December 24, 2011 at 11:20

## 1. Introduction.

# MagBragg

A simulator of nonlinear magneto-optical Bragg gratings
(Version 1.44 of January 10, 2007)

Written by Fredrik Jonsson

This CWEB[†] computer program calculates reflection and transmission spectra of nonlinear magneto-optical Bragg gratings, in a stratified geometry where the material parameters vary only in one Cartesian coordinate. The MAGBRAGG program also simulates the propagation of the electromagnetic field of an optical wave as it traverses a magneto-optical Bragg grating, in linear as well as nonlinear optical regimes.

Strictly speaking, in a linear-optical domain a forward algorithm would just as fine as a backward one; however, for nonlinear optical problems, it often (as in this particular case) turns out that it is easier to compute the inverse problem, that is to say, to compute the input that corresponds to a certain given output. This is the case for, for example, optical bistability, where a given input intensity and ellipticity of the input optical wave for certain configurations correspond to a multiple-valued optical output. This means that we are not always on safe ground when it comes to the evaluation of output as function of input; meantime it makes perfectly sense to calculate the single-valued input as function of optical output. This program, in particular, is formulated in terms of the inverse problem, and hence the input parameters to the program are partly given in terms of the optical output of the magneto-optical Bragg grating. The algorithm behind the program was published as F. Jonsson and C. Flytzanis, Physical Review Letters **96**, 063902 (2006).

[†] For information on the CWEB programming language by Donald E. Knuth, as well as samples of CWEB programs, see `http://www-cs-faculty.stanford.edu/~knuth/cweb.html`. For general information on literate programming, see `http://www.literateprogramming.com`.

**2. The CWEB programming language.** For the reader who might not be familiar with the concept of the CWEB programming language, the following citations hopefully will be useful. For further information, as well as freeware compilers for compiling CWEB source code, see `http://www.literateprogramming.com`.

*I believe that the time is ripe for significantly better documentation of programs, and that we can best achieve this by considering programs to be works of literature. Hence, my title: 'Literate Programming.'*

*Let us change our traditional attitude to the construction of programs: Instead of imagining that our main task is to instruct a computer what to do, let us concentrate rather on explaining to human beings what we want a computer to do.*

*The practitioner of literate programming can be regarded as an essayist, whose main concern is with exposition and excellence of style. Such an author, with thesaurus in hand, chooses the names of variables carefully and explains what each variable means. He or she strives for a program that is comprehensible because its concepts have been introduced in an order that is best for human understanding, using a mixture of formal and informal methods that reinforce each other.*

– Donald Knuth, *The CWEB System of Structured Documentation* (Addison-Wesley, Massachusetts, 1994)

*The philosophy behind CWEB is that an experienced system programmer, who wants to provide the best possible documentation of his or her software products, needs two things simultaneously: a language like TEX for formatting, and a language like C for programming. Neither type of language can provide the best documentation by itself; but when both are appropriately combined, we obtain a system that is much more useful than either language separately.*

*The structure of a software program may be thought of as a 'WEB' that is made up of many interconnected pieces. To document such a program we want to explain each individual part of the web and how it relates to its neighbors. The typographic tools provided by TEX give us an opportunity to explain the local structure of each part by making that structure visible, and the programming tools provided by languages like C make it possible for us to specify the algorithms formally and unambiguously. By combining the two, we can develop a style of programming that maximizes our ability to perceive the structure of a complex piece of software, and at the same time the documented programs can be mechanically translated into a working software system that matches the documentation.*

*Besides providing a documentation tool, CWEB enhances the C language by providing the ability to permute pieces of the program text, so that a large system can be understood entirely in terms of small sections and their local interrelationships. The CTANGLE program is so named because it takes a given web and moves the sections from their web structure into the order required by C; the advantage of programming in CWEB is that the algorithms can be expressed in "untangled" form, with each section explained separately. The CWEAVE program is so named because it takes a given web and intertwines the TEX and C portions contained in each section, then it knits the whole fabric into a structured document.*

– Donald Knuth, "Literate Programming", in *Literate Programming* (CSLI Lecture Notes, Stanford, 1992)

**3. Theory of nonlinear magneto-optical Bragg gratings.** The algorithm used for the simulation is based on the inverse problem of nonlinear optical gratings, where the transmitted optical field is used as input to the algorithm, which successively calculates the intra-grating optical fields backwards through the structure, finally ending up with the corresponding incident field. The analysis is here for the sake of simplicity restricted to infinite plane waves under assumption of the slowly varying envelope approximation. The conventions of notation for the discrete layers in the structure of analysis are shown in Fig. 1, in which $N-1$ layers of nonlinear magneto-optical media are separated by their boundaries at $z = z_1, \ldots, z_N$.

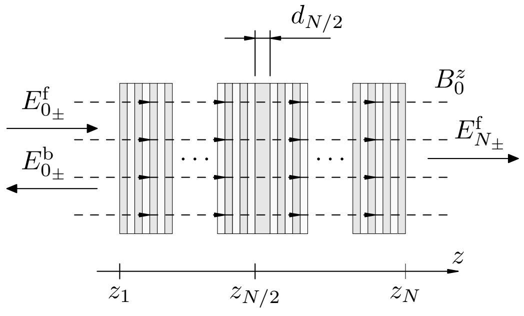

**Figure 1.** The geometry of the discretized model of the magneto-optical Bragg grating, in the Faraday configuration with the static magnetic $\mathbf{B}_0 = B_0^z \mathbf{e}_z$ applied in the direction of wave propagation. The grating consists of $N$ optically homogeneous layers with $N+1$ discrete interfaces located at spatial coordinates $z_k$, for $k = 1, 2, \ldots, N$, with the optical properties of each layer described by the optical susceptibility tensors $\chi^{\mathrm{ee}}_{ij}$ and $\chi^{\mathrm{eeee}}_{ijkl}$ and their magneto-optical counterparts $\chi^{\mathrm{eem}}_{ijk}$ and $\chi^{\mathrm{eeeem}}_{ijklm}$.

**4.** Discretization of the problem. In each homogeneous layer $z_k < z < z_{k+1}$, $k = 1, 2, \ldots, N-1$, the components of the plane-parallel electromagnetic field $\mathbf{E}(z,t) = \mathrm{Re}[\mathbf{E}_\omega \exp(-i\omega t)]$ propagate collinearly with an externally applied static magnetic field $\mathbf{B}_0 = B_0^z \mathbf{e}_z$, and the local constitutive relation for the electric polarization density $\mathbf{P}(z,t) = \mathrm{Re}[\mathbf{P}_\omega \exp(-i\omega t)]$ of the medium is [1]

$$\begin{aligned}\mathbf{P}_\omega = \varepsilon_0[&\chi^{\mathrm{ee}}_{xx}\mathbf{E}_\omega + \chi^{\mathrm{eem}}_{xyz}\mathbf{E}_\omega \times \mathbf{B}_0\\ &+ (\chi^{\mathrm{eeee}}_{xxxx} - \chi^{\mathrm{eeee}}_{xyyx})(\mathbf{E}_\omega \cdot \mathbf{E}^*_\omega)\mathbf{E}_\omega + \chi^{\mathrm{eeee}}_{xyyx}(\mathbf{E}_\omega \cdot \mathbf{E}_\omega)\mathbf{E}^*_\omega\\ &+ \chi^{\mathrm{eeeem}}_{xyyyz}(\mathbf{E}_\omega \cdot \mathbf{E}^*_\omega)\mathbf{E}_\omega \times \mathbf{B}_0 + \chi^{\mathrm{eeeem}}_{xxxyz}\mathbf{E}_\omega(\mathbf{E}_\omega \cdot (\mathbf{E}^*_\omega \times \mathbf{B}_0))].\end{aligned} \tag{1}$$

In this expression, the spatial dependence of the fields and susceptibilities is omitted in notation, as is also any slow temporal variation of the fields; from now on each appearance of a field or susceptibility implicitly implies evaluation in respective layer of context. In each layer, the envelope of the electromagnetic field satisfies the autonomous nonlinear wave equation

$$\frac{\partial^2 \mathbf{E}_\omega}{\partial z^2} + (\omega/c)^2 \mathbf{E}_\omega = -\mu_0 \omega^2 \mathbf{P}_\omega. \tag{2}$$

The field is in each layer expressed in a circularly polarized basis $\mathbf{e}_\pm = (\mathbf{e}_x \pm i\mathbf{e}_y)/\sqrt{2}$, with forward and backward traveling modes as a superposition

$$\begin{aligned}\mathbf{E}_\omega = &[\mathbf{e}_+ E^{\mathrm{f}}_{k_+}(z) + \mathbf{e}_- E^{\mathrm{f}}_{k_-}(z)]\exp[i\omega n_k(z - z_k)/c]\\ &+ [\mathbf{e}^*_+ E^{\mathrm{b}}_{k_+}(z) + \mathbf{e}^*_- E^{\mathrm{b}}_{k_-}(z)]\exp[-i\omega n_k(z - z_k)/c],\end{aligned} \tag{3}$$

where $n_k = (1 + \chi^{\mathrm{ee}}_{xx})^{1/2}$ are the linear refractive indices of the layers. We here adopt to the convention of circularly polarized field components that when looking into the oncoming wave, a left circularly polarized wave will have its vector of the electric field rotating with counterclockwise motion, while the corresponding vector of a right circularly polarized field will describe clockwise motion. This convention applies irregardless of whether we are looking into a forward propagating wave, that is to say looking in the negative $z$-direction, or looking into a backward propagating wave, in which case we instead look in the positive $z$-direction, along the $z$-axis. This convention conforms to the classical one as commonly used in optics [5]. Throughout this program, the notation $E^{\mathrm{f}}_{k_\pm}(z)$ is used to denote the left/right circularly polarized components of the forward traveling component of the electric field in the $k$th homogeneous layer, while $E^{\mathrm{b}}_{k_\pm}(z)$ in an analogous manner is used to denote the left/right circularly polarized components of the backward traveling component of the electric field, taking into account the previously described convention for circular polarization states.

**5.** General solutions in the homogeneous layers. By inserting the form of the electric field as given by Eq. (3) into the wave equation as given by Eq. (1) under the constitutive relation given by Eq. (1) and furthermore applying the slowly varying envelope approximation, the general solutions for the field components in loss-less media become [1]

$$E^{\mathrm{f}}_{k\pm} = A^{\mathrm{f}}_{k\pm} \exp[i(\omega/c)(\eta^{\mathrm{f}}_{k\pm} \pm g_k)(z - z_k) + i\psi^{\mathrm{f}}_{k\pm}], \tag{4a}$$

$$E^{\mathrm{b}}_{k\pm} = A^{\mathrm{b}}_{k\pm} \exp[-i(\omega/c)(\eta^{\mathrm{b}}_{k\pm} \mp g_k)(z - z_k) + i\psi^{\mathrm{b}}_{k\pm}]. \tag{4b}$$

In these expressions $A^{\mathrm{f,b}}_{k\pm}$ and $\psi^{\mathrm{f,b}}_{k\pm}$ are real constants of integration, determined by the boundary conditions of the $k$th layer, and $g_k = i\chi^{\mathrm{eem}}_{xyz}B^z_0/(2n_k)$ are the real linear magneto-optical gyration coefficients. In Eqs. (4) the nonlinear optical and magneto-optical light-matter interactions are described by the field-dependent parameters

$$\eta^{\mathrm{f}}_{k\pm} = p_{k\pm}(A^{\mathrm{f}\,2}_{k\pm} + 2A^{\mathrm{b}\,2}_{k\mp}) + q_{k\pm}(A^{\mathrm{f}\,2}_{k\mp} + A^{\mathrm{b}\,2}_{k\pm}), \tag{5b}$$

$$\eta^{\mathrm{b}}_{k\pm} = p_{k\mp}(A^{\mathrm{b}\,2}_{k\pm} + 2A^{\mathrm{f}\,2}_{k\mp}) + q_{k\mp}(A^{\mathrm{b}\,2}_{k\mp} + A^{\mathrm{f}\,2}_{k\pm}), \tag{5b}$$

where the coefficients $p_{k\pm}$ and $q_{k\pm}$ in the nonlinear susceptibility formalism [3] are expressed as

$$p_{k\pm} = \frac{3}{8n_k}[\chi^{\mathrm{eeee}}_{xxxx} - \chi^{\mathrm{eeee}}_{xyyx} \pm i(\chi^{\mathrm{eeeem}}_{xyyyz} - \chi^{\mathrm{eeeem}}_{xxxyz})B^z_0], \tag{6b}$$

$$q_{k\pm} = \frac{3}{8n_k}[\chi^{\mathrm{eeee}}_{xxxx} + \chi^{\mathrm{eeee}}_{xyyx} \pm i(\chi^{\mathrm{eeeem}}_{xyyyz} + \chi^{\mathrm{eeeem}}_{xxxyz})B^z_0]. \tag{6b}$$

**6.** Rotation of the polarization state across the homogeneous layers. The angle of rotation $\vartheta_k$ of the polarization ellipse of the forward traveling components around the $z$-axis in each layer is then

$$\begin{aligned}\vartheta_k = \frac{\omega(z - z_k)}{2n_k c}[&-i\chi^{\mathrm{eem}}_{xyz}B^z_0 + (3/4)\chi^{\mathrm{eeee}}_{xyyx}(A^{\mathrm{f}\,2}_{k_+} - A^{\mathrm{f}\,2}_{k_-}) + (3/8)(\chi^{\mathrm{eeee}}_{xxxx} - 3\chi^{\mathrm{eeee}}_{xyyx})(A^{\mathrm{b}\,2}_{k_+} - A^{\mathrm{b}\,2}_{k_-})\\ &- (3/4)i\chi^{\mathrm{eeeem}}_{xyyyz}(A^{\mathrm{f}\,2}_{k_+} + A^{\mathrm{f}\,2}_{k_-})B^z_0 + (3/8)i(\chi^{\mathrm{eeeem}}_{xxxyz} - 3\chi^{\mathrm{eeeem}}_{xyyyz})(A^{\mathrm{b}\,2}_{k_+} + A^{\mathrm{b}\,2}_{k_-})B^z_0].\end{aligned} \tag{7}$$

The first term in this expression describes linear Faraday rotation [4] and leads to an effective Zeeman-like polarization state splitting of the doubly degenerate Bragg resonances. The second and third terms arise from optical Kerr-effect and lead to photo-induced modification of the ellipse rotation for forward and backward propagating waves respectively. Referring to the wave equation given by Eqs. (1)–(2) these terms effectively act as to give photo-induced Stark-like shifts of the split Bragg resonances. Finally the fourth and fifth terms describe photo-induced magneto-optic Faraday rotation or equivalently photoinduced modification of the effective Zeeman-like splitting of the Bragg resonances. We note by passing that the third and fifth terms vanish whenever Kleinman symmetry [3] holds. Moreover, from Eq. (7) one obtains the Verdet coefficient $V$ of the medium as

$$V = \frac{\omega\,\mathrm{Im}[\chi^{\mathrm{eem}}_{xyz}]}{2n_k c}, \tag{8}$$

describing the Faraday rotation angle per propagation length and static magnetic field intensity.

**7.** Boundary conditions at the discrete interfaces. By neglecting any nonlinear effects at the discrete interfaces between the homogeneous layers, the continuity requirement of the transverse electromagnetic field across each interface is formulated by the boundary conditions

$$E^{\rm f}_{k_\pm}(z_k) = \tau_{k_\pm} E^{\rm f}_{k-1_\pm}(z_k)\exp(i\omega n_{k-1}d_{k-1}/c) + \rho'_{k_\mp} E^{\rm b}_{k_\mp}(z_k), \tag{9a}$$

$$E^{\rm b}_{k_\mp}(z_{k+1})\exp(-i\omega n_k d_k/c) = \tau'_{k+1_\mp} E^{\rm b}_{k+1_\mp}(z_{k+1}) + \rho_{k+1_\pm} E^{\rm f}_{k_\pm}(z_{k+1})\exp(i\omega n_k d_k/c), \tag{9b}$$

for $k = 1, 2, \ldots, N-1$, where $d_k = z_{k+1} - z_k$ are the layer thicknesses, and where

$$\rho_{k_\pm} = \frac{n_{k-1} - n_k \pm (g_{k-1} - g_k)}{n_{k-1} + n_k \pm (g_{k-1} + g_k)} = -\rho'_{k_\mp}, \tag{10a}$$

$$\tau_{k_\pm} = \frac{2(n_{k-1} \pm g_{k-1})}{n_{k-1} + n_k \pm (g_{k-1} + g_k)}, \tag{10b}$$

$$\tau'_{k_\mp} = \frac{2(n_k \pm g_k)}{n_{k-1} + n_k \pm (g_{k-1} + g_k)}, \tag{10c}$$

are the reflection and transmission coefficients at the interfaces, with primed coefficients relating to the backward travelling field components, and with all coefficients including linear magneto-optical effects. The reflection and transmission coefficients obey the Stokes relation

$$\tau_{k_\pm} = (1 - \rho^2_{k_\pm})/\tau'_{k_\mp},$$

reflecting the polarization state dependence of the boundary conditions at the layer interfaces. This linear approximation of the boundary conditions can be assumed to hold whenever processes such as the optical Kerr-effect gives a contribution to the electric polarization density which is small compared to the one given by the linear polarizability of the medium, or equivalently whenever the nonlinear index of refraction is small compared to the linear one, which for a linearly polarized optical wave can be formulated as the condition

$$(3/4)\chi^{\rm eeee}_{xxxx}|E^x_\omega|^2 \ll 1 + \chi^{\rm ee}_{xx}.$$

**8.** Solving the equations of motion for the inverse problem of transmission. In order to solve Eqs. (4) and (9) for the relation between incident ($k = 0$) and transmitted ($k = N$) fields, we impose the additional boundary condition that the backward propagating field is zero at the exit end of the grating. By iteratively using Eqs. (9) and (4) in the order $k = N-1, N-2, \ldots, 1$, consecutively solving for lower-indexed electric fields, we obtain an algorithm for solving the inverse problem, calculating the incident and reflected optical fields corresponding to a transmitted optical field. This algorithm is explicitly described in Section 10, *The algorithm of computation.*

**9.** Interpretation of the solution in terms of the Stokes parameters. The evolution of the optical field is conveniently expressed in terms of the Stokes parameters [5], which for the input light are taken as

$$S_0 = |E^{\rm f}_{0+}|^2 + |E^{\rm f}_{0-}|^2, \qquad S_1 = 2\,{\rm Re}[E^{\rm f*}_{0+} E^{\rm f}_{0-}], \tag{11a, b}$$

$$S_3 = |E^{\rm f}_{0+}|^2 - |E^{\rm f}_{0-}|^2, \qquad S_2 = 2\,{\rm Im}[E^{\rm f*}_{0+} E^{\rm f}_{0-}]. \tag{11c, d}$$

The parameter $S_0$ is a measure of the intensity of the wave, and $S_3$ a measure of the ellipticity of the polarization state, while the parameters $S_1$ and $S_2$ indicate the amount of power contained in the $x$- and $y$-directions, serving as to determine the orientation of the polarization ellipse. Similarly, the Stokes parameters for the transmitted light are taken as

$$W_0 = |E^{\rm f}_{N+}|^2 + |E^{\rm f}_{N-}|^2, \qquad W_1 = 2\,{\rm Re}[E^{\rm f*}_{N+} E^{\rm f}_{N-}], \tag{12a, b}$$

$$W_3 = |E^{\rm f}_{N+}|^2 - |E^{\rm f}_{N-}|^2, \qquad W_2 = 2\,{\rm Im}[E^{\rm f*}_{N+} E^{\rm f}_{N-}], \tag{12c, d}$$

and for the reflected light

$$V_0 = |E^{\rm b}_{0+}|^2 + |E^{\rm b}_{0-}|^2, \qquad V_1 = 2\,{\rm Re}[E^{\rm b*}_{0+} E^{\rm b}_{0-}], \tag{13a, b}$$

$$V_3 = |E^{\rm b}_{0+}|^2 - |E^{\rm b}_{0-}|^2, \qquad V_2 = 2\,{\rm Im}[E^{\rm b*}_{0+} E^{\rm b}_{0-}]. \tag{13c, d}$$

In terms of $S_0$ and $W_0$, the incident, transmitted, and reflected intensities in SI units are $I_{\rm in} = \varepsilon_0 c S_0/2$, $I_{\rm tr} = \varepsilon_0 c W_0/2$, and $I_{\rm re} = \varepsilon_0 c V_0/2$, respectively. These respective sets of Stokes parameters can be combined to form the reduced Stokes vectors

$$\mathbf{s} = (S_1, S_2, S_3)/S_0, \qquad \mathbf{w} = (W_1, W_2, W_3)/W_0, \qquad \mathbf{v} = (V_1, V_2, V_3)/V_0,$$

which map the polarization states of the respective incident, transmitted, and reflected light onto the unitary Poincaré sphere, with $|\mathbf{s}| = |\mathbf{w}| = |\mathbf{v}| = 1$, as schematically illustrated in Fig. 2.

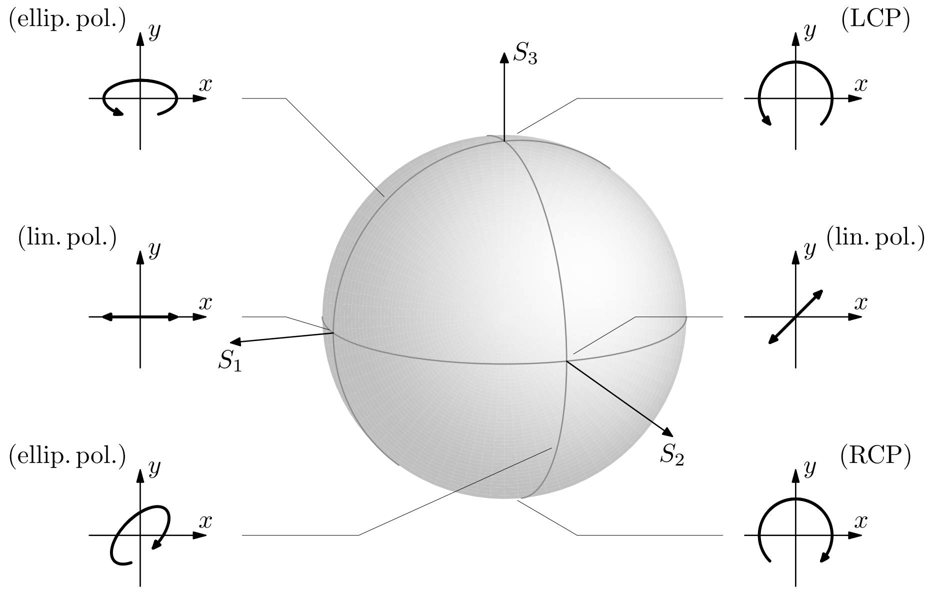

**Figure 2.** The unitary Poincaré sphere, on which the normalized Stokes vectors $\mathbf{s} = (S_1, S_2, S_3)/S_0$, $\mathbf{w} = (W_1, W_2, W_3)/W_0$, or $\mathbf{v} = (V_1, V_2, V_3)/V_0$ provide representations of the polarization state of the light. The auxiliary figures show the paths traversed by the end point of the electric field vector as the observer looks into an oncoming wave propagating in the positive $z$-direction. At the upper pole of the sphere the light is left circularly polarized (LCP), while it at the lower pole is right

circularly polarized (RCP), with the latitude there in between determining the ellipticity of the polarization state. At the equator, linearly polarized states of the light are represented, and their orientations are determined by the meridian. The figure was adopted from Ref. [9].

The described algorithm of calculation as described by iteratively using Eqs. (4)–(9) can be applied to a wide range of magneto-optical gratings, singly or multiply periodic, chirped, or even random without appreciable complication, by employing a spatial oversampling of the continuous grating profiles in terms of thin layers, so as to provide an efficient finite difference scheme of computation. Examples of computations performed on magneto-optical structures of continuous spatial distribution of the optical and magneto-optical parameters can be found in Refs.[6–8]

**10. The algorithm of computation.** With the theory outlined we are now in position to explicitly describe the algorithm of computation of the transmission and reflection properties of the grating, as obtained by iteratively using Eqs. (9) and (4) in the order $k = N-1, N-2, \ldots, 1$, consecutively solving for lower-indexed electric fields. This algorithm follows the one as described in Ref. [2].

**Algorithm A** (*Calculation of amplitude reflection and transmission spectra*). The notation of geometry as used in the algorithm as here described refers to Fig. 1. Let $E^{\rm f}_{N_\pm}(z_N)$ be the complex-valued output optical field from the grating structure, as taken a coordinate $z = z^+_N$ immediately to the positive of the last interface $z_N$. The magneto-optical grating structure is described by the material parameters $n_k$, $g_k$, $p^{(k)}_\pm$, and $q^{(k)}_\pm$, which all are constant within each homogeneous layer $z_k \le z \le z_{k+1}$, $k = 1, 2, \ldots, N-1$. The optical fields in the respective homogeneous layers are denoted as $E^{\rm f}_{k_\pm}(z)$, for $z_k \le z \le z_{k+1}$, $k = 1, 2, \ldots, N-1$.

**A1.** [Set boundary condition.] Apply the boundary condition $E^{\rm b}_{N_\mp}(z_N) = 0$, that is to say, that the backward propagating field is zero after the end of the grating.

**A2.** [Initialize forward field in layer $N-1$.] Using Eq. (9a) for $k = N$ and furthermore applying the boundary condition in step A1 of the algorithm, set

$$E^{\rm f}_{N-1_\pm}(z_N) \leftarrow [E^{\rm f}_{N_\pm}(z_N)/\tau_{N_\pm}] \exp[-i\omega n_{N-1}(z_N - z_{N-1})/c],$$

where $\tau_{N_\pm}$ is the transmission coefficient as obtained from Eq. (10b).

**A3.** [Initialize backward field in layer $N-1$.] Using Eq. (9b) for $k = N-1$ and furthermore applying the boundary condition from A1 and the field calculated in A2, set

$$E^{\rm b}_{N-1_\mp}(z_N) \leftarrow \rho_{(N)_\pm} E^{\rm f}_{N-1_\pm}(z_N) \exp[2i\omega n_{N-1}(z_N - z_{N-1})/c],$$

where $\rho_{(N)_\pm}$ is the reflection coefficient as obtained from Eq. (10a).

**A4.** [Initialize layer counter $k$.] Set $k \leftarrow N-1$. This is the counter which we use to keep track of the layers of the grating structure as this is traversed backwards, from the last layer ($k = N-1$) towards the first one ($k = 1$).

**A5.** [Calculate propagation constants in the $k$th layer.] For a lossless medium the magnitudes of the circularly polarized field components are left invariant under propagation over a homogeneous layer,

$$|E^{\rm f,b}_{k_\pm}(z_k)| = |E^{\rm f,b}_{k_\pm}(z_{k+1})|;$$

and hence the field-dependent propagation constants $\eta^{\rm f}_{k_\pm}$ and $\eta^{\rm b}_{k_\pm}$ of the layer $z^+_k \le z \le z^-_{k+1}$ are calculated using Eqs. (5) as

$$\begin{aligned}
\eta^{\rm f}_{k_\pm} &\leftarrow p^{(k)}_\pm[|E^{\rm f}_{k_\pm}(z_{k+1})|^2 + 2|E^{\rm b}_{k_\mp}(z_{k+1})|^2] + q^{(k)}_\pm[|E^{\rm f}_{k_\mp}(z_{k+1})|^2 + |E^{\rm b}_{k_\pm}(z_{k+1})|^2],\\
\eta^{\rm b}_{k_\pm} &\leftarrow p^{(k)}_\mp[|E^{\rm b}_{k_\pm}(z_{k+1})|^2 + 2|E^{\rm f}_{k_\mp}(z_{k+1})|^2] + q^{(k)}_\mp[|E^{\rm b}_{k_\mp}(z_{k+1})|^2 + |E^{\rm f}_{k_\pm}(z_{k+1})|^2].
\end{aligned}$$

**A6.** [Propagate fields over the $k$th layer.] From the obtained complex-valued fields $E^{\rm f}_{k_\pm}(z_{k+1})$ and $E^{\rm b}_{k_\mp}(z_{k+1})$, calculate the corresponding fields at the beginning $z = z^+_k$ of the $k$th layer by using Eqs. (4), that is to say, set

$$\begin{aligned}
A^{\rm f}_{k_\pm} &\leftarrow |E^{\rm f}_{k_\pm}(z_{k+1})|,\\
A^{\rm b}_{k_\pm} &\leftarrow |E^{\rm b}_{k_\pm}(z_{k+1})|,\\
\psi^{\rm f}_{k_\pm} &\leftarrow \arg[E^{\rm f}_{k_\pm}(z_{k+1})] - \omega(\eta^{\rm f}_{k_\pm} \pm g_k)(z_{k+1} - z_k)/c,\\
\psi^{\rm b}_{k_\pm} &\leftarrow \arg[E^{\rm b}_{k_\pm}(z_{k+1})] + \omega(\eta^{\rm b}_{k_\pm} \mp g_k)(z_{k+1} - z_k)/c,
\end{aligned}$$

and construct the fields $E^{\rm f}_{k_\pm}(z_k)$ and $E^{\rm b}_{k_\pm}(z_k)$ (taken immediately to the positive of $z_k$ in the $k$th layer) as

$$\begin{aligned}
E^{\rm f}_{k_\pm}(z_k) &\leftarrow A^{\rm f}_{k_\pm} \exp(i\psi^{\rm f}_{k_\pm}),\\
E^{\rm b}_{k_\pm}(z_k) &\leftarrow A^{\rm b}_{k_\pm} \exp(i\psi^{\rm b}_{k_\pm}).
\end{aligned}$$

**A7.** [Check if the whole grating structure is traversed.] If the whole grating is traversed, that is to say if the layer counter has reached $k = 1$, then proceed to A11; otherwise continue with A8.

**A8.** [Propagate forward traveling field components over interface located at $z = z_k$.] From the obtained complex-valued optical fields $E^{\rm f}_{k_\pm}(z_k)$ and $E^{\rm b}_{k_\mp}(z_k)$, use Eq. (9a) to calculate the corresponding forward propagating field at the negative side of the interface, at $z = z_k^-$, that is to say, set

$$E^{\rm f}_{k-1_\pm}(z_k) \leftarrow [E^{\rm f}_{k_\pm}(z_k) - \rho'_{k_\mp} E^{\rm b}_{k_\mp}(z_k)] \exp[-i\omega n_{k-1}(z_k - z_{k-1})/c]/\tau^{(k)}_\pm .$$

**A9.** [Propagate backward traveling field components over interface located at $z = z_k$.] From the obtained complex-valued optical fields $E^{\rm f}_{k_\pm}(z_k)$ and $E^{\rm b}_{k_\mp}(z_k)$, use Eq. (9b) to calculate the corresponding backward propagating field at the negative side of the interface, at $z = z_k^-$, that is to say, set

$$E^{\rm b}_{k-1_\mp}(z_k) \leftarrow \tau^{(k)\prime}_\mp E^{\rm b}_{k_\mp}(z_k) \exp[i\omega n_{k-1}(z_k - z_{k-1})/c] + \rho^{(k)}_\pm E^{\rm f}_{k-1_\pm}(z_k) \exp[2i\omega n_{k-1}(z_k - z_{k-1})/c].$$

**A10.** [Decrease layer counter.] Set $k \leftarrow k - 1$ and return to A5.

**A11.** [Calculate input optical field.] From the obtained forward and backward propagating optical fields in the first ($k = 1$) layer of the grating, calculate the corresponding input (forward propagating) field, that is to say, set

$$E^{\rm f}_{0_\pm}(z_1) \leftarrow [E^{\rm f}_{1_\pm}(z_1) - \rho^{(1)\prime}_\mp E^{\rm b}_{1_\mp}(z_1)]/\tau^{(1)}_\pm .$$

**A12.** [Calculate reflected optical field.] From the obtained forward and backward propagating optical fields in the first ($k = 1$) layer of the grating, and by using the calculated input optical field of A11, calculate the reflected optical field at the beginning of the grating, that is to say, set

$$E^{\rm b}_{0_\mp}(z_1) \leftarrow \tau^{(1)\prime}_\mp E^{\rm b}_{1_\mp}(z_1) + \rho^{(1)}_\pm E^{\rm f}_{0_\pm}(z_1).$$

**A13.** [Calculate reflection and transmission coefficients.] Calculate any overall properties of the grating, such as the (intensity and polarization-state dependent) reflection and transmission

$$R(\omega) \leftarrow \frac{|E^{\rm b}_{0_+}(z_1)|^2 + |E^{\rm b}_{0_-}(z_1)|^2}{|E^{\rm f}_{0_+}(z_1)|^2 + |E^{\rm f}_{0_-}(z_1)|^2}, \qquad T(\omega) \leftarrow \frac{|E^{\rm f}_{N_+}(z_N)|^2 + |E^{\rm f}_{N_-}(z_N)|^2}{|E^{\rm f}_{0_+}(z_1)|^2 + |E^{\rm f}_{0_-}(z_1)|^2},$$

and terminate the algorithm. ▌

**11. The Butcher and Cotter convention.** As a “recipe” for the degeneracy factors in theoretical nonlinear optics, Butcher and Cotter [10] provide a very useful convention which is well worth holding on to. For a superposition of monochromatic waves, and by invoking the general property of the intrinsic permutation symmetry, the monochromatic form of the $n$th order polarization density can be written as

$$(P^{(n)}_{\omega_\sigma})_\mu = \varepsilon_0 \sum_{\alpha_1} \cdots \sum_{\alpha_n} \sum_{\omega} K(-\omega_\sigma;\omega_1,\ldots,\omega_n)\chi^{(n)}_{\mu\alpha_1\cdots\alpha_n}(-\omega_\sigma;\omega_1,\ldots,\omega_n)(E_{\omega_1})_{\alpha_1}\cdots(E_{\omega_n})_{\alpha_n}. \tag{14}$$

The first summations in Eq. (14), over $\alpha_1,\ldots,\alpha_n$, is simply an explicit way of stating that the Einstein convention of summation over repeated indices holds. The summation sign $\sum_\omega$, however, serves as a reminder that the expression that follows is to be summed over *all distinct sets of* $\omega_1,\ldots,\omega_n$. Because of the intrinsic permutation symmetry, the frequency arguments appearing in Eq. (14) may be written in arbitrary order.

By “all distinct sets of $\omega_1,\ldots,\omega_n$”, we here mean that the summation is to be performed, as for example in the case of optical Kerr-effect, over the single set of nonlinear susceptibilities that contribute to a certain angular frequency as $(-\omega;\omega,\omega,-\omega)$ *or* $(-\omega;\omega,-\omega,\omega)$ *or* $(-\omega;-\omega,\omega,\omega)$. In this example, each of the combinations are considered as *distinct*, and it is left as an arbitary choice which one of these sets that are most convenient to use (this is simply a matter of choosing notation, and does not by any means change the description of the interaction).

In Eq. (14), the degeneracy factor $K$ is formally described as

$$K(-\omega_\sigma;\omega_1,\ldots,\omega_n) = 2^{l+m-n}p$$

where

$$\begin{aligned} p &= \text{the number of } \textit{distinct} \text{ permutations of } \omega_1,\omega_2,\ldots,\omega_1,\\ n &= \text{the order of the nonlinearity},\\ m &= \text{the number of angular frequencies } \omega_k \text{ that are zero, and}\\ l &= \begin{cases} 1, & \text{if } \omega_\sigma \neq 0,\\ 0, & \text{otherwise.}\end{cases}\end{aligned}$$

In other words, $m$ is the number of DC electric fields present, and $l = 0$ if the nonlinearity we are analyzing gives a static, or DC, polarization density, such as in the previously (in the spring model) described case of optical rectification in the presence of second harmonic fields (SHG).

A list of frequently encountered nonlinear phenomena in nonlinear optics, including the degeneracy factors as conforming to the above convention, is given by Butcher and Cotter [10], Table 2.1.

**12.** Note on the complex representation of the optical field. Since the observable electric field of the light, in Butcher and Cotters notation taken as

$$\mathbf{E}(\mathbf{r},t) = \frac{1}{2}\sum_{\omega_k\geq 0}[\mathbf{E}_{\omega_k}\exp(-i\omega_k t) + \mathbf{E}^*_{\omega_k}\exp(i\omega_k t)],$$

is a real-valued quantity, it follows that negative frequencies in the complex notation should be interpreted as the complex conjugate of the respective field component, or

$$\mathbf{E}_{-\omega_k} = \mathbf{E}^*_{\omega_k}.$$

**13.** Example of degeneracy factor: The Optical Kerr-effect. Assume a monochromatic optical wave (containing forward and/or backward propagating components) polarized in the $xy$-plane,

$$\mathbf{E}(z,t) = \mathrm{Re}[\mathbf{E}_\omega(z)\exp(-i\omega t)] \in \mathbf{R}^3,$$

with all spatial variation of the field contained in

$$\mathbf{E}_\omega(z) = \mathbf{e}_x E^x_\omega(z) + \mathbf{e}_y E^y_\omega(z) \in \mathbf{C}^3.$$

Optical Kerr-effect is in isotropic media described by the third order susceptibility

$$\chi^{(3)}_{\mu\alpha\beta\gamma}(-\omega;\omega,\omega,-\omega),$$

with nonzero components of interest for the $xy$-polarized beam given in Appendix 3.3 of Butcher and Cotters book as

$$\chi^{(3)}_{xxxx} = \chi^{(3)}_{yyyy}, \quad \chi^{(3)}_{xxyy} = \chi^{(3)}_{yyxx} = \left\{ \begin{array}{c} \text{intr. perm. symm.} \\ (\alpha,\omega) \rightleftharpoons (\beta,\omega) \end{array} \right\} = \chi^{(3)}_{xyxy} = \chi^{(3)}_{yxyx}, \quad \chi^{(3)}_{xyyx} = \chi^{(3)}_{yxxy},$$

with

$$\chi^{(3)}_{xxxx} = \chi^{(3)}_{xxyy} + \chi^{(3)}_{xyxy} + \chi^{(3)}_{xyyx}.$$

The degeneracy factor $K(-\omega;\omega,\omega,-\omega)$ is calculated as

$$K(-\omega;\omega,\omega,-\omega) = 2^{l+m-n}p = 2^{1+0-3}3 = 3/4.$$

From this set of nonzero susceptibilities, and using the calculated value of the degeneracy factor in the convention of Butcher and Cotter, we hence have the third order electric polarization density at $\omega_\sigma = \omega$ given as $\mathbf{P}^{(n)}(\mathbf{r},t) = \mathrm{Re}[\mathbf{P}^{(n)}_\omega \exp(-i\omega t)]$, with

$$\begin{aligned}
\mathbf{P}^{(3)}_\omega &= \sum_\mu \mathbf{e}_\mu (P^{(3)}_\omega)_\mu \\
&= \{\text{Using the convention of Butcher and Cotter}\} \\
&= \sum_\mu \mathbf{e}_\mu \left[ \varepsilon_0 \frac{3}{4} \sum_\alpha \sum_\beta \sum_\gamma \chi^{(3)}_{\mu\alpha\beta\gamma}(-\omega;\omega,\omega,-\omega)(E_\omega)_\alpha (E_\omega)_\beta (E_{-\omega})_\gamma \right] \\
&= \{\text{Evaluate the sums over } (x,y,z) \text{ for field polarized in the } xy \text{ plane}\} \\
&= \varepsilon_0 \frac{3}{4} \{ \mathbf{e}_x [\chi^{(3)}_{xxxx} E^x_\omega E^x_\omega E^x_{-\omega} + \chi^{(3)}_{xyyx} E^y_\omega E^y_\omega E^x_{-\omega} + \chi^{(3)}_{xyxy} E^y_\omega E^x_\omega E^y_{-\omega} + \chi^{(3)}_{xxyy} E^x_\omega E^y_\omega E^y_{-\omega}] \\
&\qquad + \mathbf{e}_y [\chi^{(3)}_{yyyy} E^y_\omega E^y_\omega E^y_{-\omega} + \chi^{(3)}_{yxxy} E^x_\omega E^x_\omega E^y_{-\omega} + \chi^{(3)}_{yxyx} E^x_\omega E^y_\omega E^x_{-\omega} + \chi^{(3)}_{yyxx} E^y_\omega E^x_\omega E^x_{-\omega}] \} \\
&= \{\text{Make use of } \mathbf{E}_{-\omega} = \mathbf{E}^*_\omega \text{ and relations } \chi^{(3)}_{xxyy} = \chi^{(3)}_{yyxx}, \text{ etc.}\} \\
&= \varepsilon_0 \frac{3}{4} \{ \mathbf{e}_x [\chi^{(3)}_{xxxx} E^x_\omega |E^x_\omega|^2 + \chi^{(3)}_{xyyx} E^{y\,2}_\omega E^{x*}_\omega + \chi^{(3)}_{xyxy} |E^y_\omega|^2 E^x_\omega + \chi^{(3)}_{xxyy} E^x_\omega |E^y_\omega|^2] \\
&\qquad + \mathbf{e}_y [\chi^{(3)}_{xxxx} E^y_\omega |E^y_\omega|^2 + \chi^{(3)}_{xyyx} E^{x\,2}_\omega E^{y*}_\omega + \chi^{(3)}_{xyxy} |E^x_\omega|^2 E^y_\omega + \chi^{(3)}_{xxyy} E^y_\omega |E^x_\omega|^2] \} \\
&= \{\text{Make use of intrinsic permutation symmetry}\} \\
&= \varepsilon_0 \frac{3}{4} \{ \mathbf{e}_x [(\chi^{(3)}_{xxxx} |E^x_\omega|^2 + 2\chi^{(3)}_{xxyy} |E^y_\omega|^2) E^x_\omega + (\chi^{(3)}_{xxxx} - 2\chi^{(3)}_{xxyy}) E^{y\,2}_\omega E^{x*}_\omega \\
&\qquad \mathbf{e}_y [(\chi^{(3)}_{xxxx} |E^y_\omega|^2 + 2\chi^{(3)}_{xxyy} |E^x_\omega|^2) E^y_\omega + (\chi^{(3)}_{xxxx} - 2\chi^{(3)}_{xxyy}) E^{x\,2}_\omega E^{y*}_\omega .
\end{aligned}$$

For the optical field being linearly polarized, say in the $x$-direction, the expression for the polarization density is significantly simplified, to yield

$$\mathbf{P}^{(3)}_\omega = \varepsilon_0 (3/4) \mathbf{e}_x \chi^{(3)}_{xxxx} |E^x_\omega|^2 E^x_\omega,$$

i. e. taking a form that can be interpreted as an intensity-dependent ($\sim |E^x_\omega|^2$) contribution to the refractive index (cf. Butcher and Cotter §6.3.1).

## 14. Rigorous theory of wave propagation in isotropic media.

**15.** The tensor form of the electric polarizability. In the nonlinear susceptibility formalism, with the real-valued electric field and polarization density of the medium taken according to

$$\mathbf{E}(z,t) = \mathrm{Re}[\mathbf{E}_\omega \exp(-i\omega t)], \qquad \mathbf{P}(z,t) = \mathrm{Re}[\mathbf{P}_\omega \exp(-i\omega t)]$$

respectively, the complex-valued envelope for the polarization density is here given by the expression

$$\mathbf{P}_\omega = \varepsilon_0[\mathbf{e}_\mu \chi^{\mathrm{ee}}_{\mu\alpha} E^\alpha_\omega + \mathbf{e}_\mu \chi^{\mathrm{eem}}_{\mu\alpha\beta} E^\alpha_\omega B^\beta_0 + \mathbf{e}_\mu K \chi^{\mathrm{eeee}}_{\mu\alpha\beta\gamma} E^\alpha_\omega E^\beta_\omega E^{\gamma *}_\omega + \mathbf{e}_\mu K \chi^{\mathrm{eeeem}}_{\mu\alpha\beta\gamma\delta} E^\alpha_\omega E^\beta_\omega E^{\gamma *}_\omega B^\delta_0], \tag{15}$$

in which the complex conjugated field components are to be associated with negative angular frequencies, as following from the reality condition $\mathbf{E}_{-\omega} = \mathbf{E}^*_\omega$, and where $K = 3/4$ is the degeneracy factor as explicitly included in the Butcher and Cotter convention of nonlinear optical susceptibilities [10], as described in the previous section. For isotropic media, in particular then homogeneous layers of stacked gratings or elements of a discretized continuous-profile grating, the tensor elements involved in this expression are for arbitrary frequencies of the electric and magnetic fields listed in Table 1.

