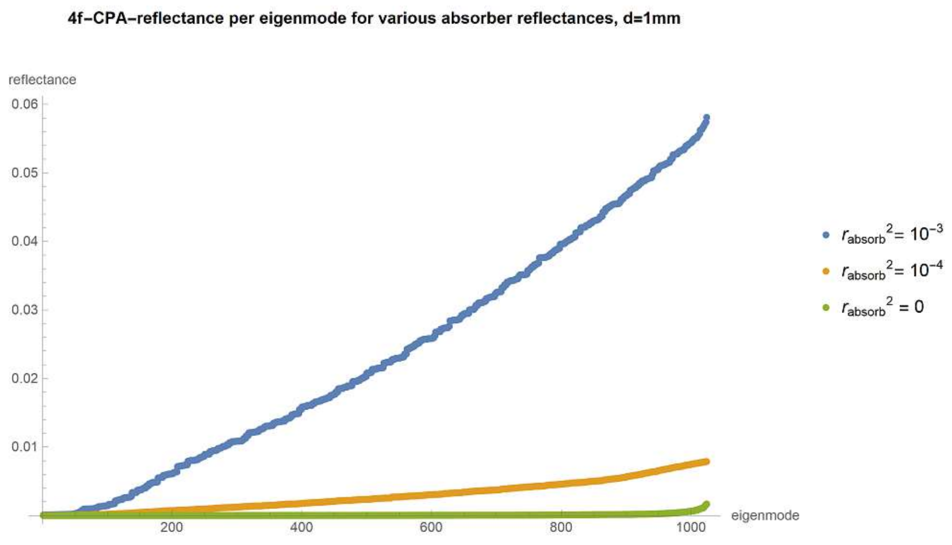

Figure 5.8.: Squared absolute reflection-matrix eigenvalues of a 4f-cavity-CPA with a reflective absorber element in the center position plotted in ascending order for different absorber reflectances. Parameters: $f = 100\,\mathrm{mm}, r_0^2 = 0.8,\ \lambda_0 = 785\,\mathrm{nm},$ absorber $d = 1\,\mathrm{mm}.$

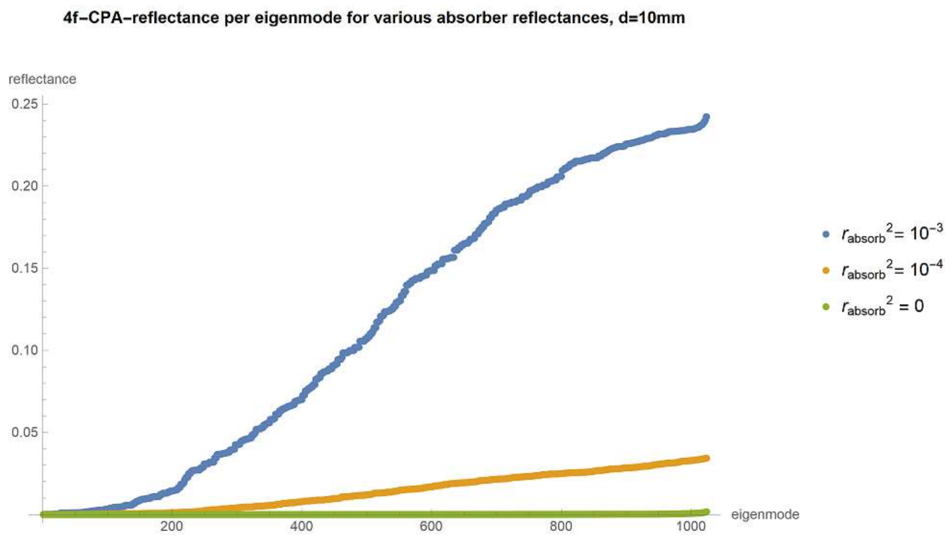

Figure 5.9.: Squared absolute reflection-matrix eigenvalues of a 4f-cavity-CPA with a reflective absorber element in the center position plotted in ascending order for different absorber reflectances. Parameters: $f = 100\,\mathrm{mm}, r_0^2 = 0.8,\ \lambda_0 = 785\,\mathrm{nm},$ absorber $d = 10\,\mathrm{mm}.$

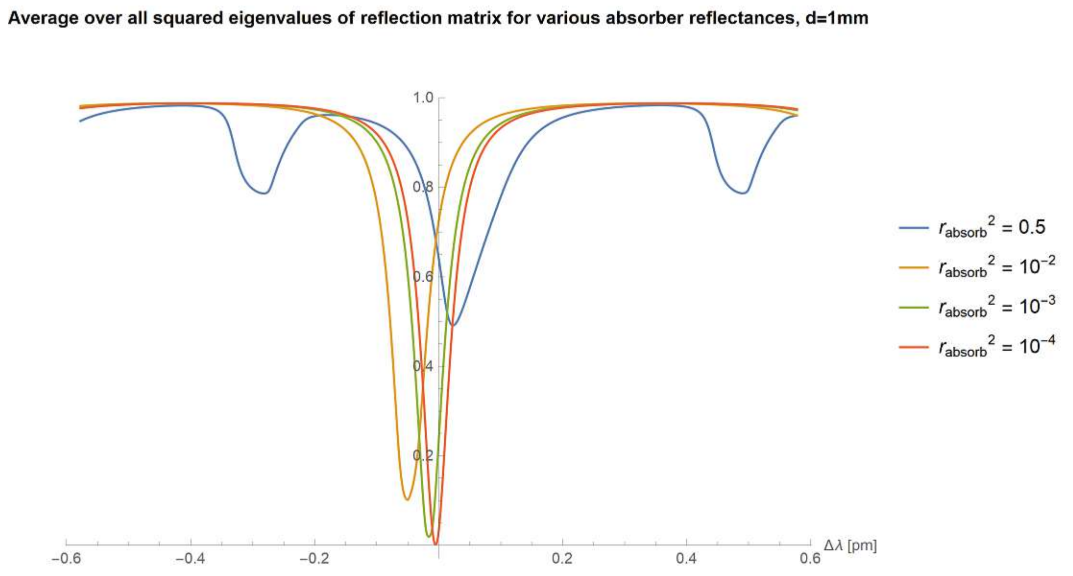

Figure 5.10.: Average over all squared absolute reflection-matrix eigenvalues of a 4f-cavity-CPA with a reflective absorber element in the middle position, plotted as a function of the wavelength around the resonance point of an optimally configured 4f-cavity. Parameters: $f = 100\,\mathrm{mm}, r_0^2 = 0.8,\ \lambda_0 = 785\,\mathrm{nm}$, absorber $d = 1\,\mathrm{mm}$.

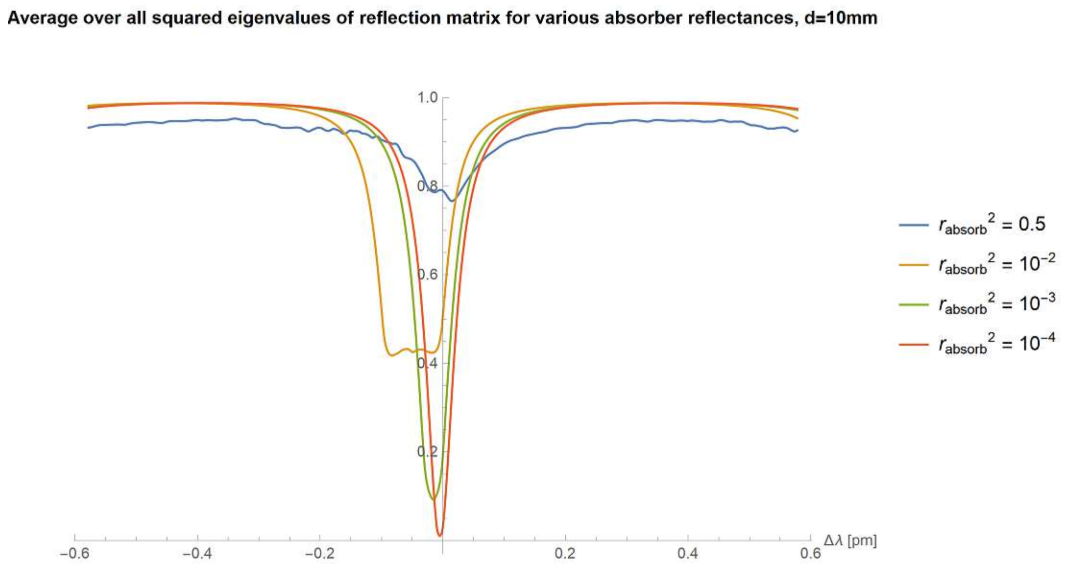

Figure 5.11.: Average over all squared absolute reflection-matrix eigenvalues of a 4f-cavity-CPA with a reflective absorber element in the center position, plotted as a function of the wavelength around the resonance point of an optimally configured 4f-cavity. Parameters: $f = 100\,\mathrm{mm}, r_0^2 = 0.8,\ \lambda_0 = 785\,\mathrm{nm}$, absorber $d = 10\,\mathrm{mm}$.

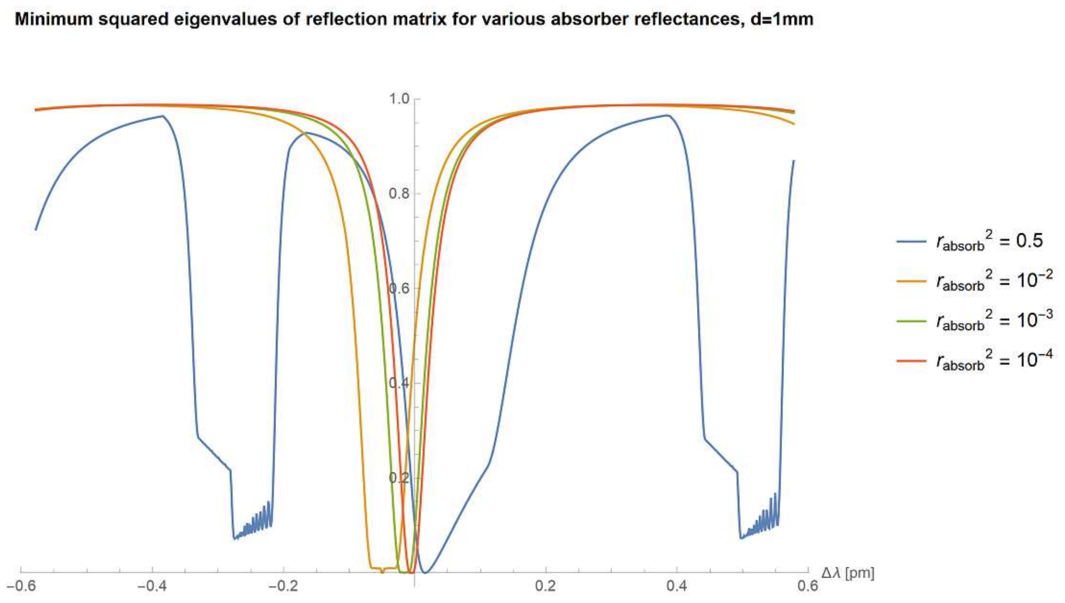

Figure 5.12.: Minimum squared absolute reflection-matrix eigenvalues of a 4f-cavity-CPA with a reflective absorber element in the middle position, plotted as a function of the wavelength around the resonance point of an optimally configured 4f-cavity. Parameters: $f = 100\,\text{mm}, r_0^2 = 0.8,\ \lambda_0 = 785\,\text{nm}$, absorber $d = 1\,\text{mm}$.

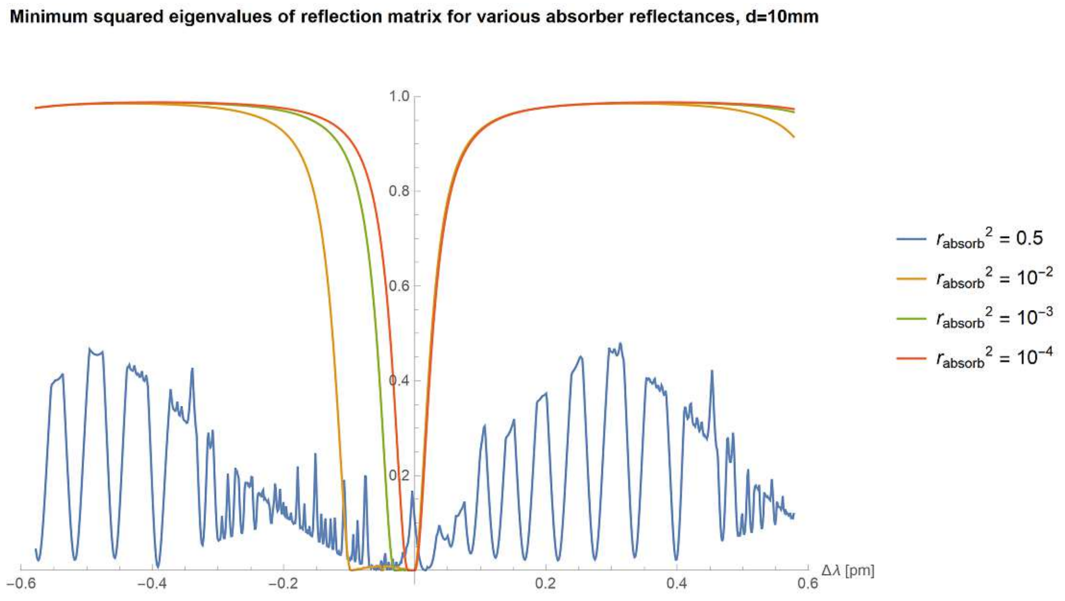

Figure 5.13.: Minimum squared absolute reflection-matrix eigenvalues of a 4f-cavity-CPA with a reflective absorber element in the middle position, plotted as a function of the wavelength around the resonance point of an optimally configured 4f-cavity. Parameters: $f = 100\,\text{mm}, r_0^2 = 0.8,\ \lambda_0 = 785\,\text{nm}$, absorber $d = 10\,\text{mm}$.

Discussion

If the effective reflectance of the absorber's facets becomes larger than 10^{-4}, the CPA-effect strongly deteriorates. In an experimental implementation it seems advisable to insert the absorbing element at an angle, so that all potential residual reflections are directed outside the optical path. The simulations also show that the thinner absorber with a thickness $d = 1\,\mathrm{mm}$ causes less deterioration of the CPA-effect than the thicker $d = 10\,\mathrm{mm}$ absober.

6. Conclusion and Outlook

Currently existing implementations of a Coherent Perfect Absorber (CPA) are in general two-port devices, where two incident beams must be carefully controlled with regard to the exact shape of their transverse mode profile, and also with regard to the mutual phase relations. Because of this, these implementations are not able to absorb just any incoming coherent light-beam of matching wavelength, but only carefully engineered wavefronts.

In this thesis, we have presented an entirely new type of "universal" CPA, consisting of a plane-parallel 4f-cavity with critical absorption. The new type of CPA has just one single port to absorb the incident light wave, which does not need any special wavefront-shaping anymore to be absorbed. Rather, the 4f-cavity-CPA presented in this thesis is capable to absorb any superposition of a very large number of transverse modes, which is a significant advancement compared to currently existing CPAs.

The new concept of a 4f-cavity CPA was confirmed by means of theoretical derivations and corresponding computer simulations. Further, we have shown by simulating the effects of certain deviations from optimal parameter values that implementing such a 4f-cavity CPA should be well within the current experimental possibilities.

Therefore, we hope that the experimental implementation of a 4f-cavity CPA, which is already on its way at the time of completion of this thesis, will soon prove the efficiency of this new concept, which promises to be a big step forward towards a simple, massively parallel and universal single-port CPA implementation. In future research we hope to be able to further improve the CPA efficiency, and also to explore how a 4f-cavity CPA might be used for sub-diffraction focusing.

Appendix

A. Derivations

A.1. Derivation of the Helmholtz-equation (2.9)

By letting $\vec{\nabla}\times$ act on the Maxwell equation (2.7) from the left side, we get:

$$
\begin{aligned}
&\vec{\nabla}\times\vec{\nabla}\times\vec{E} = -\vec{\nabla}\times\frac{\partial\vec{B}}{\partial t}\\
&\vec{\nabla}\times\vec{\nabla}\times\vec{E} = -\frac{\partial}{\partial t}\left(\vec{\nabla}\times\vec{B}\right) \overset{(2.8)}{\Longrightarrow}\\
&\vec{\nabla}\times\vec{\nabla}\times\vec{E} = -\mu_0\epsilon_r\epsilon_0\frac{\partial^2\vec{E}}{\partial t^2}\\
&\vec{\nabla}\left(\vec{\nabla}\cdot\vec{E}\right) - \vec{\nabla}^2\vec{E} = -\mu_0\epsilon_r\epsilon_0\frac{\partial^2\vec{E}}{\partial t^2} \overset{(2.5)}{\Longrightarrow}\\
&\vec{\nabla}^2\vec{E} = \mu_0\epsilon_r\epsilon_0\frac{\partial^2\vec{E}}{\partial t^2} && \Big|\mu_0\epsilon_0 = \frac{1}{c^2}\\
&\vec{\nabla}^2\vec{E} = \frac{\epsilon_r}{c^2}\frac{\partial^2\vec{E}}{\partial t^2} && \Big|\epsilon_r = \frac{\epsilon_r\epsilon_0}{\epsilon_0} = \frac{\epsilon}{\epsilon_0} = n^2\\
&\vec{\nabla}^2\vec{E} = \frac{n^2}{c^2}\frac{\partial^2\vec{E}}{\partial t^2}\\
&\underline{\vec{\nabla}^2\vec{E} - \frac{n^2}{c^2}\frac{\partial^2\vec{E}}{\partial t^2} = 0} && (1)
\end{aligned}
$$

In equation (1), which is the wave equation for the electric field, the symbol c stands for the speed of light, and n is the refractive index.

By letting $\vec{\nabla}\times$ act on the Maxwell equation (2.8) from the left side, we get a wave equation equivalent to equation (1), but this time for the magnetic field:

$$
\begin{aligned}
&\vec{\nabla}\times\vec{\nabla}\times\vec{B} = \mu_0\epsilon_r\epsilon_0\vec{\nabla}\times\frac{\partial\vec{E}}{\partial t} \\
&\vec{\nabla}\times\vec{\nabla}\times\vec{B} = \mu_0\epsilon_r\epsilon_0\frac{\partial}{\partial t}\left(\vec{\nabla}\times\vec{E}\right) \overset{(2.7)}{\Longrightarrow} \\
&\vec{\nabla}\times\vec{\nabla}\times\vec{B} = -\mu_0\epsilon_r\epsilon_0\frac{\partial^2\vec{B}}{\partial t^2} \\
&\vec{\nabla}\left(\vec{\nabla}\cdot\vec{B}\right) - \vec{\nabla}^2\vec{B} = -\mu_0\epsilon_r\epsilon_0\frac{\partial^2\vec{B}}{\partial t^2} \overset{(2.6)}{\Longrightarrow} \\
&\vec{\nabla}^2\vec{B} = \mu_0\epsilon_r\epsilon_0\frac{\partial^2\vec{B}}{\partial t^2} && \Big|\mu_0\epsilon_0 = \frac{1}{c^2} \\
&\vec{\nabla}^2\vec{B} = \frac{\epsilon_r}{c^2}\frac{\partial^2\vec{B}}{\partial t^2} && \Big|\epsilon_r = \frac{\epsilon_r\epsilon_0}{\epsilon_0} = \frac{\epsilon}{\epsilon_0} = n^2 \\
&\underline{\vec{\nabla}^2\vec{B} - \frac{n^2}{c^2}\frac{\partial^2\vec{B}}{\partial t^2} = 0} && (2)
\end{aligned}
$$

By re-writing equations (1) and (2) in Einstein notation, we get:

$$\partial_j\partial_j E_i - \frac{n^2}{c^2}\partial_t^2 E_i = 0 \tag{3}$$

$$\partial_j\partial_j B_i - \frac{n^2}{c^2}\partial_t^2 B_i = 0 \tag{4}$$

This makes it obvious that the vectorial Laplacian in equation (1) and equation (2) actually generates six uncoupled, identical equations of the form

$$\underline{\vec{\nabla}^2 U(\vec{r},t) - \frac{n^2}{c^2}\frac{\partial^2}{\partial t^2}U(\vec{r},t) = 0} \tag{5}$$

where the scalar function $U(\vec{r},t)$ stands for any of the E_i or B_i components.

The components of the $\vec{E}$ and $\vec{B}$ field can be decomposed into fields with harmonic time dependencies, so that we can assume the following separation ansatz:

$$U(\vec{r},t) = U(\vec{r})e^{-i\omega t} \tag{6}$$

Following the derivation sketch in [24, p. 3], we insert equation (6) into equation (5), and get:

$$\begin{aligned}
&\vec{\nabla}^2 \left(U(\vec{r})e^{-i\omega t}\right) - \frac{n^2}{c^2}\frac{\partial^2}{\partial t^2}\left(U(\vec{r})e^{-i\omega t}\right) = 0 \\
&\left(\vec{\nabla}^2 U(\vec{r})\right) e^{-i\omega t} - \frac{n^2}{c^2} U(\vec{r}) \frac{\partial^2}{\partial t^2} e^{-i\omega t} = 0 \\
&\left(\vec{\nabla}^2 U(\vec{r})\right) \cancel{e^{-i\omega t}} + \frac{n^2}{c^2}\omega^2 U(\vec{r}) \cancel{e^{-i\omega t}} = 0 \qquad \Big| \frac{\omega^2}{c^2} = k^2 \\
&\underline{\vec{\nabla}^2 U(\vec{r}) + n^2 k^2 U(\vec{r}) = 0} \qquad (7)
\end{aligned}$$

This is the Helmholtz equation, which will be the basis for the scalar propagation mechanism. In empty space, $n = 1$, and equation (7) becomes

$$\underline{\vec{\nabla}^2 U(\vec{r}) + k^2 U(\vec{r}) = 0} \qquad (2.9)$$

A.2. Derivation of differential equation (2.12)

This is how inserting equation (2.11) into equation (2.9) results in equation (2.12):

$$\cancel{\frac{1}{(2\pi)^2}}\vec{\nabla}^2 \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} A(k_x,k_y;z)\ e^{i(k_x x+k_y y)}\,\mathrm{d}k_x\,\mathrm{d}k_y\ +$$
$$\cancel{\frac{1}{(2\pi)^2}}k^2 \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} A(k_x,k_y;z)\ e^{i(k_x x+k_y y)}\,\mathrm{d}k_x\,\mathrm{d}k_y = 0$$

$$\left(\frac{\partial^2}{\partial x^2}+\frac{\partial^2}{\partial y^2}+\frac{\partial^2}{\partial z^2}\right)\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} A(k_x,k_y;z)\ e^{ik_x x}e^{ik_y y}\,\mathrm{d}k_x\,\mathrm{d}k_y\ +$$
$$k^2\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} A(k_x,k_y;z)\ e^{ik_x x}e^{ik_y y}\,\mathrm{d}k_x\,\mathrm{d}k_y = 0$$

$$\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\frac{\partial^2}{\partial x^2}\left(A(k_x,k_y;z)\ e^{ik_x x}e^{ik_y y}\right)\mathrm{d}k_x\,\mathrm{d}k_y\ +$$
$$\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\frac{\partial^2}{\partial y^2}\left(A(k_x,k_y;z)\ e^{ik_x x}e^{ik_y y}\right)\mathrm{d}k_x\,\mathrm{d}k_y\ +$$
$$\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\frac{\partial^2}{\partial z^2}\left(A(k_x,k_y;z)\ e^{ik_x x}e^{ik_y y}\right)\mathrm{d}k_x\,\mathrm{d}k_y\ +$$
$$\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} A(k_x,k_y;z)k^2\ e^{ik_x x}e^{ik_y y}\,\mathrm{d}k_x\,\mathrm{d}k_y = 0$$

$$\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} A(k_x,k_y;z)\ (-1)k_x^2\ e^{ik_x x}e^{ik_y y}\,\mathrm{d}k_x\,\mathrm{d}k_y\ +$$
$$\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} A(k_x,k_y;z)\ (-1)k_y^2\ e^{ik_x x}e^{ik_y y}\,\mathrm{d}k_x\,\mathrm{d}k_y\ +$$
$$\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\frac{\partial^2}{\partial z^2}A(k_x,k_y;z)\ e^{ik_x x}e^{ik_y y}\,\mathrm{d}k_x\,\mathrm{d}k_y\ +$$
$$\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} A(k_x,k_y;z)\ k^2\ e^{ik_x x}e^{ik_y y}\,\mathrm{d}k_x\,\mathrm{d}k_y = 0$$

$$\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\Bigg(\frac{\partial^2}{\partial z^2}A(k_x,k_y;z)\ e^{ik_x x}e^{ik_y y} + A(k_x,k_y;z)\ k^2\ e^{ik_x x}e^{ik_y y}$$
$$-A(k_x,k_y;z)\ k_x^2\ e^{ik_x x}e^{ik_y y} - A(k_x,k_y;z)\ k_y^2\ e^{ik_x x}e^{ik_y y}\Bigg)\,\mathrm{d}k_x\,\mathrm{d}k_y = 0$$

$$\left(\frac{\partial^2}{\partial z^2}A + Ak^2 - Ak_x^2 - Ak_y^2\right) \cancel{e^{ik_x x} e^{ik_y y}} = 0$$

$$\underline{\underline{\frac{\partial^2 A}{\partial z^2} + \left(k^2 - k_x^2 - k_y^2\right) A = 0}} \tag{2.12}$$

A.3. Derivation of equations (2.25) and (2.27)

Looking at image (a) of figure 2.3, one can derive the thickness function of the first lens segment $\Delta_1(x,y)$ as follows:

$$\Delta_1(x,y) = \Delta_{01} - \delta_1 \qquad |\delta_1 = R_1 - a_1$$

$$\Delta_1(x,y) = \Delta_{01} - R_1 + a_1 \qquad \left|a_1 = \sqrt{R_1^2 - r^2}\right.$$

$$\Delta_1(x,y) = \Delta_{01} - R_1 + \sqrt{R_1^2 - r^2} \qquad \left|r^2 = x^2 + y^2\right.$$

$$\Delta_1(x,y) = \Delta_{01} - R_1 + \sqrt{R_1^2 - x^2 - y^2}$$

$$\Delta_1(x,y) = \Delta_{01} - R_1 + R_1\sqrt{1 - \frac{x^2 + y^2}{R_1^2}}$$

$$\underline{\underline{\Delta_1(x,y) = \Delta_{01} - R_1\left(1 - \sqrt{1 - \frac{x^2 + y^2}{R_1^2}}\right)}} \tag{2.25}$$

The derivation of the thickness function of the third lens segment Δ_3 as depicted in image (c) of figure 2.3 is equivalent to the derivation of Δ_1, with the only difference that the radii of convex and concave curvature must have different signs.

