# Time 4: IFRS-CXS

## Solução da questão 3:

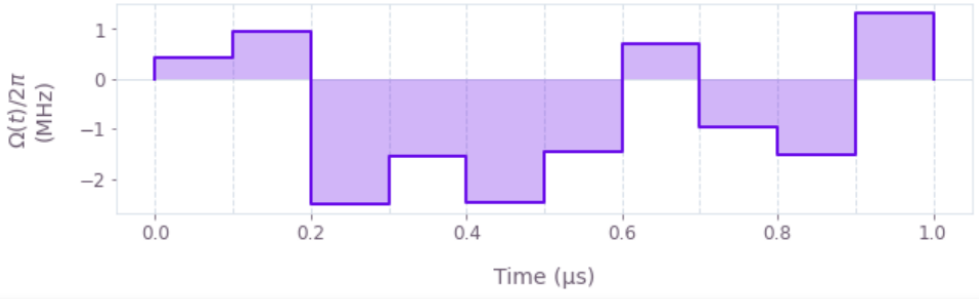
Para resolver este problema nós consideramos a rotina de otimização *closed-loop,* seguindo a explicação do notebook disponível no link. No nosso código, criamos a variável **U_t** que recebe qual porta (controle) escolhemos para rodar a otimização, ver figura abaixo:

**Aqui escolhe o controle U_t = sigma_x ou h**

In [4]:
```
# Define standard matrices.
sigma_x = np.array([[0, 1], [1, 0]], dtype=complex)
sigma_y = np.array([[0, -1j], [1j, 0]], dtype=complex)
sigma_z = np.array([[1, 0], [0, -1]], dtype=complex)
h = np.array([[1/np.sqrt(2), 1/np.sqrt(2)], [1/np.sqrt(2), -1/np.sqrt(2)]], dtype=complex)

# Gate (controle) de interesse
U_t = h

# Define control parameters.
duration = 1e-6  # s

# Create a random unknown operator.
rng = np.random.default_rng(seed=10)
phi = rng.uniform(-np.pi, np.pi)
u = rng.uniform(-1, 1)
Q_unknown = (
    u * sigma_z + np.sqrt(1 - u ** 2) * (np.cos(phi) * sigma_x + np.sin(phi) * sigma_y)
) / 4
```

Ao rodar a rotina descrita no notebook do closed-loop, obtemos os valores do pulso de controle. Assim, criamos um dicionário **control_x** e **control_h** para guardar estes valores e durações. Veja abaixo o resultado que obtemos para a porta H (controle h).

```
Your task calculate_graph (action_id="708981") has completed.
Infidelity: 0.01314502572325854
```




```
controls_h = []
for k in range(control_count):
    # Create a random string of complex numbers for each controls.
    #real_part = np.random.random(size=[segment_count])
    #imag_part = np.random.random(size=[segment_count])
    values = controls/10**(9)

    controls_h.append({"duration": duration, "values": values})
```

A partir daí, com o pulso de controle otimizado definido, começamos a usá-lo para testar no qubit na nuvem:

**Começa a usar o pulso para controle**

```
# Obtain the results of the experiment.
experiment_results = qctrl.functions.calculate_hackathon_measurements(
    controls=controls_h, shot_count=shot_count
)
```

```
Your task calculate_hackathon_measurements (action_id="708984") has completed.
```

```
measurements = experiment_results.measurements
for k, measurement_counts in enumerate(measurements):
    print(f"control #{k}: {measurement_counts}")
```

```
control #0: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
control #1: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
control #2: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
control #3: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0,
control #4: [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

Fizemos várias repetições, tal como exemplificado no notebook de enunciado da questão 3, para o caso do controle **U_t = sigma_x** e **U_t = h** (definimos no código a porta H como pode ser visto na primeira figura desta explicação). Por fim, os resultados foram arquivados num arquivo **JSON** compartilhado no repositório do time.