Quantum Mechanics: Electromagnetically-Induced transparency

Abstract

This experiment simulates a Lambda three-level atomic system in order to examine the phenomenon of electromagnetically induced transparency (EIT). The tasks involve calculating the Hamiltonian, density matrix, and numerically solving the dynamics of the system. Research includes factors influencing transitions, the impact of probe fields on atomic populations, and simulations using different intensities of pump fields and activation of probe fields. The study offers valuable insights into the phenomena of electromagnetically induced transparency (EIT) and their relationships with field strengths and activation timings. This contributes to a better understanding of the interaction between light and atoms in three-level atomic systems.

1 INTRODUCTION & THEORETICAL BACKGROUND

Quantum mechanics arose due to the limitations of classical physics in describing phenomena occurring at the atomic and subatomic scales. While classical mechanics performed a great job of explaining the macroscopic universe, it was unable to explain the unusual behaviour of particles such as electrons and photons. This motivated a new theory and approach, quantum mechanics, to resolve inconsistencies in describing phenomena such as black-body radiation, the photoelectric phenomenon, and the atomic spectra (Mann, 2022).

Max Planck's pivotal proposal in 1900 introduced a revolutionary concept: that light, with frequency ν, is emitted in discrete packets of energy, quantized as integral multiples of $E = h\nu = \hbar\omega$, where h is the Planck's constant, $\hbar$ is the reduced Planck's constant, and ν and ω are the frequency and angular frequency respectively (Morin, 2008). This groundbreaking idea laid the groundwork for quantum theory, challenging the very foundations of classical physics.

Quantum mechanics diverges from classical mechanics' deterministic laws, embracing a framework grounded in probability and statistics. Moreover, it uncovered the wave-particle duality, introducing concepts like wave functions, superposition, and entanglement while establishing fundamental principles such as the Heisenberg uncertainty principle.

The evolution of the atomic model began with Dalton's idea of tiny, unbreakable spheres and continued through Thomson's discovery of electrons and Rutherford's nuclear model. Bohr proposed that electrons revolve around the nucleus in fixed orbits (Gallagher & Ingram, 2015), but the subsequent quantum mechanical model developed by Schrödinger and Heisenberg embraced a different concept. Instead of specific orbits, they described where electrons might be found using probability within regions called orbitals.

In this new model, electrons are viewed as waves, making their behaviour a matter of probability. By employing the Schrödinger equation, various wave functions are derived to describe electron behaviour and how these waves change over time. Consequently, these functions help calculate the likelihood of finding electrons in different positions. These probabilities give rise to atomic orbitals, illustrating where an electron is likely to be found (Romforn, 2023).

Additionally, this model relies on four quantum numbers: n, l, m_l, and m_s for a wave function. The principal quantum number n defines the orbital's size and energy, while the azimuthal quantum number l represents the orbital's angular momentum. The magnetic quantum number m_l describes the electron's orientation within a sub-level, and the spin quantum number m_s signifies the electron's spin (either "spin-up" or "spin-down") (Silberberg et al., 2006). Furthermore, m_s must adhere to the Pauli exclusion principle, indicating that within the same orbital, two electrons must have opposite spins.

Lasers function based on the principle of stimulated emission, wherein excited atoms emit photons that induce identical emissions in a gain medium. This leads to a population inversion, where there are a greater number of atoms in an excited state compared to those in a lower-energy state (Svelto, Hanna, et al., 2010) Inside an optical resonator (set of reflecting mirrors), these photons undergo amplification and reflection, resulting in the generation of coherent and monochromatic light. The essential elements consist of the gain medium and resonator, which are vital for producing a concentrated, high-energy beam. The foundation was established by Einstein's hypothesis of stimulated emission. Laser applications encompass a wide range of disciplines such as medicine, telecommunications, production, and others, exerting influence on various industries due to their distinct qualities and accurate light attributes.

A relatively simple laser system is comprised of three levels: a ground state, an excited state, and a metastable state. Atoms are commonly stimulated from their lowest energy level, known as the ground state, to a higher energy state called the excited state. However, they rapidly transition back to a relatively long-lived level known as the metastable state. Nevertheless, the metastable state, characterised by an long lifetime duration, requires a greater amount of time for atoms to undergo decay and return to the ground state. The transition from state 2 to 1 is known as the lasing transition. Imposing additional limitations on this system leads to the formation of a "new" system known as a (Λ) lambda 3 level system. In this system, transitions are possible between states 1 and 3, 2 and 3, but not between states 1 and 2. Prior to undergoing transitions, they are required to obey the selection rules, which are: $\Delta l = \pm 1$ and $\Delta m_l = 0$ or ± 1 (Sakurai & Commins, 1995).

Electromagnetically Induced Transparency (EIT) is a phenomenon that occurs in a three-level system, specifically a (Λ) lambda system, when it is exposed to a strong control field. Typically, atoms that are stimulated to reach an excited state rapidly transition to a metastable state. Nevertheless, if a potent control field is synchronised with the transition from the excited to the metastable states, it modifies the atomic behaviour in a manner that causes the excited state to divide into further levels. Under these conditions, the atoms exhibit transparency to a probing field inserted between the ground state and the excited state, preventing the atoms from moving from the ground state to the excited state (Finkelstein et al., 2023).

The objective of the experiment is to reproduce a (Λ) lambda three-level laser configuration and confirm the existence of electromagnetically induced transparency (EIT) in the presence of a strong electromagnetic field. This entails the examination of fluctuations in state populations when using different pump and probe fields. The objective of simulating this system is to replicate the circumstances in which EIT occurs and demonstrate the alterations in the population distribution among various atomic states based on the intensity and interaction of the pump and probe fields.

2 MATERIALS & COMPUTATIONAL TOOLS

The simulation employed Python, utilizing `SymPy` for symbolic mathematics to specify crucial system components such as the Hamiltonian and density matrices. `NumPy` was responsible for performing numerical computations, while `SciPy`'s `odeint` function was utilized for solving differential equations. `Matplotlib` was employed to visualize the population dynamics over time.

3 METHODOLOGY

3.1 Task 1

The Hamiltonian governing the system was provided as,

$$\hat{H} = \hat{H}_0 + \hat{H}_{\text{pump}} + \hat{H}_{\text{probe}} \tag{1}$$

where,

$$\begin{aligned}
\hat{H}_0 &= \hbar\omega_1|1\rangle\langle 1| + \hbar\omega_2|2\rangle\langle 2| + \hbar\omega_3|3\rangle\langle 3|, \\
\hat{H}_{\text{pump}} &= \hbar\varepsilon_{\text{pump}} e^{-i\omega_{\text{pump}}t}|2\rangle\langle 3| + \hbar\varepsilon_{\text{pump}} e^{i\omega_{\text{pump}}t}|3\rangle\langle 2|, \\
\hat{H}_{\text{probe}} &= \hbar\varepsilon_{\text{probe}} e^{-i\omega_{\text{probe}}t}|1\rangle\langle 3| + \hbar\varepsilon_{\text{probe}} e^{i\omega_{\text{probe}}t}|3\rangle\langle 1|.
\end{aligned}$$

Where the states 1, 2 and 3 are denoted as $|j\rangle$ (for $j = 1, 2, 3$) with the respective frequency ω_j. The strengths of the pump and probe fields, denoted as $\varepsilon_{\text{pump}}$ and $\varepsilon_{\text{probe}}$. Moreover, the frequencies $\omega_{\text{pump}}, \omega_{\text{probe}}$ are defined as: $\omega_{\text{pump}} = \omega_{23} + \Delta_{\text{pump}}$ and $\omega_{\text{probe}} = \omega_{13} + \Delta_{\text{probe}}$, where the difference frequencies are represented as $\omega_{jk} = \omega_k - \omega_j$ (for $j, k = 1, 2, 3$). $\Delta_{\text{pump}}, \Delta_{\text{probe}}$ denote the difference between the frequency of the pump field/probe field and and resonant frequency of the state interacting with (detuning). In this case pump field is between state 2 and 3, while the probe field is between state 1 and 3.

First we set following : $\Omega_1 = \omega_1 - \Delta_{\text{probe}}$, $\Omega_2 = \omega_2 - \Delta_{\text{pump}}$, and $\Omega_3 = \omega_3$.

then the following substitution was done on the Hamiltonian, to remove any oscillations and time dependent terms.

$$\begin{aligned}
|j_i\rangle &\to e^{-i\Omega_j t}|j_i\rangle \quad (\text{for } j = 1, 2, 3), \\
\hat{H} &\to \hat{H} - \sum_j \hbar\Omega_j |j_i\rangle\langle j_i|.
\end{aligned}$$

3.2 Task 2

The final form of the Hamiltonian obtained in task 1 was determined and simplified. Then noting that the atom has three levels one could represent such Hamiltonian in a 3 dimensional system, i.e. a matrix of size 3×3.

This can easily be done by observing the outer product of the states, which indicate the row and column of such matrix. For example the state:

$$\hbar\varepsilon_{\text{pump}}|2\rangle\langle 3| \tag{2}$$

represent the element, $\hbar\varepsilon_{\text{pump}}$ in the matrix corresponding to row 2 and column 3. This was done for all states to obtain a 3-dimensional matrix representing the Hamiltonian:

$$\hat{H} = \begin{pmatrix} \hbar\Delta_{\text{probe}} & 0 & \hbar\varepsilon_{\text{probe}} \\ 0 & \hbar\Delta_{\text{pump}} & \hbar\varepsilon_{\text{pump}} \\ \hbar\varepsilon_{\text{probe}} & \hbar\varepsilon_{\text{pump}} & 0 \end{pmatrix}$$

Moreover the density matrix was defined as:

$$\rho = \begin{pmatrix} \rho_{11} & \rho_{12} & \rho_{13} \\ \rho_{21} & \rho_{22} & \rho_{23} \\ \rho_{31} & \rho_{32} & \rho_{33} \end{pmatrix}$$

3.3 Task 3

The following differential equation governing this system is given by ,

$$\dot{\rho} = \frac{i}{\hbar}[\rho, \hat{H}] + \mathcal{L}[\rho], \tag{3}$$

where

$$\mathcal{L}[\rho] = \gamma \sum_{j=1,2} |j\rangle\langle 3|\rho|3\rangle\langle j| - \frac{1}{2}|3\rangle\langle 3|\rho + \rho|3\rangle\langle 3| \tag{4}$$

Initially, we split the differential equation into two parts, the real part and the imaginary part. Then we solved each part separately but simultaneously. This was done by using the `odeint` function in Python, where `odeint` is a function that numerically solves ordinary differential equations. After solving and obtaining the elements of the density matrix, the diagonal elements were extracted, since they consist of the varying population in states $|1\rangle, |2\rangle$ and $|3\rangle$. This was repeated multiple times, but each time changing the value and behaviour of $\varepsilon_{\text{pump}}$ and $\varepsilon_{\text{probe}}$

After deriving the elements of the density matrix, the subsequent step involved plotting graphs showcasing the population against time in seconds. This was executed across diverse strengths of the probe and pump fields, along with different initial population states. Additionally, two specific tests were conducted: one where the probe field was abruptly activated and another where its strength was gradually increased. It's essential to note that the probe field strength consistently remained significantly lower than that of the pump field.

4 RESULTS

Figure 1 illustrates the population dynamics when initialised in state $|2\rangle$, with only the probe field activated. Evidently, the population remains constant as the probe field lacks coherence with the $|2\rangle \rightarrow |3\rangle$ transition, resulting in no atom transitions.

Moving to Figure 2, it displays the population evolution starting from state $|1\rangle$, solely with the probe field active. Here, a transition from state $|1\rangle$ to state $|3\rangle$ is observed, followed by decay due to spontaneous emission to state $|2\rangle$, where the population stabilises. It stabilises in state $|2\rangle$ since, there is no pump field active and thus, transition from state $|2\rangle \rightarrow |3\rangle$ do not occur.

Figures 3 to 7 depict the population variation with the initial state as $|1\rangle$ and the probe field suddenly activated at $t = 10$. As the subsequent graphs progress, there's an increase in the pump field. It's noticeable that with the increase in the pump field strength, the probe field becomes increasingly transparent, indicative of the occurrence of electromagnetically induced transparency (EIT). Notably, oscillations arise when the probe field is abruptly switched on.

In contrast, Figures 8 to 12 maintain the same initial conditions but introduce a slow increase in the strength of the probe field. Eventually, with the pump field surpassing the probe field significantly, EIT emerges once more. However, in this scenario, there are no oscillations; instead, a smooth transition is observed.

It is important to note that throughout this results the value of the probe field strength was always much less than the value of the pump field strength. This is done in order to observe and verify the quantum phenomena EIT.

