Coupled Wave Implementation NOTEs

DAVID FLUCKIGER[1, 2]

[1]*Independent Researcher*
[2]*david@gsolver.com*

Abstract: Following is a description of the implementation of the so-called Coupled Wave Algorithm (CWA) for the analysis of diffraction grating structures. While there is extensive literature on CWA, perhaps the novel element in the notes below is with the solution of the boundary conditions. Generally, the usual approach is to group the E and H fields at the top of a layer together and solve for the E and H fields at the bottom of a layer. This creates a potential issue with a possible numerical singularity in the solution of the resulting linear equations. The so-called R, S, T (transfer matrix) algorithmic methods are used to mitigate numerical problems. By regrouping the E and h fields at the top/bottom of the layer the potentially singular matrices, regardless of layer thickness, give a well-posed numerical algorithm.

I. Coordinate System

Incident plane wave travels towards +inf note that +z is shown down: plane wave phase is taken as

Incident wave = $\widehat{\boldsymbol{E_o}}\exp(i(\mathbf{k}\cdot\mathbf{r}-\omega t))$ (1)

(same phase convention as in Jackson).

Use the following transform on (normalized) $\hat{\boldsymbol{k}}$ = [0,0,1] to the desired input $\hat{\boldsymbol{k}}$ for some θ, ϕ.

$$T = \begin{bmatrix} \cos(\theta)\cos(\varphi) & -\sin(\varphi) & \sin(\theta)\cos(\varphi) \\ \cos(\theta)\sin(\varphi) & \cos(\varphi) & \sin(\theta)\sin(\varphi) \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix} \quad (2)$$

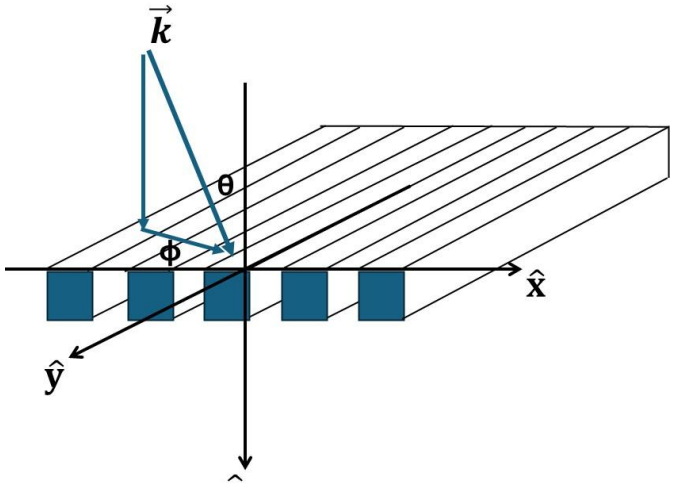

Figure 1 Coordinate System

For **normal incidence** the TE mode E-field is E_0 = [0, 1, 0]', for TM mode the E-field is E_0 = [1,0,0]', and for arbitrary polarization (see next section) E_0 = [e_x, e_y,0], where e_x, and e_y are complex numbers. The $\mathbf{E_0}$ vector for arbitrary angle of incidence becomes

$$\boldsymbol{E_{inc}} = T(\theta,\varphi)\begin{bmatrix} e_x \\ e_y \\ 0 \end{bmatrix} \quad (3)$$

II. Polarization

At normal incidence the (arbitrary) polarization $\mathbf{E_0}$ vector is given by

e_x = sin(alpha)*cos(beta) + *i**cos(alpha)*sin(beta) (4)
e_y = cos(alpha)*cos(beta) - *i**sin(alpha)*sin(beta)

for the complex magnitudes of e_x and e_y. Note that the complex part determines the secondary (E2) field and is delayed by 90°. For example:

α = 0°, β = 0° is TE mode
α = 90°, β = 0° is TM mode
α = any°, β = 45° is left-circular (and -45 for right-circular)

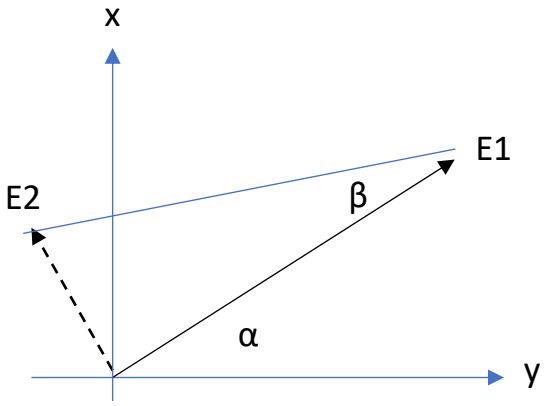

Figure 2 Polarization Angles

III. Fourier Basis

Coupled Wave theory uses a Fourier basis to (infinite plane waves) define the fields in all regions (superstrate, grating layers, substrate). The periodicity (grating period = Λ) of the problem allows a Fourier base to be used in every computational region: superstrate (semi-infinite z<0, homogeneous, isotropic, linear medium, with real index of refraction); substrate (semi-infinite z>thickness of grating, LHI medium with real or complex index of refraction); grating region, defined by the periodic, piecewise constant approximation of one or several layers each with grating period = Λ.

