Nota 6: Descomposición *CS* de matrices

La descomposición *CS* (coseno-seno, [2]) constituye la base algebraica de un método de síntesis de operadores unitarios utilizando operadores de rotación $R_y(\theta)$ y $R_z(\theta)$, y operadores *CNOT*.

1. Descomposición *CS*

Considérese una matriz M, unitaria, de tamaño $n \times n$, con n par. Se pueden calcular matrices U_0, U_1, C, S, V_0, V_1, unitarias, de tamaño $n/2 \times n/2$, tales que

$$M = \begin{bmatrix} U_0 & 0 \\ 0 & U_1 \end{bmatrix} \begin{bmatrix} C & -S \\ S & C \end{bmatrix} \begin{bmatrix} V_0 & 0 \\ 0 & V_1 \end{bmatrix}^+, \tag{1}$$

donde C y S son matrices diagonales cuyos coeficientes son $\cos\theta_0$, $\cos\theta_1$, … , $\cos\theta_{n/2}$, en el caso de C, y $\sin\theta_0$, $\sin\theta_1$, … , $\sin\theta_{n/2}$, en el caso de S, siendo θ_0, θ_1, …, $\theta_{n/2}$ ángulos comprendidos entre 0 y $\pi/2$ radianes.

Ejemplo 1

Considérese la matriz

$$U = \begin{bmatrix} 0.5 & 0.5 & 0.5 & 0.5 \\ 0.5 & 0.5j & -0.5 & -0.5j \\ 0.5 & -0.5 & 0.5 & -0.5 \\ 0.5 & -0.5j & -0.5 & 0.5j \end{bmatrix}. \tag{2}$$

Es la matriz que define la transformada cuántica de Fourier de un registro de dos qubits. Para calcular la descomposición *CS* de U se utiliza el paquete `scipy` de Python. En [3] se describe la función `scipy.linalg.cossin`. El programa siguiente (`nota7_1.py`) calcula la descomposición (1) en el caso de la matriz (2):

```
import numpy as np
import scipy
U = np.array([
```

```
            [0.5, 0.5, 0.5, 0.5],
            [0.5, 0.5j, -0.5, -0.5j],
            [0.5, -0.5, 0.5, -0.5],
            [0.5, -0.5j, -0.5, 0.5j]
])
UCSVB = scipy.linalg.cossin(U,p=2,q=2,separate = True,compute_u = True,
compute_vh = True)
U0 = UCSVB[0][0]
U1 = UCSVB[0][1]
THETA = UCSVB[1]
cos_theta0 = np.cos(THETA[0])
cos_theta1 = np.cos(THETA[1])
sin_theta0 = np.sin(THETA[0])
sin_theta1 = np.sin(THETA[1])
V0adj = UCSVB[2][0]
V1adj = UCSVB[2][1]
print("U0 = ", U0, "\n")
print("U1 = ", U1, "\n")
print("THETA = ", THETA, "\n")
print("cos_theta0 = ", cos_theta0)
print("cos_theta1 = ", cos_theta1)
print("sin_theta0 = ", sin_theta0)
print("sin_theta1 = ", sin_theta1, "\n")
print("V0+ = ", V0adj, "\n")
print("V1+ = ", V1adj, "\n")
```

Resultado:

```
U0 =  [[0.65328148-0.27059805j 0.27059805+0.65328148j]
 [0.65328148+0.27059805j 0.27059805-0.65328148j]]

U1 =  [[0.27059805+0.65328148j 0.65328148-0.27059805j]
 [0.27059805-0.65328148j 0.65328148+0.27059805j]]

THETA =  [0.39269908 1.17809725]

cos_theta0 =  0.9238795325112867
cos_theta1 =  0.38268343236508984
sin_theta0 =  0.3826834323650898
sin_theta1 =  0.9238795325112867

V0+ =  [[ 0.70710678-0.j   0.5       +0.5j]
 [ 0.70710678+0.j  -0.5       -0.5j]]

V1+ =  [[ 6.26858359e-17-0.70710678j -5.00000000e-01+0.5j       ]
 [-6.26858359e-17+0.70710678j -5.00000000e-01+0.5j       ]]
```

