Rabi oscillations in a two-level atom
Theoretical and numerical analysis

Author: Edy Alberto Flores Leal

[†] *Instituto Tecnológico y de Estudios Superiores de Monterrey*
Av. Eugenio Garza Sada 2501 Sur, Tecnológico, 64849 Monterrey, N.L.

1 Theoretical description

Consider a two-level system, i.e., it can only access two eigenstates, $|1\rangle$ and $|2\rangle$. Also, the Hamiltonian is such that

$$\hat{\mathcal{H}} = \hat{\mathcal{H}}_0 + \hat{\mathcal{H}}_{\mathrm{I}}(t), \tag{1}$$

where $\hat{\mathcal{H}}_{\mathrm{I}}(t)$ describes the *atom-light* interaction and perturbates the eigenfunctions of $\hat{\mathcal{H}}_0$ (Foot, 2003). The Hamiltonian $\hat{\mathcal{H}}_0$ produces the following energies:

$$\begin{aligned} \hat{\mathcal{H}}_0|1\rangle &= \hbar\omega_1|1\rangle, \\ \hat{\mathcal{H}}_0|2\rangle &= \hbar\omega_2|2\rangle. \end{aligned} \tag{2}$$

On the other hand, the *atom-light interaction Hamiltonian* is described by an oscillating electric field of a plane monochromatic wave,

$$\hat{\mathcal{H}}_{\mathrm{I}}(t) = -\mathbf{d}\cdot\mathbf{E} \tag{3}$$

where

- $\mathbf{E} = 2E_0\hat{\mathbf{e}}\cos(\mathbf{k}\cdot\mathbf{r} - \omega t)$ is the corresponding *electric field*,
- $\mathbf{d} = -e\mathbf{r}$ is the *electric dipole*,
- $\mathbf{r}$ is the *position vector*, and
- $\mathbf{k}$, ω, and $\hat{\mathbf{e}}$ are the *wave vector*, *angular frequency*, and *polarization*, respectively.

As expected, the system must obey Schrödinger's equation,

$$\mathrm{i}\hbar\frac{\partial}{\partial t}\Psi(t) = [\hat{\mathcal{H}}_0 + \hat{\mathcal{H}}_{\mathrm{I}}(t)]\Psi(t), \tag{4}$$

so we consider a solution where the probability is time-dependent,

$$\Psi(t) = c_1(t)|1\rangle + c_2(t)|2\rangle. \tag{5}$$

First, we plug in the general solution (5) into Schrödinger's equation in the left-hand side:

$$\mathrm{i}\hbar\frac{\partial}{\partial t}\Psi(t) = \mathrm{i}\hbar\frac{\partial}{\partial t}c_1(t)|1\rangle + \mathrm{i}\hbar\frac{\partial}{\partial t}c_2(t)|2\rangle. \tag{6}$$

For the right-hand side, we have

$$\begin{aligned} [\hat{\mathcal{H}}_0 + \hat{\mathcal{H}}_{\mathrm{I}}(t)]\Psi(t) &= \hat{\mathcal{H}}_0[c_1(t)|1\rangle + c_2(t)|2\rangle] + \hat{\mathcal{H}}_{\mathrm{I}}(t)[c_1(t)|1\rangle + c_2(t)|2\rangle] \\ &= \hbar\omega_1 c_1(t)|1\rangle + \hbar\omega_2 c_2(t)|2\rangle + \hat{\mathcal{H}}_{\mathrm{I}}(t)c_1(t)|1\rangle + \hat{\mathcal{H}}_{\mathrm{I}}(t)c_2(t)|2\rangle. \end{aligned} \tag{7}$$

Putting everything together, we see that

$$\mathrm{i}\hbar\frac{\partial}{\partial t}c_1(t)|1\rangle + \mathrm{i}\hbar\frac{\partial}{\partial t}c_2(t)|2\rangle = c_1(t)[\hbar\omega_1 + \hat{\mathcal{H}}_{\mathrm{I}}(t)]|1\rangle + c_2(t)[\hbar\omega_2 + \hat{\mathcal{H}}_{\mathrm{I}}(t)]|2\rangle. \tag{8}$$

