

ELEC 4700 Final Report

Rish Jain, 101162547

June 29, 2025

Contents

1	Introduction and Background	1
2	Carrier Equation (Milestone 6)	2
2.1	Derivation	2
2.2	Code	3
2.3	Simulation	4
2.4	Conclusion	5
3	Optical Amplifiers and SPE (Milestone 7)	6
3.1	Derivation	6
3.2	Code	7
3.3	Simulation	8
3.4	Conclusion	11
4	Lasers (Milestone 8)	12
4.1	Derivation	12
4.2	Code	12
4.3	Simulation	12
4.4	Conclusion	19
5	Final Investigation (Milestone 9)	20
5.1	Mode Evolution During Start-Up	20
5.2	Back reflection with phase shift (Idea 4)	23
6	Appendix: Code Listings	27
6.1	Section A: Milestone 6 - Carrier Equations	27
6.2	Section B: Milestone 7 – Optical Amplifiers and SPE	30
6.3	Section C: Milestone 8 - Lasers	33
6.4	Section D: Milestone 9 – Final Investigation 1	37
6.5	Section E: Milestone 9 – Final Investigation 2	42
7	References	48

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:

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

with $S(z) = C_{\text{EtoP}} (|\hat{E}_f(z)|^2 + |\hat{E}_r(z)|^2)$, 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:

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

Multiply both sides by Δt :

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

Expand the right-hand side equation

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

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

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

Factor N_i on the left-hand side:

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

After rearranging, the final expression obtained is:

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

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

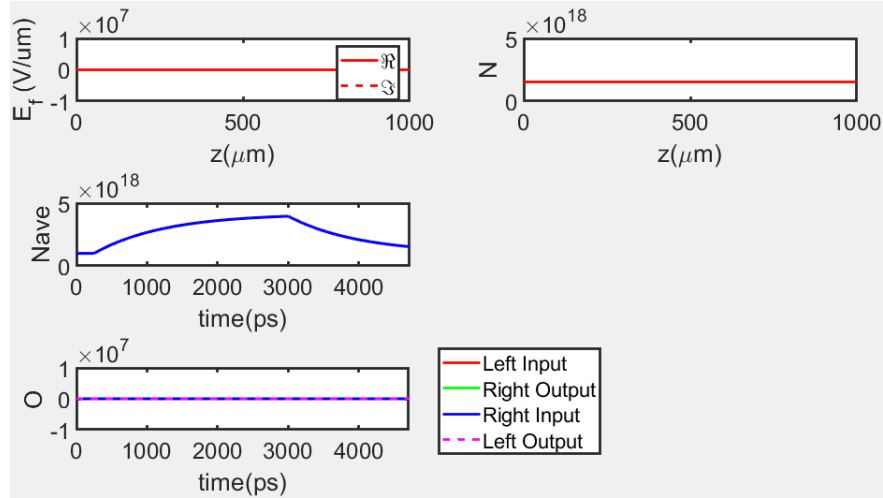


Figure 1: The change in the averaged carrier density with no optical input

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

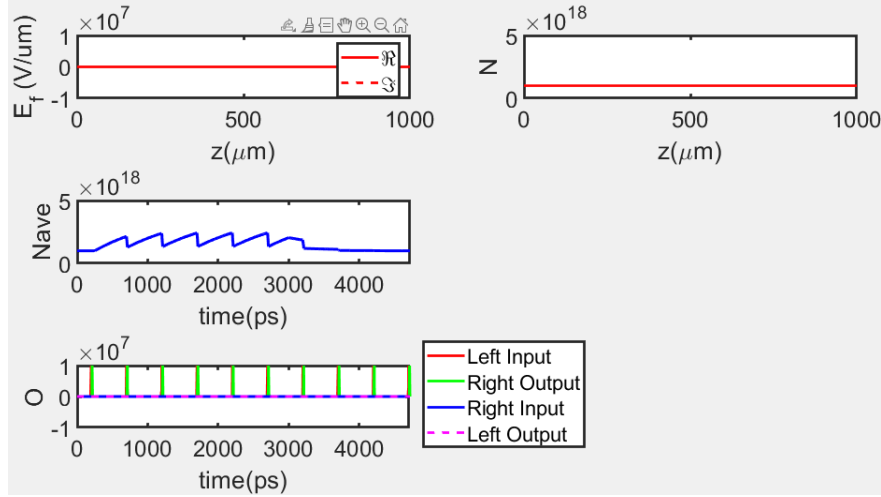


Figure 2: The change in the averaged carrier density with a gaussian optical pulse input

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:

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

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

with

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

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:

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

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

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

Here, \mathbf{eTf} and \mathbf{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 \mathbf{eTf} and \mathbf{eTr} are added to variables \mathbf{EsR} and \mathbf{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 varying 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

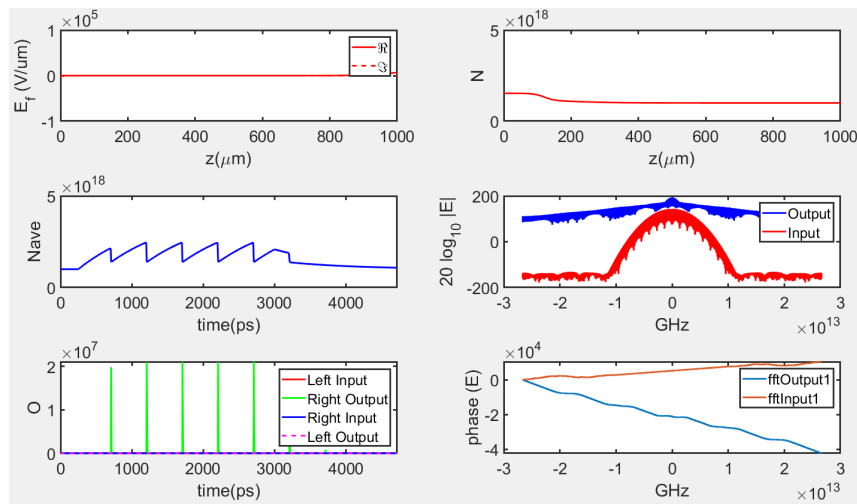


Figure 3: The amplification of the Gaussian pulse stream input with SPE turned off

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

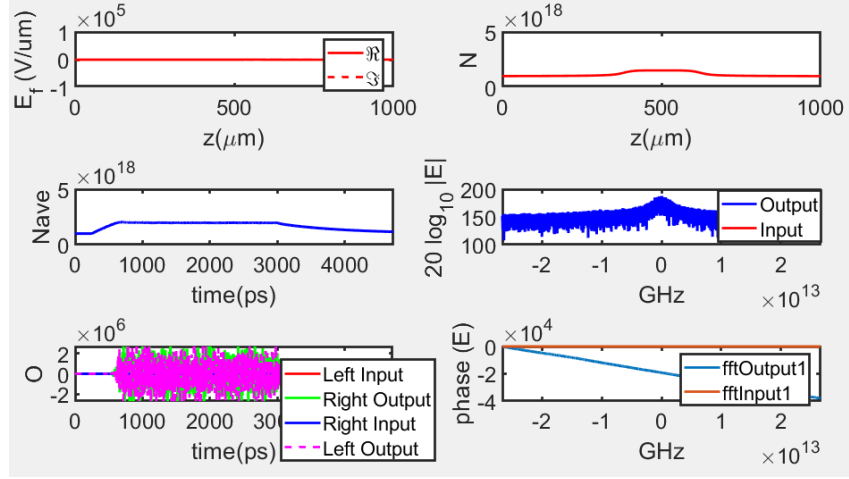


Figure 4: The change in the averaged carrier density with no optical input

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

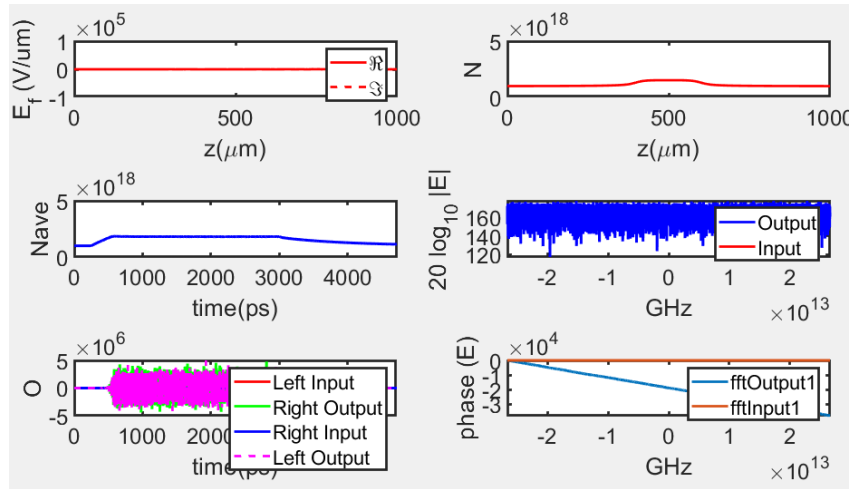


Figure 5: The change in the averaged carrier density with no optical input

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} .

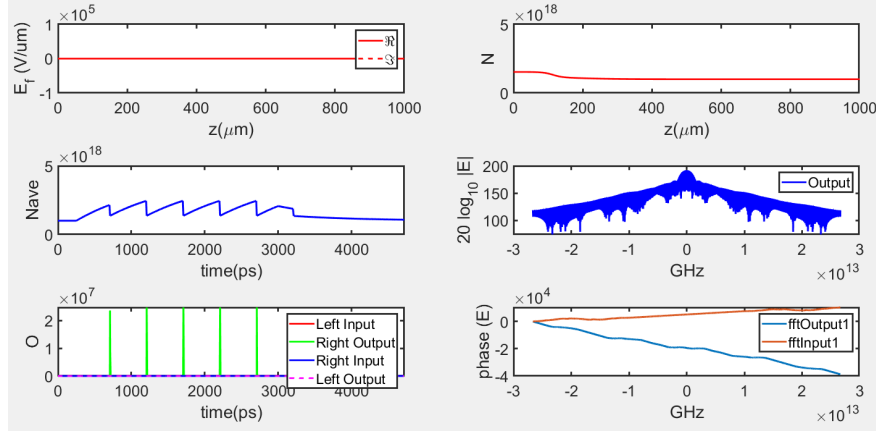


Figure 6: The response of the system to an I_D of 0.1 with an optical input

Figure 6 shows the simulation where the optical pulse's width (w_g) 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.

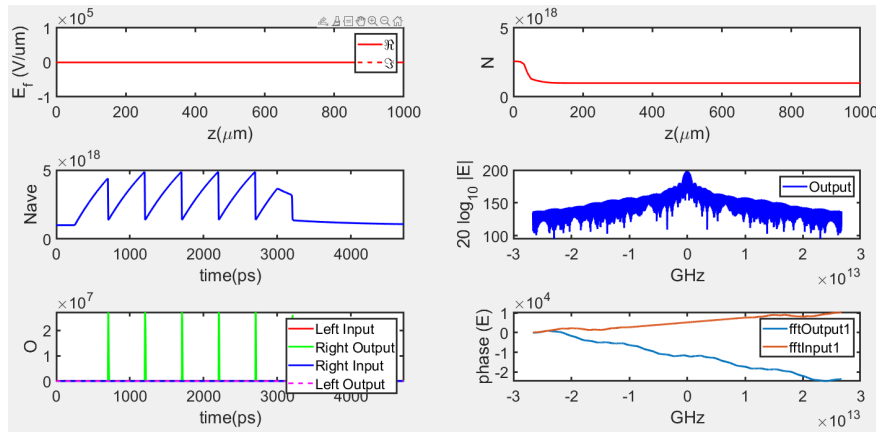


Figure 7: The response of the system to an I_D of 0.25 with an optical input

Figure 7 shows the simulation where the optical pulse's width (w_g) 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 Gaussian pulse is increased through the variable `wg`.

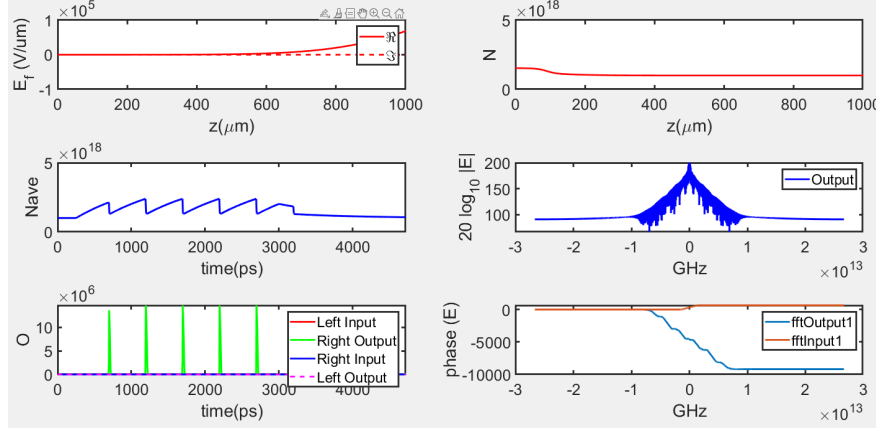


Figure 8: The response of the system when a wider Gaussian optical pulse is input

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 $\text{SPE} > 0$ 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:

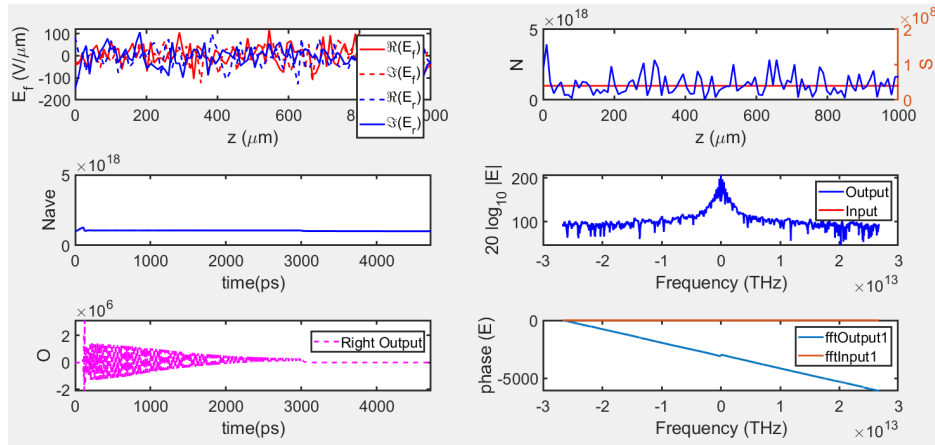


Figure 9: The systems response to spontaneous emission when mirrors are added

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.