Obsérvese que $\theta_0 = 0.39\ldots = \pi/8$ y $\theta_1 = 1.17\ldots = 3\pi/8$. Para comprobar el resultado, se ejecuta la misma función, con la opción `separate = False`, en cuyo caso se generan directamente las tres matrices de la relación (1) en lugar de las submatrices U_0, U_1, V_0^+, V_1^+, y de los ángulos θ_0 y θ_1, por separado:

```
UCSVB = scipy.linalg.cossin(U,p=2,q=2,separate = False,compute_u =
True, compute_vh = True)
U0U1 = UCSVB[0]
CS = UCSVB[1]
V0V1adj = UCSVB[2]
XX = (U0U1@CS)@V0V1adj
print("U0U1·CS·V0V1+ = ", XX)
```

Este es el resultado:

```
U0U1·CS·V0V1+ =
[[ 5.00000000e-01+1.22519038e-16j  5.00000000e-01+0.00000000e+00j
   5.00000000e-01+1.14700783e-16j  5.00000000e-01+1.38777878e-16j]
 [ 5.00000000e-01-1.46907298e-16j  1.38777878e-16+5.00000000e-01j
  -5.00000000e-01+1.04554246e-16j -2.77555756e-17-5.00000000e-01j]
 [ 5.00000000e-01+2.77555756e-17j -5.00000000e-01+0.00000000e+00j
   5.00000000e-01+7.20811553e-17j -5.00000000e-01+1.94289029e-16j]
 [ 5.00000000e-01-1.14967359e-17j -5.55111512e-17-5.00000000e-01j
  -5.00000000e-01-1.65700040e-17j  0.00000000e+00+5.00000000e-01j]]
```

Se observa que es la matriz (2), con pequeñas imprecisiones ($10^{-16} \cong 0$).

2. Operadores cuánticos

La descomposición *CS* sugiere la definición de dos tipos de operadores cuánticos.

2.1 Multiplexor cuántico 2 a 1

Considérese un registro de n qubits (q_{n-1}, q_{n-2}, …, q_0) y dos operadores unitarios U_0 y U_1 que actúan sobre n-1 qubit. Defínase un operador $M(U_0, U_1)$ que ejecuta la siguiente operación sobre el registro de n qubits ($N = 2^n$):

$$M(U_0, U_1)\left(\sum_{k=0}^{N-1} a_k|k_{n-1}k_{n-2}\dots k_0\rangle\right) =$$

$$\sum_{k=0}^{N/2-1} a_k|0\rangle\times U_0|k_{n-2}\dots k_0\rangle + \sum_{k=0}^{N/2-1} a_k|1\rangle\times U_1|k_{n-2}\dots k_0\rangle. \qquad (3)$$

Según el estado del qubit q_{n-1}, este operador ejecuta la operación U_0 o U_1 sobre los otros n-1 qubits, sin modificar el estado del qubit q_{n-1}. Es el tipo de operador representado por la primera y la tercera de las tres matrices (1). En la Fig.1 se muestra el símbolo correspondiente.

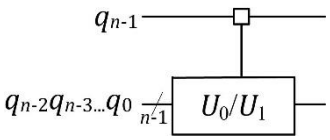

Figura 1 Multiplexor cuántico 2 a 1

2.2 Multiplexor cuántico 2^{n-1} a 1

Considérese un registro de n qubits (q_{n-1}, q_{n-2}, …, q_0) y 2^{n-1} operadores unarios unitarios U_0, U_1, …, $U_{N/2-1}$. Defínase un operador $M(U_0, U_1, \ldots, U_{N/2-1})$ que ejecuta la siguiente operación sobre el registro de n qubits ($N = 2^n$):

$$M(U_0, U_1, \ldots, U_{N/2-1})\left(\sum_{k=0}^{N-1} a_k|k_{n-1}k_{n-2}\ldots k_0\rangle\right) =$$

$$\sum_{k=0}^{N-1} a_k U_{k_{n-2}\ldots k_0}|k_{n-1}\rangle\times|k_{n-2}\ldots k_0\rangle. \quad (4)$$

Este operador ejecuta, en función del estado de los qubits q_{n-2}, q_{n-3}, …, q_0, una de las operaciones U_0, U_1, …, o $U_{N/2-1}$ sobre el qubit q_{n-1}, sin modificar el estado de los otros qubits. En la Fig.2 se muestra el símbolo correspondiente.