The magnitude of the incident k-vector in vacuum is

$$k_0 = \frac{2\pi}{\lambda} \qquad (5)$$

It is convenient to normalize all k-vectors by k_0. Thus, $k_{inc} = k_1$ (region 1 is taken to be the superstrate, region 3 is substrate) for normal incidence, is given by

$$k_1 = T(\theta,\varphi)\begin{bmatrix}0\\0\\n_1\end{bmatrix} = n_1[\sin\theta\cos\varphi\,,\sin\theta\sin\varphi\,,\cos\theta] = \left[k_{1x},k_{1y},k_{1z}\right] \qquad (6)$$

The fields in the superstrate (incident plane wave, and reflected diffracted waves, and the substrate, transmitted diffracted waves) are expanded in the following way (consistent with Floquet theory).

$$\begin{aligned}\boldsymbol{E_{inc}} &= \boldsymbol{E_0}\exp\left(i\left(k_{incx}x + k_y y + k_{incz}z\right)\right.\\ \boldsymbol{R}_{ref} &= \sum \boldsymbol{R}_n \exp\left(-i(k_{nx}x + k_y y - k_{1nz}z)\right.\\ \boldsymbol{T}_{trn} &= \sum \boldsymbol{T}_n \exp\left(-i(k_{nx}x + k_y y - k_{3nz}z)\right.\end{aligned} \qquad (7)$$

where $\mathbf{E_0}$ is incident E-field vector magnitude (eqn. 6), and $\mathbf{R}_n$, $\mathbf{T}_n$ are the E-fields for the reflected and transmitted waves. Note that boundary conditions require that the transverse E-fields are continuous across layer boundaries (thus k_y is uniform across each layer interface). Expanding on k_x (and k_z) are found as follows

$$k_{xn} = k_x - \frac{\lambda}{\Lambda}(-N{:}N) \qquad (8)$$

Where N is an integer (-N:N = [-N, -N+1, . . ., 0 ,1,. . .,N-1,N]) and represents the upper spatial cut-off frequency retained in the analysis. There are therefore a total of 2N+1 Fourier terms. k_{zn} is formed from k_{xn} and k_y as follows

$$\begin{aligned}k_{z1n} &= conj\left(sqrt\left(e_1 - {k_{xn}}^2 - {k_y}^2\right)\right)\\ k_{z3n} &= conj\left(sqrt\left(e_3 - {k_{xn}}^2 - {k_y}^2\right)\right)\end{aligned} \qquad (9)$$

Where e1 = ${n_1}^2$ and $e_3 = {n_3}^2$ (n_3 is the possibly complex index of refraction of the substrate). The conj() function takes the conjugate of the sqrt branch to ensure that we have wave attenuation in absorbing media. In the following k_{xn}, k_{z1n}, k_{z3n}, are configured as collum vectors from -N to N. For the choice of phase (eqn. 1), the index of refraction has real(n)>0, imag(n)>=0). Note that the conjugate function makes Im(k_{1nz}, k_{3nz}) < 0 for attenuated/evanescent waves.

IV. Field Expansions in a grating layer

The **E** and **H** fields in each grating layer may be expanded in Fourier Series as

$$\boldsymbol{E}_{layer} = \sum \boldsymbol{S}_n(z)\exp\left(-i(k_{xn}\,x + k_y y)\right) \qquad (10)$$
$$\boldsymbol{H}_{layer} = \sum \boldsymbol{U}_n(z)\exp\left(-i(k_{xn}\,x + k_y y)\right)$$

Where the expansion coefficients are functions of z: $\mathbf{S}_n = [S_{nx}(z), S_{ny}(z), S_{nz}(z)]$, and similarly for the **H** fields ($\mathbf{U}_n(z)$).

It is convenient to normalize the H field by

$$H \to -i\sqrt{\frac{\mu_0}{\varepsilon_0}} \qquad [\text{with } \omega = \frac{k_o}{\sqrt{\varepsilon_0\mu_0}}] \qquad (11)$$

the curl equations then become

$$\boldsymbol{\nabla}\times\boldsymbol{E} = \boldsymbol{H}, \quad \boldsymbol{\nabla}\times\boldsymbol{H} = \varepsilon(x)\boldsymbol{E} \qquad (12)$$

Note that I set $\mu_r = 1$. (for convenience, this could be expanded similarly to the permittivity for analysis of magnetic materials), and ε(x) is a periodic function of x (in each grating layer). Expanding the curl equations gives

$$\nabla\times\boldsymbol{E} = \boldsymbol{H} = \begin{bmatrix}\hat{x} & \hat{y} & \hat{z}\\ \partial x & \partial y & \partial z\\ S_x & S_y & S_z\end{bmatrix} = \begin{bmatrix}U_x\\ U_y\\ U_z\end{bmatrix} = \begin{bmatrix}-ik_y S_z - S'_y\\ S'_x + ik_{xn}S_z\\ -ik_{xn}S_y + ik_y S_x\end{bmatrix}$$
$$\nabla\times\boldsymbol{H} = \epsilon\boldsymbol{E} = \begin{bmatrix}\hat{x} & \hat{y} & \hat{z}\\ \partial x & \partial y & \partial z\\ U_x & U_y & U_z\end{bmatrix} = \begin{bmatrix}\epsilon S_x\\ \epsilon S_y\\ \epsilon S_z\end{bmatrix} = \begin{bmatrix}-ik_y U_z - U'_y\\ U'_x + ik_{xn}U_z\\ -ik_{xn}U_y + ik_y U_x\end{bmatrix} \qquad (13)$$

Note that each S/U component is a column vector of 2N+1 coefficients (which depend on z).

Eliminate the z components (of S and U) and solve for the two transverse components:

$$\begin{bmatrix}S'_x\\ S'_y\end{bmatrix} = P\begin{bmatrix}U_x\\ U_y\end{bmatrix} = \begin{bmatrix}k_{xn}\epsilon^{-1}k_y & I - k_{xn}\epsilon^{-1}k_{xn}\\ k_y\epsilon^{-1}k_y - I & -k_y\epsilon^{-1}k_{xn}\end{bmatrix}\begin{bmatrix}U_x\\ U_y\end{bmatrix}$$

$$\begin{bmatrix}U'_x\\ U'_y\end{bmatrix} = Q\begin{bmatrix}S_x\\ S_y\end{bmatrix} = \begin{bmatrix}k_{xn}k_y & \epsilon - k_{xn}^2\\ k_y^2 - \epsilon & -k_y k_{xn}\end{bmatrix}\begin{bmatrix}S_x\\ S_y\end{bmatrix} \qquad (14)$$

Which yields the second order equation

$$\begin{bmatrix}S''_x\\ S''_y\end{bmatrix} = PQ\begin{bmatrix}S_x\\ S_y\end{bmatrix} \qquad (15)$$

This second order differential (matrix) equation has the solution

$$\begin{bmatrix} S_x \\ S_y \end{bmatrix} = We^{\lambda z}c_1 + We^{\lambda(d-z)}c_2 \qquad (16)$$

Where c1, and c2 are 2(2N+1) column vectors to be found from the boundary conditions, and W is the eigenvector (matrix) solution of PQ, and λ = -sqrt(Λ2), where Λ2 are the eigenvalues of the matrix PQ. Note that the -1 sign gives decreasing terms.