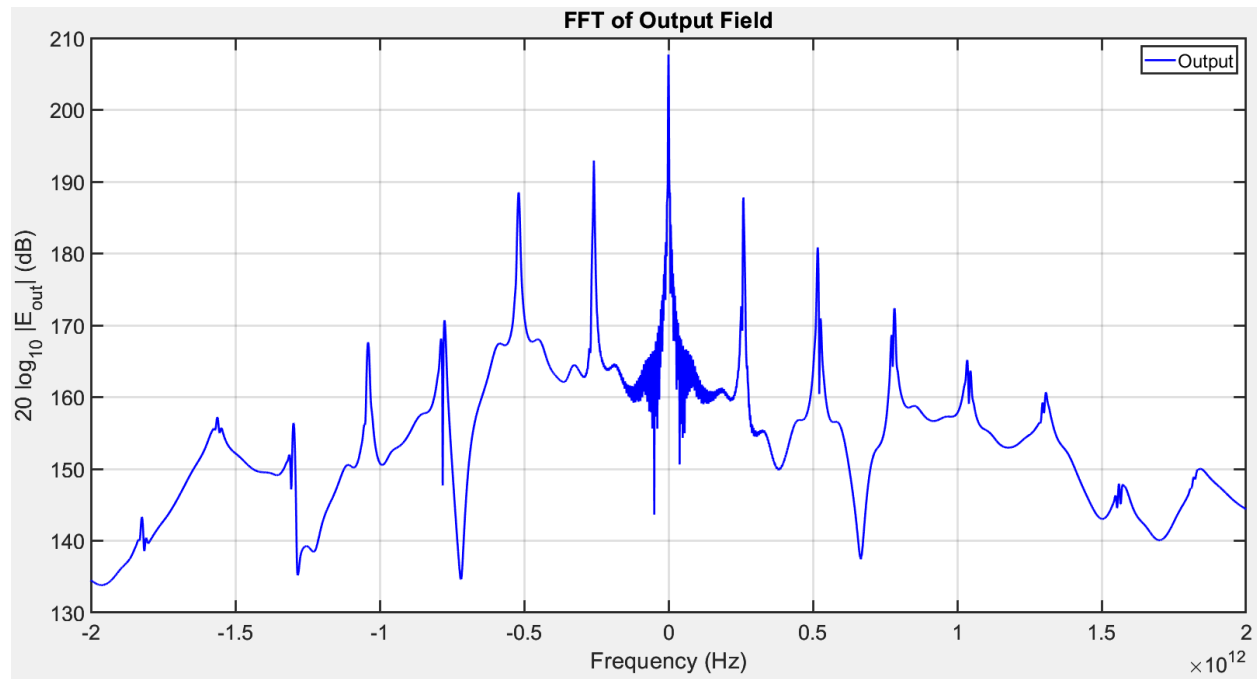


Figure 10: Closer look at the spectral response of the system by reducing the bounds of the FFT plot in Figure 9

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

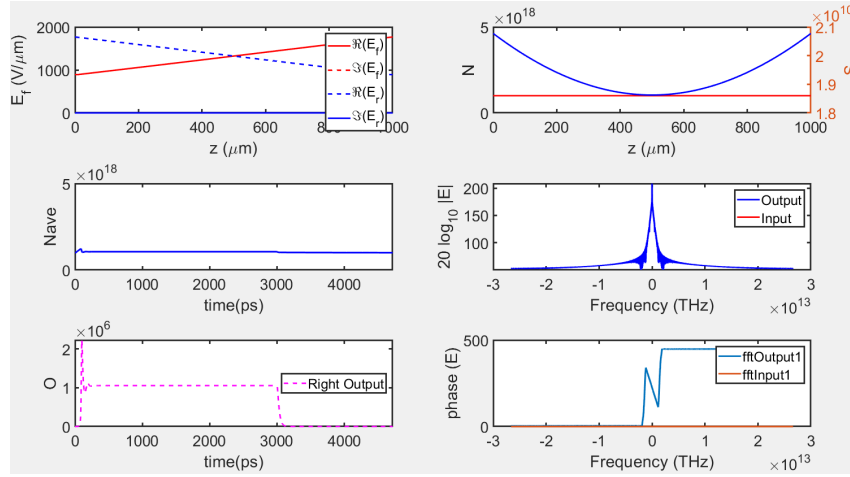


Figure 11: System's response to a deterministic spontaneous emission when mirrors are added

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?

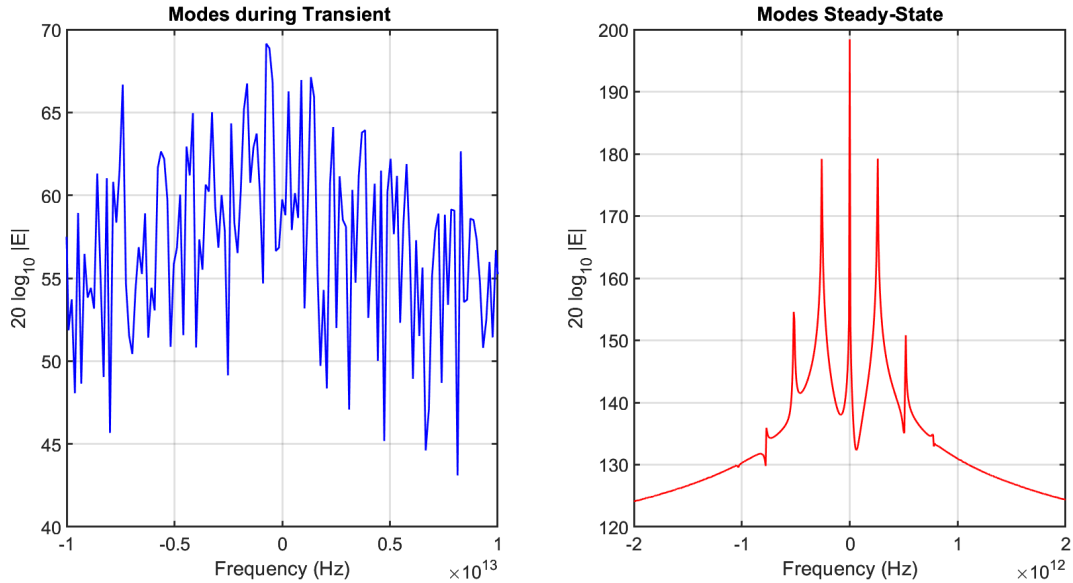


Figure 12: Comparison of the laser output spectrum during transient and steady-state

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?

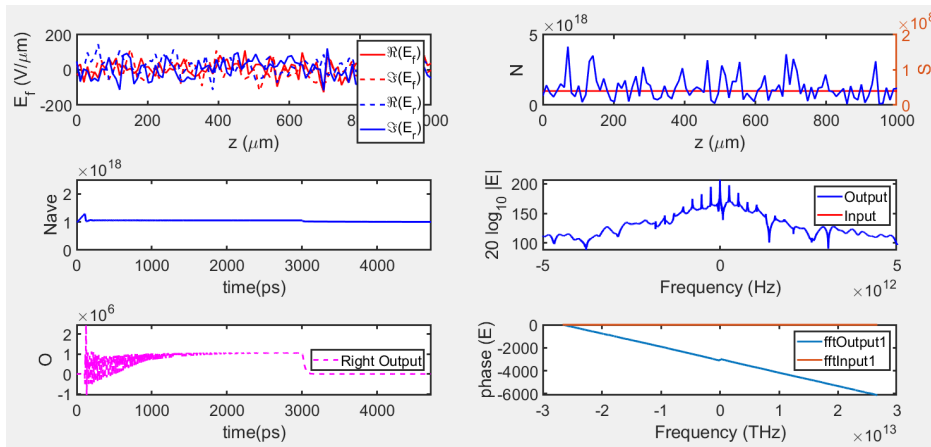


Figure 13: The lasing of the laser when the I_D is set to 0.1

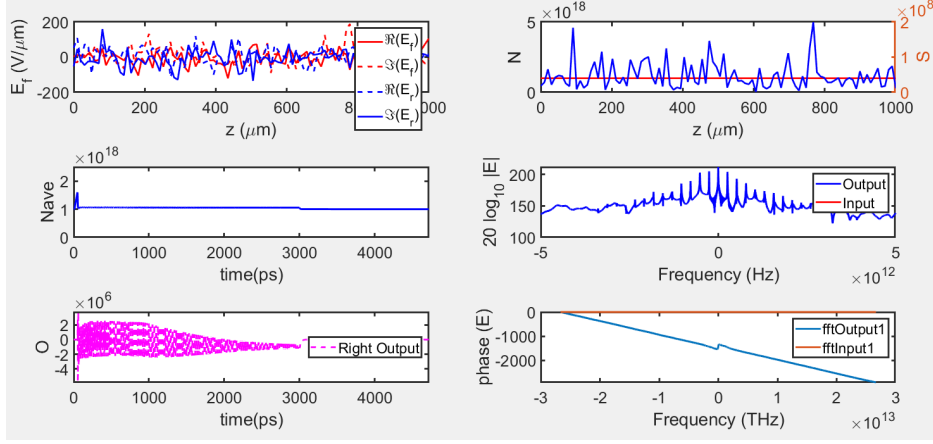


Figure 14: The lasing of the laser when the I_D is set to 0.25

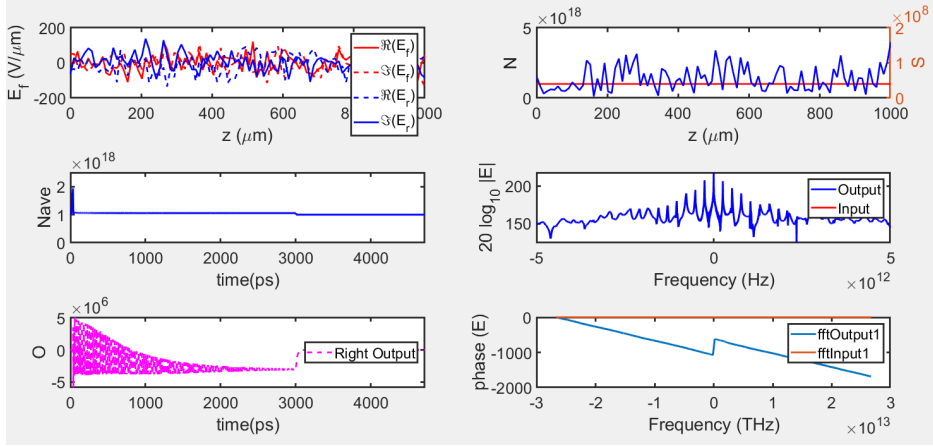


Figure 15: The lasing of the laser when the I_D is set to 0.7

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

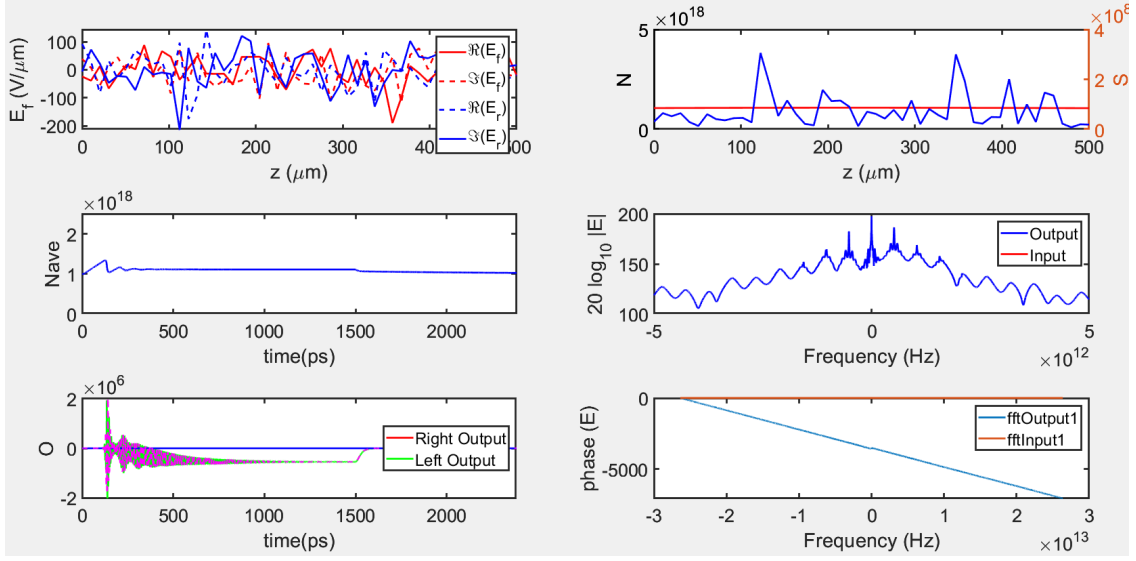


Figure 16: Lasing behavior when $I_D = 0.1$, waveguide width is $500 \mu\text{m}$, and $I_{\text{off}} = 1.5 \times 10^{-9}$

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.

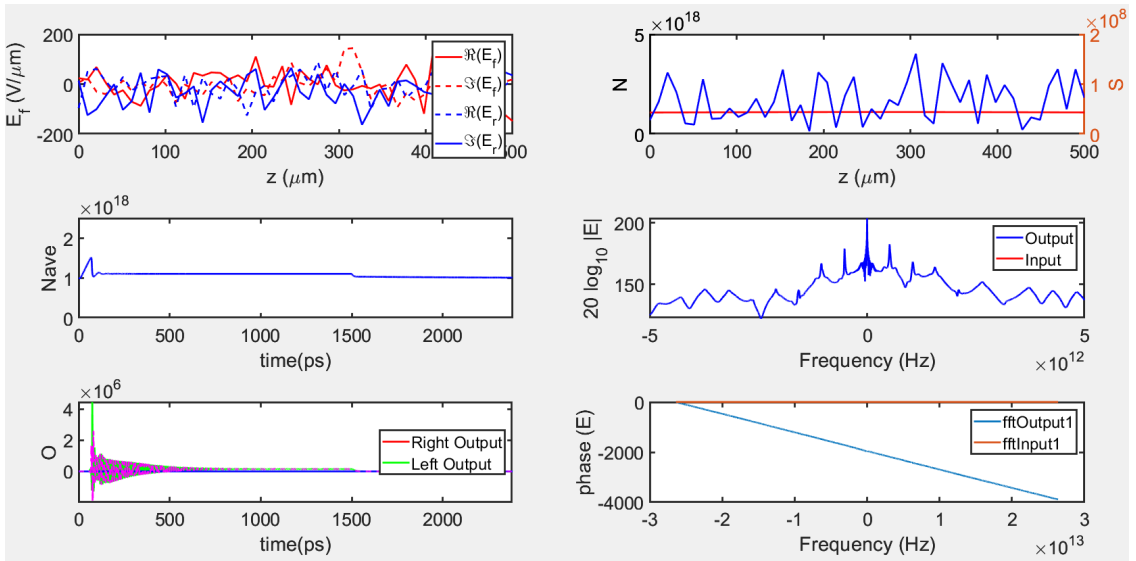


Figure 17: Lasing behavior when $I_D = 0.25$, waveguide width is $500 \mu\text{m}$, and $I_{\text{off}} = 1.5 \times 10^{-9}$

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.

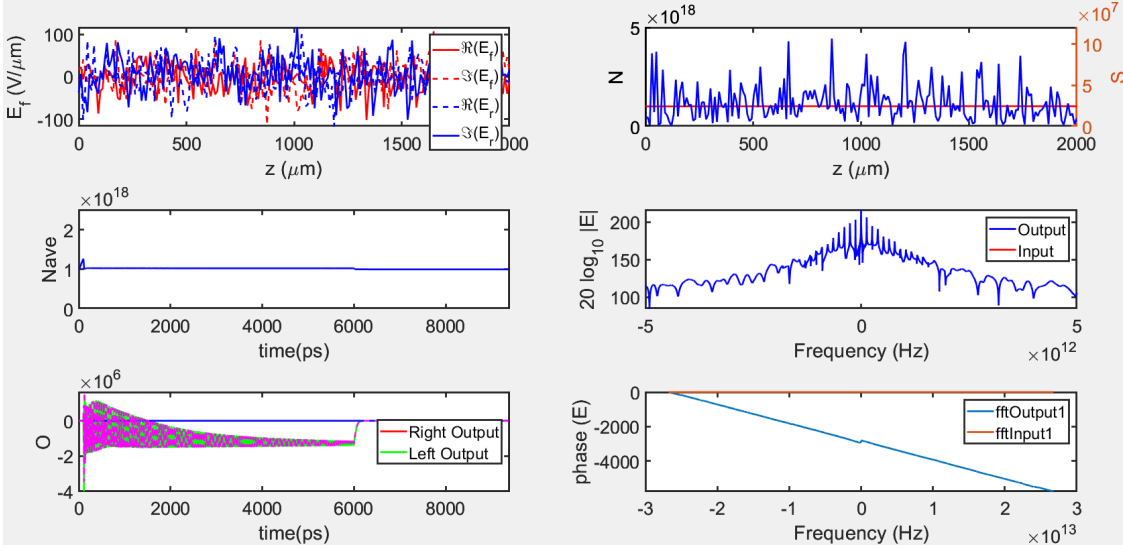


Figure 18: Lasing behavior when $I_D = 0.1$, waveguide width is $2000 \mu\text{m}$, and $I_{\text{off}} = 6 \times 10^{-9}$

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.

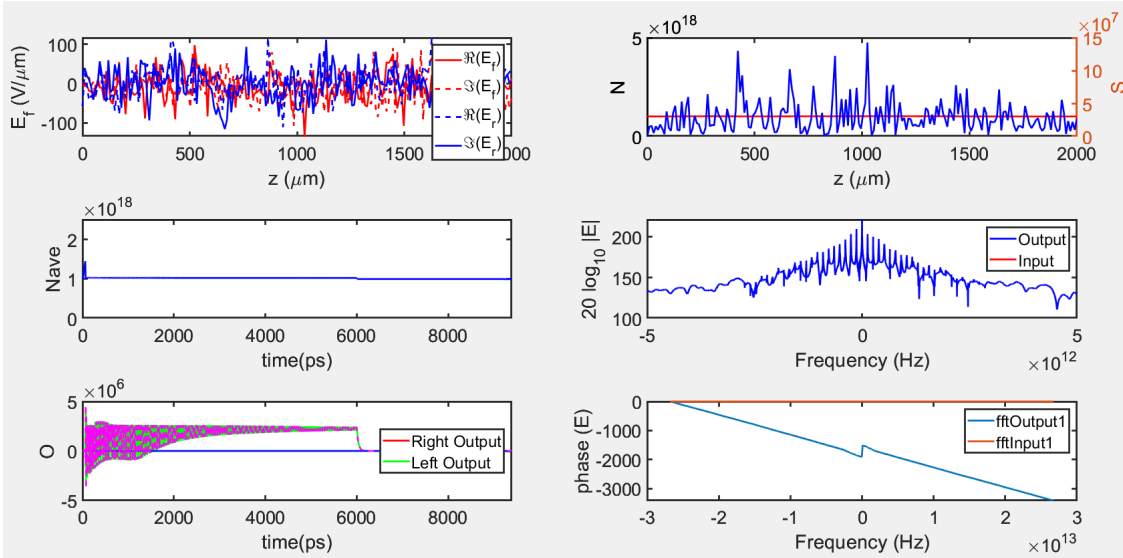


Figure 19: Lasing behavior when $I_D = 0.25$, waveguide width is $2000 \mu\text{m}$, and $I_{\text{off}} = 6 \times 10^{-9}$

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.