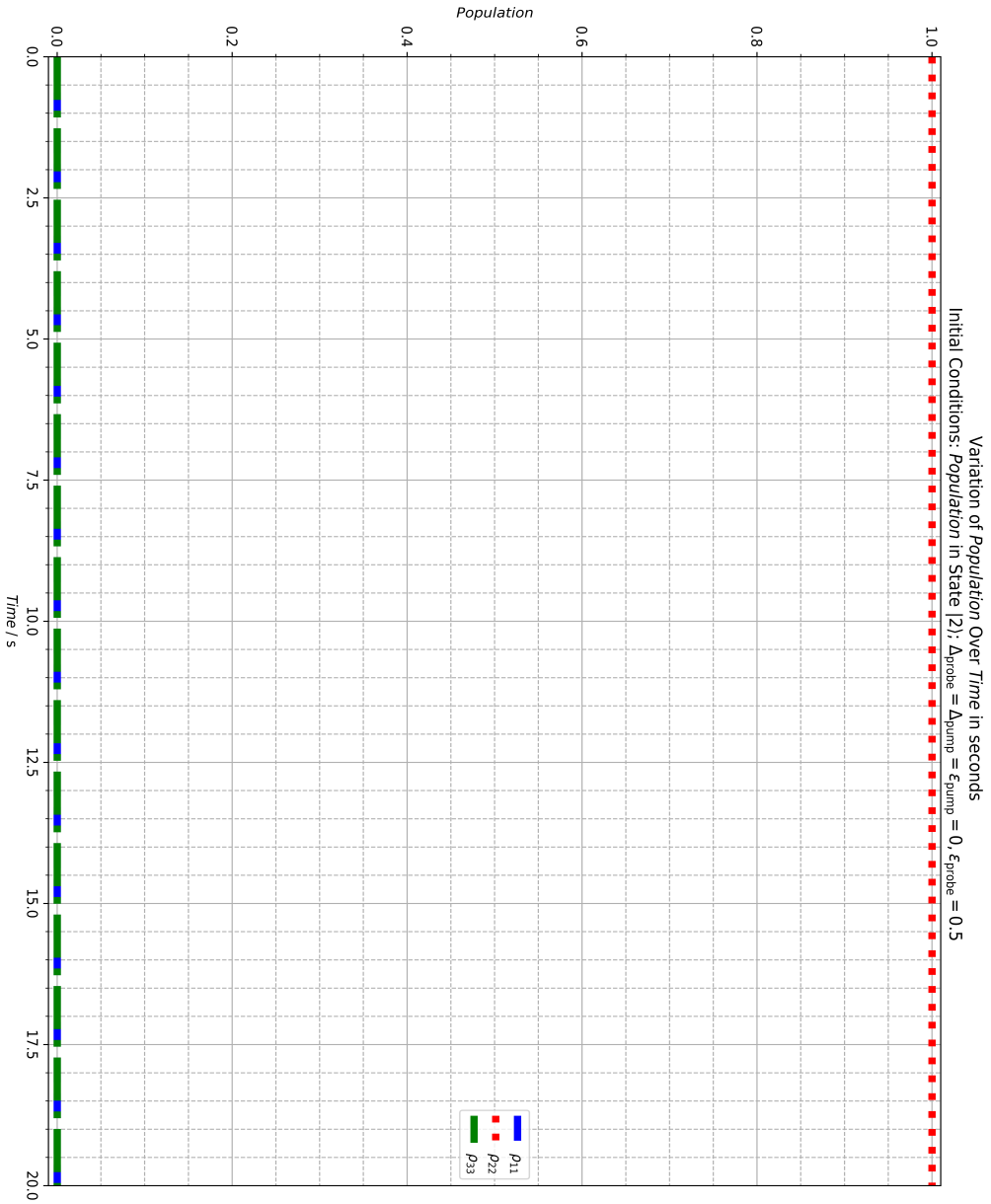

Fig. 1. Showing how the $Population$ vary with $time$ in seconds.

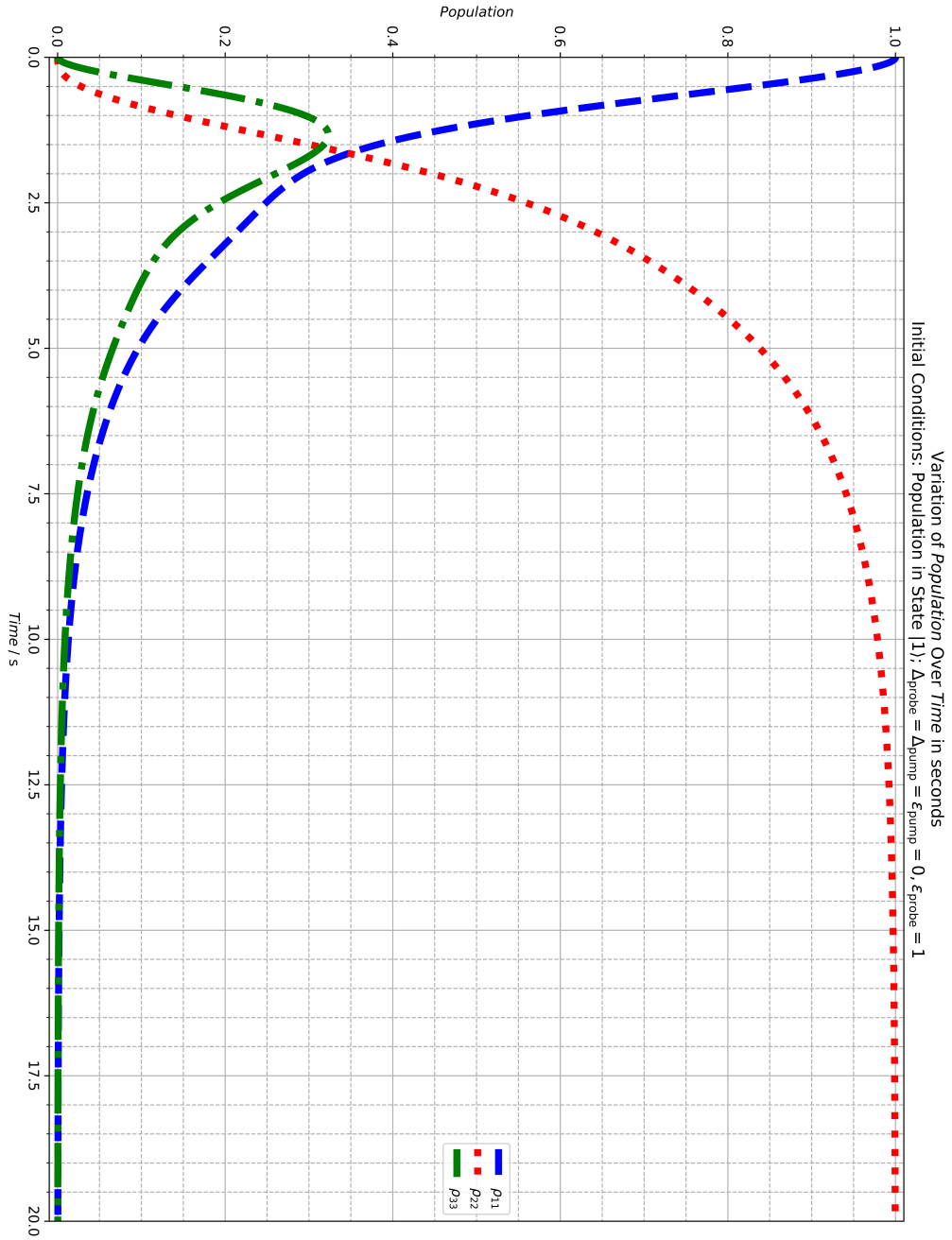

Fig. 2. Showing how the $Population$ vary with $time$ in seconds.

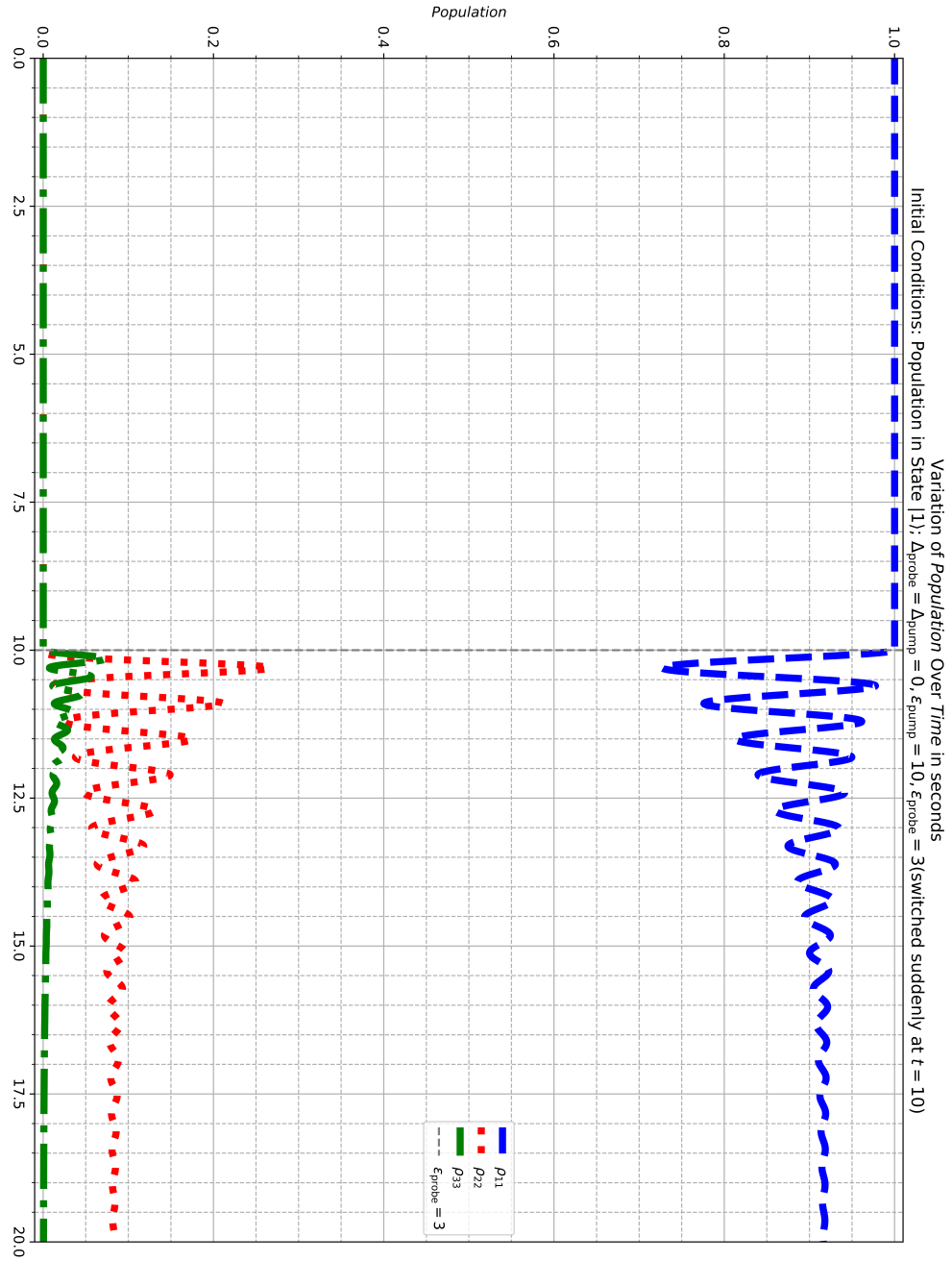

Fig. 3. Showing how the $Population$ vary with $time$ in seconds.