The permittivity convolution matrix (**ε**) is formed from the Fourier expansion of the permittivity in the grating layer. 2*(2N+1) terms are needed. The Fourier terms are then order as a Toeplitz matrix in the usual convolutional matrix operator.

To find the H fields (U_x, U_y) assume the following solution

$$\begin{bmatrix} U_x \\ U_y \end{bmatrix} = Ve^{\lambda z}c_1 - Ve^{\lambda(d-z)}c_2 \qquad (17)$$

Which (using the Q matrix to relate to first derivative of S) implies that

$$V = QW\lambda^{-1} \qquad (18)$$

Where λ is the diagonal matrix of the PQ eigenvalues. Note that the dimensions of P, Q, W are [2*(2N+1) x 2*(2N+1)].

V. Super/Substrate Fields (H from E as needed)

Take the reflected/transmitted E-fields in super/substrate regions as the unknown (R, T). Use the curl equations (12) to find the linear operators (label as C matrices) which give the $H_{x,y}$ fields in terms of the $E_{x,y}$ field components**.** The divergence free criteria, **k.E** = 0, is used to find the remaining z component.

Incident Field

$$\begin{bmatrix} H_{0x} \\ H_{0y} \end{bmatrix} = C_1 \begin{bmatrix} E_{0x} \\ E_{0y} \end{bmatrix}, \quad C_1 = i \begin{bmatrix} \frac{k_{1x}k_{1y}}{k_{1z}} & \left(\frac{k_{1y}^2}{k_{1z}} + k_{1z}\right) \\ -\left(\frac{k_{1x}^2}{k_{1z}} + k_{1z}\right) & -\frac{k_{1x}k_{1y}}{k_{1z}} \end{bmatrix} \qquad (19)$$

Reflected (Superstrate) Fields

$$\begin{bmatrix} H_{Rnx} \\ H_{Rny} \end{bmatrix} = C_2 \begin{bmatrix} R_{nx} \\ R_{ny} \end{bmatrix}, \quad C_2 = i \begin{bmatrix} -\frac{k_{nx}k_{1y}}{k_{1nz}} & -\left(\frac{k_{1y}^2}{k_{1nz}} + k_{1nz}\right) \\ \left(\frac{k_{nx}^2}{k_{1nz}} + k_{1nz}\right) & \frac{k_{nx}k_{1y}}{k_{1nz}} \end{bmatrix} \qquad (20)$$

Note that H_{Rnx}, and H_{Tny}, are each (2N+1 x 1) column vectors, and that C2 is a (4N+2 x 4N+2) matrix.

Transmitted Fields

$$\begin{bmatrix} H_{Tnx} \\ H_{Tny} \end{bmatrix} = C_3 \begin{bmatrix} T_{nx} \\ T_{ny} \end{bmatrix}, \qquad C_3 = i \begin{bmatrix} \frac{k_{nx}k_{1y}}{k_{3nz}} & \left(\frac{k_{1y}^2}{k_{3nz}} + k_{3nz}\right) \\ -\left(\frac{k_{nx}^2}{k_{3nz}} + k_{3nz}\right) & -\frac{k_{nx}k_{1y}}{k_{3nz}} \end{bmatrix} \qquad (21)$$

VI. Fields at top/bottom of grating layers from Boundary Conditions

For each layer find the **P** and **Q** matrices (eqn. 14). Form PQ and solve for the eigenvector (matrix) **W** and the (diagonal eigenvalue matrix) **Λ**, then construct the **V** matrix (eqn. 18), and the **X** matrix (=exp(λk_0d), where d = thickness of layer), **X** is a diagonal matrix. Now note that the fields at the top and bottom of the grating layer can be written as follows:

$$\begin{bmatrix} S_x \\ S_y \\ U_x \\ U_y \end{bmatrix}_{top} = \begin{bmatrix} W & WX \\ V & -VX \end{bmatrix} \begin{bmatrix} c1 \\ c2 \end{bmatrix}, \qquad \begin{bmatrix} S_x \\ S_y \\ U_x \\ U_y \end{bmatrix}_{bot} = \begin{bmatrix} WX & W \\ VX & -V \end{bmatrix} \begin{bmatrix} c1 \\ c2 \end{bmatrix} \qquad (22)$$

$$X = e^{\lambda k_0 d}$$

Also note that real(λ) <= 0. The usual formulation of the boundary conditions is to solve for the unknown constants (c's) using equations (22) iteratively through the layer stack. Observing eqn. 22 indicates that the matrices might be ill-conditioned for the case there the X matrix becomes singular (in particular when there are large (negative) eigenvalues and/or for thick gratings). There are several algorithms (R, S, T, etc.) which employ methods to mitigate the effects of near singular matrices. However, it is a simple matter to avoid/eliminate the impact of small X entries by re-arranging (22) as follows:

$$\begin{bmatrix} S_x^{top} \\ S_y^{top} \\ U_x^{bot} \\ U_y^{bot} \end{bmatrix} = \begin{bmatrix} W & WX \\ VX & -V \end{bmatrix} \begin{bmatrix} WX & W \\ V & -VX \end{bmatrix}^{-1} \begin{bmatrix} S_x^{bot} \\ S_y^{bot} \\ U_x^{top} \\ U_y^{top} \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} \\ h_{21} & h_{22} \end{bmatrix} \begin{bmatrix} S_x^{bot} \\ S_y^{bot} \\ U_x^{top} \\ U_y^{top} \end{bmatrix} \qquad (23)$$

