GRAPE for Transmon Qubits

L. Van Damme

September 29, 2025

Contents

Overview **2**

Example 1: Leakage-Free NOT Gate **3**

Example 2: Two-photon transition **5**

Example 3: Analytically shaped Hadamard Gate **7**

Example 4: Robust Fourier pulse **9**

Example 5: Selective pulse **12**

Overview

This python code provides tools to optimize control pulses in transmon qubits. The qubit dynamics are described by the Hamiltonian in the rotating-wave approximation (with $\hbar = 1$):

$$H/(2\pi) = \frac{\nu_{\mathrm{Ref}}}{2}\Big(u(t)\, a^\dagger + u^*(t)\, a\Big) + (\nu_q - \nu_d)\, a^\dagger a + \frac{\alpha}{2}\, a^{\dagger 2} a^2, \tag{1}$$

where:

- ν_{Ref} is a reference frequency (Hz) that sets the overall drive scale.
- $u(t) = u_x(t) + i\, u_y(t)$ is the normalized control pulse in complex form, with $u_x(t)$ (in-phase) and $u_y(t)$ (quadrature) real and dimensionless.
- ν_q is the qubit frequency (the $|0\rangle \leftrightarrow |1\rangle$ transition) in Hz.
- ν_d is the carrier (drive) frequency in Hz.
- α is the transmon anharmonicity in Hz.
- a and $a^\dagger$ are the annihilation and creation operators, respectively.

The user can freely adjust these parameters to match experimental requirements or explore different regimes. The parameter space may be one-dimensional or multi-dimensional, allowing optimization over grids, multiple target gates (or target states), arbitrary computational subspaces, or more complex sweeps. Parameters can also be time-dependent, enabling the study of systems affected by temporal noise or drift. Control pulses can be decomposed into a linear combination of analytical functions to generate experimentally feasible shapes and can be constrained to comply with hardware limitations. The user can optimize gate operations or state probability.

Limitations. The code supports only single or uncoupled transmons, does not implement multi-qubit gates, does not allow for multiple control channels with different drive frequencies, neglects relaxation processes, and is valid only within the rotating-wave approximation.

Requirement

This code uses numpy, numba, scipy (in particular scipy.optimize), matplotlib, and copy.

Example 1: Leakage-Free NOT Gate

`Example01_LeakageFreeGate.ipynb`

General problem. We consider a three-level system driven on resonance. The system Hamiltonian under the rotating-wave approximation is given by

$$H(t) = \frac{2\pi\nu_{\mathrm{Ref}}}{2}\Big(u(t)\,a^\dagger + u^*(t)\,a\Big) + \frac{2\pi\alpha}{2}\,a^{\dagger 2}a^2,$$

where $\alpha = -100$ MHz and ν_{Ref} is set to 10 MHz. To account for three energy levels, the operators a and $a^\dagger$ are truncated as

$$a = |0\rangle\langle 1| + \sqrt{2}\,|1\rangle\langle 2|\,,\quad a^\dagger = |1\rangle\langle 0| + \sqrt{2}\,|2\rangle\langle 1|\,,$$

The goal is to optimize the pulse shape $u(t)$ to implement the gate

$$U_{\mathrm{T}} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

on the computational subspace defined by $|0\rangle$ and $|1\rangle$. The time is discretized into 120 steps of 0.1 ns, corresponding to a 12 ns-long control pulse.

From a control perspective, this problem is underconstrained, meaning that infinitely many pulse shapes can reach the target to numerical precision. To select among these degenerate solutions, we minimize the pulse energy

$$(2\pi\nu_R)^2\int_0^T u_x^2(t) + u_y^2(t)dt,$$

which restricts the search to solutions that achieve the target while using minimal control amplitude.

Problem parameters. This problem is specified by calling the function `ProblemParameters` with the following arguments:

```
p = ProblemParameters(
            alpha = -100e6,      # Anharmonicity in Hz
            nuRef = 10e6,        # Reference pulse amplitude
            NLevels = 3,         # Number of energy levels
            dt = 0.1e-9,         # Timestep in s.
            Nt = 120,            # Number of time steps
            CompSpace = [0,1],   # Computational states indices
            Target = [[0,1],     # Target gate
                      [1,0]]
            )
```

Note that the carrier and qubit frequencies do not need to be specified for this simple problem, as the drive is assumed to be on-resonance with the qubit transition by default.

