ETH Zurich

Simulating grating MOT designs

Author:

Johannes Eberle

Supervisors:

Gillenhaal Beck

Prof. Dr. Jonathan Home

September 15, 2023

ETH zürich

Abstract

This report presents a simulation-based study aimed at determining the optimal parameters for a grating in a magneto-optical trap (MOT) for applications in trapped ion quantum computers. The study investigates the effects of different grating geometries and parameter configurations on the performance of the MOT.

Throughout the study, we have developed a code in a modular way, allowing for easy modifications and the addition of other grating geometries and analysis methods as needed.

By leveraging advanced simulation techniques, we have identified an optimal grating design that exhibits high grating efficiency while having minimal impact on the polarization of the laser source. Specifically, we have found a grating with cylindrical holes as unit cells that exhibits a high grating efficiency where 82.4% is emitted into the first order which is emitted at an azimuthal angle of 42.6°, while inducing only a slight change in polarization. The resulting ratio of the major to the minor axis in the polarization ellipse is $r = 1.10$. Additionally, we have found parameters for a blazed grating where 74% of the source power is emitted into a single order emitted at an azimuthal angle of 43.5°. In this case, the ratio of the major to the minor axis in the polarization ellipse is $r = 1.3$.

This finding is of importance for trapped ion quantum computers, as it provides a technique to transport ions from the source to the ion trap and eventually achieving higher ion loading efficiencies.

Contents

1 Introduction

Trapped ion quantum computers have emerged as one of the leading platforms for realizing large-scale, fault-tolerant quantum computing. Central to the success of these systems is the precise control and manipulation of individual ions within an electromagnetic trap. One possible approach to achieving this involves utilizing a grating within the setup of the magneto-optical trap (MOT).

A grating MOT consists of two parts: the magnetic trapping and the optical trapping. To realize the latter, a diffraction grating is used which converts a single laser beam at normal incidence into several first order beams at an azimuthal angle of 45°, effectively creating a trapping potential.

This report focuses on a simulation-based study aimed at determining the optimal parameters for a grating in a magneto-optical trap (MOT) for trapped ion quantum computers. Our objective is to gain deeper insights into how different grating parameters impact the performance of the MOT.

In the forthcoming sections, we will discuss the principles of Fraunhofer diffraction, light polarization, and grating MOTs. Subsequently, we will introduce the simulation methods and the implemented code. Finally, we will present and analyze the outcomes of the simulations,

1.1 Fraunhofer diffraction

Diffraction occurs when waves encounter obstacles, pass through apertures or scatter from surfaces with periodic structures.

In general, we distinguish between two major regimes: the near field and the far field.

In the near field, which is also known as the Fresnel regime, the observation point is relatively close to the diffracting elements compared to their overall dimensions. As a result, the diffracted wave fronts remain curved, and the individual diffracted waves interfere with each other.

On the other hand, the far field, also referred to as the Fraunhofer regime, corresponds to observation points that are sufficiently far away from the diffracting element - this is

described by the Fraunhofer condition:

$$N_F^{'} = b^2/\lambda d \ll 1, \tag{1}$$

where $N_F^{'}$ is the Fresnel number, d is the distance between the diffracting element and the observer, λ is the wavelength and b is the size of the diffracting element (e.g. the size of the aperture) [14]. In this regime, the wave fronts can be considered nearly planar, and the individual diffracted waves are approximately parallel (see figure 1). As a consequence, the interference patterns formed are simpler and can be described using the principles of Fourier optics. In Fourier optics, the propagation of light is described using the Fourier transform. By decomposing complex light patterns into simpler sinusoidal components through Fourier transforms, spatial frequencies are used to analyze how light propagates through optical systems, such as lenses or apertures [14]. [14].

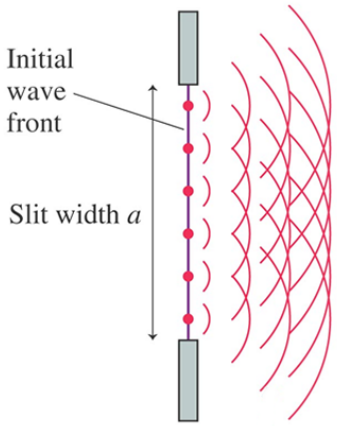


Figure 1: Huygen's principle illustrates the formation of wavelets. In the far field, the wavefronts can be considered parallel. Figure taken from [17].

To do so, let us assume the simple case of a transmission grating at $z = 0$ with the aperture function

$$p(x, y) = \begin{cases} 1, & \text{inside the aperture} \\ 0, & \text{outside the aperture} \end{cases} \tag{2}$$

and the intensity of the incoming light is I_i. In the Fraunhofer approximation, the complex amplitude at $z = d$, where d is the distance from the grating, is then given by

$$g(x,y) \approx \sqrt{I_i} h_0 P(\frac{x}{\lambda d}, \frac{y}{\lambda d}), \tag{3}$$

where $P(\nu_x, \nu_y)$ is the Fourier transform of $p(x,y)$ and $h_0 = \frac{i}{\lambda d}\exp(-ikd)$.

The intensity at (x,y,d) is therefore given by

$$I(x,y) = \frac{I_i}{(\lambda d)^2} \left| P(\frac{x}{\lambda d}, \frac{y}{\lambda d}) \right|^2. \tag{4}$$

Often, the geometry is more complex and the usage of different materials, each with unique optical properties, lead to a non-trivial transmission and therefore to a non-trivial aperture function. Hence, solving the problem analytically becomes unfeasible, and a numerical simulation is needed to determine the diffraction pattern.

We can already estimate the period of the grating by applying the grating equation, which reads [5]:

$$sin(\theta_n) = \frac{n\lambda}{d} \tag{5}$$

that relates the emission angle θ_n of the n-th order at wavelength λ and normal incidence of light to the periodicity d of the grating (see figure 2).

We see that the range of emission angles of the first order between 40° and 50° corresponds to a period range between 550 nm and 660 nm.

1.2 Polarization

The polarization of light $\xi(\mathbf{r}, t)$ is defined as the space- and time dependent direction of the electric field [14]. For a plane wave at a fixed position $\mathbf{r}$, the endpoint of the polarization vector moves along an ellipse which lies in the plane that is orthogonal to the propagation direction of the electromagnetic field. The polarization ellipse - among other descriptions, such as the Poincaré sphere, can be used to characterize the polarization. In the following chapters, we will use two parameters that fully characterize the normalized polarization axis: The ratio of the major to the minor axis and the angle of the major axis relative to the axis of P-polarization. Note that we impose the condition $(a^2 + b^2) = 1$,

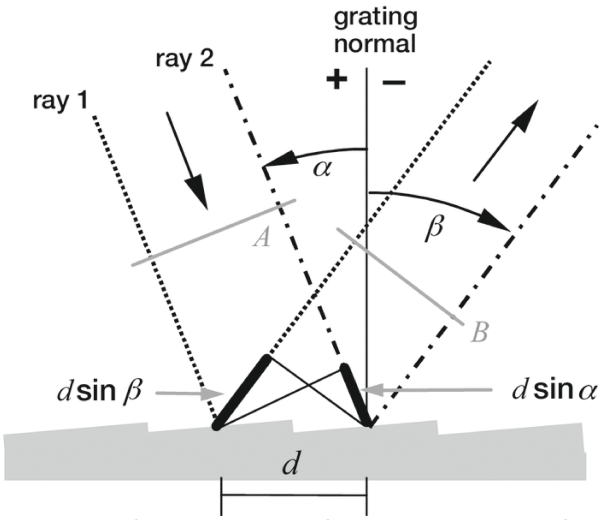


Figure 2: A reflective grating with incoming and outgoing rays. The path difference between two reflected rays is $d(sin\beta + sin\alpha)$. For constructive interference, we require this difference to be a multiple of the wavelength, such that $n\lambda = d(sin\beta + sin\alpha)$, where $n \in \mathbb{Z}$ is referred to as the diffraction order. For normal incidence ($\alpha = 0$) and by defining $\theta_n = \beta$, we obtain the grating equation 5. Figure taken from [13]

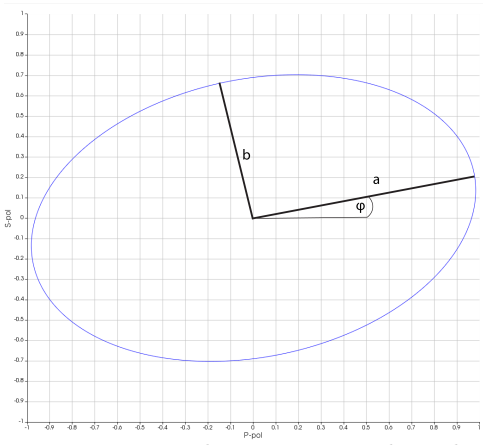


Figure 3: The ratio of the major axis a to the minor axis b and the angle φ of the major axis determine the shape of the ellipse.

where a and b are the lengths of the major and the minor axes, respectively. In general, the reflection off a medium is polarization-dependent, and so is also the diffraction off a reflection-grating. This needs to be considered when choosing the geometry of the grating. Circularly polarized light is defined by a ratio $r = 1$ of the major to the minor axis. In the following sections, this ratio will be simply referred to as "ratio".

1.3 Grating MOT

Charged ions can be confined using magnetic and electric fields [2]. This technique, however, does not work for neutral atoms. A commonly used method of confining neutral atoms is magneto-optical trapping. Here, a combination of magnetic fields and optical

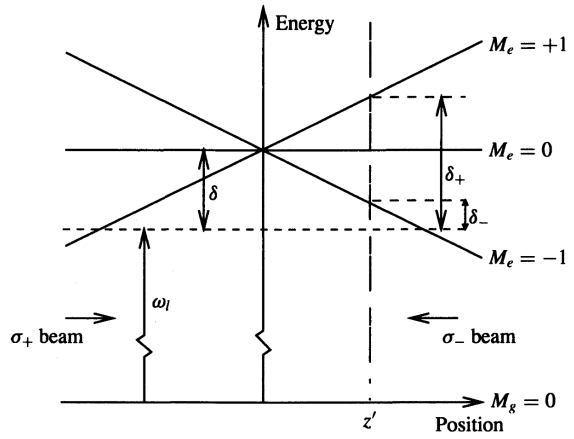


Figure 4: The energy levels of the atom in a magnetic field. Figure taken from [6] with permission from Springer Nature.

forces is used to cool and confine atoms. This can be done in all three spatial dimensions. To illustrate the fundamental concept, we will focus solely on the trapping mechanism in one direction. However, the trapping in the other two dimensions functions similarly. To create a confining potential in z-direction, the magnetic field is chosen to be linearly inhomogeneous: $B(z) = B_0 z$ [6]. We now focus on an atom with a ground and an excited state. The ground state has a total angular momentum of $J = 0$, while the excited state has a total angular momentum of $J = 1$. The magnetic field introduces a location-dependent Zeeman shift of the exited state levels with $m_J = \pm 1$ and therefore lifts the degeneracy of the excited state. Additionally, two counter-propagating beams with opposite circular polarization σ_+ and σ_- are focused on the center of the trap. The beams are red-detuned by δ, such that $\omega_l + \delta = \delta_U$, where δ_U is the energy difference between the $\langle J = 0, m_J = 0)|$ ground state and the $\langle J = 1, m_J = 0|$ excited state.

If an atom is located in the center of the trap, it will not experience a Zeeman shift due to the vanishing magnetic field and therefore the interaction probability between the photon and the atom remain low. If the atom is moving in $\pm z$ direction, the energy difference between the $\langle J = 1, m_J = \pm 1|$ and the $\langle J = 1, m_J = 0|$ state increases until it is equal to δ. It will then absorb a $\sigma_\mp$ polarized photon that is incident from the $\mp z$ direction which results in a restoring force the center of the trap [6].

In a grating MOT, a grating is illuminated by a laser and due to diffraction, intensity peaks of higher order can be observed. In our case, we are determining an optimal grating design for which the first-order beams are high in intensity and approximately at a 45° angle to the surface normal of the grating.

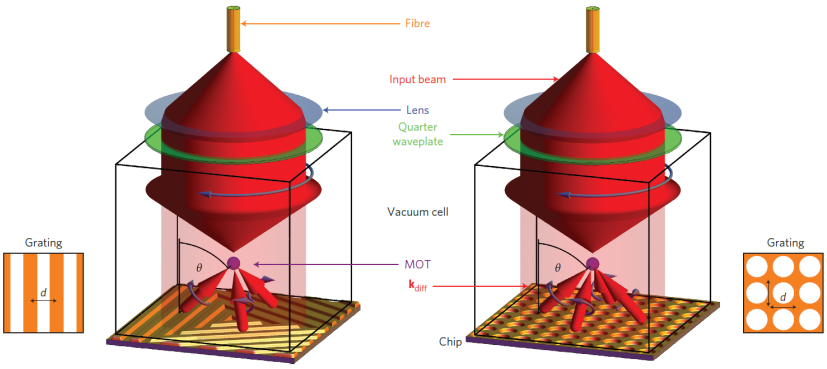


Figure 5: General design of a grating MOT. The left figure depicts a linear grating and the right figure depicts a circular grating geometry. Both gratings produce a diffraction pattern that is suitable to trap ions. The red arrows show the propagation vectors of the first-order peaks. Figure taken from [7].

In 2013, Nshii et al. found that a grating of cylindrical elements (see figure 5 on the right side) achieves the highest number of trapped atoms. [7]. Based on this observation, we will simulate a grating MOT with a cylindrical grating, as depicted in figure 5. Additionally, we will simulate a cylindrical grating with a sidewall angle as well as a linear blazed grating and determine the optimal parameters for a maximal trapping efficiencies.

Lastly, the source produces circularly polarized light. The ideal gratingMOT conserves polarization and therefore, we aim to find a geometry that leaves the polarization of the first order as little affected as possible. As a figure of merit, we use the ratio of the major to the minor axis (see section 1.2), that is $r = 1$ for circularly polarized light, and $r > 1$ for non-circularly polarized light.

1.4 Finite Difference Time Domain method

For the simulation, the software Lumerical was used. Specifically, the Finite Difference Time Domain (FDTD) method was used for all simulations.

The Finite-Difference Time-Domain (FDTD) method is a numerical technique used for solving Maxwell's equations in both time and space domains. It provides a powerful computational approach for simulating and analyzing a wide range of electromagnetic phenomena, such as wave propagation, scattering, and interaction with various structures [9].

The method is based on Yee's algorithm that Kane Yee introduced in 1966 [18] and that

gained popularity with increasing computing power.

In the FDTD (or Yee) method, the space volume is discretized by dividing it into a rectangular and structured mesh. For each mesh point, the time-dependent solution of Maxwell's equation, i.e. the electric and magnetic field components, are calculated [3]. By discretizing the space and time, the method approximates the continuous Maxwell's equations.

The FDTD method operates in a time-stepping manner, advancing the fields through time in small increments. At each time step, the electric and magnetic fields are updated based on their previous values and the interactions with the surrounding environment, including sources and boundaries. This iterative process allows the simulation to capture the temporal and spatial behavior of electromagnetic fields accurately. The derivatives in the equations are expressed as finite differences between neighboring mesh points [18]

One of the notable advantages of the FDTD method is its ability to handle complex geometries and material properties. It can simulate the interaction of electromagnetic fields with various objects, including metallic structures, dielectric materials, and dispersive media. This makes it a versatile tool in photonics. [9].

There are several mesh termination techniques, out of which the ABC (Absorbing Boundary Condition) and the PML (Perfectly Matched Layer) are the most common [3]. Both techniques absorb the light at the boundary, however, the PML method is generally considered the state-of-the-art [15]. A more elaborate description of these two methods and the different variations within these methods can be found in [3] and [15].

The periodic boundary is another frequently utilized mesh termination. It allows simulating a single unit cell in a periodic structure, such as a grating. The fields leaving trough one side of the cell are simply injected at the opposite side.[11]

1.5 Particle Swarm Optimization

Originally developed by James Kennedy, Russell C. Eberhart and Yuhui Shi [4] [16], the particle swarm optimization (PSO) is an optimization method that does not use the gradient of the function that is optimized. The general idea is that a swarm of particles is used to search the search space for the global maximum or minimum. An analogy

in nature can be found in a bee swarm searching for the best nectar sources, where the bees communicate with each other. A particle is a point in the search space and the properties are its function value, location and velocity. At the beginning of each iteration, the particle moves one step in the search space along the velocity vector. The particle then receives information from its informants and based on this, the velocity is updated. The informants are other particles - the choice of how many particles serve as informants varies between different implementations of the PSO [1]. Specifically, it is necessary to find a balance here between the propagation speed and the diversity. A small number of informants leads to a large diversity, as the single particles are less dependent on each other. However, as less information is exchanged, the particles take more iterations to find maxima. On the other hand, a large number of informants leads to less diversity, as many particles are moving uniformly since they have the same information. However, as more information is exchanged, the particles need less iterations to find maxima. [1].

