HLL Algorithm in Qiskit

QComp

06/10/2025

Centrale-Supélec

Étienne BONNAND

Professeur référent : Benoît Valiron

Sommaire

4 Influence of parameters

4.1

Cette fois-ci, pour définir l'inverse, j'applique le code suivant 1. Ce qui change dans mon code, c'est bien sûr la valeur de `n`, mais également j'ajoute une condition sur la valeur de l'angle pour appliquer ou non un `RYGate`. Le `Vectorstate` est bien le même qu'auparavant 2 (Cela a augmenté uniquement la précision pour l'encodage de l'angle).

```
n = 5
q = QuantumRegister(n, name="q")
r = QuantumRegister(1, name="r")
qcinv = QuantumCircuit(q,r)

C = 1 / 4

for  i in range(1, 2**n):
    angle = C * 2**n / i
    if -1 <= angle <= 1:
        ry_gate = RYGate(2 * np.arcsin(angle))
        bin_string = nat2bs(n, i)
        controlled_ry = ry_gate.control(num_ctrl_qubits=n, ctrl_state=bin_string)
        qcinv.append(controlled_ry,  list(q) + [r[0]])

invCirc = qcinv.to_gate(label="inv")
```

Listing 1: Création du bloc Inverse

```
 0.43301 +  0.00000j |0000000>
 0.75000 + -0.00000j |0000001>
-0.43301 +  0.00000j |1000000>
 0.25000 + -0.00000j |1000001>
```

Listing 2: Statevector en sortie de HLL

4.2

Dans cette partie, je change C de $\frac{1}{4}$ à $\frac{1}{8}$ Les coefficients qui m'intéressent dans ce cas sont toujours ceux dont le qubit le plus à droite est 1 puisque cela signifie que l'inversion a effectivement eu lieu. Je peux calculer ces coeffients: 1 2. Après avoir lancé le code j'obtiens bien le statevector voulu 3.

$$\text{coef}_1 = \frac{2\pi C x_1}{t} = \frac{2\pi \times \frac{1}{8} \times \frac{9}{8}}{2\pi \times \frac{3}{8}} = 0,375 \tag{1}$$

$$\text{coef}_2 = \frac{2\pi C x_2}{t} = \frac{2\pi \times \frac{1}{8} \times \frac{3}{8}}{2\pi \times \frac{3}{8}} = 0,125 \tag{2}$$

```
 0.91714 + -0.00000j |0000000>
 0.37500 + -0.00000j |0000001>
-0.05111 + -0.00000j |1000000>
 0.12500 + -0.00000j |1000001>
```

Listing 3: Statevector en sortie de HLL avec $C = \frac{1}{8}$

Après avoir divisé t par 2, le statevector reste identique à 2 puisque diviser C et t par deux revient à ne rien changer dans la formule de calculs des coefficients. 3
Si je choisis $t = \frac{2\pi \times 3.2}{16}$, le statevector final est une superposition de tous les états possible sur n qubits. Je n'obtiens pas un résultat exact puisque pour QPE comme QFT, le résultat est exact si la phase recherchée est un multiple de $\frac{2\pi}{2^n}$ où n est le nombre de qubit. Ici, il n'y a aucune correspondance exacte. On va avoir donc un pic autour de la valeur de $|x_1\rangle$ et de $|x_2\rangle$. Le résultat final ce lit quand le qubit le plus à droite est égal à 1: signe que l'inversion s'est bien produite. Ensuite je cherche $|x_1\rangle$ et $|x_2\rangle$ qui sont réciproquement $x_1|0\rangle$, $x_2|1\rangle$. 4 En effet en effectuant le calcul : 3 4 le résultat est moins précis.

$$\text{coef}_1 = \frac{2\pi C x_1}{t} = \frac{2\pi \times \frac{1}{8} \times \frac{9}{8}}{2\pi \times \frac{3.2}{16}} \simeq 0.703 \tag{3}$$

$$\text{coef}_2 = \frac{2\pi C x_2}{t} = \frac{2\pi \times \frac{1}{8} \times \frac{3}{8}}{2\pi \times \frac{3.2}{16}} \simeq 0.234 \tag{4}$$

```
...
0.67909 + -0.00000j |0000001>
...
...
...
0.21617 + -0.00000j |1000001>
...
...
...
```

Listing 4: Statevector en sortie de HLL avec $t = \frac{2\pi \times 3.2}{16}$

4.3

Dans cette partie:

$$t = \frac{6\pi}{16} \quad A = \begin{pmatrix} 1 & -\frac{1}{2} \\ -\frac{1}{2} & 1 \end{pmatrix} \quad b = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