**Table 1.** Optical and magneto-optical susceptibilities of isotropic nonlinear magneto-optical media, for arbitrary angular frequency arguments of the tensors $\chi^{\mathrm{ee}}_{\alpha\beta}(-\omega_\sigma;\omega_\sigma)$, $\chi^{\mathrm{eem}}_{\alpha\beta\gamma}(-\omega_\sigma;\omega_1,\omega_2)$, $\chi^{\mathrm{eeee}}_{\alpha\beta\gamma\delta}(-\omega_\sigma;\omega_1,\omega_2,\omega_3)$, and $\chi^{\mathrm{eeeem}}_{\alpha\beta\gamma\delta\epsilon}(-\omega_\sigma;\omega_1,\omega_2,\omega_3,\omega_4)$. In the table, $M$ denotes the number of nonzero elements while $N$ denotes the number of nonzero and independent elements.

| Order | Source | $M$ | $N$ | Nonzero tensor elements |
|---|---|---|---|---|
| 1st | [1] | 3 | 1 | $\chi^{\mathrm{ee}}_{xx} = \chi^{\mathrm{ee}}_{yy} = \chi^{\mathrm{ee}}_{zz}$ |
| 2nd | [1] | 6 | 1 | $\chi^{\mathrm{eem}}_{xyz} = \chi^{\mathrm{eem}}_{zxy} = \chi^{\mathrm{eem}}_{yzx} = -\chi^{\mathrm{eem}}_{xzy} = -\chi^{\mathrm{eem}}_{zyx} = -\chi^{\mathrm{eem}}_{yxz}$ |
| 3rd | [2] | 21 | 3 | $\chi^{\mathrm{eeee}}_{xxxx} = \chi^{\mathrm{eeee}}_{yyyy} = \chi^{\mathrm{eeee}}_{zzzz} = \chi^{\mathrm{eeee}}_{xxyy} + \chi^{\mathrm{eeee}}_{xyxy} + \chi^{\mathrm{eeee}}_{xyyx}$,<br>$\chi^{\mathrm{eeee}}_{yyzz} = \chi^{\mathrm{eeee}}_{zzyy} = \chi^{\mathrm{eeee}}_{zzxx} = \chi^{\mathrm{eeee}}_{xxzz} = \chi^{\mathrm{eeee}}_{xxyy} = \chi^{\mathrm{eeee}}_{yyxx}$,<br>$\chi^{\mathrm{eeee}}_{yzyz} = \chi^{\mathrm{eeee}}_{zyzy} = \chi^{\mathrm{eeee}}_{zxzx} = \chi^{\mathrm{eeee}}_{xzxz} = \chi^{\mathrm{eeee}}_{xyxy} = \chi^{\mathrm{eeee}}_{yxyx}$,<br>$\chi^{\mathrm{eeee}}_{yzzy} = \chi^{\mathrm{eeee}}_{zyyz} = \chi^{\mathrm{eeee}}_{zxxz} = \chi^{\mathrm{eeee}}_{xzzx} = \chi^{\mathrm{eeee}}_{xyyx} = \chi^{\mathrm{eeee}}_{yxxy}$ |
| 4th | [3] | 60 | 6 | $\chi^{\mathrm{eeeem}}_{xxxyz} = \chi^{\mathrm{eeeem}}_{yyyzx} = \chi^{\mathrm{eeeem}}_{zzzxy} = -\chi^{\mathrm{eeeem}}_{xxxzy} = -\chi^{\mathrm{eeeem}}_{yyyxz} = -\chi^{\mathrm{eeeem}}_{zzzyx}$<br>$= \chi^{\mathrm{eeeem}}_{xxyzx} + \chi^{\mathrm{eeeem}}_{xyxzx} + \chi^{\mathrm{eeeem}}_{yxxzx}$<br>$\chi^{\mathrm{eeeem}}_{xxyxz} = \chi^{\mathrm{eeeem}}_{yyzyx} = \chi^{\mathrm{eeeem}}_{zzxzy} = -\chi^{\mathrm{eeeem}}_{xxzxy} = -\chi^{\mathrm{eeeem}}_{yyxyz} = -\chi^{\mathrm{eeeem}}_{zzyzx}$<br>$= -\chi^{\mathrm{eeeem}}_{xxyzx} + \chi^{\mathrm{eeeem}}_{xyzxx} + \chi^{\mathrm{eeeem}}_{yxzxx}$<br>$\chi^{\mathrm{eeeem}}_{xyxxz} = \chi^{\mathrm{eeeem}}_{yzyyx} = \chi^{\mathrm{eeeem}}_{zxzzy} = -\chi^{\mathrm{eeeem}}_{xzxxy} = -\chi^{\mathrm{eeeem}}_{yxyyz} = -\chi^{\mathrm{eeeem}}_{zyzzx}$<br>$= -\chi^{\mathrm{eeeem}}_{xyxzx} - \chi^{\mathrm{eeeem}}_{xyzxx} + \chi^{\mathrm{eeeem}}_{yzxxx}$<br>$\chi^{\mathrm{eeeem}}_{yxxxz} = \chi^{\mathrm{eeeem}}_{zyyyx} = \chi^{\mathrm{eeeem}}_{xzzzy} = -\chi^{\mathrm{eeeem}}_{zxxxy} = -\chi^{\mathrm{eeeem}}_{xyyyz} = -\chi^{\mathrm{eeeem}}_{yzzzx}$<br>$= -\chi^{\mathrm{eeeem}}_{yxxzx} - \chi^{\mathrm{eeeem}}_{yxzxx} - \chi^{\mathrm{eeeem}}_{yzxxx}$<br>$\chi^{\mathrm{eeeem}}_{xxyzx} = \chi^{\mathrm{eeeem}}_{yyzxy} = \chi^{\mathrm{eeeem}}_{zzxyz} = -\chi^{\mathrm{eeeem}}_{yyxzy} = -\chi^{\mathrm{eeeem}}_{zzyxz} = -\chi^{\mathrm{eeeem}}_{xxzyx}$<br>$\chi^{\mathrm{eeeem}}_{xyxzx} = \chi^{\mathrm{eeeem}}_{yzyxy} = \chi^{\mathrm{eeeem}}_{zxzyz} = -\chi^{\mathrm{eeeem}}_{xzxyx} = -\chi^{\mathrm{eeeem}}_{yxyzy} = -\chi^{\mathrm{eeeem}}_{zyzxz}$<br>$\chi^{\mathrm{eeeem}}_{yxxzx} = \chi^{\mathrm{eeeem}}_{zyyxy} = \chi^{\mathrm{eeeem}}_{xzzyz} = -\chi^{\mathrm{eeeem}}_{zxxyx} = -\chi^{\mathrm{eeeem}}_{xyyzy} = -\chi^{\mathrm{eeeem}}_{yzzxz}$<br>$\chi^{\mathrm{eeeem}}_{xyzxx} = \chi^{\mathrm{eeeem}}_{yzxyy} = \chi^{\mathrm{eeeem}}_{zxyzz} = -\chi^{\mathrm{eeeem}}_{xzyxx} = -\chi^{\mathrm{eeeem}}_{yxzyy} = -\chi^{\mathrm{eeeem}}_{zyxzz}$<br>$\chi^{\mathrm{eeeem}}_{yxzxx} = \chi^{\mathrm{eeeem}}_{zyxyy} = \chi^{\mathrm{eeeem}}_{xzyzz} = -\chi^{\mathrm{eeeem}}_{zxyxx} = -\chi^{\mathrm{eeeem}}_{xyzyy} = -\chi^{\mathrm{eeeem}}_{yzxzz}$<br>$\chi^{\mathrm{eeeem}}_{yzxxx} = \chi^{\mathrm{eeeem}}_{zxyyy} = \chi^{\mathrm{eeeem}}_{xyzzz} = -\chi^{\mathrm{eeeem}}_{zyxxx} = -\chi^{\mathrm{eeeem}}_{xzyyy} = -\chi^{\mathrm{eeeem}}_{yxzzz}$ |

**Table 2.** Optical and magneto-optical susceptibilities of isotropic nonlinear magneto-optical media, corresponding to those as listed in Table 1, but for angular frequency arguments corresponding to the particular choice of optical Kerr-effect and photo-induced Faraday rotation in presence of a static magnetic field, $\chi^{\text{ee}}_{\alpha\beta}(-\omega;\omega)$, $\chi^{\text{eem}}_{\alpha\beta\gamma}(-\omega;\omega,0)$, $\chi^{\text{eeee}}_{\alpha\beta\gamma\delta}(-\omega;\omega,\omega,-\omega)$, and $\chi^{\text{eeeem}}_{\alpha\beta\gamma\delta\epsilon}(-\omega;\omega,\omega,-\omega,0)$. As in Table 1, $M$ denotes the number of nonzero elements while $N$ denotes the number of nonzero and independent elements.

| Order | Source | $M$ | $N$ | Nonzero tensor elements |
|---|---|---|---|---|
| 1st | [1] | 3 | 1 | $\chi^{\text{ee}}_{xx} = \chi^{\text{ee}}_{yy} = \chi^{\text{ee}}_{zz}$ |
| 2nd | [1] | 6 | 1 | $\chi^{\text{eem}}_{xyz} = \chi^{\text{eem}}_{zxy} = \chi^{\text{eem}}_{yzx} = -\chi^{\text{eem}}_{xzy} = -\chi^{\text{eem}}_{zyx} = -\chi^{\text{eem}}_{yxz}$ |
| 3rd | [2] | 21 | 3 | $\chi^{\text{eeee}}_{xxxx} = \chi^{\text{eeee}}_{yyyy} = \chi^{\text{eeee}}_{zzzz} = \chi^{\text{eeee}}_{xxyy} + \chi^{\text{eeee}}_{xyxy} + \chi^{\text{eeee}}_{xyyx}$,<br>$\chi^{\text{eeee}}_{yyzz} = \chi^{\text{eeee}}_{zzyy} = \chi^{\text{eeee}}_{zzxx} = \chi^{\text{eeee}}_{xxzz} = \chi^{\text{eeee}}_{xxyy} = \chi^{\text{eeee}}_{yyxx}$,<br>$\chi^{\text{eeee}}_{yzyz} = \chi^{\text{eeee}}_{zyzy} = \chi^{\text{eeee}}_{zxzx} = \chi^{\text{eeee}}_{xzxz} = \chi^{\text{eeee}}_{xyxy} = \chi^{\text{eeee}}_{yxyx}$,<br>$\chi^{\text{eeee}}_{yzzy} = \chi^{\text{eeee}}_{zyyz} = \chi^{\text{eeee}}_{zxxz} = \chi^{\text{eeee}}_{xzzx} = \chi^{\text{eeee}}_{xyyx} = \chi^{\text{eeee}}_{yxxy}$ |
| 4th | [3] | 60 | 6 | $\chi^{\text{eeeem}}_{xxxyz} = \chi^{\text{eeeem}}_{yyyzx} = \chi^{\text{eeeem}}_{zzzxy} = -\chi^{\text{eeeem}}_{xxxzy} = -\chi^{\text{eeeem}}_{yyyxz} = -\chi^{\text{eeeem}}_{zzzyx}$<br>$= \chi^{\text{eeeem}}_{xxyzx} + \chi^{\text{eeeem}}_{xyxzx} + \chi^{\text{eeeem}}_{yxxzx}$<br>$\chi^{\text{eeeem}}_{xxyxz} = \chi^{\text{eeeem}}_{yyzyx} = \chi^{\text{eeeem}}_{zzxzy} = -\chi^{\text{eeeem}}_{xxzxy} = -\chi^{\text{eeeem}}_{yyxyz} = -\chi^{\text{eeeem}}_{zzyzx}$<br>$= -\chi^{\text{eeeem}}_{xxyzx} + \chi^{\text{eeeem}}_{xyzxx} + \chi^{\text{eeeem}}_{yxzxx}$<br>$\chi^{\text{eeeem}}_{xyxxz} = \chi^{\text{eeeem}}_{yzyyx} = \chi^{\text{eeeem}}_{zxzzy} = -\chi^{\text{eeeem}}_{xzxxy} = -\chi^{\text{eeeem}}_{yxyyz} = -\chi^{\text{eeeem}}_{zyzzx}$<br>$= -\chi^{\text{eeeem}}_{xyxzx} - \chi^{\text{eeeem}}_{xyzxx} + \chi^{\text{eeeem}}_{yzxxx}$<br>$\chi^{\text{eeeem}}_{yxxxz} = \chi^{\text{eeeem}}_{zyyyx} = \chi^{\text{eeeem}}_{xzzzy} = -\chi^{\text{eeeem}}_{zxxxy} = -\chi^{\text{eeeem}}_{xyyyz} = -\chi^{\text{eeeem}}_{yzzzx}$<br>$= -\chi^{\text{eeeem}}_{yxxzx} - \chi^{\text{eeeem}}_{yxzxx} - \chi^{\text{eeeem}}_{yzxxx}$<br>$\chi^{\text{eeeem}}_{xxyzx} = \chi^{\text{eeeem}}_{yyzxy} = \chi^{\text{eeeem}}_{zzxyz} = -\chi^{\text{eeeem}}_{yyxzy} = -\chi^{\text{eeeem}}_{zzyxz} = -\chi^{\text{eeeem}}_{xxzyx}$<br>$\chi^{\text{eeeem}}_{xyxzx} = \chi^{\text{eeeem}}_{yzyxy} = \chi^{\text{eeeem}}_{zxzyz} = -\chi^{\text{eeeem}}_{xzxyx} = -\chi^{\text{eeeem}}_{yxyzy} = -\chi^{\text{eeeem}}_{zyzxz}$<br>$\chi^{\text{eeeem}}_{yxxzx} = \chi^{\text{eeeem}}_{zyyxy} = \chi^{\text{eeeem}}_{xzzyz} = -\chi^{\text{eeeem}}_{zxxyx} = -\chi^{\text{eeeem}}_{xyyzy} = -\chi^{\text{eeeem}}_{yzzxz}$<br>$\chi^{\text{eeeem}}_{xyzxx} = \chi^{\text{eeeem}}_{yzxyy} = \chi^{\text{eeeem}}_{zxyzz} = -\chi^{\text{eeeem}}_{xzyxx} = -\chi^{\text{eeeem}}_{yxzyy} = -\chi^{\text{eeeem}}_{zyxzz}$<br>$\chi^{\text{eeeem}}_{yxzxx} = \chi^{\text{eeeem}}_{zyxyy} = \chi^{\text{eeeem}}_{xzyzz} = -\chi^{\text{eeeem}}_{zxyxx} = -\chi^{\text{eeeem}}_{xyzyy} = -\chi^{\text{eeeem}}_{yzxzz}$<br>$\chi^{\text{eeeem}}_{yzxxx} = \chi^{\text{eeeem}}_{zxyyy} = \chi^{\text{eeeem}}_{xyzzz} = -\chi^{\text{eeeem}}_{zyxxx} = -\chi^{\text{eeeem}}_{xzyyy} = -\chi^{\text{eeeem}}_{yxzzz}$ |

**16.** The vectorial form of the polarization density. By expanding the tensorial form of the polarization denisty as given by Eq. (15) and using the nonzero elements of the susceptibility tensors as given in Table 2, one after some straightforward algebra obtains the polarization density of the medium in a vectorial form as

$$\begin{aligned}\mathbf{P}_\omega =&\varepsilon_0[\chi^{\text{ee}}_{xx}\mathbf{E}_\omega + \chi^{\text{eem}}_{xyz}\mathbf{E}_\omega \times \mathbf{B}_0 + \tfrac{3}{4}(\chi^{\text{eeee}}_{xxxx} - \chi^{\text{eeee}}_{xyyx})(\mathbf{E}_\omega \cdot \mathbf{E}^*_\omega)\mathbf{E}_\omega + \tfrac{3}{4}\chi^{\text{eeee}}_{xyyx}(\mathbf{E}_\omega \cdot \mathbf{E}_\omega)\mathbf{E}^*_\omega \\ &+ \tfrac{3}{4}\chi^{\text{eeeem}}_{xyyyz}(\mathbf{E}_\omega \cdot \mathbf{E}^*_\omega)\mathbf{E}_\omega \times \mathbf{B}_0 + \tfrac{3}{4}\chi^{\text{eeeem}}_{xxxyz}\mathbf{E}_\omega(\mathbf{E}_\omega \cdot (\mathbf{E}^*_\omega \times \mathbf{B}_0))],\end{aligned} \tag{16}$$

where the degeneracy factor $K = 3/4$ was explicitly stated with its numerical value.

**17.** The Polarization density in the Faraday configuration. In the Faraday configuration [1], the optical field propagates collinearly with an externally applied static magnetic field $\mathbf{B}_0$. Taking the direction of propagation as the $z$-axis in a Cartesian coordinate system $(\mathbf{e}_x, \mathbf{e}_y, \mathbf{e}_z)$ and furthermore assuming the infinite plane wave approximation to hold for the transverse profile of the waves, the probem of wave propagation becomes a one-dimensional nonlinear system.

As a convention for circular polarization states, we employ the circularly polarized basis vectors

$$\mathbf{e}_\pm = (\mathbf{e}_x \pm i\mathbf{e}_y)/\sqrt{2},$$

which possesses the properties

$$\mathbf{e}_\pm \times \mathbf{e}_z = \pm i\mathbf{e}_\pm, \qquad \mathbf{e}^*_\pm \cdot \mathbf{e}_\pm = 1, \qquad \mathbf{e}^*_\pm \cdot \mathbf{e}_\mp = 0.$$

In the circularly polarized basis, the fields are hence taken as $\mathbf{B}_0 = \mathbf{e}_z B^z_0$ and $\mathbf{E}_\omega = \mathbf{e}_+ E^+_\omega + \mathbf{e}_- E^-_\omega$, in which the electric field is the total one, including any forward or backward traveling components. As this is inserted into Eq. (2), the electric polarization density of the medium hence becomes

$$\begin{aligned}\mathbf{e}^*_\pm \cdot \mathbf{P}_\omega = \varepsilon_0\{(\chi^{\text{ee}}_{xx} \pm i\chi^{\text{eem}}_{xyz} B^z_0)E^\pm_\omega + \tfrac{3}{4}[(\chi^{\text{eeee}}_{xxxx} - \chi^{\text{eeee}}_{xyyx})|E^\pm_\omega|^2 + (\chi^{\text{eeee}}_{xxxx} + \chi^{\text{eeee}}_{xyyx})|E^\mp_\omega|^2]E^\pm_\omega \\ \pm \tfrac{3}{4}[i(\chi^{\text{eeeem}}_{xyyyz} - \chi^{\text{eeeem}}_{xxxyz})B^z_0|E^\pm_\omega|^2 + i(\chi^{\text{eeeem}}_{xyyyz} + \chi^{\text{eeeem}}_{xxxyz})B^z_0|E^\mp_\omega|^2]E^\pm_\omega\}\end{aligned}$$

From now on, the susceptibility tensor elements will for the sake of simplicity in notation be incorporated into the index of refraction $n = (1 + \chi^{\text{ee}}_{xx})^{1/2}$, the gyration coefficient $\gamma = \chi^{\text{eem}}_{xyz} B^z_0$, and the nonlinear optical and magneto-optical parameters $p_\pm$ and $q_\pm$, defined as

$$p_\pm = \frac{3}{8n_\pm}[(\chi^{\text{eeee}}_{xxxx} - \chi^{\text{eeee}}_{xyyx}) \pm i(\chi^{\text{eeeem}}_{xyyyz} - \chi^{\text{eeeem}}_{xxxyz})B^z_0], \tag{2a}$$

$$q_\pm = \frac{3}{8n_\pm}[(\chi^{\text{eeee}}_{xxxx} + \chi^{\text{eeee}}_{xyyx}) \pm i(\chi^{\text{eeeem}}_{xyyyz} + \chi^{\text{eeeem}}_{xxxyz})B^z_0], \tag{2b}$$

where $n_\pm = (n^2 \pm \gamma)^{1/2}$ are the effective refractive indices for the circularly polarized components of the light, to give the polarization density in Eq. (1) in the simpler form

$$\mathbf{e}^*_\pm \cdot \mathbf{P}_\omega = \varepsilon_0[n^2_\pm - 1 + 2n_\pm(p_\pm|E^\pm_\omega|^2 + q_\pm|E^\mp_\omega|^2)]E^\pm_\omega. \tag{5}$$

The reason for includingthe refractive index in the particular scaling of the nonlinear coefficients as apparing in $p_\pm$ and $q_\pm$ will become obvious as the analysis proceeds with the wave equation. As the polarization density given by Eq. (5) is inserted into the wave equation for the electric field inside the medium, one obtains the equation of motion

$$\frac{\partial^2 E^\pm_\omega}{\partial z^2} + \frac{\omega^2 n^2_\pm}{c^2}E^\pm_\omega + 2n_\pm\frac{\omega^2}{c^2}(p_\pm|E^\pm_\omega|^2 + q_\pm|E^\mp_\omega|^2)E^\pm_\omega = 0. \tag{5}$$

This equation determines the spatial evolution of the total electromagnetic field, which may be composed of forward as well as backward traveling components of arbitrary polarization state. The task that now lies ahead is the separation of these components so as to form a system which provides the basis for further analytical investigation.

**18.** Separation into forward and traveling components. It may from Eq. (5) be noticed that in the absence of the nonlinear source terms, the general solutions for the left and right circularly polarized components of a forward traveling wave become

$$E^{+}_{\omega} = E^{\mathrm{f}}_{+}\exp(in_{+}\omega z/c), \qquad E^{-}_{\omega} = E^{\mathrm{f}}_{-}\exp(in_{-}\omega z/c),$$

respectively, where $E^{\mathrm{f}}_{+}$ and $E^{\mathrm{f}}_{-}$ are constants determined by the initial conditions at some arbitrary point along the direction of propagation. Meanwhile, the solution for the left/right circularly polarized components of a backward traveling wave becomes

$$E^{+}_{\omega} = E^{\mathrm{b}}_{+}\exp(in_{-}\omega z/c), \qquad E^{-}_{\omega} = E^{\mathrm{b}}_{-}\exp(in_{+}\omega z/c),$$

in which we here emphasize the change of effective refractive indices as experienced compared to the forward traveling component. This change is due to the fact that a backward traveling wave will experience the applied static magnetic field as pointing in the opposite direction as compared to the forward traveling wave, and hence the birefringence experienced from the Faraday effect will be different in sign. For a linearly polarized wave, this is manifested in that the polarization state of the backward traveling wave will rotate in opposite direction around the axis pointing in the direction of propagation, as compared to the forward traveling wave. Hence, by employing a separation according to

$$\begin{aligned}\mathbf{E}_{\omega} = \mathbf{e}_{+}E^{\mathrm{f}}_{+}\exp(in_{+}\omega z/c) + \mathbf{e}_{-}E^{\mathrm{f}}_{-}\exp(in_{-}\omega z/c)\\ + \mathbf{e}^{*}_{+}E^{\mathrm{b}}_{+}\exp(-in_{-}\omega z/c) + \mathbf{e}^{*}_{-}E^{\mathrm{b}}_{-}\exp(-in_{+}\omega z/c),\end{aligned} \tag{6}$$

or equivalently in the scalar form

$$E^{\pm}_{\omega} = E^{\mathrm{f}}_{\pm}\exp(in_{\pm}\omega z/c) + E^{\mathrm{b}}_{\mp}\exp(-in_{\pm}\omega z/c), \tag{6}$$

the wave equation for the envelopes $E^{\mathrm{f}}_{\pm}$ and $E^{\mathrm{b}}_{\pm}$, which generally are dependent on the coordinate $z$, naturally relaxes towards the solution to the linear wave propagation problem, with $E^{\mathrm{f}}_{\pm}$ and $E^{\mathrm{b}}_{\pm}$ becoming constants whenever the nonlinear terms may be neglected. In addition, due to the separation of the linear phase evolution in this one may also expect the field envelopes to be slowly varying functions of the spatial coordinate $z$ under any reasonable nonlinear effects.

By inserting Eq. (6) into The polarization density given in Eq. (5), the wave equation (0) takes the form

$$\begin{aligned}&\Big\{\frac{\partial^2 E^{\mathrm{f}}_{\pm}}{\partial z^2} + 2ik_{\pm}\frac{\partial E^{\mathrm{f}}_{\pm}}{\partial z} + 2\frac{\omega^2 n_{\pm}}{c^2}[p_{\pm}(|E^{\mathrm{f}}_{\pm}|^2 + 2|E^{\mathrm{b}}_{\mp}|^2) + q_{\pm}(|E^{\mathrm{f}}_{\mp}|^2 + |E^{\mathrm{b}}_{\pm}|^2)]E^{\mathrm{f}}_{\pm}\Big\}\exp(ik_{\pm}z)\\
&+\Big\{\frac{\partial^2 E^{\mathrm{b}}_{\mp}}{\partial z^2} - 2ik_{\pm}\frac{\partial E^{\mathrm{b}}_{\mp}}{\partial z} + 2\frac{\omega^2 n_{\pm}}{c^2}[p_{\pm}(|E^{\mathrm{b}}_{\mp}|^2 + 2|E^{\mathrm{f}}_{\pm}|^2) + q_{\pm}(|E^{\mathrm{b}}_{\pm}|^2 + |E^{\mathrm{f}}_{\mp}|^2)]E^{\mathrm{b}}_{\mp}\Big\}\exp(-ik_{\pm}z)\\
&+2\frac{\omega^2 n_{\pm}}{c^2}p_{\pm}[E^{\mathrm{f}\,2}_{\pm}E^{\mathrm{b}*}_{\mp}\exp(3ik_{\pm}z) + E^{\mathrm{b}\,2}_{\mp}E^{\mathrm{f}*}_{\pm}\exp(-3ik_{\pm}z)]\\
&+2\frac{\omega^2 n_{\pm}}{c^2}q_{\pm}[E^{\mathrm{f}}_{\pm}E^{\mathrm{f}}_{\mp}E^{\mathrm{b}*}_{\pm}\exp(i(k_{\pm}+2k_{\mp})z) + E^{\mathrm{b}}_{\mp}E^{\mathrm{b}}_{\pm}E^{\mathrm{f}*}_{\mp}\exp(-i(k_{\pm}+2k_{\mp})z)]\\
&+2\frac{\omega^2 n_{\pm}}{c^2}q_{\pm}[E^{\mathrm{f}}_{\mp}E^{\mathrm{b}}_{\mp}E^{\mathrm{b}*}_{\pm}\exp(i(2k_{\mp}-k_{\pm})z) + E^{\mathrm{b}}_{\pm}E^{\mathrm{f}}_{\pm}E^{\mathrm{f}*}_{\mp}\exp(-i(2k_{\mp}-k_{\pm})z)] = 0,\end{aligned} \tag{7}$$

where the notation $k_{\pm} \equiv \omega n_{\pm}/c$ was introduced for the sake of algebraic simplicity. In this equation, it may me noticed that the linear terms of the polarization density have been eliminated, due to the particular choice of separation of variables according to Eq. (6). However, in this form the wave propagation problem is extremely complex in its analysis, and in order to proceed, two general approximations may be employed without loosing much of generality. These are the slowly varying envelope approximation and the method of projecting out spatially phase mismatched terms. The slowly varying envelope approximation simply assumes that the second-order spatial derivative of the field envelope is much smaller in magnitude than the first-order derivative multiplied by the wavevector, or in terms of the here employed variables,

$$\Big|\frac{\partial^2 E^{\rm f}_{\pm}}{\partial z^2}\Big| \ll 2k_{\pm}\Big|\frac{\partial E^{\rm f}_{\pm}}{\partial z}\Big|, \qquad \Big|\frac{\partial^2 E^{\rm b}_{\mp}}{\partial z^2}\Big| \ll 2k_{\pm}\Big|\frac{\partial E^{\rm b}_{\mp}}{\partial z}\Big|.$$

By applying this approximation to Eq. (7), one obtains the slightly simpler system of equations

$$\begin{aligned}
&\Big\{\frac{\partial E^{\rm f}_{\pm}}{\partial z} - i\frac{\omega}{c}[p_{\pm}(|E^{\rm f}_{\pm}|^2 + 2|E^{\rm b}_{\mp}|^2) + q_{\pm}(|E^{\rm f}_{\mp}|^2 + |E^{\rm b}_{\pm}|^2)]E^{\rm f}_{\pm}\Big\}\exp(ik_{\pm}z)\\
&\quad - \Big\{\frac{\partial E^{\rm b}_{\mp}}{\partial z} + i\frac{\omega}{c}[p_{\pm}(|E^{\rm b}_{\mp}|^2 + 2|E^{\rm f}_{\pm}|^2) + q_{\pm}(|E^{\rm b}_{\pm}|^2 + |E^{\rm f}_{\mp}|^2)]E^{\rm b}_{\mp}\Big\}\exp(-ik_{\pm}z)\\
&\quad - i\frac{\omega}{c}p_{\pm}[E^{\rm f\,2}_{\pm}E^{\rm b*}_{\mp}\exp(3ik_{\pm}z) + E^{\rm b\,2}_{\mp}E^{\rm f*}_{\pm}\exp(-3ik_{\pm}z)]\\
&\quad - i\frac{\omega}{c}q_{\pm}[E^{\rm f}_{\pm}E^{\rm f}_{\mp}E^{\rm b*}_{\pm}\exp(i(k_{\pm}+2k_{\mp})z) + E^{\rm b}_{\mp}E^{\rm b}_{\pm}E^{\rm f*}_{\mp}\exp(-i(k_{\pm}+2k_{\mp})z)]\\
&\quad - i\frac{\omega}{c}q_{\pm}[E^{\rm f}_{\mp}E^{\rm b}_{\mp}E^{\rm b*}_{\pm}\exp(i(2k_{\mp}-k_{\pm})z) + E^{\rm b}_{\pm}E^{\rm f}_{\pm}E^{\rm f*}_{\mp}\exp(-i(2k_{\mp}-k_{\pm})z)] = 0.
\end{aligned} \tag{8}$$

Next step is now to project out terms which are closely phase matched, and in particular then the terms related to the envelopes of the forward and backward traveling components. This is for the forward traveling parts done by multiplying Eq. (8) by $\exp(-ik_{\pm}z)$ and average the resulting equation over a few spatial periods, assuming slowly varying field envelopes. In performing this averaging, essentially two levels of approximation may be applied: Either we keep also terms which are closely phase matched, that is to say the term involving the exponent $\exp(i(2k_{\mp}-k_{\pm})z)$, or we may assume that also this terms is averaged out for a sufficiently strong Faraday effect. This is in many cases a fully adequate approximation, in particular since the main contribution to effects such as the ellipse rotation, optcal Kerr-effect and photo-induced Faraday effect anyway are dominated by the terms involving the absolute magnitude of the fields. In the rigorous theory as here developed, however, these terms are kept for the time being, so as to fully encounter for any effects introduced by these. This method of projecting out the forward components is analogously applied to the backward traveling ones, but for this case by instead multiplying the equation by $\exp(ik_{\pm}z)$ prior to the averaging. The resulting system of coupled equations for the envelopes of the forward and backward traveling components of the field envelopes yield

$$\frac{\partial E^{\rm f}_{\pm}}{\partial z} = i\frac{\omega}{c}\{[p_{\pm}(|E^{\rm f}_{\pm}|^2 + 2|E^{\rm b}_{\mp}|^2) + q_{\pm}(|E^{\rm f}_{\mp}|^2 + |E^{\rm b}_{\pm}|^2)]E^{\rm f}_{\pm} + q_{\pm}E^{\rm f}_{\mp}E^{\rm b}_{\mp}E^{\rm b*}_{\pm}\exp(\mp i\eta z)\} = 0, \tag{9a}$$

$$\frac{\partial E^{\rm b}_{\mp}}{\partial z} = -i\frac{\omega}{c}\{[p_{\pm}(|E^{\rm b}_{\mp}|^2 + 2|E^{\rm f}_{\pm}|^2) + q_{\pm}(|E^{\rm b}_{\pm}|^2 + |E^{\rm f}_{\mp}|^2)]E^{\rm b}_{\mp} + q_{\pm}E^{\rm b}_{\pm}E^{\rm f}_{\pm}E^{\rm f*}_{\mp}\exp(\pm i\eta z)\} = 0, \tag{9b}$$

where the notation $\eta \equiv 2(k_{+} - k_{-})$ was introduced.

**19.** Separation of amplitude and phase of the field envelopes. In order to proceed with Eqs. (7), it is convenient to separate the wave propagation into parts affecting the phase and amplitude of the forward and backward traveling field envelopes. In order to do so, the amplitude and phase of the field envelopes are taken according to

$$E^{\rm f,b}_{\pm}(z) = A^{\rm f,b}_{\pm}(z)\exp(i\psi^{\rm f,b}_{\pm}(z)),$$

where $A^{\rm f,b}_{\pm}(z) \equiv |E^{\rm f,b}_{\pm}(z)|$ are the amplitudes and $\psi^{\rm f,b}_{\pm}(z)$ the corresponding phases of $E^{\rm f,b}_{\pm}(z)$. This ansatz leads to Eqs. (7) assuming the form

$$\begin{aligned}\Big(\frac{\partial A^{\rm f}_{\pm}}{\partial z} + iA^{\rm f}_{\pm}\frac{\partial \psi^{\rm f}_{\pm}}{\partial z}\Big)\exp(i\psi^{\rm f}_{\pm}) = i\frac{\omega}{c}\{[p_{\pm}(A^{\rm f\,2}_{\pm} + 2A^{\rm b\,2}_{\mp}) + q_{\pm}(A^{\rm f\,2}_{\mp} + A^{\rm b\,2}_{\pm})]A^{\rm f}_{\pm}\exp(i\psi^{\rm f}_{\pm})\\ + q_{\pm}A^{\rm f}_{\mp}A^{\rm b}_{\mp}A^{\rm b}_{\pm}\exp(\mp i\eta z + i\psi^{\rm f}_{\mp} + i\psi^{\rm b}_{\mp} - i\psi^{\rm b}_{\pm})\}, \quad (11a)\\ \Big(\frac{\partial A^{\rm b}_{\mp}}{\partial z} + iA^{\rm b}_{\mp}\frac{\partial \psi^{\rm b}_{\mp}}{\partial z}\Big)\exp(i\psi^{\rm b}_{\mp}) = -i\frac{\omega}{c}\{[p_{\pm}(A^{\rm b\,2}_{\mp} + 2A^{\rm f\,2}_{\pm}) + q_{\pm}(A^{\rm b\,2}_{\pm} + A^{\rm f\,2}_{\mp})]A^{\rm b}_{\mp}\exp(i\psi^{\rm b}_{\mp})\\ + q_{\pm}A^{\rm b}_{\pm}A^{\rm f}_{\pm}A^{\rm f}_{\mp}\exp(\pm i\eta z + i\psi^{\rm b}_{\pm} + i\psi^{\rm f}_{\pm} - i\psi^{\rm f}_{\mp})\}, \quad (11b)\end{aligned}$$

or equivalently

$$\begin{aligned}\frac{\partial A^{\rm f}_{\pm}}{\partial z} + iA^{\rm f}_{\pm}\frac{\partial \psi^{\rm f}_{\pm}}{\partial z} &= i\frac{\omega}{c}\{[p_{\pm}(A^{\rm f\,2}_{\pm} + 2A^{\rm b\,2}_{\mp}) + q_{\pm}(A^{\rm f\,2}_{\mp} + A^{\rm b\,2}_{\pm})]A^{\rm f}_{\pm} + q_{\pm}A^{\rm f}_{\mp}A^{\rm b}_{\mp}A^{\rm b}_{\pm}\exp(\mp i\psi)\}, \quad (12a)\\ \frac{\partial A^{\rm b}_{\mp}}{\partial z} + iA^{\rm b}_{\mp}\frac{\partial \psi^{\rm b}_{\mp}}{\partial z} &= -i\frac{\omega}{c}\{[p_{\pm}(A^{\rm b\,2}_{\mp} + 2A^{\rm f\,2}_{\pm}) + q_{\pm}(A^{\rm b\,2}_{\pm} + A^{\rm f\,2}_{\mp})]A^{\rm b}_{\mp} + q_{\pm}A^{\rm b}_{\pm}A^{\rm f}_{\pm}A^{\rm f}_{\mp}\exp(\pm i\psi)\}, \quad (12b)\end{aligned}$$

where the phase differences between the fields were incorporated into the single variable $\psi = \psi(z)$, defined as

$$\psi(z) \equiv \eta z + \psi^{\rm f}_{+}(z) - \psi^{\rm f}_{-}(z) + \psi^{\rm b}_{+}(z) - \psi^{\rm b}_{-}(z). \quad (13)$$

By multiplying Eqs. (12) by respective amplitudes $A^{\rm f}_{\pm}$ and $A^{\rm b}_{\mp}$, extracting the real parts of the left and right hand sides of Eqs. (12), and assumin a nonresonant medium in which $p_{\pm}$ and $q_{\pm}$ are real-valued quantities, one obtains the amplitude equations

$$\begin{aligned}\frac{\partial A^{\rm f\,2}_{\pm}}{\partial z} &= \pm\, 2(\omega/c)q_{\pm}A^{\rm f}_{+}A^{\rm f}_{-}A^{\rm b}_{+}A^{\rm b}_{-}\sin(\psi), \quad (14a)\\ \frac{\partial A^{\rm b\,2}_{\mp}}{\partial z} &= \pm\, 2(\omega/c)q_{\pm}A^{\rm f}_{+}A^{\rm f}_{-}A^{\rm b}_{+}A^{\rm b}_{-}\sin(\psi), \quad (14b)\end{aligned}$$

while the analogous extraction of the imaginary parts instead provides the phase equations

$$\begin{aligned}A^{\rm f\,2}_{\pm}\frac{\partial \psi^{\rm f}_{\pm}}{\partial z} &= \frac{\omega}{c}[p_{\pm}(A^{\rm f\,2}_{\pm} + 2A^{\rm b\,2}_{\mp}) + q_{\pm}(A^{\rm f\,2}_{\mp} + A^{\rm b\,2}_{\pm})]A^{\rm f\,2}_{\pm} + \frac{\omega}{c}q_{\pm}A^{\rm f}_{+}A^{\rm f}_{-}A^{\rm b}_{+}A^{\rm b}_{-}\cos(\psi(z)), \quad (15a)\\ A^{\rm b\,2}_{\mp}\frac{\partial \psi^{\rm b}_{\mp}}{\partial z} &= -\,\frac{\omega}{c}[p_{\pm}(A^{\rm b\,2}_{\mp} + 2A^{\rm f\,2}_{\pm}) + q_{\pm}(A^{\rm b\,2}_{\pm} + A^{\rm f\,2}_{\mp})]A^{\rm b\,2}_{\mp} - \frac{\omega}{c}q_{\pm}A^{\rm f}_{+}A^{\rm f}_{-}A^{\rm b}_{+}A^{\rm b}_{-}\cos(\psi(z)). \quad (15b)\end{aligned}$$

The inclusion of the phases as differences into the single variable $\psi(z)$ is not, as one at a first glance might think, only just a matter of convenient and compact notation. In fact, by differentiating $\psi(z)$ with respect to $z$ and using the phase evolution according to Eqs. (10), it actually turns out that the individual phases of the components of the optical wave can be eliminated in favour of the single variable $\psi(z)$, hence providing an effective reduction of the dimensionality of the problem, as will be shown in the following sections.

**20.** Invariants of motion. As a short side track to the analysis, before proceeding with actually solving the derived equations of motion for the amplitudes and phases of the field components, we will now consider a few important points regarding conserved quantities. These are important in the final stage when we separate out one single differentail equation for one single field variable from the so far complex and coupled system. From the spatial evolution of the amplitudes as given in Eqs. (14), one finds that the magnitudes of the envelopes obey the invariants of motion

$$\frac{\partial}{\partial z}(A_{+}^{\mathrm{f}\,2} - A_{-}^{\mathrm{b}\,2}) = 0, \qquad \frac{\partial}{\partial z}(A_{-}^{\mathrm{f}\,2} - A_{+}^{\mathrm{b}\,2}) = 0. \tag{16}$$

Similarly, one also finds that the respective forward and backward traveling circularly polarized components obey the invariants of motion

$$\frac{\partial}{\partial z}(q_{-}A_{+}^{\mathrm{f}\,2} + q_{+}A_{-}^{\mathrm{f}\,2}) = 0, \qquad \frac{\partial}{\partial z}(q_{+}A_{+}^{\mathrm{b}\,2} + q_{-}A_{-}^{\mathrm{b}\,2}) = 0. \tag{17}$$

The invariants of motion given by Eqs. (16) and (17) can hence be summarized as

$$A_{+}^{\mathrm{f}\,2} - A_{-}^{\mathrm{b}\,2} = \text{ const. } \equiv C_{+}/q_{-}, \tag{18a}$$

$$A_{-}^{\mathrm{f}\,2} - A_{+}^{\mathrm{b}\,2} = \text{ const. } \equiv C_{-}/q_{+}, \tag{18b}$$

$$q_{-}A_{+}^{\mathrm{f}\,2} + q_{+}A_{-}^{\mathrm{f}\,2} = \text{ const. } \equiv I_{\mathrm{f}}, \tag{18c}$$

$$q_{+}A_{+}^{\mathrm{b}\,2} + q_{-}A_{-}^{\mathrm{b}\,2} = \text{ const. } \equiv I_{\mathrm{b}}. \tag{18d}$$

The reason for this particular choice of the form of the constants of motion, with $C_{+}$ and $C_{-}$ scaled to be in units of $q_{-}$ and $q_{+}$, respectively, is motivated later on by simplifying the notation when it comes to choosing a normalized form for the equations of motion in the final stage of their solving. As Eqs. (18) imply that the invariants of motion for the general, $z$-dependent envelopes can be formulated as the linear algebraic system

$$\begin{pmatrix} q_{-} & 0 & 0 & -q_{-} \\ 0 & q_{+} & -q_{+} & 0 \\ q_{-} & q_{+} & 0 & 0 \\ 0 & 0 & q_{+} & q_{-} \end{pmatrix} \begin{pmatrix} A_{+}^{\mathrm{f}\,2}(z) \\ A_{-}^{\mathrm{f}\,2}(z) \\ A_{+}^{\mathrm{b}\,2}(z) \\ A_{-}^{\mathrm{b}\,2}(z) \end{pmatrix} = \begin{pmatrix} C_{+} \\ C_{-} \\ I_{\mathrm{f}} \\ I_{\mathrm{b}} \end{pmatrix}, \tag{19}$$

one may be tempted to draw the conclusion that all envelopes are constant with respect to the spatial coordinate $z$. However, this is a wrong conclusion, as one easily can verify that the system (18) is underdetermined, with a zero determinant of the system matrix, as appearing in Eq. (19).

**21.** Elimination of absolute phase dependence. In order to reduce the algebraic complexity of Eqs. (14) and (15), which is necessary in order proceed with the analytical theory of their evolution, we will now eliminate the absolute phases of the fields in favour of the single variable $\psi = \psi(z)$, which describes the phase difference between the four field components $E^{\rm f}_{+}$, $E^{\rm f}_{-}$, $E^{\rm b}_{+}$ and $E^{\rm b}_{-}$. This will reduce the dimension of of the problem from the original eight variables present in Eqs. (14) and (15), down to a total of five coupled variables and equations. By differentiating the variable $\psi(z)$ with respect to $z$ and using Eqs. (10) one finds

$$\begin{aligned}
\frac{\partial\psi}{\partial z} &= \eta + \frac{\partial\psi^{\rm f}_{+}}{\partial z} - \frac{\partial\psi^{\rm f}_{-}}{\partial z} + \frac{\partial\psi^{\rm b}_{+}}{\partial z} - \frac{\partial\psi^{\rm b}_{-}}{\partial z}\\
&= \eta + \frac{\omega}{c}[p_{+}(A^{\rm f\,2}_{+} + 2A^{\rm b\,2}_{-}) + q_{+}(A^{\rm f\,2}_{-} + A^{\rm b\,2}_{+})] + \frac{\omega}{c}q_{+}\frac{A^{\rm f}_{-}A^{\rm b}_{-}A^{\rm b}_{+}}{A^{\rm f}_{+}}\cos(\psi)\\
&\qquad - \frac{\omega}{c}[p_{-}(A^{\rm f\,2}_{-} + 2A^{\rm b\,2}_{+}) + q_{-}(A^{\rm f\,2}_{+} + A^{\rm b\,2}_{-})] - \frac{\omega}{c}q_{-}\frac{A^{\rm f}_{+}A^{\rm b}_{+}A^{\rm b}_{-}}{A^{\rm f}_{-}}\cos(\psi)\\
&\qquad - \frac{\omega}{c}[p_{-}(A^{\rm b\,2}_{+} + 2A^{\rm f\,2}_{-}) + q_{-}(A^{\rm b\,2}_{-} + A^{\rm f\,2}_{+})] - \frac{\omega}{c}q_{-}\frac{A^{\rm b}_{-}A^{\rm f}_{-}A^{\rm f}_{+}}{A^{\rm b}_{+}}\cos(\psi)\\
&\qquad + \frac{\omega}{c}[p_{+}(A^{\rm b\,2}_{-} + 2A^{\rm f\,2}_{+}) + q_{+}(A^{\rm b\,2}_{+} + A^{\rm f\,2}_{-})] + \frac{\omega}{c}q_{+}\frac{A^{\rm b}_{+}A^{\rm f}_{+}A^{\rm f}_{-}}{A^{\rm b}_{-}}\cos(\psi)\\
&= \eta + \frac{\omega}{c}[3p_{+}(A^{\rm f\,2}_{+} + A^{\rm b\,2}_{-}) + 2q_{+}(A^{\rm f\,2}_{-} + A^{\rm b\,2}_{+})] - \frac{\omega}{c}[3p_{-}(A^{\rm f\,2}_{-} + A^{\rm b\,2}_{+}) + 2q_{-}(A^{\rm f\,2}_{+} + A^{\rm b\,2}_{-})]\\
&\qquad + \frac{\omega}{c}\Big(q_{+}\frac{A^{\rm f}_{-}A^{\rm b}_{-}A^{\rm b}_{+}}{A^{\rm f}_{+}} - q_{-}\frac{A^{\rm f}_{+}A^{\rm b}_{+}A^{\rm b}_{-}}{A^{\rm f}_{-}} - q_{-}\frac{A^{\rm b}_{-}A^{\rm f}_{-}A^{\rm f}_{+}}{A^{\rm b}_{+}} + q_{+}\frac{A^{\rm b}_{+}A^{\rm f}_{+}A^{\rm f}_{-}}{A^{\rm b}_{-}}\Big)\cos(\psi)\\
&= \Big\{\text{ Use Eqs. (14) in substituting for terms in the second line }\Big\}\\
&= \eta + \frac{\omega}{c}[(3p_{+} - 2q_{-})(A^{\rm f\,2}_{+} + A^{\rm b\,2}_{-}) - (3p_{-} - 2q_{+})(A^{\rm f\,2}_{-} + A^{\rm b\,2}_{+})]\\
&\qquad + \Big(\frac{1}{A^{\rm f}_{+}}\frac{\partial A^{\rm f}_{+}}{\partial z} + \frac{1}{A^{\rm f}_{-}}\frac{\partial A^{\rm f}_{-}}{\partial z} + \frac{1}{A^{\rm b}_{+}}\frac{\partial A^{\rm b}_{+}}{\partial z} + \frac{1}{A^{\rm b}_{-}}\frac{\partial A^{\rm b}_{-}}{\partial z}\Big)\frac{\cos(\psi)}{\sin(\psi)}\\
&= \left\{\text{ Use } \frac{1}{A^{\rm f\,b}_{\pm}}\frac{\partial A^{\rm f,b}_{\pm}}{\partial z} = \frac{\partial}{\partial z}\ln A^{\rm f,b}_{\pm} \text{ and } \cos(\psi)/\sin(\psi) \equiv \cot(\psi)\right\}\\
&= \eta + \frac{\omega}{c}[(3p_{+} - 2q_{-})(A^{\rm f\,2}_{+} + A^{\rm b\,2}_{-}) - (3p_{-} - 2q_{+})(A^{\rm f\,2}_{-} + A^{\rm b\,2}_{+})]\\
&\qquad + \Big(\frac{\partial}{\partial z}\ln A^{\rm f}_{+} + \frac{\partial}{\partial z}\ln A^{\rm f}_{-} + \frac{\partial}{\partial z}\ln A^{\rm b}_{+} + \frac{\partial}{\partial z}\ln A^{\rm b}_{-}\Big)\cot(\psi)\\
&= \eta + \frac{\omega}{c}[(3p_{+} - 2q_{-})(A^{\rm f\,2}_{+} + A^{\rm b\,2}_{-}) - (3p_{-} - 2q_{+})(A^{\rm f\,2}_{-} + A^{\rm b\,2}_{+})] + \cot(\psi)\frac{\partial}{\partial z}\ln(A^{\rm f}_{+}A^{\rm f}_{-}A^{\rm b}_{+}A^{\rm b}_{-}).
\end{aligned}$$

Thus, by returning to the amplitude evolution described by Eqs. (14) and by defining the short-hand notation

$$r_{\pm} \equiv (3p_{\pm} - 2q_{\mp})$$

for the coefficients of the nonlinear terms, the evolution of the optical field can be summarized with the considerably simplified system of coupled and nonlinear differential equations

$$\frac{\partial A^{\rm f\,2}_{\pm}}{\partial z} = \pm 2(\omega/c)q_{\pm}A^{\rm f}_{+}A^{\rm f}_{-}A^{\rm b}_{+}A^{\rm b}_{-}\sin(\psi), \tag{19a}$$

$$\frac{\partial A^{\rm b\,2}_{\mp}}{\partial z} = \pm 2(\omega/c)q_{\pm}A^{\rm f}_{+}A^{\rm f}_{-}A^{\rm b}_{+}A^{\rm b}_{-}\sin(\psi), \tag{19b}$$

$$\frac{\partial\psi}{\partial z} = \eta + (\omega/c)[r_{+}(A^{\rm f\,2}_{+} + A^{\rm b\,2}_{-}) - r_{-}(A^{\rm f\,2}_{-} + A^{\rm b\,2}_{+})] + \cot(\psi)\frac{\partial}{\partial z}\ln(A^{\rm f}_{+}A^{\rm f}_{-}A^{\rm b}_{+}A^{\rm b}_{-}). \tag{19c}$$

Notice that the absolute phases of the field components now have been entirely eliminated in favour of $\psi = \psi(z)$, as was the goal outset in the beginning of this section. The next step in the analysis is to eliminate also the relative phase from the equations of motion, so as to provide an autonomous system only involving the field amplitudes $A^{\rm f}_{\pm}$ and $A^{\rm b}_{\pm}$.

**22.** Elimination of the relative phase. The equations of motion given by Eqs. (19) are considerably reduced in their algebraic complexity as compared to the original ones, as given by Eqs. (1). However, there are still some simplifications which can be applied to further reduce the complexity, in particular then the elimination of the phase altogether, as will now be shown.

The trick to apply is to first multiply the left and right hand sides of Eq. (19c) with $A^{\rm f}_+A^{\rm f}_-A^{\rm b}_+A^{\rm b}_- \sin(\psi)$, expanding the spatial derivative on the right hand side, and rearrange the terms to obtain

$$\begin{aligned}\cos(\psi)\frac{\partial}{\partial z}(A^{\rm f}_+A^{\rm f}_-A^{\rm b}_+A^{\rm b}_-) - A^{\rm f}_+A^{\rm f}_-A^{\rm b}_+A^{\rm b}_- \sin(\psi)\frac{\partial\psi}{\partial z}&\\ = -\{\eta + (\omega/c)[r_+(A^{{\rm f}\,2}_+ + A^{{\rm b}\,2}_-) - r_-(A^{{\rm f}\,2}_- + A^{{\rm b}\,2}_+)]\}A^{\rm f}_+A^{\rm f}_-A^{\rm b}_+A^{\rm b}_- \sin(\psi).&\end{aligned} \tag{20}$$

In this equation, we immediately find that the left hand side is the spatial derivative of $A^{\rm f}_+A^{\rm f}_-A^{\rm b}_+A^{\rm b}_- \cos(\psi)$, so we may start looking for rewriting the right hand side as a spatial derivative as well, in which case we would end up with an integrable equation. In this search for a form of the right hand side which could be integrated, the appearance of the factor $A^{\rm f}_+A^{\rm f}_-A^{\rm b}_+A^{\rm b}_- \sin(\psi)$ leads to using either of Eqs. (19a) or (19b) to express this term as a derivative of the amplitudes instead, hopefully leading to the right hand side as a polynomial form which is easily integrated. Thus, Eq. (20) can be rewritten as

$$\begin{aligned}&\frac{\partial}{\partial z}(A^{\rm f}_+A^{\rm f}_-A^{\rm b}_+A^{\rm b}_- \cos(\psi))\\
&\quad= -\{\eta + (\omega/c)[r_+(A^{{\rm f}\,2}_+ + A^{{\rm b}\,2}_-) - r_-(A^{{\rm f}\,2}_- + A^{{\rm b}\,2}_+)]\}A^{\rm f}_+A^{\rm f}_-A^{\rm b}_+A^{\rm b}_- \sin(\psi).\\
&\quad= \big\{\text{ Use Eq. (19a) in substituting for the factor } A^{\rm f}_+A^{\rm f}_-A^{\rm b}_+A^{\rm b}_- \sin(\psi)\ \big\}\\
&\quad= -\{\eta + (\omega/c)[r_+(A^{{\rm f}\,2}_+ + A^{{\rm b}\,2}_-) - r_-(A^{{\rm f}\,2}_- + A^{{\rm b}\,2}_+)]\}\frac{1}{2(\omega/c)q_+}\frac{\partial A^{{\rm f}\,2}_+}{\partial z}\\
&\quad= \big\{\text{ Eliminate backward traveling components using Eqs. (18a) and (18b) }\big\}\\
&\quad= -\frac{1}{q_+}\Big\{\frac{\eta}{2(\omega/c)} + \tfrac{1}{2}[r_+(2A^{{\rm f}\,2}_+ - C_+/q_-) - r_-(2A^{{\rm f}\,2}_- - C_-/q_+)]\Big\}\frac{\partial A^{{\rm f}\,2}_+}{\partial z}\\
&\quad= \big\{\text{ Eliminate } A^{{\rm f}\,2}_- \text{ using Eq. (18c) }\big\}\\
&\quad= -\frac{1}{q_+}\Big\{\frac{\eta}{2(\omega/c)} + \tfrac{1}{2}[r_+(2q_-A^{{\rm f}\,2}_+ - C_+)/q_- - r_-(2(I_{\rm f} - q_-A^{{\rm f}\,2}_+) - C_-)/q_+]\Big\}\frac{\partial A^{{\rm f}\,2}_+}{\partial z}\\
&\quad= \big\{\text{ Collect terms in powers of } A^{{\rm f}\,2}_+\ \big\}\\
&\quad= -\frac{1}{q_+}\Big\{\Big(\frac{\eta}{2(\omega/c)} - \frac{(q_+r_+C_+ - q_-r_-C_-)}{2q_+q_-} - \frac{r_-I_{\rm f}}{q_+}\Big) + \frac{(q_+r_+ + q_-r_-)}{q_+q_-}q_-A^{{\rm f}\,2}_+\Big\}\frac{\partial A^{{\rm f}\,2}_+}{\partial z}\\
&\quad= \big\{\text{ Identify as differential of polynomial of form } p(A^{{\rm f}\,2}_+)\ \big\}\\
&\quad= -\frac{1}{q_+}\frac{\partial}{\partial z}\Big\{\Big(\frac{\eta}{2(\omega/c)} - \frac{(q_+r_+C_+ - q_-r_-C_-)}{2q_+q_-} - \frac{r_-I_{\rm f}}{q_+}\Big)A^{{\rm f}\,2}_+ + \frac{(q_+r_+ + q_-r_-)}{2q_+q_-}q_-A^{{\rm f}\,4}_+\Big\}\end{aligned} \tag{21}$$

which directly integrates to yield

$$\begin{aligned}q_+A^{\rm f}_+A^{\rm f}_-A^{\rm b}_+A^{\rm b}_- \cos(\psi) &= \Gamma/q_- - \Big[\Big(\frac{\eta}{2(\omega/c)} - \frac{(q_+r_+C_+ - q_-r_-C_-)}{2q_+q_-} - \frac{r_-I_{\rm f}}{q_+}\Big)A^{{\rm f}\,2}_+ + \frac{(q_+r_+ + q_-r_-)}{2q_+q_-}q_-A^{{\rm f}\,4}_+\Big]\\ &= \Gamma/q_- - aA^{{\rm f}\,2}_+ - bq_-A^{{\rm f}\,4}_+,\end{aligned} \tag{22}$$

where $\Gamma$ is a yet undetermined constant of integration, to be determined later on by the boundary conditions of the homogeneous domain, and where the short-hand notations

$$a \equiv \frac{\eta}{2(\omega/c)} - \frac{(q_+r_+C_+ - q_-r_-C_-)}{2q_+q_-} - \frac{r_-I_{\rm f}}{q_+}, \qquad b \equiv \frac{(q_+r_+ + q_-r_-)}{2q_+q_-}, \tag{24}$$

were introduced. The relative phase $\psi$ is now from Eq. (22) determined in terms of the field amplitudes, and in order to get an expression for the factor $\sin(\psi)$ which appears in the amplitude equations, for example in Eq. (19a), we may employ the trigonometric identity $\sin^2(\psi)+\cos^2(\psi)=1$, from which we obtain Eq. (19a) as

$$\frac{\partial A_+^{\mathrm{f}\,2}}{\partial z} = (-1)^k 2(\omega/c)[q_+^2 A_+^{\mathrm{f}\,2} A_-^{\mathrm{f}\,2} A_+^{\mathrm{b}\,2} A_-^{\mathrm{b}\,2} - (\Gamma/q_- - aA_+^{\mathrm{f}\,2} - bq_- A_+^{\mathrm{f}\,4})^2]^{1/2}, \tag{23}$$

where the undeterminacy of the sign of $\sin(\psi) = \pm(1-\cos^2(\psi))^{1/2}$ is included in the factor $(-1)^k$, with $k$ being a yet undetermined integer. In this nonlinear differential equation, also the relative phase $\psi$ is eliminated, and the mathematical dimension of the problem at hand has been reduced from in total eight variables to the present four ones given by the linearly independent field amplitudes. It may be observed that in deriving Eq. (23), the invariants of motion given by Eqs. (18) had to be employed, in order to be able to write the right-hand side as a total derivative. As the invariants of motion equally well still can be employed to further reduce the dimensionality of the problem by eliminating $A_-^{\mathrm{f}\,2}$, $A_+^{\mathrm{b}\,2}$ and $A_-^{\mathrm{b}\,2}$, we can at this stage easily see the outline to finally formulate the problem of wave propagation as one single differential equation involving only the single variable $A_+^{\mathrm{f}\,2}$.