1.6 First order efficiency

The grating efficiency, or, as we will call in the following sections, first order efficiency η is defined here as $\eta = \frac{P_1}{P_{source}}$, where P_1 is the integrated intensity in the far field of the first order intensity peak with the highest intensity [1] and P_{source} is the power of the source. In a grating where the unit cell is radially symmetric, all first order intensity peaks have the same intensity. However, in the opposite case, such as in a blazed grating, the first order peaks do not have the same intensity in general. In this case, the peak with higher intensity is determined and used for calculating the efficiency. A more elaborate discussion can be found further below. The integrated intensity is determined using the built-in grating projection functions provided by Lumerical. Specifically, the function *grating* returns the fraction of transmitted power into each grating order as a fraction of the source power. In the following sections, we use the terms "grating efficiency" and "first order efficiency" interchangeably. This choice stems from the parameter regime under consideration, where higher orders beyond the first order are significantly suppressed. As a result, the overall efficiency of the grating is primarily determined by the first order efficiency.

[1] To avoid confusion, it is important to note that the far field functions were not used in any calculation. Only the grating functions were used, which calculate the far field for a grating. More information on the differences between these two cases can be found in section 1.7

1.7 Far field

The far field is defined as the field at a distance from the grating much larger than the size of the grating. In Lumerical, there exist built-in functions to gain information on the far field - here, the distance to the grating is 1 m [8]. However, it is important to mention that when using the far field functions in Lumerical, only the reflection off a single unit cell is considered. One can choose to simulate a number of periods that are considered, however, the developers are advising to use the built-in grating projections functions [10]. For this project, only grating projection functions were used for calculations. Far field projection functions were found to give slightly different results compared to the grating projection functions, if no periodicity was chosen. With increasing number of periods in the far field projection, the peaks become sharper and equal to the grating projection.

2 Methods

The goal of the simulation is to find the optimal design of a grating MOT. This problem was approached by choosing different geometries and varying their parameters. The parameters that can be adjusted in this setting are: the period, the radius of the hole (cylindrical grating) or the width of the hole (linear grating), [2] the depth of the hole, the thickness of the coating and the material of the coating. Figure 6 shows the different geometries that were used for the grating. There is a large space of possible combinations of the parameters, and it is important to understand how varying each parameter affects the diffraction pattern of the grating. Therefore, several simulations were performed, for which the results will be shown in the following chapters. To make the code easily adaptable for different geometries and future adaptations, it is built in a modular way. This means that the different parts of the simulation are written in separate scripts, which are then executed one after another by a meta-script. The separation is as follows:

1. Set up the model: define all physical components of the grating
2. Set up light sources and monitors
3. Set up analysis of monitor results

[2] A distinction between the hole width and the hole radius is only relevant (and only makes sense) in 3d simulations. In 2d simulations, both terms can be used interchangeably

4. Define the parameter sweep or parameter optimization

Therefore, different geometries can be simulated in the exact same way while minimizing the risk of errors caused by transferring code between different scripts.

2.1 Code implementation details

In the following, a detailed overview over each code section is given. More details can be found in the code (see appendix).

2.1.1 Metascript

The metascript calls every code block that is needed for the specific simulation. The variables for determining the specific geometry of the grating are stored in the model as user properties. This makes sure that a parameter sweep or an optimization sweep can change these properties. All code blocks are called in the setup script of the model which itself is executed in the metascript - except for the parameter sweep or optimize sweep, which are called directly in the metascript. Therefore, while performing a sweep, the user properties can be updated and since the setup script of the model is executed in every step of the sweep, the geometry is updated as well. At the end of the script, a file name and storage location is specified.

2.1.2 Set up the grating geometry

The implementation of the grating geometry is straightforward: The rectangular silicon base and the other elements of the grating are implemented as components in a structure group. The entire structure is implemented in 3d and rotated so that a 2d simulation is performed in the x-y plane and a 3d simulation is possible by extending the simulation region along the z-axis. This is necessary due to the way the simulation region is configured in Lumerical.

2.1.3 Set up light sources and monitors

All light sources and monitors are implemented in 3d and extend over a range wider than the simulation region. Two light sources are implemented as plane waves with a relative polarisation angle of 90° and a relative phase of 90°. The resulting light wave

is a circularly polarised plane wave. The monitors record the field at the top end of the simulation region.

2.1.4 Analysis of monitor results

The goal of the analysis is to determine the intensity peaks in the far field and to check whether the first order intensity peak is emitted close to an angle of 45°. The code uses built-in grating functions and polarization analysis provided by Lumerical and determines all necessary information, such as the transmission of the source power into the far field, the emission angle and transmission of the first order intensity peaks as well as the polarization ellipses of all grating orders.

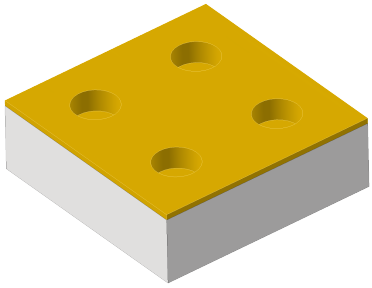
2.1.5 Define the parameter sweep or parameter optimization

As a final step, the parameter sweep or parameter optimization is defined. For each case, there exists a separate script where the sweep parameters and the parameter ranges are defined. A more elaborate discussion of the parameter sweeps and optimization can be found further below.

2.2 Grating geometries

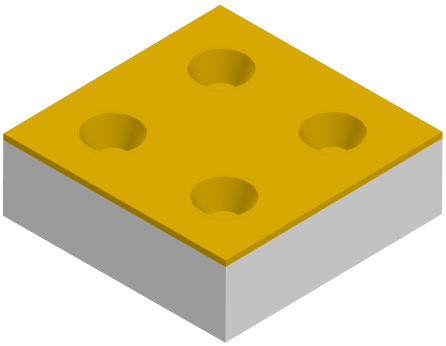
2.2.1 Linear (blazed) grating

A common type of grating is the blazed grating, which resembles the shape of a sawtooth. In this context, we will not restrict ourselves to the conventional blazed grating. Instead, we will adopt a geometry similar to the one illustrated in figure 6e. It is important to note that by choosing the period to be equal to the hole width, a sawtooth geometry can be implemented. Therefore, the set of sawtooth geometries is contained within the set that we examine. The blazed grating is not radially symmetric, from which follows that the diffraction pattern is not symmetric either. The ideal diffraction pattern consists of a suppressed zeroth order peak as well as three suppressed first order peaks, while one first order peak (in our case $(n, m) = (1, 0)$) is maximal in intensity. An example of such a diffraction spectrum can be found in section 3. The grating needs then to be assembled similarly to the linear grating in figure 5 so that the high intensity first order peaks are directed towards the center of the trap. Simulating the grating in a 2d FDTD simulation



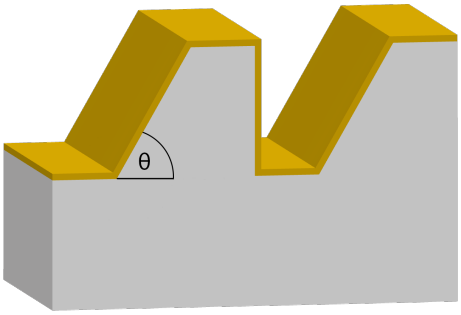
(a)

(b)



(c)

(d)



(e)

Figure 6: **a** - **e** show several unit cells of the grating for different grating geometries. **a** and **b** depict the circular grating that was used in the simulations. **a** shows the grating from the top-view, **b** shows a cross-section. **c** and **d** show the cylindrical grating with a sidewall angle. **e** shows the blazed grating, θ is the blaze angle. Here it can be seen that for a blaze angle of 90°, the grating is simply linear, without a blazing. Note that the sidewall angle in **c** is defined analogously to the blaze angle in **e**. The wafer consists of silicon, while the top layer is composed of a coating material.

proved to be equivalent to simulating it in a 3d FDTD simulation. This can be explained by the fact that the structure in 3d is an infinite extension of the 2d structure along the third axis. It is important to note that the 2d simulation accurately models both s- and p-polarized light, and thus also circularly polarized light. However, the built-in polarization analysis functions provided by Lumerical can only be applied in a 3d simulation.

2.2.2 Cylindrical grating (with sidewall angle)

Nshii et al. have found that the highest grating efficiency in a grating MOT is achieved with a cylindrical grating [7], as depicted on the right side of figure 5. Four unit cells of the grating and a cross-section are shown in figure 6. The cylindrical grating can be adjusted by adding a sidewall angle, analogously to the linear blazed grating. An example is depicted in figure 6d. The sidewall angle Θ is defined analogously to the blaze angle in the linear blazed grating. No 2d FDTD simulation was found to be equivalent to a 3d simulation of the structure. Therefore, it is necessary to use the 3d FDTD simulation. In the case of a grating with no sidewall angle, the optimal hole depth is $\lambda/4$ [12], where λ is the wavelength. This can be explained by the fact that a wave that is reflected at the bottom of the hole, has a phase shifted by $\lambda/2$ compared to a wave that is reflected outside the hole - leading to destructive interference.

2.3 Simulations

2.3.1 Parameter sweeps

In Lumerical, parameter sweeps can be performed, where one chooses the range of a parameter and the number of steps k. Lumerical allows for a parameter sweep of any number of parameters. It needs to be noted, however, that the number of simulations per n-parameter sweep is k^n, where k is the number of steps per sweep (the same number of steps per sweep is chosen for all simulations). Therefore, we limit ourselves to varying one or two parameters. For the 2d FDTD method, a 3-parameter sweep is feasible, however, for the 3d FDTD method, this would take too long. There exists a set of parameters where the diameter of the hole is greater than the grating period. In this regime, the structure is simply a flat surface of silicon that is coated. There are two ways for setting the period of the grating. First, the period can be set as an absolute number. Second,

it can be set as a multiple of the radius - by choosing the factor to be greater than 2, the regime can be avoided in which the diameter is greater than the period. Both ways are implemented in the code, and they can be chosen depending on whether a sweep or an optimization sweep is performed. When performing a sweep, the period is set as an absolute number for visual reasons (this makes the figures easier to read). When performing an optimization sweep, the period is set as a multiple of the diameter, since here no visualization of the optimization process is made. [3]

To find optimal parameters, first the hole depth is set to 105.75 nm and the coating thickness is set to 150 nm. The selection of the coating thickness is based on the suitability for fabrication, ensuring the resulting gratings meet the necessary fabrication criteria. The hole depth is chosen to be $\lambda/4$, as it is described above, where λ is the wavelength. These values are flexible and can be adjusted as needed based on specific fabrication requirements. For the cylindrical grating, a parameter sweep over the radius of the hole and the period of the grating is performed. From this sweep, the optimal radius and period are determined. These are then used in a parameter sweep over the hole depth and the coating thickness. Finally, a sweep over the sidewall angle is performed. This determines all necessary parameters.

2.3.2 Parameter optimization

A particle swarm optimization method is provided by Lumerical. Since the coating thickness is irrelevant, as discussed in section 3, this parameter will not be included as an optimization parameter - instead a common thickness of 150 nm is chosen. The hole depth [4], hole width/radius and period are set as optimization parameters, i.e. the parameters that are variable throughout the search. Additionally, for the linear and cylindrical blazed grating, the blaze or sidewall angle is set as an optimization parameter. For the PSO, it is necessary to choose a particle number as well as a maximum step number. The particle number corresponds to the number of simulations that are performed in each iteration

[3] Alternatively, one can create a list of all possible combinations of the period and the radius, which lie outside the regime where the hole diameter is larger than the period. This list is then used for setting up the sweep over the period and the radius. This method was implemented together with a separate analysis script to post-process the results and create figures. Details as well as the code are available upon request.

[4] Below we discuss that for the silver and aluminum coating, the hole depth does not need to be included as a variational parameter. However, we are still including it. The PSO was found to converge to an optimal solution, no matter if the hole depth is included as variational parameter or not

step, while the maximum step number corresponds to the maximum number of iterations. As recommended by M. Clerc [1], a particle number of 20 is chosen for all optimizations. The maximum step number is set to 100. The value to be maximized by the PSO is the efficiency of the first order. There exists an additional condition that the emission angle of the first order intensity maximum ϕ is within the range $42° < \phi < 48°$. To include this constraint, the grating efficiency is set to zero for a solution that does not fulfill this condition. We have not found a way to perform the PSO on the Euler cluster of ETH and due to the high computational complexity of the 3d simulation, the PSO could be only performed for 2d simulation. The parameter ranges are shown in table 1.

Parameter	Range
Hole width	$50\,\text{nm} - 400\,\text{nm}$
P_H	$2 - 5$
Hole depth	$10\,\text{nm} - 1\,\mu\text{m}$
θ_f	0 - 1

Table 1: The parameter ranges for the PSO sweep. The period P is given by $P = P_H \cdot R$, where R is the radius and the blaze angle θ is given by $\theta = \theta_F * \theta_{max}$, where θ_{max} is the angle for which the blazing forms a sawtooth shape.

3 Results

The observations mentioned here were found to be valid for all grating geometries. Details are discussed in the respective subsections. While the diffraction off the grating is dependent on the hole depth, the hole width (or radius), the period and the blaze (or sidewall) angle, it does not depend on the coating thickness. This was found by performing parameter sweeps over all mentioned parameters. Parameter sweeps over the hole depth and the coating thickness for three different coating materials can be found in figure 9. Here, the sweep is for the cylindrical geometry, however, this behavior was found for all gratings and coatings. This reduces the parameter space significantly, and the coating thickness was set to 150 nm, which is a common thickness in fabrication. An exception is the gold coating, where the first order efficiency is highest for a small coating thickness. This could be due to a reflection off the silicon for a small gold coating thickness. A further analysis was not performed, since the gold coating performs poorly at the wavelength of our setup. For all coating materials, a periodicity of the first order efficiency with respect to the hole depth was found. However, the following observation

arose: by initially optimizing hole width and period with arbitrary hole depth, followed by subsequent variations in hole depth, the peak first-order efficiency consistently coincided with the originally chosen hole depth. From this, we can conclude that the hole depth can be chosen arbitrarily. By varying the hole depth, the transmission of source power into the far field is periodic, too. However, the maxima of the first order efficiency and of the transmission generally do not overlap, which is why the values of the maxima of first order efficiency vary at each period.

3.1 Linear (blazed) grating

To find optimal parameters for the linear grating, a particle swarm optimization sweep (PSO) is performed with the parameter ranges, as shown in table 1.

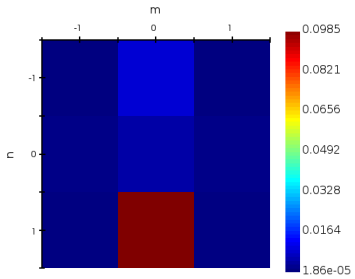
Coating material	Hole width	Period	Hole depth	Coat. Thick.	θ_f	fom
Gold	315.49 nm	632.16 nm	186.18 nm	150 nm	0.94	0.31
Silver	281.17 nm	632.04 nm	166.58 nm	150 nm	0.94	0.66
Aluminum	178.383 nm	616.67 nm	298.75 nm	150 nm	0.62	0.74

Table 2: The optimal parameters for the linear grating found with the PSO sweep for different coating materials. Coat. Thick. stands for coating thickness. θ_f denotes the fraction $\theta_f = \frac{\theta}{\theta_{max}}$, where θ denotes the blaze angle and θ_{max} is the angle for which the grating is given by a sawtooth shape. The coating thickness was fixed to 150 nm, since the diffraction is independent of the coating thickness. The figure of merit (fom) is the transmission of source power to the (n,m)=(1,0) mode. Since for the linear geometry, the maximum transmission is through this mode, this value is chosen as the figure of merit for the optimization algorithm.

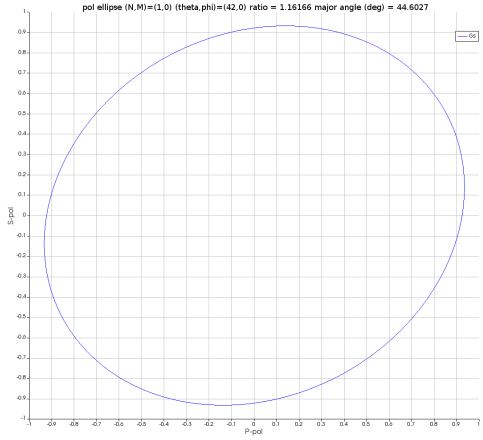
The diffraction and the polarization ellipses of the first order of the best solutions that were found are shown in figure 7. We see here that aluminum gives the highest first order efficiency and that any order than $(n, m) = (1, 0)$ is strongly suppressed. Also, a ratio of the major to the minor axis of the polarization ellipse of 1.33 is acceptable. We have therefore found a good solution to our problem.