At this point, there are two scenarios:

- Applying $|1\rangle$ yields
$$\mathrm{i}\hbar\frac{\partial}{\partial t}c_1(t) = \hbar\omega_1 c_1(t) + H_{\mathrm{I},12}c_2(t). \tag{9}$$
- Applying $|2\rangle$ yields
$$\mathrm{i}\hbar\frac{\partial}{\partial t}c_2(t) = \hbar\omega_2 c_2(t) + H_{\mathrm{I},21}c_1(t). \tag{10}$$

Notice that both equations (9) and (10) are dependent on a term of the form $H_{\mathrm{I},ij} = \langle i|\hat{\mathcal{H}}_{\mathrm{I}}(t)|j\rangle$. This would be equivalent to write

$$\begin{aligned} H_{\mathrm{I},ij} &= -\langle i\,|\,\mathbf{d}\cdot\mathbf{E}\,|\,j\rangle \\ &= -\langle i\,|\,\mathbf{d}\cdot\hat{\mathbf{e}}\,|\,j\rangle E_0\left[\mathrm{e}^{\mathrm{i}(\mathbf{k}\cdot\mathbf{r}-\omega t)} + \mathrm{e}^{-\mathrm{i}(\mathbf{k}\cdot\mathbf{r}-\omega t)}\right]. \end{aligned} \tag{11}$$

To simplify equation (11), we could neglect the phase $\mathbf{k}\cdot\mathbf{r}$ if we assume that the impinging electromagnetic wave has a wavelength bigger than the atom itself (Foot, 2003). Hence equation (11) could be expressed as

$$H_{\mathrm{I},ij} = -\langle i\,|\,\mathbf{d}\cdot\hat{\mathbf{e}}\,|\,j\rangle E_0\left(\mathrm{e}^{\mathrm{i}\omega t} + \mathrm{e}^{-\mathrm{i}\omega t}\right). \tag{12}$$

This assumption is called *dipole approximation*. Now, we are left with a system of two Partial Differential Equations. To solve it, we introduce some substitutions:

$$\begin{aligned} c_1(t) &= \tilde{c}_1(t)\mathrm{e}^{-\mathrm{i}\omega_1 t}, \\ c_2(t) &= \tilde{c}_2(t)\mathrm{e}^{-\mathrm{i}\omega_2 t}. \end{aligned} \tag{13}$$

We start with $c_1(t)$. The left-hand side of equation (9) takes the form of

$$\begin{aligned} \mathrm{i}\hbar\frac{\partial}{\partial t}c_1(t) &= \mathrm{i}\hbar\frac{\partial}{\partial t}[\tilde{c}_1(t)\mathrm{e}^{-\mathrm{i}\omega_1 t}] \\ &= \mathrm{i}\hbar\frac{\partial}{\partial t}\tilde{c}_1(t)\mathrm{e}^{-\mathrm{i}\omega_1 t} + \hbar\omega_1\tilde{c}_1(t)\mathrm{e}^{-\mathrm{i}\omega_1 t}. \end{aligned} \tag{14}$$

Now, the right-hand side:

$$\hbar\omega_1 c_1(t) + H_{\mathrm{I},12}c_2(t) = \hbar\omega_1\tilde{c}_1(t)\mathrm{e}^{-\mathrm{i}\omega_1 t} + H_{\mathrm{I},12}\tilde{c}_2(t)\mathrm{e}^{-\mathrm{i}\omega_2 t}. \tag{15}$$