**23.** Elimination of redundant field amplitudes. In this section, the dimensionality of the wave propagation problem is finally reduced to yield a one-dimensional problem in one single variable $A_+^{\mathrm{f}\,2}$. By eliminating the field amplitudes $A_-^{\mathrm{f}\,2}$, $A_+^{\mathrm{b}\,2}$ and $A_-^{\mathrm{b}\,2}$ with the use of the invariants of motion as given by Eqs. (18), one obtains Eq. (23) as

$$\begin{aligned}\frac{\partial A_+^{\mathrm{f}\,2}}{\partial z} &= (-1)^k 2(\omega/c)[q_+^2 A_+^{\mathrm{f}\,2}\underbrace{q_+^{-1}(I_{\mathrm{f}} - q_- A_+^{\mathrm{f}\,2})}_{=A_-^{\mathrm{f}\,2}(z)}\underbrace{q_+^{-1}(I_{\mathrm{f}} - q_- A_+^{\mathrm{f}\,2} - C_-)}_{=A_+^{\mathrm{b}\,2}(z)}\underbrace{q_-^{-1}(q_- A_+^{\mathrm{f}\,2} - C_+)}_{=A_-^{\mathrm{b}\,2}(z)}\\ &\qquad - q_-^{-2}(\Gamma - aq_- A_+^{\mathrm{f}\,2} - bq_-^2 A_+^{\mathrm{f}\,4})^2]^{1/2}\\ &= (-1)^k 2\frac{(\omega/c)}{q_-}[q_- A_+^{\mathrm{f}\,2}(I_{\mathrm{f}} - q_- A_+^{\mathrm{f}\,2})(I_{\mathrm{f}} - q_- A_+^{\mathrm{f}\,2} - C_-)(q_- A_+^{\mathrm{f}\,2} - C_+) - (\Gamma - aq_- A_+^{\mathrm{f}\,2} - bq_-^2 A_+^{\mathrm{f}\,4})^2]^{1/2}.\end{aligned} \tag{24}$$

Hence, by taking the normalized and dimensionless field amplitude variable $v$ and the normalized and dimensionless coordinate $\zeta$ according to

$$v \equiv q_- A_+^{\mathrm{f}\,2}, \qquad \zeta \equiv 2\omega z/c, \tag{25}$$

we obtain the normalized equation for the amplitude of the left circularly polarized forward traveling field component as

$$\frac{\partial v}{\partial \zeta} = (-1)^k [v(I_{\mathrm{f}} - v)(I_{\mathrm{f}} - v - C_-)(v - C_+) - (\Gamma - av - bv^2)^2]^{1/2}. \tag{26}$$

This nonlinear ordinary differential equation is now well suited for numerical evaluation, or even analytical as will be shown next. It should however be noticed that the field amplitudes $A_+^{\mathrm{f}}$, $A_-^{\mathrm{f}}$, $A_+^{\mathrm{b}}$, and $A_-^{\mathrm{b}}$ are all involved implicitly through the invariants of motion $I_{\mathrm{f}}$, $C_+$, and $C_-$, as given by Eqs. (18), and also via the introduced short-hand notations $a$ and $b$ as introduced in Eq. (24); also the relative phase $\psi$ of the fields evaluated at some coordinate $\zeta_0$ enters as one parameter to encounter for, via the integration constant $\Gamma$ and Eq. (22).

**24.** Formulation in terms of an elliptic integral. The nonlinear ordinary differential equation (26) for $v = v(\zeta)$ can be formulated as

$$\frac{\partial v}{\partial \zeta} = (-1)^k [f(v)]^{1/2}, \tag{27}$$

where

$$f(v) \equiv a_4 v^4 + 4a_3 v^3 + 6a_2 v^2 + 4a_1 v + a_0$$

is a quartic polynomial in the normalized field amplitudes $v$, and in which the coefficients $a_j$, for $j = 1, \ldots, 4$, are explicitly given in terms of the previously used algebraic symbols as

$$a_0 = -\Gamma^2, \tag{28a}$$

$$a_1 = (2\Gamma a - I_{\mathrm{f}}^2 C_+ + I_{\mathrm{f}} C_+ C_-)/4, \tag{28b}$$

$$a_2 = (I_{\mathrm{f}}^2 + 2\Gamma b + 2I_{\mathrm{f}} C_+ - I_{\mathrm{f}} C_- - a^2 - C_+ C_-)/6, \tag{28c}$$

$$a_3 = (-2I_{\mathrm{f}} + C_- - C_- - 2ab)/4, \tag{28d}$$

$$a_4 = 1 - b^2. \tag{28e}$$

The solution $v(\zeta)$ is then from Eq. (27) given by the integral equation

$$\int_{v(\zeta_0)}^{v(\zeta)} \frac{dx}{(a_4 x^4 + 4a_3 x^3 + 6a_2 x^2 + 4a_1 x + a_0)^{1/2}} = (-1)^k \int_{\zeta_0}^{\zeta} d\zeta = (-1)^k (\zeta - \zeta_0), \tag{29}$$

where $v_0 \equiv v(\zeta_0)$ with $\zeta_0$ being an arbitrarily chosen reference coordinate $\zeta_0$ along the axis of wave propagation.

The left-hand side of Eq. (29) constitutes an elliptic integral which principally can be reduced to the Legendre-Jacobi normal form by means of suitable homographic substitutions and the use of Jacobian elliptic functions. This implies the solving for the roots of the characteristic polynomial equation $f(v) = 0$ which, however perfectly well analytically solvable, though is an algebraically very cumbersome task.

A more convenient method is to instead employ notation and method of solution by Weierstrass, which turns out to considerably simplify the algebra, and which provides an analytic solution which can be computed analytically in terms of the Weierstrass elliptic function [11–13] $\wp(\zeta) = \wp(\zeta; g_2, g_3)$ with the quartic invariants $g_2$ and $g_3$ as

$$g_2 \equiv a_0 a_4 - 4a_1 a_3 + 3a_2^2, \qquad g_3 \equiv \begin{vmatrix} a_0 & a_1 & a_2 \\ a_1 & a_2 & a_3 \\ a_2 & a_3 & a_4 \end{vmatrix} = a_0 a_2 a_4 + 2a_1 a_2 a_3 - a_2^3 - a_0 a_3^2 - a_1^2 a_4.$$

The solution $v(\zeta)$ of Eq. (29) is then explicitly given as

$$v(\zeta) = v_0 + \frac{\sqrt{f(v_0)}\wp'(\zeta) + \frac{1}{2}f'(v_0)[\wp(\zeta) - \frac{1}{24}f''(v_0)] + \frac{1}{24}f(v_0)f'''(v_0)}{2[\wp(\zeta) - \frac{1}{24}f''(v_0)]^2 - \frac{1}{48}f(v_0)f^{(\mathrm{iv})}(v_0)}. \tag{30}$$

That the solution given by Eq. (30) actually *is* a solution to the differential equation (27) can easily be verified using the MapleV code

```
restart:
f:=a[0]*v^4+4*a[1]*v^3+6*a[2]*v^2+4*a[3]*v+a[4];
g[2]:=a[0]*a[4]-4*a[1]*a[3]+3*a[2]^2;
g[3]:=a[0]*a[2]*a[4]+2*a[1]*a[2]*a[3]-a[2]^3-a[0]*a[3]^2-a[1]^2*a[4];
df[0]:=eval(f,v=v0):
df[1]:=eval(diff(f,v$1),v=v0):
df[2]:=eval(diff(f,v$2),v=v0):
df[3]:=eval(diff(f,v$3),v=v0):
```

```
df[4]:=eval(diff(f,v$4),v=v0):
tmp[1]:=sqrt(df[0])*WeierstrassPPrime(z,g[2],g[3]):
tmp[2]:=(1/2)*df[1]*(WeierstrassP(z,g[2],g[3])-(1/24)*df[2]):
tmp[3]:=(1/24)*df[0]*df[3]:
tmp[4]:=2*(WeierstrassP(z,g[2],g[3])-(1/24)*df[2])^2:
tmp[5]:=(1/48)*df[0]*df[4]:
v:=v0+(tmp[1]+tmp[2]+tmp[3])/(tmp[4]-tmp[5]):
p:=a[0]*v^4+4*a[1]*v^3+6*a[2]*v^2+4*a[3]*v+a[4]:
testfunc:=(diff(v,z))^2-p:
testfunc:=simplify(testfunc);
```

**25.** Boundary conditions.

**26. Revision history of the program.**

**2002-10-28** [v.1.0] `<fredrik.jonsson@proximion.com>`
First properly working version of the MAGBRAGG program, written in plain (ANSI-conformant) C.

**2002-11-15** [v.1.1] `<fredrik.jonsson@proximion.com>`
Minor changes to the blocks of the program for the generation of reflection spectra, providing data for a talk at the MRS 2002 Fall Meeting [F. Jonsson and C. Flytzanis, *Theoretical model for magneto-optical Bragg gratings*, Talk O4.7 presented at the 2002 Materials Research Society (MRS) Fall Meeting, Boston, United States (December 2–6, 2002)].

**2003-02-18** [v.1.2] `<fredrik.jonsson@proximion.com>`
Modified the part of the program that writes the spatial optical field distribution along the grating to file, so that both forward and backward left and right circularly polarized components of the propagating fields are written to file (using the `--fieldevolution` command line option).

**2003-02-25** [v.1.3] `<fredrik.jonsson@proximion.com>`
Transferred all source code of the MAGBRAGG program from C to CWEB. For information on the CWEB programming language, see `http://www.literateprogramming.com`.

**2003-04-18** [v.1.4] `<fredrik.jonsson@proximion.com>`
Added the `--modifylayer` option, enabling the user to manually modify an arbitrary layer of the grating structure in linear as well as nonlinear optical parameters, or modifying the layer thickness.

**2003-04-28** [v.1.5] `<fredrik.jonsson@proximion.com>`
Added the `--intensityevolution` option.

**2003-07-15** [v.1.6] `<hakkasberra@hotmail.com>`
Added some points in the documentation regarding the philosophy behind creating two-dimensional plots from topological graphs of Stokes-parameter hypersurfaces.

**2003-07-22** [v.1.7] `<hakkasberra@hotmail.com>`
Added the possibility of specifiying whether the electrical field displacement or Stokes parameters should be written to file, when saving the intra-grating field distribution via the `--fieldevolution` command line option.

**2003-07-23** [v.1.8] `<hakkasberra@hotmail.com>`
Changed the reference phase of the calculated, final intra-grating optical field, so that the main axis of the polarization ellipse always is directed along the $x$-axis (corresponding to $S_2 = 0$ in a Stokes parameter description) at the beginning of the grating, at $z = 0$. (The naturally appearing reference phase is relative the output field, at the end of the grating, since $z = L$ is the spatial starting point in the inverse algorithm.)

**2003-08-04** [v.1.9] `<hakkasberra@hotmail.com>`
Added the `--normalize_length_to_um` option, which causes the program to write spatial distances in micrometers instead of meters. Useful for pre-normalizing data prior to importing it into programs for making graphs of the spatial intra-grating distribution of intensity or polarization state. Also added the `--normalize_intensity` option, which causes the program to write the intensity-related Stoke parameter $S_0(z)$ as the quote with the input intensity instead, as $S_0(z)/S_0(0)$.

**2003-08-07** [v.1.10] `<hakkasberra@hotmail.com>`
Added the `--intensityspectrumfile` option, to save a regular intensity spectrum as function of wavelength, and not only just the complex reflection and transmission spectra. Using this option, the character string following it will be used as the base filename for the generated intensity reflection spectrum, which will be named

⟨basename⟩`.irsp.dat`, and the intensity transmission spectrum, which will be named ⟨basename⟩`.itsp.dat`. The format of these files are simply that the first column is the vacuum wavelength in nanometers, and the second column the reflection or transmission coefficients. In addition to these two files, an additional file ⟨basename⟩`.chec.dat` will be generated, containing the sum of the respective reflection and transmission coefficients. Ideally, the second column of this file should be identically one; any deviation from this is an artefact of the limited numerical precision in the simulation, and can be taken as a measure on the correctness of the obtained data.

**2003-08-19** [v.1.11] `<hakkasberra@hotmail.com>`
Added the `--normalize_ellipticity` option, which switch the program to writing the normalized ellipticity $S_3/S_0$ (with a numerical value between $-1$ and 1) instead of the third Stokes parameter, whenever it is to be written to disk. Also added the `--scale_stokesparams` $\langle a\rangle$ option, which at the stage of saving the Stokes parameters to disk divides the sets $(S_0, S_1, S_2, S_3)$, $(W_0, W_1, W_2, W_3)$, and $(V_0, V_1, V_2, V_3)$ by the scalefactor $\langle a\rangle$ prior to writing them to disk. This is typically a good thing to do if the program that is to post-analyze the generated data has a poor way of handling large numbers (typically larger than $\sim 10^{14}$ for the squared amplitudes of the components of the electric field).

**2003-08-20** [v.1.12] `<hakkasberra@hotmail.com>`
Modified the `Makefile` to provide a somewhat more intelligent check on which files that need to be updated on compilation. Also updated section five, *Compiling the source code*, of the documentation of the program.

**2003-08-23** [v.1.13] `<hakkasberra@hotmail.com>`
Added a check in the blocks that saves the full grating structure to file, so that the program now fully recognizes the `--normalize_length_to_um` option.

**2003-10-06** [v.1.14] `<jonsson@uni-wuppertal.de>`
Added a few clarifying paragraphs on the conventions used for the magneto-optical material parameters and the exact conventions for the circularly polarized modes as here used.

**2003-12-10** [v.1.15] `<jonsson@uni-wuppertal.de>`
Added a few clarifying paragraphs on the scaling of the Stokes parameters that are written to file after the finished calculations. As a default, and as a matter of convention in electrodynamics in SI units, all Stokes parameters are given in $\mathrm{V}^2/\mathrm{m}^2$, through their definition. For example, the incident field (which is calculated by the program in this inverse formulation of the problem) is expressed in terms of the Stokes parameters as

$$\begin{aligned} S_0 &= |E^{\mathrm{f}}_{0+}|^2 + |E^{\mathrm{f}}_{0-}|^2, \qquad & S_1 &= 2\,\mathrm{Re}[E^{\mathrm{f}*}_{0+}E^{\mathrm{f}}_{0-}],\\ S_3 &= |E^{\mathrm{f}}_{0+}|^2 - |E^{\mathrm{f}}_{0-}|^2, \qquad & S_2 &= 2\,\mathrm{Im}[E^{\mathrm{f}*}_{0+}E^{\mathrm{f}}_{0-}]. \end{aligned}$$

However, the direct interpretation of these quantities in terms of squared Volts per square metres is sometimes somewhat inconvenient; therefore, those parameters can be scaled to give an interpretation of the intensity (in regular SI units measured in Watts per square metres), as $S'_k = (\varepsilon_0 c/2)S_k$, or explicitly

$$\begin{aligned} S'_0 &= (\varepsilon_0 c/2)[|E^{\mathrm{f}}_{0+}|^2 + |E^{\mathrm{f}}_{0-}|^2], \qquad & S'_1 &= (\varepsilon_0 c/2)2\,\mathrm{Re}[E^{\mathrm{f}*}_{0+}E^{\mathrm{f}}_{0-}],\\ S'_3 &= (\varepsilon_0 c/2)[|E^{\mathrm{f}}_{0+}|^2 - |E^{\mathrm{f}}_{0-}|^2], \qquad & S'_2 &= (\varepsilon_0 c/2)2\,\mathrm{Im}[E^{\mathrm{f}*}_{0+}E^{\mathrm{f}}_{0-}]. \end{aligned}$$

In this representation, $S'_0$ is now identical to the incident intensity $I_{\mathrm{in}}$ $[\mathrm{W}/\mathrm{m}^2]$. In order to have those scaled Stokes parameters $S'_k$ written to file, rather than the default ones, one convenient possibility is to use the previously added `--scale_stokesparams` option, to include `--scale_stokesparams 1.327209e-3` at the command line when invoking the program. This numerical value of the scaling is obtained from

$$\begin{aligned}\varepsilon_0 c/2 &= (8.854187817\ldots \times 10^{-12}\ \mathrm{F/m}) \times (2.99792458 \times 10^{8}\ \mathrm{m/s})/2\\ &\approx 1.327209 \times 10^{-3}\ \mathrm{F/s}.\end{aligned}$$

In regular SI units as here used, the physical dimension of the quantity $\varepsilon_0 c/2$ is $[(\mathrm{A}\cdot\mathrm{s})/(\mathrm{V}\cdot\mathrm{m})]\cdot[\mathrm{m/s}] = [\mathrm{A/V}]$, so the physical dimension of $(\varepsilon_0 c/2)S_k$ is hence $[\mathrm{A/V}]\cdot[\mathrm{V}^2/\mathrm{m}^2] = [\mathrm{W/m}^2]$, as expected for an intensity measure (power per unit area in the plane orthogonal to the direction of wave propagation).

**2003-12-10** [v.1.16] `<jonsson@uni-wuppertal.de>`
Added the `--intensityinfo` option, in order to track down the maximum optical intensity that is present inside (or outside) the magneto-optical grating structure.

**2003-12-17** [v.1.17] `<jonsson@uni-wuppertal.de>`
Added the `--trmtraject` option, in order to check the polarization state evolution of the transmitted light for a varying incident intensity, keeping the incident polarization state fixed. Typically, we use two-dimensional interpolation to get the trajectory of the transmitted polarization state as function of the incident light (the incident light typically being of a fixed polarization state, with a varying intensity). This trajectory can now be used as input to the MAGBRAGG program in the invserse formulation of the problem, for the generation of Poincaré maps corresponding to cases with constant incident polarization states and varying input intensity.

**2004-03-10** [v.1.18] `<jonsson@uni-wuppertal.de>`
Last week in Stockholm I bought a new computer, a silvery Apple Macintosh Powerbook G4 running Apple OS X (10.3 Panther). As I recompiled the CWEB source for the MAGBRAGG program, still using the GNU C-compiler (GCC) for the executable file, I got complaints about the definition of the `cabs` routine as shadowing a previously defined function. This is a complaint that I never previously had with GCC under CYGWIN and Windows 2000, and it is obvious that the `math.h` header file of GCC has been slightly changed lately. However, after globally changing the routine name from `cabs` to `cdabs` (which anyway is better since the new name also indicates that it takes complex numbers in *double* precision as input), there were no more complaints, and the program now executes as expected in the OS X environment.

**2004-04-23** [v.1.19] `<jonsson@uni-wuppertal.de>`
Fixed a bug in the initialization of chirped modulation of magneto-optical parameters, for which the last $z$-coordinate of the discretized grating, $z_N$, never was set.

**2004-04-24** [v.1.20] `<jonsson@uni-wuppertal.de>`
Added the `--gyroperturb` option in order to provide a way of locally manipulating and perturbing the gyration constant of the medium. This option was added since I got this idea that a perturbation introduced by external means, for example via a current carrying wire oriented orthogonally to the direction of propagation of light in the Faraday configuration, could be used for opening up a window in the transmission window of a chirped Bragg grating. The syntax of the `--gyroperturb` option is simply `--gyroperturb` $\langle z_{\mathrm{p}}\rangle$ $\langle a_{\mathrm{p}}\rangle$ $\langle w_{\mathrm{p}}\rangle$, where $\langle z_{\mathrm{p}}\rangle$ is the centre position, $\langle a_{\mathrm{p}}\rangle$ is the zero-to-peak amplitude of change of the gyration constant $g$, and $\langle w_{\mathrm{p}}\rangle$ is the corresponding full width half maximum of the perturbation.

**2004-04-26** [v.1.21] `<jonsson@uni-wuppertal.de>`
Added the `--stokesspectrum` option, so that it is possible to generate full Poincaré maps of the spectral properties of a magneto-optical Bragg grating. However, after having introduced this option, I get the following message from CWEAVE:

```
cweave magbragg
This is CWEAVE, Version 3.64 (Web2C 7.5.2)
*1*3*4*5*6*7*36*37*38*55*60*67*68*70
Writing the output file...*1*3*4*5*6*7*36*37*38*55*60
!  Sorry, scrap/token/text capacity exceeded.  (l.  2912)
   sprintf(outfilename_w1,"%s%s",
                                    outfilename,".w1.dat");
(That was a fatal error, my friend.)
make:  *** [magbragg.tex] Error 1
```

The C code generated by CTANGLE compiles and executes perfectly, but obviously something is obstructing CWEAVE to properly generating the TEX code for the documentation. Most annoying.

**2004-05-07** [v.1.22] `<fredrik.jonsson@nmrc.ie>`
Added the `--displaysurrmedia`, which toggles if the program should write also the surrounding media to saved grating profiles or not. This is useful if one wish to just generate some part of a grating for a figure illustrating a particular refractive index distribution, cut exactly to the specified spatial interval of interest. As default, the MAGBRAGG program writes also the surrounding media to the ends of the grating file, so by using this option only once at the command line forces the program to cut the grating file exactly to the specified spatial interval. The annoying compilation error from 2004-04-26 is still present, preventing me from generating the documentation.

**2004-07-03** [v.1.23] `<fredrik.jonsson@nmrc.ie>`
I am in Germany for laboratory work in Wuppertal during two weeks, and this free Saturday morning I decided to once and for all trace down and eliminate the annoying parsing error that CWEAVE produces. (I am currently writing this entry at Starbucks in Cologne, being the only café in town with a non-smoking policy ...) Since this particular error seemed to stem from the block related to parsing of the command line options, and since this block anyway by now has grown over any reasonable size, I decided split it into several smaller blocks instead. Having done so, CWEAVE immediately accepted the CWEB source and extracted the TEX source, which subsequently were compiled without any errors. It thus seems like I on April 26th must have passed some upper size limit on the source allowed in one single CWEB block. Not that I really expected such a built-in constraint in CWEB, but in some sense I can agree on that by putting some hard upper limit, one will at least force the programmer to structure the code into smaller blocks, probably increasing the readability. As of today, the CWEB source for the MAGBRAGG program (source file `magbragg.w`) comprises 171418 bytes and 4138 lines of code. The size of the compiled executable is 70340 bytes, and the PostScript documentation is 808345 bytes (92 pages of A4 output in 10pt).

**2004-11-14** [v.1.24] `<fredrik.jonsson@nmrc.ie>`
Added the `--apodize` option, to be able to get rid of some unwanted Gibbs oscillations at the ends of the reflectance band of spectra generated for chirped gratings, aimed to generate nice spectra for a talk to be presented at the MRS 2004 Fall Meeting [F. Jonsson and C. Flytzanis, *Artificially Induced Perturbations in Chirped Magneto-Optical Bragg Gratings*, in *Magneto-Optical Materials for Photonics and Recording*, Eds. Koji Ando, W. Challener, R. Gambino and M. Levy, Mater. Res. Soc. Symp. Proc. **834**, J1.8 (Materials Research Society, Warrendale, 2005)]. Also added the `--phasejump` option (short form `-j`) to allow specification of a discrete phase jump to appear in sinusoidal or chirped grating structures. In adding these options, the same error as of April 26th appeared again, in the block related to the parsing of command line options. Since this block has again grown over any reasonably readable size, I hence started to split this

block into smaller CWEB sub-blocks, after which there were no further complaints from CWEAVE in extracting the TeX documentation of the program. As of today, the CWEB source for the MAGBRAGG program (source file `magbragg.w`) comprises 179296 bytes and 4312 lines of code. The size of the compiled executable is 74436 bytes, and the PostScript documentation is 835671 bytes (95 pages of A4 output in 10pt).

**2004-12-04** [v.1.25] `<fredrik.jonsson@nmrc.ie>`
Changed the way of output of Stokes parameters to file. Previously, the program always opened files with extensions `.s0.dat`,..., `.s3.dat` for output of the incident $S_k$ parameters (and similarly for the reflected and transmitted Stokes parameters $V_k$ and $W_k$), irregardless of the number of sampling points in intensity and ellipticity. However, for sampling of spectral characteristics we most often only encounter a simulation performed at one single intensity and ellipticity ($M_{\rm i} = 1$ and $M_{\rm e} = 1$), which means that, previously, twelve empty files were opened and closed for each simulation. This is now changed so that the program only opens those files for output in "topological" mode of operation, in which $M_{\rm i} \geq 2$ and $M_{\rm e} \geq 2$. Compiling the C code with GCC then caused some novel complaints using the `--pedantic` option, regarding the risk of using the associated file pointers uninitialized. That this complaint appear at all might be an indicator that GCC is not that strict in checking the logical state of the program in which the file pointers were to be used, since I verified that there was no reason whatsoever for the warnings. The "quick-and-dirty" solution to get rid of the annoying and non-justified warnings was to simply initialize all file pointers to `NULL` at the beginning of the program. Also started to write a separate section on all command line options currently supported by MAGBRAGG. The documentation has now for quite a while been in urgent need of such a section, since the number of options have grown quite a lot during the last year. As a start, the documentation for the `--grating`, `--phasejump`, and `--apodize` options was updated. As of today, the CWEB source for the MAGBRAGG program (source file `magbragg.w`) comprises 192945 bytes and 4611 lines of code. The size of the compiled executable is 74436 bytes, and the PostScript documentation is 873757 bytes (102 pages of A4 output in 10pt).

**2004-12-05** [v.1.26] `<fredrik.jonsson@nmrc.ie>`
Added the feature that the program now uses `time.h` to extrapolate what the estimated time of arrival (ETA) of the simulation is. This is useful for predicting the total simulation time for large numbers of sampled layers, such in long sinusoidal or chirped gratings. The estimation is simply done via linear extrapolation as

$$t_{\rm ETA} = t_0 + \frac{t - t_0}{\{\%\,\text{finished}\}} \times 100,$$

to produce run-time messages of the form

```
Program execution started Sun Dec  5 20:55:27 2004
-----------------------------------------------------------------
 ...10 percent finished...        ETA: Sun Dec  5 21:44:57 2004
 ...20 percent finished...        ETA: Sun Dec  5 21:42:12 2004
 ...30 percent finished...        ETA: Sun Dec  5 21:41:17 2004
 ...40 percent finished...        ETA: Sun Dec  5 21:40:49 2004
 ...50 percent finished...        ETA: Sun Dec  5 21:40:33 2004
 ...60 percent finished...        ETA: Sun Dec  5 21:40:20 2004
 ...70 percent finished...        ETA: Sun Dec  5 21:40:12 2004
 ...80 percent finished...        ETA: Sun Dec  5 21:40:05 2004
 ...90 percent finished...        ETA: Sun Dec  5 21:40:01 2004
 ...done.                      Elapsed execution time:  2644 s
-----------------------------------------------------------------
```

```
Program execution closed Sun Dec  5 21:39:31 2004
```

Also wrote documentation on command-line specifications of chirped gratings, and cleaned up the declarations of local variables somewhat. As of today, the CWEB source for the MAGBRAGG program (source file `magbragg.w`) comprises 203691 bytes and 4830 lines of code. The size of the compiled executable is 74532 bytes, and the PostScript documentation is 898974 bytes (105 pages of A4 output in 10pt).

**2004-12-11** [v.1.27] `<fredrik.jonsson@nmrc.ie>`
Added two schematic figures to the documentation on the discretization of the grating.

**2005-04-22** [v.1.28] `<fj@phys.soton.ac.uk>`
Removed an unused block in the code for saving spectra to file, after having thoroughly checked that it would have no affect on the backward compatibility of the program to earlier data generated. Also increased the numerical precision of the data written to file for the intensity reflection and transmission spectra, which now yields `%-10.8f` in floating point conversion as conforming to ANSI C.

**2005-04-28** [v.1.29] `<fj@phys.soton.ac.uk>`
Added four blocks of text in the documentation of command line options.

**2005-06-08** [v.1.30] `<fj@phys.soton.ac.uk>`
In Stockholm for two months. Added the standard C library function *fflush*( ) for enforcement in writing of all buffered calculated data to file. This in order to be able to follow the process of calculation more direct by file inspection. When executing MAGBRAGG under the new distribution of CYGWIN for Windows XP, the block for the displaying of status of calculation suddenly behaves odd, showing estimated execution times well below any reasonable estimated time of arrival, and also going well beyond the maximum 100 percent in relative execution progress. This will have to be checked, and has never previously appeared when compiling with GCC, neither under Apple OS X(BSD), nor under CYGWIN.

**2005-08-10** [v.1.31] `<fj@phys.soton.ac.uk>`
Back in Southampton again with my family after a hot summer. Wrote the code for the *strip_away_path*( ) routine originally for the POINCARE program and immediately decided to adopt the code also into the MAGBRAGG program in order to finally solve the problem with long path strings that appear in the program name string whenever poincare is called with an explicit path specified at the command line. The call to the *strip_away_path*( ) routine is located in the beginning of the block for command line parsing. As of today, the CWEB source for the MAGBRAGG program (source file `magbragg.w`) comprises 219666 bytes and 5172 lines of code. The size of the compiled (CYGWIN) executable is 96647 bytes, and the PostScript documentation is 967813 bytes (112 pages of A4 output in 10pt).

**2005-08-11** [v.1.32] `<fj@phys.soton.ac.uk>`
Cleaned up the blocks for displaying on-screen help messages. Wrote two routines *hl*( ) and *fhl*( ) to assist a coherent style in displaying help on command-line options.

**2005-08-19** [v.1.33] `<fj@phys.soton.ac.uk>`
Fixed two remaining bugs in the *hl*( ) and *fhl*( ), in which GCC under OS X this evening complained that long unsigned integers were sent to standard terminal output using the regular integer conversion of standard C. This warning did not show as the code was compiled with GCC under CYGWIN, hence there seem to be some discrepancy between different ports of the otherwise reliable compilator.

**2005-09-15** [v.1.34] `<fj@phys.soton.ac.uk>`
Corrected an error in the documentation of the options concerning specifications of chirped sinusoidal grating structures. The chirp parameter is now properly defined and described by example.

**2005-12-31** [v.1.35] `<fj@phys.soton.ac.uk>`
The theoretical description of the algorithm behind the solving of the inverse problem in the MAGBRAGG program is included in an article which today has been accepted for publication in Physical Review Letters. Added a section on the theoretical basis for the algorithm behind the program, deriving the algorithm from the electromagnetic wave equation in a manner similar to the description which soon will appear in the published article.

**2006-01-22** [v.1.36] `<fj@phys.soton.ac.uk>`
Added a section on the Butcher and Cotter convention of degeneracy factors in nonlinear optics, picked from my *Lecture Notes on Nonlinear Optics* from the course I gave at the Royal Institute of Technology in 2003. Also edited the section on the theoretical basis for the algorithm of calculation and added a figure describing the representation of the polarization state on the Poincaré sphere. As of today, the CWEB source for the MAGBRAGG program (source file `magbragg.w`) comprises 252108 bytes and 5937 lines of code. The size of the compiled (OS X) executable is 78808 bytes, and the PostScript documentation is 1651977 bytes (122 pages of A4 output in 10pt).

**2006-01-24** [v.1.37] `<fj@phys.soton.ac.uk>`
Added a significant number of comments on the use of variables and their initialization. Also cleaned up and added more instructive text on the actual compilation of the CWEB source code. As of today, the CWEB source for the MAGBRAGG program (source file `magbragg.w`) comprises 261933 bytes and 6086 lines of code. The size of the compiled (OS X) executable is 78828 bytes, and the PostScript documentation is 1673601 bytes (125 pages of A4 output in 10pt).

**2006-02-09** [v.1.38] `<fj@phys.soton.ac.uk>`
As I now for a longer time of period have sketched on implementing the exact methodology of solution for the waves inside the homogeneous elements of the discretized medium, I today added the first sections on the theoretical part of a rigorous theory of wave propagation. As of today, the CWEB source for the MAGBRAGG program (source file `magbragg.w`) comprises 305905 bytes and 7095 lines of code. The size of the compiled (OS X) executable is 88002 bytes, and the PostScript documentation is 1778021 bytes (135 pages of A4 output in 10pt).

**2006-02-11** [v.1.39] `<fj@phys.soton.ac.uk>`
Added two sections on how the amplitude evolution in the rigorous theory of wave propagation actually can be reduced to the evaluation of an elliptic function of the standard form

$$\int_{v(\zeta_0)}^{v(\zeta)} \frac{dv}{(a_4v^4+a_3v^3+a_2v^2+a_1v+a_0)^{1/2}},$$

with its solution $v(z)$ expressed in terms of the Weierstrass elliptic function $\wp(\zeta;g_2,g_3)$ with the invariants $g_2$ and $g_3$ explicitly expressed in terms of $a_0$, $a_1$, $a_2$, $a_3$, and $a_4$.

**2006-03-13** [v.1.40] `<fj@phys.soton.ac.uk>`
After having spent some considerable time trying to verify that the integral equation

$$\int_{v(\zeta_0)}^{v(\zeta)} \frac{dv}{(a_4v^4+4a_3v^3+6a_2v^2+4a_1v+a_0)^{1/2}},$$

which naturally appear in the rigorous theory of cross-phase modulation in nonlinear magneto-optical media, really has the explicit solution

$$v(\zeta)=v_0+\frac{\sqrt{f(v_0)}\wp'(\zeta)+\frac{1}{2}f'(v_0)[\wp(\zeta)-\frac{1}{24}f''(v_0)]+\frac{1}{24}f(v_0)f'''(v_0)}{2[\wp(\zeta)-\frac{1}{24}f''(v_0)]^2-\frac{1}{48}f(v_0)f^{(\mathrm{iv})}(v_0)}.$$

with quartic invariants

$$g_2 \equiv a_0a_4 - 4a_1a_3 + 3a_2^2, \qquad g_3 \equiv \begin{vmatrix} a_0 & a_1 & a_2 \\ a_1 & a_2 & a_3 \\ a_2 & a_3 & a_4 \end{vmatrix} = a_0a_2a_4 + 2a_1a_2a_3 - a_2^3 - a_0a_3^2 - a_1^2a_4,$$

I today realized that the first edition of E. T. Whittaker's *A Course of Modern Analysis* (Cambridge University Press, Cambridge, 1902) actually has a printing error in one of the terms in the denominator of the solution. This error has obviously been fixed in later editions of the book, for example in E. T. Whittaker and G. N. Watson, *A Course of Modern Analysis*, 4th Reprinted Edn. (Cambridge University Press, Cambridge, 1996), ISBN 0-521-58807-3, but it caused be a considerable nuisance before I was able to track the error down. The verification of the solution was checked using the following blocks of MapleV code:

```
restart:
f:=a[0]*v^4+4*a[1]*v^3+6*a[2]*v^2+4*a[3]*v+a[4];
g[2]:=a[0]*a[4]-4*a[1]*a[3]+3*a[2]^2;
g[3]:=a[0]*a[2]*a[4]+2*a[1]*a[2]*a[3]-a[2]^3-a[0]*a[3]^2-a[1]^2*a[4];
df[0]:=eval(f,v=v0):
df[1]:=eval(diff(f,v$1),v=v0):
df[2]:=eval(diff(f,v$2),v=v0):
df[3]:=eval(diff(f,v$3),v=v0):
df[4]:=eval(diff(f,v$4),v=v0):
tmp[1]:=sqrt(df[0])*WeierstrassPPrime(z,g[2],g[3]):
tmp[2]:=(1/2)*df[1]*(WeierstrassP(z,g[2],g[3])-(1/24)*df[2]):
tmp[3]:=(1/24)*df[0]*df[3]:
tmp[4]:=2*(WeierstrassP(z,g[2],g[3])-(1/24)*df[2])^2:
tmp[5]:=(1/48)*df[0]*df[4]:
v:=v0+(tmp[1]+tmp[2]+tmp[3])/(tmp[4]-tmp[5]):
p:=a[0]*v^4+4*a[1]*v^3+6*a[2]*v^2+4*a[3]*v+a[4]:
testfunc:=(diff(v,z))^2-p:
testfunc:=simplify(testfunc);
```

As this result was the only missing link in the formulation of an explicit solution to the wave propagation problem in nonlinear magneto-optical media, taking into account also weakly phase-mismatched nonlinear source terms, the only task remaining now is to formulate the algorithm for solving the inverse problem using this more stringent method.

**2006-03-20** [v.1.41] `<fj@phys.soton.ac.uk>`
Added a few paragraphs on the rigorous theory of wave propagation, clarifying the role of the constant of integration $\Gamma$ and its calculation from the phase of the transmitted field and the amplitude-related invariants of propagation. This now interconnects more naturally to the formulation of the wave propagation as an inverse problem, being the natural mode of solving for reflectances as well as transmittances. I also took the opportunity to write an email to MathWorld, pointing out that their solution to the Weierstrass form of the integral equation, `http://mathworld.wolfram.com/EllipticIntegral.html`, Eqs. (51)–(56), actually contains errors; of course I could not resist the opportunity to provide the MapleV blocks I used for my own verification, rather than using Mathematica code!

**2006-05-01** [v.1.42] `<fj@phys.soton.ac.uk>`
Added support for initialization of the grating structure as a fractal set, of the Cantor fractal type. This possibility is in the program now accessed via the command-line option `--grating fractal cantor [options]`. The initialization is done using recursion with the function *init_cantor_fractal_grating*( ), which was finished today. As of today, the CWEB source for the MAGBRAGG program (source file `magbragg.w`) comprises 335649 bytes and 7753 lines of code. The size of the compiled (CYGWIN) executable is 95217 bytes, and the PostScript documentation is 1874643 bytes (148 pages of A4 output in 10pt).

**2007-01-10** [v.1.43] `<http://jonsson.eu>`
Seefeld, Austria. Midnight. Cleaned up the theoretical part preceeding the algorithm of computation and merged all bibliographical references into one separate section at the end of the document.

**2011-12-18** [v.1.44] `<http://jonsson.eu>`
Updated `Makefile`:s for the generation of figures. Also corrected a rather stupid way of removing preceeding paths of file names.

**27. Compiling the source code.** The program is written in CWEB, generating ANSI C (ISO C90) conforming source code and documentation as plain TEX-source, and is to be compiled using the sequences as outlined in the `Makefile` listed below.

```
#
# Makefile designed for use with ctangle, cweave, gcc, and plain TeX.
#
# The CTANGLE program converts a CWEB source document into a C program which
# may be compiled in the usual way.  The CWEAVE program converts the same CWEB
# file into a TeX file that may be formatted and printed in the usual way.
#
# Copyright (C) 2002-2006, Fredrik Jonsson <fj@phys.soton.ac.uk>
#
CTANGLE   = ctangle
CWEAVE    = cweave
CC        = gcc
CCOPTS    = -O2 -Wall -ansi -std=iso9899:1990 -pedantic
LNOPTS    = -lm
TEX       = tex
DVIPS     = dvips
DVIPSOPTS = -ta4 -D1200
METAPOST  = mp

all:  magbragg magbragg.ps

magbragg:  magbragg.o
         $(CC) $(CCOPTS) -o magbragg magbragg.o $(LNOPTS)

magbragg.o:  magbragg.w
         $(CTANGLE) magbragg
         $(CC) $(CCOPTS) -c magbragg.c

magbragg.ps:  magbragg.dvi
         $(DVIPS) $(DVIPSOPTS) magbragg.dvi -o magbragg.ps

magbragg.dvi:  magbragg.w
         make -C figures/
         $(CWEAVE) magbragg
         $(TEX) magbragg.tex

clean:
         make clean -ik -C figures/
         -rm -Rf magbragg *  *.c *.o *.exe *.dat *.tgz *.pdf
         -rm -Rf *.tex *.aux *.log *.toc *.idx *.scn *.dvi *.ps

archive:
         make -ik clean
         tar --gzip --directory=../ -cf magbragg.tgz magbragg
```

This `Makefile` essentially executes two major calls. First, the CTANGLE program parses the CWEB source document `magbragg.w` to extract a C source file `magbragg.c` which may be compiled in the usual way using any ANSI C conformant compiler. The output source file includes `#line` specifications so that any debugging can be done conveniently in terms of the original CWEB source file. Second, the CWEAVE program parses the same CWEB source file to extract a plain TEX source file `magbragg.tex` which may be compiled in the usual way. It takes appropriate care of typographic details like page layout and the use of indentation, italics, boldface, and so on, and it supplies extensive cross-index information that it gathers automatically.

After having executed `make` in the same catalogue where the files `magbragg.w` and `Makefile` are located, one is left with an executable file `magbragg`, being the ready-to-use compiled program, and a PostScript file `magbragg.ps` which contains the full documentation of the program, that is to say the document you currently are reading. Notice that on platforms running Windows NT, Windows 2000, Windows ME, or any other operating system by Microsoft, the executable file will instead automatically be called `magbragg.exe`. This convention also applies to programs compiled under the UNIX-like environment CYGWIN.

**28. Running the program.** The program is entirely controlled by the command line options supplied when invoking the program. Since the command line for some problems tend to be quite lengthy, since all material parameters and optical field inputs to the program must be specified, it is convenient to put blocks for the program calls into a Makefile, which makes it easy to maintain a structure in the simulations, as well as ensuring traceability of the steps in the generation of graphs.

As an example of such a call, the following block is included as an example in the enclosed Makefile:

```
testsimulation:
     @for g in 0.0 1.0 2.0; do \
          ./magbragg --verbose --outputfile fig1-$$g \
               --spectrumfile fig1-$$g.rsp.dat \
               --gratinglength 7.326376e-6 -N 1800 \
               --grating sinusoidal n 2.0550 0.1250 366.3188e-9 \
                       g $$g'e-3' 0.0 1.0 \
                       pe 0.0 0.0 1.0 \
                       pm 0.0 0.0 1.0 \
                       qe 0.0 0.0 1.0 \
                       qm 0.0 0.0 1.0 \
               --refindsurr 2.0550 -M 2000 \
               --lambdastart 1300.0e-9 --lambdastop 1700.0e-9 \
               --trmintensity 7.0e8 7.0e8 1 \
               --trmellipticity 0.0 0.0 1 ;\
      done
```

In the following sections, a complete listing of all command line options accepted by the MAGBRAGG program is presented.

**29. Specifying grating types.** The MAGBRAGG program currently accepts specifications of three different main types of gratings: stepwise gratings, sinusoidal gratings, and chirped sinusoidal gratings. The stepwise gratings are simply stacks of homogeneous layers, stacked to form a grating, and these gratings can be composed of two or more different materials. As an important subclass of the stepwise gratings, the binary gratings are probably the most important ones, being composed of alternating layers of only two types of media.

Throughout the program, the following definitions of the material parameters apply, to the refractive index $n(z)$, gyration coefficient $g(z)$, nonlinear optical parameters $p^{(\mathrm{e})}(z)$ and $q^{(\mathrm{e})}(z)$, and nonlinear magneto-optical parameters $p^{(\mathrm{m})}(z)$ and $q^{(\mathrm{m})}(z)$,

$$\begin{aligned}
n[k] &= n_k = [1+\chi^{(\mathrm{ee})}_{xx}]^{1/2},\\
g[k] &= g_k = i\chi^{(\mathrm{eem})}_{xyz}B^z_0/(2n_k),\\
pe[k] &= p^{(\mathrm{e})}_k = \chi^{\mathrm{eeee}}_{xxxx} - \chi^{\mathrm{eeee}}_{xyyx},\\
qe[k] &= q^{(\mathrm{e})}_k = \chi^{\mathrm{eeee}}_{xxxx} + \chi^{\mathrm{eeee}}_{xyyx},\\
pm[k] &= p^{(\mathrm{m})}_k = i(\chi^{\mathrm{eeeem}}_{xyyyz} - \chi^{\mathrm{eeeem}}_{xxxyz})B^z_0,\\
qm[k] &= q^{(\mathrm{m})}_k = i(\chi^{\mathrm{eeeem}}_{xyyyz} + \chi^{\mathrm{eeeem}}_{xxxyz})B^z_0,
\end{aligned}$$

where $n[k]$, $g[k]$, $pe[k]$, $qe[k]$, $pm[k]$, $qm[k]$ denote the variables as internally used in the MAGBRAGG program to store the corresponding material data for the $N-1$ homogeneous segments $z_k \le z < z_{k+1}$, $k = 1, 2, \ldots, N-1$.

In terms of the Verdet constant $V_k$ of a homogeneous layer, as being the commonly used measure of the linear magneto-optical rotational strength of the material, the $g$-parameter of the MAGBRAGG program is expressed as

$$g_k = V_k B^z_0 c/\omega,$$

which simply follows from the general relation

$$i\chi^{(\mathrm{eem})}_{xyz} = 2ncV/\omega.$$

**30.** Stepwise grating structures. The stepwise grating structures are specified using the command line option

```
--grating stepwise [twolevel⟨...⟩|threelevel⟨...⟩]
```

where `twolevel`, `threelevel` etc., states the number of materials used in the stacking of the layers. For the `twolevel` (binary) type of gratings, the syntax is

`--grating stepwise twolevel t1` $\langle t_1\rangle$ `t2` $\langle t_2\rangle$ `n1` $\langle n_1\rangle$ `n2` $\langle n_2\rangle$ `g1` $\langle g_1\rangle$ `g2` $\langle g_2\rangle$
`pe1` $\langle p^{(\mathrm{e})}_1\rangle$ `pe2` $\langle p^{(\mathrm{e})}_2\rangle$ `pm1` $\langle p^{(\mathrm{m})}_1\rangle$ `pm2` $\langle p^{(\mathrm{m})}_2\rangle$
`qe1` $\langle q^{(\mathrm{e})}_1\rangle$ `qe2` $\langle q^{(\mathrm{e})}_2\rangle$ `qm1` $\langle q^{(\mathrm{m})}_1\rangle$ `qm2` $\langle q^{(\mathrm{m})}_2\rangle$

where $t_1$ and $t_2$ are the geometrical layer thicknesses of the first two layers $0 \le z \le t_1$ and $t_1 \le z \le t_1 + t_2$, and $n_1$ and $n_2$ are the corresponding refractive indices of these layers, respectively.

The grating is then composed by repeating the basic pair of layers, to give a grating composed of in total $N-1$ homogeneous layers, as specified with the `-N` option. Notice that if the total number of layers $N-1$ is an odd number (i. e. specifying an even number of interfaces $N$), the last layer will possess the same material properties and geometrical thickness as the first layer.

**31.** Sinusoidal structures. The sinusoidal grating structures are specified using the command line option

$$\texttt{--grating sinusoidal n}\ \langle n_0\rangle\ \langle\Delta n\rangle\ \langle\Lambda_n\rangle\ \texttt{g}\ \langle g_0\rangle\ \langle\Delta g\rangle\ \langle\Lambda_g\rangle$$
$$\texttt{pe}\ \langle p_0^{(\mathrm{e})}\rangle\ \langle\Delta p_0^{(\mathrm{e})}\rangle\ \langle\Lambda_{\mathrm{p}}^{(\mathrm{e})}\rangle\ \texttt{pm}\ \langle p_0^{(\mathrm{m})}\rangle\ \langle\Delta p_0^{(\mathrm{m})}\rangle\ \langle\Lambda_{\mathrm{p}}^{(\mathrm{m})}\rangle$$
$$\texttt{qe}\ \langle q_0^{(\mathrm{e})}\rangle\ \langle\Delta q_0^{(\mathrm{e})}\rangle\ \langle\Lambda_{\mathrm{q}}^{(\mathrm{e})}\rangle\ \texttt{qm}\ \langle q_0^{(\mathrm{m})}\rangle\ \langle\Delta q_0^{(\mathrm{m})}\rangle\ \langle\Lambda_{\mathrm{q}}^{(\mathrm{m})}\rangle$$

In terms of the supplied command line options, the resulting grating structure is then described by the continuous distribution of refractive index, gyration coefficient, and nonlinear optical and magneto-optical coefficients as

$$\begin{aligned}
n(z) &= n_0 + \Delta n \sin(2\pi z/\Lambda_n),\\
g(z) &= g_0 + \Delta g \sin(2\pi z/\Lambda_g),\\
p^{(\mathrm{e})}(z) &= p_0^{(\mathrm{e})} + \Delta p^{(\mathrm{e})} \sin(2\pi z/\Lambda_{\mathrm{p}}^{(\mathrm{e})}),\\
q^{(\mathrm{e})}(z) &= q_0^{(\mathrm{e})} + \Delta q^{(\mathrm{e})} \sin(2\pi z/\Lambda_{\mathrm{q}}^{(\mathrm{e})}),\\
p^{(\mathrm{m})}(z) &= p_0^{(\mathrm{m})} + \Delta p^{(\mathrm{m})} \sin(2\pi z/\Lambda_{\mathrm{p}}^{(\mathrm{m})}),\\
q^{(\mathrm{m})}(z) &= q_0^{(\mathrm{m})} + \Delta q^{(\mathrm{m})} \sin(2\pi z/\Lambda_{\mathrm{q}}^{(\mathrm{m})}),
\end{aligned}$$

which in the discretized and oversampled model as used in the internal representration of the MAGBRAGG program hence becomes

$$\begin{aligned}
n_j &= n_0 + \Delta n \sin(2\pi z_j/\Lambda_n),\\
g_j &= g_0 + \Delta g \sin(2\pi z_j/\Lambda_g),\\
p_j^{(\mathrm{e})} &= p_0^{(\mathrm{e})} + \Delta p^{(\mathrm{e})} \sin(2\pi z_j/\Lambda_{\mathrm{p}}^{(\mathrm{e})}),\\
q_j^{(\mathrm{e})} &= q_0^{(\mathrm{e})} + \Delta q^{(\mathrm{e})} \sin(2\pi z_j/\Lambda_{\mathrm{q}}^{(\mathrm{e})}),\\
p_j^{(\mathrm{m})} &= p_0^{(\mathrm{m})} + \Delta p^{(\mathrm{m})} \sin(2\pi z_j/\Lambda_{\mathrm{p}}^{(\mathrm{m})}),\\
q_j^{(\mathrm{m})} &= q_0^{(\mathrm{m})} + \Delta q^{(\mathrm{m})} \sin(2\pi z_j/\Lambda_{\mathrm{q}}^{(\mathrm{m})}),
\end{aligned}$$

for $j = 1, 2, \ldots, N-1$. For sinusoidal gratings, discrete phase jumps of the refractive index distribution can also be applied, by using the `--phasejump` command line option. Subsequent apodization of the structure can also be applied, using the –apodize option.

