ELEC 4700 Final Report

Rish Jain, 101162547

June 29, 2025

Contents

1 Introduction and Background

This project focuses on simulating the behavior of optical amplifiers and lasers using the Traveling Wave Model (TWM). The goal is to understand the interaction between optical fields and carrier dynamics inside semiconductor devices, and to explore the transition from optical amplification to lasing under different physical conditions.

The project is divided into multiple milestones, where we progressively add carrier dynamics, stimulated and spontaneous emission, and optical feedback through mirrors to the model. These additions allow us to simulate laser startup, mode competition and steady-state behavior.

The final stages of the project include open-ended investigations that builds upon all the previous iterations. These explore mode evolution during startup, the effects of back reflection with phase shifts, and how different system parameters influence the spectral output of the laser. The simulations help provide insight into real-world behavior of photonic systems, such as mode selection, transient dynamics, and the impact of spontaneous emission in laser startup.

Overall, this project serves as both a numerical and physical exploration of laser physics using simplified but powerful modeling techniques. The findings align well with theoretical expectations and help in understanding how design choices affect the performance and stability of semiconductor-based optical devices.

2 Carrier Equation (Milestone 6)

2.1 Derivation

This report is treated as an extension of the midterm report, so basic derivation from scratch is not included in the report.

Milestone 6 deals with using the Traveling Wave model from Milestone 4 by introducing a carrier equation to model the stimulated emission. The equation that models the carrier equation is given below:

$$\frac{dN(z)}{dt} = \underbrace{\frac{\eta I_D}{qV_l}}_{\text{Source Term}} - \underbrace{G_0(N(z) - N_{\text{tr}})S(z)}_{\text{Stimulated Emission}} - \underbrace{\frac{N(z)}{\tau_n}}_{\text{Spontaneous Emission}} \tag{1}$$

with $S(z) = C_{\text{EtoP}}\left(|\hat{E}_f(z)|^2 + |\hat{E}_r(z)|^2\right)$, C_{EtoP} converts field intensity to photon density.

To observe this in the simulation, this equation needs to be adapted into a form suitable for numerical implementation, i.e., discretize them. In other words, the continuous-time differential equation shown above needs to be converted into a discrete-time differential equation and then implement it using MATLAB code.

The derivation starts with using the back difference approximation Using the backward difference approximation:

$$\frac{dN(z)}{dt} \approx \frac{N_i - N_{i-1}}{\Delta t}$$

Substitute into the equation:

$$\frac{N_i - N_{i-1}}{\Delta t} = \frac{\eta I_D}{qV_l} - G_0(N_{i-1} - N_{\text{tr}})S - \frac{N_i}{\tau_n} \tag{2}$$

Multiply both sides by Δt:

$$N_i - N_{i-1} = \Delta t\left(\frac{\eta I_D}{qV_l} - G_0(N_{i-1} - N_{\text{tr}})S - \frac{N_i}{\tau_n}\right) \tag{3}$$

Expand the right-hand side equation

$$N_i - N_{i-1} = \Delta t\left(\frac{\eta I_D}{qV_l} - G_0(N_{i-1} - N_{\text{tr}})S\right) - \Delta t \cdot \frac{N_i}{\tau_n} \tag{4}$$

Now N_i gets isolated by moving the $\frac{\Delta t}{\tau_n}N_i$ term to the left-hand side:

$$N_i + \frac{\Delta t}{\tau_n}N_i = \Delta t\left(\frac{\eta I_D}{qV_l} - G_0(N_{i-1} - N_{\text{tr}})S\right) + N_{i-1} \tag{5}$$

Factor N_i on the left-hand side:

$$N_i\left(1 + \frac{\Delta t}{\tau_n}\right) = \Delta t\left(\frac{\eta I_D}{qV_l} - G_0(N_{i-1} - N_{\text{tr}})S\right) + N_{i-1} \tag{6}$$

After rearranging, the final expression obtained is:

$$N_i = \frac{N_{i-1} + \Delta t \left(\frac{\eta I_D}{q V_l} - G_0 (N_{i-1} - N_{\text{tr}}) S \right)}{1 + \frac{\Delta t}{\tau_n}} \tag{7}$$

Equation (7) is a fully discretized version of the carrier equation used to model the stimulated emission given by Equation (1). It is important to note that a backward difference approximation was employed because N_i depends on an initial value. This initial condition is required both to start the simulation from the correct physical state and to iteratively compute the carrier density at each subsequent time step.

To put it more simply, N must be defined at the beginning (captured by the N_{i-1} term) and is then updated using Equation (7). As a simple analogy, consider the line of code `a = a + 2;`. This operation only works if `a` already has an initial value (At least in MATLAB). This is the way I understood this.

2.2 Code

Now that the discretized, code-able equations have been derived, they are implemented in MATLAB to verify whether the the method used to discretize them is correct. The code used to represent Equations (7) is given below:

```
N = ones(size(z)) * Ntr;
Nave(1) = mean(N);
gain = v_g * 2.5e-16;
eVol = 1.5e-10 * c_q;
Ion = 0.25e-9;
Ioff = 3e-9;
I_off = 0.024;
I_on = 0.1;
taun = 1e-9;
Zg = sqrt(c_mu_0 / c_eps_0) / n_g;
EtoP = 1 / (Zg * f0 * v_g * 1e-2 * c_hb);
alpha = 0;
```

In the code above, the term `a` appears in the form `N = ones(size(z)) * Ntr;`. This line initializes N and corresponds to N_{i-1} in the discretized equation (7). All other variables are parameters used to set the start and stop times, as well as the magnitudes of I_d (or I_{inj}) and other related quantities.

```
n_g = 3.5;
v_g = c_c / n_g * 1e2;       % TWM cm/s group velocity
Lambda = 1550e-9;            % cm
f0 = c_c / Lambda;
```

This piece of code defines important constants, including the group index, group velocity, central wavelength, and the corresponding frequency.

```
S = (abs(Ef).^2 + abs(Er).^2) .* EtoP * 1e-6;

if t < Ion || t > Ioff
    I_injv = I_off;
else
    I_injv = I_on;
end

Stim = gain .* (N - Ntr) .* S;
N = (N + dt * (I_injv / eVol - Stim)) ./ (1 + dt / taun);
Nave(i) = mean(N);
```

The actual structure of equation (7) is reflected in the lines defining `S`, `Stim`, and `N`. When these lines of code are compared to their corresponding terms in equation (7), they appear to match. This suggests that the derivation is likely correct, although the specific discretization approach used in the derivation of Equation (7) may not be exactly correct since it was done based on assumptions.

2.3 Simulation

The simulation of the physical model from Milestone 6 was carried out, and the resulting plots are shown below:

Response to pulsed I_D with optical sources turned off

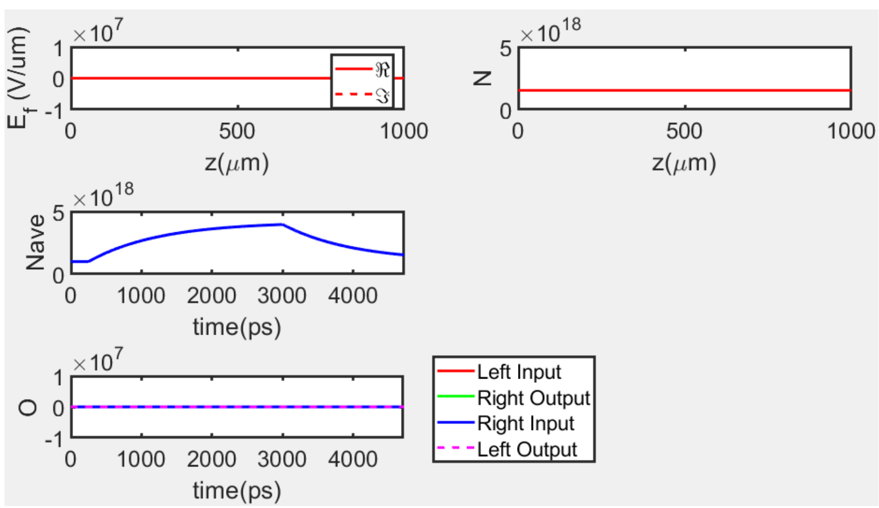

Figure 1: The change in the averaged carrier density with no optical input

Figure 1 illustrates the evolution of carrier density $N(z)$ over time, as represented by the plot of N_{ave} versus time. Since there is no optical input present, the stimulated emission term in the carrier continuity equation is effectively zero. This simplifies the system to a first-order differential equation with a constant source term. As a result, the carrier density exhibits exponential growth of the form $1-\exp(-at)$ when the injection current I_{on} is applied. Around 3000 ps, the current switches to I_{off}, and the source term is removed. The carrier density then decays exponentially as $\exp(-at)$, governed primarily by the recombination lifetime τ_n. This behavior confirms the expected time-dependent dynamics in the absence of photon-induced stimulated emission.

Response to a gaussian pulse stream

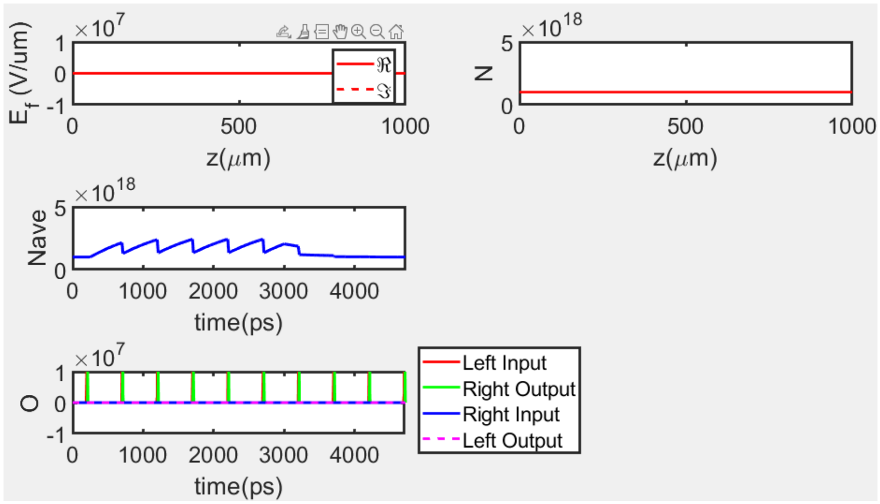

Figure 2: The change in the averaged carrier density with a gaussian optical pulse input

Figure 2 shows the system response when a stream of Gaussian optical pulses is introduced from the left input. These pulses travel through the waveguide and interact with the carriers via stimulated emission, which depletes the carrier density $N(z)$ along the propagation path. This effect is visible in the N_{ave} plot, where the average carrier density exhibits a sawtooth-like pattern: periodic drops coincide with the arrival of each pulse, followed by recovery due to continued carrier injection. Since gain due to stimulated emission has not yet been included in the optical field equations, there is no amplification of the optical pulses; they pass through the device without being amplified, but still consume carriers as they propagate.

2.4 Conclusion

In conclusion, Milestone 6 involved deriving a numerically implementable form of the travelling wave equation with a carrier equation to model the stimulated emission, implementing the derived equation 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.

3 Optical Amplifiers and SPE (Milestone 7)

3.1 Derivation

Milestone 7 deals with using the Traveling Wave model from Milestone 6 (which is built upon Milestone 4) by introducing stimulated and spontaneous emission to the optical equations. The optical equations that model this are given below:

$$\frac{1}{v_g}\frac{\partial \hat{E}_f}{\partial t} = -\frac{\partial \hat{E}_f}{\partial z} - \underbrace{i\hat{\beta}(N,S)\hat{E}_f}_{\text{Stimulated Emission}} + \underbrace{\hat{F}_f}_{\text{Spontaneous Emission}} \tag{8}$$

$$\frac{1}{v_g}\frac{\partial \hat{E}_r}{\partial t} = +\frac{\partial \hat{E}_r}{\partial z} - \underbrace{i\hat{\beta}(N,S)\hat{E}_r}_{\text{Stimulated Emission}} + \underbrace{\hat{F}_r}_{\text{Spontaneous Emission}} \tag{9}$$

with

$$\hat{\beta}(N) = \frac{i}{2}\left(\underbrace{g_f G_0(N - N_{tr})}_{\text{gain}} - \underbrace{\alpha_l}_{\text{loss}}\right) \tag{10}$$

Equations (8) and (9) are essentially the same as those from Milestones 3 and 4, except with some terms removed and others added to reflect the physics introduced in this Milestone. Since this Milestone builds upon the previous ones, whose base equations and derivations were already covered in the Midterm report, so only the components that are new will be discussed here. One of these new equations is the expression for Equation (10), but it does not need to be discretized, as all the variables and constants it depends on are already implemented in the code from earlier Milestones, so only some additional code is added and is discussed in Section 3.2. The other new addition is the inclusion of $\hat{F}_f$ and $\hat{F}_r$, which are related to spontaneous emission. The corresponding expression for the amplitude A, used in the calculations of the spontaneous emission terms, is given below:

$$A = \sqrt{\frac{\gamma\, \beta_{\text{spe}}\, c\, h_b\, f_0 \cdot 10^{-2}\, L}{2N_z\, \tau_n}} \tag{11}$$

$$E_{sF} = eT_f \cdot |\text{SPE}| \cdot \sqrt{N \cdot 10^6} \tag{12}$$

$$E_{sR} = eT_r \cdot |\text{SPE}| \cdot \sqrt{N \cdot 10^6} \tag{13}$$

Here, `eTf` and `eTr` represent arrays of complex-valued random numbers, with each entry generated by combining two independent Gaussian random variables to form the real and imaginary parts.

3.2 Code

As shown in Equations (8) to (10), stimulated emission is introduced using the complex gain term $\hat{\beta}(N)$, which modifies the propagation of the forward and backward fields. The code implementation of this stimulated emission term is given below:

```
beta_r = 0;
gain_z = gain.*(N - Ntr)./v_g;
beta_i = (gain_z - alpha)./2;
beta = beta_r + 1i*beta_i;
```

In this code, since $\alpha_l = 0$ in this case, there is no loss and the complex value of β is used in the field update equations to model the interaction of the fields with the carrier population.

In addition to stimulated emission, spontaneous emission is introduced to model random field fluctuations in both the forward and backward propagating fields. The code below shows how a random electric field is generated and added to both directions:

```
A = sqrt(gamma*beta_spe*c_hb*f0*L*1e-2/taun/(2*Nz));
if SPE > 0
    eTf = (randn(Nz,1)+1i*randn(Nz,1))*A;
    eTr = (randn(Nz,1)+1i*randn(Nz,1))*A;
else
    eTf = (ones(Nz,1))*A;
    eTr = (ones(Nz,1))*A;
end

EsF = eTf+abs(SPE).*sqrt(N.*1e6);
EsR = eTr+abs(SPE).*sqrt(N.*1e6);
```

This block corresponds to the spontaneous emission terms $\hat{F}_f$ and $\hat{F}_r$ in Equations (8) and (9), where complex random numbers are scaled by the amplitude A from Equation (11). The resulting complex vectors `eTf` and `eTr` are added to variables EsR and EsF, and these variables add variation to the actual field solution of the field in the forward and reverse directions. The value of A depends on both physical constants and simulation parameters.

