Equivalent Circuits

Lee Farrugia

10^{th} November 2022

Abstract

1 Introduction & Theoretical Background

This experiment was conducted in order to compute and understand what is referred to as a Lambda system and the electromagnetically-induced transparency (EIT). The Lambda system is a system in Atomic Physics when an atom/s inside a gas can transition only between specific levels which from the greek letter Λ. The transitions allowed are between $1 \leftrightarrow 3$ and $2 \leftrightarrow 3$, while eliminating the $1 \leftrightarrow 2$ transition by making stop absorbing light from a probe field. This results that many of the atoms in the gas become transparent hence EIT . These transitions can be seen in figure 1.

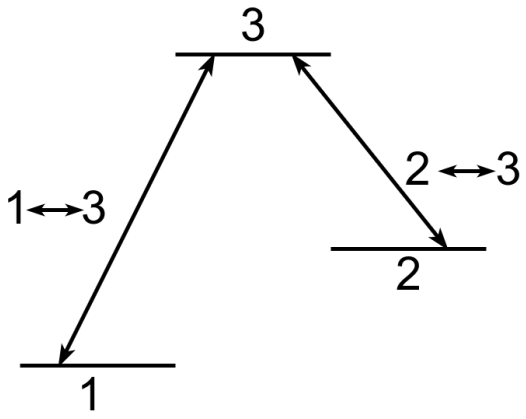

Figure 1: Diagram of the transitions in the lambda system

2 Materials & Methods

2.1 Language and Packages

Python 3.10.6, Numpy, Scipy, Matplotlib.pyplot .

2.2 Methodology

1. In order to simulate the system and compute it, the Hamiltonian governing the system is considered, which can be written as:

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

where

$$\hat{H}_0 = \hbar\omega_1 \left|1\right\rangle \left\langle 1\right| + \hbar\omega_2 \left|2\right\rangle \left\langle 2\right| + \hbar\omega_3 \left|3\right\rangle \left\langle 3\right|\,, \tag{2}$$

$$\hat{H}_{pump} = \hbar\varepsilon_{pump}(e^{-i\omega_{pump}t} \left|2\right\rangle \left\langle 3\right| + e^{i\omega_{pump}t} \left|3\right\rangle \left\langle 2\right|)\,, \tag{3}$$

$$\hat{H}_{probe} = \hbar\varepsilon(e^{-i\omega_{probe}t} \left|1\right\rangle \left\langle 3\right| + e^{i\omega_{probe}t} \left|3\right\rangle \left\langle 1\right|)\,. \tag{4}$$

By setting $\Omega_1 = \omega_1 - \Delta_{probe}$, $\Omega_2 = \omega_3 - \Delta_{pump}$ and $\Omega_3 = \omega_3$ the following transformation are to be done on the Hamiltonian,

$$\left|j\right\rangle \rightarrow e^{-i\Omega_j t} \left|j\right\rangle, j(= 1, 2, 3) \tag{5}$$

$$\hat{H} \rightarrow \hat{H} - \sum_j \hbar\Omega_j \left|j\right\rangle \left\langle j\right|\,. \tag{6}$$

These transformations will result in a three dimensional matrix without time dependence.

2. The density matrix that is obtained from the hamiltonian is also a three dimensional matrix.

3. The Hamiltonian and Density matrices are coded and derived using python 3.10.6 and numpy to confirm that the matrices obtained by derivation on pen and paper will match. The differential equation that governs the density matrix of the system is as follows:

$$\dot{\rho} = \frac{i}{\hbar}\left[\rho, \hat{H}\right] + \mathcal{L}[\rho]\,, \tag{7}$$

where $\mathcal{L}[\rho] = \gamma \sum_{j=1,2} \Big[\left|j\right\rangle \left\langle 3|\rho|3\right\rangle \left\langle j\right| - \frac{1}{2}(\left|3\right\rangle \left\langle 3|\rho + \rho|3\right\rangle \left\langle 3|)\Big]$, where γ is the decay rate of level $|3\rangle$. The differential equation for each position in the density matrix can be determined and solved numerically using scipy.integrate.odeint.

4. The behavior of the system with different initial conditions, the atom population in different levels, and the ε_{pump} increasing while the ε_{probe} is either suddenly turning on or slowing turning on. In all cases $\varepsilon_{probe} \ll \varepsilon_{pump}$.