Each field entry consists of 2N+1 unknowns, so the column vectors have dimensions [4(2N+1), 1]. For each layer there is an H matrix (eqn. 23) that relates the E_{top}, H_{bot} fields to the E_{bot}, H_{top} fields. What is needed is a method to combine H matrices that relates the E_{top}, H_{bot} fields from layer n, to the E_{top}, H_{bot} fields of layer n+1, and iterate through all layers so that one ends up with the interface at the superstrate, related to the interface at the substrate. The combined H(layer n) matrix and H(layer n+1) matrix is as follows:

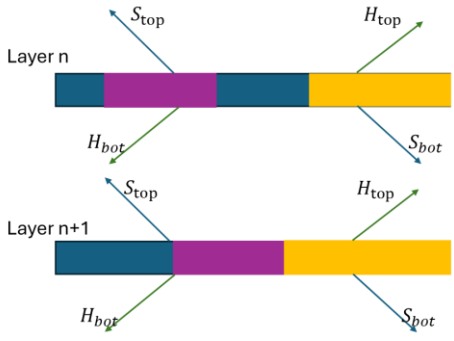

Figure 3 Combining Layers

Call the layer n H, H^A, and n+1 H, H^B. Define a D and E matrix as follows:

$$D = h_{11}^B \left(I - h_{12}^A h_{21}^B\right)^{-1} \qquad E = h_{22}^A \left(I - h_{21}^B h_{12}^A\right)^{-1} \qquad (23)$$

And then the combined H is given by (after some algebra)

$$
\begin{aligned}
h_{11} &= Dh_{11}^{A} \\
h_{12} &= Dh_{12}^{A}h_{22}^{B} + h_{12}^{B} \\
h_{21} &= Eh_{21}^{B}h_{11}^{A} + h_{21}^{A} \\
h_{22} &= Eh_{22}^{B}
\end{aligned}
\qquad (24)
$$

This yields an H matrix which relates fields top/bottom for the combined n and n+1 layers. This new H matrix is then combined (using eqn 23, 24) with the next layer. The result is a global H matrix that relates the fields at the top/bottom for all the layers with 4(2N+1) unknowns. The final boundary condition solver uses the global H matrix to relate the fields in the superstrate to substrate and solves for all constants/field magnitudes in terms of the input field.

VII. Final Boundary Condition

The known fields are the input E_0, H_0 of the incident plane wave. Form a 2(2N+1) vector of 0's and assign E_{0x} to the N+1st position, and E_{0y} to the 3N+2nd position. Note that the final boundary conditions (given the global H matrix) can be written as the following coupled equations:

$$
\begin{bmatrix} E_{0x} \\ E_{oy} \end{bmatrix} + \begin{bmatrix} R_x \\ R_y \end{bmatrix} = h_{11}\begin{bmatrix} T_x \\ T_y \end{bmatrix} + h_{12}\left(\begin{bmatrix} H_{0x} \\ H_{0y} \end{bmatrix} + C_2\begin{bmatrix} R_x \\ R_y \end{bmatrix}\right) \qquad (25)
$$

$$
C_3\begin{bmatrix} T_x \\ T_y \end{bmatrix} = h_{21}\begin{bmatrix} T_x \\ T_y \end{bmatrix} + h_{22}\left(\begin{bmatrix} H_{0x} \\ H_{0y} \end{bmatrix} + C_2\begin{bmatrix} R_x \\ R_y \end{bmatrix}\right)
$$

Where the C matrices are given in eqn's 19-21. These simultaneous equations can be solved as follows:

$$
\begin{bmatrix} I & -h_{12} \\ 0 & -h_{22} \end{bmatrix}\begin{bmatrix} E_{0x} \\ E_{0y} \\ H_{0x} \\ H_{0y} \end{bmatrix} = \begin{bmatrix} h_{12}C_2 - I & h_{11} \\ h_{22}C_2 & h_{21} - C_3 \end{bmatrix}\begin{bmatrix} R_x \\ R_y \\ T_x \\ T_y \end{bmatrix} \qquad (26)
$$

The column vector on the left is a [4(2N+1) x 1] vector of 0's with the E_{0x} value in position N+1, E_{0y} in 3N+2, H_{0x} in 5N+3, and H_{0y} in 7N+4. Equation 26 is solved for the R/T column vector.

This gives the transverse components of the (E/H) field magnitudes at the super/substrate interfaces. To find the diffraction efficiencies we also need the z components which are found from the divergence free condition as follows:

```
Rz = (kxn'.*Rx+ky*Ry)./kzn1';                    (27)
Tz = -(kxn'.*Tx+ky*Ty)./kzn3';
```

Find the field magnitudes:

```
Tn =  Tnx.*conj(Tnx)+Tny.*conj(Tny)+ Tnz.*conj(Tnz)                    (28)
Rn = Rnx.*conj(Rnx)+Rny.*conj (Rny)+Rnz.*conj(Rnz)
```

And the diffraction efficiencies:

DER_n = real(k_{zn1}/(k_{0z}).*R_n') (29)
DET_n = real(k_{zn3}/(k_{0z}).*T_n')

VIII. Polarization

To recover the polarization of each diffracted wave, transform each E-field (e.g., **R**(n) = (Rn_x, Rn_y, Rn_z)) back to the normal, using the correct k_{1nz} term (for super/substrate as in k_{zn1}, k_{zn3}). This will set R_{nz} = 0 (or T_{nz}) then eqn. 4 can be inverted to give the polarization angles for each component:

α_n = atan2(real(E_x), real(E_y))*180/π (30)
β_n = atan2(imag(E_y), real($E0_x$))*180/π

We are only interested in the field magnitudes that propagate away from the super/substrate (not the evanescent waves.