**32.** Sinusoidal chirped structures. The chirped grating structures, being sinusoidal gratings with a spatially varying grating period, are specified using the somewhat extensive command line option

$$\texttt{--grating chirped n}\ \langle n_0\rangle\ \langle\Delta n\rangle\ \langle\Lambda_n\rangle\ \langle\eta_n\rangle\ \texttt{g}\ \langle g_0\rangle\ \langle\Delta g\rangle\ \langle\Lambda_g\rangle\ \langle\eta_g\rangle$$
$$\texttt{pe}\ \langle p_0^{(\mathrm{e})}\rangle\ \langle\Delta p_0^{(\mathrm{e})}\rangle\ \langle\Lambda_\mathrm{p}^{(\mathrm{e})}\rangle\ \langle\eta_\mathrm{p}^{(\mathrm{e})}\rangle\ \texttt{pm}\ \langle p_0^{(\mathrm{m})}\rangle\ \langle\Delta p_0^{(\mathrm{m})}\rangle\ \langle\Lambda_\mathrm{p}^{(\mathrm{m})}\rangle\ \langle\eta_\mathrm{p}^{(\mathrm{m})}\rangle$$
$$\texttt{qe}\ \langle q_0^{(\mathrm{e})}\rangle\ \langle\Delta q_0^{(\mathrm{e})}\rangle\ \langle\Lambda_\mathrm{q}^{(\mathrm{e})}\rangle\ \langle\eta_\mathrm{q}^{(\mathrm{e})}\rangle\ \texttt{qm}\ \langle q_0^{(\mathrm{m})}\rangle\ \langle\Delta q_0^{(\mathrm{m})}\rangle\ \langle\Lambda_\mathrm{q}^{(\mathrm{m})}\rangle\ \langle\eta_\mathrm{q}^{(\mathrm{m})}\rangle$$

Here $\langle n_0\rangle$ is the bias refractive index across the grating, $\langle\Delta n\rangle$ the modulation of the refractive index around the bias value, that is to say half the bottom-to-peak value of the modulated index of refraction, $\langle\Lambda_n\rangle$ is the initial geometrical grating period at $z = 0$, and $\langle\eta_n\rangle$ the chirp of the grating, being the relative change of the grating period over the length of the grating. The other grating parameters are specified analogously, for gyration coefficient and nonlinear optical and magneto-optical coefficients. The chirp parameter is here defined as the change in geometrical grating period per unit geometrical distance along the grating, or

$$\eta_n = \frac{\Lambda(L) - \Lambda(0)}{L}$$

where $\Lambda(z)$ denotes the geometrical grating period as function of the spatial coordinate $z$ along the grating.

To illustrate the meaning of the chirp parameter, consider a chirped refractive index distribution over a geometrical distance $L$ and with a bias refractive index $n_0$. To give a resonance in reflection ranging from vacuum wavelength $\lambda_\mathrm{a}$ to $\lambda_\mathrm{b}$, say, the grating period could be either increasing or decreasing with spatial coordinate $z$. By choosing the initial part of the grating (at $z = 0$) to be resonant in reflection for a vacuum wavelength $\lambda_\mathrm{a}$, the initial grating period $\Lambda_n$ should be chosen as half the optical period at resonance, or

$$\Lambda_n = \frac{\lambda_\mathrm{a}}{2n_0}.$$

With this initial grating period the chirp parameter should, in order to give a final resonance in reflection at vacuum wavelength $\lambda_\mathrm{b}$ at the end of the grating (at $z = L$), then simply be chosen as

$$\eta_n = \frac{\lambda_\mathrm{b} - \lambda_\mathrm{a}}{2n_0 L}.$$

Notice that whenever $\lambda_\mathrm{b} < \lambda_\mathrm{a}$, the chirp parameter is negative, indicating a grating period which is decreasing with increasing spatial coordinate $z$. This convention of sign for the chirp parameter is consistently kept throughout the algorithms of the MAGBRAGG program.

In terms of the supplied command line options, the resulting chirped grating structure is described by the continuous distribution of refractive index, gyration coefficient, and nonlinear optical and magneto-optical coefficients as

$$\begin{aligned}
n(z) &= n_0 + \Delta n \sin((2\pi/\eta_n)\ln(1 + \eta_n z/\Lambda_n)),\\
g(z) &= g_0 + \Delta g \sin((2\pi/\eta_g)\ln(1 + \eta_g z/\Lambda_g)),\\
p^{(\mathrm{e})}(z) &= p_0^{(\mathrm{e})} + \Delta p^{(\mathrm{e})} \sin((2\pi/\eta_\mathrm{p}^{(\mathrm{e})})\ln(1 + \eta_\mathrm{p}^{(\mathrm{e})} z/\Lambda_\mathrm{p}^{(\mathrm{e})})),\\
p^{(\mathrm{m})}(z) &= p_0^{(\mathrm{m})} + \Delta p^{(\mathrm{m})} \sin((2\pi/\eta_\mathrm{p}^{(\mathrm{m})})\ln(1 + \eta_\mathrm{p}^{(\mathrm{m})} z/\Lambda_\mathrm{p}^{(\mathrm{m})})),\\
q^{(\mathrm{e})}(z) &= q_0^{(\mathrm{e})} + \Delta q^{(\mathrm{e})} \sin((2\pi/\eta_\mathrm{q}^{(\mathrm{e})})\ln(1 + \eta_\mathrm{q}^{(\mathrm{e})} z/\Lambda_\mathrm{q}^{(\mathrm{e})})),\\
q^{(\mathrm{m})}(z) &= q_0^{(\mathrm{m})} + \Delta q^{(\mathrm{m})} \sin((2\pi/\eta_\mathrm{q}^{(\mathrm{m})})\ln(1 + \eta_\mathrm{q}^{(\mathrm{m})} z/\Lambda_\mathrm{q}^{(\mathrm{m})})),
\end{aligned}$$

which in the discretized and oversampled model as used in the internal representration of the MAGBRAGG program hence becomes

$$\begin{aligned}
n_j &= n_0 + \Delta n \sin((2\pi/\eta_n)\ln(1 + \eta_n z_j/\Lambda_n)),\\
g_j &= g_0 + \Delta g \sin((2\pi/\eta_g)\ln(1 + \eta_g z_j/\Lambda_g)),\\
p_j^{(\mathrm{e})} &= p_0^{(\mathrm{e})} + \Delta p^{(\mathrm{e})} \sin((2\pi/\eta_\mathrm{p}^{(\mathrm{e})})\ln(1 + \eta_\mathrm{p}^{(\mathrm{e})} z_j/\Lambda_\mathrm{p}^{(\mathrm{e})})),\\
p_j^{(\mathrm{m})} &= p_0^{(\mathrm{m})} + \Delta p^{(\mathrm{m})} \sin((2\pi/\eta_\mathrm{p}^{(\mathrm{m})})\ln(1 + \eta_\mathrm{p}^{(\mathrm{m})} z_j/\Lambda_\mathrm{p}^{(\mathrm{m})})),\\
q_j^{(\mathrm{e})} &= q_0^{(\mathrm{e})} + \Delta q^{(\mathrm{e})} \sin((2\pi/\eta_\mathrm{q}^{(\mathrm{e})})\ln(1 + \eta_\mathrm{q}^{(\mathrm{e})} z_j/\Lambda_\mathrm{q}^{(\mathrm{e})})),\\
q_j^{(\mathrm{m})} &= q_0^{(\mathrm{m})} + \Delta q^{(\mathrm{m})} \sin((2\pi/\eta_\mathrm{q}^{(\mathrm{m})})\ln(1 + \eta_\mathrm{q}^{(\mathrm{m})} z_j/\Lambda_\mathrm{q}^{(\mathrm{m})})),
\end{aligned}$$

for $j = 1, 2, \ldots, N-1$.

Notice that the geometrical period of any of the optical/physical properties described by the above distributions is given by

$$\frac{2\pi}{\{\text{geometrical period}\}} = \frac{\partial}{\partial z}\{\text{argument of } \sin(\ldots)\}.$$

For example, the refractive index distribution

$$n(z) = n_0 + \Delta n \sin((2\pi/\eta_n)\ln(1 + \eta_n z/\Lambda_n))$$

has its spatially varying local grating period $\Lambda_{\text{loc}}(z)$ given by

$$\frac{2\pi}{\Lambda_{\text{loc}}(z)} = \frac{\partial}{\partial z}[(2\pi/\eta_n)\ln(1 + \eta_n z/\Lambda_n)] = (2\pi/\eta_n)\frac{\eta_n/\Lambda_n}{(1 + \eta_n z/\Lambda_n)} = \frac{2\pi}{(\Lambda_n + \eta_n z)}$$

that is to say, the geometrical local period is given as a linear function which in terms of the vacuum resonance wavelengths $\lambda_\text{a}$ and $\lambda_\text{b}$ becomes

$$\Lambda_{\text{loc}}(z) = \Lambda_n + \eta z = \frac{\lambda_\text{a}}{2n_0} + \frac{(\lambda_\text{b} - \lambda_\text{a})}{2n_0 L}z = \frac{1}{2n_0}(\lambda_\text{a} + (\lambda_\text{b} - \lambda_\text{a})z/L).$$

For the sake of consistency with the discussion earlier in this section, $\lambda_\text{a}$ is taken as the vacuum resonance wavelength at $z = 0$, while $\lambda_\text{b}$ is the corresponding wavelength of resonance at $z = L$. For chirped gratings, as well as for the sinusoidal gratings, discrete phase jumps of the refractive index distribution can also be applied, by using the `--phasejump` command line option. Subsequent apodization of the structure can also be applied, using the –apodize option.

**33.** Fractal structures.
[TEXT STILL TO BE WRITTEN]

**34.** Discrete phase jumps. Discrete phase jumps in sinusoidal type gratings (of sinusoidal of chirped type as previously described) can be specified using the `--phasejump` option, with syntax

`--phasejump` $\langle\varphi_{\text{jump}}\rangle$ $\langle z_{\text{jump}}\rangle$

where $\varphi_{\text{jump}}$ is the phase shift in radians to be applied for the grating at all spatial coordinates $z \geq z_{\text{jump}}$. For sinusoidal type gratings, the effect is that the refractive index and gyration coefficient distributions become

$$n(z) = n_0 + \Delta n \sin(2\pi z/\Lambda_n + \varphi(z)),$$
$$g(z) = g_0 + \Delta g \sin(2\pi z/\Lambda_g + \varphi(z)),$$

where

$$\varphi(z) = \begin{cases} 0, & z < z_{\text{jump}} \\ \varphi_{\text{jump}}, & z \geq z_{\text{jump}} \end{cases},$$

Notice that the parameters supplied with the `--phasejump` option only affects the (linear) refractive index and gyration coefficient distributions $n(z)$ and $g(z)$; all other (nonlinear) material parameters are left unaffected, irregardless of potential spatial modulation schemes.

**35.** The refractive index surrounding the grating. The refractive index surrounding the grating is specified using the command line option

`--refindsurr` $\langle n_{\rm ext}\rangle$

where $\langle n_{\rm ext}\rangle$ is the refractive index surrounding the grating. If the surrounding refractive index is not specified at the command line at execution of the program, a default value of $n_{\rm ext} = 1.0$ (vacuum) is used in the simulation.

**36.** Apodization of the grating. Apodization of the grating structure is typically applied in order to get rid of any occurring Gibbs oscillations due to a rapid change of the index modulation at the ends of the grating. For a more detailed description of Gibbs oscillations, see for example Refs. [14,15]. By applying apodization, the MAGBRAGG program force a smoother transition between modulated and non-modulated regions of the grating. The apodization is invoked by the `--apodize` option, with syntax

`--apodize` $\langle L_{\rm apod}\rangle$

where $L_{\rm apod}$ is the apodization length at each end of the grating. The apodization is performed at the ends of the grating according to a multiplicative factor of the modulation amplitudes of the refractive index and gyration coefficient, of the form

$$f(z) = \begin{cases} [1-\cos(\pi z/L_{\rm apod})]/2, & 0 \le z \le L_{\rm apod}, \\ 1, & L_{\rm apod} < z < L - L_{\rm apod}, \\ [1-\cos(\pi(z-L)/L_{\rm apod})]/2, & L - L_{\rm apod} \le z \le L, \end{cases}$$

and otherwise $f(z) = 0$, for any $z$ outside the above domains of definition, where $L_{\rm apod}$ is the effective apodization length, being the floating point parameter specified after the `--apodize` option, and $L$ the geometrical overall length of the grating, e. g. as specified with the `--gratinglength` option. Notice that as in the previously described `--phasejump` option, the parameter supplied with the `--apodize` option only apodizes the (linear) refractive index and gyration coefficient distributions $n(z)$ and $g(z)$; all other (nonlinear) material parameter distributions are left unaffected.

**37.** Specifying transmitted intensity. The generated data relating a specified optical output to an optical input can in the program automatically be iterated over a range of values of the transmitted optical intensity, expressed in regular SI units as Watts per square meter (W/m$^2$). The command line syntax for specifying that the program should generate the inverse relation for $M_{\rm i}$ discrete values of the transmitted optical intensity in a range from $I_{\rm start}$ to $I_{\rm stop}$ is

`--trmintensity` $\langle I_{\rm start}\rangle$ $\langle I_{\rm stop}\rangle$ $\langle M_{\rm i}\rangle$

Notice that parameters throughout the program are to be specified in terms of regular SI units. In case the in optical physics quite commonly used quantity of gigawatts per square centimeter is preferred, just use the conversion $1\,{\rm GW/cm^2} = 10^{13}\,{\rm W/m^2}$.

**38.** Specifying transmitted ellipticity. The command line syntax for specification of the range of ellipticity of the transmitted polarization state is analogous to that of the transmitted intensity, as

`--trmellipticity` $\langle \epsilon_{\rm start}\rangle$ $\langle \epsilon_{\rm stop}\rangle$ $\langle M_{\rm e}\rangle$

where $\langle \epsilon_{\rm start}\rangle$ and $\langle \epsilon_{\rm stop}\rangle$ are the start and stop values for the normalized transmitted ellipticity $\epsilon$, which in terms of the transmitted Stokes parameters is defined as $\epsilon \equiv W_3/W_0$. (For the definitions of the Stokes parameters in terms of the electric field of the light wave, see the corresponding separate section included in this documentation.) Here $\epsilon = -1$ corresponds to right circular polarization (RCP) while $\epsilon = 1$ corresponds to left circular polarization (LCP), with $\epsilon = 0$ corresponding to linear polarization. Values intermediate of those give the various degrees of elliptic polarization states. As in the case of specification of the transmitted intensity, $\langle M_{\rm e}\rangle$ is the number of discrete steps of the ellipticity that will be iterated over in the simulation.

**39.** Specifying optical vacuum wavelength interval to be scanned. For the sampling of optical spectra, with the reflectance and transmittance sampled as function of optical vacuum wavelength, the interval of computation is specified using the syntax

`--lambdastart` $\langle\lambda_{\text{start}}\rangle$ `--lambdastop` $\langle\lambda_{\text{stop}}\rangle$

where $\lambda_{\text{start}}$ and $\lambda_{\text{stop}}$ specifies the wavelength domain. The number of samples $M$ to be calculated is specified using the syntax

`-M` $\langle M\rangle$

The specific discrete vacuum wavelength $\lambda_k$ at which the computation is performed are

$$\lambda_k = \lambda_{\text{start}} + \frac{(k-1)}{(M-1)}(\lambda_{\text{stop}} - \lambda_{\text{start}}),$$

for $k = 1, 2, \ldots, M$.

**40.** Scaling of Stokes parameters to yield optical intensity. To force the output Stokes parameters to be expressed in terms of optical intensity units of Watts per square meter, rather than the regular square Volts per meter, use the command line option

```
--scale_stokesparams 1.327209e-3
```

forcing the program to save the scaled Stokes parameters $S'_k = 1.327209 \times 10^{-3} S_k$ to file instead of the original $S_k$, $k = 0, 1, 2, 3$. As a default, and as a matter of convention of electrodynamics in SI units, all Stokes parameters are given in $\text{V}^2/\text{m}^2$, through their definition. For example, the incident field (which is calculated by the program in this inverse formulation of the problem) is expressed in terms of the Stokes parameters as

$$S_0 = |E^{\text{f}}_{0+}|^2 + |E^{\text{f}}_{0-}|^2, \qquad S_1 = 2\,\text{Re}[E^{\text{f}*}_{0+} E^{\text{f}}_{0-}],$$
$$S_3 = |E^{\text{f}}_{0+}|^2 - |E^{\text{f}}_{0-}|^2, \qquad S_2 = 2\,\text{Im}[E^{\text{f}*}_{0+} E^{\text{f}}_{0-}].$$

However, the direct interpretation of these quantities in terms of squared Volts per square metres is sometimes somewhat inconvenient; therefore, those parameters can be scaled to give an interpretation of the intensity (in regular SI units measured in Watts per square metres), as $S'_k = (\varepsilon_0 c/2) S_k$, or explicitly

$$S'_0 = (\varepsilon_0 c/2)[|E^{\text{f}}_{0+}|^2 + |E^{\text{f}}_{0-}|^2], \qquad S'_1 = (\varepsilon_0 c/2) 2\,\text{Re}[E^{\text{f}*}_{0+} E^{\text{f}}_{0-}],$$
$$S'_3 = (\varepsilon_0 c/2)[|E^{\text{f}}_{0+}|^2 - |E^{\text{f}}_{0-}|^2], \qquad S'_2 = (\varepsilon_0 c/2) 2\,\text{Im}[E^{\text{f}*}_{0+} E^{\text{f}}_{0-}].$$

In this representation, $S'_0$ is now identical to the incident intensity $I_{\text{in}}$ $[\text{W}/\text{m}^2]$. In order to have those scaled Stokes parameters $S'_k$ written to file, rather than the default ones, one convenient possibility is to use the previously added `--scale_stokesparams` option, to include `--scale_stokesparams 1.327209e-3` at the command line when invoking the program. The numerical value of this scaling was obtained from

$$\begin{aligned}\varepsilon_0 c/2 &= (8.854187817\ldots \times 10^{-12}\ \text{F/m}) \times (2.99792458 \times 10^{8}\ \text{m/s})/2 \\ &\approx 1.327209 \times 10^{-3}\ \text{F/s}.\end{aligned}$$

In regular SI units as here used, the physical dimension of the quantity $\varepsilon_0 c/2$ is $[(\text{A}\cdot\text{s})/(\text{V}\cdot\text{m})]\cdot[\text{m/s}] = [\text{A/V}]$, so the physical dimension of $(\varepsilon_0 c/2) S_k$ is hence $[\text{A/V}] \cdot [\text{V}^2/\text{m}^2] = [\text{W}/\text{m}^2]$, as expected for an intensity measure (energy flow per unit area in the plane orthogonal to the direction of wave propagation). For any other scaling factor of the Stokes parameters, use the general syntax

`--scale_stokesparams` $\langle a\rangle$

which forces the program to save the scaled Stokes parameters $S'_k = a S_k$ to file instead of the original $S_k$, $k = 0, 1, 2, 3$.

**41.** Saving intra-grating spatial evolution of optical field and intensity. The optical intensity and/or general spatial evolution of the electromagnetic field inside the grating can be saved to file using the command line option of syntax

`-M` $\langle M\rangle$

The data for the field evolution is written to file in the format

$$\begin{array}{lllll} z_1 & \langle E^{\rm f}_+(z_1)\rangle & \langle E^{\rm f}_-(z_1)\rangle & \langle E^{\rm b}_+(z_1)\rangle & \langle E^{\rm b}_-(z_1)\rangle \\ z_2 & \langle E^{\rm f}_+(z_2)\rangle & \langle E^{\rm f}_-(z_2)\rangle & \langle E^{\rm b}_+(z_2)\rangle & \langle E^{\rm b}_-(z_2)\rangle \\ \vdots & & \vdots & & \vdots \\ z_N & \langle E^{\rm f}_+(z_N)\rangle & \langle E^{\rm f}_-(z_N)\rangle & \langle E^{\rm b}_+(z_N)\rangle & \langle E^{\rm b}_-(z_N)\rangle \end{array}$$

where each blank space-separated entry $\langle E^{\rm f}_+(z_k)\rangle$, $\langle E^{\rm f}_-(z_k)\rangle$, $\langle E^{\rm b}_+(z_k)\rangle$, or $\langle E^{\rm b}_-(z_k)\rangle$ for an electric field component constitutes the pair of its real and imaginary parts,

$$\langle E^{\rm f}_+(z_k)\rangle \equiv \langle {\rm Re}[E^{\rm f}_+(z_k)]\rangle \quad \langle {\rm Im}[E^{\rm f}_+(z_k)]\rangle,$$

where each number $\langle {\rm Re}[E^{\rm f}_+(z_k)]\rangle$ or $\langle {\rm Im}[E^{\rm f}_+(z_k)]\rangle$ in the pair in the saved file is represented in plain ASCII by 16 significant digits.

[TEXT STILL TO BE WRITTEN]

**42.** Saving the spatial profile of grating structure. The spatial distribution of the refractive index, gyration coefficient, and nonlinear optical and magneto-optical parameters can in the progress of the simulation be written to file using the command line option

`--writegratingfile` ⟨filename⟩

[TEXT STILL TO BE WRITTEN]

**43.** Intragrating intensity information. In many cases the density distribution of optical intensity inside the medium is of particular interest, since cavity-effects or other resonant phenomena can considerably increase the intensity locally along the grating. The program can automatically track down the maximum present intensity in the grating and the corresponding location, presenting the information either at the terminal window in which the program was started or written to file. The syntax for getting this information calculated and displayed in the terminal window is simply

`--intensityinfo`.

To get the information instead written to file, use

`--intensityinfologfile` ⟨filename⟩,

where ⟨filename⟩ is the name of the destination file.

**44. Postprocessing of the data to get the direct relation.** The data generated by the program naturally relate the inverse relation between the incident and transmitted (or reflected) optical fields. In other words, the input data are typically equidistantly spaced in the transmitted intensity and/or ellipticity, while the spacing of the calculated data (the incident optical field) is ....

For topological mappings of the data, such as in Fig. X [FIGURE TO BE INSERTED], it is for most cases of no importance whether it is the direct relation (transmitted or reflected field as function of incident field) or its inverse (incident or reflected field as function of transmitted field) that is described by the set of data. However, in many cases we are interested in a maping that corresponds to a direct experimental setup. For example, whenever we have a linearly polarized incident light in the experimental setup of evaluation, the transmitted polarization state generally differ from the input one, and we must take into account that the input data will ....

[TEXT STILL TO BE WRITTEN]

**45. The main program.** Here follows the general outline of the main program.

```
⟨ Library inclusions 46 ⟩
⟨ Global definitions 47 ⟩
⟨ Data structure definitions 48 ⟩
⟨ Global variables 49 ⟩
⟨ Subroutines 50 ⟩
int main(int argc, char *argv[])
{
  ⟨ Declaration of local variables 51 ⟩
  ⟨ Initialize variables 61 ⟩
  ⟨ Parse command line for parameter values 116 ⟩
  ⟨ Check for specified trajectory of transmitted Stokes parameters 134 ⟩
  ⟨ Open files for output 135 ⟩
  ⟨ Allocate optical field vectors 62 ⟩
  ⟨ Initiate grating structure 64 ⟩
  ⟨ Initiate surrounding medium 70 ⟩
  ⟨ Calculate intragrating layer reflectances 71 ⟩
  ⟨ Calculate incident optical field spectrum 72 ⟩
  ⟨ Print information on maximum optical intensity in grating 87 ⟩
  ⟨ Deallocate optical field vectors 63 ⟩
  ⟨ Close output files 136 ⟩
  return (SUCCESS);
}
```

**46.** The standard ANSI C libraries included in this program are:

| | |
|---|---|
| `math.h` | For access to mathematical functions. |
| `time.h` | For time stamps and estimation of computation time. |
| `stdio.h` | For file access and any block involving *fprintf*. |
| `stdlib.h` | For memory allocation, *malloc*, *exit*, *free* etc. |
| `string.h` | For string manipulation, *strcpy*, *strcmp* etc. |
| `ctype.h` | For access to the *isalnum* routine. |

```
⟨ Library inclusions 46 ⟩ ≡
#include <math.h>
#include <time.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
```

This code is used in section 45.

**47.** Global definitions. There are just a few global definitions present in the MAGBRAGG program:

| | |
|---|---|
| `VERSION` | The current program revision number. |
| `COPYRIGHT` | The copyright banner. |
| `SUCCESS` | The return code for successful program termination. |
| `FAILURE` | The return code for unsuccessful program termination. |
| `NCHMAX` | The maximum number of characters allowed in strings for storing file names, including path. |

⟨ Global definitions 47 ⟩ ≡
```
#define VERSION "1.43"
#define COPYRIGHT "Copyright␣(C)␣2002-2007,␣Fredrik␣Jonsson"
#define SUCCESS (0)
#define FAILURE (1)
#define NCHMAX (256)
```
This code is used in section 45.

**48.** Data structure definitions. The *dcomplex* data structure contains real and imaginary parts in double precision, and is also the basic data structure used for the allocation of complex valued vectors of double precision.

⟨ Data structure definitions 48 ⟩ ≡
```
  typedef struct DCOMPLEX {
    double r, i;
  } dcomplex;
```
This code is used in section 45.

**49.** Declaration of global variables. The only global variables allowed in my programs are *optarg*, which is a pointer to the the string of characters that specified the call from the command line, and *progname*, which simply is a pointer to the string containing the name of the program, as it was invoked from the command line.

⟨ Global variables 49 ⟩ ≡
```
  extern char *optarg;
  char *progname;
```
This code is used in section 45.

**50.** Listing of subroutines called by the main program.

⟨ Subroutines 50 ⟩ ≡
```
  ⟨ Routine for checking for numerical characters 88 ⟩
  ⟨ Routine for initialization of Cantor type fractal gratings 89 ⟩
  ⟨ Routines for removing preceding path of filenames 90 ⟩
  ⟨ Routines for memory allocation of vectors 111 ⟩
  ⟨ Routines for generation of random numbers 93 ⟩
  ⟨ Routines for complex arithmetics 94 ⟩
  ⟨ Routines for displaying help message 130 ⟩
```
This code is used in section 45.

**51. Declaration of local variables of the main program.** In CWEB one has the option of adding variables along the program, for example by locally adding temporary variables related to a given sub-block of code. However, my philosophy in the MAGBRAGG program is to keep all variables of the *main* section collected together, so as to simplify tasks as, for example, tracking down a given variable type definition.

[ADD A LIST OF VARIABLE DESCRIPTIONS WITH THE 'VARITEM' MACRO]

| | |
|---|---|
| *somevariable* | Some description of *somevariable*. |

⟨ Declaration of local variables 51 ⟩ ≡
  ⟨ Physical and mathematical constants 52 ⟩
  ⟨ Time variables 53 ⟩
  ⟨ Declaration of complex arrays storing the electrical field distribution 54 ⟩
  ⟨ Declaration of Boolean variables for execution control 55 ⟩
  ⟨ Discretization parameters 56 ⟩
  ⟨ Grating parameters 57 ⟩
  ⟨ Declaration of file pointers 58 ⟩
  ⟨ Declaration of strings and file names 59 ⟩
  ⟨ Declaration of local dummy variables 60 ⟩

```
long nne, jje, mmtraject;
long ranseed;    /* seed for random number generator */
long maxintens_layer = 0;
int fractal_level = 0;    /* The recursion level in construction of Cantor-type fractal gratings */
int maximum_fractal_level = 0;
  /* The maximum allowed recursion level in construction of Cantor-type fractal gratings */
double *w0traj = Λ, *w3traj = Λ;
double lambdastart;    /* The start vacuum wavelength λ_start of the spectrum to be sampled */
double lambdastop;    /* The stop vacuum wavelength λ_stop of the spectrum to be sampled */
double lambda;    /* Dummy variable to hold the vacuum wavelength at a discrete spectral sample */
double omega;    /* Ditto for the angular frequency ω ≡ 2π/λ */
double ievolambda;
double apolength;    /* Length L_apod over which each end of the grating should be apodized */
double phasejumpangle;    /* The magnitude of the phase discontinuity φ_jump measured in radians */
double phasejumpposition;    /* The position z_jump in metres of the phase discontinuity φ_jump */
double phi;    /* Dummy variable to hold the reference phase over the spatial extent of the grating */
double gyroperturb_position;    /* The position z_p in metres of the peak of the added Lorentzian
    perturbation Δg(z) of the gyration coefficient g(z) */
double gyroperturb_amplitude;
  /* The zero-to-peak amplitude a_p of the added Lorentzian perturbation Δg(z) of the gyration
    coefficient g(z) */
double gyroperturb_width;    /* The full width at half maximum w_p of the added Lorentzian
    perturbation Δg(z) of the gyration coefficient g(z) */
double nsurr;    /* The linear index of refraction of the medium surrounding the grating */
double n1, n2, t1, t2, nper, ncrp, g1, g2, gper, gcrp, pe1, pe2, peper, pecrp, pm1, pm2, pmper,
    pmcrp, qe1, qe2, qeper, qecrp, qm1, qm2, qmper, qmcrp;
double modn1, modt1, modg1, modpe1, modpm1, modqe1, modqm1, aafp2, aafm2, aabp2, aabm2;
double trmintensity, trmintenstart, trmintenstop, trmellipticity, trmellipstart, trmellipstop,
    stoke_scalefactor;
static double nndef = 600;    /* Number of samples in spectrum */
static double mmdef = 1000;    /* Number of samples in grating */
static double mmedef = 1;    /* Number of samples in ellipticity */
static double mmidef = 1;    /* Number of samples in intensity */
static double lldef = 0.050;    /* Length of grating in metres */
static double nsurrdef = 1.0;    /* Default surrounding refractive index */
```

**static double** $\mathit{lambdastartdef} = 1536.0 \cdot 10^{-9}$, $\mathit{lambdastopdef} = 1546.0 \cdot 10^{-9}$;

This code is used in section 45.

**52.** Declaration of numerical values of physical constants. All calculations performed by this program are done in SI units, in which the fundamental physical constants take the values below.

⟨ Physical and mathematical constants 52 ⟩ ≡
**static double** *pi* = 3.1415926535897932384626433832795; /* $\pi$ */
**static double** *twopi* = 2.0 ∗ 3.1415926535897932384626433832795; /* $2\pi$ */
**static double** $c = 2.997925 \cdot 10^8$; /* Speed of light in vacuum, $c$ [m/s] */
**static double** $\mathit{epsilon0} = 8.854 \cdot 10^{-12}$; /* Permittivity of vacuum, $\varepsilon_0$ [As/Vm] */

This code is used in section 51.

**53.** Declaration of time variables. Here any variables related to timing and estimation of computing performance are declared. The significance of the variables are as follows:

| | |
|---|---|
| *initime* | The time at which the MAGBRAGG program is initialized. This variable is initialized already in its declaration. |
| *now* | Dummy variable for extraction of current time from the system. |
| *eta* | Estimated time of arrival (ETA, not to be confused with the Greek letter $\eta$) of successful termination of the MAGBRAGG program. |

⟨ Time variables 53 ⟩ ≡
**time_t** *initime* = *time*(Λ), *now* = *time*(Λ), *eta* = *time*(Λ);

This code is used in section 51.

**54.** Declaration of pointers to arrays of fields of **dcomplex** class, holding the intra-grating electrical field distribution. These pointers will after the memory allocation for complex-valued vectors hold the spatial distribution of the electric field inside the grating. The significance of the variables are as follows:

| | |
|---|---|
| $\mathit{efp}[k] \equiv E^{\mathrm{f}}_{+}(z^{+}_{k})$, | The left circularly polarized forward traveling field component at $z = z_k$ taken in the $k$th homogeneous layer of the discretized grating. |
| $\mathit{efm}[k] \equiv E^{\mathrm{f}}_{-}(z^{+}_{k})$, | The right circularly polarized forward traveling field component at $z = z_k$ taken in the $k$th homogeneous layer of the discretized grating. |
| $\mathit{ebp}[k] \equiv E^{\mathrm{b}}_{+}(z^{+}_{k})$, | The left circularly polarized backward traveling field component at $z = z_k$ taken in the $k$th homogeneous layer of the discretized grating. |
| $\mathit{ebm}[k] \equiv E^{\mathrm{b}}_{-}(z^{+}_{k})$, | The right circularly polarized backward traveling field component at $z = z_k$ taken in the $k$th homogeneous layer of the discretized grating. |

that is to say, in a loss-less medium in which the nonlinear correction to the linear refractive index is determined by the magnitude of the circularly polarized components of the field envelopes, these arrays hold the full complex spatial evolution of the electric field.

⟨ Declaration of complex arrays storing the electrical field distribution 54 ⟩ ≡
**dcomplex** ∗*efp*, ∗*efm*, ∗*ebp*, ∗*ebm*;

This code is used in section 51.

**55.** Declaration of Boolean variables. The Boolean variables are used as flags that internally are set and recognized by the MAGBRAGG program, for example to determine states or modes of operation as specified by the user via command line options. In the MAGBRAGG program, we use integer variables of **short** precision for storing Boolean values, with the value 0 corresponding to the "false" state and 1 corresponding to the "true" state.

```
⟨ Declaration of Boolean variables for execution control 55 ⟩ ≡
  short verbose;   /* If nonzero, display information at terminal output during program execution */
  short randomdistribution;   /* If nonzero, use random layer thicknesses of a binary-type grating */
  short writegratingtofile;   /* If nonzero, save the grating structure to file */
  short scale_stokesparams;   /* If nonzero, then scale Stokes parameters by stoke_scalefactor */
  short normalize_length_to_micrometer;   /* If nonzero, */
  short normalize_intensity;   /* If nonzero, */
  short normalize_ellipticity;   /* If nonzero, */
  short normalize_internally;   /* If nonzero, */
  short odd_layer;   /* Keeps track of odd and even grating layers in initialization */
  short chirpflag;   /* If nonzero, then apply chirp to periodicity of sinusoidal grating structures */
  short apodize;   /* If nonzero, apply apodization to the ends of the grating */
  short phasejump;
    /* If nonzero, apply at least one discrete phase discontinuity of the grating profile */
  short fieldevoflag;   /* If nonzero, then save spatial field distribution to file */
  short fieldevoflag_efield;   /* If nonzero while fieldevoflag is nonzero, then save the complex electric
        field as the spatial field distribution */
  short intensityevoflag;   /* If nonzero while fieldevoflag is nonzero, then save the field intensity as the
        spatial field distribution */
  short fieldevoflag_stoke;   /* If nonzero while fieldevoflag is nonzero, then save the Stokes parameters
        as the spatial field distribution */
  short intensityinfo;   /* If nonzero, then display the maximum and minimum field intensities found
        within the grating at terminal output before closing the program */
  short saveintensityinfologfile;
    /* If nonzero, then also save the displayed maximum and minimum intensities to a log file */
  short trmtraject_specified;   /* If nonzero, */
  short save_dbspectra;
    /* If nonzero, then save any transmission spectrum in logarithmic (dB) scale */
  short stokes_parameter_spectrum;   /* If nonzero, then save any transmission spectrum in terms of the
        corresponding Stokes parameters */
  short display_surrounding_media;   /* If nonzero, then append also the surrounding medium when
        saving the spatial grating profile to file */
  short perturbed_gyration_constant;   /* If nonzero, */
```

This code is used in section 51.

**56.** Declaration of discretization parameters. These parameters determine the discretization of the problem at hand, such as the number of points in intensity, ellipticity or wavelength to be scanned. The significance of the variables are as follows:

| | |
|---|---|
| *mm* | The number of points $M$ to be sampled in the spectrum between vacuum wavelengths $\lambda_{\rm start}$ and $\lambda_{\rm stop}$. |
| *mme* | The number $M_{\rm e}$ of ellipticities $\varepsilon_{\rm tr}$ of the transmitted light to be sampled in the interval $\varepsilon_{\rm tr,start} \le \varepsilon \le \varepsilon_{\rm tr,stop}$. |
| *mmi* | The number $M_{\rm i}$ of intensities $I_{\rm tr}$ of the transmitted light to be sampled in the interval $I_{\rm tr,start} \le I_{\rm tr} \le I_{\rm tr,stop}$. |

⟨ Discretization parameters 56 ⟩ ≡
  **long** *mm*, *mme*, *mmi*;

This code is used in section 51.

**57.** Declaration of grating parameters. These parameters fully describe the geometry and material properties of the medium in which the wave propagation is to be performed. The significance of the variables are as follows:

| | |
|---|---|
| *ll* | Geometrical length $L$ of the grating measured in metres. |
| *nn* | The number $N$ of discrete interfaces separating the $N-1$ homogeneous layers of the grating and the surrounding medium. |
| *modnum* | If positive, this is the number of a manually modified layer, for instance a defect or cavity layer possessing different geometrical or optical properties than the rest of the structure described by any of the standard grating options. |
| $*z$ | Pointer to an array of $N$ elements for storing the coordinates of the discrete interfaces $z_1, \ldots, z_N$ between homogeneous layers of the grating. |
| $*dz$ | Pointer to an array of $N-1$ elements containing the layer thicknesses $dz[k] \equiv z_{k+1} - z_k$, $k = 1, \ldots, N-1$. |
| $*n$, $*g$ | Pointers to arrays of $N-1$ elements containing the refractive indices $n[k] = n_k$ and magneto-optical gyration coefficients $g[k] = g_k = i\chi^{(\text{eem})}_{xyz} B^z_0/(2n_k)$ of the homogeneous layers. |
| $*pe$, $*qe$ | Pointers to arrays of $N-1$ elements containing the nonlinear optical coefficients $pe[k] = \chi^{\text{eeee}}_{xxxx} - \chi^{\text{eeee}}_{xyyx}$ and $qe[k] = \chi^{\text{eeee}}_{xxxx} + \chi^{\text{eeee}}_{xyyx}$ of the homogeneous layers. |
| $*pm$, $*qm$ | Pointers to arrays of $N-1$ elements containing the nonlinear magneto-optical coefficients $pm[k] = i(\chi^{\text{eeeem}}_{xyyyz} - \chi^{\text{eeeem}}_{xxxyz})B^z_0$ and $qm[k] = i(\chi^{\text{eeeem}}_{xyyyz} + \chi^{\text{eeeem}}_{xxxyz})B^z_0$ of the layers. |
| $*etafp$, $*etafm$ | Pointers to arrays of $N-1$ elements for storing the nonlinear coefficients of propagation $\eta^{\text{f}}_{+}$ related to the phase evolution of forward (f) traveling left (+) and right (−) circularly polarized components of in respective layer $z_j < z < z_{j+1}$, with $etafp[j] = \eta^{\text{f}}_{+}(z^{+}_{j})$ and $etafm[j] = \eta^{\text{f}}_{-}(z^{+}_{j})$. |
| $*etabp$, $*etabm$ | Analogous to $*etafp$ and $*etafm$ but instead for the nonlinear phase contributions $etabp[j] = \eta^{\text{b}}_{+}(z^{+}_{j})$ and $etabm[j] = \eta^{\text{b}}_{-}(z^{+}_{j})$ for backward traveling field components. |
| $*taup$, $*taum$ | Pointers to arrays of $N$ elements for storing the forward intra-grating layer transmittances $\tau_{k_+}$ and $\tau_{k_-}$. |
| $*taupp$, $*taupm$ | Pointers to arrays of $N$ elements for storing the backward intra-grating layer transmittances $\tau'_{k_+}$ and $\tau'_{k_-}$. |
| $*rhop$, $*rhom$ | Pointers to arrays of $N$ elements for storing the forward intra-grating layer reflectances $\rho_{k_+}$ and $\rho_{k_-}$. |
| $*rhopp$, $*rhopm$ | Pointers to arrays of $N$ elements for storing the backward intra-grating layer reflectances $\rho'_{k_+}$ and $\rho'_{k_-}$. |

For the definitions of the intra-grating layer reflectances and transmittances, see the separate section *Calculation of intra-grating layer reflectances.*

⟨ Grating parameters 57 ⟩ ≡

```
long nn, modnum;
double ll, *z, *dz, *n, *g, *pe, *pm, *qe, *qm, *etafp, *etafm, *etabp, *etabm;
double *taup, *taum, *taupp, *taupm, *rhop, *rhom, *rhopp, *rhopm;
```

This code is used in section 51.

**58.** Declaration of file pointers.

```
⟨ Declaration of file pointers 58 ⟩ ≡
  FILE *fp_s0, *fp_s1, *fp_s2, *fp_s3;    /* Stokes parameters of incident wave */
  FILE *fp_v0, *fp_v1, *fp_v2, *fp_v3;    /* Stokes parameters of reflected wave */
  FILE *fp_w0, *fp_w1, *fp_w2, *fp_w3;    /* Stokes params. of transmitted wave */
  FILE *fp_evo = Λ, *fp_ievo = Λ, *fp_spec = Λ, *fp_traject = Λ;
  FILE *fp_irspec = Λ, *fp_itspec = Λ, *fp_icspec = Λ, *fp_gr = Λ;
  FILE *fp_evo_s0 = Λ, *fp_evo_s1 = Λ, *fp_evo_s2 = Λ, *fp_evo_s3 = Λ;
  FILE *intensinfologfile = Λ;
```

This code is used in section 51.

**59.** Declaration of character strings holding file names. Generally in this program, the maximum number of characters a file name string can contain is NCHMAX, as defined in the definitions section of the program.

```
⟨ Declaration of strings and file names 59 ⟩ ≡
  char gratingtype[NCHMAX], gratingsubtype[NCHMAX];
  char fieldevofilename[NCHMAX], intensityevofilename[NCHMAX];
  char spectrumfilename[NCHMAX], trmtraject_filename[NCHMAX];
  char intensity_reflection_spectrumfilename[NCHMAX], intensity_transmission_spectrumfilename[NCHMAX],
      intensity_check_spectrumfilename[NCHMAX];
  char fieldevofilename_s0[NCHMAX], fieldevofilename_s1[NCHMAX], fieldevofilename_s2[NCHMAX],
      fieldevofilename_s3[NCHMAX];
  char intensinfologfilename[NCHMAX];
  char outfilename[NCHMAX], gratingfilename[NCHMAX];
  char outfilename_s0[NCHMAX], outfilename_s1[NCHMAX], outfilename_s2[NCHMAX], outfilename_s3[NCHMAX];
  char outfilename_v0[NCHMAX], outfilename_v1[NCHMAX], outfilename_v2[NCHMAX], outfilename_v3[NCHMAX];
  char outfilename_w0[NCHMAX], outfilename_w1[NCHMAX], outfilename_w2[NCHMAX], outfilename_w3[NCHMAX];
```

This code is used in section 51.

**60.** Declaration of dummy variables. Here dummy and temporary variables are declared, where the "dummy" variables typically are integer counters for iteration over wavelength, ellipticity, intensity and so on, while the temporary variables are internally used for in some cases saving computational time, and also for the sake of increasing the readability of the program somewhat.

```
⟨ Declaration of local dummy variables 60 ⟩ ≡
  dcomplex tmpfp, tmpfm, tmpbp, tmpbm;
  double tmp, zt, s0, s1, s2, s3, v0, v1, v2, v3, w0, w1, w2, w3, stn, maxintens = 0.0,
      maxintens_inintens = 0.0, maxintens_inellip = 0.0, maxintens_trintens = 0.0, maxintens_trellip = 0.0;
  int no_arg, status = 0, tmpch;
  long j, k, ke, ki;
```

This code is used in section 51.

**61. Initialization of variables.**

⟨ Initialize variables 61 ⟩ ≡
```
  trmtraject_specified = 0;   /* Is a trajectory of (W_0, W_3) specified? */
  intensityinfo = 0;   /* Printing basic intensity info to stdout; default: off */
  saveintensityinfologfile = 0;   /* Printing basic intensity info to file */
  randomdistribution = 0;   /* Random shuffling of layers; default: off */
  writegratingtofile = 0;   /* Save spatial grating structure; default: off */
  scale_stokesparams = 0;   /* Scale Stokes parameters by stoke_scalefactor; default: off */
  stoke_scalefactor = 1.0;   /* Scale factor to be used for Stokes parameters */
  normalize_length_to_micrometer = 0;
  normalize_intensity = 0;   /* Write normalized intensity to file; default: off */
  normalize_ellipticity = 0;   /* Ditto for the ellipticity */
  ranseed = 1097;   /* Seed for random number generator */
  modnum = −1;   /* If positive, the manually modified layer number */
  verbose = 0;   /* Verbose mode is off by default */
  apodize = 0;   /* Apodization is off by default */
  phasejump = 0;   /* Discrete phase jump is off by default */
  normalize_internally = 0;   /* Default state of internal normalization is off */
  odd_layer = 0;   /* Counter flag for odd and even grating layers */
  chirpflag = 1;
  save_dbspectra = 0;
  stokes_parameter_spectrum = 0;
  display_surrounding_media = 1;
    /* Append also the surrounding medium when saving the spatial grating profile to file; default: on */
  perturbed_gyration_constant = 0;
  fieldevoflag = 0;   /* Save spatial field distribution to file; default: off */
  fieldevoflag_efield = 0;   /* Save as complex electric field; default: off */
  fieldevoflag_stoke = 0;   /* Save as Stokes parameters; default: off */
  intensityevoflag = 0;   /* Save spatial intensity distribution; default: off */
  mm = mmdef;   /* Number of sampling points in optical spectrum */
  nn = nndef;   /* Number of layer interfaces of grating; num of layers is nn-1 */
  mme = mmedef;   /* Number of sampling points in transmitted ellipticity */
  mmi = mmidef;   /* Number of sampling points in transmitted intensity */
  nne = 1;   /* Default number extra, intra-layer sampling points */
  ll = lldef;   /* Default physical length of the grating structure */
  lambdastart = lambdastartdef;
  lambdastop = lambdastopdef;
  phasejumpangle = 0.0;   /* Discrete phase jump is off by default */
  phasejumpposition = 0.0;   /* Discrete phase jump is off by default */
  phi = 0.0;   /* Discrete phase jump is off by default */
  nsurr = nsurrdef;   /* Default index of refraction of the surrounding medium */
  strcpy(outfilename, "out.stok");   /* Default output file basename */
  strcpy(fieldevofilename, "out.fevo.dat");   /* Default output file basename */
  strcpy(fieldevofilename_s0, "out.fevo.s0.dat");
  strcpy(fieldevofilename_s1, "out.fevo.s1.dat");
  strcpy(fieldevofilename_s2, "out.fevo.s2.dat");
  strcpy(fieldevofilename_s3, "out.fevo.s3.dat");
  strcpy(spectrumfilename, "out.rsp.dat");   /* Default output file name */
  strcpy(intensity_reflection_spectrumfilename, "out.irsp.dat");
  strcpy(intensity_transmission_spectrumfilename, "out.trsp.dat");
  strcpy(intensity_check_spectrumfilename, "out.chec.dat");
  fp_s0 = Λ;
  fp_s1 = Λ;
  fp_s2 = Λ;
  fp_s3 = Λ;
  fp_v0 = Λ;
  fp_v1 = Λ;
  fp_v2 = Λ;
  fp_v3 = Λ;
  fp_w0 = Λ;
  fp_w1 = Λ;
  fp_w2 = Λ;
  fp_w3 = Λ;
```

This code is used in section 45.

**62. Memory allocation.** Allocate memory for the temporary storage of the spatial distribution of material parameters of the grating structure and internal, complex-valued electromagnetic fields. In TEX-style notation, the parameters are interpreted in terms of the optical and magneto-optical susceptibilities [Fredrik Jonsson, *The Nonlinear Optics of Magneto-Optic Media*, PhD thesis (The Royal Institute of Technology, Stockholm, 2000)] as follows, starting with the all-optical (electric-dipolar related) parameters

$$\begin{aligned} n[k] &= n_k = [1+\chi^{(\mathrm{ee})}_{xx}]^{1/2},\\ pe[k] &= \chi^{\mathrm{eeee}}_{xxxx} - \chi^{\mathrm{eeee}}_{xyyx},\\ qe[k] &= \chi^{\mathrm{eeee}}_{xxxx} + \chi^{\mathrm{eeee}}_{xyyx}, \end{aligned}$$

where all susceptibilities of the right-hand sides are to be evaluated in the domains $z_k < z < z_{k+1}$, $k = 1, 2, \ldots N-1$. In addition to the all-optical parameters, the magneto-optical (magnetic-dipolar related) parameters are defined as

$$\begin{aligned} g[k] &= g_k = i\chi^{(\mathrm{eem})}_{xyz} B^z_0/(2n_k),\\ pm[k] &= i(\chi^{\mathrm{eeeem}}_{xyyyz} - \chi^{\mathrm{eeeem}}_{xxxyz})B^z_0,\\ qm[k] &= i(\chi^{\mathrm{eeeem}}_{xyyyz} + \chi^{\mathrm{eeeem}}_{xxxyz})B^z_0, \end{aligned}$$

using the same conventions of evaluation for the involved susceptibilities in the righ-hand sides. With these definitions, the left circularly polarized (LCP) and right circularly polarized (RCP) modes that propagate in the forward direction in the $k$th layer will experience the linear, field-independent refractive indices $n_{k_+}$ and $n_{k_-}$, respectively, with

$$n_{k_+} = n_k[1 + i\chi^{(\mathrm{eem})}_{xyz} B^z_0/(2n^2_k)] = n_k + g_k, \qquad \text{(LCP)}$$

$$n_{k_-} = n_k[1 - i\chi^{(\mathrm{eem})}_{xyz} B^z_0/(2n^2_k)] = n_k - g_k. \qquad \text{(RCP)}$$

For backward propagating light, the experienced indices of refraction are reversed, that is to say, backward propagating left circularly polarized light experience $n_-$ as the refractive index, while the right circularly polarized experience $n_+$.

We here throughout adopt to common standard definition of circularly polarized light, that when looking into an oncoming wave propagating in the positive $z$-direction, the vector of the LCP electric field component $E^{\mathrm{f}}_+$ of the wave describes counterclockwise motion, while it for the RCP field component $E^{\mathrm{f}}_-$ instead describes clockwise motion. This convention conforms to the classical one as used in optics [J. D. Jackson, Classical Electrodynamics (Wiley, New York, 1975); M. Born and E. Wolf, Principles of Optics (Cambridge University Press, Cambridge, 1980)]. Similarly, by looking into an oncoming wave propagating in the negative $z$-direction, the vector of the LCP electric field component $E^{\mathrm{b}}_+$ describes counterclockwise motion, while it for the RCP field component $E^{\mathrm{b}}_-$ describes clockwise motion.