Equating both sides, we have

$$\begin{aligned} \mathrm{i}\hbar\frac{\partial\tilde{c}_1(t)}{\partial t}\mathrm{e}^{-\mathrm{i}\omega_1 t} + \hbar\omega_1\tilde{c}_1(t)\mathrm{e}^{-\mathrm{i}\omega_1 t} &= \hbar\omega_1\tilde{c}_1(t)\mathrm{e}^{-\mathrm{i}\omega_1 t} + H_{\mathrm{I},12}\tilde{c}_2(t)\mathrm{e}^{-\mathrm{i}\omega_2 t} \\ \mathrm{i}\hbar\frac{\partial\tilde{c}_1(t)}{\partial t}\mathrm{e}^{-\mathrm{i}\omega_1 t} &= H_{\mathrm{I},12}\tilde{c}_2(t)\mathrm{e}^{-\mathrm{i}\omega_2 t} = -E_0\left(\mathrm{e}^{\mathrm{i}\omega t} + \mathrm{e}^{-\mathrm{i}\omega t}\right)\langle 1|\mathbf{d}\cdot\hat{\mathbf{e}}|2\rangle\tilde{c}_2(t)\mathrm{e}^{-\mathrm{i}\omega_2 t}. \end{aligned} \tag{16}$$

We do the same for the second equation:

$$\begin{aligned} \mathrm{i}\hbar\frac{\partial c_2(t)}{\partial t} &= \mathrm{i}\hbar\frac{\partial}{\partial t}\left[\tilde{c}_2(t)\mathrm{e}^{-\mathrm{i}\omega_2 t}\right] \\ &= \mathrm{i}\hbar\frac{\partial}{\partial t}\tilde{c}_2(t)\mathrm{e}^{-\mathrm{i}\omega_2 t} + \hbar\omega_2\tilde{c}_2(t)\mathrm{e}^{-\mathrm{i}\omega_2 t}. \end{aligned} \tag{17}$$

For the right-hand side, we have

$$\hbar\omega_2 c_2(t) + H_{\mathrm{I},21}c_1(t) = \hbar\omega_2\tilde{c}_2(t)\mathrm{e}^{-\mathrm{i}\omega_2 t} + H_{\mathrm{I},21}\tilde{c}_1(t)\mathrm{e}^{-\mathrm{i}\omega_1 t}. \tag{18}$$

Equating both sides, we get

$$
\begin{aligned}
\mathrm{i}\hbar\frac{\partial}{\partial t}\tilde{c}_2(t)\mathrm{e}^{-\mathrm{i}\omega_2 t} + \hbar\omega_2\tilde{c}_2(t)\mathrm{e}^{-\mathrm{i}\omega_2 t} &= \hbar\omega_2\tilde{c}_2(t)\mathrm{e}^{-\mathrm{i}\omega_2 t} + H_{\mathrm{I},21}\tilde{c}_1(t)\mathrm{e}^{-\mathrm{i}\omega_1 t} \\
\mathrm{i}\hbar\frac{\partial}{\partial t}\tilde{c}_2(t)\mathrm{e}^{-\mathrm{i}\omega_2 t} &= H_{\mathrm{I},21}\tilde{c}_1(t)\mathrm{e}^{-\mathrm{i}\omega_1 t} = -E_0\left(\mathrm{e}^{\mathrm{i}\omega t} + \mathrm{e}^{-\mathrm{i}\omega t}\right)\langle 2|\mathbf{d}\cdot\hat{\mathbf{e}}|1\rangle\tilde{c}_1(t)\mathrm{e}^{-\mathrm{i}\omega_1 t}.
\end{aligned} \tag{19}
$$

In summary, we have the following result:

$$
\frac{\partial}{\partial t}\tilde{c}_1(t) = \mathrm{i}\frac{d_{12}E_0}{\hbar}\left(\mathrm{e}^{\mathrm{i}\omega t} + \mathrm{e}^{-\mathrm{i}\omega t}\right)\tilde{c}_2(t)\mathrm{e}^{-\mathrm{i}(\omega_2-\omega_1)t}, \tag{20}
$$

$$
\frac{\partial}{\partial t}\tilde{c}_2(t) = \mathrm{i}\frac{d_{21}E_0}{\hbar}\left(\mathrm{e}^{\mathrm{i}\omega t} + \mathrm{e}^{-\mathrm{i}\omega t}\right)\tilde{c}_1(t)\mathrm{e}^{\mathrm{i}(\omega_2-\omega_1)t}. \tag{21}
$$