Initial pulse guess. Any arbitrary initial guess $u_{x,i}(t)$ and $u_{y,i}(t)$ can be defined by the user. For convenience, the utility function `InitPulse` provides several predefined pulse types, as described in its documentation header. In this example, we use a DRAG-like pulse of the form

$$u_{x,i}(t) = A\sin\!\left(\tfrac{\pi t}{T}\right),\quad u_{y,i}(t) = B\cos\!\left(\tfrac{\pi t}{T}\right),$$

where the parameters A and B are calibrated to minimize leakage under the system parameters specified in p. This initial guess is generated with the command:

```
ux0, uy0 = InitPulse(p,PulseType = "CosDrag", ThetaTar=np.pi)
```

Here, the option ThetaTar=np.pi specifies that DRAG should implement a π-rotation in the computational basis.

Optimization. The pulse optimization is carried out with the class grape, which supports several options documented in its header. In this example, we search for a minimum-energy solution using the options:

```
gopt=Grape(
        Maxiter=500,         # Maximum number of iterations
        EnergyMinimum=True   # Find minimum-energy solution
        )
```

The pulse optimization is obtained via the command:

```
ux, uy, J = gopt.Optimize(p,ux0, uy0)
```

which returns the optimal pulses ux and uy and the value of the cost function J.

Results. Since the optimal controls u_x and u_y are normalized, the physical pulse components (in Hz) are given by $\nu_{\rm Ref}u_x$ and $\nu_{\rm Ref}u_y$. They can be visualized with the command:

```
plt.figure()
plt.plot(p.tc,p.nuRef*ux,p.tc,p.nuRef*uy)
plt.show()
```

The resulting pulse is shown in Fig. 1. Additional functions are available for analyzing results. The complete script is provided in Example01_LeakageFreeGate.ipynb.

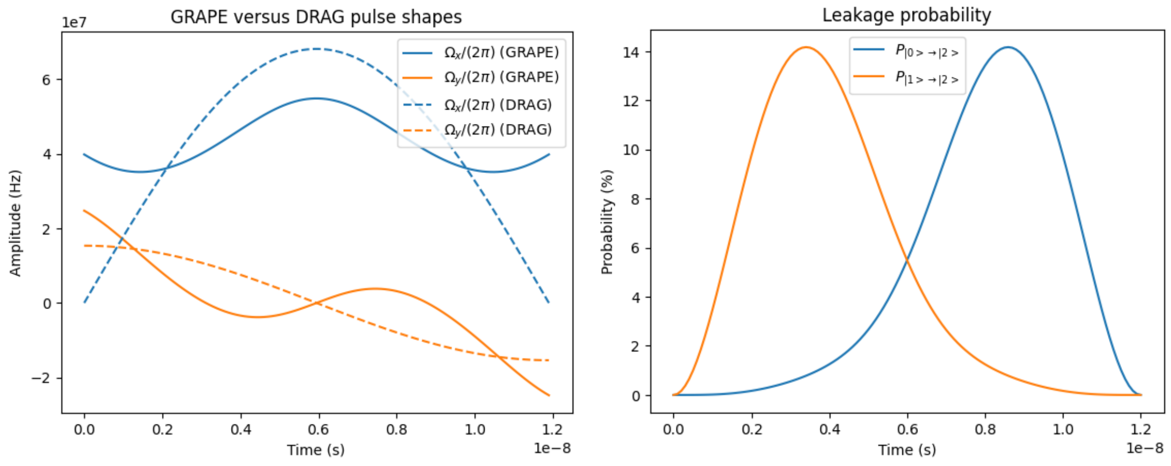

Figure 1: Left panel: Pulse shapes in Hz before and after optimization. Right panel: Leakage to state $|2\rangle$ as a function of time.