The source light is circularly polarized. In figure 7 we see that the diffracted light is not circularly polarized anymore, which is expected and can not be avoided entirely. However, as we will show further below, the effect on the polarization is slightly larger compared to the cylindrical grating.

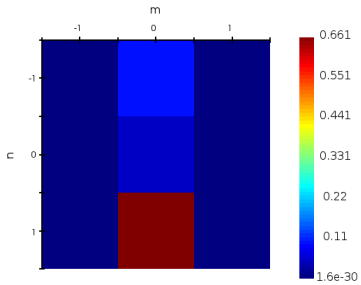
In the simulations, it was found that the regions of highest grating efficiency are located in a regime where the ratio is < 2. To reduce the ratio and therefore get closer to a circular polarization of the diffracted light, we would need to find a balance between the



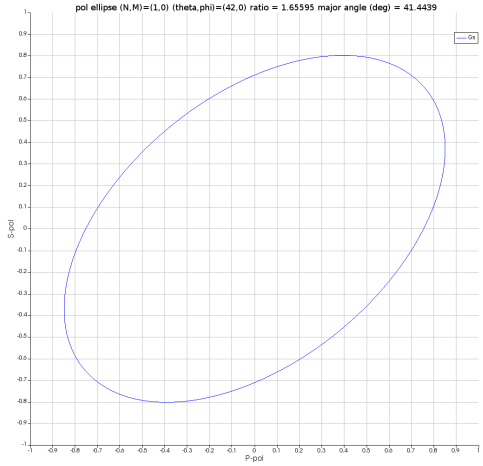
(a) First order efficiency, gold coating



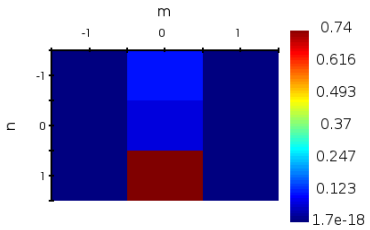
(b) Polarization ellipse, gold coating



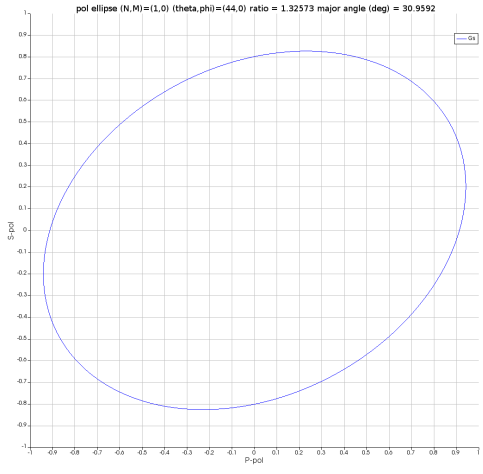
(c) First order efficiency, silver coating



(d) Polarization ellipse, silver coating



(e) First order efficiency, aluminum coating



(f) Polarization ellipse, aluminum coating

Figure 7: The efficiencies of the diffraction orders of the best solutions that were found with the PSO algorithm for different coating materials, as well as the corresponding polarization ellipses. We find that the ratio for the gold coating is the smallest, therefore the gold coating affects the polarization the least. In the case of the aluminum coating, we find a ratio of $r = 1.33$, which is acceptable.

ratio and the grating efficiency. Since we have set the priority to the grating efficiency, we will not do this. However, all the tools are given in the code and one could for example do an optimization sweep with the figure of merit [5] set to $\frac{\eta}{r}$, where η is the grating efficiency and r is the ratio of the major to the minor axis in the polarization ellipse. Since the polarization ratio makes only sense in a 3d simulation, it is necessary to perform the PSO with 3d FDTD simulations, which was not feasible given our computational resources.

3.2 Cylindrical grating (with sidewall angle)

Since the PSO sweep could not be implemented for the ETH High-Performance-Cluster Euler, no PSO was performed for this geometry. Instead, at first, a parameter sweep over the period and the hole radius was performed (see figure 8), followed by a sweep over the coating thickness and the hole depth (see figure 9), using the optimal solution from the previous sweep. From these two sweeps, the aluminum coating proved to be performing the best. For the grating with aluminum coating, a sweep over the sidewall angle was performed, again using the optimal solution of the previous sweep. Here, an optimal solution was found, maximizing the efficiency of the first order and minimizing the ratio of the polarization ellipse. When examining figure 8, two striking observations emerge. Firstly, there exist two discontinuities at a period of ca. 400 nm and at 600 nm. This could be due to plasmonic effects or due to simulation effects. The specific reason for these two discontinuities was not found. Our optimal solutions are located close to but not directly at the discontinuities. Secondly, there are areas of maxima and minima in the first order efficiency [6]. It is important to mention that the azimuthal emission angle of the first order only depends on the period. It is independent of the hole radius (see fig. 10), hole depth and the coating thickness.

In figures 11a and 11b it is visible that there are regions of higher and lower ratio of the polarization ellipse. These figures would provide important information if we optimized for the ratio. However, since we are optimizing for a high first order efficiency, we do

[5] The figure of merit is the value that the PSO algorithm will maximize

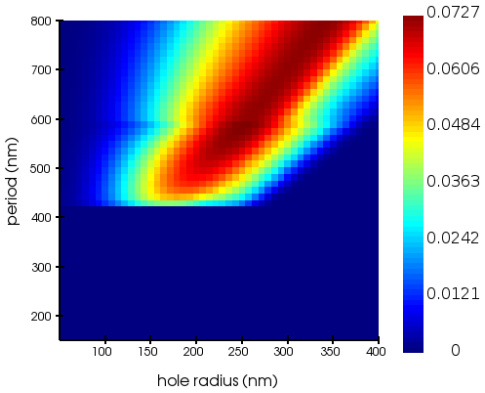
[6] Interestingly, in all three coating materials there exists the (more or less) same parameter regime for the hole radius and the period where the first order efficiency is globally maximal. In the case of the aluminum coating, there exists one further regime with slightly smaller radius and equal peak first order intensity, which is discontinuous. For a solution in the latter regime, however, the ratio of the polarization ellipse is 1.6 and therefore higher than the ratio for a solution in the former regime, where the ratio is $r = 1.1$

not analyze these figures further. More important for our purpose is figure 11c, where in a final step we see that by varying the sidewall angle in order to optimize the first order efficiency, we also minimize the ratio of the polarization ellipse. This results in a solution with high first order efficiency and low ratio of the polarization ellipse of 1.10. The diffraction and the polarization ellipse of this grating are shown in figure 12.

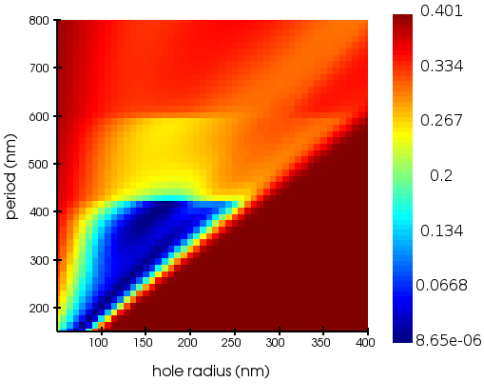
	Hole radius	Period	Hole depth	Coating thickness
Gold coating	242.86 nm	574.50 nm	105.75 nm	10 nm
Silver coating	250.00 nm	587.76 nm	105.75 nm	150 nm
Aluminum coating	242.86 nm	614.28 nm	105.75 nm	150 nm

Table 3: The optimal parameters for the cylindrical grating for different coating materials. The coating thickness was chosen to be 150 nm, since the diffraction is independent of the coating thickness, except in the case of the gold coating. The reasons for this are not known and were not further investigated, since the gold coating performs poorly due to the low reflectivity at a wavelength of 423 nm. It is striking that the values are very similar for all coating material. From this follows that the coating material has an influence on the grating, yet the influence is limited. Furthermore, the hole depth is the same for all coating materials. This value was chosen randomly for the period-radius sweep and by optimizing the first order efficiency by varying the hole radius and the period, we already found an optimal solution. The azimuthal angles of the first order are $42.6°, 44°$ and $42.6°$ for the gold, silver and aluminum coating, respectively.

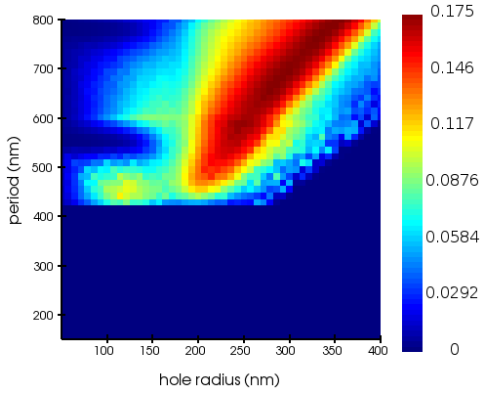
The source light is, as mentioned above, circularly polarized, and we see that the grating affects the ratio of the polarization ellipse. In figure 11a and 11b, one can see that there exist regions of higher and lower ratio. This was the case for all coatings. However, except for the sidewall angle, all parameters were chosen to maximize the first grating order efficiency, ignoring the ratio. As it is visible in figure 11c, the sidewall angle can be chosen such that the first order is still maximal, and the ratio is low. This leads to an optimal solution, with a polarization ellipse and the efficiencies of the grating as shown in figure 12. Note that the emission angle of the first order was observed to be independent of the sidewall angle.



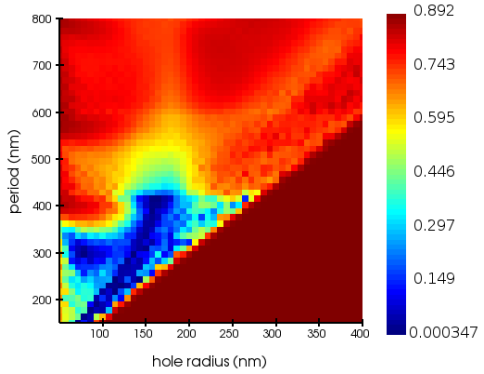
(a) First order efficiency, gold coating



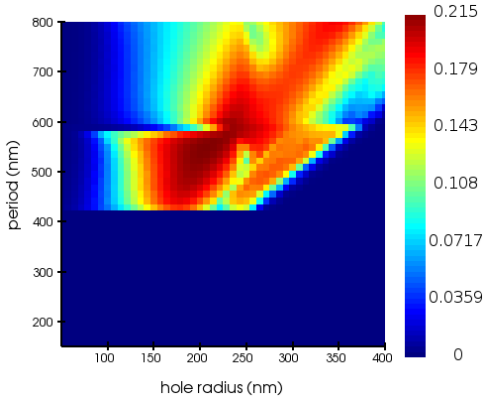
(b) Far field transmission, gold coating



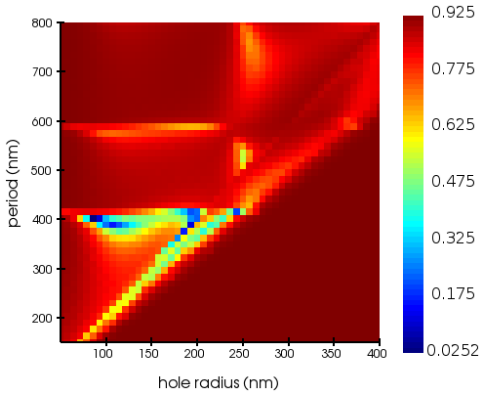
(c) First order efficiency, silver coating



(d) Far field transmission, silver coating

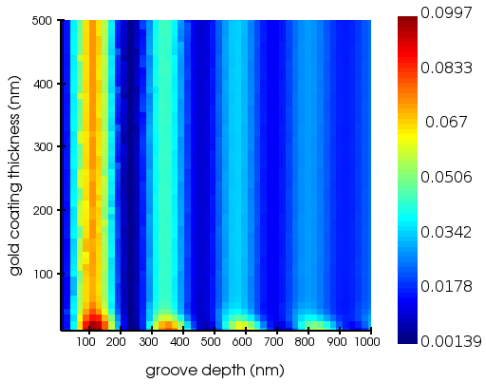


(e) First order efficiency, aluminum coating

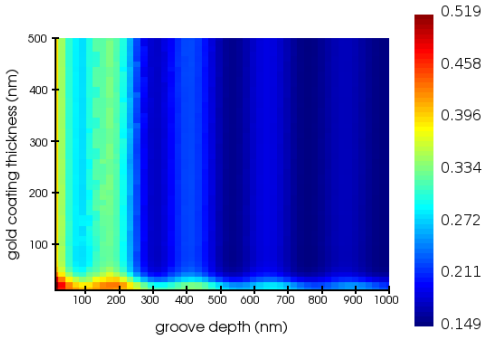


(f) Far field transmission, aluminum coating

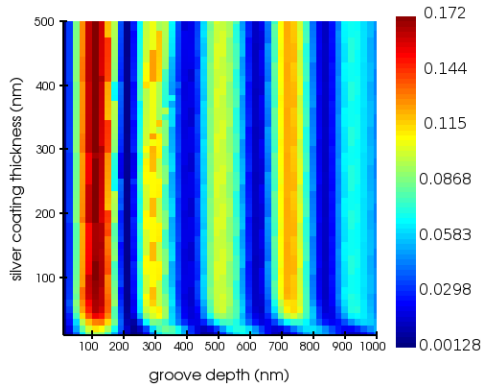
Figure 8: The first order efficiencies and the transmission into the far field for different coating materials. The varied parameters are the period and the radius, and the grating is cylindrical without a sidewall angle.



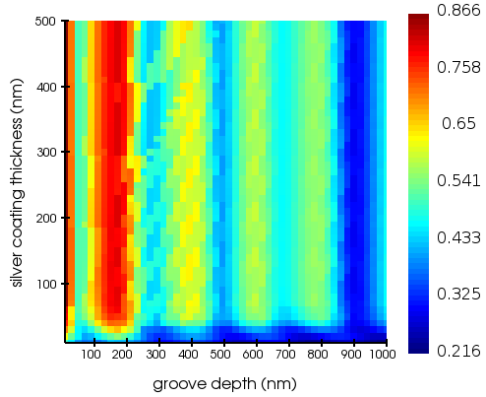
(a) First order efficiency, gold coating



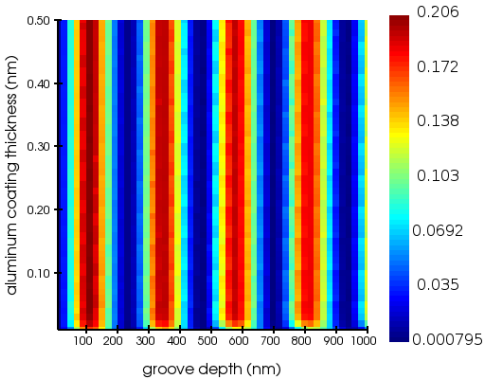
(b) Far field transmission, gold coating



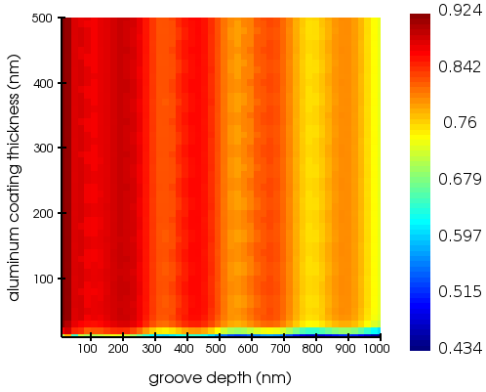
(c) First order efficiency, silver coating



(d) Far field transmission, silver coating



(e) First order efficiency, aluminum coating



(f) Far field transmission, aluminum coating

Figure 9: The first order efficiencies and the transmission into the far field for different coating materials. The varied parameters are the coating thickness and the hole depth, and the grating is cylindrical without a sidewall angle. The radius and period of the grating that resulted in maxima at the radius-period parameter sweep were utilized.

