GEORG-AUGUST-UNIVERSITÄT
GÖTTINGEN

Master's Thesis

Numerical methods for paraxial stationary and time-dependent propagation of x-ray and EUV radiation

Numerische Methoden zur paraxialen stationären und zeitabhängigen Propagation von Röntgenstrahlung und extremer ultravioletter Strahlung

prepared by

Lars Melchior

from Göttingen

at the Institut für Röntgenphysik

First referee: Prof. Dr. Tim Salditt
Second referee: Prof. Dr. Claus Ropers

ABSTRACT

Through a rigorous derivation of a scalar diffraction theory from Maxwell's equations, we generalize scalar diffraction to time-dependent problems. Well-known stationary propagation algorithms are explicitly derived and extended to radially symmetric and curved geometries. Periodic time-dependent solutions are obtained using discrete Fourier transforms of the stationary solutions. The algorithms are implemented in the Python package *PyPropagate*, allowing time-dependent full-wavefront propagation for arbitrary initial problems and matter distributions. We verify the implementation by comparing numerical results with analytical solutions. Using this framework we analyze time-dependent beam propagation in waveguides.

ZUSAMMENFASSUNG

Die skalare Beugungstheorie wird durch eine genaue Herleitung aus den Maxwell-Gleichungen auf zeitabhängige Fragestellungen verallgemeinert. Wohlbekannte Algorithmen zur stationären Propagation werden explizit hergeleitet und auf radialsymmetrische und gekrümmte Geometrien verallgemeinert. Periodische, zeitabhängige Lösungen werden über diskrete Fouriertransformationen aus den stationären Lösungen gewonnen. Die Algorithmen sind in dem *PyPropagate* Paket für Python implementiert und erlauben das volle Wellenfeld zeitabhängig für beliebige Anfangswerte und Materieverteilungen zu berechnen. Die Implementation wird durch den Vergleich mit analytischen Lösungen validiert. Das Paket wird schließlich benutzt um zeitabhängige Strahlpropagation in Wellenleitern zu untersuchen.

CONTENTS

1

INTRODUCTION

Numerical simulations play an important role in modern science and engineering, as they allow us to study the behaviour of systems that are too complex or impossible to solve analytically. In x-ray optics they are used, among other things, for the design of optical elements, for reconstruction problems and for the preparation and interpretation of experiments. With the upcoming of new radiation sources that offer high-energy ultra-short coherent beams, such as free electron lasers [1], there is a need for fast and accurate simulation algorithms for time-resolved propagation of x-ray and extreme ultraviolet radiation.

Waveguides are versatile optical elements used for many optical manipulations, such as filtering, confining, guiding, coupling or splitting beams. Their potential for x-ray optics is just getting discovered [2] and they could become a useful tool in time-resolved x-ray imaging. Accurate simulations of electromagnetic beam propagation in waveguides is essential for determining optimal design parameters.

The physics of optics is governed by Maxwell's equations, which have been published in 1865 and describe the dynamics of two coupled vectorial fields, the electric and magnetic field [3]. In optical diffraction one is usually not interested in the full electromagnetic field vectors but in the average energy density transported by these fields, as it corresponds to an easily measurable observable. Here, one frequently works with a single complex scalar field, often referred to as the light disturbance, whose squared magnitude is proportional to the average energy density. The dynamics of the disturbance are then determined by the scalar Helmholtz equation, a wave equation that is also found in many other physical fields. However, the exact nature of the disturbance has historically been a topic of discussion (see for example [4, 5, 6, 7, 8]) and, to our knowledge, there is no well-known rigorous derivation of how the disturbance is related to Maxwell's equations in matter. By fully understanding this light disturbance, we are able to develop methods that allow us to determine the time-dependent dynamics of the electromagnetic field, while considering only a single scalar potential.

In small-angle scattering one regards electromagnetic beams propagating large distances through homogeneous matter or free space. The scalar Helmholtz equation can then solved numerically using algorithms based on discrete Fourier transforms, such as Fraunhofer and Fresnel diffraction [3]. In piecewise homogeneous media the Helmholtz equation is solved by multislice or finite difference algorithms [9, 10]. However these algorithms are unsuited for large-scale propagation in curved geometries such as curved waveguides, where the

small-angle approximation is violated. By generalizing the small angle approximation into coordinate systems that curves with the geometry, we can circumvent this restriction.

Structure of this work

In **Part I** we present a scalar diffraction theory and show that the electromagnetic vectorial fields can be determined from the scalar potential. We then derive the well-known Fresnel, Fraunhofer, multislice and Finite difference propagation schemes. Using coordinate transforms we extend these propagators to cylindrically symmetric and curved coordinate systems. The algorithms are implemented into *PyPropagate*, a novel Python framework for paraxial beam propagation. We validate the implementation by comparing numerical with analytical results.

In **Part II** we show how the stationary propagation algorithms can be used for time-dependent propagation of the disturbance. We will derive and verify a straightforward algorithm to determine the time-dependent energy density. This is used to analyze time-dependent beam propagation in slab, circular and curved waveguides.

All simulations presented in this work were run by *PyPropagate* which is available online [11]. The script for each simulation, some additional figures and mathematical details to some derivations can be found in the appendix, **Part III** of this work.

Part I

STATIONARY WAVE PROPAGATION

2

PROPAGATION OF ELECTROMAGNETIC WAVES IN MATTER

2.1 Maxwell's equations

The dynamics of electromagnetic waves are described by Maxwell's Equations. They describe the relationship between the electric field $\vec{E}\,[\mathrm{N/C}]$, the magnetic field $\vec{B}\,[\mathrm{N/Am}]$, the charge density $\rho\,[\mathrm{C/m^3}]$ and the current density field $\vec{J}\,[\mathrm{A/m^2}]$. Formulated in SI units they take the form

$$\vec{\nabla}\times\vec{E} = -\frac{\partial\vec{B}}{\partial t} \qquad \vec{\nabla}\times\vec{B} = \mu_0\left(\epsilon_0\frac{\partial\vec{E}}{\partial t} + \vec{J}\right)$$
$$\vec{\nabla}\cdot\vec{E} = \frac{\rho}{\epsilon_0} \qquad \vec{\nabla}\cdot\vec{B} = 0,$$

where $\mu_0 = 4\pi\cdot 10^{-7}\ \mathrm{N/A^2}$ is the free space permeability, $\epsilon_0 = 1/(\mu_0 c^2)$ is the free space permittivity and $c = 299,792,458\ \mathrm{m/s}$ is the speed of light. While these equations are generally applicable, interactions with matter are usually summarized by two auxiliary fields, the displacement field $\vec{D}\,[\mathrm{C/m^2}]$ and the magnetizing field $\vec{H}\,[\mathrm{A/m}]$. Expressed through these auxiliary fields Maxwell's Equations are

$$\vec{\nabla}\times\vec{E} = -\frac{\partial\vec{B}}{\partial t} \qquad \vec{\nabla}\times\vec{H} = \frac{\partial\vec{D}}{\partial t} + \vec{J}_f$$
$$\vec{\nabla}\cdot\vec{D} = \rho_f \qquad \vec{\nabla}\cdot\vec{B} = 0,$$

where ρ_f is the density and $\vec{J}_f$ the current density of free charges in the medium. The nature of the auxiliary fields is generally quite complicated as they include microscopic interactions of the electromagnetic fields with the bound charges of the medium. A first approximation is to express the auxiliary fields through the macroscopic polarization density $\vec{P}\ [\mathrm{C/m^2}]$ and magnetization density $\vec{M}\ [\mathrm{A/m}]$ of the medium

$$\vec{D} = \epsilon_0\vec{E} + \vec{P} \qquad \text{and} \qquad \vec{H} = \frac{1}{\mu_0}\vec{B} - \vec{M}.$$

Media is often characterized based on its response to a monochromatic electromagnetic wave. When the polarization density has the same orientation as the electric field and the magnetization density has the same orientation as the magnetic field, the medium is said to be *isotropic*. In *linear* media the magnetization and polarization densities are directly proportional to the field amplitudes. If the macroscopic properties of the medium are constant within the regarded domain the medium is said to be *homogeneous*.

2.2 The Helmholtz equation

We derive a well known wave equation for the Fourier-transformed electric field. The time-domain Fourier transforms of the fields are denoted by a hat

$$\hat{\vec{E}} = \mathcal{F}_t[\vec{E}](\omega) = \frac{1}{\sqrt{2\pi}} \int_{\mathbb{R}} \vec{E}\, e^{-i\omega t}\, \mathrm{d}t \quad \text{etc.}$$

Note that the Fourier-transformed fields are complex valued, while the fields in time-domain are real valued, implying Hermitian symmetry: $\hat{\vec{E}}(-\omega) = \hat{\vec{E}}(\omega)^*$, where the asterisk denotes the complex conjugation. The inverse transform is given by

$$\begin{aligned}\vec{E} &= \mathcal{F}_\omega^{-1}[\hat{\vec{E}}](t) = \frac{1}{\sqrt{2\pi}} \int_{\mathbb{R}} \hat{\vec{E}}\, e^{i\omega t}\, \mathrm{d}\omega \\ &= \sqrt{\frac{2}{\pi}} \int_0^\infty \Re(\hat{\vec{E}}) \cos(\omega t) - \Im(\hat{\vec{E}}) \sin(\omega t)\, \mathrm{d}\omega \quad \text{etc.}\end{aligned}$$

In a medium without free charges, the Fourier transformed Maxwell's equations are

$$\begin{aligned}\vec{\nabla} \times \hat{\vec{E}} &= i\omega \hat{\vec{B}} & \vec{\nabla} \times \hat{\vec{H}} &= -i\omega \hat{\vec{D}} \\ \vec{\nabla} \cdot \hat{\vec{D}} &= 0 & \vec{\nabla} \cdot \hat{\vec{B}} &= 0,\end{aligned}$$

where the auxiliary fields are expressed through the Fourier-transformed polarization and magnetization fields

$$\hat{\vec{D}} = \epsilon_0 \hat{\vec{E}} + \hat{\vec{P}} \qquad \text{and} \qquad \hat{\vec{H}} = \frac{1}{\mu_0} \hat{\vec{B}} - \hat{\vec{M}}.$$

By applying the curl of curl identity from vector calculus $\vec{\nabla} \times \vec{\nabla} \times \vec{E} = \vec{\nabla}(\vec{\nabla}\vec{E}) - \Delta\vec{E}$, we obtain a vectorial wave equation

$$k^2 \hat{\vec{E}} + \Delta \hat{\vec{E}} = -i\omega\mu_0 \left(\vec{\nabla} \times \hat{\vec{M}} \right) - \omega^2 \mu_0 \hat{\vec{P}} - \frac{1}{\epsilon_0} \vec{\nabla} \left(\vec{\nabla} \hat{\vec{P}} \right), \tag{2.1}$$

with the *wavenumber* $k := \omega/c$. If the field amplitudes are sufficiently low, we can approximate the polarization through its Taylor series around $\hat{\vec{E}} = 0$ up to first order $\hat{\vec{P}} \approx \hat{\vec{P}}|_{\hat{\vec{E}}=0} + \left(\partial_{E_x} \hat{\vec{P}} \quad \partial_{E_y} \hat{\vec{P}} \quad \partial_{E_z} \hat{\vec{P}} \right)\Big|_{\hat{\vec{E}}=0} \hat{\vec{E}}$, neglecting the following nonlinear terms. When the medium contains no permanent dipoles, the constant polarization term vanishes $\hat{\vec{P}}|_{\hat{\vec{E}}=0} = 0$. In isotropic media the Jacobian becomes a scalar matrix and can be expressed as $\left(\partial_{E_x} \hat{\vec{P}} \quad \partial_{E_y} \hat{\vec{P}} \quad \partial_{E_z} \hat{\vec{P}} \right)\Big|_{\hat{\vec{E}}=0} = \epsilon_0 (n^2 - 1)\, \mathbf{I}$, where $\mathbf{I}$ is the identity matrix and n is the complex *refractive index* of the medium[1]. The refractive index defined as $n := v/c - i\kappa$, where v is the average propagation velocity and κ is the absorption coefficient of light in the medium. The polarization may then be approximated as

$$\hat{\vec{P}} \approx \hat{\vec{E}}\, \epsilon_0 (n^2 - 1).$$

1 Obtained by comparing with results from [12, page 55 ff.].

As the divergence-free displacement field is directly proportional to the electric field $\hat{\vec{D}} = \epsilon_0 \hat{\vec{E}} + \hat{\vec{P}} \approx \epsilon_0 n^2 \, \hat{\vec{E}}$, the divergence of the electric field and thus of the polarization field must vanish as well. For non-magnetic media the magnetization also vanishes $\vec{M} = 0$. When all these conditions are met, the wave equation Eq. (2.1) can be reduced into the *Helmholtz equation*

$$\Delta \hat{\vec{E}} + k^2 n^2 \hat{\vec{E}} = 0. \tag{2.2}$$

The Helmholtz equation can be analogously derived for the magnetic field $\vec{B}$. We conclude with the observation that the large-scale electromagnetic dynamics for relatively small field amplitudes in a homogeneous, isotropic, non-magnetic medium without free charges or permanent dipoles are fully described by the Helmholtz equation (2.2) and the Maxwell equations

$$\vec{\nabla} \cdot \hat{\vec{E}} = 0 \qquad \text{and} \qquad \hat{\vec{B}} = \frac{1}{i\omega} \vec{\nabla} \times \hat{\vec{E}}. \tag{2.3, 2.4}$$

2.3 Scalar diffraction theory

> "Notwithstanding this, it is not immediately apparent why one can use a single scalar field to describe the electromagnetic disturbance in free space, the squared modulus of which corresponds to the optical intensity."
>
> *David M. Paganin*, *Coherent X-Ray Optics* [9], page 5.

In optical diffraction problems one usually works a single complex scalar field called the light disturbance, whose squared magnitude corresponds to the optical intensity, as opposed to explicitly solving the vectorial Maxwell system Eq. (2.2), Eq. (2.3) and Eq. (2.4). In many optics textbooks (e.g. [13, 14]) this is justified by the assumption that the components of the electric field are uncoupled and can therefore be regarded independently. However, in almost all cases this is a violation of Eq. (2.3), which requires the divergence of the electric field to vanish. A rigorous derivation by H. S. Green and E. Wolf shows that real-valued divergence-free fields can be fully described by a complex scalar potential and that under certain circumstances the optical intensity corresponds to the squared magnitude of such a potential [4, 5]. However the applicability is limited to free-space (or absorption-free) propagation, as only then the Fourier transformed electromagnetic field vectors can be real-valued, due to the complex nature of the refractive index in the Helmholtz equation. Other authors have used the fact that an arbitrary vector field may be represented by two scalar Debye potentials [6]. This is helpful in multipole expansion problems, however the construction of the electromagnetic fields is quite exhaustive and the direct connection to the optical intensity remains unclear. Other noteworthy publications introducing a scalar theory in optics are [15, 8]. Yet, to our knowledge, there is no well-known rigorous justification of why a single

scalar field, whose squared magnitude corresponds to the optical intensity, can be used for optical diffraction problems in media.

We attempt to bridge this gap through a straightforward definition of a scalar potential and a rigorous derivation of the optical intensity.

2.3.1 Scalar potential

A divergence-free solution $\hat{\vec{E}}$ of the vectorial Helmholtz equation (2.2) can be constructed from any not necessarily divergence-free solution $\vec{\psi}$ [V/m] by taking the curl of $\vec{\psi}$

$$\hat{\vec{E}} = \frac{i}{k} \vec{\nabla} \times \vec{\psi}. \tag{2.5}$$

This directly follows from the vector calculus identities

$$\begin{aligned} \vec{\nabla} \cdot \hat{\vec{E}} &= \frac{i}{k} \vec{\nabla} \cdot (\vec{\nabla} \times \vec{\psi}) = 0, \\ \Delta \hat{\vec{E}} &= \frac{i}{k} \Delta(\vec{\nabla} \times \vec{\psi}) = \frac{i}{k} \vec{\nabla} \times (\Delta \vec{\psi}) = -n^2 k^2 \hat{\vec{E}}. \end{aligned}$$

The prefactor $\frac{i}{k}$ is chosen to simplify upcoming calculations. For any vector $\vec{e}$, the projection $(\vec{\psi} \cdot \vec{e})\, \vec{e}$ is also a solution of the Helmholtz equation. In particular, given a solution ψ of the scalar Helmholtz equation

$$\Delta \psi + k^2 n^2 \psi = 0, \tag{2.6}$$

we can construct solutions of the Maxwell system Eq. (2.2) to Eq. (2.4) by setting $\vec{\psi} = \psi\, \vec{e}_p$ for any unit vector $\vec{e}_p$ and determining[2]

$$\hat{\vec{E}} = \frac{i}{k} (\vec{\nabla}\psi) \times \vec{e}_p \qquad \text{and} \qquad \hat{\vec{B}} = \frac{1}{c} \left(\frac{1}{k^2} \vec{\nabla}(\vec{e}_p \cdot \vec{\nabla}\psi) + n^2 \psi\, \vec{e}_p \right). \tag{2.7}$$

We will call ψ the *scalar potential* of the electromagnetic field. General solutions can be constructed from linear combinations of three independent scalar potentials ψ_x, ψ_y, ψ_z by setting $\vec{\psi} = \psi_x \vec{e}_x + \psi_y \vec{e}_y + \psi_z \vec{e}_z$. The electromagnetic fields are then given by the sum of the fields determined individually from the potentials.

2.3.2 Plane wave

A trivial solution of the scalar Helmholtz equation Eq. (2.6) in free space (where $n = 1$) is $\psi = \psi_0 \exp(-i\vec{k} \cdot \vec{x})$ with the initial amplitude ψ_0 [V/m] and the wave vector $\vec{k} = k\, \vec{e}_k$ for any unit vector $\vec{e}_k$. The gradient of the solution is $\vec{\nabla}\psi = -i\psi\, \vec{k}$. To construct a solution of the Maxwell system we choose an arbitrary unit vector $\vec{e}_p$ and apply Eq. (2.7), resulting in the fields $\hat{\vec{E}} = \psi\, (\vec{e}_p \times \vec{e}_k)$ and $\hat{\vec{B}} = \frac{\psi}{c} (\vec{e}_p - \vec{e}_k(\vec{e}_p \cdot \vec{e}_k))$. We see that $\hat{\vec{E}}$, $\hat{\vec{B}}$ and $\vec{k}$ are perpendicular to each other and form a right-handed coordinate system. Without loss of generality we choose $\vec{e}_k = \vec{e}_z$ and $\vec{e}_p = \vec{e}_y$, so that $\hat{\vec{E}} = \psi\, \vec{e}_x$ and $\hat{\vec{B}} = \frac{\psi}{c}\, \vec{e}_y$. Note that physical solutions are band-limited and contain an additional frequency-dependent factor $\rho(\omega)$ with Hermitian symmetry. For a monochromatic plane wave with frequency

2 see Lemma 12.1 for an explicit derivation.

ω_0 this factor could be $\rho(\omega) = \sqrt{\pi/2}\,(\delta(\omega+\omega_0)+\delta(\omega-\omega_0))$, resulting in the fields $\vec{E} = \psi_0 \cos(k_0 x - \omega_0 t)\,\vec{e}_x$ and $\vec{B} = \frac{\psi_0}{c}\cos(k_0 x - \omega_0 t)\,\vec{e}_y$, where $k_0 = \omega_0/c$.

2.3.3 Paraxial beam

For large wave numbers k, solutions of the scalar Helmholtz equation (2.6) can often be locally approximated by a slowly varying envelope u multiplied with a fast oscillating term $\psi = u\,\exp(-in\vec{k}\cdot\vec{x})$. The magnitude of the gradient of u can be neglected compared to $|nk|$, so that we can approximate $\vec{\nabla}\psi \approx -in\psi\,\vec{k}$. Choosing $\vec{e}_p = \vec{e}_y$ and $\vec{e}_k = \vec{e}_z$ as before, we obtain the approximate solution $\hat{\vec{E}} \approx n\psi\,\vec{e}_x$ and $\hat{\vec{B}} \approx \frac{n^2}{c}\psi\,\vec{e}_y$.

2.4 Energy transfer

In many optical problems one is not interested in the explicit direction of the electric and magnetic fields, but in the average energy flux density, as it is an easily measurable observable of the electromagnetic field.

2.4.1 Poynting vector

The energy flux transported by the electromagnetic field is given by the Poynting vector [12]

$$\vec{S} = \vec{E} \times \vec{H}\ [\mathrm{W/m^2}]. \tag{2.8}$$

The integral of the Poynting vector over some detector surface and a certain exposure time corresponds to the energy transported to the detector. This energy is observable, for example by our eyes or CCD chips.

2.4.2 Intensity

Usually exposure times are very large compared to the fields' oscillations so that for periodic paraxial signals the transported energy is approximately proportional to the time-averaged magnitude of the Poynting vector. This is called the local wave intensity

$$I = \left\langle \|\vec{S}\| \right\rangle = \left\langle \|\vec{E} \times \vec{H}\| \right\rangle, \tag{2.9}$$

where the angle brackets denote the time-average $\langle f(t)\rangle = \lim_{T\to\infty}\frac{1}{T}\int_0^T f(t)\mathrm{d}t$. For paraxial beams as defined in section 2.3.3, the magnetic and electric field vectors are nearly perpendicular ($\hat{\vec{E}} \approx n\psi\,\vec{e}_x$ and $\hat{\vec{B}} \approx \frac{n^2}{c}\psi\,\vec{e}_y$). As we assume the signal to be periodic, it contains only discrete frequencies ω_i for $i \in \mathbb{N}$ and the inverse Fourier

transforms can be written as a sum over the frequencies. With this in mind we rewrite the electric field as

$$\begin{aligned}\vec{E} &= \mathcal{F}_\omega^{-1}[\hat{\vec{E}}](t) \approx \vec{e}_x\, \mathcal{F}_\omega^{-1}[n\psi](t)\\ &= \vec{e}_x \sqrt{\frac{2}{\pi}} \sum_{i\in\mathbb{N}} \Re(n(\omega_i)\,\psi(\omega_i)) \cos(\omega_i t) - \Im(n(\omega_i)\,\psi(\omega_i)) \sin(\omega_i t)\\ &\approx \vec{e}_x \sqrt{\frac{2}{\pi}} \sum_{i\in\mathbb{N}} \Re(n(\omega_i))\,(\Re(\psi(\omega_i)) \cos(\omega_i t) - \Im(\psi(\omega_i)) \sin(\omega_i t)),\end{aligned}$$

where $k_i = \omega_i/c$ and we use the fact that usually $\Re(n) \gg \Im(n)$. The same can be done for the magnetic field resulting in

$$\vec{B} \approx \vec{e}_y \frac{1}{c}\sqrt{\frac{2}{\pi}} \sum_{i\in\mathbb{N}} \Re(n(\omega_i))^2\,(\Re(\psi(\omega_i)) \cos(\omega_i t) - \Im(\psi(\omega_i)) \sin(\omega_i t)).$$

Inserting these fields into the definition of the intensity Eq. (2.9) results in the double sum

$$\begin{aligned}I \approx \frac{c\,\epsilon_0}{\pi} \sum_{i,j\in\mathbb{N}} \Re(n(\omega_i))\Re(n(\omega_j))^2\, \langle &(\Re(\psi(\omega_i)) \cos(\omega_i t) - \Im(\psi(\omega_i)) \sin(\omega_i t))\\ &(\Re(\psi(\omega_j)) \cos(\omega_j t) - \Im(\psi(\omega_j)) \sin(\omega_j t))\rangle\,.\end{aligned}$$

When taking the time average, the trigonometric products are reduced to $\langle \cos(\omega_i t)\, \sin(\omega_j t)\rangle = 0$ and $\langle \cos(\omega_i t) \cos(\omega_j t)\rangle = \delta_{i,j}/2$, where δ_{ij} is the Kronecker delta symbol[3]. The double sum therefore is equivalent to the single sum

$$I \approx \frac{c\,\epsilon_0}{\pi} \sum_{i\in\mathbb{N}} \Re(n(\omega_i))^3\, |\psi(\omega_i)|^2, \tag{2.10}$$

which is the local wave intensity for periodic paraxial fields.

2.4.3 Time-averaged Poynting vector

While the intensity is a easily measurable observable, it is also desirable to know the average direction of energy flux, which is determined by taking the time-average of the Poynting vector. When the signal is time-periodic, the fields are determined by the sum over discrete frequencies ω_i for $i \in \mathbb{N}$

$$\begin{aligned}\vec{E} &= \mathcal{F}_\omega^{-1}[\hat{\vec{E}}](t)\\ &= \sqrt{\frac{2}{\pi}} \sum_{i\in\mathbb{N}} \Re(\hat{\vec{E}}(\omega_i)) \cos(\omega_i) + \Im(\hat{\vec{E}}(\omega_i)) \sin(\omega_i) \text{ etc.}\end{aligned}$$

3 see Lemma 12.2 for proof

In the time averaged Poynting vector, we again find a double sum that can be reduced to a single sum by applying Lemma 12.2, resulting in

$$\left\langle \vec{S} \right\rangle = \frac{1}{\mu_0 \pi} \sum_{i \in \mathbb{N}} \Re(\hat{\vec{E}}(\omega_i)) \times \Re(\hat{\vec{B}}(\omega_i)) + \Im(\hat{\vec{E}}(\omega_i)) \times \Im(\hat{\vec{B}}(\omega_i)).$$

When the fields are constructed from a single scalar potential ψ that oscillates slowly in $\vec{e}_p$ direction and therefore fulfills the condition

$$\|\vec{\nabla}(\vec{e}_p \cdot \vec{\nabla}\psi)\| \ll |n^2 k^2 \psi|, \tag{2.11}$$

we can approximate

$$\begin{aligned}
\Re(\hat{\vec{E}}) \times \Re(\hat{\vec{B}}) = &\ \Re\Big(\frac{i}{k}\vec{\nabla}\psi \times \vec{e}_p\Big) \times \Re\Big(\frac{1}{c}\left(\frac{1}{k^2}\,\vec{\nabla}(\vec{e}_p \cdot \vec{\nabla}\psi) + n^2 \psi\, \vec{e}_p\right)\Big) \\
= &-\frac{1}{ck}\,\Im(\vec{\nabla}\psi \times \vec{e}_p) \times \Re\Big(\frac{1}{k^2}\,\vec{\nabla}(\vec{e}_p \cdot \vec{\nabla}\psi) + n^2 \psi\, \vec{e}_p\Big) \\
\approx &-\frac{1}{\omega}\,\Im(\vec{\nabla}\psi \times \vec{e}_p) \times \Re\Big(n^2 \psi\, \vec{e}_p\Big) \\
\approx &-\frac{\Re(n)^2}{\omega}\,\Re(\psi)\,\Im((\vec{\nabla}\psi \times \vec{e}_p) \times \vec{e}_p) \\
= &\ \frac{\Re(n)^2}{\omega}\,\Re(\psi)\,\Im(\vec{\nabla}\psi)
\end{aligned}$$

and, analogously

$$\Im(\hat{\vec{E}}) \times \Im(\hat{\vec{B}}) \approx -\frac{\Re(n)^2}{\omega}\,\Im(\psi)\,\Re(\vec{\nabla}\psi).$$

The condition Eq. (2.11) is valid for two dimensional beams that are constant in $\vec{e}_p$ direction and for paraxial beams when $\vec{e}_p$ is chosen perpendicular to the direction of propagation. A straightforward calculation[4] shows that

$$\Re(\psi)\,\Im(\vec{\nabla}\psi) - \Im(\psi)\,\Re(\vec{\nabla}\psi) = |\psi|^2\,\vec{\nabla}\arg(\psi).$$

This allows us to approximate the time-averaged Poynting vector for periodic signals with negligible derivative in $\vec{e}_p$ direction as

$$\left\langle \vec{S} \right\rangle \approx \frac{c\epsilon_0}{\pi} \sum_{i \in \mathbb{N}} \Re(n(\omega_i))^2\, |\psi(\omega_i)|^2\, \frac{\vec{\nabla}\arg(\psi(\omega_i))}{k_i}. \tag{2.12}$$

4 see Lemma 12.3 for details

When the signal is monochromatic with wave number k_0, the time-averaged Poynting vector can therfore be expressed through the intensity and the gradient of the phases of the scalar potential

$$\begin{aligned}\left\langle \vec{S} \right\rangle &\approx \frac{1}{\Re(n)k_0}\, I\, \vec{\nabla} \arg(\psi) \\ &\approx I\, \frac{\vec{\nabla} \arg(\psi)}{\|\vec{\nabla} \arg(\psi)\|},\end{aligned}$$

where the latter approximation holds for paraxial beams, where $\|\vec{\nabla} \arg(\psi)\| \approx \Re(n)k_0$. We conclude that the magnitude of the scalar potential determines the intensity, while the gradient of the phase determines the orientation of the time-averaged Poynting vector.

2.5 Propagation equations

In many optical problems one is interested in the forward propagation of electromagnetic waves through a medium. When backward propagating waves can be neglected, the Helmholtz equation may be reduced to a first order differential equation.

The derivations of this section are based on derivations of nonlinear propagation equations from the dissertation of Anton Husakou [16]. For now, we restrict our discussion to homogeneous media.

2.5.1 Forward Helmholtz equation

The Helmholtz equation may be separated into two differential equations, one describing forward and one describing backward propagation. We choose our coordinate system so that z is the forward propagation direction. By taking a two-dimensional Fourier transform with respect to the perpendicular space directions $\vec{x}_\perp := (x \;\; y)^T$, the scalar Helmholtz equation (2.6) is transformed to a second-order ordinary differential equation

$$\frac{\partial^2 \tilde{\psi}}{\partial z^2} + (k^2 n^2 - k_\perp^2)\, \tilde{\psi} = \left(\frac{\partial^2}{\partial z^2} + \beta^2 \right) \tilde{\psi} = 0, \tag{2.13}$$

where $\tilde{\psi} = \mathcal{F}_{\vec{x}_\perp}[\psi](\vec{k}_\perp)$ and $\beta := \sqrt{k^2 n^2 - k_\perp^2}$. The differential operator may be written as the product of two operators

$$\left(\frac{\partial}{\partial z} + i\beta \right) \left(\frac{\partial}{\partial z} - i\beta \right) \tilde{\psi} = \left(\frac{\partial}{\partial z} - i\beta \right) \left(\frac{\partial}{\partial z} + i\beta \right) \tilde{\psi} = 0,$$

as a straightforward calculation will verify. The differential equation is therefore fulfilled if $\tilde{\psi}$ solves either of the differential equations

$$\frac{\partial \tilde{\psi}}{\partial z} = i\beta\, \tilde{\psi} \qquad\qquad \frac{\partial \tilde{\psi}}{\partial z} = -i\beta\, \tilde{\psi} \tag{2.14a, 2.14b}$$

and the general solution can be written as a sum of both solutions. By comparing with the plane wave solutions (see Sec. 2.3) it is evident that equation Eq. (2.14a) describes a beam with backward moving wavefronts while equation Eq. (2.14b) describes a beam

with forward moving wavefronts with regard to the z axis. We therefore call them the backward and forward Helmholtz equations.

2.5.2 Paraxial Helmholtz equation

For beams traveling paraxial to the z axis, the support of $\tilde{\psi}$ is contained in a small region where $k_\perp^2 \ll k^2$. We can therefore approximate the square root in β through its first-order Taylor series around $k_\perp^2 = 0$

$$\beta = \sqrt{k^2 n^2 - k_\perp^2} \approx kn - \frac{k_\perp^2}{2kn}.$$

Substituting this into the forward Helmholtz equation Eq. (2.14b) results in the paraxial Helmholtz equation

$$\frac{\partial \tilde{\psi}}{\partial z} = i\left(\frac{k_\perp^2}{2kn} - kn\right)\tilde{\psi}. \tag{2.15}$$

When transformed back into real space, it takes the form

$$\frac{\partial \psi}{\partial z} = \frac{1}{2ikn}\,\Delta_\perp \psi - ikn\,\psi. \tag{2.16}$$

The paraxial Helmholtz equation is valid for forward propagating paraxial beams in homogeneous media[5].

2.5.3 Cylindrically symmetric propagation equations

For systems with cylindrical symmetry the wave equations can be reduced by one dimension. By transforming the Helmholtz equation (2.6) into cylindrical coordinates $(x, y, z) \to (r, \theta, z)$ and disregarding the θ-dependency of ψ, we find

$$\frac{\partial^2 \psi}{\partial z^2} + \left(\frac{\partial^2 \psi}{\partial r^2} + \frac{1}{r}\frac{\partial \psi}{\partial r}\right) + k^2 n^2 \psi = 0, \tag{2.17}$$

or, analogously for the paraxial case with Eq. (2.16)

$$\frac{\partial \psi}{\partial z} = \frac{1}{2ikn}\left(\frac{\partial^2 \psi}{\partial r^2} + \frac{1}{r}\frac{\partial \psi}{\partial r}\right) - ikn\,\psi, \tag{2.18}$$

which is the cylindrically symmetric paraxial Helmholtz equation.

2.6 Propagation kernels

Propagation kernels can be interpreted as the response of the propagation equations to an infinitesimal small impulse. They can be used to solve the propagation equations through a two dimensional convolution, given an initial field distribution in the plane perpendicular to the direction of propagation.

5 In literature one usually works with the parabolic wave equation, which is obtained by applying a slowly varying envelope approximation to the Helmholtz equation. See Chap. 11.1 for a brief discussion.