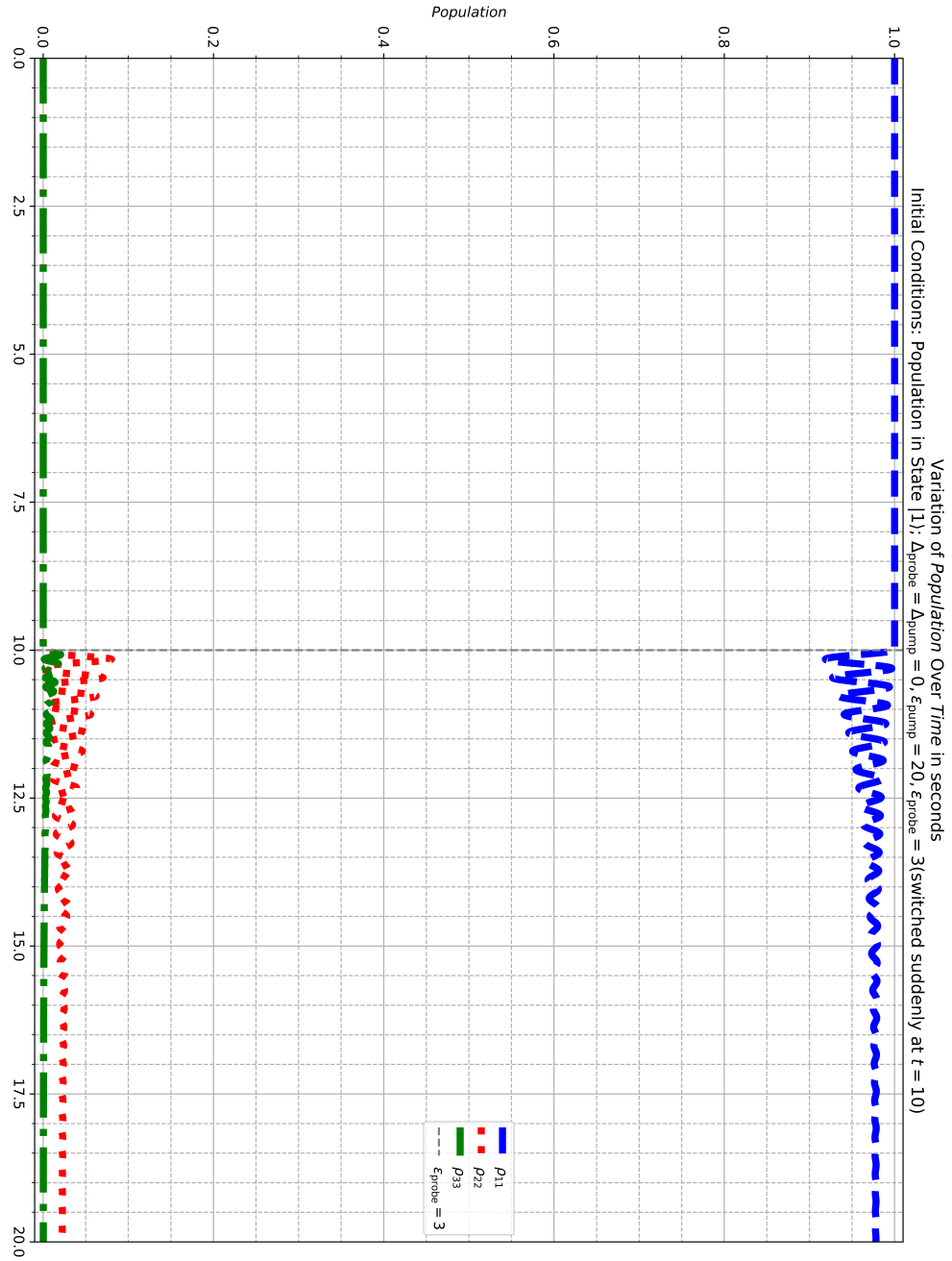

Fig. 4. Showing how the $Population$ vary with $time$ in seconds.

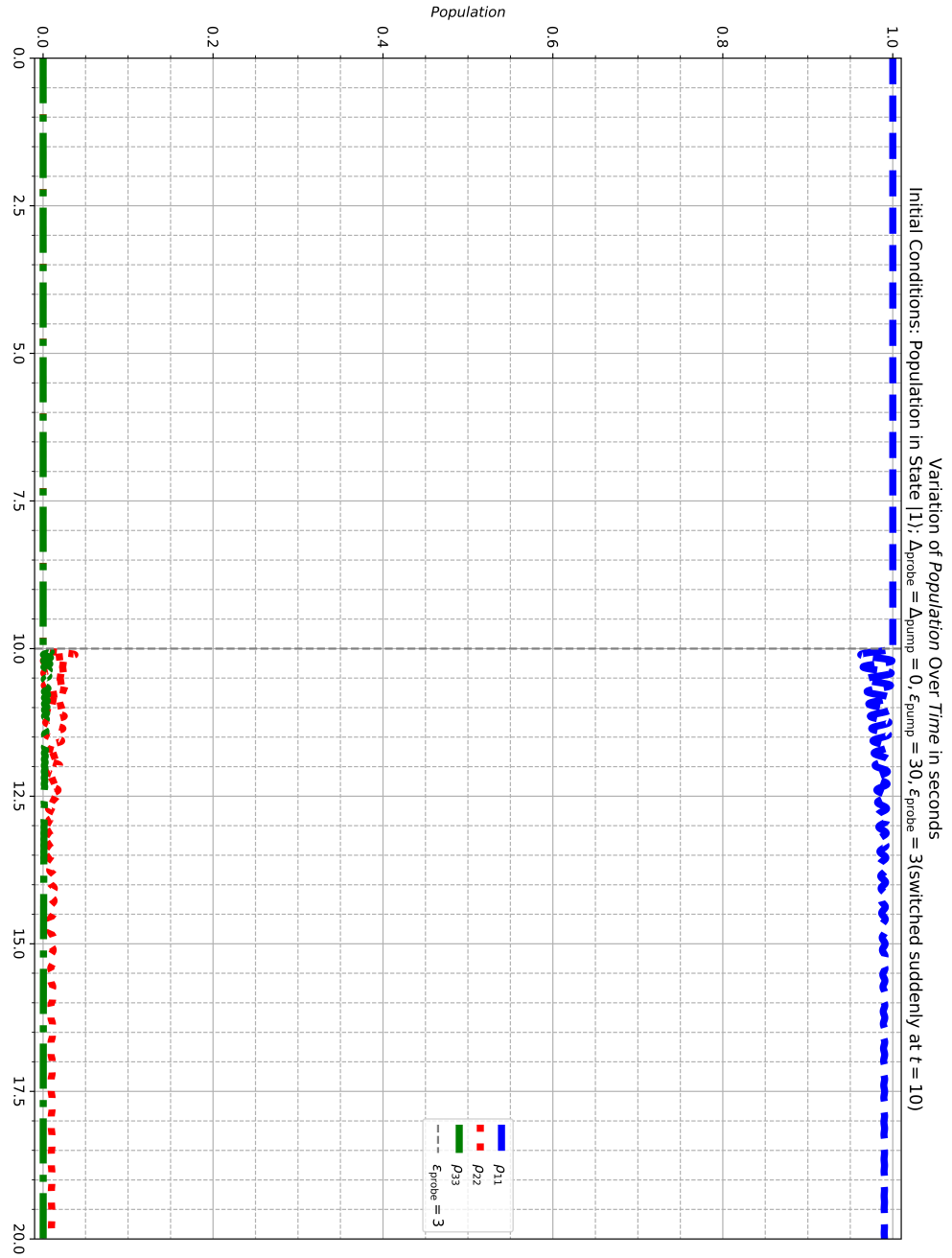

Fig. 5. Showing how the $Population$ vary with $time$ in seconds.

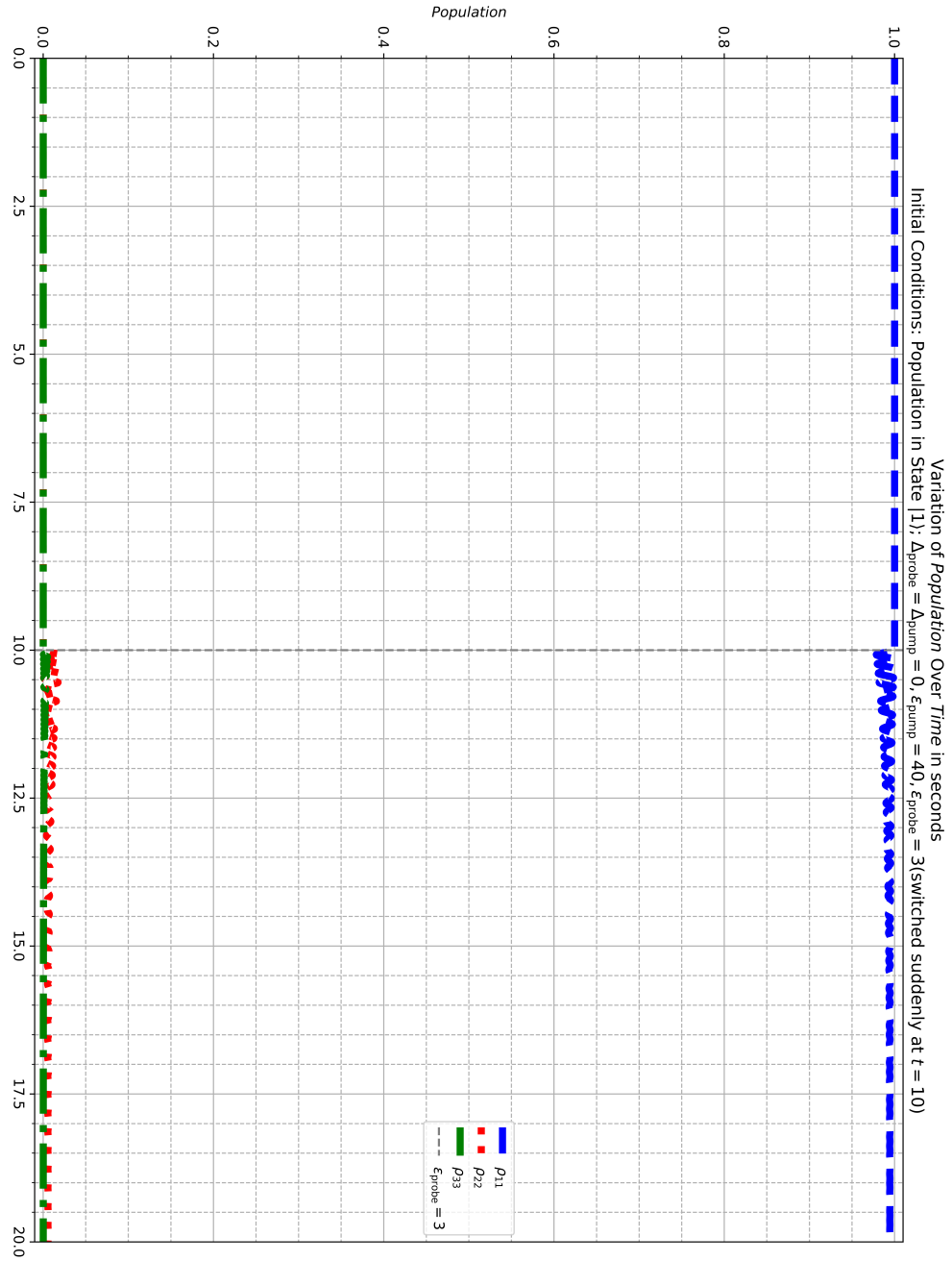

Fig. 6. Showing how the $Population$ vary with $time$ in seconds.

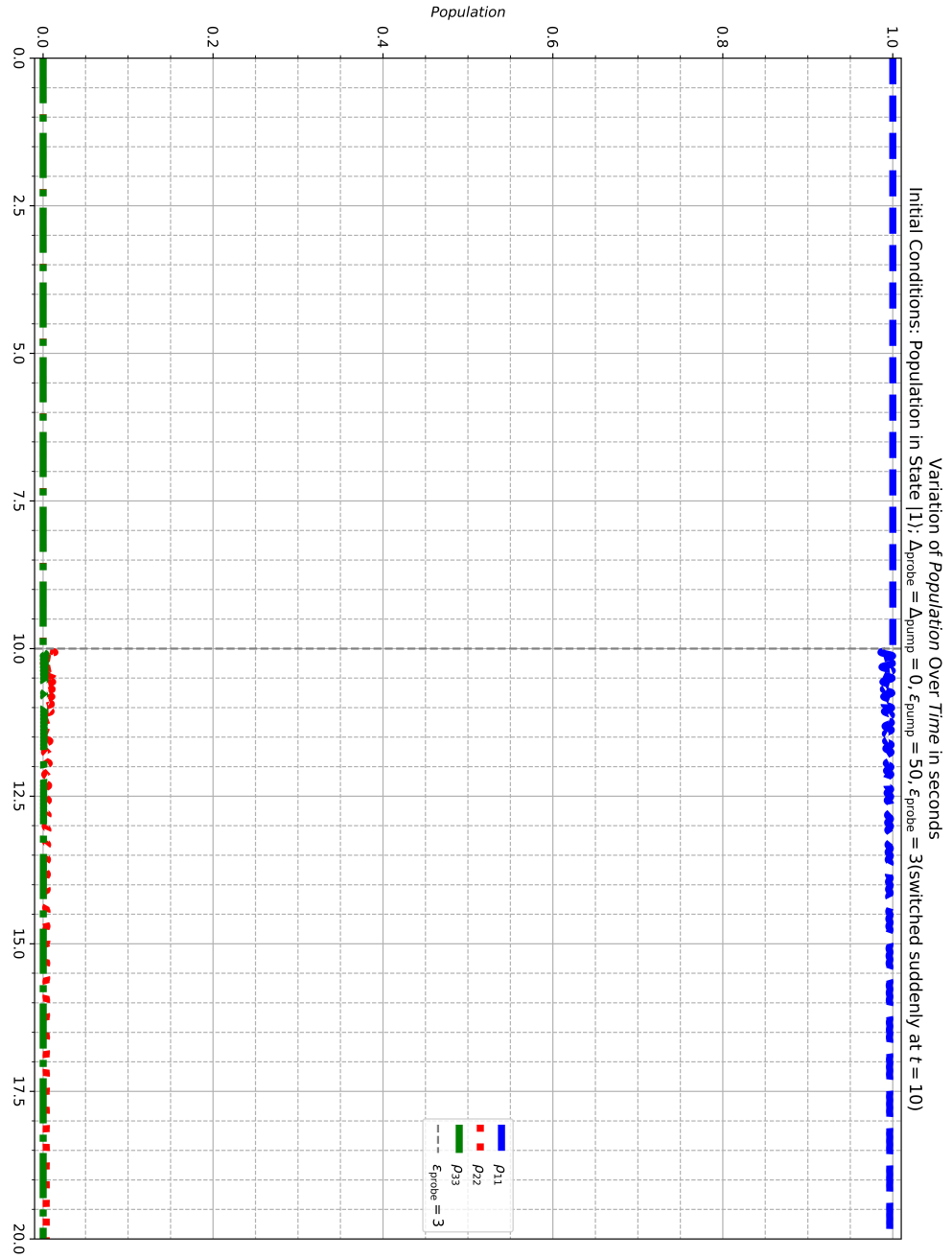

Fig. 7. Showing how the $Population$ vary with $time$ in seconds.