Finally, the total electric fields are updated by summing the variating field to the actual field solution as seen below.

```
Ef = Ef + EsF;
Er = Er + EsR;
```

These updated fields E_f and E_r now represent the full behavior of the electromagnetic wave under the contributions of amplification and spontaneous variation.

The constants used for controlling the spontaneous emission level are defined separately:

```
beta_spe = .3e-5;
gamma = 1.0;
SPE = 7;
```

The above code set the values for important parameters that control how the code and the system behaves.

3.3 Simulation

Discuss the role of the SPE parameter and the power distribution in the modes

The `SPE` parameter controls how much spontaneous emission is added to the simulation. When `SPE` is greater than zero, the model adds random noise that mimics the physical process of spontaneous emission. These noise fields are amplified according to the gain profile and gain polarization, which determine which frequency components grow and which ones don't. The injected noise spreads across multiple modes, but only some modes get amplified significantly, showing how power is naturally distributed among modes during laser startup.

Describe what `randn` does and what role it's playing

The `randn` function generates random numbers from a standard normal distribution. In this context, it is used to create the real and imaginary parts of the spontaneous emission field by generating two independent random arrays. These arrays are multiplied by a scaling factor and used to build a complex noise field that is added to both the forward and backward propagating optical fields. Since the random values are different every time the code runs, the resulting spontaneous emission field varies between simulations, which matches the unpredictable nature of spontaneous emission in real lasers. This noise plays an important role in initiating the optical field when there's no external input, allowing the model to simulate laser startup from random fluctuations, just as in physical devices.

The simulation of the physical model from Milestone 7 was carried out, and the resulting plots are shown below:

Amplification of a pulse stream with no SPE

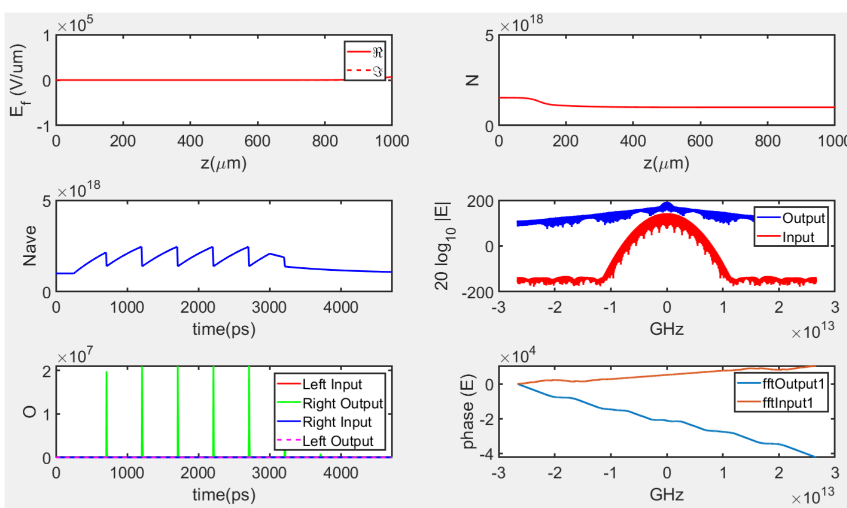

Figure 3: The amplification of the Gaussian pulse stream input with SPE turned off

Figure 3 shows a case where the input Gaussian pulse stream is getting amplified, as seen by the magnitude of the green pulses with the SPE turned off and its associated effect on the average carrier density. The FFT plots show the spectral characteristics of the input and output. It can be seen from the gain plot that the output (blue) has higher and broader frequency content than the input (red), indicating amplification and spectral broadening.

Amplification of SPE with no input source with gain polarization

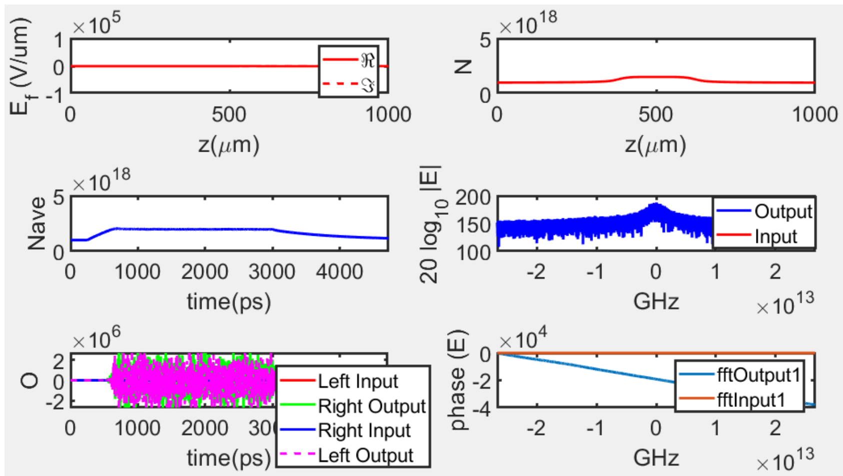

Figure 4: The change in the averaged carrier density with no optical input

Figure 4 depicts the amplification of random spontaneous emission with no optical pulse input and a gain polarization value $L_{\text{Gain}} = 0.05$. From the bottom left plot, we observe that the magnitude reaches approximately 2×10^6, with the spectral plot showing a distinct peak near the center. This indicates that the random spontaneous emission is being amplified, and a certain band is amplified more than the others.

Amplification of SPE with no input source without gain polarization

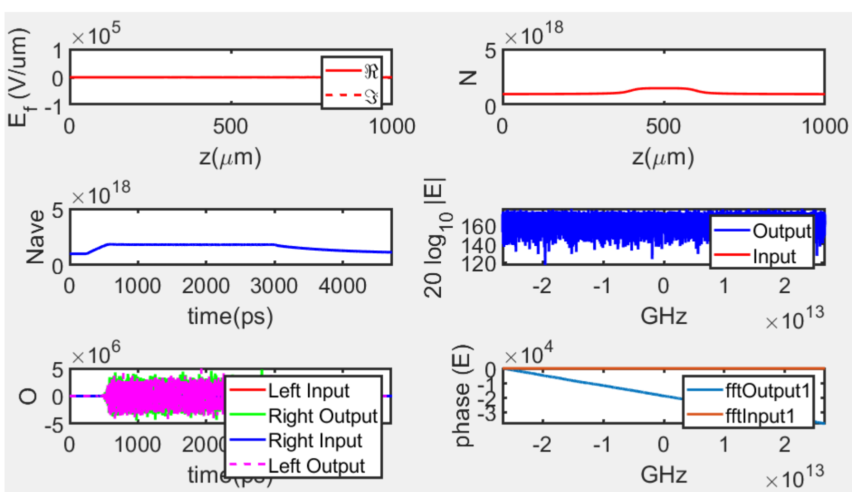

Figure 5: The change in the averaged carrier density with no optical input

Figure 5, on the contrary, uses the same parameters and has no input, similar to the setup in Figure 4, except that there is no gain polarization. This effect can be seen in the bottom left plot, where the magnitude reaches around 5×10^6, but no distinct peak is observed in the spectral response. This suggests that when gain polarization is absent, the system distributes power more evenly across the band. In contrast, when gain polarization is present, the system preferentially amplifies a certain band of frequencies, while the other bands are not amplified as much.

Effect of I_D and Input Pulse Configuration

For this part, the value of I_D is varied through the variable `I_on`.

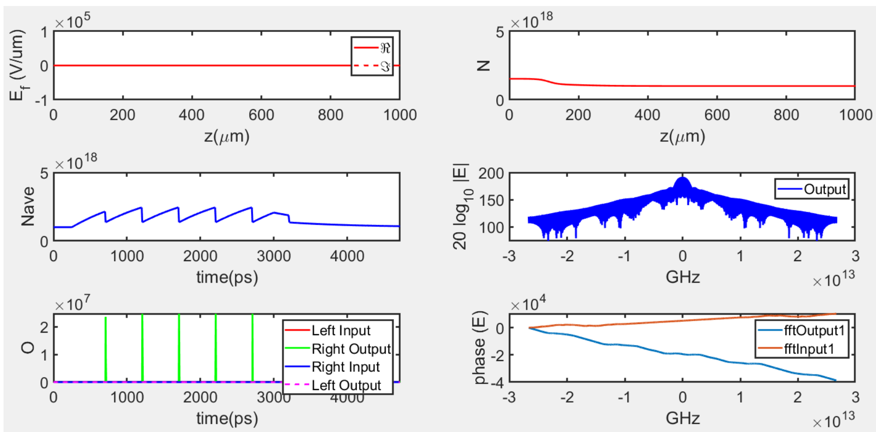

Figure 6: The response of the system to an I_D of 0.1 with an optical input

Figure 6 shows the simulation where the optical pulse's width (`wg`) was 10^{-13} and $I_D = 0.1$. From the spectral response, we observe a well-defined peak with significant amplification. The peak is relatively broadband, and there is at least one lower-power but still clearly defined sideband peak.

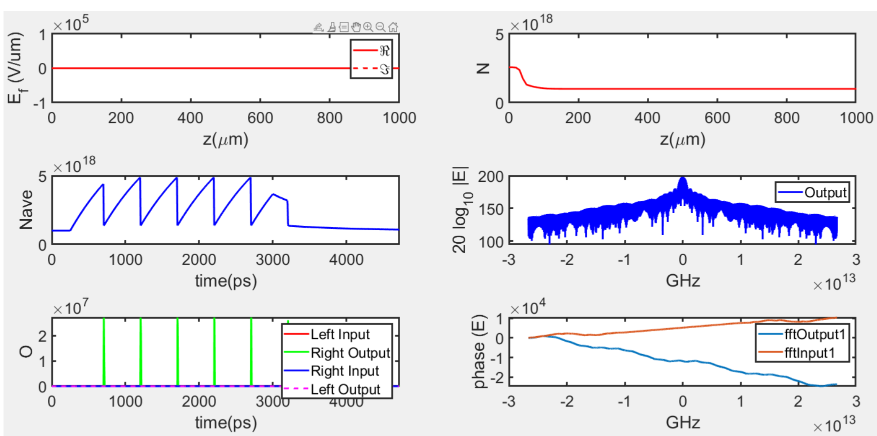

Figure 7: The response of the system to an I_D of 0.25 with an optical input

Figure 7 shows the simulation where the optical pulse's width (`wg`) was 10^{-13} and $I_D = 0.25$. From the spectral response, we observe a well-defined peak with strong amplification, though

the peak is narrower but higher power compared to that in Figure 6. Additionally, at least three sideband peaks with lower power but relatively well-defined shapes emerge.
These results indicate that increasing I_D narrows the primary amplification band while increasing its magnitude. Moreover, more sideband peaks begin to appear as I_D increases.

For this part, the value of I_D is set back to 0.1 but the width of the Gaussin pulse is increased through the variable `wg`.

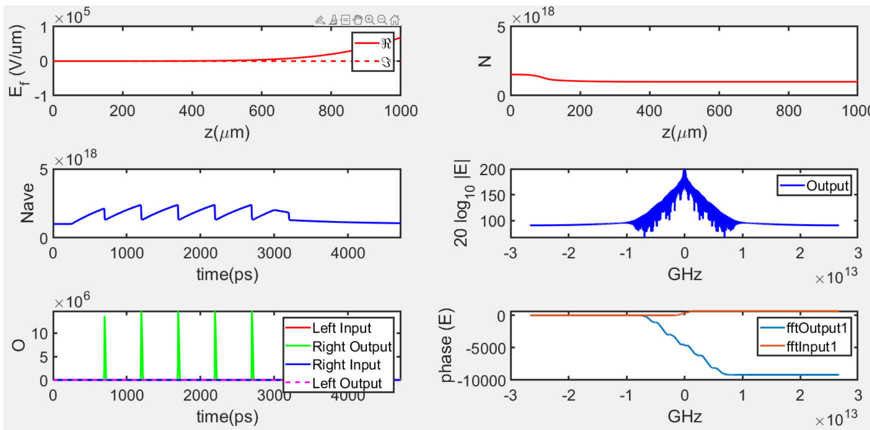

Figure 8: The response of the system when a wider Gaussian optical pulse is input

Figure 8 shows the response of the system when a Gaussian pulse with a width of 50×10^{-13} is input into the system. Compared to Figure 6, where the Gaussian pulse width is 10×10^{-13}, the spectral response in Figure 8 shows that only a very narrow band of frequencies is amplified, while frequencies outside this band are not amplified as much. The envelope of the magnitude of the frequencies beyond the significantly amplified band appears to drop almost linearly. This indicates that as the input optical pulse becomes wider, the frequency band experiencing significant amplification becomes narrower and outside this band, the frequencies exhibit significantly lower power and amplification.

3.4 Conclusion

In conclusion, Milestone 7 involved adding new terms for stimulated and spontaneous emission into the optical equations, implementing these updated equations in MATLAB, and generating simulation results in the form of plots. Both the derivation and the code were verified to be correct, as the simulation results behaved as expected and aligned with the theoretical predictions.

4 Lasers (Milestone 8)

4.1 Derivation

Milestone 8 builds on the implementation of stimulated and spontaneous emission from Milestone 7 by introducing optical feedback through mirrors. This is essential to transition from an optical amplifier to a functioning laser.The only change to the equations from Milestone 7 is the addition of boundary conditions to simulate reflection. These are defined using the reflectivities R_l and R_r for the left and right mirrors, respectively. For this milestone, both are set to 0.5. Spontaneous emission (`SPE`) is necessary to initiate the lasing process in the absence of external optical input. These random fluctuations are amplified through the gain medium and reflected by the mirrors and lasing occurs if the gain exceeds the loss. Initially, many longitudinal modes may appear during the transient startup, but the system settles into a steady state dominated by few modes near the center of the gain bandwidth.

4.2 Code

The required code changes are minimal. No new terms are added to the equations; only the boundary conditions are updated to reflect the addition of mirrors. This is done by setting:

```
Rf = 0.5;
Rr = 0.5;
```

This sets the reflectivity of both ends to 0.5. Additionally, ensuring that `SPE > 0` allows spontaneous emission to trigger the lasing process. The rest of the code for field updates and gain remains unchanged from Milestone 7.