2.6.1 Forward propagation

Given a boundary condition in the plane perpendicular to the direction of propagation the forward Helmholtz equation (2.14b) is solved by a convolution. We assume the field distribution is known at $z = 0$ in the $\vec{x}_\perp$ plane. The forward Helmholtz equation then has the unique solution $\tilde{\psi} = \tilde{\psi}(z = 0)\exp(-i\beta z)$, where $\tilde{\psi}(z = 0)$ is the Fourier transformed boundary condition with respect ot the perpendicular directions $\tilde{\psi}(z = 0) = \mathcal{F}_{\vec{x}_\perp}[\psi(z = 0)]\,(\vec{k}_\perp)$. To determine ψ in real space, we perform the inverse Fourier transform

$$\psi = \mathcal{F}^{-1}_{\vec{k}_\perp}\left[\mathcal{F}_{\vec{x}_\perp}[\psi(z=0)](\vec{k}_\perp)\cdot e^{-iz\sqrt{k^2n^2-k_\perp^2}}\right](\vec{x}_\perp). \tag{2.19}$$

As the solution involves a multiplication in Fourier space, we can write it as a two dimensional convolution over $x_\perp$ for a convolution kernel $\mathcal{T}$

$$\psi = \psi(\vec{x}_\perp, z = 0) * \mathcal{T}(\vec{x}_\perp, z),$$

where the convolution kernel $\mathcal{T}$ is determined from the inverse Fourier transform of $\exp(-iz\sqrt{k^2n^2 - k_\perp^2})$ and we assume its convergence. In [17, pp. 311] it is shown that under certain conditions a convolution kernel for the Helmholtz equation in free space is

$$\mathcal{T}'(\vec{x}_\perp, z) = \frac{z}{2\pi}\frac{\exp(ikr)}{r^2}\left(ik - \frac{1}{r}\right), \tag{2.20}$$

where $r = \sqrt{|\vec{x}_\perp|^2 + z^2}$. Without proof and further implications we will assume that $\mathcal{T}$ is given by substituting $r \to -\operatorname{sign}(z)\, r$ in $\mathcal{T}'$, which keeps the direction of propagation pointing forward in z. A plot of this kernel is shown in Fig. 2.1 (above).

2.6.2 Paraxial propagation

When the beam is paraxial, we solve the paraxial Helmholtz equation Eq. (2.15) with the boundary condition $\tilde{\psi}(z = 0)$ and perform an inverse Fourier transform to find the solution

$$\psi = \mathcal{F}^{-1}_{\vec{k}_\perp}\left[\mathcal{F}_{\vec{x}_\perp}[\psi(z=0)](\vec{k}_\perp)\cdot\exp\left(iz\left(\frac{k_\perp^2}{2kn} - kn\right)\right)\right](\vec{x}_\perp). \tag{2.21}$$

Again we can write this as the 2D convolution over $\vec{x}_\perp$ with a convolution kernel $\mathcal{P}$

$$\psi = \psi(\vec{x}_\perp, z = 0) * \mathcal{P}(\vec{x}_\perp, z). \tag{2.22}$$

We see from Cor. 12.1 that the convolution kernel is given by

$$\mathcal{P}(\vec{x}_\perp, z) = \frac{ikn}{2\pi z}\exp\left(-ikn\left(z + \frac{|\vec{x}_\perp|^2}{2z}\right)\right). \tag{2.23}$$

Both kernels are shown side-by-side in Fig. 2.1. We note that at large angles from the origin the paraxial approximation created a 'phase dipole' which is not present in

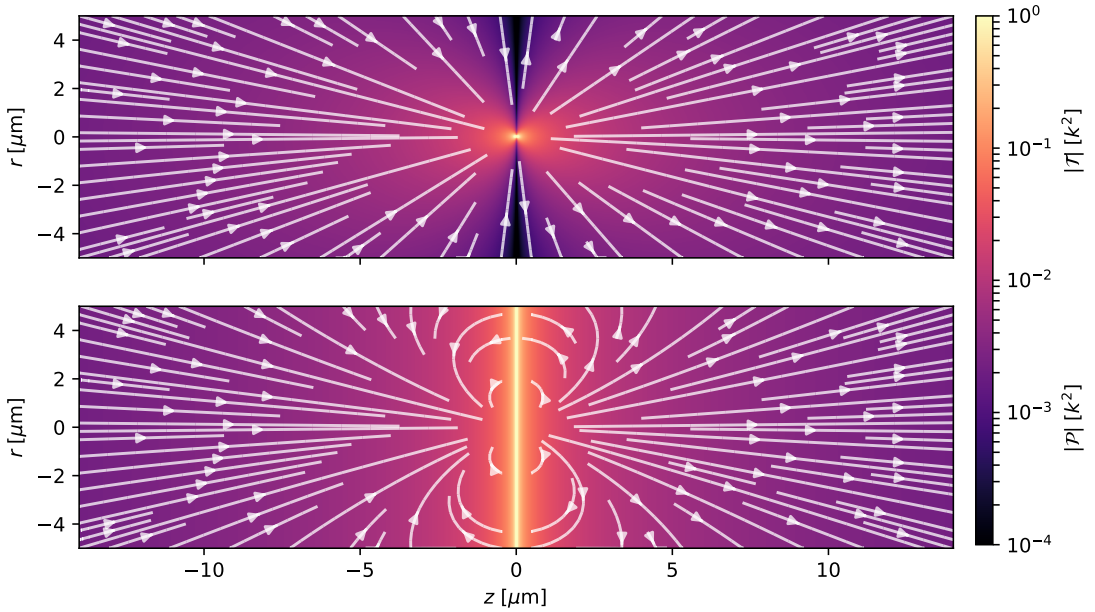

Figure 2.1: Plot of the convolution kernels $\mathcal{T}$ (above) and $\mathcal{P}$ (below) in free space at $E = 1$ eV photon energy ($k = E/\hbar c$). The magnitude is color coded and the direction of the gradient of the phase is indicated with white streamlines. While the intensities and Poynting directions differ significantly at large angles to the z axis, they are in good agreement at small angles. The streamlines were numerically determined using the techniques described in Sec. 4.5. The script to create this figure is Code 14.1.

the exact solution. However, for small propagation angles both solutions are in good agreement.

2.6.3 Frauenhofer propagation

In free space the paraxial propagation convolution reduces to an inverse Fourier transformation if the initial beam has a finite support $\Omega \subset \mathbb{R}^2$ and the propagation distance is sufficiently large. To show this, we evaluate the paraxial convolution Eq. (2.22) for $n = 1$

$$\begin{aligned}\psi &= \frac{ik}{2\pi z}\,\exp(-ikz)\int_\Omega \psi_0(\tilde{\vec{x}}_\perp)\,\exp\!\left(-\frac{ik}{2z}\left|\vec{x}_\perp - \tilde{\vec{x}}_\perp\right|^2\right)\mathrm{d}\tilde{\vec{x}}_\perp \\ &= \frac{ik}{2\pi z}\,\exp\!\left(-ik\left(z+\frac{|\vec{x}_\perp|^2}{2z}\right)\right)\int_\Omega \psi_0(\tilde{\vec{x}}_\perp)\,\exp\!\left(-\frac{ik}{2z}\left(\left|\tilde{\vec{x}}_\perp\right|^2 - 2\vec{x}_\perp\cdot\tilde{\vec{x}}_\perp\right)\right)\mathrm{d}\tilde{\vec{x}}_\perp,\end{aligned}$$

where $\psi_0(\tilde{\vec{x}}_\perp) := \psi(z=0)$. For sufficiently large propagation distances z we may approximate $\exp(-\frac{ik}{2z}|\tilde{\vec{x}}_\perp|^2) \approx 1$ over the full support. The propagation then reduces to

$$\begin{aligned}\psi &\approx \frac{ik}{2\pi z}\,\exp\!\left(-ik\left(z+\frac{|\vec{x}_\perp|^2}{2z}\right)\right)\int_\Omega \psi_0(\tilde{\vec{x}}_\perp)\,\exp\!\left(\frac{ik}{z}\,\vec{x}_\perp\cdot\tilde{\vec{x}}_\perp\right)\mathrm{d}\tilde{\vec{x}}_\perp \\ &= \frac{ik}{z}\,\exp\!\left(-ik\left(z+\frac{|\vec{x}_\perp|^2}{2z}\right)\right)\mathcal{F}^{-1}_{\vec{x}_\perp}[\psi_0(\vec{x}_\perp)]\left(\frac{k\,\vec{x}_\perp}{z}\right).\end{aligned} \tag{2.24}$$

This is essentially the inverse Fourier transform of the initial field times a constant and phase factor.

2.7 Boundary conditions

The propagation equations may be generalized to piecewise homogeneous media by imposing proper boundary conditions at the boundaries. Without going into any details we will assume the scalar potential to be continuously differentiable everywhere.

3

ELEMENTARY SOLUTIONS

3.1 Gaussian Beam

Gaussian beams are solutions of the paraxial wave equation where the intensity profile along the axis perpendicular to the direction of propagation follows a Gaussian distribution. They are a good first approximation for many optical beam profiles. We define the beam with flat phases and a symmetrical intensity distribution on the $z = 0$ plane

$$\psi(\vec{x}_\perp, z = 0) = \psi_0 \, \exp\left(-\frac{r^2}{2\sigma_r^2}\right),$$

where $r = \sqrt{x^2 + y^2}$, σ_r is the waist size at the focus and ψ_0 is the beam's amplitude in the focus. From Cor. 12.2 we see that the solution of the paraxial Helmholtz equation (2.16) for this boundary problem is

$$\psi = \psi_0 \, \frac{\sigma_r^2}{w_r(z)^2} \, \exp\left(-iknz - \frac{r^2}{2\, w_r(z)^2}\right), \tag{3.1}$$

where $w_r = \sqrt{\frac{z}{ikn} + \sigma_r^2}$. Using Cor. 12.2 we can also derive asymmetrical or two dimensional Gaussian beams. A symmetrical Gaussian beam is shown in Fig. 3.1.

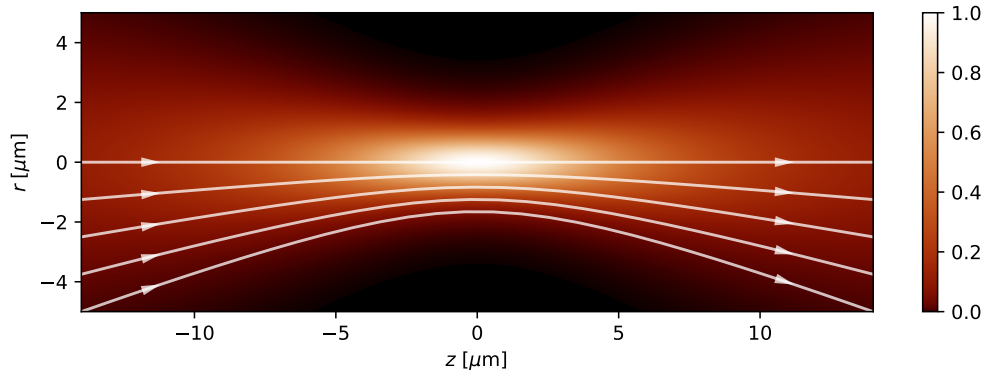

Figure 3.1: Plot of a monochromatic Gaussian beam in free space with 1 eV photon energy and a waist size of $\sigma_r = 1$ μm as described by Eq. (3.1). The field intensity is normalized and color coded while the direction of the time-averaged Poynting vector is indicated by white streamlines. The script to produce this figure is Code 14.2.

3.2 Waveguides

Waveguides are optical channels consisting of two or more propagation media. They are used for many applications such as signal transfer or mode selection. Here, we perform a short derivation of symmetrical waveguide modes. For a more thorough discussion about the theory of waveguides see the vast literature available on the topic (for example [18]).

3.2.1 Symmetric Slab Waveguide

The slab waveguide consists of a guiding layer with complex refractive index n_1 and thickness d sandwiched between two semi-infinite cladding regions of complex refractive index n_2 where $\Re(n_2) < \Re(n_1)$. We choose the orientation of the waveguide so the refractive index is a function of x

$$n = \begin{cases} n_1 & \text{if } |x| \leq d/2 \\ n_2 & \text{otherwise.} \end{cases}$$

We call the n_1 the refractive index of the core and n_2 the refractive index of the cladding of the waveguide.

We find the analytical solution for ψ inside the waveguide by initially neglecting the imaginary part of the refractive index (thus ignoring absorption) and using the ansatz $\psi(x,z) = \psi_0\,\psi_m(x)\exp(-i\beta_m z)$, where ψ_0 is the beam's initial amplitude, $\psi_m(x) \in \mathbb{R}$ is called the guided mode m and $\beta_m \in \mathbb{R}$ is called the propagation constant of the guided mode m. Inserting this into the scalar Helmholtz equation Eq. (2.6) gives a one-dimensional differential equation for ψ_m

$$\frac{\partial^2 \psi_m}{\partial x^2} = (\beta^2 - n^2 k^2)\,\psi_m.$$

We demand the solutions to be symmetrical and to vanish as $x \to \infty$. A class of solutions satisfying these criteria is

$$\psi_m = \begin{cases} \cos(\kappa_m x) & \text{if } |x| \leq d/2 \\ a_m \exp(-\gamma_m |x|) & \text{otherwise} \end{cases}$$

where $\kappa_m = \sqrt{n_1^2 k^2 - \beta_m^2}$, $\gamma_m = \sqrt{\beta_m^2 - n_2^2 k^2}$ and $a_m = \cos(\kappa_m d/2)\exp(\gamma_m d/2)$ are positive real numbers. We call these solutions the symmetrical guided modes of the waveguide. Since we require the solutions to be continuously differentiable we find the additional constraint

$$\kappa \tan(\kappa_m d/2) = \gamma_m = \sqrt{k^2(n_1^2 - n_2^2) - \kappa_m^2}\,,$$

which we call the characteristic equation of the waveguide. The propagation constants β_m are determined by numerically solving this equation.

We now show that the characteristic equation has a finite number N of solutions for κ_m. The left side is strictly monotonously increasing almost everywhere and jumps from

∞ to $-\infty$ with period in $2\pi/d$ while the right side is strictly monotonously decreasing. Therefore there is exactly one intersection in every period of the left argument. Since γ_m is real, κ_m has the upper bound $k\sqrt{(n_1^2 - n_2^2)}$, which also bounds N

$$\left\lfloor \frac{d\,k\sqrt{(n_1^2 - n_2^2)}}{2\pi} \right\rfloor \leq N \leq \left\lceil \frac{d\,k\sqrt{(n_1^2 - n_2^2)}}{2\pi} \right\rceil .$$

We reintroduce absorption through the effective linear attenuation coefficient μ_m [10]

$$\mu_m = \frac{1}{\|\psi_m\|_2^2} \int_{\mathbb{R}} \mu(x)\, |\psi_m(x)|^2 \,\mathrm{d}x,$$

where $\|\psi_m\|_2$ is the L^2 norm of ψ_m. Far away from the entrance the field in the waveguide consists only of the guided modes

$$\psi(x, z) = \sum_{m=1}^{N} c_m\, \psi_m(x)\, \exp(-(i\beta_m + \mu_m)\, z),$$

where coefficients c_m are determined by projecting the incident symmetrical field $\psi(z = 0)$ onto the guided modes

$$c_m = \frac{1}{\|\psi_m\|_2^2} \int_{\mathbb{R}} \psi_m(x) \cdot \psi(z = 0)\, \mathrm{d}x,$$

where ψ_0 is the field strength at $z = 0$. We show example field intensities for two slab waveguides in Fig. 3.2.

3.2.2 Cylindrical Waveguide

The cylindrical waveguide is a waveguide consisting of a circular guiding layer with complex refractive index n_1 and radius R embedded in a material with complex refractive index n_2 where $\Re(n_2) < \Re(n_1)$. If the waveguide is oriented in z direction, the refractive index of the waveguide is described by

$$n = \begin{cases} n_1 & \text{if } r \leq R \\ n_2 & \text{otherwise,} \end{cases}$$

where $r = \sqrt{x^2 + y^2}$.

As with the slab waveguide, we find the analytical solution by initially ignoring absorption and making the ansatz $\psi(x, z) = \psi_0\, \psi_m(r) \exp(-i\beta_m z)$, where ψ_0 is the beam's initial amplitude and $\beta_m \in \mathbb{R}$ is called the propagation constant of mode m. Inserting this into the radially symmetric Helmholtz equation Eq. (2.17) gives us a one-dimensional differential equation for ψ_m

$$\frac{\partial^2 \psi_m}{\partial r^2} + \frac{1}{r}\, \frac{\partial \psi_m}{\partial r} + (n^2 k^2 - \beta^2)\, \psi_m = 0.$$

Solutions of this are the Bessel functions of zeroth order [19]. The symmetrical guided modes are [10]

$$\psi_m = \begin{cases} J_0(\kappa_m r) & \text{if } r \leq R \\ a_m K_0(\gamma_m r) & \text{otherwise} \end{cases}$$

where $\kappa_m = \sqrt{n_1^2 k^2 - \beta_m^2}$, $\gamma_m = \sqrt{\beta_m^2 - n_2^2 k^2}$ and $a_m = J_0(\kappa_m R)/K_0(\gamma_m R)$ are positive real numbers. The characteristic equation follows from the requirement that the solution must be continuously differentiable

$$\kappa_m J_1(\kappa_m R) K_0(\gamma_m R) = \gamma_m K_1(\gamma_m R) J_0(\kappa_m R).$$

This equation can be solved numerically and determines the mode parameters. As with the slab waveguide, we determine the effective linear attenuation coefficients and calculate the full field as the sum of the guided modes. We show example field intensities for two cylindrical waveguides in Fig. 3.3.

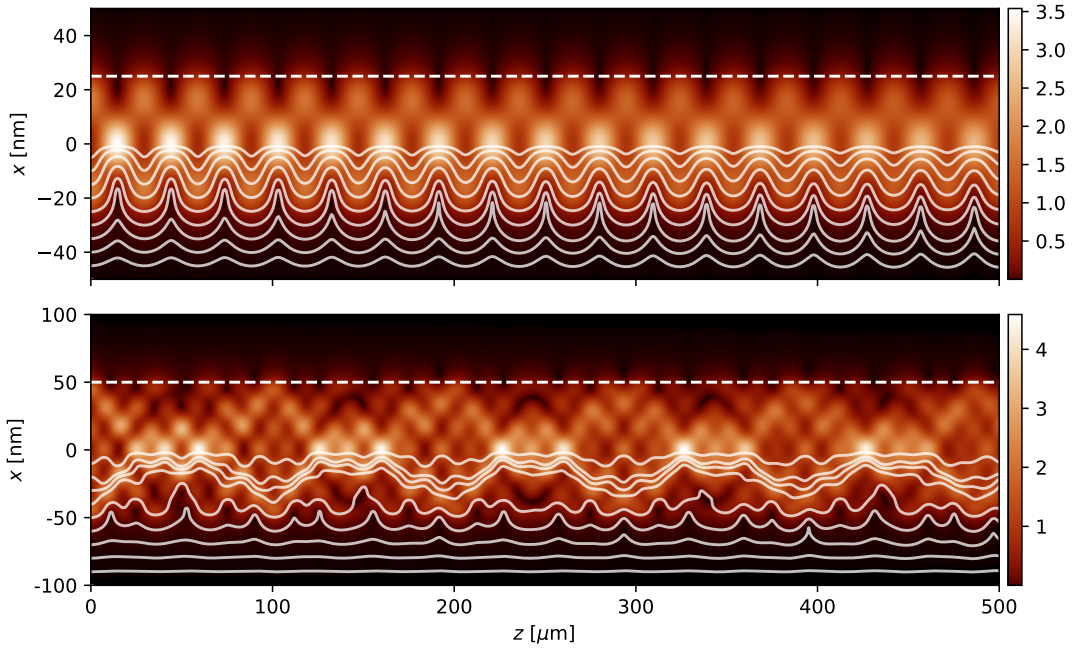

Figure 3.2: Analytical solution for the field inside slab waveguides with 25 nm (top) and 50 nm (bottom) diameter. The core is free space and cladding is composed of Germanium at room temperature. The initial field is a plane wave at 10 keV photon energy. The intensity is normalized and color coded. In the upper image half the contour of the waveguides is indicated by a white dashed line. In the lower half the direction of the time-averaged Poynting vector is indicated by white streamlines. They were determined numerically using the method mentioned in Sec. 4.5. The script to produce this figure is Code 14.4.

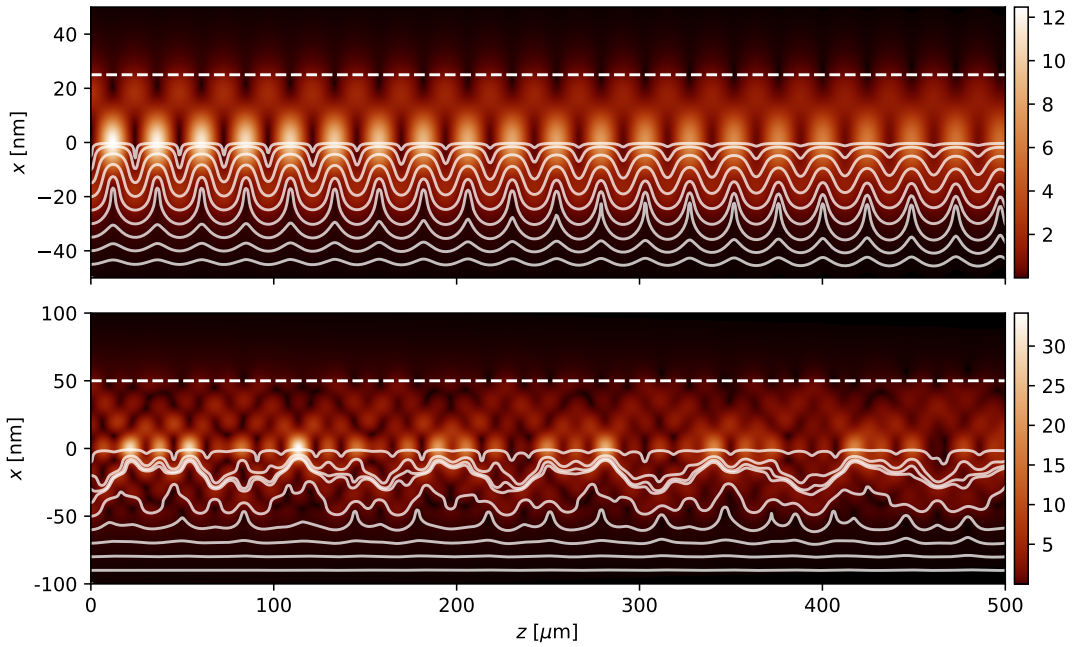

Figure 3.3: Analytical solution for the field inside cylindrical waveguides with 25 nm (top) and 50 nm (bottom) diameter. The core is free space and cladding is composed of Germanium at room temperature. This initial field is a plane wave at 10 keV photon energy. The script to produce this figure is Code 14.4.

4

NUMERICAL PROPAGATION

For many optical problems it is cumbersome or impossible to find exact analytical descriptions and it becomes necessary to apply numerical methods. In this chapter we present numerical methods for beam propagation, allowing the numerical propagation of arbitrary, but discretized scalar fields according to the paraxial scalar Helmholtz equation (2.16).

4.1 Discretization

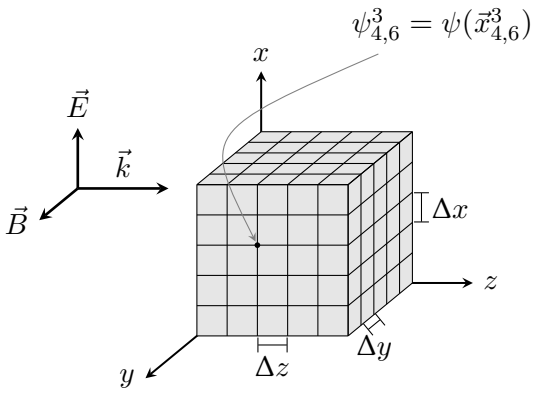

Figure 4.1: Visualization of the discretization of the scalar electromagnetic potential ψ. Each axis is split into $N_x = N_y = N_z = 6$ equidistant values. The approximate direction of the paraxial field vectors is indicated at the upper left.

We discretize the coordinate system into a regular grid with fixed node distances. Discretized field points are denoted by their grid indices, where the index of the z-axis is denoted by a superscript, while other indices are denoted by subscripts. The nodes of a three dimensional grid originating at $\vec{p} \in \mathbb{R}^3$ are $\vec{x}^k_{i,j} = \vec{p} + i\Delta x\, \vec{e_x} + j\Delta y\, \vec{e_y} + k\Delta z\, \vec{e_z}$, for the index $i \in \{1..N_x\}$, step count $N_x \in \mathbb{N}$ and step size $\Delta x \in \mathbb{R}$ etc. The values of the fields are assumed to be known only on the grid's nodes and are written as

$$\psi(\vec{x}) \to \psi(\vec{x}^k_{i,j}) := \psi^k_{i,j} \text{ etc.},$$

as shown in Fig. 4.1. The indices $i \in \{0, N_x + 1\}$ etc. are used to denote boundary conditions.

4.2 Fresnel Propagation

In this section we derive a class of numerical propagation schemes based on propagation kernels and the discrete Fourier transform (DFT).

4.2.1 Homogeneous media

For homogeneous media, the scalar field ψ can be propagated according to the forward Helmholtz equation, resulting in the convolution described by equation Eq. (2.19). If we assume ψ to be periodic in x and y direction, the Fourier transform may be approximated as a discrete Fourier transform [20], where the Fourier space is discretely sampled $\tilde{\psi}(\vec{k}) \to \tilde{\psi}^c_{a,b} = \mathrm{DFT}_{a,b}[\psi^c_{a,b}](\vec{k}_{a,b})$ and $\psi^c_{a,b} = \mathrm{DFT}^{-1}_{a,b}[\tilde{\psi}^c_{a,b}](\vec{x}^c_{a,b})$ and we chose new discretization indices $i, j, k \to a, b, c$ to avoid ambiguities. Given the boundary condition $\psi^0_{a,b}$ we apply Eq. (2.19) to calculate the field at all other discretized points

$$\psi^c_{a,b} = \mathrm{DFT}^{-1}_{a,b}\left[\mathrm{DFT}_{a,b}\left[\psi^0_{a,b}\right](\vec{k}_{a,b}) \cdot e^{-i\, c\Delta z \sqrt{k^2 n^2 - \|\vec{k}_{a,b}\|^2}}\right](\vec{x}^c_{a,b}). \tag{4.1}$$

Using divide-and-conquer algorithms such as the fast Fourier transform algorithm [20], the discrete Fourier transforms may be evaluated very efficiently, which leads to a $\mathcal{O}(N_x N_y \log(N_x N_y))$ overall time-complexity per step $\psi^c_{a,b} \to \psi^{c+1}_{a,b}$.

4.2.2 Inhomogeneous media

The approach may be generalized to inhomogeneous matter, as long as the beam is paraxial and the refractive indices are close to 1. We separate the potential ψ into the product of two potentials $\psi = \psi_\mathrm{d}\psi_\mathrm{r}$, where the factor ψ_d describes the homogeneously diffracted field and ψ_r describes the deviations due to inhomogeneous diffraction and refraction. At $z = 0$ we define the diffracted field to be equal to the scalar potential $\psi_\mathrm{d} = \psi$, and therefore require the refracted field to be equal to one $\psi_r(z = 0) = 1$. Assuming that the disturbances due to inhomogeneous refraction grow smoothly, we choose the propagation distance z small enough so that the gradient of the refracted field is negligible $\|\vec{\nabla}_\perp \psi_r\| \approx 0$. We can then approximate the paraxial Helmholtz equation Eq. (2.16) as

$$\begin{aligned}\frac{\partial \psi}{\partial z} &= \frac{1}{2ikn}\,\Delta_\perp(\psi_\mathrm{r}\psi_\mathrm{d}) - ikn\,\psi_\mathrm{r}\psi_\mathrm{d} \\ &\approx \psi_\mathrm{r}\,\frac{1}{2ikn}\,\Delta_\perp\psi_\mathrm{d} - \psi_\mathrm{d}\,ikn\,\psi_\mathrm{r}.\end{aligned}$$

Comparing coefficients with the product derivative $\frac{\partial \psi}{\partial z} = \psi_\mathrm{r}\frac{\partial \psi_d}{\partial z} + \psi_\mathrm{d}\frac{\partial \psi_r}{\partial z}$, we find a differential equation for both the diffraction and refraction term

$$\frac{\partial \psi_d}{\partial z} = \frac{1}{2ikn}\,\Delta_\perp\psi_d \qquad\qquad \frac{\partial \psi_r}{\partial z} = -ikn\,\psi_r. \tag{4.2, 4.3}$$

Assuming that $n \approx 1$, we approximate $\frac{1}{n} \approx 1$ in Eq. (4.2), for which the coefficient becomes constant. This allows us to apply corollary 12.1 and solve ψ_d with the boundary condition $\psi_d(z=0)$ through the convolution

$$\psi_d = \frac{2ik}{4\pi z}\, \psi_d(z=0) * \exp\!\left(-x_\perp^2\, \frac{2ik}{4z}\right).$$

ψ_r with the boundary condition $\psi_r(z=0) = 1$ is solved by integration

$$\psi_r = \exp\!\left(-ik \int_0^z n \, \mathrm{d}z'\right).$$

When n remains constant with regards to z during each step and the scalar potential is periodic in x and y directions, we may evaluate the convolution by two discrete Fourier transforms. We obtain the discrete propagation scheme

$$\psi_{a,b}^{c+1} = \exp(-ikn_{a,b}^c \Delta z)\, \mathrm{DFT}_{a,b}^{-1}\!\left[\mathrm{DFT}_{a,b}\!\left[\psi_{a,b}^c\right](\vec{k}_{a,b}) \cdot \exp\!\left(i\,\Delta z\, \frac{\|\vec{k}_{a,b}\|^2}{2k}\right)\right](\vec{x}_{a,b}^c). \quad (4.4)$$

The propagation scheme is valid for paraxial beams if the boundaries are periodic, the refraction index is close to 1 and the step size Δz can be chosen sufficiently small so that $\|\vec{\nabla}_\perp \psi_r\| \approx 0$. The same propagation scheme can also be derived by multislice methods [9]. Using fast Fourier transforms, the time complexity per step is $\mathcal{O}(N_x N_y \log(N_x N_y))$.

4.2.3 Hankel Transform

If the regarded system has cylindrical symmetry, the memory and performance requirements of the propagation schemes can be drastically reduced by switching to cylindrical coordinates. The two dimensional Fourier Transform of a radially symmetric function $f(\vec{x}_\perp) = f(r)$ is

$$\begin{aligned}
\mathcal{F}[f(r)]_{\vec{x}_\perp}(\vec{k}_\perp) &= \frac{1}{2\pi} \int_{\mathbb{R}^2} f(r)\, e^{-i\vec{k}\cdot\vec{x}_\perp} \mathrm{d}\vec{x}_\perp \\
&= \frac{1}{2\pi} \int_0^\infty r f(r) \int_0^{2\pi} e^{-i\vec{k}_\perp \cdot \vec{x}_\perp} \mathrm{d}\varphi \, \mathrm{d}r \\
&= \frac{1}{2\pi} \int_0^\infty r f(r) \int_0^{2\pi} e^{-ir\|\vec{k}_\perp\| \cos(\varphi)} \, \mathrm{d}\varphi \, \mathrm{d}r \\
&= \int_0^\infty r f(r)\, J_0(r\|\vec{k}_\perp\|) \, \mathrm{d}r \\
&:= \mathcal{H}_0[f(r)]_{\vec{x}_\perp}(\|\vec{k}_\perp\|),
\end{aligned}$$

which is the zero-order Hankel transform of f. Note that the Hankel transform does not depend on the direction, but only the magnitude of $\vec{k}_\perp$. If there exists a radius $R \in \mathbb{R}$ for which $r > R \Rightarrow f(r) = 0$ holds, the Hankel transformation can be evaluated numerically using the discrete Hankel transformation (DHT) [21]. A disadvantage of the DHT is that it requires the function to be sampled in a non-uniform manner and does not include the value at $r = 0$. However, using sampling theorems, we are able to resample the function at any point with high accuracy [22]. The time complexity of the

discrete Hankel transform is $\mathcal{O}(N_r^2)$, where N_r is the number of radial samples. Faster algorithms such as the fast Hankel transform [23] with $\mathcal{O}(N_r \log(N_r))$ time complexity exist, however, as far as we can tell, they are quite inaccurate and unstable and therefore unsuited for numerical beam propagation.

4.3 Finite Differences

A more general numerical method for inhomogeneous propagation is to solve the paraxial Helmholtz equation by approximating the partial derivatives through their finite difference quotients. This reduces the differential equation to a linear system that can be solved by matrix inversion.