From equation (20), we define the *detuning* and *Rabi frequency*, respectively, as

$$
\Delta = \omega - \omega_{12}, \tag{22}
$$

$$
\Omega = -\frac{d_{12}E_0}{\hbar}, \tag{23}
$$

where $\omega_{12} = \omega_2 - \omega_1$ and $d_{ij} = \langle i\,|\,\mathbf{d}\cdot\hat{\mathbf{e}}\,|\,j\rangle$. With these expressions, equations (20) and (21) take the following shape:

$$
\frac{\partial}{\partial t}\tilde{c}_1(t) = -\mathrm{i}\Omega\mathrm{e}^{\mathrm{i}\Delta t}\tilde{c}_2(t), \tag{24}
$$

$$
\frac{\partial}{\partial t}\tilde{c}_2(t) = -\mathrm{i}\Omega^*\mathrm{e}^{-\mathrm{i}\Delta t}\tilde{c}_1(t). \tag{25}
$$

It is relevant to mention that we ignore the terms oscillating with a frequency of $\omega + \omega_{12}$. The physical reason for doing so is that we want to neglect fast oscillations since these will tend to go to zero; this procedure is called the *rotating wave approximation*. To solve this system of equations, we start from the ground state, assuming that $\tilde{c}_1(0) = 1$ and $\tilde{c}_2(0) = 0$. Also, we take Δ and Ω as real-valued. We will solve this system *numerically*. This system of equations could be solved analytically as well. Take into account that equation (25) can be derived respect to time and then use equation (24) in it. From there, the remaining procedure is to deal with the algebra. We should arrive to the next solutions:

$$
\tilde{c}_1(t) = \left[\cos\left(\tilde{\Omega}t\right) - \frac{\mathrm{i}\Delta}{2\tilde{\Omega}}\sin\left(\tilde{\Omega}t\right)\right]\mathrm{e}^{\frac{\mathrm{i}\Delta t}{2}}, \tag{26}
$$

$$
\tilde{c}_2(t) = -\frac{\mathrm{i}\Omega^*}{\tilde{\Omega}}\sin\left(\tilde{\Omega}\right)\mathrm{e}^{-\frac{\mathrm{i}\Delta t}{2}}, \tag{27}
$$

where

$$
\tilde{\Omega} \equiv \sqrt{|\Omega|^2 + \frac{\Delta^2}{4}}. \tag{28}
$$

2 Numerical approach

We will solve the system of equations using the Fourth-Order Runge-Kutta method.

2.1 `Python's code`

Here we provide a detailed explanation of the code used to solve this system of two differential equations. First, we start defining the parameters:

```python
# Libraries
import numpy as np
import matplotlib.pyplot as plt

# Plot parameters
plt.rcParams.update({
    'lines.linewidth': 2,     # linewidth
    'text.usetex': True,      # LaTeX font
    'font.family': 'serif',   # Serif family
    'font.size': 16,          # font size
    'axes.titlesize': 20,     # title size
    'axes.grid': True,        # grid
    'grid.linestyle': "-.",   # grid style
})

# Physical parameters
Ω = 1  # Rabi frequency
Δ = 1  # Detuning
```

The selection of the values of Ω and Δ, in this case, is arbitrary, and we consider them equal to 1 for practical purposes. Now, we define the time array

```python
# Time parameters
t_0 = 0                         # initial time (s)
t_f = 10                        # final time   (s)
Δt = 0.01                       # step size    (s)
n = int((t_f - t_0) / Δt) + 1  # iterations
t = np.linspace(t_0, t_f, n)    # time array   (s)
```