In accordance with [11] we adopt the sign convention that as rays travel from left to right, each convex surface encountered is taken to have a positive radius of curvature, while each concave surface is taken to have a negative radius of curvature. Thus in figure 2.3 the radius of curvature of the left-hand surface of the lens is a *positive* number R_1, while the radius of curvature of the right-hand surface is a *negative* number R_2. This leads to the following derivation:

$$\begin{aligned}
\Delta_3(x,y) &= \Delta_{03} - \delta_2 && |\delta_2 = -R_2 - a_2 \\
\Delta_3(x,y) &= \Delta_{03} + R_2 + a_1 && \Big|a_2 = \sqrt{(-R_2)^2 - r^2} \\
\Delta_3(x,y) &= \Delta_{03} + R_2 + \sqrt{(-R_2)^2 - r^2} && \Big|r^2 = x^2 + y^2 \\
\Delta_3(x,y) &= \Delta_{03} + R_2 + \sqrt{(-R_2)^2 - x^2 - y^2} \\
\Delta_3(x,y) &= \Delta_{03} + R_2 - R_2\sqrt{1 - \frac{x^2 + y^2}{(-R_2)^2}} \\
\Delta_3(x,y) &= \Delta_{03} + R_2\left(1 - \sqrt{1 - \frac{x^2 + y^2}{R_2^2}}\right) && (2.27)
\end{aligned}$$

A.4. Derivation of equation (2.29)

If we assume that $x \ll R_1, R_2$ and $y \ll R_1, R_2$ (paraxial approximation), then $\sqrt{1 - \frac{x^2+y^2}{R^2}} \approx 1 - \frac{x^2-y^2}{2R^2}$. Inserting this into equation (2.28) results in

$$\begin{aligned}
\Delta(x,y) &= \Delta_0 - R_1\left(1 - \left(1 - \frac{x^2 - y^2}{2R_1^2}\right)\right) + R_2\left(1 - \left(1 - \frac{x^2 - y^2}{2R_2^2}\right)\right) \\
\Delta(x,y) &= \Delta_0 - R_1\left(\cancel{1 - 1} + \frac{x^2 - y^2}{2R_1^2}\right) + R_2\left(\cancel{1 - 1} + \frac{x^2 - y^2}{2R_2^2}\right) \\
\Delta(x,y) &= \Delta_0 - \frac{x^2 - y^2}{2R_1} + \frac{x^2 - y^2}{2R_2} \\
\Delta(x,y) &= \Delta_0 - \frac{x^2 - y^2}{2}\left(\frac{1}{R_1} - \frac{1}{R_2}\right) && (2.29)
\end{aligned}$$

A.5. Calculation of the integral in the derivation of equation (2.35)

$$\Delta\phi(r) = \int_0^r \frac{-kr'}{\sqrt{f^2+r'^2}}\,\mathrm{d}r' \qquad \Big| u \overset{\text{def}}{=} f^2+r'^2 \implies \frac{\mathrm{d}u}{\mathrm{d}r'} = 2r' \implies \mathrm{d}r' = \frac{\mathrm{d}u}{2r'}$$

$$\Delta\phi(r) = -k\int_{r'=0}^{r'=r} \frac{\cancel{r'}}{\sqrt{u}}\frac{\mathrm{d}u}{2\cancel{r'}}$$

$$\Delta\phi(r) = -k\frac{1}{2}\int_{r'=0}^{r'=r} u^{-\frac{1}{2}}\,\mathrm{d}u$$

$$\Delta\phi(r) = -k\frac{1}{\cancel{2}}\cancel{2}\,u^{\frac{1}{2}}\Big|_{r'=0}^{r'=r}$$

$$\Delta\phi(r) = -k\frac{1}{\cancel{2}}\cancel{2}\,\sqrt{f^2+r'^2}\Big|_{r'=0}^{r'=r}$$

$$\Delta\phi(r) = -k\left(\sqrt{f^2+r^2}-\sqrt{f^2}\right)$$

$$\underline{\underline{\Delta\phi(r) = -k\left(\sqrt{r^2+f^2}-f\right)}}$$

A.6. Derivation of the reflection coefficient (3.5) and the transmission coefficient (3.6)

The first boundary condition at $x = 0$ for the one-dimensional Helmholtz-Equation (3.3) is simply the continuity condition (see figure 3.2).

$$\psi_1(0) = \psi_2(0) \tag{8}$$

By inserting equations (3.1) and (3.2) into condition (8), we get:

$$\cancel{e^0} + r\cancel{e^0} = t\cancel{e^0}$$
$$\underline{t = 1 + r} \tag{9}$$

This eliminates the phasor t from equations (3.1) and (3.2):

$$\psi_1(x) = e^{ikx} + re^{-ikx} \tag{10}$$
$$\psi_2(x) = (1+r)e^{ikx} \tag{11}$$

We will need the first derivatives of equations (10) and (11):

$$\psi_1'(x) = ike^{ikx} - irke^{-ikx} \tag{12}$$
$$\psi_2'(x) = ik(1+r)e^{ikx} \tag{13}$$

The second boundary condition at $x = 0$ can be derived by integrating the one-dimensional Helmholtz-Equation (3.3) over an infinitesimally small range around $x = 0$:

$$\begin{aligned}
&\lim_{\epsilon\to 0}\int_{-\epsilon}^{\epsilon} \frac{1}{k^2}\frac{\partial^2\psi(x)}{\partial x^2}\,\mathrm{d}x + \lim_{\epsilon\to 0}\int_{-\epsilon}^{\epsilon} V_0\delta(x)\psi(x)\,\mathrm{d}x = 0\\
&\frac{1}{k^2}\left(\psi_2'(0) - \psi_1'(0)\right) + V_0\psi_2(0) = 0\\
&\psi_2'(0) - \psi_1'(0) + V_0k^2\psi_2(0) = 0\\
&\underline{\psi_2'(0) = \psi_1'(0) - V_0k^2\psi_2(0)} \qquad (14)
\end{aligned}$$

We can now insert equations (11) to (13) into the second boundary condition (14):

$$\begin{aligned}
&i(1+r)k\cancel{e^0} = ik\cancel{e^0} - irk\cancel{e^0} - k^2V_0(1+r)\cancel{e^0}\\
&\cancel{ik} + irk = \cancel{ik} - irk - k^2V_0 - k^2V_0r\\
&2ir\cancel{k} + k^{\cancel{2}}V_0r = -k^{\cancel{2}}V_0\\
&r(kV_0 + 2i) = -kV_0\\
&\underline{\underline{r = -\frac{kV_0}{kV_0 + 2i}}} \quad \overset{(9)}{\Longrightarrow} \qquad (3.5)
\end{aligned}$$

$$\begin{aligned}
&t = 1 - \frac{kV_0}{kV_0 + 2i}\\
&t = \frac{\cancel{kV_0} + 2ik - \cancel{kV_0}}{kV_0 + 2i}\\
&\underline{\underline{t = \frac{2ik}{V_0 + 2ik}}} \qquad (3.6)
\end{aligned}$$

A.7. Derivation of equation (3.8)

We start from equation (3.7):

$$
\begin{aligned}
&r_0 \overset{\text{def}}{=} |r| \\
&r_0 = \sqrt{rr^*} \\
&r_0^2 = rr^* \quad \overset{(3.5)}{\Longrightarrow} \\
&r_0^2 = \left(-\frac{kV_0}{kV_0+2i}\right)\left(-\frac{kV_0}{kV_0-2i}\right) \\
&r_0^2 = \frac{k^2V_0^2}{k^2V_0^2+4} \\
&r_0^2\left(k^2V_0^2+4\right) = k^2V_0^2 \\
&k^2V_0^2r_0^2 + 4r_0^2 = k^2V_0^2 \\
&k^2V_0^2r_0^2 - k^2V_0^2 = -4r_0^2 \\
&V_0^2k^2\left(r_0^2-1\right) = -4r_0^2 \\
&V_0^2k^2\left(1-r_0^2\right) = 4r_0^2 \\
&V_0^2 = \frac{4r_0^2}{k^2\left(1-r_0^2\right)} \\
&\underline{\underline{V_0(r_0) = \pm\frac{2r_0}{k\sqrt{1-r_0^2}}}} \qquad (3.8)
\end{aligned}
$$

A.8. Derivation of equations (3.9) and (3.10)

We start from equation (3.5):

$$
\begin{aligned}
&r(r_0) = -\frac{V_0(r_0)}{2ik - V_0(r_0)} \quad \overset{(3.8)}{\Longrightarrow} \\
&r(r_0) = -\frac{\pm\cancel{k}\frac{2r_0}{\cancel{k}\sqrt{1-r_0^2}}}{\pm\cancel{k}\frac{2r_0}{\cancel{k}\sqrt{1-r_0^2}} + 2ik} && \left|\cdot\frac{\sqrt{1-r_0^2}}{\sqrt{1-r_0^2}}\right. \\
&r(r_0) = -\frac{\pm\cancel{2}r_0}{\pm\cancel{2}r_0 + i\cancel{2}\sqrt{1-r_0^2}} && \left|\cdot\frac{\pm r_0 - i\sqrt{1-r_0^2}}{\pm r_0 - i\sqrt{1-r_0^2}}\right. \\
&r(r_0) = -\frac{r_0^2 \mp ir_0\sqrt{1-r_0^2}}{\cancel{r_0^2} + 1 - \cancel{r_0^2}} \\
&\underline{\underline{r(r_0) = -r_0^2 \pm ir_0\sqrt{1-r_0^2}}} \qquad (3.9)
\end{aligned}
$$

The transmission coefficient $t(r_0)$ can be derived in a one-liner by considering equation (9) in appendix A.6:

$$t(r_0) = 1 + r(r_0) \overset{(3.9)}{\Longrightarrow}$$
$$\underline{\underline{t(r_0) = 1 - r_0^2 \pm i r_0 \sqrt{1 - r_0^2}}} \qquad (3.10)$$

A.9. Proof of energy-conservation condition (3.13)

Proof that the energy-conservation condition (equation (3.13)) is satisfied:

$$|r(r_0)|^2 + |t(r_0)|^2 = 1$$
$$rr^* + tt^* = 1 \overset{(3.11)}{\Longrightarrow} \overset{(3.12)}{\Longrightarrow}$$
$$\left(-r_0^2 - i r_0 \sqrt{1 - r_0^2}\right)\left(-r_0^2 + i r_0 \sqrt{1 - r_0^2}\right) +$$
$$\left(1 - r_0^2 - i r_0 \sqrt{1 - r_0^2}\right)\left(1 - r_0^2 + i r_0 \sqrt{1 - r_0^2}\right) = 1$$
$$r_0^4 + r_0^2\left(1 - r_0^2\right) + \left(1 - r_0^2\right)^2 + r_0^2\left(1 - r_0^2\right) = 1$$
$$\cancel{r_0^4} + r_0^2 - \cancel{r_0^4} + 1 - 2r_0^2 + \cancel{r_0^4} + r_0^2 - \cancel{r_0^4} = 1$$
$$\cancel{r_0^2 + r_0^2 - 2r_0^2} + 1 = 1$$
$$\underline{\underline{1 = 1}} \ \square$$

A.10. Proof of phase-condition (3.14)

Proof that the phase-condition in equation (3.14) is satisfied:

$$\arg\left(t(r_0)\right) - \arg\left(r(r_0)\right) = \frac{\pi}{2} \overset{(3.11)}{\Longrightarrow} \overset{(3.12)}{\Longrightarrow}$$

$$\arg\left(1 - r_0^2 - ir_0\sqrt{1-r_0^2}\right) - \arg\left(r_0^2 - ir_0\sqrt{1-r_0^2}\right) = \frac{\pi}{2}$$

$$\underbrace{\arg\left(1 - r_0^2 - ir_0\sqrt{1-r_0^2}\right)}_{\mathrm{Re}>0 \implies \arg=\arctan\left(\frac{\mathrm{Im}}{\mathrm{Re}}\right)} - \underbrace{\arg\left(r_0^2 - ir_0\sqrt{1-r_0^2}\right)}_{\mathrm{Re}<0,\mathrm{Im}<0 \implies \arg=\arctan\left(\frac{\mathrm{Im}}{\mathrm{Re}}\right)-\pi} = \frac{\pi}{2}$$

$$\arctan\left(\frac{-r_0\sqrt{1-r_0^2}}{1-r_0^2}\right) - \left(\arctan\left(\frac{-r_0\sqrt{1-r_0^2}}{-r_0^2}\right) - \pi\right) = \frac{\pi}{2}$$

$$-\arctan\left(\frac{r_0}{\sqrt{1-r_0^2}}\right) - \arctan\left(\frac{\sqrt{1-r_0^2}}{r_0}\right) + \pi = \frac{\pi}{2} \qquad \Bigg|\, s \overset{\mathrm{def}}{=} \frac{\sqrt{1-r_0^2}}{r_0^2}$$

$$-\arctan\left(\frac{1}{s}\right) - \arctan(s) = -\frac{\pi}{2} \qquad \Big|\arctan(x) = \frac{i}{2}\left(\ln(1-ix) - \ln(1+ix)\right)$$

$$-\frac{i}{2}\left(\ln\left(1-\frac{i}{s}\right) - \ln\left(1+\frac{i}{s}\right)\right) - \frac{i}{2}\left(\ln(1-is) - \ln(1+is)\right) = -\frac{\pi}{2} \qquad |\cdot i$$

$$\frac{1}{2}\left(\ln\left(1-\frac{i}{s}\right) - \ln\left(1+\frac{i}{s}\right)\right) + \frac{1}{2}\left(\ln(1-is) - \ln(1+is)\right) = -i\frac{\pi}{2}$$

$$\frac{1}{2}\left(\ln\left(\frac{s-i}{s}\right) - \ln\left(\frac{s+i}{s}\right)\right) + \frac{1}{2}\left(\ln(1-is) - \ln(1+is)\right) = -i\frac{\pi}{2}$$

$$\frac{1}{2}\ln\left(\frac{s-i}{s}\right) - \frac{1}{2}\ln\left(\frac{s+i}{s}\right) + \frac{1}{2}\ln(1-is) - \frac{1}{2}\ln(1+is) = -i\frac{\pi}{2}$$

$$\ln\left(\sqrt{\frac{s-i}{s}}\right) - \ln\left(\sqrt{\frac{s+i}{s}}\right) + \ln\left(\sqrt{1-is}\right) - \ln\left(\sqrt{1+is}\right) = -i\frac{\pi}{2}$$

$$\ln\left(\sqrt{\frac{s-i}{s}}\right) + \ln\left(\sqrt{\frac{s}{s+i}}\right) + \ln\left(\sqrt{\frac{1-is}{1+is}}\right) = -i\frac{\pi}{2} \qquad |e^{\cdot}$$

$$\sqrt{\frac{s-i}{\cancel{s}}}\sqrt{\frac{\cancel{s}}{s+i}}\sqrt{\frac{1-is}{1+is}} = e^{-i\frac{\pi}{2}} \qquad \Big|^{2}$$

$$\frac{s-i}{s+i}\cdot\frac{1-is}{1+is}\cdot\frac{(-i)}{(-i)} = e^{-i\pi}$$

$$\frac{\cancel{s-i}}{s+i}\cdot\frac{-s-i}{\cancel{s-i}} = -1$$

$$-\frac{s+i}{s+i} = -1$$

$$\underline{\underline{-1=-1}} \;\square$$

A.11. Proof of unitarity condition (3.15)

Proof that the scattering matrix from equation (3.15) is unitary:

$$
\begin{aligned}
\underline{\underline{SS^\dagger}} &= \begin{bmatrix} r & t \\ t & r \end{bmatrix} \begin{bmatrix} r^* & t^* \\ t^* & r^* \end{bmatrix} \\
\underline{\underline{SS^\dagger}} &= \begin{bmatrix} rr^*+tt^* & rt^*+tr^* \\ rt^*+tr^* & rr^*+tt^* \end{bmatrix} \\
\underline{\underline{SS^\dagger}} &= \begin{bmatrix} |r|^2+|t|^2 & rt^*+tr^* \\ rt^*+tr^* & |r|^2+|t|^2 \end{bmatrix} \overset{(3.13)}{\Longrightarrow} \\
\underline{\underline{SS^\dagger}} &= \underline{\begin{bmatrix} 1 & rt^*+tr^* \\ rt^*+tr^* & 1 \end{bmatrix}}
\end{aligned} \tag{15}
$$

Considering equations (3.11) and (3.12) we can write:

$$
\begin{aligned}
rt^*+tr^* =& \left(-r_0^2 - ir_0\sqrt{1-r_0^2}\right)\left(1-r_0^2+ir_0\sqrt{1-r_0^2}\right)+ \\
& \left(1-r_0^2 - ir_0\sqrt{1-r_0^2}\right)\left(-r_0^2+ir_0\sqrt{1-r_0^2}\right) \\
rt^*+tr^* =& -r_0^2 - \cancel{ir_0\sqrt{1-r_0^2}} + r_0^4 + \cancel{ir_0^3\sqrt{1-r_0^2}} - \cancel{ir_0^3\sqrt{1-r_0^2}} + r_0^2\left(1-r_0^2\right) \\
& -r_0^2 + r_0^4 + \cancel{ir_0^3\sqrt{1-r_0^2}} + \cancel{ir_0\sqrt{1-r_0^2}} - \cancel{ir_0^3\sqrt{1-r_0^2}} + r_0^2\left(1-r_0^2\right) \\
rt^*+tr^* =& -\cancel{2r_0^2} + \cancel{2r_0^4} + \cancel{r_0^2} - \cancel{r_0^4} + \cancel{r_0^2} - \cancel{r_0^4} \\
rt^*+tr^* =& \; 0 \overset{(15)}{\Longrightarrow}
\end{aligned}
$$

$$
\underline{\underline{SS^\dagger = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}}} \quad \square
$$

A.12. Derivation of the condition for the critical complex wavenumber (3.18)

Derivation for the condition that has to be fulfilled by the critical wavenumber $\tilde{k}_c$ (equation (3.18)).

$$r_{cav}(\tilde{k}_c) = 0 \overset{(3.17)}{\Longrightarrow}$$

$$-r_0^2 - ir_0\sqrt{1-r_0^2} - \frac{\left(1 - r_0^2 - ir_0\sqrt{1-r_0^2}\right)^2}{e^{-2i\tilde{k}_c l} - r_0^2 - ir_0\sqrt{1-r_0^2}} = 0$$

$$\left(-r_0^2 - ir_0\sqrt{1-r_0^2}\right)\left(e^{-2i\tilde{k}_c l} - r_0^2 - ir_0\sqrt{1-r_0^2}\right) - \left(1 - r_0^2 - ir_0\sqrt{1-r_0^2}\right)^2 = 0$$

$$\left(-r_0^2 - ir_0\sqrt{1-r_0^2}\right)\left(e^{-2i\tilde{k}_c l} - r_0^2 - ir_0\sqrt{1-r_0^2}\right) = \left(1 - r_0^2 - ir_0\sqrt{1-r_0^2}\right)^2$$

$$e^{-2i\tilde{k}_c l} - r_0^2 - ir_0\sqrt{1-r_0^2} = \frac{\left(1 - r_0^2 - ir_0\sqrt{1-r_0^2}\right)^2}{-r_0^2 - ir_0\sqrt{1-r_0^2}}$$

$$e^{-2i\tilde{k}_c l} = \frac{\left(1 - r_0^2 - ir_0\sqrt{1-r_0^2}\right)^2}{-r_0^2 - ir_0\sqrt{1-r_0^2}} + r_0^2 + ir_0\sqrt{1-r_0^2}$$

$$e^{-2i\tilde{k}_c l} = \frac{\left(1 - r_0^2 - ir_0\sqrt{1-r_0^2}\right)^2 + \left(r_0^2 + ir_0\sqrt{1-r_0^2}\right)\left(-r_0^2 - ir_0\sqrt{1-r_0^2}\right)}{-r_0^2 - ir_0\sqrt{1-r_0^2}}$$

$$e^{-2i\tilde{k}_c l} = \frac{\left(1-r_0^2\right)^2 - 2i\left(1-r_0^2\right)r_0\sqrt{1-r_0^2} - \cancel{r_0^2\left(1-r_0^2\right)} - r_0^4 - 2ir_0^3\sqrt{1-r_0^2} + \cancel{r_0^2\left(1-r_0^2\right)}}{-r_0^2 - ir_0\sqrt{1-r_0^2}}$$

$$e^{-2i\tilde{k}_c l} = \frac{1 - 2r_0^2 + \cancel{r_0^4} - 2ir_0\sqrt{1-r_0^2} + \cancel{2ir_0^3\sqrt{1-r_0^2}} - \cancel{r_0^4} - \cancel{2ir_0^3\sqrt{1-r_0^2}}}{-r_0^2 - ir_0\sqrt{1-r_0^2}}$$

$$e^{-2i\tilde{k}_c l} = \frac{1 - 2r_0^2 - 2ir_0\sqrt{1-r_0^2}}{-r_0^2 - ir_0\sqrt{1-r_0^2}} \cdot \frac{-r_0^2 + ir_0\sqrt{1-r_0^2}}{-r_0^2 + ir_0\sqrt{1-r_0^2}}$$

$$e^{-2i\tilde{k}_c l} = \frac{-r_0^2 + ir_0\sqrt{1-r_0^2} + 2r_0^4 - \cancel{2ir_0^3\sqrt{1-r_0^2}} + \cancel{2ir_0^3\sqrt{1-r_0^2}} + 2r_0^2\left(1-r_0^2\right)}{r_0^4 + r_0^2\left(1-r_0^2\right)}$$

$$e^{-2i\tilde{k}_c l} = \frac{-r_0^2 + ir_0\sqrt{1-r_0^2} + \cancel{2r_0^4} + 2r_0^2 - \cancel{2r_0^4}}{\cancel{r_0^4} + r_0^2 - \cancel{r_0^4}}$$

$$e^{-2i\tilde{k}_c l} = \frac{r_0^2 + ir_0\sqrt{1-r_0^2}}{r_0^2}$$

$$e^{-2i\tilde{k}_c l} = \frac{r_0 + i\sqrt{1-r_0^2}}{r_0}$$

$$e^{2i\tilde{k}_c l} = \frac{r_0}{r_0 + i\sqrt{1-r_0^2}} \cdot \frac{r_0 - i\sqrt{1-r_0^2}}{r_0 - i\sqrt{1-r_0^2}}$$

$$e^{2i\tilde{k}_c l} = \frac{r_0^2 - ir_0\sqrt{1-r_0^2}}{\cancel{r_0^2} + 1 - \cancel{r_0^2}}$$

$$2i\tilde{k}_c l = \ln\left(r_0^2 - ir_0\sqrt{1-r_0^2}\right) \pm 2\pi i n$$

$$\tilde{k}_c = \frac{1}{2il}\ln\left(r_0^2 - ir_0\sqrt{1-r_0^2}\right) \pm \frac{\pi n}{l}$$

$$\underline{\tilde{k}_c = \frac{\pi}{l}n - \frac{i}{2l}\ln\left(r_0^2 - ir_0\sqrt{1-r_0^2}\right) \quad \text{with} \;\; n \in \mathbb{Z}} \tag{16}$$

To separate the real part and the imaginary part of $\tilde{k}_c$, we can exploit the mathematical relation $\log(\alpha) = \log|\alpha| + i\arg(\alpha)$ to re-write equation (16).

$$\tilde{k}_c = \frac{\pi}{l}n - i\frac{1}{2l}\left(\ln\left|r_0^2 - ir_0\sqrt{1-r_0^2}\right| + i\arg\left(r_0^2 - ir_0\sqrt{1-r_0^2}\right)\right)$$

$$\tilde{k}_c = \frac{\pi}{l}n - i\frac{1}{2l}\left(\ln\left(\sqrt{r_0^4 + r_0^2\left(1-r_0^2\right)}\right) + i\arg\left(r_0^2 - ir_0\sqrt{1-r_0^2}\right)\right)$$

$$\tilde{k}_c = \frac{\pi}{l}n - i\frac{1}{2l}\left(\ln\left(\sqrt{\cancel{r_0^4} + r_0^2 - \cancel{r_0^4}}\right) + i\underbrace{\arg\left(r_0^2 - ir_0\sqrt{1-r_0^2}\right)}_{\text{Re}>0 \implies \arg=\arctan\left(\frac{\text{Im}}{\text{Re}}\right)}\right)$$

$$\tilde{k}_c = \frac{\pi}{l}n - i\frac{1}{2l}\left(\ln\left(r_0\right) + i\arctan\left(\frac{\sqrt{1-r_0^2}}{r_0}\right)\right)$$

$$\tilde{k}_c = \frac{\pi}{l}n - i\frac{1}{2l}\ln\left(r_0\right) - \frac{1}{2l}\arctan\left(\frac{\sqrt{1-r_0^2}}{r_0}\right)$$

$$\underline{\underline{\tilde{k}_c = \frac{1}{2l}\left(2\pi n - \arctan\left(\frac{\sqrt{1-r_0^2}}{r_0}\right)\right) - i\frac{1}{2l}\ln\left(r_0\right) \quad \text{with} \;\; n \in \mathbb{Z}}} \tag{3.18}$$

A.13. Derivation of equation (3.22)

We start from equation (3.17).

$$r_{cav} = - r_0^2 - ir_0\sqrt{1-r_0^2} - \frac{\left(1 - r_0^2 - ir_0\sqrt{1-r_0^2}\right)^2}{e^{-2ikl} - r_0^2 - ir_0\sqrt{1-r_0^2}}$$

$$r_{cav} = \frac{\left(-r_0^2 - ir_0\sqrt{1-r_0^2}\right)\left(e^{-2ikl} - r_0^2 - ir_0\sqrt{1-r_0^2}\right) - \left(1 - r_0^2 - ir_0\sqrt{1-r_0^2}\right)^2}{e^{-2ikl} - r_0^2 - ir_0\sqrt{1-r_0^2}}$$

$$\begin{aligned} r_{cav} =& \frac{1}{e^{-2ikl} - r_0^2 - ir_0\sqrt{1-r_0^2}} \cdot \Bigg[-r_0^2\left(e^{-2ikl} - r_0^2 - ir_0\sqrt{1-r_0^2}\right) \\ & -ir_0\sqrt{1-r_0^2}\left(e^{-2ikl} - r_0^2 - ir_0\sqrt{1-r_0^2}\right) - \left(1 - r_0^2 - ir_0\sqrt{1-r_0^2}\right)^2\Bigg] \end{aligned}$$

$$\begin{aligned} r_{cav} =& \frac{1}{e^{-2ikl} - r_0^2 - ir_0\sqrt{1-r_0^2}} \cdot \Bigg[-r_0^2 e^{-2ikl} + r_0^4 + ir_0^3\sqrt{1-r_0^2} - ir_0\sqrt{1-r_0^2}e^{-2ikl} \\ & +ir_0^3\sqrt{1-r_0^2} - r_0^2\left(1-r_0^2\right) - \left(1 - r_0^2 - ir_0\sqrt{1-r_0^2}\right)^2\Bigg] \end{aligned}$$

$$\begin{aligned} r_{cav} =& \frac{1}{e^{-2ikl} - r_0^2 - ir_0\sqrt{1-r_0^2}} \cdot \Bigg[-r_0^2 e^{-2ikl} + r_0^4 + 2ir_0^3\sqrt{1-r_0^2} - ir_0\sqrt{1-r_0^2}e^{-2ikl} \\ & -r_0^2 + r_0^4 - \left(1-r_0^2\right)^2 + 2i\left(1-r_0^2\right)r_0\sqrt{1-r_0^2} + r_0^2\left(1-r_0^2\right)\Bigg] \end{aligned}$$

$$\begin{aligned} r_{cav} =& \frac{1}{e^{-2ikl} - r_0^2 - ir_0\sqrt{1-r_0^2}} \cdot \Bigg[-r_0^2 e^{-2ikl} + \cancel{2r_0^4} + \cancel{2ir_0^3\sqrt{1-r_0^2}} - ir_0\sqrt{1-r_0^2}e^{-2ikl} \\ & -\cancel{r_0^2} - 1 + 2r_0^2 - \cancel{r_0^4} + 2ir_0\sqrt{1-r_0^2} - \cancel{2ir_0^3\sqrt{1-r_0^2}} + \cancel{r_0^2} - \cancel{r_0^4}\Bigg] \end{aligned}$$

$$r_{cav} = \frac{-r_0^2 e^{-2ikl} - ir_0\sqrt{1-r_0^2}e^{-2ikl} - 1 + 2r_0^2 + 2ir_0\sqrt{1-r_0^2}}{e^{-2ikl} - r_0^2 - ir_0\sqrt{1-r_0^2}}$$

$$r_{cav} = \frac{1}{\cos(-2kl) + i\sin(-2kl) - r_0^2 - ir_0\sqrt{1-r_0^2}} \cdot \Big[-r_0^2\left(\cos(-2kl) + i\sin(-2kl)\right)$$
$$-ir_0\sqrt{1-r_0^2}\left(\cos(-2kl) + i\sin(-2kl)\right) - 1 + 2r_0^2 + 2ir_0\sqrt{1-r_0^2}\Big]$$

$$r_{cav} = \frac{1}{\cos(-2kl) + i\sin(-2kl) - r_0^2 - ir_0\sqrt{1-r_0^2}} \cdot \Big[-r_0^2\cos(-2kl) - ir_0^2\sin(-2kl)$$
$$-ir_0\sqrt{(1-r_0^2)}\cos(-2kl) + r_0\sqrt{(1-r_0^2)}\sin(-2kl) - 1 + 2r_0^2 + 2ir_0\sqrt{(1-r_0^2)}\Big]$$

$$r_{cav} = \frac{1}{(\cos(-2kl) - r_0^2) + i\left(\sin(-2kl) - r_0\sqrt{1-r_0^2}\right)} \cdot$$
$$\left[\left(2r_0^2 - 1 - r_0^2\cos(-2kl) + r_0\sqrt{(1-r_0^2)}\sin(-2kl)\right)\right.$$
$$\left.+i\left(2r_0\sqrt{(1-r_0^2)} - r_0^2\sin(-2kl) - r_0\sqrt{(1-r_0^2)}\cos(-2kl)\right)\right]$$

$$r_{cav} = \frac{1}{(\cos(2kl) - r_0^2) - i\left(\sin(2kl) + r_0\sqrt{1-r_0^2}\right)} \cdot$$
$$\left[\left(2r_0^2 - 1 - r_0^2\cos(2kl) - r_0\sqrt{(1-r_0^2)}\sin(2kl)\right)\right.$$
$$\left.+i\left(2r_0\sqrt{(1-r_0^2)} + r_0^2\sin(2kl) - r_0\sqrt{(1-r_0^2)}\cos(2kl)\right)\right] \cdot$$
$$\cdot \frac{(\cos(2kl) - r_0^2) + i\left(\sin(2kl) + r_0\sqrt{1-r_0^2}\right)}{(\cos(2kl) - r_0^2) + i\left(\sin(2kl) + r_0\sqrt{1-r_0^2}\right)}$$

$$r_{cav} = \frac{1}{(\cos(2kl) - r_0^2)^2 + \left(\sin(2kl) + r_0\sqrt{(1-r_0^2)}\right)^2} \cdot$$
$$\Big\{\Big[2r_0^2 - 1 - r_0^2\cos(2kl) - r_0\sqrt{1-r_0^2}\sin(2kl)$$
$$+i\left(2r_0\sqrt{1-r_0^2} + r_0^2\sin(2kl) - r_0\sqrt{1-r_0^2}\cos(2kl)\right)\Big] \cdot$$
$$\left[\cos(2kl) - r_0^2 + i\left(\sin(2kl) + r0\sqrt{1-r_0^2}\right)\right]\Big\}$$

$$
\begin{aligned}
r_{cav} =& \frac{1}{\cos^2(2kl) - 2r_0^2\cos(2kl) + r_0^4 + \sin^2(2kl) + 2r_0\sqrt{1-r_0^2}\sin(2kl) + r_0^2(1-r_0^2)} \cdot \\
& \left\{\left[2r_0^2 - 1 - r_0^2\cos(2kl) - r_0\sqrt{1-r_0^2}\sin(2kl)\right.\right. \\
& \left.+ i\left(2r_0\sqrt{1-r_0^2} + r_0^2\sin(2kl) - r_0\sqrt{1-r_0^2}\cos(2kl)\right)\right] \cdot \\
& \left.\left[\cos(2kl) - r_0^2 + i\left(\sin(2kl) + r0\sqrt{1-r_0^2}\right)\right]\right\} \qquad \Big| \cos^2(2kl) + \sin^2(2kl) = 1
\end{aligned}
$$

$$
\begin{aligned}
r_{cav} =& \frac{1}{1 - 2r_0^2\cos(2kl) + \cancel{r_0^4} + 2r_0\sqrt{1-r_0^2}\sin(2kl) + r_0^2 - \cancel{r_0^4}} \cdot \\
& \left\{\left[2r_0^2 - 1 - r_0^2\cos(2kl) - r_0\sqrt{1-r_0^2}\sin(2kl)\right.\right. \\
& \left.+ i\left(2r_0\sqrt{1-r_0^2} + r_0^2\sin(2kl) - r_0\sqrt{1-r_0^2}\cos(2kl)\right)\right] \cdot \\
& \left.\left[\cos(2kl) - r_0^2 + i\left(\sin(2kl) + r0\sqrt{1-r_0^2}\right)\right]\right\}
\end{aligned}
$$

$$
\begin{aligned}
r_{cav} =& \frac{1}{1 + r_0^2 - 2r_0^2\cos(2kl) + 2r_0\sqrt{(1-r_0^2)}\sin(2kl)} \cdot \\
& \left[\left(2r_0^2 - 1 - r_0^2\cos(2kl) - r_0\sqrt{(1-r_0^2)}\sin(2kl)\right)\left(\cos(2kl) - r_0^2\right)\right. \\
& - \left(2r_0\sqrt{1-r_0^2} + r_0^2\sin(2kl) - r_0\sqrt{1-r_0^2}\cos(2kl)\right)\left(\sin(2kl) + r_0\sqrt{1-r_0^2}\right) \\
& + i\left(2r_0\sqrt{1-r_0^2} + r_0^2\sin(2kl) - r_0\sqrt{1-r_0^2}\cos(2kl)\right)\left(\cos(2kl) - r_0^2\right) \\
& \left.+ i\left(2r_0^2 - 1 - r_0^2\cos(2kl) - r_0\sqrt{(1-r_0^2)}\sin(2kl)\right)\left(\sin(2kl) + r_0\sqrt{(1-r_0^2)}\right)\right]
\end{aligned}
$$

$$
\begin{aligned}
r_{cav} =& \frac{1}{1+r_0^2-2r_0^2\cos(2kl)+2r_0\sqrt{(1-r_0^2)}\sin(2kl)} \cdot \\
&\Big[2r_0^2\cos(2kl)-\cos(2kl)-\underline{r_0^2\cos^2(2kl)}-\cancel{r_0\sqrt{1-r_0^2}\sin(2kl)\cos(2kl)} \\
&-\cancel{2r_0^4}+r_0^2+\cancel{r_0^4\cos(2kl)}+r_0^3\sqrt{1-r_0^2}\sin(2kl) \\
&-2r_0\sqrt{1-r_0^2}\sin(2kl)-\underline{r_0^2\sin^2(2kl)}+\cancel{r_0\sqrt{1-r_0^2}\sin(2kl)\cos(2kl)} \\
&-2r_0^2+\cancel{2r_0^4}-r_0^3\sqrt{1-r_0^2}\sin(2kl)+r_0^2\cos(2kl)-\cancel{r_0^4\cos(2kl)} \\
&+2ir_0\sqrt{1-r_0^2}\cos(2kl)+\cancel{ir_0^2\sin(2kl)\cos(2kl)}-\underline{ir_0\sqrt{1-r_0^2}\cos^2(2kl)} \\
&-\cancel{2ir_0^3\sqrt{1-r_0^2}}-\cancel{ir_0^4\sin(2kl)}+ir0^3\sqrt{1-r_0^2}\cos(2kl) \\
&+2ir_0^2\sin(2kl)-i\sin(2kl)-\cancel{ir_0^2\sin(2kl)\cos(2kl)}-\underline{ir_0\sqrt{1-r_0^2}\sin^2(2kl)} \\
&+\cancel{2ir_0^3\sqrt{1-r_0^2}}-ir_0\sqrt{1-r_0^2}-ir_0^3\sqrt{1-r_0^2}\cos(2kl)-ir_0^2\sin(2kl)+\cancel{ir_0^4\sin(2kl)}\Big]
\end{aligned}
$$

$$
\begin{aligned}
r_{cav} =& \frac{1}{1+r_0^2-2r_0^2\cos(2kl)+2r_0\sqrt{(1-r_0^2)}\sin(2kl)} \cdot \\
&\Big[3r_0^2\cos(2kl)-\cos(2kl)-\cancel{r_0^2}+\cancel{r_0^2}+\cancel{r_0^3\sqrt{1-r_0^2}\sin(2kl)} \\
&-2r_0\sqrt{1-r_0^2}\sin(2kl)-2r_0^2-\cancel{r_0^3\sqrt{1-r_0^2}\sin(2kl)} \\
&+2ir_0\sqrt{1-r_0^2}\cos(2kl)-2ir_0\sqrt{1-r_0^2}+\cancel{ir_0^3\sqrt{1-r_0^2}\cos(2kl)} \\
&+ir_0^2\sin(2kl)-i\sin(2kl)-\cancel{ir_0^3\sqrt{1-r_0^2}\cos(2kl)}\Big]
\end{aligned}
$$

$$
\begin{aligned}
r_{cav} =& \frac{(3r_0^2-1)\cos(2kl)-2r_0\sqrt{1-r_0^2}\sin(2kl)-2r_0^2}{1+r_0^2-2r_0^2\cos(2kl)+2r_0\sqrt{(1-r_0^2)}\sin(2kl)} \\
&+ i\,\frac{2r_0\sqrt{1-r_0^2}\cos(2kl)+(r_0^2-1)\sin(2kl)-2r_0\sqrt{(1-r_0^2)}}{1+r_0^2-2r_0^2\cos(2kl)+2r_0\sqrt{(1-r_0^2)}\sin(2kl)}
\end{aligned}
\tag{3.22}
$$