4.3 Simulation

The simulation of the physical model from Milestone 8 was carried out, and the resulting plot are shown below:

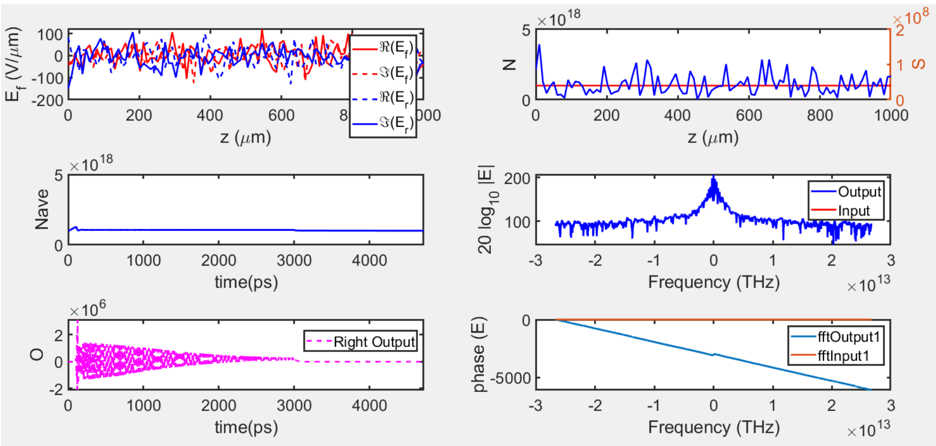

Figure 9: The systems response to spontaneous emission when mirrors are added

Figure 9 shows the field spontaneously building up due to random spontaneous emission, amplified and sustained by feedback from the mirrors. A steady-state mode structure emerges after initial transients.

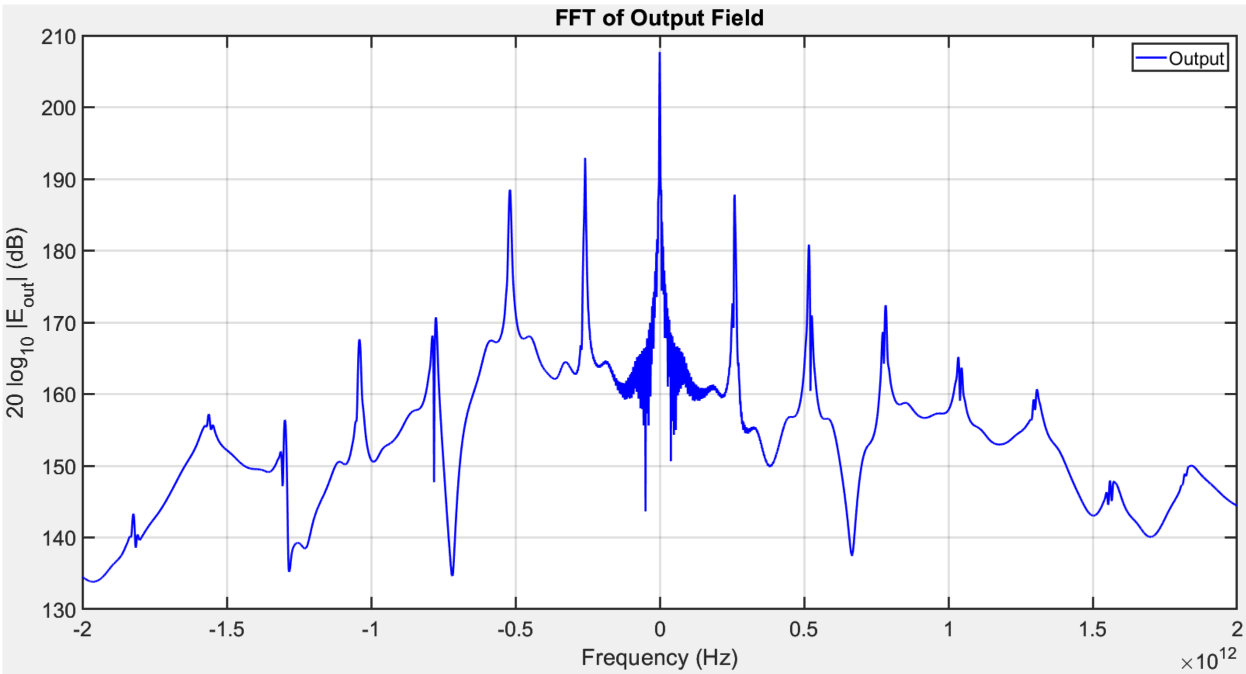

Figure 10: Closer look at the spectral response of the system by reducing the bounds of the FFT plot in Figure 9

Figure 10 shows a closer view of the spectral response by narrowing the x-axis limits of the FFT plot in Figure 9. This reveals well-defined peaks, which correspond to the modes in this laser simulation run. These modes indicate the frequencies where the power is concentrated and most strongly amplified.

If you set SPE to a negative value it will be non-random! How is this done in the code? Would you say it is a good way to do it?

In the code, the variable `SPE` controls whether spontaneous emission noise is random or not. When `SPE > 0`, the noise is generated using `randn`, which produces random values. However, if `SPE` is set to a negative value, the `else` part of the condition is triggered, and fixed values are used instead of random numbers. This removes the randomness from the spontaneous emission term, making it deterministic.

Using the sign of `SPE` to switch behavior is one way to do it, but it can be confusing because there is no such thing as negative spontaneous emission. A clearer way to implement this would be to use a flag: if the flag is set to `true`, the emission is random; if set to `false`, it is deterministic.

Contrast constant and stochastic runs

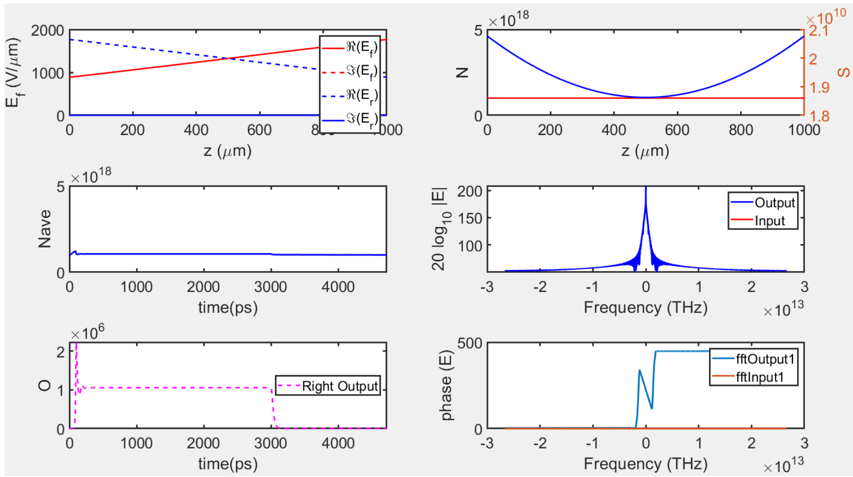

Figure 11: System's response to a deterministic spontaneous emission when mirrors are added

Figure 11 shows the result of a constant run where `SPE = -7`. This results in a repeatable simulation with a consistent optical field, spectrum, and phase. The output's FFT is symmetric, showing a narrow dominant mode due to the deterministic conditions.

In summary, stochastic runs simulate physical randomness in laser startup, while constant runs provide controlled, repeatable conditions useful for analysis.

Look at the modes at SS and during transient using the FFT. What can you say?

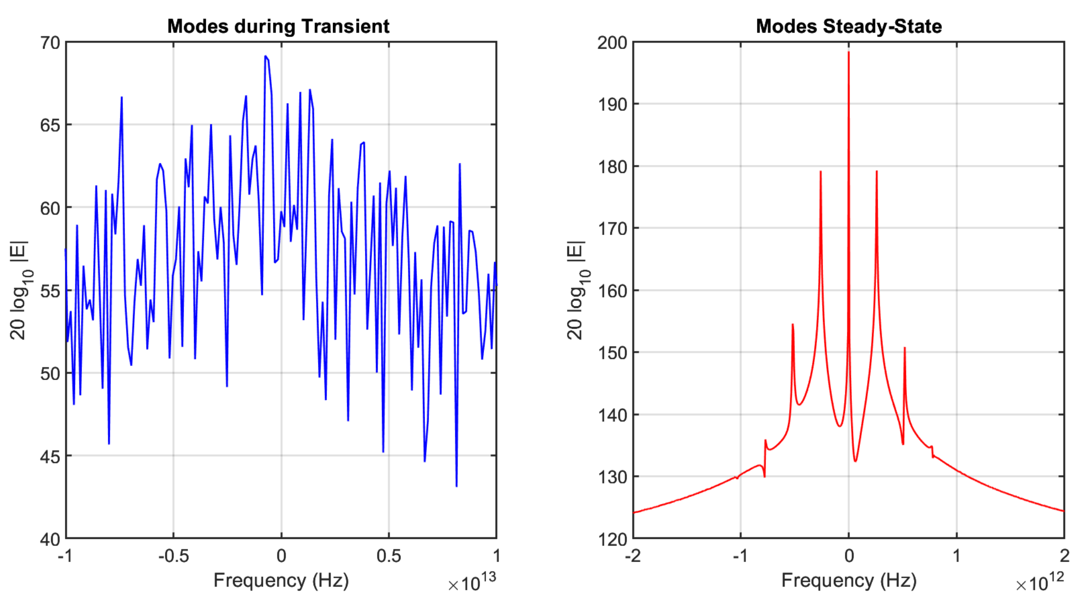

Figure 12: Comparison of the laser output spectrum during transient and steady-state

From Figure 12, we observe that during the transient phase (left plot), there are many spectral peaks, indicating that energy is spread across a wide range of modes. However, in the steady-state phase (right plot), only three well-defined peaks remain, corresponding to the dominant modes of the laser system. In other words, during startup or the transient period, noise distributes power across many frequencies. As the system evolves and reaches steady state, only the frequencies that best match the laser's gain profile are amplified. These become the modes of the system. The code used to obtain the above plot is given below:

```
transient_range = 1:round(0.009*Nt);
steady_range = round(0.3*Nt):round(0.5*Nt);
output_transient = OutputR(transient_range);
output_steady = OutputR(steady_range);
time_transient = time(transient_range);
time_steady = time(steady_range);
omega_transient = fftshift(wspace(time_transient));
omega_steady = fftshift(wspace(time_steady));
fft_transient = fftshift(fft(output_transient));
fft_steady = fftshift(fft(output_steady));
figure('Name', 'Transient vs Steady-State', 'Color', 'w');
subplot(1,2,1);
plot(omega_transient, 20*log10(abs(fft_transient)), 'b');
xlabel('Frequency (Hz)');
ylabel('20 log_{10} |E|');
title('Modes during Transient ');
grid on;
xlim([-0.1e14 0.1e14]);
subplot(1,2,2);
plot(omega_steady, 20*log10(abs(fft_steady)), 'r');
xlabel('Frequency (Hz)');
ylabel('20 log_{10} |E|');
title('Modes Steady-State');
grid on;
xlim([-0.02e14 0.02e14]);
```

Change the current ID . What is the effect?

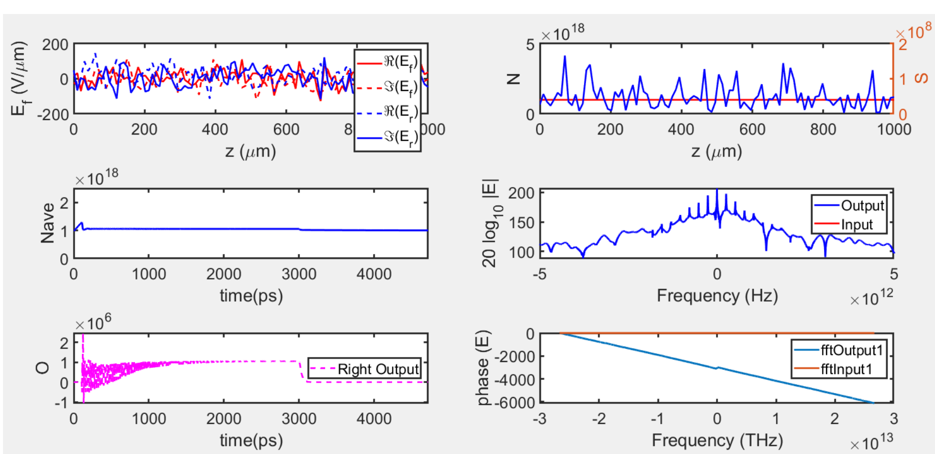

Figure 13: The lasing of the laser when the I_D is set to 0.1

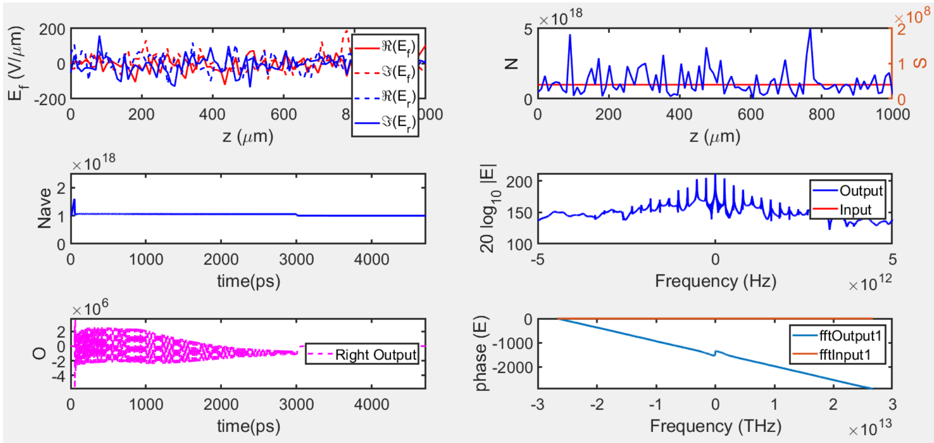

Figure 14: The lasing of the laser when the I_D is set to 0.25

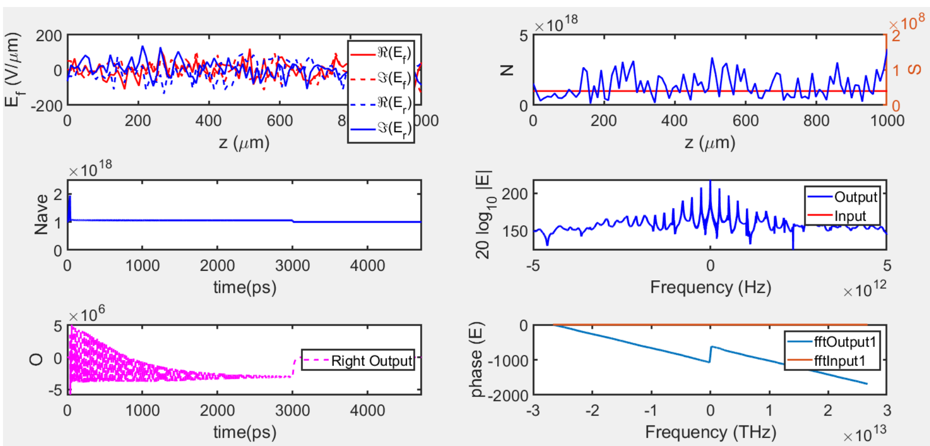

Figure 15: The lasing of the laser when the I_D is set to 0.7

Figures 13, 14, and 15 show the lasing behavior of the system at different values of I_D. As I_D increases, not only does the magnitude of the dominant mode increase, but the number of lasing modes also grows. Additionally, the phase plots reveal a shift near the 0 Hz mark. Specifically, as I_D increases, the phase near 0 Hz approaches zero phase more and more before the phase begins to diverge again.