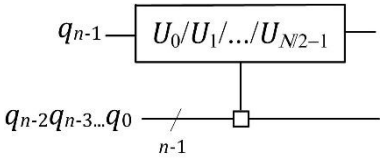

Figura 2 Multiplexor cuántico 2^{n-1} a 1

La segunda de las tres matrices (1) define un Multiplexor cuántico 2^{n-1} a 1. En este caso, son **rotaciones multiplexadas** alrededor del eje $0y$ (Fig.3.a) ya que, según la definición de la descomposición *CS*,

$$U_k = \begin{bmatrix} c_{kk} & -s_{kk} \\ s_{kk} & c_{kk} \end{bmatrix} = \begin{bmatrix} \cos\theta_k & -\sin\theta_k \\ \sin\theta_k & \cos\theta_k \end{bmatrix} = R_y(2\theta_k),\ \forall k = 0, 1, \ldots, N/2 - 1. \quad (5)$$

Otro ejemplo de rotación multiplexada, en este caso alrededor del eje $0z$, es el operador unitario

$$U = \begin{bmatrix} D & 0 \\ 0 & D^{+} \end{bmatrix} \tag{6}$$

donde D es una submatriz diagonal. De (6) y de la propiedad 1.11 de [1] se deduce que

$$U(\sum_{k=0}^{N-1} a_k |k_{n-1} k_{n-2} \dots k_0\rangle) =$$

$$\sum_{k=0}^{N/2-1} a_k e^{i\varphi_k} |0\rangle \times |k_{n-2} \dots k_0\rangle + \sum_{k=0}^{N/2-1} a_k e^{-i\varphi_k} |1\rangle \times |k_{n-2} \dots k_0\rangle =$$

$$\sum_{k=0}^{N-1} a_k U_{k_{n-2} \dots k_0} |k_{n-1}\rangle \times |k_{n-2} \dots k_0\rangle \tag{7}$$

donde

$$U_k = \begin{bmatrix} e^{i\varphi_k} & 0 \\ 0 & e^{-i\varphi_k} \end{bmatrix} = R_z(-2\varphi_k),\ \forall k = 0, 1, \dots, N/2 - 1. \tag{8}$$

El símbolo se muestra en la Fig.3.b.

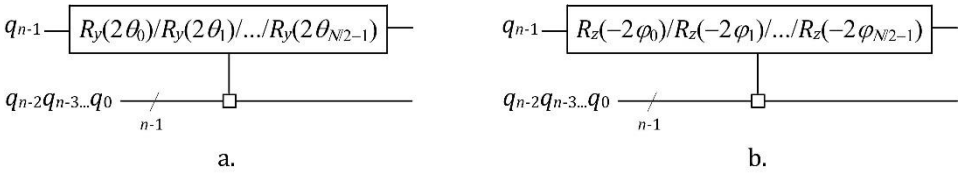

Figura 3 Rotaciones multiplexadas

3. Síntesis

De la descomposición *CS* (1) y de las definiciones de la Sec.2 se deduce el circuito de la Fig.4 compuesto de dos multiplexores 2 a 1 (Sec.2.1) y de $N/2$ rotaciones R_y multiplexadas (Sec.2.2).