A.14. Derivation of equation (3.38)

The value of $\frac{\mathrm{d}\varphi_{QLE}}{\mathrm{d}k}$ becomes maximal when the denominator in equation (3.37) takes on a minimal value. This is the case when $k = k_c$. Therefore, we can write:

$$\left.\frac{\mathrm{d}\varphi_{QLE}}{\mathrm{d}k}\right|_{max} = \left.\frac{\mathrm{d}\varphi_{QLE}}{\mathrm{d}k}\right|_{k=k_c} \overset{(3.37)}{\Longrightarrow}$$
$$\underline{\left.\frac{\mathrm{d}\varphi_{QLE}}{\mathrm{d}k}\right|_{max} = \frac{2}{\kappa_c}} \tag{17}$$

The half-maximum-value is defined as

$$\left.\frac{\mathrm{d}\varphi_{QLE}}{\mathrm{d}k}\right|_{max/2} = \frac{1}{2}\left.\frac{\mathrm{d}\varphi_{QLE}}{\mathrm{d}k}\right|_{max} \overset{(17)}{\Longrightarrow}$$
$$\underline{\left.\frac{\mathrm{d}\varphi_{QLE}}{\mathrm{d}k}\right|_{max/2} = \frac{1}{\kappa_c}} \tag{18}$$

This allows us to calculate the full-width-at-half-maximum (FWHM) value:

$$\left.\frac{\mathrm{d}\varphi_{QLE}}{\mathrm{d}k}\right|_{k=k_{max/2}} \overset{!}{=} \left.\frac{\mathrm{d}\varphi_{QLE}}{\mathrm{d}k}\right|_{max/2} \overset{(3.37)}{\Longrightarrow} \overset{(18)}{\Longrightarrow}$$
$$\frac{2\kappa_c}{\left(k_{max/2} - k_c\right)^2 + \kappa_c^2} = \frac{1}{\kappa_c}$$
$$\frac{2\kappa_c^2}{\left(k_{max/2} - k_c\right)^2 + \kappa_c^2} = 1$$
$$2\kappa_c^2 = \left(k_{max/2} - k_c\right)^2 + \kappa_c^2$$
$$\kappa_c^2 = \left(k_{max/2} - k_c\right)^2$$
$$\kappa_c = \sqrt{\left(k_{max/2} - k_c\right)^2} \qquad \left| \sqrt{\left(k_{max/2} - k_c\right)^2} = \frac{1}{2}\,\mathrm{FWHM}\left(\frac{\mathrm{d}\varphi_{QLE}}{\mathrm{d}k}\right)\right.$$
$$\underline{\underline{\kappa_c = \frac{1}{2}\,\mathrm{FWHM}\left(\frac{\mathrm{d}\varphi_{QLE}}{\mathrm{d}k}\right)}} \tag{3.38}$$

A.15. Derivation of equation (3.42)

It is possible to (approximately) express matrix $\underline{\underline{T}}_c$ from equation (3.41) as a function of r_0. This can be done using relation (3.19), as is demonstrated in the following derivation:

$$\underline{\underline{T}}_c(r_0) \approx -\mathbb{1} e^{2ik_c l} \overset{(3.19)}{\Longrightarrow}$$

$$\underline{\underline{T}}_c(r_0) \approx -\mathbb{1} e^{i\cancel{2l}\frac{1}{\cancel{2l}}\left(\cancel{2\pi n} - \arctan\frac{\sqrt{1-r_0^2}}{r_0}\right)}$$

$$\underline{\underline{T}}_c(r_0) \approx -\mathbb{1} e^{-i\arctan\frac{\sqrt{1-r_0^2}}{r_0}} \qquad \Big| \; s \overset{\text{def}}{=} \frac{\sqrt{1-r_0^2}}{r_0}$$

$$\underline{\underline{T}}_c(r_0) \approx -\mathbb{1} e^{-i\arctan(s)} \qquad \Big| \arctan(s) = \frac{i}{2}\left(\ln(1-is) - \ln(1+is)\right)$$

$$\underline{\underline{T}}_c(r_0) \approx -\mathbb{1} e^{\frac{1}{2}(\ln(1-is) - \ln(1+is))}$$

$$\underline{\underline{T}}_c(r_0) \approx -\mathbb{1} e^{\frac{1}{2}\ln(1-is) - \frac{1}{2}\ln(1+is)}$$

$$\underline{\underline{T}}_c(r_0) \approx -\mathbb{1} e^{\frac{1}{2}\ln(1-is)} e^{-\frac{1}{2}\ln(1+is)}$$

$$\underline{\underline{T}}_c(r_0) \approx -\mathbb{1} e^{\ln\sqrt{1-is}} e^{\ln\frac{1}{\sqrt{1+is}}}$$

$$\underline{\underline{T}}_c(r_0) \approx -\mathbb{1}\sqrt{1-is}\frac{1}{\sqrt{1+is}}$$

$$\underline{\underline{T}}_c(r_0) \approx -\mathbb{1}\sqrt{\frac{1-is}{1+is}} \qquad \Big| \; s \overset{\text{def}}{=} \frac{\sqrt{1-r_0^2}}{r_0}$$

$$\underline{\underline{T}}_c(r_0) \approx -\mathbb{1}\sqrt{\frac{1-i\frac{\sqrt{1-r_0^2}}{r_0}}{1+i\frac{\sqrt{1-r_0^2}}{r_0}}}$$

$$\underline{\underline{T}}_c(r_0) \approx -\mathbb{1}\sqrt{\frac{r_0 - i\sqrt{1-r_0^2}}{r_0 + i\sqrt{1-r_0^2}}} \qquad \Big| \cdot \sqrt{\frac{r_0 - i\sqrt{1-r_0^2}}{r_0 - i\sqrt{1-r_0^2}}}$$

$$\underline{\underline{T}}_c(r_0) \approx -\mathbb{1}\sqrt{\frac{\left(r_0 - i\sqrt{1-r_0^2}\right)^2}{\cancel{r_0^2} + 1 - \cancel{r_0^2}}} \overset{(3.11)}{\Longrightarrow}$$

$$\underline{\underline{T}}_c(r_0) \approx \mathbb{1}\left(-r_0 + i\sqrt{1-r_0^2}\right) \tag{3.42}$$

B. Software and hardware tools

All computer software developed in the course of this thesis was written in the MatLab Language (Version 2020b), and executed on a Windows 10 PC equipped with a 64-core AMD Ryzen Threadripper CPU and 256 GB RAM. Where possible, double precision calculations were sped up by two NVIDIA RTX 8000 48GB GPUs. When precision beyond double precision was required, the Multiprecision Computing Toolbox for MATLAB from Advanpix[3] was used.

C. Software implementation of base functionality

C.1. create_ax

The function `create_ax(gpu, N, L)` creates a vector with `N` entries containing the axis coordinates (almost) symmetric around zero so that the total length corresponds to the target length `L` minus one step according to the requirements of FFT, as explained in section 2.2.2.

Input values:

- `gpu`: 0: use CPU, 1: use GPU, 2: use mp library
- `N`: number of steps
- `L`: nominal length in meters

Return values:

- `ax`: vector with `N` entries containing coordinates
- `dx`: step width in meters
- `amax`: largest coordinate value in meters
- `amin`: smallest coordinate value in meters
- `axlen`: (reduced) axis length in meters

For example, calling `create_ax(0, 4, 2)` returns `ax = [-1, -0.5, 0, 0.5]`, `amax = 0.5`, `amin = -1` and `axlen = 1.5`. With `gpu=1`, the returned vector is of type `gpuArray`. With `gpu=2`, all returned parameters are high-precision values of type `mp`.

[3] www.advanpix.com

Source code:

```
function [ax, dx, axmax, axmin, axlen] = create_ax(gpu, N, L)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Creates a vector with N entries containing the axis
% coordinates (almost) symmetric around zero
% so that the total length corresponds to target length L
% minus one step according to the requirements of FFT.
%
% example: create_ax(0, 4, 2) returns
%  ax = [-1, -0.5, 0, 0.5]
%  dx = 0.5, axmax = 0.5, axmin = -1, axlen = 1.5
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Input parameters:
% gpu ... 0: use CPU, 1: use GPU, 2: use mp library
% N   ... number of steps
% L   ... side length in m
%
% Return values:
% ax    ... vector with position-space-array axis coordinates in m
% dx    ... step width in m
% axmax ... maximum coordinate value in m
% axmin ... minimum coordinate value in m
% axlen ... (reduced) axis length in m
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if gpu==2
    % calculate in mp precision
    L = mp(L);
    N = mp(N);
    one = mp('1');
    two = mp('2');
else
    % CPU or GPU
    one = 1;
    two = 2;
end
dx = L/N;
ax = ((one:N) - N/two) * dx - dx;
if gpu==1
    % if GPU, convert to GPU vector
    ax = gpuArray(ax);
end
axmax = max(ax);
axmin = min(ax);
axlen = axmax - axmin;
```

C.2. fft2_phys_spatial

The function `fft2_phys_spatial(X, ax)` performs a 2D Fast Fourier Transform (FFT) of the position-space-array `X` holding spatial function values, with xy-coordinates according to the axis coordinates provided in the `ax` vector. The output is the Fast Fourier Transform of `X` (according to physics sign convention as described in section 2.2.1), i.e., an array of the same size with spatial-frequency Fourier coefficients as described in section 2.2.3.

If the input is normalized, so that integrating the squared absolute values over the whole surface area (according to the axes scaling provided by `ax`) equals one, i.e., if `trapz(ax, trapz(ax, X .* conj(X), 2)) == 1`, then the output is also normalized so that `norm(fourier_coeff) == 1`.

Input values:

- `X`: position-space $N \times N$ array holding spatial function values
- `ax`: vector with N entries containing coordinate data

Output values:

- `fourier_coeff`: Fast Fourier Transform of `X`

Source code:

```
 1  function fourier_coeff = fft2_phys_spatial(X, ax)
 2  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 3  % 2D spatial Fast Fourier Transform according to
 4  % Physics Spatial Fourier Transform Convention
 5  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 6  % Input parameters:
 7  % X  ... position-space-array array holding spatial function values
 8  % ax ... vector with axis coordinates in m
 9  %
10  % Output:
11  % fourier_coeff ... array with spatial-frequency Fourier coefficients
12  %
13  % if the input is normalied, so that
14  % trapz(ax, trapz(ax, X .* conj(X), 2)) == 1,
15  % then the output is also normalized so that norm(fourier_coeff)==1
16  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
17  ax_len = max(ax)-min(ax);
18  fourier_coeff = fftshift(ifft2(X * ax_len));
```

The fact that the `ifft2` function is used in line 18 instead of the `fft2` function is because MatLab generically supports the “Engineering Sign Convention” in FFT, whereas we have implemented the “Physic Sign Convention” as described in section 2.2.1.

C.3. ifft2_phys_spatial

The function `ifft2_phys_spatial(fourier_coeff, ax)` performs a 2D Inverse Fast-Fourier-Transform (IFFT) of the spatial-frequency Fourier coefficients array `fourier_coeff` and generates a position-space-array `Y` holding spatial function values, with xy-coordinates according to the axis coordinates provided in the `ax` vector as described in section 2.2.3.

If the input is normalized, so that `norm(fourier_coeff) == 1`, then the output is also normalized so that the integral of the squared absolute values over the whole surface area (according to the axes scaling provided by `ax`) equals one, i.e., `trapz(ax, trapz(ax, Y .* conj(Y), 2)) == 1`.

Input values:

- `fourier_coeff`: position-space $N \times N$ array holding spatial function values
- `ax`: vector with N entries containing spatial axis coordinates

Output values:

- `fourier_coeff`: Fast Fourier Transform of `X`

Source code:

```
 1 function Y = ifft2_phys_spatial(fourier_coeff, ax)
 2 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 3 % 2D spatial Inverse Fast Fourier Transform according to
 4 % Physics Spatial Fourier Transform Convention
 5 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 6 % Input parameters:
 7 % fourier_coeff  ... array with spatial-frequency Fourier coefficients
 8 % ax ... vector with axis coordinates in m
 9 %
10 % Output:
11 % Y ... position-space-array holding spatial function values
12 %
13 % if the input is normalied, so that norm(fourier_coeff)==1
14 % then the output is also normalized so that
15 % trapz(ax, trapz(ax, X .* conj(X), 2)) == 1
16 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
17 ax_len = max(ax)-min(ax);
18 Y = fft2(ifftshift(fourier_coeff)) / ax_len;
```

The fact that the `fft2` function is used in line 18 instead of the `ifft2` function is because MatLab generically supports the “Engineering Sign Convention” in IFFT, whereas we have implemented the “Physics Sign Convention” as described in section 2.2.1. The factor `ax_len` ensures normalization and corresponds to the normalization term in equation (2.45).

C.4. fft2_basis_func

The function `fft2_basis_func(gpu, nx, ny, ax, f_space_out)` creates a normalized (`nx, ny`) Fast-Fourier basis function. The output is either in spatial-frequency space, or in position space as described in section 2.2.3.

The output-array `psi` is always normalized. So, if `psi` is in position-space, then integrating the squared absolute values of `psi` over the whole surface area (according to the axes scaling provided by `ax`) results in one, i.e., `trapz(ax, trapz(ax, psi .* conj(psi), 2)) == 1`. If `psi` is in spatial-frequency space, then `norm(psi) == 1`.

Input values:

- `gpu`: 0: use CPU, 1: use GPU, 2: use mp library
- `nx`: mode number in x-direction
- `ny`: mode number in y-direction
- `ax`: vector with spatial axis coordinates
- `f_space_out`:

  `true`: output in spatial-frequency space

  `false`: output in position space

Output values:

- `psi`: Array containing the requested normalized (`nx, ny`) basis function, depending on the boolean value `f_space_out` either in position space or in spatial-frequency space .

Source code:

```matlab
function psi = fft2_basis_func(gpu, nx, ny, ax, f_space_out)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Creates the normalized (nx, ny) Fast-Fourier basis function
% either in position space or in spatial-frequency space
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Input parameters:
% gpu   ... 0: use CPU, 1: use GPU, 2: use mp library
% nx    ... mode number in x-direction
% ny    ... mode number in y-direction
% ax    ... vector with spatial axis coordinates in m
% f_space_out .. if true, output in f-space, otherwise in position space
%
% Output:
% psi .. normalized basis function
% normalization condition:
% if ~f_space_out: trapz(ax, trapz(ax, psi .* conj(psi), 2)) == 1
% if  f_space_out: norm(psi) == 1
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
N = length(ax); % side length (number of steps along each axis)

% create empty basis array in f-space
if gpu == 2
    psi = zeros(N, 'mp');
elseif gpu == 1
    psi = zeros(N, 'gpuArray');
else
    psi = zeros(N);
    ax = gather(ax);
end

c = floor(N/2)+1; % row- and column index of the nx=ny=0 cell
if c-nx>=1 && c-nx <= N && c-ny>=1 && c-ny <= N
    psi(c-ny, c-nx)=1; % create normalized basis function in f-space
end

if ~f_space_out
    % output in position space required:
    % convert to spatial coordinates by means of inverse FFT
    psi = ifft2_phys_spatial(psi, ax);
end
```

C.5. create_k_ax

The function `create_k_ax(gpu, ax)` creates a vector containing wave numbers matching the axis of the FFT-transformed position-space-array (i.e., the spatial-frequency-space-array), as explained in section 2.2.2.

Input values:

- `gpu`: 0: use CPU, 1: use GPU, 2: use mp library
- `ax`: vector with spatial axis coordinates

Return values:

- `k_ax`: wave number coordinate vector

Source code:

```
 1 function  k_ax = create_k_ax(gpu, ax)
 2 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 3 % Creates a vector with the same number of entries as the
 4 % ax vector, but containing wave numbers matching the
 5 % axis of the FFT-transformed position-space-array
 6 % (i.e., the spatial-frequency-space-array)
 7 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 8 % Input parameters:
 9 % gpu ... 0: use CPU, 1: use GPU, 2: use mp library
10 % ax    ... vector with position-space coordinates in m
11 %
12 % Return value:
13 % k_ax   ... wave number coordinate vector
14 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
15 
16 N = size(ax,2);         % number of entries in ax vector
17 if gpu == 2
18     onehalf = mp('0.5');
19     one = mp('1');
20     N = mp(N);
21     Nhalf = N/mp('2');
22     twopi = mp('2*pi');
23 else
24     onehalf = 0.5;
25     one = 1;
26     Nhalf = N/2;
27     twopi = 2*pi;
28 end
29 even = (rem(N,2)==0); % true, if N is even
30 % Create vector with all possible mode numbers, beginning with highest
31 if even
32     n = (Nhalf:-one:-Nhalf+onehalf);
33 else
34     n = (Nhalf-onehalf:-one:-Nhalf+onehalf);
35 end
36 L = ax(end)-ax(1);     % reduced length of axis in position-space
37 L = L*N/(N-one);       % nominal length of axis in position-space
38 k_ax = n * twopi/L;    % calculate k-space coordinate vector
```

Lines 17-22 ensure full precision when using the `mp` library. In line 37 the reduced axis length is converted into the nominal axis length (see section 2.2.2 for more information). The formula used in line 38 corresponds to equations (2.54) and (2.55).

C.6. k_vec_tilt

The function `k_vec_tilt(gpu, nx, ny, ax, lambda)` returns the angle with which the k-vector is inclined to the z-axis given `nx` and `ny`.

Input values:

- `gpu`: 0: use CPU, 1: use GPU, 2: use mp library
- `nx`: mode number in x-direction
- `ny`: mode number in y-direction
- `ax`: vector with spatial axis coordinates
- `lambda`: wavelength in propagation direction

Output value:

- `alpha`: angle between $\vec{k}$-vector and z-axis.

Source code:

```
 1 function alpha = k_vec_tilt(gpu, nx, ny, ax, lambda)
 2 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 3 % returns the angle with which the k-vector is inclined to
 4 % the z-axis given nx and ny
 5 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 6 % Input parameters:
 7 % gpu    ...  0: use CPU, 1: use GPU, 2: use mp library
 8 % nx     ...  mode number in x-direction
 9 % ny     ...  mode number in y-direction
10 % ax     ...  vector with spatial axis coordinates in m
11 % lambda .  wavelength in propagation direction
12 %
13 % Output:
14 % alpha .. The angle with which the k-vector is inclined to the z-axis
15 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
16 if gpu==2
17     onepi = mp('pi');
18     twopi = mp('2*pi');
19     nx = mp(nx);
20     ny = mp(ny);
21 else
22     onepi = pi;
23     twopi = 2*pi;
24 end
25 N = size(ax,2);        % number of entries in ax vector
26 L = ax(end)-ax(1);     % reduced length of axis
27 L = L*N/(N-1);         % nominal length of axis
28 k0 = twopi/lambda;     % length of total k-vector
29 kx = twopi*nx/L;       % length of kx vector
30 ky = twopi*ny/L;       % length of ky vector
31 if k0^2-kx^2-ky^2>=0
32     alpha = acos(sqrt(k0^2-kx^2-ky^2)/k0);
33 else
34     alpha = onepi;
35 end
```

Lines 16-20 ensure full precision when using the `mp` library. In line 26 the reduced axis length is converted into the nominal axis length (see section 2.2.2 for more information). The formula used in line 32 is derived in section 2.2.4. Line 34 will be reached if there is no propagation in z-direction anymore.

C.7. sorted_mode_numbers

The function `sorted_mode_numbers(gpu, N)` returns a vector with all combinations of n_x and n_y allowed for an array with side-length `N`. The entries are sorted according to the angle that the associated $\vec{k}$-vectors have with respect to the z-axis.

Input values:

- `gpu`: 1: use GPU, else: use CPU
- `N`: side length of array

Output value:

- `result`: vector with all combinations of n_x and n_y allowed for an array with side-length `N`.

Source code:

```matlab
function result = sorted_mode_numbers(gpu, N)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Returns a vector with all combinations of nx and ny up to n_max
% ordered by increasing angle of corresponding k-vectors with respect
% to the z-axis
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Input parameters:
% gpu     : 1: use GPU, else: use CPU
% N       : side length of array
%
% Output:
% result: vector with all combinations of nx and ny up to n_max
%         (first column: nx-values, second column: ny-values)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
n_max = floor(N/2);  % maximum value for nx, ny
odd = (rem(N,2)~=0); % if true, then N is odd
% (1) create result column vector with three columns and as many lines as
%     required for all possible nx/ny combinations
% (2) create temp line vector with all values nx or ny can take on
if odd
    if gpu == 1
        result = zeros((1+n_max*2)^2,3, 'gpuArray');
        temp = gpuArray([-n_max:n_max]);
    else
        result = zeros((1+n_max*2)^2,3);
        temp = [-n_max:n_max];
    end
else
    if gpu == 1
        result = zeros((n_max*2)^2,3, 'gpuArray');
        temp = gpuArray([-n_max+1:n_max]);
    else
        result = zeros((n_max*2)^2,3);
        temp = [-n_max+1:n_max];
    end
end
% Fill first column with increasing nx-values
% so that each nx-value is repeated n times
result(:,1) = repelem(temp,width(temp))';
% Fill second column with ny-values so that the first two columns
% are now filled with all possible nx and ny combinations
result(:,2) = repmat(temp,1,width(temp))';
% Fill third column with nx^2+ny^2
result(:,3) = result(:,1).^2+result(:,2).^2;
% Sort primarily by the third column, and if ambiguous,
% subsequently by the first and second column.
result = sortrows(result,[3,1,2]);
% return first and second column as result
result = result(:,1:2);
```

C.8. fft2_arr_to_vec

The function `fft2_arr_to_vec(input_array, ax, mode_numbers, f_space_in)` converts `input_array`, which may be a position-space-array or a spatial-frequency-space-array, into a Fourier-coefficient-vector with the coefficients ordered according to the `mode_numbers` index vector. The boolean parameter `f_space_in` determines whether `input_array` is a spatial-frequency-space-array, or a position-space-array.

Input values:

- `input_array`: spatial-frequency-space or position-space input array
- `ax`: vector with spatial axis coordinates
- `mode_numbers`: column vector with combinations of n_x and n_y, determining which modes are to be considered and also determining the order of the entries in the generated Fourier-coefficients vector
- `f_space_in`:

  if `true`: `input_array` is a spatial-frequency-space-array

  if `false`: `input_array` is a position-space-array.

Output value:

- `FFT_vec`: vector with Fourier coefficients, ordered according to the `mode_numbers` vector

Source code:

```
function FFT_vec = fft2_arr_to_vec(input_array, ax, mode_numbers, f_space_in)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Converts position-space-array or spatial-frequency-space-array
% into a Fourier-coefficient-vector with the coefficients ordered
% according to the provided  mode_numbers input vector
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Input:
% input_array .. spatial-frequency-space or position-space input array
% ax ........... vector with spatial axis coordinates in m
% mode_numbers . vector with ny, ny quantum numbers in a specific order
% f_space_in ... if true:  FFT_arr is a spatial-frequency-space-array,
%                if false: FFT_arr is a position-space-array.
%
% Output:
% FFT_vec ...... vector with Fourier coefficients, ordered according to
%                mode_numbers vector
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
N = width(input_array);
% Determine c so that A(c,c) corresponds to nx=ny=0 (A being f-space-array)
c = floor(N/2)+1;

if ~f_space_in
    % position-space-array as input ->
    % convert to spatial-frequency-space-array first
    input_array = fft2_phys_spatial(input_array, ax);
end
% convert nx, ny values from mode_numbers vector into
% linear index vector for the input array
idx_vec = (c-mode_numbers(:,1)-1)*N + c-mode_numbers(:,2);
% convert array into vector according to sort index
FFT_vec = input_array(idx_vec);
```

C.9. fft2_vec_to_arr

The function `fft2_vec_to_arr(gpu, FFT_vec, ax, mode_numbers, f_space_out)` converts the Fourier-coefficient vector `FFT_vec` into a position-space-array or a spatial-frequency-space-array. The boolean parameter `f_space_out` determines, whether the output is a spatial-frequency-space-array, or a position-space-array.

Input values:

- `gpu`: 0: use CPU, 1: use GPU, 2: use mp library
- `FFT_vec`: Vector with Fourier coefficients, ordered according to the `mode_numbers` vector
- `ax`: vector with spatial axis coordinates
- `mode_numbers`: column vector with combinations of n_x and n_y, determining which modes are to be considered and also determining how to interpret the order of the entries in the `FFT_vec` input vector.
- `f_space_out`:

  if `true`: The output is a spatial-frequency-space-array

  if `false`: The output is a position-space-array.

Output value:

- `FFT_arr`: spatial-frequency-space or position-space output array (according to `f_space_out`)

Source code:

```matlab
function FFT_arr = fft2_vec_to_arr(gpu, FFT_vec, ax, mode_numbers, f_space_out)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Converts the Fourier-coefficient vector FFT_vec into a
% position-space-array or a spatial-frequency-space-array.
% The entries in the input vector FFT_vec are associated to the
% according nx/ny-modes by means of the mode_numbers vector
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Input:
% FFT_vec ...... vector with Fourier coefficients, ordered according to
%                mode_numbers vector
% ax ........... vector with spatial axis coordinates in m
% mode_numbers . vector with ny, ny quantum numbers
% f_space_out .. if true:  FFT_arr is a spatial-frequency-space-array,
%                if false: FFT_arr is a position-space-array.
%
% Output:
% FFT_arr ...... spatial-frequency-space or position-space output array
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% side-length of output array
N = width(ax);
% the input vector FFT_vec is probably smaller than the NxN output array
% n is the sidelength the output array would have if its was matching FFT_vec
n = sqrt(height(FFT_vec));
% Determine c so that A(c,c) corresponds to nx=ny=0 (A being f-space-array)
c = floor(N/2)+1;

% create empty NxN output array
if gpu == 2
    FFT_arr = zeros(N, 'mp');
elseif gpu == 2
    FFT_arr = zeros(N, 'gpuArray');
else
    FFT_arr = zeros(N);
end

% convert nx, ny values from mode_numbers vector into
% linear index vector for the NxN output array
idx_vec = (c-mode_numbers(:,1)-1)*N + c-mode_numbers(:,2);
% insert the values from FFT_vec at the corresponding positions in FFT_arr
FFT_arr(idx_vec) = reshape(FFT_vec,n,n);
if ~f_space_out
    % output in position-space requested -> inverse FFT
    FFT_arr = ifft2_phys_spatial(FFT_arr, ax);
end
```

C.10. RSTF_prop

The function `RSTF_prop(gpu, E_in, ax, z, lambda, n, f_space_in, f_space_out)` takes the xy-input field `E_in`, which can be either a position-space-array or a spatial-frequency-space-array, and propagates it in z-direction over the distance `z` using the **Rayleigh-Sommerfeld transfer function method**, as described in section 2.1.2. For propagation in free space, the parameter `n` is to be set $n = 1$, otherwise `n` is the material-dependent (real or complex) refractive index. The returned output field is again optionally either a position-space-array or a spatial-frequency-space-array.

Input values:

- `gpu`: 0: use CPU, 1: use GPU, 2: use mp library
- `E_in`: xy-input-field, can be either a position-space-array or a spatial-frequency-space-array
- `ax`: vector with spatial axis coordinates
- `z`: distance by which `E_in` is to be propagated in z-direction
- `lambda`: empty space wavelength
- `n`: refractive index (`1` for empty space propagation)
- `f_space_in`:

  if `true`: `E_in` is a spatial-frequency-space-array

  if `false`: `E_in` is a position-space-array
- `f_space_out`:

  if `true`: output as a spatial-frequency-space-array

  if `false`: output as a position-space-array

Output value:

- `E_out`: spatial-frequency-space or position-space output array (according to `f_space_out`), containing the propagated field

Source code:

```
 1 function E_out=RSTF_prop(gpu, E_in, ax, z, lambda, n, f_space_in, f_space_out)
 2 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 3 % Propagates the input field E_in in z-direction
 4 % Rayleigh Sommerfeld transfer function approach.
 5 % see: Voelz D, "Computational Fourier optics", (4.19)
 6 % requires z >> lambda
 7 % propagation only if sqrt(kx^2+ky^2)<2*pi/lambda
 8 %
 9 % Input parameters:
10 % gpu   ...... 0: use CPU, 1: use GPU, 2: use mp library
11 % E_in  ...... input field (in either position-space or f-space)
12 % ax  ........ vector with spatial axis coordinates in m
13 % z ......... propagation distance in z-direction
14 % lambda .... empty space wavelength
15 % n ......... refractive index
16 % f_space_in  if true:  E_in is a spatial-frequency-space array
17 %             if false: E_in is a position-space array
18 % f_space_out if true:  output is a spatial-frequency-space array
19 %             if false: output  is a position-space array
20 % Output:
21 % E_out ..... field after propagation in f-space or position-space-array
22 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
23 kxy = create_k_ax(gpu, ax); % axes coordinates in k-space
24 [kx ky] = meshgrid(kxy);    % coordinate grid in k-space
25 if gpu==2
26     % high-precision constants when using mp library
27     k_tot = mp('2 * pi') / lambda ;
28     two = mp('2');
29 else
30     % double-precision constants when using CPU or GPU
31     k_tot = 2 * pi / lambda ;
32     two = 2;
33 end
34 % calculate kz-vector
35 kz = sqrt(complex(k_tot^two - kx.^two - ky.^two));
36 % angular spectrum of propagator in z-direction
37 angs_prop_z = exp(complex(i * n * kz * z));
38 
39 if ~f_space_in
40     % input not in f-space, convert to f-space
41     E_in = fft2_phys_spatial(E_in, ax);
42 end
43 
44 % Apply angular spectrum of propagator in f-space
45 E_out = E_in .* angs_prop_z ;
46 
47 if ~f_space_out
48     % output not in f-space, convert to position-space
49     E_out = ifft2_phys_spatial(E_out, ax);
50 end
```

Lines 25-28 ensure full numerical precision when using the `mp` library. For the theory of the propagator in lines 34-37 see section 2.1.2.