What happens if you change the laser length? Keep Δz approximately the same

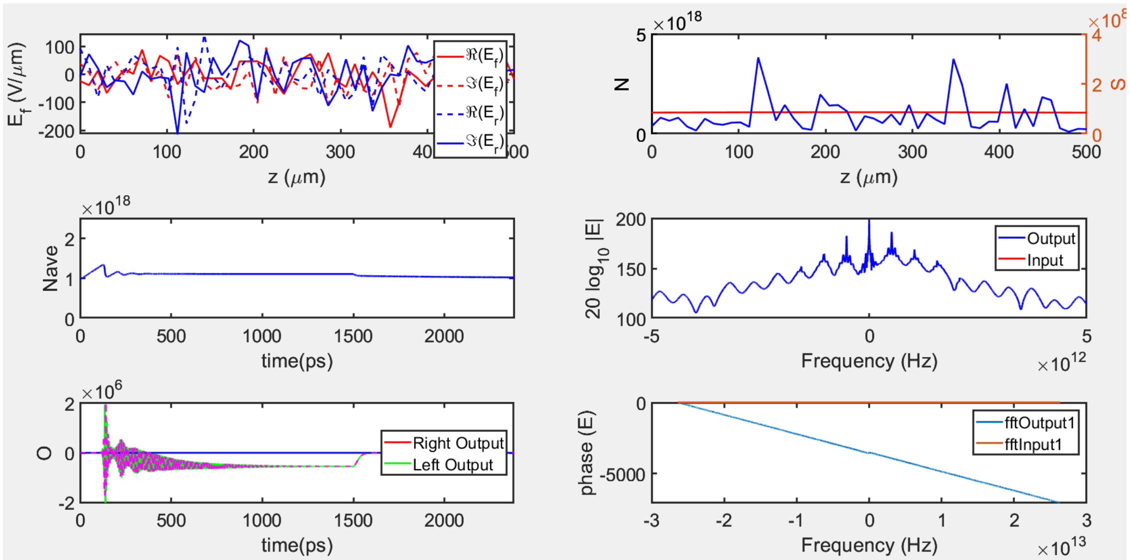

Figure 16: Lasing behavior when $I_D = 0.1$, waveguide width is 500 μm, and $I_{\text{off}} = 1.5 \times 10^{-9}$

From Figure 16, we can see from the FFT plot that there is a dominant frequency with two acceptable sideband peaks, but aside from that, the power is spread out over other frequencies.

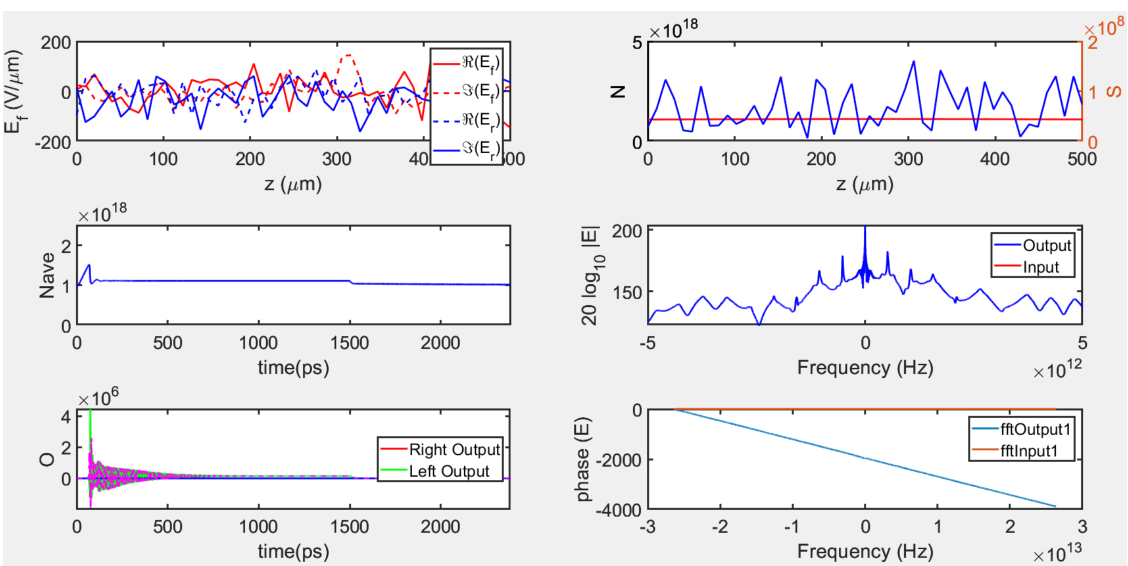

Figure 17: Lasing behavior when $I_D = 0.25$, waveguide width is 500 μm, and $I_{\text{off}} = 1.5 \times 10^{-9}$

From Figure 17, we again see a dominant frequency with two acceptable sideband peaks, similar to Figure 16. However, in this case, the peaks are relatively more defined, indicating stronger amplification due to the higher injection current.

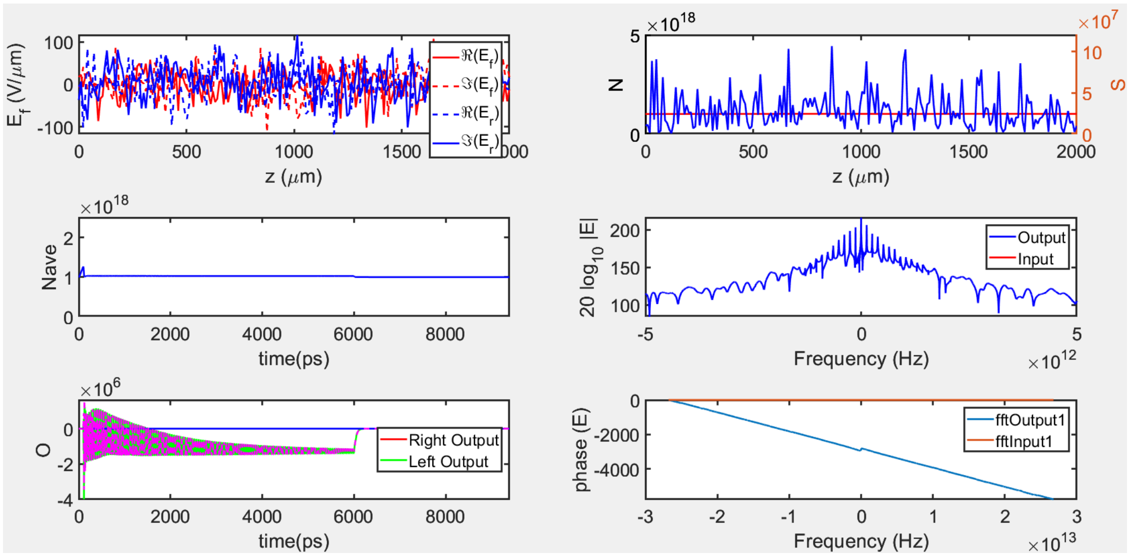

Figure 18: Lasing behavior when $I_D = 0.1$, waveguide width is 2000 μm, and $I_{\text{off}} = 6 \times 10^{-9}$

From Figure 18, the FFT plot shows a dominant frequency with nearly ten acceptable sideband peaks. This indicates that the increased waveguide length allows for significant amplification across multiple modes.

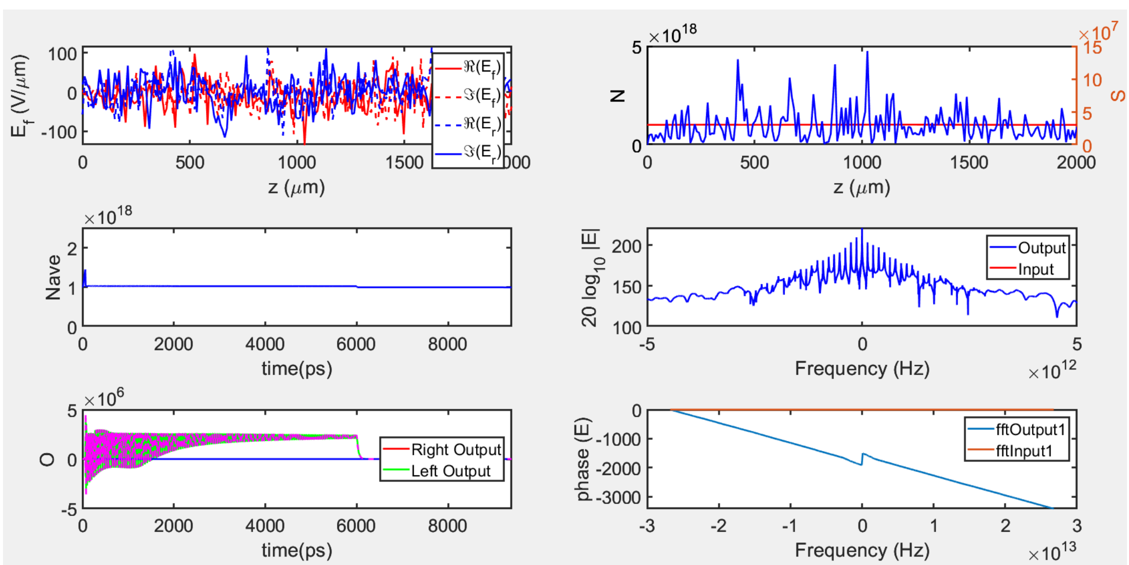

Figure 19: Lasing behavior when $I_D = 0.25$, waveguide width is 2000 μm, and $I_{\text{off}} = 6 \times 10^{-9}$

From Figure 19, the FFT plot reveals a dominant frequency with nearly eleven or twelve noticeable sideband peaks. Compared to Figure 18, there is a wider spectral spread and stronger amplification in the dominant mode, due to the higher injection current. This results in more sidebands receiving power, especially at higher orders.

These four figures together demonstrate how both the waveguide length and the injection current I_D affect the lasing characteristics. A longer waveguide supports the amplification of more longitudinal modes, as seen by the increased number of sidebands in Figures 18 and 19. Additionally, increasing I_D strengthens the gain, making the dominant modes more pronounced and extending the energy distribution into higher-order sidebands. In summary,longer waveguide length leads to a broader spectra of modes, while shorter waveguides results in narrower spectral profiles with lower spectral gain and fewer active modes. Ultimately, it all comes down to time, as more time means more amplification, and more amplification leads to greater power distribution. This allows additional modes to emerge and increases the gain of existing modes.

4.4 Conclusion

In conclusion, Milestone 8 introduced feedback through mirrors to enable lasing. No structural changes or additions to the equations developed up to Milestone 7 were required. Only the boundary conditions and the parameter controlling spontaneous emission were adjusted. Simulations confirmed laser behavior: spontaneous emission triggered field growth, and feedback from the mirrors helped select dominant longitudinal modes. The results aligned well with theoretical expectations.

5 Final Investigation (Milestone 9)

This Milestone deals with the open-ended section of the project and builds upon the foundational work developed through Milestones 1 to 8. Using the modeling framework and simulation tools established earlier, this section explores two focused investigations from final milestone ideas. Specifically, we examine:

- Mode evolution during start-up (Idea 1)
- Back reflection with phase shift (Idea 4)

Both of these topics provide a deeper insight into the various relationships between the parameters of a laser system. The following subsections describe each investigation in detail, beginning with mode evolution during start-up.

5.1 Mode Evolution During Start-Up

I am unsure if any of my plots are correct, therefore all my explanations may be wrong. But I have tried my best to give a proper explanation of what is happening.

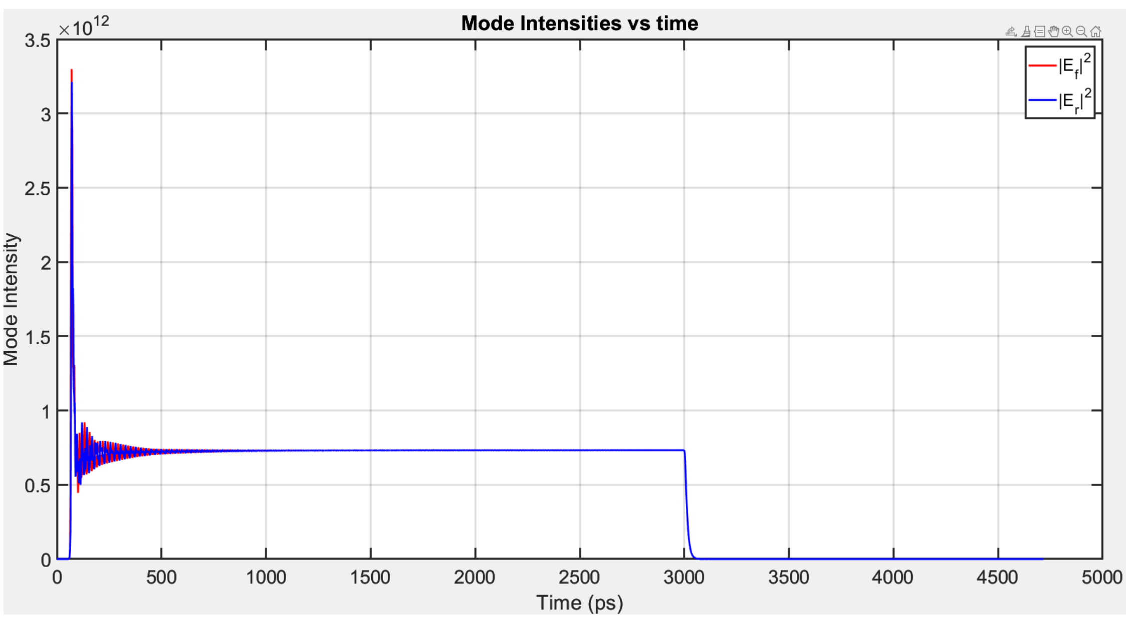

Figure 20: The mode evolution of the system with $I_d = 0.1$ along a waveguide length of 1000 µm

Figure 20 depicts the mode evolution with respect to time. We can see that in the beginning stages of laser start-up, the change in mode intensity is significant as it spikes and then drops quickly, before starting to oscillate. During this phase, the power begins to distribute among the modes, where the modes compete, and finally, the system settles at a constant intensity. That constant in Figure 20 is around 0.7×10^{12}, which, when converted to dB, corresponds to around 237 dB. The dominant mode gain for our simulations has been up to

210 dB, meaning that this plot is relatively accurate in showing how the mode intensities evolve over time.