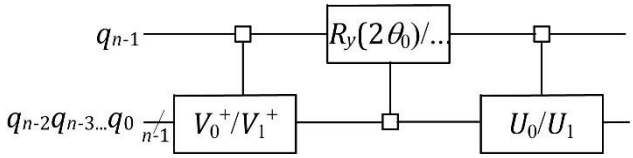

Figura 4 Descomposición *CS*

3.1 Multiplexor 2 a 1

Considérese el operador unitario definido por la matriz

$$U = \begin{bmatrix} U_0 & 0 \\ 0 & U_1 \end{bmatrix}. \tag{9}$$

La matriz unitaria $U_0 U_1^+$ puede descomponerse como sigue:

$$U_0 U_1^+ = V \Delta V^+, \tag{10}$$

donde Δ es la matriz diagonal compuesta de los valores propios de la matriz unitaria $U_0 U_1^+$, es decir, $e^{i\varphi_0}, e^{i\varphi_1}, \ldots, e^{i\varphi_{N/2-1}}$. Defínanse

$$D = \Delta^{0,5}, \tag{11}$$

es decir, la matriz diagonal compuesta de las raíces cuadradas $e^{i\varphi_0/2}, e^{i\varphi_1/2}, \ldots, e^{i\varphi_{N/2-1}/2}$ de los valores propios de $U_0 U_1^+$, y

$$W = DV^+ U_1. \tag{12}$$

Entonces, según (10), (11) y (12)

$$VDW = VD^2V^+U_1 = V\Delta V^+U_1 = U_0 U_1^+ U_1 = U_0,$$

$$VD^+W = V\,D^+DV^+U_1 = U_1,$$

con lo cual,

$$U = \begin{bmatrix} U_0 & 0 \\ 0 & U_1 \end{bmatrix} = \begin{bmatrix} V & 0 \\ 0 & V \end{bmatrix} \begin{bmatrix} D & 0 \\ 0 & D^+ \end{bmatrix} \begin{bmatrix} W & 0 \\ 0 & W \end{bmatrix}, \tag{13}$$

donde D es una matriz diagonal cuyos coeficientes son $e^{i\varphi_0/2}, e^{i\varphi_1/2}, \ldots$. Tal como se vio antes (relaciones 6 a 8 y Fig.3.b), a la matriz central de la descomposición (13) corresponde el circuito de la Fig.5 compuesto de dos operadores unitarios sobre n-1 qubits y de $N/2$ rotaciones multiplexadas $R_z(-2\varphi_k/2) = R_z(-\varphi_k)$.

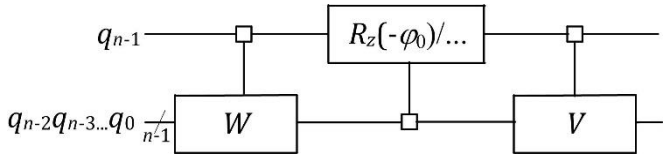

Figura 5 Multiplexor 2 a 1

Ejemplo 2

Aplíquese la descomposición (13) a la matriz compuesta de las submatrices U_0 y U_1 del ejemplo 1. En el programa siguiente (`nota7_2.py`) se utiliza el método `diagonalize()`, incluido en el paquete `scipy` [4], para la diagonalización (10) de la matriz $U_0 U_1^+$.

```
import numpy as np
import scipy
U = …
])
UCSVB = scipy.linalg.cossin(U,p=2,q=2,separate = True,compute_u = True,
compute_vh = True)
U0 = UCSVB[0][0]
U1 = UCSVB[0][1]
U1_adj = U1.conjugate().transpose()
U0_U1 = U0@U1_adj
###DIAGONALIZE WITH SYMPY
from sympy import *
V, Delta = Matrix(U0_U1).diagonalize()
print("V = ", V, "\n")
print("Delta = ", Delta, "\n")
```

Se obtienen las matrices V y Δ siguientes:

```
V =  Matrix([[0.707106781186548 - 7.0646339395761e-17*I, 0.5 + 0.5*I],
[-0.5 + 0.5*I, 0.707106781186548 + 5.55111512312579e-17*I]])

Delta =  Matrix([[-1.0 + 4.79063375800168e-17*I, 0], [0, 1.0 -
3.0598285354172e-17*I]])
```