C.11. FRTF_prop

The function `FRTF_prop(gpu, E_in, ax, z, lambda, n, f_space_in, f_space_out)` takes the xy-input field `E_in`, which can be either a position-space-array or a spatial-frequency-space-array, and propagates it in z-direction over the distance `z`, using the **Fresnel transfer function method**, as described in section 2.1.3. For propagation in free space, the parameter `n` is to be set $n = 1$, otherwise `n` is the material-dependent (real or complex) refractive index. The returned output field is again optionally either a position-space-array or a spatial-frequency-space-array.

Input values:

- `gpu`: 0: use CPU, 1: use GPU, 2: use mp library
- `E_in`: xy-input-field, can be either a position-space-array or a spatial-frequency-space-array
- `ax`: vector with spatial axis coordinates
- `z`: distance by which `E_in` is to be propagated in z-direction
- `lambda`: empty space wavelength
- `n`: refractive index (`1` for empty space propagation)
- `f_space_in`:

  if `true`: `E_in` is a spatial-frequency-space-array

  if `false`: `E_in` is a position-space-array
- `f_space_out`:

  if `true`: output as a spatial-frequency-space-array

  if `false`: output as a position-space-array

Output value:

- `E_out`: spatial-frequency-space or position-space output array (according to `f_space_out`), containing the propagated field

Source code:

```
 1 function E_out=FRTF_prop(gpu, E_in, ax, z, lambda, n, f_space_in, f_space_out)
 2 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 3 % Propagates the input field E_in in z-direction
 4 % Fresnel transfer function approach.
 5 % (paraxial approximation)
 6 %
 7 % Input parameters:
 8 % gpu   ...... 0: use CPU, 1: use GPU, 2: use mp library
 9 % E_in  ...... input field (in either position-space or f-space)
10 % ax ........ vector with spatial axis coordinates in m
11 % z ......... propagation distance in z-direction
12 % lambda .... empty space wavelength
13 % n ......... refractive index
14 % f_space_in  if true:  E_in is a spatial-frequency-space array
15 %             if false: E_in is a position-space array
16 % f_space_out if true:  output is a spatial-frequency-space array
17 %             if false: output  is a position-space array
18 % Output:
19 % E_out ..... field after propagation in f-space or position-space-array
20 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
21 kxy = create_k_ax(gpu, ax); % axes coordinates in k-space
22 [kx ky] = meshgrid(kxy);    % coordinate grid in k-space
23 if gpu==2
24     % high-precision constants when using mp library
25     k_tot = mp('2*pi') / lambda ;
26     two = mp('2');
27     fourpi = mp('4*pi');
28 else
29     % double-precision constants when using CPU or GPU
30     k_tot = 2 * pi / lambda ;
31     two = 2;
32     fourpi = 4 * pi;
33 end
34 % angular spectrum of propagator in z-direction
35 angs_prop_z = exp(complex(i * z * n * (k_tot - ...
36     lambda * (kx.^two + ky.^two) / fourpi)));
37 
38 if ~f_space_in
39     % input not in f-space, convert to f-space
40     E_in = fft2_phys_spatial(E_in, ax);
41 end
42 
43 % Apply angular spectrum of propagator in f-space
44 E_out = E_in .* angs_prop_z ;
45 
46 if ~f_space_out
47     % output not in f-space, convert to position-space
48     E_out = ifft2_phys_spatial(E_out, ax);
49 end
```

Lines 23-27 ensure full numerical precision when using the `mp` library. For the theory of the propagator in lines 34-36 see section 2.1.3.

C.12. prop

The function `prop(gpu, E_in, TF, ax, z, lambda, n, f_space_in, f_space_out)` takes the xy-input field `E_in`, which can be either a position-space-array or a spatial-frequency-space-array, and propagates it in z-direction over the distance `z`, using **either the Rayleigh-Sommerfeld transfer function method** as described in section 2.1.2, **or the Fresnel transfer function method**, as described in section 2.1.3. The distinction is made by means of the `TF` input parameter. For propagation in free space, the parameter `n` is to be set $n = 1$, otherwise `n` is the material-dependent (real or complex) refractive index. The returned output field is again optionally either a position-space-array or a spatial-frequency-space-array.

Input values:

- `gpu`: 0: use CPU, 1: use GPU, 2: use mp library
- `E_in`: xy-input-field, can be either a position-space-array or a spatial-frequency-space-array
- `TF`: propagation method to be used

  `0`: Rayleigh-Sommerfeld transfer function

  `1`: Fresnel transfer function
- `ax`: vector with spatial axis coordinates
- `z`: distance by which `E_in` is to be propagated in z-direction
- `lambda`: empty space wavelength.
- `n`: refractive index (`1` for empty space propagation).
- `f_space_in`:

  if `true`: `E_in` is a spatial-frequency-space-array

  if `false`: `E_in` is a position-space-array.
- `f_space_out`:

  if `true`: output as a spatial-frequency-space-array

  if `false`: output as a position-space-array.

Output value:

- `E_out`: spatial-frequency-space or position-space output array (according to `f_space_out`), containing the propagated field

Source code:

```
function E_out=prop(gpu, E_in, TF, ax, z, lambda, n, ...
    f_space_in, f_space_out)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Propagates the input field E_in in z-direction
% either with Rayleigh Sommerfeld transfer function approach
% or with Fresnel Transfer Function approach
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Input parameters:
% gpu   ......  0: use CPU,  1: use GPU,  2: use mp library
% E_in  ......  input field (in either position-space or f-space)
% TF ........   0: use Rayleigh Sommerfeld Transfer fuction
%               1: use Fresnel Transfer Function
% ax ........   vector with spatial axis coordinates in m
% z .........   propagation distance in z-direction
% lambda ....   empty space wavelength
% n .........   refractive index
% f_space_in    if true:  E_in is a spatial-frequency-space array
%               if false: E_in is a position-space array
% f_space_out   if true:  output is a spatial-frequency-space array
%               if false: output  is a position-space array
% Output:
% E_out .....   field after propagation in f-space or position-space-array
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if TF == 0
    E_out = RSTF_prop(gpu, E_in, ax, z, lambda, n, f_space_in, f_space_out);
else
    E_out = FRTF_prop(gpu, E_in, ax, z, lambda, n, f_space_in, f_space_out);
end
```

C.13. lens

The function `lens(gpu, ax, lambda, pupil, NA, f, lens_type)` returns the phase-mask of a thin spherical lens or perfect aspherical lens in a position-space-array `lens_mask`. This array can be applied by simply multiplying it component-wisely to an input-field-array in position-space.

Input values:

- `gpu`: 0: use CPU, 1: use GPU, 2: use mp library
- `ax`: vector with spatial axis coordinates
- `lambda`: empty space wavelength
- `pupil`: if `true`, a pupil is simulated, otherwise not
- `NA`: numerical aperture of the pupil (if activated)
- `f`: focal length
- `lens_type`: 1: thin spherical lens, 2: thin, perfect aspherical lens.

Output values:

- `lens_mask`: Phase-mask of the lens. To be applied to an input-field given in a position-space-array by component-wise multiplication: `.* lens_mask`.
- `lens_pupil`: position-space-array containing the pupil-mask. If `lens_mask == false`, or `NA` is very large, then this array contains only ones.

Source code:

```matlab
 1 function [lens_mask, lens_pupil] = ...
 2          lens(gpu, ax, lambda, pupil, NA, f, lens_type)
 3 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 4 % Returns the phase-shift-mask of a thin spherical or perfect aspherical lens
 5 % see: https://www.iue.tuwien.ac.at/phd/kirchauer/node51.html
 6 % see: https://en.wikipedia.org/wiki/Thin_lens
 7 % wave optics: https://onlinelibrary.wiley.com/doi/10.1002/0471213748.ch2
 8 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 9 %% Input values:
10 % gpu ....... 0: use CPU, 1: use GPU, 2: use mp library
11 % ax ........ vector with spatial axis coordinates in m
12 % lambda .... ermpty space wavelength
13 % pupil ..... acitivate pupil true/false
14 % NA ........ numerical aperture
15 % f ......... focal length of lens
16 % lens_type . 1: thin spherical lens, 2: thin perfect aspherical lens
17 %
18 % Outputs:
19 % lens_mask .. apply to position-space array with .* lens_mask;
20 % lens_pupil . masks off the area outside the lens
21 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
22 [X,Y] = meshgrid(ax); % coordinate grid in xy-position-space
23 if gpu==2
24     % high-precision constants when using mp library
25     k0 = mp('2*pi')/lambda;
26     two = mp('2');
27 else
28     % double-precision constants when using CPU or GPU
29     k0 = 2*pi/lambda;
30     two = 2;
31 end
32
33 R = sqrt(X.^two+Y.^two);  % radial coordinate grid
34 lens_R = f*tan(asin(NA)); % lens aperture radius
35 if pupil
36     % pupil activated: return real pupil mask
37     lens_pupil = (R<lens_R);
38 else
39     % pupil not activated: return just neutral "ones" mask
40     if gpu == 2
41         lens_pupil = ones(width(ax), 'mp');
42     elseif gpu==1
43         lens_pupil = ones(width(ax), 'gpuArray');
44     else
45         lens_pupil = ones(width(ax));
46     end
47 end
48
49 % Create lens mask
50 if lens_type == 1
51     % thin spherical lens
52     lens_mask = exp(-i * k0/(two*f) .* (X.^two + Y.^two));
53 else
54     % thin aspherical perfect lens
55     lens_mask = exp(-i * k0 .* (sqrt(f^two + X.^two + Y.^two)-f));
56 end
57
58 % Apply pupil mask
59 if pupil
60     lens_mask = lens_mask .* lens_pupil;
61 end
```

Lines 23-26 ensure full numerical precision when using the `mp` library. For the theory of the lens simulations in lines 49-56 see sections 2.1.4 and 2.1.5.

C.14. tilt

The function `tilt(gpu, ax, lambda, alpha)` returns a phase-mask which tilts the wavefront by the angle `alpha` against the z-axis in the yz-plane. The returned phase-mask-array can be applied by simply multiplying it component-wisely to an input-field-array in position-space.

Input values:

- `gpu`: 0: use CPU, 1: use GPU, 2: use mp library
- `ax`: vector with spatial axis coordinates
- `lambda`: empty space wavelength.
- `alpha`: angle by which the wavefront should be tilted against the z-axis in the yz-plane.

Output values:

- `tilt_mask`: Phase-mask of the tilt. To be applied to an input-field given in a position-space-array by component-wise multiplication: `.* tilt_mask`.

Source code:

```
 1 function tilt_mask = tilt(gpu, ax, lambda, alpha)
 2 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 3 % Returns the phase-shift-mask which tilts the wavefront with alpha
 4 % in the yz plane
 5 % see: D. Voelz: Computational Fourier Optics, p.89ff
 6 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 7 %% Input values:
 8 % gpu ....... 0: use CPU, 1: use GPU, 2: use mp library
 9 % ax ........ vector with spatial axis coordinates in m
10 % lambda .... empty space wavelength
11 % alpha ..... angle by which the wavefront should be tilted in the yz plane
12 %
13 % Outputs:
14 % tilt_mask .. apply to position-space array with .* tilt_mask;
15 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
16 
17 [one, two, four, eight, twopi]  = precision_constants(gpu);
18 [X,Y] = meshgrid(ax); % simulation grid
19 k0 = twopi/lambda;
20 tilt_mask = exp(i * k0 * Y * tan(alpha)); % tilt mask
```

The theory leading to the phase-mask formula in line 20 is explained in section 4.4.4.

C.15. opt_grid_params

The main input parameters of this function are the desired side-length of the "guarded" input-grid (`L_des`, corresponding to L_1 in figure 2.12), a `factor` defining how much larger the "guarding" grid should be (side-length L_2 in figure 2.12), and the propagation distance `z`. As these input values generally do *not* result in an integer value for the optimal number of discrete coordinate points along the x- and y-axis according to equation (2.71), the best-matching integer value for N_1 is calculated, and returned together with the corresponding side-length L_1 (close to `L_des`).

Equivalently, it is ensured that the larger side-length L_2 of the "guarding" area exactly matches an integer value N_2, and these two values are also returned. The boolean parameter `even` allows to force N_1 and N_2 to be even, or odd.

Further, a parameter `z_max` has to be passed to the function. It indicates the longest propagation distance in the simulation (in case that the propagation function is applied multiple times in a row). This input allows the function to also return the maximum tilt angle α_{max} and the maximum transverse mode number n_{max} according to equations (2.73) and (2.74).

Input values:

- `L_des`: desired side-length of the embedded (smaller) "guarded" grid
- `factor`: factor by which the side length of the embedding "guarding" grid is larger than the "guarded" grid
- `z`: propagation distance (single propagation)
- `z_max`: maximum propagation distance (i.e., the total maximum distance in case multiple propagations in a row are simulated)
- `even`: if `true`: the return values `N1` and `N2` are even numbers; if `false`: the return values `N1` and `N2` are odd numbers
- `lambda`: wavelength in empty space

Output values:

- `N1`: number of discrete coordinate points along the x- and y-axis of the (smaller) "guarded" grid
- `L1`: side-length of the (smaller) "guarded" grid (close to `L_des`)
- `N2`: number of discrete coordinate points along the x- and y-axis of the (larger) "guarding" grid

- `L2`: side-length of the (larger) “guarding” grid
- `alpha_max`: maximum tilt angle α_{max} (as depicted in figure 2.12)
- `n_max`: maximum allowed n_x and n_y mode numbers

Source code:

```
function [N1, L1, N2, L2, alpha_max, n_max] = ...
    opt_grid_params(L_des, factor, z, z_max, even, lambda)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Calculates the optimal grid parameters for propagation distance z
% L_des is the side length of the input grid
% To allow for diffraction waves to be detected after propagation,
% the input grid is embedded in a larger grid, which sides are larger
% by "factor". The grid after propgation is then large enough to still
% detect the diffracted waves, provided all waves fulfill ny=ny<n_krit.
%
% Input parameters:
% L_des ........ desired side-length of (smaller) "guarded" grid
% factor ....... factor by which the embedding grid's sides are larger
% z ........... propagation distance (single propagation)
% z_max ........ maximum propagation distance (if multiple propagations)
% even ......... If true: even number of steps, if false: odd number
% lambda ....... wave length in empty space
%
% Outputs:
% N1 ........... Number of steps, embedded (smaller) grid
% L1 ........... Side-length of embedded (smaller) grid
% N2 ........... Number of steps, embedding (larger) grid
% L2 ........... Side-length of embedding (larger) grid
% alpha_max .... maximum angle for propagating rays
% n_max ........ maximum allowed ny ny mode numbers
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% estimation for the side-length of the larger grid
L2 = L_des * factor;
% best-matching integer optimal number of steps for L2
N2 = round(L2^2/(lambda*z));
if even
    % even number of steps demanded
    if rem(N2,2)~=0
        % make N2 even
        N2 = N2 + 1;
    end
else
    % odd number of steps demanded
    if rem(N2,2)==0
        % make N2 odd
        N2 = N2 + 1;
    end
end
% back-calculate exact side-length L2 of the larger grid matching N2
L2 = sqrt(N2*lambda*z);

N1 = round(N2/factor); % number of steps for smaller grid
if even
    % even number of steps demanded
    if rem(N1,2)~=0
        % make N1 even
        N1 = N1 + 1;
    end
else
    % odd number of steps demanded
    if rem(N1,2)==0
        % make N1 odd
        N1 = N1 + 1;
    end
end
L1 = L2*N1/N2; % exact side-length of smaller grid
s = (L2-L1)/2; % excess side length in one axis direction
alpha_max = atan(s/z_max); % critical angle
n_max = floor(L1/lambda*sin(alpha_max));
```

C.16. embed_image

Function `embed_image(gpu, E_in_small, ax_large)` takes the smaller position-space-grid `E_in_small` and embeds it in a larger grid whose side-length is defined by `ax_large` (see figure 2.12).

Input values:

- `gpu`: 0: use CPU, 1: use GPU, 2: use mp library
- `E_in_small`: “smaller”, to-be-embedded position-space array
- `ax_large`: Vector with spatial axis coordinates of larger grid. Alternatively, `ax_large` can be an integer value. If that is the case, it defines the N_2 value, i.e., the number of discrete coordinate points along the axes of the larger grid.

Output value:

- `E_out_large`: larger “guarding” grid, embedding the input-grid in its center

Source code:

```matlab
function E_out_large  = embed_image(gpu, E_in_small, ax_large)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Embeds the smaller (position-space) grid E_in_small
% in a larger "guarding" grid.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Input parameters:
% gpu  ......... 0: use CPU, 1: use GPU, 2: use mp library
% E_in_small ... "smaller", to-be-embedded position-space array
% ax_large ..... coordinate vector of larger grid.
%                Alternatively ax_large can be an integer value.
%                It that it the case, it defines the N2 value, i.e., the
%                number of discrete coordinate points along the axes
% Output:
% e_out_large .. larger grid, embedding the input-grid in its center
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
s_small = size(E_in_small);
s_large = size(ax_large);
pixels_large = s_large(2);
pixels_small = s_small(2);
if pixels_large==1
    % if ax is int value instead of a vector -> just interpret as N2 value
    pixels_large = ax_large;
end
offs = (pixels_large-pixels_small)/2+1;
if gpu == 2
    E_out_large=zeros(pixels_large, 'mp');
elseif gpu==1
    E_out_large=zeros(pixels_large, 'gpuArray');
else
    E_out_large=zeros(pixels_large);
end
E_out_large(offs:offs+pixels_small-1,offs:offs+pixels_small-1) = E_in_small;
```

C.17. extract_center_image

Function `extract_center_image(E_in_large, ax_small)` extracts the smaller sub-grid from the center of position-space-grid `E_in_large`. The size of the smaller grid is defined by `ax_small` (see figure 2.12).

Input values:

- `gpu`: 0: use CPU, 1: use GPU, 2: use mp library
- `E_in_large`: “larger” position-space grid in which the smaller grid is embedded
- `ax_small`: Vector with spatial axis coordinates of smaller grid. Alternatively, `ax_small` can be an integer value. If that is the case, it defines the N_1 value, i.e., the number of discrete coordinate points along the axes of the smaller grid.

Output value:

- `E_out_small`: smaller grid, extracted from the center of the larger grid

Source code:

```matlab
function E_out_small  = extract_center_image(E_in_large, ax_small)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Extracts the smaller (position-space) grid from the center
% of a larger "guarding" grid.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Input parameters:
% E_in_large ... "larger" embedding position-space array
% ax_small ..... coordinate vector of smaller grid.
%                Alternatively ax_small can be an integer value.
%                It that it the case, it defines the N1 value, i.e., the
%                number of discrete coordinate points along the axes
% Output:
% E_out_small .. smaller grid, extracted from the center of the larger grid
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% Creates an input field with some image
%
%% Inputs:
% E_in_large    : large image in spatial coordinates
% ax2           : spatial coordinate vector with real physical length units
%               : (assuming y is the same) of small image

%% Output:
% E_out     : small image
s_large = size(E_in_large);
s_small = size(ax_small);
pixels_large = s_large(2);
pixels_small = s_small(2);
if pixels_small==1
    % if ax is int value instead of a vector -> just interpret as N1 value
    pixels_small = ax_small;
end
offs = (pixels_large-pixels_small)/2+1;

E_out_small = E_in_large(offs:offs+pixels_small-1,offs:offs+pixels_small-1);
```

C.18. round_trip_no_atten

The function `round_trip_no_atten(gpu, E_in, ax_small, ax_large, TF, lens_mask, f, lambda, f_space_in, f_space_out)` takes the input field `E_in` and sends it on a round-trip through a 4f-cavity with given focal length `f` and perfect rear-mirror, assuming no attenuation. Two coordinate-axis-vectors `ax_small` for the smaller input-grid and `ax_large` for the larger guarding-grid (see figure 2.12) must be provided together with the wavelength `lambda` and an array `lens_mask`, representing the phase-mask of choice for the lenses. The output is again optionally either in position-space or in spatial-frequency-space.

Input values:

- `gpu`: 0: use CPU, 1: use GPU, 2: use mp library
- `E_in`: xy-input-field, can be either a position-space-array or a spatial-frequency-space-array
- `ax_small`: vector with spatial axis-coordinates matching `E_in`
- `ax_large`: vector with spatial axis-coordinates for larger guarding-grid
- `TF`: propagation method to be used

  `0`: Rayleigh-Sommerfeld transfer function

  `1`: Fresnel transfer function
- `lens_mask`: position-space phase-mask of the lenses (dimensions must match `ax_large`)
- `lambda`: empty space wavelength.
- `f`: focal length of the lenses
- `f_space_in`:

  if `true`: `E_in` is a spatial-frequency-space-array

  if `false`: `E_in` is a position-space-array.
- `f_space_out`:

  if `true`: output as a spatial-frequency-space-array

  if `false`: output as a position-space-array.

Return value:

- `E_out`: array containing the resulting field after a single round-trip

Source code:

```
function E_out  = round_trip_no_atten(gpu, E_in, ax_small, ax_large, ...
TF, lens_mask, f, lambda, f_space_in, f_space_out)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Simulates a single round-trip through a 4f-cavity without attenuation
%
% Input parameters:
% gpu   ...... 0: use CPU, 1: use GPU, 2: use mp library
% E_in  ...... input field (in either position-space or f-space)
% ax_small .. vector with spatial axis coordinates matching E_in
% ax_large .. vector with spatial axis coordinates for larger guarding grid
% TF ........ 0: use Rayleigh Sommerfeld Transfer function
%             1: use Fresnel Transfer Function
% lens_mask . position-space phase-mask of the lenses
% f ......... focal length
% lambda .... empty space wavelength
% f_space_in  if true:  E_in is a spatial-frequency-space array
%             if false: E_in is a position-space array
% f_space_out if true:  output is a spatial-frequency-space array
%             if false: output  is a position-space array
%
% Output:
% E_out ..... field after propagation in f-space or position-space-array
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% prepare input
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if f_space_in
    % if input in f-space, convert to position-space
    E_in = ifft2_phys_spatial(E_in, ax_small);
end
E = embed_image(gpu, E_in, ax_large); % embed input in "large" grid

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% "left-to-right" first trip
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% propagate to first lens
E = prop(gpu, E, TF, ax_large, f, lambda, 1, false, false);
% apply lens mask
E = E .* lens_mask;
% propagate 2f distance
E = prop(gpu, E, TF, ax_large, f, lambda, 1, false, true);
E = prop(gpu, E, TF, ax_large, f, lambda, 1, true, false);
% apply lens mask
E = E .* lens_mask;
% proagate f
E = prop(gpu, E, TF, ax_large, f, lambda, 1, false, false);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Phase-shift on back-mirror
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
E = -E;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% "right-to-left" return trip
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% propagate to first lens
E = prop(gpu, E, TF, ax_large, f, lambda, 1, false, false);
% apply lens mask
E = E .* lens_mask;
% propagate 2f distance
E = prop(gpu, E, TF, ax_large, f, lambda, 1, false, true);
E = prop(gpu, E, TF, ax_large, f, lambda, 1, true, false);
% apply lens mask
E = E .* lens_mask;
% proagate f
E = prop(gpu, E, TF, ax_large, f, lambda, 1, false, false);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% prepare output
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
E_out = extract_center_image(E, ax_small); % extract smaller center image
if f_space_out
    % if output in f-space, convert from position-space to to f-space
    E_out = fft2_phys_spatial(E_out, ax_small);
end
```

C.19. round_trip_no_atten2

The function `round_trip_no_atten2` is an enhanced version of the function `round_trip_no_atten` as described in appendix C.18. It allows to separately define the distances between the partially reflective mirror and the first lens, between the first lens and the second lens, and between the second lens and the total reflective, perfect back-mirror.

Input values:

- `gpu`: 0: use CPU, 1: use GPU, 2: use mp library
- `E_in`: xy-input-field, can be either a position-space-array or a spatial-frequency-space-array
- `ax_small`: vector with spatial axis-coordinates matching `E_in`
- `ax_large`: vector with spatial axis-coordinates for larger guarding-grid
- `TF`: propagation method to be used

  `0`: Rayleigh-Sommerfeld transfer function

  `1`: Fresnel transfer function
- `lens_mask`: position-space phase-mask of the lenses (dimensions must match `ax_large`)
- `lambda`: empty space wavelength
- `d1`: distance between partially reflective mirror and first lens
- `d2`: distance between first lens and second lens
- `d3`: distance between second lens and total reflective back-mirror
- `f_space_in`:

  if `true`: `E_in` is a spatial-frequency-space-array

  if `false`: `E_in` is a position-space-array
- `f_space_out`:

  if `true`: output as a spatial-frequency-space-array

  if `false`: output as a position-space-array.

Return value:

- `E_out`: array containing the resulting field after a single round-trip

Source code:

```
function E_out  = round_trip_no_atten2(gpu, E_in, ax_small, ax_large, ...
TF, lens_mask, d1, d2, d3, lambda, f_space_in, f_space_out)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Simulates a single round-trip through a 4f-cavity without attenuation
% the distance between second lens and rear mirror is adjustabble (param d)
%
% Input parameters:
% gpu   ...... 0: use CPU, 1: use GPU, 2: use mp library
% E_in  ...... input field (in either position-space or f-space)
% ax_small .. vector with spatial axis coordinates matching E_in
% ax_large .. vector with spatial axis coordinates for larger guarding grid
% TF ........ 0: use Rayleigh Sommerfeld Transfer function
%             1: use Fresnel Transfer Function
% lens_mask . position-space phase-mask of the lenses
% d1 ........ distance between partially reflective mirror and first lens
% d2 ........ distance between first lens and second lens
% d3 ........ distance between second lens and total reflective back-mirror
% lambda .... empty space wavelength
% f_space_in   if true:  E_in is a spatial-frequency-space array
%              if false: E_in is a position-space array
% f_space_out  if true:  output is a spatial-frequency-space array
%              if false: output  is a position-space array
%
% Output:
% E_out ..... field after propagation in f-space or position-space-array
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
d2half = d2 / pval(gpu, '2');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% prepare input
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if f_space_in
    % if input in f-space, convert to position-space
    E_in = ifft2_phys_spatial(E_in, ax_small);
end
E = embed_image(gpu, E_in, ax_large); % embed input in "large" grid

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% "left-to-right" first trip
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% propagate to first lens
E = prop(gpu, E, TF, ax_large, d1, lambda, 1, false, false);
% apply lens mask
E = E .* lens_mask;
% propagate to second lens
E = prop(gpu, E, TF, ax_large, d2half, lambda, 1, false, true);
E = prop(gpu, E, TF, ax_large, d2half, lambda, 1, true, false);
% apply lens mask
E = E .* lens_mask;
% proagate d3 to rear-mirror
E = prop(gpu, E, TF, ax_large, d3, lambda, 1, false, false);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Phase-shift on back-mirror
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
E = -E;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% "right-to-left" return trip
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% propagate back to second lens
E = prop(gpu, E, TF, ax_large, d3, lambda, 1, false, false);
% apply lens mask
E = E .* lens_mask;
% propagate d2 distance
E = prop(gpu, E, TF, ax_large, d2half, lambda, 1, false, true);
E = prop(gpu, E, TF, ax_large, d2half, lambda, 1, true, false);
% apply lens mask again
E = E .* lens_mask;
% propagate d1
E = prop(gpu, E, TF, ax_large, d1, lambda, 1, false, false);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% prepare output
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
E_out = extract_center_image(E, ax_small); % extract smaller center image
if f_space_out
    % if output in f-space, convert from position-space to to f-space
    E_out = fft2_phys_spatial(E_out, ax_small);
end
```

C.20. round__trip__no__atten3

The function `round_trip_no_atten3` is an enhanced version of the function `round_trip_no_atten2` as described in appendix C.19. By employing the `tilt`-function, documented in C.14, it allows to define the angle in degrees by which the total reflective back-mirror is tilted relative to the z-axis in the yz-plane (see lines 57-61 in the following source-code).

Input values:

- `gpu`: 0: use CPU, 1: use GPU, 2: use mp library
- `E_in`: xy-input-field, can be either a position-space-array or a spatial-frequency-space-array
- `ax_small`: vector with spatial axis-coordinates matching `E_in`
- `ax_large`: vector with spatial axis-coordinates for larger guarding-grid
- `TF`: propagation method to be used

  `0`: Rayleigh-Sommerfeld transfer function

  `1`: Fresnel transfer function
- `lens_mask`: position-space phase-mask of the lenses (dimensions must match `ax_large`)
- `lambda`: empty space wavelength
- `d1`: distance between partially reflective mirror and first lens
- `d2`: distance between first lens and second lens
- `d3`: distance between second lens and total reflective back-mirror
- `mirror_tilt`: angle in degrees by which the total reflective back-mirror is tilted relative to the z-axis in the yz-plane
- `f_space_in`:

  if `true`: `E_in` is a spatial-frequency-space-array

  if `false`: `E_in` is a position-space-array.
- `f_space_out`:

  if `true`: output as a spatial-frequency-space-array

  if `false`: output as a position-space-array.