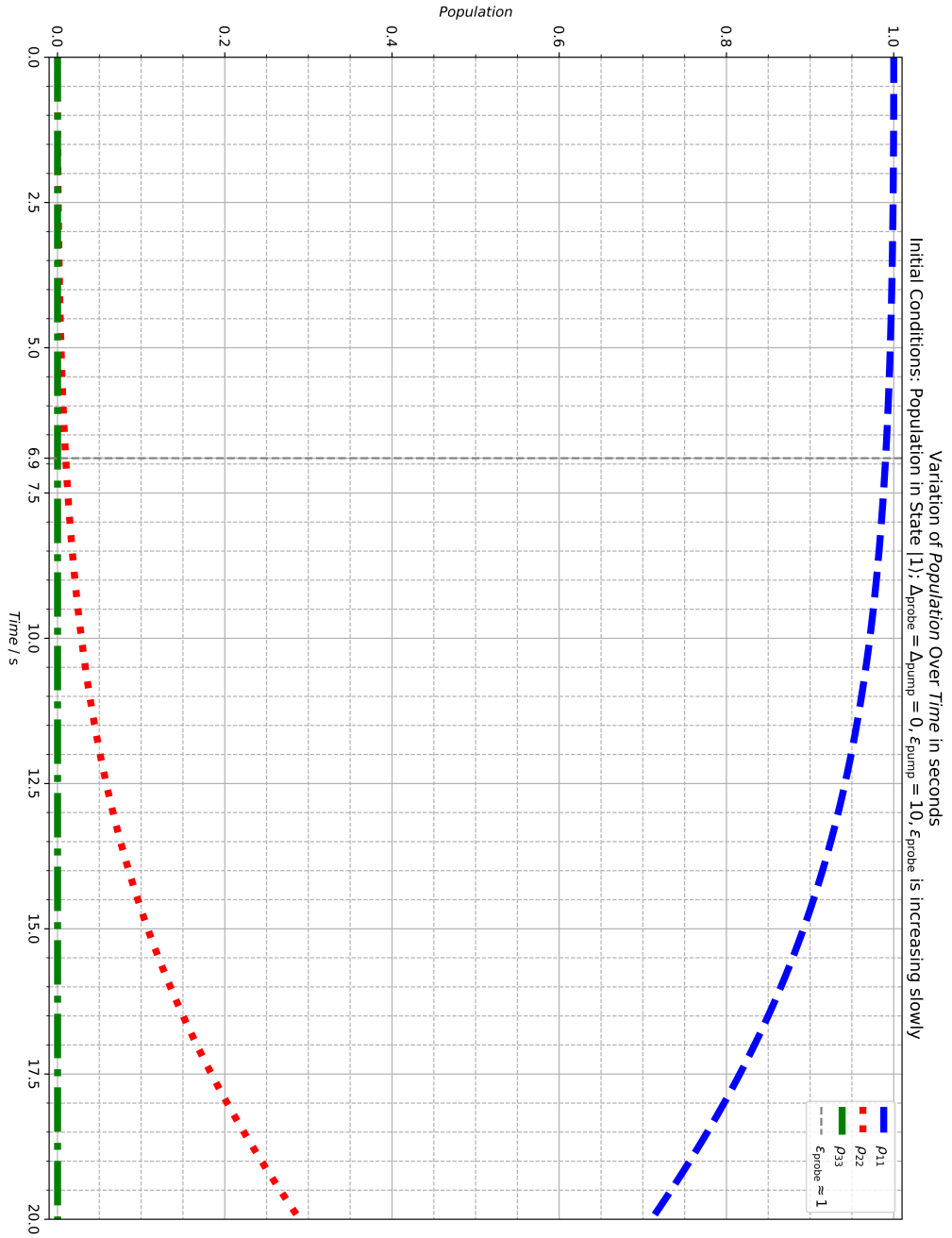

Fig. 8. Showing how the $Population$ vary with $time$ in seconds.

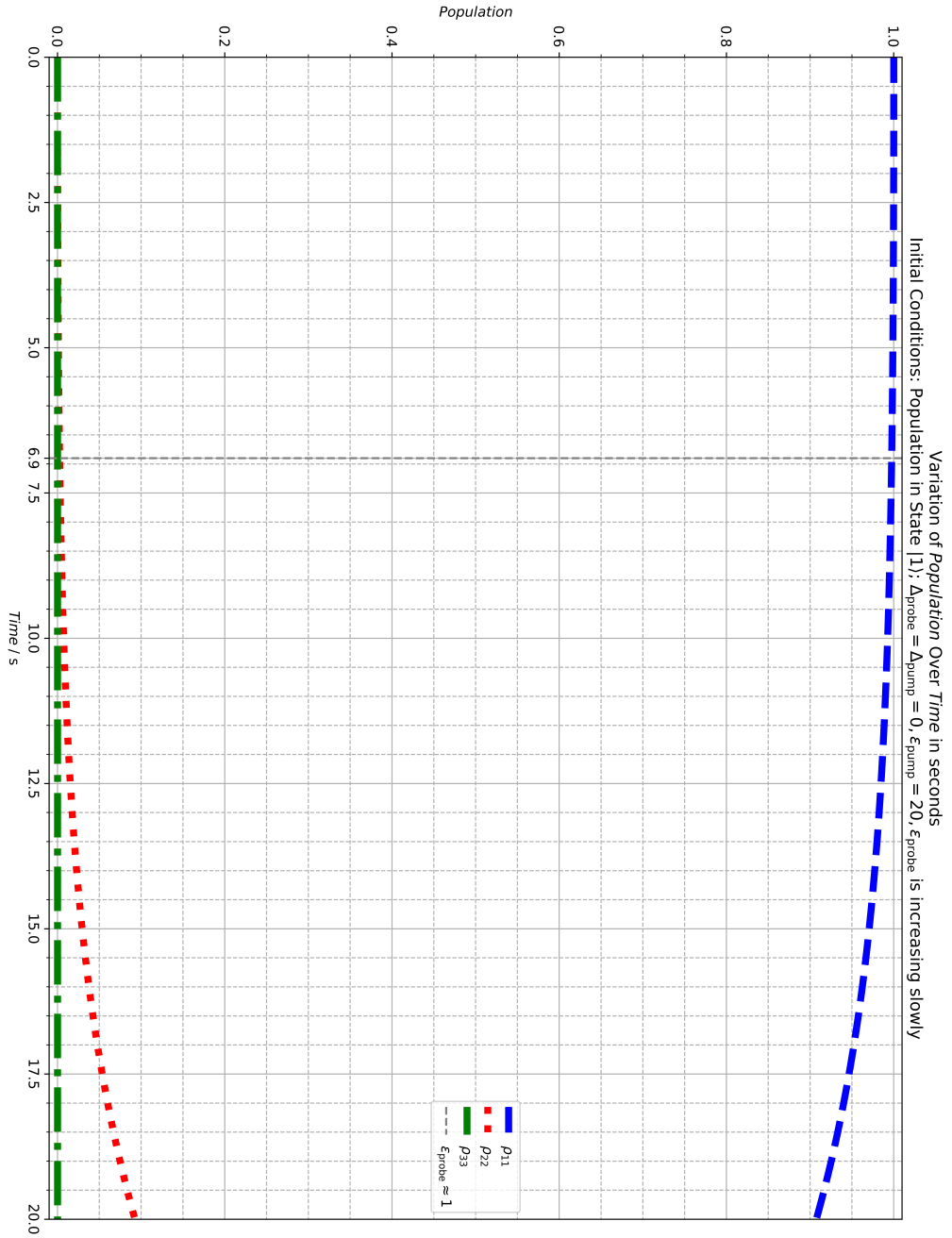

Fig. 9. Showing how the $Population$ vary with $time$ in seconds.

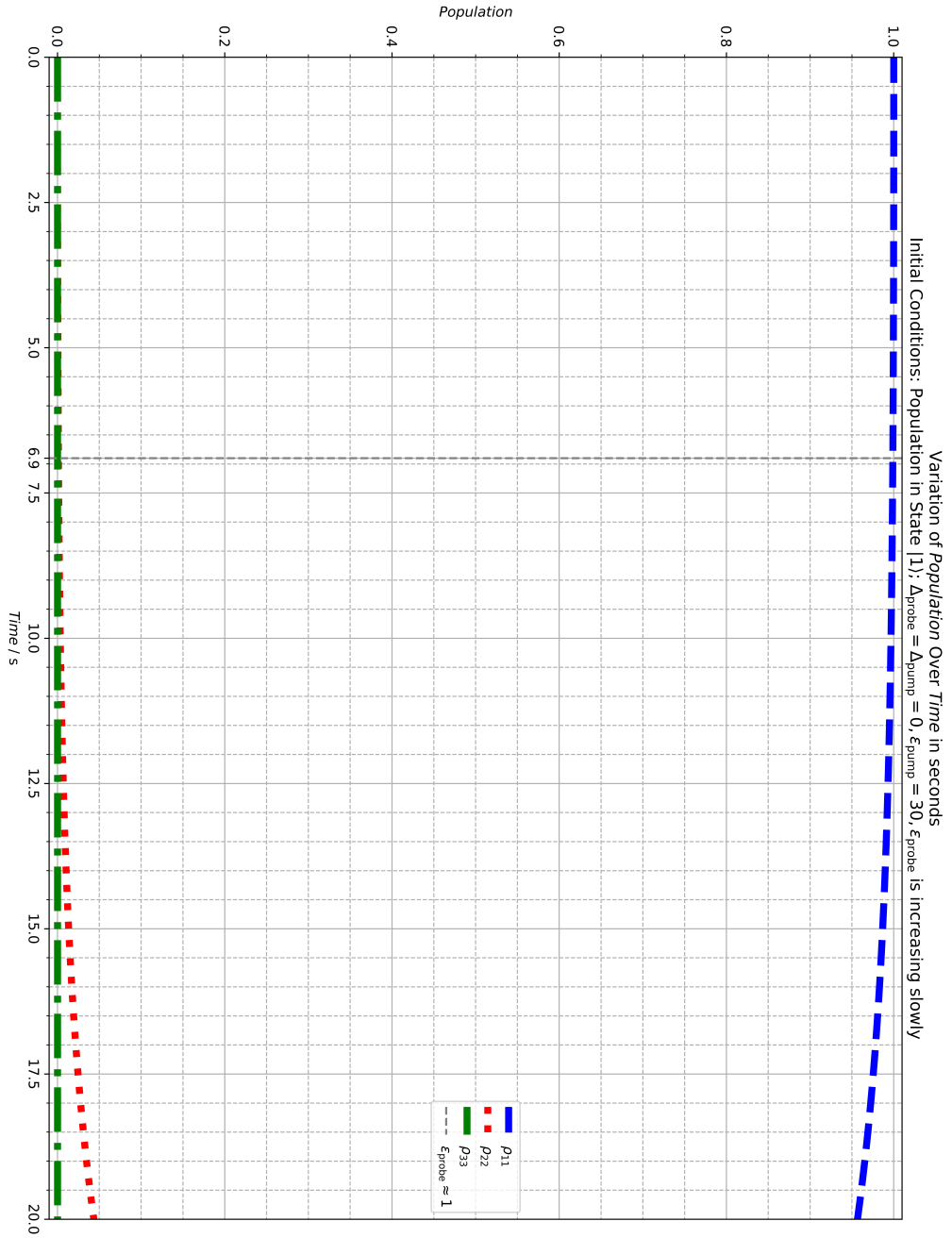

Fig. 10. Showing how the $Population$ vary with $time$ in seconds.