3 Results & Discussion

4 Conclusion

5 References

6 Appendix

```
import matplotlib.pyplot as plt
import numpy as np
from cmath import *
from math import *
from scipy import *
from sympy import *
from scipy.integrate import odeint
from sympy.physics.quantum import Bra, Ket

##Obtaining the Hamiltonian matrix and the density matrix of
   the three-dimensional lambda system, symbolically using the
   sympy package

#Defining the symbols to be used
Omega_1,Omega_2,Omega_3,omega_1,omega_2,omega_3,delta_probe,delta_pump,omeg
   =
   symbols('Omega_1,Omega_2,Omega_3,omega_1,omega_2,omega_3,delta_probe,del

#Defining the terms to be used
Omega_1 = omega_1 - delta_probe   #Rabi frequency of level 1
Omega_2 = omega_2 - delta_pump    #Rabi frequency of level 2
Omega_3 = omega_3                 #Rabi frequency of level 3

omega_probe = omega_13 + delta_probe  #frequency of probe field
omega_pump = omega_23 + delta_pump    #frequency of pump field

H_probe = (h * epsilon_probe * (Ket('1') * Bra('3'))) + (h *
   epsilon_probe * (Ket('3') * Bra('1')))   #Hamiltonian of the
   system with probe field
H_pump = (h * epsilon_pump * (Ket('2') * Bra('3'))) + (h *
   epsilon_pump * (Ket('3') * Bra('2')))      #Hamiltonian of
   the system with pump field
H_0 = (h * (Omega_1 + delta_probe)) + (h * (Omega_2 +
   delta_pump)) + (h * Omega_3)                    #Hamiltonian
   of the system without external fields

#Defining the Hamiltonian of the lambda system
H = H_0 + H_pump + H_probe
print(f'Hamiltonian: {H}')

#Defining each element of the Hamiltonian matrix
H_11 = h * delta_probe
H_12 = 0
H_13 = h * epsilon_probe * (Ket('1') * Bra('3'))
H_21 = 0
H_22 = h * delta_pump
H_23 = h * epsilon_pump * (Ket('2') * Bra('3'))
H_31 = h * epsilon_probe * (Ket('3') * Bra('1'))
```

```
H_32 = h * epsilon_pump * (Ket('3') * Bra('2'))
H_33 = 0

#Defining the Hamiltonian matrix
H_matrix = Matrix([[H_11, H_12, H_13], [H_21, H_22, H_23],
   [H_31, H_32, H_33]])
print('\n')
print(f'Hamiltonian matrix: {H_matrix}')

#Defining each element of the density matrix
rho_11 = 1
rho_12 = 1j * ((epsilon_probe * (Ket('1') * Bra('3'))) / (2 *
   h))
rho_13 = - gamma_13 - (1j * delta_probe) - (1j *
   ((epsilon_probe * (Ket('1') * Bra('3'))) / (2 * h)))
rho_21 = - (1j) * ((epsilon_probe * (Ket('1') * Bra('3'))) / (2
   * h))
rho_22 = 0
rho_23 = - gamma_23 - (1j * delta_pump)
rho_31 = - gamma_13 + (1j * delta_probe) + (1j *
   ((epsilon_probe * (Ket('1') * Bra('3'))) / (2 * h)))
rho_32 = - gamma_23 + (1j * delta_pump)
rho_33 = 0

#Defining the density matrix
density_matrix = Matrix([[rho_11, rho_12, rho_13], [rho_21,
   rho_22, rho_23], [rho_31, rho_32, rho_33]])
print('\n')
print(f'Density matrix: {density_matrix}')

##Obtaining the Hamiltonian matrix and the density matrix of
   the three-dimensional lambda system, numerically using the
   numpy package

#Defining ket mathematically
def ket(a,b,c):
  return np.array([[a],[b],[c]])

#Defining bra mathematically
def bra(a,b,c=None):
  if c==None:
    return (a.conjugate()).transpose()
  else:
    return np.array([[a,b,c]])

#Defining a constant to be used
gamma = 1   #decay rate level of level 3
```

```
def Hamiltonian_matrix(epsilon_probe, epsilon_pump):
  #Defining constants to be used
  h = 1.054571817e-34   #reduced Planck's constant
  delta_probe = 0
  delta_pump = 0

  #Working out each element of the Hamiltonian matrix
  H_11 = np.multiply(h, delta_probe)
  H_12 = 0
  H_13 = np.multiply(np.multiply(h, epsilon_probe),