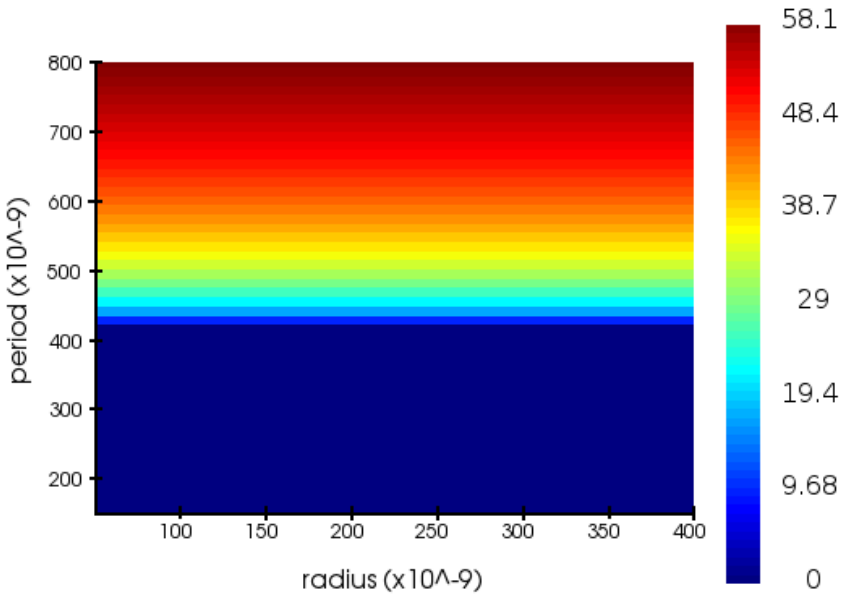


Figure 10: The azimuthal emission angle of the (n,m) = (1,0) order as a function of the hole radius and the grating period. The grating is cylindrical, and the coating is made of aluminum. It can be observed that the azimuthal emission angle only depends on the period and not on the radius.

4 Conclusion

Aluminum proved to be the most suitable coating material. We found optimal parameters for both grating geometries that we considered: one linear blazed grating and one cylindrical grating with a sidewall angle. The ratio of the major to the minor axis of the polarization ellipse in the cylindrical grating ($r = 1.10$) was slightly lower than in the linear grating ($r = 1.3$). However, one could find a solution of lower grating efficiency and lower ratio of the polarization ellipse. The tools for this are provided by the code in the appendix. This could for example be done by using as figure of merit $\frac{\eta}{r}$, where η is the first order efficiency and r is the ratio. Using several parameter sweeps, we got an understanding of how each parameter affects the diffraction, and we found the optimal parameters, which are given in table 3 and 2. The code that was developed for the simulations is provided in the appendix and may be freely used and adapted.

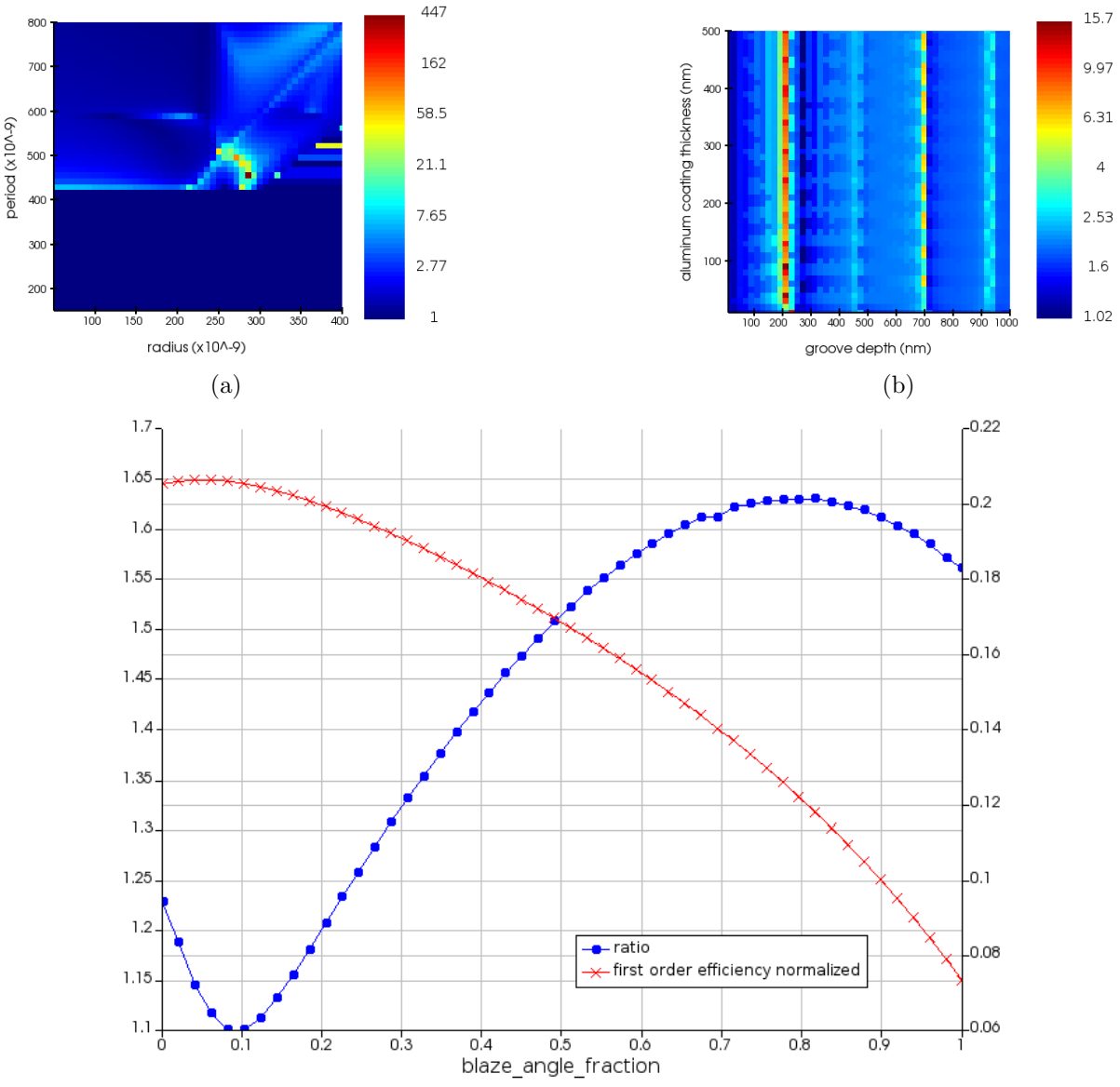
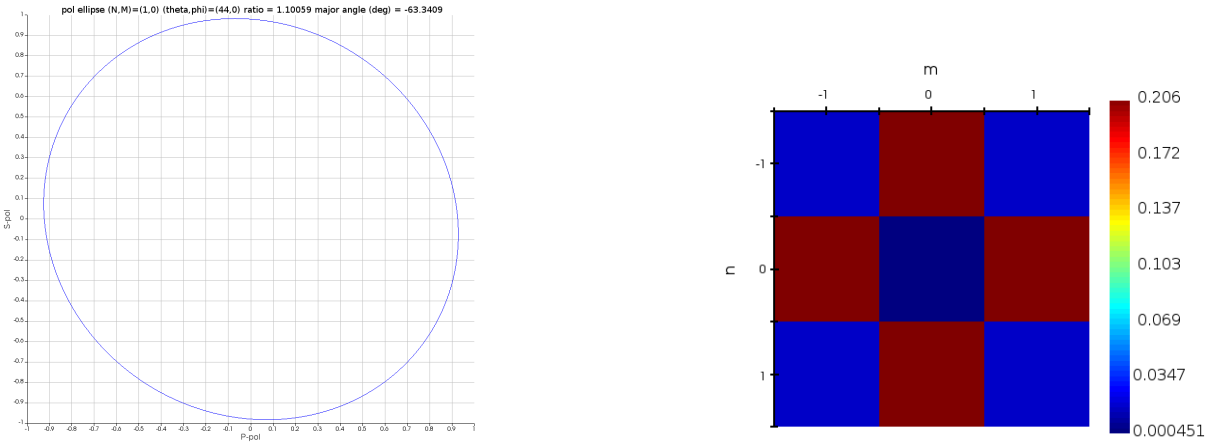


Figure 11: The ratio of the major to the minor axis of the polarization ellipse in the **a** radius-period sweep, **b** groove depth - coating thickness sweep and the **c** blaze angle fraction sweep. All three results are with the aluminum coating and cylindrical grating. In **c**, the left axis denotes the efficiency of the first order and the right axis denotes the ratio. The blazing here is in the form of a sidewall and the sidewall angle corresponds to the blazing angle in a blazed grating. A sidewall angle ratio of 1 corresponds to a geometry where the sidewall angle is such that the sidewall extends to the center point of the cylinder. In **a-b**, the color scale is logarithmic.



(a) (b)

Figure 12: The polarization ellipse and the grating efficiency for different orders for the optimal solution of the cylindrical aluminum grating with sidewall angle.

Acknowledgements

I am incredibly grateful to my supervisor, Gillen, whose mentorship has been truly invaluable. Gillen not only introduced me to a wide range of simulation tools but also shared an immense amount of knowledge about photonics, shaping my understanding and skills in this field.

I would like to extend my sincere thanks to Prof. Jonathan Home for making this project a reality and making the TIQI group a place where I felt welcome from day one.

To everyone in the group, I want to express my heartfelt appreciation. The positive and collaborative atmosphere that you all have created has made working together a great experience.

References

[1] Maurice Clerc. *Particle swarm optimization*. ISTE, 2010.

[2] Hans Dehmelt. A single atomic particle forever floating at rest in free space: New value for electron radius. *Physica Scripta*, 1988.

[3] Stephen D. Gedney. *Introduction to the Finite-Difference Time-Domain (FDTD) Method for Electromagnetics*. Morgan & Claypool, San Rafael, California, 2011.

[4] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of ICNN'95 - International Conference on Neural Networks*, volume 4, pages 1942–1948 vol.4, 1995.

[5] R. S. Longhurst. *Geometrical and Physical Optics 2nd edition*. Longman, 1967.

[6] Harold J Metcalf and Peter Van Der Straten. *Laser Cooling and Trapping*. Springer, New York, NY, 1999.

[7] C. C. Nshii, M. Vangeleyn, J. Cotter, et al. A surface-patterned chip as a strong source of ultracold atoms for quantum technologies. *Nature Nanotech 8*, page 321–324, 2013.

[8] ANSYS Optics. Far field projections in fdtd overview. `https://optics.ansys.com/hc/en-us/articles/360034914713-Far-field-projections-in-FDTD-overview`, Accessed 2023.

[9] ANSYS Optics. Finite difference time domain (fdtd) solver introduction. `https://optics.ansys.com/hc/en-us/articles/360034914633-Finite-Difference-Time-Domain-FDTD-solver-introduction`, Accessed 2023.

[10] ANSYS Optics. Grating projections in fdtd overview. `https://optics.ansys.com/hc/en-us/articles/360034394354`, Accessed 2023.

[11] ANSYS Optics. Periodic boundary conditions in fdtd and mode. `https://optics.ansys.com/hc/en-us/articles/360034382734-Periodic-boundary-conditions-in-FDTD-and-MODE`, Accessed 2023.

[12] O'Shea, Donald and Suleski, Thomas and Kathman, Alan and Prather, Dennis. *Diffractive Optics: Design, Fabrication, and Test*. SPIE Press, 2003.

[13] Christopher Palmer. *Diffraction Grating Handbook, 8th edition*. MKS Instruments, Inc., 2020.

[14] B.E.A. Saleh and M.C. Teich. *Fundamentals of photonics, 2nd edition*. John Wiley & Sons, Hoboken, New Jersey, 2007.

[15] John B. Schneider. Understanding the finite-difference time-domain method. `www.eecs.wsu.edu/~schneidj/ufdtd`, 2010.

[16] Y. Shi and R. Eberhart. A modified particle swarm optimizer. In *1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence (Cat. No.98TH8360)*, pages 69–73, 1998.

[17] Oregon State University. Single slit diffraction. `https://sites.science.oregonstate.edu/~hadlekat/COURSES/ph212/waveOptics/single-slit.html`, Accessed 2023.

[18] Kane Yee. Numerical solution of initial boundary value problems involving maxwell's equations in isotropic media. *IEEE Transactions on Antennas and Propagation*, 14(3):302–307, 1966.

Appendix: Code

Meta script

```
1  newproject;
2  deleteall;
3  clear;
4  switchtolayout;
5
6  # The gold and the aluminium coating material is taken from the following
       material database. The database is available upon request.
7  importmaterialdb("/scratch/Gillen_ChipDesign/MaterialData/
       LumericalMaterialData.mdf");
8
9  ##### define variables for simulation ##########
10 # Input properties:
11 # grating_geometry:  This defines the geometry.
12 #      The options are: linear_one (linear blazed grating), linear_two (
       linear grating with symmetrical blazed grating)
13 #           cylindrical_zero (cylindrical grating), cylindrical_two (
       cylindrical grating with side wall angle)
14 #
15 # simulationdim:   The dimension of the simulation - 2 for 2d and 3 for 3d
16 #   true_period:        "True", if in a sweep the period itself is a
       parameter,
17 #                       "False", if in a sweep the period_radiusfraction is
        a parameter
18 # coating_material:  The material of the coating
19 #   folder_path:  The absolute path of the folder in which all scripts are
       located
20 #
21 ################################################
22
23 folder_path = "/scratch/SemesterProjects/gratingMOT/SimulationsJohannes/
       scripts/modular scripts V2"
24 grating_geometry = "linear_one";
25 simulationdim = 2;
26 true_period = "False";
27 coating_material = "Ag (Silver) - Palik (0-2um)";
28 #coating_material = "Au (Gold) - Palik";
```

```
29 #coating_material = "Al (Aluminium) - Palik";
30 sweep = "two_d_sweep";
31 sweep = "optimize_sweep";
32
33 adduserprop("folder_path", 1, folder_path);
34 adduserprop("coating_material", 1, coating_material);
35 adduserprop("lambda_um", 2, 0.423e-6); # the wavelength of the source
36 adduserprop("coating_thickness", 2, 0.15e-6); # the coating thickness
37 adduserprop("groove_depth", 2, 98.55e-9);
38 adduserprop("hole_radius", 2, 206.54e-9);
39 adduserprop("period_radius_fraction", 0, 656.11/206.54);
40 adduserprop("blaze_angle_fraction", 0, 0.77);
41 adduserprop("grating_geometry", 1, grating_geometry);
42 adduserprop("simulationdim", 0, simulationdim);
43 adduserprop("true_period", 1, true_period);
44
45 ##########################################################
46
47 clearpath;
48 addpath(folder_path);
49 geometry_universal; # inserts the code of the script geometry_universal
       which cotains the function geometry(...)
50 sources_and_monitors_universal; # inserts the code of the script
       sources_and_monitors_universal which contains the function
       sources_monitors(...)
51 geometry("True", grating_geometry, coating_material); # build the grating
52 sources_monitors("True", simulationdim, true_period); #set up all sources
       and monitors
53
54 select("::model");
55 # This script will be called every time the model is set up - for example
       in a parameter or optimization sweep
56 set("Setup script", '
57     addpath(folder_path);
58     geometry_universal;
59     sources_and_monitors_universal;
60
61     geometry("False", grating_geometry, coating_material);
62     sources_monitors("False", simulationdim, true_period);
```

```
63 ');
64 runsetup;
65
66 analysis_grating;
67
68 if(sweep == "two_d_sweep") {
69  two_d_sweep;
70 }
71 else if (sweep == "optimize_sweep") {
72  optimize_sweep;
73 }
74
75 # save the simulation file
76 basename = "test";
77 f_save_name = folder_path+"/"+basename;
78 save(f_save_name);
```