Teniendo en cuenta las pequeñas imprecisiones, se deduce que

$$V = \begin{bmatrix} \frac{\sqrt{2}}{2} & \frac{1+i}{2} \\ \frac{-1+i}{2} & \frac{\sqrt{2}}{2} \end{bmatrix}, \Delta = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}, \tag{14}$$

con lo cual

$$D = \Delta^{1/2} = \begin{bmatrix} i & 0 \\ 0 & 1 \end{bmatrix}. \tag{15}$$

Queda por calcular $W = DV^+U_1$:

```
W = D@V_adj@U1
print("W = ", W, "\n")
```

Se obtiene como resultado

```
W =
[[-0.65328148-0.27059805j  0.65328148+0.27059805j]
 [ 0.65328148-0.27059805j  0.65328148-0.27059805j]]
```

Teniendo en cuenta que

$\cos\pi/8+\sin\pi/8)/2 = 0.65328148$ y $(\cos\pi/8-\sin\pi/8)/2 = 0.27059805$,

$$W = \begin{bmatrix} -\frac{e^{-i\pi/8}+e^{3i\pi/8}}{2} & \frac{e^{-i\pi/8}+e^{3i\pi/8}}{2} \\ \frac{e^{i\pi/8}+e^{-3i\pi/8}}{2} & \frac{e^{i\pi/8}+e^{-3i\pi/8}}{2} \end{bmatrix}. \qquad (16)$$

Finalmente, se definen las matrices

```
VV = np.array([
              [(2**0.5)/2, 0.5+0.5j, 0, 0],
              [-0.5+0.5j,(2**0.5)/2, 0, 0],
              [0, 0, (2**0.5)/2, 0.5+0.5j],
              [0, 0, -0.5+0.5j,(2**0.5)/2]
])
DDadj = np.array([
              [1j, 0, 0, 0],
              [0, 1, 0, 0],
              [0, 0, -1j, 0],
              [0, 0, 0, 1]
])
a = (-np.exp(-np.pi*1j/8)-np.exp(3*np.pi*1j/8))/2
b = (np.exp(np.pi*1j/8)+np.exp(-3*np.pi*1j/8))/2
WW = ([
    [a, -a, 0, 0],
    [b, b, 0, 0],
    [0, 0, a, -a],
    [0, 0, b, b]
])
```

y se ejecutan las instrucciones

```
XX = VV@DDadj@WW
print("VV·DDadj·WW =", XX, "\n")
```

Este es el resultado :

```
VV´·DDadj·WW =
[[0.65328148-0.27059805j 0.27059805+0.65328148j 0. +0.j  0. +0.j]
 [0.65328148+0.27059805j 0.27059805-0.65328148j 0. +0.j  0. +0.j]
 [0. +0.j  0. +0.j 0.27059805+0.65328148j 0.65328148-0.27059805j]
 [0. +0.j  0. +0.j 0.27059805-0.65328148j 0.65328148+0.27059805j]]
```

Se observa que es la matriz compuesta de las submatrices U_0 y U_1 del ejemplo 1.

De la descomposición *CS* y de las figuras 4 y 5 se deduce que cualquier operador unitario sobre *n* qubits puede ejecutarse según el esquema de la Fig.6, es decir, con operadores unitarios sobre *n*-1 qubits y rotaciones multiplexadas, alrededor de los ejes 0*y* y 0*z*, controladas por *n*-1 qubits. Una aplicación iterativa de esta descomposición lleva a la conclusión de que cualquier operador unitario puede ejecutarse con operadores unitarios sobre un qubit y con rotaciones multiplexadas.

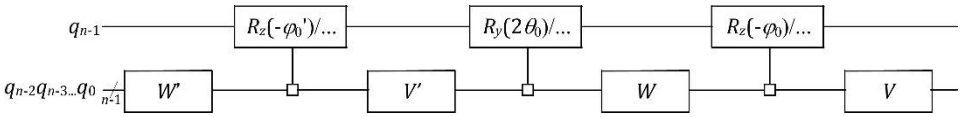

Figura 6 Operador unitario sobre *n* qubits

Obsérvese que el esquema de la Fig.6 se puede interpretar como una generalización del teorema 6.1 de [1].