En résolvant le système,j'obtiens:

$$x_1 = \frac{4}{3} \quad x_2 = \frac{2}{3}$$

Les coefficients en sortie de HLL vont être :

$$\text{coef}_1 = \frac{2\pi C x_1}{t} = \frac{2\pi \times \frac{1}{8} \times \frac{4}{3}}{2\pi \times \frac{3}{16}} = \frac{8}{9} \tag{5}$$

$$\text{coef}_2 = \frac{2\pi C x_2}{t} = \frac{2\pi \times \frac{1}{8} \times \frac{2}{3}}{2\pi \times \frac{3}{16}} = \frac{4}{9} \tag{6}$$

J'obtient finalement: 5

```
0.94790 + -0.00000j |0000000>
0.22222 +  0.00000j |0000001>
0.05210 + -0.00000j |1000000>
-0.22222 +  0.00000j |1000001>
```

Listing 5: Statevector en sortie de HLL avec la deuxième matrice

Ce qui ne correspond pas du tout à ce que prédisent les calculs. Je pense que cela est dû à mon choix de C. Ici de nombreuse rotations n'ont pas lieu. Car *arcsin* n'est pas défini dans la majorité des cas. En prenant $C = \frac{1}{32}$ cette fois-ci les calculs donnent:

$$\text{coef}_1 = \frac{2\pi C x_1}{t} = \frac{2\pi \times \frac{1}{32} \times \frac{4}{3}}{2\pi \times \frac{3}{16}} = \frac{2}{9} \simeq 2.22 \tag{7}$$

$$\text{coef}_2 = \frac{2\pi C x_1}{t} = \frac{2\pi \times \frac{1}{32} \times \frac{4}{3}}{2\pi \times \frac{3}{16}} = \frac{1}{9} \simeq 1.11 \tag{8}$$

Et j'obtiens comme statevector: 6. Ce qui montre que le programme fonctionne bien. Pour diminuer l'erreur, augmenter C fonctionne bien car plus au augmente C, plus les valeurs de $\frac{C2^n}{i}$ 1 sont comprises entre -1 et 1 et donc davantage de rotations sont prises en compte.

```
0.96831 + -0.00000j |0000000>
0.22222 +  0.00000j |0000001>
-0.02550 +  0.00000j |1000000>
0.11111 + -0.00000j |1000001>
```

Listing 6: Statevector en sortie de HLL avec la deuxième matrice pour $C = \frac{1}{16}$

Pour pouvoir calculer l'erreur automatiquement je modifie le contenu de la fonction `runStateVector` 7. Ici je calcule l'erreur comme la norme deux de la différence entre la valeur théorique du coefficient et la valeur du coefficient en sortie de HLL. Pour un codage de l'angle sur 5 qubit je calcule l'erreur en fonction de C. 1 Au dessus de $C = \frac{1}{2^n}$ il n'y a pas d'évolution majeure de l'erreur, c'est normal puisqu'à partir de cette valeur de C tous les angles sont pris en compte.

```
def runStateVector(qc, keys=None):
    simulator = StatevectorSimulator(statevector_parallel_threshold=6)
    job = simulator.run(qc.decompose(reps=6), memory=True)
    job_result = job.result()
    result = job_result.results[0].to_dict()['data']['statevector']
    if keys is not None:
        reals = {k: result[int(k, 2)].real for k in keys}
        return reals
    else:
        printFinalRes(result, keys)
```

Listing 7: Modification de runStateVector pour calculer l'erreur

Un deuxième facteur qui influence bien plus sur l'erreur est la valeur de n, le nombre de qubit sur lequel est encodé l'angle. En effet, plus ce nombre est précis plus la rotation est précise et plus QPE est précis puisque la précision pour QTF est aussi augmentée. La figure 2 montre une diminution exponentielle de l'erreur en fonction du nombre de qubit pour la précision de l'angle.

