ELEC 4700 Midterm Report

Rish Jain, 101162547

June 29, 2025

Contents

1 Introduction and Background

Over the past century, extensive experimental and theoretical research has been conducted to understand the behavior of light, a form of electromagnetic radiation. Light propagation can be described by Maxwell's equations, which provide a fundamental framework for understanding it's behavior. Various models have been developed to simulate light interaction with different media, aiding in advancements in fields such as photonics, optical communication, and laser technology.
This project uses a simple Travelling Wave Model (TWM) to simulate light emission within a waveguide under various conditions. The model accounts for complex sinusoidal modulation, boundary reflections, gain and loss mechanisms, detuning effects, and gratings that act as high-pass or low-pass filters. Additionally, it incorporates dispersion effects to analyze wave evolution over time. By simulating these physical phenomena, the project aims to improve our understanding of light behavior in waveguides, which has implications for developing more efficient optical devices and communication systems.

2 Basic Propagation (Milestone 1)

Milestone 1 deals with using the simple Travelling Wave Model equation and re-writing it in a code-able form and plotting it to see a Gaussian pulse propagating along a waveguide.

The simplest TLM model for propagation in a waveguide is given by these two equations:

$$\frac{1}{v_g}\frac{\partial \hat{E}_f}{\partial t} = -\frac{\partial \hat{E}_f}{\partial z} \tag{1}$$

$$\frac{1}{v_g}\frac{\partial \hat{E}_r}{\partial t} = +\frac{\partial \hat{E}_r}{\partial z} \tag{2}$$

To have a codeable form, we need to re-write the equation using the **Upwind Finite Difference Method** with the synchronization condition:

$$\Delta z = v_g \Delta t. \tag{3}$$

Derivation

Consider a generic function $f(z)$. Its derivative can be found using the **definition of the derivative**:

$$\frac{df}{dz} = \lim_{\Delta z \to 0} \frac{f(z + \Delta z) - f(z)}{\Delta z}. \tag{4}$$

Here, Δz is an infinitesimally small value, which is suitable for theoretical analysis but impractical for computational methods since computers cannot handle infinitesimal steps. In numerical computations, we approximate derivatives using **finite differences**, which rely on function values at discrete points rather than a continuous domain. Therefore, to make the problem computationally feasible, we need to **discretize** the spatial domain in the z-direction.

Spatial Discretization

To apply finite difference methods, we start by discretizing the z-spatial domain:

- Let the total length in the z-direction be finite.
- Divide the domain into N segments to obtain a uniform grid spacing:

$$\Delta z = \frac{z_{\max} - z_{\min}}{N}. \tag{5}$$

- Define each grid point as:

$$z_i = i\Delta z, \quad i = 0, 1, 2, \ldots, N. \tag{6}$$

This discretization transforms the continuous variable z into a series of discrete points, making it suitable for numerical computations.

Finite Difference Approximation

Starting with the definition of the derivative and substituting the discrete grid points:

$$\frac{df}{dz} = \lim_{\Delta z \to 0} \frac{f(z_i + \Delta z) - f(z_i)}{\Delta z}. \tag{7}$$

However, taking the limit is not practical for numerical methods. Instead, we approximate the derivative by using the finite Δz:

$$\left.\frac{df}{dz}\right|_{z_i} \approx \frac{f(z_i + \Delta z) - f(z_i)}{\Delta z}. \tag{8}$$

Since $z_i + \Delta z = z_{i+1}$, this can be rewritten as:

$$\left.\frac{df}{dz}\right|_{z_i} \approx \frac{f(z_{i+1}) - f(z_i)}{\Delta z}. \tag{9}$$

This expression is known as the **forward difference approximation**.

Similarly, we can also formulate the derivative this way

$$\left.\frac{df}{dz}\right|_{z_i} \approx \frac{f(z_i) - f(z_{i-1})}{\Delta z}. \tag{10}$$

This expression is known as the **backward difference approximation**.

Equations (4) through (10) establish the theoretical foundation for computing the derivative of a generic function. Building on these concepts, we will now apply them to solve an **advection** equation, specifically the TWM equations given by Equations (1) and (2).
We will skip the "definition of derivative" form of E_f and E_r (similar to equation (4)) and proceed directly to their finite difference forms, similar to equation (8). Before doing so, we need to discretize both variables, z and t. This is done by defining each grid point as follows:

$$t_j = j\Delta t, \quad j = 0, 1, 2, \ldots, M, \tag{11}$$

$$z_i = i\Delta z, \quad i = 0, 1, 2, \ldots, N. \tag{12}$$

From mathematical principles, when computing the derivative of a two-variable function, one variable must be held constant while differentiating with respect to the other. Since E_f depends on both z and t, its derivative involves holding either z or t constant, depending on the context.

For wave equations, it is crucial to define the direction of the positive axis. In this analysis, it is assumed that the positive z-direction is defined as the direction in which a wave propagates from left to right. Consequently, the sign in front of the spatial derivative $\frac{\partial \hat{E}_f}{\partial z}$ indicates the direction of wave propagation.

- A "+" sign indicates that the Wave moves left (negative z-direction).
- A "−" sign indicates that the Wave moves right (positive z-direction).

Based on this, we can derive finite difference equations for Equations (1) and (2). However, before doing so, it is important to understand what the upwind finite difference method is, how it allows us to numerically compute the TWM equations, and how it ensures stability by using information from the direction the wave is coming from. For a rightward-moving wave, a backward difference in space is used to incorporate upstream information from the left, whereas for a leftward-moving wave, a forward difference in space is applied to capture upstream information from the right. Future states depend on known past states, ensuring both causality and stability. Causality is maintained by preventing reliance on unknown future values, while stability is achieved by using previously computed time step values, allowing for an accurate progression of the wave solution.

Finite Differences Derivation for Equation (1)

- **Time derivative (forward difference):**

$$\left.\frac{\partial \hat{E}_f}{\partial t}\right|_{(z_i,t_j)} \approx \frac{\hat{E}_f(z_i,t_j) - \hat{E}_f(z_i,t_{j-1})}{\Delta t} \tag{13}$$

- **Space derivative (upwind backward difference)** *(wave propagates in the $+z$ direction)*:

$$\left.\frac{\partial \hat{E}_f}{\partial z}\right|_{(z_i,t_j)} \approx \frac{\hat{E}_f(z_i,t_{j-1}) - \hat{E}_f(z_{i-1},t_{j-1})}{\Delta z} \tag{14}$$

Substituting these finite difference approximations back into the original PDE (equation (1)):

$$\frac{1}{v_g} \cdot \frac{\hat{E}_f(z_i,t_j) - \hat{E}_f(z_i,t_{j-1})}{\Delta t} = -\frac{\hat{E}_f(z_i,t_{j-1}) - \hat{E}_f(z_{i-1},t_{j-1})}{\Delta z} \tag{15}$$

Substituting the synchronization condition from equation (3) into the discretized PDE gives:

$$\frac{1}{v_g} \cdot \frac{\hat{E}_f(z_i,t_j) - \hat{E}_f(z_i,t_{j-1})}{\Delta t} = -\frac{\hat{E}_f(z_i,t_{j-1}) - \hat{E}_f(z_{i-1},t_{j-1})}{v_g \Delta t}. \tag{16}$$

Multiplying both sides by $v_g \Delta t$ simplifies the equation to:

$$\hat{E}_f(z_i, t_j) - \hat{E}_f(z_i, t_{j-1}) = -\hat{E}_f(z_i, t_{j-1}) + \hat{E}_f(z_{i-1}, t_{j-1}). \tag{17}$$

Rearranging to solve for $\hat{E}_f(z_i, t_j)$:

$$\hat{E}_f(z_i, t_j) = \hat{E}_f(z_{i-1}, t_{j-1}). \tag{18}$$

Finite Differences Derivation for Equation (2)

- **Time derivative (forward difference):**

$$\left.\frac{\partial \hat{E}_r}{\partial t}\right|_{(z_i, t_j)} \approx \frac{\hat{E}_r(z_i, t_j) - \hat{E}_r(z_i, t_{j-1})}{\Delta t} \tag{19}$$

  The time derivative for Equation (2), as given by Equation (19), is the same as the one derived for Equation (1), as shown in Equation (13). This is because time advances in only one direction, regardless of the direction of wave propagation. Therefore, both Equation (1) and Equation (2) share the same finite difference approximation for the time derivative.

- **Space derivative (upwind forward difference)** *(wave propagates in the $-z$ direction)*:

$$\left.\frac{\partial \hat{E}_r}{\partial z}\right|_{(z_i, t_j)} \approx \frac{\hat{E}_r(z_{i+1}, t_{j-1}) - \hat{E}_r(z_i, t_{j-1})}{\Delta z} \tag{20}$$

Substituting these finite difference approximations into the original PDE:

$$\frac{1}{v_g} \cdot \frac{\hat{E}_r(z_i, t_j) - \hat{E}_r(z_i, t_{j-1})}{\Delta t} = \frac{\hat{E}_r(z_{i+1}, t_{j-1}) - \hat{E}_r(z_i, t_{j-1})}{\Delta z} \tag{21}$$

Applying the synchronization condition $v_g \Delta t = \Delta z$:

$$\frac{1}{v_g} \cdot \frac{\hat{E}_r(z_i, t_j) - \hat{E}_r(z_i, t_{j-1})}{\Delta t} = \frac{\hat{E}_r(z_{i+1}, t_{j-1}) - \hat{E}_r(z_i, t_{j-1})}{v_g \Delta t} \tag{22}$$

Multiplying both sides by $v_g \Delta t$:

$$\hat{E}_r(z_i, t_j) - \hat{E}_r(z_i, t_{j-1}) = \hat{E}_r(z_{i+1}, t_{j-1}) - \hat{E}_r(z_i, t_{j-1}) \tag{23}$$

Rearranging to solve for $\hat{E}_r(z_i, t_j)$:

$$\hat{E}_r(z_i, t_j) = \hat{E}_r(z_{i+1}, t_{j-1}) \tag{24}$$

These results (Equations (18) and (24)) show that under the synchronization condition, the field at position z_i and time t_j is determined by the field at the previous position and time. Equation (18), which corresponds to a wave propagating in the $+z$-direction, shows that the field at (z_i, t_j) depends on the field at (z_{i-1}, t_{j-1}), reflecting the physical intuition that information travels from left to right. In contrast, Equation (24), which represents wave propagation in the $-z$-direction, indicates that the field at (z_i, t_j) is determined by the field at (z_{i+1}, t_{j-1}), capturing the concept that information travels from right to left. In both cases, future wave states are computed step-by-step from known states. These fully discretized equations provide a stable framework for numerically solving the travelling wave model and ensure that the direction of wave propagation is accurately represented through the appropriate upwind finite difference method.

Code

The full code corresponding to Milestone 1 can be found in Appendix under Section 7.2.

Now that the discretized, code-able TWM equations have been derived, they are implemented in MATLAB to verify that the upwind finite difference method is not only correct but also stable. The code used to represent Equations (18) and (24) come from lines 106 and 107 and is placed inside the main loop to enable continuous field updates required for smooth wave propagation. The code is:

```
Ef(2:Nz) = fsync * Ef(1:Nz-1);
Er(1:Nz-1) = fsync * Er(2:Nz);
```

The code above accurately represents Equation (18) because the present values of E_f are computed using past values, ensuring that information flows in the correct direction and no future data is used to infer the present. This preserves the physical principle of causality, where the current state depends only on prior states. The range for the present values (left-hand side of Equation (18)) starts at 2 because the forward propagation calculation requires information from the previous spatial point. Since MATLAB indexing starts at 1, starting the present value range at the first element would require referencing a non-existent zero index for the past values, which would

result in an indexing error. The range for the present values ends at Nz because it corresponds to the last discrete spatial point along the z-axis. Including Nz in the range for past values would attempt to access an out-of-bounds index beyond the waveguide's physical domain. For example, if the past values range ended at Nz, the present values of E_f on the left-hand side of the equation would extend to $Nz+1$, which lies outside the physical boundary of the problem and would cause an indexing error in the code. Capping the past range at $Nz-1$ ensures that all accessed indices are valid while maintaining consistency with the physical boundary. Together, these index ranges ensure that the updated field correctly reflects the wave's propagation and aligns with the discretized form given by Equation (18).
The same logic applies to the backward-propagating field E_r, where the range $1 : Nz-1$ preserves causality by ensuring that present field values depend on past values at spatial points located upstream along the propagation path.

Additionally, the `fsync` term is the stability factor that ensures the numerical method remains stable and allows the Gaussian pulse to propagate without distortion.

At the boundaries of the waveguide, boundary conditions can be added to act like mirrors. The code below from lines 102 and 103 implements the boundary conditions at both ends of the waveguide. At the left boundary ($z = 0$), the forward-propagating field is set as the sum of the input field from the left and the reflected backward-propagating field, scaled by the left reflection coefficient. In the code, `InputL(i)` represents the input wave entering from the left at time step i, while `RL*Er(1)` accounts for the portion of the backward field reflected back into the waveguide.

At the right boundary ($z = L$), the backward-propagating field is similarly updated. It is computed as the sum of the right input field and the forward-propagating field reflected at the boundary, scaled by the right reflection coefficient. Here, `InputR(i)` denotes the input wave from the right side, and `RR*Ef(Nz)` represents the reflection of the forward field at the right boundary. In short, these boundary conditions model mirror like reflections at both ends of the waveguide.

```
Ef(1) = InputL(i) + RL*Er(1);
Er(Nz) = InputR(i) + RR*Ef(Nz);
```

The code snippet below lines 110 and 111 calculates the output fields at the waveguide boundaries. At the right boundary ($z = L$), `OutputR(i)` represents the transmitted portion of the forward-propagating field after accounting for reflections. The factor `(1 - RR)` ensures that only the part of the wave not reflected at the boundary is considered as output.

Similarly, at the left boundary ($z = 0$), `OutputL(i)` calculates output of the backward-propagating field. The factor `(1 - RL)` removes the reflected portion, ensuring that only the transmitted part of the backward wave is included in the output.
These factors act as transmission coefficients, ensuring that the reflected portion of the field is subtracted from the total field at the boundary, allowing the code to capture the correct transmitted outputs.

```
OutputR(i) = Ef(Nz)*(1-RR);
OutputL(i) = Er(1)*(1-RL);
```

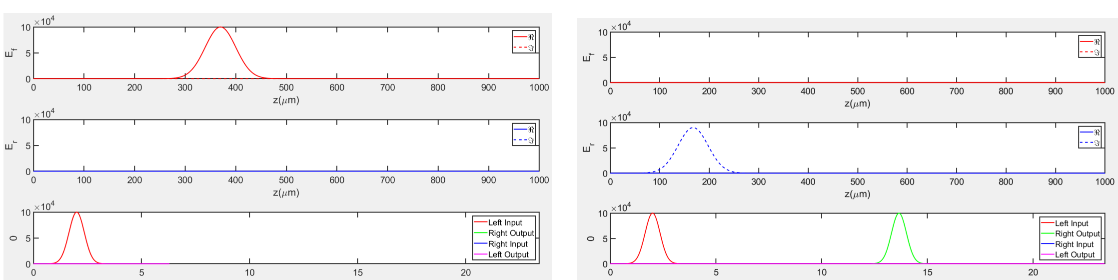
The code snippet below, found in lines 94 and 95, defines the reflection coefficients at the left and right boundaries. Their values are set to $0.9i$, indicating that they are complex reflection coefficients.

```
RL = 0.9i;
RR = 0.9i;
```

Simulation Results

The code from Section 2.7.2 was simulated for left and right inputs, including a Gaussian and a modulated Gaussian, and the resulting plots are presented below:

Normal Gaussian Pulse Input from the Left and Reflection at the Right Side



(a) Forward-propagating unmodulated Gaussian pulse from the left input.