It is relevant to mention that we define the time parameter as a `np.linspace` array. However, it can also be declared as an empty vector that gets updated in every iteration. In the end, this choice is a matter of preference. We now define the Fourth-Order Runge-Kutta method:

```python
# Fourth-Order Runge-Kutta function
def RK4(f, x0, y0, z0, h):
    k1y = h * f(x0, y0, z0)[0]
    k1z = h * f(x0, y0, z0)[1]
    #
    k2y = h * f(x0 + h / 2, y0 + k1y / 2, z0 + k1z / 2)[0]
    k2z = h * f(x0 + h / 2, y0 + k1y / 2, z0 + k1z / 2)[1]
    #
    k3y = h * f(x0 + h / 2, y0 + k2y / 2, z0 + k2z / 2)[0]
    k3z = h * f(x0 + h / 2, y0 + k2y / 2, z0 + k2z / 2)[1]
    #
    k4y = h * f(x0 + h, y0 + k3y, z0 + k3z)[0]
    k4z = h * f(x0 + h, y0 + k3y, z0 + k3z)[1]

    # Approximation
    y1 = y0 + (k1y + 2 * k2y + 2 * k3y + k4y) / 6
    z1 = z0 + (k1z + 2 * k2z + 2 * k3z + k4z) / 6
    return y1, z1
```

This function takes the following inputs:

- `f`: functions to evaluate.
- `x0`: initial condition (independent variable).
- `y0`: initial condition (dependent variable).
- `z0`: initial condition (dependent variable).
- `h`: step size.

The next step is to define the functions.

```python
# Differential Equations
def f(t, c1, c2):
    return complex(-1j * Ω * np.exp(1j * Δ * t) * c2), complex(-1j * Ω * np.exp(-1j * Δ
    ↪  * t) * c1)
```

That is, `f(t, c1, c2)` defines the equations

$$\frac{\partial}{\partial t}\tilde{c}_1(t) = -\mathrm{i}\Omega \mathrm{e}^{\mathrm{i}\Delta t}\tilde{c}_2(t),$$
$$\frac{\partial}{\partial t}\tilde{c}_2(t) = -\mathrm{i}\Omega^* \mathrm{e}^{-\mathrm{i}\Delta t}\tilde{c}_1(t).$$

We initialize our storage arrays:

```python
# Initial arrays
c1, c1[0] = np.zeros(n, dtype = 'complex'), 1 + 0j
c2, c2[0] = np.zeros(n, dtype = 'complex'), 0 + 0j
```

In the previous code, we create an empty array of n elements and store the initial conditions as the first value. Finally, we perform the `for` cycle:

```python
# RK4 method evaluation
for i in range(n - 1):
    c1[i + 1], c2[i + 1] = RK4(f, t[i], c1[i], c2[i], Δt)
```

If we plot t versus $|c_1(t)|^2$, $|c_2(t)|^2$, and $|c_1(t) + c_2(t)|^2$, we obtain the next plot:

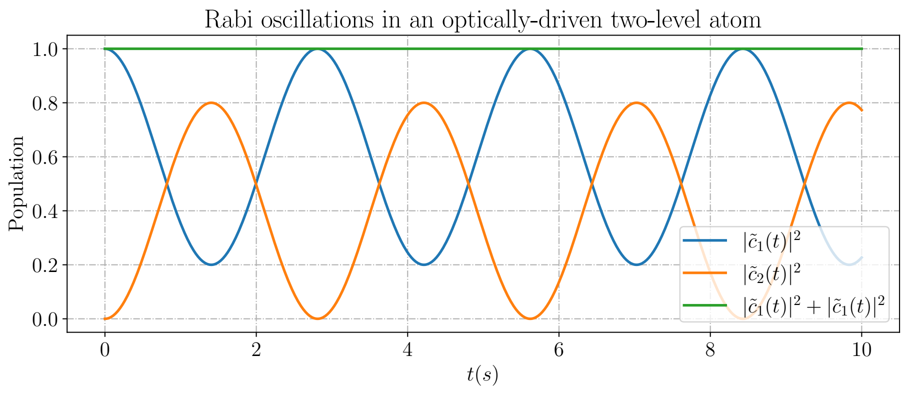

Figure 1: Evolution of the change of state probability through time. Here, we use $\Omega = 1$ and $\Delta = 1$.