Geometry setup

```
1  function geometry(firstsetup, grating_geometry, coating_material) {
2      ##########################################################################################
3      # input:
4      # firstsetup: boolean
5      #           - true if this function is used to set up the geometry in a
6      #             new simulation
7      #           - false if this function is called to update parameters
8      # grating_geometry: char string-determines the geometry of the grating;
9      #   The options are: linear_one (linear blazed grating),
10     #                    linear_two (linear with symmetrical blazing)
11     #                    cylindrical_zero (cylindrical grating),
12     #                    cylindrical_two (cylindrical with side wall angle)
13     # This function can easily be extended by other geometries!
14     #
15     # coating_material: char string - specifies the material of the coating
16     #
17     # output: nothing - this function builds the grating or updates all
18     #           necessary parameters
19     ##########################################################################################
20
21     select("::model");
```

```
22      hole_radius = get("hole_radius");
23      coating_thickness = get("coating_thickness");
24      groove_depth = get("groove_depth");
25      blaze_angle_fraction = get("blaze_angle_fraction");
26
27
28      # Substrate and coating layer are the same in every geometry
29
30      if(firstsetup == "True") {
31
32          addstructuregroup;
33          adduserprop("coating_material", 1, coating_material);
34          adduserprop("hole_radius",2, hole_radius);
35          adduserprop("coating_thickness",2, coating_thickness);
36          adduserprop("groove_depth",2, groove_depth);
37          adduserprop("blaze_angle_fraction",0, blaze_angle_fraction);
38          adduserprop("firstsetup",1, firstsetup);
39          set("name", "structure group");
40          set("x", 0);
41          set("y", 0);
42          set("z", 0);
43      }
44      else {
45          select("::model::structure group");
46          set("hole_radius", hole_radius);
47          set("coating_thickness", coating_thickness);
48          set("groove_depth", groove_depth);
49          set("blaze_angle_fraction", blaze_angle_fraction);
50      }
51
52      script = '
53          deleteall;
54
55          if(firstsetup == "True") {
56              addrect;
57           set("name", "Substrate");
58           set("material", "Si (Silicon) - Palik");
59              set("x", 0);
60              set("x span", 100e-6);
```

```
61              set("y", 0);
62              set("y span", 100e-6);
63              set("z min", -20e-6);
64              set("z max", 0);
65              set("render type", 1);
66              set("detail", 0.3);
67              set("alpha", 0.3);
68              set("override mesh order from material database", 1);
69              set("mesh order", 4);
70
71              addrect;
72              set("name", "coating layer");
73              set("material", coating_material);
74              set("x", 0);
75              set("x span", 100e-6);
76              set("y", 0);
77              set("y span", 100e-6);
78              set("z min", 0);
79              set("z max", coating_thickness);
80              set("render type", 1);
81              set("detail", 0.3);
82              set("alpha", 0.3);
83              set("override mesh order from material database", 1);
84              set("mesh order", 3);
85          }
86          else {
87              select("structure group::coating layer");
88              set("z min", 0);
89              set("z max", coating_thickness);
90          }
91      ';
92      # End of first part of script, Now, the script is extended, depending
            on the geometry
93
94      if(grating_geometry == "cylindrical_zero") {
95          script = script + '
96              if(firstsetup == "True") {
97                  addcircle;
98                  set("name", "coating circle");
```

```
99                  set("material", coating_material);
100                 set("render type", 1);
101                 set("detail", 0.3);
102                 set("alpha", 0.3);
103                 set("override mesh order from material database", 1);
104                 set("mesh order", 3);
105                 set("radius", hole_radius + coating_thickness);
106                 set("z max", 0);
107                 set("z min", -1*groove_depth);
108
109                 addcircle;
110                 set("name", "Vacuum circle");
111                 set("material", "etch");
112                 set("render type", 1);
113                 set("detail", 0.3);
114                 set("alpha", 0.3);
115                 set("override mesh order from material database", 1);
116                 set("mesh order", 1);
117                 set("radius", hole_radius);
118                 set("z max", coating_thickness);
119                 set("z min", coating_thickness - groove_depth);  # -(
                        groove_depth - coating_thickness)
120             }
121             else {
122                 select("structure group::coating circle");
123                 set("radius", hole_radius + coating_thickness);
124                 set("z min", -1*groove_depth);
125
126                 select("structure group::Vacuum circle");
127                 set("radius", hole_radius);
128                 set("z max", coating_thickness);
129                 set("z min", coating_thickness - groove_depth);  # -(
                        groove_depth - coating_thickness);
130             }
131     ';
132     }
133
134     else if(grating_geometry == "linear_one") {
135         script = script + '
```

```
136             if(firstsetup == "True"){
137                 addpoly;
138                 set("name", "coating poly");
139              set("material", coating_material);
140              set("render type", 1);
141              set("detail", 0.3);
142              set("alpha", 0.3);
143              set("override mesh order from material database", 1);
144              set("mesh order", 3);
145              x = hole_radius + coating_thickness;
146              set("vertices", [x,2e-6;-x,2e-6;-x,-2e-6;x,-2e-6]);
147              set("z max", 0);
148              set("z min", -1*groove_depth);
149
150                 addpoly;
151                 set("name", "Vacuum poly");
152                 set("material", "etch");
153                 set("render type", 1);
154                 set("detail", 0.3);
155                 set("alpha", 0.3);
156                 set("override mesh order from material database", 1);
157                 set("mesh order", 2);
158                 x = hole_radius;
159                 set("vertices", [x,2e-6;-x,2e-6;-x,-2e-6;x,-2e-6]);
160                 set("z max", coating_thickness);
161                 set("z min", coating_thickness - groove_depth);  # -(
                        groove_depth - coating_thickness)
162
163                 # add a blaze structure
164                 min_angle = atan(groove_depth / (2*hole_radius));
165                 blaze_angle = 0.5*pi - blaze_angle_fraction * (0.5*pi -
                        min_angle);
166                 x = groove_depth/tan(blaze_angle);
167
168                 addtriangle;
169                 set("name", "blaze");
170                 set("material", coating_material);
171                 set("render type", 1);
172                 set("detail", 0.3);
```

```
173                 set("alpha", 0.3);
174                 set("override mesh order from material database", 1);
175                 set("mesh order", 1);
176                 set("first axis",2);
177                 set("rotation 1",90);
178                 set("vertices", [0,0;x,0;0,groove_depth]);
179                 set("z max", hole_radius);
180                 set("z min", -hole_radius);  # -(groove_depth -
                        coating_thickness)
181                 set("z", 0);
182                 set("z span", 4e-6);
183                 set("z",coating_thickness-groove_depth);
184                 set("x",-hole_radius);
185                 set("y",0);
186             }
187             else{
188                 select("structure group::coating poly");
189              x = hole_radius + coating_thickness;
190              set("vertices", [x,2e-6;-x,2e-6;-x,-2e-6;x,-2e-6]);
191                set("z min", -1*groove_depth);
192
193                select("structure group::Vacuum poly");
194                x = hole_radius;
195                set("vertices", [x,2e-6;-x,2e-6;-x,-2e-6;x,-2e-6]);
196                set("z max", coating_thickness);
197                set("z min", coating_thickness - groove_depth);  # -(
                       groove_depth - coating_thickness)
198
199                select("structure group::blaze");
200                set("vertices", [0,0;x,0;0,groove_depth]);
201                set("z max", hole_radius);
202                set("z min", -hole_radius);  # -(groove_depth -
                       coating_thickness);
203                set("z",coating_thickness-groove_depth);
204                set("x",-hole_radius);
205         }
206         ';
207     }
208
```

```
209
210     else if(grating_geometry == "linear_two") {
211         script = script + '
212             if(firstsetup == "True"){
213                 addpoly;
214                 set("name", "coating poly");
215                 set("material", coating_material);
216                 x = hole_radius + coating_thickness;
217                 set("vertices", [x,x;-x,x;-x,-x;x,-x]);
218                 set("z max", 0);
219                 set("z min", -1*groove_depth);
220                 set("render type", 1);
221                 set("detail", 0.3);
222                 set("alpha", 0.3);
223                 set("override mesh order from material database", 1);
224                 set("mesh order", 3);
225
226                 addpoly;
227                 set("name", "Vacuum poly");
228                 set("material", "etch");
229                 x = hole_radius;
230                 set("vertices", [x,x;-x,x;-x,-x;x,-x]);
231                 set("z max", coating_thickness);
232                 set("z min", coating_thickness - groove_depth);  # -(
                        groove_depth - coating_thickness);
233                 set("render type", 1);
234                 set("detail", 0.3);
235                 set("alpha", 0.3);
236                 set("override mesh order from material database", 1);
237                 set("mesh order", 2);
238
239                 # add a blaze structure
240                 min_angle = atan(groove_depth / hole_radius);
241                 blaze_angle = 0.5*pi - blaze_angle_fraction * (0.5*pi -
                        min_angle);
242                 x = groove_depth/tan(blaze_angle);
243
244                 addtriangle;
245                 set("name", "blaze");
```

```
246             set("material", coating_material);
247             set("vertices", [0,0;x,0;0,groove_depth]);
248             set("z max", hole_radius);
249             set("z min", -hole_radius);  # -(groove_depth -
                    coating_thickness)
250             set("z",coating_thickness-groove_depth);
251             set("x",-hole_radius);
252             set("y",0);
253             set("render type", 1);
254             set("detail", 0.3);
255             set("alpha", 0.3);
256             set("override mesh order from material database", 1);
257             set("mesh order", 1);
258             set("first axis",2);
259             set("rotation 1",90);
260
261             # add a second blaze structure
262             addtriangle;
263             set("name", "second blaze");
264             set("material", coating_material);
265             set("vertices", [-x,0;0,0;0,groove_depth]);
266             set("z max", hole_radius);
267             set("z min", -hole_radius);  # -(groove_depth -
                    coating_thickness)
268             set("z",coating_thickness-groove_depth);
269             set("x",hole_radius);
270             set("y",0);
271             set("render type", 1);
272             set("detail", 0.3);
273             set("alpha", 0.3);
274             set("override mesh order from material database", 1);
275             set("mesh order", 1);
276             set("first axis",2);
277             set("rotation 1",90);
278         }
279         else {
280             select("structure group::coating poly");
281             x = hole_radius + coating_thickness;
282             set("vertices", [x,x;-x,x;-x,-x;x,-x]);
```

```
283                 set("z min", -1*groove_depth);
284
285                 select("structure group::Vacuum poly");
286                 x = hole_radius;
287                 set("vertices", [x,x;-x,x;-x,-x;x,-x]);
288                 set("z max", coating_thickness);
289                 set("z min", coating_thickness - groove_depth);  # -(
                        groove_depth - coating_thickness);
290
291                 min_angle = atan(groove_depth / hole_radius);
292                 blaze_angle = 0.5*pi - blaze_angle_fraction * (0.5*pi -
                        min_angle);
293                 x = groove_depth/tan(blaze_angle);
294
295                 select("structure group::blaze");
296                 set("vertices", [0,0;x,0;0,groove_depth]);
297                 set("z max", hole_radius);
298                 set("z min", -hole_radius);  # -(groove_depth -
                        coating_thickness)
299                 set("z",coating_thickness-groove_depth);
300                 set("x",-hole_radius);
301
302                 select("structure group::second blaze");
303                 set("vertices", [-x,0;0,0;0,groove_depth]);
304                 set("z max", hole_radius);
305                 set("z min", -hole_radius);  # -(groove_depth -
                        coating_thickness)
306                 set("z",coating_thickness-groove_depth);
307                 set("x",hole_radius);
308             }
309         ';
310
311     }
312     else if(grating_geometry == "cylindrical_two") {
313         script = script + '
314             if(firstsetup == "True"){
315                 addcircle;
316                 set("name", "coating circle");
317                 set("material", coating_material);
```

```
318             set("radius", hole_radius + coating_thickness);
319             set("z max", 0);
320             set("z min", -1*groove_depth);
321             set("render type", 1);
322             set("detail", 0.3);
323             set("alpha", 0.3);
324             set("override mesh order from material database", 1);
325             set("mesh order", 3);
326
327             addcircle;
328             set("name", "Vacuum circle");
329             set("material", "etch");
330             set("radius", hole_radius);
331             set("z max", coating_thickness);
332             set("z min", coating_thickness - groove_depth);  # -(
                    groove_depth - coating_thickness);
333             set("render type", 1);
334             set("detail", 0.3);
335             set("alpha", 0.3);
336             set("override mesh order from material database", 1);
337             set("mesh order", 2);
338
339             # add polygon toroid
340             theta_start = 0;
341             theta_stop = 360;
342             material = coating_material;
343             resolution = 1000;
344             radius = hole_radius;
345
346             ################################################
347             # General polygon toroid
348             # This object created a 3D structure by revolving an
                    arbitrary outline,
349             # as defined by a set of polygon vertices, around the Z
                    axis with a radius R.
350             # The vertices of the polygon shape can be defined
                    following the
351             # instructions at http://docs.lumerical.com/en/fdtd/
                    user_guide_set_polygon_vertices.html
```

```
352 #
353 # Input properties
354 # theta start: starting angle of toroid
355 # theta stop:  stopping angle of toroid
356 # radius:      distance from the center of the toroid to
        the center of
357 #              the each slice that makes up the toroid
358 # material:    material of object
359 # resolution:  number of slices that make up overall shape
360 #
361 # Tags: toroid ring general custom polygon
362 #
363 # Copyright 2010 Lumerical Solutions Inc
364 ################################################
365
366 # USER specifies polygon vertices here. The 3D structure
        will be created by revolving this shape around Z axis,
        with a radius R.
367 # Note: It is OK, but not necessary to close the polygon
368 min_angle = atan(groove_depth / hole_radius);
369 blaze_angle = 0.5*pi - blaze_angle_fraction * (0.5*pi -
        min_angle);
370 x = groove_depth/tan(blaze_angle);
371
372 V=matrix(4,2);
373 V(1,1:2) = [ -x,0 ];
374 V(2,1:2) = [ 0,  groove_depth ];
375 V(3,1:2) = [ 0, 0 ];
376 V(4,1:2) = [ -x,0 ];
377
378
379 #plot(pinch(V,2,1)*1e6,pinch(V,2,2)*1e6,"x (um)","y (um)","
        Polygon outline");   # plot vertices (for debugging)
380
381
382 # calculate slice thickness
383 th = 4*pi*radius/resolution;  # divide circumference by
        resolution
384 th = th * 1.1;       # scale up thickess slighly. Required
```

```
            when polygon vertices extend beyond zero, which
            increases the maximum radius.
385
386         # if partial revolution, use only a fraction of slices
387         resolution=round(resolution*abs(theta_start-theta_stop)
            /360);
388
389         # Calculate revolution angle vector
390         theta = linspace(theta_start*pi/180,theta_stop*pi/180,
            resolution);
391
392         for(i=1:resolution) {
393             addpoly;
394             set("name", "coating blazing");
395             set("vertices",V);
396             set("first axis","x");
397             set("rotation 1",90);
398             set("second axis","z");
399             set("rotation 2",theta(i)*180/pi);
400             set("x",radius*cos(theta(i)));
401             set("y",radius*sin(theta(i)));
402
403             set("z min",-th/2 + coating_thickness - groove_depth);
404             set("z max",th/2 + coating_thickness - groove_depth);
405             set("material",material);
406             set("override mesh order from material database", 1);
407             set("mesh order", 1);
408         }
409     }
410     else {
411         select("structure group::coating circle");
412         set("radius", hole_radius + coating_thickness);
413         set("z max", 0);
414         set("z min", -1*groove_depth);
415
416         select("structure group::Vacuum circle");
417         set("radius", hole_radius);
418         set("z max", coating_thickness);
419         set("z min", coating_thickness - groove_depth);  # -(
```

```
                groove_depth - coating_thickness);
420
421             select("structure group::coating blazing");
422             #perform same calculations as above
423             min_angle = atan(groove_depth / hole_radius);
424             blaze_angle = 0.5*pi - blaze_angle_fraction * (0.5*pi -
                    min_angle);
425             x = groove_depth/tan(blaze_angle);
426             V=matrix(4,2);
427             V(1,1:2) = [ -x,0 ];
428             V(2,1:2) = [ 0,  groove_depth ];
429             V(3,1:2) = [ 0, 0 ];
430             V(4,1:2) = [ -x,0 ];
431             th = 4*pi*radius/resolution;  # divide circumference by
                    resolution
432             th = th * 1.1;        # scale up thickess slighly. Required
                    when polygon vertices extend beyond zero, which
                    increases the maximum radius.
433             resolution=round(resolution*abs(theta_start-theta_stop)
                    /360);
434             theta = linspace(theta_start*pi/180,theta_stop*pi/180,
                    resolution);
435             set("vertices",V);
436             set("rotation 2",theta(i)*180/pi);
437             set("x",radius*cos(theta(i)));
438             set("y",radius*sin(theta(i)));
439             set("z min",-th/2 + coating_thickness - groove_depth);
440             set("z max",th/2 + coating_thickness - groove_depth);
441         }
442     ';
443 }
444
445 select("::model::structure group");
446 set("first axis", 2); # first axis is x axis
447 set("rotation 1", 270);
448 set("script", script);
449 }
```