4.3.1 Two Dimensions

We develop a finite difference method for solving the two-dimensional, second order partial differential equation

$$\alpha \frac{\partial u}{\partial z} = A \frac{\partial^2 u}{\partial x^2} + B \frac{\partial u}{\partial x} + C\, u$$

with position dependent $A, B, C, \alpha \in \mathbb{C}$. We keep the redundant term α to later avoid division by zero in cylindrical coordinate systems. Since the field values are only known at discretized grid points, we approximate the derivatives by their first-order finite difference quotients

$$\left.\frac{\partial u}{\partial x}\right|_{\substack{x=x_i \\ z=z_k}} \to \frac{u_{i+1}^k - u_{i-1}^k}{2\Delta x}, \qquad \left.\frac{\partial^2 u}{\partial x^2}\right|_{\substack{x=x_i \\ z=z_k}} \to \frac{u_{i+1}^k - 2u_i^k + u_{i-1}^k}{\Delta x^2}$$

and

$$\left.\frac{\partial u}{\partial z}\right|_{\substack{x=x_i \\ z=z_k}} \to \frac{u_i^{k+1} - u_i^k}{\Delta z},$$

resulting in the forward Euler step

$$\frac{\alpha_i^{k+1} u_i^{k+1} - \alpha_i^k u_i^k}{\Delta z} = \frac{A_i^k}{\Delta x^2}\left(u_{1+i}^k + u_{i-1}^k - 2\,u_i^k\right) + \frac{B_i^k}{2\,\Delta x}\left(u_{1+i}^k - u_{i-1}^k\right) + C_i^k\, u_i^k,$$

Which can be used to determine u_i^{k+1} from u_i^k. This is the Euler method, which is first-order accurate and numerically unstable at large step sizes. We instead choose to implement our propagator using the Crank–Nicolson method, an implicit method that is second-order accurate in Δx and Δz and unconditionally stable [24]. By defining the right side of the Euler step as f_i^k, the according Crank-Nicolson step is written as

$$\frac{\alpha_i^{k+1} u_i^{k+1} - \alpha_i^k u_i^k}{\Delta z} := \frac{f_i^k + f_i^{k+1}}{2}.$$

Reordering terms and introducing the auxiliary variables $a_i^k = \frac{A_i^k}{2\Delta x^2}$, $b_i^k = \frac{B_i^k}{4\Delta x}$, $c_i^k = \frac{C_i^k}{2}$ and $d_i^k = \frac{\alpha_i^k}{\Delta z}$ gives

$$\begin{aligned} & u_i^{k+1}(2\,a_i^{k+1} - c_i^{k+1} + d_i^{k+1}) - u_{i-1}^{k+1}(a_i^{k+1} - b_i^{k+1}) - u_{i+1}^{k+1}(a_i^{k+1} + b_i^{k+1}) \\ = u_i^k \quad & (d_i^k \quad + c_i^k \quad - 2\,a_i^k \quad) + u_{i-1}^k\,(a_i^k \quad - b_i^k \quad) + u_{i+1}^k\,(a_i^k \quad + b_i^k \quad). \end{aligned} \tag{4.5}$$

This is written as the matrix equation

$$M^{k+1}\boldsymbol{u}^{k+1} = \boldsymbol{e}^k, \tag{4.6}$$

where $\left(\boldsymbol{u}^k\right)_i = u_i^k$, M^k is the $N_x \times N_x$ tridiagonal matrix

$$M^k = \begin{pmatrix} 2\,a_1^k - c_1^k + d_1^k & -a_1^k - b_1^k & 0 & \\ -a_2^k + b_2^k & 2\,a_2^k - c_2^k + d_2^k & -a_2^k - b_2^k & \dots \\ 0 & -a_3^k + b_3^k & 2\,a_3^k - c_3^k + d_3^k & \\ & \vdots & & \ddots \end{pmatrix},$$

and e^k is a vector containing the right side of equation Eq. (4.5), which we conveniently define as f_i^k, and the boundary conditions u_0^k and $u_{N_x+1}^k$ at the edges of the grid

$$\boldsymbol{e}^k = \begin{pmatrix} f_1^k + (a_1^{k+1} - b_1^{k+1})\,u_0^{k+1} \\ f_2^k \\ \vdots \\ f_{N_x-1}^k \\ f_{N_x}^k + (a_{N_x}^{k+1} + b_{N_x}^{k+1})\,u_{N_x+1}^{k+1} \end{pmatrix}.$$

Since the matrix B^n is tridiagonal, this linear system can be solved in $\mathcal{O}(N_x)$ operations.

4.3.2 Three dimensions

In three dimensions we solve the paraxial Helmholtz equation using a two-step alternating-direction scheme. This results in a second-order accurate method in Δx, Δy and Δz and time-complexity of order $\mathcal{O}(N_y\,N_x)$. The derivation is very similar to the previous chapter and shown explicitly in Sec. 11.2 in the appendix.

4.4 Envelope equations

Finite-Difference schemes can be used to solve a variety of first and second order linear differential equations. To achieve a high numerical accuracy, the step size Δz must be chosen short enough to resolve the smallest features of the solution. In many paraxial systems the wavelength of the beam is much smaller than scale on which the beams amplitude changes. Therefore, instead of directly determining the scalar potential ψ, we determine the much more slowly varying complex envelope $u = \psi/\exp(-ikz)$ of the

potential. Substituting the envelope into the paraxial Helmholtz equation (2.16) we find the paraxial envelope equation

$$\frac{\partial u}{\partial z} = \frac{1}{2ikn}\,\Delta_\perp u - ik(n-1)\,u. \tag{4.7}$$

For systems where the solution does not depend on y, we formulate the two-dimensional envelope equation

$$\frac{\partial u}{\partial z} = \frac{1}{2ikn}\,\frac{\partial^2 u}{\partial^2 x} - ik(n-1)\,u. \tag{4.8}$$

For cylindrically symmetric systems the envelope equation is determined by insertion of the envelope into the cylindrically symetrical Helmholtz equation (2.18)

$$r\,\frac{\partial u}{\partial z} = \frac{1}{2ikn}\left(r\frac{\partial^2 u}{\partial r^2} + \frac{\partial u}{\partial r}\right) - ikr(n-1)\,u, \tag{4.9}$$

where we multiplied both sides with r to avoid a numerical singularity at $r = 0$. For consistency, the Fresnel propagation schemes of section 4.2 may also be straightforwardly adapted to solve the envelope equations.

4.5 Intensity and Poynting vector

Many important properties of the electromagnetic field can be determined directly from the envelope of the scalar potential. As the envelope differs only in phase from the scalar potential, the field intensity for monochromatic beams (see Eq. (2.10)) is proportional to the squared magnitude of the envelope

$$I \propto |\psi|^2 = |u|^2. \tag{4.10}$$

The time averaged Poynting vector is proportional to the gradient of the phase (see Eq. (2.12)) and can be directly determined from the envelope

$$\left\langle \vec{S} \right\rangle \propto I\,\vec{\nabla}\arg(\psi) = I\,(\vec{\nabla}\arg(u) + k\,\vec{e}_z). \tag{4.11}$$

In numerics, the phase of a complex number is usually computed using the $\mathrm{atan2} : \mathbb{R}^2 \to [-\pi, \pi]$ function [25]. However this induces a discontinuity after each oscillation that has to be taken into account when calculating the gradient. We determine the gradient using central differences (and forward and backward differences at the edges) of the phases, after locally centering them in the interval $[0, 2\pi)$

$$\left.\frac{\partial \varphi}{\partial x}\right|_{\substack{x=x_i \\ z=z_k}} \to \frac{((\varphi^k_{i+1} - \varphi^k_i + \pi) \bmod 2\pi) - ((\varphi^k_{i-1} - \varphi^k_i + \pi) \bmod 2\pi)}{2\Delta x} \text{ etc.}, \tag{4.12}$$

where $\varphi = \mathrm{atan2}(\Im(u), \Re(u))$ is the phase of u. This allows us to determine derivatives in the range $\frac{\partial \varphi}{\partial x} \in [0, \pi/\Delta x)$.

An elegant way to visualize the Poynting vector is to plot streamlines of the vector field on top of the intensity distribution, as seen in figures throughout this work. We determine the streamlines by interpreting $\vec{S}$ as a velocity field and calculating particle trajectories in the field. This is implemented using an explicit Runge-Kutta method of order (4)5 [26], where velocities between discretized points are determined by linearly interpolating $\vec{S}$ from neighboring points.

4.6 Tridiagonal matrix inversion

Using Thomas' Algorithm [20], which is a form of Gaussian elimination, the matrix equations (4.6) and (11.3) can be solved very efficiently. This algorithm has been shown to be stable in many cases, particularly when the matrix involved is diagonally dominant [20]. For the 2D case, solving the matrix equation (4.6) is therfore stable if for all i and k

$$\begin{aligned} |M^k_{i,i}| \geq \sum_{j\neq i} |M^k_{i,j}| &\Leftarrow |2\,a^k_i - c^k_i + d^k_i| \geq |b^k_i - a^k_i| + |a^k_i + b^k_i| \\ &\Leftarrow \Big||2\,a^k_i + d^k_i| - |c^k_i|\Big| \geq 2\,|a^k_i| + 2\,|b^k_i|, \end{aligned} \tag{4.13}$$

where $a^k_i = \frac{A^k_i}{2\Delta x^2}$, $b^k_i = \frac{B^k_i}{4\Delta x}$, $c^k_i = \frac{C^k_i}{2}$ and $d^k_i = \frac{\alpha^k_i}{\Delta z}$. When $\Re(n) \gg \Im(n)$, $\Delta z \gg \Delta x$ and $\Delta x \sim \lambda$, as is desired in many paraxial setups, we can assume that $a^k_i, b^k_i, c^k_i \in \mathbb{I}$, $d^k_i \in \mathbb{R}$, $|d^k_i|/|a^k_i| \ll 1$ and we may approximate $|2a^k_i + d^k_i| \approx 2|a^k_i| + |d^k_i|^2/4|a^k_i|$. The inequality Eq. (4.13) can then be approximated by

$$|d^k_i| \geq 2\sqrt{|a^k_i|(2|b^k_i| + |c^k_i|)}.$$

From this we can derive a lower bound for the the maximum stable ratio of the step sizes $\Delta z/\Delta x$. For the two dimensional paraxial envelope equation Eq. (4.8) this leads to the condition

$$\frac{\Delta z}{\Delta x} \leq \min_{i,k} \sqrt{\frac{2\Re(n)}{\Re(n) - 1}}, \tag{4.14}$$

where the minimum is taken over all discretized points. In regimes where the index of refraction is very close to one, Δz may therefore be many orders of magnitude larger than Δx or λ.

Note that the accuracy is independent of stability and in the Crank-Nicolson scheme numerical errors scale with the square of the step size.

4.7 Piecewise paraxial propagation

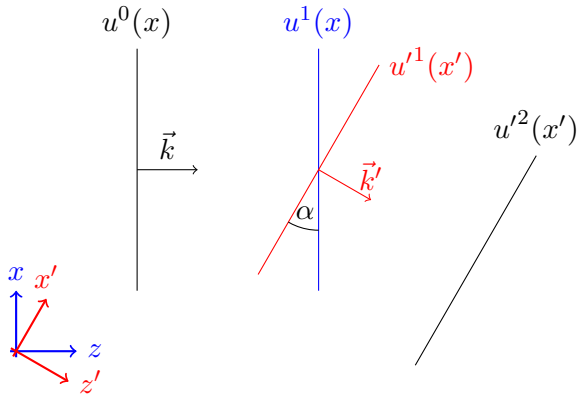

Figure 4.2: Illustration of piecewise paraxial propagation. After a paraxial propagation step $u^0 \to u^1$ in $\vec{k}$ direction, the field is tilted by a very small angle α to perform a paraxial propagation step $u'^1 \to u'^2$ in $\vec{k}'$ direction.

The curved coordinates are approximated by linear segments in which we perform a linear propagation step, as shown in Fig. 4.2. After each step the coordinate system is slightly tilted by resampling the envelope u to match the next segment. For a coordinate system $x'z'$ tilted by an angle α in the xz plane, the resampled envelope u' is obtained by

$$\begin{aligned} \psi &= u\, e^{-ikz} \\ &= u'\, e^{-ikz'} \\ &= u'\, e^{-ik(x\sin(\alpha)+z\cos(\alpha))} \\ \Rightarrow u' &= u\, e^{ikx\sin(\alpha)}. \end{aligned}$$

When the angle α is very small, the envelope is approximately identical in both coordinate systems $u'(x') \approx u'(x)$. The new envelope in the tilted coordinate system is then obtained by multiplying $\exp(ikx\sin(\alpha))$ with the original envelope.

5

PYPROPAGATE

We implement the numerical algorithms for stationary propagation presented in Chap. 4 into the *PyPropagate* framework. This offers a high-performance C++ toolbox with a Python 2.7 front-end for paraxial propagation. An object-oriented approach, that decouples simulation parameters from the propagation algorithms allows users to setup simulations in very few lines of code and solve them using any of the implemented propagators. See Code 5.1 for an example.

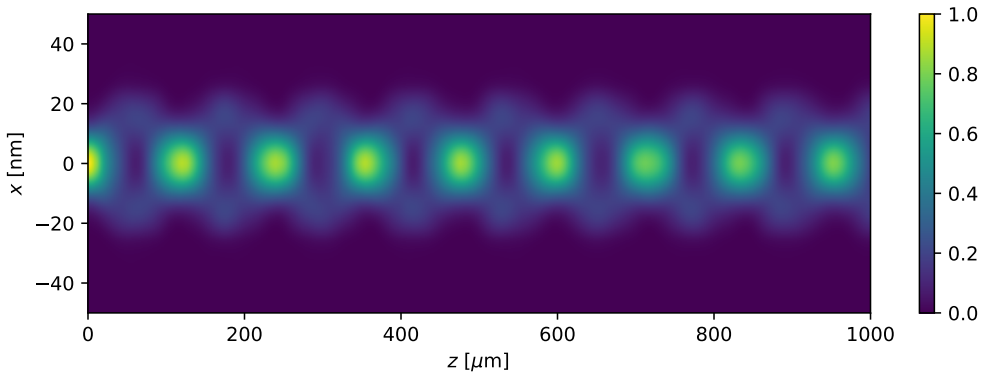

Figure 5.1: Output of Code 5.1.

```
from PyPropagate import *
settings = presets.settings.create_paraxial_wave_equation_settings() # stores simulation parameters
s = settings.symbols # shortcut to relevant parameters
s.u0 = pc.exp(-(s.x**2+s.y**2)/(2*(10*units.nm)**2)) # initial condition
s.u_boundary = 0 # boundary condition
nGe = presets.medium.create_material('Ge',settings) # use xraylib to determine refractive index
s.n = pc.piecewise((1   ,s.x**2+s.y**2 <= (25*units.nm)**2), # define the waveguide
                   (nGe,True))
settings.simulation_box.set((100*units.nm,100*units.nm,1*units.mm),(1024,1024,1000)) # set simulation box
     physical and voxel size
settings.wave_equation.set_energy(50*units.keV) # set photon energy
propagator = propagators.FiniteDifferences3D(settings) # initialize propagator
field = propagator.run_slice()[:,0,:] # propagate and store the slice y = 0
plot(field) # plot result
```

Code Listing 5.1: Example code for *PyPropagate* that propagates a symmetrical Gaussian beam with initial waist size $\sigma_r = 10$ nm through a cylindrical Germanium waveguide at room temperature with Vacuum core and 50 nm diameter using three dimensional Finite Differences. The beam is monochromatic with 50 keV photon energy. This script outputs a plot of the field intensity shown in Fig. 5.1.

5.1 Installation

PyPropagate can be installed through pip by running the single command

```
> pip install pypropagate
```

The current version of *PyPropagate* (1.2) runs on the Linux and Mac operating systems and requires fairly recent versions of the following open-source software to be installed and activated on the system

- python 2.7
- pip
- boost-python
- gcc or equivalent compiler.

The source code is freely available online and can be accessed at `https://github.com/TheLartians/PyPropagate`. It is licensed under the GNU General Public License [27] and third-party contributions are strongly encouraged.

5.2 Propagators

PyPropagate implements different numerical propagators based on different algorithms. These solve the envelope equations from section 4.4 for propagating the discretized field u, which is stored in a one or two dimensional complex floating point array with double precision (128 bit per scalar). The following table gives an overview of the available propagators.

Propagator	Equation	Dimensions	Edge condition	Algorithm
Finite Differences 2D	Eq. (4.8)	1+1	Dirichlet	Sec. 4.3.1
Finite Differences 3D	Eq. (4.7)	2+1	Dirichlet	Sec. 11.2
Finite Differences CS	Eq. (4.9)	1+1	Dirichlet	Sec. 4.3.1
Fresnel 2D	Eq. (4.8)	1+1	Periodic	Sec. 4.2.2
Fresnel 3D	Eq. (4.7)	2+1	Periodic	Sec. 4.2.2
Fresnel CS	Eq. (4.9)	1+1	Dirichlet (0)	Sec. 4.2.2, Sec. 4.2.3

Table 5.1: Overview of the propagators implemented by *PyPropagate*. The dimensions are written as the sum of the perpendicular degrees of freedom and the propagation direction. The boundary conditions of the propagators differ depending on the propagation scheme.

5.3 Refractive indices

To determine the refractive indices of different materials we use functions provided by the *xraylib* library [28, 29]. If not explicitly mentioned the density of elements is taken from tabulated values at room temperature from *xraylib*.

5.4 Performance

The computation time depends on the internal algorithm used and differs for each propagator. The table below gives an overview of the time-complexity per step for each propagator and an example runtime.

Propagator	Time complexity	Example runtime
Finite Differences 2D	$\mathcal{O}(N_x)$	0.3 s
Finite Differences 3D	$\mathcal{O}(N_x\, N_y)$	120 s
Finite Differences CS	$\mathcal{O}(N_r)$	0.3 s
Fresnel 2D	$\mathcal{O}(N_x\, \log(N_x))$	0.6 s
Fresnel 3D	$\mathcal{O}(N_x N_y\, \log(N_x N_y))$	260 s
Fresnel CS	$\mathcal{O}(N_r^2)$	30 s

Table 5.2: Time complexity per step of the propagators with an example runtime for Code 5.1 ($N_x = N_y = N_r = 1024$, $N_z = 1000$) using different propagators on a 2011 dual-core consumer notebook. Note that two dimensional propagation schemes are not parallelized.

5.5 Validation

We validate the implementation of the propagators by comparing numerically obtained results with analytical results from previous chapters.

5.5.1 Free space

To verify the results of free space propagation, we propagate a symmetrical Gaussian beam as defined in Sec. 3.1. As initial condition we set the field distribution of a two or three dimensional monochromatic Gaussian beam with 12 keV photon energy, a waist size of $\sigma_x = \sigma_r = 0.25\,\mu\text{m}$, at 1 mm from the focus. The step sizes of the numerical propagators are $\Delta x = \Delta y = 10$ nm and $\Delta z = 20\,\mu\text{m}$. The full script for the simulation is Code 14.5. Fig. 5.2 shows the intensity distribution and energy flux of the analytical and numerically propagated fields.

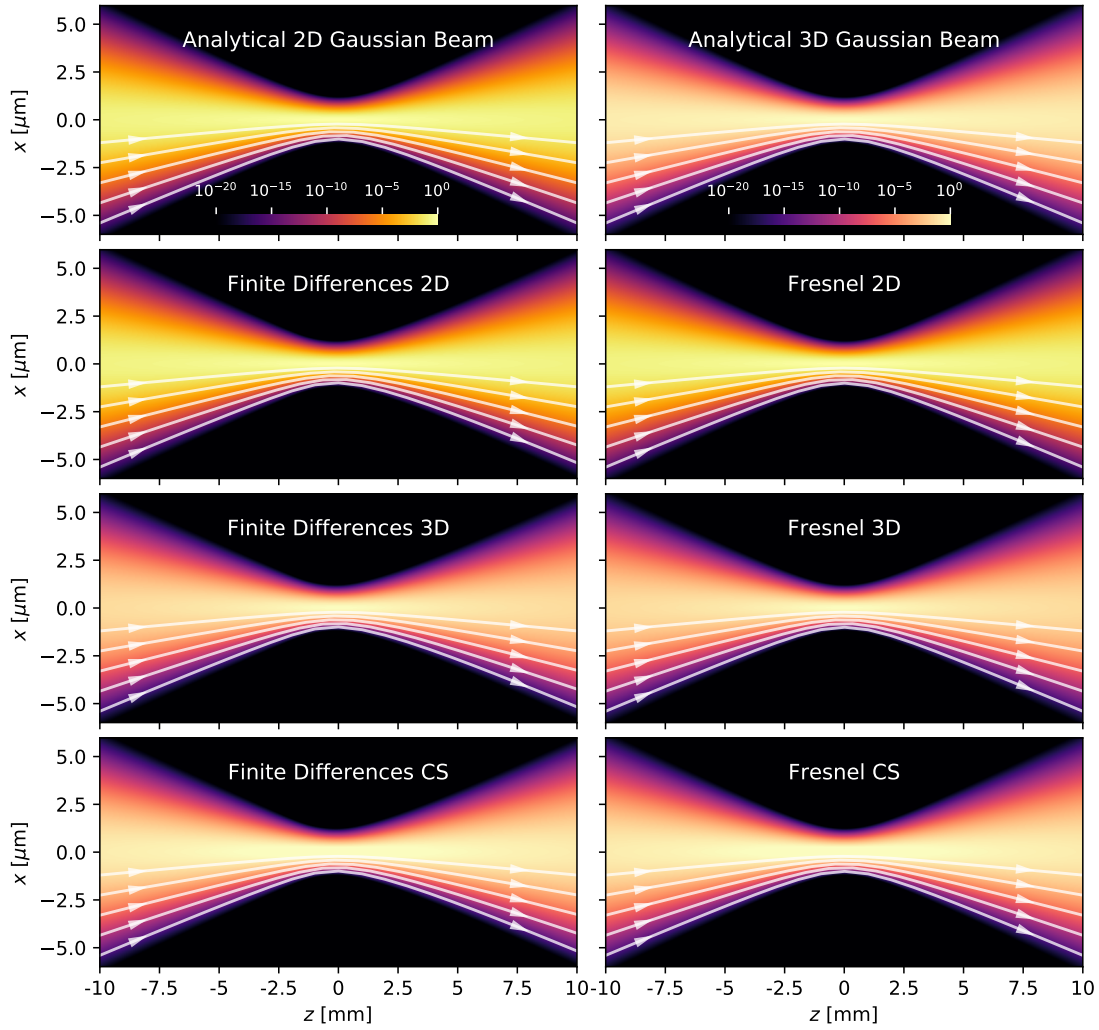

Figure 5.2: Comparison of Gaussian beams calculated by different propagators. The field intensity is normalized and color coded while the direction of the time-averaged Poynting vector is indicated by white streamlines. The top row shows the analytically obtained solutions by applying Cor. 12.2 to the paraxial Helmholtz equation. The script to create this figure is Code 14.5.

5.5.2 Inhomogeneous matter

To verify the propagators in inhomogeneous matter, we compare numerically propagated results from slab and cylindrical waveguides with the analytical solutions from Sec. 3.2. The waveguides consist of a vacuum core and a Germanium cladding at room temperature. The diameter of the waveguides is chosen as 50 nm and the initial beam is Gaussian distributed with a waist size of $\sigma_x = \sigma_r = 100$ nm. The step sizes of the numerical propagators are $\Delta x = \Delta y = 0.7$ nm and $\Delta z = 16$ nm. The full details of the simulation parameters are in Code 14.6. Fig. 5.3 shows the intensity distribution and energy flow of the analytical and numerically propagated fields. A validation for 100 nm waveguide diameter is shown on Fig. 13.1 in the appendix.

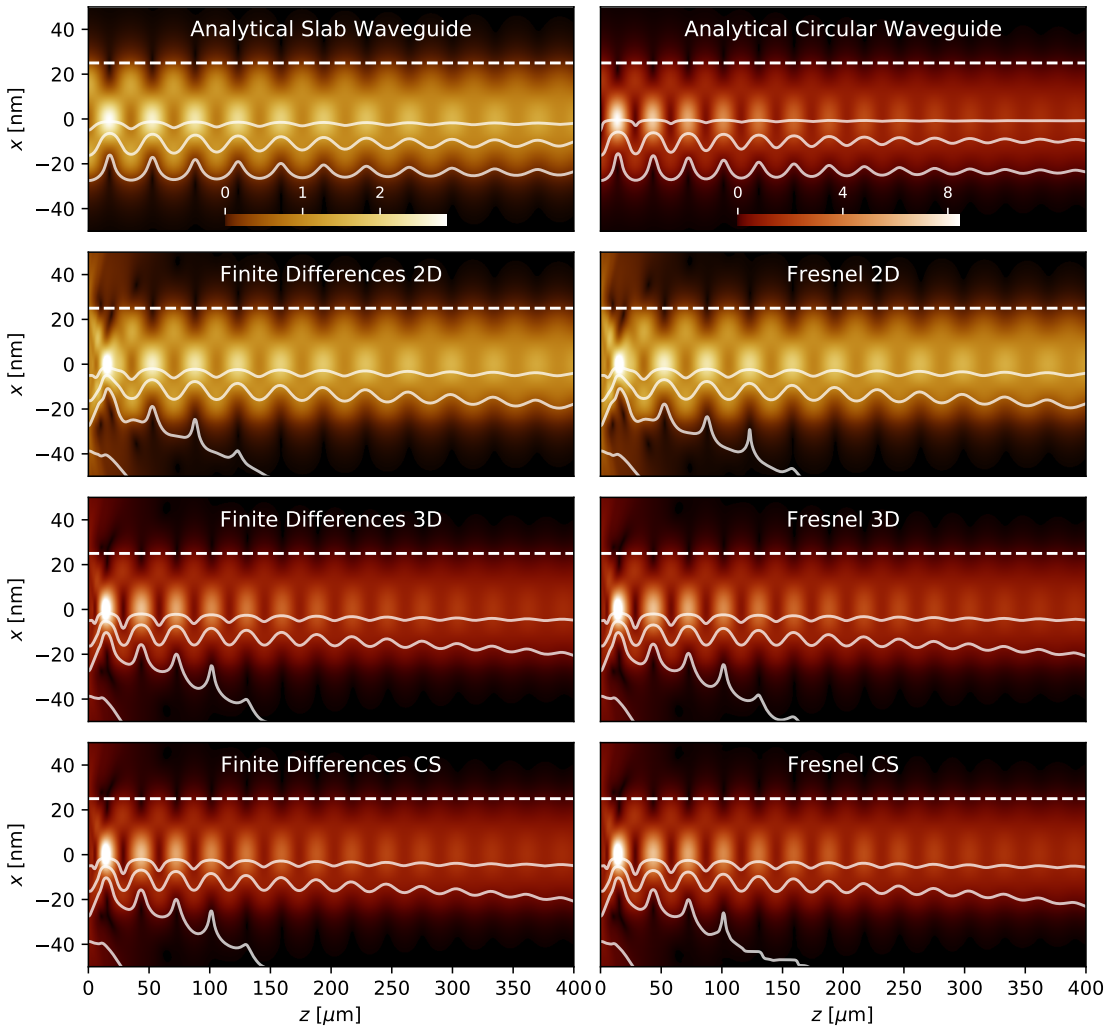

Figure 5.3: Comparison of fields in Germanium waveguides with 25 nm diameter obtained by different propagators. The field intensity is normalized and color coded while the direction of the time-averaged Poynting vector is indicated by white streamlines. The upper waveguide edge is marked by a dashed line. Note that the analytical solutions are determined by projection of the incident field onto the guided modes, while the simulations contain other modes that dissipate into the cladding. The script to create this figure is Code 14.6.

5.6 Propagator accuracy

While the Fresnel propagator is superior in homogeneous matter, finite difference propagation is better suited for in propagation in inhomogeneous matter distributions. In free space of homogeneous matter, the accuracy of Fresnel propagation does not depend on the step size Δz allowing to make arbitrary large steps, while finite differences propagators create significant numerical errors at large step sizes. The artifacts for a Gaussian beam are shown in Fig. 5.4.

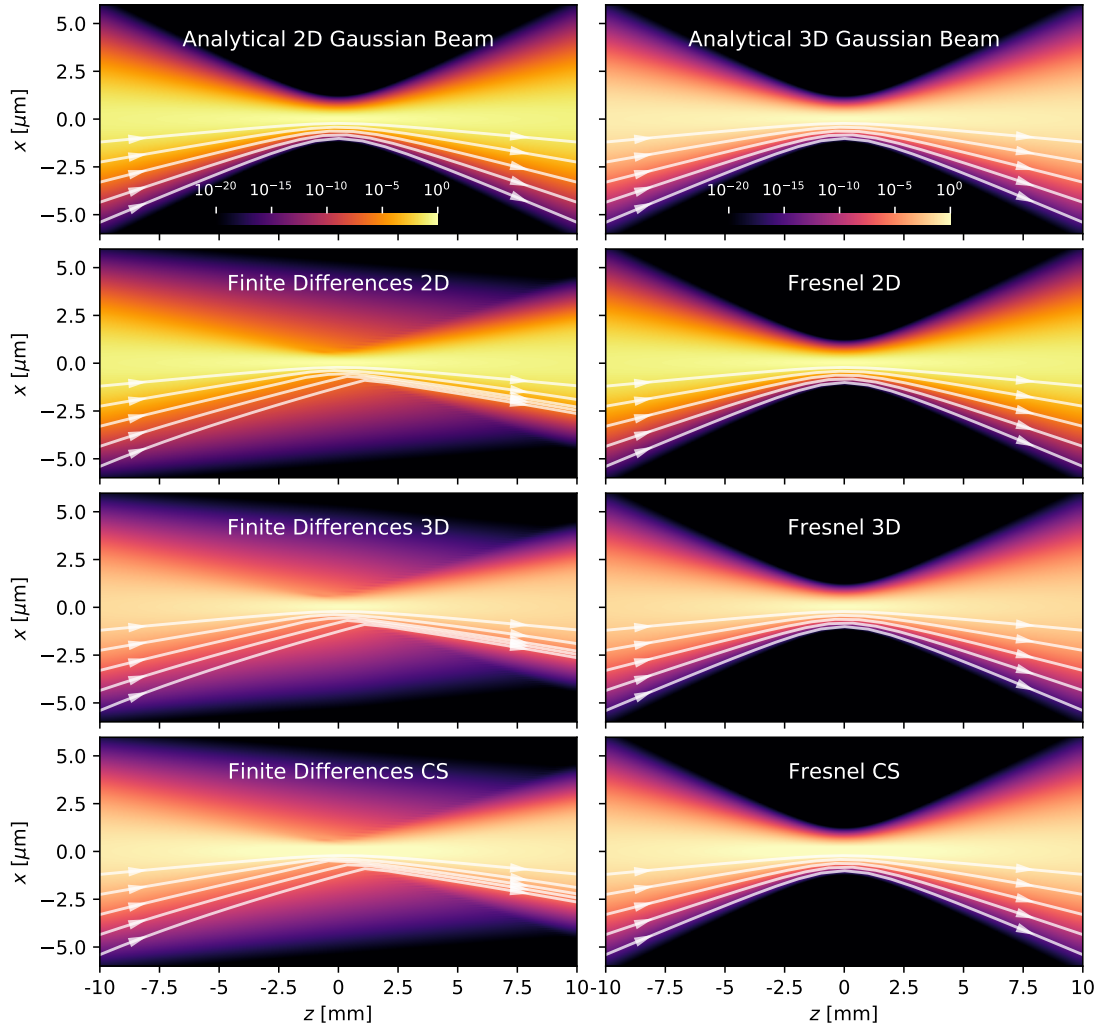

Figure 5.4: Simulations of Fig. 5.4 rerun with a step size of $\Delta z = 400$ µm. As the accuracy of Fresnel propagation does not depend on the step size, the Fresnel results remain unchanged. However the finite difference algorithms produce significant numerical errors. The script to create this figure is Code 14.5.

In inhomogeneous matter, the step size must be chosen extremely small for Fresnel propagation to produce accurate results. The finite differences propagator is still accurate at fairly large step sizes. Fig. 5.5 shows the artifacts for waveguide propagation.

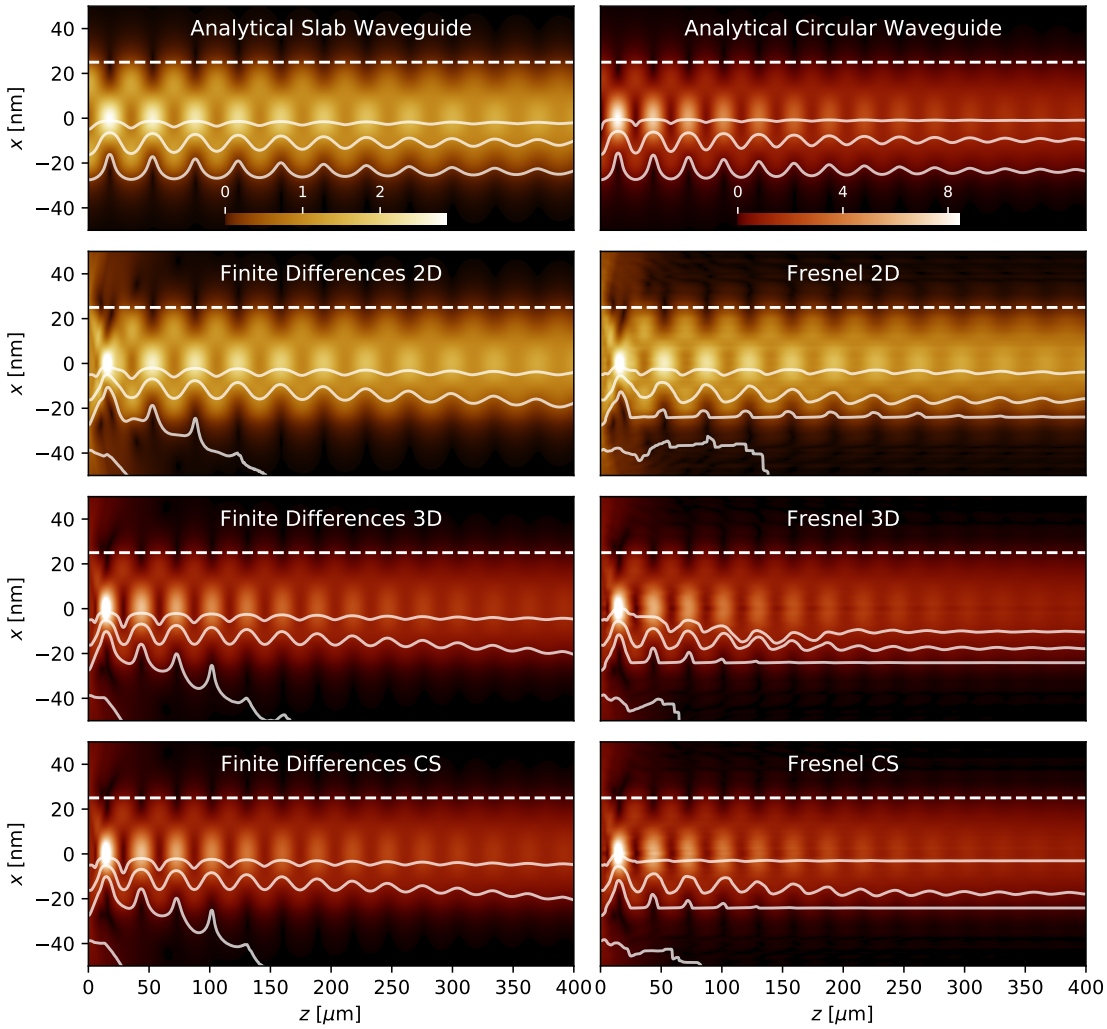

Figure 5.5: Simulations of Fig. 5.3 rerun with a step size of $\Delta z = 0.8$ µm. With these parameters the solutions created by finite difference propagation are still accurate, however Fresnel propagation introduces horizontal artifacts that prevent an accurate determination of the energy flow. The script to create this figure is Code 14.6.

5.6.1 Piecewise paraxial propagation

To verify the piecewise paraxial propagation scheme we compare results by the finite difference 2D propagator in a curved slab waveguide in cartesian and curved coordinates. The waveguide consists of a curved vacuum guiding layer with 200 nm diameter, 40 mm radius of curvature and a tantalum cladding at room temperature. The incident field is a plane wave with 7.9 keV photon energy. In Fig. 5.6 we compare fields obtained by finite difference propagation in cartesian and curved coordinates.