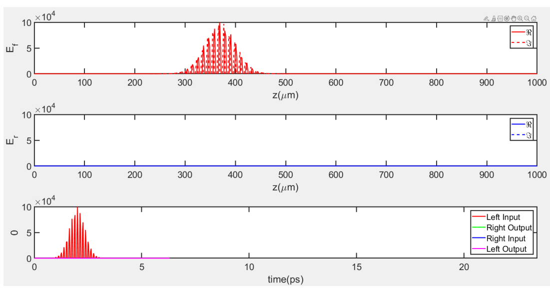
(b) Backward-propagating unmodulated Gaussian pulse reflected from the right boundary.

Figure 1: An un-modulated Gaussian pulse inputted from the left input and reflected at the right boundary of the waveguide, with E_{field} units as $V/\mu m$

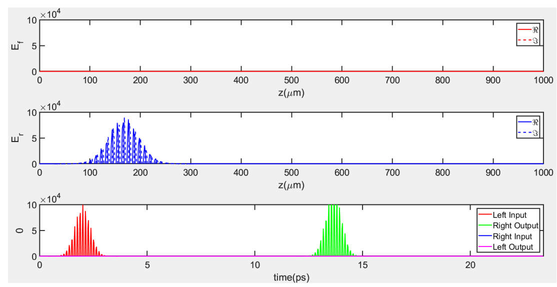
Modulated Gaussian Pulse Input from the Left and Reflection at the Right Side

To generate a sinusoidally modulated Gaussian pulse, the code was slightly modified by updating the frequency of the sinusoidal modulation parameter from 0 Hz to 50 THz, as shown below:

```
InputParasL.we = 5e13;  % Frequency of complex modulation on Gaussian pulse
```



(a) Sinusoidally modulated Gaussian pulse propagating forward from the left input.



(b) Sinusoidally modulated Gaussian pulse propagating backward after reflection at the right boundary.

Figure 2: A 50 THz sinusoidally modulated Gaussian pulse inputted from the left input and reflected at the right boundary of the waveguide, with E_{field} units as $V/\mu m$

Normal Gaussian Pulse Input from the Right and Reflection at the Left Side

To generate an input from the right side, `InputParasR` was activated with minor code modifications. `InputParasL` was disabled to clearly observe the wave propagating from the right input and reflecting off the left boundary. The code changes are as follows:

```
InputParasR = 0;
```

Updated to (1st change):

```
InputParasR.E0 = 1e5;          % Amplitude of the right-side input E-field / E_r
InputParasR.we = 5e13;         % Frequency of modulation on Gaussian pulse
InputParasR.t0 = 2e-12;        % Time shift for the right-side input
InputParasR.wg = 5e-13;        % Width of Gaussian distribution for right input
InputParasR.phi = 0;           % Initial phase of the right input field
```

Then,

```
InputR(i) = ErN(t, 0); % No input from the right previously
```

Updated to (2nd change):

```
InputR(i) = ErN(t, InputParasR); % Input signal from right side at time step i
```

Then, `InputParasL`:

```
InputParasL.E0 = 1e5;           % Amplitude of the input E-field / E_f
InputParasL.we = 0;             % Frequency of modulation on Gaussian pulse
InputParasL.t0 = 2e-12;         % Time shift for the left-side input
InputParasL.wg = 5e-13;         % Width of the Gaussian distribution
InputParasL.phi = 0;            % Initial phase of the left input field
```

Updated to (3rd change):

```
InputParasL = 0;
```

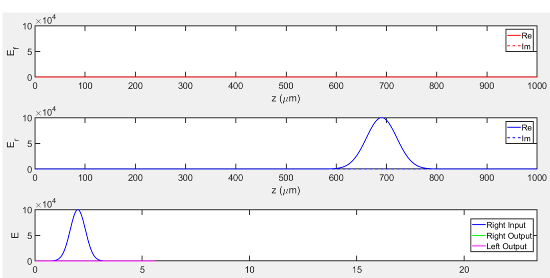
Then:

```
InputL(i) = Ef1(t, InputParasL);
```

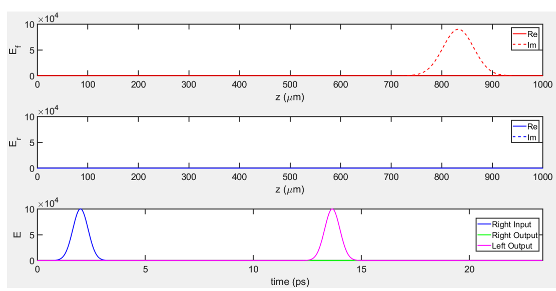
Updated to (4th Change):

```
InputL(i) = Ef1(t, 0);
```

After these modifications, simulating the code produced the following plots:



(a) Forward-propagating Gaussian pulse from the right input.



(b) Backward-propagating Gaussian pulse reflected from the left boundary.

Figure 3: An un-modulated Gaussian pulse inputted from the right input and reflected at the left boundary of the waveguide, with E_{field} units as $V/\mu m$

Although it is not clear from Figure 2, we can see from Figures 1 and 3 that for a purely real input from either the right or left side, when reflected at the left or right boundaries, the magnitude of the wave decreases and the reflected waves become purely imaginary. This is consistent with the reflection coefficients R_R and R_L mentioned above, which are set to $0.9i$. A purely real wave reflecting at either boundary will have its amplitude scaled by 0.9 and become purely imaginary due to the i term.

A sinusoidally modulated plot for the right input was not performed, as it would be identical to Figure 2 but mirrored.

In conclusion, Milestone 1 involved deriving the code-able form of the Travelling Wave Equation, implementing the derived equations in MATLAB, and obtaining simulation results in the form of plots. The derivation and code were found to be correct since the simulation results behaved as intended and matched the expected outcomes.

3 Static Gain, Detuning, and Frequency Domain (Milestone 2)

Milestone 2 deals with using the simple Travelling Wave Model (TWM) and introducing static gain and detuning to the pulse, then examining the changes to the pulse in the spectral domain. The modified equations that model the static gain and detuning are given below:

$$\frac{1}{v_g}\frac{\partial \hat{E}_f}{\partial t} = -\frac{\partial \hat{E}_f}{\partial z} - i\hat{\beta}\hat{E}_f \tag{25}$$

$$\frac{1}{v_g}\frac{\partial \hat{E}_r}{\partial t} = +\frac{\partial \hat{E}_r}{\partial z} - i\hat{\beta}\hat{E}_r \tag{26}$$

The exact physical derivation for Equations (25) and (26) starts in the same way as the derivation of Equations (1) and (2) but deviates at the point where Equation (27), given below, is not neglected.

$$\mu\sigma\frac{\partial \hat{E}(z,t)}{\partial t} \tag{27}$$

To be specific, it deviates where the material property parameter σ in the equation above is not assumed or set to zero. Milestone 1 dealt with propagation in a lossless medium, whereas Milestone 2 requires the physical material parameters not to be neglected because they control whether the pulse experiences gain or loss in the waveguide.

The derivation continues by assuming a harmonic time dependence of the form $\hat{E}(z,t) = \hat{E}(z)e^{-i\omega t}$ and defining the complex propagation constant $\hat{\beta} = \beta_r + i\beta_i$. This transforms the wave equation into a second-order differential equation in space. Solving this PDE leads directly to the gain/loss-modified TWM Equations (25) and (26), where the imaginary part of $\hat{\beta}$ introduces exponential growth (gain) or decay (loss) into the traveling waves and the real part of $\hat{\beta}$ introduces de-tuning of the wave.

To be able to see this physical phenomena, we need to transform Equations (25) and (26) into something that is code-able i.e. discretize the equations. This process will involves starting with the Forward Wave Equation (25) , then rearranging it to isolate the spatial derivative:

$$\frac{\partial \hat{E}_f}{\partial z} = -\frac{1}{v_g}\frac{\partial \hat{E}_f}{\partial t} - i\hat{\beta}\hat{E}_f \tag{28}$$

Assuming a quasi-static solution where the time variation is negligible over the step Δz, the time derivative term can be dropped:

$$\frac{\partial \hat{E}_f}{\partial z} = -i\hat{\beta}\hat{E}_f \tag{29}$$

This equation shows how the forward-traveling wave evolves spatially, with the term $-i\hat{\beta}\hat{E}_f$ accounting for both phase shift (from the real part of $\hat{\beta}$) and gain or loss (from the imaginary part of $\hat{\beta}$).

To determine the wave behavior over a small spatial step Δz, both sides are integrated from z_{i-1} to z_i:

$$\int_{E_f(z_{i-1})}^{E_f(z_i)} \frac{1}{\hat{E}_f} d\hat{E}_f = -i\int_{z_{i-1}}^{z_i} \hat{\beta}(z)dz \tag{30}$$

Using the integral identity:

$$\int \frac{1}{x}dx = \ln|x| + C$$

The left-hand side simplifies to a natural logarithm:

$$\ln\left(\frac{\hat{E}_f(z_i)}{\hat{E}_f(z_{i-1})}\right) = -i\int_{z_{i-1}}^{z_i} \hat{\beta}(z)dz \tag{31}$$

Assuming that $\hat{\beta}$ is constant over the interval Δz, the integral on the right-hand side simplifies using the property:

$$\int_a^b c\, dx = c(b-a) \quad \text{for a constant } c$$

resulting in:

$$\ln\left(\frac{\hat{E}_f(z_i)}{\hat{E}_f(z_{i-1})}\right) = -i\hat{\beta}(z_i)\Delta z \tag{32}$$

Exponentiating both sides with the identity:

$$e^{\ln(x)} = x$$

gives:

$$\frac{\hat{E}_f(z_i)}{\hat{E}_f(z_{i-1})} = e^{-i\hat{\beta}(z_i)\Delta z} \tag{33}$$

Thus, the forward wave after one spatial step is:

$$\hat{E}_f(z_i) = \hat{E}_f(z_{i-1})e^{-i\hat{\beta}(z_i)\Delta z} \tag{34}$$

In time-domain marching methods, the wave at position z_i and time t_j is computed from the previous time step t_{j-1} as:

$$\hat{E}_f(t_j, z_i) = \hat{E}_f(t_{j-1}, z_{i-1})e^{-i\hat{\beta}(z_i)\Delta z} \tag{35}$$

For the reverse-traveling wave, the corresponding equation is:

$$\hat{E}_r(t_j, z_i) = \hat{E}_r(t_{j-1}, z_{i+1})e^{-i\hat{\beta}(z_i)\Delta z} \tag{36}$$

The exponential term in these equations introduces both phase rotation (from the real part of $\hat{\beta}$) and amplitude changes (from the imaginary part of $\hat{\beta}$). If $\beta_i < 0$, the wave amplitude increases, indicating gain. Conversely, if $\beta_i > 0$, the amplitude decreases, indicating loss. Incorporating a complex propagation constant into the TWM equations results in an exponential factor that models the gain or loss experienced by the traveling waves during propagation.

Code

The full code corresponding to Milestone 2 can be found in Appendix under 7.5 Section E.

Now that the discretized, code-able modified TWM equations have been derived, they are implemented in MATLAB to verify that the derivations for static gain and denutuning is correctly implemented. The code used to represent Equations (35) and (36) and its validity can be found at lines 17, 18, 99, 100, 111, 112, 119, 120 and 161-174.

The various pieces of code are as follows:

```
beta_r = 0;
beta_i = 0;
```

This code initializes the real and imaginary parts of the complex propagation constant β. Currently, both are set to zero, but they can be modified as needed. Different values of β_r and β_i will be explored in the simulation section for Milestone 2.

```
beta = ones(size(z)) * (beta_r + 1i * beta_i);
exp_det = exp(-1i * dz * beta);
```

This code initializes an array with the same size as the waveguide for the complex propagation constant β. Each element of this array is assigned the value $\beta_r + i\beta_i$, where β_r and β_i are defined earlier. Since β is treated as a constant along the waveguide, it is initialized outside the main loop to improve efficiency.
The variable `exp_det` represents the exponential term that appears in Equations (35) and (36), effectively capturing the phase and amplitude variations introduced by β. Like `beta`, `exp_det` is also an array of the same length as the waveguide.

```
Ef(2:Nz) = fsync * exp_det(1:Nz-1) .* Ef(1:Nz-1);
Er(1:Nz-1) = fsync * exp_det(2:Nz) .* Er(2:Nz);
```

This code is similar to the one used in Milestone 1 but includes the additional `exp_det` term, which accounts for the exponential behavior described in Equations (35) and (36). This term effectively introduces gain/loss and/or de-tuning to the propagating wave, depending on the value of β_i and β_r respectively.
It is also important to note that the indexing of `exp_det` matches the indexing of the `Ef` on the right-hand side of Equation (35) . This indexing matching maintains numerical stability and follows the same principles discussed in Milestone 1 for handling wave propagation in discrete steps.

```
fftOutput = fftshift(fft(OutputR));    % Get FFT data for OutputR
omega = fftshift(wspace(time));
```

This code computes the Fourier Transform of the output signal `OutputR` to obtain its frequency-domain representation. The variable `omega` corresponds to the frequency axis.

```
subplot(3,2,2);
plot(omega, abs(fftOutput));
xlabel('Frequency (THz)');
ylabel('|E|');
xlim([-1.5e14, 1.5e14]);

subplot(3,2,4);
phase = unwrap(angle(fftOutput));  % Unwrap the phase
plot(omega, phase);
xlabel('Frequency (THz)');
ylabel('Phase (E)');
xlim([-1.5e14, 1.5e14]);
```

This code plots the magnitude and phase of the Gaussian-modulated pulse in the frequency domain, illustrating the effects of gain, loss, and de-tuning on its spectrum.

Simulation Results

Time domain, magnitude, and phase plots for $B_r = B_i = 0$

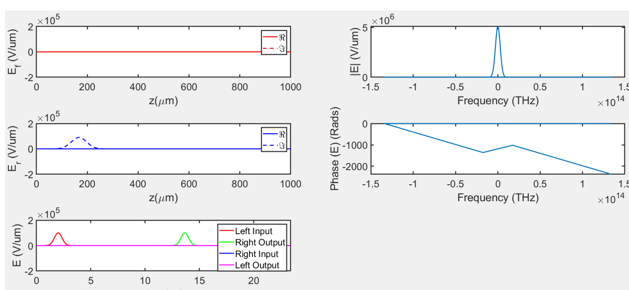


Figure 4: Magnitude and phase of a Normal Gaussian pulse with no gain or de-tuning

Figure 4 represents the simulation of a normal Gaussian pulse propagating through the waveguide with no modulation, gain or de-tuning. The field detection plot the the bottom left confirms that there is no gain present whereas the magnitude vs frequency plot confirms that there is no modulation present.

Time domain, magnitude, and phase plots for $B_r = 0$, $B_i = 8$ with a simple Gaussian

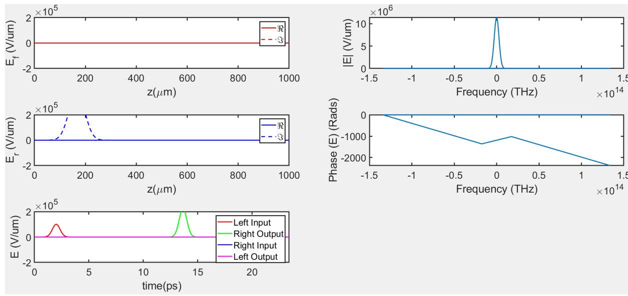


Figure 5: Magnitude and phase of a normal Gaussian pulse with a gain but no de-tuning

Figure 5 represents the simulation of a normal Gaussian pulse propagating through the waveguide with some gain but no modulation or de-tuning. The field detection plot the the bottom left confirms that there is some gain present whereas the magnitude vs frequency plot confirms that there is no modulation present.

