LCU-toymodel in first quantization.

- De LCU-decompositie van de 2×2 Hamiltoniaan
- De `PREP`-stap (superpositie van LCU-coëfficiënten)
- De `SELECT`-unitary die de juiste Pauli-operator toepast
- De block-encoding van de Hamiltoniaan
- De constructie van de walk-operator voor qubitization
- Quantum Phase Estimation (QPE) om het energieniveau van het systeem te bepalen

Overzicht en Hamiltoniaan in LCU-vorm

We willen de 1-qubit Hamiltoniaan $H = 1.5\,I + 0.5\,X + 0.5\,(-Z)$ block-encoden via *Linear Combination of Unitaries* (LCU). Dit betekent dat we H schrijven als lineaire combinatie van unitaire operatoren U_i met coëfficiënten ω_i. In dit geval hebben we drie termen: $U_0 = I$, $U_1 = X$ en $U_2 = Z$ (voor de $-Z$-term nemen we $U_2 = Z$ en verwerken het minteken apart). De gewichten zijn $\omega_0 = 1.5$, $\omega_1 = 0.5$ en $\omega_2 = -0.5$.

We bepalen eerst de normaleerfactor $\lambda = \sum_i |\omega_i| = 1.5 + 0.5 + 0.5 = 2.5$. Voor de LCU-constructie bereiden we een ancillatoestand met amplituden $\sqrt{\omega_i/\lambda}$ (voor negatieve ω_i nemen we de fase -1 mee in de selectie-operator). Hier wordt dus:

- $\sqrt{\omega_0/\lambda} = \sqrt{1.5/2.5} \approx 0.7746$,
- $\sqrt{\omega_1/\lambda} = \sqrt{0.5/2.5} \approx 0.4472$,
- $\sqrt{|\omega_2|/\lambda} = \sqrt{0.5/2.5} \approx 0.4472$ (de fase $-$ verwerken we apart).

De ancillaregister heeft $m = \lceil \log_2 3 \rceil = 2$ qubits nodig (vier basisstates, waarvan er één ongebruikt blijft). We zullen de amplituden verdelen over $|00\rangle$, $|01\rangle$, $|10\rangle$ overeenkomstig $\{\sqrt{0.6}, \sqrt{0.2}, \sqrt{0.2}\}$, en $|11\rangle$ krijgt amplitude 0.

PREP: Voorbereiden van de amplitude-superpositie

De eerste stap is een **PREP**-subcircuit dat de 2-qubit ancillaregister vanuit $|00\rangle$ naar de superpositie $ \sqrt{0.6},|00\rangle + \sqrt{0.2},|01\rangle + \sqrt{0.2},|10\rangle
In Qiskit kunnen we dit direct doen met `QuantumCircuit.prepare_state` voor de gewenste amplituden. Hieronder construeren we de PREP-gate:

```python
from qiskit import QuantumCircuit, QuantumRegister
import numpy as np

# Definieer gewichten en genormaliseerde amplituden
omega = [1.5, 0.5, -0.5]
lam = sum(abs(w) for w in omega)            # lambda = 2.5
alpha = [np.sqrt(abs(w)/lam) for w in omega]  # amplitudes voor |00>,|01>,|10>
alpha.append(0)                              # pad amplitude voor |11>
alpha = np.array(alpha, dtype=complex)
alpha = alpha / np.linalg.norm(alpha)      # normaliseer (zou al genormaliseerd moeten zijn)

# Bouw PREP subcircuit
anc = QuantumRegister(2, name="anc")
prep_circ = QuantumCircuit(anc, name="PREP")
prep_circ.prepare_state(alpha, anc)  # bereidt ancilla-toestand volgens alpha
prep_gate = prep_circ.to_gate(label="PREP")
```

Hier hebben we de ancillatoestand voorbereid met de gespecificeerde amplituden. (Qiskit's `prepare_state` decomponeert deze state in elementaire poortjes voor ons.)