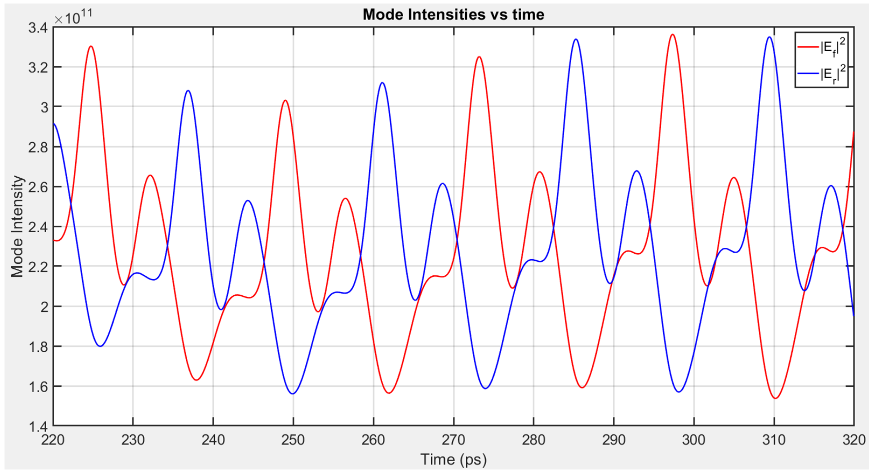

Figure 21: A zoomed-in version of the mode evolution of the system with the same parameters as the one in the figure above

Figure 21 shows a zoomed-in version of Figure 20. Here, it can be seen that the intensities of the forward and backward fields are around 270 degrees out of phase and repeat with changing amplitudes. Note: this plot is not from the same simulation; it comes from a different simulation run, but it can be replicated using the same code used to generate Figure 20. All it requires is changing the start and end times so that the code only captures the waveform within that time frame.

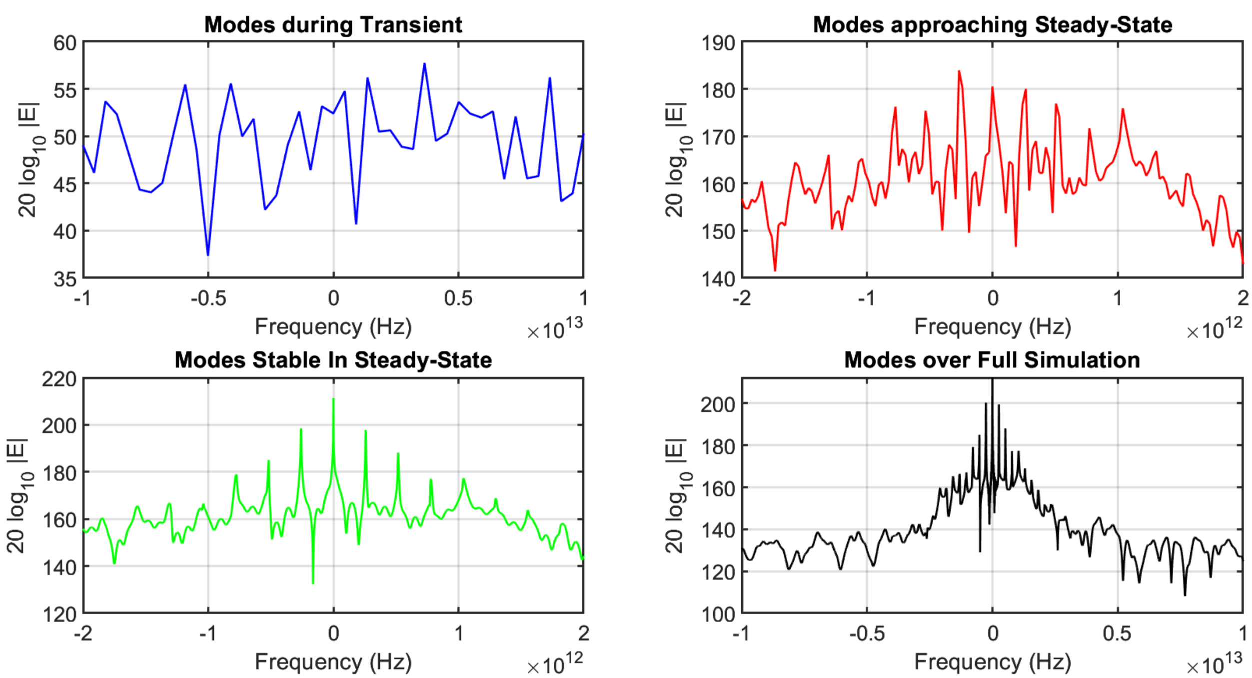

Figure 22: The mode intensity of the system over time

Figure 22 depicts the evolution of modes in the frequency domain. First from start to the transient phase, then from start to the point where it is just entering steady-state, then from start to steady-state, and finally from start to the end of simulation time. From the four plots, we see that during the transient phase, there is no clearly defined mode, and power is distributed across the entire spectrum. As the system starts approaching steady-state, some modes begin to receive more power through amplification, although they are not fully developed yet. Once we reach steady-state, the modes can be clearly seen and distinguished from the rest of the spectrum. Finally, by the end of the simulation, some very distinct modes can be identified by their significantly higher gain than the others.

To convert from decibels (dB) to linear gain, we use the formula:

$$G = 10^{\frac{\text{Gain (dB)}}{20}} \tag{14}$$

For a gain of 220 dB:

$$G_{220} = 10^{\frac{220}{20}} = 10^{11} = 100{,}000{,}000{,}000 \tag{15}$$

For a gain of 180 dB:

$$G_{180} = 10^{\frac{180}{20}} = 10^{9} = 1{,}000{,}000{,}000 \tag{16}$$

The ratio between the two linear gains is:

$$\frac{G_{220}}{G_{180}} = \frac{10^{11}}{10^{9}} = 10^{2} = 100 \tag{17}$$

So, a difference of 40 dB corresponds to a 100× difference in gain. From the plots, the difference between the dominant first and third mode suggests that the dominant mode has 100 times more power or amplification than the third mode.

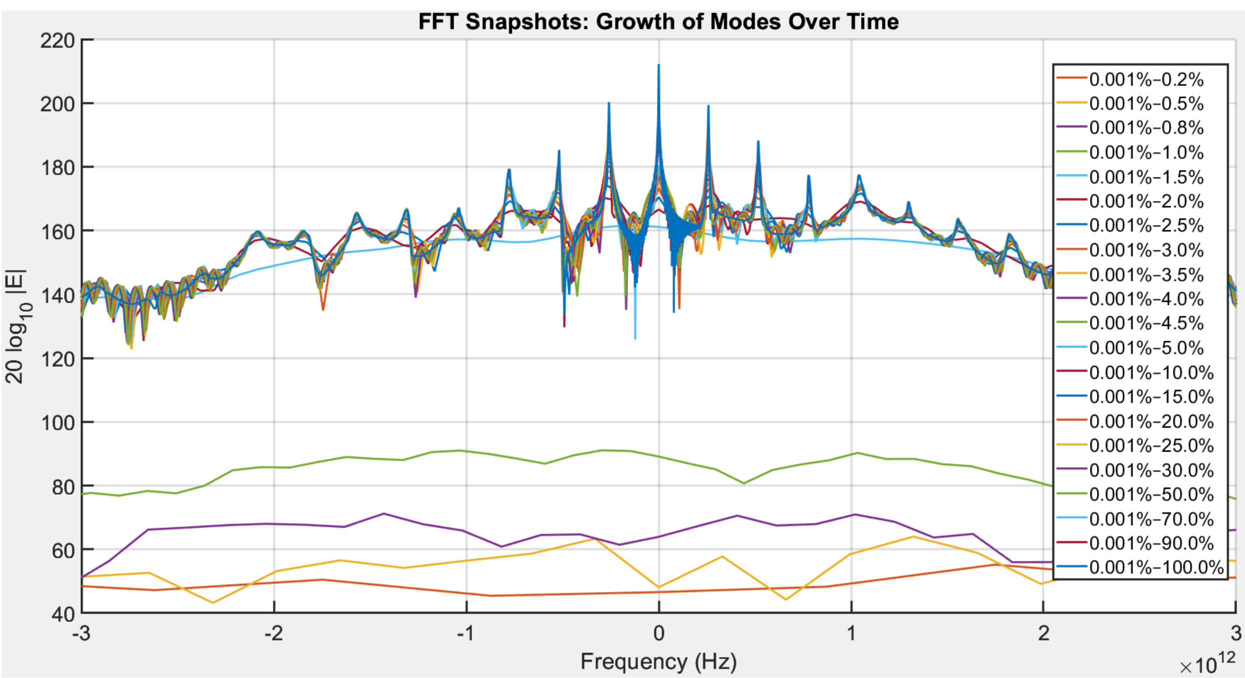

Figure 23: Mode evolution of the laser system as time increases

Figure 23 is a more advanced form of Figure 22, where instead of showing four distinct plots at different time frames, it displays a semi-continuous progression. This is a good representation of how to look at the mode intensity evolving with time because we can see that when only 1% of the total simulation time has passed, the laser is still in its transient phase. However, the amplification of spontaneous emission really starts to take off just after 1%, and we begin to that the gain is starting to concentrate closer to the middle. By 3%, we have a better idea of where the power is starting to gather. At 5%, the mode locations become even clearer. Once enough time has passed and the field intensities have exited the oscillatory phase shown in Figure 20, they stabilize, and very distinct peaks form.
What this tells us is that within approximately 250 ps, the laser begins to stabilize enough for the modes to start distinguishing themselves from the rest of the spectrum. This is an important observation, as it indicates how quickly lasers can stabilize, which is critical for systems that rely on lasers to perform time-sensitive operations. For example, the laser etching machines built by ASML would require extremely short stabilization times, as they need to etch millions of patterns very quickly. A laser with a long stabilization time may not be ideal for such tasks. By adjusting the parameters of this laser, we may be able to reduce the stabilization time even further.

The code used to obtain the plots in Figures 20, 21, 22, and 23 can be found in the Milestone 9 code.

5.2 Back reflection with phase shift (Idea 4)

This one deals with back reflection with a phase shift after a certain time delay. In other words, this subsection deals with taking a laser output, delaying it, and phase-shifting it before injecting it back again.

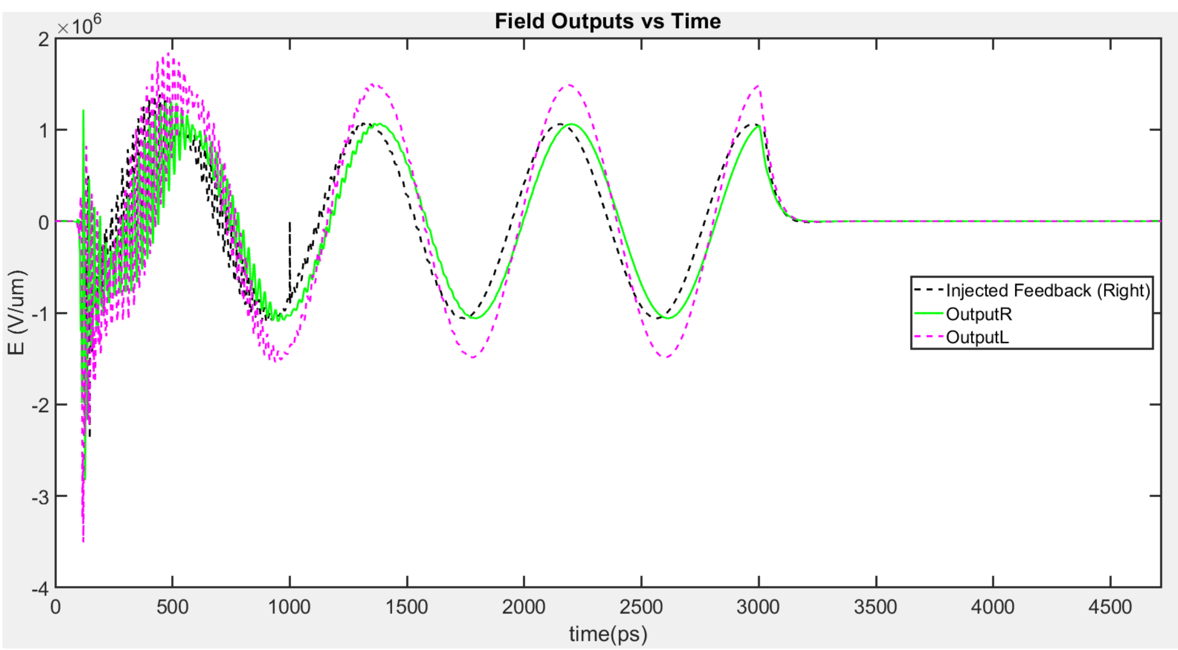

Figure 24: The characteristic of lasing looking to move in like a sine wave

Figure 24 represents the time domain response of the system, when there is phase-shifted feedback from the right back into the system. The feedback happens merely 10 ps after the simulation starts, and the sine wave-like characteristic starts to show up very quickly.

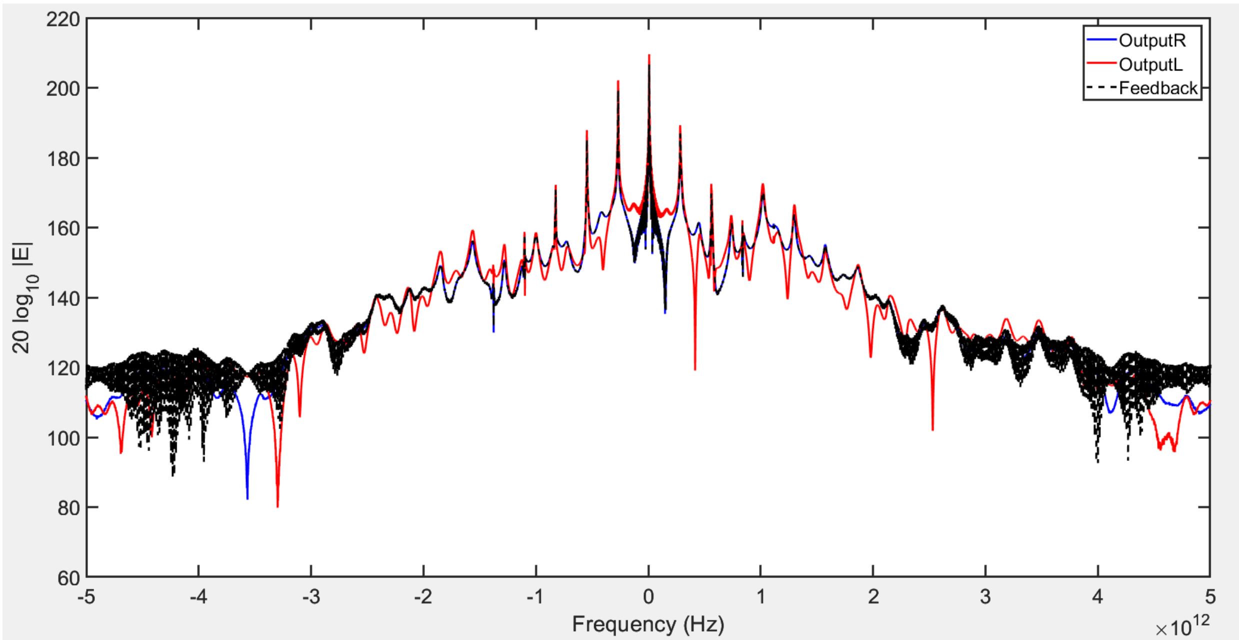

Figure 25: The spectral response of the system with respect to the feedback with a $\pi/6$ phase shift coming into the system

We can see from Figure 25 that while the spectral responses of the feedback, the output at the right boundary, and the left boundary may not match, as expected, their peaks, or modes, do align, even if the gain at those modes is not the same for all three.
Although we receive a sinusoidal pattern in Figure 24, it may not be true if the delay is changed, since a laser is a very sensitive system and even the smallest changes can have significant effects on it.
For this next one, the delay is increased to 500 ps, well after the laser has stabilized with distinct modes. The back injection may disturb the laser, may not have any effect, or may have some kind of effect that is good for the system.