Time domain, magnitude, and phase plots for $B_r = 0$, $B_i = 8$ with a modulated Gaussian

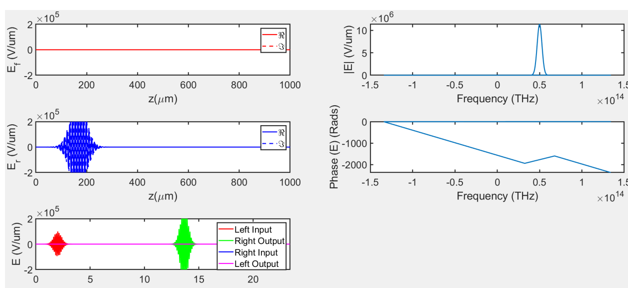


Figure 6: Magnitude and phase of a modulated Gaussian pulse with some gain but no de-tuning

Figure 6 represents the simulation of a modulated Gaussian pulse propagating through the waveguide with some gain but no de-tuning. The field detection plot the the bottom left confirms that there is some gain present whereas the magnitude vs frequency plot also confirms that there modulation present since the magnitude curve is shifted to the right with the same amount as the frequency of the modulation, which in this case is 50 THz.

Time domain, magnitude, and phase plots for $B_r = 80$, $B_i = 0$

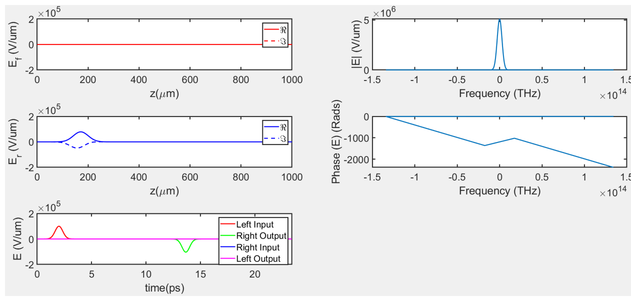


Figure 7: Magnitude and phase of a normal Gaussian pulse with no gain but phase shift due to de-tuning

Figure 7 represents the simulation of a normal Gaussian pulse propagating through the waveguide with no gain and no modulation but some phase shift due to de-tuning. The field detection plot the the bottom left confirms that there is no gain present whereas the magnitude vs frequency plot also confirms that there is no modulation present.

Time domain, magnitude, and phase plots for $B_r = 80$, $B_i = 8$

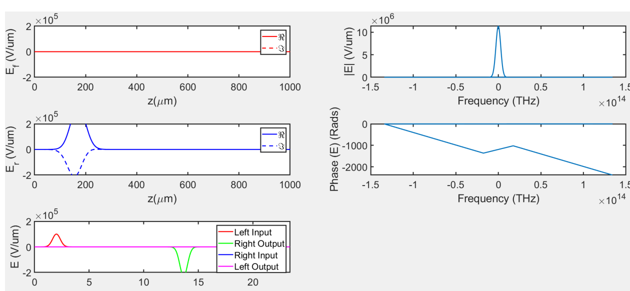


Figure 8: Magnitude and phase of a normal Gaussian pulse some gain and phase shift due to de-tuning

Figure 8 represents the simulation of a normal Gaussian pulse propagating through the waveguide with some gain and some phase shift due to de-tuning, but no modulation. The field detection plot the the bottom left confirms that there is some gain present whereas the magnitude vs frequency plot confirms that there is no modulation present.

In conclusion, Milestone 2 involved deriving a numerically implementable form of the travelling wave equation with static gain and de-tuning to the pulse, then examining the changes to the pulse in the spectral domain, and implementing the derived equations in MATLAB, and finally, obtaining simulation results in the form of plots. The derivation and code were verified to be correct, as the simulation results matched the expected outcomes.

4 Gratings and the use of $\hat{\kappa}$ (Milestone 3)

Milestone 3 builds upon the simple Travelling Wave Model (TWM) by introducing gratings into the waveguide, which reflect the wave upon interaction and cause back reflection. Additionally, we examine how this reflection affects the propagation of a Gaussian pulse in both the time and frequency domains. The modified equations that model the grating effects are given below:

$$\frac{1}{v_g}\frac{\partial \hat{E}_f}{\partial t} = -\frac{\partial \hat{E}_f}{\partial z} + i\kappa_f \hat{E}_r \tag{37}$$

$$\frac{1}{v_g}\frac{\partial \hat{E}_r}{\partial t} = +\frac{\partial \hat{E}_r}{\partial z} + i\kappa_r \hat{E}_f \tag{38}$$

To observe this physical phenomenon, we need to transform the above equations into a form suitable for numerical implementation, i.e., discretize them. The discretization process follows the same methodology used in the derivations for Equations (1) and (2). Hence, derived equations from Milestone 1, such as the time derivatives, Equation (13) which corresponds to $\left.\frac{\partial \hat{E}_f}{\partial t}\right|_{(z_i,t_j)}$ and Equation (19) which corresponds to $\left.\frac{\partial \hat{E}_r}{\partial t}\right|_{(z_i,t_j)}$, can be reused. Similarly, the spatial derivatives, Equation (14) representing $\left.\frac{\partial \hat{E}_f}{\partial z}\right|_{(z_i,t_j)}$ and Equation (20) representing $\left.\frac{\partial \hat{E}_r}{\partial z}\right|_{(z_i,t_j)}$, can also be reused.

By substituting the previously derived Equations (13) and (14) into Equation (37), we obtain:

$$\frac{1}{v_g}\frac{\hat{E}_f(z_i,t_j)-\hat{E}_f(z_i,t_{j-1})}{\Delta t}=-\frac{\hat{E}_f(z_i,t_{j-1})-\hat{E}_f(z_{i-1},t_{j-1})}{\Delta z}+i\kappa_f\hat{E}_r \tag{39}$$

Rearranging the synchronization condition from Equation (3) to the form $v_g = \frac{\Delta z}{\Delta t}$ and substituting it into the above equation yields

$$\frac{\Delta t}{\Delta z}\frac{\hat{E}_f(z_i,t_j)-\hat{E}_f(z_i,t_{j-1})}{\Delta t}=-\frac{\hat{E}_f(z_i,t_{j-1})-\hat{E}_f(z_{i-1},t_{j-1})}{\Delta z}+i\kappa_f\hat{E}_r \tag{40}$$

Canceling Δt from the left-hand side gives:

$$\frac{\hat{E}_f(z_i,t_j)-\hat{E}_f(z_i,t_{j-1})}{\Delta z}=-\frac{\hat{E}_f(z_i,t_{j-1})-\hat{E}_f(z_{i-1},t_{j-1})}{\Delta z}+i\kappa_f\hat{E}_r \tag{41}$$

Multiplying both sides by Δz results in:

$$\hat{E}_f(z_i,t_j)-\hat{E}_f(z_i,t_{j-1})=-\left[\hat{E}_f(z_i,t_{j-1})-\hat{E}_f(z_{i-1},t_{j-1})\right]+i\Delta z\kappa_f\hat{E}_r \tag{42}$$

Solving for $\hat{E}_f(z_i,t_j)$:

$$\hat{E}_f(z_i,t_j)=\hat{E}_f(z_i,t_{j-1})-\left[\hat{E}_f(z_i,t_{j-1})-\hat{E}_f(z_{i-1},t_{j-1})\right]+i\Delta z\kappa_f\hat{E}_r \tag{43}$$

Simplifying the terms:

$$\hat{E}_f(z_i,t_j)=\hat{E}_f(z_{i-1},t_{j-1})+i\Delta z\kappa_f\hat{E}_r \tag{44}$$

Up to this point, the discrete indices for $\hat{E}_r$ within the term $i\Delta z\kappa_f\hat{E}_r$ had not been specified. However, in Equation (45), they are defined as $\hat{E}_r(z_{i+1},t_{j-1})$. The reasoning behind this choice dates back to Milestone 1, where the selection of discrete indices for a forward or reverse-propagating field was explained. For a forward-propagating wave (moving left to right), the value at the ith point depends on the $(i-1)$th point, which lies upstream in its propagation direction. Conversely, for a reverse-propagating wave (moving right to left), the field at the ith grid point relies on the $(i+1)$th point. Thus, when forward and reverse-propagating pulses converge at the ith grid point, their interaction is determined by the upstream data from their respective directions: $i-1$ for the forward wave and $i+1$ for the reverse wave. Equation (44) specifically models how the forward coupling coefficient of the grating at the ith point affects the reverse-propagating pulse. This indexing choice ensures causality is maintained, since using future states to compute the present state lead to numerical instability. Additionally, κ_f is updated to $\hat{\kappa}_f$ to account for the fact that the grating does not extend along the entire waveguide. As a result, the coupling coefficient is not constant and varies with position. To capture this spatial dependence, it is best to represent $\hat{\kappa}_f$ as a position-dependent variable. When the discretized equations are implemented in code, $\hat{\kappa}_f$ can be treated as an array or matrix, where each element corresponds to the coupling coefficient at a specific discrete grid point z_i. This approach ensures that the grating's non-uniform profile is accurately modeled in the simulation. Hence, the final equation is written as:

$$\hat{E}_f(z_i, t_j) = \hat{E}_f(z_{i-1}, t_{j-1}) + i\Delta z \hat{\kappa}_f \hat{E}_r(z_{i+1}, t_{j-1}) \tag{45}$$

Since $\hat{E}_f$ and $\hat{E}_r$ are mathematical mirrors of each other, it is reasonable to assume that the derivation for $\hat{E}_r$ follows the same approach as that for $\hat{E}_f$. Hence, we can directly obtain the discretized equation for $\hat{E}_r$ without performing the full derivation. The equation for $\hat{E}_r$ is:

$$\hat{E}_r(z_i, t_j) = \hat{E}_r(z_{i+1}, t_{j-1}) + i\Delta z \hat{\kappa}_r \hat{E}_f(z_{i-1}, t_{j-1}) \tag{46}$$

While the equations share the same mathematical structure, the coupling coefficients $\hat{\kappa}_f$ and $\hat{\kappa}_r$ may differ depending on the physical properties of the grating. However, for our case, $\hat{\kappa}_f = \hat{\kappa}_r = \hat{\kappa}$.

Code

The full code corresponding to Milestone 3 can be found in Appendix under 7.5 Section E.

Now that the discretized, code-able modified TWM equations have been derived, they are implemented in MATLAB to verify that the derivations for position dependent grating and coupling of forward and reverse propagating waves at the grating boundaries is correctly implemented. The code used to represent Equations (45)and (46) and its validity can be found at lines 19-21, 74-76, 122, 123, 131, 132 174-199.

The various pieces of code are as follows:

```
kappa0 = 100;
kappaStart = 1/3;
kappaStop = 2/3;
```

These lines define the parameters for the coupling coefficient. `kappa0` specifies the coupling value, while `kappaStart` and `kappaStop` set the start and stop positions of the coupling region as fractions of the total waveguide length.

```
kappa = kappa0*ones(size(z));
kappa(z<L*kappaStart) = 0;
kappa(z>L*kappaStop) = 0;
```

The code above initializes an array with the same size as the waveguide for the coupling coefficient `kappa`. Each element of the array is assigned the value `kappa0`. The code then reassigns the values of `kappa` to zero outside the defined coupling region, determined by the positions `kappaStart` and `kappaStop`. This ensures that coupling occurs only within the specified section of the waveguide.

```
Ef(2:Nz) = fsync*exp_det(1:Nz-1).*Ef(1:Nz-1) + 1i*dz*kappa(2:Nz).*Er(2:Nz);
Er(1:Nz-1) = fsync*exp_det(2:Nz).*Er(2:Nz) + 1i*dz*kappa(1:Nz-1).*Ef(1:Nz-1);
```

This code extends the implementation from Milestone 2 by introducing the coupling term `kappa` as described in Equations (45) and (46). These terms model the interaction between the forward and backward propagating fields within the grating region. Additionally, it is important to note that the indexing of `kappa` must match the indexing of the forward or backward wave it affects; otherwise, the coupling will not be applied at the correct spatial points, leading to numerical inaccuracies and incorrect field evolution.

```
fftOutput2 = fftshift(fft(OutputL));
fftInput1 = fftshift(fft(InputL));
```

This code computes the Fourier Transform of the output signal `OutputL` and input signal `InputL` to obtain their frequency-domain representation.

```
subplot(3,2,2);
plot(omega, abs(fftOutputR));
hold on;
plot(omega, abs(fftOutputL));
plot(omega, abs(fftInputL));
hold off;
xlabel('Frequency (THz)');
ylabel('|E|');
xlim([-0.1e14, 0.1e14]);
legend('fftOutputR','fftOutputL','fftInputL');

subplot(3,2,4);
phase1 = unwrap(angle(fftOutputR));
phase2 = unwrap(angle(fftOutputL));
phase3 = unwrap(angle(fftInputL));
plot(omega, phase1);
hold on;
plot(omega, phase2);
plot(omega, phase3);
hold off;
xlabel('Frequency (THz)');
ylabel('Phase (E)');
xlim([-1.5e14, 1.5e14]);
legend('fftOutputR','fftOutputL','fftInputL');
```

This code plots the magnitude and phase of output signal `OutputL` and input signal `InputL` in the frequency domain, illustrating the effects of grating on Gaussian pulse.

Simulation Results

Plot of the time domain and frequency domain magnitude and phase passing through the grating

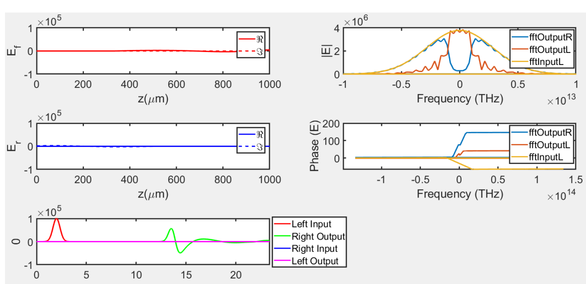


Figure 9: Mag. and phase of the pulse through a grating in time and freq. domains

Figure 9 (with E_{field} units as $V/\mu m$) shows the propagation of a Gaussian pulse through a grating region inside a waveguide. FFT is applied to the input pulse and the pulses at both the left and right outputs. Based on the magnitude versus frequency plot in Figure 9, the left output primarily contains low-frequency components, while the right output contains high-frequency components. This indicates that the grating region acts as a low-pass filter for the Gaussian pulse: high frequencies pass through the grating, whereas low frequencies are reflected at the grating boundary and propagate toward the left boundary.

In conclusion, Milestone 3 involved deriving a numerically implementable form of the travelling wave equation with a position-dependent grating, implementing the derived equations in MATLAB, and obtaining simulation results in the form of plots. The derivation and code were verified to be correct, as the simulation results behaved as intended and matched the expected outcomes.

5 Gain/Loss Dispersion (Milestone 4)

Milestone 4 builds upon the simple Travelling Wave Model (TWM) by introducing a frequency-dependent material response (gain/loss) into the waveguide. This response is modeled using the equations:

$$\frac{1}{v_g}\frac{\partial \hat{E}_f}{\partial t} = -\frac{\partial \hat{E}_f}{\partial z} - i\hat{\beta}(N,S)\hat{E}_f - k_p\left(\hat{E}_f - \hat{P}_f(z)\right) \tag{47}$$

$$\frac{1}{v_g}\frac{\partial \hat{E}_r}{\partial t} = +\frac{\partial \hat{E}_r}{\partial z} - i\hat{\beta}(N,S)\hat{E}_r - k_p\left(\hat{E}_r - \hat{P}_r(z)\right) \tag{48}$$

Where N and S control β_r and β_i and the two new terms $k_p\left(\hat{E}_f - \hat{P}_f(z)\right)$ and $k_p\left(\hat{E}_r - \hat{P}_r(z)\right)$ model the material polarization through the equations for $\hat{P}_f$ and $\hat{P}_r$. The equation for $\hat{P}_f$ and $\hat{P}_r$ are:

$$\frac{d\hat{P}_f}{dt} = i\omega_0'\hat{P}_f + \gamma(\hat{E}_f - \hat{P}_f), \tag{49}$$

$$\frac{d\hat{P}_r}{dt} = i\omega_0'\hat{P}_r + \gamma(\hat{E}_r - \hat{P}_r). \tag{50}$$

These equations represent a low-pass filter response to the exciting field, known as a Lorentzian response.