In the following allocation of memory, the $k$th element of respective vector contains the numerical value for the respective material parameter evaluated in the domain $z_k < z < z_{k+1}$, for $k = 1, 2, \ldots, N-1$, while the vectors containing the electrical field components contain the respective component evaluated at the "beginning" of respective layer, at $z = z^+_k$. From material parameters such as the linear refractive indices $n_k$ and the magneto-optical gyration constants $g_k$ of the compound structure, derived numerical quantities, such as the transmission coefficients $\tau_{k_\pm}$ and $\tau'_{k_\pm}$ and the reflection coefficients $\rho_{k_\pm}$ and $\rho'_{k_\pm}$ across the interfaces at $z_k$, are later on calculated and stored in vectors that here are allocated.

```
⟨ Allocate optical field vectors 62 ⟩ ≡
  {
    z = dvector(1, nn);    /* Spatial coordinate z_k along the Bragg grating */
    dz = dvector(1, nn − 1);    /* dz[k] ≡ z[k + 1] − z[k], k = 1, 2, ..., nn − 1 */
    efp = dcvector(0, nn);    /* Equals E^f_{k_+} in TEX-style notation */
    efm = dcvector(0, nn);    /* Equals E^f_{k_-} in TEX-style notation */
    ebp = dcvector(0, nn);    /* Equals E^b_{k_+} in TEX-style notation */
    ebm = dcvector(0, nn);   /* Equals $E^b_{k_-}$ in TEX-style notation */
    taup = dvector(1, nn);   /* Forward LCP transmission coefficient $\tau_{k_+}$ */
    taum = dvector(1, nn);   /* Forward RCP transmission coefficient $\tau_{k_+}$ */
    taupp = dvector(1, nn);   /* Backward LCP transmission coeff. $\tau'_{k_+}$ */
    taupm = dvector(1, nn);   /* Backward RCP transmission coeff. $\tau'_{k_-}$ */
    rhop = dvector(1, nn);   /* Forward LCP reflection coeff. $\rho_{k_+}$ */
    rhom = dvector(1, nn);   /* Forward RCP reflection coeff. $\rho_{k_-}$ */
    rhopp = dvector(1, nn);   /* Backward LCP reflection coeff. $\rho'_{k_+}$ */
    rhopm = dvector(1, nn);   /* Backward RCP reflection coeff. $\rho'_{k_-}$ */
    n = dvector(0, nn);   /* Linear electric-dipolar refractive index $n(z)$ */
    g = dvector(0, nn);   /* Linear magneto-optical contribution to $n(z)$ */
    pe = dvector(1, nn - 1);   /* Nonlinear all-optical contribution to $n(z)$ */
    pm = dvector(1, nn - 1);   /* Nonlinear magneto-optical contribution to $n(z)$ */
    qe = dvector(1, nn - 1);   /* Nonlinear all-optical contribution to $n(z)$ */
    qm = dvector(1, nn - 1);   /* Nonlinear magneto-optical contribution to $n(z)$ */
    etafp = dvector(1, nn - 1);   /* Nonlinear LCP forward contribution to $n(z)$ */
    etabp = dvector(1, nn - 1);   /* Nonlinear LCP backward contribution to $n(z)$ */
    etafm = dvector(1, nn - 1);   /* Nonlinear RCP forward contribution to $n(z)$ */
    etabm = dvector(1, nn - 1);   /* Nonlinear RCP backward contribution to $n(z)$ */
  }
```

This code is used in section 45.

**63.** At the end of the program, we would also like to properly deallocate the memory occupied by the previously allocated real- and complex-valued vectors that have been used in the simulation. This is done by executing the following block.

```
⟨ Deallocate optical field vectors 63 ⟩ ≡
  {
    free_dcvector(efp, 0, nn);
    free_dcvector(efm, 0, nn);
    free_dcvector(ebp, 0, nn);
    free_dcvector(ebm, 0, nn);
    free_dvector(taup, 1, nn);
    free_dvector(taum, 1, nn);
    free_dvector(taupp, 1, nn);
    free_dvector(taupm, 1, nn);
    free_dvector(rhop, 1, nn);
    free_dvector(rhom, 1, nn);
    free_dvector(rhopp, 1, nn);
    free_dvector(rhopm, 1, nn);
    free_dvector(n, 0, nn);
    free_dvector(g, 0, nn);
    free_dvector(pe, 1, nn − 1);
    free_dvector(pm, 1, nn − 1);
    free_dvector(qe, 1, nn − 1);
    free_dvector(qm, 1, nn − 1);
    free_dvector(etafp, 1, nn − 1);
    free_dvector(etabp, 1, nn − 1);
    free_dvector(etafm, 1, nn − 1);
    free_dvector(etabm, 1, nn − 1);
    if (trmtraject_specified) {
      free_dvector(w0traj, 1, mmtraject);
      free_dvector(w3traj, 1, mmtraject);
    }
  }
```

This code is used in section 45.

**64. Initialization of the grating structure.** Using the previously allocated memory for the temporary storage of the spatially distributed grating structure, initiate the material parameters of the interior of the grating. If the user via the command line options has specified that a perturbation of the gyration constant should be present somewhere along the grating, then add a perturbation in the functional form of a magneto-optical effect that would arise due to a current carrying wire, placed orthogonal to the direction of propagation, and in a close vicinity of the medium.

```
⟨ Initiate grating structure 64 ⟩ ≡
  {
    if (¬strcmp(gratingtype, "stepwise")) {
      ⟨ Initiate binary grating structure 65 ⟩;
    }
    else if (¬strcmp(gratingtype, "sinusoidal")) {
      ⟨ Initiate sinusoidal grating structure 66 ⟩;
    }
    else if (¬strcmp(gratingtype, "chirped")) {
      ⟨ Initiate chirped grating structure 67 ⟩;
    }
    else if (¬strcmp(gratingtype, "fractal")) {
      ⟨ Initiate fractal grating structure 68 ⟩;
    }
    else {
      fprintf(stderr, "%s:␣Error:␣Specified␣grating␣type␣is␣invalid.\n", progname);
      showsomehelp();
      exit(FAILURE);
    }
    if (perturbed_gyration_constant) {
      ⟨ Add perturbation of gyration constant along grating structure 69 ⟩;
    }
  }
```

This code is used in section 45.

**65.** For the binary type of stepwise gratings, with the refractive index and all other material parameters of the medium spatially alternating between two distinct values, the odd layers (i.e. $z_k < z < z_{k+1}$, with $k$ being an odd integer) have all the layer thickness given by the parameter *t1*, while all even layers (i.e. $z_k < z < z_{k+1}$, with $k$ being an even integer) have the layer thickness given by *t2*. This means that in this case the total length of the grating is given as

$$L = \{\text{the number of odd layers}\} \times t1 + \{\text{the number of even layers}\} \times t2,$$

and hence the value of any specified grating length (by using the `--gratinglength` option) will be neglected for the `twolevel` stepwise type of grating.

The material parameters *pe*, *qe*, *pm*, and *qm*, are (as previously) defined as

$$\begin{aligned}
pe[k] &= \chi^{\text{eeee}}_{xxxx}(-\omega;\omega,\omega,-\omega) - \chi^{\text{eeee}}_{xyyx}(-\omega;\omega,\omega,-\omega),\\
qe[k] &= \chi^{\text{eeee}}_{xxxx}(-\omega;\omega,\omega,-\omega) + \chi^{\text{eeee}}_{xyyx}(-\omega;\omega,\omega,-\omega),\\
pm[k] &= i(\chi^{\text{eeeem}}_{xyyyz}(-\omega;\omega,\omega,-\omega,0) - \chi^{\text{eeeem}}_{xxxyz}(-\omega;\omega,\omega,-\omega,0))B^z_0,\\
qm[k] &= i(\chi^{\text{eeeem}}_{xyyyz}(-\omega;\omega,\omega,-\omega,0) + \chi^{\text{eeeem}}_{xxxyz}(-\omega;\omega,\omega,-\omega,0))B^z_0,
\end{aligned}$$

where $\chi^{\text{eeee}}_{\mu\alpha\beta\gamma}(-\omega;\omega,\omega,-\omega)$ and $\chi^{\text{eeeem}}_{\mu\alpha\beta\gamma\delta}(-\omega;\omega,\omega,-\omega,0)$ are the nonlinear optical and magneto-optical susceptibility tensors governing the intensity-dependent refractive index and Faraday effect, respectively, taken in a notation conforming to P. N. Butcher and D. Cotter [P. N. Butcher and D. Cotter, *The Elements of Nonlinear Optics* (Cambridge University Press, New York, 1990)], and to be evaluated in respective layer $z_k < z < z_{k+1}$, $k = 1, 2, \ldots, N-1$.

When initializing the binary type of stepwise gratings, the program will check if the *randomdistribution* flag (a numerical integer) was set through the command line parameters. If so (i. e. if *randomdistribution* = 1), then the layers of different indices and material parameters will be randomly ordered.

```
⟨ Initiate binary grating structure 65 ⟩ ≡
  {
    if (¬strcmp(gratingsubtype, "twolevel")) {
      if (randomdistribution) {
        for ($j = 1$; $j \le nn - 1$; $j$++) {
          if ($j \equiv modnum$) {
            if ($j \equiv 1$) $z[j] = 0.0$;
            $z[j+1] = z[j] + modt1$;
            $dz[j] = modt1$;
            $n[j] = modn1$;
            $g[j] = modg1$;
            $pe[j] = modpe1$;
            $pm[j] = modpm1$;
            $qe[j] = modqe1$;
            $qm[j] = modqm1$;
          }
          else {
            $ranseed = ranseed + j$;
            if (ran1(&ranseed) > 0.5) {
              if (verbose) fprintf(stdout, "Random␣number:␣1\n");
              if ($j \equiv 1$) $z[j] = 0.0$;
              $z[j+1] = z[j] + t1$;
              $dz[j] = t1$;
              $n[j] = n1$;
              $g[j] = g1$;
              $pe[j] = pe1$;
              $pm[j] = pm1$;
              $qe[j] = qe1$;
              $qm[j] = qm1$;
            }
            **else** {
              **if** ($verbose$) $fprintf(stdout, "Random␣number:␣0\n")$;
              **if** ($j \equiv 1$) $z[j] = 0.0$;
              $z[j+1] = z[j] + t2$;
              $dz[j] = t2$;
              $n[j] = n2$;
              $g[j] = g2$;
              $pe[j] = pe2$;
              $pm[j] = pm2$;
              $qe[j] = qe2$;
              $qm[j] = qm2$;
            }
          }
        }
      }
      **else** {      /* else, if non-random distribution */
        **if** ($modnum < 0$) {      /* if no modified layer of the grating, ... */
          **for** ($j = 1$; $j \le nn - 1$; $j = j + 2$) {      /* all odd layers, $j = 1, 3, 5, \ldots$ */
            $z[j] = 0.5 * ((\mathbf{double})(j - 1)) * (t1 + t2)$;
            $dz[j] = t1$;
            $n[j] = n1$;
            $g[j] = g1$;
            $pe[j] = pe1$;
            $pm[j] = pm1$;
            $qe[j] = qe1$;
            $qm[j] = qm1$;
            **if** ($j \equiv nn - 1$) $z[nn] = z[nn - 1] + t1$;
          }
          **for** ($j = 2$; $j \le nn - 1$; $j = j + 2$) {      /* all even layers, $j = 2, 4, 6, \ldots$ */
            $z[j] = z[j - 1] + t1$;
            $dz[j] = t2$;
            $n[j] = n2$;
            $g[j] = g2$;
            $pe[j] = pe2$;
            $pm[j] = pm2$;
            $qe[j] = qe2$;
            $qm[j] = qm2$;
            **if** ($j \equiv nn - 1$) $z[nn] = z[nn - 1] + t2$;
          }
        }
        **else** {      /* ... else, if at least one modified layer of the grating */
          **for** ($j = 1$; $j \le nn - 1$; $j{+}{+}$) {
            **if** ($j \equiv 1$) $z[j] = 0.0$;
            **if** ($j \equiv modnum$) {      /* the modified layer */
              $z[j+1] = z[j] + modt1$;
              $dz[j] = modt1$;
              $n[j] = modn1$;
              $g[j] = modg1$;
```

```
              pe[j] = modpe1;
              pm[j] = modpm1;
              qe[j] = modqe1;
              qm[j] = modqm1;
            }
            else {
              tmp = ((double) j)/((double) 2);
              if (tmp − floor(tmp) > 0.25) {     /* if j odd */
                z[j + 1] = z[j] + t1;
                dz[j] = t1;
                n[j] = n1;
                g[j] = g1;
                pe[j] = pe1;
                pm[j] = pm1;
                qe[j] = qe1;
                qm[j] = qm1;
              }
              else {     /* if j even */
                z[j + 1] = z[j] + t2;
                dz[j] = t2;
                n[j] = n2;
                g[j] = g2;
                pe[j] = pe2;
                pm[j] = pm2;
                qe[j] = qe2;
                qm[j] = qm2;
              }
            }
          }
        }
      }
    }
    else {
      fprintf(stderr, "%s:␣Error.\n", progname);
      fprintf(stderr, "%s:␣(No␣valid␣grating␣subtype␣found).\n", progname);
      exit(FAILURE);
    }
  }
```

This code is used in section 64.

**66.** For the sinusoidal type gratings, the grating structure is spatially oversampled and modelled as a large number of thin homogeneous slices, as in the oversampling in some algorithms for calculation of transmission properties of fiber Bragg gratings. In the oversampling, the thickness of each of the layers is equal, using an equidistanly spaced spatial increment in the beam propagation performed across each of the layers. Here, *nper* is the physical, spatial period of the refractive index distribution $n(z)$, while *gper* is the spatial period of the linear magneto-optical gyration constant $g(z)$, etc.

For sinusoidal type gratings, any stated apodization profile will also be applied, in order to get rid of any occurring Gibbs oscillations due to a rapid change of the index modulation at the ends of the grating. From a user perspective, the `--apodize` option is used at startup time of the program for specifying a smoother transition between modulated and non-modulated regions of the grating. The apodization is performed at the ends of the grating according to a multiplicative factor of the *n2* and *g2* modulation amplitudes of the refractive index and gyration coefficient, of the form

$$f(z)=\begin{cases} [1-\cos(\pi z/a)]/2, & 0\le z\le a,\\ 1, & a<z<L-a,\\ [1-\cos(\pi(z-L)/a)]/2, & L-a\le z\le L,\end{cases}$$

and otherwise $f(z)=0$, for any $z$ outside the above domains of definition, where $a$ is the effective apodization length, being the floating point parameter specified after the `--apodize` option, and $L$ as usual the geometrical overall length of the grating.

In the generation of the sinusoidal grating structure, we also include any possible discrete spatial phase jump, as specified by the `--phasejump` command line option.

⟨ Initiate sinusoidal grating structure 66 ⟩ ≡

```
{
  t1 = ll/((double)(nn - 1));
  for (j = 1; j ≤ nn - 1; j++) {
    z[j] = ((double)(j - 1)) * t1;
    dz[j] = t1;
    if (apodize) {
      if ((0.0 ≤ z[j]) ∧ (z[j] ≤ apolength)) {
        tmp = (1.0 - cos(pi * z[j]/apolength))/2.0;
      }
      else if ((apolength ≤ z[j]) ∧ (z[j] ≤ ll - apolength)) {
        tmp = 1.0;
      }
      else if ((ll - apolength ≤ z[j]) ∧ (z[j] ≤ ll)) {
        tmp = (1.0 - cos(pi * (z[j] - ll)/apolength))/2.0;
      }
      else {
        tmp = 0.0;
        fprintf(stderr, "%s:␣Impossible␣apodization␣event␣occurred.", progname);
        fprintf(stderr, "%s:␣(Please␣check␣grating␣initialization.)", progname);
      }
    }
    if (phasejump) {
      if (z[j] ≥ phasejumpposition) {
        phi = phasejumpangle;
      }
      else {
        phi = 0.0;
      }
    }
```

```
      if (apodize) {
        n[j] = n1 + n2 * tmp * sin(twopi * z[j]/nper + phi);
        g[j] = g1 + g2 * tmp * sin(twopi * z[j]/gper + phi);
      }
      else {
        n[j] = n1 + n2 * sin(twopi * z[j]/nper + phi);
        g[j] = g1 + g2 * sin(twopi * z[j]/gper + phi);
      }
      pe[j] = pe1 + pe2 * sin(twopi * z[j]/peper);
      pm[j] = pm1 + pm2 * sin(twopi * z[j]/pmper);
      qe[j] = qe1 + qe2 * sin(twopi * z[j]/qeper);
      qm[j] = qm1 + qm2 * sin(twopi * z[j]/qmper);
    }
    z[nn] = ll;
  }
```

This code is used in section 64.

**67.** Just as for the sinusoidal type gratings, the chirped grating structure is spatially oversampled and modelled as a large number of thin homogeneous slices of equal thickness. As in the case of a pure sinusoidal grating, we here also check whether an apodization should be applied to the grating profile, subsequently applying the apodization. In the generation of the chirped grating structure, we also include any possible discrete spatial phase jump, as specified by the `--phasejump` command line option.

```
⟨ Initiate chirped grating structure 67 ⟩ ≡
  {
    $t1 = ll/((\mathbf{double})(nn - 1))$;
    for ($j = 1$; $j \le nn - 1$; $j{+}{+}$) {
      $z[j] = ((\mathbf{double})(j - 1)) * t1$;
      $dz[j] = t1$;
      if ($apodize$) {
        if ($(0.0 \le z[j]) \land (z[j] \le apolength)$) {
          $tmp = (1.0 - cos(pi * z[j]/apolength))/2.0$;
        }
        else if ($(apolength \le z[j]) \land (z[j] \le ll - apolength)$) {
          $tmp = 1.0$;
        }
        else if ($(ll - apolength \le z[j]) \land (z[j] \le ll)$) {
          $tmp = (1.0 - cos(pi * (z[j] - ll)/apolength))/2.0$;
        }
        else {
          $tmp = 0.0$;
          fprintf(stderr, "%s:␣Impossible␣apodization␣event␣occurred.", progname);
          fprintf(stderr, "%s:␣(Please␣check␣grating␣initialization.)", progname);
        }
      }
      if ($phasejump$) {
        if ($z[j] \ge phasejumpposition$) {
          $phi = phasejumpangle$;
        }
        else {
          $phi = 0.0$;
        }
      }
      if ($apodize$) {
        if ($(fabs(ncrp) > 0.0)$)      /* if nonzero chirp of $n(z)$ */
          $n[j] = n1 + n2 * tmp * sin((twopi/ncrp) * log(1.0 + ncrp * z[j]/nper) + phi)$;
        else $n[j] = n1 + n2 * tmp * sin(twopi * z[j]/nper + phi)$;
        if ($(fabs(gcrp) > 0.0)$)      /* if nonzero chirp of $g(z)$ */
          $g[j] = g1 + g2 * tmp * sin((twopi/gcrp) * log(1.0 + gcrp * z[j]/gper) + phi)$;
        else $g[j] = g1 + g2 * tmp * sin(twopi * z[j]/gper + phi)$;
      }
      else {
        if ($(fabs(ncrp) > 0.0)$)      /* if nonzero chirp of $n(z)$ */
          $n[j] = n1 + n2 * sin((twopi/ncrp) * log(1.0 + ncrp * z[j]/nper) + phi)$;
        else $n[j] = n1 + n2 * sin(twopi * z[j]/nper + phi)$;
        if ($(fabs(gcrp) > 0.0)$)      /* if nonzero chirp of $g(z)$ */
          $g[j] = g1 + g2 * sin((twopi/gcrp) * log(1.0 + gcrp * z[j]/gper) + phi)$;
        else $g[j] = g1 + g2 * sin(twopi * z[j]/gper + phi)$;
      }
      if ($(fabs(pecrp) > 0.0)$)      /* if nonzero chirp of $p_e(z)$ */
        $pe[j] = pe1 + pe2 * sin((twopi/pecrp) * log(1.0 + pecrp * z[j]/peper));$
      else $pe[j] = pe1 + pe2 * sin(twopi * z[j]/peper);$
      if $(pmcrp * pmcrp > 0.0)$    /* if nonzero chirp of $p_{\mathrm{m}}(z)$ */
        $pm[j] = pm1 + pm2 * sin((twopi/pmcrp) * log(1.0 + pmcrp * z[j]/pmper));$
      else $pm[j] = pm1 + pm2 * sin(twopi * z[j]/pmper);$
      if $(fabs(qecrp) > 0.0)$    /* if nonzero chirp of $q_{\mathrm{e}}(z)$ */
        $qe[j] = qe1 + qe2 * sin((twopi/qecrp) * log(1.0 + qecrp * z[j]/qeper));$
      else $qe[j] = qe1 + qe2 * sin(twopi * z[j]/qeper);$
      if $(fabs(qmcrp) > 0.0)$    /* if nonzero chirp of $q_{\mathrm{m}}(z)$ */
        $qm[j] = qm1 + qm2 * sin((twopi/qmcrp) * log(1.0 + qmcrp * z[j]/qmper));$
      else $qm[j] = qm1 + qm2 * sin(twopi * z[j]/qmper);$
    }
    $z[nn] = ll;$
  }
```

This code is used in section 64.

**68.** For the fractal type gratings, the grating structure is composed of a self-similar structure with certain scaling properties. Currently only the Cantor-type fractal is possible to apply to the grating structure.

The initialization of the Cantor-type grating is performed by one single call to *init_cantor_fractal_grating*( ), which intiates the positions $z_k$ at which the interfaces between media of different properties are located. The *init_cantor_fractal_grating*( ) routine then makes recursive calls to itself, until the bottom level of the fractal initialization is reached. After this, the material parameters of these regions are set sequentially in the same way as for binary type gratings.

In the initialization of the Cantor-type fractal grating, the program uses the vector $z[1..N]$ with upper bound determined by the 'level' $p$ of the fractal as $N = 2^p$. The value of $N$ is calculated and set immediately after the program has parsed the level from the command line options, and hence any additional specifications of $N$ are superfluous, as this is set by the fractal level. If the number of elements in $z$ is *not* an integer power of two, then the call to *init_cantor_fractal_grating*( ) will fail, leaving the error message of this routine on exit. This will happen if, for example, a command line option specifying $N$ appears after the specification of the fractal type grating, if the specified value for $N$ does not conform with the convention that $N = 2^p$.

```
⟨ Initiate fractal grating structure 68 ⟩ ≡
{
   tmp = 1.0;
   for (j = 1; j ≤ fractal_level − 1; j++) tmp = 2.0 ∗ tmp;
   for (j = 1; j ≤ maximum_fractal_level − fractal_level; j++) tmp = 3.0 ∗ tmp;
      /∗ leaves tmp = 2(p − 1)3(p_max − p) ∗/
   ll = tmp ∗ t1 − tmp ∗ t2;
   tmp = 1.0;
   for (j = 1; j ≤ maximum_fractal_level − 1; j++) tmp = 3.0 ∗ tmp;
   ll = ll + tmp ∗ t2;
   if (verbose) {
      fprintf(stdout, "%s:␣Minimum␣layer␣thickness␣at␣maximum␣recursion␣depth:\n", progname);
      fprintf(stdout, "%s:␣␣␣␣␣t2=%f␣[nm]\n", progname, t1 ∗ 1.0 · 10^9);
      fprintf(stdout, "%s:␣␣␣␣␣t2=%f␣[nm]\n", progname, t2 ∗ 1.0 · 10^9);
      fprintf(stdout, "%s:␣Fractal␣grating␣length␣calculated␣as␣L=%e␣[m]\n", progname, ll);
      fprintf(stdout, "%s:␣Based␣on␣fractal␣recursion␣of␣%d␣out␣of␣a␣maximum␣of␣%d␣levels.\n",
            progname, fractal_level, maximum_fractal_level);
   }
   init_cantor_fractal_grating(z, 1, nn, 0.0, ll, n1, n2);
   for (j = 1; j ≤ nn − 1; j++) {
      dz[j] = z[j + 1] − z[j];
      if (j ≡ 1) {
         odd_layer = 1;
      }
      else {
         odd_layer = (odd_layer ? 0 : 1);
      }
      n[j] = (odd_layer ? n1 : n2);
      g[j] = (odd_layer ? g1 : g2);
      pe[j] = (odd_layer ? pe1 : pe2);
      pm[j] = (odd_layer ? pm1 : pm2);
      qe[j] = (odd_layer ? qe1 : qe2);
      qm[j] = (odd_layer ? qm1 : qm2);
   }
}
```

This code is used in section 64.

**69.** Adding any present perturbation of gyration constant. If the user via the command line has specified that a perturbation of the gyration constant should be added to the present spatial distribution, then *perturbed_gyration_constant* will be set to unity (true), and a perturbation of the profile corresponding to the magnetic field strength of a current carrying wire, orthogonal to the direction of propagation of light, will be added. The added perturbation $\Delta g(z)$ will be a "bump" of the Lorentzian spatial shape

$$\Delta g(z) = \frac{a_{\rm p}}{1 + 4(z - z_{\rm p})^2/w_{\rm p}^2},$$

where $z_{\rm p}$ is the position, $a_{\rm p}$ is the zero-to-peak amplitude, and $w_{\rm p}$ the full width half maximum of the perturbation. This form of perturbation applied to chirped sinusoidal magneto-optical Bragg gratings has in a linear optical regime been analyzed for spectral windowing and filtering, in [F. Jonsson and C. Flytzanis, JOSAB (2005); F. Jonsson and C. Flytzanis, Proc. MRS Fall meeting (2005)].

⟨Add perturbation of gyration constant along grating structure 69⟩ ≡

```
{
  for (j = 1; j ≤ nn − 1; j++) {
    tmp = 2.0 ∗ (z[j] − gyroperturb_position)/gyroperturb_width;
    g[j] += gyroperturb_amplitude/(1.0 + tmp ∗ tmp);
  }
}
```

This code is used in section 64.

**70.** Set the refractive index of the medium surrounding the magneto-optical Bragg grating. The first and last elements of the double vectors $n[0, nn]$ and $g[0, nn]$ are defined according to the convention

$$\begin{aligned} n[0] &\equiv n(z_0^-), & n[nn] &\equiv n(z_N^+), \\ g[0] &\equiv g(z_0^-), & g[nn] &\equiv g(z_N^+), \end{aligned}$$

and the surrounding medium is here assumed to be linear,

$$p_{\rm e,m}(z_0^-) = p_{\rm e,m}(z_N^+) = q_{\rm e,m}(z_0^-) = q_{\rm e,m}(z_N^+) = 0,$$

and non-gyrotropic ($g_0 = g_N = 0$).

⟨Initiate surrounding medium 70⟩ ≡

```
{
  n[0] = nsurr;    /* n(z_0^-) */
  n[nn] = nsurr;    /* n(z_N^+) */
  g[0] = 0.0;    /* g(z_0^-) */
  g[nn] = 0.0;    /* g(z_N^+) */
}
```

This code is used in section 45.

**71. Calculation of intra-grating layer reflectances.** Calculate the intrinsic reflectances over the layer interfaces prior to entering the algorithm of calculation of field distributions or reflection and transmission coefficients. The reflectance and transmission coefficients across the layers are calculated in terms of their linear optical and magneto-optical material parameters as

$$\begin{aligned}
taup[j] &= \tau_+(z_j) = \frac{2(n_{j-1}+g_{j-1})}{(n_{j-1}+n_j+g_{j-1}+g_j)}, & taum[j] &= \tau_-(z_j) = \frac{2(n_{j-1}-g_{j-1})}{(n_{j-1}+n_j-g_{j-1}-g_j)},\\
taupp[j] &= \tau'_+(z_j) = \frac{2(n_j-g_j)}{(n_{j-1}+n_j-g_{j-1}-g_j)}, & taupm[j] &= \tau'_-(z_j) = \frac{2(n_j+g_j)}{(n_{j-1}+n_j+g_{j-1}+g_j)},\\
rhop[j] &= \rho_+(z_j) = \frac{(n_{j-1}-n_j+g_{j-1}-g_j)}{(n_{j-1}+n_j+g_{j-1}+g_j)}, & rhom[j] &= \rho_-(z_j) = \frac{(n_{j-1}-n_j-g_{j-1}+g_j)}{(n_{j-1}+n_j-g_{j-1}-g_j)},\\
rhopp[j] &= \rho'_+(z_j) = -\rho_-(z_j), & rhopm[j] &= \rho'_-(z_j) = -\rho_+(z_j),
\end{aligned}$$

for interface $z = z_j$, $j = 1, 2, \ldots, nn$, where $n_j$ and $g_j$ are the respective refractive indices and gyration constants of the layers $z_j < z < z_{j+1}$. Outside the grating a zero gyration coefficient is assumed while the refractive indices are specified by $n_0 = n_N = nsurr$, as given through the command line options.

In these expressions, $\tau_\pm(z_j)$ are the layer reflectances for forward propagating left/right circularly polarized light (i. e. for light coming from the negative $z$-direction), while $\tau'_\pm(z_j)$ are the layer reflectances for backward propagating left/right circularly polarized light (i. e. for light coming from the positive $z$-direction).

These polarization selective amplitude reflectances satisfy the Stokes relations

$$\rho_\pm(z_j) = -\rho'_\mp(z_j), \qquad \tau_\pm(z_j)\tau'_\mp(z_j) = 1 - \rho^2_\pm(z_j),$$

in this particular case generalized to interfaces between magneto-optic media. Notice the reversed order of the subscripts of the reflectances in the Stokes relations, reflecting the nonreciprocity of the magneto-optical contributions to the refractive index.

```
⟨ Calculate intragrating layer reflectances 71 ⟩ ≡
{
  for (j = 1; j ≤ nn; j++) {
    taup[j] = 2.0 * (n[j − 1] + g[j − 1])/(n[j − 1] + n[j] + g[j − 1] + g[j]);
    taum[j] = 2.0 * (n[j − 1] − g[j − 1])/(n[j − 1] + n[j] − g[j − 1] − g[j]);
    taupp[j] = 2.0 * (n[j] − g[j])/(n[j − 1] + n[j] − g[j − 1] − g[j]);
    taupm[j] = 2.0 * (n[j] + g[j])/(n[j − 1] + n[j] + g[j − 1] + g[j]);
    rhop[j] = (n[j − 1] − n[j] + g[j − 1] − g[j])/(n[j − 1] + n[j] + g[j − 1] + g[j]);
    rhom[j] = (n[j − 1] − n[j] − g[j − 1] + g[j])/(n[j − 1] + n[j] − g[j − 1] − g[j]);
    rhopp[j] = −rhom[j];
    rhopm[j] = −rhop[j];
  }
}
```

This code is used in section 45.

**72. Calculating the electrical field distribution inside the grating.** Having constructed all data needed for the analysis of optical wave propagation in the grating structure, in linear as well as nonlinear, all-optically as well as magneto-optically, we are now in position of entering the actual electromagnetic field calculations. The field calculation is performed for a set of vacuum wavelengths within the spectral range specified by *lambdastart* and *lambdastop*. The spectrum is sampled using equidistantly spaced wavelength increments.

If the user instead of stating equidistantly spaced transmitted intensities and ellipticities has specified a file where to find the trajectory of Stokes parameters $(W_0, W_3)$ for the transmitted light, by using the `--trmtraject` option, then *mme* is set to be equal to the number of points *mmtraject* on this trajectory, and *mmi* set to one. For this case, for every value of $ke = 1, 2, 3, \ldots, mme$, the *trmintensity* and *trmellipticity* variables will be set according to the data supplied in the specified trajectory file.

```
⟨ Calculate incident optical field spectrum 72 ⟩ ≡
  {
    for (k = 1; k ≤ mm; k++) {      /* for all wavelengths in the spectrum window */
      if (mm > 1) {      /* if more than one sampling point in the spectrum */
        lambda = lambdastart + (((double)(k − 1))/((double)(mm − 1))) ∗ (lambdastop − lambdastart);
      }
      else {
        lambda = lambdastart;
      }
      omega = twopi ∗ c/lambda;      /* angular frequency of the light */
      if (trmtraject_specified) {
        mme = mmtraject;
        mmi = 1;
      }
      ⟨ Scan transmitted optical field in ellipticity and intensity 74 ⟩;
    }
    if (verbose) ⟨ Display elapsed execution time 73 ⟩;
  }
```

This code is used in section 45.

**73.** Display the total execution time consumed by the simulation. This is the last block to be executed by the program, and employs the *difftime* routine of the standard C library `time.h`.

```
⟨ Display elapsed execution time 73 ⟩ ≡
  {
    fprintf(stdout, "␣...done.␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣");
    now = time(Λ);
    fprintf(stdout, "Elapsed␣execution␣time:␣%d␣s\n", ((int) difftime(now, initime)));
    for (k = 1; k ≤ 64; k++) fprintf(stdout, (k < 64 ? "-" : "\n"));
    fprintf(stdout, "Program␣execution␣closed␣%s", ctime(&now));
  }
```

This code is used in section 72.

**74.** For a specified range of the intensity and ellipticity of polarization state of the transmitted optical field, calculate the corresponding incident fields in this inverse formulation of the electromagnetic wave propagation problem. The range of the ellipticity $\varepsilon_{\rm T}$ of the polarization state of the transmitted optical field is given as

$$\varepsilon_{\rm T} \in [\textit{trmellipstart}, \textit{trmellipstop}],$$

where *trmellipstart* and *trmellipstop* are the parameters provided at startup of the program through the command line option `--trmellipticity` ⟨*trmellipstart*⟩ ⟨*trmellipstop*⟩ ⟨*mme*⟩. Via the definition of the normalized ellipticity $\varepsilon_{\rm T}$, the *trmellipstart* and *trmellipstop* parameters are bound to

$$-1 \le \textit{trmellipstart} \le \textit{trmellipstop} \le 1.$$

```
⟨ Scan transmitted optical field in ellipticity and intensity 74 ⟩ ≡
  {
    for (ke = 1; ke ≤ mme; ke++) {
      if (trmtraject_specified) {
        trmellipticity = w3traj[ke]/w0traj[ke];
      }
      else {
        if (mme > 1) {
          trmellipticity = trmellipstart + (((double)(ke − 1))/((double)(mme − 1))) ∗ (trmellipstop −
              trmellipstart);
        }
        else {
          trmellipticity = trmellipstart;
        }
      }
      ⟨ Scan transmitted optical field in intensity 75 ⟩;
    }
  }
```

This code is used in section 72.

**75.** For a specified range of the intensity of the transmitted optical field, calculate the corresponding incident fields in this inverse formulation of the electromagnetic wave propagation problem. The range of the intensity $I_{\rm T}$ of the transmitted optical field is given as

$$I_{\rm T} \in [\mathit{trmintenstart}, \mathit{trmintenstop}],$$

where *trmintenstart* and *trmintenstop* are the parameters provided at startup of the program through the command line option `--trmintensity` ⟨*trmintenstart*⟩ ⟨*trmintenstop*⟩ ⟨*mmi*⟩.

```
⟨ Scan transmitted optical field in intensity 75 ⟩ ≡
  {
    for (ki = 1; ki ≤ mmi; ki++) {
      if (trmtraject_specified) {
        trmintensity = (epsilon0 * c/2.0) * w0traj[ke];
      }
      else {
        if (mmi > 1) {
          trmintensity = trmintenstart + (((double)(ki − 1))/((double)(mmi − 1))) * (trmintenstop −
                trmintenstart);
        }
        else {
          trmintensity = trmintenstart;
        }
      }
      ⟨ Set boundary conditions at end of grating 76 ⟩;
      ⟨ Calculate optical field in last layer of the grating 77 ⟩;
      ⟨ Propagate optical fields from last to first layer of the grating 78 ⟩;
      ⟨ Write Stokes parameters and reflection coefficients to file 83 ⟩;
      ⟨ Write intragrating field evolution to file 84 ⟩;
      ⟨ Write intragrating intensity evolution to file 85 ⟩;
      ⟨ Write spatial grating structure to file 86 ⟩;
    }
  }
```

This code is used in section 74.

**76.** In this inverse formulation of the algorithm, apply the boundary condition that the backward propagating wave at the end of the grating is zero,

$$E^{\rm b}_{N_\pm}(z_N) \equiv E^{\rm b}_\pm(z_N^+) = 0,$$

and that the forward propagating field is a given quantity, specified in terms of intensity and ellipticity of the polarization state, as

$$E^{\rm f}_{N_+}(z_N) \equiv E^{\rm f}_+(z_N^+) = [(1+\varepsilon_{\rm T})I_{\rm T}/(\varepsilon_0 c)]^{1/2}\,,$$
$$E^{\rm f}_{N_-}(z_N) \equiv E^{\rm f}_-(z_N^+) = [(1-\varepsilon_{\rm T})I_{\rm T}/(\varepsilon_0 c)]^{1/2}\,,$$

with values of transmitted intensity and ellipticity in the range specified by parameters parsed from the command line options, supplied during startup of the program. Here the transmitted intensity $I_{\rm T}$ (in the program described by the variable *trmintensity*) is expressed in regular SI units, in W/m$^2$ (Watts per square meter), as

$$I_{\rm T} = (c\varepsilon_0/2)(|E^{\rm f}_+(z_N^+)|^2 + |E^{\rm f}_-(z_N^+)|^2),$$

and the normalized transmitted ellipticity $\varepsilon_{\rm T}$ of the transmitted polarization state (in the program described by the variable *trmellipticity*)

$$\varepsilon_{\rm T} = \frac{|E^{\rm f}_+(z_N^+)|^2 - |E^{\rm f}_-(z_N^+)|^2}{|E^{\rm f}_+(z_N^+)|^2 + |E^{\rm f}_-(z_N^+)|^2},$$

is a number in the range from −1 to 1, with −1 corresponding to right circularly polarized (RCP), 0 to linearly polarized, and 1 left circularly polarized (LCP) light.

⟨ Set boundary conditions at end of grating 76 ⟩ ≡

```
{
  ebp[nn] = complex(0.0, 0.0);
  ebm[nn] = complex(0.0, 0.0);
  efp[nn] = complex(sqrt((1.0 + trmellipticity) * trmintensity/(c * epsilon0)), 0.0);
  efm[nn] = complex(sqrt((1.0 − trmellipticity) * trmintensity/(c * epsilon0)), 0.0);
}
```

This code is used in section 75.

**77.** Having applied the boundary conditions at the end of the grating, calculate the optical fields in the last layer, for which $j = N - 1$. These fields are taken immediately next to the last interface, at $z = z_N^-$.

⟨ Calculate optical field in last layer of the grating 77 ⟩ ≡

```
{
  efp[nn − 1] = cmul(crdiv(efp[nn], taup[nn]), crexpi(−omega * n[nn − 1] * dz[nn − 1]/c));
  efm[nn − 1] = cmul(crdiv(efm[nn], taum[nn]), crexpi(−omega * n[nn − 1] * dz[nn − 1]/c));
  ebp[nn − 1] = cmul(rcmul(rhom[nn], efm[nn − 1]), crexpi(2.0 * omega * n[nn − 1] * dz[nn − 1]/c));
  ebm[nn − 1] = cmul(rcmul(rhop[nn], efp[nn − 1]), crexpi(2.0 * omega * n[nn − 1] * dz[nn − 1]/c));
}
```

This code is used in section 75.

**78.** Given the right and left circularly polarized components of the forward and backward propagating optical fields in the last layer, iterate the propagation over the whole grating structure in order to find the corresponding input optical fields. This is done in an iterative manner over all the layers $z_j$, starting with the last layer (for which $j = N - 1$) and ending up at the first layer (for which $j = 1$), successively propagating the optical field over one layer at the time.

As we enter the loop in the order $j = N-1, N-2, \ldots, 1$, the forward and backward propagating optical field components immediately to the "left" of $z_{j+1}$ are contained in the complex vector elements *efp*[*j*], *efm*[*j*], *ebp*[*j*], and *ebm*[*j*], with

$$efp[j] = E^f_{j_+}(z^-_{j+1}), \qquad efm[j] = E^f_{j_-}(z^-_{j+1}), \qquad ebp[j] = E^b_{j_+}(z^-_{j+1}), \qquad ebm[j] = E^b_{j_-}(z^-_{j+1}),$$

where $z = z^-_j$ denotes the position immediately to the "left" of $z = z_j$. We will now propagate the fields from $z_{j+1}$ to $z_j$, for $j = N-1, N-2, \ldots, 1$ (in that order), by first calculating the nonlinear (optical field-dependent) propagation constants of the current layer $z_j < z < z_{j+1}$.

This block of code also deals with displaying information on the progress of calculation, using the standard ANSI C time library for the calculation of "estimated time of arrival" (ETA) for the finishing of execution.

```
⟨ Propagate optical fields from last to first layer of the grating 78 ⟩ ≡
  {
    for (j = nn − 1; j ≥ 1; j−−) {
      ⟨ Calculate nonlinear propagation constants of layer 79 ⟩;
      ⟨ Propagate fields over homogeneous layer 80 ⟩;
      ⟨ Propagate fields over interface to next layer 81 ⟩;
      if (verbose) {
        ⟨ Display simulation status and estimated time of arrival 82 ⟩;
      }
    }
  }
```

This code is used in section 75.

**79.** Calculate the nonlinear, optical field- and polarization-dependent contributions to the refractive index, including electric dipolar (all-optical) as well as magetic dipolar (magneto-optical) contributions.

Here the local variables are defined as

$$\begin{aligned} aafp2 &= |E^f_{j+}(z_j)|^2, & etafp[j] &= \eta^{\rm f}_{+}(z_j)\\ aafm2 &= |E^f_{j-}(z_j)|^2, & etafm[j] &= \eta^{\rm f}_{-}(z_j)\\ aabp2 &= |E^b_{j+}(z_j)|^2, & etabp[j] &= \eta^{\rm b}_{+}(z_j)\\ aabm2 &= |E^b_{j-}(z_j)|^2, & etabm[j] &= \eta^{\rm b}_{-}(z_j) \end{aligned}$$

and the optical field-dependent propagation constants $\eta^{\rm f}_{\pm}$ and $\eta^{\rm b}_{\pm}$ are given in terms of the optical fields and material parameters of the layer $z_j < z < z_{j+1}$ as

$$\begin{aligned} etafp[j] &= \frac{3}{8n_j}((pe[j]+pm[j])(|E^f_{j+}(z_j)|^2+2|E^b_{j-}(z_j)|^2)+(qe[j]+qm[j])(|E^f_{j-}(z_j)|^2+|E^b_{j+}(z_j)|^2)),\\ etafm[j] &= \frac{3}{8n_j}((pe[j]-pm[j])(|E^f_{j-}(z_j)|^2+2|E^b_{j+}(z_j)|^2)+(qe[j]-qm[j])(|E^f_{j+}(z_j)|^2+|E^b_{j-}(z_j)|^2)),\\ etabp[j] &= \frac{3}{8n_j}((pe[j]-pm[j])(|E^b_{j+}(z_j)|^2+2|E^f_{j-}(z_j)|^2)+(qe[j]-qm[j])(|E^b_{j-}(z_j)|^2+|E^f_{j+}(z_j)|^2)),\\ etabm[j] &= \frac{3}{8n_j}((pe[j]+pm[j])(|E^b_{j-}(z_j)|^2+2|E^f_{j+}(z_j)|^2)+(qe[j]+qm[j])(|E^b_{j+}(z_j)|^2+|E^f_{j-}(z_j)|^2)). \end{aligned}$$

For a more strict derivation of these parameters, as governing the phase evolution of light in homogeneous nonlinear magneto-optical Kerr-media, see F. Jonsson and C. Flytzanis, *Phys. Rev. Lett.* **82**, 1426 (1999).

⟨ Calculate nonlinear propagation constants of layer 79 ⟩ ≡

```
{
  aafp2 = cdabs(efp[j]);
  aafm2 = cdabs(efm[j]);
  aabp2 = cdabs(ebp[j]);
  aabm2 = cdabs(ebm[j]);
  aafp2 *= aafp2;    /* equals $|E^f_{j+}(z_j)|^2$ */
  aafm2 *= aafm2;    /* equals $|E^f_{j-}(z_j)|^2$ */
  aabp2 *= aabp2;    /* equals $|E^b_{j+}(z_j)|^2$ */
  aabm2 *= aabm2;    /* equals $|E^b_{j-}(z_j)|^2$ */
  tmp = 3.0/(8.0 * n[j]);
  etafp[j] = tmp * ((pe[j] + pm[j]) * (aafp2 + 2.0 * aabm2) + (qe[j] + qm[j]) * (aafm2 + aabp2));
  etafm[j] = tmp * ((pe[j] - pm[j]) * (aafm2 + 2.0 * aabp2) + (qe[j] - qm[j]) * (aafp2 + aabm2));
  etabp[j] = tmp * ((pe[j] - pm[j]) * (aabp2 + 2.0 * aafm2) + (qe[j] - qm[j]) * (aabm2 + aafp2));
  etabm[j] = tmp * ((pe[j] + pm[j]) * (aabm2 + 2.0 * aafp2) + (qe[j] + qm[j]) * (aabp2 + aafm2));
}
```

This code is used in section 78.

**80.** Having calculated the nonlinear propagation constants of the layer, now propagate the optical fields to the next interface in the negative $z$-direction. This wave propagation in homogeneous slices of the medium is performed using the analytical solution for loss-less propagation as published in F. Jonsson and C. Flytzanis, Phys. Rev. Lett. **82**, 1426 (1999).

After this propagation over the homogeneous slice (layer), the temporary variables *tmpfp*, *tmpfm*, *tmpbp*, and *tmpbm* contain the forward and backward travelling optical fields taken at the "beginning" of the current, homogeneous segment.

```
⟨ Propagate fields over homogeneous layer 80 ⟩ ≡
  {
    tmpfp = cmul(efp[j], crexpi(−omega ∗ (etafp[j] + g[j]) ∗ dz[j]/c));
    tmpfm = cmul(efm[j], crexpi(−omega ∗ (etafm[j] − g[j]) ∗ dz[j]/c));
    tmpbp = cmul(ebp[j], crexpi(omega ∗ (etabp[j] − g[j]) ∗ dz[j]/c));
    tmpbm = cmul(ebm[j], crexpi(omega ∗ (etabm[j] + g[j]) ∗ dz[j]/c));
  }
```

This code is used in section 78.

**81.** Make the passage over the interface to the next layer. The infinitesimal propagation of the waves over the interfaces between homogeneous layers is performed by using the previously calculated reflection and transmission coefficients, taking the linear magneto-optical effect into account as well.

```
⟨ Propagate fields over interface to next layer 81 ⟩ ≡
  {
    if (j > 1) {
      efp[j − 1] = crdiv(cmul(csub(tmpfp, rcmul(rhopm[j], tmpbm)),
            crexpi(−omega ∗ n[j − 1] ∗ dz[j − 1]/c)), taup[j]);
      efm[j − 1] = crdiv(cmul(csub(tmpfm, rcmul(rhopp[j], tmpbp)),
            crexpi(−omega ∗ n[j − 1] ∗ dz[j − 1]/c)), taum[j]);
      ebp[j − 1] = cadd(rcmul(taupp[j], cmul(tmpbp, crexpi(omega ∗ n[j − 1] ∗ dz[j − 1]/c))),
            rcmul(rhom[j], cmul(efm[j − 1], crexpi(2.0 ∗ omega ∗ n[j − 1] ∗ dz[j − 1]/c))));
      ebm[j − 1] = cadd(rcmul(taupm[j], cmul(tmpbm, crexpi(omega ∗ n[j − 1] ∗ dz[j − 1]/c))),
            rcmul(rhop[j], cmul(efp[j − 1], crexpi(2.0 ∗ omega ∗ n[j − 1] ∗ dz[j − 1]/c))));
    }
    else {
      efp[0] = crdiv(csub(tmpfp, rcmul(rhopm[1], tmpbm)), taup[1]);
      efm[0] = crdiv(csub(tmpfm, rcmul(rhopp[1], tmpbp)), taum[1]);
      ebp[0] = cadd(rcmul(taupp[1], tmpbp), rcmul(rhom[1], efm[0]));
      ebm[0] = cadd(rcmul(taupm[1], tmpbm), rcmul(rhop[1], efp[0]));
    }
  }
```

This code is used in section 78.

**82.** Display the real-time progress in the simulation and calculate the estimated time of arrival for finishing of the simulation. The completeness of the simulation is displayed for every ten percent up to successful termination of the program at one hundred percent.

The formula for calculation of the progress is

$$\left\{\begin{matrix}\text{progress in}\\ \text{percent}\end{matrix}\right\} = 100 \times \frac{(N-j-1)+(k_{\rm i}-1)(N-1)+k_{\rm e}M_{\rm i}(N-1)+(k-1)M_{\rm e}M_{\rm i}(N-1)}{MM_{\rm e}M_{\rm e}(N-1)},$$

where

$$1 \le j \le N \qquad \text{[Layer counter index]}$$
$$1 \le k \le M \qquad \text{[Wavelength counter index]}$$
$$1 \le k_{\rm i} \le M_{\rm i} \qquad \text{[Intensity counter index]}$$
$$1 \le k_{\rm e} \le M_{\rm e} \qquad \text{[Ellipticity counter index]}$$

⟨ Display simulation status and estimated time of arrival 82 ⟩ ≡

```
{
  modf(100.0 * ((float)((nn − j − 1) + (ki − 1) * (nn − 1) + (ke − 1) * mmi * (nn − 1) + (k − 1) * mme *
      mmi * (nn − 1)))/((float)(mm * mme * mmi * (nn − 1))), &stn);
  if (stn > status + 10) {
    status = status + 10;
    now = time(Λ);
    eta = initime + ((int)((100.0/((double) status)) * difftime(now, initime)));
    fprintf(stdout, "␣...%2d␣percent␣finished...␣␣␣", status);
    fprintf(stdout, "␣ETA:␣%s", ctime(&eta));
  }
}
```

This code is used in section 78.

**83.** Having calculated the input electromagnetic field from the given output field (as in this case, using the implicit way of calculating the transmission characteristics of a nonlinear magneto-optical Bragg grating), now calculate the corresponding incident $(S_0, S_1, S_2, S_3)$, transmitted $(W_0, W_1, W_2, W_3)$, and reflected $(V_0, V_1, V_2, V_3)$ Stokes parameters of the optical fields. Notice that in this inverse formulation of the algorithm, the transmitted Stokes parameters are not really calculated in a strict sense, since they follow directly from the given input to the program – the transmitted optical field.