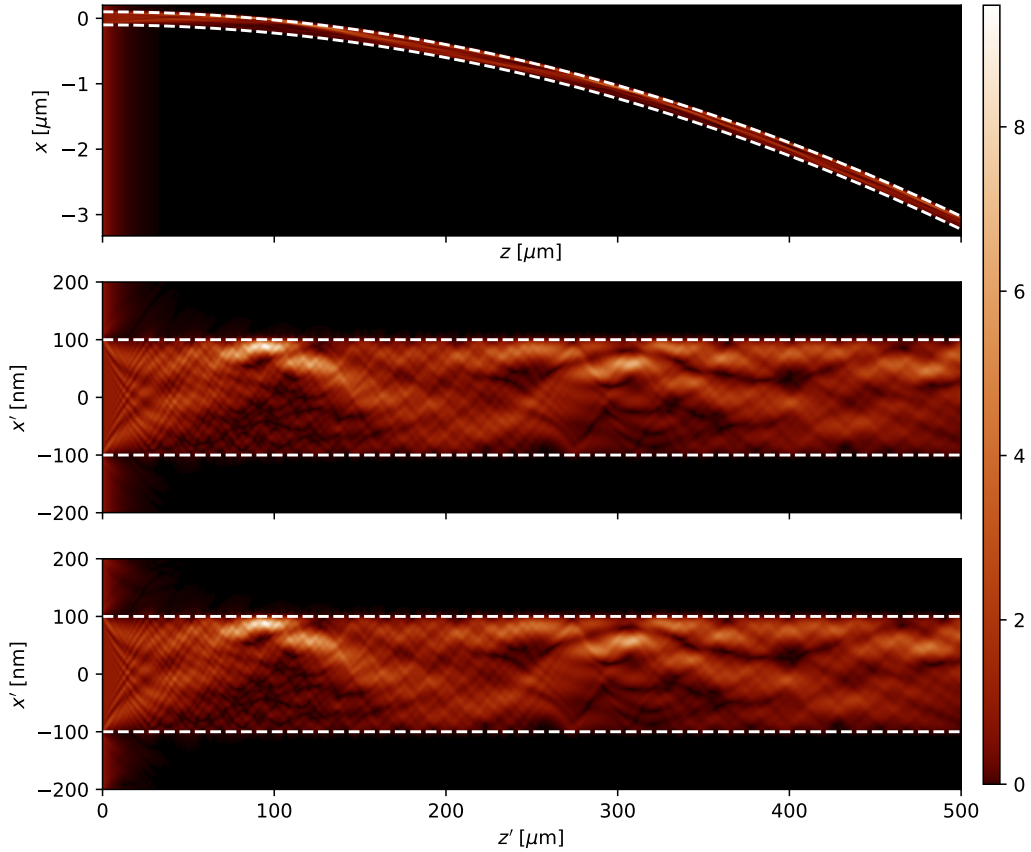

Figure 5.6: Above: normalized intensity distribution of a 7.9 keV plane wave in a tantalum curved slab waveguide with 200 nm diameter and a radius of curvature of 40 mm. The waveguide edges are illustrated by dashed white lines. The step sizes are $\Delta x = 350$ pm and $\Delta z = 50$ nm. Center: intensity distribution from above simulation, straightened by a coordinate system that curves with the waveguide. Below: the intensity distribution of the curved coordinate system obtained directly by piecewise paraxial propagation. Due to the straightened geometry the size of the simulation box can be chosen much smaller while keeping the step-count as before, resulting in a step size of $\Delta x' = 40$ pm. The script to produce this figure is Code 14.7.

6

CONCLUSION I

We have developed a strong theoretical and numerical framework for paraxial electromagnetic propagation in non-magnetic, isotropic, linear and piecewise homogeneous media at relatively low field amplitudes.

Through a rigorous deviation we have derived a scalar diffraction theory from Maxwell equations and have shown that these lead to the usual closed-form expressions for the intensity and time-averaged energy flux density. We have derived the paraxial Helmholtz equation (2.16), which generalizes the widely used parabolic wave equation (11.1) to systems where the refractive index differs greatly from one. We then have shown that the usual Fresnel and Fraunhofer propagation schemes are straightforwardly derivable from the paraxial Helmholtz equation. The n-dimensional formulation of Fresnel propagation allows us to derive the n-dimensional Gaussian beam (Cor. 12.2), which is an exact solution of the paraxial Helmholtz equation in one and two dimensions.

The scalar theory has been heuristically extended to piecewise homogeneous media by postulating that the scalar potential is contiguously differentiable at the material boundaries. The domain of validity of this assumption remains unclear and must be investigated further to identify how polarization dependent phenomena such transverse electric or magnetic polarization and Brewster's angle can be included in the scalar theory.

By discretizing the scalar potential into a regular grid, we have derived methods to numerically solve the paraxial Helmholtz equation for arbitrary initial conditions and matter distributions. For refractive indices close to one, we have separated the potential into a diffracted and refracted term to generalize the Fresnel propagation scheme to piecewise homogeneous matter distributions. This leads to a numerical method that is essentially identical to the well known multi-slice propagation method [9]. The more general finite difference method directly solves the propagation equations using the Crank-Nicolson method in two dimensions or an implicit two-step alternating-direction scheme in three dimensions, achieving very high numerical performance and accuracy.

In free space and homogeneous matter, Fresnel propagation is superior as it can make arbitrary large propagation steps and numerical errors do not accumulate. In inhomogeneous matter, the finite difference propagators produce far better results at lower time complexity. For refractive indices close to one, both schemes solve the same differential equation and should converge to identical results. Therefore Fresnel and finite difference propagation can be used to verify each other and to detect numerical artifacts.

In systems with cylindrical symmetry, such as circular waveguides or zoneplates [30], the cylindrical propagation schemes yield similar accuracy as three dimensional propagation at much lower lower computational cost and memory requirement. With radially symmetrical finite differences the performance gain is dramatic, as runtime and memory consumption scale with $\mathcal{O}(N_r)$ per step, as opposed to $\mathcal{O}(N_r^2)$ per step in three dimensional finite difference propagation. For cylindrically symmetrical Fresnel propagation, the runtime scales with $\mathcal{O}(N_r^2)$, compared to $\mathcal{O}(N_r^2 \log(N_r))$ in three dimensions. Another disadvantage of cylindrically symmetrical Fresnel propagation is that the potential at the boundaries must drop to zero, as required by the discrete Hankel transform. It should therefore only be preferred against cylindrically symmetrical finite differences when propagating very large steps in free space or homogeneous matter.

We have generalized paraxial propagation to curved geometries through piecewise paraxial propagation. This opens new possibilities for numerical propagation as it allows to simulate systems where the paraxial approximation is only locally valid, for example in rings or strongly curved waveguides.

The use of streamlines to indicate the direction of the time-averaged Poynting vector elegantly enriches classical intensity plots by adding information about the energy flow in the electromagnetic field. We predict that this method has a large potential for visual data analysis and publications.

All algorithms have been implemented in the open-source propagation framework *PyPropagate* [11], which is freely available online. The framework allows scientists to quickly define, run and share code for optical simulations while using fast and reliable algorithms. The Python frontend allows the seamless integration of *PyPropagate* with a huge set of optical frameworks, such as *xraylib* [28], which we have used to determine material properties. So far *PyPropagate* has been tested on Linux and Mac operating systems and runs on Python 2.7.

When the beam is not paraxial or the medium does not fulfill the conditions above, one must resort to more general algorithms, such as *Finite Difference Frequency Domain* [14], which directly solve Maxwell's field equations. These algorithms however usually require a much denser spaced discretization grid as the full vectorial electromagnetic fields need to be resolved accurately. The domain that may be regarded is therefore strongly limited by available memory and computational power.

In future work *PyPropagate* may be extended to include non-paraxial propagation methods (such as [31]) or other more general discretization schemes such as the finite element method [32]. Latter could also be used for the derivation of a more general paraxial propagation method, as each element can be equipped with an individual envelope and paraxial propagation direction. The current propagation schemes could also be adapted to make use of advanced graphics processing units available in modern hardware to further improve performance. By combining *PyPropagate* with beamline simulation libraries such as *ShadowOui* [33] or the *Synchrotron Radiation Workshop* [34] we can simulate full synchrotron beamlines for more application-oriented initial field distributions.

Part II

TIME DEPENDENT PROPAGATION

7

TIME-DEPENDENT FIELDS

7.1 Narrow-banded high-energy signals

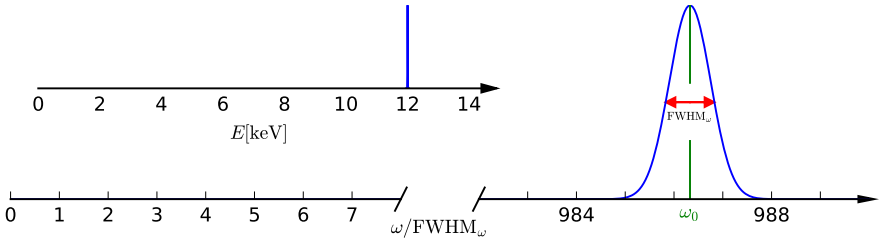

Figure 7.1: Illustration of the spectrum of a narrow-banded high-energy pulse enveloped by a Gaussian. The pulse duration FWHM_t (full width at half maximum) is 300 attoseconds at a photon energy of $E_0 = \hbar\omega_0 = 12$ keV. In the lower plot frequencies are shown in units of $\mathrm{FWHM}_\omega = 8\log(2)/\mathrm{FWHM}_t$, which is the full width at half maximum of the pulse in frequency domain.

Current generation free electron lasers (FELs) are capable of generating very short, high-energy wavepackets. For example, the XFELs at TESLA are build to create 100 fs pulses with photon energies as high as 12 keV [35]. The pulses consist of short coherent wavepackets with durations as low as 300 as. The spectrum of such a wavepacket is quite narrow-banded as illustrated in Fig. 7.1.

7.2 Instantaneous intensity

While the time-averaged energy density or intensity is approximated by the scalar potential summed over all frequencies, the time-dependent energy density, or instantaneous intensity I_{inst}, is approximately proportional to the square of the Fourier transform of the scalar potential. The energy flux density of the electromagnetic field at a given point in time is given by the magnitude of the Poynting vector. Following the definition Eq. (2.8) it is determined from

$$\begin{aligned} I_{\mathrm{inst}} &= \|\vec{S}\| = \|\vec{E} \times \vec{H}\| \\ &= \|\vec{E}\|\|\vec{H}\| \sin(\theta), \end{aligned}$$

where θ is the angle between the $\vec{E}$ and the $\vec{H}$ field. In Sec. 2.3.3 we see that the Fourier transformed fields of paraxial beams described by a scalar potential ψ are approximately orthogonal and can be approximated as $\hat{\vec{E}} \approx n\psi\,\vec{e}_x$ and $\hat{\vec{B}} \approx \frac{n^2}{c}\psi\,\vec{e}_y$. We can then approximate the instantaneous intensity as

$$I_{\text{inst}} \approx c\epsilon_0 \left|\mathcal{F}_\omega^{-1}[n\psi](t)\right| \left|\mathcal{F}_\omega^{-1}[n^2\psi](t)\right|.$$

For narrow banded signals where the refractive index does not change significantly in the support of the spectrum, we may approximate it as a constant $n \approx n(\omega_0) := n_0$ and extract it from the Fourier transform

$$\begin{aligned} I_{\text{inst}} &\approx c\epsilon_0 |n_0|^3 \left|\mathcal{F}_\omega^{-1}[\psi](t)\right|^2 \\ &= c\epsilon_0 |n_0|^3 \, A^2, \end{aligned}$$

where we defined the field amplitude

$$A := \mathcal{F}_\omega^{-1}[\psi](t),$$

which is approximately proportional to the $\vec{E}$ and $\vec{B}$ field amplitudes of paraxial beams.

7.3 Complex field amplitude

Using the Hermitian symmetry of the scalar potential, we define the complex field amplitude A', whose spectrum contains only positive frequencies

$$\begin{aligned} A' &:= \mathcal{F}_\omega^{-1}\Big[\psi\ \mathbf{1}_{\{\omega\geq 0\}}\Big](t) \\ &= \frac{1}{\sqrt{2\pi}} \int_0^\infty \psi\ e^{i\omega t}\ \mathrm{d}\omega, \end{aligned}$$

where $\mathbf{1}_{\{\omega\geq 0\}}$ is the indicator function of the set $\{\omega \geq 0\}$ and is defined as $\mathbf{1}_{\{\omega\geq a\}} = 1$ if $\omega \geq a$ and $\mathbf{1}_{\{\omega\geq a\}} = 0$ otherwise, for any $a \in \mathbb{R}$. The actual field amplitude is obtained by taking twice the real component of the complex field amplitude

$$\begin{aligned} A &= \mathcal{F}_\omega^{-1}[\psi](t) \\ &= \sqrt{\frac{2}{\pi}} \int_0^\infty \Re(\psi)\cos(\omega t) - \Im(\psi)\sin(\omega t)\ \mathrm{d}\omega \\ &= \sqrt{\frac{2}{\pi}} \int_0^\infty \Re\Big(\psi\ e^{i\omega t}\Big)\ \mathrm{d}\omega \\ &= 2\,\Re(A'). \end{aligned}$$

By regarding the complex field amplitude we are able to simplify many calculations regarding time dependent propagation.

7.4 Quasi-stationary solutions

If the envelope $u = \psi/\exp(-ikz)$ of the scalar potential can be approximated by the product of a stationary field $\Psi(\vec{x})$ and a spectrum $\xi(\omega)$, the field amplitude is proportional to the stationary field enveloped by a signal moving with speed c

$$\begin{aligned} A &= \mathcal{F}_\omega^{-1}\Big[\Psi(\vec{x})\ \xi(\omega)\, e^{-ikz}\Big](t) \\ &= \Psi(\vec{x})\ \mathcal{F}_\omega^{-1}[\xi(\omega)]\left(t - \frac{z}{c}\right), \end{aligned}$$

where we used the Fourier identities F1 and F2 from Sec. 12.1. Extremely narrow-banded high-energy signals are usually quasi-stationary, as the envelope is often approximately constant throughout the support of the spectrum. However, when the spectrum contains frequencies close to absorption edges of the propagation medium this is no longer valid, as material properties change abruptly at these points.

7.5 Pulsed plane wave

The pulsed plane wave is the one-dimensional time-dependent solution of the forward Helmholtz equation (2.14b). We use it to study characteristic properties of time-dependent pulses such as phase velocity, group velocity and dispersion.

7.5.1 Definition

Given the one dimensional boundary condition $\psi(z = 0) = \psi_0$, where $\partial_x\psi_0 = \partial_y\psi_0 = 0$, the forward Helmholtz equation (2.14b) has the unique solution

$$\psi(z) = \psi_0\ e^{-iknz}.$$

If ψ_0 is narrow banded around a base frequency ω_0 and the refractive index n is a smooth function of ω around ω_0, we may approximate $K(\omega) := kn$ through the Taylor series

$$\begin{aligned} K &\approx \sum_{j=0}^{\infty} \frac{(\omega-\omega_0)^j}{j!}\ \frac{\mathrm{d}^j K}{\mathrm{d}\omega^j}\bigg|_{\omega=\omega_0} \\ &:= \sum_{j=0}^{\infty} \frac{(\omega-\omega_0)^j}{j!}\, K_j, \end{aligned}$$

where we defined $K_j := \frac{\mathrm{d}^j K}{\mathrm{d}\omega^j}|_{\omega=\omega_0}$.

The complex field amplitude is then

$$\begin{aligned}
A' &= \mathcal{F}_\omega^{-1}[\psi_0(\omega)\, e^{-iknz}\, \mathbf{1}_{\{\omega\geq 0\}}](t) \\
&= e^{-iK_0 z}\, \mathcal{F}_\omega^{-1}\left[\psi_0(\omega)\, \exp\left(-iz\sum_{j=1}^{\infty}\frac{(\omega-\omega_0)^j}{j!}\,K_j\right)\, \mathbf{1}_{\{\omega\geq 0\}}\right](t) \\
&= e^{i(\omega_0 t - K_0 z)}\, \mathcal{F}_\omega^{-1}\left[\psi_0(\omega+\omega_0)\, \exp\left(-iz\sum_{j=1}^{\infty}\frac{\omega^j}{j!}\,K_j\right)\, \mathbf{1}_{\{\omega\geq -\omega_0\}}\right](t) \\
&= e^{i(\omega_0 t - K_0 z)}\, \mathcal{F}_\omega^{-1}\left[\psi_0(\omega+\omega_0)\, \exp\left(-iz\sum_{j=2}^{\infty}\frac{\omega^j}{j!}\,K_j\right)\, \mathbf{1}_{\{\omega\geq -\omega_0\}}\right](t-K_1 z), \qquad (7.1)
\end{aligned}$$

where we used the Fourier identities F1, F2 and F3 from Sec. 12.1.

7.5.2 Phase and group velocity

Eq. (7.1) can be interpreted as a plane wave, whose wavefronts travel with the phase velocity $c/n(\omega_0)$, multiplied with a time-dependent envelope traveling in z direction with the group speed $v := 1/\Re(K_1) = c/(\Re(n_0) + \omega_0\Re(n_1))$, where $n_j = \frac{\mathrm{d}^j n}{\mathrm{d}\omega^j}\Big|_{\omega=\omega_0}$.

7.5.3 Dispersion

As the envelope $u_{\omega_0} := A'/\exp(i(\omega_0 t - K_0 z))$ is determined by a multiplication in Fourier space, we can use the identity F5 to write it as a convolution over time

$$u_{\omega_0} = \frac{1}{\sqrt{2\pi}}\, \mathcal{F}_\omega^{-1}\Big[\psi_0(\omega+\omega_0)\, \mathbf{1}_{\{\omega\geq -\omega_0\}}\Big](t) * \mathcal{F}_\omega^{-1}\left[\exp\left(-iz\sum_{j=2}^{\infty}\frac{\omega^j}{j!}\,K_j\right)\right](t-K_1 z).$$

Setting z to 0 in Eq. (7.1), we identify the left argument of the convolution as the boundary condition of the envelope at $z = 0$ with the solution

$$u_{\omega_0} = \frac{1}{\sqrt{2\pi}}\, u_{\omega_0}(z=0,t) * \mathcal{F}_\omega^{-1}\left[\exp\left(-iz\sum_{j=2}^{\infty}\frac{\omega^j}{j!}\,K_j\right)\right](t-K_1 z).$$

When the second derivative of n is negligible, the K series terminates after the second term, allowing us to determine the inverse Fourier transform through the Fourier identity F4. The envelope is then

$$u_{\omega_0} = u_{\omega_0}(z=0,t) * \frac{\exp\Big(-\frac{(t-K_1 z)^2}{iK_2 z}\Big)}{2\pi\sqrt{iK_2 z}}.$$

The convolution kernel takes the shape of a Gaussian distribution that increases in duration as it propagates forward in z. The speed with which the kernels duration increases is set by the $K_2 = (2n_1 + \omega_0 n_2)/c$ term and effectively smooths out the intitial signal.

7.5.4 Gaussian pulse

If the initial envelope is itself Gaussian distributed and given by $u_{\omega_0}(z=0,t) = \exp(-t^2/2\sigma_t^2)$, with an initial duration σ_t, we evaluate the convolution to

$$u_{\omega_0} = \frac{\sigma_t}{\sqrt{\sigma_t^2 - iK_2 z}} \exp\left(-\frac{(t-K_1 z)^2}{2(\sigma_t^2 - iK_2 z)}\right), \tag{7.2}$$

where we used the Fourier identities F4 and F5. After propagating a distance z, the signal is still Gaussian distributed, however the pulse duration Σ_t is now a function of z

$$\begin{aligned}\Sigma_t &= \Re\left(\frac{1}{\sigma_t^2 - iK_2 z}\right)^{-1/2} \\ &= \sqrt{\frac{\left(\sigma_t^2 + z\,\Im(K_2)\right)^2 + z^2\,\Re(K_2)^2}{\sigma_t^2 + z\,\Im(K_2)}}.\end{aligned}$$

For very large propagation distances z the pulse width is approximately given by $\lim_{z\to\infty}\Sigma_t = \sqrt{z\,|K_2|^2/\Im(K_2)}$ or, when absorption is negligible, $\lim_{z\to\infty}\Sigma_t \approx z\,\Re(K_2)/\sigma_t$. We find a dimensionless expression for Σ_t by substituting $z' := z\,\Im(K_2)/\sigma_t^2$ into above equation

$$\frac{\Sigma_t}{\sigma_t} = \sqrt{\frac{(1+z')^2 + (z'\gamma)^2}{1+z'}}, \tag{7.3}$$

where $\gamma := \Re(K_2)/\Im(K_2)$. For negligible absorption, the dimensionless expression becomes

$$\frac{\Sigma_t}{\sigma_t} = \sqrt{1 + (z'')^2},$$

where $z'' = z\,\Re(K_2)/\sigma_t^2$.

8

NUMERICAL TECHNIQUES

8.1 Time-dependent envelope

As narrow-banded high-energy fields oscillate at very high frequencies (ω_0 in time and $k_0 = \omega_0/c$ in space domain), we consider the time-dependent envelope $u_{\omega_0}(t) = A'/\exp(i(\omega_0 t - k_0 z))$ of the complex field amplitude. The relationship of the time-dependent envelope with the stationary envelope $u(\omega) = \psi/e^{-ik_0 z}$, as used by previous stationary propagation schemes in Chap. 4, is given by

$$\begin{aligned} A' =& u_{\omega_0}(t)\; e^{i(\omega_0 t - k_0 z)} = \mathcal{F}_\omega^{-1}\Big[\psi\; \mathbf{1}_{\{\omega\geq 0\}}\Big](t) = \mathcal{F}_\omega^{-1}\Big[u(\omega)\; e^{-ikz}\; \mathbf{1}_{\{\omega\geq 0\}}\Big](t) \\ &\Rightarrow \mathcal{F}_t[u_{\omega_0}(t)](\omega-\omega_0)\; e^{-ik_0 z} = u(\omega)\; e^{-ikz}\; \mathbf{1}_{\{\omega\geq 0\}} \\ &\Rightarrow \mathcal{F}_t[u_{\omega_0}(t)](\omega) = u(\omega+\omega_0)\; e^{-ikz}\; \mathbf{1}_{\{\omega\geq -\omega_0\}}, \end{aligned}$$

where we used the Fourier identity F2. Taking the inverse transform gives the identity

$$u_{\omega_0}(t) = \mathcal{F}_\omega^{-1}\Big[u(\omega+\omega_0)\; e^{-ikz}\; \mathbf{1}_{\{\omega\geq -\omega_0\}}\Big](t), \tag{8.1}$$

which shows that the time-dependent envelope may be determined by the Fourier transform of the positive-frequency stationary envelopes, centered around the base frequency ω_0.

8.2 Locally averaged instantaneous intensity

When local oscillations due to the high-frequency nature of the beam are not relevant we may regard the locally averaged instantaneous intensity. The instantaneous intensity expressed by the complex envelope is

$$\begin{aligned} I_{\text{inst}} &\approx c\epsilon_0 A^2 \\ &= 2c\epsilon_0 \Re\Big(u_{\omega_0}(t)\; e^{i(\omega_0 t - k_0 z)}\Big)^2 \\ &= 2c\epsilon_0 (\Re(u_{\omega_0}(t))\cos(\omega_0 t - k_0 z) - \Im(u_{\omega_0}(t))\sin(\omega_0 t - k_0 z))^2. \end{aligned}$$

At high photon energies the envelope u is much smoother in z than the base oscillation $\left|\frac{\partial u_{\omega_0}}{\partial z}\right| \ll |k_0 u_{\omega_0}|$. We therefore regard the locally averaged instantaneous intensity $\bar{I}_{\text{inst}}$,

that is locally averaged in z over many oscillations. Treating u_{ω_0} as a constant we apply Lemma 12.2 to find

$$I_{\text{inst,avg}} \approx c\epsilon_0 \, |u_{\omega_0}|^2. \tag{8.2}$$

We see that the squared magnitude of the time-dependent envelope is proportional to the locally averaged instantaneous intensity.

8.3 Envelope stretching

When determining solutions of the wave equation for very short pulses, it may become useful to 'stretch' the solution in z direction, as the pulse length would otherwise be too small to sample accurately. This can be done through a coordinate transform $t \to t' = t - a\,z/c$, where $a \in [0,1]$ is the stretching parameter. When the beam is described by an envelope $f(t - \frac{z}{c})$ with a spacial length of σ_z, the coordinate transform results in beam with the stretched length $\sigma_z' = \sigma_z/(1-a)$. For $a = 0$, the beam's length remains unchanged while for $a = 1$ the beam's length is stretched to infinity. It is convenient to introduce the stretching factor $s := \sigma_z'/\sigma_z = 1/(1-a)$. We then determine the stretched complex envelope as a function of t'

$$\begin{aligned} u_{\omega_0}(t) &= \mathcal{F}_\omega^{-1}\Big[u(\omega+\omega_0)\; e^{-ikz}\; \mathbf{1}_{\{\omega \geq -\omega_0\}}\Big](t' + a\,z/c) \\ &= \mathcal{F}_\omega^{-1}\Big[u(\omega+\omega_0)\cdot e^{-ikz/s}\; \mathbf{1}_{\{\omega \geq -\omega_0\}}\Big](t'), \end{aligned} \tag{8.3}$$

where we used the the Fourier identity F2. Note that for fixed z the stretched envelope $u_{\omega_0}(t')$ is identical to the actual envelope $u_{\omega_0}(t)$ with a temporal offset.

8.4 Determining group velocity

When determining the group velocity of an ultra-short periodic pulse we use the envelope stretching technique to accurately sample the full pulse in z-direction, allowing us to follow the propagation of a single pulse through the discretized simulation box. We can then determine the group velocity v_g of the pulse from the group velocity in the stretched coordinate system $v_g' = \Delta z/\Delta t'$

$$v_g = \frac{\Delta z}{\Delta t} = \frac{\Delta z}{\Delta t' + a\frac{\Delta z}{c}} = \frac{c}{a + c/v_g'},$$

where $a \in [0,1]$ is the stretching parameter.

8.5 Periodic envelope propagation

In numerical contexts it becomes necessary to discretize the spectrum. In the simplest case we assume the signal to be time-periodic, thus being composed of discrete frequencies. We develop the following algorithm to determine the stretched time-dependent envelope from the periodic initial signal $u_{\omega_0}|_{z=0}$.

1. Determine the envelope of the scalar potential $u(\omega) = \mathcal{F}_t[u_{\omega_0}|_{z=0}](\omega - \omega_0)$.
2. Use a suitable propagator from Sec. 5.2 to calculate the spatially discretized envelope $u^g_{ef}(\omega_h)$ for a set of N_ω equidistant frequencies ω_h centered around ω_0, where e, f, g, h are the discretization indices.
3. Choose a suitable stretching factor s to determine the stretched time-dependent envelope $u_{\omega_0}(\vec{x}^g_{ef}, {t'}_h) = \mathrm{DFT}^{-1}_h\Big[u^g_{ef}(\omega_h)\; e^{-iz\omega_h/cs}\Big]({t'}_h)$.

Note that we can make use of the fast Fourier transform algorithm to efficiently calculate the discrete Fourier transform. Usually $N_\omega \ll N_z$, resulting in a total $\mathcal{O}(N_\omega N_x N_y N_z)$ time complexity when using finite difference propagators.

8.6 Validation

Together with the stationary propagation schemes, the periodic envelope propagation algorithm has been implemented into the *PyPropagate* framework. We validate the implementation using two example problems and control the results for consistency.

8.6.1 Angled pulses

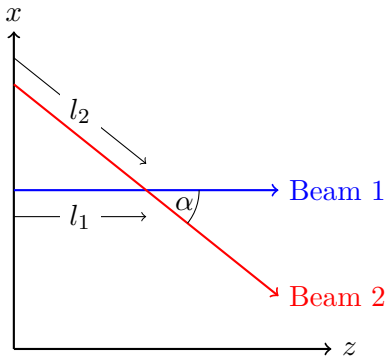

Figure 8.1: Schematic illustration of the angled pulse test-case. Two beams simultaneously start at $z = 0$, one of which propagates downward with an angle α. Due to the different propagation angles, the upper beam travels further to reach the intersection point, resulting in a delay between the two pulses.

We design a simple geometric test-case shown in Fig. 8.1 where two beams are simultaneously pulsed at different positions and angles at $z = 0$ and intersect at $z = l_1$. At the intersection point the straight beam has traveled the distance l_1, while beam angled by α has traveled the distance $l_2 = l_1/\cos(\alpha)$ and is therefore delayed by $\Delta t = l_1/c\,(1/\cos(\alpha) - 1)$. For pulse durations larger than Δt the pulses will interfere with each other while they will miss each other for shorter durations.

The beams are two dimensional Gaussian beams with a width of $\sigma_x = 8.5$ μm and a Gaussian distributed time dependence with a duration of $\mathrm{FWHM}_t = 0.3$ fs. We choose the base photon energy as $E_0 = 12$ keV. The geometric parameters are $\alpha = 20$ mrad and $l_1 = 4\ \mathrm{FWHM}_t/c(\cos(\alpha)^{-1} - 1) \approx 1.8$ mm. As the pulses travel with the group velocity c in free space, the time difference between the pulses at the intersection point is $\Delta t = 4\ \mathrm{FWHM}_t = 1.2$ fs, so that the pulses should not interfere. For stationary propagation we use the Fresnel propagator with discretization step sizes $\Delta x = 0.8$ nm, $\Delta z = 3.6$ μm and $\Delta\omega = 0.07\ \mathrm{FWHM}_\omega$. The full simulation parameters are listed in Code 14.8.

In Fig. 8.2 we compare the intensity of the pulsed beams to monochromatic beams with 12 keV photon energy. As expected, the pulsed beams show no interference patterns at the intersection. The length of the straight pulse in z direction is $\mathrm{FWHM}_z = c/\mathrm{FWHM}_t \approx 90$ nm. As this is approximately 4 times larger than the sampling distance in z, we choose a stretching factor $s = 2000$, to elongate the pulse length to approximately $\mathrm{FWHM}'_z \approx 180$ μm. The angled beam reaches the intersection point $z = l_1$ at $t = \tau := l_2/c \approx 60$ ps. In the stretched coordinate system t' this corresponds

to the time point $\tau' = \tau - a\,l_1/c \approx 4.2$ fs. In Fig. 8.3 (left) we show the locally averaged instantaneous intensity at $t' = \tau'$, which confirms the spacial separation of the two beams. In Fig. 8.3 (right) we see locally averaged instantaneous intensity plotted at $z = l_1$ as a function of t'. By fitting Gaussian distributions to the locally averaged instantaneous intensity at $z = l_1$ and $x = 0$, we measure the delay between the pulses as $\Delta t = 1.21(2)$ fs and the stretched pulse width as $\mathrm{FWHM}'_z = 180(9)$ μm, which are both consistent with the assumptions.

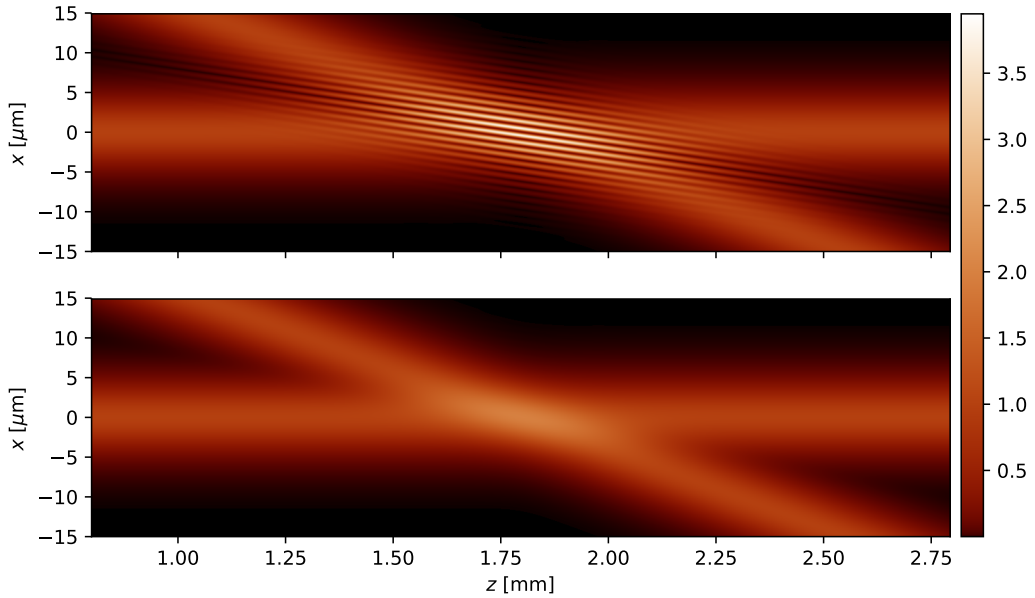

Figure 8.2: Comparison of the normalized intensity of two intersecting monochromatic Gaussian beams (above) and two periodic Gaussian pulses (below) with a duration of 0.3 fs. Due to the different path lengths the pulses do not interfere, as the angled pulse reaches the center 1.2 fs after the straight pulse. The script to create this figure is Code 14.8.