Example 2: Two-photon transition

Example02_TwoPhotonTransition.ipynb

General problem. We consider a transmon with anharmonicity $\alpha = -200$ MHz and $|0\rangle \leftrightarrow |1\rangle$ frequency $\nu_q = 5$ GHz, driven at the two-photon resonance

$$\nu_d = \nu_q + \alpha/2 = \nu_{02}/2.$$

In the frame rotating at ν_d and under the RWA, the control Hamiltonian is

$$H = \frac{2\pi\nu_{\text{Ref}}}{2}\Big(u(t)\,a^\dagger + u^*(t)\,a\Big) - \frac{2\pi\alpha}{4}\,a^\dagger a + \frac{2\pi\alpha}{2}\,a^{\dagger 2}a^2,$$

To capture leakage, we simulate the system with four energy levels by truncating the operators a and $a^\dagger$ as

$$a = |0\rangle\langle 1| + \sqrt{2}\,|1\rangle\langle 2| + \sqrt{3}\,|2\rangle\langle 3|\,, \quad a^\dagger = |1\rangle\langle 0| + \sqrt{2}\,|2\rangle\langle 1| + \sqrt{3}\,|3\rangle\langle 2|\,.$$

Starting from $|\psi(0)\rangle = |0\rangle$, the objective is to optimize $u(t)$ to transfer the state to $|2\rangle$, such that

$$|\langle 2|\psi(T)\rangle|^2 = 1,$$

where T is the duration of the control pulse. The time is discretized into 200 steps of 0.1 ns, corresponding to $T = 20$ ns.

As for the previous example, this problem is underconstrained, meaning that infinitely many pulse shapes can reach the target to numerical precision. To select among these degenerate solutions, we minimize the pulse energy

$$(2\pi\nu_R)^2\int_0^T u_x^2(t) + u_y^2(t)dt,$$

which restricts the search to solutions that achieve the target while using minimal control amplitude.

Problem parameters. This problem is specified by calling the function `ProblemParameters` with the following arguments:

```
p = ProblemParameters(
            alpha=-200e6,           # Anharmonicity parameter in Hz
            QubitFreq=5e9,          # Qubit frequency in Hz
            CarrFreq = 5e9-200e6/2, # Carrier frequency in Hz
            nuRef = 100e6,          # Reference frequency in Hz
            NLevels = 4,            # Number of energy levels
            Nt=200,                 # Number of time steps
            dt=0.1e-9,              # Time step in s.
            Psi0 = [1,0,0,0],       # Initial state: |0>
            Target =[0,0,1,0],      # Target state: |2>
            )
```

Note that the computational subspace does not need to be specified, as it is not used for state-to-state problems, because the initial and target states are defined in the full Hilbert space.

Initial pulse guess. In this example, we use a symmetric/antisymmetric Fourier series of the form

$$u_{x,i}(t) = \sum_{k=1}^{3} a_k \sin\left(\frac{(2k-1)\pi t}{T}\right), \quad u_{y,i}(t) = \sum_{k=1}^{3} b_k \sin\left(\frac{2k\pi t}{T}\right),$$

where a_k and b_k are chosen randomly. This initial guess is generated with the command:

```
ux0, uy0 = InitPulse(p,PulseType="RandSymAnt")
```

Optimization. The pulse optimization is carried out with the class `grape`, which supports several options documented in its header. In this example, we search for a minimum-energy solution using the options:

```
gopt=Grape(
        Maxiter=1000,        # Maximum number of iterations
        EnergyMinimum=True   # Find minimum-energy solution
        )
```

The pulse optimization is obtained via the command:

```
ux, uy, J = gopt.Optimize(p,ux0, uy0)
```

which returns the optimal pulses `ux` and `uy` and the value of the cost function `J`.

Results. Since the optimal controls u_x and u_y are normalized, the physical pulse components (in Hz) are given by $\nu_{\text{Ref}} u_x$ and $\nu_{\text{Ref}} u_y$. They can be visualized with the command:

```
plt.figure()
plt.plot(p.tc,p.nuRef*ux,p.tc,p.nuRef*uy)
plt.show()
```

The resulting pulse is shown in Fig. 2. Additional functions are available for analyzing results. The complete script is provided in `Example02_TwoPhotonTransition.ipynb`.