The full code is shown here:

```python
# Libraries
import numpy as np
import matplotlib.pyplot as plt

# Plot parameters
plt.rcParams.update({
    'lines.linewidth': 2,      # linewidth
    'text.usetex': True,       # LaTeX font
    'font.family': 'serif',    # Serif family
    'font.size': 16,           # font size
    'axes.titlesize': 20,      # title size
    'axes.grid': True,         # grid
    'grid.linestyle': "-.",    # grid style
})

# Physical parameters
Ω = 1  # Rabi frequency
Δ = 1  # Detuning

# Time parameters
t_0 = 0                         # initial time (s)
t_f = 10                        # final time   (s)
Δt = 0.01                       # step size    (s)
n = int((t_f - t_0) / Δt) + 1   # iterations
t = np.linspace(t_0, t_f, n)    # time vector  (s)

# Fourth-Order Runge-Kutta function
def RK4(f, x0, y0, z0, h):
    k1y = h * f(x0, y0, z0)[0]
    k1z = h * f(x0, y0, z0)[1]
    #
    k2y = h * f(x0 + h / 2, y0 + k1y / 2, z0 + k1z / 2)[0]
    k2z = h * f(x0 + h / 2, y0 + k1y / 2, z0 + k1z / 2)[1]
    #
    k3y = h * f(x0 + h / 2, y0 + k2y / 2, z0 + k2z / 2)[0]
    k3z = h * f(x0 + h / 2, y0 + k2y / 2, z0 + k2z / 2)[1]
    #
    k4y = h * f(x0 + h, y0 + k3y, z0 + k3z)[0]
    k4z = h * f(x0 + h, y0 + k3y, z0 + k3z)[1]

    # Approximation
    y1 = y0 + (k1y + 2 * k2y + 2 * k3y + k4y) / 6
    z1 = z0 + (k1z + 2 * k2z + 2 * k3z + k4z) / 6
    return y1, z1

# Differential Equations
def f(t, c1, c2):
    return complex(-1j * Ω * np.exp(1j * Δ * t) * c2), complex(-1j * Ω * np.exp(-1j * Δ
    ↪  * t) * c1)

# Initial arrays
c1, c1[0] = np.zeros(n, dtype = 'complex'), 1 + 0j
```

```python
c2, c2[0] = np.zeros(n, dtype = 'complex'), 0 + 0j

# RK4 method evaluation
for i in range(n - 1):
    c1[i + 1], c2[i + 1] = RK4(f, t[i], c1[i], c2[i], Δt)

# Plot
plt.figure(figsize = (10, 4.5))
plt.plot(t, np.abs(c1)**2, label = r'$|\tilde{c}_{1}(t)|^2$')
plt.plot(t, np.abs(c2)**2, label = r'$|\tilde{c}_{2}(t)|^2$')
plt.plot(t, np.abs(c1)**2 + np.abs(c2)**2, label = r'$|\tilde{c}_{1}(t)|^2 +
↪  |\tilde{c}_{1}(t)|^2$')
plt.title(r'Rabi oscillations in an optically-driven two-level atom')
plt.xlabel(r'$t(s)$')
plt.ylabel(r'Population')
plt.legend(loc = 'lower right', borderpad = 0.2)
plt.tight_layout()
plt.savefig('pyplot.pdf')
```

2.2 Julia's code

In this section, we provide a detailed description of the code to simulate the change-of-state probability evolution through time, but this time using `Julia`. We expect to do the same for other programming languages. Once again, we write the whole Runge-Kutta method instead of using the ODE solvers available because we want to do it as a matter of practice. We establish the initial conditions:

```julia
# Libraries
using LinearAlgebra
using Plots, ColorSchemes, LaTeXStrings

# Plot parameters
default(fontfamily = "Computer modern",
            legend = true,
          tickfont = font(12),
         guidefont = font(16),
         titlefont = font(20),
         linewidth = 2,
              grid = true,
              size = (800, 400),
          margin = 10Plots.mm)
```