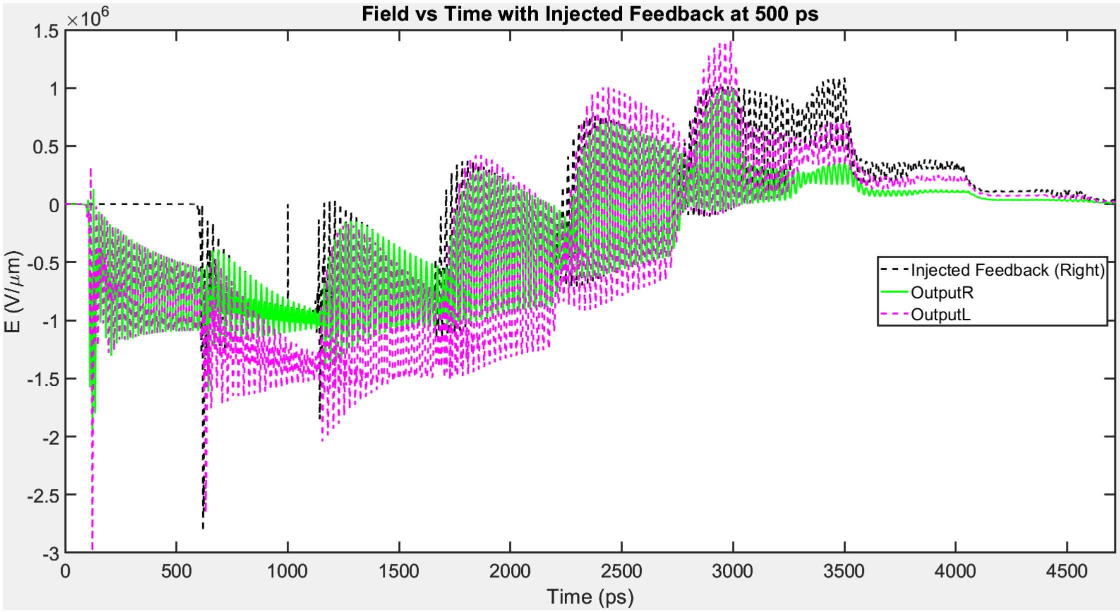

Figure 26: The time evolution of the fields with respect to the feedback with a $\pi/6$ phase shift coming into the system and a delay of 500 ps

From Figure 26, we can see that after the initial feedback at around 500 ps, the system registers the feedback, but what seems to be happening is that there is back injection every 500 ps. There is no clear way to describe the pattern, but it looks like it increases with every back injection, until I_d is turned off, and then it starts to reduce. This effects seems to be more mathematical than physical, so can be disregarded from an analytical perspective but interesting nonetheless.

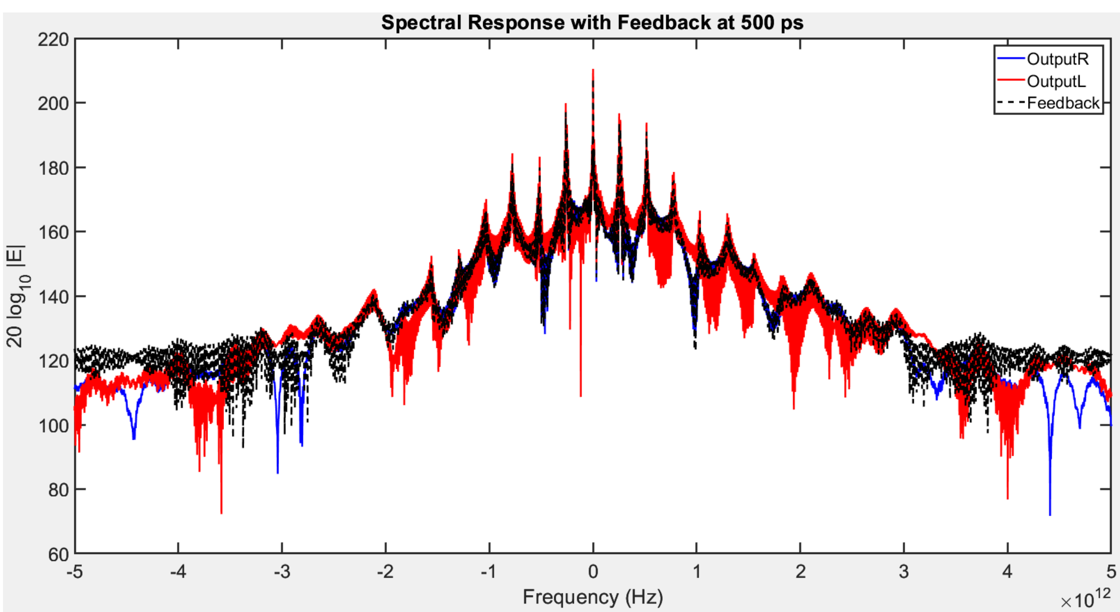

Figure 27: The spectral response of the system with respect to the feedback with a $\pi/6$ phase shift coming into the system and a delay of 500 ps

Figure 27 is the spectral response of the fields and feedback from Figure 26. When comparing this to Figure 25, it can be seen that there are almost the same number of modes, but in Figure 26, there seems to be more spectral oscillation between any two consecutive modes. The best guess for this happening is that inputting a delay at 500 ps—or more specifically, after the laser has stabilized and reached steady state causes some kind of interference, which creates structured bands of lower power between the main frequency components (between the modes).

This tells us that it is best to inject feedback into the system when it is still in the transient phase or just entering steady state. This also confirms the sensitive nature of a laser system, where just about any change can disturb the laser and its lasing.

6 Appendix: Code Listings

6.1 Section A: Milestone 6 - Carrier Equations

```
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 = 100e5;               % Amplitude of the input E-field / E_f
InputParasL.we = 0;                 % Frequency of the complex sinusoidal modulation on the gaussian pulse
InputParasL.t0 = 200e-12;             % The constant we are shifting the time by
InputParasL.wg = 10e-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
InputParasL.rep = 500e-12;

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
f0 = c_c/Lambda;

plotN = 10;                         % Divisior constant
L = 1000e-6*1e2;                    % length of the waveguide in cm

% Material Polarization Information
g_fwhm = 0;            % Frequency
LGamma = g_fwhm*2*pi;
Lw0 = 0;
LGain = 0;                       % 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 = 100;                           % 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(400*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;

Nave = nan(1,Nt);
Ntr = 1e18;
N = ones(size(z))*Ntr;
Nave(1) = mean(N);

gain = vg*2.5e-16;
eVol = 1.5e-10*c_q;
Ion = 0.25e-9;
Ioff = 3e-9;
I_off = 0.024;
I_on = 0.1;
taun = 1e-9;
Zg = sqrt(c_mu_0/c_eps_0)/n_g;
EtoP = 1/(Zg*f0*vg*1e-2*c_hb);
alpha = 0;

figure('name', 'Fields')

% Forward field E_f
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')

% Carrier Density N
subplot(3,2,2)
plot(z*10000, N, 'r');
xlim(XL * 1e4)
ylim([0, 5 * Ntr])
xlabel('z(\mum)')
ylabel('N')

% Average Carrier Density Over Time
subplot(3,2,3)
plot(time * 1e12, Nave, 'b');
xlim([0, Nt * dt * 1e12])
ylim([0, 5 * Ntr])
xlabel('time(ps)')
ylabel('Nave')

% Input and Output Fields over time
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


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, 0);  % 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

    S = (abs(Ef).^2 +abs(Er).^2).*EtoP*1e-6;

    if t < Ion || t > Ioff
        I_injv = I_off;
    else
        I_injv = I_on;
    end

    Stim = gain.*(N - Ntr).*S;
    N = (N + dt*(I_injv/ eVol - Stim))./(1+ dt/taun);
    Nave(i) = mean(N);

    % % 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,1000) == 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 (V/um)')
        legend('\Re', '\Im')
        hold off

        % Carrier Density N
        subplot(3,2,2)
        plot(z * 10000, N, 'r');
        xlim(XL * 1e4)
        ylim([0, 5 * Ntr])
        xlabel('z(\mum)')
        ylabel('N')

        % Average Carrier Density Over Time
        subplot(3,2,3)
        plot(time * 1e12, Nave, 'b');
        xlim([0, Nt * dt * 1e12])
        ylim([0, 5 * Ntr])
        xlabel('time(ps)')
        ylabel('Nave')

        % Input and Output Fields over time
        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('O')
        legend('Left Input', 'Right Output', 'Right Input', 'Left Output', 'Location', 'east')
        hold off
        pause(0.01)
    end

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

Listing 1: MATLAB code for Milestone 6

6.2 Section B: Milestone 7 – Optical Amplifiers and SPE

```

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


beta_spe = .3e-5;
gamma = 1.0;
% SPE = 7;
SPE = 0;

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 = 200e-12;           % The constant we are shifting the time by
InputParasL.wg = 50e-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
InputParasL.rep = 500e-12;

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
f0 = c_c/Lambda;

plotN = 10;                       % Division 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 = 0.05;                     % 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 = 100;                         % 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(400*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
```

```matlab

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;

Nave = nan(1,Nt);
Ntr = 1e18;
N = ones(size(z))*Ntr;
Nave(1) = mean(N);

gain = vg*2.5e-16;
eVol = 1.5e-10*c_q;
Ion = 0.25e-9;
Ioff = 3e-9;
% I_off = 0.024;
% I_on = 0.1;
I_off = 0.024;
I_on = 0.25;
taun = 1e-9;
Zg = sqrt(c_mu_0/c_eps_0)/n_g;
EtoP = 1/(Zg*f0*vg*1e-2*c_hb);
alpha = 0;

figure('name', 'Fields')

% Forward field E_f
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')

% Carrier Density N
subplot(3,2,2)
plot(z*10000, N, 'r');
xlim(XL * 1e4)
ylim([0, 5 * Ntr])
xlabel('z(\mum)')
ylabel('N')

% Average Carrier Density Over Time
subplot(3,2,3)
plot(time * 1e12, Nave, 'b');
xlim([0, Nt * dt * 1e12])
ylim([0, 5 * Ntr])
xlabel('time(ps)')
ylabel('Nave')

% Input and Output Fields over time
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


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

    % Input
```

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

    S = (abs(Ef).^2 + abs(Er).^2).*EtoP*1e-6;

    if t < Ion || t > Ioff
        I_injv = I_off;
    else
        I_injv = I_on;
    end

    Stim = gain.*(N - Ntr).*S;
    N = (N + dt*(I_injv/ eVol - Stim))./(1+ dt/taun);
    Nave(i) = mean(N);

    gain_z = gain.*(N - Ntr)./vg;  % Compute gain coefficient
    beta_i = (gain_z - alpha)./2;  % Compute imaginary part of propagation constant
    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

    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
    Tf = LGamma * Ef(1:Nz-2) + Cw0 * Pfp(2:Nz-1) + LGamma * Efp(1:Nz-2);  % Computes the forward polarization response
        based on previous field values.
    Pf(2:Nz-1) = (Pfp(2:Nz-1) + 0.5 * dt * Tf) ./ (1 - 0.5 * dt * Cw0);   % Updates the forward polarization field for
        every time step.
    Tr = LGamma * Er(3:Nz) + Cw0 * Prp(2:Nz-1) + LGamma * Erp(3:Nz);      % Computes the reverse polarization response
        based on previous field values.
    Pr(2:Nz-1) = (Prp(2:Nz-1) + 0.5 * dt * Tr) ./ (1 - 0.5 * dt * Cw0);   % Updates the reverse polarization field for
        every time step.

    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

    A = sqrt(gamma*beta_spe*c_hb*f0*L*1e-2/taun)/(2*Nz);
    if SPE > 0
        eTf = ((randn(Nz,1)+1i*randn(Nz,1))*A).';
        eTr = ((randn(Nz,1)+1i*randn(Nz,1))*A).';
    else
        eTf = ((ones(Nz,1))*A).';
        eTr = ((ones(Nz,1))*A).';
    end

    EsF = eTf*abs(SPE).*sqrt(N.*1e6);
    Esr = eTr*abs(SPE).*sqrt(N.*1e6);

    Ef = Ef + EsF;
    Er = Er + Esr;

    % % 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,1000) == 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 (V/um)')
        legend('\Re', '\Im')
        hold off

        % Carrier Density N
        subplot(3,2,2)
        plot(z * 10000, N, 'r');
        xlim(XL * 1e4)
        ylim([0, 5 * Ntr])
```

```
        xlabel('z(\mum)')
        ylabel('N')

        % Average Carrier Density Over Time
        subplot(3,2,3)
        plot(time * 1e12, Nave, 'b');
        xlim([0, Nt * dt * 1e12])
        ylim([0, 5 * Ntr])
        xlabel('time(ps)')
        ylabel('Nave')

        % Input and Output Fields over time
        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 auto
        xlabel('time(ps)')
        ylabel('0')
        legend('Left Input', 'Right Output', 'Right Input', 'Left Output', 'Location', 'east')
        hold off


        % Output Field Spectrum (Magnitude in dB)
        subplot(3,2,4)
        plot(omega, 20*log10(abs(fftOutput1)), 'b'); hold on
        %plot(omega, 20*log10(abs(fftInput1)), 'r'); %
        xlabel('GHz')
        ylabel('20 log_{10} |E|')
        legend('Output', 'Input')
        hold off

        % Phase of Output Field
        subplot(3,2,6)
        phase1 = unwrap(angle(fftOutput1));  % Unwrap the phase1
        phase2 = unwrap(angle(fftInput1));  % Unwrap the phase for Input
        plot(omega, phase1);
        hold on;
        plot(omega, phase2);
        hold off;
        xlabel('GHz')
        ylabel('phase (E)')
        legend('fftOutput1', 'fftInput1');
        ylim auto

        pause(0.01)


    end

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

Listing 2: MATLAB code for Milestone 7

6.3 Section C: Milestone 8 - Lasers