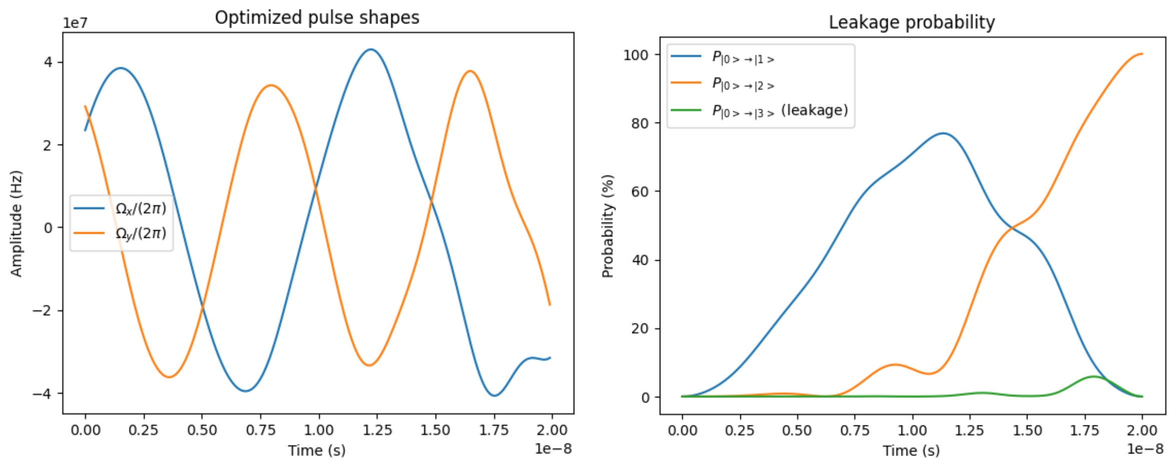

Figure 2: Left: Pulse amplitudes (in Hz) implementing a two-photon transition from $|0\rangle$ to $|2\rangle$. Right: Transition probabilities as a function of time.

Example 3: Analytically shaped Hadamard Gate

Example03_AnalyticallyShapedPulse.ipynb

General problem. We consider a three-level system driven on resonance. The system Hamiltonian under the rotating-wave approximation is given by

$$H(t) = \frac{2\pi\nu_{\mathrm{Ref}}}{2}\Big(u(t)\,a^\dagger + u^*(t)\,a\Big) + \frac{2\pi\alpha}{2}\,a^{\dagger 2}a^2,$$

where $\alpha = -200$ MHz and ν_{Ref} is set to 10 MHz. To account for three energy levels, the operators a and $a^\dagger$ are truncated as

$$a = |0\rangle\langle 1| + \sqrt{2}\,|1\rangle\langle 2|\,, \quad a^\dagger = |1\rangle\langle 0| + \sqrt{2}\,|2\rangle\langle 1|\,,$$

where $|0\rangle$, $|1\rangle$, and $|2\rangle$ correspond to the three lowest transmon eigenstates. The objective is to implement a Hadamard gate,

$$U_{\mathrm{T}} = \tfrac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix},$$

on the computational subspace spanned by $|0\rangle$ and $|1\rangle$. In contrast to examples 1 and 2, the pulse shape is restricted to a basis of analytical functions. Specifically, we use the parametrization

$$u_x(t) = \sum_{k=1}^{3} a_{x_k}\,\sin\Big(\tfrac{(2k-1)\pi t}{T}\Big)\,,$$
$$u_y(t) = \sum_{k=1}^{3} a_{y_k}\,\cos\Big(\tfrac{(2k-1)\pi t}{T}\Big)\,,$$

and search for coefficients $\{a_{x_k}, a_{y_k}\}$ that realize the target gate. The control pulse is discretized into 160 time steps of 0.1 ns, resulting in a total duration of 16 ns.

Problem parameters. This problem is specified by calling the function `ProblemParameters` with the following arguments:

```
p = ProblemParameters(
            QubitFreq = 5e9,     # QubitFreq in Hz
            alpha = -200e6,      # Anharmonicity in Hz
            NLevels = 3,         # Number of energy levels
            dt = 0.1e-9,         # Timestep in s.
            Nt = 160,            # Number of time steps
            CompSpace = [0,1],   # Indices of the computational states
            Target = [[1, 1],    # Target: Hadamard gate
                      [1,-1]]/np.sqrt(2)
            )
```

Note that the carrier frequency does not need to be specified, as the drive is assumed to be on-resonance by default.