   float(np.dot(bra(0,0,3), ket(0,0,1))))
  H_21 = 0
  H_22 = np.multiply(h, delta_pump)
  H_23 = np.multiply(np.multiply(h, epsilon_pump),
   float(np.dot(bra(0,0,3), ket(0,0,2))))
  H_31 = np.multiply(np.multiply(h, epsilon_probe),
   float(np.dot(bra(0,0,1), ket(0,0,3))))
  H_32 = np.multiply(np.multiply(h, epsilon_pump),
   float(np.dot(bra(0,0,2), ket(0,0,3))))
  H_33 = 0

  #Defining the Hamiltonian matrix
  H_mat = np.array([[H_11, H_12, H_13], [H_21, H_22, H_23],
   [H_31, H_32, H_33]])
  return H_mat

def Density_matrix(epsilon_probe):
  #Defining constants to be used
  h_2 = 2.109143634e-34   #twice the reduced Planck's constant
  delta_probe = 0
  delta_pump = 0
  gamma_13 = 1
  gamma_23 = 1

  #Working out each element of the density matrix
  bra_ket = np.dot(bra(0,0,3), ket(0,0,1))
  bra_ket_2 = np.dot(bra(0,0,3), ket(0,0,2))
  ep_pr_bra_ket = np.multiply(epsilon_probe, bra_ket)
  ep_pp_bra_ket = np.multiply(epsilon_pump, bra_ket_2)
  ep_pr_bra_ket_h_2 = ep_pr_bra_ket / h_2
  ep_pp_bra_ket_h_2 = ep_pp_bra_ket / h_2

  rho_11 = 1
  rho_12 = complex(0, ep_pr_bra_ket_h_2) - complex(0,
   ep_pp_bra_ket_h_2)
  rho_13 = - gamma_13 - complex(0, delta_probe) - complex(0,
   ep_pr_bra_ket_h_2)
  rho_21 = - complex(0, ep_pr_bra_ket_h_2) + complex(0 ,
```

```
    ep_pp_bra_ket_h_2)
  rho_22 = 0
  rho_23 = - gamma_23 - complex(0, delta_pump)
  rho_31 = - gamma_13 + complex(0, delta_probe) + complex(0,
    ep_pr_bra_ket_h_2)
  rho_32 = - gamma_23 + complex(0, delta_pump)
  rho_33 = 0

  #Defining the density matrix
  density_mat = np.array([[rho_11, rho_12, rho_13], [rho_21,
    rho_22, rho_23], [rho_31, rho_32, rho_33]])
  return density_mat

def commutator(density_mat, H_mat):
  #Defining a constant to be used
  gamma = 1   #decay rate level of level 3

  #Working out the commutator of the Hamiltonian matrix and the
    density matrix
  com_rho_H = (np.matmul(density_mat, H_mat)) -
    (np.matmul(H_mat, density_mat))

  #Working out the first term of the differential equation that
    governs the density matrix of the lambda system, rho_dot
  y = (1/h) * com_rho_H
  return y

def L(j, density_mat):
  #Defining the second term of the differential equation that
    governs the density matrix of the lambda system, rho_dot
  if j>0:
    z = gamma * (np.dot(bra(0,0,3), ket(0,0,j)) * density_mat *
    np.dot(bra(0,0,j), ket(0,0,3))) - (0.5) *
    (np.dot(bra(0,0,3), ket(0,0,3)) * (density_mat +
    density_mat) * np.dot(bra(0,0,3), ket(0,0,3)))
  return z

def real_imaginary(y, z):
  #Obtaining each element of the matrix defining the first term
    of rho_dot
  y_11 = np.array(y[0,0])
  y_12 = np.array(y[0,1])
  y_13 = np.array(y[0,2])
  y_21 = np.array(y[1,0])
  y_22 = np.array(y[1,1])
  y_23 = np.array(y[1,2])
  y_31 = np.array(y[2,0])
  y_32 = np.array(y[2,1])
```

```
y_33 = np.array(y[2,2])

#Obtaining the real and imaginary part of each element of the
 matrix defining the first term of rho_dot
real_y_11 = y_11.real
imag_y_11 = y_11.imag
real_y_12 = y_12.real
imag_y_12 = y_12.imag
real_y_13 = y_13.real
imag_y_13 = y_13.imag
real_y_21 = y_21.real
imag_y_21 = y_21.imag
real_y_22 = y_22.real
imag_y_22 = y_22.imag
real_y_23 = y_23.real
imag_y_23 = y_23.imag
real_y_31 = y_31.real
imag_y_31 = y_31.imag
real_y_32 = y_32.real
imag_y_32 = y_32.imag
real_y_33 = y_33.real
imag_y_33 = y_33.imag

#Obtaining each element of the matrix defining the second
 term of rho_dot
z_11 = np.array(z[0,0])
z_12 = np.array(z[0,1])
z_13 = np.array(z[0,2])
z_21 = np.array(z[1,0])