Sources and monitors setup

```
1  function sources_monitors(firstsetup, dimension, trueperiod) {
2      ##########################################################################
3      # input:
4      #   firstsetup: boolean - true if this function is used to set up
5      #                         the geometry in a new simulation
6      #                       - false if this function is called to
7      #                         update parameters
8      #   dimension: integer - the dimension of the FDTD simulation region
9      #       The options are: 2 (2D) or 3 (3D)
10     #   trueperiod: char string - specifies whether the
11     #                             period_radius_fraction is used for
12     #                             determining the period or whether the
13     #                             period is set as an absolute number.
14     #                             This information is necessary for
15     #                             updating the FDTD simulation region
16     #                             in parameter sweeps
17     #               - "true" if the period is set an an absolute number
18     #               - "false" if set by the period_radius_fraction
19     #
20     # output: nothing - this function sets up the monitors or updates
21     #         all necessary parameters
22     # This function can easily be extended by other monitors!
23     ##########################################################################
24
25     select("::model");
26     lambda_um = get("lambda_um");
27     coating_thickness = get("coating_thickness");
28     groove_depth = get("groove_depth");
29     hole_radius = get("hole_radius");
30     period_radius_fraction = get("period_radius_fraction");
31     blaze_angle_fraction = get("blaze_angle_fraction");
32     lambda_um = get("lambda_um");
33
34     period = period_radius_fraction * hole_radius;
35
36     if(firstsetup == "True") {
37         #add a plane wave source
38         addplane;
```

```
39          set("injection axis", "y");
40          set("direction", "backward");
41          set("x", 0);
42          set("x span", 100e-6);
43          set("z", 0);
44          set("z", 0);
45          set("z span", 100e-6);
46          set("wavelength start", lambda_um);
47          set("wavelength stop", lambda_um);
48          set("polarization angle", 90);
49          set("phase", 90);
50          set("y", coating_thickness + 1e-6);
51
52
53          #add a secomd plane wave source
54          addplane;
55          set("injection axis", "y");
56          set("direction", "backward");
57          set("x", 0);
58          set("x span", 100e-6);
59          set("z", 0);
60          set("z span", 100e-6);
61          set("wavelength start", lambda_um);
62          set("wavelength stop", lambda_um);
63          set("y", coating_thickness + 1e-6);
64
65
66
67          #add a power monitor
68
69          addpower;
70          set("name", "3dpowermonitor");
71          if(dimension==3){
72              set("monitor type", 8); # 3d
73              set("y span", 0);
74          }
75          else{
76              set("monitor type", 6); # 2d y normal
77          }
```

```
78          set("x", 0);
79          set("x span", 100e-6);
80          set("y", coating_thickness + 1.2e-6);
81          set("z", 0);
82          set("z span", 100e-6);
83          set("override global monitor settings", 1);
84          set("use source limits", 0);
85          set("frequency points", 1);
86          set("wavelength center", lambda_um);
87          set("wavelength span", 0);
88
89
90
91          #add a side view power monitor
92          addpower;
93          set("name", "sideview");
94          set("monitor type", 7); # 2d z normal
95          set("x", 0);
96          set("x span", 10e-6);
97          set("y", 0);
98          set("y span", 10e-6);
99          set("z", 0);
100         set("override global monitor settings", 1);
101         set("use source limits", 0);
102         set("frequency points", 1);
103         set("wavelength center", lambda_um);
104         set("wavelength span", 0);
105
106
107
108
109         # add the simulation region
110         addfdtd;
111         set("dimension", dimension-1); # 1 = 2D, 2 = 3D
112         set("x", 0);
113         set("z", 0);
114         set("x min bc", "Periodic");
115         set("y min bc", "PML");
116         set("y max bc", "PML");
```

```
117         if(dimension == 3) {
118             set("z max bc", "Periodic");
119             set("z min bc", "Periodic");
120         }
121         set("x span", period);
122         set("y min", -1.0 * (groove_depth + 0.5e-6));
123         set("y max", coating_thickness + 1.3e-6);
124         set("z span", period);
125         set("mesh accuracy", 4);
126     }
127     else {
128         select("3dpowermonitor");
129         set("y", coating_thickness + 1.2e-6);
130 
131         select("FDTD");
132         if(trueperiod == "False") {
133             set("x span", period);
134             set("z span", period);
135         }
136         set("y min", -1.0 * (groove_depth + 0.5e-6));
137         set("y max", coating_thickness + 1.3e-6);
138 
139         select("source");
140         set("y", coating_thickness + 1e-6);
141 
142         select("source_1");
143         set("y", coating_thickness + 1e-6);
144     }
145 }
```