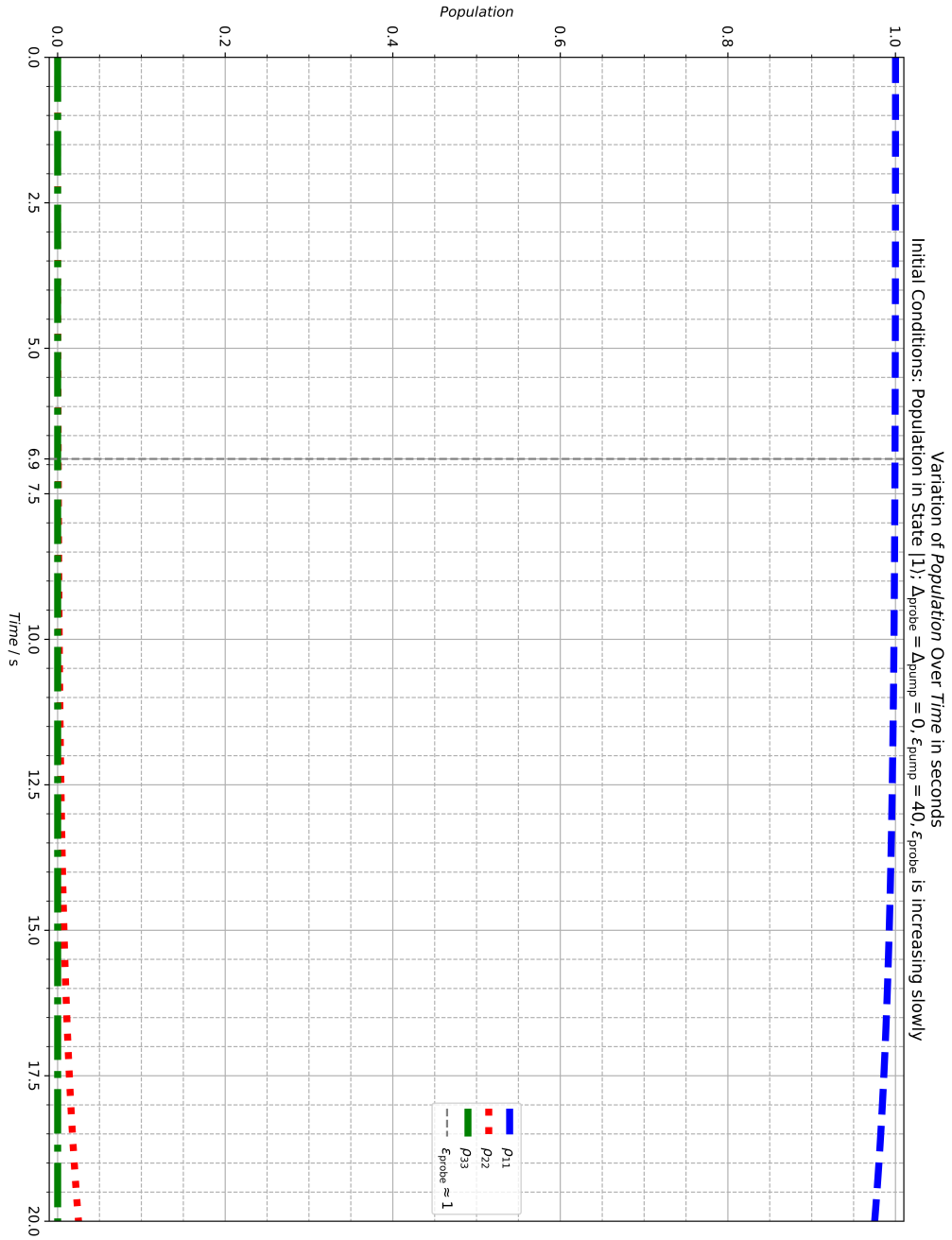

Fig. 11. Showing how the $Population$ vary with $time$ in seconds.

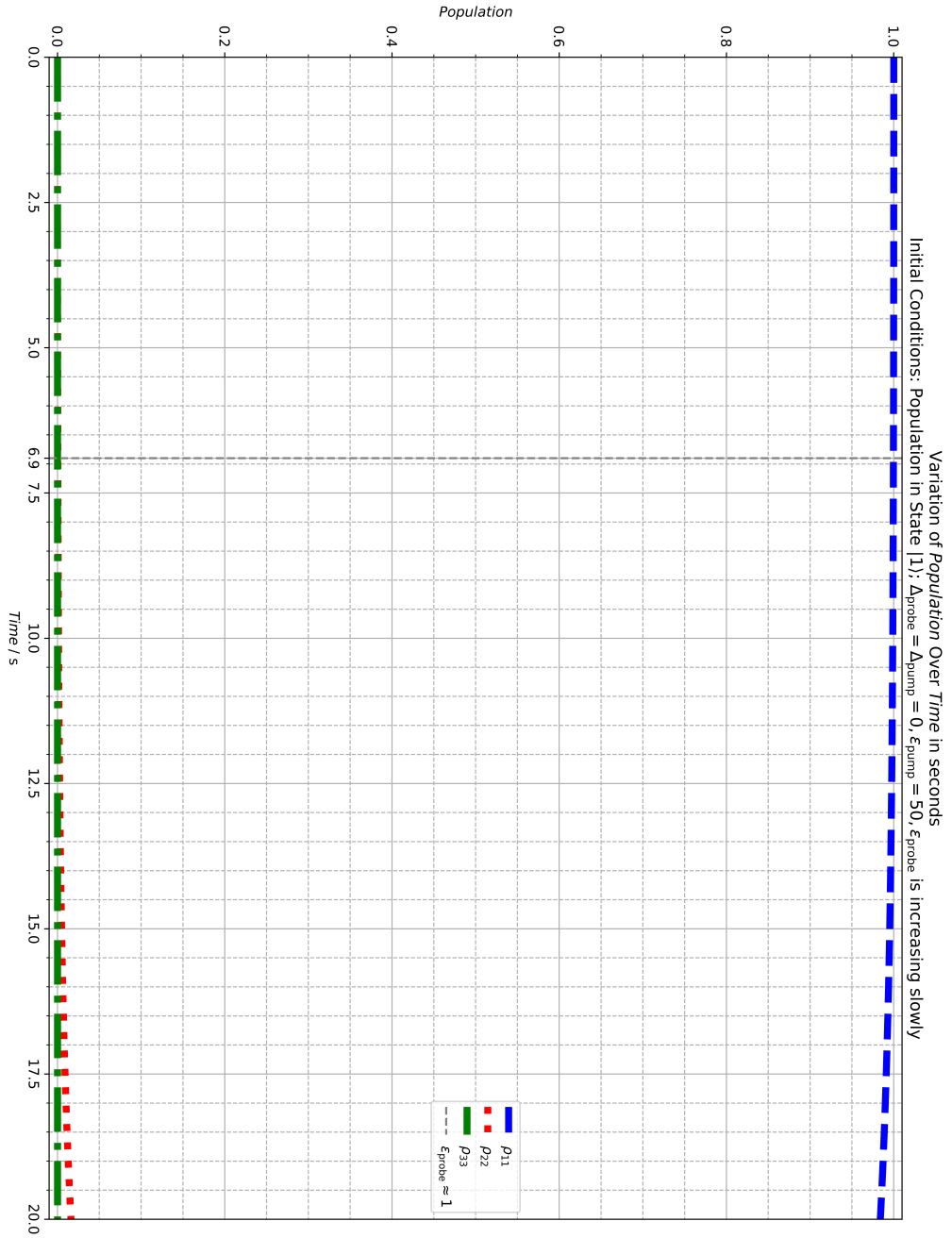

Fig. 12. Showing how the $Population$ vary with $time$ in seconds.

5 ANALYSIS

5.1 Task 1A

Setting the following substitutions for equation 1:

$$\Omega_1 = \omega_1 - \Delta_{\text{probe}}$$
$$\Omega_2 = \omega_2 - \Delta_{\text{pump}}$$
$$\Omega_3 = \omega_3.$$

$$|j_i\rangle \to e^{-i\Omega_j t}|j_i\rangle \quad (\text{for } j = 1, 2, 3),$$
$$\hat{H} \to \hat{H} - \sum_j \hbar\Omega_j |j_i\rangle\langle j_i|.$$

then we get:

$$\hat{H_0} = \hbar\Big(\omega_1|1\rangle\langle 1| + \omega_2|2\rangle\langle 2| + \omega_3|3\rangle\langle 3|\Big) \tag{5}$$

$$\hat{H}_{\text{pump}} = \hbar\varepsilon_{\text{pump}}\Big(e^{-it(\Omega_3-\Omega_2+\Omega_2-\Omega_3)}|2\rangle\langle 3| + e^{it(\Omega_3-\Omega_2-\Omega_3+\Omega_2)}|3\rangle\langle 2|\Big) \tag{6}$$

simplifying,

$$\hat{H}_{\text{pump}} = \hbar\varepsilon_{\text{pump}}\Big(|2\rangle\langle 3| + |3\rangle\langle 2|\Big) \tag{7}$$

$$\hat{H}_{\text{probe}} = \hbar\varepsilon_{\text{probe}}\Big(e^{-it(\Omega_3-\Omega_1+\Omega_1-\Omega_3)}|1\rangle\langle 3| + e^{it(\Omega_3-\Omega_1-\Omega_3+\Omega_1)}|3\rangle\langle 1|\Big) \tag{8}$$

$$\hat{H}_{\text{probe}} = \hbar\varepsilon_{\text{probe}}\Big(|1\rangle\langle 3| + |3\rangle\langle 1|\Big) \tag{9}$$

moreover,

$$\sum_j \hbar\Omega_j|j_i\rangle\langle j_i| = \hbar\Big[\Big(\omega_1 - \Delta_{\text{probe}}\Big)|1\rangle\langle 1| + \Big(\omega_2 - \Delta_{\text{pump}}\Big)|2\rangle\langle 2| + \omega_3|3\rangle\langle 3|\Big] \tag{10}$$

Therefore, The new Hamiltonian is given by,

$$\hat{H} - \sum_j \hbar\Omega_j |j_i\rangle\langle j_i| = \hbar\varepsilon_{\text{pump}}\Big(|2\rangle\langle 3| + |3\rangle\langle 2|\Big) + \hbar\varepsilon_{\text{probe}}\Big(|1\rangle\langle 3| + |3\rangle\langle 1|\Big) + \hbar\Big(\Delta_{\text{probe}}|1\rangle\langle 1| + \Delta_{\text{pump}}|2\rangle\langle 2|\Big) \tag{11}$$

5.2 Task 2

The resultant Hamiltonian can be easily represented in matrix formed, since the outer product is a representation of the row (ket) and the column (bra) in a matrix.

$$\hat{H} = \begin{pmatrix} \hbar\Delta_{\text{probe}} & 0 & \hbar\varepsilon_{\text{probe}} \\ 0 & \hbar\Delta_{\text{pump}} & \hbar\varepsilon_{\text{pump}} \\ \hbar\varepsilon_{\text{probe}} & \hbar\varepsilon_{\text{pump}} & 0 \end{pmatrix}$$

Moreover the density matrix can be defined as,

$$\rho = \begin{pmatrix} \rho_{11} & \rho_{12} & \rho_{13} \\ \rho_{21} & \rho_{22} & \rho_{23} \\ \rho_{31} & \rho_{32} & \rho_{33} \end{pmatrix}$$

Where the diagonal elements represent the population in states 1, 2 and 3 respectively.