Additionally, a full derivation of the discretized forms of Equations (47) and (48) is unnecessary, as the discretized TWM equations from Milestone 2 (Equations (35) and (36)) can be reused. However, the new term $k_p\left(\hat{E}_f - \hat{P}_f(z)\right)$ must be discretized for numerical implementation. This discretization can be performed using the trapezoidal rule for finite differences, with the derivation outlined as follows:

$$\frac{\hat{P}_f(z_i,t_j) - \hat{P}_f(z_i,t_{j-1})}{\Delta t} = 0.5\left[f(\hat{P}_f(z_i,t_{j-1}),t_{j-1}) + f(\hat{P}_f(z_i,t_j),t_j)\right] \tag{51}$$

where

$$f(\hat{P}_f(z_i,t_j),t_j) = i\omega_0'\hat{P}_f(z_i,t_j) + \gamma(\hat{E}_f(z_i,t_j) - \hat{P}_f(z_i,t_j)) \tag{52}$$

Substituting into the trapezoidal formula:

$$\begin{aligned}\frac{\hat{P}_f(z_i,t_j) - \hat{P}_f(z_i,t_{j-1})}{\Delta t} = 0.5\Big[&i\omega_0'\hat{P}_f(z_i,t_{j-1}) + \gamma(\hat{E}_f(z_i,t_{j-1}) - \hat{P}_f(z_i,t_{j-1})) \\ &+i\omega_0'\hat{P}_f(z_i,t_j) + \gamma(\hat{E}_f(z_i,t_j) - \hat{P}_f(z_i,t_j))\Big]\end{aligned} \tag{53}$$

Expanding the right-hand side:

$$\begin{aligned}\frac{\hat{P}_f(z_i,t_j) - \hat{P}_f(z_i,t_{j-1})}{\Delta t} = 0.5\Big[&(i\omega_0' - \gamma)\hat{P}_f(z_i,t_{j-1}) + \gamma\hat{E}_f(z_i,t_{j-1}) \\ &+(i\omega_0' - \gamma)\hat{P}_f(z_i,t_j) + \gamma\hat{E}_f(z_i,t_j)\Big]\end{aligned} \tag{54}$$

Rearranging:

$$\begin{aligned}\hat{P}_f(z_i,t_j)\left[1 - 0.5\Delta t(i\omega_0' - \gamma)\right] = &\hat{P}_f(z_i,t_{j-1})\left[1 + 0.5\Delta t(i\omega_0' - \gamma)\right] \\ &+0.5\Delta t\gamma(\hat{E}_f(z_i,t_{j-1}) + \hat{E}_f(z_i,t_j))\end{aligned} \tag{55}$$

Substituting $(i\omega_0' - \gamma) = C\omega_0$:

$$\begin{aligned}\hat{P}_f(z_i,t_j)\left[1 - 0.5\Delta t(C\omega_0)\right] = &\hat{P}_f(z_i,t_{j-1})\left[1 + 0.5\Delta t(C\omega_0)\right] \\ &+0.5\Delta t\gamma(\hat{E}_f(z_i,t_{j-1}) + \hat{E}_f(z_i,t_j))\end{aligned} \tag{56}$$

Solve for $\hat{P}_f(z_i,t_j)$:

$$\hat{P}_f(z_i,t_j) = \frac{\left[1+0.5\Delta t(C\omega_0)\right]\hat{P}_f(z_i,t_{j-1}) + 0.5\Delta t\gamma(\hat{E}_f(z_i,t_{j-1}) + \hat{E}_f(z_i,t_j))}{1-0.5\Delta t(C\omega_0)} \tag{57}$$

Expanding and rearranging the numerator:

$$\left[1+0.5\Delta t(C\omega_0)\right]\hat{P}_f(z_i,t_{j-1}) = \hat{P}_f(z_i,t_{j-1}) + 0.5\Delta t(C\omega_0)\hat{P}_f(z_i,t_{j-1}) \tag{58}$$

Substituting back:

$$\begin{aligned}\hat{P}_f(z_i,t_j) = {} & \frac{\hat{P}_f(z_i,t_{j-1}) + 0.5\Delta t(C\omega_0)\hat{P}_f(z_i,t_{j-1})}{1-0.5\Delta t(C\omega_0)} \\ & + \frac{0.5\Delta t\gamma(\hat{E}_f(z_i,t_{j-1}) + \hat{E}_f(z_i,t_j))}{1-0.5\Delta t(C\omega_0)}\end{aligned} \tag{59}$$

Factoring out $0.5\Delta t$, we get:

$$\hat{P}_f(z_i,t_j) = \frac{\hat{P}_f(z_i,t_{j-1}) + 0.5\Delta t\Big[(C\omega_0)\hat{P}_f(z_i,t_{j-1}) + \gamma(\hat{E}_f(z_i,t_{j-1}) + \hat{E}_f(z_i,t_j))\Big]}{1-0.5\Delta t(C\omega_0)} \tag{60}$$

Define a new term T_f:

$$T_f = C\omega_0\hat{P}_f(z_i,t_{j-1}) + \gamma(\hat{E}_f(z_{i-1},t_{j-1}) + \hat{E}_f(z_{i-1},t_j)) \tag{61}$$

Hence, the equation becomes:

$$\hat{P}_f(z_i,t_j) = \frac{\hat{P}_f(z_i,t_{j-1}) + 0.5\Delta t T_f}{1-0.5\Delta t C\omega_0} \tag{62}$$

Similarly, the same the trapezoidal finite difference derivation is done for $\hat{P}_r$ from Equation (50) to obtain:

$$\hat{P}_r(z_i,t_j) = \frac{\hat{P}_r(z_i,t_{j-1}) + 0.5\Delta t\Big[C\omega_0\hat{P}_r(z_i,t_{j-1}) + \gamma(\hat{E}_r(z_{i+1},t_{j-1}) + \hat{E}_r(z_{i+1},t_j))\Big]}{1-0.5\Delta t(C\omega_0)} \tag{63}$$

Where T_r is define as:

$$T_r = C\omega_0 \hat{P}_r(z_i, t_{j-1}) + \gamma(\hat{E}_r(z_{i+1}, t_j) + \hat{E}_r(z_{i+1}, t_{j-1})) \tag{64}$$

Hence, the equation becomes:

$$\hat{P}_r(z_i, t_j) = \frac{\hat{P}_r(z_i, t_{j-1}) + 0.5\Delta t T_r}{1 - 0.5\Delta t C\omega_0} \tag{65}$$

The reasoning behind the indices of $\hat{E}_f$ inside $\hat{T}_f$ and $\hat{E}_r$ inside $\hat{T}_r$ comes from the derivation for $\hat{E}_f$ and $\hat{E}_r$ in Milestone 1.

Now that the code-able discretized TWM equations with dispersion terms have been derived, they need to be represented using some MATLAB code so they can be simulated to check for their validity.

Code

The full code corresponding to Milestone 4 can be found in Appendix under 7.6 Section F.
Now that the code-able discretized TWM equations with dispersion terms have been derived, they need to be represented using some MATLAB code so they can be simulated to check for their validity. The code used to represent Equations (61),(62) and (64),(65) and its validity can be found at lines 40-43, 89-92, 139-154, 224-227.

The various pieces of code are as follows:

```
% Material Polarization Information
g_fwhm = 3.5e+012/10;          % Frequency full width at half maximum
LGamma = g_fwhm * 2 * pi;      % Polarization decay rate
Lw0 = 0;                       % Resonance frequency offset
LGain = 0.01;                  % Gain constant
```

This piece of code above defines the various constants such as the frequency, polarization, offset, and gain. These parameters are used in functions within the main simulation loop to model the polarization and field interactions accurately.

```
Efp = Ef;
Erp = Er;
Pfp = Pf;
Prp = Pr;
```

This section creates four new variables `Efp`, `Erp`, `Pfp`, and `Prp` that store the current iteration's E_{field} and Polarization. These variables save present state of the system, allowing for calculations that depend on both current and previous time steps.

```
Pf(1) = 0;     % Zero polarization at left boundary
Pf(Nz) = 0;    % Zero polarization at right boundary
Pr(1) = 0;     % Zero polarization at right boundary
Pr(Nz) = 0;    % Zero polarization at left boundary

Cw0 = -LGamma + 1i * Lw0;  % Complex response function of the material

% Dispersion calculations
Tf = LGamma * Ef(1:Nz-2) + Cw0 * Pfp(2:Nz-1) + LGamma * Efp(1:Nz-2);
Pf(2:Nz-1) = (Pfp(2:Nz-1) + 0.5 * dt * Tf) ./ (1 - 0.5 * dt * Cw0);

Tr = LGamma * Er(3:Nz) + Cw0 * Prp(2:Nz-1) + LGamma * Erp(3:Nz);
Pr(2:Nz-1) = (Prp(2:Nz-1) + 0.5 * dt * Tr) ./ (1 - 0.5 * dt * Cw0);

Ef(2:Nz-1) = Ef(2:Nz-1) - LGain * (Ef(2:Nz-1) - Pf(2:Nz-1));
Er(2:Nz-1) = Er(2:Nz-1) - LGain * (Er(2:Nz-1) - Pr(2:Nz-1));
```

The above code performs several critical operations. Lines 1–4 set the boundary conditions for the polarization fields. Specifically, `Pf` (forward polarization field) is set to zero at the left and right boundaries, while `Pr` (backward polarization field) is set to zero at the right and left boundaries, ensuring no polarization exists at the domain edges. Lines 9-10 perform dispersion calculations for the forward polarization field `Pf`, corresponding to equations (49) (or (61), (62)), whereas lines 12-13 calculate the backward polarization field `Pr`, which corresponds to equations (50) (or (64), (65)). The updated `Pr` and `Pf` values use previous polarization values from `Pfp` and `Prp`. Lastly, lines 16 and 17 update the forward and backward electric fields `Ef` and `Er` by adding the newly calculated polarization fields `Pf` and `Pr`. These updates correspond to equations (47) and (48), ensuring the electric fields account for polarization effects and completing the coupled field-polarization calculations

```
% Update previous values for next iteration
    Efp = Ef;
    Erp = Er;
    Pfp = Pf;
    Prp = Pr;
```

Although this section of the code may seem repetitive, it is important for updating the previous iteration's values with the current iteration's values, allowing the next time step to use the most recent data.

Simulation

Plot of the time domain and frequency domain magnitude and phase of a pulse at the beginning and end

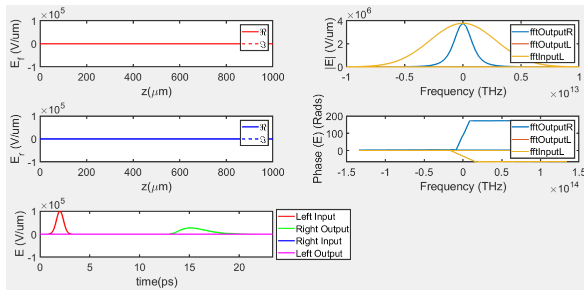


Figure 10: Magnitude and phase of a polarized Gaussian pulse propagating through the waveguide in the time and frequency domains

Figure 10 represents the simulation of a polarized Gaussian pulse propagating through the waveguide. We can see from the E_{field}-vs-time plot that when the input it detected, it has a certain shape, but when it is detected at the right output, its magnitude is much lower whereas the pulse is much wider. We can see this phenomenon in the frequency domain, where the width of the pulse at the right output is far lesses than that of the input pulse. This makes sense because as a pulse widens in time domain, its width decreases in frequency domain, as seen in Figure 10.

Plot of the time domain and frequency domain magnitude and phase of a pulse at the beginning and end with gratings added

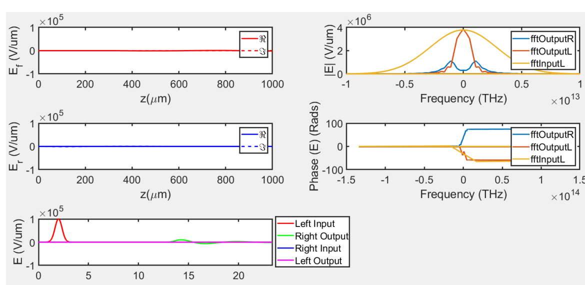


Figure 11: Mag. and phase of a Gauss. pulse passing through grating in the time and freq. domains

Figure 11 represents a similar simulation as done in Figure 10, except there is a grating region inside the waveguide. In this simulation. the input Gaussian pulse not only decreases over time due to polarization, it also gets affected by the grating region. We can see this more clearly in the frequency domain plots, where the grating filters a small band of low frequency components, whereas the only a thin of slightly higher frequencies with highly attenuated magnitude are allowed to pass.

In conclusion, Milestone 4 involved deriving a numerically implementable form of the travelling wave equation with polarization, implementing the derived equations in MATLAB, and obtaining simulation results in the form of plots. The derivation and code were verified to be correct, as the simulation results behaved as intended and matched the expected outcomes.

6 Passive Device Exploration (Milestone 5)

Milestone 5 focuses on finding a solution to a problem by designing an experiment that uses the available physical parameters of the waveguide and the Gaussian pulse. The objective is to understand how varying these parameters affects the numerical solution.

The topic under investigation is the implementation of polarization for gain/loss dispersion. This is explored by varying the parameters ω_0 and γ, and by replacing the polarization code for P_r and P_f with a version derived from a Backward Euler finite difference scheme.

Part A

Investigation of ω_0 Variation

First, we assume that the derivations and code from Milestone 4 are accurate and free of critical errors. Under this assumption, ω_0 will be varied and the results compared with those from Milestone 4 to analyze its effect on pulse propagation.

In Milestone 4, an ω_0 value of 0 was used, meaning ω_0 did not influence wave propagation. In this milestone, ω_0 is varied across four values: 100 MHz, 10 GHz, 1 THz, and 100 THz. These values were chosen to observe a clear trend in how ω_0 affects wave propagation.

Before delving into the simulations, it is useful to examine the effect from an analytical perspective. According to equation (56), ω_0 is related to C_{ω_0}, a term in the polarization equations. When $\omega_0 = 0$, C_{ω_0} is purely real. By setting $\omega_0 \neq 0$, an imaginary component is introduced, making C_{ω_0} complex. Mathematically, adding an imaginary component increases the complexity of the expression, which should manifest in the system's behavior.

Plot for $\omega_0 = 100$ MHz

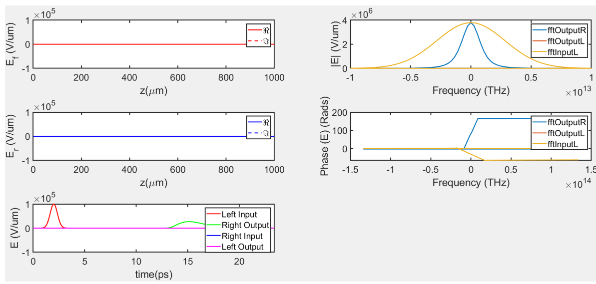


Figure 12: Frequency domain response of the Gaussian pulse for $\omega_0 = 100$ MHz

Figure 12 shows the frequency domain response of the system for $\omega_0 = 100$ MHz. It is the same as what we got in Milestone 4, meaning that for this value of $\omega_0 = 100$, there is no significant effect on the system.