Analysis scripts

```
mname = "::model::3dpowermonitor";
select("::model::FDTD");
dimension = get("dimension");
select("::model");
lambda_um = get("lambda_um");


################################################################################
# all analysis scripts are set up in this script:
#   1. Grating Transmission (Lumerical built-in plus own code)
#  2. Polarization Ellipse (Lumerical built-in)
#
# Results:
#   Grating Transmission:
#       T:  total transmitted power vs frequency. Will be negative for
#           power flowing in negative direction
#       T_grating:  (3D) fraction of source power transmitted to each
#                   grating order, S & P polarization components, direction
#                   cosine vectors, and theta, phi angles
#       T_grating: (2D) fraction of source power transmitted to each
#                   grating order, S & P polarization components,
#                   and theta angle
#       num_orders: the number or orders that were detected
#  first_order_efficiency: fraction of far field power transmitted to
#                                the (n,m)=(1,0) mode
#                    for 2D: Can be set to 0 outside a range of
#                                angles that are desired; The code for this
#                                can be commented/uncommented further below!
#  first_order_theta: emission angle of the (n,m) = (1,0) mode
#  first_order_efficiency_normalized: fraction of source power
#                                          transmitted to the
#                                          (n,m)=(1,0) mode
#
#   Polarization ellipse:
#       Gs_plot, Gp_plot: returns polarization in S and P direction for all
#                         orders as a function of lambda
#  num_orders:  returns the number of valid grating order vs lambda
#                   (may also include orders due to numberical error, e.g
```

```
    .,
39 #                    G=10^-30)
40 #        pol: contains the fraction transmitted power (G), phase differece
41 #             in s and p polarization (phase_diff), polarization handedness
42 #             (pol_handed), angle of major axis (major_angle) and ratio of
43 #             major/minor axis (ratio) for all grating orders and
44 #             wavelengths.
45 #        ratio_1_0: returns the ratio of the major to the minor axis for
46 #                   the (n,m) = (1,0) mode
47 ##########################################################################################
48
49
50 addanalysisgroup;
51 set("name", "Grating Transmission");
52 set("x", 0);
53 set("y", 0);
54 set("z", 0);
55 adduserprop("normal", 1, "y");
56 adduserprop("x span",2,5e-6);
57 adduserprop("y span",2,0);
58 adduserprop("z span",2,5e-6);
59 addanalysisprop("make plots", 0, 0);
60 addanalysisprop("n target", 0, 1);
61 addanalysisprop("m target", 0, 0);
62 addanalysisprop("lambda target", 2, lambda_um);
63 addanalysisresult("T");
64 addanalysisresult("T_grating");
65 addanalysisresult("num_orders");
66 addanalysisresult("first_order_efficiency");
67 addanalysisresult("first_order_theta");
68 addanalysisresult("first_order_efficiency_normalized");
69
70
71
72 analysis_script= '
73     mname = "::model::3dpowermonitor";
74
75     ##################################################################################
76     # Grating transmission
```

```
77  # This object calculates the fraction of source power transmitted to
78  # each grating order (total, S and P polarization) at all frequency
79  # points recorded by the monitor. It also calculates the number of
80  # propagating grating orders
81  #
82  # Input properties
83  #   make plots: 1 to make plots, 0 otherwise
84  #   n,m target: grating order to plot. These parameters only affect
85  #               the plots. They do not affect the output results.
86  #   lambda target: wavelength to plot. These parameters only affect
87  #                  the plots. They do not affect the output results.
88  # Output properties
89  #   T(f): total transmitted power vs frequency. Will be negative for
90  #         power flowing in negative direction
91  #   T_grating(n,m,f): (3D) fraction of source power transmitted to
92  #                     each grating order,S&P polarization components,
93  #                     direction cosine vectors, and theta,phi angles
94  #   T_grating(n,f): (2D) fraction of source power transmitted to each
95  #                   grating order, S & P polarization components,
96  #                   and theta angle
97  #   num_orders(f):  number of supported grating orders
98  #
99  # Notes
100 #   - grating_S, grating_P are normalized so that |grating_S|^2 gives
101 #     the fraction of the source power to each grating order that is
102 #     S polarized. |grating_S|^2+|grating_P|^2 = T_grating.
103 #   - Interpretation of results for various monitor orientations for
104 #     3D simulations:
105 #     XY plane: n, U1 correspond to X axis. m, U2 correspond to Y axis
106 #     XZ plane: n, U1 correspond to X axis. m, U2 correspond to Z axis
107 #     YZ plane: n, U1 correspond to Y axis. m, U2 correspond to Z axis
108 #
109 # Tags: far field grating order transmission
110 # Copyright 2016 Lumerical Solutions Inc
111 ########################################################################
112
113
114
115 # simplify input variable names by removing spaces
```

```
116     make_plots = %make plots%;
117     n_target = %n target%;
118     m_target = %m target%;
119     lambda_target = %lambda target%;
120
121     # specify monitor name
122     mname="::model::3dpowermonitor";
123
124     # get frequency vector
125     f=getdata(mname,"f");
126     size_f=length(f);
127
128     # get total net power transmitted through monitor
129     T=transmission(mname);
130
131     if (getdata(mname,"dimension") == 3) { # 3D simulation
132
133
134         # find the maximum possible number of grating orders
135         # this occurs at the maximum frequency
136         n=gratingn(mname,size_f);
137         m=gratingm(mname,size_f);
138         size_n=length(n);
139         size_m=length(m);
140
141         # initialize matrices
142         T_grating   = matrix(size_n,size_m,size_f); # grating order
                strength vs f
143         grating_S   = matrix(size_n,size_m,size_f); # |grating_S|^2 gives
                the fraction of the source power to each grating order that is S
                 polarized
144         grating_P   = matrix(size_n,size_m,size_f); # |grating_P|^2 gives
                the fraction of the source power to each grating order that is P
                 polarized
145         U1          = matrix(size_n,size_m,size_f); # first direction
                cosine  (if monitor is in XY plane, this corresponds to Ux)
146         U2          = matrix(size_n,size_m,size_f); # second direction
                cosine  (if monitor is in XY plane, this corresponds to Uy)
147
```

```
148
149         # loop over each frequency point
150         for (i=1:size_f) {
151
152             # get the grating numbers at this frequency
153             n_tmp = gratingn(mname,i);
154             m_tmp = gratingm(mname,i);
155
156             # calculate indices for inserting these results into final
                    matrix
157             n1    = find(n,n_tmp(1));
158             n2    = find(n,n_tmp(length(n_tmp)));
159             m1    = find(m,m_tmp(1));
160             m2    = find(m,m_tmp(length(m_tmp)));
161
162             # calculate grating order angles (direction cosine units)
163             # and save into U1, U2 matrix.
164             # set unused orders to -1 or +1
165             u1            = matrix(size_n);
166             u2            = matrix(size_m);
167             u1(1:n1)      =-1;
168             u2(1:m1)      =-1;
169             u1(n2:size_n) = 1;
170             u2(m2:size_m) = 1;
171             u1(n1:n2)     = gratingu1(mname,i);
172             gratingu2(mname,i);
173             u2(m1:m2)     = gratingu2(mname,i);
174             U1(1:size_n,1:size_m,i) = meshgridx(u1,u2);
175             U2(1:size_n,1:size_m,i) = meshgridy(u1,u2);
176
177
178             # calculate grating orders and save into T_grating matrix
179             grating_temp             = gratingpolar(mname,i);
180             grating_temp             = grating_temp * sqrt(abs(T(i))); #
                    normalize result such that sum of all grating orders of |
                    Etheta|^2+|Ephi|^2 equals
181                                 # the fraction of source power transmitted
                                        through the monitor.
182             grating_S(n1:n2,m1:m2,i) = pinch(grating_temp,3,3);        # |
```

```
            grating_S|^2 gives the fraction of the source power to each
            grating order that is S polarized
183         grating_P(n1:n2,m1:m2,i) = pinch(grating_temp,3,2);        # |
            grating_P|^2 gives the fraction of the source power to each
            grating order that is P polarized
184         T_grating(n1:n2,m1:m2,i) = abs(pinch(grating_temp,3,2))^2 +
            abs(pinch(grating_temp,3,3))^2; # fraction of source power
            to each grating order
185
186     }
187
188     # calculate U3 and convert grating directions to theta,phi.  If
            monitor is in XY plane, U3=Uz
189     U3    = sqrt ( 1-U1^2-U2^2 );
190     theta = real ( acos(U3) ) * 180/pi;
191     phi   = atan2( U2,U1  )   * 180/pi;
192
193
194     # Calculate the number of grating orders (theta < 90)
195     # NOTE: this script for counting grating orders assumes a
            rectangular
196     # unit cell. The count will not be correct for triangular lattices.
197     num_orders_matrix = sum(sum(   (real(theta) < 89.9)   , 2),1);
198
199     T_matrix=T;
200     T_grating_matrix=T_grating;
201
202     T = matrixdataset("T");
203     T.addparameter("lambda",c/f,"f",f);
204     T.addattribute("T",T_matrix);
205
206     num_orders = matrixdataset("num_orders");
207     num_orders.addparameter("lambda",c/f,"f",f);
208     num_orders.addattribute("num_orders",num_orders_matrix);
209
210     T_grating = matrixdataset("T_grating");
211     T_grating.addparameter("n",n);
212     T_grating.addparameter("m",m);
213     T_grating.addparameter("lambda",c/f,"f",f);
```

```
214         T_grating.addattribute("T_grating",T_grating_matrix);
215         T_grating.addattribute("grating_S",grating_S);
216         T_grating.addattribute("grating_P",grating_P);
217         T_grating.addattribute("U1",U1);
218         T_grating.addattribute("U2",U2);
219         T_grating.addattribute("U3",U3);
220         T_grating.addattribute("theta",theta);
221         T_grating.addattribute("phi",phi);
222
223
224         if (make_plots) {
225
226             # plot number of orders
227             plot(c/f*1e6, num_orders_matrix,
228             "wavelength (um)","","Number of grating orders");
229
230
231             # plot data for a particular grating order
232             T_grating_plot = pinch(pinch(T_grating_matrix,2,find(m,m_target
                    )) ,1,find(n,n_target));
233             theta_plot     = pinch(pinch(theta,2,find(m,m_target))      ,1,
                    find(n,n_target));
234             phi_plot       = pinch(pinch(phi,2,find(m,m_target))        ,1,
                    find(n,n_target));
235
236             plot(c/f*1e6, abs(T_matrix), T_grating_plot,
237             "wavelength (um)","Transmission","Transmission");
238             legend("Total","To order ("+num2str(n_target)+","+num2str(
                    m_target)+")");
239             plot(c/f*1e6, theta_plot, phi_plot,
240             "wavelength (um)","angle (deg)","Propagation direction for
                    order ("+num2str(n_target)+","+num2str(m_target)+")");
241             legend("Theta","Phi");
242
243
244             ##########################################################################
245             # plot results at one frequency point
246             fi             = find(c/f,lambda_target);
247             u1_plot        = pinch(pinch(U1,3,fi),2,1);
```

```
248             u2_plot         = pinch(pinch(U2,3,fi),1,1);
249             T_grating_plot = pinch(T_grating_matrix,3,fi);
250             image(u1_plot,u2_plot,T_grating_plot,
251             "u1","u2","Transmission at "+num2str(c/f(fi)*1e6)+"um","polar
                    plot");
252
253             # re-plot at higher resolution for a nicer plot
254             pts             = 35;
255             u1_plot2        = linspace(-1,1,pts);
256             u2_plot2        = linspace(-1,1,pts);
257             T_grating_plot2 = matrix(pts,pts);
258
259             for (i=1:size_n) {
260                 for (j=1:size_m) {
261                     u1i = find(u1_plot2,u1_plot(i));
262                     u2j = find(u2_plot2,u2_plot(j));
263
264                     T_grating_plot2(u1i,u2j) = T_grating_plot(i,j);
265                 }
266             }
267
268             image(u1_plot2,u2_plot2,T_grating_plot2,
269             "u1","u2","Transmission at "+num2str(c/f(fi)*1e6)+"um","polar
                    plot");
270             image(u1_plot2,u2_plot2,log10(abs(T_grating_plot2)+1e-5),
271             "u1","u2","Log10(|Transmission|) at "+num2str(c/f(fi)*1e6)+"um
                    ","polar plot");
272         }
273
274     } else { # 2D simulation
275
276
277         # find the maximum possible number of grating orders
278         # this occurs at the maximum frequency
279         n=gratingn(mname,size_f);
280         size_n=length(n);
281
282         # initialize matrices
283         T_grating = matrix(size_n,size_f); # grating order strength vs f
```

```
284         theta     = matrix(size_n,size_f); # angle matrix
285         grating_S   = matrix(size_n,size_f); # |grating_S|^2 gives the
                fraction of the source power to each grating order that is S
                polarized
286         grating_P   = matrix(size_n,size_f); # |grating_P|^2 gives the
                fraction of the source power to each grating order that is P
                polarized
287
288
289         # loop over each frequency point
290         for (i=1:size_f) {
291
292             # get the grating numbers at this frequency
293             n_tmp = gratingn(mname,i);
294
295             # calculate indices for inserting these results into final
                    matrix
296             n1    = find(n,n_tmp(1));
297             n2    = find(n,n_tmp(length(n_tmp)));
298
299             # calculate grating order angles
300             # set unused orders to -90 or +90
301             theta(1:n1,i)       = -90;
302             theta(n2:size_n,i) = 90;
303             theta(n1:n2,i)      = gratingangle(mname,i);
304
305             # calculate grating orders and save into T_grating matrix
306             grating_temp         = gratingpolar(mname,i);
307             grating_temp         = grating_temp * sqrt(abs(T(i))); #
                    normalize result such that sum of all grating orders of |
                    Etheta|^2+|Ephi|^2
308                         # equals the fraction of source power
                                transmitted through the monitor.
309             grating_S(n1:n2,i)  = pinch(grating_temp,2,3);        # |
                    grating_S|^2 gives the fraction of the source power to each
                    grating order that is S polarized
310             grating_P(n1:n2,i)  = pinch(grating_temp,2,2);        # |
                    grating_P|^2 gives the fraction of the source power to each
                    grating order that is P polarized
```

```
311             T_grating(n1:n2,i)  = abs(pinch(grating_temp,2,2))^2 + abs(
                    pinch(grating_temp,2,3))^2; # fraction of source power to
                    each grating order
312         }
313
314
315         # Calculate the number of grating orders (theta < 90)
316         num_orders_matrix = sum( (abs(theta) < 89.9)  ,1);
317
318
319         T_matrix=T;
320         T_grating_matrix=T_grating;
321
322         T = matrixdataset("T");
323         T.addparameter("lambda",c/f,"f",f);
324         T.addattribute("T",T_matrix);
325
326         num_orders = matrixdataset("num_orders");
327         num_orders.addparameter("lambda",c/f,"f",f);
328         num_orders.addattribute("num_orders",num_orders_matrix);
329
330         T_grating = matrixdataset("T_grating");
331         T_grating.addparameter("n",n);
332         T_grating.addparameter("lambda",c/f,"f",f);
333         T_grating.addattribute("T_grating",T_grating_matrix);
334         T_grating.addattribute("grating_S",grating_S);
335         T_grating.addattribute("grating_P",grating_P);
336         T_grating.addattribute("theta",theta);
337
338
339
340         if (make_plots) {
341
342             # plot number of orders
343             plot(c/f*1e9, num_orders_matrix,
344             "wavelength (nm)","","Number of grating orders");
345
346             # plot data for a particular grating order
347             T_grating_plot = pinch( T_grating_matrix,1,find(n,n_target) );
```

```
348             theta_plot      = pinch( theta      ,1,find(n,n_target) );
349
350             plot(c/f*1e9, abs(T_matrix), T_grating_plot,
351             "wavelength (nm)","Transmission","Transmission");
352             legend("Total","To order ("+num2str(n_target)+")");
353             plot(c/f*1e9, theta_plot,
354             "wavelength (nm)","angle (deg)","Propagation angle for order
                    ("+num2str(n_target)+")");
355
356
357             # plot results at one frequency point
358             fi              = find(c/f,lambda_target);
359             theta_plot      = pinch(theta,2,fi);
360             T_grating_plot = pinch(T_grating_matrix,2,fi);
361             plot(theta_plot,T_grating_plot,
362             "theta (deg)","Transmission","Transmission at "+num2str(round(c
                    /f(fi)*1e9))+"nm","plot points");
363         }
364
365     }
366 ';
367 ########## my own implementation starts here #################
368
369
370 if (dimension == "3D") { # 3D simulation
371     analysis_script = analysis_script + '
372         orders = gratingordercount(mname);
373         u1 = gratingu1(mname);
374         u2 = gratingu2(mname);
375         uz = sqrt(1 - u1*u1 - u2*u2);
376         theta = acos(u2) * 180 / pi; # azimuth angle
377         phi = atan(u2/u1) * 180 / pi; # polar angle
378         T = transmission(mname);
379
380         grating_efficiencies = grating(mname);
381
382         grating_n = gratingn(mname);
383         grating_m = gratingm(mname);
384
```

```
385
386     # Depending on the number of orders, the first order can be found with
            a different index
387     # The case of 3 orders was observed in a blazed grating. In a radially
            symmetric grating, the number of orders is 1, 5 or 9
388         if(orders == 1) {
389             firstordern = 1;
390             firstorderm = 1;
391             first_order_efficiency = 0;
392             firstorderu2 = 1;
393             first_order_theta = acos(firstorderu2) * 180/ pi;
394         }
395
396         else if(orders == 3) {
397             firstordern = 1;
398             firstorderm = 0;
399             first_order_efficiency = grating_efficiencies(3);
400             firstorderu2 = u1(3);
401             first_order_theta = acos(firstorderu2) * 180/ pi;
402         }
403
404         else if(orders == 5) {
405             firstordern = 2;
406             firstorderm = 3;
407             first_order_efficiency = grating_efficiencies(firstordern,
                    firstorderm);
408             firstorderu2 = u2(3);
409             first_order_theta = acos(firstorderu2) * 180/ pi;
410         }
411
412         else if(orders == 9) {
413             firstordern = 2;
414             firstorderm = 3;
415             first_order_efficiency = grating_efficiencies(firstordern,
                    firstorderm);
416             firstorderu2 = u2(3);
417             first_order_theta = acos(firstorderu2) * 180/ pi;
418         }
419
```

```
420   # Here a more experimental algorithm that finds the first order
          automatically. It was not thoroughly tested and needs to be adapted
421           else {
422            print("Universal algorithm");
423               firstordern = find(gratingn(mname), 0);
424               firstorderm = find(gratingm(mname), 1);
425               first_order_efficiency = grating_efficiencies(firstordern,
                      firstorderm);
426
427               zerou2 = find(u2, 0);
428               if(zerou2 == 1) {
429                   firstorderu2 = 1;
430               }
431               else {
432                   firstorderu2 = u2(zerou2 + 1);
433               }
434               first_order_theta = acos(firstorderu2) * 180/ pi;
435           }
436           # grating(mname) gives only the power relative to the farfield
                  power.
437           # To get the power relative to the source power, we need to
                  multiply with T (farfield power / source power)
438           first_order_efficiency_normalized = T * first_order_efficiency;
439     ';
440   }
441   else if (dimension == "2D") { # 2D simulation
442       analysis_script = analysis_script + '
443           orders = gratingordercount(mname);
444           theta = theta;
445           first_order_efficiencies = T_grating.T_grating;
446           T = transmission(mname);
447
448           if(orders == 3) {
449               first_order_efficiency_normalized = first_order_efficiencies(3)
                      ;
450               first_order_theta = theta(3);
451           }
452           else {
453               first_order_efficiency_normalized = 0;
```

```
454             first_order_theta = 90;
455         }
456
457         first_order_efficiency = first_order_efficiency_normalized / T;
458         first_order_n = 1;
459         first_order_m = 0;
460
461         # for the optimization algorithm we can set the
                first_order_efficiency_normalized to 0 if the emission angle of
                the first order is not within the
462         # range [42 degrees, 48 degrees]. Comment/uncomment as needed
463         if((first_order_theta <=42) or (first_order_theta >= 48)){
464             first_order_efficiency = 0;
465             first_order_efficiency_normalized = 0;
466         }
467
468     ';
469 }
470
471 set("analysis script", analysis_script);
472
473
474 # add a polarization analysis
475 addanalysisgroup;
476 set("name", "polarization ellipse");
477 set("x", 0);
478 set("y", 1.35e-6);
479 set("z", 0);
480 adduserprop("normal", 1, "y");
481 adduserprop("x span",2,5e-6);
482 adduserprop("y span",2,0);
483 adduserprop("z span",2,5e-6);
484 addanalysisprop("make plots", 0, 0);
485 addanalysisprop("n target", 0, 1);
486 addanalysisprop("m target", 0, 0);
487 addanalysisprop("lambda target", 2, lambda_um);
488 addanalysisprop("ellipse res", 0, 1000);
489 addanalysisresult("Gp_plot");
490 addanalysisresult("Gs_plot");
```

```
491 addanalysisresult("pol");
492 addanalysisresult("num_orders");
493 addanalysisresult("ratio_1_0");
494
495
496 analysis_script = '
497         ##########################################################################
498         # Polarization ellipse
499         # This script calculates the polarization of all grating orders.
500         # The results of all orders (n,m) are returned to the Gs_plot and
501         # Gp_plot datasets
502         # The results of a user-specified grating order and frequency
503         # point can be plotted in terms of the polarization ellipse
504         #
505         # Input parameters:
506         # make plots: 1 to make plots, 0 otherwise
507         # n_target, m_target: grating order of interest (n,m).
508         # These parameters only affect the ellipse plot. They do not
509         # affect the output results.
510         # lambda_target: wavelength of interest. This parameter only
511         # affects the ellipse plot. It does not affect the output results.
512         # ellipse_res: resolution of the polarization ellipse.
513         # This parameter affects the ellipse plot, it may also affect the
514         # output results.
515         #
516         # Output results:
517         # Gs_plot, Gp_plot: returns polarization in S and P direction for
518         # all orders as a function of lambda
519         # num_orders: returns the number of valid grating order vs lambda
520         # (may also include orders due to numberical error,e.g.,G=10^-30)
521         # pol: contains the fraction transmitted power (G), phase differece
522         # in s and p polarization (phase_diff), polarization handedness
523         # (pol_handed), angle of major axis (major_angle) and ratio of
524         # major/minor axis (ratio) for all grating orders and wavelengths.
525         #
526         # Note - Since the size of the pol dataset is a function of the
527         #        number of grating orders
528         #        Therefore, a larger dataset is created and then stitch
529         #        data into the whole matrix
```

```
530         #          The ratio of major/minor axis can be a "not a number"
531         #          (NAM)
532         #          For some invalid grating orders, the major and minor
533         #          axis can be both zero that causes NAM
534         #          The visualizer may not be able to properly display
535         #          these entries
536         #          Users can use the View Data function in the visualizer
537         #          to see the exact numbers
538         #
539         #       - The polarization handedness is defined from the point
540         #          of view of the receiver
541         #
542         # Plot:
543         # ellipse:  plots a polarization ellipse based on the s,
544         #           p polarization components of a user-specified n,
545         #           m and lambda. Numbers in the plot title are rounded
546         #           to integer
547         #
548         # Tags: far field polarization ellipse grating
549         #
550         # Copyright 2014 Lumerical Solutions Inc.
551         ########################################################################################
552
553         mname = "::model::3dpowermonitor";
554
555
556         # simplify input variable names by removing spaces
557         make_plots = %make plots%;
558         n_target = %n target%; # target grating order for the ellipse plot
559         m_target = %m target%;
560         lambda_target = %lambda target%; # target frequency point for the
                ellipse plot
561         ellipse_res = %ellipse res%; # pol ellipse resolution. This number
                affects the accuracy
562
563         # specify monitor name, get frequency vector
564         f=getdata(mname,"f");
565
566         # find the maximum possible number of grating orders, this occurs
```

```
        at the maximum frequency
567     n=gratingn(mname,length(f));
568     m=gratingm(mname,length(f));
569
570     # initialize matrices
571     u1 = matrix(length(n));
572     u2 = matrix(length(m));
573     U1 = matrix(length(n),length(m),length(f)); # first direction
        cosine  (if monitor is in XY plane, this corresponds to Ux)
574     U2 = matrix(length(n),length(m),length(f)); # second direction
        cosine  (if monitor is in XY plane, this corresponds to Uy)
575     Gs_all = matrix(ellipse_res,length(n),length(m),length(f)); # s pol
        . vs res,n,m,f
576     Gp_all = matrix(ellipse_res,length(n),length(m),length(f)); # p pol
        . vs res,n,m,f
577     kappa  = linspace(0,360,ellipse_res)*pi/180;    # setting kappa for
         Gs and Gp
578     G  = matrix(length(n),length(m),length(f)); # grating order
        strength vs n,m,f
579     major_angle_all = matrix(length(n),length(m),length(f)); # major
        angle vs n,m,f (this is also called the orientation or tilt
        angle)
580     ratio_all  = matrix(length(n),length(m),length(f));  # ratio vs n,m
        ,f (for linear pol., ratio can be very high)
581     phase_diff_all = matrix(length(n),length(m),length(f)); # phase vs
        n,m,f (the phase difference between s and p polarization (p
        minus s))
582     pol_handed_all = matrix(length(n),length(m),length(f));
583     # left or right handed polarization. 1 is right-handed, -1 is left-
        handed, 0 means no entry/linear.
584     # In other words, from phase_diff, -180<p-s<0 is left, 0<p-s<180 is
         right.
585     # This is defined from the point of view of the receiver (against
        the progation direction).
586     # User can multiply -1 to this matrix if from the point of view of
        the source.
587
588     # loop over each frequency point
589     for (fi=1:length(f)) {
```

```
590
591         # get the grating numbers at this frequency point
592         n_tmp = gratingn(mname,fi);
593         m_tmp = gratingm(mname,fi);
594
595         # calculate indices for inserting these results into the final
                matrix
596         # that the final matrix is filled from n1 to n2, and m1 to m2
597         n1    = find(n,n_tmp(1));
598         n2    = find(n,n_tmp(length(n_tmp)));
599         m1    = find(m,m_tmp(1));
600         m2    = find(m,m_tmp(length(m_tmp)));
601
602         # calculate grating order angles (direction cosine units)
603         # and save into U1, U2 matrix. Set unused orders to -1 or +1
604         u1(1:n1)       =-1;
605         u2(1:m1)       =-1;
606         u1(n2:length(n)) = 1;
607         u2(m2:length(m)) = 1;
608         u1(n1:n2)      = gratingu1(mname,fi);
609         u2(m1:m2)      = gratingu2(mname,fi);
610         U1(1:length(n),1:length(m),fi) = meshgridx(u1,u2);
611         U2(1:length(n),1:length(m),fi) = meshgridy(u1,u2);
612
613         # calculate grating orders and save into G matrix
614         G_vec  = gratingpolar(mname,fi);
615         Gtheta = pinch(G_vec,3,2);
616         Gphi   = pinch(G_vec,3,3);
617
618         # grating order strength (fraction of transmitted power)
619         # stitch data into the central columns of the final result matrices
620         # to have all data from different grating orders packaged in a
                single matrix dataset
621         G(n1:n2,m1:m2,fi) = abs(Gtheta)^2+abs(Gphi)^2;
622
623             #loop over all grating numbers
624             for ( ni=1:length(n_tmp) )  {
625             for ( mi=1:length(m_tmp) )  {
626
```

```
627         # convert spherical coordinates polarization into s,p polarization
                and select the grating order
628         # calculate the polarization ellipse. kappa is the resolution
629         Gs = Gphi(ni,mi);
630         Gp = Gtheta(ni,mi);
631         Gs = real( Gs*exp(1i*kappa) );
632         Gp = real( Gp*exp(1i*kappa) );
633
634         # measure parameters(ratio,major_angle) from the ellipse. normalize
                 Gs and Gp
635         diameter   = sqrt( (Gs)^2+(Gp)^2 );
636         major_axis = max( diameter );      # for some grating numbers, the
                major_axis can be 0
637         minor_axis = min( diameter );      # for some grating numbers, the
                minor_axis can be 0
638         ratio      = major_axis / minor_axis;      # for some grating
                numbers, the ratio can be a NAM (0/0)
639         if (almostequal(ratio,1,0.0001,0.0001)){major_axis_i = find( Gp,
                max( Gp )); # if ratio ~ 1, then it will have no major axis
640         }else{ major_axis_i = find( diameter, max(diameter) );}
641         major_angle  = atan2( Gs( major_axis_i ), Gp( major_axis_i ) )*180/
                pi;
642         if (major_angle < -90) { major_angle = major_angle+180; }
643         if (major_angle > 90)  { major_angle = major_angle-180; }
644         Gs = Gs/major_axis;
645         Gp = Gp/major_axis; # normalization
646
647         # measure phase difference from the s and p component
648         phase_diff = (angle(Gtheta(ni,mi))-angle(Gphi(ni,mi)))*180/pi; #
                phase difference of s and p pol. in degree
649         if (Gphi(ni,mi)==0+1i*0) { phase_diff = 0; }    # define linearly
                pol light
650         if (Gtheta(ni,mi)==0+1i*0) { phase_diff = 0; }    # define linearly
                 pol light
651         if (phase_diff > 180) { phase_diff = phase_diff - 180; } # -180<
                phase_diff <180
652         if (phase_diff < -180) { phase_diff = phase_diff + 180; }  # -180<
                phase_diff <180
653
```

```
654         # measure polarization handedness based on phase_diff
655         pol_handed = 0; # initiallise an entry
656         if (phase_diff < 0)   { pol_handed = -1; } # define left-handed
657         if (phase_diff > 0)   { pol_handed = 1; }  # define right-handed
658
659         # stitch data into the central columns of the final result matrices
660         # to have all data from different grating orders packaged in a
                single matrix dataset (as a function of n, m, f)
661         Gs_all(1:ellipse_res,n1+ni-1,m1+mi-1,fi) = Gs;
662         Gp_all(1:ellipse_res,n1+ni-1,m1+mi-1,fi) = Gp;
663         phase_diff_all(n1+ni-1,m1+mi-1,fi) = phase_diff;
664         pol_handed_all(n1+ni-1,m1+mi-1,fi) = pol_handed;
665         ratio_all(n1+ni-1,m1+mi-1,fi)  = ratio;
666         major_angle_all(n1+ni-1,m1+mi-1,fi)= major_angle;
667             }
668         }
669         }
670
671         # calculate U3 and convert grating directions to theta,phi. If
                monitor is in XY plane, U3=Uz
672         U3    = sqrt ( 1-U1^2-U2^2 );
673         theta = real ( acos(U3) ) * 180/pi;
674         phi   = atan2( U2,U1  )   * 180/pi;
675
676         # Calculate the number of grating orders (theta < 90)
677         # NOTE: this script for counting grating orders assumes a
                rectangular
678         # unit cell. The count will not be correct for triangular lattices.
679         # this will count all non-zero entries, including numerical error
                roders
680         num_orders_matrix = sum(sum(    (real(theta) < 89.9)   , 2),1);
681
682         # Create datasets
683         num_orders = matrixdataset("num_orders");
684         num_orders.addparameter("lambda",c/f,"f",f);
685         num_orders.addattribute("num_orders",num_orders_matrix);
686
687         pol = matrixdataset("polarization");
688         pol.addparameter("n",n);
```

```
689         pol.addparameter("m",m);
690         pol.addparameter("lambda",c/f,"f",f);
691         pol.addattribute("G",G);
692         pol.addattribute("phase_diff",phase_diff_all);
693         pol.addattribute("pol_handed",pol_handed_all);
694         pol.addattribute("major_angle",major_angle_all);
695         pol.addattribute("ratio",ratio_all);
696
697         Gs_plot = matrixdataset("Gs_all");
698         Gs_plot.addparameter("ellipse_point",linspace(1,ellipse_res,
                ellipse_res));
699         Gs_plot.addparameter("n",n);
700         Gs_plot.addparameter("m",m);
701         Gs_plot.addparameter("lambda",c/f,"f",f);
702         Gs_plot.addattribute("Gs_all",Gs_all);
703
704         Gp_plot = matrixdataset("Gp_all");
705         Gp_plot.addparameter("ellipse_point",linspace(1,ellipse_res,
                ellipse_res));
706         Gp_plot.addparameter("n",n);
707         Gp_plot.addparameter("m",m);
708         Gp_plot.addparameter("lambda",c/f,"f",f);
709         Gp_plot.addattribute("Gp_all",Gp_all);
710
711         # make ellipse plot for the user-specified grating orders and
                frequency point
712         if (make_plots) {
713
714             ni = find(n,n_target);
715             mi = find(m,m_target);
716             fi=find(f,c/lambda_target);
717
718             # these command lines can be also used to plot Gs and Gp in
                    other tools, such as excel
719             Gp_all=Gp_plot.Gp_all;
720             Gs_all=Gs_plot.Gs_all;
721             Gp=pinch(pinch(pinch(Gp_all,2,ni),2,mi),2,fi);
722             Gs=pinch(pinch(pinch(Gs_all,2,ni),2,mi),2,fi);
723
```

```
724             # plot ellipse, and ellipse parameters
725             # numbers are rounded to integer to save title space
726             plot(Gp,Gs,"P-pol","S-pol"," pol ellipse (N,M)=(" +num2str(n(ni
                    ))+","+num2str(m(mi))+ ")" +
727                                             " (theta,phi)=(" +num2str(round(
                                                  theta(ni,mi,fi)))+","+num2str(
                                                  round(phi(ni,mi,fi)))+")" +
728                                             " ratio = " +num2str(ratio_all(ni
                                                  ,mi,fi)) +
729                                             " major angle (deg) = " +num2str(
                                                  major_angle_all(ni,mi,fi)) );
730
731             setplot("x min",-1);
732             setplot("x max", 1);
733             setplot("y min",-1);
734             setplot("y max", 1);
735         }
736         ni = find(n,n_target);
737         mi = find(m,m_target);
738         fi=find(f,c/lambda_target);
739         ratio_1_0 = ratio_all(ni,mi,fi);
740 ';
741
742 set("analysis script", analysis_script);
```