5.3 Task3

Starting from,

$$\dot{\rho} = \frac{i}{\hbar}[\rho, \hat{H}] + \mathcal{L}[\rho], \tag{12}$$

where

$$\mathcal{L}[\rho] = \gamma \sum_{j=1,2} |j\rangle\langle 3|\rho|3\rangle\langle j| - \frac{1}{2}|3\rangle\langle 3|\rho + \rho|3\rangle\langle 3| \tag{13}$$

The Hamiltonian and density matrix were defined as in Task 2. Now the Lindblad $L[\rho]$ was easily found by setting $\gamma = 1$ and simplifying the expression in equation 13 using Mathematica,

```
ket1 = {{1}, {0}, {0}};
bra1 = {{1, 0, 0}};

ket2 = {{0}, {1}, {0}};
bra2 = {{0, 1, 0}};

```

```
ket3 = {{0}, {0}, {1}};
bra3 = {{0, 0, 1}};

L = Dot[ket1, Dot[bra3, Dot[p, Dot[ket3, bra1]]]] +
   Dot[ket2, Dot[bra3, Dot[p, Dot[ket3, bra2]]]] -
   Dot[ket3, Dot[bra3, p]] - Dot[p, Dot[ket3, bra3]];
```

Thus,

$$L[\rho] = \begin{pmatrix} \rho_{33} & 0 & -\rho_{13} \\ 0 & \rho_{33} & -\rho_{32} \\ -\rho_{31} & -\rho_{32} & -2\rho_{33} \end{pmatrix}$$

Then we proceeded in finding the commutator in equation 12 using Python, along with splitting the expression in real part and imaginary part and solving the differential equation. The complex function was defined as follow,

```

def combined_differential_equations(current_rho_values, t):
    real_part = current_rho_values[:9].reshape((3, 3)).astype(float)
    imag_part = current_rho_values[9:].reshape((3, 3)).astype(float)

    rho = real_part + 1j * imag_part

    L_values = L.subs({
        rho_11: rho[0, 0], rho_12: rho[0, 1], rho_13: rho[0, 2],
        rho_21: rho[1, 0], rho_22: rho[1, 1], rho_23: rho[1, 2],
        rho_31: rho[2, 0], rho_32: rho[2, 1], rho_33: rho[2, 2]
    })

    dp = np.matmul(rho, H) - np.matmul(H, rho) + L_values
    dp_real = sp.re(sp.expand(dp))
    dp_imag = sp.im(sp.expand(dp))

    dp_real_values = np.array(dp_real).astype(float).flatten()
    dp_imag_values = np.array(dp_imag).astype(float).flatten()

    return np.concatenate((dp_real_values, dp_imag_values)).tolist()
```

Furthermore, this function was solved over a period of time t and reshaped into a matrix form, from which the diagonal elements were extracted.

```

# Initial conditions
t = np.linspace(0, 20, 2000)
```

```
y0 = np.array([0, 0, 0, 0, 1, 0, 0, 0, 0] + [0, 0, 0, 0, 0, 0, 0, 0, 0])

# Solving complex differential equation
Y = odeint(combined_differential_equations, y0, t)

# Extracting real and imaginary parts from the results
real_part = Y[:, :9].reshape((len(t), 3, 3))
imag_part = Y[:, 9:].reshape((len(t), 3, 3))

rho_matrices = real_part + 1j * imag_part

rho_11_values_all = rho_matrices[:, 0, 0]
rho_22_values_all = rho_matrices[:, 1, 1]
rho_33_values_all = rho_matrices[:, 2, 2]
```

The graphs were then plotted, showing how the variation of the population varies from state to state,

```

# Plotting, labelling.
plt.figure(figsize=(15, 11))
plt.ylim(-0.01, 1.01)
plt.xlim(0,20)
plt.plot(t, rho_11_values_all, 'b--', label=r'$\rho_{11}$', linewidth=5)
plt.plot(t, rho_22_values_all, 'r:', label=r'$\rho_{22}$', linewidth=5)
plt.plot(t, rho_33_values_all, 'g-.', label=r'$\rho_{33}$', linewidth=5)

plt.minorticks_on()
plt.grid(visible=True, which='major', linestyle='-')
plt.grid(visible=True, which='minor', linestyle='--')

plt.title('Variation of $Population$ Over $Time$ in seconds  \n Initial Conditions:
    $Population$ in State $|2\rangle$; $\Delta_{\mathrm{probe}} = \Delta_{\mathrm{pump}} = \
    epsilon_{\mathrm{pump}} = 0 , \epsilon_{\mathrm{probe}} = 0.5$')

plt.xlabel(r'$Time\;$/ s ')
plt.ylabel(r'$Population$')
plt.legend()
plt.savefig('Graph1.png', dpi=1000)
plt.show()
```

This process was repeated for different values and behaviour of the pump field and probe field.

6 DISCUSSION

The study presents simulations conducted to examine the dynamics of a 3-level (Λ) lambda system under various field conditions. The results obtained from these simulations closely align with the theoretical predictions based on quantum mechanics, specifically on the strict selection rules that govern atomic transitions. The plots derived from our tests consistently showed trends and behaviours that practically were comparable to the anticipated results. Furthermore the agreement between the simulated results and the expected theoretical predictions highlights the reliability and precision of our computational model in accurately representing the complex behaviours of the system. This not only supports the adherence to fundamental rules in quantum physics, but also strengthens the reliability of the computational approach in recreating complicated atomic transitions in the Λ system.

In order to enhance the study and eliminate limitations, it is suggested to broaden the investigation by integrating a more authentic model that encompasses the influences of decoherence ($\Delta_{\text{pump}} = \Delta_{\text{probe}} \neq 0$) and dephasing mechanisms within the lambda system. By incorporating these elements into the simulation, the research could further explore the behaviour of the system under more realistic circumstances, providing insights into the influence of environmental factors on the phenomena of electromagnetically induced transparency (EIT) (Finkelstein et al., 2023). Gaining insight into these decoherence effects could reveal the strength of EIT against many sources of noise or disruptions, thereby facilitating practical implementations in quantum information processing or quantum computing, where preserving coherence is crucial. Expanding the simulation will not only increase the range of the study but also provide essential insights into the practicality and reliability of using EIT based devices in everyday circumstances.

An interesting phenomenon was shown in the experiment. The probe field became transparent when a strong pump field was applied. This interesting result showed an expected correlation between the probe field's behaviour and the pump field's strength, known as electromagnetically induced transparency. Essentially, when the pump field was strong enough, it acted as a strong field (control field), affecting the state 3 'level' by splitting into more bands that are now incoherent with the probe field (Finkelstein et al., 2023). Resulting in the probe field becoming transparent to the medium. Moreover, the experimental findings repeatedly showed that there are no transitions from state 1 to state 2. This lack of presence confirmed the vital concept that changes between these states were not possible because of the constraints set by the selection rules, i.e. $l = \pm 1$ and $m_l = 0$ or ± 1.

The experiment on pump-probe interaction in the (Λ) lambda system yielded promising results that align with theoretical predictions. Strong pump fields can induce transparency to the probe field, as suggested by electromagnetically induced transparency theory (EIT). Interestingly, the strong control field targeting the state 2 and 3 transition effectively stops light absorption from the probe field in the state 1 and 3 transition, as predicted. To determine the optimal circumstances for transparency, one can enhance the experiment by exploring variations in the intensity and duration of the pump field. By altering the initial atomic states or field frequencies, one can also investigate the dynamic changes of the system.

REFERENCES

Finkelstein, R Bali, S Firstenberg, O & Novikova, I (2023) A practical guide to electromagnetically induced transparency in atomic vapor. *New Journal of Physics*, *25*(3) 035001. https://doi.org/10.1088/1367-2630/acbc40

Gallagher, R & Ingram, P (2015) *Complete chemistry for cambridge igcse®*. Oxford University Press-Children.

Mann, A (2022) *What is quantum mechanics?* https://www.livescience.com/33816-quantum-mechanics-explanation.html Accessed on December 13, 2023.

Morin, D (2008) Introduction to quantum mechanics. *Ch*, *10*, 1–20.

Romforn, L (2023) *What is a quantum mechanical model?* https://academichelp.net/stem/physics/quantum-mechanical-model.html Accessed on December 12, 2023.

Sakurai, J J & Commins, E D (1995) Modern quantum mechanics, revised edition.

Silberberg, M S Amateis, P Venkateswaran, R & Chen, L (2006) *Chemistry: The molecular nature of matter and change* (Vol. 4) McGraw-Hill New York.

Svelto, O Hanna, D C et al. (2010) *Principles of lasers* (Vol. 1) Springer.

APPENDIX

1 CODE LISTINGS

1.1 Python Code

```
import sympy as sp
import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt
from scipy.linalg import eigh

# Define symbolic variables for Delta_probe, Delta_pump, epsilon_probe, epsilon_pump, rho_ij
Delta_probe, Delta_pump, epsilon_probe, epsilon_pump = sp.symbols('Delta_probe Delta_pump
    epsilon_probe epsilon_pump')
rho_11, rho_12, rho_13, rho_21, rho_22, rho_23, rho_31, rho_32, rho_33 = sp.symbols('rho_11
    rho_12 rho_13 rho_21 rho_22 rho_23 rho_31 rho_32 rho_33')

# Defining imaginary number
j = sp.I

# Define specific values
values = {
    Delta_probe: 0,
    Delta_pump: 0,
    epsilon_probe: 0.5,
    epsilon_pump: 0
}

# Defining the Hamiltonian, Density matrix and lindblad matrix.
H = sp.Matrix([[Delta_probe * j, 0, epsilon_probe * j],
               [0, Delta_pump * j, epsilon_pump * j],
               [epsilon_probe * j, epsilon_pump * j, 0]])

rho = sp.Matrix([[rho_11, rho_12, rho_13],
                 [rho_21, rho_22, rho_23],
                 [rho_31, rho_32, rho_33]])

L = sp.Matrix([[rho_33, 0, -1 * rho_13],
               [0, rho_33, -1 * rho_23],
               [-1 * rho_31, -1 * rho_32, -2 * rho_33]])