```

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


beta_spe = .3e-5;
gamma = 1.0;
SPE = 7;
```

```matlab

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 = 200e-12;          % The constant we are shifting the time by
InputParasL.wg = 10e-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
InputParasL.rep = 500e-12;

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
f0 = c_c/Lambda;

plotN = 10;                      % Divisior constant

% L = 1000e-6*1e2;                 % length of the waveguide in cm
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 = 0.015;                    % Gain Constant

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

% Nz = 100;                        % Number of divisions
Nz = 100;                        % 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(400*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;

Nave = nan(1,Nt);
Ntr = 1e18;
N = ones(size(z))*Ntr;
Nave(1) = mean(N);

gain = vg*2.5e-16;
eVol = 1.5e-10*c_q;
Ion = 0.01e-9;
% Ion = 0.25e-9;
Ioff = 3e-9;
```

```
I_off = 0.024;
I_on = 0.1;
taun = 1e-9;
Zg = sqrt(c_mu_0/c_eps_0)/n_g;
EtoP = 1/(Zg*f0*vg*1e-2*c_hb);
alpha = 0;

figure('name', 'Fields')

% Forward field E_f
subplot(3,2,1)
plot(z*10000, real(Ef), 'r'); hold on
plot(z*10000, imag(Ef), 'r--');
plot(z*10000, real(Er), 'b--');
plot(z*10000, imag(Er), 'b');
hold off
xlim(XL*1e4)
ylim auto
xlabel('z (\mum)')
ylabel('E_f (V/\mum)')
legend('\Re(E_f)', '\Im(E_f)', '\Re(E_r)', '\Im(E_r)')


% Carrier Density N
subplot(3,2,2)
plot(z * 10000, N, 'r'); % Primary plot
xlim(XL * 1e4)
ylim([0, 5 * Ntr])
xlabel('z (\mum)')
ylabel('N')

% % Add a second y-axis for S
% yyaxis right
% plot(z * 10000, S, 'b'); % Plot S on the right axis
% ylabel('S') % Change the label to the appropriate units


% Average Carrier Density Over Time
subplot(3,2,3)
plot(time * 1e12, Nave, 'b');
xlim([0, Nt * dt * 1e12])
ylim([0, 5 * Ntr])
xlabel('time(ps)')
ylabel('Nave')

subplot(3,2,5)
plot(time * 1e12, real(OutputR), 'g'); hold on
plot(time * 1e12, real(OutputL), 'm--');
xlim([0, Nt * dt * 1e12])
ylim auto
xlabel('time(ps)')
ylabel('E (V/um)')
legend('Right Output', 'Left Output', 'Location', 'east')
hold off

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

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

    RL = 0.5;                               % The left side reflection coefficient
    RR = 0.5;                               % The right side reflection coefficient

    % Input
    InputL(i) = Ef1(t,0);
    % 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);

    S = (abs(Ef).^2 + abs(Er).^2).*EtoP*1e-6;

    if t < Ion || t > Ioff
        I_injv = I_off;
    else
        I_injv = I_on;
    end

    Stim = gain.*(N - Ntr).*S;
    N = (N + dt*(I_injv/ eVol - Stim))./(1+ dt/taun);
    Nave(i) = mean(N);

    gain_z = gain.*(N - Ntr)./vg;  % Compute gain coefficient
    beta_i = (gain_z - alpha)./2;  % Compute imaginary part of propagation constant
    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

    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
    Tf = LGamma * Ef(1:Nz-2) + Cw0 * Pfp(2:Nz-1) + LGamma * Efp(1:Nz-2);  % Computes the forward polarization response
         based on previous field values.
    Pf(2:Nz-1) = (Pfp(2:Nz-1) + 0.5 * dt * Tf) ./ (1 - 0.5 * dt * Cw0);   % Updates the forward polarization field for
         every time step.
    Tr = LGamma * Er(3:Nz) + Cw0 * Prp(2:Nz-1) + LGamma * Erp(3:Nz);      % Computes the reverse polarization response
         based on previous field values.
    Pr(2:Nz-1) = (Prp(2:Nz-1) + 0.5 * dt * Tr) ./ (1 - 0.5 * dt * Cw0);   % Updates the reverse polarization field for
         every time step.

    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

    A = sqrt(gamma*beta_spe*c_hb*f0*L*1e-2/taun)/(2*Nz);
    if SPE > 0
        eTf = ((randn(Nz,1)+1i*randn(Nz,1))*A).';
        eTr = ((randn(Nz,1)+1i*randn(Nz,1))*A).';
    else
        eTf = ((ones(Nz,1))*A).';
        eTr = ((ones(Nz,1))*A).';
    end

    EsF = eTf*abs(SPE).*sqrt(N.*1e6);
    Esr = eTr*abs(SPE).*sqrt(N.*1e6);

    Ef = Ef + EsF;
    Er = Er + Esr;

    % % 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,2000) == 0       % Only executed when i is multiple of plotN

        % Forward field E_f
        subplot(3,2,1)
        plot(z*10000, real(Ef), 'r'); hold on
        plot(z*10000, imag(Ef), 'r--');
        plot(z*10000, real(Er), 'b--');
        plot(z*10000, imag(Er), 'b');
        hold off
        xlim(XL*1e4)
        ylim auto
        xlabel('z (\mum)')
        ylabel('E_f (V/\mum)')
        legend('\Re(E_f)', '\Im(E_f)', '\Re(E_r)', '\Im(E_r)')


        % Carrier Density N
        subplot(3,2,2)
        plot(z * 10000, N, 'r'); % Primary plot
        xlim(XL * 1e4)
        ylim([0, 5 * Ntr])
        xlabel('z (\mum)')
        ylabel('N')

        % Add a second y-axis for S
        yyaxis right
        plot(z * 10000, S, 'b'); % Plot S on the right axis
        ylabel('S') % Change the label to the appropriate units

        % Average Carrier Density Over Time
        subplot(3,2,3)
        plot(time * 1e12, Nave, 'b');
        xlim([0, Nt * dt * 1e12])
        ylim([0, 2.5 * Ntr])
        xlabel('time(ps)')
        ylabel('Nave')

        % Input and Output Fields over time
        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 auto
        xlabel('time(ps)')
```

```
        ylabel('O')
        legend( 'Right Output', 'Left Output', 'Location', 'east')
        hold off


        % Output Field Spectrum (Magnitude in dB)
        subplot(3,2,4)
        plot(omega, 20*log10(abs(fftOutput1)), 'b'); hold on
        plot(omega, 20*log10(abs(fftInput1)), 'r'); %
        xlabel('Frequency (Hz)')
        ylabel('20 log_{10} |E|')
        legend('Output', 'Input')
        xlim([-0.05e14 0.05e14])
        hold off

        % Phase of Output Field
        subplot(3,2,6)
        phase1 = unwrap(angle(fftOutput1));  % Unwrap the phase1
        phase2 = unwrap(angle(fftInput1));  % Unwrap the phase for Input
        plot(omega, phase1);
        hold on;
        plot(omega, phase2);
        hold off;
        xlabel('Frequency (Hz)')
        ylabel('phase (E)')
        legend('fftOutput1', 'fftInput1');

        pause(0.01)


    end


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

Listing 3: MATLAB code for Milestone 8

6.4 Section D: Milestone 9 – Final Investigation 1

```

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


beta_spe = .3e-5;
gamma = 1.0;
SPE = 7;

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 = 200e-12;          % The constant we are shifting the time by
InputParasL.wg = 10e-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
InputParasL.rep = 500e-12;

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
f0 = c_c/Lambda;

```

```
plotN = 10;                       % Division constant

% L = 1000e-6*1e2;                  % length of the waveguide in cm
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 = 0.015;                     % Gain Constant

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

% Nz = 100;                         % Number of divisions
Nz = 100;                         % 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(400*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;

Nave = nan(1,Nt);
Ntr = 1e18;
N = ones(size(z))*Ntr;
Nave(1) = mean(N);

gain = vg*2.5e-16;
eVol = 1.5e-10*c_q;
Ion = 0.01e-9;
% Ion = 0.25e-9;
Ioff = 3e-9;
I_off = 0.024;
I_on = 0.25;
taun = 1e-9;
Zg = sqrt(c_mu_0/c_eps_0)/n_g;
EtoP = 1/(Zg*f0*vg*1e-2*c_hb);
alpha = 0;

figure('name', 'Fields')
If = nan(1, Nt);  % Forward mode intensity over time
Ir = nan(1, Nt);  % Backward mode intensity over time


% Forward field E_f
subplot(3,2,1)
plot(z*10000, real(Ef), 'r'); hold on
plot(z*10000, imag(Ef), 'r--');
plot(z*10000, real(Er), 'b--');
plot(z*10000, imag(Er), 'b');
hold off
```

```
xlim(XL*1e4)
ylim auto
xlabel('z (\mum)')
ylabel('E_f (V/\mum)')
legend('\Re(E_f)', '\Im(E_f)', '\Re(E_r)', '\Im(E_r)')


% Carrier Density N
subplot(3,2,2)
plot(z * 10000, N, 'r'); % Primary plot
xlim(XL * 1e4)
ylim([0, 5 * Ntr])
xlabel('z (\mum)')
ylabel('N')

% % Add a second y-axis for S
% yyaxis right
% plot(z * 10000, S, 'b'); % Plot S on the right axis
% ylabel('S') % Change the label to the appropriate units


% Average Carrier Density Over Time
subplot(3,2,3)
plot(time * 1e12, Nave, 'b');
xlim([0, Nt * dt * 1e12])
ylim([0, 5 * Ntr])
xlabel('time(ps)')
ylabel('Nave')

subplot(3,2,5)
plot(time * 1e12, real(OutputR), 'g'); hold on
plot(time * 1e12, real(OutputL), 'm--');
xlim([0, Nt * dt * 1e12])
ylim auto
xlabel('time(ps)')
ylabel('E (V/um)')
legend('Right Output', 'Left Output', 'Location', 'east')
hold off

% Define time range for plotting mode intensities
t_start = 0.000001e-12;  % 100 ps
t_end = Nt;    % 300 ps



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

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

    RL = 0.5;                                  % The left side reflection coefficient
    RR = 0.5;                                  % The right side reflection coefficient

    % Input
    InputL(i) = Ef1(t,0);
    % 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);

    S = (abs(Ef).^2 + abs(Er).^2).*EtoP*1e-6;

    if t < Ion || t > Ioff
        I_injv = I_off;
    else
        I_injv = I_on;
    end

    Stim = gain.*(N - Ntr).*S;
    N = (N + dt*(I_injv/ eVol - Stim))./(1+ dt/taun);
    Nave(i) = mean(N);

    gain_z = gain.*(N - Ntr)./vg;  % Compute gain coefficient
    beta_i = (gain_z - alpha)./2;  % Compute imaginary part of propagation constant
    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

    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
    Tf = LGamma * Ef(1:Nz-2) + Cw0 * Pfp(2:Nz-1) + LGamma * Efp(1:Nz-2);  % Computes the forward polarization response
         based on previous field values.
```

```
    Pf(2:Nz-1) = (Pfp(2:Nz-1) + 0.5 * dt * Tf) ./ (1 - 0.5 * dt * Cw0);   % Updates the forward polarization field for
         every time step.
    Tr = LGamma * Er(3:Nz) + Cw0 * Prp(2:Nz-1) + LGamma * Erp(3:Nz);      % Computes the reverse polarization response
         based on previous field values.
    Pr(2:Nz-1) = (Prp(2:Nz-1) + 0.5 * dt * Tr) ./ (1 - 0.5 * dt * Cw0);   % Updates the reverse polarization field for
         every time step.

    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

    % --- Mode intensities as a function of time ---
    If(i) = trapz(z, abs(Ef).^2);   % Forward mode intensity
    Ir(i) = trapz(z, abs(Er).^2);   % Backward mode intensity


    A = sqrt(gamma*beta_spe*c_hb*f0*L*1e-2/taun)/(2*Nz);
    if SPE > 0
        eTf = ((randn(Nz,1)+1i*randn(Nz,1))*A).';
        eTr = ((randn(Nz,1)+1i*randn(Nz,1))*A).';
    else
        eTf = ((ones(Nz,1))*A).';
        eTr = ((ones(Nz,1))*A).';
    end

    EsF = eTf*abs(SPE).*sqrt(N.*1e6);
    Esr = eTr*abs(SPE).*sqrt(N.*1e6);

    Ef = Ef + EsF;
    Er = Er + Esr;

    % % 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));

    % Find indices for the selected time range
    range_idx = find(time >= t_start & time <= t_end);

    if mod(i,2000) == 0       % Only executed when i is multiple of plotN

        % Forward field E_f
        subplot(3,2,1)
        plot(z*10000, real(Ef), 'r'); hold on
        plot(z*10000, imag(Ef), 'r--');
        plot(z*10000, real(Er), 'b--');
        plot(z*10000, imag(Er), 'b');
        hold off
        xlim(XL*1e4)
        ylim auto
        xlabel('z (\mum)')
        ylabel('E_f (V/\mum)')
        legend('\Re(E_f)', '\Im(E_f)', '\Re(E_r)', '\Im(E_r)')


        % Carrier Density N
        subplot(3,2,2)
        plot(z * 10000, N, 'r'); % Primary plot
        xlim(XL * 1e4)
        ylim([0, 5 * Ntr])
        xlabel('z (\mum)')
        ylabel('N')

        % Add a second y-axis for S
        yyaxis right
        plot(z * 10000, S, 'b'); % Plot S on the right axis
        ylabel('S') % Change the label to the appropriate units

        % Average Carrier Density Over Time
        subplot(3,2,3)
        plot(time * 1e12, Nave, 'b');
        xlim([0, Nt * dt * 1e12])
        ylim([0, 2.5 * Ntr])
        xlabel('time(ps)')
        ylabel('Nave')

        % Input and Output Fields over time
        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 auto
        xlabel('time(ps)')
        ylabel('O')
        legend( 'Right Output', 'Left Output', 'Location', 'east')
        hold off
```

```matlab


        % Output Field Spectrum (Magnitude in dB)
        subplot(3,2,4)
        plot(omega, 20*log10(abs(fftOutput1)), 'b'); hold on
        plot(omega, 20*log10(abs(fftInput1)), 'r'); %
        xlabel('Frequency (Hz)')
        ylabel('20 log_{10} |E|')
        legend('Output', 'Input')
        xlim([-0.05e14 0.05e14])
        hold off

        % Phase of Output Field
        subplot(3,2,6)
        phase1 = unwrap(angle(fftOutput1));  % Unwrap the phase1
        phase2 = unwrap(angle(fftInput1));  % Unwrap the phase for Input
        plot(omega, phase1);
        hold on;
        plot(omega, phase2);
        hold off;
        xlabel('Frequency (Hz)')
        ylabel('phase (E)')
        legend('fftOutput1', 'fftInput1');

        figure('name', 'Mode Intensities Over Time')
        plot(time(range_idx) * 1e12, If(range_idx), 'r', 'DisplayName', '|E_f|^2'); hold on
        plot(time(range_idx) * 1e12, Ir(range_idx), 'b', 'DisplayName', '|E_r|^2');
        xlabel('Time (ps)')
        ylabel('Mode Intensity')
        legend
        title('Mode Intensities vs time')
        grid on


        pause(0.01)


    end


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






end

%% UNCOMMENT THIS PART IF YOU WISH TO SEE PLOTS SIMILAR TO THE ONES IN MY FINAL REPORT
% figure('Name', 'FFT plots at different time samples');
% hold on;
%
% % Time points in % of total simulation
% pcts = [0.001, 0.25, 0.5, 0.75, 1, ...
%         1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5, ...
%         10, 15, 20, 25, 30, ...
%         50, 70, 90, 100];
%
% colors = lines(length(pcts));
% startIdx = round(0.001 * Nt);
%
% for k = 1:length(pcts)
%     endIdx = round((pcts(k)/100) * Nt);
%     idxRange = startIdx:endIdx;
%
%     if length(idxRange) < 2
%         continue
%     end
%
%     fftData = fftshift(abs(fft(OutputR(idxRange))));
%     omega = fftshift(wspace(time(idxRange)));
%
%     plot(omega, 20*log10(fftData), 'Color', colors(k,:), ...
%         'DisplayName', sprintf('%.3f%%   %.1f%%', 0.001, pcts(k)));
% end
%
% xlabel('Frequency (Hz)');
% ylabel('20 log_{10} |E|');
% title('FFT Snapshots: Growth of Modes Over Time');
% legend('Location', 'bestoutside');
% grid on;
% xlim([-0.03e14, 0.03e14]);
%
%
% disp('Simulation finished. Running FFT analysis...');
%
% transient_range = round(0.0001*Nt):round(0.003*Nt);
```

```
% steady_range = round(0.0001*Nt):round(0.05*Nt);
% stable_range = round(0.0001*Nt):round(0.59*Nt);
% full_range = 1:Nt;
%
% output_transient = OutputR(transient_range);
% output_steady = OutputR(steady_range);
% output_stable = OutputR(stable_range);
% output_full = OutputR(full_range);
%
% time_transient = time(transient_range);
% time_steady = time(steady_range);
% time_stable = time(stable_range);
% time_full = time(full_range);
%
% omega_transient = fftshift(wspace(time_transient));
% omega_steady = fftshift(wspace(time_steady));
% omega_stable = fftshift(wspace(time_stable));
% omega_full = fftshift(wspace(time_full));
%
% fft_transient = fftshift(fft(output_transient));
% fft_steady = fftshift(fft(output_steady));
% fft_stable = fftshift(fft(output_stable));
% fft_full = fftshift(fft(output_full));
%
%
%
% figure('Name', 'FFT Comparison (All Phases)', 'Color', 'w');
%
% % Transient FFT
% subplot(2,2,1);
% plot(omega_transient, 20*log10(abs(fft_transient)), 'b');
% xlabel('Frequency (Hz)');
% ylabel('20 log_{10} |E|');
% title('Modes during Transient');
% grid on;
% xlim([-0.1e14 0.1e14]);
%
% % Steady-State FFT
% subplot(2,2,2);
% plot(omega_steady, 20*log10(abs(fft_steady)), 'r');
% xlabel('Frequency (Hz)');
% ylabel('20 log_{10} |E|');
% title('Modes approaching Steady-State');
% grid on;
% xlim([-0.02e14 0.02e14]);
%
% % Stable FFT
% subplot(2,2,3);
% plot(omega_stable, 20*log10(abs(fft_stable)), 'g');
% xlabel('Frequency (Hz)');
% ylabel('20 log_{10} |E|');
% title('Modes Stable In Steady-State');
% grid on;
% xlim([-0.02e14 0.02e14]);
%
% % Full FFT
% subplot(2,2,4);
% plot(omega_full, 20*log10(abs(fft_full)), 'k');
% xlabel('Frequency (Hz)');
% ylabel('20 log_{10} |E|');
% title('Modes over Full Simulation');
% grid on;
% xlim([-0.1e14 0.1e14]);
```