Defining the function basis. Instead of optimizing the pulse directly at each time step, we describe the control fields as a combination of analytical functions. These basis functions can be generated with the following commands:

```
NCoeffs = 3                        # Number of coefficients ax and ay
tau = np.linspace(0, 1, p.Nt)   # Normalized time: shape (Nt,)
fx = np.zeros((NCoeffs,p.Nt))
fy = np.zeros((NCoeffs,p.Nt))
for n in range(NCoeffs):
    fx[n,:] = np.sin((2*n+1)*np.pi*tau)
    fy[n,:] = np.cos((2*n+1)*np.pi*tau)
```

To use this parametrization in the optimization, we add the basis functions to the parameter structure:

```
p.Set(uxBasis = fx, uyBasis = fy)
```

With this setup, the code will no longer optimize u_x and u_y at each time step. Instead, it will only optimize the coefficients a_{x_k} and a_{y_k}.

Initial pulse guess. The pulse guess is now a guess for the coefficients a_{x_k} and a_{y_k}. While the function `InitPulse` is also adapted in this context, in this example, we use:

```
ax0 = np.random.randn(NCoeffs)
ay0 = np.random.randn(NCoeffs)
```

Optimization. The pulse optimization is carried out with the class `grape`, which supports several options documented in its header. In this example, we use:

```
gopt=Grape(Maxiter=500)
```

The pulse optimization is obtained via the command:

```
ax, ay, J = gopt.Optimize(p,ax0, ay0)
```

which returns the optimal coefficients a_{x_k} and a_{y_k} and the value of the cost function `J`.

Results. The pulse component $u_x(t)$ and $u_y(t)$ can be obtained with the command

```
ux = p.TruxBasis @ ax
uy = p.TruyBasis @ ay
```

Since the optimal controls u_x and u_y are normalized, the physical pulse components (in Hz) are given by $\nu_{\text{Ref}}u_x$ and $\nu_{\text{Ref}}u_y$. They can be visualized with the command:

```
plt.figure()
plt.plot(p.tc,p.nuRef*ux,p.tc,p.nuRef*uy)
plt.show()
```

The complete script is provided in `Example03_AnalyticallyShapedPulse.ipynb`.

Example 4: Robust Fourier pulse

`Example04_RobustFourierPulse.ipynb`

General problem. We consider a transmon system subject to both pulse amplitude deviations and detuning errors. Within the rotating-wave approximation, the system Hamiltonian is

$$H(\epsilon,\delta,t) = \frac{2\pi\nu_{\mathrm{Ref}}}{2}\,(1+\epsilon)\Big(u(t)\,a^\dagger + u^*(t)\,a\Big) + 2\pi(\nu_q - \nu_d + \delta)\,a^\dagger a + \frac{2\pi\alpha}{2}\,a^{\dagger 2}a^2,$$

where $\alpha = -200$ MHz, $\nu_{\mathrm{Ref}} = 10$ MHz, and $\nu_q = \nu_d = 5$ GHz. Here, ϵ (dimensionless) models a relative error in the drive amplitude, while δ (in Hz) represents a detuning error.

Our goal is to design a control pulse $u(t)$ that implements a $X_{\pi/2}$ gate:

$$U_{\mathrm{T}} = \tfrac{1}{\sqrt{2}}\begin{pmatrix} 1 & -i \\ -i & 1 \end{pmatrix}$$

on the computational subspace spanned by $|0\rangle$ and $|1\rangle$. To make the solution robust, we require that the same pulse achieves the target gate for all error values within the ranges

$$\epsilon \in [-0.1,\,0.1], \quad \delta \in [-0.2 \text{ MHz},\, 0.2 \text{ MHz}].$$

In practice, the error parameters ϵ and δ are discretized into N_ϵ and N_δ points within their respective ranges. The goal is then to find a pulse that minimizes the average infidelity over all these error values:

$$J = \frac{1}{N_\epsilon N_\delta}\sum_{n=1}^{N_\epsilon}\sum_{m=1}^{N_\delta}\left(1 - \tfrac{1}{2}\left|\,\mathrm{Tr}\left(U_T^\dagger U_{nm}^{01}(T)\right)\right|^2\right),$$

where $U_{nm}^{01}(T)$ is the evolution operator in the computational subspace $\{|0\rangle, |1\rangle\}$ associated to the pair (ϵ_n, δ_m). Since the computational subspace does not cover the full Hilbert space, $U_{nm}^{01}(T)$ may not be strictly unitary due to leakage to the $|2\rangle$ level.