The Stokes parameters are defined as conforming to the standard definition in J. D. Jackson's *Classical Electrodynamics* (Wiley, New York, 1975), as

$$S_0 = |E^f_+(z_0^-)|^2 + |E^f_-(z_0^-)|^2, \qquad S_1 = 2\,\mathrm{Re}[E^{f*}_+(z_0^-)E^f_-(z_0^-)],$$
$$S_3 = |E^f_+(z_0^-)|^2 - |E^f_-(z_0^-)|^2, \qquad S_2 = 2\,\mathrm{Im}[E^{f*}_+(z_0^-)E^f_-(z_0^-)],$$

for the incident wave,

$$W_0 = |E^f_+(z_N^+)|^2 + |E^f_-(z_N^+)|^2, \qquad W_1 = 2\,\mathrm{Re}[E^{f*}_+(z_N^+)E^f_-(z_N^+)],$$
$$W_3 = |E^f_+(z_N^+)|^2 - |E^f_-(z_N^+)|^2, \qquad W_2 = 2\,\mathrm{Im}[E^{f*}_+(z_N^+)E^f_-(z_N^+)],$$

for the transmitted wave, and finally

$$V_0 = |E^b_+(z_0^-)|^2 + |E^b_-(z_0^-)|^2, \qquad V_1 = 2\,\mathrm{Re}[E^{b*}_+(z_0^-)E^b_-(z_0^-)],$$
$$V_3 = |E^b_+(z_0^-)|^2 - |E^b_-(z_0^-)|^2, \qquad V_2 = 2\,\mathrm{Im}[E^{b*}_+(z_0^-)E^b_-(z_0^-)],$$

for the reflected wave of the grating structure. These parameters fully specify the reflection and transmission characteristics for the nonlinear magneto-optical grating structure, in a linear as well as nonlinear optical domain. In particular, the intensity transmission and reflection coefficients are expressed in as $T = W_0/S_0$ and $R = V_0/S_0$, respectively.

In this calculation, a check is being performed regarding if the simulation is being performed for a multiple set of intensities and ellipticities of the transmitted light; if found to be so, then the set of Stokes parameters are calculated in order to create topological graphs as in F. Jonsson and C. Flytzanis, *Phys. Rev. Lett.* **82**, 1426 (1999), in which case only one particular wavelength should being considered for the simulation (otherwise we would end up with too many topological graphs to keep track of); otherwise the complex-valued amplitude reflection and transmission coefficients,

$$\rho_\pm(\omega) = E^{\rm b}_{0_\mp}(z_0)/E^{\rm f}_{0_\pm}(z_0), \qquad \tau_\pm(\omega) = E^{\rm f}_{N_\pm}(z_N)/E^{\rm f}_{0_\pm}(z_0),$$

are calculated, in order to create graphs of the spectral reflection and transmission properties of the magneto-optical grating for circularly polarized fields (in which case $k > 1$ is implicitly assumed).

If Stokes parameters are generated as data ouput, the resulting output files (named according to the suffix convention *outfilename*`.s0.dat`, *outfilename*`.s1.dat`, … *outfilename*`.v0.dat`, …, etc.), can after the simulation be used for creating topological graphs.

For instance, by mapping the contents of Stokes parameter files *outfilename*`.s0.dat`, *outfilename*`.s3.dat`, and *outfilename*`.w0.dat` (using, for example, the MATLAB command `mesh(s0,s3./s0,w0)`, assuming that the variables `s0`, `s3`, and `w0` were loaded using the MATLAB commands `s0=load(\outfilename.s0.dat); s3=load(\outfilename.s3.dat); w0=load(\outfilename.w0.dat);`), one gets a topological graph showing (*s0*,*s3*,*w0*) as defining a surface showing the transmitted intensity *w0* as function of the input intensity *s0* and input ellipticity *s3*.

Similarly, two-dimensional graphs can be generated by, as an example, instead using the MATLAB command `contour(s3./s0,w3./w0,s0)`, to get graphs of the transmitted ellipticity `w3./w0` (*w3*/*w0*, the "$y$-coordinate") as function of the input ellipticity `s3./s0` (*s3*/*s0*, the "$x$-coordinate"), for various values of the input intensity `s0` (*s0*, the "$z$-coordinate"). This way of creating two-dimensional graphs from a parametric hypersurface of Stokes-parameters might seem to be a cumbersome way of generating a visible output; however, it actually turns out to be a very convenient method. Once one has adopted to this somewhat topologically directed way of thinking, any element of the Stokes vector is easily visualized as function of any of the other variables, and one avoids the difficulty that arise whenever the parameter of interest is a multivalued function of the other parameters (particularly appreciated when considering, for example, optically bistable behaviour of the structure, since one for those cases carefully must sort out data that belong to different state branches).

As a default, and as a matter of convention of electrodynamics, all Stokes parameters are given in $\mathrm{V}^2/\mathrm{m}^2$, through their definition. For example, the incident field (which is calculated by the program in this inverse formulation of the problem) is expressed in terms of the Stokes parameters as

$$S_0 = |E^{\mathrm{f}}_{0_+}|^2 + |E^{\mathrm{f}}_{0_-}|^2, \qquad S_1 = 2\,\mathrm{Re}[E^{\mathrm{f}*}_{0_+} E^{\mathrm{f}}_{0_-}],$$
$$S_3 = |E^{\mathrm{f}}_{0_+}|^2 - |E^{\mathrm{f}}_{0_-}|^2, \qquad S_2 = 2\,\mathrm{Im}[E^{\mathrm{f}*}_{0_+} E^{\mathrm{f}}_{0_-}].$$

However, the direct interpretation of these quantities in terms of squared Volts per square metres is sometimes somewhat inconvenient; therefore, those parameters can be scaled to give an interpretation of the intensity (in regular SI units measured in Watts per square metres), as $S'_k = (\varepsilon_0 c/2)S_k$, or explicitly

$$S'_0 = (\varepsilon_0 c/2)[|E^{\mathrm{f}}_{0_+}|^2 + |E^{\mathrm{f}}_{0_-}|^2], \qquad S'_1 = (\varepsilon_0 c/2)2\,\mathrm{Re}[E^{\mathrm{f}*}_{0_+} E^{\mathrm{f}}_{0_-}],$$
$$S'_3 = (\varepsilon_0 c/2)[|E^{\mathrm{f}}_{0_+}|^2 - |E^{\mathrm{f}}_{0_-}|^2], \qquad S'_2 = (\varepsilon_0 c/2)2\,\mathrm{Im}[E^{\mathrm{f}*}_{0_+} E^{\mathrm{f}}_{0_-}].$$

In this representation, $S'_0$ is now identical to the incident intensity $I_{\mathrm{in}}$ [W/m$^2$]. In order to have those scaled Stokes parameters $S'_k$ written to file, rather than the default ones, one convenient possibility is to use the previously added `--scale_stokesparams` option, to include `--scale_stokesparams 1.327209e-3` at the command line when invoking the program. The numerical value of this scaling was obtained from†

$$\begin{aligned}\varepsilon_0 c/2 &= (8.854187817\ldots \times 10^{-12}\ \mathrm{F/m}) \times (2.99792458 \times 10^8\ \mathrm{m/s})/2\\ &\approx 1.327209 \times 10^{-3}\ \mathrm{F/s}.\end{aligned}$$

```
⟨ Write Stokes parameters and reflection coefficients to file 83 ⟩ ≡
  {   /* Scan the grating spatially for the maximum intra-grating intensity. */
      /* This is performed with the original, unscaled Stokes parameters. */
    if (intensityinfo) {
      for (j = 0; j ≤ nn; j++) {
        tmp = (epsilon0 * n[j] * c/2) * (cabs2(efp[j]) + cabs2(efm[j]));
        if (maxintens < tmp) {
          maxintens = tmp;
          maxintens_layer = j;
          maxintens_inintens = (epsilon0 * nsurr * c/2) * (cabs2(efp[0]) + cabs2(efm[0]));
          maxintens_inellip = (cabs2(efp[0]) − cabs2(efm[0]))/(cabs2(efp[0]) + cabs2(efm[0]));
          maxintens_trintens = (epsilon0 * nsurr * c/2) * (cabs2(efp[nn]) + cabs2(efm[nn]));
          maxintens_trellip = (cabs2(efp[nn]) − cabs2(efm[nn]))/(cabs2(efp[nn]) + cabs2(efm[nn]));
        }
      }
    }
    if ((mme > 1) ∧ (mmi > 1)) {   /* "topological" mode */
        /* Stokes parameters of input optical wave */
      s0 = cabs2(efp[0]) + cabs2(efm[0]);
      s1 = 2.0 * cmul(conjg(efp[0]), efm[0]).r;
```

† In regular SI units, the physical dimension of the quantity $\varepsilon_0 c/2$ is [F/s] = [C/(V · s)], so the physical dimension of $(\varepsilon_0 c/2)S_k$ is easily verified as [C/(V · s)][V$^2$/m$^2$] = [C · V/(m$^2$ · s)] = [J/(m$^2$ · s)] = [W/m$^2$], as expected for a physical quantity describing an energy flow per unit area.

```
s2 = 2.0 * cmul(conjg(efp[0]), efm[0]).i;
s3 = cabs2(efp[0]) - cabs2(efm[0]);
if (scale_stokesparams) {
  s0 = s0 * stoke_scalefactor;
  s1 = s1 * stoke_scalefactor;
  s2 = s2 * stoke_scalefactor;
  s3 = s3 * stoke_scalefactor;
}
if (normalize_ellipticity) s3 = s3/s0;
fprintf(fp_s0, "%16.12e␣␣", s0);
fprintf(fp_s1, "%16.12e␣␣", s1);
fprintf(fp_s2, "%16.12e␣␣", s2);
fprintf(fp_s3, "%16.12e␣␣", s3);
fflush(fp_s0);
fflush(fp_s1);
fflush(fp_s2);
fflush(fp_s3);      /* Stokes parameters of reflected optical wave */
v0 = cabs2(ebp[0]) + cabs2(ebm[0]);
v1 = 2.0 * cmul(conjg(ebp[0]), ebm[0]).r;
v2 = 2.0 * cmul(conjg(ebp[0]), ebm[0]).i;
v3 = cabs2(ebp[0]) - cabs2(ebm[0]);
if (scale_stokesparams) {
  v0 = v0 * stoke_scalefactor;
  v1 = v1 * stoke_scalefactor;
  v2 = v2 * stoke_scalefactor;
  v3 = v3 * stoke_scalefactor;
}
if (normalize_ellipticity) v3 = v3/v0;
fprintf(fp_v0, "%16.12e␣␣", v0);
fprintf(fp_v1, "%16.12e␣␣", v1);
fprintf(fp_v2, "%16.12e␣␣", v2);
fprintf(fp_v3, "%16.12e␣␣", v3);
fflush(fp_v0);
fflush(fp_v1);
fflush(fp_v2);
fflush(fp_v3);      /* Stokes parameters of transmitted optical wave */
w0 = cabs2(efp[nn]) + cabs2(efm[nn]);
w1 = 2.0 * cmul(conjg(efp[nn]), efm[nn]).r;
w2 = 2.0 * cmul(conjg(efp[nn]), efm[nn]).i;
w3 = cabs2(efp[nn]) - cabs2(efm[nn]);
if (scale_stokesparams) {
  w0 = w0 * stoke_scalefactor;
  w1 = w1 * stoke_scalefactor;
  w2 = w2 * stoke_scalefactor;
  w3 = w3 * stoke_scalefactor;
}
if (normalize_ellipticity) w3 = w3/w0;
fprintf(fp_w0, "%16.12e␣␣", w0);
fprintf(fp_w1, "%16.12e␣␣", w1);
fprintf(fp_w2, "%16.12e␣␣", w2);
fprintf(fp_w3, "%16.12e␣␣", w3);
fflush(fp_w0);
```

```
    fflush(fp_w1);
    fflush(fp_w2);
    fflush(fp_w3);
  }
  else {      /* "spectrum" mode */
    if (stokes_parameter_spectrum) {      /* spectrum of Stokes parameters */
        /* Stokes parameters of input optical wave */
      s0 = cabs2(efp[0]) + cabs2(efm[0]);
      s1 = 2.0 * cmul(conjg(efp[0]), efm[0]).r;
      s2 = 2.0 * cmul(conjg(efp[0]), efm[0]).i;
      s3 = cabs2(efp[0]) - cabs2(efm[0]);
      fprintf(fp_spec, "%16.12e␣%16.12e␣%16.12e␣%16.12e␣%16.12e\n", lambda, omega, s1/s0, s2/s0,
          s3/s0);
      fflush(fp_spec);      /* Stokes parameters of reflected optical wave */
      v0 = cabs2(ebp[0]) + cabs2(ebm[0]);
      v1 = 2.0 * cmul(conjg(ebp[0]), ebm[0]).r;
      v2 = 2.0 * cmul(conjg(ebp[0]), ebm[0]).i;
      v3 = cabs2(ebp[0]) - cabs2(ebm[0]);      /* CODE FOR WRITING TO BE INSERTED HERE
          */      /* Stokes parameters of transmitted optical wave */
      w0 = cabs2(efp[nn]) + cabs2(efm[nn]);
      w1 = 2.0 * cmul(conjg(efp[nn]), efm[nn]).r;
      w2 = 2.0 * cmul(conjg(efp[nn]), efm[nn]).i;
      w3 = cabs2(efp[nn]) - cabs2(efm[nn]);
        /* CODE FOR WRITING TO BE INSERTED HERE */
    }
    if (save_dbspectra) {      /* spectrum in dB */
      tmp = (cabs2(ebp[0]) + cabs2(ebm[0]))/(cabs2(efp[0]) + cabs2(efm[0]));
      fprintf(fp_irspec, "%-10.8f␣%-10.8f\n", lambda * 1.0 · 10^9, 10.0 * log10(tmp));
      fprintf(fp_itspec, "%-10.8f␣%-10.8f\n", lambda * 1.0 · 10^9, 10.0 * log10(1.0 - tmp));
      fflush(fp_irspec);
      fflush(fp_itspec);
    }
    else {      /* linear scale between zero and unity */
      fprintf(fp_irspec, "%-10.8f␣%-10.8f\n", lambda * 1.0 · 10^9,
          (cabs2(ebp[0]) + cabs2(ebm[0]))/(cabs2(efp[0]) + cabs2(efm[0])));
      fprintf(fp_itspec, "%-10.8f␣%-10.8f\n", lambda * 1.0 · 10^9,
          (cabs2(efp[nn]) + cabs2(efm[nn]))/(cabs2(efp[0]) + cabs2(efm[0])));
      fflush(fp_irspec);
      fflush(fp_itspec);
    }
    fprintf(fp_icspec,
        "%-10.8f␣%-10.8f\n", lambda * 1.0 · 10^9, (cabs2(ebp[0]) + cabs2(ebm[0]))/(cabs2(efp[0]) +
        cabs2(efm[0])) + (cabs2(efp[nn]) + cabs2(efm[nn]))/(cabs2(efp[0]) + cabs2(efm[0])));
    fflush(fp_icspec);
  }
  if (ki ≥ mmi) {      /* Write linefeed at and of each scan of intensity */
    if ((mme > 1) ∧ (mmi > 1)) {
      fprintf(fp_s0, "\n");
      fprintf(fp_s1, "\n");
      fprintf(fp_s2, "\n");
      fprintf(fp_s3, "\n");
      fprintf(fp_v0, "\n");
```

```
        fprintf(fp_v1, "\n");
        fprintf(fp_v2, "\n");
        fprintf(fp_v3, "\n");
        fprintf(fp_w0, "\n");
        fprintf(fp_w1, "\n");
        fprintf(fp_w2, "\n");
        fprintf(fp_w3, "\n");
      }
    }
  }
```

This code is used in section 75.

**84.** Check if the user has specified a filename for saving the electromagnetic field as it propagates through the grating structure, with *nne* being the number of discrete sampling points within each homogeneous layer. (This is quite useful since we otherwise just would get samples of the intra grating electromagnetic field at the boundaries of the homogeneous layers of the model; for twolevel grating types with comparatively thick layers this would otherwise, for example, cause unwanted discrete jumps in the plots of the polarization state evolution in the grating structure.) If so, write the full information of the electromagnetic field to file, to be used for later graphs (not necessarily just in terms of Stokes parameters).

⟨ Write intragrating field evolution to file 84 ⟩ ≡

```
{
  if (fieldevoflag) {
    if (fieldevoflag_efield) {
      if (verbose) fprintf(stdout, "Writing␣spatial␣field␣evolution␣to␣file.\n");
      if (fp_evo ≠ Λ) {
        for (j = 1; j ≤ nn − 1; j++) {
          for (jje = 1; jje ≤ nne; jje++) {
            if (nne > 1) {
              zt = z[j] + ((double)(jje − 1)) ∗ dz[j]/((double)(nne));
            }
            else {
              zt = z[j];
            }
            tmpfp = cmul(efp[j], crexpi(omega ∗ (etafp[j] + g[j]) ∗ (zt − z[j])/c));
            tmpfm = cmul(efm[j], crexpi(omega ∗ (etafm[j] − g[j]) ∗ (zt − z[j])/c));
            tmpbp = cmul(ebp[j], crexpi(−omega ∗ (etabp[j] − g[j]) ∗ (zt − z[j])/c));
            tmpbm = cmul(ebm[j], crexpi(−omega ∗ (etabm[j] + g[j]) ∗ (zt − z[j])/c));
            if (normalize_length_to_micrometer) {
              fprintf(fp_evo, "%16.12e␣%16.12e␣%16.12e␣%16.12e" "␣%16.12e␣%16.12e␣%1\
                  6.12e␣%16.12e" "␣%16.12e\n", zt ∗ 1.0 · 10^6, tmpfp.r, tmpfp.i, tmpfm.r, tmpfm.i,
                  tmpbp.r, tmpbp.i, tmpbm.r, tmpbm.i);
            }
            else {
              fprintf(fp_evo, "%16.12e␣%16.12e␣%16.12e␣%16.12e" "␣%16.12e␣%16.12e␣%1\
                  6.12e␣%16.12e" "␣%16.12e\n", zt, tmpfp.r, tmpfp.i, tmpfm.r, tmpfm.i, tmpbp.r,
                  tmpbp.i, tmpbm.r, tmpbm.i);
            }
          }
        }
        if (normalize_length_to_micrometer) {
          fprintf(fp_evo, "%16.12e␣%16.12e␣%16.12e␣%16.12e␣%16.12e" "␣%16.12e␣%16.12e␣%1\
              6.12e␣%16.12e\n", z[nn] ∗ 1.0 · 10^6, efp[nn].r, efp[nn].i, efm[nn].r, efm[nn].i, ebp[nn].r,
              ebp[nn].i, ebm[nn].r, ebm[nn].i);
        }
        else {
          fprintf(fp_evo, "%16.12e␣%16.12e␣%16.12e␣%16.12e␣%16.12e" "␣%16.12e␣%16.12e␣%1\
              6.12e␣%16.12e\n", z[nn], efp[nn].r, efp[nn].i, efm[nn].r, efm[nn].i, ebp[nn].r, ebp[nn].i,
              ebm[nn].r, ebm[nn].i);
        }
      }
      else {
        fprintf(stderr, "%s:␣Could␣not␣write␣to␣file␣%s!\n", progname, fieldevofilename);
        exit(FAILURE);
      }
```

```
}
else if (fieldevoflag_stoke) {
  if (verbose)
    fprintf(stdout, "Writing␣spatial␣evolution␣of␣Stokes␣parameters␣to␣file.\n");
  if ((fp_evo_s0 ≠ Λ) ∧ (fp_evo_s1 ≠ Λ) ∧ (fp_evo_s2 ≠ Λ) ∧ (fp_evo_s3 ≠ Λ)) {
    for (j = 1; j ≤ nn − 1; j++) {
      for (jje = 1; jje ≤ nne; jje++) {
        if (nne > 1) {
          zt = z[j] + ((double)(jje − 1)) ∗ dz[j]/((double)(nne));
        }
        else {
          zt = z[j];
        }
        tmpfp = cmul(efp[j], crexpi(omega ∗ (etafp[j] + g[j]) ∗ (zt − z[j])/c));
        tmpfm = cmul(efm[j], crexpi(omega ∗ (etafm[j] − g[j]) ∗ (zt − z[j])/c));
        tmpbp = cmul(ebp[j], crexpi(−omega ∗ (etabp[j] − g[j]) ∗ (zt − z[j])/c));
        tmpbm = cmul(ebm[j], crexpi(−omega ∗ (etabm[j] + g[j]) ∗ (zt − z[j])/c));
        s0 = cabs2(tmpfp) + cabs2(tmpfm);
        s1 = 2.0 ∗ cmul(conjg(tmpfp), tmpfm).r;
        s2 = 2.0 ∗ cmul(conjg(tmpfp), tmpfm).i;
        s3 = cabs2(tmpfp) − cabs2(tmpfm);
        if (normalize_intensity) s0 = s0/(cabs2(efp[1]) + cabs2(efm[1]));
        if (normalize_length_to_micrometer) {
          fprintf(fp_evo_s0, "%16.12e␣%16.12e\n", zt ∗ 1.0 · 10^6, s0);
          fprintf(fp_evo_s1, "%16.12e␣%16.12e\n", zt ∗ 1.0 · 10^6, s1);
          fprintf(fp_evo_s2, "%16.12e␣%16.12e\n", zt ∗ 1.0 · 10^6, s2);
          fprintf(fp_evo_s3, "%16.12e␣%16.12e\n", zt ∗ 1.0 · 10^6, s3);
        }
        else {
          fprintf(fp_evo_s0, "%16.12e␣%16.12e\n", zt, s0);
          fprintf(fp_evo_s1, "%16.12e␣%16.12e\n", zt, s1);
          fprintf(fp_evo_s2, "%16.12e␣%16.12e\n", zt, s2);
          fprintf(fp_evo_s3, "%16.12e␣%16.12e\n", zt, s3);
        }
      }
    }
    s0 = cabs2(efp[nn]) + cabs2(efm[nn]);
    s1 = 2.0 ∗ cmul(conjg(efp[nn]), efm[nn]).r;
    s2 = 2.0 ∗ cmul(conjg(efp[nn]), efm[nn]).i;
    s3 = cabs2(efp[nn]) − cabs2(efm[nn]);
    if (normalize_intensity) s0 = s0/(cabs2(efp[1]) + cabs2(efm[1]));
    if (normalize_length_to_micrometer) {
      fprintf(fp_evo_s0, "%16.12e␣%16.12e\n", z[nn] ∗ 1.0 · 10^6, s0);
      fprintf(fp_evo_s1, "%16.12e␣%16.12e\n", z[nn] ∗ 1.0 · 10^6, s1);
      fprintf(fp_evo_s2, "%16.12e␣%16.12e\n", z[nn] ∗ 1.0 · 10^6, s2);
      fprintf(fp_evo_s3, "%16.12e␣%16.12e\n", z[nn] ∗ 1.0 · 10^6, s3);
    }
    else {
      fprintf(fp_evo_s0, "%16.12e␣%16.12e\n", z[nn], s0);
      fprintf(fp_evo_s1, "%16.12e␣%16.12e\n", z[nn], s1);
      fprintf(fp_evo_s2, "%16.12e␣%16.12e\n", z[nn], s2);
      fprintf(fp_evo_s3, "%16.12e␣%16.12e\n", z[nn], s3);
```

```
          }
        }
        else {
          fprintf(stderr, "%s:␣Could␣not␣write␣to␣file␣%s,␣%s,␣%s,␣or␣%s!\n", progname,
              fieldevofilename_s0, fieldevofilename_s1, fieldevofilename_s2, fieldevofilename_s3);
          exit(FAILURE);
        }
      }
      else {
        fprintf(stderr, "%s:␣Unknown␣field␣evolution␣flag.\n" "%s:␣(This␣cannot␣happen)\n",
            progname, progname);
        exit(FAILURE);
      }
    }
  }
```

This code is used in section 75.

**85.** Check if the user has specified a filename for saving the electromagnetic intensity distribution inside the grating structure, with *nne* being the number of discrete sampling points within each homogeneous layer.

⟨ Write intragrating intensity evolution to file 85 ⟩ ≡

```
{
  if (intensityevoflag) {
    if (fabs(ievolambda − lambda) < fabs(lambdastop − lambdastart)/((double)(mm))) {
      if (verbose)
        fprintf(stdout, "%s:␣Saving␣intensity␣evolution␣at␣lambda=%8.4e\n", progname, lambda);
      if (strcmp(intensityevofilename, "")) {
        if (fp_ievo ≠ Λ) {
          for (j = 1; j ≤ nn − 1; j++) {
            if (normalize_length_to_micrometer) {
              fprintf(fp_ievo,
                  "%16.12e␣%16.12e\n", z[j] ∗ 1.0 · 10^6, (cdabs(efp[j]) ∗ cdabs(efp[j]) + cdabs(efm[j]) ∗
                  cdabs(efm[j]))/(cdabs(efp[1]) ∗ cdabs(efp[1]) + cdabs(efm[1]) ∗ cdabs(efm[1])));
              fprintf(fp_ievo, "%16.12e␣%16.12e\n", z[j + 1] ∗ 1.0 · 10^6,
                  (cdabs(efp[j]) ∗ cdabs(efp[j]) + cdabs(efm[j]) ∗ cdabs(efm[j]))/(cdabs(efp[1]) ∗
                  cdabs(efp[1]) + cdabs(efm[1]) ∗ cdabs(efm[1])));
            }
            else {
              fprintf(fp_ievo, "%16.12e␣%16.12e\n", z[j], (cdabs(efp[j]) ∗ cdabs(efp[j]) + cdabs(efm[j]) ∗
                  cdabs(efm[j]))/(cdabs(efp[1]) ∗ cdabs(efp[1]) + cdabs(efm[1]) ∗ cdabs(efm[1])));
              fprintf(fp_ievo, "%16.12e␣%16.12e\n", z[j + 1],
                  (cdabs(efp[j]) ∗ cdabs(efp[j]) + cdabs(efm[j]) ∗ cdabs(efm[j]))/(cdabs(efp[1]) ∗
                  cdabs(efp[1]) + cdabs(efm[1]) ∗ cdabs(efm[1])));
            }
          }
        }
        else {
          fprintf(stderr, "%s:␣Could␣not␣write␣to␣file␣%s!\n", progname, intensityevofilename);
          exit(FAILURE);
        }
      }
    }
  }
}
```

This code is used in section 75.

**86.** Check if the user has specified a filename for saving the spatial grating structure to. If so, save the whole grating structure to disk; this is useful as reference in graphs of the intragrating intensity and polarization state evolution.

```
⟨ Write spatial grating structure to file 86 ⟩ ≡
  {
    if (writegratingtofile) {
      if ((fp_gr = fopen(gratingfilename, "w")) ≡ Λ) {
        fprintf(stderr, "%s:␣Could␣not␣open␣file␣%s␣for␣output!\n", progname, gratingfilename);
        exit(FAILURE);
      }
      if (normalize_length_to_micrometer) {      /* length z in micrometer */
        if (display_surrounding_media) {
          fprintf(fp_gr, "%16.12e␣%16.12e␣%16.12e␣%16.12e""␣%16.12e␣%16.12e␣%16.12e\n",
                $(z[1] - 0.1 * (z[nn] - z[1])) * 1.0 \cdot 10^6$, nsurr, 0.0, 0.0, 0.0, 0.0, 0.0);
          fprintf(fp_gr, "%16.12e␣%16.12e␣%16.12e␣%16.12e""␣%16.12e␣%16.12e␣%16.12e\n",
                $z[1] * 1.0 \cdot 10^6$, nsurr, 0.0, 0.0, 0.0, 0.0, 0.0);
        }
        for (j = 1; j ≤ nn − 1; j++) {
          fprintf(fp_gr, "%16.12e␣%16.12e␣%16.12e␣%16.12e""␣%16.12e␣%16.12e␣%16.12e\n",
                $z[j] * 1.0 \cdot 10^6$, n[j], g[j], pe[j], pm[j], qe[j], qm[j]);
          fprintf(fp_gr, "%16.12e␣%16.12e␣%16.12e␣%16.12e""␣%16.12e␣%16.12e␣%16.12e\n",
                $z[j + 1] * 1.0 \cdot 10^6$, n[j], g[j], pe[j], pm[j], qe[j], qm[j]);
        }
        if (display_surrounding_media) {
          fprintf(fp_gr, "%16.12e␣%16.12e␣%16.12e␣%16.12e""␣%16.12e␣%16.12e␣%16.12e\n",
                $z[nn] * 1.0 \cdot 10^6$, nsurr, 0.0, 0.0, 0.0, 0.0, 0.0);
          fprintf(fp_gr, "%16.12e␣%16.12e␣%16.12e␣%16.12e""␣%16.12e␣%16.12e␣%16.12e\n",
                $(z[nn] + 0.1 * (z[nn] - z[1])) * 1.0 \cdot 10^6$, nsurr, 0.0, 0.0, 0.0, 0.0, 0.0);
        }
      }
      else {      /* length z in meter */
        if (display_surrounding_media) {
          fprintf(fp_gr, "%16.12e␣%16.12e␣%16.12e␣%16.12e""␣%16.12e␣%16.12e␣%16.12e\n",
                z[1] − 0.1 * (z[nn] − z[1]), nsurr, 0.0, 0.0, 0.0, 0.0, 0.0);
          fprintf(fp_gr, "%16.12e␣%16.12e␣%16.12e␣%16.12e""␣%16.12e␣%16.12e␣%16.12e\n", z[1],
                nsurr, 0.0, 0.0, 0.0, 0.0, 0.0);
        }
        for (j = 1; j ≤ nn − 1; j++) {
          fprintf(fp_gr, "%16.12e␣%16.12e␣%16.12e␣%16.12e""␣%16.12e␣%16.12e␣%16.12e\n", z[j],
                n[j], g[j], pe[j], pm[j], qe[j], qm[j]);
          fprintf(fp_gr, "%16.12e␣%16.12e␣%16.12e␣%16.12e""␣%16.12e␣%16.12e␣%16.12e\n", z[j + 1],
                n[j], g[j], pe[j], pm[j], qe[j], qm[j]);
        }
        if (display_surrounding_media) {
          fprintf(fp_gr, "%16.12e␣%16.12e␣%16.12e␣%16.12e""␣%16.12e␣%16.12e␣%16.12e\n", z[nn],
                nsurr, 0.0, 0.0, 0.0, 0.0, 0.0);
          fprintf(fp_gr, "%16.12e␣%16.12e␣%16.12e␣%16.12e""␣%16.12e␣%16.12e␣%16.12e\n",
                z[nn] + 0.1 * (z[nn] − z[1]), nsurr, 0.0, 0.0, 0.0, 0.0, 0.0);
        }
      }
      fclose(fp_gr);
    }
  }
```

This code is used in section 75.

**87.** Display the maximum optical intensity detected in the grating.

⟨ Print information on maximum optical intensity in grating 87 ⟩ ≡

```
{
  if (intensityinfo) {
    for (k = 1; k ≤ 64; k++) fprintf(stdout, (k < 64 ? "-" : "\n"));
    fprintf(stdout, "Summary␣of␣intra-grating␣intensities:\n");
    fprintf(stdout, "The␣maximum␣intensity␣%1.4e␣[W/m^2]␣(%1.4f␣GW/cm^2)␣was␣detected\n",
        maxintens, maxintens ∗ 1.0 · 10^{−13});
    fprintf(stdout, "in␣layer␣%ld.␣The␣maximum␣intensity␣was␣detected␣at␣a␣transmitted\n",
        maxintens_layer);
    fprintf(stdout, "intensity␣of␣%1.4e␣[W/m^2]␣(%1.4f␣GW/cm^2),␣and␣at␣a␣transmitted\n",
        maxintens_trintens, maxintens_trintens ∗ 1.0 · 10^{−13});
    fprintf(stdout, "normalized␣ellipticity␣of␣S3/S0=%1.4f.␣", maxintens_trellip);
    fprintf(stdout, "(where␣S3/S0=1␣for␣LCP,␣and␣-1\n""for␣RCP).");
    fprintf(stdout, "At␣this␣state,␣the␣calculated␣optical␣intensity␣incident␣to␣the\n");
    fprintf(stdout, "crystal␣was␣%1.4e␣[W/m^2]␣(%1.4f␣GW/cm^2),␣or␣%1.1f␣percent\n",
        maxintens_inintens, maxintens_inintens ∗ 1.0 · 10^{−13}, 100.0 ∗ maxintens_inintens/maxintens);
    fprintf(stdout, "of␣the␣maximum␣intra-grating␣optical␣intensity.\n");
    fprintf(stdout, "The␣calculated␣normalized␣incident␣ellipticity␣was␣%1.4f.\n",
        maxintens_inellip);
    fprintf(stdout, "The␣intensity␣transmission␣was␣for␣this␣state␣%1.1f␣percent.\n",
        100.0 ∗ maxintens_trintens/maxintens_inintens);
    for (k = 1; k ≤ 64; k++) fprintf(stdout, (k < 64 ? "-" : "\n"));
    if (saveintensityinfologfile) {      /∗ also save log to file ∗/
      if ((intensinfologfile = fopen(intensinfologfilename, "w")) ≡ Λ) {
        fprintf(stderr, "%s:␣Could␣not␣open␣%s␣for␣intensity␣log!\n", progname,
            intensinfologfilename);
        exit(FAILURE);
      }
      for (k = 1; k ≤ 32; k++) fprintf(intensinfologfile, (k < 32 ? "-" : "\n"));
      fprintf(intensinfologfile, "Summary␣of␣intra-grating␣intensities:\n");
      fprintf(intensinfologfile,
          "Maximum␣intensity␣%1.4e␣[W/sq.m]␣(%1.4f␣GW/sq.cm)␣was␣detected\n", maxintens,
          maxintens ∗ 1.0 · 10^{−13});
      fprintf(intensinfologfile,
          "in␣layer␣%ld.␣Maximum␣intensity␣was␣detected␣at␣a␣transmitted\n",
          maxintens_layer);
      fprintf(intensinfologfile,
          "intensity␣of␣%1.4e␣[W/sq.m]␣(%1.4f␣GW/sq.cm),␣and␣at␣transmitted\n",
          maxintens_trintens, maxintens_trintens ∗ 1.0 · 10^{−13});
      fprintf(intensinfologfile, "normalized␣ellipticity␣of␣S3/S0=%1.4f.␣", maxintens_trellip);
      fprintf(intensinfologfile, "(where␣S3/S0=1␣for␣LCP,␣and␣-1\n""for␣RCP).␣");
      fprintf(intensinfologfile,
          "At␣this␣state,␣the␣calculated␣optical␣intensity␣incident␣to␣the\n");
      fprintf(intensinfologfile,
          "crystal␣was␣%1.4e␣[W/sq.m]␣(%1.4f␣GW/sq.cm),␣or␣%1.1f␣percent\n",
          maxintens_inintens, maxintens_inintens ∗ 1.0 · 10^{−13}, 100.0 ∗ maxintens_inintens/maxintens);
      fprintf(intensinfologfile, "of␣the␣maximum␣intra-grating␣optical␣intensity.\n");
      fprintf(intensinfologfile, "The␣calculated␣normalized␣incident␣ellipticity␣was␣%1.4f.\n",
          maxintens_inellip);
      fprintf(intensinfologfile,
          "The␣intensity␣transmission␣was␣for␣this␣state␣%1.1f␣percent.\n",
          100.0 * maxintens_trintens/maxintens_inintens);
      fclose(intensinfologfile);
      for (k = 1; k ≤ 32; k++) fprintf(intensinfologfile, (k < 32 ? "-" : "\n"));
    }
  }
}
```

This code is used in section 45.

**88. Routine for checking for numerical characters.** The *numeric*( ) routine takes one character *ch* as argument, and returns 1 ("true") if the character is a sign or numeric, otherwise 0 ("false") is returned. Notice that numerical fields of the form ".314159" are not allowed or recognized here.

```
⟨ Routine for checking for numerical characters 88 ⟩ ≡
  int numeric(char ch)
  {
    if ((ch ≡ '0') ∨ (ch ≡ '1') ∨ (ch ≡ '2') ∨ (ch ≡ '3') ∨ (ch ≡ '4') ∨ (ch ≡ '5') ∨ (ch ≡ '6') ∨ (ch ≡
          '7') ∨ (ch ≡ '8') ∨ (ch ≡ '9') ∨ (ch ≡ '+') ∨ (ch ≡ '-')) {
      return (1);    /* yes, this is a sign or numeric character */
    }
    else {
      return (0);    /* no, this is not a sign or numeric character */
    }
  }
```

This code is used in section 50.

**89. Routine for initialization of Cantor type fractal gratings.**

```
⟨ Routine for initialization of Cantor type fractal gratings 89 ⟩ ≡
  void init_cantor_fractal_grating(double *z, int indxmin, int indxmax, double llmin, double
          llmax, double n1, double n2)
  {
    int indxmid;
    double dll;

    if (indxmax − indxmin ≡ 1) {
      z[indxmin] = llmin;
      z[indxmax] = llmax;
    }
    else if (indxmax − indxmin ≥ 3) {
      indxmid = (indxmin + indxmax)/2;
      dll = (n2/(n1 + 2 ∗ n2)) ∗ (llmax − llmin);
      init_cantor_fractal_grating(z, indxmin, indxmid, llmin, llmin + dll, n1, n2);
      init_cantor_fractal_grating(z, indxmid + 1, indxmax, llmax − dll, llmax, n1, n2);
    }
    else {
      fprintf(stderr, "%s:␣Error␣in␣indexes␣detected␣in␣routine␣%s", progname,
          "init_cantor_fractal_grating()!\n");
      fprintf(stderr, "%s:␣Please␣verify␣that␣the␣number␣of␣discrete␣layers\n", progname);
      fprintf(stderr, "%s:␣in␣the␣grating␣is␣N-1,␣where␣N␣is␣an␣integer""␣power␣of␣2.\n",
          progname);
      exit(FAILURE);
    }
  }
```

This code is used in section 50.

**90. Routines for removing preceding path of filenames.** In this block all routines related to removing preceding path strings go. Not really fancy programming, and no contribution to any increase of numerical efficiency or precision; just for the sake of keeping a tidy terminal output of the program. The *strip_away_path*( ) routine is typically called when initializing the program name string *progname* from the command line string *argv*[0], and is typically located in the blocks related to parsing of the command line options.

```
⟨ Routines for removing preceding path of filenames 90 ⟩ ≡
  ⟨ Routine for checking valid path characters 91 ⟩
  ⟨ Routine for stripping away path string 92 ⟩
```

This code is used in section 50.

**91.** Checking for a valid path character. The *pathcharacter* routine takes one character *ch* as argument, and returns 1 ("true") if the character is valid character of a path string, otherwise 0 ("false") is returned. The *isalnum* requires `ctype.h`.

```
⟨ Routine for checking valid path characters 91 ⟩ ≡
  short pathcharacter(int ch)
  {
    return (isalnum(ch) ∨ (ch ≡ '.') ∨ (ch ≡ '/') ∨ (ch ≡ '\\') ∨ (ch ≡ '_') ∨ (ch ≡ '-') ∨ (ch ≡ '+'));
  }
```

This code is used in section 90.

**92.** Stripping path string from a file name. The *strip_away_path* routine takes a character string *filename* as argument, and returns a pointer to the same string but without any preceding path segments. This routine is, for example, useful for removing paths from program names as parsed from the command line.

```
⟨ Routine for stripping away path string 92 ⟩ ≡
  char *strip_away_path(char filename[ ])
  {
    int j, k = 0;
    while (pathcharacter(filename[k])) k++;
    j = (−−k);    /* this is the uppermost index of the full path+file string */
    while (isalnum((int)(filename[j]))) j−−;
    j++;    /* this is the lowermost index of the stripped file name */
    return (&filename[j]);
  }
```

This code is used in section 90.

**93. Routines for generation of random numbers.** The *ran1*( ) routine is taken from Numerical Recipes in C, 2nd ed. (Cambridge University Press, New York, 1994).

```
⟨ Routines for generation of random numbers 93 ⟩ ≡
#define IA 16807
#define IM 2147483647
#define AM (1.0/IM)
#define IQ 127773
#define IR 2836
#define NTAB 32
#define NDIV (1 + (IM − 1)/NTAB)
#define EPS 1.2 · 10^−7
#define RNMX (1.0 − EPS)
  float ran1(long *idum)
  {
    int j;
    long k;
    static long iy = 0;
    static long iv[NTAB];
    float temp;
    if (*idum ≤ 0 ∨ ¬iy) {
      if (−(*idum) < 1) *idum = 1;
      else *idum = −(*idum);
      for (j = NTAB + 7; j ≥ 0; j−−) {
        k = (*idum)/IQ;
        *idum = IA * (*idum − k * IQ) − IR * k;
        if (*idum < 0) *idum += IM;
        if (j < NTAB) iv[j] = *idum;
      }
      iy = iv[0];
    }
    k = (*idum)/IQ;
    *idum = IA * (*idum − k * IQ) − IR * k;
    if (*idum < 0) *idum += IM;
    j = iy/NDIV;
    iy = iv[j];
    iv[j] = *idum;
    if ((temp = AM * iy) > RNMX) return RNMX;
    else return temp;
  }
#undef IA
#undef IM
#undef AM
#undef IQ
#undef IR
#undef NTAB
#undef NDIV
#undef EPS
#undef RNMX
```

This code is used in section 50.

**94. Routines for doing complex arithmetics.** By using the data structure **dcomplex**, which here is the basic construct for complex numbers, containing real and imaginary parts in double precision, several routines for doing arithmetics with this representation have been implemented.

```
⟨ Routines for complex arithmetics 94 ⟩ ≡
  ⟨ Complex number 95 ⟩
  ⟨ Complex conjugation 96 ⟩
  ⟨ Absolute value of complex number 97 ⟩
  ⟨ Squared absolute value of complex number 98 ⟩
  ⟨ Complex addition 99 ⟩
  ⟨ Complex subtraction 100 ⟩
  ⟨ Complex multiplication 101 ⟩
  ⟨ Complex division 104 ⟩
  ⟨ Complex square root 107 ⟩
  ⟨ Complex exponentiation 108 ⟩
```

This code is used in section 50.

**95.** The function *complex*$(a, b)$ takes two real valued arguments $a$ and $b$ as input, and returns the complex number $c = a + ib$. This basic construct is useful for storing temporary results, and for internal use in routines dealing with complex arithmetics.

```
⟨ Complex number 95 ⟩ ≡
  dcomplex complex(double re, double im)
  {
    dcomplex c;

    c.r = re;
    c.i = im;
    return c;
  }
```

This code is used in section 94.

**96.** The function *conjg*$(z)$ takes a complex valued argument $z$ of type **dcomplex** as input, and returns the complex conjugate $\bar{z} = \mathrm{Re}(z) - i\,\mathrm{Im}(z)$.

```
⟨ Complex conjugation 96 ⟩ ≡
  dcomplex conjg(dcomplex z)
  {
    dcomplex c;

    c.r = z.r;
    c.i = −z.i;
    return c;
  }
```

This code is used in section 94.

**97.** The function *cdabs*$(z)$ takes a complex valued argument $z$ as input, and returns the absolute value $|z| = [\mathrm{Re}(z)^2 + \mathrm{Im}(z)^2]^{1/2}$.

```
⟨ Absolute value of complex number 97 ⟩ ≡
  double cdabs(dcomplex z)
  {
    double x, y, c, tmp;
    x = fabs(z.r);
    y = fabs(z.i);
    if (x ≡ 0.0) c = y;
    else if (y ≡ 0.0) c = x;
    else if (x > y) {
      tmp = y/x;
      c = x ∗ sqrt(1.0 + tmp ∗ tmp);
    }
    else {
      tmp = x/y;
      c = y ∗ sqrt(1.0 + tmp ∗ tmp);
    }
    return c;
  }
```

This code is used in section 94.

**98.** The function *cabs2*$(z)$ takes a complex valued argument $z$ as input, and returns the squared absolute value $|z|^2 = \mathrm{Re}(z)^2 + \mathrm{Im}(z)^2$. This function is identical to the *cdabs*$(z)$ function, with the exception that the squared result is returned.

```
⟨ Squared absolute value of complex number 98 ⟩ ≡
  double cabs2(dcomplex z)
  {
    double x, y, c, tmp;
    x = fabs(z.r);
    y = fabs(z.i);
    if (x ≡ 0.0) c = y ∗ y;
    else if (y ≡ 0.0) c = x ∗ x;
    else if (x > y) {
      tmp = y/x;
      c = x ∗ x ∗ (1.0 + tmp ∗ tmp);
    }
    else {
      tmp = x/y;
      c = y ∗ y ∗ (1.0 + tmp ∗ tmp);
    }
    return c;
  }
```

This code is used in section 94.

**99.** The function *cadd*$(a,b)$ takes complex valued arguments $a$ and $b$ as inputs, and returns the complex valued sum $a+b$.

```
⟨ Complex addition 99 ⟩ ≡
  dcomplex cadd(dcomplex a, dcomplex b)
  {
    dcomplex c;
    c.r = a.r + b.r;
    c.i = a.i + b.i;
    return c;
  }
```

This code is used in section 94.

**100.** The function *csub*$(a,b)$ takes complex valued arguments $a$ and $b$ as inputs, and returns the complex valued difference $a-b$.

```
⟨ Complex subtraction 100 ⟩ ≡
  dcomplex csub(dcomplex a, dcomplex b)
  {
    dcomplex c;
    c.r = a.r − b.r;
    c.i = a.i − b.i;
    return c;
  }
```

This code is used in section 94.

**101.** When dealing with multiplication involving complex numbers, the case where one of the numbers is entirely real is distinguished from the general multiplication by two complex valued numbers, in order to speed up the multiplication somewhat.

```
⟨ Complex multiplication 101 ⟩ ≡
  ⟨ Multiplication by two complex numbers 102 ⟩
  ⟨ Multiplication by real and complex numbers 103 ⟩
```

This code is used in section 94.

**102.** The function *cmul*$(a,b)$ takes complex valued arguments $a$ and $b$ as inputs, and returns the complex valued product $ab$.

```
⟨ Multiplication by two complex numbers 102 ⟩ ≡
  dcomplex cmul(dcomplex a, dcomplex b)
  {
    dcomplex c;
    c.r = a.r * b.r − a.i * b.i;
    c.i = a.i * b.r + a.r * b.i;
    return c;
  }
```

This code is used in section 101.

**103.** The function *rcmul*$(a, b)$ takes a real valued argument $a$ and a complex valued arguments $b$ as inputs, and returns the complex valued product $ab = a\,\mathrm{Re}(b) + ia\,\mathrm{Im}(b)$.

```
⟨ Multiplication by real and complex numbers 103 ⟩ ≡
  dcomplex rcmul(double a, dcomplex b)
  {
    dcomplex c;
    c.r = a * b.r;
    c.i = a * b.i;
    return c;
  }
```

This code is used in section 101.

**104.** When dealing with division involving complex numbers, the case where the denominator is entirely real is distinguished from the general division by two complex valued numbers, in order to speed up the division somewhat.

```
⟨ Complex division 104 ⟩ ≡
  ⟨ Division by complex numbers 105 ⟩
  ⟨ Division by complex and real number 106 ⟩
```

This code is used in section 94.

**105.** The function *cdiv*(*a*, *b*) takes complex valued arguments *a* and *b* as inputs, and returns the complex valued quote *a*/*b*. If the denominator is found to be real, this routine speed up the division in similar to the *crdiv* routine, by instead applying the rules of real-valued division.

⟨ Division by complex numbers 105 ⟩ ≡
```
dcomplex cdiv(dcomplex a, dcomplex b)
{
  dcomplex c;
  double r, den;

  if (cdabs(b) > 0.0) {
    if (fabs(b.i) ≡ 0.0) {
      c.r = a.r/b.r;
      c.i = a.i/b.r;
    }
    else {
      if (fabs(b.r) ≥ fabs(b.i)) {
        r = b.i/b.r;
        den = b.r + r * b.i;
        c.r = (a.r + r * a.i)/den;
        c.i = (a.i − r * a.r)/den;
      }
      else {
        r = b.r/b.i;
        den = b.i + r * b.r;
        c.r = (a.r * r + a.i)/den;
        c.i = (a.i * r − a.r)/den;
      }
    }
  }
  else {
    fprintf(stderr, "Error␣in␣cdiv():␣Division␣by␣zero!\n");
    exit(FAILURE);
  }
  return c;
}
```
This code is used in section 104.

**106.** The function *crdiv*$(a,b)$ takes a complex valued argument $a$ of and a real valued argument $b$ as inputs, and returns the complex valued quote $a/b = \mathrm{Re}(a)/b + i\,\mathrm{Im}(a)/b$.

```
⟨ Division by complex and real number 106 ⟩ ≡
  dcomplex crdiv(dcomplex a, double b)
  {
    dcomplex c;
    if (fabs(b) > 0.0) {
      c.r = a.r/b;
      c.i = a.i/b;
    }
    else {
      fprintf(stderr, "Error␣in␣crdiv():␣Division␣by␣zero!\n");
      exit(FAILURE);
    }
    return c;
  }
```

This code is used in section 104.

**107.** The function *csqrt*$(z)$ takes a complex valued argument $z$ as input and returns the complex valued square root $z^{1/2}$.

```
⟨ Complex square root 107 ⟩ ≡
  dcomplex csqrt(dcomplex z)
  {
    dcomplex c;
    double x, y, w, r;
    if ((z.r ≡ 0.0) ∧ (z.i ≡ 0.0)) {
      c.r = 0.0;
      c.i = 0.0;
      return c;
    }
    else {
      x = fabs(z.r);
      y = fabs(z.i);
      if (x ≥ y) {
        r = y/x;
        w = sqrt(x) * sqrt(0.5 * (1.0 + sqrt(1.0 + r * r)));
      }
      else {
        r = x/y;
        w = sqrt(y) * sqrt(0.5 * (r + sqrt(1.0 + r * r)));
      }
      if (z.r ≥ 0.0) {
        c.r = w;
        c.i = 0.5 * z.i/w;
      }
      else {
        c.i = ((z.i ≥ 0) ? w : −w);
        c.r = 0.5 * z.i/c.i;
      }
      return c;
    }
  }
```

This code is used in section 94.

**108.** For complex exponentiation, we distinguish between the case when the argument is complex valued, with nonzero real and imaginary parts, and the case where the argument is entirely imaginary. In the former case, the *cexp* routine should be used, while in the latter case, the *crexpi* routine is more appropriate.

```
⟨ Complex exponentiation 108 ⟩ ≡
  ⟨ Exponentiation by complex number 109 ⟩
  ⟨ Exponentiation by imaginary number 110 ⟩
```

This code is used in section 94.

**109.** The function *cexp*(*a*) takes a complex valued argument $a$ as input and returns the complex valued exponential $\exp(a) = \exp(\mathrm{Re}(a))[\cos(\mathrm{Im}(a)) + i\sin(\mathrm{Im}(a))]$.

```
⟨ Exponentiation by complex number 109 ⟩ ≡
  dcomplex cexp(dcomplex a)
  {
    dcomplex c;
    double tmp = exp(a.r);
    c.r = tmp * cos(a.i);
    c.i = tmp * sin(a.i);
    return c;
  }
```

This code is used in section 108.

**110.** The function *crexpi*(*a*) takes a real valued argument $a$ as input and returns the complex valued exponential $\exp(ia) = \cos(a) + i\sin(a)$.

```
⟨ Exponentiation by imaginary number 110 ⟩ ≡
  dcomplex crexpi(double a)
  {
    dcomplex c;
    c.r = cos(a);
    c.i = sin(a);
    return c;
  }
```

This code is used in section 108.