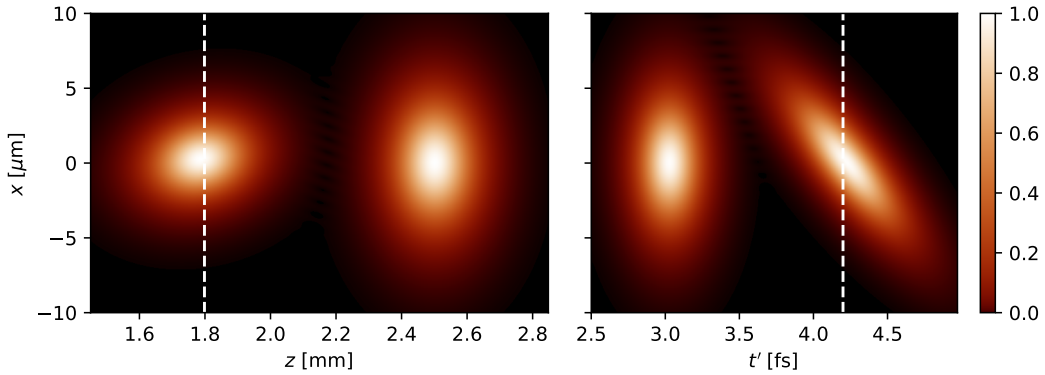

Figure 8.3: Left: normalized locally averaged instantaneous intensity of the pulsed Gaussian beams at $t' = \tau'$. The dashed line indicates the intersection point $z = l_1$. Right: normalized locally averaged instantaneous intensity at $z = l_1$. The angled beam appears distorted as it propagates downward with t'. The intersection time $t' = \tau'$ is indicated by a dashed line. The script to create this figure is Code 14.8.

8.6.2 Dispersion

The algorithm is validated in a system with matter by determining the dispersion of an ultrashort pulse and comparing with the analytical result Eq. (7.3). We choose the initial beam as a one-dimensional Gaussian pulse with the base energy $E_0 = 12$ keV and a duration of $\mathrm{FWHM}_t = 10$ as. The propagation medium is water. As the analytical solution is only valid when the second derivative of n is negligible, we approximate n through its Taylor series up to first order around the base energy, as shown in Fig. 8.4 (Left). Using 2D Fresnel propagation we numerically determine the time dependent field with step sizes $\Delta z = 100$ μm and $\Delta\omega = 0.06\ \mathrm{FWHM}_\omega$. The script for the simulation is Code 14.9. Through Gaussian fits we deterime the pulse duration for all values of z and compare it with the analytical solution Eq. (7.3). Both curves are shown in Fig. 8.4. The maximum deviation of the numerical values from the analytical solution is around 1%.

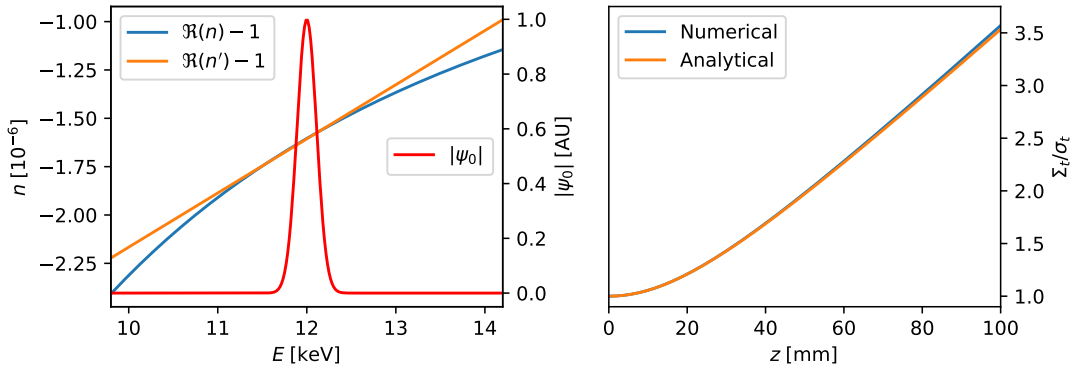

Figure 8.4: Left: The refractive index of water n is approximated by n', the Taylor series of first order around E_0, to match the analytical approximation. Right: Comparison of numerical pulse duration with analytical values from Eq. (7.3) for a Gaussian pulse in water at $E_0 = 12$ keV base photon energy and $\mathrm{FWHM}_t = 10$ as. The script to create this figure is Code 14.9.

9

RESULTS

From our validation we conclude that we may trust the results of the periodic envelope propagation algorithm to an accuracy of 1%, when using similar discretization parameters. We can now use this algorithm to determine the time-dependent electromagnetic fields for arbitrary optical propagation problems in matter under the usual conditions (isotropic non-magnetic media, paraxial low-intensity beam, etc.).

Keeping our focus on waveguides, we analyze time-dependent beam propagation through waveguides.

9.1 Pulse propagation in waveguides

In the analytical analysis in Sec. 3.2 we found that waveguides support a finite number of guided modes, each with its own propagation constant and effective absorption index. We therefore suspect that the modes travel through the waveguide with their own effective dispersion and group velocity depending on the derivatives of the effective refractive indices. Using time-dependent numerical propagation we determine the time dependent field for waveguides with 100 nm diameter with silicon cladding with a Va core. To be able to separate different modes by their group velocity, we choose a Gaussian pulse with an extremely short duration of $\mathrm{FWHM}_t = 5$ as. The spectrum of the pulse and the refractive index of silicon is shown in Fig. 9.1. The script for all simulations and figures in this section is Code 14.10.

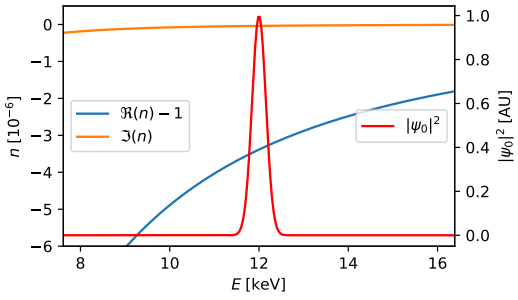

Figure 9.1: Left axis: real and imaginary part of the refractive index of silicon. Right axis: Spectrum of the Gaussian pulse with base energy $E_0 = 12$ kev and $FWHM_t = 5$ as pulse duration.

9.1.1 Slab waveguide

We determine the field in a slab waveguide using the two dimensional Finite difference propagator. The step sizes are chosen as $\Delta x = 0.2$ nm, $\Delta z = 0.6$ μm and $\Delta\omega = 0.07$ FWHM$_\omega$.

Fig. 9.2 shows the instantaneous intensity of the beam at different propagation distances. We observe 3 guided modes that separate spatially and temporally while propagating through the waveguide. As the beam is extremely short and periodic it is difficult to accurately determine the group velocity directly from the instantaneous intensity. We therefore reconstruct the signal with a stretching factor of around 66700 as seen in Fig. 9.3 (left) for different t' values. By regarding the averaged field intensity in the waveguide core as a function of z and t' as seen in Fig. 9.3 (right), we immediately see the different propagation velocity of the modes. By fitting Gaussian distributions to the stretched field at $z = 5$ mm, we determine the group velocities and dispersion of the modes shown in Tab. 9.1.

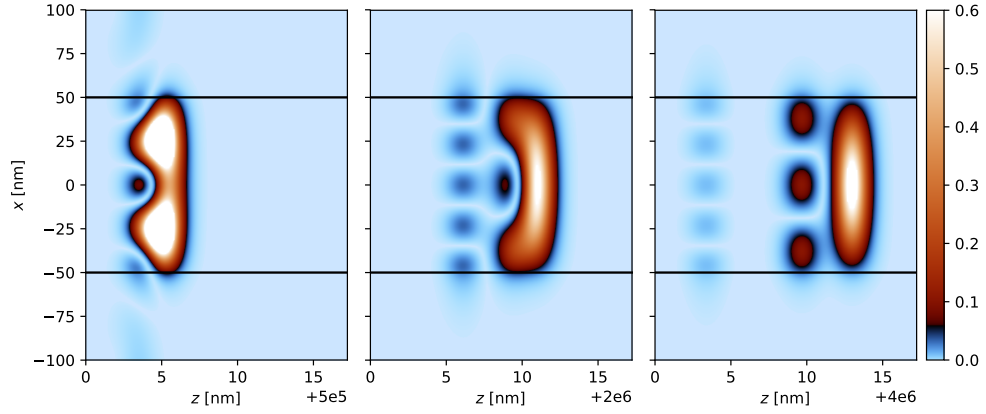

Figure 9.2: Normalized instantaneous intensity of the 5 attosecond beam for different propagation distances $z \approx 0.5$ mm (left), $z \approx 2$ mm (center), $z \approx 4$ mm (right) in the silicon slab waveguide of 100 nm diameter. The waveguide's edges are indicated by black lines.

Mode	v_G/c	$\Sigma_t(z = 5\text{ mm})/\Sigma_t(z = 0)$
1	$1 - 0.18\text{e}{-6}$	1
2	$1 - 1\text{e}{-6}$	1.02
3	$1 - 2.6\text{e}{-6}$	1.12

Table 9.1: Group velocity and pulse duration for the first three symmetrical guided modes in the 100 nm diameter silicon slab waveguide for a Gaussian pulse with FWHM$_t = 5$ as. Values are rounded to the estimated precision of three decimals.

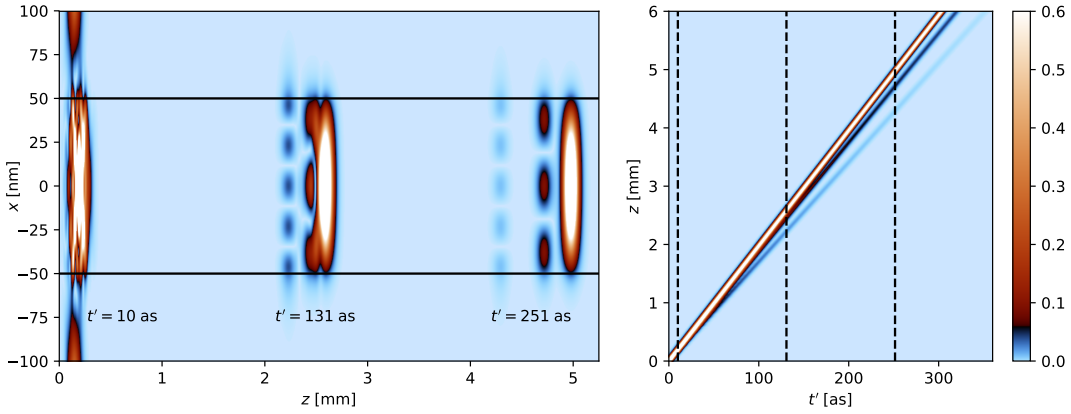

Figure 9.3: Left: normalized locally averaged instantaneous intensity of a pulsed beam with $\mathrm{FWHM}_t = 5$ attoseconds in a 100 nm diameter silicon slab waveguide at different times points t' with a stretching factor of $s = 66700$. The waveguide edges are indicated as black lines. Right: normalized instantaneous intensity locally averaged in z and averaged over the waveguide core. Dashed lines represent the time points on the left.

9.1.2 Cylindrical waveguide

We repeat the procedure for a cylindrical waveguide with 100 nm diameter. The step sizes are identical as with the slab waveguide and $\Delta r = 0.2$ nm. The instantaneous intensity for different propagation distances is shown in Fig. 9.4. Fig. 9.5 shows the stretched instantaneous intensity for determining the group velocity of the waveguide's modes. Again we observe three guided modes with group velocities and dispersion shown in Tab. 9.2.

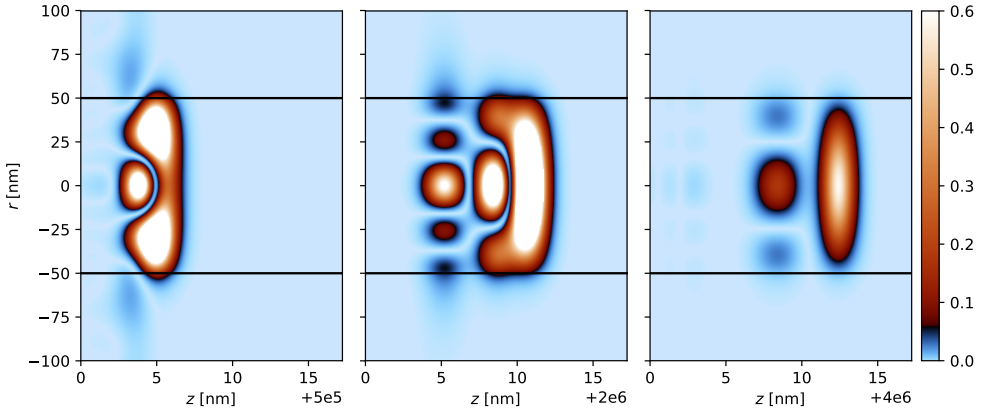

Figure 9.4: Normalized instantaneous intensity of the 5 attosecond beam for different propagation distances $z \approx 0.5$ mm (left), $z \approx 2$ mm (center), $z \approx 4$ mm (right) in the silicon cylindrical waveguide of 100 nm diameter. The waveguide's edges are indicated by black lines.

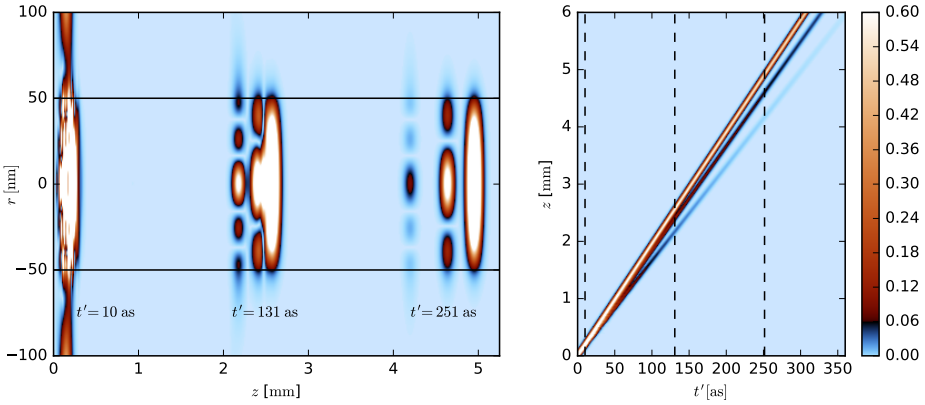

Figure 9.5: Left: normalized locally averaged instantaneous intensity of a pulsed beam with $\mathrm{FWHM}_t = 5$ as in a 100 nm diameter silicon cylindrical waveguide at different times points t' with a stretching factor of $s = 66700$. The waveguide edges are indicated as black lines. Right: normalized instantaneous intensity locally averaged in z and averaged over the waveguide core. Dashed lines represent the time points on the left.

Mode	v_G/c	$\Sigma_t(z = 5\ \mathrm{mm})/\Sigma_t(z = 0)$
1	$1 - 0.33\mathrm{e}{-6}$	1
2	$1 - 1.36\mathrm{e}{-6}$	1.04
3	$1 - 3.07\mathrm{e}{-6}$	1.18

Table 9.2: Group velocity and pulse duration for the first three symmetrical guided modes in the 100 nm diameter silicon cylindrical waveguide for a Gaussian pulse with $\mathrm{FWHM}_t = 5$ as. Values are rounded to the estimated precision of three decimals.

9.1.3 Curved slab waveguide

We repeat the procedure for a curved slab waveguide with a radius of curvature of 40 nm. The instantaneous intensity for different propagation distances is shown in Fig. 9.6. Fig. 9.5 shows the stretched instantaneous intensity for determining the group velocity of the waveguide's modes. In the curved waveguide the modes no longer separate, resulting in the single measurable group velocity and dispersion shown in Tab. 9.3.

Mode	v_G/c	$\Sigma_t(z = 5\ \mathrm{mm})/\Sigma_t(z = 0)$
?	$1 - 1.13\mathrm{e}{-6}$	1.18

Table 9.3: Group velocity and pulse duration for the symmetrical guided modes in the 100 nm diameter silicon curved slab waveguide with a radius of curvature of $R = 40$ µm. Without further analysis we can make no statement about the number of guided modes as they are no longer spatially separated. Values are rounded to the estimated precision of three decimals.

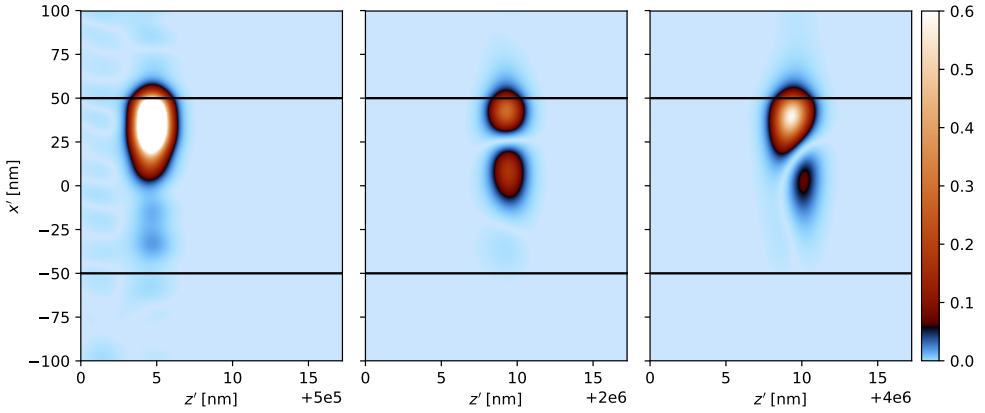

Figure 9.6: Normalized instantaneous intensity of the 5 attosecond beam for different propagation distances $z \approx 0.5$ mm (left), $z \approx 2$ mm (center), $z \approx 4$ mm (right) in the silicon curved waveguide of 100 nm diameter. The waveguide's edges are indicated by black lines.

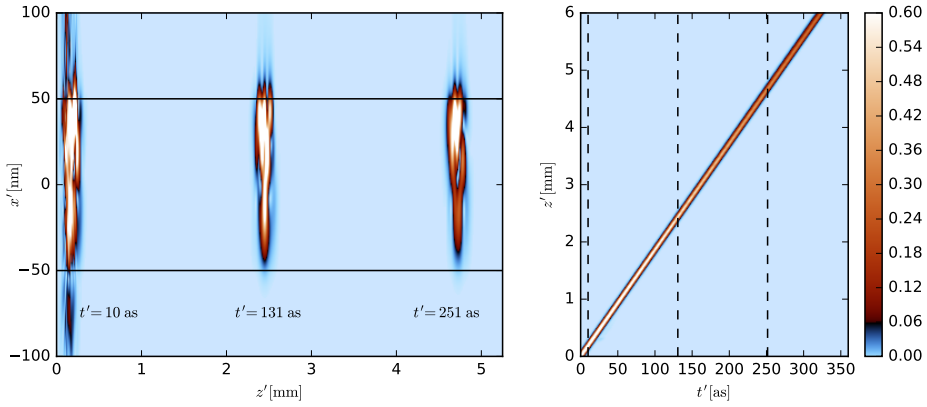

Figure 9.7: Left: normalized locally averaged instantaneous intensity of a pulsed beam with $\mathrm{FWHM}_t = 5$ as in a 100 nm diameter silicon curved slab waveguide at different times points t' with a stretching factor of $s = 66700$. The waveguide edges are indicated as black lines. Right: normalized instantaneous intensity locally averaged in z and averaged over the waveguide core. Dashed lines represent the time points on the left.

9.2 Dispersion at absorption edge

At the absorption edge of a medium the magnitude of the imaginary part of the refractive index instantly increases as shown in Fig. 9.8 for the Nickel K absorption edge at $E \approx 8.33$ keV. To observe the propagation of narrow-banded pulses with energies close to absorption edges, we propagate a Gaussian pulse with a base energy of $E_0 \approx 8.33$ keV and $\mathrm{FWHM}_t = 300$ as in different Nickel structures. The pulse's spectrum is shown on Fig. 9.8. Due to the different absorption coefficients, the high-frequency beam components are absorbed faster by the medium, thus deforming the pulse and spectrum as seen in Fig. 9.9. To determine the effects on the pulse duration, we fit Voigt profiles [36] to the locally averaged instantaneous intensity using *lmfit* [37], resulting in the curve Fig. 9.10 (left). We observe that the pulse's duration initially increases and seems to saturate at to about 1.82 times of the initial duration. Propagating the pulse through a slab and cylindrical waveguide with nickel cladding and Va core creates a similar pattern on a much larger scale and again saturates at approximately the same value, as shown in Fig. 9.10 (center) and (right). The code to produce the figures is Code 14.11.

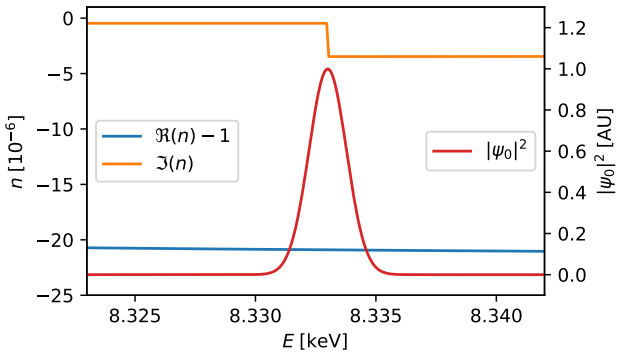

Figure 9.8: Left axis: real and imaginary part of the refractive index of nickel around the K absorption edge. The data is taken from *xraylib* [28]. Right axis: spectrum of the 300 as pulse with a base energy of $E_0 \approx 8.33$ keV.

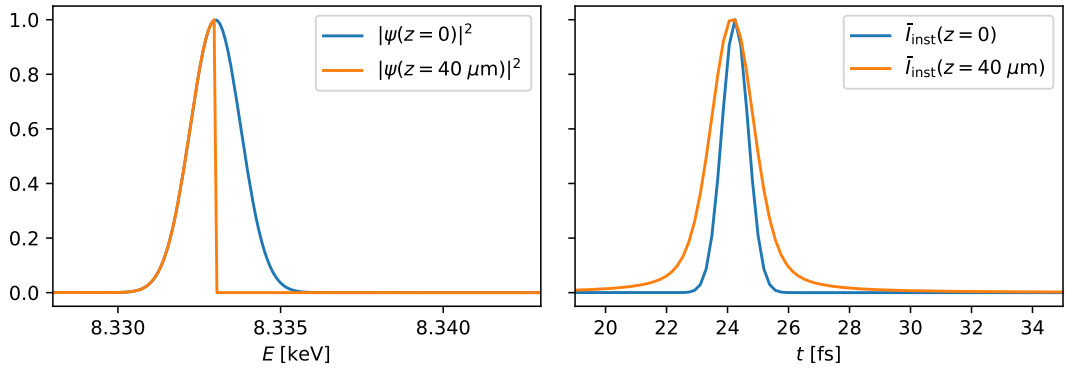

Figure 9.9: Normalized spectrum (left) and locally centered and averaged intensity (right) of an 300 as pulse with a base energy of $E_0 \approx 8.33$ keV propagating in nickel at the propagation distances $z = 0$ and $z = 42$ mm.

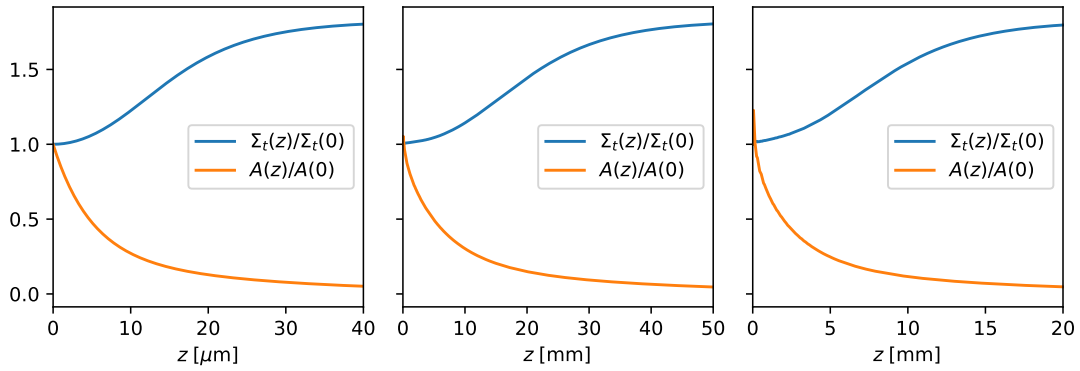

Figure 9.10: Normalized pulse width $\Sigma(z)$ and normalized amplitude A of 300 as pulses with a base energy of $E_0 = 8.3328$ keV in homogeneous nickel (left), a 100 nm Va slab waveguide (center) and cylindrical waveguide (right) in a nickel cladding. The pulse width and intensity are determined by fitting Voigt profiles to the locally averaged intensity profiles. The sudden rise in pulse intensity at the waveguide entrance is due to propagation of the initial plane wave from outside into the waveguide.

10

CONCLUSION II

Building on the stationary propagation framework from Part I, we have developed a theoretical and numerical framework for time-dependent propagation in media with frequency dependent refractive indices for narrow-banded high-energy signals. By generalizing the scalar potential to the time-dependent case and discretizing the frequency space, we have been able to use the well understood stationary propagation algorithms to determine the time-dependent envelope. As we have required the spectrum to be discrete, the scheme may only be used to regard time-periodic signals. Special care must be given to the discretization as periodic signals are able interfere with themselves if the separation time is not large enough.

We have verified the algorithm using a simple geometrical setup with angled Gaussian beams in free space and by comparing the dispersion of a pulse in media with analytical results. In both cases the maximal deviation from our expectations has been around 1 %. We assume the deviation to be due to sampling errors, as the expected values lie within the confidence intervals from the fits. Nonetheless, the results match the expectations sufficiently well and the algorithm can be used for time-dependent propagation of arbitrary initial fields and matter distributions, where the usual conditions for the beam and medium apply.

In domains where these conditions are not applicable, one must resort to more general methods, such as finite-difference time-domain [38] and the finite-element time-domain method [39]. As is the case with stationary propagation, they require a much finer discretization and are therefore very limited by the available computational power.

As FEL signals are composed of very short coherent spikes that are difficult to sample spatially, we have introduced a 'beam-stretching' coordinate transform into the algorithm. This allows us to visualize and analyze the time-dependent beam as it propagates through a sample, even when the sampling distance is much larger than the beam's length.

Applying the algorithm to the propagation of extremely short pulses in waveguides, we have verified that the propagated modes separate temporally and spatially, as they travel with different propagation velocities through the core. We suspect that the propagation velocity may be determined by the derivatives of the effective refractive index of each mode. This could be easily verified by numerical determination of the derivatives of the effective refractive index or by further simulations. The mode separation and group velocities are similar for slab and circular waveguides, however when the waveguide is strongly bent, we do not observe mode separation.

When the base frequency of the beam matches an absorption edge of the propagation medium, the higher-frequency part of the spectrum is absorbed stronger than the lower frequency part, resulting in a deformation of the beam. After a certain propagation distance the upper half is nearly completely absorbed and the beam continues to propagate with a duration of approximately 1.82 times the initial beam's duration. In waveguides, the same deformation takes place, but at a much larger scale, as only a small part of the beam is propagates inside the medium. The model of the refractive index given by *xraylib* may not be completely accurate around absorption edges and simulations using more accurate models may yield other interesting phenomena.

So far we have only analyzed beams with Gaussian distributed spectra, which is a good initial approximation for individual FEL spikes. However, much more application-oriented results can be obtained by simulating the full beamline and pulse-train. For example, a python package that makes this possible is *WaveProperGator* [40], which has been used for time-dependent wavefront propagation through full FEL beamlines [41].

In future work the framework can be extended to support non-linear optical effects, for example by including non-linear terms in the polarization expansion from Sec. 2.2 or by using other methods such as the split-step Fourier method [42].

REFERENCES

[1] Charles A. Brau. "Free-electron lasers". In: *AIP Conference Proceedings* 184.2 (1989), pp. 1615–1706. DOI: 10.1063/1.38022. eprint: http://aip.scitation.org/doi/pdf/10.1063/1.38022. URL: http://aip.scitation.org/doi/abs/10.1063/1.38022.

[2] Sarah Hoffmann-Urlaub and Tim Salditt. "Miniaturized beamsplitters realized by X-ray waveguides". In: *Acta Crystallographica Section A* 72.5 (Sept. 2016), pp. 515–522. DOI: 10.1107/S205327331601144X. URL: https://doi.org/10.1107/S205327331601144X.

[3] Max Born and Emil Wolf. *Principles of Optics (7th Ed)*. Cambridge University Press, 1999.

[4] H S Green and E Wolf. "A Scalar Representation of Electromagnetic Fields". In: *Proceedings of the Physical Society. Section A* 66.12 (1953), p. 1129. URL: http://stacks.iop.org/0370-1298/66/i=12/a=308.

[5] E Wolf. "A Scalar Representation of Electromagnetic Fields: II". In: *Proceedings of the Physical Society* 74.3 (1959), p. 269. URL: http://stacks.iop.org/0370-1328/74/i=3/a=305.

[6] C. G. Gray and B. G. Nickel. "Debye potential representation of vector fields". In: *American Journal of Physics* 47.8 (1979), pp. 736–736. DOI: http://dx.doi.org/11.1119/1.11111. URL: http://scitation.aip.org/content/aapt/journal/ajp/47/7/10.1119/1.11111.

[7] P Roman. "A Scalar Representation of Electromagnetic Fields: III". In: *Proceedings of the Physical Society* 74.3 (1959), p. 281. URL: http://stacks.iop.org/0370-1328/74/i=3/a=306.

[8] Marco Ornigotti and Andrea Aiello. "The Hertz vector revisited: A simple physical picture". In: *Journal of Optics* 16 (2014). URL: http://iopscience.iop.org/2040-8986/16/10/105705.

[9] David M. Paganin. *Coherent X-Ray Optics*. Oxford University Press, 2006. URL: http://www.oup.com/uk/catalogue/?ci=9780198567288.

[10] Christian Fuhse. "X-ray waveguides and waveguide-based lensless imaging". PhD thesis. Germany: Georg-August-Universität Göttingen, 2006.

[11] Lars Melchior. *PyPropagate*. 2016. URL: https://github.com/TheLartians/PyPropagate.

[12] D. Attwood. *Soft X-Rays and Extreme Ultraviolet Radiation: Principles and Applications*. Cambridge University Press, 2007. ISBN: 9780521029971.

[13] J.W. Goodman. *Introduction to Fourier Optics*. McGraw-Hill Series in Electrical and Computer Engineering: Communications and Signal Processing. McGraw-Hill, 1996. ISBN: 9780070242548. URL: https://books.google.de/books?id=QllRAAAAMAAJ.

[14] J.D. Schmidt. *Numerical Simulation of Optical Wave Propagation with Examples in MATLAB*. Press monograph. SPIE, 2010. ISBN: 9780819483263. URL: https://books.google.de/books?id=RqPZSAAACAAJ.

[15] A. S. Marathay and G. B. Parrent. "Use of Scalar Theory in Optics". In: *J. Opt. Soc. Am.* 60.2 (Feb. 1970), pp. 243–245. DOI: 10.1364/JOSA.60.000243. URL: http://www.osapublishing.org/abstract.cfm?URI=josa-60-2-243.

[16] Anton Husakou. "Nichtlineare Phaenomene spektral ultrabreiter Strahlung in Photonischen Kristallfasern und Hohlen Wellenleitern". PhD thesis. Germany: Freie Universität Berlin, 2002.

[17] R.K. Luneburg and M. Herzberger. *Mathematical Theory of Optics*. University of California Press, 1964. URL: https://books.google.de/books?id=-t4bLCrwf60C.

[18] D. Marcuse. *Theory of dielectric optical waveguides*. Quantum electronics–principles and applications. Academic Press, 1974. ISBN: 9780124709508. URL: https://books.google.de/books?id=kZwxAAAAIAAJ.

[19] Wikipedia. *Bessel function*. [Online; accessed 2-November-2016]. 2016. URL: https://en.wikipedia.org/w/index.php?title=Bessel_function&oldid=747522820.

[20] William H. Press et al. *Numerical Recipes in C (2Nd Ed.): The Art of Scientific Computing*. New York, NY, USA: Cambridge University Press, 1992. ISBN: 0-521-43108-5.

[21] Natalie Baddour and Ugo Chouinard. "Theory and Operational Rules for the Discrete Hankel Transform". In: *Journal of the Optical Society of America* 32(4) (2015). URL: https://www.researchgate.net/publication/272942976_Theory_and_Operational_Rules_for_the_Discrete_Hankel_Transform#pf3.

[22] Andrew W. Norfolk and Edward J. Grace. "Reconstruction of optical fields with the Quasi-discrete Hankel transform". In: *Opt. Express* 18.10 (May 2010), pp. 10551–10556. DOI: 10.1364/OE.18.010551. URL: http://www.opticsexpress.org/abstract.cfm?URI=oe-18-10-10551.

[23] Timothy M. Pritchett. "Fast Hankel Transform Algorithms for Optical Beam Propagation". In: *Army Research Labratory* (2001). URL: http://www.arl.army.mil/arlreports/2001/ARL-TR-2492.pdf.

[24] Encyclopedia of Mathematics. *Crank-Nicolson method*. [Online; accessed 9-February-2017]. 2014. URL: https://www.encyclopediaofmath.org/index.php/Crank-Nicolson_method.

[25] cppreference.com. *atan2*. [Online; accessed 10-February-2017]. 2016. URL: http://en.cppreference.com/w/c/numeric/math/atan2.

[26] Ernst Hairer, Syvert P. Nørsett, and Gerhard Wanner. *Solving Ordinary Differential Equations I*. Springer, 1993. ISBN: 978-3-540-56670-0.

[27] *GNU General Public License*. Version 3. Free Software Foundation, June 29, 2007. URL: http://www.gnu.org/licenses/gpl.html.

[28] Tom Schoonjans et al. *xraylib 3.1.0*. Oct. 2014. DOI: 10.5281/zenodo.12378. URL: https://doi.org/10.5281/zenodo.12378.

[29] A. Brunetti et al. "A library for X-ray–matter interaction cross sections for X-ray fluorescence applications". In: *Spectrochimica Acta Part B: Atomic Spectroscopy* 59.10–11 (2004). 17th International Congress on X-Ray Optics and Microanalysis, pp. 1725–1731. ISSN: 0584-8547. DOI: http://dx.doi.org/10.1016/j.sab.2004.03.014. URL: http://www.sciencedirect.com/science/article/pii/S0584854704001843.

[30] Wikipedia. *Zone plate — Wikipedia, The Free Encyclopedia*. [Online; accessed 21-March-2017]. 2017. URL: https://en.wikipedia.org/w/index.php?title=Zone_plate&oldid=768320510.

[31] Anurag Sharma and Arti Agrawal. "New method for nonparaxial beam propagation". In: *J. Opt. Soc. Am. A* 21.6 (June 2004), pp. 1082–1087. DOI: 10.1364/JOSAA.21.001082. URL: http://josaa.osa.org/abstract.cfm?URI=josaa-21-6-1082.