```

```
H = H.subs(values)
L = L.subs(values)
rho = rho.subs(values)

# Defining the complex differential equation
def combined_differential_equations(current_rho_values, t):
    real_part = current_rho_values[:9].reshape((3, 3)).astype(float)
    imag_part = current_rho_values[9:].reshape((3, 3)).astype(float)

    rho = real_part + 1j * imag_part

    L_values = L.subs({
        rho_11: rho[0, 0], rho_12: rho[0, 1], rho_13: rho[0, 2],
        rho_21: rho[1, 0], rho_22: rho[1, 1], rho_23: rho[1, 2],
        rho_31: rho[2, 0], rho_32: rho[2, 1], rho_33: rho[2, 2]
    })

    dp = np.matmul(rho, H) - np.matmul(H, rho) + L_values
    dp_real = sp.re(sp.expand(dp))
    dp_imag = sp.im(sp.expand(dp))

    dp_real_values = np.array(dp_real).astype(float).flatten()
    dp_imag_values = np.array(dp_imag).astype(float).flatten()

    return np.concatenate((dp_real_values, dp_imag_values)).tolist()


# Initial conditions
t = np.linspace(0, 20, 2000)
y0 = np.array([0, 0, 0, 0, 1, 0, 0, 0, 0] + [0, 0, 0, 0, 0, 0, 0, 0, 0])

# Solving complex differential equation
Y = odeint(combined_differential_equations, y0, t)

# Extracting real and imaginary parts from the results
real_part = Y[:, :9].reshape((len(t), 3, 3))
imag_part = Y[:, 9:].reshape((len(t), 3, 3))

rho_matrices = real_part + 1j * imag_part

rho_11_values_all = rho_matrices[:, 0, 0]
```

```
rho_22_values_all = rho_matrices[:, 1, 1]
rho_33_values_all = rho_matrices[:, 2, 2]


# Plotting, labelling.
plt.figure(figsize=(15, 11))
plt.ylim(-0.01, 1.01)
plt.xlim(0,20)
plt.plot(t, rho_11_values_all, 'b--', label=r'$\rho_{11}$', linewidth=5)
plt.plot(t, rho_22_values_all, 'r:', label=r'$\rho_{22}$', linewidth=5)
plt.plot(t, rho_33_values_all, 'g-.', label=r'$\rho_{33}$', linewidth=5)

plt.minorticks_on()
plt.grid(visible=True, which='major', linestyle='-')
plt.grid(visible=True, which='minor', linestyle='--')

plt.title('Variation of $Population$ Over $Time$ in seconds  \n Initial Conditions:
    $Population$ in State $|2\rangle$; $\Delta_{\mathrm{probe}} = \Delta_{\mathrm{pump}} = \
    epsilon_{\mathrm{pump}} = 0 , \epsilon_{\mathrm{probe}} = 0.5$')
plt.xlabel(r'$Time\;$/ s ')
plt.ylabel(r'$Population$')
plt.legend()
plt.savefig('Graph1.png', dpi=1000)
plt.show()



# Question 3

values = {
    Delta_probe: 0,
    Delta_pump: 0,
    epsilon_probe: 1,
    epsilon_pump: 0
}

H = sp.Matrix([[Delta_probe * j, 0, epsilon_probe * j],
               [0, Delta_pump * j, epsilon_pump * j],
               [epsilon_probe * j, epsilon_pump * j, 0]])

rho = sp.Matrix([[rho_11, rho_12, rho_13],
```

```
                  [rho_21, rho_22, rho_23],
                  [rho_31, rho_32, rho_33]])

L = sp.Matrix([[rho_33, 0, -1 * rho_13],
               [0, rho_33, -1 * rho_23],
               [-1 * rho_31, -1 * rho_32, -2 * rho_33]])

H = H.subs(values)
L = L.subs(values)
rho = rho.subs(values)

def combined_differential_equations(current_rho_values, t):
    real_part = current_rho_values[:9].reshape((3, 3)).astype(float)
    imag_part = current_rho_values[9:].reshape((3, 3)).astype(float)

    rho = real_part + 1j * imag_part

    L_values = L.subs({
        rho_11: rho[0, 0], rho_12: rho[0, 1], rho_13: rho[0, 2],
        rho_21: rho[1, 0], rho_22: rho[1, 1], rho_23: rho[1, 2],
        rho_31: rho[2, 0], rho_32: rho[2, 1], rho_33: rho[2, 2]
    })

    dp = np.matmul(rho, H) - np.matmul(H, rho) + L_values
    dp_real = sp.re(sp.expand(dp))
    dp_imag = sp.im(sp.expand(dp))

    dp_real_values = np.array(dp_real).astype(float).flatten()
    dp_imag_values = np.array(dp_imag).astype(float).flatten()

    return np.concatenate((dp_real_values, dp_imag_values)).tolist()


# Initial conditions
t = np.linspace(0, 20, 2000)
y0 = np.array([1, 0, 0, 0, 0, 0, 0, 0, 0] + [0, 0, 0, 0, 0, 0, 0, 0, 0])
Y = odeint(combined_differential_equations, y0, t)


real_part = Y[:, :9].reshape((len(t), 3, 3))
imag_part = Y[:, 9:].reshape((len(t), 3, 3))
```

```
156
157 rho_matrices = real_part + 1j * imag_part
158
159 rho_11_values_all = rho_matrices[:, 0, 0]
160 rho_22_values_all = rho_matrices[:, 1, 1]
161 rho_33_values_all = rho_matrices[:, 2, 2]
162
163
164 # Plotting
165 plt.figure(figsize=(15, 11))
166 plt.ylim(-0.01, 1.01)
167 plt.xlim(0,20)
168 plt.plot(t, rho_11_values_all, 'b--', label=r'$\rho_{11}$', linewidth=5)
169 plt.plot(t, rho_22_values_all, 'r:', label=r'$\rho_{22}$', linewidth=5)
170 plt.plot(t, rho_33_values_all, 'g-.', label=r'$\rho_{33}$', linewidth=5)
171
172 plt.minorticks_on()
173 plt.grid(visible=True, which='major', linestyle='-')
174 plt.grid(visible=True, which='minor', linestyle='--')
175
176 plt.title('Variation of $Population$ Over $Time$ in seconds \nInitial Conditions: Population
      in State $|1\rangle$; $\Delta_{\mathrm{probe}} = \Delta_{\mathrm{pump}} = \epsilon_{\
    mathrm{pump}} = 0, \epsilon_{\mathrm{probe}} = 1$')
177 plt.xlabel(r'$Time\;$/ s ')
178 plt.ylabel(r'$Population$')
179 plt.legend()
180 plt.savefig('Graph2.png', dpi=1000)
181 plt.show()
182
183
184
185 # Question 4i)
186
187
188
189 H = sp.Matrix([[Delta_probe * j, 0, epsilon_probe * j],
190                [0, Delta_pump * j, epsilon_pump * j],
191                [epsilon_probe * j, epsilon_pump * j, 0]])
192
193 rho = sp.Matrix([[rho_11, rho_12, rho_13],
194                  [rho_21, rho_22, rho_23],
```

```
                  [rho_31, rho_32, rho_33]])

L = sp.Matrix([[rho_33, 0, -1 * rho_13],
               [0, rho_33, -1 * rho_23],
               [-1 * rho_31, -1 * rho_32, -2 * rho_33]])


L = L.subs(values)
rho = rho.subs(values)




def combined_differential_equations(current_rho_values, t):
    real_part = current_rho_values[:9].reshape((3, 3)).astype(float)
    imag_part = current_rho_values[9:].reshape((3, 3)).astype(float)

    rho = real_part + 1j * imag_part

    L_values = L.subs({
        rho_11: rho[0, 0], rho_12: rho[0, 1], rho_13: rho[0, 2],
        rho_21: rho[1, 0], rho_22: rho[1, 1], rho_23: rho[1, 2],
        rho_31: rho[2, 0], rho_32: rho[2, 1], rho_33: rho[2, 2]
    })

    if t > 10:

        epsilon_probe_values = 3
    else:
        epsilon_probe_values = 0

    H_values = H.subs({
        Delta_probe: 0,
        Delta_pump: 0,
        epsilon_probe: epsilon_probe_values,
        epsilon_pump: 10
    })

    dp = np.matmul(rho, H_values) - np.matmul(H_values, rho) + L_values
    dp_real = sp.re(sp.expand(dp))
    dp_imag = sp.im(sp.expand(dp))
```

```
236
237     dp_real_values = np.array(dp_real).astype(float).flatten()
238     dp_imag_values = np.array(dp_imag).astype(float).flatten()
239
240     return np.concatenate((dp_real_values, dp_imag_values)).tolist()
241
242
243 # Initial conditions
244 t = np.linspace(0, 20, 2000)
245 y0 = np.array([1, 0, 0, 0, 0, 0, 0, 0, 0] + [0, 0, 0, 0, 0, 0, 0, 0, 0])
246
247 Y = odeint(combined_differential_equations, y0, t)
248
249
250 real_part = Y[:, :9].reshape((len(t), 3, 3))
251 imag_part = Y[:, 9:].reshape((len(t), 3, 3))
252
253 rho_matrices = real_part + 1j * imag_part
254
255 rho_11_values_all = rho_matrices[:, 0, 0]
256 rho_22_values_all = rho_matrices[:, 1, 1]
257 rho_33_values_all = rho_matrices[:, 2, 2]
258
259 # Plotting
260 plt.figure(figsize=(15, 11))
261 plt.ylim(-0.01, 1.01)
262 plt.xlim(0,20)
263 plt.plot(t, rho_11_values_all, 'b--', label=r'$\rho_{11}$', linewidth=5)
264 plt.plot(t, rho_22_values_all, 'r:', label=r'$\rho_{22}$', linewidth=5)
265 plt.plot(t, rho_33_values_all, 'g-.', label=r'$\rho_{33}$', linewidth=5)
266
267 plt.minorticks_on()
268 plt.grid(visible=True, which='major', linestyle='-')
269 plt.grid(visible=True, which='minor', linestyle='--')
270
271 plt.title('Variation of $Population$ Over $Time$ in seconds\nInitial Conditions: Population
      in State $|1\rangle$; $\Delta_{\mathrm{probe}} = \Delta_{\mathrm{pump}} = 0 ,  \epsilon_
      {\mathrm{pump}} = 10 \ , \epsilon_{\mathrm{probe}} = 3$(switched suddenly at $t=10$)')
272 plt.xlabel(r'$Time\;$/ s ')
273 plt.ylabel(r'$Population$')
274 plt.axvline(x=10, color='grey', linestyle='--', label='$\epsilon_{\mathrm{probe}} = 3$')
```

```
plt.legend()
plt.savefig('Graph3.png', dpi=1000)
plt.show()


# Question 4ii)


H = sp.Matrix([[Delta_probe * j, 0, epsilon_probe * j],
               [0, Delta_pump * j, epsilon_pump * j],
               [epsilon_probe * j, epsilon_pump * j, 0]])

rho = sp.Matrix([[rho_11, rho_12, rho_13],
                 [rho_21, rho_22, rho_23],
                 [rho_31, rho_32, rho_33]])

L = sp.Matrix([[rho_33, 0, -1 * rho_13],
               [0, rho_33, -1 * rho_23],
               [-1 * rho_31, -1 * rho_32, -2 * rho_33]])

L = L.subs(values)
rho = rho.subs(values)

def expo(t, k):
    return np.exp(k*t)-1


def combined_differential_equations(current_rho_values, t):
    real_part = current_rho_values[:9].reshape((3, 3)).astype(float)
    imag_part = current_rho_values[9:].reshape((3, 3)).astype(float)

    rho = real_part + 1j * imag_part

    L_values = L.subs({
        rho_11: rho[0, 0], rho_12: rho[0, 1], rho_13: rho[0, 2],
        rho_21: rho[1, 0], rho_22: rho[1, 1], rho_23: rho[1, 2],
        rho_31: rho[2, 0], rho_32: rho[2, 1], rho_33: rho[2, 2]
    })


    epsilon_probe_values = expo(t, k=0.1)
```

```
316
317     H_values = H.subs({
318         Delta_probe: 0,
319         Delta_pump: 0,
320         epsilon_probe: epsilon_probe_values,
321         epsilon_pump: 10
322     })
323
324     dp = np.matmul(rho, H_values) - np.matmul(H_values, rho) + L_values
325     dp_real = sp.re(sp.expand(dp))
326     dp_imag = sp.im(sp.expand(dp))
327
328     dp_real_values = np.array(dp_real).astype(float).flatten()
329     dp_imag_values = np.array(dp_imag).astype(float).flatten()
330
331     return np.concatenate((dp_real_values, dp_imag_values)).tolist()
332
333
334 # Initial conditions
335 t = np.linspace(0, 20, 2000)
336 y0 = np.array([1, 0, 0, 0, 0, 0, 0, 0, 0] + [0, 0, 0, 0, 0, 0, 0, 0, 0])
337
338 Y = odeint(combined_differential_equations, y0, t)
339
340 real_part = Y[:, :9].reshape((len(t), 3, 3))
341 imag_part = Y[:, 9:].reshape((len(t), 3, 3))
342
343 rho_matrices = real_part + 1j * imag_part
344
345 rho_11_values_all = rho_matrices[:, 0, 0]
346 rho_22_values_all = rho_matrices[:, 1, 1]
347 rho_33_values_all = rho_matrices[:, 2, 2]
348
349 # Plotting
350 plt.figure(figsize=(15, 11))
351 plt.ylim(-0.01, 1.01)
352 plt.xlim(0,20)
353 plt.plot(t, rho_11_values_all, 'b--', label=r'$\rho_{11}$', linewidth=5)
354 plt.plot(t, rho_22_values_all, 'r:', label=r'$\rho_{22}$', linewidth=5)
355 plt.plot(t, rho_33_values_all, 'g-.', label=r'$\rho_{33}$', linewidth=5)
356
```

```
plt.minorticks_on()
plt.grid(visible=True, which='major', linestyle='-')
plt.grid(visible=True, which='minor', linestyle='--')

plt.title('Variation of $Population$ Over $Time$ in seconds \nInitial Conditions: Population in State $|1\rangle$; $\Delta_{\mathrm{probe}} = \Delta_{\mathrm{pump}} = 0 , \epsilon_{\mathrm{pump}} = 10 $, $\epsilon_{\mathrm{probe}}$ (is increasing slowly),')
plt.xlabel(r'$Time\;$/ s ')
plt.ylabel(r'$Population$')

plt.axvline(x=6.9, color='grey', linestyle='--', label=r'$\epsilon_{\mathrm{probe}}\approx 1$')
plt.legend()
plt.xticks(ticks=[*plt.xticks()[0], 6.9])
plt.savefig('Graph4.png', dpi=1000)
plt.show()


# Creating Gifs for question 4i and 4ii

import sympy as sp
import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt
from scipy.linalg import eigh
import imageio

# Define symbolic variables for Delta_probe, Delta_pump, epsilon_probe, epsilon_pump, rho_ij
Delta_probe, Delta_pump, epsilon_probe, epsilon_pump = sp.symbols('Delta_probe Delta_pump epsilon_probe epsilon_pump')
rho_11, rho_12, rho_13, rho_21, rho_22, rho_23, rho_31, rho_32, rho_33 = sp.symbols('rho_11 rho_12 rho_13 rho_21 rho_22 rho_23 rho_31 rho_32 rho_33')

# Defining imaginary number
j = sp.I

# Define specific values
values = {
    Delta_probe: 0,
    Delta_pump: 0,
}
```

```
393
394 H = sp.Matrix([[Delta_probe * j, 0, epsilon_probe * j],
395                [0, Delta_pump * j, epsilon_pump * j],
396                [epsilon_probe * j, epsilon_pump * j, 0]])
397
398 rho = sp.Matrix([[rho_11, rho_12, rho_13],
399                  [rho_21, rho_22, rho_23],
400                  [rho_31, rho_32, rho_33]])
401
402 L = sp.Matrix([[rho_33, 0, -1 * rho_13],
403                [0, rho_33, -1 * rho_23],
404                [-1 * rho_31, -1 * rho_32, -2 * rho_33]])
405
406
407 L = L.subs(values)
408 rho = rho.subs(values)
409
410
411 # Generating plots for various epsilon_pump values
412 epsilon_pump_values = [10, 20, 30, 40, 50]  # Replace with desired epsilon_pump values
413 images = []
414
415 for epsilon_pump_value in epsilon_pump_values:
416     def combined_differential_equations(current_rho_values, t):
417         real_part = current_rho_values[:9].reshape((3, 3)).astype(float)
418         imag_part = current_rho_values[9:].reshape((3, 3)).astype(float)
419
420         rho = real_part + 1j * imag_part
421
422         L_values = L.subs({
423             rho_11: rho[0, 0], rho_12: rho[0, 1], rho_13: rho[0, 2],
424             rho_21: rho[1, 0], rho_22: rho[1, 1], rho_23: rho[1, 2],
425             rho_31: rho[2, 0], rho_32: rho[2, 1], rho_33: rho[2, 2]
426         })
427
428         if t > 10:
429
430             epsilon_probe_values = 3
431         else:
432             epsilon_probe_values = 0
433
```

```
        H_values = H.subs({
            Delta_probe: 0,
            Delta_pump: 0,
            epsilon_probe: epsilon_probe_values,
            epsilon_pump: epsilon_pump_values
        })

        dp = np.matmul(rho, H_values) - np.matmul(H_values, rho) + L_values
        dp_real = sp.re(sp.expand(dp))
        dp_imag = sp.im(sp.expand(dp))

        # Evaluate symbolic expressions with numerical values
        dp_real_values = np.array(dp_real.subs({Delta_probe: 0, Delta_pump: 0, epsilon_probe
    : 3, epsilon_pump: epsilon_pump_value})).astype(float).flatten()
        dp_imag_values = np.array(dp_imag.subs({Delta_probe: 0, Delta_pump: 0, epsilon_probe
    : 3, epsilon_pump: epsilon_pump_value})).astype(float).flatten()

        return np.concatenate((dp_real_values, dp_imag_values)).tolist()


    # Initial conditions
    t = np.linspace(0, 20, 2000)
    y0 = np.array([1, 0, 0, 0, 0, 0, 0, 0, 0] + [0, 0, 0, 0, 0, 0, 0, 0, 0])

    Y = odeint(combined_differential_equations, y0, t)


    real_part = Y[:, :9].reshape((len(t), 3, 3))
    imag_part = Y[:, 9:].reshape((len(t), 3, 3))

    rho_matrices = real_part + 1j * imag_part

    rho_11_values_all = rho_matrices[:, 0, 0]
    rho_22_values_all = rho_matrices[:, 1, 1]
    rho_33_values_all = rho_matrices[:, 2, 2]

    # Plotting
    plt.figure(figsize=(15, 11))
    plt.ylim(-0.01, 1.01)
    plt.xlim(0,20)
    plt.plot(t, rho_11_values_all, 'b--', label=r'$\rho_{11}$', linewidth=5)
```

```
    plt.plot(t, rho_22_values_all, 'r:', label=r'$\rho_{22}$', linewidth=5)
    plt.plot(t, rho_33_values_all, 'g-.', label=r'$\rho_{33}$', linewidth=5)

    plt.minorticks_on()
    plt.grid(visible=True, which='major', linestyle='-')
    plt.grid(visible=True, which='minor', linestyle='--')

    plt.title(f'Variation of $Population$ Over $Time$ in seconds\nInitial Conditions:
    Population in State $|1\rangle$; $\Delta_{{\mathrm{{probe}}}} = \Delta_{{\mathrm{{pump
    }}}} = 0 ,  \epsilon_{{\mathrm{{pump}}}} = {epsilon_pump_value}, \epsilon_{{\mathrm{{
    probe}}}} = 3$(switched suddenly at $t=10$)')

    plt.ylabel(r'$Population$')
    plt.axvline(x=10, color='grey', linestyle='--', label='$\epsilon_{\mathrm{probe}} = 3$')
    plt.legend()
    plt.savefig(f'Graph 3_epsilon_pump_{epsilon_pump_value}.png', dpi = 1000)
    plt.close()
    images.append(imageio.imread(f'Graph 3_epsilon_pump_{epsilon_pump_value}.png'))

# Saving the images as a GIF
imageio.mimsave('graph_3_epsilon_pump_variation.gif', images, duration=3)

# Question 4ii GIF
# Define symbolic variables for Delta_probe, Delta_pump, epsilon_probe, epsilon_pump, rho_ij
Delta_probe, Delta_pump, epsilon_probe, epsilon_pump = sp.symbols('Delta_probe Delta_pump
    epsilon_probe epsilon_pump')
rho_11, rho_12, rho_13, rho_21, rho_22, rho_23, rho_31, rho_32, rho_33 = sp.symbols('rho_11
    rho_12 rho_13 rho_21 rho_22 rho_23 rho_31 rho_32 rho_33')

# Defining imaginary number
j = sp.I

# Define specific values
values = {
    Delta_probe: 0,
    Delta_pump: 0,
}

H = sp.Matrix([[Delta_probe * j, 0, epsilon_probe * j],
               [0, Delta_pump * j, epsilon_pump * j],
               [epsilon_probe * j, epsilon_pump * j, 0]])
```

```

rho = sp.Matrix([[rho_11, rho_12, rho_13],
                 [rho_21, rho_22, rho_23],
                 [rho_31, rho_32, rho_33]])

L = sp.Matrix([[rho_33, 0, -1 * rho_13],
               [0, rho_33, -1 * rho_23],
               [-1 * rho_31, -1 * rho_32, -2 * rho_33]])


L = L.subs(values)
rho = rho.subs(values)


# Generating plots for various pump field values
epsilon_pump_values = [10, 20, 30, 40, 50]  # Replace with desired epsilon_pump values
images = []

def expo(t, k):
    return np.exp(k*t)-1

for epsilon_pump_value in epsilon_pump_values:
    def combined_differential_equations(current_rho_values, t):
        real_part = current_rho_values[:9].reshape((3, 3)).astype(float)
        imag_part = current_rho_values[9:].reshape((3, 3)).astype(float)

        rho = real_part + 1j * imag_part

        L_values = L.subs({
            rho_11: rho[0, 0], rho_12: rho[0, 1], rho_13: rho[0, 2],
            rho_21: rho[1, 0], rho_22: rho[1, 1], rho_23: rho[1, 2],
            rho_31: rho[2, 0], rho_32: rho[2, 1], rho_33: rho[2, 2]
        })

        epsilon_probe_values = expo(t, k=0.1)

        H_values = H.subs({
            Delta_probe: 0,
            Delta_pump: 0,
            epsilon_probe: epsilon_probe_values,
            epsilon_pump: epsilon_pump_values
```

```
        })

        dp = np.matmul(rho, H_values) - np.matmul(H_values, rho) + L_values
        dp_real = sp.re(sp.expand(dp))
        dp_imag = sp.im(sp.expand(dp))

        # Evaluate symbolic expressions with numerical values
        dp_real_values = np.array(dp_real.subs({Delta_probe: 0, Delta_pump: 0, epsilon_probe
    : 3, epsilon_pump: epsilon_pump_value})).astype(float).flatten()
        dp_imag_values = np.array(dp_imag.subs({Delta_probe: 0, Delta_pump: 0, epsilon_probe
    : 3, epsilon_pump: epsilon_pump_value})).astype(float).flatten()

        return np.concatenate((dp_real_values, dp_imag_values)).tolist()


# Initial conditions
t = np.linspace(0, 20, 2000)
y0 = np.array([1, 0, 0, 0, 0, 0, 0, 0, 0] + [0, 0, 0, 0, 0, 0, 0, 0, 0])

Y = odeint(combined_differential_equations, y0, t)

real_part = Y[:, :9].reshape((len(t), 3, 3))
imag_part = Y[:, 9:].reshape((len(t), 3, 3))

rho_matrices = real_part + 1j * imag_part

rho_11_values_all = rho_matrices[:, 0, 0]
rho_22_values_all = rho_matrices[:, 1, 1]
rho_33_values_all = rho_matrices[:, 2, 2]

# Plotting
plt.figure(figsize=(15, 11))
plt.ylim(-0.01, 1.01)
plt.xlim(0,20)
plt.plot(t, rho_11_values_all, 'b--', label=r'$\rho_{11}$', linewidth=5)
plt.plot(t, rho_22_values_all, 'r:', label=r'$\rho_{22}$', linewidth=5)
plt.plot(t, rho_33_values_all, 'g-.', label=r'$\rho_{33}$', linewidth=5)

plt.minorticks_on()
plt.grid(visible=True, which='major', linestyle='-')
plt.grid(visible=True, which='minor', linestyle='--')
```

```
589
590     plt.title(f'Variation of $Population$ Over $Time$ in seconds\nInitial Conditions:
        Population in State $|1\rangle$; $\Delta_{{\mathrm{{probe}}}} = \Delta_{{\mathrm{{pump
        }}}} = 0 ,  \epsilon_{{\mathrm{{pump}}}} = {epsilon_pump_value}, \epsilon_{{\mathrm{{
        probe}}}} $ is increasing slowly')
591
592     plt.xlabel(r'$Time\;$/ s ')
593     plt.ylabel(r'$Population$')
594
595     plt.axvline(x=6.9, color='grey', linestyle='--', label=r'$\epsilon_{\mathrm{probe}}\
        approx 1$')
596     plt.legend()
597     plt.xticks(ticks=[*plt.xticks()[0], 6.9])
598     plt.savefig(f'Graph 4_epsilon_pump_{epsilon_pump_value}.png', dpi = 1000)
599     plt.close()
600     images.append(imageio.imread(f'Graph 4_epsilon_pump_{epsilon_pump_value}.png'))
601
602 #Saving the images as a GIF
603 imageio.mimsave('graph4_epsilon_pump_variation.gif', images, duration=3)
```