IX. Simplifications for TE Mode

For TE, k_y=0, E0 = [0, 1, 0]

The transverse component of H is H_x.

$$S''_y = PQS_y$$
$$PQ = -\varepsilon + k_{xn}^2$$
$$[W, \Lambda 2] = eig(PQ)$$
$$V = QW\lambda^{-1}$$

Note that now size(PQ) = 2N+1 x 2N+1.

$$C_1 = diag(i\, k_{0z1}), \qquad C_2 = diag(-i\, k_{nz1}) \qquad C_3 = diag(i\, k_{nz3})$$

```
res = zeros(2*nn,1);
res(1:nn) = -Hg.h12(:,N+1)*(1i*k0z);
res(N+1) = res(N+1)+1;
res(nn+(1:nn)) = -Hg.h22(:,N+1)*1i*k0z;
```

$$\begin{bmatrix} I & -h_{12} \\ 0 & -h_{22} \end{bmatrix} \begin{bmatrix} E_{0y} \\ H_{0x} \end{bmatrix} = \begin{bmatrix} h_{12}C_2 - I & h_{11} \\ h_{22}C_2 & h_{21} - C_3 \end{bmatrix} \begin{bmatrix} R_y \\ T_y \end{bmatrix}$$

Ry, Ty are the only non-zero TE E-field components.

X. Simplifications for TM

For TM, k_y = 0, H0 = [0, n1, 0]

The transverse component of E is E_x.

$$U''_y = QPU_y$$
$$QP = -\varepsilon(I - k_{kn}\varepsilon^{-1}k_{kn})$$
$$[V, \Lambda 2] = eig(QP)$$

$$U'_y = QS_x = -\epsilon S_x$$

$$S'_x = PU_y = (I - k_{xn}\epsilon^{-1}k_{xn})U_y$$

The V matrix, and λ are known

$$U_y = Ve^{\lambda z}c_1 - Ve^{\lambda(d-z)}c_2$$

$$S_x = We^{\lambda z}c_1 + We^{\lambda(d-z)}c_2 \quad \text{implies that} \quad W = -\epsilon^{-1}V\Lambda$$

We now have (see above) $C_1^{-1}\begin{bmatrix}H_{0x}\\H_{0y}\end{bmatrix} = \begin{bmatrix}E_{0x}\\E_{0y}\end{bmatrix}$, or just get E0 for the transformation matrix

Use above final BC solver to finish off (note need Rz, and Tz from divergence free condition).