[32] J. Tinsley Oden. "Finite element method". In: *Scholarpedia* 5.5 (2010). revision #122201, p. 9836.

[33] Luca Rebuffi and Manuel Sanchez del Rio. "*ShadowOui*: a new visual environment for X-ray optics and synchrotron beamline simulations". In: *Journal of Synchrotron Radiation* 23.6 (Nov. 2016), pp. 1357–1367. DOI: 10.1107/S1600577516013837. URL: https://doi.org/10.1107/S1600577516013837.

[34] O Chubar et al. "Wavefront propagation simulations for beamlines and experiments with Synchrotron Radiation Workshop". In: *Journal of Physics: Conference Series* 425.16 (2013), p. 162001. URL: http://stacks.iop.org/1742-6596/425/i=16/a=162001.

[35] G. Materlik and Th. Tschentscher. *TESLA Technical Design Report*. Mar. 2001. URL: http://tesla.desy.de/new_pages/TDR_CD/PartV/xfel.pdf.

[36] Wikipedia. *Voigt profile*. [Online; accessed 23-February-2017]. 2016. URL: https://en.wikipedia.org/w/index.php?title=Voigt_profile&oldid=753555013.

[37] Matt Newville et al. *lmfit-py 0.9.5*. July 2016. DOI: 10.5281/zenodo.58759. URL: https://doi.org/10.5281/zenodo.58759.

[38] Kane Yee. "Numerical solution of initial boundary value problems involving maxwell's equations in isotropic media". In: *IEEE Transactions on Antennas and Propagation* 14.3 (May 1966), pp. 302–307. ISSN: 0018-926X. DOI: 10.1109/TAP.1966.1138693.

[39] Gary Cohen et al. “Efficient Mixed Finite Elements for Maxwell’S Equations in Time Domain”. In: *Ultra-Wideband, Short-Pulse Electromagnetics 6*. Boston, MA: Springer US, 2003, pp. 251–259. ISBN: 978-1-4419-9146-1. DOI: 10.1007/978-1-4419-9146-1_22. URL: http://dx.doi.org/10.1007/978-1-4419-9146-1_22.

[40] Samoylova L et al. “WavePropaGator: interactive framework for X-ray free-electron laser optics design and simulations”. In: *Journal of Applied Crystallography* 49.4 (2016), pp. 1347–1355. DOI: doi:10.1107/S160057671600995X.

[41] S. Roling et al. “Time-dependent wave front propagation simulation of a hard x-ray split-and-delay unit: Towards a measurement of the temporal coherence properties of x-ray free electron lasers”. In: *Phys. Rev. ST Accel. Beams* 17 (11 Nov. 2014), p. 110705. DOI: 10.1103/PhysRevSTAB.17.110705. URL: http://link.aps.org/doi/10.1103/PhysRevSTAB.17.110705.

[42] G.M. Muslu and H.A. Erbay. “Higher-order split-step Fourier schemes for the generalized nonlinear Schrödinger equation”. In: *Mathematics and Computers in Simulation* 67.6 (2005), pp. 581–595. ISSN: 0378-4754. DOI: http://dx.doi.org/10.1016/j.matcom.2004.08.002. URL: http://www.sciencedirect.com/science/article/pii/S037847540400254X.

[43] Wikipedia. *Vector calculus identities*. [Online; accessed 20-February-2017]. 2016. URL: https://en.wikipedia.org/w/index.php?title=Vector_calculus_identities&oldid=753562955.

[44] Wikipedia. *Fourier transform*. [Online; accessed 30-March-2016]. 2016. URL: https://en.wikipedia.org/w/index.php?title=Fourier_transform&oldid=712371627.

Part III

APPENDIX

11

ADDITIONAL DERIVATIONS

11.1 The parabolic wave equation

In literature it is common to work with the parabolic wave equation, which was introduced by Leontovich and Fock in the late 1940s for radio wave propagation and has since been used for the study of many X-ray optical problems [10]. It is derived from the scalar Helmholtz equation

$$\Delta\psi + k^2 n^2 \psi = 0, \tag{2.6}$$

by approximating $\psi = u\, e^{-ikz}$, where u is a slowly varying envelope with the condition

$$\left|\frac{\partial^2 u}{\partial z^2}\right| \ll \left|k\frac{\partial u}{\partial z}\right|.$$

Inserting this into the Helmholtz equation and neglecting the second derivative, we find the parabolic wave equation

$$\frac{\partial u}{\partial z} = \frac{1}{2ik}\,\Delta_\perp u - ik\,\frac{n^2-1}{2}\,u. \tag{11.1}$$

It may also be derived by inserting the envelope into the paraxial Helmholtz equation (2.16)

$$\frac{\partial u}{\partial z} = \frac{1}{2ikn}\,\Delta_\perp u - ik(n-1)\,u$$

and approximating $\frac{1}{n} \approx 1$ and $n - 1 \approx (n^2-1)/2$, which is valid when $n \approx 1$. This assumption was implicitly implemented through the envelope approximation, as we can also verify by regarding the Beer–Lambert law $\psi \sim \exp(-iknz)$, which does not fulfill the envelope approximation for large n. It is therfore a solution of the paraxial Helmholtz equation (2.16), but not the parabolic wave equation (11.1). We therefore believe that the paraxial Helmholtz equation has enhanced validity for paraxial beams, as it requires no additional constraints for the refractive index.

11.2 Three dimensional finite differences

We regard the three dimensional partial differential equation

$$\frac{\partial u}{\partial z} = A\,\frac{\partial^2 u}{\partial x^2} + B\,\frac{\partial^2 u}{\partial y^2} + C\,u$$

with position dependent $A, B, C \in \mathbb{C}$. In three dimensions the Crank-Nicolson scheme will not result in a tridiagonal system and the algorithm will therefore become comparably slow. Instead, we choose a two-step implicit alternating-direction scheme that gives us linear time complexity in N_x and N_y and second order accuracy in Δx, Δy and Δz as well as unconditional stability [10]. The algorithm consists of two steps in which the partial derivatives with respect to x and y are alternately evaluated implicitly and explicitly. For each step the equation is approximated by a finite difference equation

$$I. \qquad \frac{u_{i,j}^{k+\frac{1}{2}} - u_{i,j}^{k}}{\Delta z/2} = A_{i,j}^{k+\frac{1}{2}}\,\frac{\partial_x^2 u_{i,j}^{k+\frac{1}{2}}}{\Delta x^2} + B_{i,j}^{k}\,\frac{\partial_y^2 u_{i,j}^{k}}{\Delta y^2} + \frac{C_{i,j}^{k} u_{i,j}^{k} + C_{i,j}^{k+\frac{1}{2}} u_{i,j}^{k+\frac{1}{2}}}{2}$$

$$II. \qquad \frac{u_{i,j}^{k+1} - u_{i,j}^{k+\frac{1}{2}}}{\Delta z/2} = A_{i,j}^{k+\frac{1}{2}}\,\frac{\partial_x^2 u_{i,j}^{k+\frac{1}{2}}}{\Delta x^2} + B_{i,j}^{k+1}\,\frac{\partial_y^2 u_{i,j}^{k+1}}{\Delta y^2} + \frac{C_{i,j}^{k+\frac{1}{2}} u_{i,j}^{k+\frac{1}{2}} + C_{i,j}^{k+1} u_{i,j}^{k+1}}{2}$$

where

$$\partial_x^2 u_{i,j}^{k} := u_{i-1,j}^{k} - 2u_{i,j}^{k} + u_{i+1,j}^{k}, \qquad \partial_y^2 u_{i,j}^{k} := u_{i,j-1}^{k} - 2u_{i,j}^{k} + u_{i,j+1}^{k},$$

Similar as in the 2D case, reordering terms and introducing the auxiliary variables $a_{i,j}^{k} = \frac{A_{i,j}^{k}}{2}\frac{\Delta z}{\Delta x^2}$, $b_{i,j}^{k} = \frac{B_{i,j}^{k}}{2}\frac{\Delta z}{\Delta y^2}$ and $c_{i,j}^{k} = \frac{\Delta z}{4} C_{i,j}^{k}$ transforms step $I.$ to

$$(1 - a_{i,j}^{k+\frac{1}{2}}\partial_x^2 - b_{i,j}^{k+\frac{1}{2}})u_i^{k+\frac{1}{2}} = (1 + b_{i,j}^{k}\partial_y^2 + c_{i,j}^{k})u_i^{k}. \tag{11.2}$$

Taking into account the edge boundary conditions $u_{0,j}^{k}$, $u_{N_x+1,j}^{k}$,$u_{i,0}^{k}$, u_{i,N_y+1}^{k} we obtain N_y systems of linear equations

$$M_j^{k+\frac{1}{2}}\,\boldsymbol{u}_j^{k+\frac{1}{2}} = \boldsymbol{d}_j^{k}, \tag{11.3}$$

where $\left(\boldsymbol{u}_j^{k}\right)_i = u_{i,j}^{k}$, M_j^{k} is the $N_x \times N_x$ tridiagonal matrix

$$M_j^{k} = \begin{pmatrix} (1 + 2\,a_{1,j}^{k} - c_{1,j}^{k}) & -a_{1,j}^{k} & 0 & \\ -a_{2,j}^{k} & (1 + 2\,a_{2,j}^{k} - c_{2,j}^{k}) & -a_{2,j}^{k} & \dots \\ 0 & -a_{3,j}^{k} & (1 + 2\,a_{3,j}^{k} - c_{3,j}^{k}) & \\ & \vdots & & \ddots \end{pmatrix},$$

and d_j^k is a vector containing the right side of equation Eq. (11.3) and the edge boundary conditions

$$\boldsymbol{d}_j^k = \begin{pmatrix} (1 + b_{1,j}^k \partial_y^2 + c_{1,j}^k)\, u_{1,j}^k + a_{0,j}^{k+\frac{1}{2}}\, u_{0,j}^{k+\frac{1}{2}} \\ (1 + b_{2,j}^k \partial_y^2 + c_{2,j}^k)\, u_{2,j}^k \\ \vdots \\ (1 + b_{N_x-1,j}^k \partial_y^2 + c_{N_x-1,j}^k)\, u_{N_x-1,j}^k \\ (1 + b_{N_x,j}^k \partial_y^2 + c_{N_x,j}^k)\, u_{N_x,j}^k + a_{N_x+1,j}^{k+\frac{1}{2}}\, u_{N_x+1,j}^{k+\frac{1}{2}} \end{pmatrix}.$$

Each of these N_y tridiagonal systems can be solved in $\mathcal{O}(N_x)$ operations. For step II. we can deviate a similar group of N_x tridiagonal systems of size $N_y \times N_y$ resulting in $\mathcal{O}(N_x\ N_y)$ total time complexity.

12

MATHEMATICAL DETAILS

12.1 Identities

List of identities used in this work. Sources: [43, 44].

12.1.1 Vector calculus identities

For complex vector fields $\vec{A}, \vec{B} : \mathbb{R}^3 \to \mathbb{C}^3$ the following identities hold.

V1: $\vec{\nabla} \cdot \left(\vec{\nabla} \times \vec{A}\right) = 0.$

V2: $\vec{\nabla} \times \left(\vec{A} \times \vec{B}\right) = \vec{A}\left(\vec{\nabla} \cdot \vec{B}\right) - \vec{B}\left(\vec{\nabla} \cdot \vec{A}\right) + \left(\vec{B} \cdot \vec{\nabla}\right)\vec{A} - \left(\vec{A} \cdot \vec{\nabla}\right)\vec{B}.$

V3: $\vec{\nabla} \times \left(\psi \vec{A}\right) = \psi(\vec{\nabla} \times \vec{A}) + (\vec{\nabla}\psi) \times \vec{A}.$

V4: $\vec{\nabla} \times \left(\vec{A} \times \vec{B}\right) = \vec{A}\left(\vec{\nabla} \cdot \vec{B}\right) - \vec{B}\left(\vec{\nabla} \cdot \vec{A}\right) + \left(\vec{B} \cdot \vec{\nabla}\right)\vec{A} - \left(\vec{A} \cdot \vec{\nabla}\right)\vec{B}.$

12.1.2 Fourier identities

For integrable functions $f(x), g(x) : \mathbb{R} \to \mathbb{C}$, $a \in \mathbb{R}$ and $\alpha \in \mathbb{C}$ the following identities hold.

F1: $\mathcal{F}_x^{-1}[f(x)](k) = \mathcal{F}_x[f(x)](-k).$

F2: $\mathcal{F}_x[f(x)\, e^{iax}](k) = \mathcal{F}_x[f(x)](k-a).$

F3: $\mathcal{F}_x[f(x+a)](k) = \mathcal{F}_x[f(x)](k)\, \exp(iak).$

F4: $\mathcal{F}_x[e^{-\alpha x^2}](k) = \frac{1}{\sqrt{2\alpha}}\, e^{-k^2/4\alpha}$, when $\Re(\alpha) > 0.$

F5: $\mathcal{F}_x[f(x)\, g(x)](k) = \frac{1}{\sqrt{2\pi}}\, \mathcal{F}_x[f](k) * \mathcal{F}_x[g](k).$

F6: $\mathcal{F}_x[\frac{\partial f(x)}{\partial x}](k) = ik\mathcal{F}_x[g](k).$

12.2 Calculations

12.2.1 Scalar Potential

Lemma 12.1. *Assuming Eq.* (2.2) *and Eq.* (2.3) *and Eq.* (2.4) *are valid and given the electromagnetic potential* $\vec{\psi} = \psi\,\vec{e}_p$*, the electric field is* $\hat{\vec{E}} = \frac{i}{k}\,(\vec{\nabla}\psi) \times \vec{e}_p$ *and the magnetic field is* $\hat{\vec{B}} = \frac{1}{c}\left(\frac{1}{k^2}\,\vec{\nabla}(\vec{e}_p \cdot \vec{\nabla}\psi) + n^2\psi\,\vec{e}_p\right)$.

Proof. The electric field follows directly from Eq. (2.5) using the vector calculus identity V3

$$\begin{aligned}\hat{\vec{E}} &= \frac{i}{k}\,\vec{\nabla} \times (\psi\vec{e}_p)\\ &= \frac{i}{k}\,((\vec{\nabla}\psi) \times \vec{e}_p + \psi\,(\vec{\nabla} \times \vec{e}_p))\\ &= \frac{i}{k}\,(\vec{\nabla}\psi) \times \vec{e}_p,\end{aligned}$$

while the magnetic field then follows from Eq. (2.4) using the vector calculus identity V4

$$\begin{aligned}\hat{\vec{B}} &= \frac{1}{\omega k}\,\vec{\nabla} \times ((\vec{\nabla}\psi) \times \vec{e}_p)\\ &= \frac{1}{\omega k}\,((\vec{e}_p \cdot \vec{\nabla})\vec{\nabla}\psi - \Delta\psi)\\ &= \frac{1}{c}\left(\frac{1}{k^2}\,\vec{\nabla}(\vec{e}_p \cdot \vec{\nabla}\psi) + n^2\psi\,\vec{e}_p\right).\end{aligned}$$

□

12.2.2 Poynting vector

Lemma 12.2. *The time-average of the product of two* cos *functions with positive non-zero frequencies* ω_1, ω_2 *and phase-shift* $\varphi \in \mathbb{R}$ *is* $\langle \cos(\omega_1 t)\cos(\omega_2 t + \varphi)\rangle = \mathbf{1}_{\{\omega_1=\omega_2\}}\cos(\varphi)/2$*, where* $\mathbf{1}_A : \Omega \to \{0,1\}$ *is the indicator function of the subset* $A \subset \Omega$*.*

Proof. To begin, note that for any frequency ω, the time-average of $\cos(\omega t + \varphi)$ is

$$\begin{aligned}\langle \cos(\omega t + \varphi)\rangle &= \lim_{T\to\infty}\frac{1}{T}\int_0^T \cos(\omega t + \varphi)\,\mathrm{d}t\\ &= \begin{cases}\cos(\varphi), & \text{if } \omega = 0\\ \lim_{T\to\infty}\frac{1}{\omega T}\,\sin(\omega T + \varphi), & \text{otherwise}\end{cases} = \cos(\varphi)\,\mathbf{1}_{\{\omega=0\}}.\end{aligned}$$

Now we see that

$$\begin{aligned}\langle \cos(\omega_1 t)\cos(\omega_2 t + \varphi)\rangle &= \lim_{T\to\infty}\frac{1}{T}\int_0^T \cos(\omega_1 t)\cos(\omega_2 t + \varphi)\,\mathrm{d}t\\ &= \lim_{T\to\infty}\frac{1}{2T}\int_0^T (\cos(t\,(\omega_1 - \omega_2) - \varphi) + \cos(t\,(\omega_1 + \omega_2) + \varphi))\,\mathrm{d}t\\ &= \frac{1}{2}(\langle\cos(t\,(\omega_1 - \omega_2) - \varphi)\rangle + \langle\cos(t\,(\omega_1 + \omega_2) + \varphi)\rangle)\\ &= \frac{\cos(\varphi)}{2}\,\left(\mathbf{1}_{\{\omega_1=\omega_2\}} + \mathbf{1}_{\{\omega_1=-\omega_2\}}\right).\end{aligned}$$

The claim follows for positive, non-zero frequencies. □

Lemma 12.3. *For any differentiable function* $\psi : \mathbb{R}^3 \to \mathbb{C}$ *the identity* $\Re(\psi)\,\Im(\vec{\nabla}\psi) - \Im(\psi)\,\Re(\vec{\nabla}\psi) = |\psi|^2\,\vec{\nabla}\arg(\psi)$ *holds.*

Proof. We begin by writing $\psi = A\exp(i\varphi)$ with the real-valued functions $A = |\psi|$ and $\varphi = \arg(\psi)$. Then the gradient of ψ is $\vec{\nabla}\psi = (\vec{\nabla}A + iA\vec{\nabla}\varphi)\,e^{i\varphi}$ with the real part $\Re(\vec{\nabla}\psi) = \cos(\varphi)\vec{\nabla}A - \sin(\varphi)A\vec{\nabla}\varphi$ and imaginary part $\Im(\vec{\nabla}\psi) = \sin(\varphi)\vec{\nabla}A + \cos(\varphi)A\vec{\nabla}\varphi$. Inserting this into the original statement eliminates the $\vec{\nabla}A$ term and leaves

$$\begin{aligned}\Re(\psi)\,\Im(\vec{\nabla}\psi) - \Im(\psi)\,\Re(\vec{\nabla}\psi) &= A\,(\cos(\varphi)\,\Im(\vec{\nabla}\psi) - \sin(\varphi)\,\Re(\vec{\nabla}\psi))\\ &= A^2\,\vec{\nabla}\varphi,\end{aligned}$$

which proves the claim. □

12.2.3 Paraxial Helmholtz equation

Lemma 12.4. *For* $n \in \mathbb{N}$, $i \in \{1..n\}$, $A_i, F \in \mathbb{C}$ *and* $\Re(A_i) > 0$, *the partial differential equation* $\partial_z u = F\,u + \sum_{i=1}^n A_i\,\partial_{x_i}^2 u$ *is solved by* $u = e^{Fz}\,\prod_{i=1}^n \frac{1}{\sqrt{2A_i z}}\exp\!\left(-\frac{x_i^2}{4A_i z}\right)$.

Proof. By taking the Fourier transform of u in the perpendicular directions $\tilde{u} = \mathcal{F}_{x_1,x_2,\dots}[u](k_1,k_2,\dots)$, the differential equation becomes an ordinary differential equation $\partial_z\tilde{u} = (F + \sum_{i=1}^n A_i k_i^2)\,\tilde{u}$. With the exponential ansatz we immediately find the solution $\tilde{u} = \exp(F\,z - \sum_i A_i k_i^2\,z)$. The solution in real space can then be obtained by applying the Fourier identity F4: $u = e^{Fz}\,\prod_{i=1}^n \frac{1}{\sqrt{2A_i z}}\exp\!\left(-\frac{x_i^2}{4A_i z}\right)$. □

Lemma 12.5. *For* $\mathcal{U} := e^{Fz}\,\prod_{i=1}^n \frac{1}{\sqrt{4\pi A_i z}}\exp\!\left(-\frac{x_i^2}{4A_i z}\right)$ *where* $n \in \mathbb{N}$, $i \in \{1..n\}$, $A_i, F \in \mathbb{C}$ *and* $\Re(A_i) > 0$, *it follows that* $\lim_{z\to 0}\mathcal{U} = \delta(\vec{x}_\perp)$, *where* $(\vec{x}_\perp)_i = x_i$.

Proof. We define $g_z(x) := \exp(-x^2/4Az)/\sqrt{4\pi Az}$. Then, for any bounded contiguous function f,

$$\begin{aligned}\lim_{z\to 0}\int_{\mathbb{R}} f(x)\,g_z(x)\,\mathrm{d}x &= \lim_{z\to 0}\frac{1}{\sqrt{A\pi}}\int_{\mathbb{R}} f(\sqrt{4z}\,u)\,\exp(-u^2/A)\,\mathrm{d}u\\ &= \frac{1}{\sqrt{A\pi}}\int_{\mathbb{R}} f(0)\,\exp(-u^2/A)\,\mathrm{d}u\\ &= f(0) = \int_{\mathbb{R}} f(x)\,\delta(x)\,\mathrm{d}x,\end{aligned}$$

where we used the monotone convergence theorem in the second step. This result implies weak convergence $g_z \xrightarrow{w} \delta$. The claim follows from the product distribution $\lim_{z\to 0}\mathcal{U}(\vec{x}_\perp, z) = \lim_{z\to 0}\prod_{i=1}^n g_z(x_i) \xrightarrow{w} \prod_{i=1}^n \delta(x_i) = \delta(\vec{x}_\perp)$. □

Corollary 12.1 (Fresnel Propagator)**.** *From lemma 12.4 and 12.5 directly follows that for $A_i, F \in \mathbb{C}$ and $\Re(A_i) > 0$ a solution of the $n+1$ dimensional partial differential equation $\partial_z u = F\,u + \sum_{i=1}^{n} A_i\,\partial_{x_i}^2 u$, given the boundary condition $u(\vec{x}_\perp, 0)$, is*

$$\begin{aligned} u(\vec{x}_\perp, z) &= \int_{\mathbb{R}^n} u(\vec{x}_\perp - \vec{r}, 0) \cdot \mathcal{U}(\vec{r}, z)\; d\vec{r} \\ &:= u(\vec{x}_\perp, 0) * \mathcal{U}(\vec{x}_\perp, z). \end{aligned}$$

Corollary 12.2 (n-Dimensional Gaussian Beam)**.** *Using corollary 12.1 we can derive a class of solutions of the differential equation characterized by a Gaussian intensity profile. Inserting the initial condition $u(\vec{x}_\perp, 0) = \prod_{i=1}^{n} \exp(-x_i^2/2\sigma_i^2)$ with $\sigma_i \in \mathbb{R}^+$ we find the solution*

$$\begin{aligned} u(\vec{x}_\perp, z) &= u(\vec{x}_\perp, 0) * \mathcal{U}(\vec{x}_\perp, z) \\ &= e^{Fz} \prod_{i=1}^{n} \int_{\mathbb{R}} \frac{1}{\sqrt{4\pi A_i z}} \exp\left(-\frac{r_i^2}{4A_i z} - \frac{(r_i - x_i)^2}{2\sigma_i^2} \right) dr_i \\ &= e^{Fz} \prod_{i=1}^{n} \frac{\sigma_i}{w_i(z)} \exp\left(-\frac{{x_i}^2}{2w_i(z)^2} \right), \end{aligned}$$

where $w_i(z) = \sqrt{2\,A_i\,z + \sigma_i^2}$ is the complex waist size. The actual waist size on the x_i axis is determined by the real part of the exponential argument

$$\Re\left(\frac{1}{w_i(z)^2} \right)^{-1/2} = \sqrt{\frac{4\,z^2\,|A_i|^2 + 4\,\Re(A_i)\,z\,\sigma_i^2 + \sigma_i^4}{2\,\Re(A_i)\,z + \sigma_i^2}}.$$

13

ADDITIONAL FIGURES

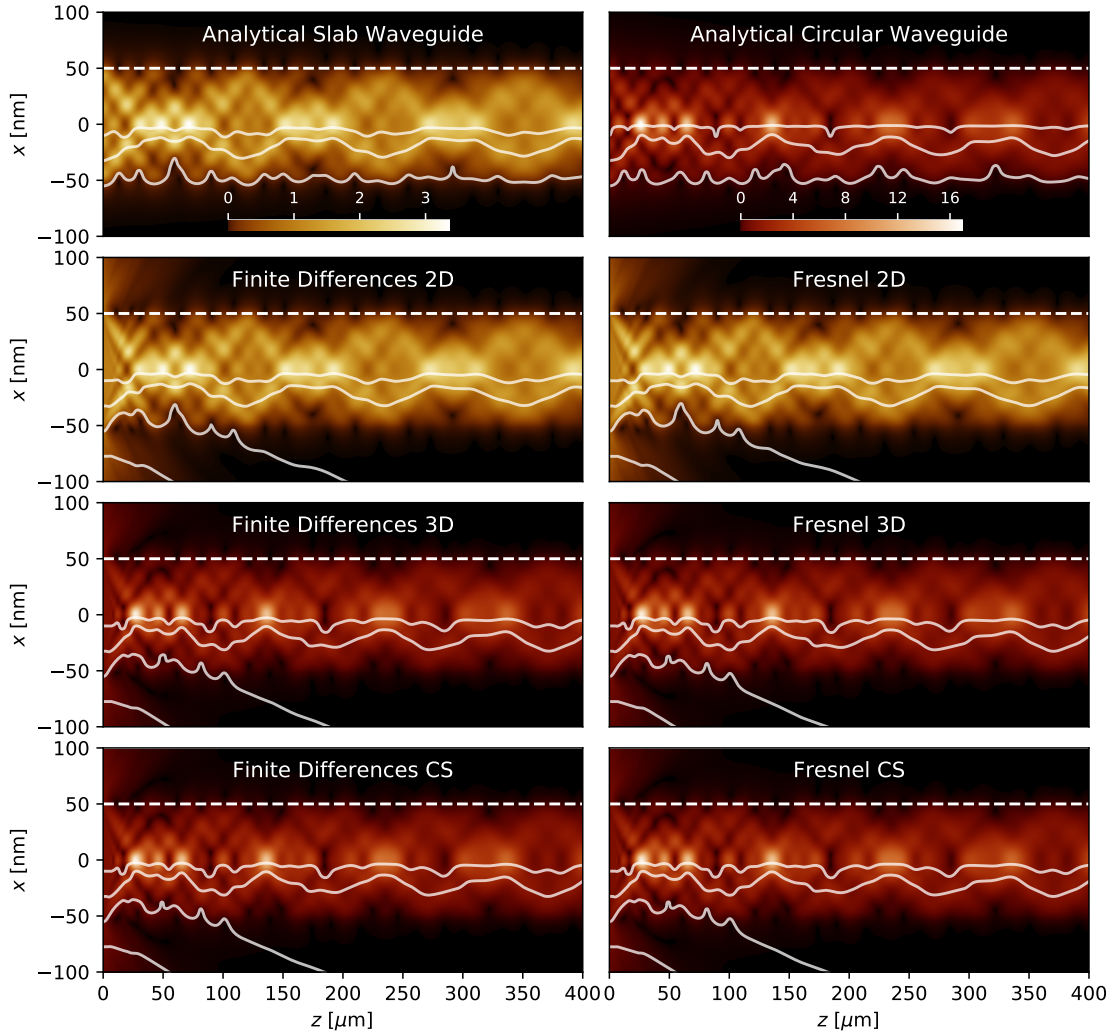

Figure 13.1: Propagator validation with germanium slab and cylindrical waveguides with 100 nm diameter and vacuum core. Normalized intensity is color coded and streamlines indicate the direction of the time-averaged Poynting vector. The white dashed line indicates the upper edge of the waveguides. The top row shows the analytical solutions in which only guided modes are regarded. The script to create this figure is Code 14.6.

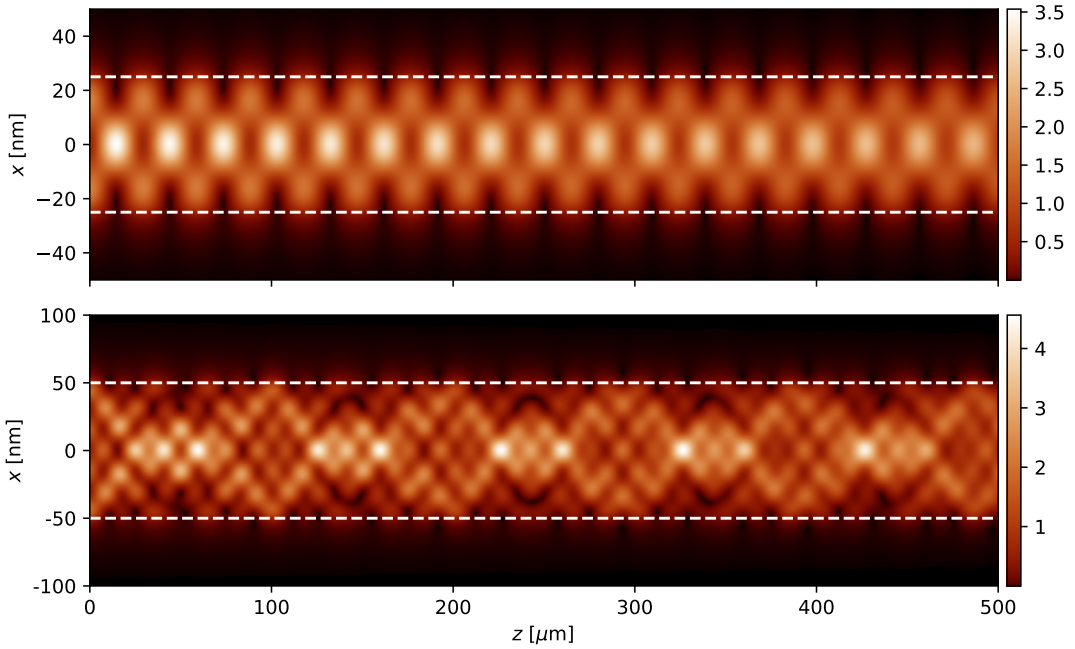

Figure 13.2: Analytical solution for the field inside germanium slab waveguides with Vacuum core and 25 nm (top) and 50 nm (bottom) diameter for an incident plane wave at 10 keV photon energy. The intensity is normalized to an incident intensity of 1 and color coded. The edge of the waveguides is indicated by a white dashed lines. The script to create this figure is Code 14.4.

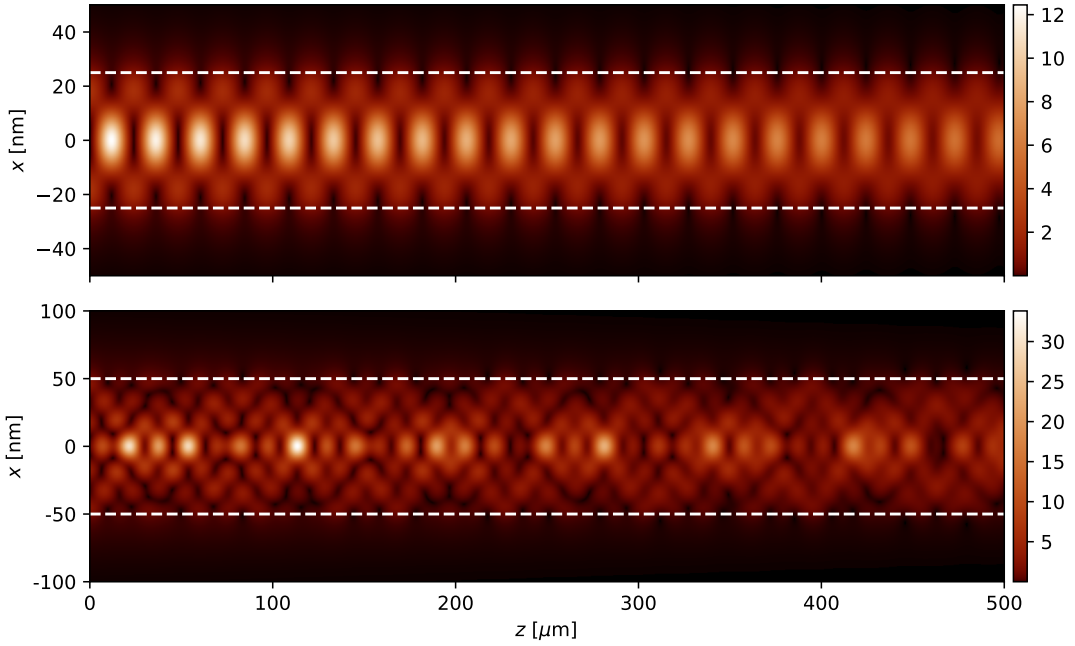

Figure 13.3: Analytical solution for the field inside germanium cylindrical waveguides with Vacuum core with 25 nm (top) and 50 nm (bottom) diameter for an incident plane wave at 10 keV photon energy. The edges of the waveguides is indicated by white dashed lines. The script to create this figure is Code 14.4.

14

CODE

The numerical algorithms presented and used in this work are implemented in the PyPropagate framework, which consists of around 3800 lines of Python and C++ code. The source code is publicly available online at `https://github.com/TheLartians/PyPropagate`. Images and Simulations in this work have been created by the scripts below. Note about half of the code is needed to defines and run the simulations, while the other half is needed to create the figures.