Similarly to example 3, the pulse shape is restricted to a basis of analytical functions. Specifically, we use the parametrization

$$u_x(t) = \sum_{k=1}^{3} a_k \sin\!\Big(\tfrac{(2k-1)\pi t}{T}\Big), \quad u_y(t) = \sum_{k=1}^{3} b_k \sin\!\Big(\tfrac{2k\pi t}{T}\Big),$$

and search for coefficients $\{a_k, b_k\}$ minimizing the average infidelity J. In addition, we impose amplitude constraints on $u_x(t)$ and $u_y(t)$ of the form:

$$\nu_{\mathrm{Ref}}u_x(t) \le 15 \text{ MHz}, \quad \nu_{\mathrm{Ref}}u_y(t) \le 15 \text{ MHz}.$$

The control pulse is discretized into 200 time steps of 0.5 ns, resulting in a total duration of 100 ns.

Hamiltonian parameter grids. The error parameters δ and ϵ are discretized into 5 points each within their respective ranges using:

```
Na = 5  # Number of sampling points in the amplitude range
Nd = 5  # Number of sampling points in the detuning range
AmpError = np.linspace(-0.1,0.1,Na)   # Amplitude error [-10%, 10%]
Detuning = 0.2e6*np.linspace(-1,1,Nd) # Detuning [-0.2MHz, 0.2MHz]
```

corresponding to a total of 25 parameter sets $\{\epsilon_n, \delta_m\}$. These vectors are then combined to form a full grid of parameter pairs with:

```
DetMap, AmpMap = np.meshgrid(Detuning, AmpError)  # Make grid of inhomogeneities
AmpVec = np.reshape(AmpMap,(Na*Nd,))              # Reshape as vector (Na*Nb,)
DetVec = np.reshape(DetMap,(Na*Nd,))              # Reshape as vector (Na*Nb,)
```

This grid allows the pulse to be evaluated over all combinations of detuning and amplitude errors.

Problem parameters. This problem is specified by calling the function `ProblemParameters` with the following arguments:

```
p = ProblemParameters(
                alpha=-200e6,          # Anharmonicity parameter in Hz
                QubitFreq=5e9+DetVec,  # Qubit freq (+ detuning) in Hz
                CarrFreq = 5e9,        # Carrier frequency in Hz
                nuRef=5e6,             # Reference pulse amplitude in Hz
                AmpScale=1+AmpVec,     # Amplitude scale factor
                NLevels = 3,           # Number of energy levels
                Nt=200,                # Number of time steps
                dt=0.5e-9,             # Time step in s.
                CompSpace=[0,1],       # Computational states |0>, |1>
                Target =[[1,-1j],      # Target gate on comp. space
                         [-1j,1]]/np.sqrt(2)
                )
```

Note that the parameters `AmpScale` and `QubitFreq` accept arrays as input. This is also the case for `alpha`.

Defining the function basis. Instead of optimizing the pulse directly at each time step, we describe the control fields as a combination of analytical functions. These basis functions can be generated with the following commands:

```
Ncoeffs = 5
fx = np.zeros((Ncoeffs,p.Nt))
fy = np.zeros((Ncoeffs,p.Nt))
tau=np.linspace(0,1,p.Nt)
for n in range(Ncoeffs):
    fx[n,:] = np.sin((2*n+1)*np.pi*tau) # Harmonics of ux
    fy[n,:] = np.sin(2*(n+1)*np.pi*tau) # Harmonics of uy
```

To use this parametrization in the optimization, we add the basis functions to the parameter structure:

```
p.Set(uxBasis = fx, uyBasis = fy)  # Add to the list of parameters
```

With this setup, the code will no longer optimize u_x and u_y at each time step. Instead, it will only optimize the coefficients a_k and b_k in $u_x = \sum a_n f_x(n,t)$ and $u_y = \sum b_n f_y(n,t)$.

Initial pulse guess. The pulse initialization is now a guess for the coefficients a_k and b_k. The utility function `InitPulse` is also adapted in this context. We use a Drag-like pulse initialization using the command:

```
a0, b0 = InitPulse(p,PulseType="CosDrag",ThetaTar=np.pi/2)
```

Optimization. The pulse optimization is carried out with the class `grape.py`, which supports several options documented in its header. In this example, we need to account for the constraints on u_x and u_y. To do so, we use the command:

```
gopt=Grape(
        Maxiter=2000,           # Maximum number of iterations
        MaxInPhase = 15e6,      # Maximum In-phase amplitude allowed
        MaxQuadrature = 15e6    # Maximum quadrature amplitude allowed
        )
```

The pulse optimization is obtained via the command:

```
a, b, J = gopt.Optimize(p, a0, b0)
```

which returns the optimal coefficients a_k and b_k and the value of the cost function `J`.