Appendix

Matlab syntax code for general polarization in conical mount example

Code for Conical Mode

```
% Conical mode CWA code
%       Copyright (c) 2025 David Fluckiger
%       Author: David Fluckiger
%       Date: February 2025, revision: 1.0
%
% Permission is hereby granted, free of charge, to any person obtaining a copy of this software
% and associated documentation files (the "Software"), to deal in the Software without restriction,
% including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense,
% and/or sell copies of the Software, and to permit persons to whom the Software is furnished to
% do so, subject to the following conditions:
%
% The above copyright notice and this permission notice shall be included in all copies or
%substantial portions of the Software.

%% ===========Global Grating Problem Definition===============
%clear all;
%close all;

% number of orders (+/-N)
N = 15;

% define input plane wave
lambda = 0.6328;
Period =1.2;
theta = 30.;            % angle in degrees
phi =10;
alpha = 0.;             %polarization angles
beta =0.;

n1 = 1.;                %superstrate index of refraction (expected to be real)
n3 = 3.13737 + 0.07562i;    %substrate index of refraction
e1 = n1^2;              %permittivites in super/substrate
e3 = n3^2;

% define layers for device
xl{1} = [0., .5,1.];    %position of index regions (note normalized scaling)
eps{1} = [n3,n1].^2;    %permittivity in each region
dl = .5;                %thickness of this layer

xl{2} = [0., .25,1.];   %another layer
eps{2} = [n1,2.17+.04i].^2;
dl(2) = .5;
```

```
xl{3} = [0., .72,1.];        %another alyer
eps{3} = [1.8, n1].^2;
dl(3) = .3;

% This loop will be for varying theta, as an example
loop = linspace(0,99,45)+1.e-5;

% preallocate some arrays
Ref = zeros(length(loop), 5);          %superstrate/reflection diffraction efficiencies
Trn = zeros(length(loop), 5);          %substrate/transmission diffraction efficiencies
TrnAlpha = zeros(length(loop),3);      %polarization angles
RefAlpha = zeros(length(loop),3);
TrnBeta = zeros(length(loop),3);
RefBeta = zeros(length(loop),3);

kk = 1;          % convenient indexing variable

n = 2*N+1;       %number of total Fourier components

% main loop. This one is for theta, use as example for looping on other parameters
for theta =  loop
        %find the input plane wave E-field components
        [k0I,E0,~,Tr] = Transform(theta,phi,alpha,beta);
        k0 = 2*pi/lambda;
        nn = n;  %for sizing arrays
        % derived quantities
        kx = n1*k0I(1);                          % all k's are normalized to k0 = 2*pi/lambda
        ky = n1*k0I(2);
        kz = n1*k0I(3);
        kxn = kx - (-N:N)*lambda/Period;
        kzn1 = conj(sqrt(e1-kxn.^2-ky^2));
        kzn3 = conj(sqrt(e3-kxn.^2-ky^2));

        II = eye(nn);
        Z = zeros(2*nn);
        Kx = diag(kxn);
        Ky = ky*eye(nn);
        Kz1 = diag(kzn1);
        Kz3 = diag(kzn3);

        nn = 2*nn;
        II2 = eye(nn);
        % initialize the global H matrix
        Hg.h11 = eye(nn);
        Hg.h12 = Z;
        Hg.h21 = Z;
        Hg.h22 = eye(nn);

% here is the loop through the grating layers
        for j = 1:length(xl)
```

```
        gxl = xl{j};
        epsl = eps{j};
    amn= FourierCoef(epsl,gxl,2*N);
    Amn=toeplitz(amn(2*N+1:4*N+1),amn(2*N+1:-1:1));
    bmn= FourierCoef(1./epsl,gxl,2*N);
    Bmn=toeplitz(bmn(2*N+1:4*N+1),bmn(2*N+1:-1:1));

    P = [(Kx*Bmn)*Ky,    II-Kx*Bmn*Kx;...
            Ky*Bmn*Ky-II, -Ky*(Bmn*Kx)];
    Q = [Kx*Ky, Amn-Kx*Kx;...
            Ky*Ky-Amn, -Ky*Kx];
    [W,Lam] = eig(P*Q);
    Lam = -(sqrt(diag(Lam)));
    V = Q*W*diag(1./Lam);
    X = diag(exp(k0*dl(j)*Lam));
    H1 = [W, W*X;V*X, -V]/[W*X, W;V, -V*X];
    H.h11 = H1(1:nn,1:nn);
    H.h12 = H1(1:nn,nn+(1:nn));
    H.h21 = H1(nn+(1:nn),1:nn);
    H.h22 = H1(nn+(1:nn),nn+(1:nn));
    Hg = Star(Hg,H,II2);
end      % end of loop through layers, have global H matrix

% final boundary conditions, construct the C matrices
C1 = 1i*[ky*kx/kz, ky*ky/kz+kz; -kx*kx/kz-kz, -kz*ky/kz];
H0 = C1*[E0(1);E0(2)];

C2 = 1i*[diag(-ky*kxn./kzn1), diag(-ky*ky./kzn1-kzn1);...
            diag(kxn.^2./kzn1+kzn1), diag(kxn*ky./kzn1)];
C3 = 1i*[diag(ky*kxn./kzn3), diag(ky*ky./kzn3+kzn3);...
            diag(-kxn.^2./kzn3-kzn3), diag(-kxn*ky./kzn3)];

% initialize the input E and H array
res = zeros(2*nn,1);
res(N+1) = E0(1);
res(3*N+2) = E0(2);
res(5*N+3) = H0(1);
res(7*N+4) = H0(2);

% solve the final BC here
F1 = [II2, -Hg.h12; Z, -Hg.h22];
F2 = [Hg.h12*C2-II2, Hg.h11; Hg.h22*C2, Hg.h21-C3];
RT = F2\F1*res;

% extract out the x, y E-field components
Rx = RT(1:n);
Ry = RT(n+(1:n));
Rz = (kxn'.*Rx+ky*Ry)./kzn1';
Tx = RT(2*n+(1:n));
Ty = RT(3*n+(1:n));
```

```
        Tz = -(kxn'.*Tx+ky*Ty)./kzn3';

        % calculate the field magnitudes
        T =  Tx.*conj(Tx)+Ty.*conj(Ty)+ Tz.*conj(Tz);
        R = Rx.*conj(Rx)+Ry.*conj(Ry)+Rz.*conj(Rz);

        % diffraction efficiency is fraction of energy that propagates away from the grating
        R = real(kzn1/(kz).*R');
        T = real(kzn3/(kz).*T');

        % here are the Diffraction efficiencies, will select out some to plot
        Ref(kk,:) = R((-2:2)+N+1);
        Trn(kk,:) = T((-2:2)+N+1);

        % look at phase of Ref(-1:1) and Trn(-1:1) orders
        for j = -1:1
                kp = [kxn(N+1+j); ky; real(kzn1(N+1+j))];
                EE = [Rx(N+1+j); Ry(N+1+j); Rz(N+1+j)];
                [alp, bet,kh0,Eh0,iT] = invTransform(phi,kp,EE);
                RefAlpha(kk,j+2) = alp;
                RefBeta(kk,j+2) = bet;

                kp = [kxn(N+1+j); ky; kzn3(N+1+j)];
                EE = [Tx(N+1+j); Ty(N+1+j); Tz(N+1+j)];
                [alp, bet,kh0,Eh0,iT] = invTransform(phi,kp,EE);
                TrnAlpha(kk,j+2) = alp;
                TrnBeta(kk,j+2) = bet;
        end
        kk = kk+1;

        % makes some plots of results as they accrue
        figure(1);
        plot(loop,Ref,loop,Trn);grid
        title('Defraction Efficiency');

        figure(2);
        plot(loop,TrnAlpha,loop,TrnBeta); grid
        title('Transmission \alpha, \beta');
        figure(3);
        plot(loop,RefAlpha,loop,RefBeta); grid;
        title('Reflection \alpha, \beta');
        drawnow;
end     %get next parameter
```