Plot for $\omega_0 = 10$ GHz

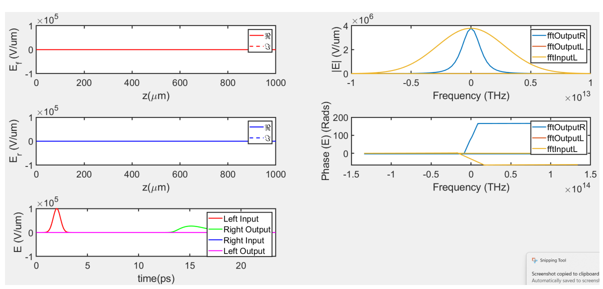
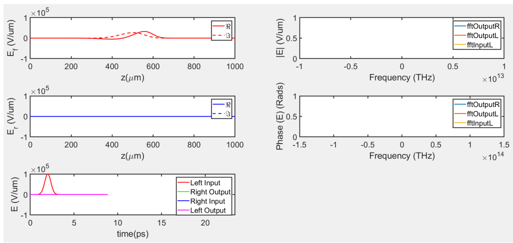


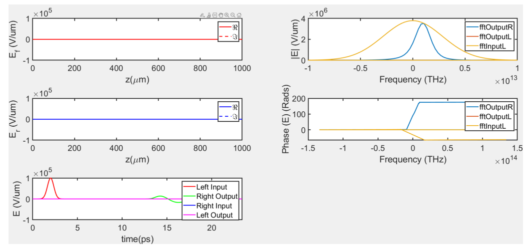
Figure 13: Frequency domain response of the Gaussian pulse for $\omega_0 = 10$ GHz

Figure 13 shows the frequency domain response of the system for $\omega_0 = 10$ GHz. It is the same as what we got in Milestone 4, meaning that for this value of $\omega_0 = 100$, there is no significant effect on the system.

Plot for $\omega_0 = 1$ THz



(a) Propagation behavior of the pulse in the waveguide

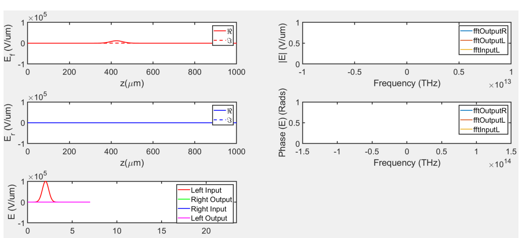


(b) Frequency domain response for $\omega_0 = 1$ THz

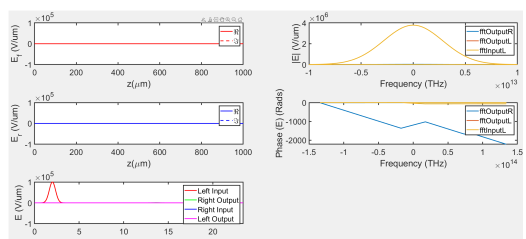
Figure 14: Pulse propagation and frequency response for $\omega_0 = 1$ THz

Figure 14 shows the Gaussian pulse's propagation and frequency response for $\omega_0 = 1$ THz. In (a), the pulse appears to undergo detuning or modulation during propagation along the z-direction. This is confirmed by the frequency domain response in (b), which shows a rightward shift in the magnitude vs. frequency plot, indicating modulation of the pulse.

Plot for $\omega_0 = 100$ THz



(a) Rapid amplitude reduction of the propagating pulse



(b) Frequency domain response for $\omega_0 = 100$ THz

Figure 15: Pulse propagation and frequency response for $\omega_0 = 100$ THz

Figure 15 shows the Gaussian pulse's response for $\omega_0 = 100$ THz. In (a), the amplitude of the propagating pulse quickly diminishes to nearly zero, indicating complete dispersion before reaching the waveguide's right boundary. This is further confirmed in (b), where the magnitude vs. frequency plot shows no detectable signal at the right output.

Analysis of ω_0 Variation

The results demonstrate how ω_0 influences pulse propagation. At lower frequencies (100 MHz and 10 GHz), the behavior resembles the dispersion observed in Milestone 4: the pulse broadens in the time domain while narrowing in the frequency domain. At 1 THz, de-tuning and high-frequency modulation effects emerge, with a noticeable shift in the frequency response. By 100 THz, the pulse is completely attenuated, indicating full energy dispersion before it reaches the waveguide's end. This allows us to draw a very strong conclusion that if we want to ensure that the phenomenon of dispersion doesn't disperse the input pulse, the frequency of ω_0 must be below a set threshold, which for us lies between 10 GHz and 1 THz, although probably closer to 10 GHz.

Investigation of γ Variation

A similar approach is applied to γ to explore its effect on pulse propagation. From equation (56), γ corresponds to the real part of C_{ω_0} and appears throughout the T_f and T_r equations, suggesting that even small variations can significantly affect the system. To test this hypothesis, the system will be simulated for multiple γ values to identify trends in pulse propagation. Additionally, it is important to note that the ω_0 value has been reset to 0.

The plot when gamma's term $g_{\mathbf{fwhm}}$ is set to 3.5 GHz

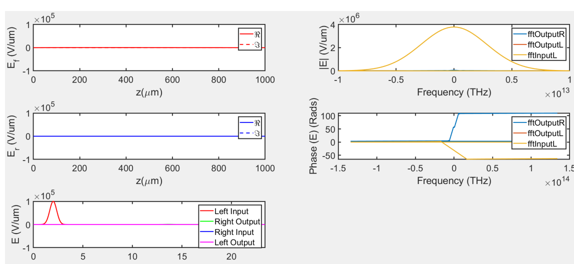


Figure 16: Magnitude and phase of a polarized Gaussian pulse with gamma being 3.5 GHz propagating through the waveguide in the time and frequency domains

Figure 16 shows the time evolution of a polarized Gaussian pulse, where its amplitude is completely attenuated before it is detected at the right boundary.

The plot when gamma's term $g_{\mathbf{fwhm}}$ is set to 35 GHz

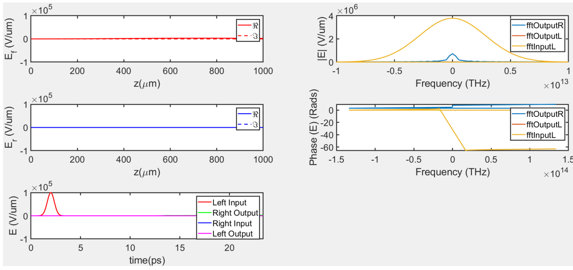


Figure 17: Magnitude and phase of a polarized Gaussian pulse with gamma being 35 GHz propagating through the waveguide in the time and frequency domains

Figure 17 shows the time evolution of a polarized Gaussian pulse, where its amplitude is mostly attenuated by the time it reaches the end of the waveguide. From the magnitude vs. frequency plot, the heavily attenuated low-frequency components can be observed at the output.

The plot when gamma's term $g_{\mathbf{fwhm}}$ is set to 3.5 THz

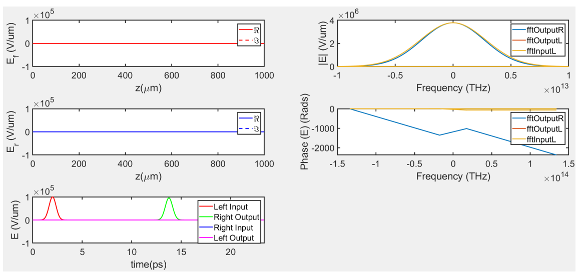


Figure 18: Magnitude and phase of a polarized Gaussian pulse with gamma being 3.5 THz propagating through the waveguide in the time and frequency domains

Figure 18 shows the time evolution of a polarized Gaussian pulse. The magnitude vs. frequency and field detection plots indicate that there is almost no attenuation of the input signal. Both plots show that the detected pulse maintains most of its amplitude.

The plot when gamma's term $g_{\mathbf{fwhm}}$ is set to 35 THz

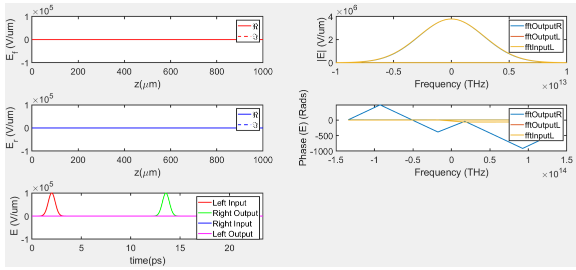


Figure 19: Magnitude and phase of a polarized Gaussian pulse with gamma being 35 THz propagating through the waveguide in the time and frequency domains

Figure Figure 19 shows the time evolution of a polarized Gaussian pulse, where it can be inferred from the magnitude vs frequency and the field detection plot that basically no attenuation happened to the input signal as it propagated down the waveguide. Both those plots show that the pulse that was detected maintains the same amplitude at the beginning and end of its lifespan.

Analysis of γ Variation

This section involves simulating the system for four different values of γ to identify trends in the behavior of the polarized Gaussian pulse. The simulations revealed a clear trend. At $\gamma = 3.5$ GHz, the propagating Gaussian pulse was fully attenuated before reaching the end of the waveguide. When γ was increased to 35 GHz, a similar phenomenon was observed; however, some highly attenuated low-frequency components were detected at the right output of the waveguide. This indicates that a γ of 35 GHz caused slightly less attenuation compared to the case with $\gamma = 3.5$ GHz. Further evidence supporting this trend was observed at $\gamma = 3.5$ THz. As the Gaussian pulse propagated through the waveguide, it experienced almost no attenuation. This was confirmed by the magnitude versus frequency plot, where the Fast Fourier Transform (FFT) curves for the input and the pulse detected at the right boundary were nearly identical. Finally, the response for $\gamma = 35$ THz reinforced the trend that increasing the frequency of γ reduces the attenuation caused by dispersion. At sufficiently high frequencies, γ effectively counteracts the dispersion phenomenon. The parameter γ is thus a significant numerical and physical quantity, playing a crucial role in controlling the attenuation and dispersion behavior of the propagating Gaussian pulse.

Part B

This part of Milestone 5 deals with re-deriving Equations (49) and (50) using the Backward Euler finite difference scheme. The derivation is as follows:

$$\frac{\hat{P}_f(z_i,t_j)-\hat{P}_f(z_i,t_{j-1})}{\Delta t} = f(\hat{P}_f(z_i,t_j),t_j) \tag{66}$$

where

$$f(\hat{P}_f(z_i,t_j),t_j) = i\omega_0'\hat{P}_f(z_i,t_j)+\gamma(\hat{E}_f(z_i,t_j)-\hat{P}_f(z_i,t_j)) \tag{67}$$

Substituting into the Backward Euler formula:

$$\frac{\hat{P}_f(z_i,t_j)-\hat{P}_f(z_i,t_{j-1})}{\Delta t} = i\omega_0'\hat{P}_f(z_i,t_j)+\gamma(\hat{E}_f(z_i,t_j)-\hat{P}_f(z_i,t_j)) \tag{68}$$

Expanding and rearranging terms:

$$\hat{P}_f(z_i,t_j)-\hat{P}_f(z_i,t_{j-1}) = \Delta t[(i\omega_0'-\gamma)\hat{P}_f(z_i,t_j)+\gamma\hat{E}_f(z_i,t_j)] \tag{69}$$

Rearranging to isolate $\hat{P}_f(z_i,t_j)$:

$$\hat{P}_f(z_i,t_j)[1-\Delta t(i\omega_0'-\gamma)] = \hat{P}_f(z_i,t_{j-1})+\Delta t\gamma\hat{E}_f(z_i,t_j) \tag{70}$$

Solving for $\hat{P}_f(z_i,t_j)$:

$$\hat{P}_f(z_i,t_j) = \frac{\hat{P}_f(z_i,t_{j-1})+\Delta t\gamma\hat{E}_f(z_i,t_j)}{1-\Delta t(i\omega_0'-\gamma)} \tag{71}$$

Substituting the same constant from Equation (56), we get:

$$\hat{P}_f(z_i,t_j) = \frac{\hat{P}_f(z_i,t_{j-1})+\Delta t\gamma\hat{E}_f(z_i,t_{j-1})}{1-\Delta tC_{w_0}} \tag{72}$$

Similarly, for the backward wave polarization $\hat{P}_r$, applying the same Backward Euler discretization derives to :

$$\hat{P}_r(z_i,t_j) = \frac{\hat{P}_r(z_{i+1},t_{j-1})+\Delta t\gamma\hat{E}_r(z_{i+1},t_{j-1})}{1-\Delta tC_{w_0}} \tag{73}$$

Code

Based on the discrete equations derived in (72) and (73), MATLAB code was built to simulate the equations properly. Full code can be found in Section 7.7. The code is as follows:

```
% Dispersion Calculations
% Backward Euler Polarization Update
% Forward polarization
Tf = LGamma * Efp(2:Nz-1);  % Source term for forward polarization
Pf(2:Nz-1) = (Pfp(2:Nz-1) + dt * Tf) ./ (1 - dt * Cw0);  % Fwd polarization

% Backward polarization
Tr = LGamma * Erp(2:Nz-1);  % Source term for backward polarization
Pr(2:Nz-1) = (Prp(2:Nz-1) + dt * Tr) ./ (1 - dt * Cw0);  % Bwd polarization
```

Simulation

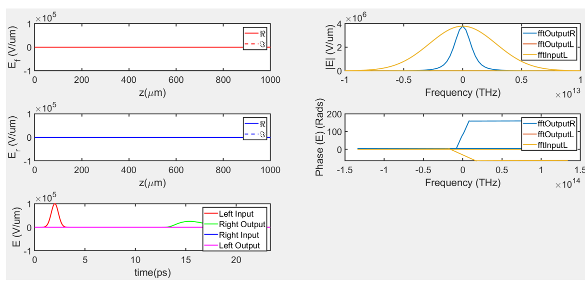


Figure 20: The effect of dispersion on a normal Gaussian pulse propagating through the waveguide

From Figure 20, we can see that the response of the magnitude vs frequency plot is the same as the one in Milestone 4, this means that the derivation and the code modeling the derivation was correct. This also tells us that the trapezoidal rule and the backward Euler both work for this kind of derivation, meaning there are more than one way to solve a problem.

7 Appendix: Code Listings

7.1 Section A: Source Function

```
function E = SourceFct(t, InputParas)
% SourceFct computes a source signal E based on the input time t and parameters.
%
% There are two ways the function works:
% 1. If InputParas has a 'rep' field, the time t is adjusted so that it
%    always falls within one period of the repetition.
% 2. If InputParas is a structure, the function calculates E as a
%    Gaussian envelope / pulse modulated by a complex exponential. Otherwise,
%    If InputParas is not a structure (for example, if it's already a
%    precomputed numeric signal), the function simply assigns it to E and returns it

% Check if the parameters include a repetition period ('rep')
if isfield(InputParas, 'rep')
    % Calculate the number of full periods that fit into time t
    n = floor(t / InputParas.rep);
    % Subtract the full periods from t, so t is now within one period
    t = t - n * InputParas.rep;
end

% If InputParas is not a structure, assume it directly represents the signal E itself
if ~isstruct(InputParas)
    E = InputParas;
else
    % If InputParas is a structure, calculate the signal E using parameters from that structure:
    % E0  - Amplitude of the signal (scaling factor)
    % t0  - Center (in time) of the Gaussian envelope / pulse
    % wg  - Width of the Gaussian envelope / pulse
    % we  - Angular frequency of the oscillation (how fast it oscillates)
    % phi - Phase offset of the oscillation
    %
    % The computed signal E is the product of:
    %   - A Gaussian envelope/function : exp(-((t-t0)^2 / wg^2))
    %   - A complex oscillation: exp(1i*(we*t + phi))
    E = InputParas.E0 * exp(- (t - InputParas.t0)^2 / InputParas.wg^2) ...
        * exp(1i * (InputParas.we * t + InputParas.phi));
end

end
```