SELECT: Geconditioneerde Pauli-operaties

De **SELECT**-stap brengt het systeemqubit in contact met de ancilla. Afhankelijk van de ancillatoestand $|i\rangle$ passen we de juiste unitaire U_i toe op het systeem Onze U's zijn: I, X en Z (waarbij Z bedoeld is voor de $-Z$ term). Om het minteken te verwerken definiëren we $U_2 = -Z$ als unitair (wat fysiek gelijk is aan Z op een globale fase na, maar die faseverschillen zijn essentieel voor interferentie in de block-encode). We construeren daarom $-Z$ als apart quantum-gate door Z te conjugeren met X-poorten (oftewel $-Z = XZX$):

```python
from qiskit.circuit.library import XGate, IGate

# Definieer één-qubit gates voor I, X en -Z
id_gate = IGate()                   # identiteitspoort (I)
x_gate = XGate()                    # Pauli-X poort
# Bouw -Z gate via conjugatie X Z X
negz_circ = QuantumCircuit(1, name="-Z")
negz_circ.x(0); negz_circ.z(0); negz_circ.x(0)
negz_gate = negz_circ.to_gate(label="-Z")
```

```python
# Maak lijst van unitaire gates [I, X, -Z]
U_list = [id_gate, x_gate, negz_gate]

# Bouw SELECT subcircuit: toepassen U_i afhankelijk van ancilla |i>
sys = QuantumRegister(1, name="sys")
select_circ = QuantumCircuit(anc, sys, name="SELECT")
m = anc.size  # =2 control qubits
for i, U_gate in enumerate(U_list):
    ctrl_state = format(i, f"0{m}b")      # binaire representatie met 2 bits
    controlled_U = U_gate.control(m, ctrl_state=ctrl_state)
    select_circ.append(controlled_U, anc[:] + sys[:])
select_gate = select_circ.to_gate(label="SELECT")
```

In het bovenstaande subcircuit gebruiken we multi-control volgens het binaire patroon van de ancilla. Concreet:

- Voor $i = 0$ (ctrl_state "00") wordt $U_0 = I$ toegepast als ancilla $|00\rangle$ is (dit laat het systeem ongewijzigd, zoals bedoeld).
- Voor $i = 1$ (ctrl_state "01") wordt X toegepast als ancilla $|01\rangle$ is.
- Voor $i = 2$ (ctrl_state "10") wordt onze $-Z$-gate toegepast als ancilla $|10\rangle$ is.

Qiskit's `.control(m, ctrl_state=...)` methode verzorgt automatisch de vereiste X-poorten op de controlequbits voor ctrl_state met nullen, zodat bijvoorbeeld ctrl_state "00" betekent dat beide ancilla-qubits 0 moeten zijn (in plaats van de standaard 1)

Block-encoding U = PREP → SELECT → PREP†

Nu construeren we het volledige block-encoding circuit **U** door de bovenstaande componenten te combineren. Dit circuit voert eerst PREP (prepare ancilla-superpositie) uit, vervolgens SELECT (geconditioneerde Pauli-operaties op het systeem), en ten slotte PREP† (uncompute ancilla) (). Het netto-effect is dat $\langle 0|_{\text{anc}} U |0\rangle_{\text{anc}} = H/\lambda$ – met andere woorden, in de subruimte waar de ancilla terugkeert naar $|00\rangle$, gedraagt U zich als de genormaliseerde Hamiltoniaan (

```python
# Combineer PREP, SELECT, PREP† tot block-encoding unitary U
blockenc_circ = QuantumCircuit(anc, sys, name="U_block")
blockenc_circ.append(prep_gate, anc[:])
blockenc_circ.append(select_gate, anc[:] + sys[:])
blockenc_circ.append(prep_gate.inverse(), anc[:])
U_block = blockenc_circ.to_gate(label="U")  # gate voor de gehele block-encode
```

We hebben nu een unitary `U_block` die onze 3×3 Hamiltoniaan in een 4×4 blok van de totale unitaire matrix bevat (embedded in een 8×8 unitary op 3 qubits).