Subroutines:

```
function [k0,E0,H0,T] = Transform(theta,phi,alpha,beta)
%       Copyright (c) 2025 David Fluckiger
%       Author: David Fluckiger
```

```
%       Date: February 2025, revision: 1.0
%
% Permission is hereby granted, free of charge, to any person obtaining a copy of this software
% and associated documentation files (the "Software"), to deal in the Software without restriction,
% including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense,
% and/or sell copies of the Software, and to permit persons to whom the Software is furnished to
% do so, subject to the following conditions:
%
% The above copyright notice and this permission notice shall be included in all copies or
%substantial portions of the Software.

% geometric transform to rotate vector in 3-space

T = [cosd(theta)*cosd(phi), -sind(phi), sind(theta)*cosd(phi); ...
        cosd(theta)*sind(phi), cosd(phi), sind(theta)*sind(phi); ...
        -sind(theta), 0., cosd(theta)];

ax = sind(alpha)*cosd(beta)+1i*cosd(alpha)*sind(beta);
ay = cosd(alpha)*cosd(beta)-1i*sind(alpha)*sind(beta);

k0 = T*[0,0,1]';
E0 = T*[ax;ay;0];
H0 = cross(k0,E0);
```

```
function [alpha, beta,k0,E0,iT] = invTransform(phi,k,E)
%       Copyright (c) 2025 David Fluckiger
%       Author: David Fluckiger
%       Date: February 2025, revision: 1.0
%
% Permission is hereby granted, free of charge, to any person obtaining a copy of this software
% and associated documentation files (the "Software"), to deal in the Software without restriction,
% including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense,
% and/or sell copies of the Software, and to permit persons to whom the Software is furnished to
% do so, subject to the following conditions:
%
% The above copyright notice and this permission notice shall be included in all copies or
%substantial portions of the Software.

% geometric transform to rotate vector in 3-space
k = real(k);
if size(k,1)==1, k = k'; end
k = k/norm(k);
E = E - (k'*E)*k;                       % subtract off the evanescent portion of wave

theta = acosd(k(3));
iT = [cosd(theta)*cosd(phi), cosd(theta)*sind(phi),-sind(theta); ...
        -sind(phi), cosd(phi), 0; ...
        cosd(phi)*sind(theta), sind(theta)*sind(phi), cosd(theta)];
```

```
if size(E,1)==1, E = conj(E'); end
k0 = iT*k;
E0 = iT*E;
if abs(E0(3))>1E-7
	theta = -theta;
	iT = [cosd(theta)*cosd(phi), cosd(theta)*sind(phi),-sind(theta); ...
		-sind(phi), cosd(phi), 0; ...
		cosd(phi)*sind(theta), sind(theta)*sind(phi), cosd(theta)];
	k0 = iT*k;
	E0 = iT*E;
end

alpha = atan2(real(E0(1)),real(E0(2)))*180/pi;
alpha = mod(alpha,90);
beta = atan2(imag(E0(1)),real(E0(2)))*180/pi;
if beta==-135,
	beta = -45;
else
	beta = 45-mod(beta+45,90);
end
```

```
function t = FourierCoef(eps,xn,N)
%	Copyright (c) 2025 David Fluckiger
%	Author: David Fluckiger
%	Date: February 2025, revision: 1.0
%
% Permission is hereby granted, free of charge, to any person obtaining a copy of this software
% and associated documentation files (the "Software"), to deal in the Software without restriction,
% including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense,
% and/or sell copies of the Software, and to permit persons to whom the Software is furnished to
% do so, subject to the following conditions:
%
% The above copyright notice and this permission notice shall be included in all copies or
%substantial portions of the Software.

%compute the 2N+1 Fourier Coefficients of the piecewise function

if length(N) == 1
	t = zeros(2*N+1,1);
	for i = -N:N
		for j = 1:length(eps)
		if i~=0
		t(i+N+1) = t(i+N+1)+eps(j)*1i*(exp(2*1i*pi*i*xn(j))-
exp(2*1i*pi*i*xn(j+1)))/2/pi/i;
		else
		t(i+N+1) = t(i+N+1)+eps(j)*(xn(j+1)-xn(j));
		end
```

```
                end
                %t(i+N+1) = t(i+N+1)+diff([eps(end),eps])/2*exp(1i*2*pi*i*xn(1:end-1))';
        end
else
        k = 0;
        t = zeros(length(N),1);
        for i = 1:length(N)
                k = k+1;
                n = N(i);
                for j = 1:length(eps)
                if n~=0
                t(k) = t(k)+eps(j)*1i*(exp(2*1i*pi*n*xn(j))-exp(2*1i*pi*n*xn(j+1)))/2/pi/n;
                else
                t(k) = t(k)+eps(j)*(xn(j+1)-xn(j));
                end
                end
        end
end
        t = conj(t);
```

```
function H = Star(H1,H2,II)
%       Copyright (c) 2025 David Fluckiger
%       Author: David Fluckiger
%       Date: February 2025, revision: 1.0
%
% Permission is hereby granted, free of charge, to any person obtaining a copy of this software
% and associated documentation files (the "Software"), to deal in the Software without restriction,
% including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense,
% and/or sell copies of the Software, and to permit persons to whom the Software is furnished to
% do so, subject to the following conditions:
%
% The above copyright notice and this permission notice shall be included in all copies or
%substantial portions of the Software.

% Combining two H matrices
D = H2.h11/(II-H1.h12*H2.h21);
E = H1.h22/(II-H2.h21*H1.h12);
H.h11 = D*H1.h11;
H.h12 = D*H1.h12*H2.h22+H2.h12;
H.h21 = E*H2.h21*H1.h11+H1.h21;
H.h22 = E*H2.h22;
```

Code for TM Mode

```
% TM mode CWA code
%       Copyright (c) 2025 David Fluckiger
%       Author: David Fluckiger
%       Date: February 2025, revision: 1.0
%
% Permission is hereby granted, free of charge, to any person obtaining a copy of this software
% and associated documentation files (the "Software"), to deal in the Software without restriction,
% including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense,
% and/or sell copies of the Software, and to permit persons to whom the Software is furnished to
% do so, subject to the following conditions:
%
% The above copyright notice and this permission notice shall be included in all copies or
%substantial portions of the Software.

%% ==========Global Grating Problem Definition===============
%clear all;
%close all;
%H = [];  dl = [];  eps = [];  xl=[];

    N = 35;

    lambda = .6328;
    Period = .5;
    theta = 30.;

    n1 = 1.;
    n3 = 10i;
    e1 = n1^2;
    e3 = n3^2;

    % define layers for device
    xl{1} = [0., .5,1.];
    eps{1} = [e3,e1];
    dl(1) =.5;

    % This example will loop through the duty cycle
    loop = linspace(0,1,725);

    Ref = zeros(length(loop), 3);
    Trn = zeros(length(loop), 3);
    kk = 1;

    % main loop through parameter (device duty cycle in this example
    for dx =  loop
        nn = 2*N+1;      % for duty cycle these assignments can be taken out of the loop
        E0 = [cosd(theta), 0, -sind(theta)];
```

```
  k0I = [sind(theta), 0, cosd(theta)];

  % derived quantities
  k0 = 2*pi/lambda;
  k1x = n1*sind(theta);                    % all k's are normalized to k0 = 2*pi/lambda
  k1z = n1*cosd(theta);
  kxn = k1x - (-N:N)*lambda/Period;
  kzn1 = conj(sqrt(e1-kxn.^2));
  kzn3 = conj(sqrt(e3-kxn.^2));

  II = eye(2*N+1);
  Z = zeros(2*N+1);
  Kx = diag(kxn);
Hg.h11 = II;
Hg.h12 = Z;
Hg.h21 = Z;
Hg.h22 = II;

  % loop through device layers
  for j = 1:length(xl)
        gxl(2) = dx;
        epsl = eps{j};
        amn= FourierCoef(epsl,gxl,2*N);
        bmn= FourierCoef(1./epsl,gxl,2*N);

        Amn=toeplitz(amn(2*N+1:4*N+1),amn(2*N+1:-1:1));
        Bmn=toeplitz(bmn(2*N+1:4*N+1),bmn(2*N+1:-1:1));

        Q = Bmn\(-II+Kx*(Amn\Kx));
        [V,Lam] = eig(Q);
        Lam = -sqrt(diag(Lam));

        W = -Bmn*V*diag(Lam);
        X = diag(exp(k0*dl(j)*Lam));
        H1 = [W, W*X;V*X, -V]/[W*X, W;V, -V*X];
        H.h11 = H1(1:nn,1:nn);
        H.h12 = H1(1:nn,nn+(1:nn));
        H.h21 = H1(nn+(1:nn),1:nn);
        H.h22 = H1(nn+(1:nn),nn+(1:nn));
        Hg = Star(Hg,H,II);
  end

  % final boundary condition solver
  H0y = -1i*n1;
  res = zeros(2*nn,1);
  res(N+1) = E0(1);
  res(3*N+2) = H0y;
  C2 =1i* diag(kxn.^2./kzn1+kzn1);
```

```
    C3 = -1i*diag(kxn.^2./kzn3+kzn3);
    F1 = [II, -Hg.h12; Z, -Hg.h22];
    F2 = [Hg.h12*C2-II, Hg.h11;  Hg.h22*C2, Hg.h21-C3];
    RT = F2\F1*res;

    % extract field components
    Rx = RT(1:nn);
    Rz = -(kxn'.*Rx)./kzn1';

    Tx = RT(nn+(1:nn));
    Tz = -(kxn'.*Tx)./kzn3';

    R = Rx.*conj(Rx)+Rz.*conj(Rz);
    T = Tx.*conj(Tx)+Tz.*conj(Tz);

    R = real(kzn1/(k1z).*R');
    T = real(kzn3/(k1z).*T');

    Ref(kk,:) = R((-1:1)+N+1);
    Trn(kk,:) = T((-1:1)+N+1);
    kk = kk+1;

    figure(1);
    plot(loop,Ref,loop,Trn);grid;axis([loop(1),loop(end),0,1]);
    drawnow;

end
```