**111. Subroutines for memory allocation.** Here follows the routines for memory allocation and deallocation of double precision real and complex valued vectors, as used for storing the optical field distribution in the grating, the refractive index distribution of the grating, etc.

```
⟨ Routines for memory allocation of vectors 111 ⟩ ≡
  ⟨ Allocation of real-valued vectors 112 ⟩
  ⟨ Allocation of complex-valued vectors 113 ⟩
  ⟨ Deallocation of real-valued vectors 114 ⟩
  ⟨ Deallocation of complex-valued vectors 115 ⟩
```

This code is used in section 50.

**112.** The *dvector* routine allocate a real-valued vector of double precision, with vector index ranging from *nl* to *nh*, while the *dcvector* routine allocate a complex-valued vector of double precision, with vector index ranging from *nl* to *nh*.

```
⟨ Allocation of real-valued vectors 112 ⟩ ≡
  double *dvector(long nl, long nh)
  {
    double *v;
    v = (double *) malloc((size_t)((nh − nl + 2) * sizeof(double)));
    if (¬v) {
      fprintf(stderr, "Error:␣Allocation␣failure␣in␣dvector()\n");
      exit(FAILURE);
    }
    return v − nl + 1;
  }
```

This code is used in section 111.

**113.** The *dcvector* routine allocate a complex-valued vector of double precision, with vector index ranging from *nl* to *nh*. The basic building type of the obtained complex valued vector is the globally declared **dcomplex** data structure.

```
⟨ Allocation of complex-valued vectors 113 ⟩ ≡
  dcomplex *dcvector(long nl, long nh)
  {
    dcomplex *v;
    v = (dcomplex *) malloc((size_t)((nh − nl + 2) * sizeof(dcomplex)));
    if (¬v) {
      fprintf(stderr, "Error:␣Allocation␣failure␣in␣dcvector()\n");
      exit(FAILURE);
    }
    return v − nl + 1;
  }
```

This code is used in section 111.

**114.** The *free_dvector* routine release the memory occupied by the real-valued vector $v[nl \mathinner{..} nh]$.

```
⟨ Deallocation of real-valued vectors 114 ⟩ ≡
  void free_dvector(double *v, long nl, long nh)
  {
    free((char *)(v + nl − 1));
  }
```

This code is used in section 111.

**115.** The *free_dcvector* routine release the memory occupied by the complex-valued vector $v[nl \mathrel{..} nh]$.

⟨ Deallocation of complex-valued vectors 115 ⟩ ≡

```
void free_dcvector(dcomplex *v, long nl, long nh)
{
  free((char *)(v + nl - 1));
}
```

This code is used in section 111.

**116. Parsing command line options.** All input parameters are passed to the program through command line options and arguments to the program. The syntax of command line options is listed whenever the program is invoked without any options, or whenever any of the `--help` or `-h` options are specified at startup.

```
⟨ Parse command line for parameter values 116 ⟩ ≡
  {
    progname = strip_away_path(argv[0]);
    fprintf(stdout, "This␣is␣%s␣v.%s.␣%s\n", progname, VERSION, COPYRIGHT);
    no_arg = argc;
    while (−− argc) {
      if (¬strcmp(argv[no_arg − argc], "-o") ∨ ¬strcmp(argv[no_arg − argc], "--outputfile")) {
        −− argc;
        strcpy(outfilename, argv[no_arg − argc]);
      }
      else if (¬strcmp(argv[no_arg − argc], "-w") ∨ ¬strcmp(argv[no_arg − argc], "--writegratingfile"))
        {
        −− argc;
        strcpy(gratingfilename, argv[no_arg − argc]);
        writegratingtofile = 1;
      }
      else if (¬strcmp(argv[no_arg − argc], "--displaysurrmedia")) {
        display_surrounding_media = (display_surrounding_media ? 0 : 1);
      }
      else if (¬strcmp(argv[no_arg − argc], "-a") ∨ ¬strcmp(argv[no_arg − argc], "--apodize")) {
        ⟨ Parse apodization options 118 ⟩;
      }
      else if (¬strcmp(argv[no_arg − argc], "--phasejump")) {
        ⟨ Parse phase jump options 119 ⟩;
      }
      else if (¬strcmp(argv[no_arg − argc], "--fieldevolution")) {
        ⟨ Parse field evolution saving options 121 ⟩;
      }
      else if (¬strcmp(argv[no_arg − argc], "--intensityevolution")) {
        −− argc;
        if (¬sscanf(argv[no_arg − argc], "%lf", &ievolambda)) {
          fprintf(stderr, "%s:␣Error␣in␣'--intensityevolution'␣option.\n", progname);
          exit(FAILURE);
        }
        −− argc;
        strcpy(intensityevofilename, argv[no_arg − argc]);
        intensityevoflag = 1;
      }
      else if (¬strcmp(argv[no_arg − argc], "--spectrumfile")) {
        −− argc;
        strcpy(spectrumfilename, argv[no_arg − argc]);
      }
      else if (¬strcmp(argv[no_arg − argc], "--stokesspectrum")) {
        stokes_parameter_spectrum = 1;
      }
      else if (¬strcmp(argv[no_arg − argc], "--trmtraject")) {
        −− argc;
        strcpy(trmtraject_filename, argv[no_arg − argc]);
        trmtraject_specified = 1;
      }
      else if (¬strcmp(argv[no_arg − argc], "--intensityspectrumfile")) {
        −− argc;
        strcpy(intensity_reflection_spectrumfilename, argv[no_arg − argc]);
        strcpy(intensity_transmission_spectrumfilename, argv[no_arg − argc]);
        strcpy(intensity_check_spectrumfilename, argv[no_arg − argc]);
        sprintf(intensity_reflection_spectrumfilename, "%s.irsp.dat",
             intensity_reflection_spectrumfilename);
        sprintf(intensity_transmission_spectrumfilename, "%s.itsp.dat",
             intensity_transmission_spectrumfilename);
        sprintf(intensity_check_spectrumfilename, "%s.chec.dat", intensity_check_spectrumfilename);
      }
      else if (¬strcmp(argv[no_arg − argc], "--logarithmicspectrum")) {
        save_dbspectra = 1;
      }
      else if (¬strcmp(argv[no_arg − argc], "-v") ∨ ¬strcmp(argv[no_arg − argc], "--verbose")) {
        verbose = (verbose ? 0 : 1);
      }
      else if (¬strcmp(argv[no_arg − argc], "--scale_stokesparams")) {
        scale_stokesparams = 1;
        −− argc;
        if (¬sscanf(argv[no_arg − argc], "%lf", &stoke_scalefactor)) {
          fprintf(stderr, "%s:␣Error␣in␣'--scale_stokesparams'␣option.\n", 39progname);
          exit(FAILURE);
        }
      }
      else if (¬strcmp(argv[no_arg − argc], "--intensityinfo")) {
        intensityinfo = (intensityinfo ? 0 : 1);
      }
      else if (¬strcmp(argv[no_arg − argc], "--intensityinfologfile")) {
        intensityinfo = 1;
        saveintensityinfologfile = 1;
        −− argc;
        strcpy(intensinfologfilename, argv[no_arg − argc]);
      }
      else if (¬strcmp(argv[no_arg − argc], "--gyroperturb")) {
        ⟨ Parse gyration constant perturbation options 120 ⟩;
      }
      else if (¬strcmp(argv[no_arg − argc], "--normalize_length_to_um")) {
        normalize_length_to_micrometer = (normalize_length_to_micrometer ? 0 : 1);
      }
      else if (¬strcmp(argv[no_arg − argc], "--normalize_intensity")) {
        normalize_intensity = (normalize_intensity ? 0 : 1);
      }
      else if (¬strcmp(argv[no_arg − argc], "--normalize_ellipticity")) {
        normalize_ellipticity = (normalize_ellipticity ? 0 : 1);
      }
      else if (¬strcmp(argv[no_arg − argc], "--normalizedrepresentation")) {
        normalize_internally = (normalize_internally ? 0 : 1);
      }
      else if (¬strcmp(argv[no_arg − argc], "-r") ∨ ¬strcmp(argv[no_arg − argc], "--random")) {
```

```
  randomdistribution = (randomdistribution ? 0 : 1);
}
else if (¬strcmp(argv[no_arg − argc], "-h") ∨ ¬strcmp(argv[no_arg − argc], "--help")) {
  showsomehelp( );
}
else if (¬strcmp(argv[no_arg − argc], "-g") ∨ ¬strcmp(argv[no_arg − argc], "--grating")) {
  −− argc;
  if (¬strcmp(argv[no_arg − argc], "stepwise")) {
    ⟨ Parse for stepwise grating options 122 ⟩
  }
  else if (¬strcmp(argv[no_arg − argc], "sinusoidal")) {
    ⟨ Parse for sinusoidal grating options 123 ⟩
  }
  else if (¬strcmp(argv[no_arg − argc], "chirped")) {
    ⟨ Parse for chirped grating options 124 ⟩
  }
  else if (¬strcmp(argv[no_arg − argc], "fractal")) {
    ⟨ Parse for fractal grating options 125 ⟩
  }
  else {
    fprintf(stderr, "%s:␣Error␣in␣'-g'␣or␣'--grating'␣option.\n", progname);
    fprintf(stderr, "%s:␣(No␣valid␣grating␣type␣found!)\n", progname);
    exit(FAILURE);
  }
}
else if ((¬strcmp(argv[no_arg − argc], "-L")) ∨ (¬strcmp(argv[no_arg − argc], "--gratinglength")))
  {
  −− argc;
  if (¬sscanf(argv[no_arg − argc], "%lf", &ll)) {
    fprintf(stderr, "%s:␣Error␣in␣'-L'␣option.\n", progname);
    exit(FAILURE);
  }
}
else if (¬strcmp(argv[no_arg − argc], "--modifylayer")) {
  ⟨ Parse for options for modified layer of grating structure 126 ⟩
}
else if (¬strcmp(argv[no_arg − argc], "--refindsurr")) {
  −− argc;
  if (¬sscanf(argv[no_arg − argc], "%lf", &nsurr)) {
    fprintf(stderr, "%s:␣Error␣in␣'--refindsurr'␣option.\n", progname);
    exit(FAILURE);
  }
}
else if (¬strcmp(argv[no_arg − argc], "--trmintensity")) {
  ⟨ Parse the command line for transmitted intensity range 127 ⟩
}
else if (¬strcmp(argv[no_arg − argc], "--trmellipticity")) {
  ⟨ Parse the command line for transmitted ellipticity range 128 ⟩
}
else if (¬strcmp(argv[no_arg − argc], "--lambdastart")) {
  −− argc;
  if (¬sscanf(argv[no_arg − argc], "%lf", &lambdastart)) {
        fprintf(stderr, "%s:␣Error␣in␣'--lambdastart'␣option.\n", progname);
        exit(FAILURE);
      }
    }
    else if (¬strcmp(argv[no_arg − argc], "--lambdastop")) {
      −− argc;
      if (¬sscanf(argv[no_arg − argc], "%lf", &lambdastop)) {
        fprintf(stderr, "%s:␣Error␣in␣'--lambdastop'␣option.\n", progname);
        exit(FAILURE);
      }
    }
    else if (¬strcmp(argv[no_arg − argc], "--wlenlinz")) {
      chirpflag = 1;
    }
    else if (¬strcmp(argv[no_arg − argc], "--freqlinz")) {
      chirpflag = 0;
    }
    else if (¬strcmp(argv[no_arg − argc], "-N")) {
      −− argc;
      if (¬sscanf(argv[no_arg − argc], "%ld", &nn)) {
        fprintf(stderr, "%s:␣Error␣in␣'-N'␣option.\n", progname);
        exit(FAILURE);
      }
    }
    else if (¬strcmp(argv[no_arg − argc], "-M")) {
      −− argc;
      if (¬sscanf(argv[no_arg − argc], "%ld", &mm)) {
        fprintf(stderr, "%s:␣Error␣in␣'-M'␣option.\n", progname);
        exit(FAILURE);
      }
    }
    else {
      fprintf(stderr, "%s:␣Specified␣option␣invalid!\n", progname);
      showsomehelp();
      exit(FAILURE);
    }
  }
  ⟨ Create outfile suffixes 117 ⟩
  ⟨ Display parameters parsed from the command line 129 ⟩
}
```

This code is used in section 45.

**117.** Create suffixes for output filenames. The output files for the Stokes parameters are named according to the convention that....

```
⟨ Create outfile suffixes 117 ⟩ ≡
  {
    sprintf(outfilename_s0, "%s.s0.dat", outfilename);
    sprintf(outfilename_s1, "%s.s1.dat", outfilename);
    sprintf(outfilename_s2, "%s.s2.dat", outfilename);
    sprintf(outfilename_s3, "%s.s3.dat", outfilename);
    sprintf(outfilename_v0, "%s.v0.dat", outfilename);
    sprintf(outfilename_v1, "%s.v1.dat", outfilename);
    sprintf(outfilename_v2, "%s.v2.dat", outfilename);
    sprintf(outfilename_v3, "%s.v3.dat", outfilename);
    sprintf(outfilename_w0, "%s.w0.dat", outfilename);
    sprintf(outfilename_w1, "%s.w1.dat", outfilename);
    sprintf(outfilename_w2, "%s.w2.dat", outfilename);
    sprintf(outfilename_w3, "%s.w3.dat", outfilename);
  }
```

This code is used in section 116.

**118.** Parse the command line for options related to apodization [15] of the spatial modulation of the refractive index and gyration coefficient.

```
⟨ Parse apodization options 118 ⟩ ≡
  {
    apodize = 1;
    −− argc;
    if (¬sscanf(argv[no_arg − argc], "%lf", &apolength)) {
      fprintf(stderr, "%s:␣Error␣in␣'--apodize'␣option.\n", progname);
      exit(FAILURE);
    }
  }
```

This code is used in section 116.

**119.** Parse the command line for options related to discrete phase jump in the spatial modulation of the refractive index and gyration coefficient.

```
⟨ Parse phase jump options 119 ⟩ ≡
  {
    phasejump = 1;
    −− argc;
    if (¬sscanf(argv[no_arg − argc], "%lf", &phasejumpangle)) {
      fprintf(stderr, "%s:␣Error␣in␣1st␣arg␣of␣'--phasejump'␣option.\n", progname);
      exit(FAILURE);
    }
    −− argc;
    if (¬sscanf(argv[no_arg − argc], "%lf", &phasejumpposition)) {
      fprintf(stderr, "%s:␣Error␣in␣2nd␣arg␣of␣'--phasejump'␣option.\n", progname);
      exit(FAILURE);
    }
  }
```

This code is used in section 116.

**120.** Parse the command line for options related to a Lorentzian perturbation of the magneto-optical gyration coefficient of the grating.

```
⟨ Parse gyration constant perturbation options 120 ⟩ ≡
  {
    perturbed_gyration_constant = 1;
    -- argc;
    if (¬sscanf(argv[no_arg − argc], "%lf", &gyroperturb_position)) {
      fprintf(stderr, "%s:␣Error␣in␣'--gyroperturb'␣option.\n", progname);
      exit(FAILURE);
    }
    -- argc;
    if (¬sscanf(argv[no_arg − argc], "%lf", &gyroperturb_amplitude)) {
      fprintf(stderr, "%s:␣Error␣in␣'--gyroperturb'␣option.\n", progname);
      exit(FAILURE);
    }
    -- argc;
    if (¬sscanf(argv[no_arg − argc], "%lf", &gyroperturb_width)) {
      fprintf(stderr, "%s:␣Error␣in␣'--gyroperturb'␣option.\n", progname);
      exit(FAILURE);
    }
  }
```

This code is used in section 116.

**121.** Parse the command line for options related to saving the intra grating optical field spatial evolution to file. If Stokes parameters are preferred for the output of the spatial intra-grating field evolution, the specified filename will be used as the base name for the output file; the suffixes `.s0.dat`, `.s1.dat`, `.s2.dat`, and `.s3.dat` will then be appended to the base name in order to keep track of them.

⟨ Parse field evolution saving options 121 ⟩ ≡

```
{
  --argc;
  if (¬strcmp(argv[no_arg − argc], "efield")) {
    fieldevoflag_efield = 1;
  }
  else if (¬strcmp(argv[no_arg − argc], "stoke")) {
    fieldevoflag_stoke = 1;
  }
  else {
    fprintf(stderr, "%s:␣Unknown␣field␣evolution␣flag␣'%s'␣in␣second␣argument␣of\
        \n""␣--fieldevolution␣option.\n", progname, argv[no_arg − argc]);
    exit(FAILURE);
  }
  --argc;
  if (¬sscanf(argv[no_arg − argc], "%ld", &nne)) {
    fprintf(stderr, "%s:␣Error␣in␣'--fieldevolution'␣option.\n", progname);
    exit(FAILURE);
  }
  --argc;
  strcpy(fieldevofilename, argv[no_arg − argc]);
  fieldevoflag = 1;      /* Indicate that the field evolution shold be saved */
  if (fieldevoflag_stoke) {      /* Stokes parameter output preferred */
    sprintf(fieldevofilename_s0, "%s.s0.dat", fieldevofilename);
    sprintf(fieldevofilename_s1, "%s.s1.dat", fieldevofilename);
    sprintf(fieldevofilename_s2, "%s.s2.dat", fieldevofilename);
    sprintf(fieldevofilename_s3, "%s.s3.dat", fieldevofilename);
  }
}
```

This code is used in section 116.

**122.** Parse the command line for options related to the initiation of a stepwise grating, consisting of a set of stacked layers.

```
⟨ Parse for stepwise grating options 122 ⟩ ≡
  {
    strcpy(gratingtype, argv[no_arg − argc]);
    −− argc;
    if (¬strcmp(argv[no_arg − argc], "twolevel")) {
      strcpy(gratingsubtype, argv[no_arg − argc]);
      −− argc;
      if (strcmp(argv[no_arg − argc], "t1")) {
        fprintf(stderr, "%s:␣Error␣in␣'twolevel'␣option.\n", progname);
        fprintf(stderr, "%s:␣(Expecting␣string␣'t1').\n", progname);
        exit(FAILURE);
      }
      −− argc;
      if (¬sscanf(argv[no_arg − argc], "%lf", &t1)) {
        fprintf(stderr, "%s:␣Error␣in␣'twolevel'␣option.\n", progname);
        fprintf(stderr, "%s:␣(Could␣not␣read␣data␣for␣t1).\n", progname);
        exit(FAILURE);
      }
      −− argc;
      if (strcmp(argv[no_arg − argc], "t2")) {
        fprintf(stderr, "%s:␣Error␣in␣'twolevel'␣option.\n", progname);
        fprintf(stderr, "%s:␣(Expecting␣string␣'t2').\n", progname);
        exit(FAILURE);
      }
      −− argc;
      if (¬sscanf(argv[no_arg − argc], "%lf", &t2)) {
        fprintf(stderr, "%s:␣Error␣in␣'twolevel'␣option.\n", progname);
        fprintf(stderr, "%s:␣(Could␣not␣read␣data␣for␣t2).\n", progname);
        exit(FAILURE);
      }
      −− argc;
      if (strcmp(argv[no_arg − argc], "n1")) {
        fprintf(stderr, "%s:␣Error␣in␣'twolevel'␣option.\n", progname);
        fprintf(stderr, "%s:␣(Expecting␣string␣'n1').\n", progname);
        exit(FAILURE);
      }
      −− argc;
      if (¬sscanf(argv[no_arg − argc], "%lf", &n1)) {
        fprintf(stderr, "%s:␣Error␣in␣'twolevel'␣option.\n", progname);
        fprintf(stderr, "%s:␣(Could␣not␣read␣data␣for␣n1).\n", progname);
        exit(FAILURE);
      }
      −− argc;
      if (strcmp(argv[no_arg − argc], "n2")) {
        fprintf(stderr, "%s:␣Error␣in␣'twolevel'␣option.\n", progname);
        fprintf(stderr, "%s:␣(Expecting␣string␣'n2').\n", progname);
        exit(FAILURE);
      }
      −− argc;
      if (¬sscanf(argv[no_arg − argc], "%lf", &n2)) {
```

```
    fprintf(stderr, "%s:␣Error␣in␣'twolevel'␣option.\n", progname);
    fprintf(stderr, "%s:␣(Could␣not␣read␣data␣for␣n2).\n", progname);
    exit(FAILURE);
  }
  −− argc;
  if (strcmp(argv[no_arg − argc], "g1")) {
    fprintf(stderr, "%s:␣Error␣in␣'twolevel'␣option.\n", progname);
    fprintf(stderr, "%s:␣(Expecting␣string␣'g1').\n", progname);
    exit(FAILURE);
  }
  −− argc;
  if (¬sscanf(argv[no_arg − argc], "%lf", &g1)) {
    fprintf(stderr, "%s:␣Error␣in␣'twolevel'␣option.\n", progname);
    fprintf(stderr, "%s:␣(Could␣not␣read␣data␣for␣g1).\n", progname);
    exit(FAILURE);
  }
  −− argc;
  if (strcmp(argv[no_arg − argc], "g2")) {
    fprintf(stderr, "%s:␣Error␣in␣'twolevel'␣option.\n", progname);
    fprintf(stderr, "%s:␣(Expecting␣string␣'g2').\n", progname);
    exit(FAILURE);
  }
  −− argc;
  if (¬sscanf(argv[no_arg − argc], "%lf", &g2)) {
    fprintf(stderr, "%s:␣Error␣in␣'twolevel'␣option.\n", progname);
    fprintf(stderr, "%s:␣(Could␣not␣read␣data␣for␣g2).\n", progname);
    exit(FAILURE);
  }
  −− argc;
  if (strcmp(argv[no_arg − argc], "pe1")) {
    fprintf(stderr, "%s:␣Error␣in␣'twolevel'␣option.\n", progname);
    fprintf(stderr, "%s:␣(Expecting␣string␣'pe1').\n", progname);
    exit(FAILURE);
  }
  −− argc;
  if (¬sscanf(argv[no_arg − argc], "%lf", &pe1)) {
    fprintf(stderr, "%s:␣Error␣in␣'twolevel'␣option.\n", progname);
    fprintf(stderr, "%s:␣(Could␣not␣read␣data␣for␣pe1).\n", progname);
    exit(FAILURE);
  }
  −− argc;
  if (strcmp(argv[no_arg − argc], "pe2")) {
    fprintf(stderr, "%s:␣Error␣in␣'twolevel'␣option.\n", progname);
    fprintf(stderr, "%s:␣(Expecting␣string␣'pe2').\n", progname);
    exit(FAILURE);
  }
  −− argc;
  if (¬sscanf(argv[no_arg − argc], "%lf", &pe2)) {
    fprintf(stderr, "%s:␣Error␣in␣'twolevel'␣option.\n", progname);
    fprintf(stderr, "%s:␣(Could␣not␣read␣data␣for␣pe2).\n", progname);
    exit(FAILURE);
  }
```

```
−− argc;
if (strcmp(argv[no_arg − argc], "pm1")) {
  fprintf(stderr, "%s:␣Error␣in␣'twolevel'␣option.\n", progname);
  fprintf(stderr, "%s:␣(Expecting␣string␣'pm1').\n", progname);
  exit(FAILURE);
}
−− argc;
if (¬sscanf(argv[no_arg − argc], "%lf", &pm1)) {
  fprintf(stderr, "%s:␣Error␣in␣'twolevel'␣option.\n", progname);
  fprintf(stderr, "%s:␣(Could␣not␣read␣data␣for␣pm1).\n", 39progname);
  exit(FAILURE);
}
−− argc;
if (strcmp(argv[no_arg − argc], "pm2")) {
  fprintf(stderr, "%s:␣Error␣in␣'twolevel'␣option.\n", progname);
  fprintf(stderr, "%s:␣(Expecting␣string␣'pm2').\n", progname);
  exit(FAILURE);
}
−− argc;
if (¬sscanf(argv[no_arg − argc], "%lf", &pm2)) {
  fprintf(stderr, "%s:␣Error␣in␣'twolevel'␣option.\n", progname);
  fprintf(stderr, "%s:␣(Could␣not␣read␣data␣for␣pm2).\n", 39progname);
  exit(FAILURE);
}
−− argc;
if (strcmp(argv[no_arg − argc], "qe1")) {
  fprintf(stderr, "%s:␣Error␣in␣'twolevel'␣option.\n", progname);
  fprintf(stderr, "%s:␣(Expecting␣string␣'qe1').\n", progname);
  exit(FAILURE);
}
−− argc;
if (¬sscanf(argv[no_arg − argc], "%lf", &qe1)) {
  fprintf(stderr, "%s:␣Error␣in␣'twolevel'␣option.\n", progname);
  fprintf(stderr, "%s:␣(Could␣not␣read␣data␣for␣qe1).\n", 39progname);
  exit(FAILURE);
}
−− argc;
if (strcmp(argv[no_arg − argc], "qe2")) {
  fprintf(stderr, "%s:␣Error␣in␣'twolevel'␣option.\n", progname);
  fprintf(stderr, "%s:␣(Expecting␣string␣'qe2').\n", progname);
  exit(FAILURE);
}
−− argc;
if (¬sscanf(argv[no_arg − argc], "%lf", &qe2)) {
  fprintf(stderr, "%s:␣Error␣in␣'twolevel'␣option.\n", progname);
  fprintf(stderr, "%s:␣(Could␣not␣read␣data␣for␣qe2).\n", progname);
  exit(FAILURE);
}
−− argc;
if (strcmp(argv[no_arg − argc], "qm1")) {
  fprintf(stderr, "%s:␣Error␣in␣'twolevel'␣option.\n", progname);
  fprintf(stderr, "%s:␣(Expecting␣string␣'qm1').\n", progname);
```

```
          exit(FAILURE);
        }
        −− argc;
        if (¬sscanf(argv[no_arg − argc], "%lf", &qm1)) {
          fprintf(stderr, "%s:␣Error␣in␣'twolevel'␣option.\n", progname);
          fprintf(stderr, "%s:␣(Could␣not␣read␣data␣for␣qm1).\n", progname);
          exit(FAILURE);
        }
        −− argc;
        if (strcmp(argv[no_arg − argc], "qm2")) {
          fprintf(stderr, "%s:␣Error␣in␣'twolevel'␣option.\n", progname);
          fprintf(stderr, "%s:␣(Expecting␣string␣'qm2').\n", progname);
          exit(FAILURE);
        }
        −− argc;
        if (¬sscanf(argv[no_arg − argc], "%lf", &qm2)) {
          fprintf(stderr, "%s:␣Error␣in␣'twolevel'␣option.\n", progname);
          fprintf(stderr, "%s:␣(Could␣not␣read␣data␣for␣qm2).\n", progname);
          exit(FAILURE);
        }
      }
      else {
        fprintf(stderr, "%s:␣Error␣in␣'-g'␣or␣'--grating'␣option.\n", progname);
        fprintf(stderr, "%s:␣(No␣valid␣stepwise␣grating␣type␣found!)\n", progname);
        exit(FAILURE);
      }
    }
```

This code is used in section 116.

**123.** Parse the command line for options related to the initiation of a sinusoidal grating, consisting of a set of stacked layers.

⟨ Parse for sinusoidal grating options 123 ⟩ ≡

```
{
  strcpy(gratingtype, argv[no_arg − argc]);
  −− argc;
  if (strcmp(argv[no_arg − argc], "n")) {
    fprintf(stderr, "%s:␣Error␣in␣'sinusoidal'␣option.\n", progname);
    fprintf(stderr, "%s:␣(Expecting␣string␣'n').\n", progname);
    exit(FAILURE);
  }
  −− argc;
  if (¬sscanf(argv[no_arg − argc], "%lf", &n1)) {
    fprintf(stderr, "%s:␣Error␣in␣'sinusoidal'␣option.\n", progname);
    fprintf(stderr, "%s:␣(Could␣not␣read␣data␣for␣n␣[1st␣arg]).\n", progname);
    exit(FAILURE);
  }
  −− argc;
  if (¬sscanf(argv[no_arg − argc], "%lf", &n2)) {
    fprintf(stderr, "%s:␣Error␣in␣'sinusoidal'␣option.\n", progname);
    fprintf(stderr, "%s:␣(Could␣not␣read␣data␣for␣n␣[2nd␣arg]).\n", progname);
    exit(FAILURE);
  }
  −− argc;
  if (¬sscanf(argv[no_arg − argc], "%lf", &nper)) {
    fprintf(stderr, "%s:␣Error␣in␣'sinusoidal'␣option.\n", progname);
    fprintf(stderr, "%s:␣(Could␣not␣read␣data␣for␣n␣[3rd␣arg]).\n", progname);
    exit(FAILURE);
  }
  −− argc;
  if (strcmp(argv[no_arg − argc], "g")) {
    fprintf(stderr, "%s:␣Error␣in␣'sinusoidal'␣option.\n", progname);
    fprintf(stderr, "%s:␣(Expecting␣string␣'g').\n", progname);
    exit(FAILURE);
  }
  −− argc;
  if (¬sscanf(argv[no_arg − argc], "%lf", &g1)) {
    fprintf(stderr, "%s:␣Error␣in␣'sinusoidal'␣option.\n", progname);
    fprintf(stderr, "%s:␣(Could␣not␣read␣data␣for␣g␣[1st␣arg]).\n", progname);
    exit(FAILURE);
  }
  −− argc;
  if (¬sscanf(argv[no_arg − argc], "%lf", &g2)) {
    fprintf(stderr, "%s:␣Error␣in␣'sinusoidal'␣option.\n", progname);
    fprintf(stderr, "%s:␣(Could␣not␣read␣data␣for␣g␣[2nd␣arg]).\n", progname);
    exit(FAILURE);
  }
  −− argc;
  if (¬sscanf(argv[no_arg − argc], "%lf", &gper)) {
    fprintf(stderr, "%s:␣Error␣in␣'sinusoidal'␣option.\n", progname);
    fprintf(stderr, "%s:␣(Could␣not␣read␣data␣for␣g␣[3rd␣arg]).\n", progname);
    exit(FAILURE);
```

```
}
−− argc;
if (strcmp(argv[no_arg − argc], "pe")) {
  fprintf(stderr, "%s:␣Error␣in␣'sinusoidal'␣option.\n", progname);
  fprintf(stderr, "%s:␣(Expecting␣string␣'pe').\n", progname);
  exit(FAILURE);
}
−− argc;
if (¬sscanf(argv[no_arg − argc], "%lf", &pe1)) {
  fprintf(stderr, "%s:␣Error␣in␣'sinusoidal'␣option.\n", progname);
  fprintf(stderr, "%s:␣(Could␣not␣read␣data␣for␣pe␣[1st␣arg]).\n", progname);
  exit(FAILURE);
}
−− argc;
if (¬sscanf(argv[no_arg − argc], "%lf", &pe2)) {
  fprintf(stderr, "%s:␣Error␣in␣'sinusoidal'␣option.\n", progname);
  fprintf(stderr, "%s:␣(Could␣not␣read␣data␣for␣pe␣[2nd␣arg]).\n", progname);
  exit(FAILURE);
}
−− argc;
if (¬sscanf(argv[no_arg − argc], "%lf", &peper)) {
  fprintf(stderr, "%s:␣Error␣in␣'sinusoidal'␣option.\n", progname);
  fprintf(stderr, "%s:␣(Could␣not␣read␣data␣for␣pe␣[3rd␣arg]).\n", progname);
  exit(FAILURE);
}
−− argc;
if (strcmp(argv[no_arg − argc], "pm")) {
  fprintf(stderr, "%s:␣Error␣in␣'sinusoidal'␣option.\n", progname);
  fprintf(stderr, "%s:␣(Expecting␣string␣'pm').\n", progname);
  exit(FAILURE);
}
−− argc;
if (¬sscanf(argv[no_arg − argc], "%lf", &pm1)) {
  fprintf(stderr, "%s:␣Error␣in␣'sinusoidal'␣option.\n", progname);
  fprintf(stderr, "%s:␣(Could␣not␣read␣data␣for␣pm␣[1st␣arg]).\n", progname);
  exit(FAILURE);
}
−− argc;
if (¬sscanf(argv[no_arg − argc], "%lf", &pm2)) {
  fprintf(stderr, "%s:␣Error␣in␣'sinusoidal'␣option.\n", progname);
  fprintf(stderr, "%s:␣(Could␣not␣read␣data␣for␣pm␣[2nd␣arg]).\n", progname);
  exit(FAILURE);
}
−− argc;
if (¬sscanf(argv[no_arg − argc], "%lf", &pmper)) {
  fprintf(stderr, "%s:␣Error␣in␣'sinusoidal'␣option.\n", progname);
  fprintf(stderr, "%s:␣(Could␣not␣read␣data␣for␣pm␣[3rd␣arg]).\n", progname);
  exit(FAILURE);
}
−− argc;
if (strcmp(argv[no_arg − argc], "qe")) {
  fprintf(stderr, "%s:␣Error␣in␣'sinusoidal'␣option.\n", progname);
```

```
        fprintf(stderr, "%s:␣(Expecting␣string␣'qe').\n", progname);
        exit(FAILURE);
      }
      −− argc;
      if (¬sscanf(argv[no_arg − argc], "%lf", &qe1)) {
        fprintf(stderr, "%s:␣Error␣in␣'sinusoidal'␣option.\n", progname);
        fprintf(stderr, "%s:␣(Could␣not␣read␣data␣for␣qe␣[1st␣arg]).\n", progname);
        exit(FAILURE);
      }
      −− argc;
      if (¬sscanf(argv[no_arg − argc], "%lf", &qe2)) {
        fprintf(stderr, "%s:␣Error␣in␣'sinusoidal'␣option.\n", progname);
        fprintf(stderr, "%s:␣(Could␣not␣read␣data␣for␣qe␣[2nd␣arg]).\n", progname);
        exit(FAILURE);
      }
      −− argc;
      if (¬sscanf(argv[no_arg − argc], "%lf", &qeper)) {
        fprintf(stderr, "%s:␣Error␣in␣'sinusoidal'␣option.\n", progname);
        fprintf(stderr, "%s:␣(Could␣not␣read␣data␣for␣qe␣[3rd␣arg]).\n", progname);
        exit(FAILURE);
      }
      −− argc;
      if (strcmp(argv[no_arg − argc], "qm")) {
        fprintf(stderr, "%s:␣Error␣in␣'sinusoidal'␣option.\n", progname);
        fprintf(stderr, "%s:␣(Expecting␣string␣'qm').\n", progname);
        exit(FAILURE);
      }
      −− argc;
      if (¬sscanf(argv[no_arg − argc], "%lf", &qm1)) {
        fprintf(stderr, "%s:␣Error␣in␣'sinusoidal'␣option.\n", progname);
        fprintf(stderr, "%s:␣(Could␣not␣read␣data␣for␣qm␣[1st␣arg]).\n", progname);
        exit(FAILURE);
      }
      −− argc;
      if (¬sscanf(argv[no_arg − argc], "%lf", &qm2)) {
        fprintf(stderr, "%s:␣Error␣in␣'sinusoidal'␣option.\n", progname);
        fprintf(stderr, "%s:␣(Could␣not␣read␣data␣for␣qm␣[2nd␣arg]).\n", progname);
        exit(FAILURE);
      }
      −− argc;
      if (¬sscanf(argv[no_arg − argc], "%lf", &qmper)) {
        fprintf(stderr, "%s:␣Error␣in␣'sinusoidal'␣option.\n", progname);
        fprintf(stderr, "%s:␣(Could␣not␣read␣data␣for␣qm␣[3rd␣arg]).\n", progname);
        exit(FAILURE);
      }
    }
```

This code is used in section 116.

**124.** Parse the command line for options related to the initiation of a sinusoidal grating, consisting of a set of stacked layers.

```
⟨ Parse for chirped grating options 124 ⟩ ≡
  {
    strcpy(gratingtype, argv[no_arg − argc]);
    −− argc;
    if (strcmp(argv[no_arg − argc], "n")) {
      fprintf(stderr, "%s:␣Error␣in␣'chirped'␣grating␣option.\n", progname);
      fprintf(stderr, "%s:␣(Expecting␣string␣'n').\n", progname);
      exit(FAILURE);
    }
    −− argc;
    if (¬sscanf(argv[no_arg − argc], "%lf", &n1)) {
      fprintf(stderr, "%s:␣Error␣in␣'chirped'␣grating␣option.\n", progname);
      fprintf(stderr, "%s:␣(Could␣not␣read␣data␣for␣n␣[1st␣arg]).\n", progname);
      exit(FAILURE);
    }
    −− argc;
    if (¬sscanf(argv[no_arg − argc], "%lf", &n2)) {
      fprintf(stderr, "%s:␣Error␣in␣'chirped'␣grating␣option.\n", progname);
      fprintf(stderr, "%s:␣(Could␣not␣read␣data␣for␣n␣[2nd␣arg]).\n", progname);
      exit(FAILURE);
    }
    −− argc;
    if (¬sscanf(argv[no_arg − argc], "%lf", &nper)) {
      fprintf(stderr, "%s:␣Error␣in␣'chirped'␣grating␣option.\n", progname);
      fprintf(stderr, "%s:␣(Could␣not␣read␣data␣for␣n␣[3rd␣arg]).\n", progname);
      exit(FAILURE);
    }
    −− argc;
    if (¬sscanf(argv[no_arg − argc], "%lf", &ncrp)) {
      fprintf(stderr, "%s:␣Error␣in␣'chirped'␣grating␣option.\n", progname);
      fprintf(stderr, "%s:␣(Could␣not␣read␣data␣for␣n␣[4th␣arg]).\n", progname);
      exit(FAILURE);
    }
    −− argc;
    if (strcmp(argv[no_arg − argc], "g")) {
      fprintf(stderr, "%s:␣Error␣in␣'chirped'␣grating␣option.\n", progname);
      fprintf(stderr, "%s:␣(Expecting␣string␣'g').\n", progname);
      exit(FAILURE);
    }
    −− argc;
    if (¬sscanf(argv[no_arg − argc], "%lf", &g1)) {
      fprintf(stderr, "%s:␣Error␣in␣'chirped'␣grating␣option.\n", progname);
      fprintf(stderr, "%s:␣(Could␣not␣read␣data␣for␣g␣[1st␣arg]).\n", progname);
      exit(FAILURE);
    }
    −− argc;
    if (¬sscanf(argv[no_arg − argc], "%lf", &g2)) {
      fprintf(stderr, "%s:␣Error␣in␣'chirped'␣grating␣option.\n", progname);
      fprintf(stderr, "%s:␣(Could␣not␣read␣data␣for␣g␣[2nd␣arg]).\n", progname);
      exit(FAILURE);
    }
    −− argc;
    if (¬sscanf(argv[no_arg − argc], "%lf", &gper)) {
      fprintf(stderr, "%s:␣Error␣in␣'chirped'␣grating␣option.\n", progname);
      fprintf(stderr, "%s:␣(Could␣not␣read␣data␣for␣g␣[3rd␣arg]).\n", progname);
      exit(FAILURE);
    }
    −− argc;
    if (¬sscanf(argv[no_arg − argc], "%lf", &gcrp)) {
      fprintf(stderr, "%s:␣Error␣in␣'chirped'␣grating␣option.\n", progname);
      fprintf(stderr, "%s:␣(Could␣not␣read␣data␣for␣g␣[4th␣arg]).\n", progname);
      exit(FAILURE);
    }
    −− argc;
    if (strcmp(argv[no_arg − argc], "pe")) {
      fprintf(stderr, "%s:␣Error␣in␣'chirped'␣grating␣option.\n", progname);
      fprintf(stderr, "%s:␣(Expecting␣string␣'pe').\n", progname);
      exit(FAILURE);
    }
    −− argc;
    if (¬sscanf(argv[no_arg − argc], "%lf", &pe1)) {
      fprintf(stderr, "%s:␣Error␣in␣'chirped'␣grating␣option.\n", progname);
      fprintf(stderr, "%s:␣(Could␣not␣read␣data␣for␣pe␣[1st␣arg]).\n", progname);
      exit(FAILURE);
    }
    −− argc;
    if (¬sscanf(argv[no_arg − argc], "%lf", &pe2)) {
      fprintf(stderr, "%s:␣Error␣in␣'chirped'␣grating␣option.\n", progname);
      fprintf(stderr, "%s:␣(Could␣not␣read␣data␣for␣pe␣[2nd␣arg]).\n", progname);
      exit(FAILURE);
    }
    −− argc;
    if (¬sscanf(argv[no_arg − argc], "%lf", &peper)) {
      fprintf(stderr, "%s:␣Error␣in␣'chirped'␣grating␣option.\n", progname);
      fprintf(stderr, "%s:␣(Could␣not␣read␣data␣for␣pe␣[3rd␣arg]).\n", progname);
      exit(FAILURE);
    }
    −− argc;
    if (¬sscanf(argv[no_arg − argc], "%lf", &pecrp)) {
      fprintf(stderr, "%s:␣Error␣in␣'chirped'␣grating␣option.\n", progname);
      fprintf(stderr, "%s:␣(Could␣not␣read␣data␣for␣pe␣[4th␣arg]).\n", progname);
      exit(FAILURE);
    }
    −− argc;
    if (strcmp(argv[no_arg − argc], "pm")) {
      fprintf(stderr, "%s:␣Error␣in␣'chirped'␣grating␣option.\n", progname);
      fprintf(stderr, "%s:␣(Expecting␣string␣'pm').\n", progname);
      exit(FAILURE);
    }
    −− argc;
    if (¬sscanf(argv[no_arg − argc], "%lf", &pm1)) {
      fprintf(stderr, "%s:␣Error␣in␣'chirped'␣grating␣option.\n", progname);
      fprintf(stderr, "%s:␣(Could␣not␣read␣data␣for␣pm␣[1st␣arg]).\n", progname);
      exit(FAILURE);
    }
    −− argc;
    if (¬sscanf(argv[no_arg − argc], "%lf", &pm2)) {
      fprintf(stderr, "%s:␣Error␣in␣'chirped'␣grating␣option.\n", progname);
      fprintf(stderr, "%s:␣(Could␣not␣read␣data␣for␣pm␣[2nd␣arg]).\n", progname);
      exit(FAILURE);
    }
    −− argc;
    if (¬sscanf(argv[no_arg − argc], "%lf", &pmper)) {
      fprintf(stderr, "%s:␣Error␣in␣'chirped'␣grating␣option.\n", progname);
      fprintf(stderr, "%s:␣(Could␣not␣read␣data␣for␣pm␣[3rd␣arg]).\n", progname);
      exit(FAILURE);
    }
    −− argc;
    if (¬sscanf(argv[no_arg − argc], "%lf", &pmcrp)) {
      fprintf(stderr, "%s:␣Error␣in␣'chirped'␣grating␣option.\n", progname);
      fprintf(stderr, "%s:␣(Could␣not␣read␣data␣for␣pm␣[4th␣arg]).\n", progname);
      exit(FAILURE);
    }
    −− argc;
    if (strcmp(argv[no_arg − argc], "qe")) {
      fprintf(stderr, "%s:␣Error␣in␣'chirped'␣grating␣option.\n", progname);
      fprintf(stderr, "%s:␣(Expecting␣string␣'qe').\n", progname);
      exit(FAILURE);
    }
    −− argc;
    if (¬sscanf(argv[no_arg − argc], "%lf", &qe1)) {
      fprintf(stderr, "%s:␣Error␣in␣'chirped'␣grating␣option.\n", progname);
      fprintf(stderr, "%s:␣(Could␣not␣read␣data␣for␣qe␣[1st␣arg]).\n", progname);
      exit(FAILURE);
    }
    −− argc;
    if (¬sscanf(argv[no_arg − argc], "%lf", &qe2)) {
      fprintf(stderr, "%s:␣Error␣in␣'chirped'␣grating␣option.\n", progname);
      fprintf(stderr, "%s:␣(Could␣not␣read␣data␣for␣qe␣[2nd␣arg]).\n", progname);
      exit(FAILURE);
    }
    −− argc;
    if (¬sscanf(argv[no_arg − argc], "%lf", &qeper)) {
      fprintf(stderr, "%s:␣Error␣in␣'chirped'␣grating␣option.\n", progname);
      fprintf(stderr, "%s:␣(Could␣not␣read␣data␣for␣qe␣[3rd␣arg]).\n", progname);
      exit(FAILURE);
    }
    −− argc;
    if (¬sscanf(argv[no_arg − argc], "%lf", &qecrp)) {
      fprintf(stderr, "%s:␣Error␣in␣'chirped'␣grating␣option.\n", progname);
      fprintf(stderr, "%s:␣(Could␣not␣read␣data␣for␣qe␣[4th␣arg]).\n", progname);
      exit(FAILURE);
    }
    −− argc;
```

```
    if (strcmp(argv[no_arg − argc], "qm")) {
      fprintf(stderr, "%s:␣Error␣in␣'chirped'␣grating␣option.\n", progname);
      fprintf(stderr, "%s:␣(Expecting␣string␣'qm').\n", progname);
      exit(FAILURE);
    }
    −− argc;
    if (¬sscanf(argv[no_arg − argc], "%lf", &qm1)) {
      fprintf(stderr, "%s:␣Error␣in␣'chirped'␣grating␣option.\n", progname);
      fprintf(stderr, "%s:␣(Could␣not␣read␣data␣for␣qm␣[1st␣arg]).\n", progname);
      exit(FAILURE);
    }
    −− argc;
    if (¬sscanf(argv[no_arg − argc], "%lf", &qm2)) {
      fprintf(stderr, "%s:␣Error␣in␣'chirped'␣grating␣option.\n", progname);
      fprintf(stderr, "%s:␣(Could␣not␣read␣data␣for␣qm␣[2nd␣arg]).\n", progname);
      exit(FAILURE);
    }
    −− argc;
    if (¬sscanf(argv[no_arg − argc], "%lf", &qmper)) {
      fprintf(stderr, "%s:␣Error␣in␣'chirped'␣grating␣option.\n", progname);
      fprintf(stderr, "%s:␣(Could␣not␣read␣data␣for␣qm␣[3rd␣arg]).\n", progname);
      exit(FAILURE);
    }
    −− argc;
    if (¬sscanf(argv[no_arg − argc], "%lf", &qmcrp)) {
      fprintf(stderr, "%s:␣Error␣in␣'chirped'␣grating␣option.\n", progname);
      fprintf(stderr, "%s:␣(Could␣not␣read␣data␣for␣qm␣[4th␣arg]).\n", progname);
      exit(FAILURE);
    }
  }
```

This code is used in section 116.