Listing 1. Python code used for Task 1, 2, 3.

1.2 Mathematica computational code

```
1  ket1 = {{1}, {0}, {0}};
2  bra1 = {{1, 0, 0}};
3
4  ket2 = {{0}, {1}, {0}};
5  bra2 = {{0, 1, 0}};
6
7  ket3 = {{0}, {0}, {1}};
8  bra3 = {{0, 0, 1}};
9
10 (*Compute Hamiltonian, H, by first principles. Dot[] is used for \
11 general matrix multiplication*)
12 H = h*(Delta_pump* Dot[ket1, bra1] + Delta_pump *Dot[ket2, bra2] +
13      epsilon_pump *(Dot[ket2, bra3] + Dot[ket3, bra2]) +
14      epsilon_probe* (Dot[ket1, bra3] + Dot[ket3, bra1]));
15 (*Define a general density matrix*)
16 p = {{rho11, rho12, rho13}, {rho21, rho22, rho23}, {rho31, rho32,
17     rho33}};
18
```

```
(*To simplify the process, first calculate the commutator and L[p]*)
Commutator = Dot[p, H] - Dot[H, p];
L = Dot[ket1, Dot[bra3, Dot[p, Dot[ket3, bra1]]]] +
   Dot[ket2, Dot[bra3, Dot[p, Dot[ket3, bra2]]]] -
   Dot[ket3, Dot[bra3, p]] - Dot[p, Dot[ket3, bra3]];

(*rho-dot is the derivative of the density matrix, the imaginary \
number,i, can be entered using (ESC), then (i), and then (ESC) again*)

RhoDot = Simplify[((i/h)*Commutator) + L];
MatrixForm[RhoDot]
```

Listing 2. Mathematica code to recheck the Master equation calculations

2 APPENDIX B

Throughout listing 1, we imported the packages that were used in this experiment. The Hamiltonian, the density matrix, and the Lindblad operators were defined and used to solve the master differential equation. Before solving, however, the real and imaginary parts were separated and solved separately but simultaneously. After solving, the diagonal elements of the density matrix depicted how the population was changing throughout states 1, 2, and 3. The population was plotted against the evolution of time. The same code was repeated a couple of times with different pump and probe field strengths. Then the process was repeating over increasing values of the pump field creating ”snapshots”.

Additionally listing 2 shows the calculation performed on the Mathematica field to calculate the Hamiltonian, and the Master equation. It was used as a second reference, to be sure on the calculations.