z_22 = np.array(z[1,1])
z_23 = np.array(z[1,2])
z_31 = np.array(z[2,0])
z_32 = np.array(z[2,1])
z_33 = np.array(z[2,2])

#Obtaining the real and imaginary part of each element of the
 matrix defining the second term of rho_dot
real_z_11 = z_11.real
imag_z_11 = z_11.imag
real_z_12 = z_12.real
imag_z_12 = z_12.imag
real_z_13 = z_13.real
imag_z_13 = z_13.imag
real_z_21 = z_21.real
imag_z_21 = z_21.imag
real_z_22 = z_22.real
imag_z_22 = z_22.imag
real_z_23 = z_23.real
```

```
    imag_z_23 = z_23.imag
    real_z_31 = z_31.real
    imag_z_31 = z_31.imag
    real_z_32 = z_32.real
    imag_z_32 = z_32.imag
    real_z_33 = z_33.real
    imag_z_33 = z_33.imag

    #Working out the real part of each element of the matrix
     defining rho_dot
    real_x_11 = - imag_y_11 + real_z_11
    real_x_12 = - imag_y_12 + real_z_12
    real_x_13 = - imag_y_13 + real_z_13
    real_x_21 = - imag_y_21 + real_z_21
    real_x_22 = - imag_y_22 + real_z_22
    real_x_23 = - imag_y_23 + real_z_23
    real_x_31 = - imag_y_31 + real_z_31
    real_x_32 = - imag_y_32 + real_z_32
    real_x_33 = - imag_y_33 + real_z_33

    #Working out the imaginary part of each element of the matrix
     defining rho_dot
    imag_x_11 = real_y_11 + imag_z_11
    imag_x_12 = real_y_12 + imag_z_12
    imag_x_13 = real_y_13 + imag_z_13
    imag_x_21 = real_y_21 + imag_z_21
    imag_x_22 = real_y_22 + imag_z_22
    imag_x_23 = real_y_23 + imag_z_23
    imag_x_31 = real_y_31 + imag_z_31
    imag_x_32 = real_y_32 + imag_z_32
    imag_x_33 = real_y_33 + imag_z_33

    #Defining the matrix of the real parts of rho_dot
    real_x_mat = np.array([[real_x_11, real_x_12, real_x_13],
     [real_x_21, real_x_22, real_x_23], [real_x_31, real_x_32,
     real_x_33]])

    #Defining the matrix of the imaginary parts of rho_dot
    imag_x_mat = np.array([[imag_x_11, imag_x_12, imag_x_13],
     [imag_x_21, imag_x_22, imag_x_23], [imag_x_31, imag_x_32,
     imag_x_33]])
    return real_x_mat, imag_x_mat

    def diff_eq_func(real_x_mat, imag_x_mat):
    #Defining parameters to be used
    max_t = 3
    d = [0, 1, 2]
```

```
  diff_eq = []

  #Obtaining the 9 differential equations corresponding to the
   9 elements of the matrix defining rho_dot
  for i in range(max_t):
    for c in d:
      rho_real = real_x_mat[i, c]
      rho_imag = imag_x_mat[i, c]
      rho = (rho_real + rho_imag)
      diff_eq.append(rho)
  return diff_eq

  def odeint_func(diff_eq ,t):
  #Defining a parameter to be used
  b = (7e6, 0, 0, 0, 0, 0, 0, 1e3, 0)

  #Working out the obtained 9 differential equations using the
   odeint function
  a = odeint(drho_dt, b, t)
  return a

#Defining the function to be used by odeint to numerically
   solve the obtained 9 differential equations
def drho_dt(a, t):
  drho_dt_1 = diff_eq[0]
  drho_dt_2 = diff_eq[1]
  drho_dt_3 = diff_eq[2]
  drho_dt_4 = diff_eq[3]
  drho_dt_5 = diff_eq[4]
  drho_dt_6 = diff_eq[5]
  drho_dt_7 = diff_eq[6]
  drho_dt_8 = diff_eq[7]
  drho_dt_9 = diff_eq[8]
  return [drho_dt_1, drho_dt_2, drho_dt_3, drho_dt_4,
   drho_dt_5, drho_dt_6, drho_dt_7, drho_dt_8, drho_dt_9]

#Setting a constant to be used
h = 1.054571817e-34   #reduced Planck's constant

#Setting the initial conditions of the lambda system
epsilon_probe = 1             #quantification of the strength
   of the probe field
epsilon_pump = 0              #quantification of the strength
   of the pump field
j = 2                         #level in which the atom
   population lies
t = np.arange(0, 10000, 0.1)  #time
```