Code for TE Mode

```
% TE mode CWA code
%       Copyright (c) 2025 David Fluckiger
%       Author: David Fluckiger
%       Date: February 2020, revision: 1.0
%
% Permission is hereby granted, free of charge, to any person obtaining a copy of this software
% and associated documentation files (the "Software"), to deal in the Software without restriction,
% including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense,
% and/or sell copies of the Software, and to permit persons to whom the Software is furnished to
% do so, subject to the following conditions:
%
% The above copyright notice and this permission notice shall be included in all copies or
%substantial portions of the Software.

%% ==========Global Grating Problem Definition===============
%clear all;
%close all;
%H = [];  dl = [];  eps = [];  xl=[];

    N = 15;
    Period = .5;
    lambda = .6328;
    theta = 30;

    n1 = 1.;
    n3 =  3.42965 + 0.18833i;

    % device, define from top down
    xl = cell(1);
    eps = cell(1);

    %dl = zeros(2,1);
    xl{1} = [0., .5,1.];
    eps{1} = [n3,n1].^2;
    dl(1) = 0.5;

    %xl{2} = [0., .4,1.];
    %eps{2} = [1.5,n1].^2;
    %dl(2) = 0.5;

    loop = linspace(0,88,9);                % Loop through theta (degrees)
    Ref = zeros(length(loop), 3);
    Trn = zeros(length(loop), 3);
    nn = 2*N+1;

    kk = 1;
```

```
for theta =  loop
      % derived quantities
      k0 = 2*pi/lambda;
      k0x = n1*sind(theta);                    % all k's are normalized to k0 = 2*pi/lambda
      k0z = n1*cosd(theta);
      kx1 = k0x - (-N:N)*lambda/Period;
      kz1 = conj(sqrt(n1^2-kx1.^2));
      kz3 = conj(sqrt(n3^2-kx1.^2));

      II = eye(2*N+1);
      Z = zeros(2*N+1);
      Kx2 = diag(kx1.^2);
    Hg.h11 = II;
    Hg.h12 = Z;
    Hg.h21 = Z;
    Hg.h22 = II;

      for j = 1:length(xl)
                gxl = xl{j};
                %gxl(2) = dx;
                epsl = eps{j};
        amn= FourierCoef(epsl,gxl,2*N);
        Amn=toeplitz(amn(2*N+1:4*N+1),amn(2*N+1:-1:1));
        Q = -Amn + Kx2;
        [W,Lam] = eig(Q);
        Lam = -(sqrt(diag(Lam)));
        V = -W*diag(Lam);
        X = diag(exp(k0*dl(j)*Lam));
        H1 = [W, W*X;V*X, -V]/[W*X, W;V, -V*X];
        H.h11 = H1(1:nn,1:nn);
        H.h12 = H1(1:nn,nn+(1:nn));
        H.h21 = H1(nn+(1:nn),1:nn);
        H.h22 = H1(nn+(1:nn),nn+(1:nn));
        Hg = Star(Hg,H,II);
      end

      res = zeros(2*nn,1);
      res(1:nn) = -Hg.h12(:,N+1)*(1i*k0z);
      res(N+1) = res(N+1)+1;
      res(nn+(1:nn)) = -Hg.h22(:,N+1)*1i*k0z;
      Kz1 = diag(1i*kz1);
      Kz3 = diag(1i*kz3);
      F = [-II-Hg.h12*Kz1, Hg.h11; -Hg.h22*Kz1, Hg.h21-Kz3];
      RT = F\res;

      R = RT(1:nn);
      T = RT(nn+(1:nn));
```

```
    R = real(kz1/k0z).*abs(R').^2;
    T = real(kz3/k0z).*abs(T').^2;

    Ref(kk,:) = R((-1:1)+N+1);
    Trn(kk,:) = T((-1:1)+N+1);

    kk = kk+1;

    plot(loop,Ref,loop,Trn);grid
    drawnow;

end
```