Listing 1: MATLAB code for the input source, common to all milestones

7.2 Section B: Milestone 1 – TWM Simulation with Reflections

```
set(0,'defaultaxesfontsize',20)
set(0,'DefaultFigureWindowStyle','docked')
set(0,'DefaultLineLineWidth',2);
set(0,'Defaultaxeslinewidth',2)

set(0,'DefaultFigureWindowStyle','docked')


c_c = 299792458;                  % m/s TWM speed of light
c_eps_0 = 8.8542149e-12;          % F/m vaccum permittivity
c_eps_0_cm = c_eps_0/100;         % F/cm vaccum permittivity
c_mu_0 = 1/c_eps_0/c_c^2;         % Permiability of free space
c_q = 1.60217653e-19;             % Charge of an electon
c_hb = 1.05457266913e-34;         % Dirac / Reduced Planck constant
c_h = c_hb*2*pi;                  % Planck constant

InputParasL.E0 = 1e5;             % Amplitude of the input E-field / E_f
InputParasL.we = 0;               % Frequency of the complex sinusoidal modulation on the gaussian pulse
InputParasL.t0 = 2e-12;           % The constant we are shifting the time by
```

```
InputParasL.wg = 5e-13;             % Width of the Gaussian distribution
InputParasL.phi = 0;                % Initial Phase of the E_f / input E-field
InputParasR = 0;                    % Placeholder variable for reverse propagation

n_g = 3.5;                          % Constant to control group velocity
vg = c_c/n_g *1e2;                  % TWM cm/s group velocity
Lambda = 1550e-9;                   % Wavelength of light

plotN = 50;                         % Divisior constant

L = 1000e-6*1e2;                    % length of the waveguide in cm
XL = [0,L];                         % Start and End of the x-axis
YL = [0,InputParasL.E0];            % Start and End of the y-axis

Nz = 500;                           % Number of spatial points
dz = L/(Nz-1);                      % Distance between every point
dt = dz/vg;                         % Time step to plot every point
fsync = dt*vg/dz;                   % Equals 1, allows the Gaussian to be stable

Nt = floor(2*Nz);                   % Number of time steps
tmax = Nt*dt;                       % Maximum time for simulation
t_L = dt*Nz;                        % Time for the wave to travel the entire waveguide length (s)

z = linspace(0,L,Nz).';             % Spatial grid from 0 to L with Nz points
time = nan(1,Nt);                   % Time matrix with 1 row and Nt columns / row vector of Nt elements
InputL = nan(1,Nt);                 % Matrix with 1 row and Nt columns / row vector of Nt elements
InputR = nan(1, Nt);                % Matrix with 1 row and Nt columns / row vector of Nt elements
OutputL = nan(1,Nt);                % Matrix with 1 row and Nt columns / row vector of Nt elements
OutputR = nan(1,Nt);                % Matrix with 1 row and Nt columns / row vector of Nt elements

Ef = zeros(size(z));                % Forward field along the waveguide (initialized to 0)
Er = zeros(size(z));                % Backward field along the waveguide (initialized to 0)

Ef1 = @SourceFct;                   % Handle to the source function for forward input
ErN = @SourceFct;                   % Handle to the source function for backward input

t = 0;                              % Set t to a starting value of 0
time(1) = t;                        % Sets the first element of the time vector to 0

InputL(1) = Ef1(t, InputParasL);    % Set initial value of InputL using the source function
InputR(1) = ErN(t, InputParasR);    % Set initial value of InputR using the source function

OutputR(1) = Ef(Nz);                % Initial right output field (at the end of the waveguide)
OutputL(1) = Er(1);                 % Initial left output field (at the start of the waveguide)

Ef(1) = InputL(1);                  % Initializes forward field at z = 0 (Input signal from the left)
Er(Nz) = InputR(1);                 % Initializes backward field at z = L (Input signal from the right)

figure('name', 'Fields')
subplot(3,1,1)
plot(z*1000, real(Ef), 'r');
hold off
xlabel('z(\mum)')
ylabel('E_f')
subplot(3,1,2)
plot(z*1000, real(Er), 'b');
xlabel('z(\mum)')
ylabel('E_r')
hold off
subplot(3,1,3)
plot(time*1e12, real(InputL), 'r'); hold on
plot(time*1e12, real(OutputR), 'r--');
plot(time*1e12, real(InputR), 'b'); hold on
plot(time*1e12, real(OutputL), 'b--');
xlabel('time(ps)')
ylabel('E')

hold off

for i = 2:Nt                                 % 2 to 1000 in steps of 1
    t = dt*(i-1);                            % Update time
    time(i) = t;                             % Record current time

    RL = 0.9i;                               % The left side reflection coefficient
    RR = 0.9i;                               % The right side reflection coefficient

    % Input fields at current time step
    InputL(i) = Ef1(t, InputParasL);         % At t, we input a signal characterized by InputParasL from the left
```

```matlab
    InputR(i) = ErN(t, 0);                  % At t, we input no signal from the right (since InputParasR = 0)

    % Boundary conditions with reflections
    Ef(1) = InputL(i) + RL*Er(1);           % Boundary condition at z = 0 (left side);
    Er(Nz) = InputR(i) + RR*Ef(Nz);         % Boundary condition at z = L (right side);

    % Wave propagation using upwind FD method
    Ef(2:Nz) = fsync * Ef(1:Nz-1);          % Forward field propagation from left to right
    Er(1:Nz-1) = fsync * Er(2:Nz);          % Backward field propagation from right to left

    % Output fields recorded at boundaries
    OutputR(i) = Ef(Nz) * (1 - RR);         % Right output at z = L (compensated for reflection)
    OutputL(i) = Er(1) * (1 - RL);          % Left output z = 0 (compensated for reflection)

    if mod(i,plotN) == 0                    % Only executed when i is multiple of plotN

        % Forward Propagation of the Gaussian Pulse
        subplot(3,1,1)
        plot(z*10000,real(Ef),'r'); hold on
        plot(z*10000,imag(Ef),'r--'); hold off
        xlim(XL*1e4)
        ylim(YL)
        xlabel('z(\mum)')
        ylabel('E_f')
        legend('\Re','\Im')
        hold off

        % Reverse (Reflection) of the Gaussian Pulse
        subplot(3,1,2)
        plot(z*10000, real(Er), 'b'); hold on
        plot(z*10000, imag(Er), 'b--'); hold off
        xlim(XL*1e4)
        ylim(YL)
        xlabel('z(\mum)')
        ylabel('E_r')
        legend('\Re', '\Im')

        hold off

        % Plot showing when the time when the input and output pulse were detected
        subplot(3,1,3);
        plot(time*1e12, real(InputL), 'r'); hold on
        plot(time*1e12, real(OutputR), 'g');
        plot(time*1e12, real(InputR), 'b');
        plot(time*1e12, real(OutputL), 'm');
        xlim([0, Nt*dt*1e12])
        ylim(YL)
        xlabel('time(ps)')
        ylabel('0')
        legend('Left Input', 'Right Output', 'Right Input', 'Left Output' ...
            , 'Location', 'east')
        hold off
        pause(0.01)
    end
end
```

Listing 2: MATLAB code for TWM simulation with reflections

7.3 Section C: Wspace Function

```matlab
function w = wspace(t, nt)

% This function constructs a linearly-spaced vector of angular
% frequencies that correspond to the points in an FFT spectrum.
% The second half of the vector is aliased to negative frequencies.
%
% USAGE
% w = wspace(tv);
% w = wspace(t, nt);
%
% INPUT
% tv - vector of linearly-spaced time values
```

```
% t  - scalar representing the periodicity of the time sequence
% nt - Number of points in time sequence
%      (should only be provided if first argument is scalar)
%
% OUTPUT
% w  - vector of angular frequencies
%
% EXAMPLE
% t = linspace(-10,10,2048)';       % set up time vector
% x = exp(-t.^2);                   % construct time sequence
% w = wspace(t);                    % construct w vector
% Xhat = fft(x);                    % calculate spectrum
% plot(w, abs(Xhat))                % plot spectrum
%
% AUTHOR: Thomas E. Murphy (tem@umd.edu)

if (nargin < 2)
    nt = length(t);
    dt = t(2) - t(1);
    t = t(nt) - t(1) + dt;
end

if (nargin == 2)
    dt = t / nt;
end

w = 2 * pi * (0:nt-1) / t;
kv = find(w >= pi / dt);
w(kv) = w(kv) - 2 * pi / dt;
```

Listing 3: MATLAB code for wspace function to compute frequency space

7.4 Section D: Milestone 2 – TWM Simulation with Static Gain, Detuning, and Frequency Domain

```
set(0,'defaultaxesfontsize',20)
set(0,'DefaultFigureWindowStyle','docked')
set(0,'DefaultLineLineWidth',2);
set(0,'Defaultaxeslinewidth',2)

set(0,'DefaultFigureWindowStyle','docked')


c_c = 299792458;                    % m/s TWM speed of light
c_eps_0 = 8.8542149e-12;            % F/m vaccum permittivity
c_eps_0_cm = c_eps_0/100;           % F/cm vaccum permittivity
c_mu_0 = 1/c_eps_0/c_c^2;           % Permiability of free space
c_q = 1.60217653e-19;               % Charge of an electon
c_hb = 1.05457266913e-34;           % Dirac / Reduced Planck constant
c_h = c_hb*2*pi;                    % Planck constant

beta_r = 80;                        % Pulse De-tuning Constant
beta_i = 8;                         % Pulse Gain Constant

InputParasL.E0 = 1e5;               % Amplitude of the input E-field / E_f
InputParasL.we = 0;                 % Frequency of the complex sinusoidal modulation on the gaussian pulse
InputParasL.t0 = 2e-12;             % The constant we are shifting the time by
InputParasL.wg = 5e-13;             % Width of the Gaussian distribution
InputParasL.phi = 0;                % Initial Phase of the E_f / input E-field
InputParasR = 0;                    % Placeholder variable for reverse propagation

n_g = 3.5;                          % Constant to control group velocity
vg = c_c/n_g *1e2;                  % TWM cm/s group velocity
Lambda = 1550e-9;                   % Wavelength of light

plotN = 50;                         % Division constant

L = 1000e-6*1e2;                    % length of the waveguide in cm
XL = [0,L];                         % Start and End of the x-axis
YL = [0,InputParasL.E0];            % Start and End of the y-axis

```

```
Nz = 500;                          % Number of spatial points
dz = L/(Nz-1);                     % Distance between every point
dt = dz/vg;                        % Time step to plot every point
fsync = dt*vg/dz;                  % Equals 1, allows the Gaussian to be stable

Nt = floor(2*Nz);                  % Number of time steps
tmax = Nt*dt;                      % Maximum time for simulation
t_L = dt*Nz;                       % Time for the wave to travel the entire waveguide length (s)

z = linspace(0,L,Nz).';            % Spatial grid from 0 to L with Nz points
time = nan(1,Nt);                  % Time matrix with 1 row and Nt columns / row vector of Nt elements
InputL = nan(1,Nt);                % Matrix with 1 row and Nt columns / row vector of Nt elements
InputR = nan(1, Nt);               % Matrix with 1 row and Nt columns / row vector of Nt elements
OutputL = nan(1,Nt);               % Matrix with 1 row and Nt columns / row vector of Nt elements
OutputR = nan(1,Nt);               % Matrix with 1 row and Nt columns / row vector of Nt elements

Ef = zeros(size(z));               % Forward field along the waveguide (initialized to 0)
Er = zeros(size(z));               % Backward field along the waveguide (initialized to 0)

Ef1 = @SourceFct;                  % Handle to the source function for forward input
ErN = @SourceFct;                  % Handle to the source function for backward input

t = 0;                             % Set t to a starting value of 0
time(1) = t;                       % Sets the first element of the time vector to 0

InputL(1) = Ef1(t, InputParasL);   % Set initial value of InputL using the source function
InputR(1) = ErN(t, InputParasR);   % Set initial value of InputR using the source function

OutputR(1) = Ef(Nz);               % Initial right output field (at the end of the waveguide)
OutputL(1) = Er(1);                % Initial left output field (at the start of the waveguide)

Ef(1) = InputL(1);                 % Initializes forward field at z = 0 (Input signal from the left)
Er(Nz) = InputR(1);                % Initializes backward field at z = L (Input signal from the right)

figure('name', 'Fields')
subplot(3,1,1)
plot(z*1000, real(Ef), 'r');
hold off
xlabel('z(\mum)')
ylabel('E_f')
subplot(3,1,2)
plot(z*1000, real(Er), 'b');
xlabel('z(\mum)')
ylabel('E_r')
hold off
subplot(3,1,3)
plot(time*1e12, real(InputL), 'r'); hold on
plot(time*1e12, real(OutputR), 'r--');
plot(time*1e12, real(InputR), 'b'); hold on
plot(time*1e12, real(OutputL), 'b--');
xlabel('time(ps)')
ylabel('E')

hold off

for i = 2:Nt                                % 2 to 1000 in steps of 1
    t = dt*(i-1);                           % Update time
    time(i) = t;                            % Record current time

    RL = 0.9i;                              % The left side reflection coefficient
    RR = 0.9i;                              % The right side reflection coefficient

    beta = ones(size(z))*(beta_r+1i*beta_i);  % Complex propagation constant
    exp_det = exp(-1i*dz*beta);               % Phase shift due to propagation over a distance dz

    % Input fields at current time step
    InputL(i) = Ef1(t, InputParasL);        % At t, we input a signal characterized by InputParasL from the left
    InputR(i) = ErN(t, 0);                  % At t, we input no signal from the right (since InputParasR = 0)

    % Boundary conditions with reflections
    Ef(1) = InputL(i) + RL*Er(1);           % Boundary condition at z = 0 (left side);
    Er(Nz) = InputR(i) + RR*Ef(Nz);         % Boundary condition at z = L (right side);

    % Wave propagation using upwind FD method and quasi-static solution
    Ef(2:Nz) = fsync*exp_det(1:Nz-1).*Ef(1:Nz-1);       % Forward field propagation from left to right
    Er(1:Nz-1) = fsync*exp_det(2:Nz).*Er(2:Nz);         % Backward field propagation from right to left

    % Output fields recorded at boundaries
```

```
    OutputR(i) = Ef(Nz) * (1 - RR);         % Right output at z = L (compensated for reflection)
    OutputL(i) = Er(1) * (1 - RL);          % Left output z = 0 (compensated for reflection)

    % FFT data from the outputs
    fftOutput = fftshift(fft(OutputR));     % Get FFT data for OutputR
    omega = fftshift(wspace(time));

    if mod(i,plotN) == 0                    % Only executed when i is multiple of plotN

        % Forward Propagation of the Gaussian Pulse
        subplot(3,1,1)
        plot(z*10000,real(Ef),'r'); hold on
        plot(z*10000,imag(Ef),'r--'); hold off
        xlim(XL*1e4)
        ylim(YL)
        xlabel('z(\mum)')
        ylabel('E_f')
        legend('\Re','\Im')
        hold off

        % Reverse (Reflection) of the Gaussian Pulse
        subplot(3,1,2)
        plot(z*10000, real(Er), 'b'); hold on
        plot(z*10000, imag(Er), 'b--'); hold off
        xlim(XL*1e4)
        ylim(YL)
        xlabel('z(\mum)')
        ylabel('E_r')
        legend('\Re', '\Im')

        hold off

        % Plot showing when the time when the input and output pulse were detected
        subplot(3,1,3);
        plot(time*1e12, real(InputL), 'r'); hold on
        plot(time*1e12, real(OutputR), 'g');
        plot(time*1e12, real(InputR), 'b');
        plot(time*1e12, real(OutputL), 'm');
        xlim([0, Nt*dt*1e12])
        ylim(YL)
        xlabel('time(ps)')
        ylabel('0')
        legend('Left Input', 'Right Output', 'Right Input', 'Left Output' ...
            , 'Location', 'east')
        hold off

        % Plot the spectral content of the Gaussion (or Gaussian Modulated) Pulse
        subplot(3,2,2);
        plot(omega, abs(fftOutput));
        xlabel('Frequency (THz)');
        ylabel('|E|');
        xlim([-1.5e14, 1.5e14]);

        % Plot the Phase of the Gaussion (or Gaussian Modulated) Pulse
        subplot(3,2,4);
        phase = unwrap(angle(fftOutput));  % Unwrap the phase
        plot(omega, phase);
        xlabel('Frequency (THz)');
        ylabel('Phase (E)');
        xlim([-1.5e14, 1.5e14]);

        pause(0.01)
    end
end
```

Listing 4: MATLAB code for TWM simulation with with Static Gain, Detuning, and Frequency Domain