Quantum Walk Operator W

De volgende stap is het construeren van de *quantum walk*-operator $W = (2|0\rangle\langle 0| - I) \cdot U$ ([Block Encoding in Quantum Computing | by Hier is $|0\rangle$ het projectiepunt van de ancilla op $|00\rangle$ (d.w.z. alle ancilla-qubits in de nultoestand). De operator $R = 2|00\rangle\langle 00| - I$ reflecteert de toestand ten opzichte van de $|00\rangle$-richting in het ancillaruim (. Samen met U resulteert $W = RU$ in een ge-Qubitiseerde (qubitized) unitary waarvan de eigenfasen θ gerelateerd zijn aan de eigenwaarden E van H via $\cos\theta = E/\lambda$ We implementeren R door alle ancilla-qubits te inverteren met X-poorten (zodat $|00\rangle$ wordt omgezet in $|11\rangle$) en vervolgens een multi-controlled-Z toe te passen die een fase -1 toevoegt aan $|11\rangle$. Daarna inverteren we de ancilla's terug. Dit is equivalent aan $2|00\rangle\langle 00| - I$ (op een globale fase na, wat geen invloed heeft) – alleen toestanden **niet** gelijk aan $|00\rangle$ krijgen een faseflip van -1). In twee ancilla-qubits is een CZ-poort tussen beide qubits voldoende in deze omgekeerde basis.

```
# Bouw het quantum walk operator W = R * U
W_circ = QuantumCircuit(anc, sys, name="W")
# Eerst U toepassen:
W_circ.append(U_block, anc[:] + sys[:])
# Vervolgens reflectie R op ancilla:
W_circ.x(anc[0]); W_circ.x(anc[1])
W_circ.cz(anc[0], anc[1])            # faseflip op |11> (komt overeen met |00> origineel)
W_circ.x(anc[0]); W_circ.x(anc[1])
W_gate = W_circ.to_gate(label="W")
```

Hier hebben we W als één gecombineerde gate gedefinieerd.

Quantum Phase Estimation met W

Tot slot voeren we **kwantumfase-schatting (QPE)** uit op W om de eigenfasen (en daarmee de energieniveaus) van H te extraheren. We nemen een fase-register van bijv. 3 qubits (voor 3-bit precisie) en bereiden deze in een superpositie via Hadamard-poorten. Het systeemregister (1 qubit) bereiden we in een starttoestand (bijvoorbeeld $|0\rangle$ of een eigenvector van H als die bekend is). Daarna laten we het fase-register controle-gewijs W toepassen in toenemende machtsfactoren: $1, 2, 4, \ldots$ (dit is de kern van fase-schatting (). We bouwen ook de inverse Quantum Fourier Transform om de gemeten fasebits om te zetten naar een leesbaar resultaat.

```python
from qiskit.circuit.library import QFT

# Maak registers voor QPE, ancilla, en systeem
qpe = QuantumRegister(3, name="qpe")
anc_final = QuantumRegister(2, name="anc")   # ancilla (2 qubits)
sys_final = QuantumRegister(1, name="sys")
# klassiek register voor uitlezen fasebits
from qiskit import ClassicalRegister
c_reg = ClassicalRegister(3, name="c")

qpe_circ = QuantumCircuit(qpe, anc_final, sys_final, c_reg, name="QPE")
# 1. Initialiseer ancilla en systeem
qpe_circ.append(prep_gate, anc_final[:])          # voorbereid ancilla zoals eerder (alternatief: begin in |00> en laat PREP deel van U doen)
qpe_circ.initialize([1,0], sys_final)             # bijvoorbeeld |0> als starttoestand systeem

# 2. Hadamards op alle QPE qubits
qpe_circ.h(qpe)

# 3. Gecontroleerde W^1, W^2, W^4 toepassen
# Bereid powers van W als gates (W, W^2, W^4)
from qiskit.quantum_info import Operator
W_op = Operator(W_gate)
W2_op = W_op.power(2);  W4_op = W_op.power(4)
W2_gate = W2_op.to_instruction(name="W^2")
W4_gate = W4_op.to_instruction(name="W^4")

# Controleer met QPE qubits
qpe_circ.append(W_gate.control(1), [qpe[0]] + anc_final[:] + sys_final[:])
qpe_circ.append(W2_gate.control(1), [qpe[1]] + anc_final[:] + sys_final[:])
qpe_circ.append(W4_gate.control(1), [qpe[2]] + anc_final[:] + sys_final[:])

# 4. Inverse Quantum Fourier Transform op QPE-register
qft_inv = QFT(num_qubits=3, inverse=True).to_gate(label="QFT†")
qpe_circ.append(qft_inv, qpe)

# 5. Meet het fase-register
qpe_circ.measure(qpe, c_reg)
```