```
#Calling each previously defined function to obtain the
   solution of the 9 differential equations corresponding to
   the set initial conditions
den_mat = Density_matrix(epsilon_probe)
ham_mat = Hamiltonian_matrix(epsilon_probe, epsilon_pump)
real_x_mat = real_imaginary(commutator(den_mat, ham_mat), L(j,
   den_mat))[0]
imag_x_mat = real_imaginary(commutator(den_mat, ham_mat), L(j,
   den_mat))[1]
diff_eq = diff_eq_func(real_x_mat, imag_x_mat)
p = odeint_func(diff_eq_func(real_x_mat, imag_x_mat),t)
   #solution of the 9 differential equations corresponding to
   the set initial conditions, rho

print(f'Epsilon_probe: {epsilon_probe}')
print(f'Epsilon_pump: {epsilon_pump}')
print(f'Level, j: {j}')

#Plotting a graph of the change in rho in time
plt.figure(figsize=(7.5, 10.5))
plt.rcParams['font.family'] = 'STIXGeneral'
plt.rcParams['mathtext.fontset'] = 'stix'
plt.rcParams['font.size'] = 12
plt.rcParams['font.weight'] = 'normal'
plt.minorticks_on()
plt.grid(visible=True, which='major', linestyle='-')
plt.grid(visible=True, which='minor', linestyle='--')
plt.plot(t, p)
plt.xlabel('t / s')
plt.ylabel(r'$\rho$')
plt.xlim(0,)
plt.title(r'A graph of the change in $\mathrm{\rho}$ in time')
plt.legend(['11','12','13','21','22','23','31','32','33'])
plt.tight_layout()
plt.savefig(f'plots/Plot 1.png', dpi=800)
plt.show()


#Setting the initial conditions of the lambda system
epsilon_probe = 1    #quantification of the strength of the
   probe field
epsilon_pump = 0     #quantification of the strength of the pump
   field
j = 1                #level in which the atom population lies

#Calling each previously defined function to obtain the
   solution of the 9 differential equations corresponding to
   the set initial conditions
```

```
den_mat = Density_matrix(epsilon_probe)
ham_mat = Hamiltonian_matrix(epsilon_probe, epsilon_pump)
real_x_mat = real_imaginary(commutator(den_mat, ham_mat), L(j,
    den_mat))[0]
imag_x_mat = real_imaginary(commutator(den_mat, ham_mat), L(j,
    den_mat))[1]
diff_eq = diff_eq_func(real_x_mat, imag_x_mat)
p = odeint_func(diff_eq_func(real_x_mat, imag_x_mat),t)
    #solution of the 9 differential equations corresponding to
    the set initial conditions, rho

print(f'Epsilon_probe: {epsilon_probe}')
print(f'Epsilon_pump: {epsilon_pump}')
print(f'Level, j: {j}')

#Plotting a graph of the change in rho in time
plt.figure(figsize=(7.5, 10.5))
plt.rcParams['font.family'] = 'STIXGeneral'
plt.rcParams['mathtext.fontset'] = 'stix'
plt.rcParams['font.size'] = 12
plt.rcParams['font.weight'] = 'normal'
plt.minorticks_on()
plt.grid(visible=True, which='major', linestyle='-')
plt.grid(visible=True, which='minor', linestyle='--')
plt.plot(t, p)
plt.xlabel('t / s')
plt.ylabel(r'$\rho$')
plt.xlim(0,)
plt.title(r'A graph of the change in $\mathrm{\rho}$ in time')
plt.legend(['11','12','13','21','22','23','31','32','33'])
plt.tight_layout()
plt.savefig(f'plots/Plot 2.png', dpi=800)
plt.show()

##Question 4a

#Setting the initial conditions of the lambda system
epsilon_probe = 0    #quantification of the strength of the
    probe field
epsilon_pump = 1     #quantification of the strength of the pump
    field
j = 1                #level in which the atom population lies

for i in t:
  print(f'Epsilon_probe: {epsilon_probe}')
  print(f'Epsilon_pump: {epsilon_pump}')
  print(f'Level, j: {j}')
```

```
  #Calling each previously defined function to obtain the
   solution of the 9 differential equations corresponding to
   the set initial conditions