Return value:

- `E_out`: array containing the resulting field after a single round-trip

Source code:

```
function E_out  = round_trip_no_atten3(gpu, E_in, ax_small, ax_large, ...
TF, lens_mask, d1, d2, d3, mirror_tilt, lambda, f_space_in, f_space_out)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Simulates a single round-trip through a 4f-cavity without attenuation
% the distance between second lens and rear mirror is adjustabble (param d)
%
% Input parameters:
% gpu   ...... 0: use CPU, 1: use GPU, 2: use mp library
% E_in  ...... input field (in either position-space or f-space)
% ax_small .. vector with spatial axis coordinates matching E_in
% ax_large .. vector with spatial axis coordinates for larger guarding grid
% TF ........ 0: use Rayleigh Sommerfeld Transfer function
%             1: use Fresnel Transfer Function
% lens_mask . position-space phase-mask of the lenses
% d1 ........ distance between partially reflective mirror and first lens
% d2 ........ distance between first lens and second lens
% d3 ........ distance between second lens and total reflective back-mirror
% mirror_tilt angle in degree by which the backmirror is titled in yz plane
% lambda .... empty space wavelength
% f_space_in  if true:  E_in is a spatial-frequency-space array
%             if false: E_in is a position-space array
% f_space_out if true:  output is a spatial-frequency-space array
%             if false: output is a position-space array
%
% Output:
% E_out ..... field after propagation in f-space or position-space-array
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
d2half = d2 / pval(gpu, '2');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% prepare input
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if f_space_in
    % if input in f-space, convert to position-space
    E_in = ifft2_phys_spatial(E_in, ax_small);
end
E = embed_image(gpu, E_in, ax_large); % embed input in "large" grid

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% "left-to-right" first trip
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% propagate to first lens
E = prop(gpu, E, TF, ax_large, d1, lambda, 1, false, false);
% apply lens mask
E = E .* lens_mask;
% propagate to second lens
E = prop(gpu, E, TF, ax_large, d2half, lambda, 1, false, true);
E = prop(gpu, E, TF, ax_large, d2half, lambda, 1, true, false);
% apply lens mask
E = E .* lens_mask;
% proagate d to rear-mirror
E = prop(gpu, E, TF, ax_large, d3, lambda, 1, false, false);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% back-mirror: Phase-shift and tilt
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
E = -E; % phase-shift
if mirror_tilt ~= 0
    beam_tilt = 2 * mirror_tilt;
    tilt_mask = tilt(gpu, ax_large, lambda, beam_tilt*pi/180);
    E = E .* tilt_mask;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% "right-to-left" return trip
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% propagate back to second lens
E = prop(gpu, E, TF, ax_large, d3, lambda, 1, false, false);
% apply lens mask
E = E .* lens_mask;
% propagate d2 distance
E = prop(gpu, E, TF, ax_large, d2half, lambda, 1, false, true);
E = prop(gpu, E, TF, ax_large, d2half, lambda, 1, true, false);
% apply lens mask again
E = E .* lens_mask;
% propagate d1
E = prop(gpu, E, TF, ax_large, d1, lambda, 1, false, false);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% prepare output
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
E_out = extract_center_image(E, ax_small); % extract smaller center image
if f_space_out
    % if output in f-space, convert from position-space to to f-space
    E_out = fft2_phys_spatial(E_out, ax_small);
end
```

C.21. transmision_matrix_round_trip_no_atten

The function `transmision_matrix_round_trip_no_atten(gpu, ax_small, ax_large, TF, modes, lens_mask, f, lambda)` creates all transverse (x,y)-modes encoded in the `modes` input vector and sends these modes on a single round-trip through an unattenuated 4f-cavity with the given parameters. Based on this, a transmission matrix is created and returned. The n-th column of the returned transmission matrix represents the spatial frequency response to the n-th mode according to the `modes` input-vector.

A coordinate-axis-vector `ax_small` for the smaller input-grid and `ax_large` for the larger guarding-grid (see figure 2.12) must be provided together with the wavelength `lambda` and an array `lens_mask`, representing the phase-mask of choice for the lenses.

Input values:

- `gpu`: 0: use CPU, 1: use GPU, 2: use mp library
- `ax_small`: vector with spatial axis-coordinates matching `E_in`
- `ax_large`: vector with spatial axis-coordinates for larger guarding-grid
- `TF`: propagation method to be used

  `0`: Rayleigh-Sommerfeld transfer function

  `1`: Fresnel transfer function
- `modes`: vector with all combinations of n_x and n_y to be considered (first column: n_x-values, second column: n_y-values)
- `lens_mask`: position-space phase-mask of the lenses (dimensions must match `ax_large`)
- `lambda`: empty space wavelength
- `f`: focal length of the lenses

Return value:

- `T`: Transmission matrix. The n-th column represents the spatial frequency response to the n-th mode according to the `modes` input-vector.

Source code:

```
 1 function T =  transmision_matrix_round_trip_no_atten(...
 2     gpu, ax_small, ax_large, TF, modes, lens_mask, f, lambda);
 3 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 4 % Creates a transmission matrix for a single round-trip through a
 5 % 4f-cavity without attenuation
 6 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 7 % Input parameters:
 8 % gpu  ......  0: use CPU, 1: use GPU, 2: use mp library
 9 % ax_small ..  vector with spatial axis coordinates for the area of interest
10 % ax_large ..  vector with spatial axis coordinates for larger guarding grid
11 % TF ........  0: use Rayleigh Sommerfeld Transfer function
12 %              1: use Fresnel Transfer Function
13 % modes .....  vector with all combinations of nx and ny up to be considered
14 %              (first column: nx-values, second column: ny-values)
15 % lens_mask .  position-space phase-mask of the lenses
16 % f .........  focal length
17 % lambda ....  empty space wavelength
18 %
19 % Output:
20 % T .........  Transmission matrix. The n-th column represents the spatial
21 %              frequency response to the n-th mode according to the modes
22 %              input-vector
23 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
24
25 % generate all-zero matrix in a data format matching the gpu parameter
26 T = create_zeros(gpu, size(modes,1));
27 % iterate through all modes
28 parfor idx = 1:height(modes)
29     nx = modes(idx,1);
30     ny = modes(idx,2);
31     % generate (nx,ny) mode
32     psi_in = fft2_basis_func(gpu, nx, ny, ax_small, false);
33     % simulate round-trip
34     psi_out = round_trip_no_atten(gpu, psi_in, ax_small, ax_large, TF, ...
35         lens_mask, f, lambda, false, false);
36     % add column to transmission matrix
37     T(:,idx) = fft_arr_to_vec(psi_out, ax_small, modes, false);
38 end
```

In line 26, an empty (all-zero) `T`-matrix in the data format indicated by the `gpu`-parameter is created (see appendix D.4). The parallelized loop starting in line 28 iterates through all modes encoded in the `modes` input-vector. In line 32, an (n_x,n_y)-mode input-field `psi_in` is generated, and then sent on a single-round-trip through the 4f-cavity in lines 34 and 35 (see appendix C.18). The resulting array in position-space `psi_out` is then converted into a column-vector in spatial-frequency-space, and this vector is added to the matching column of the `T`-matrix (line 37).

C.22. transmision_matrix_round_trip_no_atten2

The function `transmision_matrix_round_trip_no_atten2` is an enhanced version of the function `transmision_matrix_round_trip_no_atten` as described in appendix C.21. It allows to separately define the distances between the partially reflective mirror and the first lens, between the first lens and the second lens, and between the second lens and the total reflective, perfect back-mirror.

Input values:

- `gpu`: 0: use CPU, 1: use GPU, 2: use mp library
- `ax_small`: vector with spatial axis-coordinates matching `E_in`
- `ax_large`: vector with spatial axis-coordinates for larger guarding-grid
- `TF`: propagation method to be used

  `0`: Rayleigh-Sommerfeld transfer function

  `1`: Fresnel transfer function
- `modes`: vector with all combinations of n_x and n_y to be considered (first column: n_x-values, second column: n_y-values)
- `lens_mask`: position-space phase-mask of the lenses (dimensions must match `ax_large`)
- `d1`: distance between partially reflective mirror and first lens
- `d2`: distance between first lens and second lens
- `d3`: distance between second lens and total reflective back-mirror
- `lambda`: empty space wavelength

Return value:

- `T`: Transmission matrix. The n-th column represents the spatial frequency response to the n-th mode according to the `modes` input-vector.

Source code:

```matlab
function T =  transmision_matrix_round_trip_no_atten2(...
    gpu, ax_small, ax_large, TF, modes, lens_mask, d1, d2, d3, lambda);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Creates a transmission matrix for a single round-trip through a
% 4f-cavity without attenuation where the distance of the total
% reflective mirror is adjustable (parameter d)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Input parameters:
% gpu  ...... 0: use CPU, 1: use GPU, 2: use mp library
% ax_small .. vector with spatial axis coordinates for the area of interest
% ax_large .. vector with spatial axis coordinates for larger guarding grid
% TF ........ 0: use Rayleigh Sommerfeld Transfer function
%             1: use Fresnel Transfer Function
% modes ..... vector with all combinations of nx and ny up to be considered
%             (first column: nx-values, second column: ny-values)
% lens_mask . position-space phase-mask of the lenses
% f ......... focal length
% d1 ........ distance between partially reflective mirror and first lens
% d2 ........ distance between first lens and second lens
% d3 ........ distance between second lens and total reflective back-mirror
% lambda .... empty space wavelength
%
% Output:
% T ......... Transmission matrix. The n-th column represents the spatial
%             frequency response to the n-th mode according to the modes
%             input-vector
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% generate all-zero matrix in a data format matching the gpu parameter
T = create_zeros(gpu, size(modes,1));
% iterate through all modes
parfor idx = 1:height(modes)
    nx = modes(idx,1);
    ny = modes(idx,2);
    % generate (nx,ny) mode
    psi_in = fft2_basis_func(gpu, nx, ny, ax_small, false);
    % simulate round-trip
    psi_out = round_trip_no_atten2(gpu, psi_in, ax_small, ax_large, TF, ...
        lens_mask, d1, d2, d3, lambda, false, false);
    % add column to transmission matrix
    T(:,idx) = fft_arr_to_vec(psi_out, ax_small, modes, false);
end
```

C.23. transmision_matrix_round_trip_no_atten3

The function `transmision_matrix_round_trip_no_atten3` is an enhanced version of the function `transmision_matrix_round_trip_no_atten2` as described in appendix C.22. It allows to define the angle in degrees by which the total reflective back-mirror is tilted against the z-axis in the yz-plane.

Input values:

- `gpu`: 0: use CPU, 1: use GPU, 2: use mp library
- `ax_small`: vector with spatial axis-coordinates matching `E_in`
- `ax_large`: vector with spatial axis-coordinates for larger guarding-grid
- `TF`: propagation method to be used

  `0`: Rayleigh-Sommerfeld transfer function

  `1`: Fresnel transfer function
- `modes`: vector with all combinations of n_x and n_y to be considered (first column: n_x-values, second column: n_y-values)
- `lens_mask`: position-space phase-mask of the lenses (dimensions must match `ax_large`)
- `d1`: distance between partially reflective mirror and first lens
- `d2`: distance between first lens and second lens
- `d3`: distance between second lens and total reflective back-mirror
- `mirror_tilt`: angle in degrees by which the total reflective back-mirror is tilted against the z-axis in the yz-plane
- `lambda`: empty space wavelength

Return value:

- `T`: Transmission matrix. The n-th column represents the spatial frequency response to the n-th mode according to the `modes` input-vector.

Source code:

```matlab
function T = transmision_matrix_round_trip_no_atten3(gpu, ax_small, ...
    ax_large, TF, modes, lens_mask, d1, d2, d3, mirror_tilt, lambda);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Creates a transmission matrix for a single round-trip through a
% 4f-cavity without attenuation where the distance of the total
% reflective mirror is adjustable (parameter d)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Input parameters:
% gpu  ...... 0: use CPU, 1: use GPU, 2: use mp library
% ax_small .. vector with spatial axis coordinates for the area of interest
% ax_large .. vector with spatial axis coordinates for larger guarding grid
% TF ........ 0: use Rayleigh Sommerfeld Transfer function
%             1: use Fresnel Transfer Function
% modes ..... vector with all combinations of nx and ny up to be considered
%             (first column: nx-values, second column: ny-values)
% lens_mask . position-space phase-mask of the lenses
% d1 ........ distance between partially reflective mirror and first lens
% d2 ........ distance between first lens and second lens
% d3 ........ distance between second lens and total reflective back-mirror
% mirror_tilt angle in degree by which the backmirror is titled in yz plane
% lambda .... empty space wavelength
%
% Output:
% T ......... Transmission matrix. The n-th column represents the spatial
%             frequency response to the n-th mode according to the modes
%             input-vector
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% generate all-zero matrix in a data format matching the gpu parameter
T = create_zeros(gpu, size(modes,1));
% iterate through all modes
parfor idx = 1:height(modes)
    nx = modes(idx,1);
    ny = modes(idx,2);
    % generate (nx,ny) mode
    psi_in = fft2_basis_func(gpu, nx, ny, ax_small, false);
    % simulate round-trip
    psi_out = round_trip_no_atten3(gpu, psi_in, ax_small, ax_large, TF, ...
        lens_mask, d1, d2, d3, mirror_tilt, lambda, false, false);
    % add column to transmission matrix
    T(:,idx) = fft_arr_to_vec(psi_out, ax_small, modes, false);
end
```

C.24. create_S_matrix

The function `create_S_matrix(gpu, R, T)` creates a scattering matrix based on the input matrices `R` and `T`, which represent the reflection-sub-matrix and the transmission-sub-matrix, respectively. The `R`-matrix goes in the top-left and the bottom-right quadrant of the resulting scattering matrix; and the `T`-matrix goes into the top-right and bottom-left quadrant of the resulting scattering matrix.

The input matrices `R` and `T` must be square and of same size. The output scattering-matrix `S` has twice the side-length than the input-matrices.

Input values:

- `gpu`: 0: use CPU, 1: use GPU, 2: use mp library
- `R`: reflection matrix, must be square
- `T`: transmission matrix, must be square and of same size as `R`

Return value:

- `S`: Scattering matrix. The `R`-matrix goes in the top-left and the bottom-right quadrant of the scattering matrix; and the `T`-matrix goes into the top-right and bottom-left quadrant of the scattering matrix.

Source code:

```
function S = create_S_matrix(gpu, R, T)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Creates a Scattering Matrix S from
% reflection matrix R and transmission matrix T
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Input parameters:
% gpu  ......  0: use CPU, 1: use GPU, 2: use mp library
% R .........  Reflection Matrix
% T .........  Transmission Marix
%
% Output:
% S .........  Scattering Matrix
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

szR = size(R);
szT = size(T);
if ~isequal(szR,szT)
     error('Error! R and T must have same dimensions!')
end
sz = szR;
if sz(1) ~= sz(2)
     error('Error! R and T must be square matrices!')
end

S = create_zeros(gpu, sz(1)*2);
if gpu == 0
     R = gather(R);
     T = gather(T);
end
S(1:sz(1),1:sz(1))=R; % left top quadrant: R
S(1:sz(1),sz(1)+1:2*sz(1))=T; % right top quadrant: T
S(sz(1)+1:2*sz(1),1:sz(1))=T; % left bottom quadrant: T
S(sz(1)+1:2*sz(1),sz(1)+1:2*sz(1))=R; % right bottom quadrant: R
```

C.25. convert_S_to_M

The function `convert_S_to_M(S)` converts the input scattering matrix `S` into the corresponding transfer matrix `M`, using the conversion method described in section 5.3.2.

Input values:

- `S`: scattering matrix, must be square and have even side-length

Return value:

- `M`: corresponding transfer matrix

Source code:

```matlab
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Converts scattering matrix S to transfer matrix M
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Input parameters:
% S ......... Scattering Matrix
%
% Output:
% M ......... Transfer Matrix
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
sz = size(S);
if sz(1) ~= sz(2)
    error('Error! S must be a sqare-shaped matrix!')
end
if rem(sz(1),2) ~= 0
    error('Error! S must have an even side length!')
end
qsz = sz(1)/2; % quadrant side length

% break down S matrix into quadrant matrices
S11 = S(1:qsz,1:qsz); % top left quadrant
S12 = S(1:qsz,qsz+1:2*qsz); % top right quadrant
S21 = S(qsz+1:2*qsz,1:qsz); % bottom left quadrant
S22 = S(qsz+1:2*qsz,qsz+1:2*qsz); % bottom right quadrant

% create Transfer-matrix
M=S;
invS21 = inv(S21);
S11invS21 = S11 * invS21;
M(1:qsz,1:qsz) = S12 - S11invS21 * S22; % top left quadrant
M(1:qsz,qsz+1:2*qsz) = S11invS21; % top right quadrant
M(qsz+1:2*qsz,1:qsz) = -invS21 * S22; % bottom left quadrant
M(qsz+1:2*qsz,qsz+1:2*qsz) = invS21; % bottom right quadrant
```

C.26. convert_M_to_R

The function `convert_M_to_R(M)` converts the input transfer matrix `M` into the top-left quadrant of the corresponding scattering matrix, which is the reflection matrix `R`, based on the theory presented in section 5.3.3.

Input values:

- `M`: transfer matrix, must be square and have even side-length

Return value:

- `R`: top-left quadrant of the corrsponding scattering matrix, which is the reflection matrix.

Source code:

```
function R = convert_M_to_R(M)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Converts transfer matrix M to left upper quadrant of scattering matrix S
% which is the Reflection Matrix R
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Input parameter:
% M ......... Transfer Matrix
%
% Output:
% R ......... Reflection Matrix
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
sz = size(M);
if sz(1) ~= sz(2)
    error('Error! T must be a sqare-shaped matrix!')
end
if rem(sz(1),2) ~= 0
    error('Error! T must have an even side length!')
end
qsz = sz(1)/2; % quadrant side length

% break down T matrix into quadrant matrices
M12 = M(1:qsz,qsz+1:2*qsz); % top right quadrant
M22 = M(qsz+1:2*qsz,qsz+1:2*qsz); % bottom right quadrant

% create R-matrix
R = M12/M22; % top left quadrant of scattering matrix = reflection matrix
```

C.27. transmision_matrix_prop

The function `transmision_matrix_prop(gpu, TF, ax, z, lambda, n, modes)` creates a transmission matrix `T` representing propagation through free space or material (with refractive index n) over a distance `z`.

Input values:

- `gpu`: 0: use CPU, 1: use GPU, 2: use mp library
- `TF`: propagation method to be used
  - `0`: Rayleigh-Sommerfeld transfer function
  - `1`: Fresnel transfer function
- `ax`: vector with spatial axis coordinates
- `z`: propagation distance
- `lambda`: empty space wavelength.
- `n`: complex refractive index (`1` for empty space propagation)
- `modes`: column vector with all combinations of n_x and n_y to be considered (first column: n_x-values, second column: n_y-values)

Output value:

- `T`: Transmission matrix. The n-th column represents the spatial frequency response to the n-th mode according to the `modes` input-vector.

Source code:

```
function T =  transmision_matrix_prop(gpu, TF, ax, z, lambda, n, modes)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Creates a transmission matrix for propagation through free space
% or absorber (with refractive index n)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Input parameters:
% gpu   ...... 0: use CPU, 1: use GPU, 2: use mp library
% TF ........ 0: use Rayleigh Sommerfeld Transfer fuction
%             1: use Fresnel Transfer Function
% ax ........ vector with spatial axis coordinates in m
% z ......... propagation distance in z-direction
% lambda .... empty space wavelength
% n ......... diffraction index
% modes ..... vector with all combinations of nx and ny up to be considered
%             (first column: nx-values, second column: ny-values)
%
% Output:
% T ......... Transmission matrix. The n-th column represents the spatial
%             frequency response to the n-th mode according to the modes
%             input-vector
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% generate all-zero matrix in a data format matching the gpu parameter
T = create_zeros(gpu, size(modes,1));
% iterate through all modes
parfor idx = 1:height(modes)
    nx = modes(idx,1);
    ny = modes(idx,2);
    % generate (nx,ny) mode
    psi_in = fft2_basis_func(gpu, nx, ny, ax, true);
    % simulate propagation
    psi_out = prop(gpu, psi_in, TF, ax, z, lambda, n, true, true);
    % add column to transmission matrix
    T(:,idx) = fft_arr_to_vec(psi_out, ax, modes, true);
end
```

C.28. transmission_matrix_lens

The function `transmission_matrix_lens(gpu, ax, lambda, pupil, NA, f, lens_type, modes)` creates a transmission matrix T representing a thin lens.

Input values:

- `gpu`: 0: use CPU, 1: use GPU, 2: use mp library
- `ax`: vector with spatial axis coordinates
- `lambda`: empty space wavelength
- `pupil`: if `true`, a pupil is simulated, otherwise not
- `NA`: numerical aperture of the pupil (if activated)
- `f`: focal length
- `lens_type`: 1: thin spherical lens, 2: thin, perfect aspherical lens
- `modes`: column vector with all combinations of n_x and n_y to be considered (first column: n_x-values, second column: n_y-values)

Output value:

- `T`: Transmission matrix. The n-th column represents the spatial frequency response to the n-th mode according to the `modes` input-vector.

Source code:

```
function T =  transmision_matrix_lens(gpu, ax, lambda, pupil, NA, f, ...
    lens_type, modes)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Creates a transmission matrix for a thin lens
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Input parameters:
% gpu  ...... 0: use CPU, 1: use GPU, 2: use mp library
% ax ........ vector with spatial axis coordinates in m
% lambda .... ermpty space wavelength
% pupil ..... acitivate pupil true/false
% NA ........ numerical aperture
% f ......... focal length of lens
% lens_type . 1: thin spherical lens, 2: thin perfect aspherical lens
% modes ..... vector with all combinations of nx and ny up to be considered
%             (first column: nx-values, second column: ny-values)
%
% Output:
% T ......... Transmission matrix. The n-th column represents the spatial
%             frequency response to the n-th mode according to the modes
%             input-vector
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% generate all-zero matrix in a data format matching the gpu parameter
T = create_zeros(gpu, size(modes,1));
% iterate through all modes
lens_mask = lens(gpu, ax, lambda, pupil, NA, f, lens_type);
parfor idx = 1:height(modes)
    nx = modes(idx,1);
    ny = modes(idx,2);
    % generate (nx,ny) mode
    psi_in = fft2_basis_func(gpu, nx, ny, ax, false);
    % simulate lens
    psi_out = psi_in .* lens_mask;
    % add column to transmission matrix
    T(:,idx) = fft_arr_to_vec(psi_out, ax, modes, false);
end
```

C.29. downscale__TR__matrix

The function `downscale_TR_matrix(gpu, TR, ax_small, ax_large, modes_large` downscales a transmission or reflection matrix `TR` which corresponds to the larger axis coordinates `ax_large` of the guarding grid to a smaller transmission or reflection matrix corresponding to the smaller observation grid coordinates `ax_small`.

Input values:

- `gpu`: 0: use CPU, 1: use GPU, 2: use mp library
- `TR`: transmission or reflection matrix to be down-scaled, corresponding to the axis coordinates encoded in the `ax_large` coordinate vector.
- `ax_small`: coordinate vector containing the axis coordinates of the smaller observation grid which corresponds to the to-be calculated output transmission or reflection matrix
- `ax_large`: coordinate vector containing the axis coordinates of the larger guarding grid
- `modes`: column vector with all combinations of n_x and n_y encoded in the `TR` input matrix (first column: n_x-values, second column: n_y-values).

Output values:

- `TR_small`: Downscaled transmission or reflection matrix. The n-th column represents the spatial frequency response to the n-th mode according to the `modes_small` output-vector.
- `modes_small`: column vector with all combinations of n_x and n_y encoded in the `TR_small` output matrix (first column: n_x-values, second column: n_y-values).

Source code:

```
function [TR_small, modes_small] = downscale_TR_matrix(...
    gpu, TR, ax_small, ax_large, modes_large)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Downscales Transmission Matrix or Reflection Matrix of larger
% embedding grid to smaller Transmission Matrix or Reflection Matrix
% of observation area
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Input parameters:
% gpu   ........ 0: use CPU, 1: use GPU, 2: use mp library
% TR .......... Tranmission Matrix or Reflection Matrix of "large" Grid
% ax_small .... vector with spatial axis coordinates of small grid
% ax_large .... vector with spatial axis coordinates of large grid
% modes_large .
%
% Output:
% TR_small .... downsized Tranmission Matrix or Reflection Matrix
% modes_small .
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
N_small = width(ax_small);
modes_small = sorted_mode_numbers(gpu, N_small);
TR_small = create_zeros(gpu, size(modes_small,1));
parfor idx = 1:height(modes_small)
    nx = modes_small(idx,1);
    ny = modes_small(idx,2);
    psi_in = fft2_basis_func(gpu, nx, ny, ax_small, false);  % generate (nx,ny) mode
    E = embed_image(gpu, psi_in, ax_large); % embed input in "large" grid
    E_vec = fft_arr_to_vec(E, ax_large, modes_large, false);
    E_vec = TR*E_vec;
    E = fft2_vec_to_arr(gpu, E_vec, ax_large, modes_large, false);
    E = extract_center_image(E, ax_small); % extract smaller center image
    TR_small(:,idx) = fft_arr_to_vec(E, ax_small, modes_small, false);
end
```

D. Helper functions

D.1. create_test_image

The function `create_test_image(gpu, ax, image, f_space_out)` creates an array either in position-space or in spatial-frequency-space, which contains a field representing a test-image. Six different test-images are available.

Input values:

- `gpu`: 0: use CPU, 1: use GPU, 2: use mp library
- `ax`: vector with spatial axis coordinates
- `image`:

  `1`: small centered T

  `2`: large centered T

  `3`: large centered T with a phase gradient

  `4`: large off-center T

  `5`: large off-center T with a phase gradient

  `6`: centered square-beam (1/5 size)
- `f_space_out`:

  if `true`: output as a spatial-frequency-space-array

  if `false`: output as a position-space-array.

Return value:

- `E_out`: array containing the test-image

Source code:

```
function E_out = create_test_image(gpu, ax, image, f_space_out)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Creates a test image
%
% Input parameters:
% gpu  ....... 0: use CPU, 1: use GPU, 2: use mp library:
% ax ......... vector with spatial axis coordinates in m
% image ...... 1: small centered T
%              2: large centered T
%              3: large centered T with a phase gradient
%              4: large off-center T
%              5: large off-center T with a phase gradient
%              6: 1/5 square beam
% f_space_out  if true:  output is a spatial-frequency-space array
%              if false: output  is a position-space array
% Output:
% E_out....... Test image eigter in position-space or f-space
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
pixels = width(ax);
if gpu==2
    E_out = zeros(pixels, 'mp');
elseif gpu == 1
    E_out = zeros(pixels, 'gpuArray');
else
    E_out = zeros(pixels);
end
switch image
    case 1
        % small T
        a = int16(2000/4096*pixels);
        b = int16(2200/4096*pixels);
        c = int16(2020/4096*pixels);
        d = int16(1940/4096*pixels);
        e = int16(2070/4096*pixels);
        E_out(a:b,a:c)=1; % horizontal bar
        E_out(a:c,d:e)=1;  % vertical bar

    case 2
        % large T
        a = int16(1300/4096*pixels);
        b = int16(1700/4096*pixels);
        c = int16((4096-1300)/4096*pixels);
        d = int16(3000/4096*pixels);
        e = int16(1900/4096*pixels);
        f = int16(2200/4096*pixels);
        E_out(a:b,a:c)=1; % horizontal bar
        E_out(a:d,e:f)=1; % vertical bar

    case 3
        % large T with phase shifts
        a = fix(1300/4096*pixels);
        b = fix(1700/4096*pixels);
        c = fix((4096-1300)/4096*pixels);
        d = fix(3000/4096*pixels);
        e = fix(1900/4096*pixels);
        f = fix(2200/4096*pixels);
        % vertical bar
        % line b+1:d, column e:f
        E_out(b+1:d,e:f) =exp(i*repelem(((1:(d-b))')./(d-b)*2*pi,1,f-e+1));
        % horizontal bar
        % line a:b, column a:c
        E_out(a:b,a:c) =exp(i*repelem(1:(c-a+1),b-a+1,1)./(c-a+1)*pi);

    case 4
        % large off-center T
        offset=-900;
        a = fix((1300+offset)/4096*pixels);
        b = fix((1700+offset)/4096*pixels);
        c = fix((4096-1300+offset)/4096*pixels);
        d = fix((3000+offset)/4096*pixels);
        e = fix((1900+offset)/4096*pixels);
        f = fix((2200+offset)/4096*pixels);
        % vertical bar
        % line b+1:d, column e:f
        E_out(b+1:d,e:f) = 1;%exp(i*repelem(((1:(d-b))')./(d-b)*2*pi,1,f-e+1));
        % horizontal bar
        % line a:b, column a:c
        E_out(a:b,a:c) = 1; %exp(i*repelem(1:(c-a+1),b-a+1,1)./(c-a+1)*pi);
    case 5
        % large off-center T with phase shifts
        offset=-900;
        a = fix((1300+offset)/4096*pixels);
        b = fix((1700+offset)/4096*pixels);
        c = fix((4096-1300+offset)/4096*pixels);
        d = fix((3000+offset)/4096*pixels);
        e = fix((1900+offset)/4096*pixels);
        f = fix((2200+offset)/4096*pixels);
        % vertical bar
        % line b+1:d, column e:f
        E_out(b+1:d,e:f) = exp(i*repelem(((1:(d-b))')./(d-b)*2*pi,1,f-e+1));
        % horizontal bar
        % line a:b, column a:c
        E_out(a:b,a:c) = exp(i*repelem(1:(c-a+1),b-a+1,1)./(c-a+1)*pi);

    case 6
        % 1/5 sqare beam
        dx = ax(2)-ax(1);
        w = (max(ax)-min(ax)+dx)/5+dx;
        [X2, Y2] = meshgrid(ax, ax);
        E_out = (abs(X2/w)<=1/2).*(abs(Y2/w)<=1/2);
        if gpu==2
            E_out = mp(E_out);
        elseif gpu==1
            E_out = gpuArray(E_out);
        end

end

if f_space_out
    E_out = fft2_phys_spatial(E_out, ax);
end
```

D.2. plot_fields

The function `plot_fields(gpu, zoom, ax, field1, title1, mode1, field2, title2, mode2, field3, title3, mode3)` plots up to three fields (which must be provided as position-space arrays) next to each other. The axes are labeled according to the coordinate-vector `ax`. Each plot can be assigned a separate title heading, and it can be chosen separately for each plot whether the real-parts, the imaginary-parts, the absolute values, or the squared absolute values are to be plotted.