Bovenstaand construeren we het volledige QPE-circuit. Eerst zetten we alle QPE-bits in $|+\rangle$ superpositie. Daarna voeren we gecontroleerde W-operaties uit: de minst significante QPE-qubit stuurt 1 toepassing van W aan, de volgende stuurt W^2 aan, de volgende W^4, etc. (Voor 3 qubits hebben we W, W^2, W^4.) Ten slotte doen we de inverse QFT op het fase-register en meten we deze uit.

Visualisatie van het circuit: hieronder zien we het complete circuit voor QPE. De subcircuit-gates `PREP`, `SELECT` en `PREPdg` vormen samen de block-encoding U (hier weergegeven als één blok `U`), en `QFT†` is de inverse Fourier-transformatie. Het fase-register (`qpe0..qpe2`) start met Hadamards, controleert vervolgens opeenvolgend W, W^2, W^4 (hier weergegeven als gecontroleerde W-blokken), waarna de inverse QFT de fasen interfereert en klaarzet voor meting:

【✏️ *Figuur: QPE-circuit voor het LCU-model. Het ancilla-register (2 qubits) en systeemqubit ondergaan eerst de block-encoding U (als onderdeel van W), en de fase qubits sturen W, W^2, W^4 aan. De inverse QFT (QFT†) zet de fase-informatie om voor uitlezen.*】

Zoals gewenst hebben we hiermee een volledig Qiskit-circuit gegenereerd dat:

1. De Hamiltoniaan voorbereidt als LCU,
2. Een PREP-stap bevat met $\sqrt{\omega_i/\lambda}$-amplituden,
3. Een SELECT-unitary implementeert die afhankelijk van het ancilla-register I, X of $-Z$ toepast op het systeem,
4. De block-encoding $U = \mathrm{PREP} \rightarrow \mathrm{SELECT} \rightarrow \mathrm{PREP}^\dagger$ realiseert,
5. De walk-operator $W = (2|00\rangle\langle 00| - I)\,U$ construeert,
6. Quantum Phase Estimation uitvoert met W (met fase-register en meting van de fase).