3.2 Rotaciones multiplexadas

Considérese primero una rotación multiplexada, alrededor del eje 0*y* o 0*z*, controlada por un solo qubit (Fig.7.a en el caso del eje 0*y*). Según el estado del qubit de control q_0, se ejecuta una rotación de θ_0 o de θ_1 radianes del estado de qubit objetivo q_1. Gracias a las relaciones

$$XR_y(\theta)X = R_y(-\theta),\ XR_z(\theta)X = R_y(-\theta), \qquad (17)$$

se deduce que esta operación es la que ejecuta el circuito de la Fig.7.b (en el caso del eje 0*y*). Si el qubit de control está en el estado $|0\rangle$, se ejecuta una rotación de

$$\frac{\theta_0+\theta_1}{2} + \frac{\theta_0-\theta_1}{2} = \theta_0 \text{ radianes,}$$

y si el qubit de control está en el estado $|1\rangle$, se ejecuta una rotación de

$$\frac{\theta_0+\theta_1}{2} - \frac{\theta_0-\theta_1}{2} = \theta_1 \text{ radianes.}$$

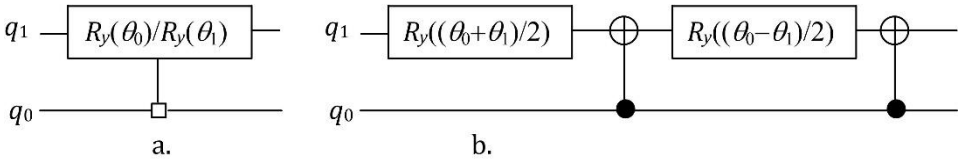

Figura 7 Rotación multiplexada controlada por un qubit

Las conclusiones son similares en el caso del eje 0*z*.

Esta construcción se puede generalizar. En le figura 8 se muestra como sintetizar una rotación multiplexada, alrededor del eje 0*y*, controlada por dos qubits. Por ejemplo, si $q_1q_0 = 00$ se ejecuta

$$R_y\left(\frac{\theta_0+\theta_1}{2}+\frac{\theta_0-\theta_1}{2}\right)=R_y(\theta_0),$$

y si $q_1q_0 = 11$ se ejecuta

$$R_y\left(\frac{\theta_2+\theta_3}{2}-\frac{\theta_2-\theta_3}{2}\right)=R_y(\theta_3).$$

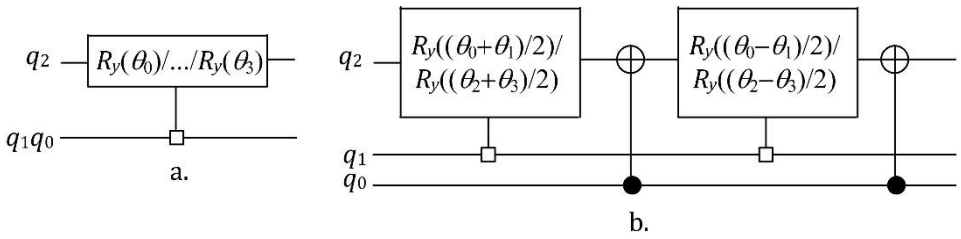

Figura 8 Rotación multiplexada controlada por dos qubits

Las conclusiones son similares en el caso del eje 0*z*.

De forma general, una rotación multiplexada, alrededor del eje 0*y* o 0*z*, controlada por *n*-1 qubits, puede ejecutarse según el esquema de la Fig.9 (en el caso del eje 0*y*), es decir, con rotaciones multiplexadas controladas por *n*-2 qubits y puertas *CNOT*.