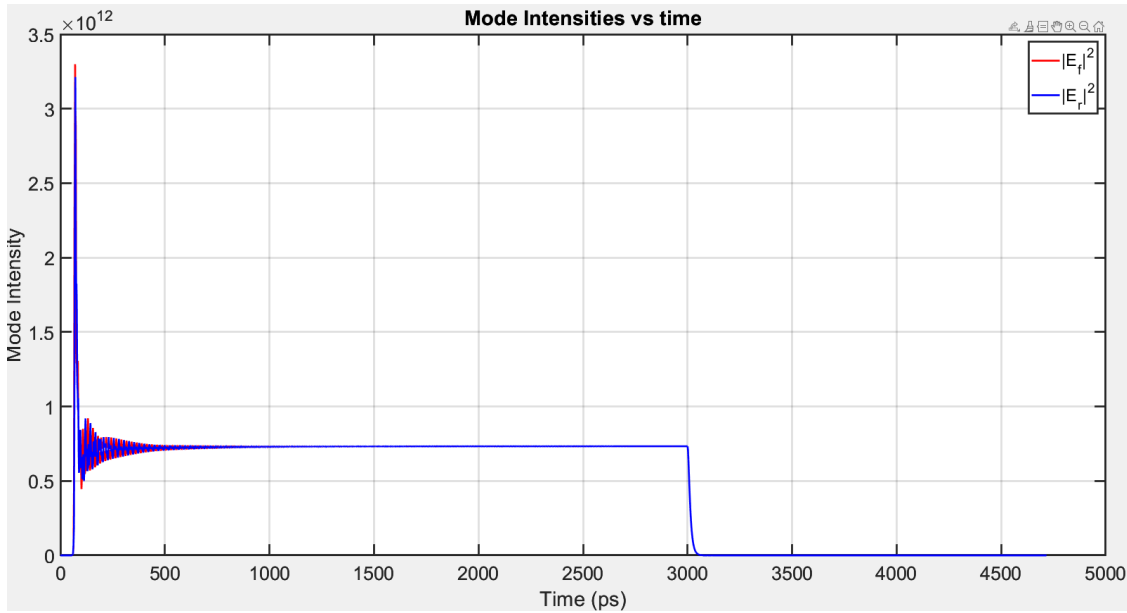


Figure 20: The mode evolution of the system with $I_d = 0.1$ along a waveguide length of $1000 \mu\text{m}$

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.

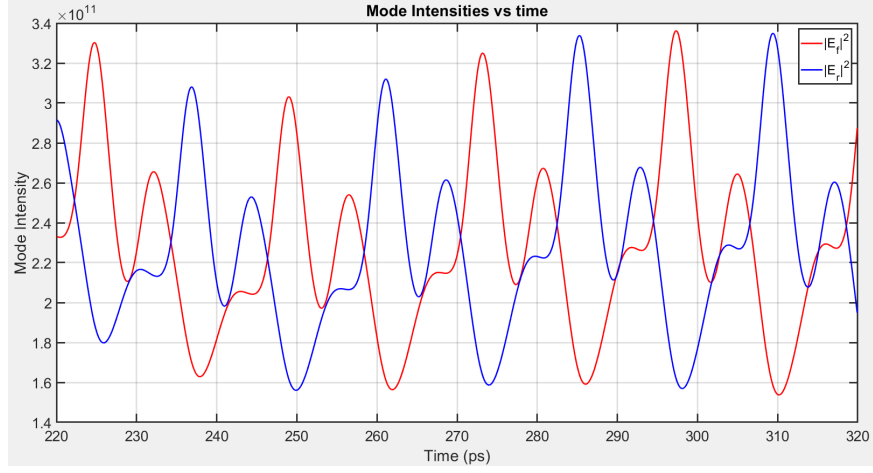


Figure 21: A zoomed-in version of the mode evolution of the system with the same parameters as the one in the figure above

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.

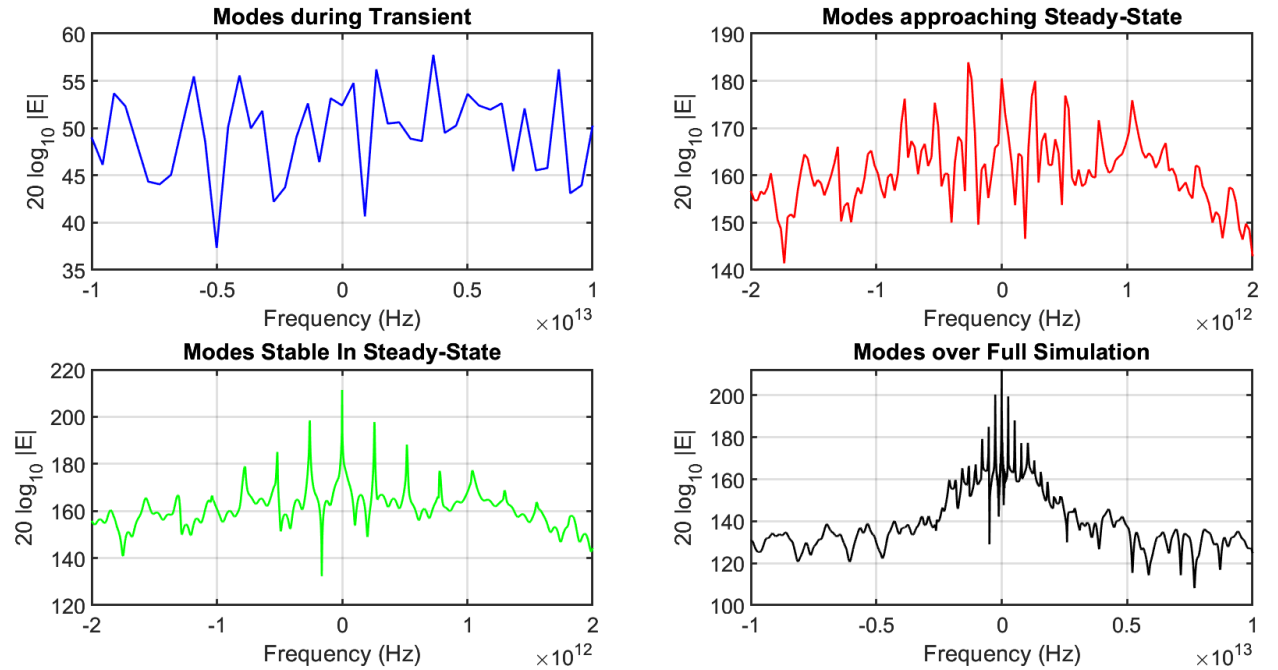


Figure 22: The mode intensity of the system over time

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:

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

For a gain of 220 dB:

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

For a gain of 180 dB:

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

The ratio between the two linear gains is:

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

So, a difference of 40 dB corresponds to a $100\times$ 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.

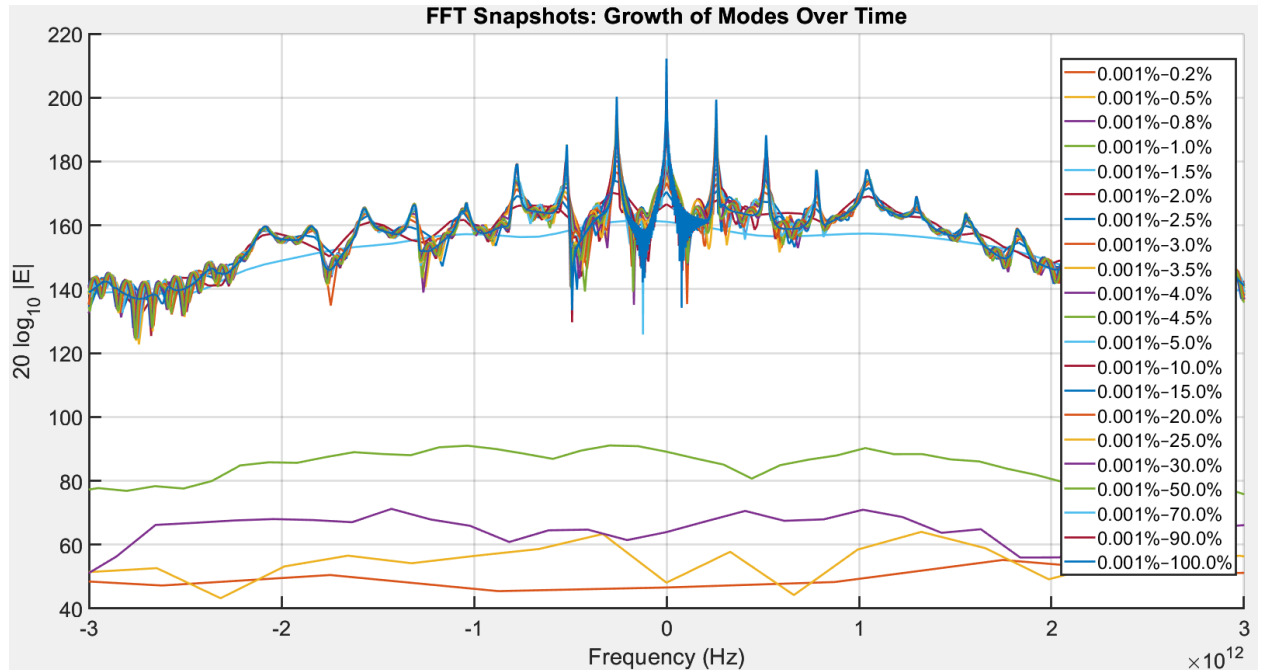


Figure 23: Mode evolution of the laser system as time increases

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 see 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.

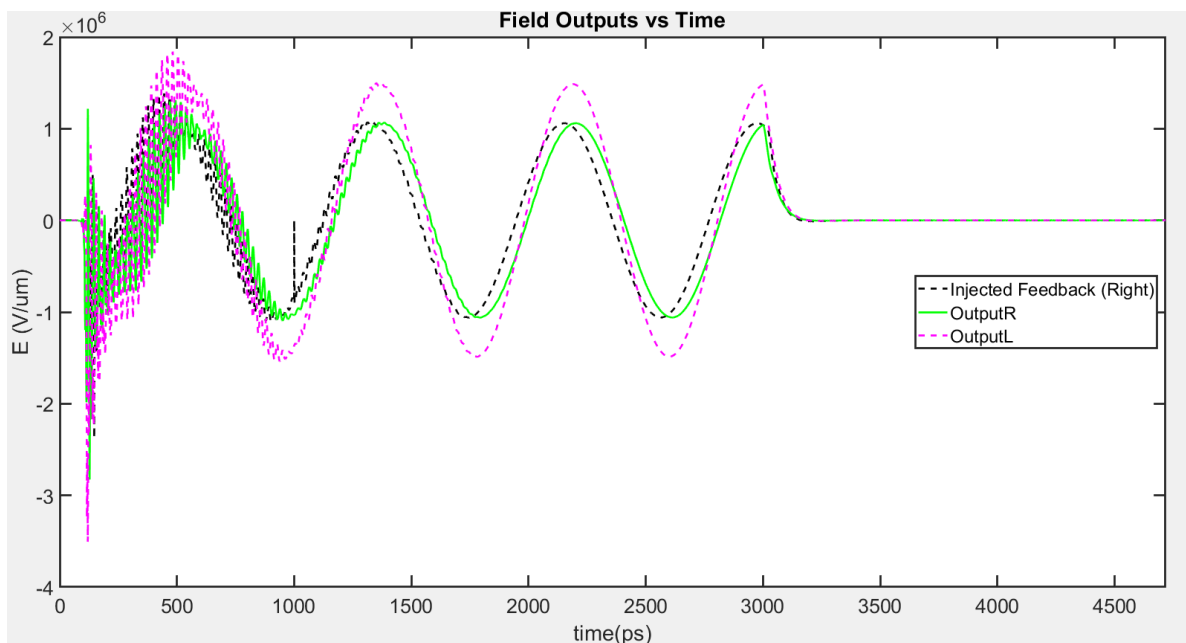


Figure 24: The characteristic of lasing looking to move in like a sine wave

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.

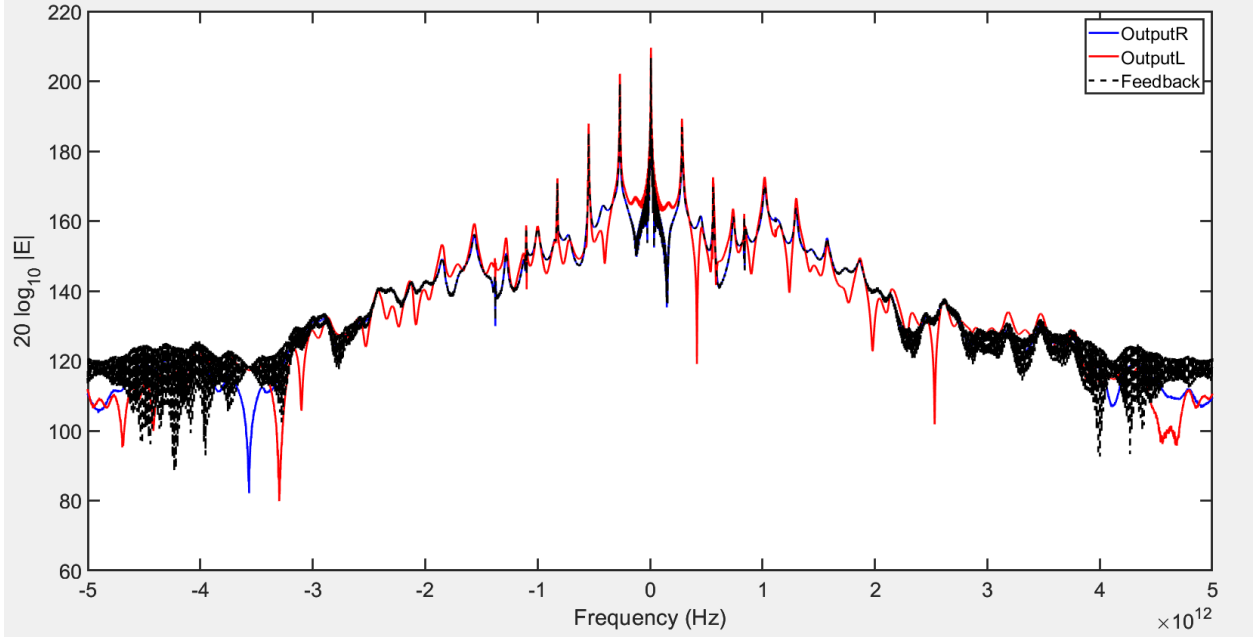


Figure 25: The spectral response of the system with respect to the feedback with a $\pi/6$ phase shift coming into the system

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.

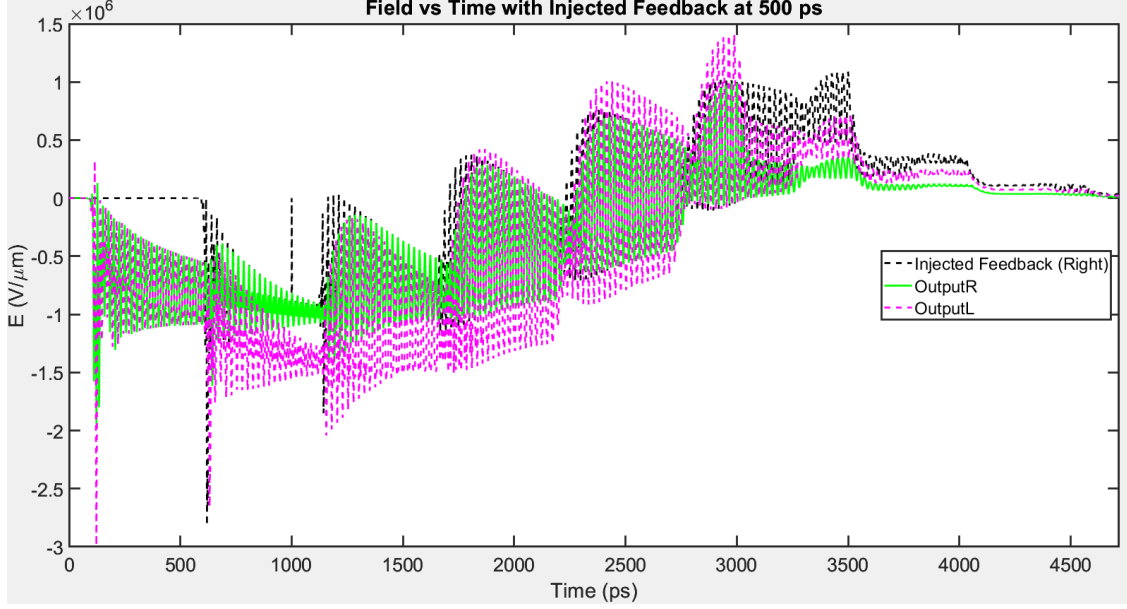


Figure 26: The time evolution of the fields with respect to the feedback with a $\pi/6$ phase shift coming into the system and a delay of 500 ps

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.