Listing 4: MATLAB code for Milestone 9 Part 1

6.5 Section E: Milestone 9 – Final Investigation 2

```

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
```

```
beta_spe = .3e-5;
gamma = 1.0;
SPE = 7;

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 = 200e-12;           % The constant we are shifting the time by
InputParasL.wg = 10e-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
InputParasL.rep = 500e-12;

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
f0 = c_c/Lambda;

plotN = 10;                       % Division constant

% L = 1000e-6*1e2;                  % length of the waveguide in cm
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 = 0.015;                     % Gain Constant

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

% Nz = 100;                         % Number of divisions
Nz = 100;                         % 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(400*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;

Nave = nan(1,Nt);
Ntr = 1e18;
N = ones(size(z))*Ntr;
Nave(1) = mean(N);

```

```
gain = vg*2.5e-16;
eVol = 1.5e-10*c_q;
Ion = 0.01e-9;
% Ion = 0.25e-9;
Ioff = 3e-9;
I_off = 0.024;
I_on = 0.1;
taun = 1e-9;
Zg = sqrt(c_mu_0/c_eps_0)/n_g;
EtoP = 1/(Zg*f0*vg*1e-2*c_hb);
alpha = 0;

InjectedFeedback = zeros(1, Nt);


figure('name', 'Fields')

% Forward field E_f
subplot(3,2,1)
plot(z*10000, real(Ef), 'r'); hold on
plot(z*10000, imag(Ef), 'r--');
plot(z*10000, real(Er), 'b--');
plot(z*10000, imag(Er), 'b');
hold off
xlim(XL*1e4)
ylim auto
xlabel('z (\mum)')
ylabel('E_f (V/\mum)')
legend('\Re(E_f)', '\Im(E_f)', '\Re(E_r)', '\Im(E_r)')


% Carrier Density N
subplot(3,2,2)
plot(z * 10000, N, 'r'); % Primary plot
xlim(XL * 1e4)
ylim([0, 5 * Ntr])
xlabel('z (\mum)')
ylabel('N')

% % Add a second y-axis for S
% yyaxis right
% plot(z * 10000, S, 'b'); % Plot S on the right axis
% ylabel('S') % Change the label to the appropriate units


% Average Carrier Density Over Time
subplot(3,2,3)
plot(time * 1e12, Nave, 'b');
xlim([0, Nt * dt * 1e12])
ylim([0, 5 * Ntr])
xlabel('time(ps)')
ylabel('Nave')

subplot(3,2,5)
plot(time * 1e12, real(OutputR), 'g'); hold on
plot(time * 1e12, real(OutputL), 'm--');
xlim([0, Nt * dt * 1e12])
ylim auto
xlabel('time(ps)')
ylabel('E (V/um)')
legend('Right Output', 'Left Output', 'Location', 'east')
hold off

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

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

    RL = 0.5;                               % The left side reflection coefficient
    RR = 0.5;                               % The right side reflection coefficient


    delay_steps = round(500e-12/dt);
    phase_shift = pi/6;

    if i > delay_steps
        if abs(OutputR(i - delay_steps)) > 1e-3
            FeedbackR = OutputR(i - delay_steps);
            InjectedFeedback(i) = FeedbackR * exp(1i * phase_shift);
            InputR(i) = InjectedFeedback(i);
        else
            InjectedFeedback(i) = 0;
            InputR(i) = 0;
        end
    else
        InjectedFeedback(i) = 0;
        InputR(i) = 0;
    end




```

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

    S = (abs(Ef).^2 + abs(Er).^2).*EtoP*1e-6;

    if t < Ion || t > Ioff
        I_injv = I_off;
    else
        I_injv = I_on;
    end

    Stim = gain.*(N - Ntr).*S;
    N = (N + dt*(I_injv/ eVol - Stim))./(1+ dt/taun);
    Nave(i) = mean(N);

    gain_z = gain.*(N - Ntr)./vg;  % Compute gain coefficient
    beta_i = (gain_z - alpha)./2;  % Compute imaginary part of propagation constant
    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

    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
    Tf = LGamma * Ef(1:Nz-2) + Cw0 * Pfp(2:Nz-1) + LGamma * Efp(1:Nz-2);  % Computes the forward polarization response based on previous field values.
    Pf(2:Nz-1) = (Pfp(2:Nz-1) + 0.5 * dt * Tf) ./ (1 - 0.5 * dt * Cw0);   % Updates the forward polarization field for every time step.
    Tr = LGamma * Er(3:Nz) + Cw0 * Prp(2:Nz-1) + LGamma * Erp(3:Nz);      % Computes the reverse polarization response based on previous field values.
    Pr(2:Nz-1) = (Prp(2:Nz-1) + 0.5 * dt * Tr) ./ (1 - 0.5 * dt * Cw0);   % Updates the reverse polarization field for every time step.

    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

    delay_steps = round(1000e-12/dt);
    phase_shift = pi/6;

    if i == delay_steps + 1
        FeedbackR = OutputR(i - delay_steps);                     % Right-side output
        InjectedFeedback(i) = FeedbackR * exp(1i * phase_shift); % Phase shift
        InputR(i) = InjectedFeedback(i);                          % Inject to right side
    end

    A = sqrt(gamma*beta_spe*c_hb*f0*L*1e-2/taun)/(2*Nz);
    if SPE > 0
        eTf = ((randn(Nz,1)+1i*randn(Nz,1))*A).';
        eTr = ((randn(Nz,1)+1i*randn(Nz,1))*A).';
    else
        eTf = ((ones(Nz,1))*A).';
        eTr = ((ones(Nz,1))*A).';
    end

    EsF = eTf*abs(SPE).*sqrt(N.*1e6);
    Esr = eTr*abs(SPE).*sqrt(N.*1e6);

    Ef = Ef + EsF;
    Er = Er + Esr;


    % % 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
    fftFeedback = fftshift(fft(InjectedFeedback));
    omega = fftshift(wspace(time));


    if mod(i,2000) == 0       % Only executed when i is multiple of plotN

        % Forward field E_f
        subplot(3,2,1)
        plot(z*10000, real(Ef), 'r'); hold on
        plot(z*10000, imag(Ef), 'r--');
```

```
        plot(z*10000, real(Er), 'b--');
        plot(z*10000, imag(Er), 'b');
        hold off
        xlim(XL*1e4)
        ylim auto
        xlabel('z (\mum)')
        ylabel('E_f (V/\mum)')
        legend('\Re(E_f)', '\Im(E_f)', '\Re(E_r)', '\Im(E_r)')


        % Carrier Density N
        subplot(3,2,2)
        plot(z * 10000, N, 'r'); % Primary plot
        xlim(XL * 1e4)
        ylim([0, 5 * Ntr])
        xlabel('z (\mum)')
        ylabel('N')

        % Add a second y-axis for S
        yyaxis right
        plot(z * 10000, S, 'b'); % Plot S on the right axis
        ylabel('S') % Change the label to the appropriate units

        % Average Carrier Density Over Time
        subplot(3,2,3)
        plot(time * 1e12, Nave, 'b');
        xlim([0, Nt * dt * 1e12])
        ylim([0, 2.5 * Ntr])
        xlabel('time(ps)')
        ylabel('Nave')

        subplot(3,2,5)
        plot(time * 1e12, real(InjectedFeedback), 'k--'); hold on
        plot(time * 1e12, real(OutputR), 'g');
        plot(time * 1e12, real(OutputL), 'm--');
        xlim([0, Nt * dt * 1e12])
        ylim auto
        xlabel('time(ps)')
        ylabel('E (V/um)')
        legend('Injected Feedback (Right)', 'OutputR', 'OutputL', 'Location', 'east')
        hold off



        subplot(3,2,4)
        plot(omega, 20*log10(abs(fftOutput1)), 'b'); hold on
        plot(omega, 20*log10(abs(fftOutput2)), 'r');
        plot(omega, 20*log10(abs(fftFeedback)), 'k--');  % From InjectedFeedback
        xlabel('Frequency (Hz)')
        ylabel('20 log_{10} |E|')
        legend('OutputR', 'OutputL', 'Feedback')
        xlim([-0.05e14 0.05e14])
        hold off


        subplot(3,2,6)
        phase1 = unwrap(angle(fftOutput1));
        phase2 = unwrap(angle(fftFeedback));
        plot(omega, phase1); hold on
        plot(omega, phase2);
        xlabel('Frequency (Hz)')
        ylabel('phase (E)')
        legend('OutputR', 'Feedback');
        hold off

        pause(0.01)
        % UNCOMMENT IF YOU WANT TO REPLICATE THE PLOTS I HAVE IN MY REPORT
%         figure(2)
%         clf
%         plot(omega, 20*log10(abs(fftOutput1)), 'b'); hold on
%         plot(omega, 20*log10(abs(fftOutput2)), 'r');
%         plot(omega, 20*log10(abs(fftFeedback)), 'k--');  % From InjectedFeedback
%         xlabel('Frequency (Hz)')
%         ylabel('20 log_{10} |E|')
%         legend('OutputR', 'OutputL', 'Feedback')
%         xlim([-0.05e14 0.05e14])
%         title('Spectral Response with Feedback at 500 ps')
%         hold off
%         drawnow
%
%         figure(3)
%         clf
%         plot(time * 1e12, real(InjectedFeedback), 'k--'); hold on
%         plot(time * 1e12, real(OutputR), 'g');
%         plot(time * 1e12, real(OutputL), 'm--');
%         xlim([0, Nt * dt * 1e12])
%         ylim auto
%         xlabel('Time (ps)')
%         ylabel('E (V/\mum)')
%         legend('Injected Feedback (Right)', 'OutputR', 'OutputL', 'Location', 'east')
%         title('Field vs Time with Injected Feedback at 500 ps')
%         hold off
```

```
%           drawnow


    end


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

Listing 5: MATLAB code for Milestone 9 Part 2

7 References

[1] ELEC 4700 Course Documents, Milestone 6 – Carrier Equation, Department of Electronics, Carleton University, Ottawa, ON, Canada, 2025.

[2] ELEC 4700 Course Documents, Milestone 7 – Optical Amplifiers and SPE, Department of Electronics, Carleton University, Ottawa, ON, Canada, 2025.

[3] ELEC 4700 Course Documents, Milestone 8 – Lasers, Department of Electronics, Carleton University, Ottawa, ON, Canada, 2025.

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