We use the same time consideration as in the `Python`'s script.

```julia
# Time parameters
t0 = 0                          # initial time (s)
t1 = 10                         # final time   (s)
Δt = 0.01                       # step size    (s)
n = Int64((t1 - t0) / Δt)       # iterations
t = t0:Δt:(t1 - Δt)             # time array (s)
```

Next, we define the Runge-Kutta method using the previous equations.

```julia
# Fourth-Order Runge-Kutta function
function RK4(f, x0, y0, z0, h)
    k1y = h * f(x0, y0, z0)[1]
    k1z = h * f(x0, y0, z0)[2]
    #
    k2y = h * f(x0 + h / 2, y0 + k1y / 2, z0 + k1z / 2)[1]
    k2z = h * f(x0 + h / 2, y0 + k1y / 2, z0 + k1z / 2)[2]
    #
    k3y = h * f(x0 + h / 2, y0 + k2y / 2, z0 + k2z / 2)[1]
    k3z = h * f(x0 + h / 2, y0 + k2y / 2, z0 + k2z / 2)[2]
    #
    k4y = h * f(x0 + h, y0 + k3y, z0 + k3z)[1]
    k4z = h * f(x0 + h, y0 + k3y, z0 + k3z)[2]

    # Approximation
    y1 = Float64(y0 + (k1y + 2 * k2y + 2 * k3y + k4y) / 6)
    z1 = Float64(z0 + (k1z + 2 * k2z + 2 * k3z + k4z) / 6)
    return y1, z1
end
```

Once we declare the Runge-Kutta expressions, we can proceed to declare the function that contains the equations to solve:

```julia
# Differential Equations
function f(t, c1, c2)
    return ComplexF64(-1im * Ω * exp(1im * Δ * t) * c2), ComplexF64(-1im * Ω * exp(-1im
    ↪  * Δ * t) * c1)
end
```

We initialize the storage arrays:

```julia
# Initial arrays
c1, c1[1] = zeros(ComplexF64, n), ComplexF64(1 + 0im)
c2, c2[1] = zeros(ComplexF64, n), ComplexF64(0 + 0im)
```

Finally, we evaluate the Runge-Kutta method with the `for` cycle:

```julia
# RK4 method evaluation
for i in 1:(n - 1)
    c1[i + 1], c2[i + 1] = RK4(f, t[i], c1[i], c2[i], Δt)
end
```

Plotting the time t versus the coefficients $|\tilde{c}_1(t)|^2$, $|\tilde{c}_2(t)|^2$ and $|\tilde{c}_1(t)|^2 + |\tilde{c}_2(t)|^2$, we get the following plot:

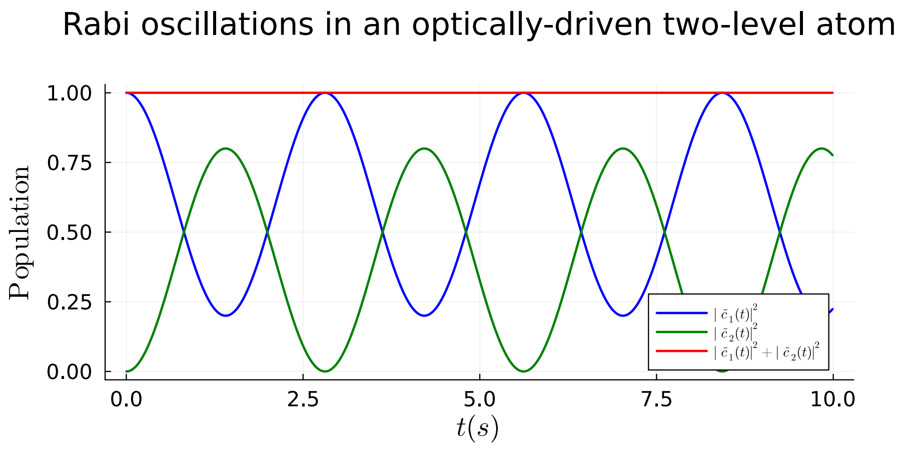

Figure 2: Evolution of the angle through time.