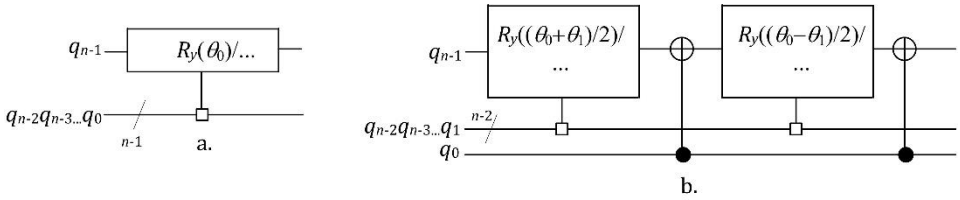

Figura 9 Rotación multiplexada controlada por n-1 qubits

En conclusión, aplicando iterativamente las construcciones de las figuras 6 y 9, se obtiene un circuito compuesto de operadores unitarios sobre un qubit, de rotaciones $R_y(\theta)$ y $R_z(\theta)$, y de puertas *CNOT*. Como los operadores unarios pueden también descomponerse en rotaciones ([1], teorema 6.1), se concluye que cualquier operador unitario puede sintetizarse con rotaciones alrededor de los ejes 0y y 0z y con puertas *CNOT*.

Ejemplo3

Para sintetizar el operador U del ejemplo 1, se debe aplicar la descomposición (13) a la matriz compuesta de las submatrices V_0 y V_1, de la misma manera que se hizo para la matriz compuesta de las submatrices U_0 y U_1 (ejemplo 2). El resultado es el siguiente (`nota7_3.py`):

$$W' = \begin{bmatrix} \frac{1-i}{2} & \frac{\sqrt{2}}{2}i \\ \frac{-1-i}{2} & -\frac{\sqrt{2}}{2} \end{bmatrix}, V' = \begin{bmatrix} \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2}i \\ \frac{\sqrt{2}}{2}i & \frac{\sqrt{2}}{2} \end{bmatrix}. \quad (18)$$

Por tanto, el operador U del ejemplo 1 puede ser ejecutado por el circuito de la figura 10, aplicando las relaciones (5) y (8), y teniendo en cuenta que

$$\begin{bmatrix} i & 0 \\ 0 & -i \end{bmatrix} = R_z(-\pi), \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = R_z(0).$$

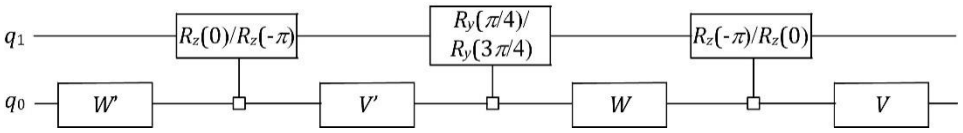

Figura 10 Esquema del circuito que ejecuta *U*

Los bloques *W'*, *V'*, *W* y *V* son operadores unitarios, definidos por las matrices (14), (15) y (18), que actúan sobre el segundo qubit. Utilizando el método de la Fig.7, para la síntesis de las rotaciones multiplexadas, se obtiene el circuito de la figura 11 descrito por el siguiente programa (`nota7_4.py`):

```
import cirq
import numpy as np
v = cirq.MatrixGate(np.array([
             [(2**0.5)/2, 0.5+0.5j],
             [-0.5+0.5j,(2**0.5)/2]
]))
a = (-np.exp(-np.pi*1j/8)-np.exp(3*np.pi*1j/8))/2
b = (np.exp(np.pi*1j/8)+np.exp(-3*np.pi*1j/8))/2
w = cirq.MatrixGate(np.array([
    [a, -a],
    [b, b]
]))
vv = cirq.MatrixGate(np.array([
             [(2**0.5)/2, ((2**0.5)/2)*1j],
             [((2**0.5)/2)*1j,(2**0.5)/2]
]))
ww = cirq.MatrixGate(
    np.array([
             [ 0.5-0.5j,((2**0.5)/2)*1j],
             [-0.5-0.5j,-(2**0.5)/2]
]))
q1,q0 = cirq.LineQubit.range(2)
U = cirq.Circuit()
U.append(ww(q0))
U.append([cirq.rz(-np.pi/2)(q1),
cirq.CX(q0,q1),cirq.rz(np.pi/2)(q1),cirq.CX(q0,q1)])
U.append(vv(q0))
U.append([cirq.ry(np.pi/2)(q1), cirq.CX(q0,q1),
   cirq.ry(-np.pi/4)(q1),cirq.CX(q0,q1)])
U.append(w(q0))
U.append([cirq.rz(-np.pi/2)(q1), cirq.CX(q0,q1),
   cirq.rz(-np.pi/2)(q1),cirq.CX(q0,q1)])
U.append(v(q0))
print(cirq.unitary(U))
```