Input values:

- `gpu`: 0: use CPU, 1: use GPU, 2: use mp library
- `zoom`: zoom-factor
- `ax`: vector with spatial axis coordinates
- `field1`: first field to be plotted (must be provided in position-space)
- `title1`: title heading for the first field
- `mode1`: determines what to display in the first plot (1: real part, 2: imaginary part, 3: absolute value, 4: absolute value squared)
- `field2` (optional): second field to be plotted (must be provided in position-space)
- `title2` (optional): title heading for the second field
- `mode2` (optional): determines what to display in the second plot (1: real part, 2: imaginary part, 3: absolute value, 4: absolute value squared)
- `field3` (optional): third field to be plotted (must be provided in position-space)
- `title3` (optional): title heading for the third field
- `mode3` (optional): determines what to display in the third plot (1: real part, 2: imaginary part, 3: absolute value, 4: absolute value squared)

Source code:

```
function plot_fields(gpu, zoom, ax, field1, title1, mode1, ...
    field2, title2, mode2, field3, title3, mode3)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Plots up to three fields next to each other
% Based on a program by Ori Katz (Hebrew University of Jerusalem)
% extended by Helmut Hoerner (Vienna university of Technology)
%
% Inputs parameters:
%
% gpu   ...... 0: use CPU, 1: use GPU, 2: use mp library
% zoom  ...... zoom factor
% ax ........ vector with spatial axis coordinates in m
% field1 .... matrix with first field to be plotted
% title1 .... text string containing title for first plot
% mode1 ..... 1: disp. real, 2: disp. imag, 3: disp. abs, 4: disp. abs^2
% field2 .... matrix with second field to be plotted
% title2 .... text string containing title for second plot
% mode2 ..... 1: disp. real, 2: disp. imag, 3: disp. abs, 4: disp. abs^2
% field3 .... matrix with third field to be plotted
% title3 .... text string containing title for third plot
% mode3 ..... 1: disp. real, 2: disp. imag, 3: disp. abs, 4: disp. abs^2
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

cm=1e-2; mm=1e-3; um=1e-6; nm=1e-9;

ROI = 2 * abs(complex(ax(1)));
if gpu == 1
    % If gpu, convert gpuArray double to "normal" double
    ROI = gather(ROI);
end
ROI = ROI / zoom;
figure(1)

img_count = 1;
if exist('field2', 'var')
    img_count = img_count + 1;
end
if exist('field3', 'var')
    img_count = img_count + 1;
end

subplot(1,img_count,1);
if mode1 == 1
    imagesc(ax/mm, ax/mm, real(field1));
elseif mode1 == 2
    imagesc(ax/mm, ax/mm, imag(field1));
elseif mode1 == 3
    imagesc(ax/mm, ax/mm, abs(field1));
elseif mode1 == 4
    imagesc(ax/mm, ax/mm, abs(field1).^2);
end
xlabel('(mm)')
ylabel('(mm)')
axis image
title(title1);
axis(ROI/2*[-1 1 -1 1]/mm);


if img_count >= 2
    % plot second image, if provided
    subplot(1,img_count,2);
    if mode2 == 1
      imagesc(ax/mm, ax/mm, real(field2));
    elseif mode2 == 2
        imagesc(ax/mm, ax/mm, imag(field2));
    elseif mode2 == 3
        imagesc(ax/mm, ax/mm, abs(field2));
    elseif mode2 == 4
        imagesc(ax/mm, ax/mm, abs(field2).^2);
    end
    xlabel('(mm)')
    ylabel('(mm)')
    axis image
    title(title2);
    axis(ROI/2*[-1 1 -1 1]/mm);
end
```

```
if img_count >= 3
    % plot third image, if provided
    subplot(1,img_count,3);
    if mode3 == 1
      imagesc(ax/mm, ax/mm, real(field3));
    elseif mode3 == 2
        imagesc(ax/mm, ax/mm, imag(field3));
    elseif mode3 == 3
        imagesc(ax/mm, ax/mm, abs(field3));
    elseif mode3 == 4
        imagesc(ax/mm, ax/mm, abs(field3).^2);
    end
    xlabel('(mm)')
    ylabel('(mm)')
    axis image
    title(title3);
    axis(ROI/2*[-1 1 -1 1]/mm);
end
```

D.3. create_eye

The function `create_eye(gpu, N)` simply creates an N-by-N identity matrix with ones on the main diagonal and zeros elsewhere. The returned matrix has a data-format matching the `gpu` parameter (`Array`, `gpuArray`, or `mp`).

Input values:

- `gpu`: 0: use CPU, 1: use GPU, 2: use mp library
- `N`: number of rows and columns

Return value:

- `T`: identity matrix

  if `gpu` is 0, this is an `Array`

  if `gpu` is 1, this is a `gpuArray`

  if `gpu` is 2, this is a `mp` array

Source code:

```
function I = create_eye(gpu, N)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% creates a N-by_N unity matrix
% in the data format required for the GPU setting
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Input parameters:
% gpu  ......... 0: use CPU, 1: use GPU, 2: use mp library
%
% Output:
% I .. N-by_N unity matrix
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if gpu == 2
    I = eye(N,'mp');
elseif gpu == 1
    I = eye(N,'gpuArray');
else
    I = eye(N);
end
```

D.4. create_zeros

The function `create_zeros(gpu, N)` simply creates an N-by-N array of all zeros. The returned matrix has a data-format matching the `gpu` parameter (`Array`, `gpuArray`, or `mp`).

Input values:

- `gpu`: 0: use CPU, 1: use GPU, 2: use mp library
- `N`: number of rows and columns

Return value:

- `Z`: array with all zeros

  if `gpu` is 0, this is an `Array`

  if `gpu` is 1, this is a `gpuArray`

  if `gpu` is 2, this is a `mp` array

Source code:

```
function Z = create_zeros(gpu, N)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% creates a N-by_N array of all zeros
% in the data format required for the GPU setting
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Input parameters:
% gpu  ......... 0: use CPU, 1: use GPU, 2: use mp library
%
% Output:
% I .. N-by_N array of all zeros
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if gpu == 2
    Z = zeros(N,'mp');
elseif gpu == 1
    Z = zeros(N,'gpuArray');
else
    Z = zeros(N);
end
```

D.5. precision_constants

The function `precision_constants(gpu)` returns the constants 1, 2, 4, 8, and 2π in a data format matching the `gpu` parameter.

Input value:

- `gpu`: 0: use CPU, 1: use GPU, 2: use mp library

Return value:

- `one`: the value 1. Data type: `mp` if `gpu == 2`, else `integer`
- `two`: the value 2. Data type: `mp` if `gpu == 2`, else `integer`
- `four`: the value 4. Data type: `mp` if `gpu == 2`, else `integer`
- `eight`: the value 8. Data type: `mp` if `gpu == 2`, else `integer`
- `twopi`: the value 2π. Data type: `mp` if `gpu == 2`, else `double`

Source code:

```
function [one, two, four, eight, twopi]  = precision_constants(gpu)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% creates the numerical constants 1, 2, 4, 8 and 2*pi
% in the data format required for the GPU setting
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Input parameters:
% gpu  ......... 0: use CPU, 1: use GPU, 2: use mp library
%
% Outputs:
% one .... constant 1
% two .... constant 2
% four ... constant 4
% twopi .. constant 2*pi
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if gpu == 2
    % mp library use
    one = mp('1');
    two = mp('2');
    four = mp('4');
    eight = mp('8');
    twopi = mp('2*pi');
else
    % CPU or GPU use
    one = 1;
    two = 2;
    four = 4;
    eight = 8;
    twopi = 2*pi;
end
```

D.6. pval

The function `pval(gpu, x)` takes a scalar value encoded in the input string-parameter `x` and returns it as value parameter matching the `gpu`-setting.

Input value:

- `gpu`: 0: use CPU, 1: use GPU, 2: use mp library
- `x`: value encoded as string

Return value:

- `y`: numerical value. Data type: `mp` if `gpu == 2`, else `double` or `integer`

Source code:

```
function y = pval(gpu, x)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% converts the scalar input x (provided as string) into the data format
% required for the GPU setting
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Input parameters:
% gpu  ....  0: use CPU, 1: use GPU, 2: use mp library
% x .......  scalar value encoded as string
%
% Output:
% y .......  numerical value.  If gpu==2: data type mp, else double
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if gpu == 2
    y = mp(x);
else
    y = eval(x);
end
```

E. Simulation main programs

E.1. CPA_sim_001_reflection_matrix

The program on the following page is used to calculate the results in section 3.3.4. It first calculates in lines 32-39 a critical wavenumber k_c based on equation (3.19), so that the corresponding critical wavelength λ_c is as close as possible to the given reference-wavelength λ_0. Then it uses the function described in appendix C.21 to create a single-round-trip transmission matrix `T` for a 4f-cavity without attenuation, corresponding to $\underline{\underline{\mathrm{T}}}_c$ in equation (3.41) (lines 55-57). Using equation (3.41), the transmission matrix `T_atten`, corresponding to $\underline{\underline{\tilde{\mathrm{T}}}}_c$, is calculated (line 69). Eventually, in lines 71-74, the cavity's total reflection matrix `R` is calculated based on equations (3.11) and (3.39). In the remaining lines of code, the unitarity of the matrix `T` is checked by calculating $\underline{\underline{\mathrm{T}}}\underline{\underline{\mathrm{T}}}^\dagger$ in line 60, and displaying the largest off-diagnonal absolute value of $\underline{\underline{\mathrm{T}}}\underline{\underline{\mathrm{T}}}^\dagger$ (line 61-62). Also, in lines 64-66, the top-left diagonal entry of `T` is compared with the expected value based on equation (3.42). Finally, `R` (with element-wisely squared absolute values) and `T` are visualized graphically.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Calculating Refleciton Matrix from
% 4f-cavity round-trip transmission matrix
% File: CPA_sim_001_reflection_matrix.m
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear all
close all
gpu = 1; % 0: CPU, 1: GPU, 2: mp library
if gpu == 1
    for ii = 1:gpuDeviceCount
        gpuDevice(ii); % initialize GPUs
    end
elseif gpu == 2
    mp.Digits(34); % Define numerical precision for mp library
end
% create constants 1, 2, 4, 8, 2*pi in required data format
[one, two, four, eight, twopi] = precision_constants(gpu);

% Basic physical simulation parameters
lambda_0 = pval(gpu, '800e-9'); % base wavelength [m]
f = pval(gpu, '75e-3');         % focal length of lenses [m]
pupil = false;                  % pupil yes/no
NA = pval(gpu, '0.05');         % numerical aperture of pupil
lens_type = 2;                  % 1: spherical 2:aspherical, perfect lens
r0 = pval(gpu, 'sqrt(0.8)');    % reflection coefficient left mirror

% technical simulation parameters
TF = 0;                         % 0: Rayleigh Sommerfeld, 1: Fresnel
L_des = pval(gpu, '2.5e-3');    % desired side length of smaller grid
factor = 2;             % factor by which embedding grid should be larger

% longitudinal mode number best matching lambda_0
l_mode = round((atan(sqrt(one-r0^two)/r0))/twopi+eight*f/lambda_0);
% critical wavenumber best matching lambda_0
k_c = twopi*l_mode/(eight*f) - (atan(sqrt(1-r0^two)/r0))/(eight*f);
% critical wavelength best matching lambda_0
lambda_c = twopi/k_c;
% We use the critical wavelength for simulation
lambda = lambda_c;

% create optimal grid
[N_small, L_small, N_large, L_large, alpha_max, n_max] = ...
opt_grid_params(L_des, factor, f, f, true, lambda);
% display grid parameters
disp(['N1=', num2str(N_small), ' N2=', num2str(N_large), ' n_max=', num2str(n_max)])
% create xy-axes coordinates for "small" and "large" grid
ax_small = create_ax(gpu, N_small, L_small);
ax_large = create_ax(gpu, N_large, L_large);
% Create vector with xy-modes up to n=32/2=16
modes = sorted_mode_numbers(gpu, 32);
no_of_modes = size(modes,1);
% create lens phase mask
[lens_mask, lens_pupil] = lens(gpu, ax_large, lambda, pupil, NA, f, lens_type);

% generate transmission matrix for one-round-trip (no attenuation)
T = transmision_matrix_round_trip_no_atten(gpu, ax_small, ax_large, ...
    TF, modes, lens_mask, f, lambda);

% check unitarity of transmission matrix
TT = T*T';
maxval = max(abs(abs(TT)-eye(size(T,1))),[],'all');
disp(['max T*T'' off-diagonal value:    ', num2str(maxval)]);

% compare diagonal value of transmission matrix with expected value
disp(['estimated diagonal value of T: ', num2str(-r0+i*sqrt(1-r0^2))]);
disp(['actual T(1,1) value:           ', num2str(T(1,1))]);

% Calculate transmission matrix *with* attenuation
T_atten = r0*T;

% Calculate reflection matrix
I = create_eye(gpu, no_of_modes);
r1 = -r0^two - i * r0 * sqrt(one - r0^two); % left mirror refl. coeff.
R = I * r1 + (one + r1)^two * T_atten / (I - r1 * T_atten);

% Display Reflection Matrix
figure(1);
imagesc(abs(R).^2)
axis square
title('Reflection Matrix Squared')

% Display Transmission Matrix
figure(2);
imagesc(abs(T))
axis square
title('Transmission Matrix (no attenuation)')
```

E.2. CPA_sim_002_r_curve

The following source-code is used to calculate the results in section 4.1.1. It allows to calculate the maximum, average, and minimum eigenvalue of a 4f cavity's reflection matrix for various wavelengths around a critical wavelength. The attenuation can be set to exactly critical attenuation, or alternatively to arbitrary overcritical or undercritical values. The results are written into a file in .xlsx (MS Excel) format.

In lines 25-32 basic physical simulation parameters are defined; amongst them the base wavelength lambda_0 in line 26, the focal length in line 27, and the type of lens to be used in line 30. The parameter r0 in line 31 is the absolute value of the left mirror's reflection coefficient. Finally, rho in line 32 is the factor by which the critical attenuation gets multiplied. Consequently, rho=1 results in exactly critical attenuation, whereas a value smaller (larger) than 1 results in undercritical (overcritical) attenuation.

```
 1 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 2 % Calculating max(EV(R)), avg(EV(R)), min(EV(R))
 3 % of a attenuated 4f-cavity
 4 % over a range of wavelengths
 5 % File: CPA_sim_002_r_curve.m
 6 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 7 
 8 clear all
 9 close all
10 folder = '002\\'; % folder for saving results
11 gpu = 1; % 0: CPU, 1: GPU, 2: mp library
12 if gpu == 1
13     % initialize GPUs
14     for ii = 1:gpuDeviceCount
15         gpuDevice(ii);
16     end
17 elseif gpu == 2
18     % Define numerical precision for mp library
19     mp.Digits(34);
20 end
21 % create constants 1, 2, 4, 8, 2*pi in required data format
22 [one, two, four, eight, twopi] = precision_constants(gpu);
23 pm = pval(gpu, '1e-12'); % one pico-meter
24 
25 % Basic physical simulation parameters
26 lambda_0 = pval(gpu, '800e-9'); % base wavelength [m]
27 f = pval(gpu, '75e-3');         % focal length of lenses [m]
28 pupil = false;                  % pupil yes/no
29 NA = pval(gpu, '0.05');         % numerical aperture of pupil
30 lens_type = 1;                  % 1: spherical 2:aspherical, perfect lens
31 r0 = pval(gpu, 'sqrt(0.8)');    % reflection coefficient left mirror
32 rho = pval(gpu, '1');      % how much crit. attenuation?
33 
34 % technical simulation parameters
35 TF = 0;                         % 0: Rayleigh Sommerfeld, 1: Fresnel
36 L_des = pval(gpu, '2.5e-3');    % desired side length of smaller grid
37 factor = 2;           % factor by which embedding grid should be larger
38 
39 % longitudinal mode number best matching lambda_0
40 l_mode = round((atan(sqrt(one-r0^two)/r0))/twopi+eight*f/lambda_0);
41 % critical wavenumber best matching lambda_0
42 k_c = twopi*l_mode/(eight*f) - (atan(sqrt(1-r0^two)/r0))/(eight*f);
43 % critical wavelength best matching lambda_0
44 lambda_c = twopi/k_c;
45 
46 % create optimal grid
47 [N_small, L_small, N_large, L_large, alpha_max, n_max] = ...
48 opt_grid_params(L_des, factor, f, f, true, lambda_0);
49 % display grid parameters
50 disp(['N1=', num2str(N_small), ' N2=', num2str(N_large), ' n_max=', num2str(n_max)])
51 % create xy-axes coordinates for "small" and "large" grid
52 ax_small = create_ax(gpu, N_small, L_small);
53 ax_large = create_ax(gpu, N_large, L_large);
```

```
 54 % Create vector with xy-modes up to n=32/2=16
 55 modes = sorted_mode_numbers(gpu, 32);
 56 no_of_modes = size(modes,1);
 57 % create lens phase mask
 58 [lens_mask, lens_pupil] = lens(gpu, ax_large, lambda_0, pupil, NA, f, lens_type);
 59 % define area of interest and iterate through it
 60 points = pval(gpu, '300');% number of points to be calculated
 61 period = twopi/(eight*f); % period between resonance points
 62 aoi_width = period/four;  % area-of-interest width: a quater of a period
 63 dk = aoi_width/points;    % delta k
 64 s=0;                      % counting-up-index
 65 for idx = round(points/two):-1:-round(points/two)
 66     s = s + 1;
 67     k = k_c +idx * dk;
 68     lambda = twopi/k;
 69 
 70     % generate transmission matrix for one-round-trip (no attenuation)
 71     T = transmision_matrix_round_trip_no_atten(gpu, ax_small, ax_large, ...
 72         TF, modes, lens_mask, f, lambda);
 73 
 74     % Calculate transmission matrix *with* attenuation
 75     T_atten = r0*rho*T;
 76 
 77     % Calculate reflection matrix
 78     I = create_eye(gpu, no_of_modes);
 79     r1 = -r0^two - i * r0 * sqrt(one - r0^two); % left mirror refl. coeff.
 80     R = I * r1 + (one + r1)^two * T_atten / (I - r1 * T_atten);
 81     EV = eig(R); % Eigenvalues of reflection matrix
 82     xls(s,1) = idx;
 83     xls(s,2) = lambda / pm; % wavelength in pm
 84     xls(s,3) = (lambda-lambda_c) / pm; % delta lambda in pm
 85     xls(s,4) = max(abs(EV)); % largest eigenvalue
 86     xls(s,5) = mean(abs(EV)); % mean eigenvalue
 87     xls(s,6) = min(abs(EV)); % smallest eigenvalue
 88     xls(s,7) = xls(s,4)^2; % largest eigenvalue squared
 89     xls(s,8) = xls(s,5)^2; % mean eigenvalue squared
 90     xls(s,9) = xls(s,6)^2; % smallest eigenvalue squared
 91 
 92     disp(['idx: ', num2str(idx), ...
 93         ' max EV: ', num2str(max(abs(EV))), ...
 94         ' mean EV: ', num2str(mean(abs(EV))), ...
 95         ' min EV: ', num2str(min(abs(EV)))])
 96 end
 97 % save results
 98 filename = strcat(folder, 'EV_atten', num2str(rho,4));
 99 if lens_type == 1
100     filename = strcat(filename, '_spherical');
101 else
102     filename = strcat(filename, '_aspherical');
103 end
104 filename = strcat(filename,'_f_', num2str(f*1000,4));
105 if TF == 0
106     filename = strcat(filename,'_RS');
107 else
108     filename = strcat(filename,'_FR');
109 end
110 filename = strcat(filename,'.xlsx');
111 writematrix(xls, filename);
```

Among the technical simulation parameters in lines 34-37 there is the parameter `TF` which defines the propagator method to be used (Rayleigh-Sommerfeld propagator as described in section 2.1.2, or Fresnel propagator, as described in section 2.1.3).

In lines 39-44, the critical wavenumber and critical wavelength best matching the reference wavelength `lambda_0` are calculated. Lines 46-53 determine the optimal sizes for the sampling grid and the guarding grid (see sections 2.6 and 2.7). Line 55 defines the highest transverse xy mode to be included in the simulation.

The core part of the program starts in line 60. The number of points to be calculated is defined in line 61, and also the area of interest (in above program it is a quarter of a period around the resonance point, see line 63). The loop starting in line 66 iterates through various wavenumbers in the area of interest.

Using method `transmission_matix_round_trip` (see appendix C.21), the transmission-matrix `T` of an unattenuated 4d-cavity with the given physical parameters and the wavenumber of the current loop-iteration is generated.

In line 76, the attenuation-factor is multiplied to the transmission matrix of the unattenuated cavity. Based on that, the total reflection matrix `R` of the cavity is calculated in lines 78-81 using equation (3.39).

Finally the eigenvalues of `R` are calculated in lines 82-96. The smallest, average, and largest eigenvalues are stored in the `xls`-array. When the loop has ended and all data-points are calculated, a file-name describing the used parameters is generated in lines 99-111, and the results are finally written into a file in line 112.

E.3. `CPA_sim_003_mode_decomposition`

The program on the following page is used to calculate the results in section 4.2. It generates the reflection-matrix of an attenuated 4f-cavity, then calculates the corresponding eigenvectors and eigenvalues, and finally writes the eigenvalues in ascending order into an `.xlsx`-file. Optionally, the corresponding eigenmodes (encoded in the eigenvectors) can also be saved as images, and/or it creates a video where the eigenmodes in ascending order are visualized in an animation.

In line 8 it can be chosen whether or not a video should be created. Line 9 allows to choose whether or not the eigenmodes should be saved as images. The output folder is determined in line 10. Basic physical simulation parameters are defined in lines 22-29; among others the type of lenses to be used (line 27). Lines 31-34 are used for defining technical simulation parameters, including the propagation simulation to be used (Rayleigh-Sommerfeld or Fresnel propagation).

In lines 36-43, the program then calculates a critical wavenumber k_c based on equation (3.19), so that the corresponding critical wavelength λ_c is as close as possible to the given reference-wavelength λ_0. Then it uses the function described in appendix C.21 to create a single-round-trip transmission matrix `T` for a 4f-cavity without attenuation, corresponding to $\underline{\underline{\mathrm{T}_c}}$ in equation (3.41) (lines 59-61). Using equation (3.41), the transmission matrix `T_atten`, corresponding to $\underline{\underline{\tilde{\mathrm{T}}_c}}$, is calculated (line 64). Eventually, in lines 66-69, the cavity's total reflection matrix `R` is calculated based on equations (3.11) and (3.39).

In lines 71-76, the eigensystem of the reflection matrix is determined, a meaningful file name is generated (lines 78-91), and the squared absolute eigenvalues are stored in ascending order into a file together with the ascending mode numbers (lines 92-95).

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Calculates all reflection-eigenmodes of a 4f-cavity CPA and saves
% them into separate image files and also in a video file
% File: CPA_sim_003_mode_decomposition.m
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear all
close all
create_video  = true;  % create video of eigenmodes?
create_images = true;  % create images of eigenmodes?
folder = '003\\';      % folder for saving results
gpu = 1; % 0: CPU, 1: GPU, 2: mp library
if gpu == 1
    for ii = 1:gpuDeviceCount
        gpuDevice(ii); % initialize GPUs
    end
elseif gpu == 2
    mp.Digits(34); % define numerical precision for mp library
end
% create constants 1, 2, 4, 8, 2*pi in required data format
[one, two, four, eight, twopi] = precision_constants(gpu);

% Basic physical simulation parameters
lambda_0 = pval(gpu, '800e-9'); % base wavelength [m]
f = pval(gpu, '99e-3');         % focal length of lenses [m]
pupil = false;                  % pupil yes/no
NA = pval(gpu, '0.05');         % numerical aperture of pupil
lens_type = 2;                  % 1: spherical 2:aspherical, perfect lens
r0 = pval(gpu, 'sqrt(0.8)');    % reflection coefficient left mirror
rho = pval(gpu, '1');           % how much crit. attenuation?

% technical simulation parameters
TF = 0;                         % 0: Rayleigh Sommerfeld, 1: Fresnel
L_des = pval(gpu, '2.5e-3');    % desired side length of smaller grid
factor = 2;         % factor by which embedding grid should be larger

% longitudinal mode number best matching lambda_0
l_mode = round((atan(sqrt(one-r0^two)/r0))/twopi+eight*f/lambda_0);
% critical wavenumber best matching lambda_0
k_c = twopi*l_mode/(eight*f) - (atan(sqrt(1-r0^two)/r0))/(eight*f);
% critical wavelength best matching lambda_0
lambda_c = twopi/k_c;
% We use the critical wavelength for simulation
lambda = lambda_c;

% create optimal grid
[N_small, L_small, N_large, L_large, alpha_max, n_max] = ...
opt_grid_params(L_des, factor, f, f, true, lambda);
% display grid parameters
disp(['N1=', num2str(N_small), ' N2=', num2str(N_large), ' n_max=', num2str(n_max)])
% create xy-axes coordinates for "small" and "large" grid
ax_small = create_ax(gpu, N_small, L_small);
ax_large = create_ax(gpu, N_large, L_large);
% Create vector with xy-modes up to n=32/2=16
modes = sorted_mode_numbers(gpu, 32);
no_of_modes = size(modes,1);
% create lens phase mask
[lens_mask, lens_pupil] = lens(gpu, ax_large, lambda, pupil, NA, f, lens_type);

% generate transmission matrix for one-round-trip (no attenuation)
T = transmision_matrix_round_trip_no_atten(gpu, ax_small, ax_large, ...
    TF, modes, lens_mask, f, lambda);

% Calculate transmission matrix *with* attenuation
T_atten = r0 * rho * T;

% Calculate reflection matrix
I = create_eye(gpu, no_of_modes);
r1 = -r0^two - i * r0 * sqrt(one - r0^two); % left mirror refl. coeff.
R = I * r1 + (one + r1)^two * T_atten / (I - r1 * T_atten);

% Eigenvalues and Eigenvectors of reflection matrix
[V, D] = eig(R); % Eigenvectors and diagonal eigenvalue matrix
EV = diag(D); % get Eigenvalues from diagonal D matrix
EVS = abs(EV).^2; % Eigenvalues squared
[EVS_sorted, sortEVS_idx] = sort(EVS); % sort sqared eigenvalues
V_sorted = V(:,sortEVS_idx); % eigenvectors sorted by eigenvalues
```

```
 78 % generate filename
 79 filename = strcat(folder, 'modes_atten', num2str(rho,4));
 80 if lens_type == 1
 81     filename = strcat(filename, '_spherical');
 82 else
 83     filename = strcat(filename, '_aspherical');
 84 end
 85 filename = strcat(filename,'_f_', num2str(f*1000,4));
 86 if TF == 0
 87     filename = strcat(filename,'_RS');
 88 else
 89     filename = strcat(filename,'_FR');
 90 end
 91 filename = strcat(filename,'.xlsx');
 92 % generate result matrix
 93 xls = [1:size(EVS,1)]'; % 1st column: ascending number of mode
 94 xls(:,2) = EVS_sorted; % second column: squared eigenvalues
 95 writematrix(xls, filename);
 96 
 97 if create_video || create_images
 98     if create_video
 99         % create video-file
100         vidfilename = strcat(folder,'modes','.mp4');
101         vidfile = VideoWriter(vidfilename,'MPEG-4');
102         open(vidfile);
103     end
104     % iterate through all eigenmodes of the reflection matrix
105     for idx = 1:size(modes);
106         EigenMode = fft2_vec_to_arr(gpu, V_sorted(:,idx), ax_small, modes, false);
107         plot_fields(gpu, 1, ax_small, EigenMode, "input", 1)
108         % color-coding symmetric around zero
109         maxval = max(real(EigenMode),[],'all');
110         minval = min(real(EigenMode),[],'all');
111         extval = gather(max([abs(minval), abs(maxval)]));
112         colorbar("off");
113         caxis([-extval, extval]);
114         title(strcat("Reflectance: ",num2str(EVS_sorted(idx))));
115         if create_video
116             % write image as to frames to video file
117             writeVideo(vidfile,getframe(1));
118             writeVideo(vidfile,getframe(1));
119         end
120         if create_images
121             imgfilename = sprintf('%05d',idx);
122             imgfilename = strcat(folder,imgfilename,'.png');
123             % save image in separate png-image-file
124             saveas(1, imgfilename);
125         end
126     end
127     if create_video
128         % close video file
129         close(vidfile);
130     end
131 end
```

The rest of the program, starting in line 97, is only executed if a video is to be created, or the eigenmodes are to be saved as images. If required, a video-file is created in lines 98-103. Then, starting from line 105, a loop iterates through all eigenvalues (in ascending order). For each eigenvalue, the corresponding eigenvector is converted into an array in line 106 using the function `fft2_vec_to_arr` (see appendix C.9). In lines 107-114 the real value part of the currently processed eigenmode is plotted, and the color-coding is adjusted so that the zero-value has always the same color for all iterations.

In lines 115-119, the created image is added to the video file twice, so that two frames in the video always display the same image and the resulting video has an acceptable speed. In lines 120-125, the created image is saved as `.png` file with a filename encoding the mode number (with leading zeros). Finally, in lines 127-130, the video file is closed.

E.4. CPA_sim_004_vary_atten

The program listed below first generates the reflection-matrix $\underline{\underline{R}}_{crit}$ of a critically attenuated 4f-cavity, and calculates the corresponding eigenvectors and eigenvalues (lines 1-73). It does so by using the same program logic as the program in appendix E.3. Then it runs through a loop (lines 81-96) where in each iteration a modified reflection-matrix $\underline{\underline{R}}(a)$ with a slightly overcritical or undercritical attenuations a is calculated.

For each of the original eigenvectors $\vec{v}_m$ (with m being the index number of the eigenmode), the reflectance $R(a,m) = |\underline{\underline{R}}(a)\ \vec{v}_m|^2$ is calculated in line 88. If the reflectance $R(a,m)$ for a specific mode m is lower than the best reflectance so far, then the program remembers this value as the new best-so-far reflectance together with the factor by which the attenuation a is larger or smaller than the critical attenuation for this mode (lines 89-94).

Finally, the results are collected into an output-array and stored into a file (lines 98-105).

```
 1 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 2 % Calculates all reflection-eigenmodes of a 4f-cavity CPA
 3 % and varies the attenuation to find the optimal attenuation for each mode
 4 % File: CPA_sim_004_vary_atten.m
 5 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 6 clear all
 7 close all
 8 folder = '004\\';       % folder for saving results
 9 gpu = 1; % 0: CPU, 1: GPU, 2: mp library
10 if gpu == 1
11     for ii = 1:gpuDeviceCount
12         gpuDevice(ii); % initialize GPUs
13     end
14 elseif gpu == 2
15     mp.Digits(34); % define numerical precision for mp library
16 end
17 % create constants 1, 2, 4, 8, 2*pi in required data format
18 [one, two, four, eight, twopi] = precision_constants(gpu);
19
20 % Basic physical simulation parameters
21 lambda_0 = pval(gpu, '800e-9'); % base wavelength [m]
22 f = pval(gpu, '50e-3');         % focal length of lenses [m]
23 pupil = false;                  % pupil yes/no
24 NA = pval(gpu, '0.05');         % numerical aperture of pupil
25 lens_type = 2;                  % 1: spherical 2:aspherical, perfect lens
26 r0 = pval(gpu, 'sqrt(0.8)');    % reflection coefficient left mirror
27
28 % technical simulation parameters
29 TF = 0;                         % 0: Rayleigh Sommerfeld, 1: Fresnel
30 L_des = pval(gpu, '2.5e-3');    % desired side length of smaller grid
31 factor = 2;          % factor by which embedding grid should be larger
32
33 % longitudinal mode number best matching lambda_0
34 l_mode = round((atan(sqrt(one-r0^two)/r0))/twopi+eight*f/lambda_0);
35 % critical wavenumber best matching lambda_0
36 k_c = twopi*l_mode/(eight*f) - (atan(sqrt(1-r0^two)/r0))/(eight*f);
37 % critical wavelength best matching lambda_0
38 lambda_c = twopi/k_c;
39 % We use the critical wavelength for simulation
40 lambda = lambda_c;
41
42 % create optimal grid
43 [N_small, L_small, N_large, L_large, alpha_max, n_max] = ...
44 opt_grid_params(L_des, factor, f, f, true, lambda);
45 % display grid parameters
46 disp(['N1=', num2str(N_small,5), ' N2=', num2str(N_large,5), ' n_max=', num2str(n_max,5)])
47 % create xy-axes coordinates for "small" and "large" grid
48 ax_small = create_ax(gpu, N_small, L_small);
49 ax_large = create_ax(gpu, N_large, L_large);
```

```
% Create vector with xy−modes up to n=32/2=16
modes = sorted_mode_numbers(gpu, 32);
no_of_modes = size(modes,1);
% create lens phase mask
[lens_mask, lens_pupil] = lens(gpu, ax_large, lambda, pupil, NA, f, lens_type);

% generate transmission matrix for one−round−trip (no attenuation)
T = transmision_matrix_round_trip_no_atten(gpu, ax_small, ax_large, ...
    TF, modes, lens_mask, f, lambda);

% Calculate transmission matrix with critical attenuation
T_atten = r0 * T;

% Calculate reflection matrix
I = create_eye(gpu, no_of_modes);
r1 = −r0^two − i * r0 * sqrt(one − r0^two); % left mirror refl. coeff.
R = I * r1 + (one + r1)^two * T_atten / (I − r1 * T_atten);

% Eigenvalues and Eigenvectors of reflection matrix
[V, D] = eig(R); % Eigenvectors and diagonal eigenvalue matrix
EV = diag(D); % get Eigenvalues from diagonal D matrix
EVS = abs(EV).^2; % Eigenvalues squared
[EVS_sorted, sortEVS_idx] = sort(EVS); % sort sqared eigenvalues
V_sorted = V(:,sortEVS_idx); % eigenvectors sorted by eigenvalues

% Results
xls1(1:no_of_modes)=0; % mode number
xls2(1:no_of_modes)=pval(gpu,'0'); % lowest reflectance
xls3(1:no_of_modes)=pval(gpu,'0'); % rho for lowest reflection


% iterate from 0.99 to 1.01 critical attenuation
for rho = pval(gpu, '0.99'):pval(gpu, '0.0001'):pval(gpu, '1.01')
    T_atten = r0 * rho * T;
    R = I * r1 + (one + r1)^two * T_atten / (I − r1 * T_atten);
    % test all modes with current damping and k
    disp(['Testing ', 'rho ',num2str(rho,5)])
    parfor m = 1:no_of_modes;
        refl = norm(R*V_sorted(:,m))^2; % calculate reflectance
        if xls1(m)==0 || refl < xls2(m)
            % first data point, or lowest reflectance so far
            xls1(m) = m;            % mode number
            xls2(m) = refl;         % lowest reflectance so far
            xls3(m) = rho;          % rho for lowest reflectance
        end
    end
end

% collect results
xls(:,1)=xls1';        % mode number
xls(:,2)=EVS_sorted'; % original reflectance
xls(:,3)=xls2';        % lowest reflectance
xls(:,4)=xls3';        % rho for lowest reflectance

filename = strcat(folder,'opt_atten.xlsx');
writematrix(xls, filename);
```

E.5. CPA_sim_005_vary_f

The program listed below is used to calculate the results in section 4.3.3. It first generates the reflection-matrix $\underline{\underline{R}}_{crit}$ of a critically attenuated 4f-cavity, and calculates the corresponding eigenvectors and eigenvalues (lines 1-74). It does so by using the same program logic as the program in appendix E.3. Then it runs through a loop (lines 83-106) where in each iteration a modified 4f-cavity-reflection-matrix $\underline{\underline{R}}(f)$ for a slightly modified focal length $f = f_0 + c$ is generated (lines 85-91).