Results. The pulse component $u_x(t)$ and $u_y(t)$ can be obtained with the command

```
ux = p.TruxBasis @ a
uy = p.TruyBasis @ b
```

which performs the operations $u_x = \sum a_n f_x(n,t)$ and $u_y = \sum b_n f_y(n,t)$. Since the optimal controls u_x and u_y are normalized, the physical pulse components (in Hz) are given by $\nu_{\mathrm{Ref}} u_x$ and $\nu_{\mathrm{Ref}} u_y$. They can be visualized with the command:

```
plt.figure()
plt.plot(p.tc,p.nuRef*ux,p.tc,p.nuRef*uy)
plt.show()
```

The complete script is provided in `Example04_RobustFourierPulse.ipynb`.

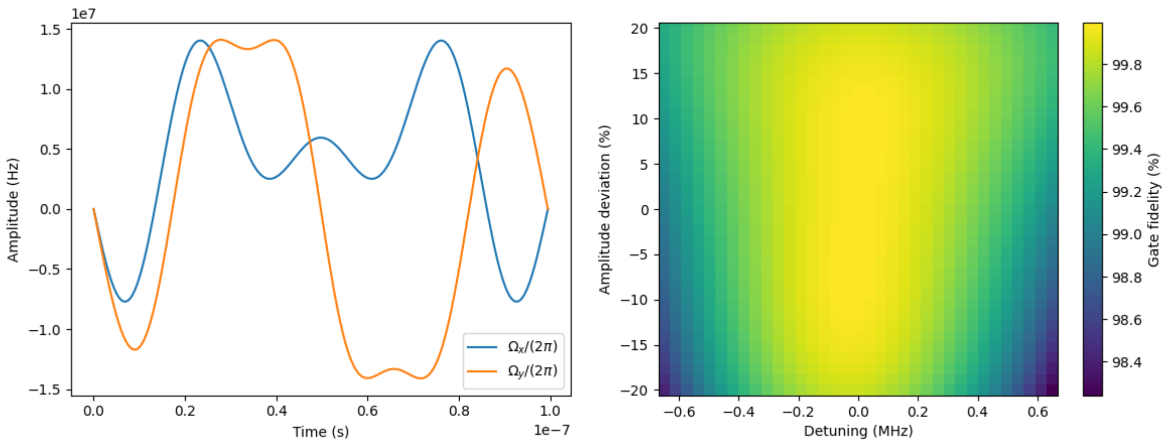

Figure 3: Left: Pulse amplitudes (in Hz) implementing a robust $X_{\pi/2}$ gate. Right: Gate fidelity as a function of detuning (δ) and amplitude error (ϵ).

Example 5: Selective pulse

`Example05_SelectivePulse.ipynb`

General problem. We consider a two-level quantum system with dynamics governed by the Hamiltonian

$$H(\delta,t) = \frac{2\pi\nu_{\text{Ref}}}{2}\Big(u(t)\,a^\dagger + u^*(t)\,a\Big) + 2\pi(\nu_q - \nu_d + \delta)\,a^\dagger a + \frac{2\pi\alpha}{2}\,a^{\dagger 2}a^2,$$

where $\alpha = -200$ MHz, $\nu_{\text{Ref}} = 5$ MHz, and $\nu_q = \nu_d = 5$ GHz. The detuning δ can take one of two values: ± 2.5 MHz. Since we are considering only the two lowest levels of the system, the ladder operators are truncated as

$$a = |0\rangle\langle 1|\,, \quad a^\dagger = |1\rangle\langle 0|\,.$$

Our goal is to design a control pulse $u(t)$ that implements the following:

$$\begin{cases} \text{Transfer } \; |0\rangle \; \text{ to } \; |1\rangle \; \text{ if } \delta = -2.5 \text{ MHz} \\ \text{Return } \; |0\rangle \; \text{ to } \; |0\rangle \; \text{ if } \delta = +2.5 \text{ MHz.} \end{cases}$$

In addition, we impose amplitude constraints on $u(t)$ such that:

$$|u(t)|\nu_{\text{Ref}} \leq 8 \text{ MHz.}$$

The control pulse is discretized into 120 time steps of 1 ns, resulting in a total duration of 120 ns.