Este es el resultado:

```
[[ 5.00000000e-01+2.77555756e-17j  5.00000000e-01+0.00000000e+00j
   5.00000000e-01+0.00000000e+00j  5.00000000e-01+5.55111512e-17j]
 [ 5.00000000e-01+0.00000000e+00j  5.55111512e-17+5.00000000e-01j
```

```
 -5.00000000e-01+2.77555756e-17j  0.00000000e+00-5.00000000e-01j]
[ 5.00000000e-01+2.77555756e-17j -5.00000000e-01+0.00000000e+00j
  5.00000000e-01+0.00000000e+00j -5.00000000e-01-5.55111512e-17j]
[ 5.00000000e-01+0.00000000e+00j -5.55111512e-17-5.00000000e-01j
 -5.00000000e-01+2.77555756e-17j  0.00000000e+00+5.00000000e-01j]]
```

Es la matriz (2), con pequeñas imprecisiones.

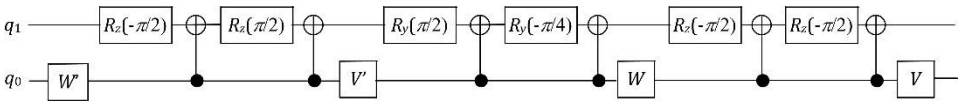

Figura 11 Circuito que ejecuta *U*

4. Comentarios

El uso de la transformada *CS* permite llegar a la misma conclusión que el uso de operadores de dos niveles ([1], Sec.6.2.1): cualquier operador unitario puede sintetizarse con rotaciones y con puertas *CNOT*. Sin embargo, en ninguno de los dos casos se garantiza que el circuito sintetizado sea óptimo. El operador *U* del ejemplo 1 puede sintetizarse con un circuito mucho más simple, compuesto de dos operadores *H*, una rotación controlada $CR_{\pi/2}$ y un operador *SWAP* ([1], Fig.7.6, `nota7_5.py`):

```
import cirq
import numpy as np
q1,q0 = cirq.LineQubit.range(2)
CRpi_2 = cirq.S.controlled()
U = cirq.Circuit()
U.append([cirq.H(q1), Crpi_2(q0,q1), cirq.H(q0)])
U.append(cirq.SWAP(q0,q1))
print(cirq.unitary(U))
```

Resultado:

```
[[ 0.5+0.j   0.5+0.j   0.5+0.j   0.5+0.j ]
 [ 0.5+0.j   0. +0.5j -0.5+0.j   0. -0.5j]
 [ 0.5+0.j  -0.5+0.j   0.5+0.j  -0.5+0.j ]
 [ 0.5+0.j   0. -0.5j -0.5+0.j   0. +0.5j]]
```

Lo que se ha podido demostrar, a partir de la descomposición *CS* (ver por ejemplo [5]), es que la complejidad máxima de un circuito que materializa un operador unitario sobre *n* qubits tiene un orden de magnitud proporcional a 4^n.

Referencias

[1] J.P.Deschamps, Computación Cuántica, Marcombo, Barcelona, 2023.

[2] Sutton, B.D. Computing the complete CS decomposition. Numer. Algor. 2009 50, 33–65.

[3] https://docs.scipy.org/doc/scipy/reference/generated/scipy.linalg.cossin.html

[4] https://docs.sympy.org/latest/tutorials/intro-tutorial/matrices.html

[5] Shende, V.; Bullock, S.; Markov, I. Synthesis of Quantum Logic Circuits. Comput.-Aided Des. Integr. Circuits Syst. IEEE Trans. 2006, 25, 1000–1010