Here, we show the full code:

```
# Libraries
using LinearAlgebra
using Plots, ColorSchemes, LaTeXStrings

# Plot parameters
default(fontfamily = "Computer modern",
            legend = true,
          tickfont = font(12),
         guidefont = font(16),
         titlefont = font(18),
         linewidth = 2,
              grid = true,
              size = (800, 400),
            margin = 10Plots.mm)

# Physical parameters
Ω = 1   # Rabi frequency
Δ = 1   # Detuning

# Time parameters
t0 = 0                          # initial time (s)
t1 = 10                         # final time   (s)
Δt = 0.01                       # step size    (s)
n = Int64((t1 - t0) / Δt)       # iterations
t = t0:Δt:(t1 - Δt)             # time vector  (s)

# Fourth-Order Runge-Kutta function
function RK4(f, x0, y0, z0, h)
    k1y = h * f(x0, y0, z0)[1]
    k1z = h * f(x0, y0, z0)[2]
    #
    k2y = h * f(x0 + h / 2, y0 + k1y / 2, z0 + k1z / 2)[1]
    k2z = h * f(x0 + h / 2, y0 + k1y / 2, z0 + k1z / 2)[2]
    #
```

```julia
    k3y = h * f(x0 + h / 2, y0 + k2y / 2, z0 + k2z / 2)[1]
    k3z = h * f(x0 + h / 2, y0 + k2y / 2, z0 + k2z / 2)[2]
    #
    k4y = h * f(x0 + h, y0 + k3y, z0 + k3z)[1]
    k4z = h * f(x0 + h, y0 + k3y, z0 + k3z)[2]

    # Approximation
    y1 = ComplexF64(y0 + (k1y + 2 * k2y + 2 * k3y + k4y) / 6)
    z1 = ComplexF64(z0 + (k1z + 2 * k2z + 2 * k3z + k4z) / 6)
    return y1, z1
end

# Differential Equations
function f(t, c1, c2)
    return ComplexF64(-1im * Ω * exp(1im * Δ * t) * c2), ComplexF64(-1im * Ω * exp(-1im
     ↪  * Δ * t) * c1)
end

# Initial arrays
c1, c1[1] = zeros(ComplexF64, n), ComplexF64(1 + 0im)
c2, c2[1] = zeros(ComplexF64, n), ComplexF64(0 + 0im)

# RK4 method evaluation
for i in 1:(n - 1)
    c1[i + 1], c2[i + 1] = RK4(f, t[i], c1[i], c2[i], Δt)
end

plot(t, [abs2.(c1), abs2.(c2), abs2.(c1) + abs2.(c2)], grid = true, label =
↪  [L"$|\tilde{c}_{1}(t)|^2$" L"$|\tilde{c}_{2}(t)|^2$" L"|\tilde{c}_{1}(t)|^2 +
↪  |\tilde{c}_{2}(t)|^2"], linewidth = 2)

# Plot
plot(t, abs2.(c1), label = L"$|\tilde{c}_{1}(t)|^2$", c=:blue, xlabel = L"$t (s)$",
↪  ylabel = L"\textrm{Population}")
plot!(t, abs2.(c2), label = L"$|\tilde{c}_{2}(t)|^2$", c=:green)
plot!(t, abs2.(c1) + abs2.(c2), label = L"|\tilde{c}_{1}(t)|^2 + |\tilde{c}_{2}(t)|^2",
↪  c=:red)
title!("Rabi oscillations in an optically-driven two-level atom")
savefig("juliaplot.pdf")
```

References

C J Foot. *Atomic Physics*. Oxford Master Series in Physics. Oxford University Press, London, England, August 2003.