7.5 Section E: Milestone 3 – Gratings and the use of $\hat{\kappa}$

```
set(0,'defaultaxesfontsize',20)
```

```
set(0,'DefaultFigureWindowStyle','docked')
set(0,'DefaultLineLineWidth',2);
set(0,'Defaultaxeslinewidth',2)

set(0,'DefaultFigureWindowStyle','docked')

c_c = 299792458;                  % m/s TWM speed of light
c_eps_0 = 8.8542149e-12;          % F/m vaccum permittivity
c_eps_0_cm = c_eps_0/100;         % F/cm vaccum permittivity
c_mu_0 = 1/c_eps_0/c_c^2;         % Permiability of free space
c_q = 1.60217653e-19;             % Charge of an electon
c_hb = 1.05457266913e-34;         % Dirac / Reduced Planck constant
c_h = c_hb*2*pi;                  % Planck constant

beta_r = 0;                       % De-tuning constant
beta_i = 0;                       % Gain Constant

kappa0 = 100;                     % Coupling coefficient
kappaStart = 1/3;                 % Constant defines starting position where coupling begins.
kappaStop = 2/3;                  % Constant defines ending position where coupling stops.

InputParasL.E0 = 1e5;             % Amplitude of the input E-field / E_f
InputParasL.we = 0;               % Frequency of the complex sinusoidal modulation on the gaussian pulse
InputParasL.t0 = 2e-12;           % The constant we are shifting the time by
InputParasL.wg = 5e-13;           % Width of the Gaussian distribution
InputParasL.phi = 0;              % Initial Phase of the E_f / input E-field
InputParasR = 0;                  % Placeholder variable for reverse propagation

n_g = 3.5;                        % Constant to control group velocity
vg = c_c/n_g *1e2;                % TWM cm/s group velocity

Lambda = 1550e-9;                 % Wavelength of light

plotN = 10;                       % Division constant

L = 1000e-6*1e2;                  % length of the waveguide in cm

XL = [0,L];                       % Start and End of the x-axis
YL = [-InputParasL.E0,InputParasL.E0];       % Start and End of the y-axis

Nz = 500;                         % Number of spatial points
dz = L/(Nz-1);                    % Distance between every point
dt = dz/vg;                       % Time step to plot every point
fsync = dt*vg/dz;                 % Equals 1, allows the Gaussian to be stable

Nt = floor(2*Nz);                 % Number of time steps
tmax = Nt*dt;                     % Maximum time for simulation
t_L = dt*Nz;                      % Time for the wave to travel the entire waveguide length (s)
z = linspace(0,L,Nz);             % Spatial grid from 0 to L with Nz points
time = nan(1,Nt);                 % Time matrix with 1 row and Nt columns / row vector of Nt elements

InputL = nan(1,Nt);               % Matrix with 1 row and Nt columns / row vector of Nt elements
InputR = nan(1, Nt);              % Matrix with 1 row and Nt columns / row vector of Nt elements
OutputL = nan(1,Nt);              % Matrix with 1 row and Nt columns / row vector of Nt elements
OutputR = nan(1,Nt);              % Matrix with 1 row and Nt columns / row vector of Nt elements
Ef = zeros(size(z));              % Forward field along the waveguide (initialized to 0)
Er = zeros(size(z));              % Backward field along the waveguide (initialized to 0)
Ef1 = @SourceFct;                 % Handle to the source function for forward input
ErN = @SourceFct;                 % Handle to the source function for backward input

t = 0;                            % Set t to a starting value of 0
time(1) = t;                      % Sets the first element of the time vector to 0

InputL(1) = Ef1(t, InputParasL);  % Set initial value of InputL using the source function
InputR(1) = ErN(t, InputParasR);  % Set initial value of InputR using the source function

OutputR(1) = Ef(Nz);              % Initial right output field (at the end of the waveguide)
OutputL(1) = Er(1);               % Initial left output field (at the start of the waveguide)

Ef(1) = InputL(1);                % Initializes forward field at z = 0 (Input signal from the left)
Er(Nz) = InputR(1);               % Initializes backward field at z = L (Input signal from the right)

kappa = kappa0*ones(size(z));     % Creates an array of size z, where all indexes hold a value of kappa0
kappa(z<L*kappaStart) = 0;        % Sets the limit such that kappa is set to zero outside the interaction region
kappa(z>L*kappaStop) = 0;         % Sets the limit such that kappa is set to zero outside the interaction region.

figure('name', 'Fields')

```

```
subplot(3,1,1)
plot(z*1000, real(Ef), 'r');
hold off
xlabel('z(\mum)')
ylabel('E_f')

subplot(3,1,2)
plot(z*1000, real(Er), 'b');
xlabel('z(\mum)')
ylabel('E_r')
hold off

subplot(3,1,3)
plot(time*1e12, real(InputL), 'r'); hold on
plot(time*1e12, real(OutputR), 'r--');
plot(time*1e12, real(InputR), 'b'); hold on
plot(time*1e12, real(OutputL), 'b--');
xlabel('time(ps)')
ylabel('E')
hold off

for i = 2:Nt                    % 2 to 1000 in steps of 1

    t = dt*(i-1);
    time(i) = t;

    RL = 0;                               % The left side reflection coefficient
    RR = 0;                               % The right side reflection coefficient

    beta = ones(size(z))*(beta_r+1i*beta_i); % Complex propagation constant
    exp_det = exp(-1i*dz*beta);              % Phase shift due to propagation over a distance dz

    % % Input fields at current time step
    InputL(i) = Ef1(t, InputParasL);  % At t, we input a signal characterized by InputParasL from the left

    InputR(i) = ErN(t, 0);            % At t, we input no signal from the right (since InputParasR = 0)

    % Boundary conditions with reflections
    Ef(1) = InputL(i) + RL*Er(1);     % Boundary condition at z = 0 (left side);
    Er(Nz) = InputR(i) + RR*Ef(Nz);   % Boundary condition at z = L (right side)

    % Propagation using upwind FD method, quasi-static solution, and position dependent coupling coefficient
    Ef(2:Nz) = fsync*exp_det(1:Nz-1).*Ef(1:Nz-1) + 1i*dz*kappa(2:Nz).*Er(2:Nz); % Forward Field Propagation
    Er(1:Nz-1) = fsync*exp_det(2:Nz).*Er(2:Nz) + 1i*dz*kappa(2:Nz).*Ef(2:Nz);   % Reverse Field Propagation

    % Output fields recorded at boundaries
    OutputR(i) = Ef(Nz) * (1 - RR); % Right output at z = L (compensated for reflection)
    OutputL(i) = Er(1) * (1 - RL);  % Left output z = 0 (compensated for reflection)

    % FFT data from the outputs
    fftOutputR = fftshift(fft(OutputR)); % Get FFT data for OutputR
    fftOutputL = fftshift(fft(OutputL)); % Get FFT data for OutputL
    fftInputL = fftshift(fft(InputL)); % Get FFT data for OutputL
    omega = fftshift(wspace(time));

    if mod(i,plotN) == 0       % Only executed when i is multiple of plotN

        % Forward Propagation of the Gaussian Pulse
        subplot(3,2,1)
        plot(z*10000,real(Ef),'r'); hold on
        plot(z*10000,imag(Ef),'r--'); hold off
        xlim(XL*1e4)
        ylim(YL)
        xlabel('z(\mum)')
        ylabel('E_f')
        legend('\Re','\Im')
        hold off

        % Reverse (Reflection) of the Gaussian Pulse
        subplot(3,2,3)
        plot(z*10000, real(Er), 'b'); hold on
        plot(z*10000, imag(Er), 'b--'); hold off
        xlim(XL*1e4)
        ylim(YL)
        xlabel('z(\mum)')
        ylabel('E_r')
        legend('\Re', '\Im')
        hold off
```

```

        % Plot showing when the time when the input and output pulse were detected
        subplot(3,2,5);
        plot(time*1e12, real(InputL), 'r'); hold on
        plot(time*1e12, real(OutputR), 'g');
        plot(time*1e12, real(InputR), 'b');
        plot(time*1e12, real(OutputL), 'm');
        xlim([0, Nt*dt*1e12])
        ylim(YL)
        xlabel('time(ps)')
        ylabel('0')
        legend('Left Input', 'Right Output', 'Right Input', 'Left Output' ...
            , 'Location', 'east')
        hold off

        % Plot the spectral content of the Gaussion (or Gaussian Modulated) Pulse
        subplot(3,2,2);
        plot(omega, abs(fftOutputR));
        hold on; %
        plot(omega, abs(fftOutputL));
        plot(omega, abs(fftInputL));
        hold off; %
        xlabel('Frequency (THz)');
        ylabel('|E|');
        xlim([-0.1e14, 0.1e14]);
        legend('fftOutputR','fftOutputL','fftInputL');

        % Plot the Phase of the Gaussion (or Gaussian Modulated) Pulse
        subplot(3,2,4);
        phase1 = unwrap(angle(fftOutputR));  % Unwrap the phase1
        phase2 = unwrap(angle(fftOutputL));  % Unwrap the phase2
        phase3 = unwrap(angle(fftInputL));  % Unwrap the phase for Input

        plot(omega, phase1);
        hold on;
        plot(omega, phase2);
        plot(omega, phase3);
        hold off;
        xlabel('Frequency (THz)');
        ylabel('Phase (E)');
        xlim([-1.5e14, 1.5e14]);
        legend('fftOutputR','fftOutputL','fftInputL');

        pause(0.01)
    end
end
```

Listing 5: MATLAB code for TWM simulation with Gratings and the use of $\hat{\kappa}$

7.6 Section F: Milestone 4 – Gain/Loss Dispersion

```
set(0,'defaultaxesfontsize',20)
set(0,'DefaultFigureWindowStyle','docked')
set(0,'DefaultLineLineWidth',2)
set(0,'Defaultaxeslinewidth',2)

set(0,'DefaultFigureWindowStyle','docked')

c_c = 299792458;                % m/s TWM speed of light
c_eps_0 = 8.8542149e-12;        % F/m vacuum permittivity
c_eps_0_cm = c_eps_0/100;       % F/cm vacuum permittivity
c_mu_0 = 1/c_eps_0/c_c^2;       % Permeability of free space
c_q = 1.60217653e-19;           % Charge of an electron
c_hb = 1.05457266913e-34;       % Dirac / Reduced Planck constant
c_h = c_hb*2*pi;                % Planck constant

beta_r = 0;                     % De-tuning constant
beta_i = 0;                     % Gain constant

kappa0 = 0;                     % Coupling coefficient
kappaStart = 1/3;               % Constant defines starting position where coupling begins
kappaStop = 2/3;                % Constant defines ending position where coupling stops
```

```
InputParasL.E0 = 1e5;           % Amplitude of the input E-field / E_f
InputParasL.we = 0;             % Frequency of the complex sinusoidal modulation on the Gaussian pulse
InputParasL.t0 = 2e-12;         % Time shift constant for the Gaussian pulse
InputParasL.wg = 5e-13;         % Width of the Gaussian distribution
InputParasL.phi = 0;            % Initial phase of the input E-field
InputParasR = 0;                % Placeholder variable for reverse propagation

n_g = 3.5;                      % Constant to control group velocity
vg = c_c/n_g * 1e2;             % TWM cm/s group velocity

Lambda = 1550e-9;               % Wavelength of light

plotN = 10;                     % Divisor constant for plotting frequency

L = 1000e-6*1e2;                % Length of the waveguide in cm

% Material Polarization Information
g_fwhm = 3.5e+012/10;           % Frequency full width at half maximum
LGamma = g_fwhm * 2 * pi;       % Polarization decay rate
Lw0 = 0;                        % Resonance frequency offset
LGain = 0.01;                   % Gain constant

XL = [0, L];                    % Start and end of the x-axis
YL = [-InputParasL.E0, InputParasL.E0]; % Start and end of the y-axis

Nz = 500;                       % Number of spatial points
dz = L/(Nz-1);                  % Distance between each spatial point
dt = dz/vg;                     % Time step ensuring Gaussian pulse stability
fsync = dt*vg/dz;               % Equals 1, ensures Gaussian stability

Nt = floor(2*Nz);               % Number of time steps
tmax = Nt*dt;                   % Maximum simulation time
t_L = dt*Nz;                    % Time for wave to travel the waveguide length
z = linspace(0, L, Nz);         % Spatial grid with Nz points
time = nan(1, Nt);              % Time vector with Nt elements

InputL = nan(1, Nt);            % Left input field over time
InputR = nan(1, Nt);            % Right input field over time
OutputL = nan(1, Nt);           % Left output field over time
OutputR = nan(1, Nt);           % Right output field over time
Ef = zeros(size(z));            % Forward field along the waveguide (initialized to zero)
Er = zeros(size(z));            % Backward field along the waveguide (initialized to zero)

Ef1 = @SourceFct;               % Handle to the source function for forward input
ErN = @SourceFct;               % Handle to the source function for backward input

t = 0;                          % Set initial time to 0
time(1) = t;                    % Initialize the first element of the time vector

InputL(1) = Ef1(t, InputParasL);  % Initial left input field from source function
InputR(1) = ErN(t, InputParasR);  % Initial right input field from source function

OutputR(1) = Ef(Nz);            % Initial right output field (at z = L)
OutputL(1) = Er(1);             % Initial left output field (at z = 0)

Ef(1) = InputL(1);              % Initializes forward field at z = 0
Er(Nz) = InputR(1);             % Initializes backward field at z = L

kappa = kappa0 * ones(size(z)); % Creates an array filled with kappa0 values
kappa(z < L*kappaStart) = 0;    % Sets kappa to zero before interaction region
kappa(z > L*kappaStop) = 0;     % Sets kappa to zero after interaction region

Pf = zeros(size(z));            % Forward field material polarization
Pr = zeros(size(z));            % Reverse field material polarization

% Variables to hold field and polarization information from the previous time step
Efp = Ef;
Erp = Er;
Pfp = Pf;
Prp = Pr;

figure('name', 'Fields')

subplot(3,1,1)
plot(z*1000, real(Ef), 'r');    % Plot of initial forward field
xlabel('z(\mum)')
ylabel('E_f')
```

```
subplot(3,1,2)
plot(z*1000, real(Er), 'b');    % Plot of initial backward field
xlabel('z(\mum)')
ylabel('E_r')

subplot(3,1,3)
plot(time*1e12, real(InputL), 'r'); hold on
plot(time*1e12, real(OutputR), 'r--');
plot(time*1e12, real(InputR), 'b'); hold on
plot(time*1e12, real(OutputL), 'b--');
xlabel('time(ps)')
ylabel('E')
hold off

for i = 2:Nt                    % Loop over time steps

    t = dt*(i-1);               % Update current time
    time(i) = t;                % Store current time

    RL = 0;                     % Left side reflection coefficient
    RR = 0;                     % Right side reflection coefficient

    beta = ones(size(z)) * (beta_r + 1i * beta_i); % Complex propagation constant
    exp_det = exp(-1i * dz * beta);                % Phase shift due to propagation over dz

    % Input fields at current time step
    InputL(i) = Ef1(t, InputParasL);
    InputR(i) = ErN(t, InputParasR);

    % Apply boundary conditions with reflections
    Ef(1) = InputL(i) + RL * Er(1);     % Boundary condition at z = 0
    Er(Nz) = InputR(i) + RR * Ef(Nz);   % Boundary condition at z = L

    % Propagation using upwind finite difference method and position-dependent coupling
    Ef(2:Nz) = fsync * exp_det(1:Nz-1) .* Ef(1:Nz-1) + 1i * dz * kappa(2:Nz) .* Er(2:Nz);
    Er(1:Nz-1) = fsync * exp_det(2:Nz) .* Er(2:Nz) + 1i * dz * kappa(1:Nz-1) .* Ef(1:Nz-1);

    % Boundary conditions for polarization
    Pf(1) = 0;     % Zero polarization at left boundary
    Pf(Nz) = 0;    % Zero polarization at right boundary
    Pr(1) = 0;     % Zero polarization at right boundary
    Pr(Nz) = 0;    % Zero polarization at left boundary

    Cw0 = -LGamma + 1i * Lw0;  % Complex response function of the material

    % Dispersion calculations
    Tf = LGamma * Ef(1:Nz-2) + Cw0 * Pfp(2:Nz-1) + LGamma * Efp(1:Nz-2);
    Pf(2:Nz-1) = (Pfp(2:Nz-1) + 0.5 * dt * Tf) ./ (1 - 0.5 * dt * Cw0);

    Tr = LGamma * Er(3:Nz) + Cw0 * Prp(2:Nz-1) + LGamma * Erp(3:Nz);
    Pr(2:Nz-1) = (Prp(2:Nz-1) + 0.5 * dt * Tr) ./ (1 - 0.5 * dt * Cw0);

    Ef(2:Nz-1) = Ef(2:Nz-1) - LGain * (Ef(2:Nz-1) - Pf(2:Nz-1));  % Forward field adjustment
    Er(2:Nz-1) = Er(2:Nz-1) - LGain * (Er(2:Nz-1) - Pr(2:Nz-1));  % Reverse field adjustment

    % Output fields recorded at boundaries
    OutputR(i) = Ef(Nz) * (1 - RR);
    OutputL(i) = Er(1) * (1 - RL);

    % FFT for spectral analysis
    fftOutput1 = fftshift(fft(OutputR));
    fftOutput2 = fftshift(fft(OutputL));
    fftInput1 = fftshift(fft(InputL));
    omega = fftshift(wspace(time));

    if mod(i, plotN) == 0       % Plot every plotN iterations

        % Forward Gaussian pulse propagation
        subplot(3,2,1)
        plot(z*10000, real(Ef), 'r'); hold on
        plot(z*10000, imag(Ef), 'r--'); hold off
        xlim(XL * 1e4)
        ylim(YL)
        xlabel('z(\mum)')
        ylabel('E_f')
        legend('\Re','\Im')