Complete code (Qiskit 0.45+) – onderstaand script combineert alle stappen en kan direct gesimuleerd worden (bijvoorbeeld met Aer's unitary simulator of statevector simulator) om de eenheidsmatrix of outputtoestand te inspecteren:

```
from qiskit import QuantumCircuit, QuantumRegister, ClassicalRegister
from qiskit.circuit.library import XGate, IGate, QFT
from qiskit.quantum_info import Operator
import numpy as np

# Parameters
omega = [1.5, 0.5, -0.5]
lam = sum(abs(w) for w in omega)
# Amplitudes for ancilla states |00>, |01>, |10>, |11>
alpha = [np.sqrt(abs(w)/lam) for w in omega] + [0]
alpha = np.array(alpha, dtype=complex)
```

```
    alpha = alpha / np.linalg.norm(alpha)

    # Define ancilla PREP gate
    anc = QuantumRegister(2, "anc")
    prep_circ = QuantumCircuit(anc, name="PREP")
    prep_circ.prepare_state(alpha, anc)
    prep_gate = prep_circ.to_gate(label="PREP")

    # Define single-qubit unitaries I, X, -Z
    id_gate = IGate()    # identity gate
    x_gate = XGate()
    negz_circ = QuantumCircuit(1, name="-Z")
    negz_circ.x(0); negz_circ.z(0); negz_circ.x(0)
    negz_gate = negz_circ.to_gate(label="-Z")
    U_list = [id_gate, x_gate, negz_gate]

    # Build SELECT gate
    sys = QuantumRegister(1, "sys")
    select_circ = QuantumCircuit(anc, sys, name="SELECT")
    m = anc.size
    for i, U_gate in enumerate(U_list):
        ctrl_state = format(i, f"0{m}b")
        controlled_U = U_gate.control(m, ctrl_state=ctrl_state)
        select_circ.append(controlled_U, anc[:] + sys[:])
    select_gate = select_circ.to_gate(label="SELECT")

    # Block-encoding U = PREP -> SELECT -> PREP†
    blockenc_circ = QuantumCircuit(anc, sys, name="U_block")
    blockenc_circ.append(prep_gate, anc[:])
    blockenc_circ.append(select_gate, anc[:] + sys[:])
    blockenc_circ.append(prep_gate.inverse(), anc[:])
    U_block = blockenc_circ.to_gate(label="U")

    # Quantum walk operator W = R * U
    W_circ = QuantumCircuit(anc, sys, name="W")
    W_circ.append(U_block, anc[:] + sys[:])
    W_circ.x(anc[0]); W_circ.x(anc[1])
    W_circ.cz(anc[0], anc[1])
    W_circ.x(anc[0]); W_circ.x(anc[1])
```

```
W_gate = W_circ.to_gate(label="W")

# QPE circuit
qpe = QuantumRegister(3, "qpe")            # phase register (3 qubits)
c_reg = ClassicalRegister(3, "c")          # classical bits for measurement
anc_final = QuantumRegister(2, "anc")      # ancilla (2 qubits)
sys_final = QuantumRegister(1, "sys")      # system qubit
qpe_circ = QuantumCircuit(qpe, anc_final, sys_final, c_reg, name="QPE")

# Initialize ancilla and system
qpe_circ.append(prep_gate, anc_final[:])       # prepare ancilla superposition
qpe_circ.initialize([1, 0], sys_final[:])      # system |0> (can be changed as needed)

# Hadamards on phase qubits
qpe_circ.h(qpe[:])

# Controlled W^1, W^2, W^4
W_op = Operator(W_gate)
W1_gate = W_gate
W2_gate = W_op.power(2).to_instruction(label="W^2")
W4_gate = W_op.power(4).to_instruction(label="W^4")
qpe_circ.append(W1_gate.control(1), [qpe[0]] + anc_final[:] + sys_final[:])
qpe_circ.append(W2_gate.control(1), [qpe[1]] + anc_final[:] + sys_final[:])
qpe_circ.append(W4_gate.control(1), [qpe[2]] + anc_final[:] + sys_final[:])

# Inverse QFT on phase register
qpe_circ.append(QFT(3, inverse=True).to_gate(label="QFT†"), qpe[:])

# Measure phase register
qpe_circ.measure(qpe[:], c_reg[:])

print(qpe_circ)
```

Als we dit circuit uitvoeren op een unitary-simulator, kunnen we de unitair matrix of outputtoestand controleren. Bijvoorbeeld, met de `AerSimulator(method='unitary')` zouden de bovenhoeks 2×2 submatrixelementen van de unitair overeen moeten komen met $H/\lambda = \frac{1}{2.5}(1.5I + 0.5X - 0.5Z)$ (Linear combination of unitaries with Qiskit circuit - Quantum Computing Stack Exchange). Bij uitvoering van QPE (bijv. op de toestand $|0\rangle_{\text{sys}}$) zal het fase-register een bitstring teruggeven die correspondeert met een van de eigenfasen θ van W, waarmee men vervolgens de eigenenergie $E = \lambda \cos\theta$ kan bepalen. (In dit eenvoudige model zou men twee mogelijke uitkomsten zien, overeenkomend met de twee eigenenergieën van H.) Hiermee is voldaan aan alle eisen, en hebben we een volledig Qiskit-circuit volgens moderne standaarden gecreëerd voor het LCU-toymodel in first quantization.

```
In [ ]:
```