Problem parameters. The code can handle multiple targets, provided that the number of targets matches the number of parameter sets. In this example, we consider two parameter sets corresponding to detunings $\delta_1 = -2.5$ MHz and $\delta_2 = +2.5$ MHz. Accordingly, we define two target states, which must be concatenateds when passed to `ParamExperiment`. The problem setup is as follows:

```
p = ProblemParameters(
        QubitFreq = 5e9 + np.array([-2.5e6, 2.5e6]),  # QubitFreq in Hz
        CarrFreq = 5e9,         # Carrier frequency in Hz
        NLevels = 2,            # Number of energy levels (ideal qubits)
        dt = 1e-9,              # Timestep in s.
        Nt = 120,               # Number of time steps
        CompSpace = [0,1],      # Indices of the computational states
        Psi0 = [1,0, 1,0],      # Initial states: [|0>, |0>]
        Target = [1,0, 0,1]     # Target states:  [|1>, |0>]
        )
```

Note that the parameter `alpha` does not need to be specified, since we are restricting the system to two levels.

Initial pulse guess. We initialize the pulse with a constant control field applied along the x-axis:

```
ux0 = np.ones(p.Nt)
uy0 = np.zeros(p.Nt)
```

Optimization. The pulse optimization is carried out with the class `grape`, which supports several options documented in its header. In this example, we need to account for the constraints on u_x and u_y. To do so, we use the command:

```
gopt=Grape(
        Maxiter=2000,          # Maximum number of iterations
        MaxAmplitude=8e6,      # Maximum amplitude allowed in Hz
        nGrad=3                # Improve gradient accuracy
        )
```

The pulse optimization is obtained via the command:

```
ux, uy, J = gopt.Optimize(p,ux0, uy0)
```

which returns the optimal pulses `ux` and `uy` and the value of the cost function `J`.

Results. Since the optimal controls u_x and u_y are normalized, the physical pulse components (in Hz) are given by $\nu_{\text{Ref}} u_x$ and $\nu_{\text{Ref}} u_y$. They can be visualized with the command:

```
plt.figure()
plt.plot(p.tc,p.nuRef*ux,p.tc,p.nuRef*uy)
plt.show()
```

The resulting pulse resembles a Ramsey experiment, with the pulse switched off between t = 32 ns and t = 88 ns. This is shown in Fig. 4. Additional functions are available for analyzing results. The complete script is provided in `Example05_SelectivePulse.ipynb`.

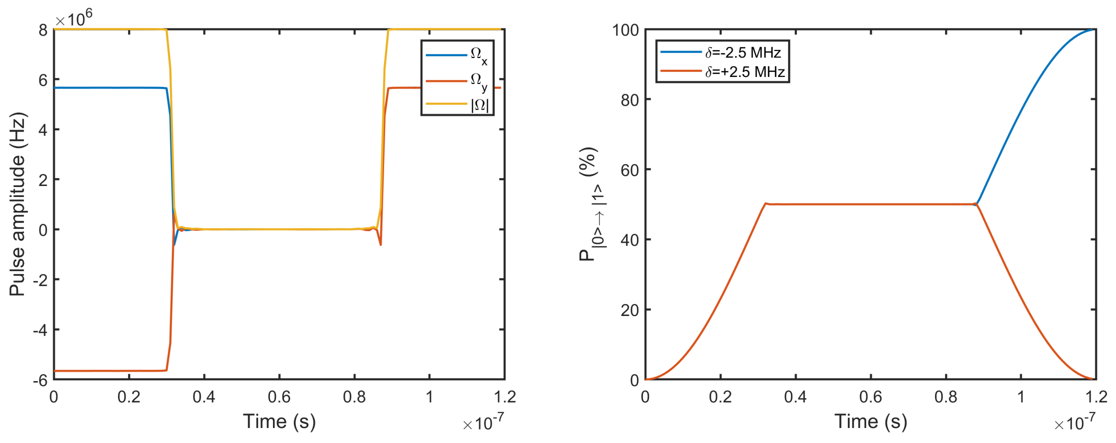

Figure 4: Left panel: Optimal pulse shape in Hz. Right panel: Transition probability as a function of time.