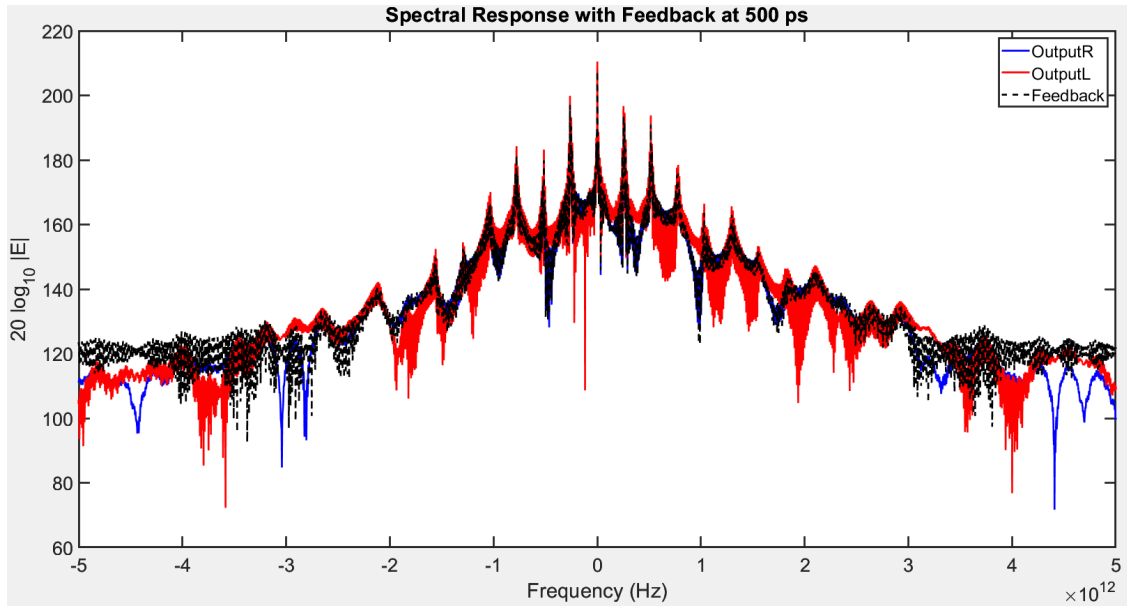


Figure 27: The spectral response of the system with respect to the feedback with a $\pi/6$ phase shift coming into the system and a delay of 500 ps

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

```
1 set(0,'defaultaxesfontsize',20)
2 set(0,'DefaultFigureWindowStyle','docked')
3 set(0,'DefaultLineLineWidth',2);
4 set(0,'Defaultaxeslinewidth',2)
5
6 set(0,'DefaultFigureWindowStyle','docked')
7
8 c_c = 299792458;           % m/s TWM speed of light
9 c_eps_0 = 8.8542149e-12;   % F/m vaccum permittivity
10 c_eps_0_cm = c_eps_0/100;  % F/cm vaccum permittivity
11 c_mu_0 = 1/c_eps_0/c_c^2;  % Permiability of free space
12 c_q = 1.60217653e-19;     % Charge of an electron
13 c_hb = 1.05457266913e-34;  % Dirac / Reduced Planck constant
14 c_h = c_hb*2*pi;          % Planck constant
15
16 beta_r = 0;               % De-tuning constant
17 beta_i = 0;               % Gain Constant
18
19 kappa0 = 0;                % Coupling coefficient
20 kappaStart = 1/3;          % Constant defines starting position where coupling begins.
21 kappaStop = 2/3;           % Constant defines ending position where coupling stops.
22
23 InputParasL.E0 = 100e5;    % Amplitude of the input E-field / E_f
24 InputParasL.we = 0;        % Frequency of the complex sinusoidal modulation on the gaussian pulse
25 InputParasL.t0 = 200e-12;  % The constant we are shifting the time by
26 InputParasL.wg = 10e-13;   % Width of the Gaussian distribution
27 InputParasL.phi = 0;       % Initial Phase of the E_f / input E-field
28 InputParasR = 0;           % Placeholder variable for reverse propagation
29 InputParasL.rep = 500e-12;
30
31 n_g = 3.5;                 % Constant to control group velocity
32 vg = c_c/n_g *1e2;         % TWM cm/s group velocity
33
34 Lambda = 1550e-9;          % Wavelength of light
35 f0 = c_c/Lambda;
36
37 plotN = 10;                % Divisor constant
38 L = 1000e-6*1e2;           % length of the waveguide in cm
39
40 % Material Polarization Information
41 g_fwhm = 0;                 % Frequency
42 LGamma = g_fwhm*2*pi;
43 Lw0 = 0;
44 LGain = 0;                  % Gain Constant
45
46 XL = [0,L];                % Start and End of the x-axis
47 YL = [-InputParasL.E0,InputParasL.E0]; % Start and End of the y-axis
48
49 Nz = 100;                  % Number of divisions
50 dz = L/(Nz-1);             % Distance between every point
51 dt = dz/vg;                % Time taken to plot every point
52 fsync = dt*vg/dz;          % Equals 1, allows the Gaussian to be stable
53
54 Nt = floor(400*Nz);        % Time steps
55 tmax = Nt*dt;              % Maximum time for simulation
56 t_L = dt*Nz;               % time to travel length
57 z = linspace(0,L,Nz);      % Nz points, Nz-1 segments
58 time = nan(1,Nt);          % Time matrix with 1 row and Nt columns / row vector of Nt elements
59
60 InputL = nan(1,Nt);         % Matrix with 1 row and Nt columns / row vector of Nt elements
61 InputR = nan(1, Nt);        % Matrix with 1 row and Nt columns / row vector of Nt elements
62 OutputL = nan(1,Nt);        % Matrix with 1 row and Nt columns / row vector of Nt elements
63 OutputR = nan(1,Nt);        % Matrix with 1 row and Nt columns / row vector of Nt elements
64 Ef = zeros(size(z));        % Matrix with the same dimensions as z, all elements initialized to 0
65 Er = zeros(size(z));        % Matrix with the same dimensions as z, all elements initialized to 0
66
67 Ef1 = @SourceFct;           % Reference to SourceFct
68 ErN = @SourceFct;           % Reference to SourceFct
69
70 t = 0;                      % Set t to a starting value of 0
71 time(1) = t;                % Sets the first element of the time vector to 0
72
73 InputL(1) = Ef1(t, InputParasL); % Set initial value of InputL using the source function
74 InputR(1) = ErN(t, InputParasR); % Set initial value of InputR using the source function
75
76 OutputR(1) = Ef(Nz);         % The end of the waveguide is the first value of the reflection (Right to Left)
77 OutputL(1) = Er(1);          % The end of the waveguide is the first value of the reflection (Left to Right)
78
79 Ef(1) = InputL(1);           % Initializes forward field at z = 0 (Input signal from the left)
80 Er(Nz) = InputR(1);          % Initializes backward field at z = L (Input signal from the right)
81
82 kappa = kappa0*ones(size(z)); % Creates an array of size z, where all indexes hold a value of kappa0
83 kappa(z<L*kappaStart) = 0;   % Sets the limit such that kappa is set to zero outside the interaction region
84 kappa(z>L*kappaStop) = 0;     % Sets the limit such that kappa is set to zero outside the interaction region.
```

```

85
86 Pf = zeros(size(z));           % Variable for the polarization of the material on the forward field
87 Pr = zeros(size(z));           % Variable for the polarization of the material on the reverse field
88
89 % Variables to hold field and polarization information
90 Efp = Ef;
91 Erp = Er;
92 Pfp = Pf;
93 Prp = Pr;
94
95 Nave = nan(1,Nt);
96 Ntr = 1e18;
97 N = ones(size(z))*Ntr;
98 Nave(1) = mean(N);
99
100 gain = vg*2.5e-16;
101 eVol = 1.5e-10*c_q;
102 Ion = 0.25e-9;
103 Ioff = 3e-9;
104 I_off = 0.024;
105 I_on = 0.1;
106 taun = 1e-9;
107 Zg = sqrt(c_mu_0/c_eps_0)/n_g;
108 EtoP = 1/(Zg*f0*vg*1e-2*c_hb);
109 alpha = 0;
110
111 figure('name', 'Fields')
112
113 % Forward field E_f
114 subplot(3,2,1)
115 plot(z*10000, real(Ef), 'r'); hold on
116 plot(z*10000, imag(Ef), 'r--'); hold off
117 xlim(XL*1e4)
118 ylim(YL)
119 xlabel('z(\mum)')
120 ylabel('E_f (V/um)')
121 legend('\Re', '\Im')
122
123 % Carrier Density N
124 subplot(3,2,2)
125 plot(z*10000, N, 'r');
126 xlim(XL * 1e4)
127 ylim([0, 5 * Ntr])
128 xlabel('z(\mum)')
129 ylabel('N')
130
131 % Average Carrier Density Over Time
132 subplot(3,2,3)
133 plot(time * 1e12, Nave, 'b');
134 xlim([0, Nt * dt * 1e12])
135 ylim([0, 5 * Ntr])
136 xlabel('time(ps)')
137 ylabel('Nave')
138
139 % Input and Output Fields over time
140 subplot(3,2,5)
141 plot(time * 1e12, real(InputL), 'r'); hold on
142 plot(time * 1e12, real(OutputR), 'g');
143 plot(time * 1e12, real(InputR), 'b');
144 plot(time * 1e12, real(OutputL), 'm--');
145 xlim([0, Nt * dt * 1e12])
146 ylim(YL)
147 xlabel('time(ps)')
148 ylabel('E (V/um)')
149 legend('Left Input', 'Right Output', 'Right Input', 'Left Output', 'Location', 'east')
150 hold off
151
152
153 for i = 2:Nt           % 2 to 1000 in steps of 1
154
155     t = dt*(i-1);
156     time(i) = t;
157
158     RL = 0;           % The left side reflection coefficient
159     RR = 0;           % The right side reflection coefficient
160
161     beta = ones(size(z))*(beta_r+1i*beta_i); % Complex propagation constant
162     exp_det = exp(-1i*dz*beta);           % Phase shift due to propagation over a distance dz
163
164     % Input
165     InputL(i) = Ef1(t, 0); % At time t, we input a signal characterized by InputParasL from the left
166     InputR(i) = ErN(t, 0); % At time t, we input no signal from the right (since InputParasR = 0)
167
168     % Reflection
169     Ef(1) = InputL(i) + RL*Er(1); % Boundary condition at z = 0 (left side);
170     Er(Nz) = InputR(i) + RR*Ef(Nz); % Boundary condition at z = L (right side);
171
172     Ef(2:Nz) = fsync*exp_det(1:Nz-1).*Ef(1:Nz-1) + 1i*dz*kappa(2:Nz).*Er(2:Nz); % Forward Field Propagation
173     Er(1:Nz-1) = fsync*exp_det(2:Nz).*Er(2:Nz) + 1i*dz*kappa(2:Nz).*Ef(2:Nz); % Reverse Field Propagation
174
175     % Boundary Conditions
176     Pf(1) = 0; % zero polarization at the left boundary

```

```

177 Pf(Nz) = 0; % zero polarization at the right boundary
178 Pr(1) = 0; % zero polarization at the right boundary
179 Pr(Nz) = 0; % zero polarization at the left boundary
180 Cw0 = -LGamma + 1i * Lw0; % Defines the complex response function of the material.
181
182 % Dispersion Calculations
183 % Backward Euler Polarization Update
184 % Forward polarization
185 Tf = LGamma * Efp(2:Nz-1); % Source term for forward polarization
186 Pf(2:Nz-1) = (Pfp(2:Nz-1) + dt * Tf) ./ (1 - dt * Cw0); % Forward polarization update
187
188 % Backward polarization
189 Tr = LGamma * Erp(2:Nz-1); % Source term for backward polarization
190 Pr(2:Nz-1) = (Prp(2:Nz-1) + dt * Tr) ./ (1 - dt * Cw0); % Backward polarization update
191
192 Ef(2:Nz-1) = Ef(2:Nz-1) - LGain * (Ef(2:Nz-1) - Pf(2:Nz-1)); % Adjusts the forward electric field
193 Er(2:Nz-1) = Er(2:Nz-1) - LGain * (Er(2:Nz-1) - Pr(2:Nz-1)); % Adjusts the reverse electric field
194
195 % Output
196 OutputR(i) = Ef(Nz) * (1 - RR); % Right output at z = L
197 OutputL(i) = Er(1) * (1 - RL); % Left output at z = 0
198
199 S = (abs(Ef).^2 + abs(Er).^2) * EtoP * 1e-6;
200
201 if t < Ion || t > Ioff
202     I_injv = I_off;
203 else
204     I_injv = I_on;
205 end
206
207 Stim = gain * (N - Ntr) * S;
208 N = (N + dt * (I_injv / eVol - Stim)) ./ (1 + dt / taun);
209 Nave(i) = mean(N);
210
211 % % FFT data from the outputs
212 % fftOutput1 = fftshift(fft(OutputR)); % Get FFT data for OutputR
213 % fftOutput2 = fftshift(fft(OutputL)); % Get FFT data for OutputL
214 % fftInput1 = fftshift(fft(InputL)); % Get FFT data for InputL
215 % omega = fftshift(wspace(time));
216
217 if mod(i,1000) == 0 % Only executed when i is multiple of plotN
218
219     % Forward Propagation of the Gaussian Pulse
220     subplot(3,2,1)
221     plot(z * 10000, real(Ef), 'r'); hold on
222     plot(z * 10000, imag(Ef), 'r--'); hold off
223     xlim(XL * 1e4)
224     ylim(YL)
225     xlabel('z (\mum)')
226     ylabel('E_f (V/um)')
227     legend('Re', 'Im')
228     hold off
229
230     % Carrier Density N
231     subplot(3,2,2)
232     plot(z * 10000, N, 'r');
233     xlim(XL * 1e4)
234     ylim([0, 5 * Ntr])
235     xlabel('z (\mum)')
236     ylabel('N')
237
238     % Average Carrier Density Over Time
239     subplot(3,2,3)
240     plot(time * 1e12, Nave, 'b');
241     xlim([0, Nt * dt * 1e12])
242     ylim([0, 5 * Ntr])
243     xlabel('time(ps)')
244     ylabel('Nave')
245
246     % Input and Output Fields over time
247     subplot(3,2,5)
248     plot(time * 1e12, real(InputL), 'r'); hold on
249     plot(time * 1e12, real(OutputR), 'g');
250     plot(time * 1e12, real(InputR), 'b');
251     plot(time * 1e12, real(OutputL), 'm--');
252     xlim([0, Nt * dt * 1e12])
253     ylim(YL)
254     xlabel('time(ps)')
255     ylabel('0')
256     legend('Left Input', 'Right Output', 'Right Input', 'Left Output', 'Location', 'east')
257     hold off
258     pause(0.01)
259 end
260
261 % Update Previous Values
262 Efp = Ef;
263 Erp = Er;
264 Pfp = Pf;
265 Prp = Pr;
266 end