For each of the original eigenvectors $\vec{v}_m$ (with m being the index number of the eigenmode), the reflectance $R(c,m) = |\underline{\underline{R}}(f_0 + c)\ \vec{v}_m|^2$ is calculated in line 95. If the reflectance $R(c,m)$ for a specific mode m is lower than the best reflectance so far, then the program remembers this value as the new best-so-far reflectance together with the additive correction factor c (lines 99-104).

Finally, the results are collected into an output-array and stored into a file (lines 108-115).

```
 1 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 2 % Calculates all reflection-eigenmodes of a 4f-cavity CPA
 3 % and varies the attenuation to find best attenuation for each mode
 4 % File: CPA_sim_005_vary_f.m
 5 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 6 clear all
 7 close all
 8 folder = '005\\';                 % folder for saving results
 9 gpu = 1;                          % 0: CPU, 1: GPU, 2: mp library
10 pm = pval(gpu, '1e-12');          % one pico-meter
11 if gpu == 1
12     for ii = 1:gpuDeviceCount
13         gpuDevice(ii); % initialize GPUs
14     end
15 elseif gpu == 2
16     mp.Digits(34); % define numerical precision for mp library
17 end
18 % create constants 1, 2, 4, 8, 2*pi in required data format
19 [one, two, four, eight, twopi] = precision_constants(gpu);
20
21 % Basic physical simulation parameters
22 lambda_0 = pval(gpu, '800e-9'); % base wavelength [m]
23 f = pval(gpu, '50e-3');          % focal length of lenses [m]
24 pupil = false;                   % pupil yes/no
25 NA = pval(gpu, '0.05');          % numerical aperture of pupil
26 lens_type = 2;                   % 1: spherical 2:aspherical, perfect lens
27 r0 = pval(gpu, 'sqrt(0.8)');     % reflection coefficient left mirror
28
29 % technical simulation parameters
30 TF = 0;                          % 0: Rayleigh Sommerfeld, 1: Fresnel
31 L_des = pval(gpu, '2.5e-3');     % desired side length of smaller grid
32 factor = 2;         % factor by which embedding grid should be larger
33
34 % longitudinal mode number best matching lambda_0
35 l_mode = round((atan(sqrt(one-r0^two)/r0))/twopi+eight*f/lambda_0);
36 % critical wavenumber best matching lambda_0
37 k_c = twopi*l_mode/(eight*f) - (atan(sqrt(1-r0^two)/r0))/(eight*f);
38 % critical wavelength best matching lambda_0
39 lambda_c = twopi/k_c;
40 % We use the critical wavelength for simulation
41 lambda = lambda_c;
42
43 % create optimal grid
44 [N_small, L_small, N_large, L_large, alpha_max, n_max] = ...
45 opt_grid_params(L_des, factor, f, f, true, lambda);
46 % display grid parameters
47 disp(['N1=', num2str(N_small), ' N2=', num2str(N_large), ' n_max=', num2str(n_max)])
48 % create xy-axes coordinates for "small" and "large" grid
49 ax_small = create_ax(gpu, N_small, L_small);
50 ax_large = create_ax(gpu, N_large, L_large);
```

```
% Create vector with xy-modes up to n=32/2=16
modes = sorted_mode_numbers(gpu, 32);
no_of_modes = size(modes,1);
% create lens phase mask
[lens_mask, lens_pupil] = lens(gpu, ax_large, lambda, pupil, NA, f, lens_type);

% generate transmission matrix for one-round-trip (no attenuation)
T = transmision_matrix_round_trip_no_atten(gpu, ax_small, ax_large, ...
    TF, modes, lens_mask, f, lambda);

% Calculate transmission matrix with critical attenuation
T_atten = r0 * T;

% Calculate reflection matrix
I = create_eye(gpu, no_of_modes);
r1 = -r0^two - i * r0 * sqrt(one - r0^two); % left mirror refl. coeff.
R = I * r1 + (one + r1)^two * T_atten / (I - r1 * T_atten);

% Eigenvalues and Eigenvectors of reflection matrix
[V, D] = eig(R); % Eigenvectors and diagonal eigenvalue matrix
EV = diag(D); % get Eigenvalues from diagonal D matrix
EVS = abs(EV).^2; % Eigenvalues squared
[EVS_sorted, sortEVS_idx] = sort(EVS); % sort sqared eigenvalues
V_sorted = V(:,sortEVS_idx); % eigenvectors sorted by eigenvalues

% Results
xls1(1:no_of_modes)=0; % mode number
xls2(1:no_of_modes)=0; % original reflectance
xls3(1:no_of_modes)=pval(gpu,'0'); % lowest reflectance
xls4(1:no_of_modes)=pval(gpu,'0'); % length_factor for lowest reflection


% iterate through various values of the correction factor c
for c = -1000:10              % additive correction in pm
    % generate transmission matrix for one-round-trip (no attenuation)
    T = transmision_matrix_round_trip_no_atten(gpu, ax_small, ax_large, ...
    TF, modes, lens_mask, f + c*pm, lambda);
    % calculate attenuated transmission matrix
    T_atten = r0 * T;
    % calulate refleciton matrix
    R = I * r1 + (one + r1)^two * T_atten / (I - r1 * T_atten);
    % test all modes with current damping and k
    disp(['testing ', 'c ',num2str(c,5)])
    parfor m = 1:no_of_modes;
        refl = norm(R*V_sorted(:,m))^2; % calculate reflectance
        if c == 0
            xls2(m) = refl;           % original reflectance
        end
        if xls1(m)==0 || refl < xls3(m)
            % first data point, or lowest reflectance so far
            xls1(m) = m;              % mode number
            xls3(m) = refl;           % reflectance
            xls4(m) = c;              % c [pm] for lowest reflectance
        end
    end
end

% collect results
xls(:,1)=xls1';         % mode number
xls(:,2)=xls2';         % original reflectance
xls(:,3)=xls3';         % improved reflectance
xls(:,4)=xls4';         % f_corr [pm]

filename = strcat(folder,'opt_f.xlsx');
writematrix(xls, filename);
```

E.6. CPA_sim_006_vary_length

The program listed below is used to calculate the results in section 4.3.4. It first generates the reflection-matrix $\underline{\underline{R}}_{crit}$ of a critically attenuated 4f-cavity, and calculates the corresponding eigenvectors and eigenvalues (lines 1-74). It does so by using the same program logic as the program in appendix E.3. Then it runs through a loop (lines 83-106) where in each iteration a modified 4f-cavity-reflection-matrix $\underline{\underline{R}}(f_0, d)$ with the same focal length f_0, but a deviating distance d between the second lens and the perfect back-mirror is generated (lines 85-91).

For each of the original eigenvectors $\vec{v}_m$ (with m being the index number of the eigenmode), the reflectance $R(d{,}m) = |\underline{\underline{R}}(f_0, d)\ \vec{v}_m|^2$ is calculated in line 95. If the reflectance $R(d{,}m)$ for a specific mode m is lower than the best reflectance so far, then the program remembers this value as the new best-so-far reflectance together with the additive correction factor d (lines 99-104).

Finally, the results are collected into an output-array and stored into a file (lines 108-115).

```
 1 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 2 % Calculates all reflection-eigenmodes of a 4f-cavity CPA
 3 % and varies the attenuation to fond best attenuation for each mode
 4 % File: CPA_sim_006_vary_length.m
 5 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 6 clear all
 7 close all
 8 folder = '006\\';                  % folder for saving results
 9 gpu = 1;                           % 0: CPU, 1: GPU, 2: mp library
10 pm = pval(gpu, '1e-12');           % one pico-meter
11 if gpu == 1
12     for ii = 1:gpuDeviceCount
13         gpuDevice(ii); % initialize GPUs
14     end
15 elseif gpu == 2
16     mp.Digits(34); % define numerical precision for mp library
17 end
18 % create constants 1, 2, 4, 8, 2*pi in required data format
19 [one, two, four, eight, twopi] = precision_constants(gpu);
20 
21 % Basic physical simulation parameters
22 lambda_0 = pval(gpu, '800e-9'); % base wavelength [m]
23 f = pval(gpu, '50e-3');            % focal length of lenses [m]
24 pupil = false;                     % pupil yes/no
25 NA = pval(gpu, '0.05');            % numerical aperture of pupil
26 lens_type = 2;                     % 1: spherical 2:aspherical, perfect lens
27 r0 = pval(gpu, 'sqrt(0.8)');       % reflection coefficient left mirror
28 
29 % technical simulation parameters
30 TF = 0;                            % 0: Rayleigh Sommerfeld, 1: Fresnel
31 L_des = pval(gpu, '2.5e-3');       % desired side length of smaller grid
32 factor = 2;             % factor by which embedding grid should be larger
33 
34 % longitudinal mode number best matching lambda_0
35 l_mode = round((atan(sqrt(one-r0^two)/r0))/twopi+eight*f/lambda_0);
36 % critical wavenumber best matching lambda_0
37 k_c = twopi*l_mode/(eight*f) - (atan(sqrt(1-r0^two)/r0))/(eight*f);
38 % critical wavelength best matching lambda_0
39 lambda_c = twopi/k_c;
40 % We use the critical wavelength for simulation
41 lambda = lambda_c;
42 
43 % create optimal grid
44 [N_small, L_small, N_large, L_large, alpha_max, n_max] = ...
45 opt_grid_params(L_des, factor, f, f, true, lambda);
46 % display grid parameters
47 disp(['N1=', num2str(N_small,5), ' N2=', num2str(N_large,5), ' n_max=', num2str(n_max,5)])
```

```
% create xy-axes coordinates for "small" and "large" grid
ax_small = create_ax(gpu, N_small, L_small);
ax_large = create_ax(gpu, N_large, L_large);
% Create vector with xy-modes up to n=32/2=16
modes = sorted_mode_numbers(gpu, 32);
no_of_modes = size(modes,1);
% create lens phase mask
[lens_mask, lens_pupil] = lens(gpu, ax_large, lambda, pupil, NA, f, lens_type);

% generate transmission matrix for one-round-trip (no attenuation)
T = transmision_matrix_round_trip_no_atten(gpu, ax_small, ax_large, ...
    TF, modes, lens_mask, f, lambda);

% Calculate transmission matrix with critical attenuation
T_atten = r0 * T;

% Calculate reflection matrix
I = create_eye(gpu, no_of_modes);
r1 = -r0^two - i * r0 * sqrt(one - r0^two); % left mirror refl. coeff.
R = I * r1 + (one + r1)^two * T_atten / (I - r1 * T_atten);

% Eigenvalues and Eigenvectors of reflection matrix
[V, D] = eig(R); % Eigenvectors and diagonal eigenvalue matrix
EV = diag(D); % get Eigenvalues from diagonal D matrix
EVS = abs(EV).^2; % Eigenvalues squared
[EVS_sorted, sortEVS_idx] = sort(EVS); % sort sqared eigenvalues
V_sorted = V(:,sortEVS_idx); % eigenvectors sorted by eigenvalues

% Results
xls1(1:no_of_modes)=0; % mode number
xls2(1:no_of_modes)=0; % original reflectance
xls3(1:no_of_modes)=pval(gpu,'0'); % lowest reflectance
xls4(1:no_of_modes)=pval(gpu,'0'); % length_factor for lowest reflection


% iterate through various values for d in pm
for d = -4000:10
    % generate transmission matrix for one-round-trip (no attenuation)
    T = transmision_matrix_round_trip_no_atten2(gpu, ax_small, ax_large, ...
    TF, modes, lens_mask, f, f, f + d*pm, lambda);
    % calculate attenuated transmission matrix
    T_atten = r0 * T;
    % calulate reflection matrix
    R = I * r1 + (one + r1)^two * T_atten / (I - r1 * T_atten);
    % test all modes with current damping and k
    disp(['Testing ', 'd ',num2str(d,5)])
    parfor m = 1:no_of_modes;
        refl = norm(R*V_sorted(:,m))^2; % calculate reflectance
        if d == 0
            xls2(m) = refl;          % original reflectance
        end
        if xls1(m)==0 || refl < xls3(m)
            % first data point, or lowest reflectance so far
            xls1(m) = m;             % mode number
            xls3(m) = refl;          % reflectance
            xls4(m) = d;             % d [pm] for lowest reflectance
        end
    end
end

% collect results
xls(:,1)=xls1';        % mode number
xls(:,2)=xls2';        % original reflectance
xls(:,3)=xls3';        % improved reflectance
xls(:,4)=xls4';        % d [pm]

filename = strcat(folder,'opt_length3.xlsx');
writematrix(xls, filename);
```

E.7. CPA_sim_007_r_curve_deviate

The program listed below is used to calculate the results in section 4.4 and is an enhanced version of the program CPA_sim_002_r_curve as documented in appendix E.2. Like CPA_sim_002_r_curve, it allows to calculate the maximum, average, and minimum eigenvalues of a 4f cavity's reflection-matrix for various wavelengths around a critical wavelength. However, this enhanced program version not only allows to vary the attenuation, but also allows

- to vary the positions of the two lenses and the back-mirror, so that the distances may deviate from the optimal f-2f-f distances, and
- to run multiple simulations in a row, each with a different set of parameters.

The result of each simulation is written into a separate file in .xlsx (MS Excel) format.

In lines 25-32, basic physical simulation parameters common for all simulation rounds are defined; amongst them the base wavelength lambda_0 in line 26, the focal length in line 27, and the type of lens to be used in line 30. The parameter r0 in line 31 is the absolute value of the left mirror's reflection coefficient.

Among the technical simulation parameters in lines 34-37 there is the parameter TF which defines the propagator method to be used (Rayleigh-Sommerfeld propagator as described in section 2.1.2, or Fresnel propagator, as described in section 2.1.3).

In lines 39-44, the critical wavenumber and critical wavelength best matching the reference wavelength lambda_0 are calculated. Lines 46-53 determine the optimal sizes for the sampling grid and the guarding grid (see sections 2.6 and 2.7). Line 55 defines the highest transverse xy-mode to be included in the simulation.

```
 1 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 2 % Calculating max(EV(R)), avg(EV(R)), min(EV(R))
 3 % of an attenuated 4f-cavity with deviating parameters
 4 % over a range of wavelengths
 5 % File: CPA_sim_007_r_curve_deviate.m
 6 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 7 
 8 clear all
 9 close all
10 folder = '007\\'; % folder for saving results
11 gpu = 1; % 0: CPU, 1: GPU, 2: mp library
12 if gpu == 1
13     % initialize GPUs
14     for ii = 1:gpuDeviceCount
15         gpuDevice(ii);
16     end
17 elseif gpu == 2
18     % Define numerical precision for mp library
19     mp.Digits(34);
20 end
21 % create constants 1, 2, 4, 8, 2*pi in required data format
22 [one, two, four, eight, twopi] = precision_constants(gpu);
23 pm = pval(gpu, '1e-12'); % one pico-meter
```

```
% Basic physical simulation parameters
lambda_0 = pval(gpu, '785e-9'); % base wavelength [m]
f = pval(gpu, '50e-3');         % focal length of lenses [m]
pupil = false;                  % pupil yes/no
NA = pval(gpu, '0.05');         % numerical aperture of pupil
lens_type = 2;                  % 1: spherical 2:aspherical, perfect lens
r0 = pval(gpu, 'sqrt(0.8)');    % reflection coefficient left mirror
rho = pval(gpu, '1');           % how much crit. attenuation?

% technical simulation parameters
TF = 0;                         % 0: Rayleigh Sommerfeld, 1: Fresnel
L_des = pval(gpu, '2.5e-3');    % desired side length of smaller grid
factor = 2;          % factor by which embedding grid should be larger

% longitudinal mode number best matching lambda_0
l_mode = round((atan(sqrt(one-r0^two)/r0))/twopi+eight*f/lambda_0);
% critical wavenumber best matching lambda_0
k_c = twopi*l_mode/(eight*f) - (atan(sqrt(1-r0^two)/r0))/(eight*f);
% critical wavelength best matching lambda_0
lambda_c = twopi/k_c;

% create optimal grid
[N_small, L_small, N_large, L_large, alpha_max, n_max] = ...
opt_grid_params(L_des, factor, f, f, true, lambda_0);
% display grid parameters
disp(['N1=', num2str(N_small,5), ' N2=', num2str(N_large,5), ' n_max=', num2str(n_max,5)])
% create xy-axes coordinates for "small" and "large" grid
ax_small = create_ax(gpu, N_small, L_small);
ax_large = create_ax(gpu, N_large, L_large);
% Create vector with xy-modes up to n=32/2=16
modes = sorted_mode_numbers(gpu, 32);
no_of_modes = size(modes,1);
% create lens phase mask
[lens_mask, lens_pupil] = lens(gpu, ax_large, lambda_0, pupil, NA, f, lens_type);

% define area of interest and iterate through it
points = pval(gpu, '900');% number of points to be calculated
period = twopi/(eight*f); % period between resonance points
aoi_width = period * pval(gpu,'1.5');  % area-of-interest width 1.5 periods
dk = aoi_width/points;     % delta k

% parameter variations
rho=[1, 1, 1, 1, 1];
d1 = [f, f, f, f, f];
d2 = [f*two, f*two, f*two, f*two, f*two];
d3 = [f, f+lambda_c/four, f+pval(gpu,'1e-3'), f+pval(gpu,'2e-3'), f+pval(gpu,'5e-3')];

% iterate through the parameter variations
for r = 1:width(d1)
    s=0;                        % counting-up-index
    for idx = round(points/two):-1:-round(points/two)
        s = s + 1;
        k = k_c +idx * dk;
        lambda = twopi/k;

        % generate transmission matrix for one-round-trip (no attenuation)
        T = transmision_matrix_round_trip_no_atten2(gpu, ax_small, ax_large, ...
            TF, modes, lens_mask, d1(r), d2(r), d3(r), lambda);

        % Calculate transmission matrix *with* attenuation
        T_atten = r0*rho(r)*T;

        % Calculate reflection matrix
        I = create_eye(gpu, no_of_modes);
        r1 = -r0^two - i * r0 * sqrt(one - r0^two); % left mirror refl. coeff.
        R = I * r1 + (one + r1)^two * T_atten / (I - r1 * T_atten);
        EV = eig(R); % Eigenvalues of reflection matrix
        xls(s,1) = idx;
        xls(s,2) = lambda / pm; % wavelength in nm
        xls(s,3) = (lambda-lambda_c) / pm; % delta lambda in pm
        xls(s,4) = max(abs(EV)); % largest eigenvalue
        xls(s,5) = mean(abs(EV)); % mean eigenvalue
        xls(s,6) = min(abs(EV)); % smallest eigenvalue
        xls(s,7) = xls(s,4)^2; % largest eigenvalue squared
        xls(s,8) = xls(s,5)^2; % mean eigenvalue squared
        xls(s,9) = xls(s,6)^2; % smallest eigenvalue squared

        disp(['idx: ', num2str(idx,5), ...
            ' max EV: ', num2str(max(abs(EV)),5), ...
            ' mean EV: ', num2str(mean(abs(EV)),5), ...
            ' min EV: ', num2str(min(abs(EV)),5)])
    end
    % save results
    filename = strcat(folder, 'EV_var_', num2str(r,5),'.xlsx');
    writematrix(xls, filename);
end
```

The core part of the program starts in line 60. The number of points to be calculated is defined in line 61, and also the area of interest (in the program on the previous page it is 1.5 times a period around the resonance point, see line 63).

In lines 66-70 a number of parameter vectors are defined. They all have to be of the same size, and the number of entries defines the number of simulations to be performed:

- `rho`: This vector in line 67 defines the factors by which the critical attenuation gets multiplied in each simulation round. Consequently, a value of `1` results in exactly critical attenuation, whereas a value smaller (larger) than `1` results in undercritical (overcritical) attenuation.
- `d1`: This vector in line 68 defines the position of the first lens in each simulation round.
- `d2`: This vector in line 69 defines the distance between the first and the second lens in each simulation round.
- `d3`: This vector in line 70 defines the distance between the second lens and the total reflective back-mirror in each simulation round.

Hence, the vectors defined in lines 66-70 specify five simulation rounds, each with critical attenuation and the first and second lens being in the optimal position, but with the back mirror in a slightly different position in each round (positions f, $f + \frac{\lambda}{4}$, $f + 1\,\mathrm{mm}$, $f + 2\,\mathrm{mm}$, and $f + 5\,\mathrm{mm}$).

The loop starting in line 73 iterates through all simulation rounds. The inner loop, starting in line 75, iterates through various wavenumbers in the area of interest.

Using method `transmission_matix_round_trip` (see appendix C.21), the transmission-matrix `T` of an unattenuated 4d-cavity with the given physical parameters and the wavenumber of the current loop-iteration is generated.

In line 85, the attenuation-factor is multiplied to the transmission matrix of the unattenuated cavity. Based on that, the total reflection matrix `R` of the cavity is calculated in lines 87-90 using equation (3.39).

Finally the eigenvalues of `R` are calculated in lines 91. The smallest, average and largest eigenvalue is stored in the `xls`-array. When the inner loop has ended and all data-points of the current simulation-round have been calculated, a unique file-name is generated in line 108, and the results are finally written into a file in line 109.

E.8. `CPA_sim_008_r_curve_deviate`

The program listed below was used to calculate the results of the "tilted-back-mirror"-simulations in section 4.4. It is an enhanced version of the program `CPA_sim_007_r_curve_deviate` as documented in appendix E.7. Like `CPA_sim_007_r_curve_deviate`, it supports multiple simulation-rounds with varying simulation-parameters, and calculates in each round the maximum, average, and minimum eigenvalues of a 4f cavity's reflection-matrix for various wavelengths around a critical wavelength. However, this enhanced program version does not only allow to vary the attenuation and the positions of the two lenses or the back-mirror in each round, but also

- allows to vary the angle of the back mirror against the z-axis in the yz-plane in each round, and
- writes all squared absolute eigenvalues at resonance wavelength in ascending order into a separate `.xlsx`-file in each simulation-round, similar to the `CPA_sim_003_mode_decomposition` program (see appendix E.3).

In lines 25-32, basic physical simulation parameters common for all simulation rounds are defined; amongst them the base wavelength `lambda_0` in line 26, the focal length in line 27, and the type of lens to be used in line 30. The parameter `r0` in line 31 is the absolute value of the left mirror's reflection coefficient.

Among the technical simulation parameters in lines 34-37 there is the parameter `TF` which defines the propagator method to be used (Rayleigh-Sommerfeld propagator as described in section 2.1.2, or Fresnel propagator, as described in section 2.1.3).

In lines 39-44, the critical wavenumber and critical wavelength best matching the reference wavelength `lambda_0` are calculated. Lines 46-53 determine the optimal sizes for the sampling grid and the guarding grid (see sections 2.6 and 2.7). Line 55 defines the highest transverse xy-mode to be included in the simulation.

The core part of the program starts in line 60. The number of points to be calculated is defined in line 61, and also the area of interest (in the program below it is 1.5 times a period around the resonance point, see line 63).

```
 1 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 2 % Calculating max(EV(R)), avg(EV(R)), min(EV(R))
 3 % of a attenuated 4f-cavity with deviating parameters
 4 % over a range of wavelengths
 5 % File: CPA_sim_008_r_curve_deviate.m
 6 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 7
 8 clear all
 9 close all
10 folder = '008\\'; % folder for saving results
11 gpu = 1; % 0: CPU, 1: GPU, 2: mp library
12 if gpu == 1
13     % initialize GPUs
14     for ii = 1:gpuDeviceCount
15         gpuDevice(ii);
16     end
17 elseif gpu == 2
18     % Define numerical precision for mp library
19     mp.Digits(34);
20 end
21 % create constants 1, 2, 4, 8, 2*pi in required data format
22 [one, two, four, eight, twopi] = precision_constants(gpu);
23 pm = pval(gpu, '1e-12'); % one pico-meter
24
25 % Basic physical simulation parameters
26 lambda_0 = pval(gpu, '785e-9'); % base wavelength [m]
27 f = pval(gpu, '50e-3');          % focal length of lenses [m]
28 pupil = false;                   % pupil yes/no
29 NA = pval(gpu, '0.05');          % numerical aperture of pupil
30 lens_type = 2;                   % 1: spherical 2:aspherical, perfect lens
31 r0 = pval(gpu, 'sqrt(0.8)');     % reflection coefficient left mirror
32 rho = pval(gpu, '1');            % how much crit. attenuation?
33
34 % technical simulation parameters
35 TF = 0;                          % 0: Rayleigh Sommerfeld, 1: Fresnel
36 L_des = pval(gpu, '2.5e-3');     % desired side length of smaller grid
37 factor = 2;          % factor by which embedding grid should be larger
38
39 % longitudinal mode number best matching lambda_0
40 l_mode = round((atan(sqrt(one-r0^two)/r0))/twopi+eight*f/lambda_0);
41 % critical wavenumber best matching lambda_0
42 k_c = twopi*l_mode/(eight*f) - (atan(sqrt(1-r0^two)/r0))/(eight*f);
43 % critical wavelength best matching lambda_0
44 lambda_c = twopi/k_c;
45
46 % create optimal grid
47 [N_small, L_small, N_large, L_large, alpha_max, n_max] = ...
48 opt_grid_params(L_des, factor, f, f, true, lambda_0);
49 % display grid parameters
50 disp(['N1=', num2str(N_small,5), ' N2=', num2str(N_large,5), ' n_max=', num2str(n_max,5)])
51 % create xy-axes coordinates for "small" and "large" grid
52 ax_small = create_ax(gpu, N_small, L_small);
53 ax_large = create_ax(gpu, N_large, L_large);
54 % Create vector with xy-modes up to n=32/2=16
55 modes = sorted_mode_numbers(gpu, 32);
56 no_of_modes = size(modes,1);
57 % create lens phase mask
58 [lens_mask, lens_pupil] = lens(gpu, ax_large, lambda_0, pupil, NA, f, lens_type);
59
60 % define area of interest and iterate through it
61 points = pval(gpu, '900');% number of points to be calculated
62 period = twopi/(eight*f); % period between resonance points
63 aoi_width = period * pval(gpu,'1.5');  % area-of-interest width 1.5 periods
64 dk = aoi_width/points;    % delta k
65
66 % parameter variations
67 rho= [1, 1, 1, 1, 1, 1];
68 d1 = [f, f, f, f, f, f];
69 d2 = [f*two, f*two, f*two, f*two, f*two, f*two];
70 d3 = [f, f, f, f, f, f];
71 mirror_tilt = [0.005, 0.003, 0.001, 0.0005, 0.0003, 0.0001]; % degrees
```

In lines 66-70 a number of parameter vectors are defined. They all have to be of the same size, and the number of entries defines the number of simulations to be performed:

```
 73 % iterate through the paramter variations
 74 for r = 1:width(d1)
 75     s=0;                    % counting-up-index
 76     for idx = round(points/two):-1:-round(points/two)
 77         s = s + 1;
 78         k = k_c +idx * dk;
 79         lambda = twopi/k;
 80 
 81         % generate single-round-trip transmission matrix no attenuation)
 82         T = transmision_matrix_round_trip_no_atten3(gpu, ax_small, ax_large, ...
 83             TF, modes, lens_mask, d1(r), d2(r), d3(r), mirror_tilt(r), lambda);
 84 
 85         % Calculate transmission matrix *with* attenuation
 86         T_atten = r0 * rho(r) * T;
 87 
 88         % Calculate reflection matrix
 89         I = create_eye(gpu, no_of_modes);
 90         r1 = -r0^two - i * r0 * sqrt(one - r0^two); % left mirror refl. coeff.
 91         R = I * r1 + (one + r1)^two * T_atten / (I - r1 * T_atten);
 92         [V, D] = eig(R); % Eigenvectors and diagonal eigenvalue matrix
 93         EV = diag(D); % get reflection matrix eigenvalues from D matrix
 94 
 95         if idx == 0
 96             %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 97             % at critical wavelength:
 98             % generate eigenmode decomposition
 99             %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
100             EVS = abs(EV).^2; % Eigenvalues squared
101             [EVS_sorted, sortEVS_idx] = sort(EVS); % sort sqared eigenvalues
102             V_sorted = V(:,sortEVS_idx); % eigenvectors sorted by eigenvalues
103             % generate result matrix
104             mode_xls = [1:size(EVS,1)]'; % 1st column: ascending number of mode
105             mode_xls(:,2) = EVS_sorted; % second column: squared eigenvalues
106             % save eigenmode decompoition results
107             filename_m = strcat(folder, 'modes_', num2str(r,5),'.xlsx');
108             writematrix(mode_xls, filename_m);
109         end
110 
111         xls(s,1) = idx;
112         xls(s,2) = lambda / pm; % wavelength in nm
113         xls(s,3) = (lambda-lambda_c) / pm; % delta lambda in pm
114         xls(s,4) = max(abs(EV)); % largest eigenvalue
115         xls(s,5) = mean(abs(EV)); % mean eigenvalue
116         xls(s,6) = min(abs(EV)); % smalles eigenvalue
117         xls(s,7) = xls(s,4)^2; % largest eigenvalue squared
118         xls(s,8) = xls(s,5)^2; % mean eigenvalue squared
119         xls(s,9) = xls(s,6)^2; % smalles eigenvalue squared
120 
121         disp(['idx: ', num2str(idx,5), ...
122             ' max EV: ', num2str(max(abs(EV)),5), ...
123             ' mean EV: ', num2str(mean(abs(EV)),5), ...
124             ' min EV: ', num2str(min(abs(EV)),5)])
125     end
126     % save results
127     filename = strcat(folder, 'EV_var_', num2str(r,5),'.xlsx');
128     writematrix(xls, filename);
129 end
```

- `rho`: This vector in line 67 defines the factors by which the critical attenuation gets multiplied in each simulation round. Consequently, a value of 1 results in exactly critical attenuation, whereas a value smaller (larger) than 1 results in undercritical (overcritical) attenuation.
- `d1`: This vector in line 68 defines the position of the first lens in each simulation round.
- `d2`: This vector in line 69 defines the distance between the first and the second lens in each simulation round.
- `d3`: This vector in line 70 defines the distance between the second lens and the total reflective back-mirror in each simulation round.
- `mirror_tilt`: Angle in degrees by which the mirror is tilted relative to the z-axis in the yz-plane.