Code Listing 14.1: Script to plot convolution kernels with streamlines.

```
from pypropagate import *
import matplotlib.pyplot as plt
from matplotlib.colors import LogNorm
from pypropagate.colormaps import fire_colormap

# Create settings
settings = presets.settings.create_paraxial_wave_equation_settings()
s = settings.symbols
settings.wave_equation.set_energy(1*units.eV)

# Define simulation box
s.xmin = -5*units.um
s.xmax =  5*units.um
s.zmin = -14*units.um
s.zmax =  14*units.um

s.Nx = 2**13
s.Nz = 2**13

# Define Kernels
r = pc.Symbol('r')
K1 = -s.z/(2*pc.pi*r)*pc.derivative(pc.exp(1j*s.k*r)/r,r).evaluate()
K2 = 1j*s.k/(2*pc.pi*s.z) * pc.exp(-1j*s.k*(s.z+s.x**2/(2*s.z)))

# Plot the kernels
fig,(ax1,ax2) = plt.subplots(2,1,figsize=(8,5),sharex=True)
K1field = expression_to_array(K1.subs(r,- pc.sign(s.z) * pc.sqrt(s.x**2+s.z**2)) / s.k**2 * pc.exp(-1j*pc.pi
    ),settings)
K2field = expression_to_array(K2/ s.k**2,settings)
img = plot(abs(K1field),ax=ax1,norm=LogNorm(10**-4),vmax=1,cmap='magma')
support_1 = s.x**2+s.z**2 > units.um**2
poynting_streamplot(K1field,0,ax=ax1,color=(1,1,1,0.75),settings=settings,support=support_1,density=1)
ax1.set_xlabel('')
ax1.set_ylabel(r'$r\;[\mu\mathrm{m}]$')
img = plot(abs(K2field),ax=ax2,norm=LogNorm(10**-4),vmax=1,cmap='magma')
support_2 = support_1 & (abs(s.z) > 0.2 * units.um)
poynting_streamplot(K2field,0,ax=ax2,color=(1,1,1,0.75),settings=settings,support=support_2,density=1)
ax2.set_ylabel(r'$r\;[\mu\mathrm{m}]$')
cbar_ax = fig.add_axes([0.92, 0.13, 0.015, 0.75])
cbar = fig.colorbar(img, cax=cbar_ax)
cbar.set_label('$|\mathcal{P}| \; [k^2] \hspace{11} |\mathcal{T}| \; [k^2]$')

plt.savefig('plots/propagation/kernels.pdf',bbox_extra_artists=(cbar_ax,), bbox_inches='tight',transparent=
    True)
```

Code Listing 14.2: Script to create analytical Gaussian beam plot.

```
from pypropagate import *
from pypropagate.colormaps import fire_colormap
import matplotlib.pyplot as plt

# Create settings
settings = presets.settings.create_paraxial_wave_equation_settings()
s = settings.symbols

# Define simulation box
s.xmin = -5*units.um
s.xmax =  5*units.um
s.zmin = -14*units.um
s.zmax =  14*units.um
s.Nx = 2**10
s.Nz = 2**10

# Define gaussian beam
r = pc.Symbol('r')
sr = pc.Symbol('sigma_r')
w = pc.sqrt(s.z/(1j*s.k) + sr**2)
gauss = pc.exp(-1j*s.k*s.z) * sr**2/w**2 * pc.exp(-r**2/(2*w**2))
settings.wave_equation.set_energy(1*units.eV)

# Plot the beam
field = expression_to_array(gauss.subs([(sr,1*units.um),(r,s.x)]),settings)
fig,(ax) = plt.subplots(1,1,figsize=(8,3))
uimg = plot(abs(field)**2,ax=ax,vmax=1,cmap=fire_colormap())
poynting_streamplot_with_start_points(field,0,[[s.zmin,i*s.xmin] for i in np.linspace(0,1,5)],ax=ax,color=(1
    ,1,1,0.75),arrowsize=5,arrowpositions=[0.1,0.9])
plt.colorbar(uimg)
plt.ylabel('$r\;[\mu\mathrm{m}]$')
plt.tight_layout()

plt.savefig('plots/elementary_solutions/gaussian.pdf',transparent=True)
```

Code Listing 14.3: Script to calculate analytical slab and circular waveguide solutions.

```
def analytical_slab_waveguide(settings):
    from pypropagate.coordinate_ndarray import CoordinateNDArray
    import expresso.pycas as pc
    from pypropagate import expression_to_array
    import scipy.optimize
    import scipy.integrate
    from scipy.special import j0,j1,k0,k1
    import numpy as np
    from numpy import sqrt,ceil,pi,sign,cos,sin,exp,tan,sum,abs
    import warnings

    s = settings.symbols
    r = settings.waveguide.r
    k = float(settings.get_numeric(s.k*r))
    n1 = settings.get_as(settings.waveguide.n_1,complex)
    n2 = settings.get_as(settings.waveguide.n_2,complex)
    d  = 2
    dx = settings.get_as(s.dx/r,float)

    # We will determine um for all guided modes as the roots of the characteristic equation:
    def Char(kappa):
        gamma = sqrt((n1.real**2-n2.real**2)*k**2-kappa**2)
        return kappa*tan(kappa*d/2) - gamma
    kappa_max = sqrt(n1.real**2-n2.real**2)*k

    # maximum number of guided modes
    N = ceil(d*kappa_max/(2*pi))

    # Now find roots of the characteristic equation by searching intervals
    kappa_values = []
    segments = N

    with warnings.catch_warnings():
        warnings.simplefilter("ignore")
        while len(kappa_values) < N:
            segments = segments*2
            if segments>10000:
                break
            kappa_values = []
            xsegs = np.linspace(0,kappa_max,segments)
            for interval in [(xi,xj) for xi,xj in zip(xsegs[1:],xsegs[:-1])]:
                va,vb = Char(np.array(interval))
                # Check if values for interval boundaries are finite and have opposite sign
```

```
                if np.isfinite(va) and np.isfinite(vb) and sign(va) != sign(vb):
                    # There might be a root in the interval. Find it using Brent's method!
                    kappa,res = scipy.optimize.brentq(Char,interval[0],interval[1],full_output=True)
                    # Check that point converged and value is small (we might have converged to a pole
                        instead)
                    if kappa!=0 and res.converged and abs(Char(kappa))<1:
                        kappa_values.append(kappa)
    kappa_values = np.array(kappa_values)

    # Define the guided modes
    def psi(kappa,r):
        gamma = sqrt((n1.real**2-n2.real**2)*k**2-kappa**2)
        return cos(kappa*r)*(r*2<=d) + cos(kappa*d/2) * exp(-gamma*(r-d/2))*(r*2>d)

    # Normalize the modes by integrating the intensity of the guided modes over R
    B_values = [(2*scipy.integrate.quad(lambda r:abs(psi(kappa,r))**2,0,np.inf)[0])**(-0.5) for kappa in
        kappa_values]

    #Project the modes onto the symmetrical initial condition
    u0 = pc.numpyfy(settings.get_numeric(s.u0.subs(s.y,0).subs(s.x,s.x*r)),restype=float)
    c_values = [2*B**2*scipy.integrate.quad(lambda r:psi(kappa,r)*u0(x=r),0,np.inf)[0] for kappa,B in zip(
        kappa_values,B_values)]

    # Determine attenuation coeffcients
    mu_values = np.array([2*B**2*k*
            (scipy.integrate.quad(lambda r:abs(psi(kappa,r))**2*n1.imag,0,d/2)[0]    +
             scipy.integrate.quad(lambda r:abs(psi(kappa,r))**2*n2.imag,d/2,np.inf)[0])
                        for kappa,B in zip(kappa_values,B_values)])

    # The full solution is the superposition of the guided modes
    def field(x,r):
        solution = None
        for c,b,kappa,mu in zip(c_values,B_values,kappa_values,mu_values):
            beta = sqrt(k**2*n1.real**2 - kappa**2)
            mode = c * psi(kappa,r) * exp((mu+1j*(beta-k))*x)
            if solution is None: solution = mode
            else: solution += mode
        return solution

    linspace = lambda a,b,N: np.linspace(a,b,int(N))
    x_values = linspace(*settings.get_as((s.zmin/r,s.zmax/r,s.Nz),float))
    r_values = abs(linspace(*settings.get_as((s.xmin/r,s.xmax/r,s.Nx),float)))
    data = np.conjugate(field(*np.meshgrid(x_values,r_values)) )
    sx = settings.get_numeric(s.sx)
    sz = settings.get_numeric(s.sz)
    res = CoordinateNDArray(data,[(-sx/2,sx/2),(0,sz)],(s.x,s.z),settings.get_numeric_transform())
    return res


def analytical_circular_waveguide(settings):
    from pypropagate.coordinate_ndarray import CoordinateNDArray
    import expresso.pycas as pc
    import warnings
    import scipy.optimize
    import scipy.integrate
    from scipy.special import j0,j1,k0,k1
    import numpy as np
    from numpy import sqrt,ceil,pi,sign,cos,sin,exp,sum,abs

    s = settings.symbols
    r = settings.waveguide.r
    dx = float(settings.get_numeric(s.dx/r))
    kn = float(settings.get_numeric(s.k*r))
    n1 = settings.get_as(settings.waveguide.n_1,complex)
    n2 = settings.get_as(settings.waveguide.n_2,complex)
    R  = 1

    # We will determine um for all guided modes as the roots of the characteristic equation:
    def char(kappa):
        gamma = sqrt(kn**2*(n1.real**2 - n2.real**2) - kappa**2)
        return gamma*k1(gamma*R)*j0(kappa*R) - kappa*j1(kappa*R)*k0(gamma*R)

    kappa_max = kn * sqrt(n1.real**2-n2.real**2)

    # Dimensionless waveguide paramter
    V = kn*R*sqrt(n1.real**2 - n2.real**2)

    # V determines the number of guided modes
    N = ceil(V/pi)

    # Now find all N roots of the characteristic equation
    kappa_values = []
```

```
    segments = N

    with warnings.catch_warnings():
        warnings.simplefilter("ignore")

        while len(kappa_values) < N and segments < 10000:
            segments = segments*2
            kappa_values = []
            xsegs = np.linspace(0,kappa_max,segments)
            for interval in [(xi,xj) for xi,xj in zip(xsegs[1:],xsegs[:-1])]:
                va,vb = char(np.array(interval))
                # Check if values for interval boundaries are finite and have opposite sign
                if np.isfinite(va) and np.isfinite(vb) and sign(va) != sign(vb):
                    # There might be a root in the interval. Find it using Brent's method!
                    kappa,res = scipy.optimize.brentq(char,interval[0],interval[1],full_output=True)
                    # Check that point converged and value is small (we might have converged to a pole
                         instead)
                    if kappa!=0 and res.converged and abs(char(kappa))<1:
                        kappa_values.append(kappa)
        # Reactivate warnings
        warnings.resetwarnings()

    kappa_values = np.array(kappa_values)

    # Define the guided modes
    def psi(kappa,r):
        gamma = sqrt(kn**2*(n1.real**2 - n2.real**2) - kappa**2)
        return np.piecewise(r,[r<R,r>=R],[lambda r:j0(kappa*r),lambda r:j0(kappa*R)*k0(gamma*r)/k0(gamma*R)]
             )

    # Normalize the modes by integrating the intensity of the guided modes over R^2
    B_values = [(2*pi*scipy.integrate.quad(lambda r:abs(psi(kappa,r))**2*r,0,R)[0]
                + (2*pi*scipy.integrate.quad(lambda r:abs(psi(kappa,r))**2*r,R,np.inf)[0]))**(-0.5) for
                     kappa in kappa_values]

    #Project the modes onto the symmetrical initial condition
    u0 = pc.numpyfy(settings.get_numeric(s.u0.subs(s.y,0).subs(s.x,s.x*r)),restype=float)
    c_values = [B**2*2*pi*(scipy.integrate.quad(lambda r:psi(kappa,r)*u0(x=r)*r,0,R)[0]
                +scipy.integrate.quad(lambda r:psi(kappa,r)*u0(x=r)*r,R,np.inf)[0]) for kappa,B in zip(
                     kappa_values,B_values)]

    # Integrate mu
    mu_values = np.array([
            2*pi*scipy.integrate.quad(lambda r:abs(psi(kappa,r)*B)**2*kn*n1.imag*r,0,R)[0] +
            2*pi*scipy.integrate.quad(lambda r:abs(psi(kappa,r)*B)**2*kn*n2.imag*r,R,np.inf)[0]
                        for kappa,B in zip(kappa_values,B_values)])
    beta_values = sqrt(n1.real**2*kn**2-kappa_values**2)

    # The full solution is the superposition of the guided modes
    def field(x,r):
        solution = None
        for i in range(len(kappa_values)):
            mode = c_values[i] * psi(kappa_values[i],np.abs(r)) * exp( (1j*(beta_values[i]-kn)+mu_values[i])
                  * x)
            if solution is None: solution = mode
            else: solution += mode
        return solution
    linspace = lambda a,b,N: np.linspace(a,b,int(N))
    x_values = linspace(*settings.get_as((s.zmin/r,s.zmax/r,s.Nz),float))
    r_values = linspace(*settings.get_as((s.xmin/r,s.xmax/r,s.Nx),float))
    data = np.conjugate(field(*np.meshgrid(x_values,r_values)))
    sx = settings.get_numeric(s.sx)
    sz = settings.get_numeric(s.sz)
    res = CoordinateNDArray(data,[(-sx/2,sx/2),(0,sz)],(s.x,s.z),settings.get_numeric_transform())
    return res
```

Code Listing 14.4: Script to plot analytical waveguide solutions. Requires Code 14.3.

```
from pypropagate import *
from pypropagate.colormaps import fire_colormap
import matplotlib.pyplot as plt
from analytical_waveguides import *
from mpl_toolkits.axes_grid1 import make_axes_locatable

def analytical_waveguide(slab_waveguide,plot_poynting_streamlines,path = None):
    # Define waveguide
    settings = presets.settings.create_paraxial_wave_equation_settings(fresnel_compatible = True)
    presets.boundaries.set_plane_wave_initial_conditions(settings)
    s = settings.symbols
    wg = settings.create_category('waveguide')
```

```
    wg.create_symbol('n_1')
    wg.create_symbol('n_2')
    wg.create_symbol('r')
    wg.create_symbol('l')
    s.n = pc.piecewise((wg.n_1,s.x**2+s.y**2<=wg.r**2),(wg.n_2,True))

    # Set paramters
    wg.n_1 = 1
    wg.n_2 = presets.medium.create_material('Ge',settings)
    wg.l = 0.6*units.mm
    settings.wave_equation.set_energy(10*units.keV)
    settings.simulation_box.set((4*wg.r,0,0.5*units.mm),(1000,0,1000))

    # Plot
    fig,(ax1,ax2) = plt.subplots(2,1,figsize=(8,5),sharex=True)
    wg.r = 25*units.nm
    field = analytical_slab_waveguide(settings) if slab_waveguide else analytical_circular_waveguide(
        settings)
    im = plot(field,ax=ax1,interpolation='bilinear',cmap=fire_colormap())
    if plot_poynting_streamlines:
        start_points = [[0,i*-2*wg.r] for i in np.linspace(0.1,1,10)]
        stream = poynting_streamplot_with_start_points(field,s.k,start_points,color=(1,1,1,0.75),
            arrowpositions=[],ax=ax1)
    else:
        ax1.plot([0,500],[-25,-25],'--',color=(1,1,1))
    ax1.plot([0,500],[25,25],'--',color=(1,1,1))
    ax1.set_ylim(-2*25,2*25)
    ax1.set_xlim(0,500)
    ax1.set_xlabel('')
    divider = make_axes_locatable(ax1)
    cax = divider.append_axes("right", "1.5%", pad="1%")
    plt.colorbar(im, cax=cax)
    wg.r = 50*units.nm
    field = analytical_slab_waveguide(settings) if slab_waveguide else analytical_circular_waveguide(
        settings)
    im = plot(field,ax=ax2,interpolation='bilinear',cmap=fire_colormap())
    if plot_poynting_streamlines:
        start_points = [[0,i*-2*wg.r] for i in np.linspace(0.1,1,10)]
        stream = poynting_streamplot_with_start_points(field,s.k,start_points,color=(1,1,1,0.75),
            arrowpositions=[],ax=ax2)
    else:
        ax2.plot([0,500],[-50,-50],'--',color=(1,1,1))
    ax2.plot([0,500],[50,50],'--',color=(1,1,1))
    ax2.set_yticklabels([-100,-50,0,50,100])
    ax2.set_ylim(-100,100)
    ax2.set_ylabel(r'$x$ [$\mathrm{nm}$]')
    divider = make_axes_locatable(ax2)
    cax = divider.append_axes("right", "1.5%", pad="1%")
    plt.colorbar(im, cax=cax)
    plt.tight_layout()
    if path: plt.savefig(path,transparent=True)
    plt.show()

analytical_waveguide(True,True,'plots/elementary_solutions/slab_waveguide_anal_25-50nm_poynting.pdf')
analytical_waveguide(False,True,'plots/elementary_solutions/circ_waveguide_anal_25-50nm_poynting.pdf')
analytical_waveguide(True,False,'plots/elementary_solutions/slab_waveguide_anal_25-50nm.pdf')
analytical_waveguide(False,False,'plots/elementary_solutions/circ_waveguide_anal_25-50nm_poynting.pdf')
```

Code Listing 14.5: Validation of PyPropagate propagators in free space (Gaussian beam). This code creates Fig. 5.2 and Fig. 5.4.

```
def gaussian_beam_validation(Nz,path=None):
    from pypropagate import np,pc,presets,propagators,plot
    import matplotlib.pyplot as plt
    from pypropagate.colormaps import ice_colormap
    from pypropagate.plot import get_metric_prefix,get_unitless_bounds
    from matplotlib.colors import LogNorm
    from matplotlib.ticker import FuncFormatter
    import warnings

    # Define the gaussian beam settings
    settings = presets.settings.create_paraxial_wave_equation_settings()
    s = settings.symbols
    r = pc.sqrt(s.x**2+s.y**2)
    g = settings.create_category('gaussian',info='Parameters of the gaussian beam')
    g.create_symbol('w_0',type=pc.Types.Real,positive=True,info = 'Waist size at z = 0')
    g.create_function('w_r',(s.z,),pc.sqrt(2)*pc.sqrt((g.w_0)**2/2-2j/(2*s.k)*s.z),info = 'Waist size')
    g.create_function('u3D',(s.x,s.y,s.z),(g.w_0)**2/g.w_r**2*pc.exp(-(r**2)/(g.w_r**2)),info='3D Gaussian')
    g.create_function('u2D',(s.x,s.y,s.z), (g.w_0)/g.w_r*pc.exp(-(s.x**2)/(g.w_r**2)),info='2D Gaussian')
```

```
    # Set paramters and simulation box
    g.w_0 = 0.25*units.um
    s.n = 1
    s.u_boundary = 0
    s.zmin = -s.sz/2
    settings.wave_equation.set_energy(12*units.keV)
    s.xmin = s.ymin = -10*units.um
    s.xmax = s.ymax = 10*units.um
    s.zmin = -10*units.mm
    s.zmax = 10*units.mm
    s.Nx = s.Ny = 2**11
    s.Nz = Nz

    # Propagate
    fields = [ expression_to_array(g.u2D,settings) , expression_to_array(g.u2D.subs(s.y,0),settings) ]
    names = ['Analytical 2D Gaussian Beam','Analytical 3D Gaussian Beam','Finite Differences 2D','Fresnel 2D
        ','Finite Differences 3D','Fresnel 3D','Finite Differences CS','Fresnel CS']
    propagators = [propagators.FiniteDifferences2D,propagators.Fresnel2D,propagators.FiniteDifferences3D,
        propagators.Fresnel3D,propagators.FiniteDifferencesCS,propagators.FresnelCS]
    for prop in propagators:
        print "Running %s..." % prop.__name__
        if prop.ndim == 1:
            s.u0 = g.u2D
            propagator = prop(settings)
            field = propagator.run_slice()[::s.sx/1000,:]
        else:
            s.u0 = g.u3D
            propagator = prop(settings)
            field = propagator.run_slice()[::s.sx/1000,0,:]
        fields.append(field)

    # Plot
    fig,axes = plt.subplots(4,2,figsize=(4*2,2.5*3),sharex=True,sharey=True)
    start_points = [(s.zmin,-6*i*units.um) for i in np.linspace(0.2,0.9,5)]
    for idx,(field,label) in enumerate(zip(fields,names)):
        if field is None:
            ax.set_xlabel('')
            ax.set_ylabel('')
            continue
        i = idx%2
        j = idx/2
        ax = axes[j][i]
        field = field[-6*units.um:6*units.um]
        cmap = 'inferno' if idx in [0,2,3] else 'magma'
        im = plot(field + 1e-50,ax=ax,vmax=1,norm=LogNorm(10**-20),cmap=cmap)
        poynting_streamplot_with_start_points(field,s.k,start_points,ax=ax,color=(1,1,1,0.75) )
        ax.text(0.5, 0.8, label,verticalalignment='bottom', horizontalalignment='center', transform=ax.
            transAxes, color='white', fontsize=11)
        if idx in [0,1]:
            pos = ax.get_position()
            cbaxes = fig.add_axes([pos.xmin+(0.08 if idx is 0 else 0.12),pos.ymin + 0.065,0.2,0.01])
            cbar = fig.colorbar(im,cax=cbaxes, orientation='horizontal')
            cbaxes.tick_params(axis='x', colors='white')
            for pos in ['top', 'bottom', 'right', 'left']:
                cbaxes.spines[pos].set_edgecolor('white')
            cbar.outline.set_visible(False)
            cbar.set_ticks([10**-20,10**-15,10**-10,10**-5,1])
            cbaxes.tick_params(labelsize=8)
            cbaxes.xaxis.set_ticks_position('top')
            cbaxes.xaxis.set_label_position('top')
        if j != 3: ax.set_xlabel('')
        else: ax.xaxis.set_major_formatter(FuncFormatter(lambda x,p:"%i" % x if x%1==0 else '%s' % x))
        if i != 0: ax.set_ylabel('')

    with warnings.catch_warnings():
        warnings.simplefilter("ignore")
        plt.tight_layout(pad=0.1, w_pad=0.1, h_pad=0.1)
    plt.subplots_adjust(left=0.1)

    if path is not None:
        plt.savefig(path,transparent=True)
    plt.show()

gaussian_beam_validation(50,'plots/validation/free_space_gaussian_large_step.pdf')
gaussian_beam_validation(1000,'plots/validation/free_space_gaussian.pdf')
```

Code Listing 14.6: Validation of PyPropagate propagators in piecewise homogeneous matter by propagation in waveguides. This code creates Fig. 5.3 and Fig. 5.5.

```
from pypropagate import *

def waveguide_validation(Nx,Nz,radius,path=None,live_plot=False):
    from pypropagate.colormaps import fire_colormap
    import matplotlib.pyplot as plt
    from analytical_waveguides import analytical_circular_waveguide,analytical_slab_waveguide
    import warnings

    # Define waveguide settings
    settings = presets.settings.create_paraxial_wave_equation_settings(fresnel_compatible = True)
    s = settings.symbols
    wg = settings.create_category('waveguide')
    wg.create_symbol('n_1')
    wg.create_symbol('n_2')
    wg.create_symbol('r')
    wg.create_symbol('l')
    s.n = pc.piecewise((wg.n_1,s.x**2+s.y**2<=wg.r**2),(wg.n_2,True))
    wg.n_1 = 1
    wg.n_2 = presets.medium.create_material('Ge',settings)
    wg.r = radius
    wg.l = 0.6*units.mm
    settings.wave_equation.set_energy(12*units.keV)
    s.u0 = pc.exp(-(s.x**2 + s.y**2) / (2 * (2*wg.r)**2))
    s.u_boundary = 0

    # Propagate
    settings.simulation_box.set((30*wg.r,30*wg.r,0.8*units.mm),(2**10,2**10,Nz))
    print "dx, dr = %f nm\tdz = %f um" % (settings.get_numeric(s.dx/units.nm),settings.get_numeric(s.dz/
        units.um))
    fields = [ analytical_slab_waveguide(settings)[-2*wg.r:2*wg.r] , analytical_circular_waveguide(settings)
        [-2*wg.r:2*wg.r] ]
    names = ['Analytical Slab Waveguide','Analytical Cylindrical Waveguide','Finite Differences 2D','Fresnel
         2D','Finite Differences 3D','Fresnel 3D','Finite Differences CS','Fresnel CS']
    props = [propagators.FiniteDifferences2D,propagators.Fresnel2D,propagators.FiniteDifferences3D,
        propagators.Fresnel3D,propagators.FiniteDifferencesCS,propagators.FresnelCS]
    for prop in props:
        print "Running %s..." % prop.__name__
        if prop.ndim == 2:
            propagator = prop(settings)
            field = propagator.run_slice(display_progress=True)[-2*wg.r:2*wg.r:wg.r/500,0,::s.sz/1000]
        else:
            propagator = prop(settings)
            field = propagator.run_slice(display_progress=True)[-2*wg.r:2*wg.r:wg.r/500,::s.sz/1000]
        fields.append(field)
        if live_plot:
            plot_poynting(field,s.k,figsize=(15,4))

    # Plot
    fig,axes = plt.subplots(4,2,figsize=(4*2,2.5*3),sharex=True,sharey=True)
    max2D = (abs(fields[0])**2).max()
    max3D = (abs(fields[1])**2).max()

    for idx,field,label in zip(range(len(fields)),fields,names):
        i = idx%2
        j = idx/2
        ax = axes[j][i]
        vmax = max2D if idx in [0,2,3] else max3D
        cmap = fire_colormap(hue_shift = 0.05 if idx in [0,2,3] else 0)
        im = plot(abs(field)**2,ax=ax,vmax=vmax,vmin=0,interpolation='bilinear',cmap=cmap)
        support = expression_for_array(s.x<0,field,settings)
        if idx in [0,1]:
            cbarmask = expression_for_array(pc.Or(s.x>-60 * units.nm,s.z<200*units.um),field,settings)
            support.data = np.logical_and( support.data , cbarmask.data )
        start_points = [[0,sp*-2*radius] for sp in np.linspace(0.1,1,5)]
        if idx in [0,1]:
            start_points = start_points[:-2]
        else:
            start_points = start_points[:-1]
        stream = poynting_streamplot_with_start_points(field,s.k,start_points,color=(1,1,1,0.75),ax=ax,
            arrowpositions=[])
        ax.text(0.5, 0.85, label,verticalalignment='bottom', horizontalalignment='center', transform=ax.
            transAxes, color='white', fontsize=11)
        if idx in [0,1]:
            pos = ax.get_position()
            cbaxes = fig.add_axes([pos.xmin+(0.08 if idx is 0 else 0.12),pos.ymin + 0.065,0.2,0.01])
            cbar = fig.colorbar(im,cax=cbaxes, orientation='horizontal')
```

```
            cbaxes.tick_params(axis='x', colors='white')
            for pos in ['top', 'bottom', 'right', 'left']:
                cbaxes.spines[pos].set_edgecolor('white')
            cbar.outline.set_visible(False)
            cbar.set_ticks([k*(1 if idx in [0,2,4] else 4) for k in range(10)])
            cbaxes.tick_params(labelsize=8)
            cbaxes.xaxis.set_ticks_position('top')
            cbaxes.xaxis.set_label_position('top')
        ax.plot([0,400],[float(radius/units.nm),float(radius/units.nm)],'w--')
        ax.set_ylim(-2*float(radius/units.nm),2*float(radius/units.nm))
        ax.set_xlim(0,400)
        if j != 3:
            ax.set_xlabel('')
            plt.setp(ax.get_xticklines(), visible=False)
        if i == 1:
            ax.set_ylabel('')
            plt.setp(ax.get_yticklines(), visible=False)
    with warnings.catch_warnings():
        warnings.simplefilter("ignore")
        plt.tight_layout(pad=0.1, w_pad=0.1, h_pad=0.1)
    if path: plt.savefig(path,transparent=True)
    plt.show()

waveguide_validation(2**10,1000,25*units.nm,'plots/validation/waveguide_prop_25nm_large_step.pdf')
waveguide_validation(2**10,50000,25*units.nm,'plots/validation/waveguide_prop_25nm.pdf')
waveguide_validation(2**10,50000,50*units.nm,'plots/validation/waveguide_prop_50nm.pdf')
```

Code Listing 14.7: Validation of piecewise paraxial propagation by curved waveguide propagation. This code creates Fig. 5.6.

```
from pypropagate import *
from pypropagate.colormaps import fire_colormap
import matplotlib.pyplot as plt
from matplotlib.colors import LogNorm

# Define waveguide
settings = presets.settings.create_paraxial_wave_equation_settings()
s = settings.symbols
wg = settings.create_category('waveguide')
wg.create_symbol('n_inside')
wg.create_symbol('n_outside')
wg.create_symbol('r')
wg.create_symbol('d')
wg.create_symbol('R')
wg.create_function('center_x',(s.z,))
wg.center_x = pc.sqrt(wg.R**2 - s.z**2) - wg.R
dist_from_center = pc.sqrt((s.x - wg.center_x)**2 + s.y**2)
s.n = pc.piecewise((wg.n_inside,pc.Or(pc.And(abs(s.x - wg.center_x) < wg.r,s.y >= -wg.d)),s.y > 0),(wg.
     n_outside,True))
settings.get(s.n)
wg.n_inside = 1
wg.n_outside = presets.medium.create_material('Ta',settings)
wg.d = wg.r
wg.r = 100*units.nm
wg.R = 40*units.mm
s.zmin = 0
s.zmax = 0.5 * units.mm
s.xmax = wg.center_x.subs(s.z,0) + 2*wg.r
s.xmin = wg.center_x.subs(s.z,s.zmax) - 2*wg.r
s.Nx = 10000
s.Nz = 10000
settings.wave_equation.set_energy(7.9*units.keV)
presets.boundaries.set_plane_wave_initial_conditions(settings)
print "dx = %f nm\tdz = %f um" % (settings.get_numeric(s.dx/units.nm),settings.get_numeric(s.dz/units.um))
propagator = propagators.FiniteDifferences2D(settings)
field_cartesian = propagator.run_slice()[::s.sx/4000,::s.sz/4000]

# Data for plot
wg_center = abs(expression_to_array(wg.center_x/units.um,settings).data)
wg_r = settings.get_as(wg.r/units.um,float)
zvalues = abs(expression_to_array(s.z/units.um,settings).data)

# Piecewise paraxial propagation
def create_curvature_initializer(R):
    def curvature_initializer(settings):
        s = settings.symbols
        alpha = settings.get_as(-pc.atan(s.dz/R),float)
        update_field = expression_to_array(pc.exp(1j*s.k*s.x*pc.sin(alpha)),settings).data
        def curvature_updater(propagator):
```

```
            field = propagator._get_field()
            field *= update_field
        settings.updaters['curvature'] = curvature_updater
    return curvature_initializer

settings.initializers['curvature'] = create_curvature_initializer(wg.R)

# Adjust center
wg.center_x = 0
print "dx = %f nm\tdz = %f um" % (settings.get_numeric(s.dx/units.nm),settings.get_numeric(s.dz/units.um))
propagator = propagators.FiniteDifferences2D(settings)
field_curved = propagator.run_slice()[::s.sx/4000,::s.sz/4000]

# Distort cartesian solution to match curved coordinate system
field_cartesian_shifted = field_cartesian.copy()
x,y = np.meshgrid(np.arange(field_cartesian.shape[1]),np.arange(field_cartesian.shape[0]))
RR = float(field_cartesian_shifted.evaluate(wg.R/s.dz))
field_cartesian_shifted.data = field_cartesian.data[ np.floor((y + 283.6 * (np.sqrt(RR**2-x**2) - RR)) %
     field_cartesian.shape[0]).astype(int) ,x]

# Plot
fig,(ax0,ax1,ax2) = plt.subplots(3,1,figsize=(8,7.5),sharex=True)
vmax = abs(field_curved).max()**2
plot(field_cartesian,ax=ax0,vmax=vmax,cmap=fire_colormap())
ax0.plot(zvalues,-wg_center+wg_r,'w--')
ax0.plot(zvalues,-wg_center-wg_r,'w--')
plot(field_cartesian_shifted[-200*units.nm:],ax=ax1,vmax=vmax,cmap=fire_colormap())
ax1.plot([0,1000],[100,100],'w--')
ax1.plot([0,1000],[-100,-100],'w--')
ax1.set_xlim(0,500)
ax1.set_xlabel('')
ax1.set_ylabel('$x\'\;[\mathrm{nm}]$')
im = plot(field_curved,ax=ax2,vmax=vmax,cmap=fire_colormap())
ax2.plot([0,1000],[100,100],'w--')
ax2.plot([0,1000],[-100,-100],'w--')
ax2.set_xlabel('$z\'\;[\mu\mathrm{m}]$')
ax2.set_ylabel('$x\'\;[\mathrm{nm}]$')
fig.subplots_adjust(right=0.9)
cbar_ax = fig.add_axes([0.92, 0.13, 0.015, 0.75])
fig.colorbar(im, cax=cbar_ax)
plt.savefig('plots/validation/curved_waveguide_comparison.pdf',transparent=True)
```