```

6.2 Section B: Milestone 7 – Optical Amplifiers and SPE

```

1
2 set(0,'defaultaxesfontsize',20)
3 set(0,'DefaultFigureWindowStyle','docked')
4 set(0,'DefaultLineLineWidth',2);
5 set(0,'Defaultaxeslinewidth',2)
6
7 set(0,'DefaultFigureWindowStyle','docked')
8
9 c_c = 299792458;           % m/s TWM speed of light
10 c_eps_0 = 8.8542149e-12;   % F/m vaccum permittivity
11 c_eps_0_cm = c_eps_0/100;  % F/cm vaccum permittivity
12 c_mu_0 = 1/c_eps_0/c_c^2;  % Permiability of free space
13 c_q = 1.60217653e-19;     % Charge of an electron
14 c_hb = 1.05457266913e-34;  % Dirac / Reduced Planck constant
15 c_h = c_hb*2*pi;          % Planck constant
16
17 beta_r = 0;               % De-tuning constant
18 beta_i = 0;               % Gain Constant
19
20
21 beta_spe = .3e-5;
22 gamma = 1.0;
23 % SPE = 7;
24 SPE = 0;
25
26 kappa0 = 0;               % Coupling coefficient
27 kappaStart = 1/3;         % Constant defines starting position where coupling begins.
28 kappaStop = 2/3;         % Constant defines ending position where coupling stops.
29
30 InputParasL.E0 = 1e5;     % Amplitude of the input E-field / E_f
31 InputParasL.we = 0;       % Frequency of the complex sinusoidal modulation on the gaussian pulse
32 InputParasL.t0 = 200e-12; % The constant we are shifting the time by
33 InputParasL.wg = 50e-13;  % Width of the Gaussian distribution
34 InputParasL.phi = 0;      % Initial Phase of the E_f / input E-field
35 InputParasR = 0;          % Placeholder variable for reverse propagation
36 InputParasL.rep = 500e-12;
37
38 n_g = 3.5;                % Constant to control group velocity
39 vg = c_c/n_g *1e2;        % TWM cm/s group velocity
40
41 Lambda = 1550e-9;         % Wavelength of light
42 f0 = c_c/Lambda;
43
44 plotN = 10;               % Divisor constant
45
46 L = 1000e-6*1e2;         % length of the waveguide in cm
47
48 % Material Polarization Information
49 g_fwhm = 3.5e+012/10;    % Frequency
50 LGamma = g_fwhm*2*pi;
51 Lw0 = 0;
52 LGain = 0.05;            % Gain Constant
53
54 XL = [0,L];              % Start and End of the x-axis
55 YL = [-InputParasL.E0,InputParasL.E0]; % Start and End of the y-axis
56
57
58 Nz = 100;                % Number of divisions
59 dz = L/(Nz-1);           % Distance between every point
60 dt = dz/vg;              % Time taken to plot every point
61 fsync = dt*vg/dz;        % Equals 1, allows the Gaussian to be stable
62
63 Nt = floor(400*Nz);      % Time steps
64 tmax = Nt*dt;            % Maximum time for simulation
65 t_L = dt*Nz;             % time to travel length
66 z = linspace(0,L,Nz);    % Nz points, Nz-1 segments
67 time = nan(1,Nt);         % Time matrix with 1 row and Nt columns / row vector of Nt elements
68
69 InputL = nan(1,Nt);       % Matrix with 1 row and Nt columns / row vector of Nt elements
70 InputR = nan(1, Nt);      % Matrix with 1 row and Nt columns / row vector of Nt elements
71 OutputL = nan(1,Nt);     % Matrix with 1 row and Nt columns / row vector of Nt elements
72 OutputR = nan(1,Nt);     % Matrix with 1 row and Nt columns / row vector of Nt elements
73 Ef = zeros(size(z));     % Matrix with the same dimensions as z, all elements initialized to 0
74 Er = zeros(size(z));     % Matrix with the same dimensions as z, all elements initialized to 0
75
76 Ef1 = @SourceFct;        % Reference to SourceFct
77 ErN = @SourceFct;        % Reference to SourceFct
78
79 t = 0;                   % Set t to a starting value of 0
80 time(1) = t;             % Sets the first element of the time vector to 0

```



```

81
82 InputL(1) = Ef1(t, InputParasL); % Set initial value of InputL using the source function
83 InputR(1) = ErN(t, InputParasR); % Set initial value of InputR using the source function
84
85 OutputR(1) = Ef(Nz); % The end of the waveguide is the first value of the reflection (Right to Left)
86 OutputL(1) = Er(1); % The end of the waveguide is the first value of the reflection (Left to Right)
87
88 Ef(1) = InputL(1); % Initializes forward field at z = 0 (Input signal from the left)
89 Er(Nz) = InputR(1); % Initializes backward field at z = L (Input signal from the right)
90
91 kappa = kappa0*ones(size(z)); % Creates an array of size z, where all indexes hold a value of kappa0
92 kappa(z<L*kappaStart) = 0; % Sets the limit such that kappa is set to zero outside the interaction region
93 kappa(z>L*kappaStop) = 0; % Sets the limit such that kappa is set to zero outside the interaction region.
94
95 Pf = zeros(size(z)); % Variable for the polarization of the material on the forward field
96 Pr = zeros(size(z)); % Variable for the polarization of the material on the reverse field
97
98 % Variables to hold field and polarization information
99 Efp = Ef;
100 Erp = Er;
101 Pfp = Pf;
102 Prp = Pr;
103
104 Nave = nan(1,Nt);
105 Ntr = 1e18;
106 N = ones(size(z))*Ntr;
107 Nave(1) = mean(N);
108
109 gain = vg*2.5e-16;
110 eVol = 1.5e-10*c_q;
111 Ion = 0.25e-9;
112 Ioff = 3e-9;
113 % I_off = 0.024;
114 % I_on = 0.1;
115 I_off = 0.024;
116 I_on = 0.25;
117 taun = 1e-9;
118 Zg = sqrt(c_mu_0/c_eps_0)/n_g;
119 EtoP = 1/(Zg*f0*vg*1e-2*c_hb);
120 alpha = 0;
121
122 figure('name', 'Fields')
123
124 % Forward field E_f
125 subplot(3,2,1)
126 plot(z*10000, real(Ef), 'r'); hold on
127 plot(z*10000, imag(Ef), 'r--'); hold off
128 xlim(XL*1e4)
129 ylim(YL)
130 xlabel('z(\mum)')
131 ylabel('E_f (V/um)')
132 legend('\Re', '\Im')
133
134 % Carrier Density N
135 subplot(3,2,2)
136 plot(z*10000, N, 'r');
137 xlim(XL * 1e4)
138 ylim([0, 5 * Ntr])
139 xlabel('z(\mum)')
140 ylabel('N')
141
142 % Average Carrier Density Over Time
143 subplot(3,2,3)
144 plot(time * 1e12, Nave, 'b');
145 xlim([0, Nt * dt * 1e12])
146 ylim([0, 5 * Ntr])
147 xlabel('time(ps)')
148 ylabel('Nave')
149
150 % Input and Output Fields over time
151 subplot(3,2,5)
152 plot(time * 1e12, real(InputL), 'r'); hold on
153 plot(time * 1e12, real(OutputR), 'g');
154 plot(time * 1e12, real(InputR), 'b');
155 plot(time * 1e12, real(OutputL), 'm--');
156 xlim([0, Nt * dt * 1e12])
157 ylim(YL)
158 xlabel('time(ps)')
159 ylabel('E (V/um)')
160 legend('Left Input', 'Right Output', 'Right Input', 'Left Output', 'Location', 'east')
161 hold off
162
163
164 for i = 2:Nt % 2 to 1000 in steps of 1
165
166     t = dt*(i-1);
167     time(i) = t;
168
169     RL = 0; % The left side reflection coefficient
170     RR = 0; % The right side reflection coefficient
171
172     % Input

```

```

173 InputL(i) = Ef1(t,InputParasL);
174 % InputL(i) = Ef1(t, InputParasL); % At time t, we input a signal characterized by InputParasL from the left
175 InputR(i) = ErN(t, 0); % At time t, we input no signal from the right (since InputParasR = 0)
176
177 % Reflection
178 Ef(1) = InputL(i) + RL*Er(1); % Boundary condition at z = 0 (left side);
179 Er(Nz) = InputR(i) + RR*Ef(Nz); % Boundary condition at z = L (right side);
180
181 S = (abs(Ef).^2 + abs(Er).^2).*EtoP*1e-6;
182
183 if t < Ion || t > Ioff
184     I_injv = I_off;
185 else
186     I_injv = I_on;
187 end
188
189 Stim = gain.*(N - Ntr).*S;
190 N = (N + dt*(I_injv/ eVol - Stim))./(1+ dt/taun);
191 Nave(i) = mean(N);
192
193 gain_z = gain.*(N - Ntr)./vg; % Compute gain coefficient
194 beta_i = (gain_z - alpha)./2; % Compute imaginary part of propagation constant
195 beta = ones(size(z)).*(beta_r + 1i * beta_i); % Complex propagation constant
196 exp_det = exp(-1i * dz * beta); % Phase shift due to propagation over dz
197
198 Ef(2:Nz) = fsync*exp_det(1:Nz-1).*Ef(1:Nz-1) + 1i*dz*kappa(2:Nz).*Er(2:Nz); % Forward Field Propagation
199 Er(1:Nz-1) = fsync*exp_det(2:Nz).*Er(2:Nz) + 1i*dz*kappa(2:Nz).*Ef(2:Nz); % Reverse Field Propagation
200
201 % Boundary Conditions
202 Pf(1) = 0; % zero polarization at the left boundary
203 Pf(Nz) = 0; % zero polarization at the right boundary
204 Pr(1) = 0; % zero polarization at the right boundary
205 Pr(Nz) = 0; % zero polarization at the left boundary
206 Cw0 = -LGamma + 1i * Lw0; % Defines the complex response function of the material.
207
208 % Dispersion Calculations
209 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.
210 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.
211 Tr = LGamma * Er(3:Nz) + Cw0 * Prp(2:Nz-1) + LGamma * Erp(3:Nz); % Computes the reverse polarization response
% based on previous field values.
212 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.
213
214 Ef(2:Nz-1) = Ef(2:Nz-1) - LGain * (Ef(2:Nz-1) - Pf(2:Nz-1)); % Adjusts the forward electric field
215 Er(2:Nz-1) = Er(2:Nz-1) - LGain * (Er(2:Nz-1) - Pr(2:Nz-1)); % Adjusts the reverse electric field
216
217 % Output
218 OutputR(i) = Ef(Nz) * (1 - RR); % Right output at z = L
219 OutputL(i) = Er(1) * (1 - RL); % Left output at z = 0
220
221 A = sqrt(gamma*beta_spe*c_hb*f0*L*1e-2/taun)/(2*Nz);
222 if SPE > 0
223     eTf = ((randn(Nz,1)+1i*randn(Nz,1))*A).';
224     eTr = ((randn(Nz,1)+1i*randn(Nz,1))*A).';
225 else
226     eTf = ((ones(Nz,1))*A).';
227     eTr = ((ones(Nz,1))*A).';
228 end
229
230 EsF = eTf*abs(SPE).*sqrt(N.*1e6);
231 Esr = eTr*abs(SPE).*sqrt(N.*1e6);
232
233 Ef = Ef + EsF;
234 Er = Er + Esr;
235
236 % % FFT data from the outputs
237 fftOutput1 = fftshift(fft(OutputR)); % Get FFT data for OutputR
238 % fftOutput2 = fftshift(fft(OutputL)); % Get FFT data for OutputL
239 fftInput1 = fftshift(fft(InputL)); % Get FFT data for InputL
240 omega = fftshift(wspace(time));
241
242
243 if mod(i,1000) == 0 % Only executed when i is multiple of plotN
244
245     % Forward Propagation of the Gaussian Pulse
246     subplot(3,2,1)
247     plot(z * 10000, real(Ef), 'r'); hold on
248     plot(z * 10000, imag(Ef), 'r--'); hold off
249     xlim(XL * 1e4)
250     ylim(YL)
251     xlabel('z (\mum)')
252     ylabel('E_f (V/\mum)')
253     legend('\Re', '\Im')
254     hold off
255
256     % Carrier Density N
257     subplot(3,2,2)
258     plot(z * 10000, N, 'r');
259     xlim(XL * 1e4)
260     ylim([0, 5 * Ntr])

```

```

261 xlabel('z(\mum)')
262 ylabel('N')
263
264 % Average Carrier Density Over Time
265 subplot(3,2,3)
266 plot(time * 1e12, Nave, 'b');
267 xlim([0, Nt * dt * 1e12])
268 ylim([0, 5 * Ntr])
269 xlabel('time(ps)')
270 ylabel('Nave')
271
272 % Input and Output Fields over time
273 subplot(3,2,5)
274 plot(time * 1e12, real(InputL), 'r'); hold on
275 plot(time * 1e12, real(OutputR), 'g');
276 plot(time * 1e12, real(InputR), 'b');
277 plot(time * 1e12, real(OutputL), 'm--');
278 xlim([0, Nt * dt * 1e12])
279 ylim auto
280 xlabel('time(ps)')
281 ylabel('I')
282 legend('Left Input', 'Right Output', 'Right Input', 'Left Output', 'Location', 'east')
283 hold off
284
285 % Output Field Spectrum (Magnitude in dB)
286 subplot(3,2,4)
287 plot(omega, 20*log10(abs(fftOutput1)), 'b'); hold on
288 %plot(omega, 20*log10(abs(fftInput1)), 'r'); %
289 xlabel('GHz')
290 ylabel('20 log_{10} |E|')
291 legend('Output', 'Input')
292 hold off
293
294 % Phase of Output Field
295 subplot(3,2,6)
296 phase1 = unwrap(angle(fftOutput1)); % Unwrap the phase1
297 phase2 = unwrap(angle(fftInput1)); % Unwrap the phase for Input
298 plot(omega, phase1);
299 hold on;
300 plot(omega, phase2);
301 hold off;
302 xlabel('GHz')
303 ylabel('phase (E)')
304 legend('fftOutput1', 'fftInput1');
305 ylim auto
306
307 pause(0.01)
308
309
310
311 end
312
313 % Update Previous Values
314 Efp = Ef;
315 Erp = Er;
316 Pfp = Pf;
317 Prp = Pr;
318 end

```

Listing 2: MATLAB code for Milestone 7

6.3 Section C: Milestone 8 - Lasers

```

1
2 set(0,'defaultaxesfontsize',20)
3 set(0,'DefaultFigureWindowStyle','docked')
4 set(0,'DefaultLineLineWidth',2);
5 set(0,'Defaultaxeslinewidth',2)
6
7 set(0,'DefaultFigureWindowStyle','docked')
8
9 c_c = 299792458; % m/s TWM speed of light
10 c_eps_0 = 8.8542149e-12; % F/m vaccum permittivity
11 c_eps_0_cm = c_eps_0/100; % F/cm vaccum permittivity
12 c_mu_0 = 1/c_eps_0/c_c^2; % Permiability of free space
13 c_q = 1.60217653e-19; % Charge of an electron
14 c_hb = 1.05457266913e-34; % Dirac / Reduced Planck constant
15 c_h = c_hb*2*pi; % Planck constant
16
17 beta_r = 0; % De-tuning constant
18 beta_i = 0; % Gain Constant
19
20
21 beta_spe = .3e-5;
22 gamma = 1.0;
23 SPE = 7;