  den_mat = Density_matrix(epsilon_probe)
  ham_mat = Hamiltonian_matrix(epsilon_probe, epsilon_pump)
  real_x_mat = real_imaginary(commutator(den_mat, ham_mat),
   L(j, den_mat))[0]
  imag_x_mat = real_imaginary(commutator(den_mat, ham_mat),
   L(j, den_mat))[1]
  diff_eq = diff_eq_func(real_x_mat, imag_x_mat)
  p = odeint_func(diff_eq_func(real_x_mat, imag_x_mat),t)
   #solution of the 9 differential equations corresponding to
   the set initial conditions, rho

  #Plotting a graph of the change in rho in time
  plt.figure(figsize=(7.5, 10.5))
  plt.rcParams['font.family'] = 'STIXGeneral'
  plt.rcParams['mathtext.fontset'] = 'stix'
  plt.rcParams['font.size'] = 12
  plt.rcParams['font.weight'] = 'normal'
  plt.minorticks_on()
  plt.grid(visible=True, which='major', linestyle='-')
  plt.grid(visible=True, which='minor', linestyle='--')
  plt.plot(t, p)
  plt.xlabel('t / s')
  plt.ylabel(r'$\rho$')
  plt.xlim(0,)
  plt.title(r'A graph of the change in $\mathrm{\rho}$ in time')
  plt.legend(['11','12','13','21','22','23','31','32','33'])
  plt.tight_layout()
  plt.savefig(f'plots/Plot 2.{i+1}.png', dpi=800)
  plt.show()

  #Increasing the strength of the pump field in steps of 20
  epsilon_pump += 20

  #Turning-on the probe field suddenly when the strength of the
   pump field exceeds 60
  if epsilon_pump > 60:
    epsilon_probe = 1

  #Stopping the simulations when the strength of the pump field
   exceeds 150
  if epsilon_pump > 150:
    break

#Setting the initial conditions of the lambda system
epsilon_probe = 0   #quantification of the strength of the
```

```
   probe field
epsilon_pump = 1    #quantification of the strength of the pump
   field
j = 1               #level in which the atom population lies

for i in t:
  print(f'Epsilon_probe: {epsilon_probe}')
  print(f'Epsilon_pump: {epsilon_pump}')
  print(f'Level, j: {j}')

  #Calling each previously defined function to obtain the
   solution of the 9 differential equations corresponding to
   the set initial conditions
  den_mat = Density_matrix(epsilon_probe)
  ham_mat = Hamiltonian_matrix(epsilon_probe, epsilon_pump)
  real_x_mat = real_imaginary(commutator(den_mat, ham_mat),
   L(j, den_mat))[0]
  imag_x_mat = real_imaginary(commutator(den_mat, ham_mat),
   L(j, den_mat))[1]
  diff_eq = diff_eq_func(real_x_mat, imag_x_mat)
  p = odeint_func(diff_eq_func(real_x_mat, imag_x_mat),t)
   #solution of the 9 differential equations corresponding to
   the set initial conditions, rho

  #Plotting a graph of the change in rho in time
  plt.figure(figsize=(7.5, 10.5))
  plt.rcParams['font.family'] = 'STIXGeneral'
  plt.rcParams['mathtext.fontset'] = 'stix'
  plt.rcParams['font.size'] = 12
  plt.rcParams['font.weight'] = 'normal'
  plt.minorticks_on()
  plt.grid(visible=True, which='major', linestyle='-')
  plt.grid(visible=True, which='minor', linestyle='--')
  plt.plot(t, p)
  plt.xlabel('t / s')
  plt.ylabel(r'$\rho$')
  plt.xlim(0,)
  plt.title(r'A graph of the change in $\mathrm{\rho}$ in time')
  plt.legend(['11','12','13','21','22','23','31','32','33'])
  plt.tight_layout()
  plt.savefig(f'plots/Plot 4.{i+1}.png', dpi=800)
  plt.show()

  #Increasing the strength of the pump field in steps of 20
  epsilon_pump += 20

  #Increasing the strength of the probe field in steps of 1 -
   turning-on the probe field slowly
```

```
    epsilon_probe += 0.2

    #Stopping the simulations when the strength of the pump field
     exceeds 150
    if epsilon_pump > 150:
      break
```