```

```
        % Reverse (reflected) Gaussian pulse
        subplot(3,2,3)
        plot(z*10000, real(Er), 'b'); hold on
        plot(z*10000, imag(Er), 'b--'); hold off
        xlim(XL * 1e4)
        ylim(YL)
        xlabel('z(\mum)')
        ylabel('E_r')
        legend('\Re', '\Im')

        % Time domain input and output fields
        subplot(3,2,5)
        plot(time*1e12, real(InputL), 'r'); hold on
        plot(time*1e12, real(OutputR), 'g');
        plot(time*1e12, real(InputR), 'b');
        plot(time*1e12, real(OutputL), 'm');
        xlim([0, Nt*dt*1e12])
        ylim(YL)
        xlabel('time(ps)')
        ylabel('E')
        legend('Left Input', 'Right Output', 'Right Input', 'Left Output', 'Location', 'east')

        % Frequency spectrum magnitude
        subplot(3,2,2)
        plot(omega, abs(fftOutput1)); hold on
        plot(omega, abs(fftOutput2));
        plot(omega, abs(fftInput1)); hold off
        xlabel('Frequency (THz)')
        ylabel('|E|')
        xlim([-0.1e14, 0.1e14])
        legend('fftOutput1','fftOutput2','fftInput1')

        % Frequency spectrum phase
        subplot(3,2,4)
        plot(omega, unwrap(angle(fftOutput1))); hold on
        plot(omega, unwrap(angle(fftOutput2)));
        plot(omega, unwrap(angle(fftInput1))); hold off
        xlabel('Frequency (THz)')
        ylabel('Phase(E)')
        xlim([-1.5e14, 1.5e14])
        legend('fftOutput1','fftOutput2','fftInput1')

        pause(0.01)  % Pause for real-time visualization
    end

    % Update previous values for next iteration
    Efp = Ef;
    Erp = Er;
    Pfp = Pf;
    Prp = Pr;
end
```

Listing 6: MATLAB code for TWM simulation with Gain/Loss Dispersion

7.7 Section G: Milestone 5 – Investigation of effects of parameters ω_0 and γ

```
set(0,'defaultaxesfontsize',20)
set(0,'DefaultFigureWindowStyle','docked')
set(0,'DefaultLineLineWidth',2);
set(0,'Defaultaxeslinewidth',2)

set(0,'DefaultFigureWindowStyle','docked')

c_c = 299792458;                 % m/s TWM speed of light
c_eps_0 = 8.8542149e-12;         % F/m vaccum permittivity
c_eps_0_cm = c_eps_0/100;        % F/cm vaccum permittivity
c_mu_0 = 1/c_eps_0/c_c^2;        % Permiability of free space
c_q = 1.60217653e-19;            % Charge of an electon
c_hb = 1.05457266913e-34;        % Dirac / Reduced Planck constant
c_h = c_hb*2*pi;                 % Planck constant
```

```
beta_r = 0;                       % De-tuning constant
beta_i = 0;                       % Gain Constant

kappa0 = 0;                     % Coupling coefficient
kappaStart = 1/3;                 % Constant defines starting position where coupling begins.
kappaStop = 2/3;                  % Constant defines ending position where coupling stops.

InputParasL.E0 = 1e5;             % Amplitude of the input E-field / E_f
InputParasL.we = 0;               % Frequency of the complex sinusoidal modulation on the gaussian pulse
InputParasL.t0 = 2e-12;           % The constant we are shifting the time by
InputParasL.wg = 5e-13;           % Width of the Gaussian distribution
InputParasL.phi = 0;              % Initial Phase of the E_f / input E-field
InputParasR = 0;                  % Placeholder variable for reverse propagation

n_g = 3.5;                        % Constant to control group velocity
vg = c_c/n_g *1e2;                % TWM cm/s group velocity

Lambda = 1550e-9;                 % Wavelength of light

plotN = 10;                       % Divisior constant

L = 1000e-6*1e2;                  % length of the waveguide in cm

% Material Polarization Information
g_fwhm = 3.5e+012/10;             % Frequency
LGamma = g_fwhm*2*pi;
Lw0 = 0;
LGain = 0.01;                     % Gain Constant

XL = [0,L];                       % Start and End of the x-axis
YL = [-InputParasL.E0,InputParasL.E0];       % Start and End of the y-axis

Nz = 500;                         % Number of divisions
dz = L/(Nz-1);                    % Distance between every point
dt = dz/vg;                       % Time taken to plot every point
fsync = dt*vg/dz;                 % Equals 1, allows the Gaussian to be stable

Nt = floor(2*Nz);                 % Time steps
tmax = Nt*dt;                     % Maximum time for simulation
t_L = dt*Nz;                      % time to travel length
z = linspace(0,L,Nz);             % Nz points, Nz-1 segments
time = nan(1,Nt);                 % Time matrix with 1 row and Nt columns / row vector of Nt elements

InputL = nan(1,Nt);               % Matrix with 1 row and Nt columns / row vector of Nt elements
InputR = nan(1, Nt);              % Matrix with 1 row and Nt columns / row vector of Nt elements
OutputL = nan(1,Nt);              % Matrix with 1 row and Nt columns / row vector of Nt elements
OutputR = nan(1,Nt);              % Matrix with 1 row and Nt columns / row vector of Nt elements
Ef = zeros(size(z));              % Matrix with the same dimensions as z, all elements initialized to 0
Er = zeros(size(z));              % Matrix with the same dimensions as z, all elements initialized to 0

Ef1 = @SourceFct;                 % Reference to SourceFct
ErN = @SourceFct;                 % Reference to SourceFct

t = 0;                            % Set t to a starting value of 0
time(1) = t;                      % Sets the first element of the time vector to 0

InputL(1) = Ef1(t, InputParasL);  % Set initial value of InputL using the source function
InputR(1) = ErN(t, InputParasR);  % Set initial value of InputR using the source function

OutputR(1) = Ef(Nz);              % The end of the waveguide is the first value of the reflection (Right to Left)
OutputL(1) = Er(1);               % The end of the waveguide is the first value of the reflection (Left to Right)

Ef(1) = InputL(1);                % Initializes forward field at z = 0 (Input signal from the left)
Er(Nz) = InputR(1);               % Initializes backward field at z = L (Input signal from the right)

kappa = kappa0*ones(size(z));   % Creates an array of size z, where all indexes hold a value of kappa0
kappa(z<L*kappaStart) = 0;      % Sets the limit such that kappa is set to zero outside the interaction region
kappa(z>L*kappaStop) = 0;       % Sets the limit such that kappa is set to zero outside the interaction region.

Pf = zeros(size(z));              % Variable for the polarization of the material on the forward field
Pr = zeros(size(z));              % Variable for the polarization of the material on the reverse field

% Variables to hold field and polarization information
Efp = Ef;
Erp = Er;
Pfp = Pf;
Prp = Pr;
```

```

figure('name', 'Fields')

subplot(3,1,1)
plot(z*1000, real(Ef), 'r');
hold off
xlabel('z(\mum)')
ylabel('E_f')

subplot(3,1,2)
plot(z*1000, real(Er), 'b');
xlabel('z(\mum)')
ylabel('E_r')
hold off

subplot(3,1,3)
plot(time*1e12, real(InputL), 'r'); hold on
plot(time*1e12, real(OutputR), 'r--');
plot(time*1e12, real(InputR), 'b'); hold on
plot(time*1e12, real(OutputL), 'b--');
xlabel('time(ps)')
ylabel('E')
hold off

for i = 2:Nt                      % 2 to 1000 in steps of 1

    t = dt*(i-1);
    time(i) = t;

    RL = 0;                                  % The left side reflection coefficient
    RR = 0;                                  % The right side reflection coefficient

    beta = ones(size(z))*(beta_r+1i*beta_i); % Complex propagation constant
    exp_det = exp(-1i*dz*beta);               % Phase shift due to propagation over a distance dz

    % Input
    InputL(i) = Ef1(t, InputParasL);  % At time t, we input a signal characterized by InputParasL from the left
    InputR(i) = ErN(t, 0);            % At time t, we input no signal from the right (since InputParasR = 0)

    % Reflection
    Ef(1) = InputL(i) + RL*Er(1);     % Boundary condition at z = 0 (left side);
    Er(Nz) = InputR(i) + RR*Ef(Nz);   % Boundary condition at z = L (right side);

    Ef(2:Nz) = fsync*exp_det(1:Nz-1).*Ef(1:Nz-1) + 1i*dz*kappa(2:Nz).*Er(2:Nz); % Forward Field Propagation
    Er(1:Nz-1) = fsync*exp_det(2:Nz).*Er(2:Nz) + 1i*dz*kappa(2:Nz).*Ef(2:Nz);   % Reverse Field Propagation

    % Boundary Conditions
    Pf(1) = 0;     % zero polarization at the left boundary
    Pf(Nz) = 0;    % zero polarization at the right boundary
    Pr(1) = 0;     % zero polarization at the right boundary
    Pr(Nz) = 0;    % zero polarization at the left boundary
    Cw0 = -LGamma + 1i * Lw0;         % Defines the complex response function of the material.

    % Dispersion Calculations
    % Backward Euler Polarization Update
    % Forward polarization
    Tf = LGamma * Efp(2:Nz-1);  % Source term for forward polarization
    Pf(2:Nz-1) = (Pfp(2:Nz-1) + dt * Tf) ./ (1 - dt * Cw0);  % Forward polarization update

    % Backward polarization
    Tr = LGamma * Erp(2:Nz-1);  % Source term for backward polarization
    Pr(2:Nz-1) = (Prp(2:Nz-1) + dt * Tr) ./ (1 - dt * Cw0);  % Backward polarization update

    Ef(2:Nz-1) = Ef(2:Nz-1) - LGain * (Ef(2:Nz-1) - Pf(2:Nz-1));  % Adjusts the forward electric field
    Er(2:Nz-1) = Er(2:Nz-1) - LGain * (Er(2:Nz-1) - Pr(2:Nz-1));  % Adjusts the reverse electric field

    % Output
    OutputR(i) = Ef(Nz) * (1 - RR); % Right output at z = L
    OutputL(i) = Er(1) * (1 - RL);  % Left output at z = 0

    % FFT data from the outputs
    fftOutput1 = fftshift(fft(OutputR)); % Get FFT data for OutputR
    fftOutput2 = fftshift(fft(OutputL)); % Get FFT data for OutputL
    fftInput1 = fftshift(fft(InputL)); % Get FFT data for OutputL
    omega = fftshift(wspace(time));

    if mod(i,plotN) == 0       % Only executed when i is multiple of plotN

```

```matlab
        % Forward Propagation of the Gaussian Pulse
        subplot(3,2,1)
        plot(z*10000,real(Ef),'r'); hold on
        plot(z*10000,imag(Ef),'r--'); hold off
        xlim(XL*1e4)
        ylim(YL)
        xlabel('z(\mum)')
        ylabel('E_f (V/um)')
        legend('\Re','\Im')
        hold off

        % Reverse (Reflection) of the Gaussian Pulse
        subplot(3,2,3)
        plot(z*10000, real(Er), 'b'); hold on
        plot(z*10000, imag(Er), 'b--'); hold off
        xlim(XL*1e4)
        ylim(YL)
        xlabel('z(\mum)')
        ylabel('E_r (V/um)')
        legend('\Re', '\Im')
        hold off

        % Plot showing when the time when the input and output pulse were detected
        subplot(3,2,5);
        plot(time*1e12, real(InputL), 'r'); hold on
        plot(time*1e12, real(OutputR), 'g');
        plot(time*1e12, real(InputR), 'b');
        plot(time*1e12, real(OutputL), 'm');
        xlim([0, Nt*dt*1e12])
        ylim(YL)
        xlabel('time(ps)')
        ylabel('E (V/um)')
        legend('Left Input', 'Right Output', 'Right Input', 'Left Output' ...
            , 'Location', 'east')
        hold off

        % Plot showing the spectral content of OutputR and OutputL
        subplot(3,2,2);
        plot(omega, abs(fftOutput1));
        hold on; %
        plot(omega, abs(fftOutput2));
        plot(omega, abs(fftInput1));
        hold off; %
        xlabel('Frequency (THz)');
        ylabel('|E| (V/um)');
        xlim([-0.1e14, 0.1e14]);
        legend('fftOutputR','fftOutputL','fftInputL');

        % Plot showing the phase of OutputR and OutputL
        subplot(3,2,4);
        phase1 = unwrap(angle(fftOutput1));  % Unwrap the phase1
        phase2 = unwrap(angle(fftOutput2));  % Unwrap the phase2
        phase3 = unwrap(angle(fftInput1));  % Unwrap the phase for Input

        plot(omega, phase1);
        hold on;
        plot(omega, phase2);
        plot(omega, phase3);
        hold off;
        xlabel('Frequency (THz)');
        ylabel('Phase (E) (Rads)');
        xlim([-1.5e14, 1.5e14]);
        legend('fftOutputR','fftOutputL','fftInputL');

        pause(0.01)
    end

    % Update Previous Values
    Efp = Ef;
    Erp = Er;
    Pfp = Pf;
    Prp = Pr;
end
```

Listing 7: Investigation of effects of parameters ω_0 and γ

8 References

[1] ELEC 4700 Course Documents, *Travelling Wave Model Project Overview*, Department of Electronics, Carleton University, Ottawa, ON, Canada, 2025.

[2] ELEC 4700 Course Documents, *Milestone 1 – Fundamentals of Wave Propagation*, Department of Electronics, Carleton University, Ottawa, ON, Canada, 2025.

[3] ELEC 4700 Course Documents, *Milestone 2 – Gain, Detuning, and Frequency Domain Analysis*, Department of Electronics, Carleton University, Ottawa, ON, Canada, 2025.

[4] ELEC 4700 Course Documents, *Milestone 3 – Grating Effects and Coupling Coefficients*, Department of Electronics, Carleton University, Ottawa, ON, Canada, 2025.

[5] ELEC 4700 Course Documents, *Milestone 4 – Frequency-Dependent Gain Dispersion and Trapezoidal Integration*, Department of Electronics, Carleton University, Ottawa, ON, Canada, 2025.

[6] OpenAI, "ChatGPT," OpenAI, San Francisco, CA, USA, 2024. [Online]. Available: `https://openai.com/chatgpt`