```

```

24
25 kappa0 = 0; % Coupling coefficient
26 kappaStart = 1/3; % Constant defines starting position where coupling begins.
27 kappaStop = 2/3; % Constant defines ending position where coupling stops.
28
29 InputParasL.E0 = 1e5; % Amplitude of the input E-field / Ef
30 InputParasL.we = 0; % Frequency of the complex sinusoidal modulation on the gaussian pulse
31 InputParasL.t0 = 200e-12; % The constant we are shifting the time by
32 InputParasL.wg = 10e-13; % Width of the Gaussian distribution
33 InputParasL.phi = 0; % Initial Phase of the Ef / input E-field
34 InputParasR = 0; % Placeholder variable for reverse propagation
35 InputParasL.rep = 500e-12;
36
37 n_g = 3.5; % Constant to control group velocity
38 vg = c_c/n_g *1e2; % TWM cm/s group velocity
39
40 Lambda = 1550e-9; % Wavelength of light
41 f0 = c_c/Lambda;
42
43 plotN = 10; % Divisor constant
44
45 % L = 1000e-6*1e2; % length of the waveguide in cm
46 L = 1000e-6*1e2; % length of the waveguide in cm
47
48
49 % Material Polarization Information
50 g_fwhm = 3.5e+012/10; % Frequency
51 LGamma = g_fwhm*2*pi;
52 Lw0 = 0;
53 LGain = 0.015; % Gain Constant
54
55 XL = [0,L]; % Start and End of the x-axis
56 %YL = [0,InputParasL.E0]; % Start and End of the y-axis
57 YL = [-InputParasL.E0,InputParasL.E0]; % Start and End of the y-axis
58
59 % Nz = 100; % Number of divisions
60 Nz = 100; % Number of divisions
61 dz = L/(Nz-1); % Distance between every point
62 dt = dz/vg; % Time taken to plot every point
63 fsync = dt*vg/dz; % Equals 1, allows the Gaussian to be stable
64
65 Nt = floor(400*Nz); % Time steps
66 tmax = Nt*dt; % Maximum time for simulation
67 tL = dt*Nz; % time to travel length
68 z = linspace(0,L,Nz); % Nz points, Nz-1 segments
69 time = nan(1,Nt); % Time matrix with 1 row and Nt columns / row vector of Nt elements
70
71 InputL = nan(1,Nt); % Matrix with 1 row and Nt columns / row vector of Nt elements
72 InputR = nan(1, Nt); % Matrix with 1 row and Nt columns / row vector of Nt elements
73 OutputL = nan(1,Nt); % Matrix with 1 row and Nt columns / row vector of Nt elements
74 OutputR = nan(1,Nt); % Matrix with 1 row and Nt columns / row vector of Nt elements
75 Ef = zeros(size(z)); % Matrix with the same dimensions as z, all elements initialized to 0
76 Er = zeros(size(z)); % Matrix with the same dimensions as z, all elements initialized to 0
77
78 Ef1 = @SourceFct; % Reference to SourceFct
79 ErN = @SourceFct; % Reference to SourceFct
80
81 t = 0; % Set t to a starting value of 0
82 time(1) = t; % Sets the first element of the time vector to 0
83
84 InputL(1) = Ef1(t, InputParasL); % Set initial value of InputL using the source function
85 InputR(1) = ErN(t, InputParasR); % Set initial value of InputR using the source function
86
87 OutputR(1) = Ef(Nz); % The end of the waveguide is the first value of the reflection (Right to Left)
88 OutputL(1) = Er(1); % The end of the waveguide is the first value of the reflection (Left to Right)
89
90 Ef(1) = InputL(1); % Initializes forward field at z = 0 (Input signal from the left)
91 Er(Nz) = InputR(1); % Initializes backward field at z = L (Input signal from the right)
92
93 kappa = kappa0*ones(size(z)); % Creates an array of size z, where all indexes hold a value of kappa0
94 kappa(z<L*kappaStart) = 0; % Sets the limit such that kappa is set to zero outside the interaction region.
95 kappa(z>L*kappaStop) = 0; % Sets the limit such that kappa is set to zero outside the interaction region.
96
97 Pf = zeros(size(z)); % Variable for the polarization of the material on the forward field
98 Pr = zeros(size(z)); % Variable for the polarization of the material on the reverse field
99
100 % Variables to hold field and polarization information
101 Efp = Ef;
102 Erp = Er;
103 Pfp = Pf;
104 Prp = Pr;
105
106 Nave = nan(1,Nt);
107 Ntr = 1e18;
108 N = ones(size(z))*Ntr;
109 Nave(1) = mean(N);
110
111 gain = vg*2.5e-16;
112 eVol = 1.5e-10*c_q;
113 Ion = 0.01e-9;
114 % Ion = 0.25e-9;
115 Ioff = 3e-9;

```

```

116 I_off = 0.024;
117 I_on = 0.1;
118 taun = 1e-9;
119 Zg = sqrt(c_mu_0/c_eps_0)/n_g;
120 EtoP = 1/(Zg*f0*vg*1e-2*c_hb);
121 alpha = 0;
122
123 figure('name', 'Fields')
124
125 % Forward field E_f
126 subplot(3,2,1)
127 plot(z*10000, real(Ef), 'r'); hold on
128 plot(z*10000, imag(Ef), 'r--');
129 plot(z*10000, real(Er), 'b--');
130 plot(z*10000, imag(Er), 'b');
131 hold off
132 xlim(XL*1e4)
133 ylim auto
134 xlabel('z (\num)')
135 ylabel('E_f (V/\num)')
136 legend('\Re(E_f)', '\Im(E_f)', '\Re(E_r)', '\Im(E_r)')
137
138
139 % Carrier Density N
140 subplot(3,2,2)
141 plot(z * 10000, N, 'r'); % Primary plot
142 xlim(XL * 1e4)
143 ylim([0, 5 * Ntr])
144 xlabel('z (\num)')
145 ylabel('N')
146
147 % % Add a second y-axis for S
148 % yyaxis right
149 % plot(z * 10000, S, 'b'); % Plot S on the right axis
150 % ylabel('S') % Change the label to the appropriate units
151
152
153 % Average Carrier Density Over Time
154 subplot(3,2,3)
155 plot(time * 1e12, Nave, 'b');
156 xlim([0, Nt * dt * 1e12])
157 ylim([0, 5 * Ntr])
158 xlabel('time(ps)')
159 ylabel('Nave')
160
161 subplot(3,2,5)
162 plot(time * 1e12, real(OutputR), 'g'); hold on
163 plot(time * 1e12, real(OutputL), 'm--');
164 xlim([0, Nt * dt * 1e12])
165 ylim auto
166 xlabel('time(ps)')
167 ylabel('E (V/um)')
168 legend('Right Output', 'Left Output', 'Location', 'east')
169 hold off
170
171 for i = 2:Nt % 2 to 1000 in steps of 1
172
173     t = dt*(i-1);
174     time(i) = t;
175
176     RL = 0.5; % The left side reflection coefficient
177     RR = 0.5; % The right side reflection coefficient
178
179     % Input
180     InputL(i) = Ef1(t,0);
181     % InputL(i) = Ef1(t, InputParasL); % At time t, we input a signal characterized by InputParasL from the left
182     InputR(i) = ErN(t, 0); % At time t, we input no signal from the right (since InputParasR = 0)
183
184     % Reflection
185     Ef(1) = InputL(i) + RL*Er(1); % Boundary condition at z = 0 (left side);
186     Er(Nz) = InputR(i) + RR*Ef(Nz); % Boundary condition at z = L (right side);
187
188     S = (abs(Ef).^2 + abs(Er).^2).*EtoP*1e-6;
189
190     if t < Ion || t > Ioff
191         I_injv = I_off;
192     else
193         I_injv = I_on;
194     end
195
196     Stim = gain.*(N - Ntr).*S;
197     N = (N + dt*(I_injv/ eVol - Stim))./(1+ dt/taun);
198     Nave(i) = mean(N);
199
200     gain_z = gain.*(N - Ntr)./vg; % Compute gain coefficient
201     beta_i = (gain_z - alpha)./2; % Compute imaginary part of propagation constant
202     beta = ones(size(z)).*(beta_r + 1i * beta_i); % Complex propagation constant
203     exp_det = exp(-1i * dz * beta); % Phase shift due to propagation over dz
204
205     Ef(2:Nz) = fsync*exp_det(1:Nz-1).*Ef(1:Nz-1) + 1i*dz*kappa(2:Nz).*Er(2:Nz); % Forward Field Propagation
206     Er(1:Nz-1) = fsync*exp_det(2:Nz).*Er(2:Nz) + 1i*dz*kappa(2:Nz).*Ef(2:Nz); % Reverse Field Propagation
207

```

```

208 % Boundary Conditions
209 Pf(1) = 0; % zero polarization at the left boundary
210 Pf(Nz) = 0; % zero polarization at the right boundary
211 Pr(1) = 0; % zero polarization at the right boundary
212 Pr(Nz) = 0; % zero polarization at the left boundary
213 Cw0 = -LGamma + 1i * Lw0; % Defines the complex response function of the material.
214
215 % Dispersion Calculations
216 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.
217 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.
218 Tr = LGamma * Er(3:Nz) + Cw0 * Prp(2:Nz-1) + LGamma * Erp(3:Nz); % Computes the reverse polarization response
% based on previous field values.
219 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.
220
221 Ef(2:Nz-1) = Ef(2:Nz-1) - LGain * (Ef(2:Nz-1) - Pf(2:Nz-1)); % Adjusts the forward electric field
222 Er(2:Nz-1) = Er(2:Nz-1) - LGain * (Er(2:Nz-1) - Pr(2:Nz-1)); % Adjusts the reverse electric field
223
224 % Output
225 OutputR(i) = Ef(Nz) * (1 - RR); % Right output at z = L
226 OutputL(i) = Er(1) * (1 - RL); % Left output at z = 0
227
228 A = sqrt(gamma*beta_spe*c_hb*f0*L*1e-2/taun)/(2*Nz);
229 if SPE > 0
230 eTf = ((randn(Nz,1)+1i*randn(Nz,1))*A).';
231 eTr = ((randn(Nz,1)+1i*randn(Nz,1))*A).';
232 else
233 eTf = ((ones(Nz,1))*A).';
234 eTr = ((ones(Nz,1))*A).';
235 end
236
237 EsF = eTf*abs(SPE).*sqrt(N.*1e6);
238 EsR = eTr*abs(SPE).*sqrt(N.*1e6);
239
240 Ef = Ef + EsF;
241 Er = Er + EsR;
242
243 % % FFT data from the outputs
244 fftOutput1 = fftshift(fft(OutputR)); % Get FFT data for OutputR
245 fftOutput2 = fftshift(fft(OutputL)); % Get FFT data for OutputL
246 fftInput1 = fftshift(fft(InputL)); % Get FFT data for OutputL
247 omega = fftshift(wspace(time));
248
249
250 if mod(i,2000) == 0 % Only executed when i is multiple of plotN
251
252 % Forward field E_f
253 subplot(3,2,1)
254 plot(z*10000, real(Ef), 'r'); hold on
255 plot(z*10000, imag(Ef), 'r--');
256 plot(z*10000, real(Er), 'b--');
257 plot(z*10000, imag(Er), 'b');
258 hold off
259 xlim(XL*1e4)
260 ylim auto
261 xlabel('z (\mum)')
262 ylabel('E_f (V/\mum)')
263 legend('\Re(E_f)', '\Im(E_f)', '\Re(E_r)', '\Im(E_r)')
264
265
266 % Carrier Density N
267 subplot(3,2,2)
268 plot(z * 10000, N, 'r'); % Primary plot
269 xlim(XL * 1e4)
270 ylim([0, 5 * Ntr])
271 xlabel('z (\mum)')
272 ylabel('N')
273
274 % Add a second y-axis for S
275 yyaxis right
276 plot(z * 10000, S, 'b'); % Plot S on the right axis
277 ylabel('S') % Change the label to the appropriate units
278
279 % Average Carrier Density Over Time
280 subplot(3,2,3)
281 plot(time * 1e12, Nave, 'b');
282 xlim([0, Nt * dt * 1e12])
283 ylim([0, 2.5 * Ntr])
284 xlabel('time(ps)')
285 ylabel('Nave')
286
287 % Input and Output Fields over time
288 subplot(3,2,5)
289 plot(time * 1e12, real(InputL), 'r'); hold on
290 plot(time * 1e12, real(OutputR), 'g');
291 plot(time * 1e12, real(InputR), 'b');
292 plot(time * 1e12, real(OutputL), 'm--');
293 xlim([0, Nt * dt * 1e12])
294 ylim auto
295 xlabel('time(ps)')

```

```

296 ylabel('0')
297 legend('Right Output', 'Left Output', 'Location', 'east')
298 hold off
299
300
301 % Output Field Spectrum (Magnitude in dB)
302 subplot(3,2,4)
303 plot(omega, 20*log10(abs(fftOutput1)), 'b'); hold on
304 plot(omega, 20*log10(abs(fftInput1)), 'r'); %
305 xlabel('Frequency (Hz)')
306 ylabel('20 log_{10} |E|')
307 legend('Output', 'Input')
308 xlim([-0.05e14 0.05e14])
309 hold off
310
311 % Phase of Output Field
312 subplot(3,2,6)
313 phase1 = unwrap(angle(fftOutput1)); % Unwrap the phase1
314 phase2 = unwrap(angle(fftInput1)); % Unwrap the phase for Input
315 plot(omega, phase1);
316 hold on;
317 plot(omega, phase2);
318 hold off;
319 xlabel('Frequency (Hz)')
320 ylabel('phase (E)')
321 legend('fftOutput1', 'fftInput1');
322
323 pause(0.01)
324
325
326 end
327
328 % Update Previous Values
329 Efp = Ef;
330 Erp = Er;
331 Pfp = Pf;
332 Prp = Pr;
333
334 end

```

Listing 3: MATLAB code for Milestone 8

6.4 Section D: Milestone 9 – Final Investigation 1

```

1
2 set(0,'defaultaxesfontsize',20)
3 set(0,'DefaultFigureWindowStyle','docked')
4 set(0,'DefaultLineLineWidth',2);
5 set(0,'Defaultaxeslinewidth',2)
6
7 set(0,'DefaultFigureWindowStyle','docked')
8
9 c_c = 299792458; % m/s TWM speed of light
10 c_eps_0 = 8.8542149e-12; % F/m vaccum permittivity
11 c_eps_0_cm = c_eps_0/100; % F/cm vaccum permittivity
12 c_mu_0 = 1/c_eps_0/c_c^2; % Permiability of free space
13 c_q = 1.60217653e-19; % Charge of an electron
14 c_hb = 1.05457266913e-34; % Dirac / Reduced Planck constant
15 c_h = c_hb*2*pi; % Planck constant
16
17 beta_r = 0; % De-tuning constant
18 beta_i = 0; % Gain Constant
19
20
21 beta_spe = .3e-5;
22 gamma = 1.0;
23 SPE = 7;
24
25 kappa0 = 0; % Coupling coefficient
26 kappaStart = 1/3; % Constant defines starting position where coupling begins.
27 kappaStop = 2/3; % Constant defines ending position where coupling stops.
28
29 InputParasL.E0 = 1e5; % Amplitude of the input E-field / E_f
30 InputParasL.we = 0; % Frequency of the complex sinusoidal modulation on the gaussian pulse
31 InputParasL.t0 = 200e-12; % The constant we are shifting the time by
32 InputParasL.wg = 10e-13; % Width of the Gaussian distribution
33 InputParasL.phi = 0; % Initial Phase of the E_f / input E-field
34 InputParasR = 0; % Placeholder variable for reverse propagation
35 InputParasL.rep = 500e-12;
36
37 n_g = 3.5; % Constant to control group velocity
38 vg = c_c/n_g *1e2; % TWM cm/s group velocity
39
40 Lambda = 1550e-9; % Wavelength of light
41 f0 = c_c/Lambda;
42

```

```

43 | plotN = 10; % Divisor constant
44 |
45 | % L = 1000e-6*1e2; % length of the waveguide in cm
46 | L = 1000e-6*1e2; % length of the waveguide in cm
47 |
48 |
49 | % Material Polarization Information
50 | g_fwhm = 3.5e+012/10; % Frequency
51 | LGamma = g_fwhm*2*pi;
52 | Lw0 = 0;
53 | LGain = 0.015; % Gain Constant
54 |
55 | XL = [0,L]; % Start and End of the x-axis
56 | %YL = [0,InputParasL.E0]; % Start and End of the y-axis
57 | YL = [-InputParasL.E0,InputParasL.E0]; % Start and End of the y-axis
58 |
59 | % Nz = 100; % Number of divisions
60 | Nz = 100; % Number of divisions
61 | dz = L/(Nz-1); % Distance between every point
62 | dt = dz/vg; % Time taken to plot every point
63 | fsync = dt*vg/dz; % Equals 1, allows the Gaussian to be stable
64 |
65 | Nt = floor(400*Nz); % Time steps
66 | tmax = Nt*dt; % Maximum time for simulation
67 | tL = dt*Nz; % time to travel length
68 | z = linspace(0,L,Nz); % Nz points, Nz-1 segments
69 | time = nan(1,Nt); % Time matrix with 1 row and Nt columns / row vector of Nt elements
70 |
71 | InputL = nan(1,Nt); % Matrix with 1 row and Nt columns / row vector of Nt elements
72 | InputR = nan(1, Nt); % Matrix with 1 row and Nt columns / row vector of Nt elements
73 | OutputL = nan(1,Nt); % Matrix with 1 row and Nt columns / row vector of Nt elements
74 | OutputR = nan(1,Nt); % Matrix with 1 row and Nt columns / row vector of Nt elements
75 | Ef = zeros(size(z)); % Matrix with the same dimensions as z, all elements initialized to 0
76 | Er = zeros(size(z)); % Matrix with the same dimensions as z, all elements initialized to 0
77 |
78 | Ef1 = @SourceFct; % Reference to SourceFct
79 | ErN = @SourceFct; % Reference to SourceFct
80 |
81 | t = 0; % Set t to a starting value of 0
82 | time(1) = t; % Sets the first element of the time vector to 0
83 |
84 | InputL(1) = Ef1(t, InputParasL); % Set initial value of InputL using the source function
85 | InputR(1) = ErN(t, InputParasR); % Set initial value of InputR using the source function
86 |
87 | OutputR(1) = Ef(Nz); % The end of the waveguide is the first value of the reflection (Right to Left)
88 | OutputL(1) = Er(1); % The end of the waveguide is the first value of the reflection (Left to Right)
89 |
90 | Ef(1) = InputL(1); % Initializes forward field at z = 0 (Input signal from the left)
91 | Er(Nz) = InputR(1); % Initializes backward field at z = L (Input signal from the right)
92 |
93 | kappa = kappa0*ones(size(z)); % Creates an array of size z, where all indexes hold a value of kappa0
94 | kappa(z<L*kappaStart) = 0; % Sets the limit such that kappa is set to zero outside the interaction region
95 | kappa(z>L*kappaStop) = 0; % Sets the limit such that kappa is set to zero outside the interaction region.
96 |
97 | Pf = zeros(size(z)); % Variable for the polarization of the material on the forward field
98 | Pr = zeros(size(z)); % Variable for the polarization of the material on the reverse field
99 |
100 | % Variables to hold field and polarization information
101 | Efp = Ef;
102 | Erp = Er;
103 | Pfp = Pf;
104 | Prp = Pr;
105 |
106 | Nave = nan(1,Nt);
107 | Ntr = 1e18;
108 | N = ones(size(z))*Ntr;
109 | Nave(1) = mean(N);
110 |
111 | gain = vg*2.5e-16;
112 | eVol = 1.5e-10*c_q;
113 | Ion = 0.01e-9;
114 | % Ion = 0.25e-9;
115 | Ioff = 3e-9;
116 | I_off = 0.024;
117 | I_on = 0.25;
118 | taun = 1e-9;
119 | Zg = sqrt(c_mu_0/c_eps_0)/n_g;
120 | EtoP = 1/(Zg*f0*vg*1e-2*c_hb);
121 | alpha = 0;
122 |
123 | figure('name', 'Fields')
124 | If = nan(1, Nt); % Forward mode intensity over time
125 | Ir = nan(1, Nt); % Backward mode intensity over time
126 |
127 |
128 | % Forward field E_f
129 | subplot(3,2,1)
130 | plot(z*10000, real(Ef), 'r'); hold on
131 | plot(z*10000, imag(Ef), 'r--');
132 | plot(z*10000, real(Er), 'b--');
133 | plot(z*10000, imag(Er), 'b');
134 | hold off