Two-parameter sweep

```
1  # A nested sweep over one or two parameters
2
3  mname = "::model::3dpowermonitor";
4
5  ##########################################################################################
6  # Here, we choose the paramters to sweep over.
7  # The sweeps that can be found below were implemented.
8  # A sweep can be selected by uncommenting it.
9  # The code can be easily adapted to add other parameter sweeps.
10 ##########################################################################################
11
12 # the number of sweep parameters
13 num_sweeppar1_sweeps=50;
14 num_sweeppar2_sweeps=50;
15
16
17 ###### period radius sweep ################################
18 sweeppar1_start = 0.15e-6;
19 sweeppar1_stop = 0.8e-6;
20 sweeppar2_start = 0.05e-6;
21 sweeppar2_stop = 0.4e-6;
22 sweeppar1_name = "period";
23 sweeppar2_name = "radius";
24 ##########################################################################
25
26 ###### coatingTickness grooveDepth sweep ##################
27 #sweeppar1_start = 0.01e-6;
28 #sweeppar1_stop = 0.5e-6;
29 #sweeppar2_start = 0.01e-6;
30 #sweeppar2_stop = 1e-6;
31 #sweeppar1_name = "coatingThickness";
32 #sweeppar2_name = "grooveDepth";
33 ##########################################################################
34
35 ####### period_radius_fraction sweep ######################
36 #sweeppar1_start = 0;
37 #sweeppar1_stop = 1;
38 #sweeppar1_name = "period_radius_fraction";
```

```
39 #sweeppar2_name = "none";
40 ##############################################################
41
42 ####### blaze angle fraction sweep ######################
43 #sweeppar1_name = "blaze_angle_fraction";
44 #sweeppar1_start = 0;
45 #sweeppar1_stop = 1;
46 #sweeppar2_name = "none";
47 ##############################################################
48
49 if (sweeppar1_name == "coatingThickness") {
50     select("::model");
51     para1 = struct;
52     para1.Name = "coating thickness";
53     para1.Parameter = "::model::coating_thickness";
54     para1.Type = "Length";
55     para1.Start = sweeppar1_start;
56     para1.Stop = sweeppar1_stop;
57     para1.Units = "microns";
58
59 }
60 if(sweeppar1_name == "period"){
61     select("::model::FDTD");
62     para1 = struct;
63     para1.Name = "period";
64     para1.Parameter = "::model::FDTD::x span";
65     para1.Type = "Length";
66     para1.Start = sweeppar1_start;
67     para1.Stop = sweeppar1_stop;
68     para1.Units = "microns";
69
70     para1_2 = struct;
71     para1_2.Name = "period";
72     para1_2.Parameter = "::model::FDTD::z span";
73     para1_2.Type = "Length";
74     para1_2.Start = sweeppar1_start;
75     para1_2.Stop = sweeppar1_stop;
76     para1_2.Units = "microns";
77 }
```

```
78 if (sweeppar2_name == "grooveDepth") {
79     select("::model");
80     para2 = struct;
81     para2.Name = "groove depth";
82     para2.Parameter = "::model::groove_depth";
83     para2.Type = "Length";
84     para2.Start = sweeppar2_start;
85     para2.Stop = sweeppar2_stop;
86     para2.Units = "microns";
87 }
88 if (sweeppar2_name == "radius") {
89     para2 = struct;
90     para2.Name = "radius";
91     para2.Parameter = "::model::hole_radius";
92     para2.Type = "Length";
93     para2.Start = sweeppar2_start;
94     para2.Stop = sweeppar2_stop;
95     para2.Units = "microns";
96 }
97 if (sweeppar1_name == "period_radius_fraction") {
98     para1 = struct;
99     para1.Name = "period_radius_fraction";
100    para1.Parameter = "::model::period_radius_fraction";
101    para1.Type = "Number";
102    para1.Start = sweeppar1_start;
103    para1.Stop = sweeppar1_stop;
104 }
105 if (sweeppar1_name == "blaze_angle_fraction") {
106    para1 = struct;
107    para1.Name = "blaze_angle_fraction";
108    para1.Parameter = "::model::blaze_angle_fraction";
109    para1.Type = "Number";
110    para1.Start = sweeppar1_start;
111    para1.Stop = sweeppar1_stop;
112 }
113
114 ######### General implementation of the nested sweep ############
115
116 # child paramter sweep over width of simulation region
```

```
117 addsweep(0);
118 sweepname = sweeppar1_name+" sweep";
119 setsweep("sweep", "name", sweepname);
120 setsweep(sweepname, "type", "Ranges");
121 setsweep(sweepname, "number of points", num_sweeppar1_sweeps);
122
123
124 addsweepparameter(sweepname, para1);
125 if(sweeppar1_name == "period"){
126     addsweepparameter(sweepname, para1_2);
127 }
128
129 result = struct;
130 result.Name = "T";
131 result.Result = "::model::Grating Transmission::T";
132 addsweepresult(sweepname, result);
133
134 result = struct;
135 result.Name = "num_orders";
136 result.Result = "::model::Grating Transmission::num_orders";
137 addsweepresult(sweepname, result);
138
139 result = struct;
140 result.Name = "grating_efficiency";
141 result.Result = "::model::Grating Transmission::grating_efficiency";
142 addsweepresult(sweepname, result);
143
144 result = struct;
145 result.Name = "first_order_efficiency";
146 result.Result = "::model::Grating Transmission::first_order_efficiency";
147 addsweepresult(sweepname, result);
148
149 result = struct;
150 result.Name = "first_order_theta";
151 result.Result = "::model::Grating Transmission::first_order_theta";
152 addsweepresult(sweepname, result);
153
154 result = struct;
155 result.Name = "grating_n";
```

```
156 result.Result = "::model::Grating Transmission::grating_n";
157 addsweepresult(sweepname, result);
158
159 result = struct;
160 result.Name = "grating_m";
161 result.Result = "::model::Grating Transmission::grating_m";
162 addsweepresult(sweepname, result);
163
164 result = struct;
165 result.Name = "first_order_efficiency_normalized";
166 result.Result = "::model::Grating Transmission::
        first_order_efficiency_normalized";
167 addsweepresult(sweepname, result);
168
169 result = struct;
170 result.Name = "ratio_1_0";
171 result.Result = "::model::polarization ellipse::ratio_1_0";
172 addsweepresult(sweepname, result);
173
174
175 # parent sweep
176 if(sweeppar2_name != "none"){
177     insertsweep(sweepname);
178     sweepname2 = sweeppar2_name+" sweep";
179     setsweep("sweep", "name", sweepname2);
180     setsweep(sweepname2, "type", "Ranges");
181     setsweep(sweepname2, "number of points", num_sweeppar2_sweeps);
182     addsweepparameter(sweepname2, para2);
183
184     result = struct;
185     result.Name = "T";
186     result.Result = "T";
187     addsweepresult(sweepname2, result);
188
189     result = struct;
190     result.Name = "num_orders";
191     result.Result = "num_orders";
192     addsweepresult(sweepname2, result);
193
```

```
194     result = struct;
195     result.Name = "grating_efficiency";
196     result.Result = "grating_efficiency";
197     addsweepresult(sweepname2, result);
198
199     result = struct;
200     result.Name = "first_order_efficiency";
201     result.Result = "first_order_efficiency";
202     addsweepresult(sweepname2, result);
203
204     result = struct;
205     result.Name = "first_order_theta";
206     result.Result = "first_order_theta";
207     addsweepresult(sweepname2, result);
208
209     result = struct;
210     result.Name = "grating_n";
211     result.Result = "grating_n";
212     addsweepresult(sweepname2, result);
213
214     result = struct;
215     result.Name = "grating_m";
216     result.Result = "grating_m";
217     addsweepresult(sweepname2, result);
218
219     result = struct;
220     result.Name = "first_order_efficiency_normalized";
221     result.Result = "first_order_efficiency_normalized";
222     addsweepresult(sweepname2, result);
223
224     result = struct;
225     result.Name = "ratio_1_0";
226     result.Result = "ratio_1_0";
227     addsweepresult(sweepname2, result);
228 }
```

PSO optimization sweep

```
1  ##########################################################################
2  # This script implements a PSO optimization sweep.
3  # Optimization parameters can be added/removed.
4  ##########################################################################
5
6
7  select("::model");
8  mname = "::model::3dpowermonitor";
9  lambda_um = get("lambda_um");
10 lambdas_SI = [lambda_um];
11 ### Here, we choose the paramters to optimize and their allowed ranges ###
12
13
14 radius_start = 0.05e-6;
15 radius_stop = 0.4e-6;
16
17 period_radius_fraction_start = 2;
18 period_radius_fraction_stop = 5;
19
20 groove_depth_start = 0.01e-6;
21 groove_depth_stop = 1e-6;
22
23 blaze_angle_fraction_start = 0;
24 blaze_angle_fraction_stop = 1;
25
26 #coating_thickness_start = 30e-9;
27 #coating_thickness_stop = 200e-9;
28
29
30 # set up parameters for period_radius_fraction
31 select("::model");
32 para1 = struct;
33 para1.Name = "period_radius_fraction";
34 para1.Parameter = "::model::period_radius_fraction";
35 para1.Type = "Number";
36 para1.Min = period_radius_fraction_start;
37 para1.Max = period_radius_fraction_stop;
38
```

```
39 #set up parameters for radius
40 para1_2 = struct;
41 para1_2.Name = "radius";
42 para1_2.Parameter = "::model::hole_radius";
43 para1_2.Type = "Length";
44 para1_2.Unit = "microns";
45 para1_2.Min = radius_start;
46 para1_2.Max = radius_stop;
47
48 #set up parameters for groove depth
49 para2 = struct;
50 para2.Name = "groove depth";
51 para2.Parameter = "::model::groove_depth";
52 para2.Type = "Length";
53 para2.Min = groove_depth_start;
54 para2.Max = groove_depth_stop;
55 para2.Units = "microns";
56
57 #set up parameters for blaze angle
58 para3 = struct;
59 para3.Name = "blaze angle fraction";
60 para3.Parameter = "::model::blaze_angle_fraction";
61 para3.Type = "Number";
62 para3.Min = blaze_angle_fraction_start;
63 para3.Max = blaze_angle_fraction_stop;
64
65 #set up parameters for coating thickness
66 #para4 = struct;
67 #para4.Name = "coating thickness";
68 #para4.Parameter = "::model::coating_thickness";
69 #para4.Type = "Length";
70 #para4.Min = coating_thickness_start;
71 #para4.Max = coating_thickness_stop;
72 #para4.Units = "microns";
73
74
75
76 ##########################################################################
77
```

```
78 # optimization over width of simulation region
79 addsweep(1);
80 sweepname = "period-radius sweep";
81 setsweep("optimization", "name", sweepname);
82 setsweep(sweepname, "type", "Maximize");
83 setsweep(sweepname, "Algorithm", "Particle Swarm");
84 setsweep(sweepname, "Maximum Generations", 200);
85 setsweep(sweepname, "Generation Size", 30);
86
87
88
89 addsweepparameter(sweepname, para1); # period-radius fraction
90 addsweepparameter(sweepname, para1_2); # radius
91 addsweepparameter(sweepname, para2); # groove depth
92 addsweepparameter(sweepname, para3); # blaze angle
93 #addsweepparameter(sweepname, para4); #coating thickness
94
95 result = struct;
96 result.Name = "T";
97 result.Result = "::model::Grating Transmission::T";
98 result.Optimize = false;
99 addsweepresult(sweepname, result);
100
101 result = struct;
102 result.Name = "num_orders";
103 result.Result = "::model::Grating Transmission::num_orders";
104 result.Optimize = false;
105 addsweepresult(sweepname, result);
106
107 result = struct;
108 result.Name = "first_order_efficiency";
109 result.Result = "::model::Grating Transmission::first_order_efficiency";
110 result.Optimize = false;
111 addsweepresult(sweepname, result);
112
113 result = struct;
114 result.Name = "first_order_efficiency_normalized";
115 result.Result = "::model::Grating Transmission::
      first_order_efficiency_normalized";
```

```
116 result.Optimize = true;
117 addsweepresult(sweepname, result);
118
119 result = struct;
120 result.Name = "first_order_theta";
121 result.Result = "::model::Grating Transmission::first_order_theta";
122 result.Optimize = false;
123 addsweepresult(sweepname, result);
```