Code Listing 14.8: Validation of periodic envelope propagation using angled Gaussian pulses. Creates Fig. 8.2 and Fig. 8.3.

```
from pypropagate import *
from pypropagate.fit import fit_gaussians
import matplotlib.pyplot as plt
from mpl_toolkits.axes_grid1 import make_axes_locatable
from pypropagate.colormaps import fire_colormap

# Parameters
FWHM_t = 0.3 * units.fs
alpha = 0.02
l = 4 * FWHM_t * units.c / (pc.cos(alpha)**-1 - 1)
h = pc.tan(alpha) * l
stretch = 2000

# Create settings
settings = presets.settings.create_2D_paraxial_frequency_settings()
s = settings.symbols

# Define gaussian beam
g = settings.create_category('gaussian',info='Parameters of the gaussian beam')
g.create_symbol('sigma_x',info = 'Waist size at z = 0',type=pc.Types.Real,positive=True)
g.create_function('w_x',(s.z,),pc.sqrt(2)*pc.sqrt(g.sigma_x**2-1j/s.k*s.z),info = 'Waist size')
g.create_function('u',(s.x,s.z),g.sigma_x/g.w_x*pc.exp(-s.x**2/(g.w_x**2)));

# Define gaussian pulses
p1 = settings.create_category('pulse_1')
p1.create_symbol('sigma_t')
p1.create_symbol('omega_0')
p1.create_symbol('sigma_omega',1/p1.sigma_t)
p1.create_function('utilde_0',(s.x,s.z,s.omega),pc.exp(-1*(s.omega - p1.omega_0)**2 /(2*p1.sigma_omega**2))
     * g.u  )
p1.create_symbol('alpha',0)
xa,za = pc.cos(p1.alpha)*s.x-pc.sin(p1.alpha)*s.z,pc.cos(p1.alpha)*s.z+pc.sin(p1.alpha)*s.x
p1.create_function('utilde',(s.x,s.z,s.omega),p1.utilde_0.function(xa,za,s.omega) * pc.exp(-1j*s.k*(za - s.z
     ) ))
```

```
p2 = settings.create_category('pulse_2')
p2.create_symbol('sigma_t')
p2.create_symbol('omega_0')
p2.create_symbol('sigma_omega',1/p2.sigma_t)
p2.create_function('utilde_0',(s.x,s.z,s.omega),pc.exp(-1*(s.omega - p2.omega_0)**2 /(2*p2.sigma_omega**2))
    * g.u  )
p2.create_symbol('alpha',0)
xa,za = pc.cos(p2.alpha)*s.x-pc.sin(p2.alpha)*s.z,pc.cos(p2.alpha)*s.z+pc.sin(p2.alpha)*s.x
p2.create_function('utilde',(s.x,s.z,s.omega),p2.utilde_0.function(xa,za,s.omega) * pc.exp(-1j*s.k*(za - s.z
    ) ))

# Set parameters
g.sigma_x = h/10
p1.sigma_t = FWHM_t * pc.sqrt(2)/ (2 * pc.sqrt(2*pc.log(2)))
p1.omega_0 = 12 * units.keV/units.hbar
p1.alpha = -alpha
p2.sigma_t = FWHM_t * pc.sqrt(2)/ (2 * pc.sqrt(2*pc.log(2)))
p2.omega_0 = 12 * units.keV/units.hbar
p2.alpha = 0 * units.degrees
s.n = 1
s.u0 = p1.utilde.subs(s.x,s.x-h) + p2.utilde
s.u_boundary = pc.piecewise( ( s.u0,s.x>0), (0,True) )
s.omegamin = p1.omega_0 - 20 * p1.sigma_omega
s.omegamax = p1.omega_0 + 20 * p1.sigma_omega
s.xmin = -h * 1.5
s.xmax = h * 1.5
s.zmin = 0
s.zmax = 2*l
s.Nomega = 2**8
s.Nx = 2**17
s.Nz = 1000
print "domega = %f FWHM_omega" % settings.get_numeric(s.domega / (p1.sigma_omega * 2*(2*pc.log(2))**0.5))
print "dx = %f nm" % settings.get_numeric(s.dx / units.nm )
print "dz = %f um" % settings.get_numeric(s.dz / units.um )

# Propagate
propagator = propagators.Fresnel3D(settings)
u = propagator.run_slice()[::s.sx/1000,:,:]
t = pc.Symbol('t')
tdfield = presets.time.periodic_envelope_propagation(u,p1.omega_0,s=float(stretch)).transpose((t,s.x,s.z))

# Plot
time_averaged_intensity = (abs(tdfield)**2).sum(axis=t)
time_averaged_intensity /= time_averaged_intensity[:,0].max()
monochromatic_intensity = abs(u[:,(s.omegamin + s.omegamax)/2,:])**2
monochromatic_intensity /= monochromatic_intensity[:,0].max()
vmax = monochromatic_intensity.max()
fig,(ax1,ax2) = plt.subplots(2,1,figsize=(8,5),sharex=True)
im1 = plot(monochromatic_intensity[-15*units.um:15*units.um,s.sz/2-1*units.mm:s.sz/2+1*units.mm],ax=ax1,vmax
    =vmax,cmap=fire_colormap())
ax1.set_xlabel('')
im2 = plot(time_averaged_intensity[-15*units.um:15*units.um,s.sz/2-1*units.mm:s.sz/2+1*units.mm],ax=ax2,vmax
    =vmax,cmap=fire_colormap())
fig.subplots_adjust(right=0.9)
cbar_ax = fig.add_axes([0.91, 0.125, 0.02, 0.755])
fig.colorbar(im1, cax=cbar_ax)
plt.savefig('plots/validation/angled_gaussian_pulses_xz.pdf',transparent=True)
t_cut = l/pc.cos(alpha)/units.c - (1-1./stretch) * l / units.c
z_cut = s.zmin + s.sz/2
cut_int_1 = (abs(tdfield[2.5*units.fs:5*units.fs,-10*units.um:10*units.um,z_cut])**2).transpose()
cut_int_1 /= cut_int_1.max()
cut_int_2 = abs((tdfield[t_cut,-10*units.um:10*units.um,1.45*units.mm:2.85*units.mm]))**2
cut_int_2 /= cut_int_2.max()
fig,(ax1,ax2) = plt.subplots(1,2,figsize=(8,3),sharey=True)
im = plot(cut_int_2,ax=ax1,cmap=fire_colormap(),vmax=1)
im = plot(cut_int_1,ax=ax2,cmap=fire_colormap(),vmax=1)
ax2.plot([settings.get_as(t_cut/units.fs,float)] * 2,[-10,10],'w--')
ax1.plot([settings.get_as(z_cut/units.mm,float)] * 2,[-10,10],'w--')
ax1.set_ylim(-10,10)
ax2.set_ylabel('')
ax2.set_xlabel(r"$t' \; [\mathrm{fs}]$")
plt.colorbar(im)
plt.tight_layout()
plt.savefig('plots/validation/angled_gaussian_pulses_xzt.pdf',transparent=True)

# Fit the time delay
popt,psig = fit_gaussians(abs(tdfield[2.5*units.fs:5*units.fs,0,z_cut])**2,2,periodic=True)
print "dt = %f fs +- %f fs" % ((popt[0][1] - popt[1][1])/units.fs,pc.sqrt((psig[0][1]/units.fs)**2+(psig[1][
    1]/units.fs)**2))
popt,psig = fit_gaussians(abs(tdfield[t_cut,0,:])**2,1,periodic=True)
print "FWHM_z = %f um +- %f um" % ((popt[0][2]*2*pc.sqrt(2*pc.log(2))/units.um,psig[0][2]*2*pc.sqrt(2*pc.log
    (2))/units.um))
```

Code Listing 14.9: Validation of Gaussian pulse width. This script creates Fig. 8.4.

```
from pypropagate import *
import matplotlib.pyplot as plt
from matplotlib.colors import LogNorm
from scipy.optimize import curve_fit

# Create settings
settings = presets.settings.create_2D_paraxial_frequency_settings()
s = settings.symbols

# Define the pulse
p = settings.create_category('pulse_1')
p.create_symbol('sigma_t')
p.create_symbol('omega_0')
p.create_symbol('sigma_omega',1/p.sigma_t)
p.create_function('u_0',(s.x,s.z,s.omega),pc.exp(-1*(s.omega - p.omega_0)**2 /(2*p.sigma_omega**2))  )
presets.boundaries.set_plane_wave_initial_conditions(settings)
s.u0 = p.u_0

FWHM_t = 0.01 * units.fs
p.sigma_t = FWHM_t * pc.sqrt(2)/ (2 * pc.sqrt(2*pc.log(2)))
p.omega_0 = 12 * units.keV/units.hbar
s.omegamin = p.omega_0 - 20 * p.sigma_omega
s.omegamax = p.omega_0 + 20 * p.sigma_omega
s.Nomega = 2**8

# Taylor the refractive index of water
n = presets.medium.create_material('H2O',settings,density=1)
domega = 100*s.domega
n0 = settings.get_numeric(n.subs(s.omega,p.omega_0)).N()
n1 = settings.get_numeric(n.subs(s.omega,p.omega_0+domega)).N()
nn1 = settings.get_numeric(n.subs(s.omega,p.omega_0-domega)).N()
dn0 = (n1 - nn1)/(2*domega)
n_taylored = n0 + (s.omega - p.omega_0) * dn0
s.n = n_taylored
s.xmin = -1*units.nm
s.xmax = 1*units.nm
s.zmin = 0
s.zmax = 100*units.mm
s.Nx = 5
s.Nz = 1000
print "domega = %f FWHM_omega" % settings.get_numeric(s.domega / (p1.sigma_omega * 2*(2*pc.log(2))**0.5))
print "dz = %f um" % settings.get_numeric(s.dz / units.um )

# Propagate and Fourier transform
propagator = propagators.Fresnel3D(settings)
field = propagator.run_slice()[0,:]
tdfield = presets.time.periodic_envelope_propagation(field,p.omega_0,s=1).transpose()

# Fit gaussian profiles to the beam shape
widths = []
intensities = []
indices = []
offsets = []
xdata = np.linspace(float((tdfield.bounds[1][0])/units.fs) , float((tdfield.bounds[1][1])/units.fs) ,
     tdfield.shape[1])
gaussian = lambda x,x0,s,a: a * np.exp(-(x-x0)**2 / (2*s**2))
for i in range(tdfield.shape[0]):
    ydata = abs(tdfield.data[i])
    ymax = ydata.max()
    ydata = np.roll(ydata,ydata.shape[0]/2 - np.where(ydata==ydata.max())[0])
    p0 = [xdata[len(xdata)/2],(xdata[-1]-xdata[0])/40,ymax]
    popt, pcov = curve_fit(gaussian, xdata, ydata,p0=p0)
    offsets.append(popt[0])
    widths.append(abs(popt[1]))
    intensities.append(popt[2])
    indices.append(i)
dz = settings.get_as(s.dz/units.mm,float)
zvalues = np.array([i*dz for i in indices])
widths = np.array(widths)/widths[0]

# Analytical solution
omega0 = p.omega_0
K0 = n0 * omega0/units.c
K1 = (n0 + omega0 * dn0)/units.c
K2 = 2*dn0/units.c
sigma_t = p.sigma_t
zprime = zvalues * complex(settings.get_numeric(units.mm*pc.imag(K2)/sigma_t**2)).real
```

```
gamma = complex(settings.get_numeric(pc.real(K2)/pc.imag(K2))).real
analytical_widths = np.sqrt(((1+zprime**2)+(zprime*gamma)**2)/(1+zprime))
print 'max width deviation: %f %%' % abs((np.array(widths) - analytical_widths) / analytical_widths).max()

# Plot
fig,(ax2,ax1) = plt.subplots(1,2,figsize=(8,3))
ax1.plot(zvalues,widths,label='Numerical')
ax1.plot(zvalues,analytical_widths,label='Analytical')
ax1.legend(loc=2)
ax1.set_xlim(0.,100)
ax1.set_ylim(0.9,3.75)
ax1.set_xlabel(r"$z \; [\mathrm{mm}]$")
ax1.set_ylabel(r"$\Sigma_t/\sigma_t$")
ax1.yaxis.tick_right()
ax1.yaxis.set_label_position('right')
n_plot = expression_to_array(n,settings)
n_taylored_plot = expression_to_array(n_taylored,settings)
u0_plot = expression_to_array(s.u0,settings)
E_plot = expression_to_array(s.E/units.keV,settings).real
ax2.plot(E_plot.data,(n_plot.data.real-1) * 1e6,label='$\Re(n)-1$')
ax2.plot(E_plot.data,(n_taylored_plot.data.real-1) * 1e6,label=r"$\Re(n')-1$")
ax2.set_ylabel(r'$n \; [10^{-6}]$')
ax2.set_xlabel('$E\;[\mathrm{keV}]$')
ax3 = ax2.twinx()
ax3.plot(E_plot.data,abs(u0_plot.data),color='r',label='$|\psi_0|$')
ax3.set_ylabel(r'$|\psi_0| \; [\mathrm{AU}]$')
ax2.set_xlim(9.8,14.2)
ax2.set_xticks(range(10,15))
ax2.legend(loc=2,prop={'size':10})
ax3.legend(loc=5,prop={'size':10})
plt.tight_layout()
plt.savefig('plots/validation/gaussian_pulse_width_comparison.pdf',bbox_extra_artists=(ax2,ax3),transparent=
    True)
```

Code Listing 14.10: Periodic pulse propagation in slab, circular and curved waveguides. This script creates Fig. 9.1, Fig. 9.3, Fig. 9.5 and Fig. 9.3.

```
from pypropagate import *
from pypropagate.colormaps import ice_and_fire_colormap
import matplotlib.pyplot as plt
from matplotlib import gridspec
from pypropagate.fit import fit_gaussians

def pulsed_waveguide_propagation(wg_type): # wg_type in {'slab','circ','curved'}
    print wg_type

    # Create the settings
    settings = presets.settings.create_2D_paraxial_frequency_settings()
    s = settings.symbols

    # Define the pulse parameters
    p = settings.create_category('pulse')
    p.create_symbol('FWHM_t') # width in intensity
    p.create_symbol('sigma_t',p.FWHM_t * pc.sqrt(2)/(2 * pc.sqrt(2*pc.log(2))))
    p.create_symbol('omega_0')
    p.create_symbol('sigma_omega',1/p.sigma_t)
    p.create_function('u_0',(s.x,s.z,s.omega),pc.exp(-1*(s.omega - p.omega_0)**2 /(2*p.sigma_omega**2))  )
    presets.boundaries.set_plane_wave_initial_conditions(settings)
    presets.boundaries.set_plane_wave_initial_conditions(settings)

    # Define the waveguide parameters
    wg = settings.create_category('waveguide')
    wg.create_symbol('d')
    wg.create_function('n_cladding',(s.omega,))
    wg.create_function('n_core',(s.omega,))
    wg.create_function('n',(s.x,s.z,s.omega,), pc.piecewise((wg.n_core,2*abs(s.x) < wg.d),(wg.n_cladding,
        True)))

    # Set parameters
    s.u0 = p.u_0
    p.omega_0 = 12*units.keV / units.hbar
    p.FWHM_t = 0.005*units.fs
    s.omegamin = p.omega_0 - 20 * p.sigma_omega
    s.omegamax = p.omega_0 + 20 * p.sigma_omega
    s.Nomega = 2**8
    wg.d = 100 * units.nm
    wg.n_core = 1
    wg.n_cladding = presets.medium.create_material('Si',settings)
    s.n = wg.n
```

```
s.xmin = -400*units.nm
s.xmax =  400*units.nm
s.zmin = 0
s.zmax = 6*units.mm
s.Nx = 2**12
s.Nz = 2**14
pulse_length = settings.get_numeric(p.FWHM_t * units.c)
stretch = float(settings.get_numeric(0.01 * 10 * units.mm / pulse_length))
print "domega = %f FWHM_omega" % settings.get_numeric(s.domega / (p.sigma_omega * 2*(2*pc.log(2))**0.5))
print "dx = %f nm" % settings.get_numeric(s.dx / units.nm )
print "dz = %f um" % settings.get_numeric(s.dz / units.um )
print "stretch = %f" % stretch

if wg_type == 'curved':
    # Piecewise paraxial propagation
    def create_curvature_initializer(R):
        def curvature_initializer(settings):
            s = settings.symbols
            alpha = settings.get_as(-pc.atan(s.dz/R),float)
            update_field = expression_to_array(pc.exp(1j*s.k*s.x*pc.sin(alpha)),settings).data.transpose
                ()
            def curvature_updater(propagator):
                field = propagator._get_field()
                field *= update_field
            settings.updaters['curvature_updater'] = curvature_updater
        return curvature_initializer

    n_padding = presets.medium.create_material('Ge',settings)
    s_padding = 200 * units.nm
    r = pc.Min(1,pc.Max(0,(s.x - s.xmax + s_padding) / s_padding))
    s.n = wg.n * (1-r) + r * n_padding
    settings.initializers['curvature'] = create_curvature_initializer(40 * units.mm)

# Run simulation
propagator_type = {'slab':propagators.FiniteDifferences3D,'circ':propagators.FiniteDifferences3DCS,'
    curved':propagators.FiniteDifferences3D}
propagator = propagator_type[wg_type](settings)
u = propagator.run_slice()[-wg.d:wg.d:2*wg.d/1024,:,::s.sz/1024]
tdfield = presets.time.periodic_envelope_propagation(u,p.omega_0,s=stretch).transpose([pc.Symbol('t'),s.
    x,s.z])

# Create large xz slice by stacking slice three times
wg_field = tdfield[:,-wg.d/2:wg.d/2,:]
if wg_type == 'circ':
    radius_wg = expression_to_array(abs(s.x/wg.d*2),settings)[-wg.d/2:wg.d/2].real
    intensity = abs(wg_field)**2
    intensity.data.transpose((2,0,1))[:] *= radius_wg.data

    sliced = (intensity).sum(axis=s.x)
    sliced /= wg_field.shape[1]
    del intensity
else:
    sliced = (abs(wg_field)**2).sum(axis=s.x)
    sliced /= wg_field.shape[wg_field.axis.index(s.x)]
stacked_data = np.vstack([sliced.data,sliced.data,sliced.data])
stacked = CoordinateNDArray(stacked_data,[[sliced.bounds[0][0],sliced.bounds[0][1]*3],sliced.bounds[1]],
    sliced.axis,sliced.evaluate)

# Remove periodic signals
stacked[0.1*units.fs:,:1*units.mm].data[:] = 0
stacked[0.15*units.fs:,:2*units.mm].data[:] = 0
stacked[0.2*units.fs:,:2.5*units.mm].data[:] = 0
stacked[0.225*units.fs:,:2.9*units.mm].data[:] = 0
stacked[0.25*units.fs:,:4*units.mm].data[:] = 0
stacked[0.31*units.fs:,:5*units.mm].data[:] = 0
stacked[:0.08*units.fs,1.9*units.mm:].data[:] = 0
stacked[:0.15*units.fs,3.1*units.mm:].data[:] = 0
stacked[:0.2*units.fs,4.1*units.mm:].data[:] = 0
stacked[:0.24*units.fs,5*units.mm:].data[:] = 0
normalization = stacked[0].max()
vmax = 0.6

# Plot refractive index
if wg_type == 'slab':
    fig,(ax2) = plt.subplots(1,1,figsize=(5,3))
    n_plot = expression_to_array(wg.n_cladding,settings)
    u0_plot = expression_to_array(s.u0,settings)
    E_plot = expression_to_array(s.E/units.keV,settings).real
    ax2.plot(E_plot.data,(n_plot.data.real-1) * 1e6,label='$\Re(n)-1$')
    ax2.plot(E_plot.data,(n_plot.data.imag) * 1e6,label='$\Im(n)$')
    ax2.set_ylabel(r'$n \; [10^{-6}]$')
    ax2.set_xlabel('$E\;[\mathrm{keV}]$')
```

```
        ax2.set_ylim(-6,0.5)
        ax3 = ax2.twinx()
        ax3.plot(E_plot.data,abs(u0_plot.data)**2,color='r',label='$|\psi_0|^2$')
        ax3.set_ylabel(r'$|\psi_0|^2 \; [\mathrm{AU}]$')
        ax2.set_xticks(range(4,20,2))
        ax2.set_xlim(E_plot.min(),E_plot.max())
        ax2.legend(loc=6,prop={'size':10})
        ax3.legend(loc=5,prop={'size':10})
        plt.tight_layout()
        plt.savefig('plots/time_dependent/n_Si_12keV_5as.pdf')

    # Plot
    def plot_intensity(pos,length,time,ax):
        exit_field_freq = CoordinateNDArray(np.repeat(u[:,:,pos].data[:,:,np.newaxis],100,axis=2),[u.bounds[
            0],u.bounds[1],[pos,pos + length]],u.axis,u.evaluate)
        exit_field = presets.time.periodic_envelope_propagation(exit_field_freq,p.omega_0,1).transpose([pc.
            Symbol('t'),s.x,s.z])
        im = plot((exit_field[time].real)**2/normalization,cmap=ice_and_fire_colormap(),vmax=vmax,ax=ax)
        ax.plot([float(b/units.nm) for b in exit_field.bounds[2]],[50]*2,'-',color='k')
        ax.plot([float(b/units.nm) for b in exit_field.bounds[2]],[-50]*2,'-',color='k')
        return im
    fig,(ax1,ax2,ax3) = plt.subplots(1,3,figsize=(10,4),sharey=True)
    im = plot_intensity(0.5 * units.mm,11.5 * pulse_length,0.0175*units.fs,ax=ax1)
    plot_intensity(2 * units.mm,11.5 * pulse_length,0.0375*units.fs,ax=ax2)
    plot_intensity(4 * units.mm,11.5 * pulse_length,0.045*units.fs,ax=ax3)
    if wg_type == 'circ': ax1.set_ylabel('$r \; [\mathrm{nm}]$')
    if wg_type == 'curved':
        ax1.set_ylabel(r"$x' \; [\mathrm{nm}]$")
        for ax in [ax1,ax2,ax3]: ax.set_xlabel(r"$z' \; [\mathrm{nm}]$")
    ax2.set_ylabel('')
    ax3.set_ylabel('')
    plt.tight_layout()
    fig.colorbar(im, ax=[ax1,ax2,ax3],pad=0.01)
    plt.savefig('plots/time_dependent/%s_waveguide_as_pulse_intensities.pdf' % wg_type)

    t0 = 0.01*units.fs
    t1 = t0 + tdfield.bounds[0][1]
    t2 = t0 + 2*tdfield.bounds[0][1]
    fig = plt.figure(figsize=(10,4))
    gs = gridspec.GridSpec(1, 2, width_ratios=[2,1.5])
    ax1 = plt.subplot(gs[1])
    im = plot(stacked.transpose()[:6*units.mm]/normalization,cmap=ice_and_fire_colormap(),ax=ax1,vmax=vmax)
    ax1.plot([float(t0*1e3/units.fs)] * 2,[0,6],'k--')
    ax1.plot([float(t1*1e3/units.fs)] * 2,[0,6],'k--')
    ax1.plot([float(t2*1e3/units.fs)] * 2,[0,6],'k--')
    ax1.set_xlim(0,360)
    ax1.set_ylim(0,6)
    cbar = plt.colorbar(im)
    plot_intensity = abs(tdfield[t0,:,:5.25*units.mm])**2
    ax2 = plt.subplot(gs[0])
    im = plot(plot_intensity/normalization,cmap=ice_and_fire_colormap(),ax=ax2,vmax=vmax)
    ax2.plot([0,6],[50,50],'-',color='k')
    ax2.plot([0,6],[-50,-50],'-',color='k')
    ax2.set_xlim(0,5.25)
    ax2.text(0.27,-75,"$t' = %.0f \; \mathrm{as}$" % (t0*1000/units.fs),horizontalalignment='left',
          verticalalignment='center')
    ax2.text(2.1,-75,"$t' = %.0f \; \mathrm{as}$" % (t1*1000/units.fs),horizontalalignment='left',
          verticalalignment='center')
    ax2.text(4.2,-75,"$t' = %.0f \; \mathrm{as}$" % (t2*1000/units.fs),horizontalalignment='left',
          verticalalignment='center')
    ax2.yaxis.labelpad = -10
    if wg_type == 'circ': ax2.set_ylabel('$r \; [\mathrm{nm}]$')
    if wg_type == 'curved':
        ax2.set_ylabel(r"$x' \; [\mathrm{nm}]$")
        ax2.set_xlabel(r"$z' \; [\mathrm{nm}]$")
        ax1.set_ylabel(r"$z' \; [\mathrm{nm}]$")
    ax1.set_xlabel(r"$t' \; [\mathrm{as}]$")
    plt.tight_layout()
    plt.savefig('plots/time_dependent/mode_group_velocities_%s_waveguide.pdf' % wg_type)

    # Determine goup velocities
    zfit = stacked._dbounds[1] * stacked.shape[1] * 0.8
    NM = 1 if wg_type == 'curved' else 3
    popt,psig = fit_gaussians(stacked[:,zfit].transpose(),NM)
    for i,v,s in zip(range(NM),popt,psig):
        print 'mode %i:' % i
        print "  v = (1-%.5e) c" % ( 1 - zfit/(v[1] + zfit/units.c * (1-1./stretch)) / units.c)
        print "  s = %.3f +- %.3f" % tuple(settings.get_as( (v[2]*pc.sqrt(2)/(p.sigma_t) , s[2]*pc.sqrt(2)/p
            .sigma_t) , float))

pulsed_waveguide_propagation('slab')
pulsed_waveguide_propagation('circ')
```

```
pulsed_waveguide_propagation('curved')
```

Code Listing 14.11: Propagation of Gaussian pulses at the K absorption edge of Ni. This script creates Fig. 9.8, Fig. 9.9, Fig. 9.10.

```
from pypropagate import *
import matplotlib.pyplot as plt
from lmfit.models import VoigtModel

settings = presets.settings.create_2D_paraxial_frequency_settings()
pde = settings.partial_differential_equation
s = settings.symbols

p = settings.create_category('pulse')
p.create_symbol('FWHM_t')
p.create_symbol('sigma_t',p.FWHM_t * pc.sqrt(2)/(2 * pc.sqrt(2*pc.log(2))))
p.create_symbol('omega_0')
p.create_symbol('sigma_omega',1/p.sigma_t)
p.create_function('u_0',(s.x,s.z,s.omega),pc.exp(-1*(s.omega - p.omega_0)**2 /(2*p.sigma_omega**2))  )
wg = settings.create_category('waveguide')
wg.create_function('n_core',(s.omega,))
wg.create_function('n_cladding',(s.omega,))
wg.create_symbol('r')

s.u0 = p.u_0
presets.boundaries.set_1D_boundary_condition(settings)
wg.n_core = 1
wg.n_cladding = presets.medium.create_material('Ni',settings)
wg.r = 50*units.nm
p.FWHM_t = 1 * units.fs
p.omega_0 = 8.333*units.keV/units.hbar
s.omegamin = p.omega_0 - 10 * p.sigma_omega
s.omegamax = p.omega_0 + 10 * p.sigma_omega
s.Nomega = 2**8
s.xmin = -0.1* units.um
s.xmax = 0.1 * units.um
s.zmin = 0
s.zmax = 50*units.mm
s.Nx = 1000
s.Nz = 100000
s.n = pc.piecewise((wg.n_core,abs(s.x) <= wg.r),(wg.n_cladding,True))

# Function to center periodic data
centered = lambda field: np.roll(field,field.shape[0]/2 - np.where(field == field.max())[0][0])

# Function to determine pulse width and intensity
def get_wi(int_field):
    xdata = np.linspace(float((int_field.bounds[0][0])/units.fs) , float((int_field.bounds[0][1])/units.fs)
          , int_field.shape[0])
    widths = []
    intensities = []
    mod = VoigtModel()
    for i in range(int_field.shape[1]):
        ydata = centered(int_field.data[:,i])
        ymax = ydata.max()
        pars = mod.guess(ydata, x=xdata)
        out = mod.fit(ydata, pars, x=xdata)
        widths.append(out.result.params.get('fwhm').value)
        intensities.append(out.result.params.get('height').value)
    return np.array(widths),np.array(intensities)

# Slab waveguide
propagator = propagators.FiniteDifferences3D(settings)
u = propagator.run_slice()[-2*wg.r:2*wg.r:4*wg.r/1000,:,::s.sz/1000]
tdfield = presets.time.periodic_envelope_propagation(u,p.omega_0).transpose([pc.Symbol('t'),s.x,s.z])
wg_field = tdfield[:,-wg.r:wg.r]
int_field = (abs(wg_field)**2).sum(axis=s.x)
slab_w,slab_i = get_wi(int_field)
dz = float(int_field._dbounds[1]/units.mm)
zvalues = np.array([i*dz for i in range(int_field.shape[1])])

# Circular waveguide
propagator = propagators.FiniteDifferences3DCS(settings)
u = propagator.run_slice()[-2*wg.r:2*wg.r:4*wg.r/1000,:,::s.sz/1000]
tdfield = presets.time.periodic_envelope_propagation(u,p.omega_0).transpose([pc.Symbol('t'),s.x,s.z])
wg_field = tdfield[:,-wg.r:wg.r,:]
radius_wg = expression_to_array(abs(s.x/wg.r),settings)[-wg.r:wg.r].real
int_field = abs(wg_field)**2
int_field.data.transpose((2,0,1))[:] *= radius_wg.data
```

```
circ_w,circ_i = get_wi(int_field.sum(axis=s.x))

# Homogeneous medium
s.n = wg.n_cladding
s.Nx = 4
s.Nz = 1000
s.zmax = 0.1*units.mm
propagator = propagators.Fresnel3D(settings)
u = propagator.run_slice()[-2*wg.r:2*wg.r:4*wg.r/1000,:,::s.sz/1000]
tdfield = presets.time.periodic_envelope_propagation(u,p.omega_0).transpose([pc.Symbol('t'),s.x,s.z])
int_field = abs(tdfield[:,0,:])**2
hom_w,hom_i = get_wi(int_field)
hom_dz = float(int_field._dbounds[1]/units.um)
hom_zvalues = np.array([i*hom_dz for i in range(int_field.shape[1])])

# Plot refractive index
E_plot = expression_to_array(s.E/units.keV,settings).data.real
n_plot = expression_to_array(wg.n_cladding,settings).data
u0_plot = expression_to_array(s.u0,settings).data
fig,ax3 = plt.subplots(figsize=(5,3))
ax4 = ax3.twinx()
ax3.plot(E_plot,(n_plot.real-1)*10**6,label='$\Re(n) - 1$')
ax3.plot(E_plot,n_plot.imag*10**6,label='$\Im(n)$')
ax3.set_ylim(-25,1)
ax3.set_xlabel(r'$E \; [\mathrm{keV}]$')
ax3.set_ylabel(r'$n \; [10^{-6}]$')
ax3.legend(loc=6)
ax3.set_xticks([8.325,8.33,8.335,8.34])
ax3.set_xlim(8.323,8.342)
ax4.plot(E_plot,abs(u0_plot)**2,color='C3',label='$|\psi_0|^2$')
ax4.set_ylabel(r'$|\psi_0|^2\;[\mathrm{AU}]$')
ax4.set_ylim(-0.1,1.3)
ax4.legend(loc=5)
plt.tight_layout()
plt.savefig('plots/time_dependent/Ni_8keV_300as.pdf',transparent=True)

# Plot beam shape
fig,(ax1,ax2) = plt.subplots(1,2,figsize=(8,3),sharey=True)
ax1.plot(E_plot,abs(u[0,:,0].data)**2,label='$|\psi(z=0)|^2$')
ax1.plot(E_plot,abs(u[0,:,40*units.um].data)**2/(abs(u[0,:,40*units.um].data)**2).max(),label='$|\psi(z=40
    \;\mu\mathrm{m})|^2$')
ax1.set_xlabel(r'$E \; [\mathrm{keV}]$')
ax1.set_xticks([8.32 + 0.005 * i for i in range(10)])
ax1.set_xlim(8.328,8.343)
ax1.legend()
plot(centered(int_field[:,0]/int_field[:,0].max()),ax=ax2,label=r'$\bar{I}_\mathrm{inst}(z=0)$')
plot(centered(int_field[:,40*units.um]/int_field[:,40*units.um].max()),ax=ax2,label=r'$\bar{I}_\mathrm{inst}
    (z=40\;\mu\mathrm{m})$')
ax2.set_xlim(19,35)
ax2.legend(loc=1)
plt.tight_layout()
plt.savefig('plots/time_dependent/Ni_propagation_pulse_shape_and_spectrum.pdf',transparent=True)

# Plot widths
fig,(ax1,ax2,ax3) = plt.subplots(1,3,sharey=True,figsize=(8,3))
ax1.plot(hom_zvalues,hom_w/hom_w[0],label=r'$\Sigma_t(z) / \Sigma_t(0)$')
ax1.plot(hom_zvalues,hom_i/hom_i[0],label=r'$A(z) / A(0)$')
ax1.set_xlim(0,40)
ax1.legend(loc=5)
ax1.set_xlabel(r'$z\;[\mu \mathrm{m}]$')
ax2.plot(zvalues,slab_w/slab_w[0],label=r'$\Sigma_t(z) / \Sigma_t(0)$')
ax2.plot(zvalues,slab_i/slab_i[0],label=r'$A(z) /A(0)$')
ax2.legend(loc=5)
ax2.set_xlabel(r'$z\;[\mathrm{mm}]$')
ax2.set_xlim(0,50)
ax3.plot(zvalues,circ_w/circ_w[0],label=r'$\Sigma_t(z) / \Sigma_t(0)$')
ax3.plot(zvalues,circ_i/circ_i[0],label=r'$A(z) / A(0)$')
ax3.legend(loc=5)
ax3.set_xlabel(r'$z\;[\mathrm{mm}]$')
ax3.set_xlim(0,20)
plt.tight_layout()
plt.savefig('plots/time_dependent/pulse_width_Ni_absorption_edge_100nm_wg_comparison.pdf',transparent=True)
```

DANKSAGUNG

An dieser Stelle möchte ich allen danken, die zum Gelingen dieser Masterarbeit beigetragen haben.

Ein ganz besonderer Dank geht dabei an **Prof. Dr. Tim Salditt**, der es mir erst ermöglicht hat meine Masterarbeit über dieses spannende Thema zu schrieben. Die konstruktiven Diskussionen und seine Begeisterung für das Thema haben mir erlaubt neue Ideen zu entwickeln und waren zudem eine ständige Motivationsquelle. Danke auch an **Prof. Dr. Claus Ropers** für seine Bereitschaft diese Arbeit als Zweitgutachter zu bewerten.

Ich danke meinen Bürokollegen **Jasper Frohn**, **Stanislav Hrivňak**, **Karlo Komorowski** und **Natalie Nowak**, die für eine tolle Arbeitsatmosphäre gesorgt haben und stets für interessante und konstruktive Diskussionen zu haben waren.

Danke an **Dipl.-Geol. Jan Goeman** für seinen unermüdbaren Einsatz alle Rechner des Instituts instand zu halten und mit benötigter Software zu versorgen.

Weiterhin danke ich **Dipl.-Phys. Sarah Hoffmann-Urlaub** für ihr ausführliches und sehr konstruktives Feedback zu dieser Arbeit.

Abschließend möchte ich mich noch bei meiner Familie bedanken, die mir durch ihre Unterstützung und ihr Vertrauen mein Studium und schließlich auch diese Masterarbeit ermöglicht haben.

DECLARATION

Hiermit erkläre ich, dass ich diese Abschlussarbeit selbständig verfasst habe, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe und alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten Schriften entnommen wurden, als solche kenntlich gemacht habe.

Darüberhinaus erkläre ich, dass diese Abschlussarbeit nicht, auch nicht auszugsweise, im Rahmen einer nichtbestandenen Prüfung an dieser oder einer anderen Hochschule eingereicht wurde.

Göttingen, den 2. April 2017.

Lars Melchior