4.4

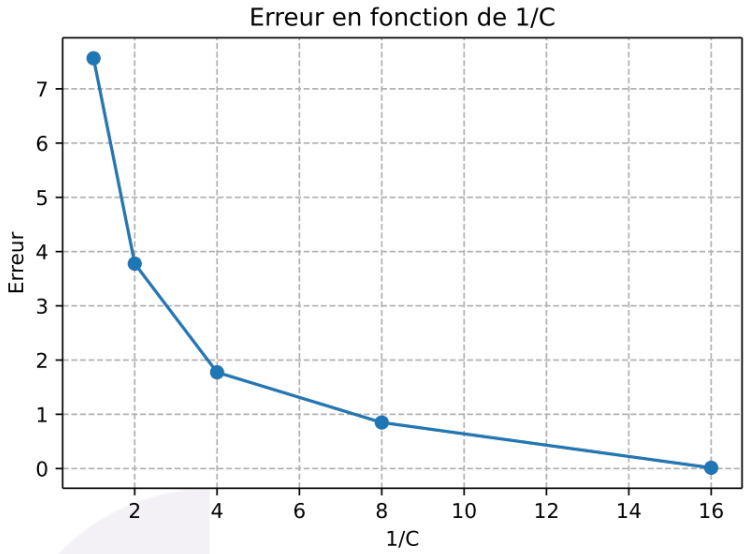

Figure 1: Erreur en fonction de l'inverse de C pour $n = 5$

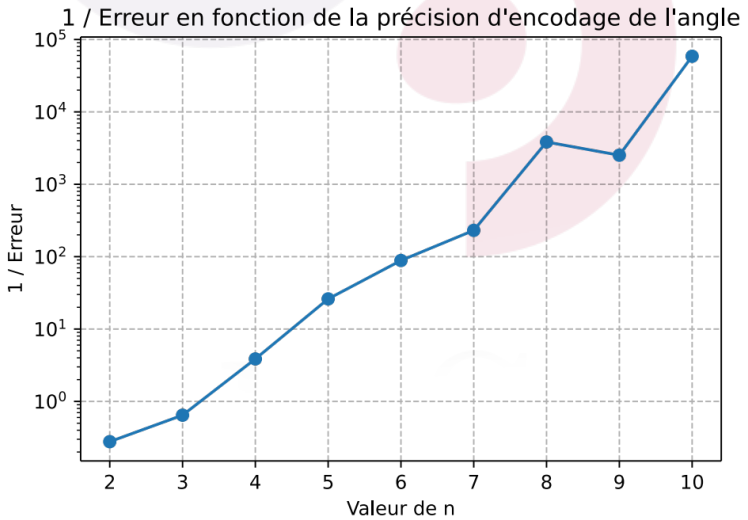

Figure 2: Inverse de l'erreur en fonction de n en échelle logarithmique avec $C = \frac{1}{2^n}$.

5 Handling more generals matrices

5.1

Pour pouvoir prendre en compte les valeurs propres négatives, j'ai adapté le code du bloc inverseur de la façon suivante: 8. J'obtiens bien les valeurs souhaitées pour les différents codes par la suite.

```
for  i in range(1, 2**n):
if i <= 2**(n - 1):
    i_angle = i
else:
    i_angle = i - 2**n
angle = C * 2**n / i_angle
if -1 <= angle <= 1:
    ry_gate = RYGate(2 * np.arcsin(angle))
    bin_string = nat2bs(n, i)
    controlled_ry = ry_gate.control(num_ctrl_qubits=n, ctrl_state=bin_string)
    qcinv.append(controlled_ry,  list(q) + [r[0]])
```

Listing 8: Modification du bloc Inverse pour gérer les valeurs propres négatives

5.2 Update HHL

Pour tester l'algorithme HLL sur le système: $A = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$, $\vec{b} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ -1 \end{pmatrix}$, il suffit de mettre à jour le code avec les nouveaux paramètres et d'implémenter le nouveau calcul de l'inverse présenté précédement. À cela ce rajoute l'implémentation de b effectuée de la sorte: 9

```
# Creating b
nb = 1
qb = QuantumRegister(nb, name="b")
qc.x(qb)
qc.h(qb)
```

Listing 9: Implémentation de b

5.3 Testing larger matrices A

Pour mettre à jour pour de grandes matrices, il faut augmenter la taille de b 10. Dans ce cas, aucune initialisation n'est nécessaire, puisque b est codé uniquement avec des qubits nuls.

```
nb = 3
qb = QuantumRegister(nb, name="b")
```

Listing 10: Implémentation de b

Le résultat obtenu est : 11. J'obtiens bien les valeurs voulue pour les coefficients.

```
Classical solution:
[ 0.16190476  0.20952381 -0.26666667 -0.07619048  0.3047619  -0.07619048
 -0.26666667  0.20952381]
Expected coefficients in the quantum solution (up to normalization):
[-0.26666667 -0.26666667 -0.07619048 -0.07619048  0.16190476  0.20952381
  0.20952381  0.3047619 ]
Results of quantum computation:
[-0.26666667 -0.26666667 -0.07611016 -0.07611016  0.16241454  0.2094435
  0.2094435   0.30425213]
```

```
L'erreur est :
0.0007386088922852989
```

Listing 11: Résutat pour une matrice plus grande