**125.** Parse the command line for options related to the initiation of a fractal grating, consisting of a set of stacked homogeneous layers of thicknesses corresponding to a Cantor-set fractal.

```
⟨ Parse for fractal grating options 125 ⟩ ≡
  {
    strcpy(gratingtype, argv[no_arg − argc]);
    −− argc;
    if (¬strcmp(argv[no_arg − argc], "cantor")) {
      strcpy(gratingsubtype, argv[no_arg − argc]);
      −− argc;
      if (strcmp(argv[no_arg − argc], "fractal_level")) {
        fprintf(stderr, "%s:␣Error␣in␣'cantor'␣grating␣option.\n", progname);
        fprintf(stderr, "%s:␣(Expecting␣string␣'fractal_level').\n", progname);
        exit(FAILURE);
      }
      −− argc;
      if (¬sscanf(argv[no_arg − argc], "%d", &fractal_level)) {
        fprintf(stderr, "%s:␣Error␣in␣'cantor'␣grating␣option.\n", progname);
        fprintf(stderr, "%s:␣(Could␣not␣read␣data␣for␣fractal_level).\n", progname);
        exit(FAILURE);
      }
      −− argc;
      if (strcmp(argv[no_arg − argc], "maximum_fractal_level")) {
        fprintf(stderr, "%s:␣Error␣in␣'cantor'␣grating␣option.\n", progname);
        fprintf(stderr, "%s:␣(Expecting␣string␣'maximum_fractal_level').\n", progname);
        exit(FAILURE);
      }
      −− argc;
      if (¬sscanf(argv[no_arg − argc], "%d", &maximum_fractal_level)) {
        fprintf(stderr, "%s:␣Error␣in␣'cantor'␣grating␣option.\n", progname);
        fprintf(stderr, "%s:␣(Could␣not␣read␣data␣for␣maximum_fractal_level).\n", progname);
        exit(FAILURE);
      }
      nn = 1;
      for (j = 1; j ≤ fractal_level; j++) nn = 2 ∗ nn;
      −− argc;
      if (strcmp(argv[no_arg − argc], "t1")) {
        fprintf(stderr, "%s:␣Error␣in␣'cantor'␣grating␣option.\n", progname);
        fprintf(stderr, "%s:␣(Expecting␣string␣'t1').\n", progname);
        exit(FAILURE);
      }
      −− argc;
      if (¬sscanf(argv[no_arg − argc], "%lf", &t1)) {
        fprintf(stderr, "%s:␣Error␣in␣'cantor'␣grating␣option.\n", progname);
        fprintf(stderr, "%s:␣(Could␣not␣read␣data␣for␣t1).\n", progname);
        exit(FAILURE);
      }
      −− argc;
      if (strcmp(argv[no_arg − argc], "t2")) {
        fprintf(stderr, "%s:␣Error␣in␣'cantor'␣grating␣option.\n", progname);
        fprintf(stderr, "%s:␣(Expecting␣string␣'t2').\n", progname);
        exit(FAILURE);
      }
```

```
−− argc;
if (¬sscanf(argv[no_arg − argc], "%lf", &t2)) {
  fprintf(stderr, "%s:␣Error␣in␣'cantor'␣grating␣option.\n", progname);
  fprintf(stderr, "%s:␣(Could␣not␣read␣data␣for␣t2).\n", progname);
  exit(FAILURE);
}
−− argc;
if (strcmp(argv[no_arg − argc], "n1")) {
  fprintf(stderr, "%s:␣Error␣in␣'cantor'␣grating␣option.\n", progname);
  fprintf(stderr, "%s:␣(Expecting␣string␣'n1').\n", progname);
  exit(FAILURE);
}
−− argc;
if (¬sscanf(argv[no_arg − argc], "%lf", &n1)) {
  fprintf(stderr, "%s:␣Error␣in␣'cantor'␣grating␣option.\n", progname);
  fprintf(stderr, "%s:␣(Could␣not␣read␣data␣for␣n1).\n", progname);
  exit(FAILURE);
}
−− argc;
if (strcmp(argv[no_arg − argc], "n2")) {
  fprintf(stderr, "%s:␣Error␣in␣'cantor'␣grating␣option.\n", progname);
  fprintf(stderr, "%s:␣(Expecting␣string␣'n2').\n", progname);
  exit(FAILURE);
}
−− argc;
if (¬sscanf(argv[no_arg − argc], "%lf", &n2)) {
  fprintf(stderr, "%s:␣Error␣in␣'cantor'␣grating␣option.\n", progname);
  fprintf(stderr, "%s:␣(Could␣not␣read␣data␣for␣n2).\n", progname);
  exit(FAILURE);
}
−− argc;
if (strcmp(argv[no_arg − argc], "g1")) {
  fprintf(stderr, "%s:␣Error␣in␣'cantor'␣grating␣option.\n", progname);
  fprintf(stderr, "%s:␣(Expecting␣string␣'g1').\n", progname);
  exit(FAILURE);
}
−− argc;
if (¬sscanf(argv[no_arg − argc], "%lf", &g1)) {
  fprintf(stderr, "%s:␣Error␣in␣'cantor'␣grating␣option.\n", progname);
  fprintf(stderr, "%s:␣(Could␣not␣read␣data␣for␣g1).\n", progname);
  exit(FAILURE);
}
−− argc;
if (strcmp(argv[no_arg − argc], "g2")) {
  fprintf(stderr, "%s:␣Error␣in␣'cantor'␣grating␣option.\n", progname);
  fprintf(stderr, "%s:␣(Expecting␣string␣'g2').\n", progname);
  exit(FAILURE);
}
−− argc;
if (¬sscanf(argv[no_arg − argc], "%lf", &g2)) {
  fprintf(stderr, "%s:␣Error␣in␣'cantor'␣grating␣option.\n", progname);
  fprintf(stderr, "%s:␣(Could␣not␣read␣data␣for␣g2).\n", progname);
```

```
    exit(FAILURE);
  }
  −− argc;
  if (strcmp(argv[no_arg − argc], "pe1")) {
    fprintf(stderr, "%s:␣Error␣in␣'cantor'␣grating␣option.\n", progname);
    fprintf(stderr, "%s:␣(Expecting␣string␣'pe1').\n", progname);
    exit(FAILURE);
  }
  −− argc;
  if (¬sscanf(argv[no_arg − argc], "%lf", &pe1)) {
    fprintf(stderr, "%s:␣Error␣in␣'cantor'␣grating␣option.\n", progname);
    fprintf(stderr, "%s:␣(Could␣not␣read␣data␣for␣pe1).\n", progname);
    exit(FAILURE);
  }
  −− argc;
  if (strcmp(argv[no_arg − argc], "pe2")) {
    fprintf(stderr, "%s:␣Error␣in␣'cantor'␣grating␣option.\n", progname);
    fprintf(stderr, "%s:␣(Expecting␣string␣'pe2').\n", progname);
    exit(FAILURE);
  }
  −− argc;
  if (¬sscanf(argv[no_arg − argc], "%lf", &pe2)) {
    fprintf(stderr, "%s:␣Error␣in␣'cantor'␣grating␣option.\n", progname);
    fprintf(stderr, "%s:␣(Could␣not␣read␣data␣for␣pe2).\n", progname);
    exit(FAILURE);
  }
  −− argc;
  if (strcmp(argv[no_arg − argc], "pm1")) {
    fprintf(stderr, "%s:␣Error␣in␣'cantor'␣grating␣option.\n", progname);
    fprintf(stderr, "%s:␣(Expecting␣string␣'pm1').\n", progname);
    exit(FAILURE);
  }
  −− argc;
  if (¬sscanf(argv[no_arg − argc], "%lf", &pm1)) {
    fprintf(stderr, "%s:␣Error␣in␣'cantor'␣grating␣option.\n", progname);
    fprintf(stderr, "%s:␣(Could␣not␣read␣data␣for␣pm1).\n", 39progname);
    exit(FAILURE);
  }
  −− argc;
  if (strcmp(argv[no_arg − argc], "pm2")) {
    fprintf(stderr, "%s:␣Error.\n", progname);
    fprintf(stderr, "%s:␣(Expecting␣string␣'pm2').\n", progname);
    exit(FAILURE);
  }
  −− argc;
  if (¬sscanf(argv[no_arg − argc], "%lf", &pm2)) {
    fprintf(stderr, "%s:␣Error␣in␣'cantor'␣grating␣option.\n", progname);
    fprintf(stderr, "%s:␣(Could␣not␣read␣data␣for␣pm2).\n", 39progname);
    exit(FAILURE);
  }
  −− argc;
  if (strcmp(argv[no_arg − argc], "qe1")) {
    fprintf(stderr, "%s:␣Error␣in␣'cantor'␣grating␣option.\n", progname);
    fprintf(stderr, "%s:␣(Expecting␣string␣'qe1').\n", progname);
    exit(FAILURE);
  }
  --argc;
  if (¬sscanf(argv[no_arg − argc], "%lf", &qe1)) {
    fprintf(stderr, "%s:␣Error␣in␣'cantor'␣grating␣option.\n", progname);
    fprintf(stderr, "%s:␣(Could␣not␣read␣data␣for␣qe1).\n", 39progname);
    exit(FAILURE);
  }
  --argc;
  if (strcmp(argv[no_arg − argc], "qe2")) {
    fprintf(stderr, "%s:␣Error␣in␣'cantor'␣grating␣option.\n", progname);
    fprintf(stderr, "%s:␣(Expecting␣string␣'qe2').\n", progname);
    exit(FAILURE);
  }
  --argc;
  if (¬sscanf(argv[no_arg − argc], "%lf", &qe2)) {
    fprintf(stderr, "%s:␣Error␣in␣'cantor'␣grating␣option.\n", progname);
    fprintf(stderr, "%s:␣(Could␣not␣read␣data␣for␣qe2).\n", progname);
    exit(FAILURE);
  }
  --argc;
  if (strcmp(argv[no_arg − argc], "qm1")) {
    fprintf(stderr, "%s:␣Error␣in␣'cantor'␣grating␣option.\n", progname);
    fprintf(stderr, "%s:␣(Expecting␣string␣'qm1').\n", progname);
    exit(FAILURE);
  }
  --argc;
  if (¬sscanf(argv[no_arg − argc], "%lf", &qm1)) {
    fprintf(stderr, "%s:␣Error␣in␣'cantor'␣grating␣option.\n", progname);
    fprintf(stderr, "%s:␣(Could␣not␣read␣data␣for␣qm1).\n", progname);
    exit(FAILURE);
  }
  --argc;
  if (strcmp(argv[no_arg − argc], "qm2")) {
    fprintf(stderr, "%s:␣Error␣in␣'cantor'␣grating␣option.\n", progname);
    fprintf(stderr, "%s:␣(Expecting␣string␣'qm2').\n", progname);
    exit(FAILURE);
  }
  --argc;
  if (¬sscanf(argv[no_arg − argc], "%lf", &qm2)) {
    fprintf(stderr, "%s:␣Error␣in␣'cantor'␣grating␣option.\n", progname);
    fprintf(stderr, "%s:␣(Could␣not␣read␣data␣for␣qm2).\n", progname);
    exit(FAILURE);
  }
}
else {
  fprintf(stderr, "%s:␣Error␣in␣'fractal'␣grating␣option.\n", progname);
  fprintf(stderr, "%s:␣(No␣valid␣fractal␣type␣found!)\n", progname);
  fprintf(stderr, "%s:␣(Currently␣only␣Cantor␣type␣implemented)\n", progname);
  exit(FAILURE);
  }
}
```

This code is used in section 116.

**126.** Parse the command line for options related to the manual modification of an arbitrary discrete layer of the grating structure (which may be of type stepwise twolevel, sinusoidal, chirped, or any other, arbitrary type).

```
⟨ Parse for options for modified layer of grating structure 126 ⟩ ≡
  {
    −− argc;
    if (strcmp(argv[no_arg − argc], "num")) {
      fprintf(stderr, "%s:␣Error␣in␣'--modifylayer'.\n", progname);
      fprintf(stderr, "%s:␣(Expecting␣string␣'num'␣after␣'--modifylayer').\n", progname);
      exit(FAILURE);
    }
    −− argc;
    if (¬sscanf(argv[no_arg − argc], "%ld", &modnum)) {
      fprintf(stderr, "%s:␣Error␣in␣'--modifylayer'.\n", progname);
      fprintf(stderr, "%s:␣(Could␣not␣read␣data␣for␣num).\n", progname);
      exit(FAILURE);
    }
    −− argc;
    if (strcmp(argv[no_arg − argc], "t")) {
      fprintf(stderr, "%s:␣Error␣in␣'--modifylayer'.\n", progname);
      fprintf(stderr, "%s:␣(Expecting␣string␣'t'␣[for␣layer␣thickness]).\n", progname);
      exit(FAILURE);
    }
    −− argc;
    if (¬sscanf(argv[no_arg − argc], "%lf", &modt1)) {
      fprintf(stderr, "%s:␣Error␣in␣'--modifylayer'.\n", progname);
      fprintf(stderr, "%s:␣(Could␣not␣read␣data␣for␣t␣[layer␣thickness]).\n", progname);
      exit(FAILURE);
    }
    −− argc;
    if (strcmp(argv[no_arg − argc], "n")) {
      fprintf(stderr, "%s:␣Error␣in␣'--modifylayer'.\n", progname);
      fprintf(stderr, "%s:␣(Expecting␣string␣'n'␣[for␣refractive␣index]).\n", progname);
      exit(FAILURE);
    }
    −− argc;
    if (¬sscanf(argv[no_arg − argc], "%lf", &modn1)) {
      fprintf(stderr, "%s:␣Error␣in␣'--modifylayer'.\n", progname);
      fprintf(stderr, "%s:␣(Could␣not␣read␣data␣for␣n␣[refractive␣index]).\n", progname);
      exit(FAILURE);
    }
    −− argc;
    if (strcmp(argv[no_arg − argc], "g")) {
      fprintf(stderr, "%s:␣Error␣in␣'--modifylayer'.\n", progname);
      fprintf(stderr, "%s:␣(Expecting␣string␣'g'␣[for␣gyration␣constant]).\n", progname);
      exit(FAILURE);
    }
    −− argc;
    if (¬sscanf(argv[no_arg − argc], "%lf", &modg1)) {
      fprintf(stderr, "%s:␣Error␣in␣'--modifylayer'.\n", progname);
      fprintf(stderr, "%s:␣(Could␣not␣read␣data␣for␣g␣[gyration␣constant]).\n", progname);
      exit(FAILURE);
    }
    --argc;
    if (strcmp(argv[no_arg - argc], "pe")) {
      fprintf(stderr, "%s:␣Error␣in␣'--modifylayer'.\n", progname);
      fprintf(stderr, "%s:␣(Expecting␣string␣'pe').\n", progname);
      exit(FAILURE);
    }
    --argc;
    if (¬sscanf(argv[no_arg - argc], "%lf", &modpe1)) {
      fprintf(stderr, "%s:␣Error␣in␣'--modifylayer'.\n", progname);
      fprintf(stderr, "%s:␣(Could␣not␣read␣data␣for␣pe).\n", progname);
      exit(FAILURE);
    }
    --argc;
    if (strcmp(argv[no_arg - argc], "pm")) {
      fprintf(stderr, "%s:␣Error␣in␣'--modifylayer'.\n", progname);
      fprintf(stderr, "%s:␣(Expecting␣string␣'pm').\n", progname);
      exit(FAILURE);
    }
    --argc;
    if (¬sscanf(argv[no_arg - argc], "%lf", &modpm1)) {
      fprintf(stderr, "%s:␣Error␣in␣'--modifylayer'.\n", progname);
      fprintf(stderr, "%s:␣(Could␣not␣read␣data␣for␣pm).\n", progname);
      exit(FAILURE);
    }
    --argc;
    if (strcmp(argv[no_arg - argc], "qe")) {
      fprintf(stderr, "%s:␣Error␣in␣'--modifylayer'.\n", progname);
      fprintf(stderr, "%s:␣(Expecting␣string␣'qe').\n", progname);
      exit(FAILURE);
    }
    --argc;
    if (¬sscanf(argv[no_arg - argc], "%lf", &modqe1)) {
      fprintf(stderr, "%s:␣Error␣in␣'--modifylayer'.\n", progname);
      fprintf(stderr, "%s:␣(Could␣not␣read␣data␣for␣qe).\n", progname);
      exit(FAILURE);
    }
    --argc;
    if (strcmp(argv[no_arg - argc], "qm")) {
      fprintf(stderr, "%s:␣Error␣in␣'--modifylayer'.\n", progname);
      fprintf(stderr, "%s:␣(Expecting␣string␣'qm').\n", progname);
      exit(FAILURE);
    }
    --argc;
    if (¬sscanf(argv[no_arg - argc], "%lf", &modqm1)) {
      fprintf(stderr, "%s:␣Error␣in␣'--modifylayer'.\n", progname);
      fprintf(stderr, "%s:␣(Could␣not␣read␣data␣for␣qm).\n", progname);
      exit(FAILURE);
    }
  }
```

This code is used in section 116.

**127.** Parse the command line for options specifying the transmitted optical intensity range. The command line syntax for specification of the transmitted optical intensity range is

`--trmintensity` $\langle I_{\rm start}\rangle$ $\langle I_{\rm stop}\rangle$ $\langle M_{\rm i}\rangle$

where the numerical values supplied are internally kept by the variables *trmintenstart*, *trmintenstop*, and *mmi*, respectively.

⟨ Parse the command line for transmitted intensity range 127 ⟩ ≡

```
{
  −− argc;
  if (¬sscanf(argv[no_arg − argc], "%lf", &trmintenstart)) {
    fprintf(stderr, "%s:␣Error␣in␣'--trmintensity'␣option.\n", 39progname);
    exit(FAILURE);
  }
  −− argc;
  if (¬sscanf(argv[no_arg − argc], "%lf", &trmintenstop)) {
    fprintf(stderr, "%s:␣Error␣in␣'--trmintensity'␣option.\n", 39progname);
    exit(FAILURE);
  }
  −− argc;
  if (¬sscanf(argv[no_arg − argc], "%ld", &mmi)) {
    fprintf(stderr, "%s:␣Error␣in␣'--trmintensity'␣option.\n", 39progname);
    exit(FAILURE);
  }
}
```

This code is used in section 116.

**128.** Parse the command line for options specifying the ellipticity range of the transmitted polarization state. The command line syntax for specification of the ellipticity of the transmitted polarization state is analogous to that of the transmitted intensity,

`--trmellipticity` $\langle \epsilon_{\rm start}\rangle$ $\langle \epsilon_{\rm stop}\rangle$ $\langle M_{\rm e}\rangle$

where the numerical values supplied are internally kept by the variables *trmellipstart*, *trmellipstop*, and *mme*, respectively.

⟨ Parse the command line for transmitted ellipticity range 128 ⟩ ≡

```
{
  −− argc;
  if (¬sscanf(argv[no_arg − argc], "%lf", &trmellipstart)) {
    fprintf(stderr, "%s:␣Error␣in␣'--trmellipticity'␣option.\n", progname);
    exit(FAILURE);
  }
  −− argc;
  if (¬sscanf(argv[no_arg − argc], "%lf", &trmellipstop)) {
    fprintf(stderr, "%s:␣Error␣in␣'--trmellipticity'␣option.\n", progname);
    exit(FAILURE);
  }
  −− argc;
  if (¬sscanf(argv[no_arg − argc], "%ld", &mme)) {
    fprintf(stderr, "%s:␣Error␣in␣'--trmellipticity'␣option.\n", progname);
    exit(FAILURE);
  }
}
```

This code is used in section 116.

**129.** If the flag *verbose* is set to true (1 in C), then display the information parsed from the command line, such as output filenames, indices of refraction, grating period, etc. This is useful for later on creating log files of sessions with the executable program MAGBRAGG.

⟨ Display parameters parsed from the command line 129 ⟩ ≡

```
{
  if (verbose) {
    for (k = 1; k ≤ 64; k++) fprintf(stdout, (k < 64 ? "-" : "\n"));
    fprintf(stdout, "Input␣parameters:\n");
    fprintf(stdout, "Grating␣type:␣%s\n", gratingtype);
    if (¬strcmp(gratingtype, "sinusoidal")) {
      fprintf(stdout, "Bias␣refractive␣index:␣␣␣␣␣␣␣␣␣␣%10.6e\n", n1);
      fprintf(stdout, "Refractive␣index␣modulation:␣␣␣␣%10.6e\n", n2);
      fprintf(stdout, "Modulation␣period:␣␣␣␣␣␣␣␣␣␣␣␣␣%10.6e␣[m]\n", nper);
    }
    else if (¬strcmp(gratingtype, "chirped")) { }
    else if (¬strcmp(gratingtype, "stepwise")) { }
    else if (¬strcmp(gratingtype, "fractal")) { }
    else {
      fprintf(stdout, "%s:␣Error:␣Unknown␣grating␣type␣'%s'\n", progname, gratingtype);
    }
    fprintf(stdout, "Geometrical␣length:␣␣␣␣␣␣␣␣␣␣␣␣" "L=%10.6e␣[m]\n", ll);
    fprintf(stdout, "Surrounding␣refractive␣index:␣␣" "nsurr=%10.6e\n", nsurr);
    fprintf(stdout, "Begin␣wavelength␣of␣spectrum:␣␣" "lambda_start=%10.6e␣[m]\n", lambdastart);
    fprintf(stdout, "End␣wavelength␣of␣spectrum:␣␣␣␣" "lambda_stop=%10.6e␣[m]\n", lambdastop);
    fprintf(stdout, "Number␣of␣samples␣in␣spectrum:␣" "M=%-12ld\n", mm);
    fprintf(stdout, "Number␣of␣discrete␣layers:␣␣␣␣␣" "N=%-12ld\n", nn);
    if (trmtraject_specified) {
      fprintf(stdout, "Trajectory␣for␣transmitted␣Stokes␣parameters:␣%s\n", trmtraject_filename);
    }
    else {
      fprintf(stdout, "Number␣of␣samples␣in␣output␣intensity:␣␣␣" "mmi=%-12ld\n", mmi);
      fprintf(stdout, "Number␣of␣samples␣in␣output␣ellipticity:␣" "mme=%-12ld\n", mme);
    }
    fprintf(stdout, "Stokes␣parameters␣will␣be␣written␣to␣files:\n");
    fprintf(stdout, "␣␣␣%s␣␣[S0␣(incident␣wave)],\n", outfilename_s0);
    fprintf(stdout, "␣␣␣%s␣␣[S1␣(incident␣wave)],\n", outfilename_s1);
    fprintf(stdout, "␣␣␣%s␣␣[S2␣(incident␣wave)],\n", outfilename_s2);
    fprintf(stdout, "␣␣␣%s␣␣[S3␣(incident␣wave)],\n", outfilename_s3);
    fprintf(stdout, "␣␣␣%s␣␣[V0␣(reflected␣wave)],\n", outfilename_v0);
    fprintf(stdout, "␣␣␣%s␣␣[V1␣(reflected␣wave)],\n", outfilename_v1);
    fprintf(stdout, "␣␣␣%s␣␣[V2␣(reflected␣wave)],\n", outfilename_v2);
    fprintf(stdout, "␣␣␣%s␣␣[V3␣(reflected␣wave)],\n", outfilename_v3);
    fprintf(stdout, "␣␣␣%s␣␣[W0␣(transmitted␣wave)],\n", outfilename_w0);
    fprintf(stdout, "␣␣␣%s␣␣[W1␣(transmitted␣wave)],\n", outfilename_w1);
    fprintf(stdout, "␣␣␣%s␣␣[W2␣(transmitted␣wave)],\n", outfilename_w2);
    fprintf(stdout, "␣␣␣%s␣␣[W3␣(transmitted␣wave)],\n", outfilename_w3);
    if (fieldevoflag) {
      if (fieldevoflag_efield) {
        if (strcmp(fieldevofilename, "")) {
          fprintf(stdout,
              "Intra␣grating␣optical␣field␣evolution␣will␣" "be␣written␣to␣file:\n");
          fprintf(stdout, "␣␣␣%s\n", fieldevofilename);
        }
        else {
          fprintf(stderr, "%s:␣Error:␣No␣file␣name␣specified␣for␣s\
              aving␣spatial\n""field␣evolution.␣(efield␣option)\n", progname);
          exit(FAILURE);
        }
        fprintf(stdout, "(Intra␣grating␣field␣evolution␣will␣be␣presented␣in␣terms␣o\
            f\n""the␣electrical␣field␣displacement.)\n");
      }
      else if (fieldevoflag_stoke) {
        if (strcmp(fieldevofilename_s0, "") ∧ strcmp(fieldevofilename_s1,
              "") ∧ strcmp(fieldevofilename_s2, "") ∧ strcmp(fieldevofilename_s3, "")) {
          fprintf(stdout,
              "Intra␣grating␣optical␣field␣evolution␣will␣""be␣written␣to␣files:\n");
          fprintf(stdout, "␣␣␣%s\n␣␣␣%s\n␣␣␣%s\n␣␣␣%s\n", fieldevofilename_s0, fieldevofilename_s1,
              fieldevofilename_s2, fieldevofilename_s3);
        }
        else {
          fprintf(stderr, "%s:␣Error:␣No␣file␣name␣specified␣for␣s\
              aving␣spatial\n""field␣evolution.␣(stoke␣option)\n", progname);
          exit(FAILURE);
        }
        fprintf(stdout, "(Intra␣grating␣field␣evolution␣will␣be␣presented␣in␣terms␣o\
            f\n""the␣Stokes␣parameters␣of␣the␣forward␣propagating␣field.)\n");
      }
      else {
        fprintf(stderr, "%s:␣Unknown␣field␣evolution␣flag.\n", progname);
        exit(FAILURE);
      }
      fprintf(stdout, "Number␣of␣intermediate␣samples␣within␣each␣layer:␣%-12ld\n", nne);
    }
    if (intensityevoflag) {
      if (strcmp(intensityevofilename, "")) {
        fprintf(stdout,
            "Intra␣grating␣optical␣intensity␣evolution␣will␣""be␣written␣to␣file:\n");
        fprintf(stdout, "␣␣␣%s\n", intensityevofilename);
      }
    }
    fprintf(stdout, "Program␣execution␣started␣%s", ctime(&initime));
    for (k = 1; k ≤ 64; k++) fprintf(stdout, (k < 64 ? "-" : "\n"));
  }
}
```

This code is used in section 116.

**130.** Routines for displaying help message to standard terminal output.

⟨ Routines for displaying help message 130 ⟩ ≡
  ⟨ Display split help line 131 ⟩
  ⟨ Display full help line 132 ⟩
  ⟨ Display help message 133 ⟩

This code is used in section 50.

**131.** Routine for proper display of split help lines. This is a very simple routine just to keep the *fprintf*(*stderr*, "...", "...") statements to a minimum. The routine also checks that the full length of the diplayed line does not exceed 80 characters, as conforming to normal line length of terminal output.

```
⟨ Display split help line 131 ⟩ ≡
  void hl(char firststring[], char secondstring[])
  {
    if (strlen(firststring) > 25) {
      fprintf(stderr, "%s:*******␣Error␣in␣hl()␣routine!␣*******\n", progname);
      fprintf(stderr, "%s:␣The␣first␣string␣argument␣is␣too␣long:\n", progname);
      fprintf(stderr, "%s:␣␣␣'%s'\n", progname, firststring);
      fprintf(stderr, "%s:␣String␣lengths␣is␣%d␣characters\n", progname, (int) strlen(firststring));
      fprintf(stderr, "%s:␣(Maximum␣25␣characters␣for␣first␣argument.)\n", progname);
      fprintf(stderr, "%s:********␣End␣of␣error␣message␣*********\n", progname);
      exit(FAILURE);
    }
    if (strlen(secondstring) > 55) {
      fprintf(stderr, "%s:*******␣Error␣in␣hl()␣routine!␣*******\n", progname);
      fprintf(stderr, "%s:␣The␣second␣string␣argument␣is␣too␣long:\n", progname);
      fprintf(stderr, "%s:␣␣␣'%s'\n", progname, secondstring);
      fprintf(stderr, "%s:␣String␣lengths␣is␣%d␣characters\n", progname, (int) strlen(secondstring));
      fprintf(stderr, "%s:␣(Maximum␣55␣characters␣for␣second␣argument.)\n", progname);
      fprintf(stderr, "%s:********␣End␣of␣error␣message␣*********\n", progname);
      exit(FAILURE);
    }
    fprintf(stderr, "%-25.25s%1.55s\n", firststring, secondstring);
  }
```

This code is used in section 130.

**132.** Routine for proper display of full help lines. This is similar to the *hl*( ) routine, with the only difference being that a full line of text is flushed instead of a line split into two parts.

```
⟨ Display full help line 132 ⟩ ≡
  void fhl(char linestring[])
  {
    if (strlen(linestring) > 80) {
      fprintf(stderr, "%s:*******␣Error␣in␣fhl()␣routine!␣*******\n", progname);
      fprintf(stderr, "%s:␣The␣following␣help␣line␣is␣too␣long:\n", progname);
      fprintf(stderr, "%s:␣␣␣'%s'\n", progname, linestring);
      fprintf(stderr, "%s:␣String␣is␣%d␣characters\n", progname, (int) strlen(linestring));
      fprintf(stderr, "%s:␣(Maximum␣80␣characters␣per␣help␣line␣is␣allowed.)\n", progname);
      fprintf(stderr, "%s:********␣End␣of␣error␣message␣*********\n", progname);
      exit(FAILURE);
    }
    fprintf(stderr, "%s\n", linestring);
  }
```

This code is used in section 130.

**133.** Show a help message at the screen, giving the full syntax of the command line options that are accepted by the program.

```
⟨ Display help message 133 ⟩ ≡
  void showsomehelp(void)
  {
    fprintf(stderr, "␣Usage:␣%s␣[options]\n", progname);
    fhl("␣Options:");
    hl("␣␣-h,␣--help", "Display␣this␣help␣message␣and␣exit␣clean.");
    hl("␣␣-N␣<int>", "");
    hl("␣␣-M␣<int>", "");
    hl("␣␣-v,␣--verbose", "");
    hl("␣␣-o,␣--outputfile␣<str>", "");
    fhl("␣␣--fieldevolution␣{efield|stoke}␣<n>␣<str>");
    fhl("␣␣--intensityevolution␣<lambda>␣<str>");
    fhl("␣␣--normalize_length_to_um");
    hl("␣␣--normalize_intensity", "");
    hl("", "When␣saving␣the␣spatial␣evolution␣of␣the␣intra-");
    hl("", "grating␣intensity,␣normalize␣the␣intensity␣with");
    hl("", "respect␣to␣the␣intensity␣at␣z=0,␣inside␣the");
    hl("", "grating␣(that␣is␣to␣say,␣normalize␣with␣respect");
    hl("", "to␣the␣initial␣intra-grating␣intensity).␣This");
    hl("", "option␣*only*␣affects␣the␣fields␣saved␣with␣the");
    hl("", "--intensityevolution␣or␣--fieldevolution");
    hl("", "options.");
    hl("␣␣-r,␣--random", "");
    hl("␣␣-a,␣--apodize␣<real>", "");
    hl("", "Apodize␣the␣grating␣structure␣over␣geometrical");
    hl("", "distance␣<real>␣at␣each␣end␣of␣the␣grating.");
    hl("", "The␣option␣only␣applies␣to␣gratings␣with␣sinus-");
    hl("", "oidal␣modulation␣of␣refractive␣index␣and");
    hl("", "gyration␣coefficient,␣in␣constant␣or␣chirped");
    hl("", "periodic␣configurations,␣as␣specified␣with␣the");
    hl("", "'--grating␣sinusoidal'␣or␣'--grating␣chirped'");
    hl("", "options␣respectively.");
    hl("", "␣␣␣The␣apodization␣is␣applied␣to␣the␣refractive");
    hl("", "index␣modulation␣and,␣in␣cases␣where␣the␣linear");
    hl("", "magneto-optical␣is␣spatially␣modulated␣as␣well,");
    hl("", "to␣the␣linear␣gyration␣coefficient.");
    fhl("␣␣-j,␣--phasejump␣<r1␣(angle)>␣<r2␣(position)>");
    hl("", "Apply␣discrete␣phase␣jump␣in␣the␣spatial␣phase");
    hl("", "of␣the␣grating␣profile.␣This␣option␣adds␣the");
    hl("", "real␣number␣<r1>␣to␣the␣argument␣of␣the␣sinus-");
    hl("", "oidal␣function␣for␣the␣grating␣profile␣for␣all");
    hl("", "spatial␣coordinates␣z␣>=␣<r2>.");
    hl("", "␣␣␣As␣in␣the␣case␣of␣apodization,␣this␣option");
    hl("", "only␣applies␣to␣gratings␣with␣sinusoidal␣modu-");
    hl("", "lation␣of␣refractive␣index␣and␣gyration␣coeffi-");
    hl("", "cient,␣in␣constant␣or␣chirped␣periodic␣configu-");
    hl("", "rations,␣as␣specified␣with␣the␣'--grating");
    hl("", "sinusoidal'␣or␣'--grating␣chirped'␣options");
    hl("", "respectively.␣The␣discrete␣phase␣jump␣applies");
    hl("", "to␣the␣linear␣linear␣refractive␣index␣modula-");
    hl("","tion␣and,␣in␣cases␣where␣the␣linear␣magneto-");
    hl("","optical␣interaction␣is␣spatially␣modulated␣as");
    hl("","well,␣also␣to␣the␣linear␣gyration␣coefficient.");
    fhl("␣␣-w,␣--writegratingfile␣<str>");
    hl("␣␣--spectrumfile␣<str>","");
    hl("","Generates␣the␣complex␣reflectance␣as␣function");
    hl("","of␣the␣vacuum␣wavelength␣in␣meters,␣and␣save");
    hl("","the␣spectrum␣in␣file␣named␣according␣to␣the");
    hl("","supplied␣character␣string␣<str>.");
    fhl("␣␣--intensityspectrumfile␣<str>");
    hl("","Generates␣the␣intensity␣reflectance␣as␣function");
    hl("","of␣the␣vacuum␣wavelength␣in␣meters,␣and␣save");
    hl("","the␣spectrum␣in␣file␣named␣according␣to␣the");
    hl("","supplied␣character␣string␣<str>.");
    fhl("␣␣-g,␣--grating␣<grating␣options>");
    hl("","Specifies␣the␣grating␣type,␣where");
    hl("","<grating␣options>␣=␣");
    hl("","␣␣␣[stepwise␣<stepwise␣options>␣|");
    hl("","␣␣␣␣sinusoidal␣<sinusoidal␣options>␣|");
    hl("","␣␣␣␣chirped␣<chirped␣options>]");
    hl("","<stepwise␣options>␣=␣");
    hl("","␣␣␣twolevel␣t1␣␣<f>␣t2␣<f>");
    hl("","␣␣␣␣␣␣␣␣␣␣␣␣n1␣␣<f>␣n2␣<f>␣␣g1␣␣<f>␣g2␣␣<f>");
    hl("","␣␣␣␣␣␣␣␣␣␣␣␣pe1␣<f>␣pe2␣<f>␣pm1␣<f>␣pm2␣<f>");
    hl("","␣␣␣␣␣␣␣␣␣␣␣␣qe1␣<f>␣qe2␣<f>␣qm1␣<f>␣qm2␣<f>");
    hl("","<sinusoidal␣options>␣=␣");
    hl("","␣␣␣␣␣␣␣␣␣␣␣␣n␣␣<n0>␣␣<dn>␣␣<nper>");
    hl("","␣␣␣␣␣␣␣␣␣␣␣␣g␣␣<g0>␣␣<dg>␣␣<gper>");
    hl("","␣␣␣␣␣␣␣␣␣␣␣␣pe␣<pe0>␣<dpe>␣<peper>");
    hl("","␣␣␣␣␣␣␣␣␣␣␣␣pm␣<pm0>␣<dpm>␣<pmper>");
    hl("","␣␣␣␣␣␣␣␣␣␣␣␣qe␣<qe0>␣<dqe>␣<qeper>");
    hl("","␣␣␣␣␣␣␣␣␣␣␣␣qm␣<qm0>␣<dqm>␣<qmper>");
    hl("","<chirped␣options>␣=␣");
    hl("","␣␣␣␣␣␣␣␣␣␣␣␣n␣␣<n0>␣␣<dn>␣␣<nper>␣␣<ncrp>");
    hl("","␣␣␣␣␣␣␣␣␣␣␣␣g␣␣<g0>␣␣<dg>␣␣<gper>␣␣<gcrp>");
    hl("","␣␣␣␣␣␣␣␣␣␣␣␣pe␣<pe0>␣<dpe>␣<peper>␣<pecrp>");
    hl("","␣␣␣␣␣␣␣␣␣␣␣␣pm␣<pm0>␣<dpm>␣<pmper>␣<pmcrp>");
    hl("","␣␣␣␣␣␣␣␣␣␣␣␣qe␣<qe0>␣<dqe>␣<qeper>␣<qecrp>");
    hl("","␣␣␣␣␣␣␣␣␣␣␣␣qm␣<qm0>␣<dqm>␣<qmper>␣<qmcrp>");
    hl("␣␣-L,--gratinglength␣<f>","Physical␣length␣of␣grating␣in␣meter␣[m]");
    fhl("␣␣␣␣␣--refindsurr␣<f>");
    fhl("␣␣␣␣␣--trmtraject␣<str>");
    fhl("␣␣␣␣␣--trmintensity␣<istart>␣<istop>␣<mmi>");
    fhl("␣␣␣(intensity␣measured␣in␣Watts␣per␣square␣meter)");
    fhl("␣␣␣␣␣--trmellipticity␣<estart>␣<estop>␣<mme>");
    fhl("␣␣␣␣␣--lambdastart␣<lambda>");
    fhl("␣␣␣(start␣vacuum␣wavelength␣measured␣in␣meter)");
    fhl("␣␣␣␣␣--lambdastop␣<lambda>");
    fhl("␣␣␣(stop␣vacuum␣wavelength␣measured␣in␣meter)");
    exit(FAILURE);
  }
```

This code is used in section 130.

**134. Check for specified trajectory of transmitted Stokes parameters.**

⟨ Check for specified trajectory of transmitted Stokes parameters 134 ⟩ ≡

```
{
  mmtraject = 0;
  if (trmtraject_specified) {      /* Was a trajectory specified at all? */
    if ((fp_traject = fopen(trmtraject_filename, "r")) ≡ Λ) {
      fprintf(stderr,
          "%s:␣Could␣not␣open␣file␣%s␣for␣reading␣Stokes␣parameter" "␣trajectory␣of␣tran\
          smitted␣wave!\n", progname, trmtraject_filename);
      exit(FAILURE);
    }
    fseek(fp_traject, 0L, SEEK_SET);
      /* Scan the specified file for the number of points of the trajectory */
    while ((tmpch = getc(fp_traject)) ≠ EOF) {
      ungetc(tmpch, fp_traject);
      fscanf(fp_traject, "%lf", &tmp);      /* Read away the W0 parameter */
      fscanf(fp_traject, "%lf", &tmp);      /* Read away the W3 parameter */
      mmtraject++;      /* Read away blanks and linefeeds */
      tmpch = getc(fp_traject);
      while ((tmpch ≠ EOF) ∧ (¬numeric(tmpch))) {
        tmpch = getc(fp_traject);
      }
      if (tmpch ≠ EOF) ungetc(tmpch, fp_traject);
    }
    if (verbose) {
      fprintf(stdout, "%s:␣I␣have␣now␣pre-parsed␣the␣specified␣trajectory␣of␣trans\
          mitted\n" "Stokes␣parameters␣(W0,W3)␣in␣file␣%s,␣and␣I␣found␣%-ld␣points.\n",
          progname, trmtraject_filename, mmtraject);
      fprintf(stdout, "%s:␣Now␣allocating␣the␣vectors␣for␣the␣transmitted␣trajectory...",
          progname);
    }
    fseek(fp_traject, 0L, SEEK_SET);      /* Rewind the file for reading */
    w0traj = dvector(1, mmtraject);      /* Allocate memory for w0traj */
    w3traj = dvector(1, mmtraject);      /* Allocate memory for w3traj */
    for (ke = 1; ke ≤ mmtraject; ke++) {
      fscanf(fp_traject, "%le", &w0traj[ke]);      /* Read the W0 parameter */
      fscanf(fp_traject, "%le", &w3traj[ke]);      /* Read the W3 parameter */
    }
    if (0 ≡ 1) {
      for (ke = 1; ke ≤ mmtraject; ke++) fprintf(stdout, "w0=%e␣␣␣w3=%e\n", w0traj[ke], w3traj[ke]);
    }
    fclose(fp_traject);
  }
}
```

This code is used in section 45.

**135. Opening and closing files for data output.** Open output files, to be used later on for saving Stokes parameters on disk. The naming convention of the files is that the *outfilename* string (at the command line specified using the `-o` ⟨*outfilename*⟩ or `--outputfile` ⟨*outfilename*⟩ option) is the base name, with suffixes `.s0.dat`, `.s1.dat`, etc., indicating the actual Stoke parameter which was written to respective file.

The following string variables contain the filenames of the files where to store the calculated data:

| | |
|---|---|
| *outfilename_s0* | The $S_0$ Stokes parameter of the incident optical wave, governing the optical intensity. |
| *outfilename_s1* | The $S_1$ Stokes parameter of the incident optical wave, together with $S_2$ governing the orientation of the main axis of the polarization ellipse. |
| *outfilename_s2* | The $S_2$ Stokes parameter of the incident optical wave, together with $S_1$ governing the orientation of the main axis of the polarization ellipse. |
| *outfilename_s3* | The $S_3$ Stokes parameter of the incident optical wave, governing the polarization state ellipticity. |
| *outfilename_v0* | The $V_0$ Stokes parameter of the reflected optical wave, governing the optical intensity. |
| *outfilename_v1* | The $V_1$ Stokes parameter of the reflected optical wave, together with $V_2$ governing the orientation of the main axis of the polarization ellipse. |
| *outfilename_v2* | The $V_2$ Stokes parameter of the reflected optical wave, together with $V_1$ governing the orientation of the main axis of the polarization ellipse. |
| *outfilename_v3* | The $V_3$ Stokes parameter of the reflected optical wave, governing the polarization state ellipticity. |
| *outfilename_w0* | The $W_0$ Stokes parameter of the transmitted optical wave, governing the optical intensity. |
| *outfilename_w1* | The $W_1$ Stokes parameter of the transmitted optical wave, together with $W_2$ governing the orientation of the main axis of the polarization ellipse. |
| *outfilename_w2* | The $W_2$ Stokes parameter of the transmitted optical wave, together with $W_1$ governing the orientation of the main axis of the polarization ellipse. |
| *outfilename_w3* | The $W_3$ Stokes parameter of the transmitted optical wave, governing the polarization state ellipticity. |

The reason for using separate files for the Stokes parameters is that in many cases sets or matrices of Stokes parameters will be generated for certain material or geometrical parameters, resulting in several different topological surfaces of, for example, transmitted intensity $W_0$ as function of input intensity $S_0$ and ellipticity $S_3/S_0$, in which case it is convenient to load separate Stokes parameters from separate files.

```
⟨ Open files for output 135 ⟩ ≡
  {
    if ((mme > 1) ∧ (mmi > 1)) {
      if ((fp_s0 = fopen(outfilename_s0, "w")) ≡ Λ) {
        fprintf(stderr, "%s:␣Could␣not␣open␣%s␣for␣saving␣incident␣wave!\n", progname,
            outfilename_s0);
        exit(FAILURE);
      }
      if ((fp_s1 = fopen(outfilename_s1, "w")) ≡ Λ) {
        fprintf(stderr, "%s:␣Could␣not␣open␣%s␣for␣saving␣incident␣wave!\n", progname,
            outfilename_s1);
        exit(FAILURE);
      }
      if ((fp_s2 = fopen(outfilename_s2, "w")) ≡ Λ) {
        fprintf(stderr, "%s:␣Could␣not␣open␣%s␣for␣saving␣incident␣wave!\n", progname,
            outfilename_s2);
        exit(FAILURE);
      }
      if ((fp_s3 = fopen(outfilename_s3, "w")) ≡ Λ) {
        fprintf(stderr, "%s:␣Could␣not␣open␣%s␣for␣saving␣incident␣wave!\n", progname,
            outfilename_s3);
        exit(FAILURE);
      }
      if ((fp_v0 = fopen(outfilename_v0, "w")) ≡ Λ) {
        fprintf(stderr, "%s:␣Could␣not␣open␣%s␣for␣saving␣reflected␣wave!\n", progname,
            outfilename_v0);
        exit(FAILURE);
      }
      if ((fp_v1 = fopen(outfilename_v1, "w")) ≡ Λ) {
        fprintf(stderr, "%s:␣Could␣not␣open␣%s␣for␣saving␣reflected␣wave!\n", progname,
            outfilename_v1);
        exit(FAILURE);
      }
      if ((fp_v2 = fopen(outfilename_v2, "w")) ≡ Λ) {
        fprintf(stderr, "%s:␣Could␣not␣open␣%s␣for␣saving␣reflected␣wave!\n", progname,
            outfilename_v2);
        exit(FAILURE);
      }
      if ((fp_v3 = fopen(outfilename_v3, "w")) ≡ Λ) {
        fprintf(stderr, "%s:␣Could␣not␣open␣%s␣for␣saving␣reflected␣wave!\n", progname,
            outfilename_v3);
        exit(FAILURE);
      }
      if ((fp_w0 = fopen(outfilename_w0, "w")) ≡ Λ) {
        fprintf(stderr, "%s:␣Could␣not␣open␣%s␣for␣saving␣transmitted␣wave!\n", progname,
            outfilename_w0);
        exit(FAILURE);
      }
      if ((fp_w1 = fopen(outfilename_w1, "w")) ≡ Λ) {
        fprintf(stderr, "%s:␣Could␣not␣open␣%s␣for␣saving␣transmitted␣wave!\n", progname,
            outfilename_w1);
        exit(FAILURE);
      }
      if ((fp_w2 = fopen(outfilename_w2, "w")) ≡ Λ) {
        fprintf(stderr, "%s:␣Could␣not␣open␣%s␣for␣saving␣transmitted␣wave!\n", progname,
            outfilename_w2);
        exit(FAILURE);
      }
      if ((fp_w3 = fopen(outfilename_w3, "w")) ≡ Λ) {
        fprintf(stderr, "%s:␣Could␣not␣open␣%s␣for␣saving␣transmitted␣wave!\n", progname,
            outfilename_w3);
        exit(FAILURE);
      }
    }
    if ((mme > 1) ∧ (mmi > 1)) {
      fseek(fp_s0, 0L, SEEK_SET);
      fseek(fp_s1, 0L, SEEK_SET);
      fseek(fp_s2, 0L, SEEK_SET);
      fseek(fp_s3, 0L, SEEK_SET);
      fseek(fp_v0, 0L, SEEK_SET);
      fseek(fp_v1, 0L, SEEK_SET);
      fseek(fp_v2, 0L, SEEK_SET);
      fseek(fp_v3, 0L, SEEK_SET);
      fseek(fp_w0, 0L, SEEK_SET);
      fseek(fp_w1, 0L, SEEK_SET);
      fseek(fp_w2, 0L, SEEK_SET);
      fseek(fp_w3, 0L, SEEK_SET);
    }
    if (fieldevoflag) {
      if (fieldevoflag_efield) {
        if (strcmp(fieldevofilename, "")) {
          if ((fp_evo = fopen(fieldevofilename, "w")) ≡ Λ) {
            fprintf(stderr, "%s:␣Could␣not␣open␣file␣%s␣for␣fieldevo␣output!\n", progname,
                fieldevofilename);
            exit(FAILURE);
          }
        }
      }
      else if (fieldevoflag_stoke) {
        if (strcmp(fieldevofilename_s0, "")) {
          if ((fp_evo_s0 = fopen(fieldevofilename_s0, "w")) ≡ Λ) {
            fprintf(stderr,
                "%s:␣Could␣not␣open␣file␣%s␣for␣saving␣spatial␣S0""␣distribution!\n",
                progname, fieldevofilename);
            exit(FAILURE);
          }
        }
        else {
          fprintf(stderr,
              "%s:␣A␣name␣for␣the␣file␣for␣saving␣spatial␣S0␣distribution""␣is␣required!\n",
              progname);
          exit(FAILURE);
        }
        if (strcmp(fieldevofilename_s1, "")) {
          if ((fp_evo_s1 = fopen(fieldevofilename_s1, "w")) ≡ Λ) {
            fprintf(stderr,
                "%s:␣Could␣not␣open␣file␣%s␣for␣saving␣spatial␣S1""␣distribution!\n",
                progname, fieldevofilename);
            exit(FAILURE);
          }
        }
        else {
          fprintf(stderr,
              "%s:␣A␣name␣for␣the␣file␣for␣saving␣spatial␣S1␣distribution""␣is␣required!\n",
              progname);
          exit(FAILURE);
        }
        if (strcmp(fieldevofilename_s2, "")) {
          if ((fp_evo_s2 = fopen(fieldevofilename_s2, "w")) ≡ Λ) {
            fprintf(stderr,
                "%s:␣Could␣not␣open␣file␣%s␣for␣saving␣spatial␣S2""␣distribution!\n",
                progname, fieldevofilename);
            exit(FAILURE);
          }
        }
        else {
          fprintf(stderr,
              "%s:␣A␣name␣for␣the␣file␣for␣saving␣spatial␣S2␣distribution""␣is␣required!\n",
              progname);
          exit(FAILURE);
        }
        if (strcmp(fieldevofilename_s3, "")) {
          if ((fp_evo_s3 = fopen(fieldevofilename_s3, "w")) ≡ Λ) {
            fprintf(stderr,
                "%s:␣Could␣not␣open␣file␣%s␣for␣saving␣spatial␣S3""␣distribution!\n",
                progname, fieldevofilename);
            exit(FAILURE);
          }
        }
        else {
          fprintf(stderr,
              "%s:␣A␣name␣for␣the␣file␣for␣saving␣spatial␣S3␣distribution""␣is␣required!\n",
              progname);
          exit(FAILURE);
        }
      }
      else {
        fprintf(stderr, "%s:␣Unknown␣field␣evolution␣flag.\n", progname);
        exit(FAILURE);
      }
    }
    if (intensityevoflag) {
      if (strcmp(intensityevofilename, "")) {
        if ((fp_ievo = fopen(intensityevofilename, "w")) ≡ Λ) {
          fprintf(stderr, "%s:␣Could␣not␣open␣file␣%s␣for␣intensityevo␣output!\n", progname,
              intensityevofilename);
          exit(FAILURE);
        }
      }
    }
    if (¬((mme > 1) ∧ (mmi > 1))) {
      if ((fp_spec = fopen(spectrumfilename, "w")) ≡ Λ) {
        fprintf(stderr, "%s:␣Could␣not␣open␣file␣%s␣for␣spectrum␣output!\n", progname,
            spectrumfilename);
        exit(FAILURE);
      }
      if ((fp_irspec = fopen(intensity_reflection_spectrumfilename, "w")) ≡ Λ) {
        fprintf(stderr, "%s:␣Could␣not␣open␣%s␣for␣intensity␣reflection␣spectrum!\n", progname,
            intensity_reflection_spectrumfilename);
        exit(FAILURE);
      }
```

```
      if ((fp_itspec = fopen(intensity_transmission_spectrumfilename, "w")) ≡ Λ) {
        fprintf(stderr, "%s:␣Could␣not␣open␣%s␣for␣intensity␣transmission␣spectrum!\n",
            progname, intensity_transmission_spectrumfilename);
        exit(FAILURE);
      }
      if ((fp_icspec = fopen(intensity_check_spectrumfilename, "w")) ≡ Λ) {
        fprintf(stderr, "%s:␣Could␣not␣open␣%s␣for␣checking␣spectra!\n", progname,
            intensity_check_spectrumfilename);
        exit(FAILURE);
      }
    }
  }
```

This code is used in section 45.

**136.** Close all open files.

```
⟨ Close output files 136 ⟩ ≡
  {
    if ((mme > 1) ∧ (mmi > 1)) {
      fclose(fp_s0);
      fclose(fp_s1);
      fclose(fp_s2);
      fclose(fp_s3);
      fclose(fp_v0);
      fclose(fp_v1);
      fclose(fp_v2);
      fclose(fp_v3);
      fclose(fp_w0);
      fclose(fp_w1);
      fclose(fp_w2);
      fclose(fp_w3);
    }
    if (fieldevoflag)
      if (strcmp(fieldevofilename, "")) fclose(fp_evo);
    if (intensityevoflag)
      if (strcmp(intensityevofilename, "")) fclose(fp_ievo);
    if (¬((mme > 1) ∧ (mmi > 1))) {
      fclose(fp_spec);
      fclose(fp_irspec);
      fclose(fp_itspec);
      fclose(fp_icspec);
    }
  }
```

This code is used in section 45.

**137. References.**

[1] F. Jonsson and C. Flytzanis, *Polarization State Controlled Multistability of a Nonlinear Magneto-optic Cavity*, Phys. Rev. Lett. **82**, 1426 (1999).

[2] F. Jonsson and C. Flytzanis, *Nonlinear Magneto-Optical Bragg Gratings*, Phys. Rev. Lett. **96**, 063902 (2006).

[3] Y. R. Shen, *The Principles of Nonlinear Optics* (Wiley, New York, 1984), ISBN 0-471-88998-9.

[4] A. K. Zvezdin and V. A. Kotov, *Modern Magnetooptics and Magnetooptical Materials*, (Institute of Physics Publishing, London, 1997), ISBN 0-7503-0362-X.

[5] J. D. Jackson, *Classical Electrodynamics*, 2nd Ed. (Wiley, New York, 1975) ISBN 0-471-43132-X.

[6] F. Jonsson and C. Flytzanis, *Optical amplitude and phase evolution in nonlinear magneto-optical Bragg gratings*, J. Nonlin. Opt. Physics and Materials **13**, 129 (2004).

[7] F. Jonsson and C. Flytzanis, *Spectral windowing with chirped magneto-optical Bragg gratings*, J. Opt. Soc. Am. B **22**, 293 (2005).

[8] F. Jonsson and C. Flytzanis, *Artificially Induced Perturbations in Chirped Magneto-Optical Bragg Gratings*, in *Magneto-Optical Materials for Photonics and Recording*, Eds. Koji Ando, W. Challener, R. Gambino and M. Levy, Mater. Res. Soc. Symp. Proc. **834**, J1.8 (Materials Research Society, Warrendale, 2005).

[9] F. Jonsson, *The Nonlinear Optics of Magneto-Optic Media*, PhD Thesis (Royal Institute of Technology, Stockholm, 2000), ISBN 91-7170-575-9.

[10] P. N. Butcher and D. Cotter, *The Elements of Nonlinear Optics* (Cambridge University Press, New York, 1990), ISBN 0-521-42424-0.

[11] E. T. Whittaker, *A Course of Modern Analysis–An Introduction to the General Theory of Infinite Processes and of Analytic Functions; With an Account of the Principal Transcendental Functions*, 1st Edn. (Cambridge University Press, Cambridge, 1902).

[12] E. T. Whittaker and G. N. Watson, *A Course of Modern Analysis–An Introduction to the General Theory of Infinite Processes and of Analytic Functions; With an Account of the Principal Transcendental Functions*, 4th Reprinted Edn. (Cambridge University Press, Cambridge, 1996), ISBN 0-521-58807-3.

[13] P. F. Byrd and M. D. Friedman, *Handbook of Elliptic Integrals for Engineers and Scientists*, 2nd Edn. (Springer–Verlag, Berlin, 1971), ISBN 3-540-05318-2.

[14] M. McCall, J. Lightwave Technol. **18**, 236 (2000).

[15] A. Othonos and K. Kalli, *Fiber Bragg Gratings* (Artech House, Boston, 1999), ISBN 0-89006-344-3.

## 138. Index.

# MAGBRAGG