Hence, the vectors defined in lines 66-71 specify six simulation rounds, each with critical attenuation and all lenses and mirrors in optimal position, but with the back mirror tilted between 0.005° and 0.0001°.

The loop starting in line 74 iterates through all simulation rounds. The inner loop, starting in line 76, iterates through various wavenumbers in the area of interest.

Using method `transmission_matix_round_trip` (see appendix C.21), the transmission-matrix `T` of an unattenuated 4d-cavity with the given physical parameters and the wavenumber of the current loop-iteration is generated.

In line 86, the attenuation-factor is multiplied to the transmission matrix of the unattenuated cavity. Based on that, the total reflection matrix `R` of the cavity is calculated in lines 88-91 using equation (3.39).

Finally the eigenvalues of `R` are calculated in lines 92-93. When the inner loop is exactly at the critical wavelength, all squared absolute eigenvalues are sorted in ascending order, and stored into a separate `.xlsx`-file (lines 95-109).

In each iteration of the inner loop, the smallest, average, and largest eigenvalues are stored in the `xls`-array. When the inner loop has ended and all data-points of the current simulation-round have been calculated, a unique file-name is generated in line 128, and the results are finally written into a file in line 128.

E.9. `CPA_sim_009`

The program below demonstrates how the concept of simulating a 4f-cavity-CPA by means of scattering matrices and transfer matrices can be extended from a one-dimensional toy-model (section 5.2) to a realistic model of a 4f-cavity (section 5.3).

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% File: CPA_sim_009D.m
% Proof of Concept: 4f-Cavity-CPA simulation with
% Scattering and Transfer Matrices
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all
close all
gpu = 1; % 0: CPU, 1: GPU, 2: mp library
if gpu == 1
    % initialize GPUs
    for ii = 1:gpuDeviceCount
        gpuDevice(ii);
    end
elseif gpu == 2
    % Define numerical precision for mp library
    mp.Digits(34);
end
% create constants 1, 2, 4, 8, 2*pi in required data format
[one, two, four, eight, twopi] = precision_constants(gpu);
pm = pval(gpu, '1e-12'); % one pico-meter

% Basic physical simulation parameters
lambda_0 = pval(gpu, '785e-9'); % base wavelength [m]
f = pval(gpu, '0.100');         % focal length of lenses [m]
pupil = false;                  % pupil yes/no
NA = pval(gpu, '0.05');         % numerical aperture of pupil
lens_type = 2;                  % 1: spherical 2:aspherical, perfect lens
r0 = pval(gpu, 'sqrt(0.8)');    % reflection coefficient left mirror

% technical simulation parameters
TF = 0;                         % 0: Rayleigh Sommerfeld, 1: Fresnel
L_des = pval(gpu, '1.4e-3');    % desired side length of smaller grid
factor = 1.25;         % factor by which embedding grid should be larger

% longitudinal mode number best matching lambda_0
l_mode = round((atan(sqrt(one-r0^two)/r0))/twopi+eight*f/lambda_0);
% critical wavenumber best matching lambda_0
k_c = twopi*l_mode/(eight*f) - (atan(sqrt(1-r0^two)/r0))/(eight*f);
% critical wavelength best matching lambda_0
lambda_c = twopi/k_c;

% create optimal grid
[N_small, L_small, N_large, L_large, alpha_max, n_max] = ...
opt_grid_params(L_des, factor, f, f, true, lambda_c);
% display grid parameters
disp(['N1=', num2str(N_small,5), ' N2=', num2str(N_large,5), ' n_max=', num2str(n_max,5)])
% create xy-axes coordinates for "small" and "large" grid
ax_small = create_ax(gpu, N_small, L_small);
ax_large = create_ax(gpu, N_large, L_large);
% Create vector with all xy-modes
modes_large = sorted_mode_numbers(gpu, N_large);
no_of_modes = size(modes_large,1);
% create lens phase mask
[lens_mask, lens_pupil] = lens(gpu, ax_large, lambda_c, pupil, NA, f, lens_type);

% Absorber data
d_absorb = pval(gpu, '0.01');
nr = pval(gpu, '1.5');
ni = -log(r0)/(two*d_absorb*k_c);
n_absorb = nr + i*ni;
d_absorb_opt = d_absorb * nr; % optical thickness absorber

% Simulation at k=k_c and lambda = lambda_c
k = k_c;
lambda = twopi/k;

I = create_eye(gpu, no_of_modes); % unity matrix
Z = create_zeros(gpu, no_of_modes); % zero matrix

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Create Left Mirror Scattering Matrix S1.
% Then create Transfer Matrix M1 from Scattering Matrix S1.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
r1 = -r0^two - i * r0 * sqrt(one - r0^two); % left mirror refl. coeff.
t1 = one + r1; % left mirror transmission coeff.
S1 = create_S_matrix(gpu, I*r1, I*t1);
M1 = convert_S_to_M(S1);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Create TP_d1:  Transmission matrix from propagation of distance d1
% (distance between first lens to begin of absorber and
% also distance between end of absorber and begin of second lens)
% Then create Scattering Matrix SP_d1 from Transmission Matrix TP_d1.
% Then create Transfer Matrix MP_d1 from Scattering Matrix SP_d1.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
d1 = f-d_absorb_opt/two;
TP_d1 = transmision_matrix_prop(...
    gpu, TF, ax_large, d1, lambda, 1, modes_large);
SP_d1 = create_S_matrix(gpu, Z, TP_d1);
MP_d1 = convert_S_to_M(SP_d1);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Create TA: Transmission matrix for propagation through absorber
% Then create Scattering Matrix SSA from Transmission Matrix TA.
% Then create Transfer Matrix MA from Scattering Matrix SA.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
TA = transmision_matrix_prop(...
    gpu, TF, ax_large, d_absorb, lambda, n_absorb, modes_large);
SA = create_S_matrix(gpu, Z, TA);
MA = convert_S_to_M(SA);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Create TL: Transmission matrix for lens
% Then create Scattering Matrix SL from Transmission Matrix TL.
% Then create Transfer Matrix ML from Scattering Matrix SL.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
TL = transmision_matrix_lens(...
    gpu, ax_large, lambda, pupil, NA, f, lens_type, modes_large);
SL = create_S_matrix(gpu, Z, TL);
ML = convert_S_to_M(SL);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Create TP_f:  Transmission matrix from propagation of distance f
% Then create Scattering Matrix SP_f from Transmission Matrix TP_f.
% Then create Transfer Matrix MP_f from Scattering Matrix SP_f.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
TP_f = transmision_matrix_prop(...
    gpu, TF, ax_large, f, lambda, 1, modes_large);
SP_f = create_S_matrix(gpu, Z, TP_f);
MP_f = convert_S_to_M(SP_f);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Create Right Mirror Scattering Matrix S2.
% Then create Transfer Matrix M2 from Scattering Matrix S2.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
epsilon = pval(gpu, '1e-18');
S2 = create_S_matrix(gpu, -I, I*epsilon);
M2 = convert_S_to_M(S2);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Create Total 4f cavity Transfer Matrix T4f
% Left mirror - propagate f - lens - propagate d1 - propagate through
% absorber - propagate d1 - lens - proagate f - right mirror
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
M4f = M1 * MP_f * ML * MP_d1 * MA * MP_d1 * ML * MP_f * M2;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% (Partially) convert total Cavity Transfer Matrix into total Cavity
% Scattering Matrix (just the top-left quadrant - which is the (ouside)
% Cavity Reflection Matrix R), and downscale to observation grid
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
R_large = convert_M_to_R(M4f);
[R_small, modes_small] = ...
    downscale_TR_matrix(gpu, R_large, ax_small, ax_large, modes_large);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Show Reflection Matrix element-wiseley squared
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
figure(2);
imagesc(abs(R_small).^2)
axis square
title('Reflection Matrix Squared')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Show effect of single-trip transmission matrix on test-image
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T_large = TP_f * TL * TP_d1 * TA * TP_d1 * TL * TP_f; % single trip transmission matrix
[T_small, modes_small] = ...
    downscale_TR_matrix(gpu, T_large, ax_small, ax_large, modes_large) ;
E_in = create_test_image(gpu, ax_small, 4, false);
E_vec = fft_arr_to_vec(E_in, ax_small, modes_small, false);
E_vec = T_small*E_vec;
E = fft2_vec_to_arr(gpu, E_vec, ax_small, modes_small, false);
plot_fields(gpu, 1, ax_small, E_in, 'Input Field', 3, E, 'Output Field', 3);
```

In lines 23-29, basic physical simulation parameters are defined; amongst them the base wavelength `lambda_0` in line 24, the focal length in line 25, and the type of lens to be used in line 28. The parameter `r0` in line 29 is the absolute value of the left mirror's reflection coefficient.

Among the technical simulation parameters in lines 31-34 there is the parameter `TF` which defines the propagator method to be used (Rayleigh-Sommerfeld propagator as described in section 2.1.2, or Fresnel propagator, as described in section 2.1.3).

In lines 36-41, the critical wavenumber and critical wavelength best matching the reference wavelength `lambda_0` are calculated. Lines 43-45 determine the optimal sizes for the sampling grid and the guarding grid (see sections 2.6 and 2.7). In lines 49 and 50, the discrete axes coordinate vectors for the smaller and larger grid are created. The transfer matrices and scattering matrices correspond to the larger guarding grid size. Consequently a vector containing all transverse n_x/n_y mode numbers matching the side length of the larger grid is created (line 52 and 53). In line 55, a phase-mask to be used for simulating the two lenses is created.

The program is to simulate a 4f-cavity with an absorber of thickness $d = 10\,\text{mm}$ in the center position. The parameters of the absorber are defined in lines 57-62. After the thickness (line 58), the real and imaginary parts of the absorber's refractive index are defined in lines 59-61. The real part of the refractive index increases the optical thickness of the absorber (line 62).

The simulation itself starts in line 71. In lines 71-78, the scattering matrix `S1` for the (partially reflective) left mirror is created; and then converted into the corresponding transfer matrix `M1`.

In lines 80-91, a transmission matrix `TP_d1` is created, which is able to simulate the propagation corresponding to the distance between the first lens and the beginning of the center-position absorber, and between the end of the center-position absorber and the second lens, respectively. From this transmission matrix `TP_d1`, a scattering matrix `S_d1` is generated, which is eventually converted into the corresponding transfer matrix `M_d1`.

In lines 93-101, a transmission matrix `TA` is created, simulating the propagation through the absorber (considering the optical propagation distance and the complex refractive index). From this transmission matrix `TA`, a scattering matrix `SA` is generated, which is eventually converted into the corresponding transfer matrix `MA`.

In lines 103-111, a transmission matrix `TL` is created, which is able to simulate the effect of the thin lens. From this transmission matrix `TL`, a scattering matrix `SL` is generated, which is eventually converted into the corresponding transfer matrix `ML`.

In lines 113-121, a transmission matrix `TP_f` is created, which is able to simulate the propagation distance of one focal length, which is the distance between the first mirror and the first lens, and also between the second lens and the total reflective mirror, respectively. From this transmission matrix `TP_f`, a scattering matrix `S_f` is generated, which is eventually converted into the corresponding transfer matrix `M_f`.

In lines 123-129, the scattering matrix `S2` for the right (total reflective) mirror is created; and then converted into the corresponding transfer matrix `M2`. Please note that the sub-matrix of the scattering-matrix that is representing the total reflective mirror's transmission behavior is not set to a zero-value-matrix, but rather to a diagonal matrix with a small `epsilon`-value to avoid singularity when `S2` is converted to `M2`. As explained in section 5.2.1, this is no problem, as this specific value does not affect the resulting total reflection matrix.

In line 136, the total transfer matrix `M4f` for the whole 4f-cavity is calculated by multiplying all transfer matrices in the correct order. From this `M4f` matrix the top-left quadrant of the corresponding scattering matrix is calculated in line 143, which happens to be the cavity's reflection matrix. As the whole calculation is done based on the larger guarding-grid, the reflection-matrix is finally down-sized to the reflection matrix `R_small`, corresponding to the smaller observation grid size (lines 114-115).

Eventually, in lines 147-154, the reflection matrix with element-wisely squared values is displayed, and in lines 155-163 the total transmission matrix for a single trip through the cavity is calculated, and its effect on a simple test image (an off-center letter "T") is demonstrated.

E.10. `CPA_sim_010`

The program presented in this section again uses the method of scattering and transfer matrices to simulate a 4f-cavity with an absorber in the middle position. In contrast to the previous proof-of-concept program `CPA_sim_009`, it allows to run several simulation rounds with different simulation parameters (namely, absorber thickness and reflectivity of the absorber facets).

The maximum, average, and minimum eigenvalues of the 4f cavity's reflection-matrix are calculated for various wavelengths around a critical wavelength, and stored into a single file per simulation round. In addition to that, separate files are generated at each calculated wavelength, containing a full eigenmode decomposition at the given wavelength with the simulation parameters of the respective round.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% File: CPA_sim_010.m
% 4f-Cavity-CPA simulation with Scattering Matrices
% and Transfer Matrices and reflective facets on absorber
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all
close all
folder = '010\\'; % folder for saving results
gpu = 1; % 0: CPU, 1: GPU, 2: mp library
if gpu == 1
    % initialize GPUs
    for ii = 1:gpuDeviceCount
        gpuDevice(ii);
    end
elseif gpu == 2
    % Define numerical precision for mp library
    mp.Digits(34);
end
% create constants 1, 2, 4, 8, 2*pi in required data format
[one, two, four, eight, twopi] = precision_constants(gpu);
pm = pval(gpu, '1e-12'); % one pico-meter

% Basic physical simulation parameters
lambda_0 = pval(gpu, '785e-9'); % base wavelength [m]
f = pval(gpu, '0.100');         % focal length of lenses [m]
pupil = false;                  % pupil yes/no
NA = pval(gpu, '0.05');         % numerical aperture of pupil
lens_type = 2;                  % 1: spherical 2:aspherical, perfect lens
r0 = pval(gpu, 'sqrt(0.8)');    % reflection coefficient left mirror

% technical simulation parameters
TF = 0;                         % 0: Rayleigh Sommerfeld, 1: Fresnel
L_des = pval(gpu, '1.4e-3');    % desired side length of smaller grid
factor = 1.25;          % factor by which embedding grid should be larger

% longitudinal mode number best matching lambda_0
l_mode = round((atan(sqrt(one-r0^two)/r0))/twopi+eight*f/lambda_0);
% critical wavenumber best matching lambda_0
k_c = twopi*l_mode/(eight*f) - (atan(sqrt(1-r0^two)/r0))/(eight*f);
% critical wavelength best matching lambda_0
lambda_c = twopi/k_c;

% create optimal grid
[N_small, L_small, N_large, L_large, alpha_max, n_max] = ...
opt_grid_params(L_des, factor, f, f, true, lambda_c);
% display grid parameters
disp(['N1=', num2str(N_small,5), ' N2=', num2str(N_large,5), ' n_max=', num2str(n_max,5)])
% create xy-axes coordinates for "small" and "large" grid
ax_small = create_ax(gpu, N_small, L_small);
ax_large = create_ax(gpu, N_large, L_large);
% Create vector with all xy-modes
modes_large = sorted_mode_numbers(gpu, N_large);
no_of_modes_large = size(modes_large,1);
% create lens phase mask
[lens_mask, lens_pupil] = lens(gpu, ax_large, lambda_c, pupil, NA, f, lens_type);
```

```
% define area of interest
points = pval(gpu, '900');% number of points to be calculated
period = twopi/(eight*f); % period between resonance points
aoi_width = period * pval(gpu,'1.5');  % area-of-interest width 1.5 periods
dk = aoi_width/points;     % delta k

I = create_eye(gpu, no_of_modes_large); % unity matrix
Z = create_zeros(gpu, no_of_modes_large); % zero matrix
r1 = -r0^two - i * r0 * sqrt(one - r0^two); % left mirror refl. coeff.
t1 = one + r1; % left mirror transmission coeff.

%%%%%%%%%%%%%%%%%%%%%%%%%%%
% parameter vectors
%%%%%%%%%%%%%%%%%%%%%%%%%%%
 % simulate reflecting absorber facets
absorber_facets = [true, true, true, true, true, true, true, true];
 % absorber reflections
raf4 = pval(gpu, 'sqrt(1e-4)');
raf3 = pval(gpu, 'sqrt(1e-3)');
raf2 = pval(gpu, 'sqrt(1e-2)');
raf05 = pval(gpu, 'sqrt(0.5)');
raf0 = [raf4, raf3, raf2, raf05, raf4, raf3, raf2, raf05];
% absorber thickness
d1mm = pval(gpu, '0.001');
d10mm = pval(gpu, '0.01');
d_absorb = [d10mm, d10mm, d10mm, d10mm, d1mm, d1mm, d1mm, d1mm];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% iterate through the parameter variations
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for r = 1:width(d_absorb)
    disp(['SIMULATION ROUND: ', num2str(r,5)])
    % Absorber data
    nr = pval(gpu, '1.5');
    ni = -log(r0)/(two*d_absorb(r)*k_c);
    n_absorb = nr + i*ni;
    d_absorb_opt = d_absorb(r) * nr; % optical thickness absorber
    raf = -raf0(r)^two - i * raf0(r) * sqrt(one - raf0(r)^two); % absorb. facet refl. coeff.
    taf = one + raf; % absorber facet transmission coeff.
    s=0; % counting-up-index
    for idx = round(points/two):-1:-round(points/two)
        s = s + 1;
        k = k_c +idx * dk;
        lambda = twopi/k;

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Create Left Mirror Scattering Matrix S1.
        % Then create Transfer Matrix M1 from Scattering Matrix S1.
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        r1 = -r0^two - i * r0 * sqrt(one - r0^two); % left mirror refl. coeff.
        t1 = one + r1; % left mirror transmission coeff.
        S1 = create_S_matrix(gpu, I*r1, I*t1);
        M1 = convert_S_to_M(S1);

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Create TP_d1:  Transmission matrix from propagation of distance d1
        % (distance between first lens to begin of absorber and
        % also distance between end of absorber and begin of second lens)
        % Then create Scattering Matrix SP_d1 from Transmission Matrix TP_d1.
        % Then create Transfer Matrix MP_d1 from Scattering Matrix SP_d1.
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        d1 = f-d_absorb_opt/two;
        TP_d1 = transmision_matrix_prop(...
            gpu, TF, ax_large, d1, lambda, 1, modes_large);
        SP_d1 = create_S_matrix(gpu, Z, TP_d1);
        MP_d1 = convert_S_to_M(SP_d1);

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Create TA: Transmission matrix for propagation through absorber
        % Then create Scattering Matrix SA from Transmission Matrix TA.
        % Then create Transfer Matrix MA from Scattering Matrix SA.
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        TA = transmision_matrix_prop(...
            gpu, TF, ax_large, d_absorb(r), lambda, n_absorb, modes_large);
        SA = create_S_matrix(gpu, Z, TA);
        MA = convert_S_to_M(SA);

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % If required: Create Absorber Facet Scattering Matrix SAF.
        % It describes the reflection on the left absorber side.
        % Then create Transfer Matrix TAF from Scattering Matrix SAF.
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        if absorber_facets(r)
            SF = create_S_matrix(gpu, I*raf, I*taf);
            MF = convert_S_to_M(SF);
        end
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Create TL: Transmission matrix for lens
        % Then create Scattering Matrix SL from Transmission Matrix TL.
        % Then create Transfer Matrix ML from Scattering Matrix SL.
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        TL = transmision_matrix_lens(...
            gpu, ax_large, lambda, pupil, NA, f, lens_type, modes_large);
        SL = create_S_matrix(gpu, Z, TL);
        ML = convert_S_to_M(SL);

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Create TP_f:  Transmission matrix from propagation of distance f
        % Then create Scattering Matrix SP_f from Transmission Matrix TP_f.
        % Then create Transfer Matrix TP_f from Scattering Matrix SP_f.
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        TP_f = transmision_matrix_prop(...
            gpu, TF, ax_large, f, lambda, 1, modes_large);
        SP_f = create_S_matrix(gpu, Z, TP_f);
        MP_f = convert_S_to_M(SP_f);

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Create Right Mirror Scattering Matrix S2.
        % Then create Transfer Matrix M2 from Scattering Matrix S2.
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        epsilon = pval(gpu, '1e-18');
        S2 = create_S_matrix(gpu, -I, I*epsilon);
        M2 = convert_S_to_M(S2);

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Create Total 4f cavity Transfer Matrix M4f
        % Left mirror - propagate f - lens - propagate d1 - propagate through
        % absorber - propagate d1 - lens - proagate f - right mirror
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        if absorber_facets(r)
            M4f = M1 * MP_f * ML * MP_d1 * MF * MA * MF * MP_d1 * ML * MP_f * M2;
        else
            M4f = M1 * MP_f * ML * MP_d1 * MA * MP_d1 * ML * MP_f * M2;
        end

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % (Partially) convert total Cavity Transfer Matrix into total Cavity
        % Scattering Matrix (just the top-left quadrant - which is the (ouside)
        % Cavity Reflection Matrix R), and downscale to observation grid
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        R_large = convert_M_to_R(M4f);
        [R_small, modes_small] = ...
            downscale_TR_matrix(gpu, R_large, ax_small, ax_large, modes_large);

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Calculate Eigenvectors and Eigenvalues
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        [V, D] = eig(R_small); % Eigenvectors and diagonal eigenvalue matrix
        EV = diag(D); % get reflection matrix eigenvalues from D matrix

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % generate eigenmode decomposition for the current wavelength
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        EVS = abs(EV).^2; % Eigenvalues squared
        [EVS_sorted, sortEVS_idx] = sort(EVS); % sort sqared eigenvalues
        V_sorted = V(:,sortEVS_idx); % eigenvectors sorted by eigenvalues
        % generate result matrix
        mode_xls = [1:size(EVS,1)]'; % 1st column: ascending number of mode
        mode_xls(:,2) = EVS_sorted; % second column: squared eigenvalues
        % save eigenmode decompoition results
        idx_str = sprintf('%04d',idx);
        filename_m = strcat(folder, 'modes_', num2str(r,5), ...
            '_', idx_str, '.xlsx');
        writematrix(mode_xls, filename_m);
        xls(s,1) = idx;
        xls(s,2) = lambda / pm; % wavelength in nm
        xls(s,3) = (lambda-lambda_c) / pm; % delta lambda in pm
        xls(s,4) = max(abs(EV)); % largest eigenvalue
        xls(s,5) = mean(abs(EV)); % mean eigenvalue
        xls(s,6) = min(abs(EV)); % smallest eigenvalue
        xls(s,7) = xls(s,4)^2; % largest eigenvalue squared
        xls(s,8) = xls(s,5)^2; % mean eigenvalue squared
        xls(s,9) = xls(s,6)^2; % smallest eigenvalue squared

        disp(['idx: ', num2str(idx,5), ...
            ' max EV: ', num2str(max(abs(EV)),5), ...
            ' mean EV: ', num2str(mean(abs(EV)),5), ...
            ' min EV: ', num2str(min(abs(EV)),5)])
    end
    filename = strcat(folder, 'EV_var_', num2str(r,5),'.xlsx');
    writematrix(xls, filename); % save results
end
```

Lines 1-56 of the above source code are practically identical to the source code in appendix E.9, except for line 9, where the folder for saving the result files is defined. In lines 58-62 the "area of interest" around the resonance point, and the number of points to be calculated are defined. In lines 66 and 67, the complex reflection and transmission coefficients of the left, partially reflective, mirror are calculated according to equations (3.9) and (3.10).

In lines 69-83, parameter vectors are defined which determine the round-dependent simulation parameters. In the source-code above, each of the parameter vectors has eight entries, meaning that there are eight simulation rounds. The vector `absorber_facets` defines that in all of the simulations an absorber with reflecting facets is to be simulated. In line 79 it is determined by vector `raf0` that the absolute value of the facets' reflection coefficient should be 0, 10^{-4}, 10^{-3}, 10^{-2}, and 0.5, in each of the first four simulation rounds, and then again in the next four rounds. Finally, vector `d_absorb` is initiated with eight entries, defining that the absorber thickness is 10 mm in the first four rounds, and then 1 mm in the next four rounds.

The loop starting in line 88 iterates through all simulation rounds. The real part of the absorber's refractive index is defined to be 1.5 in line 91. The imaginary part of the absorber's refractive index is defined in line 92 according to equation (5.23). In lines 95 and 96, the complex reflection and transmission coefficient of the absorber's facets is calculated according to equations (5.39) and (5.40).

The core part of the simulation starts with the loop in line 98, where the program iterates through all of the wavenumbers in the area of interest. The scattering matrices, and according transfer matrices are calculated in lines 103-171, just as in the `CPA_sim_009` program (see appendix E.10). The only new aspect is the calculation of the facets' scattering matrix `SF` and transfer matrix `MF` in lines 135-143. In lines 173-182 the cavity's total transfer matrix is calculated (either with or without reflecting facets). In lines 184-191, the cavity's total reflection matrix is calculated from the total transfer matrix, and then down-converted from the size of the simulation-grid to the smaller size of the observation grid by employing the method `downscale_TR_matrix` (see appendix C.29).

In lines 193-197, the eigenvalues and eigenvectors of the reflection matrix are calculated. In lines 199-212, a file is generated for each data point, containing the squared absolute value of all eigenvalues in ascending order.

Finally, in lines 213-221, the smallest, average, and largest absolute eigenvalues at each wavelength are added to the `xls` array, so that all of this data over the complete area of interest can be stored in a single, separate file after each simulation round (line 228-229).

Acknowledgments

First and foremost I am extremely grateful to my supervisors, Prof. Stefan Rotter and Dipl.-Ing. Kevin Pichler for their invaluable advice, continuous support, and their patience during this thesis research project. Additionally, I would like to express gratitude to Dr. Andre Brandstötter, who kindly supported me in the initial phase of my project. I would also like to thank Prof. Ori Katz, Gil Weinberg and Yevgeny Slobodkin from the Department of Applied Physics at the Hebrew University of Jerusalem for the experimental collaboration, which hopefully will soon back the calculations in this thesis with solid experimental data. Finally, I would like to thank my wife Michelle, who not only supported me emotionally during the compilation of this thesis, but occasionally also helped to fix some of my broken English.

List of Figures

Bibliography

[1] Y. D. Chong, L. Ge, H. Cao, and A. D. Stone. Coherent Perfect Absorbers: Time-Reversed Lasers. Physical Review Letters **105**, 053901 (2010).

[2] S. Longhi. Backward lasing yields a perfect absorber. Physics **3**, 61 (2010).

[3] W. Wan, Y. Chong, L. Ge, H. Noh, A. D. Stone, and H. Cao. Time-Reversed Lasing and Interferometric Control of Absorption. Science **331**, 889 (2011).

[4] Y. Sun, W. Tan, H.-q. Li, J. Li, and H. Chen. Experimental demonstration of a coherent perfect absorber with PT phase transition. Physical review letters **112**, 143903 (2014).

[5] C. W. Hsu, A. Goetschy, Y. Bromberg, A. D. Stone, and H. Cao. Broadband Coherent Enhancement of Transmission and Absorption in Disordered Media. Physical Review Letters **115**, 223901 (2015).

[6] H. Li, S. Suwunnarat, R. Fleischmann, H. Schanz, and T. Kottos. Random Matrix Theory Approach to Chaotic Coherent Perfect Absorbers. Physical Review Letters **118**, 044101 (2017).

[7] D. G. Baranov, A. Krasnok, and A. Alù. Coherent virtual absorption based on complex zero excitation for ideal light capturing. Optica **4**, 1457 (2017).

[8] K. Pichler, M. Kühmayer, J. Böhm, A. Brandstötter, P. Ambichl, U. Kuhl, and S. Rotter. Random anti-lasing through coherent perfect absorption in a disordered medium. Nature **567**, 351 (2019).

[9] J. C. Maxwell. A Treatise on Electricity and Magnetism. Clarendon Press, Oxford (1873).

[10] B. Kress. Applied digital optics : from micro-optics to nanophotonics. Wiley, Chichester, U.K (2009).

[11] J. W. Goodman. Introduction to Fourier Optics. McGraw Hill, second edn. (1996).

[12] J. Greivenkamp. Field guide to geometrical optics. SPIE Press, Bellingham, Wash (2004).

[13] N. George and X. Chen. Extended depth-of-field lenses and methods for their design, optimization and manufacturing. United States Patent US 7,898,746 B2 (2011).

[14] D. Voelz. Computational fourier optics : a MATLAB tutorial. SPIE Press, Bellingham, Wash (2011).

[15] D. G. Voelz and M. C. Roggemann. Digital simulation of scalar optical diffraction: revisiting chirp function sampling criteria and consequences. Appl. Opt. **48**, 6132 (2009).

[16] A. E. Siegman. Lasers (Revised). UNIVERSITY SCIENCE BOOKS (1986).

[17] Springer Handbook of Lasers and Optics. Springer-Verlag GmbH (2012).

[18] E. Salik. Quantitative investigation of Fresnel reflection coefficients by polarimetry. American Journal of Physics **80**, 216 (2012).

[19] G. Friedmann and H. S. Sandhu. Phase Change on Reflection from Isotropic Dielectrics. American Journal of Physics **33**, 135 (1965).

[20] G. B. Friedmann and H. S. Sandhu. Phase changes on reflection from a metallic surface. American Journal of Physics **56**, 270 (1988).

[21] V. Degiorgio. Phase shift between the transmitted and the reflected optical fields of a semireflecting lossless mirror is pi/2. American Journal of Physics **48**, 81 (1980).

[22] C. W. Gardiner and M. J. Collett. Input and output in damped quantum systems: Quantum stochastic differential equations and the master equation. Physical Review A **31**, 3761 (1985).

[23] J. Frei, X.-D. Cai, and S. Muller. Multiport S-Parameter and T-Parameter Conversion With Symmetry Extension. IEEE Transactions on Microwave Theory and Techniques **56**, 2493 (2008).

[24] A. Orefice, R. Giovanelli, and D. Ditto. Helmholtz wave trajectories in classical and quantum physics (2011).