```



```

135 xlim(XL*1e4)
136 ylim auto
137 xlabel('z (\num)')
138 ylabel('E_f (V/\num)')
139 legend('\Re(E_f)', '\Im(E_f)', '\Re(E_r)', '\Im(E_r)')
140
141
142 % Carrier Density N
143 subplot(3,2,2)
144 plot(z * 10000, N, 'r'); % Primary plot
145 xlim(XL * 1e4)
146 ylim([0, 5 * Ntr])
147 xlabel('z (\num)')
148 ylabel('N')
149
150 % Add a second y-axis for S
151 % yyaxis right
152 % plot(z * 10000, S, 'b'); % Plot S on the right axis
153 % ylabel('S') % Change the label to the appropriate units
154
155
156 % Average Carrier Density Over Time
157 subplot(3,2,3)
158 plot(time * 1e12, Nave, 'b');
159 xlim([0, Nt * dt * 1e12])
160 ylim([0, 5 * Ntr])
161 xlabel('time(ps)')
162 ylabel('Nave')
163
164 subplot(3,2,5)
165 plot(time * 1e12, real(OutputR), 'g'); hold on
166 plot(time * 1e12, real(OutputL), 'm--');
167 xlim([0, Nt * dt * 1e12])
168 ylim auto
169 xlabel('time(ps)')
170 ylabel('E (V/um)')
171 legend('Right Output', 'Left Output', 'Location', 'east')
172 hold off
173
174 % Define time range for plotting mode intensities
175 t_start = 0.000001e-12; % 100 ps
176 t_end = Nt; % 300 ps
177
178
179
180 for i = 2:Nt % 2 to 1000 in steps of 1
181
182     t = dt*(i-1);
183     time(i) = t;
184
185     RL = 0.5; % The left side reflection coefficient
186     RR = 0.5; % The right side reflection coefficient
187
188     % Input
189     InputL(i) = Ef1(t,0);
190     % InputL(i) = Ef1(t, InputParasL); % At time t, we input a signal characterized by InputParasL from the left
191     InputR(i) = ErN(t, 0); % At time t, we input no signal from the right (since InputParasR = 0)
192
193     % Reflection
194     Ef(1) = InputL(i) + RL*Er(1); % Boundary condition at z = 0 (left side);
195     Er(Nz) = InputR(i) + RR*Ef(Nz); % Boundary condition at z = L (right side);
196
197     S = (abs(Ef).^2 + abs(Er).^2).*EtoP*1e-6;
198
199     if t < Ion || t > Ioff
200         I_injv = I_off;
201     else
202         I_injv = I_on;
203     end
204
205     Stim = gain.*(N - Ntr).*S;
206     N = (N + dt*(I_injv/ eVol - Stim))./(1+ dt/taun);
207     Nave(i) = mean(N);
208
209     gain_z = gain.*(N - Ntr)./vg; % Compute gain coefficient
210     beta_i = (gain_z - alpha)./2; % Compute imaginary part of propagation constant
211     beta = ones(size(z)).*(beta_r + 1i * beta_i); % Complex propagation constant
212     exp_det = exp(-1i * dz * beta); % Phase shift due to propagation over dz
213
214     Ef(2:Nz) = fsync*exp_det(1:Nz-1).*Ef(1:Nz-1) + 1i*dz*kappa(2:Nz).*Er(2:Nz); % Forward Field Propagation
215     Er(1:Nz-1) = fsync*exp_det(2:Nz).*Er(2:Nz) + 1i*dz*kappa(2:Nz).*Ef(2:Nz); % Reverse Field Propagation
216
217     % Boundary Conditions
218     Pf(1) = 0; % zero polarization at the left boundary
219     Pf(Nz) = 0; % zero polarization at the right boundary
220     Pr(1) = 0; % zero polarization at the right boundary
221     Pr(Nz) = 0; % zero polarization at the left boundary
222     Cw0 = -LGamma + 1i * Lw0; % Defines the complex response function of the material.
223
224     % Dispersion Calculations
225     Tf = LGamma * Ef(1:Nz-2) + Cw0 * Ppf(2:Nz-1) + LGamma * Efp(1:Nz-2); % Computes the forward polarization response
        based on previous field values.

```

```

226 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.
227 Tr = LGamma * Er(3:Nz) + Cw0 * Prp(2:Nz-1) + LGamma * Erp(3:Nz); % Computes the reverse polarization response
    based on previous field values.
228 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.
229
230 Ef(2:Nz-1) = Ef(2:Nz-1) - LGain * (Ef(2:Nz-1) - Pf(2:Nz-1)); % Adjusts the forward electric field
231 Er(2:Nz-1) = Er(2:Nz-1) - LGain * (Er(2:Nz-1) - Pr(2:Nz-1)); % Adjusts the reverse electric field
232
233 % Output
234 OutputR(i) = Ef(Nz) * (1 - RR); % Right output at z = L
235 OutputL(i) = Er(1) * (1 - RL); % Left output at z = 0
236
237 % --- Mode intensities as a function of time ---
238 If(i) = trapz(z, abs(Ef).^2); % Forward mode intensity
239 Ir(i) = trapz(z, abs(Er).^2); % Backward mode intensity
240
241
242 A = sqrt(gamma*beta_spe*c_hb*f0*L*1e-2/taun)/(2*Nz);
243 if SPE > 0
244     eTf = ((randn(Nz,1)+1i*randn(Nz,1))*A).';
245     eTr = ((randn(Nz,1)+1i*randn(Nz,1))*A).';
246 else
247     eTf = ((ones(Nz,1))*A).';
248     eTr = ((ones(Nz,1))*A).';
249 end
250
251 EsF = eTf*abs(SPE).*sqrt(N.*1e6);
252 Esr = eTr*abs(SPE).*sqrt(N.*1e6);
253
254 Ef = Ef + EsF;
255 Er = Er + Esr;
256
257 % % FFT data from the outputs
258 fftOutput1 = fftshift(fft(OutputR)); % Get FFT data for OutputR
259 fftOutput2 = fftshift(fft(OutputL)); % Get FFT data for OutputL
260 fftInput1 = fftshift(fft(InputL)); % Get FFT data for OutputL
261 omega = fftshift(wspace(time));
262
263 % Find indices for the selected time range
264 range_idx = find(time >= t_start & time <= t_end);
265
266 if mod(i,2000) == 0 % Only executed when i is multiple of plotN
267
268     % Forward field E_f
269     subplot(3,2,1)
270     plot(z*10000, real(Ef), 'r'); hold on
271     plot(z*10000, imag(Ef), 'r--');
272     plot(z*10000, real(Er), 'b--');
273     plot(z*10000, imag(Er), 'b');
274     hold off
275     xlim(XL*1e4)
276     ylim auto
277     xlabel('z (\mum)')
278     ylabel('E_f (V/\mum)')
279     legend('\Re(E_f)', '\Im(E_f)', '\Re(E_r)', '\Im(E_r)')
280
281
282     % Carrier Density N
283     subplot(3,2,2)
284     plot(z * 10000, N, 'r'); % Primary plot
285     xlim(XL * 1e4)
286     ylim([0, 5 * Ntr])
287     xlabel('z (\mum)')
288     ylabel('N')
289
290     % Add a second y-axis for S
291     yyaxis right
292     plot(z * 10000, S, 'b'); % Plot S on the right axis
293     ylabel('S') % Change the label to the appropriate units
294
295     % Average Carrier Density Over Time
296     subplot(3,2,3)
297     plot(time * 1e12, Nave, 'b');
298     xlim([0, Nt * dt * 1e12])
299     ylim([0, 2.5 * Ntr])
300     xlabel('time(ps)')
301     ylabel('Nave')
302
303     % Input and Output Fields over time
304     subplot(3,2,5)
305     plot(time * 1e12, real(InputL), 'r'); hold on
306     plot(time * 1e12, real(OutputR), 'g');
307     plot(time * 1e12, real(InputR), 'b');
308     plot(time * 1e12, real(OutputL), 'm--');
309     xlim([0, Nt * dt * 1e12])
310     ylim auto
311     xlabel('time(ps)')
312     ylabel('0')
313     legend('Right Output', 'Left Output', 'Location', 'east')
314     hold off

```

```

315
316
317 % Output Field Spectrum (Magnitude in dB)
318 subplot(3,2,4)
319 plot(omega, 20*log10(abs(fftOutput1)), 'b'); hold on
320 plot(omega, 20*log10(abs(fftInput1)), 'r'); %
321 xlabel('Frequency (Hz)')
322 ylabel('20 log_{10} |E|')
323 legend('Output', 'Input')
324 xlim([-0.05e14 0.05e14])
325 hold off
326
327 % Phase of Output Field
328 subplot(3,2,6)
329 phase1 = unwrap(angle(fftOutput1)); % Unwrap the phase1
330 phase2 = unwrap(angle(fftInput1)); % Unwrap the phase for Input
331 plot(omega, phase1);
332 hold on;
333 plot(omega, phase2);
334 hold off;
335 xlabel('Frequency (Hz)')
336 ylabel('phase (E)')
337 legend('fftOutput1', 'fftInput1');
338
339 figure('name', 'Mode Intensities Over Time')
340 plot(time(range_idx) * 1e12, If(range_idx), 'r', 'DisplayName', '|E_f|^2'); hold on
341 plot(time(range_idx) * 1e12, Ir(range_idx), 'b', 'DisplayName', '|E_r|^2');
342 xlabel('Time (ps)')
343 ylabel('Mode Intensity')
344 legend
345 title('Mode Intensities vs time')
346 grid on
347
348
349 pause(0.01)
350
351
352 end
353
354 % Update Previous Values
355 Efp = Ef;
356 Erp = Er;
357 Pfp = Pf;
358 Prp = Pr;
359
360
361
362
363
364
365
366 end
367
368 %% UNCOMMENT THIS PART IF YOU WISH TO SEE PLOTS SIMILAR TO THE ONES IN MY FINAL REPORT
369 % figure('Name', 'FFT plots at different time samples');
370 % hold on;
371 %
372 % Time points in % of total simulation
373 % pcts = [0.001, 0.25, 0.5, 0.75, 1, ...
374 %         1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5, ...
375 %         10, 15, 20, 25, 30, ...
376 %         50, 70, 90, 100];
377 %
378 % colors = lines(length(pcts));
379 % startIdx = round(0.001 * Nt);
380 %
381 % for k = 1:length(pcts)
382 %     endIdx = round((pcts(k)/100) * Nt);
383 %     idxRange = startIdx:endIdx;
384 %
385 %     if length(idxRange) < 2
386 %         continue
387 %     end
388 %
389 %     fftData = fftshift(abs(fft(OutputR(idxRange))));
390 %     omega = fftshift(wnspace(time(idxRange)));
391 %
392 %     plot(omega, 20*log10(fftData), 'Color', colors(k,:), ...
393 %          'DisplayName', sprintf('%0.3f%% %0.1f%%', 0.001, pcts(k)));
394 % end
395 %
396 % xlabel('Frequency (Hz)');
397 % ylabel('20 log_{10} |E|');
398 % title('FFT Snapshots: Growth of Modes Over Time');
399 % legend('Location', 'bestoutside');
400 % grid on;
401 % xlim([-0.03e14, 0.03e14]);
402 %
403 %
404 % disp('Simulation finished. Running FFT analysis...');
405 %
406 % transient_range = round(0.0001*Nt):round(0.003*Nt);

```

```

407 % steady_range = round(0.0001*Nt):round(0.05*Nt);
408 % stable_range = round(0.0001*Nt):round(0.59*Nt);
409 % full_range = 1:Nt;
410 %
411 % output_transient = OutputR(transient_range);
412 % output_steady = OutputR(steady_range);
413 % output_stable = OutputR(stable_range);
414 % output_full = OutputR(full_range);
415 %
416 % time_transient = time(transient_range);
417 % time_steady = time(steady_range);
418 % time_stable = time(stable_range);
419 % time_full = time(full_range);
420 %
421 % omega_transient = fftshift(wspace(time_transient));
422 % omega_steady = fftshift(wspace(time_steady));
423 % omega_stable = fftshift(wspace(time_stable));
424 % omega_full = fftshift(wspace(time_full));
425 %
426 % fft_transient = fftshift(fft(output_transient));
427 % fft_steady = fftshift(fft(output_steady));
428 % fft_stable = fftshift(fft(output_stable));
429 % fft_full = fftshift(fft(output_full));
430 %
431 %
432 %
433 % figure('Name', 'FFT Comparison (All Phases)', 'Color', 'w');
434 %
435 % % Transient FFT
436 % subplot(2,2,1);
437 % plot(omega_transient, 20*log10(abs(fft_transient)), 'b');
438 % xlabel('Frequency (Hz)');
439 % ylabel('20 log_{10} |E|');
440 % title('Modes during Transient');
441 % grid on;
442 % xlim([-0.1e14 0.1e14]);
443 %
444 % % Steady-State FFT
445 % subplot(2,2,2);
446 % plot(omega_steady, 20*log10(abs(fft_steady)), 'r');
447 % xlabel('Frequency (Hz)');
448 % ylabel('20 log_{10} |E|');
449 % title('Modes approaching Steady-State');
450 % grid on;
451 % xlim([-0.02e14 0.02e14]);
452 %
453 % % Stable FFT
454 % subplot(2,2,3);
455 % plot(omega_stable, 20*log10(abs(fft_stable)), 'g');
456 % xlabel('Frequency (Hz)');
457 % ylabel('20 log_{10} |E|');
458 % title('Modes Stable In Steady-State');
459 % grid on;
460 % xlim([-0.02e14 0.02e14]);
461 %
462 % % Full FFT
463 % subplot(2,2,4);
464 % plot(omega_full, 20*log10(abs(fft_full)), 'k');
465 % xlabel('Frequency (Hz)');
466 % ylabel('20 log_{10} |E|');
467 % title('Modes over Full Simulation');
468 % grid on;
469 % xlim([-0.1e14 0.1e14]);

```

Listing 4: MATLAB code for Milestone 9 Part 1

6.5 Section E: Milestone 9 – Final Investigation 2

```

1
2 set(0,'defaultaxesfontsize',20)
3 set(0,'DefaultFigureWindowStyle','docked')
4 set(0,'DefaultLineLineWidth',2);
5 set(0,'DefaultAxesLineWidth',2)
6
7 set(0,'DefaultFigureWindowStyle','docked')
8
9 c_c = 299792458;           % m/s TWM speed of light
10 c_eps_0 = 8.8542149e-12;  % F/m vaccum permittivity
11 c_eps_0_cm = c_eps_0/100; % F/cm vaccum permittivity
12 c_mu_0 = 1/c_eps_0/c_c^2; % Permiability of free space
13 c_q = 1.60217653e-19;    % Charge of an electon
14 c_hb = 1.05457266913e-34; % Dirac / Reduced Planck constant
15 c_h = c_hb*2*pi;         % Planck constant
16
17 beta_r = 0;              % De-tuning constant
18 beta_i = 0;              % Gain Constant

```

```

19
20
21 beta_spe = .3e-5;
22 gamma = 1.0;
23 SPE = 7;
24
25 kappa0 = 0; % Coupling coefficient
26 kappaStart = 1/3; % Constant defines starting position where coupling begins.
27 kappaStop = 2/3; % Constant defines ending position where coupling stops.
28
29 InputParasL.E0 = 1e5; % Amplitude of the input E-field / E_f
30 InputParasL.we = 0; % Frequency of the complex sinusoidal modulation on the gaussian pulse
31 InputParasL.t0 = 200e-12; % The constant we are shifting the time by
32 InputParasL.wg = 10e-13; % Width of the Gaussian distribution
33 InputParasL.phi = 0; % Initial Phase of the E_f / input E-field
34 InputParasR = 0; % Placeholder variable for reverse propagation
35 InputParasL.rep = 500e-12;
36
37 n_g = 3.5; % Constant to control group velocity
38 vg = c_c/n_g *1e2; % TWM cm/s group velocity
39
40 Lambda = 1550e-9; % Wavelength of light
41 f0 = c_c/Lambda;
42
43 plotN = 10; % Divisor constant
44
45 % L = 1000e-6*1e2; % length of the waveguide in cm
46 L = 1000e-6*1e2; % length of the waveguide in cm
47
48 % Material Polarization Information
49 g_fwhm = 3.5e+012/10; % Frequency
50 LGamma = g_fwhm*2*pi;
51 Lw0 = 0;
52 LGain = 0.015; % Gain Constant
53
54 XL = [0,L]; % Start and End of the x-axis
55 %YL = [0,InputParasL.E0]; % Start and End of the y-axis
56 YL = [-InputParasL.E0,InputParasL.E0]; % Start and End of the y-axis
57
58 % Nz = 100; % Number of divisions
59 Nz = 100; % Number of divisions
60 dz = L/(Nz-1); % Distance between every point
61 dt = dz/vg; % Time taken to plot every point
62 fsync = dt*vg/dz; % Equals 1, allows the Gaussian to be stable
63
64 Nt = floor(400*Nz); % Time steps
65 tmax = Nt*dt; % Maximum time for simulation
66 t_L = dt*Nz; % time to travel length
67 z = linspace(0,L,Nz); % Nz points, Nz-1 segments
68 time = nan(1,Nt); % Time matrix with 1 row and Nt columns / row vector of Nt elements
69
70 InputL = nan(1,Nt); % Matrix with 1 row and Nt columns / row vector of Nt elements
71 InputR = nan(1, Nt); % Matrix with 1 row and Nt columns / row vector of Nt elements
72 OutputL = nan(1,Nt); % Matrix with 1 row and Nt columns / row vector of Nt elements
73 OutputR = nan(1,Nt); % Matrix with 1 row and Nt columns / row vector of Nt elements
74 Ef = zeros(size(z)); % Matrix with the same dimensions as z, all elements initialized to 0
75 Er = zeros(size(z)); % Matrix with the same dimensions as z, all elements initialized to 0
76
77 Ef1 = @SourceFct; % Reference to SourceFct
78 ErN = @SourceFct; % Reference to SourceFct
79
80 t = 0; % Set t to a starting value of 0
81 time(1) = t; % Sets the first element of the time vector to 0
82
83 InputL(1) = Ef1(t, InputParasL); % Set initial value of InputL using the source function
84 InputR(1) = ErN(t, InputParasR); % Set initial value of InputR using the source function
85
86 OutputR(1) = Ef(Nz); % The end of the waveguide is the first value of the reflection (Right to Left)
87 OutputL(1) = Er(1); % The end of the waveguide is the first value of the reflection (Left to Right)
88
89 Ef(1) = InputL(1); % Initializes forward field at z = 0 (Input signal from the left)
90 Er(Nz) = InputR(1); % Initializes backward field at z = L (Input signal from the right)
91
92 kappa = kappa0*ones(size(z)); % Creates an array of size z, where all indexes hold a value of kappa0
93 kappa(z<L*kappaStart) = 0; % Sets the limit such that kappa is set to zero outside the interaction region.
94 kappa(z>L*kappaStop) = 0; % Sets the limit such that kappa is set to zero outside the interaction region.
95
96 Pf = zeros(size(z)); % Variable for the polarization of the material on the forward field
97 Pr = zeros(size(z)); % Variable for the polarization of the material on the reverse field
98
99 % Variables to hold field and polarization information
100
101 Efp = Ef;
102 Erp = Er;
103 Pfp = Pf;
104 Prp = Pr;
105
106 Nave = nan(1,Nt);
107 Ntr = 1e18;
108 N = ones(size(z))*Ntr;
109 Nave(1) = mean(N);
110

```

```

111 gain = vg*2.5e-16;
112 eVol = 1.5e-10*c_q;
113 Ion = 0.01e-9;
114 % Ion = 0.25e-9;
115 Ioff = 3e-9;
116 I_off = 0.024;
117 I_on = 0.1;
118 taun = 1e-9;
119 Zg = sqrt(c_mu_0/c_eps_0)/n_g;
120 EtoP = 1/(Zg*f0*vg*1e-2*c_hb);
121 alpha = 0;
122
123 InjectedFeedback = zeros(1, Nt);
124
125
126 figure('name', 'Fields')
127
128 % Forward field E_f
129 subplot(3,2,1)
130 plot(z*10000, real(Ef), 'r'); hold on
131 plot(z*10000, imag(Ef), 'r--');
132 plot(z*10000, real(Er), 'b--');
133 plot(z*10000, imag(Er), 'b');
134 hold off
135 xlim(XL*1e4)
136 ylim auto
137 xlabel('z (\mum)')
138 ylabel('E_f (V/\mum)')
139 legend('\Re(E_f)', '\Im(E_f)', '\Re(E_r)', '\Im(E_r)')
140
141
142 % Carrier Density N
143 subplot(3,2,2)
144 plot(z * 10000, N, 'r'); % Primary plot
145 xlim(XL * 1e4)
146 ylim([0, 5 * Ntr])
147 xlabel('z (\mum)')
148 ylabel('N')
149
150 % % Add a second y-axis for S
151 % yyaxis right
152 % plot(z * 10000, S, 'b'); % Plot S on the right axis
153 % ylabel('S') % Change the label to the appropriate units
154
155
156 % Average Carrier Density Over Time
157 subplot(3,2,3)
158 plot(time * 1e12, Nave, 'b');
159 xlim([0, Nt * dt * 1e12])
160 ylim([0, 5 * Ntr])
161 xlabel('time(ps)')
162 ylabel('Nave')
163
164 subplot(3,2,5)
165 plot(time * 1e12, real(OutputR), 'g'); hold on
166 plot(time * 1e12, real(OutputL), 'm--');
167 xlim([0, Nt * dt * 1e12])
168 ylim auto
169 xlabel('time(ps)')
170 ylabel('E (V/\mum)')
171 legend('Right Output', 'Left Output', 'Location', 'east')
172 hold off
173
174 for i = 2:Nt % 2 to 1000 in steps of 1
175
176     t = dt*(i-1);
177     time(i) = t;
178
179     RL = 0.5; % The left side reflection coefficient
180     RR = 0.5; % The right side reflection coefficient
181
182
183     delay_steps = round(500e-12/dt);
184     phase_shift = pi/6;
185
186     if i > delay_steps
187         if abs(OutputR(i - delay_steps)) > 1e-3
188             FeedbackR = OutputR(i - delay_steps);
189             InjectedFeedback(i) = FeedbackR * exp(1i * phase_shift);
190             InputR(i) = InjectedFeedback(i);
191         else
192             InjectedFeedback(i) = 0;
193             InputR(i) = 0;
194         end
195     else
196         InjectedFeedback(i) = 0;
197         InputR(i) = 0;
198     end
199
200
201
202

```

```

203 % Input
204 InputL(i) = Ef1(t,0);
205 % InputL(i) = Ef1(t, InputParasL); % At time t, we input a signal characterized by InputParasL from the left
206 %InputR(i) = ErN(t, 0); % At time t, we input no signal from the right (since InputParasR = 0)
207
208 % Reflection
209 Ef(1) = InputL(i) + RL*Er(1); % Boundary condition at z = 0 (left side);
210 Er(Nz) = InputR(i) + RR*Ef(Nz); % Boundary condition at z = L (right side);
211
212 S = (abs(Ef).^2 + abs(Er).^2).*EtoP*1e-6;
213
214 if t < Ion || t > Ioff
215     I_injv = I_off;
216 else
217     I_injv = I_on;
218 end
219
220 Stim = gain.*(N - Ntr).*S;
221 N = (N + dt*(I_injv/ eVol - Stim))./(1+ dt/taun);
222 Nave(i) = mean(N);
223
224 gain_z = gain.*(N - Ntr)./vg; % Compute gain coefficient
225 beta_i = (gain_z - alpha)./2; % Compute imaginary part of propagation constant
226 beta = ones(size(z)).*(beta_r + 1i * beta_i); % Complex propagation constant
227 exp_det = exp(-1i * dz * beta); % Phase shift due to propagation over dz
228
229 Ef(2:Nz) = fsync*exp_det(1:Nz-1).*Ef(1:Nz-1) + 1i*dz*kappa(2:Nz).*Er(2:Nz); % Forward Field Propagation
230 Er(1:Nz-1) = fsync*exp_det(2:Nz).*Er(2:Nz) + 1i*dz*kappa(2:Nz).*Ef(2:Nz); % Reverse Field Propagation
231
232 % Boundary Conditions
233 Pf(1) = 0; % zero polarization at the left boundary
234 Pf(Nz) = 0; % zero polarization at the right boundary
235 Pr(1) = 0; % zero polarization at the right boundary
236 Pr(Nz) = 0; % zero polarization at the left boundary
237 Cw0 = -LGamma + 1i * Lw0; % Defines the complex response function of the material.
238
239 % Dispersion Calculations
240 Tf = LGamma * Ef(1:Nz-2) + Cw0 * Pfp(2:Nz-1) + LGamma * Efp(1:Nz-2); % Computes the forward polarization response
241 % based on previous field values.
242 Pf(2:Nz-1) = (Pfp(2:Nz-1) + 0.5 * dt * Tf) ./ (1 - 0.5 * dt * Cw0); % Updates the forward polarization field for
243 % every time step.
244 Tr = LGamma * Er(3:Nz) + Cw0 * Prp(2:Nz-1) + LGamma * Erp(3:Nz); % Computes the reverse polarization response
245 % based on previous field values.
246 Pr(2:Nz-1) = (Prp(2:Nz-1) + 0.5 * dt * Tr) ./ (1 - 0.5 * dt * Cw0); % Updates the reverse polarization field for
247 % every time step.
248
249 Ef(2:Nz-1) = Ef(2:Nz-1) - LGain * (Ef(2:Nz-1) - Pf(2:Nz-1)); % Adjusts the forward electric field
250 Er(2:Nz-1) = Er(2:Nz-1) - LGain * (Er(2:Nz-1) - Pr(2:Nz-1)); % Adjusts the reverse electric field
251
252 % Output
253 OutputR(i) = Ef(Nz) * (1 - RR); % Right output at z = L
254 OutputL(i) = Er(1) * (1 - RL); % Left output at z = 0
255
256 delay_steps = round(1000e-12/dt);
257 phase_shift = pi/6;
258
259 if i == delay_steps + 1
260     FeedbackR = OutputR(i - delay_steps); % Right-side output
261     InjectedFeedback(i) = FeedbackR * exp(1i * phase_shift); % Phase shift
262     InputR(i) = InjectedFeedback(i); % Inject to right side
263 end
264
265 A = sqrt(gamma*beta_spe*c_hb*f0*L*1e-2/taun)/(2*Nz);
266 if SPE > 0
267     eTf = ((randn(Nz,1)+1i*randn(Nz,1))*A).';
268     eTr = ((randn(Nz,1)+1i*randn(Nz,1))*A).';
269 else
270     eTf = ((ones(Nz,1))*A).';
271     eTr = ((ones(Nz,1))*A).';
272 end
273
274 EsF = eTf*abs(SPE).*sqrt(N.*1e6);
275 Esr = eTr*abs(SPE).*sqrt(N.*1e6);
276
277 Ef = Ef + EsF;
278 Er = Er + Esr;
279
280 % % FFT data from the outputs
281 fftOutput1 = fftshift(fft(OutputR)); % Get FFT data for OutputR
282 fftOutput2 = fftshift(fft(OutputL)); % Get FFT data for OutputL
283 fftInput1 = fftshift(fft(InputL)); % Get FFT data for InputL
284 fftFeedback = fftshift(fft(InjectedFeedback));
285 omega = fftshift(wspace(time));
286
287 if mod(i,2000) == 0 % Only executed when i is multiple of plotN
288     % Forward field E_f
289     subplot(3,2,1)
290     plot(z*10000, real(Ef), 'r'); hold on
291     plot(z*10000, imag(Ef), 'r--');

```

```

291 plot(z*10000, real(Er), 'b--');
292 plot(z*10000, imag(Er), 'b');
293 hold off
294 xlim(XL*1e4)
295 ylim auto
296 xlabel('z (\mum)')
297 ylabel('E_f (V/\mum)')
298 legend('\Re(E_f)', '\Im(E_f)', '\Re(E_r)', '\Im(E_r)')
299
300
301 % Carrier Density N
302 subplot(3,2,2)
303 plot(z * 10000, N, 'r'); % Primary plot
304 xlim(XL * 1e4)
305 ylim([0, 5 * Ntr])
306 xlabel('z (\mum)')
307 ylabel('N')
308
309 % Add a second y-axis for S
310 yyaxis right
311 plot(z * 10000, S, 'b'); % Plot S on the right axis
312 ylabel('S') % Change the label to the appropriate units
313
314 % Average Carrier Density Over Time
315 subplot(3,2,3)
316 plot(time * 1e12, Nave, 'b');
317 xlim([0, Nt * dt * 1e12])
318 ylim([0, 2.5 * Ntr])
319 xlabel('time(ps)')
320 ylabel('Nave')
321
322 subplot(3,2,5)
323 plot(time * 1e12, real(InjectedFeedback), 'k--'); hold on
324 plot(time * 1e12, real(OutputR), 'g');
325 plot(time * 1e12, real(OutputL), 'm--');
326 xlim([0, Nt * dt * 1e12])
327 ylim auto
328 xlabel('time(ps)')
329 ylabel('E (V/\mum)')
330 legend('Injected Feedback (Right)', 'OutputR', 'OutputL', 'Location', 'east')
331 hold off
332
333
334
335 subplot(3,2,4)
336 plot(omega, 20*log10(abs(fftOutput1)), 'b'); hold on
337 plot(omega, 20*log10(abs(fftOutput2)), 'r');
338 plot(omega, 20*log10(abs(fftFeedback)), 'k--'); % From InjectedFeedback
339 xlabel('Frequency (Hz)')
340 ylabel('20 log_{10} |E|')
341 legend('OutputR', 'OutputL', 'Feedback')
342 xlim([-0.05e14 0.05e14])
343 hold off
344
345
346 subplot(3,2,6)
347 phase1 = unwrap(angle(fftOutput1));
348 phase2 = unwrap(angle(fftFeedback));
349 plot(omega, phase1); hold on
350 plot(omega, phase2);
351 xlabel('Frequency (Hz)')
352 ylabel('phase (E)')
353 legend('OutputR', 'Feedback');
354 hold off
355
356 pause(0.01)
357 % UNCOMMENT IF YOU WANT TO REPLICATE THE PLOTS I HAVE IN MY REPORT
358 % figure(2)
359 % clf
360 % plot(omega, 20*log10(abs(fftOutput1)), 'b'); hold on
361 % plot(omega, 20*log10(abs(fftOutput2)), 'r');
362 % plot(omega, 20*log10(abs(fftFeedback)), 'k--'); % From InjectedFeedback
363 % xlabel('Frequency (Hz)')
364 % ylabel('20 log_{10} |E|')
365 % legend('OutputR', 'OutputL', 'Feedback')
366 % xlim([-0.05e14 0.05e14])
367 % title('Spectral Response with Feedback at 500 ps')
368 % hold off
369 % drawnow
370 %
371 % figure(3)
372 % clf
373 % plot(time * 1e12, real(InjectedFeedback), 'k--'); hold on
374 % plot(time * 1e12, real(OutputR), 'g');
375 % plot(time * 1e12, real(OutputL), 'm--');
376 % xlim([0, Nt * dt * 1e12])
377 % ylim auto
378 % xlabel('Time (ps)')
379 % ylabel('E (V/\mum)')
380 % legend('Injected Feedback (Right)', 'OutputR', 'OutputL', 'Location', 'east')
381 % title('Field vs Time with Injected Feedback at 500 ps')
382 % hold off

```



```
383 % drawnow
384
385
386 end
387
388
389 % Update Previous Values
390 Efp = Ef;
391 Erp = Er;
392 Pfp = Pf;
393 Prp = Pr;
394 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>