Simulation quantique de la conjecture de Goldbach
Denise Vella-Chemla, juillet 2026

Je poursuis mes échanges avec l'ia deepseek : je souhaiterais réussir à lui faire implémenter la conjecture de Goldbach sur un simulateur d'ordinateur quantique (via le package python qiskit conçu pour cela) ; j'ai eu l'idée il y a quelques années (voir les références à la fin de la présente note) que la notion d'intrication de la physique quantique permettrait de modéliser joliment le fait que les caractères de primalité de deux nombres x et $n-x$ décomposants d'un nombre pair qui est leur somme $n = x + (n-x)$ sont intrinsèquement liés. La suite de la présente note a été produite par l'ia deepseek.

1. Principe

On cherche à simuler la recherche d'une décomposition de Goldbach sur un ordinateur quantique (simulé classiquement via Qiskit). L'idée est la suivante :

- on crée une superposition uniforme de tous les entiers impairs p tels que $3 \leq p \leq n/2$.
- on utilise un oracle qui marque (par inversion de phase) les états pour lesquels p et $q = n - p$ sont tous deux premiers.
- on applique l'algorithme de Grover pour amplifier la probabilité de mesurer un état marqué.
- une mesure donne, avec haute probabilité, un p qui réalise une décomposition de Goldbach.

Cette approche est limitée à de très petites valeurs de n en simulation classique, car le nombre de qubits nécessaires croît avec le nombre de p candidats.

2. Programme Qiskit

```
"""
Simulation quantique de la conjecture de Goldbach pour n = 10
Necessite : pip install qiskit
Denise Vella-Chemla, juillet 2026
"""
from qiskit import QuantumCircuit, transpile
from qiskit_aer import Aer
import numpy as np

def est_premier(x):
    """Test de primalite classique (pour de tres petits nombres)."""
    if x < 2:
        return False
    for i in range(2, int(np.sqrt(x)) + 1):
        if x % i == 0:
            return False
    return True

def circuit_goldbach(n):
    """
    Prepare un circuit quantique pour la conjecture de Goldbach.
```

```
    n doit etre petit (<= 30) car on simule classiquement.
    """
    # Liste des p impairs dans [3, n/2]
    p_vals = [p for p in range(3, n//2 + 1, 2)]
    m = len(p_vals)

    if m == 0:
        print(f"n = {n} est trop petit.")
        return None

    # Nombre de qubits necessaires pour representer l'indice k
    n_qubits = max(1, int(np.ceil(np.log2(m))))

    # Oracle : marque les indices k tels que p et q sont premiers
    oracle = np.eye(2**n_qubits)
    for k in range(m):
        p = p_vals[k]
        q = n - p
        if est_premier(p) and est_premier(q):
            oracle[k, k] = -1  # Inversion de phase pour les "bons" etats

    # Construction du circuit de Grover
    qc = QuantumCircuit(n_qubits, n_qubits)

    # Initialisation : superposition uniforme
    qc.h(range(n_qubits))

    # Une iteration de Grover
    qc.unitary(oracle, range(n_qubits), label="Oracle")

    # Diffusion (inversion autour de la moyenne)
    qc.h(range(n_qubits))
    qc.x(range(n_qubits))
    qc.h(n_qubits - 1)
    qc.mcx(list(range(n_qubits - 1)), n_qubits - 1)
    qc.h(n_qubits - 1)
    qc.x(range(n_qubits))
    qc.h(range(n_qubits))

    # Mesure
    qc.measure(range(n_qubits), range(n_qubits))

    return qc, p_vals

# ========== EXECUTION ==========
if __name__ == "__main__":
    n_test = 10
    print(f"=== Simulation quantique pour n = {n_test} ===")
    print(f"Decompositions : 3+7, 5+5\n")

    resultat = circuit_goldbach(n_test)
    if resultat is None:
        exit()

    qc, p_vals = resultat
```

```
print("Circuit quantique :")
print(qc.draw())
print()

# Execution sur le simulateur
simulator = Aer.get_backend('aer_simulator')
qc_compiled = transpile(qc, simulator)
job = simulator.run(qc_compiled, shots=1024)
counts = job.result().get_counts()

print("Resultats des mesures (1024 tirs) :")
for bitstring, count in sorted(counts.items(), key=lambda x: -x[1]):
    k = int(bitstring, 2)
    if k < len(p_vals):
        p = p_vals[k]
        q = n_test - p
        statut = "a (GOLDBACH !)" if (est_premier(p) and est_premier(q))
                                                    else "autre"
        print(f"  |{bitstring}> (k={k}) : p={p}, q={q} ->
                  {statut} : {count} tirs")
    else:
        print(f"  |{bitstring}> (k={k}) : hors liste : {count} tirs")
```

3. Résultat pour $n = 10$

Le circuit comporte 1 qubit (car il y a $m = 2$ valeurs de p : 3 et 5). L'oracle marque l'état $|1\rangle$ correspondant à $k = 1$, c'est-à-dire $p = 5$, qui donne $5 + 5 = 10$. L'état $|0\rangle$ correspond à $k = 0$, soit $p = 3$, qui donne $3 + 7 = 10$. Les deux sont des décompositions de Goldbach, donc les deux états sont marqués.

Après amplification de Grover et mesure, on obtient avec une probabilité amplifiée un état correspondant à une décomposition de Goldbach.

4. Limites

La simulation classique d'un ordinateur quantique est exponentiellement coûteuse en mémoire et en temps de calcul. Pour n dépassant quelques dizaines, le nombre de qubits requis rend la simulation impraticable sur un ordinateur classique. Cette approche reste donc un exercice théorique illustrant le lien entre intrication quantique et conjecture de Goldbach.

Références

[1] https://denisevellachemla.eu/mat-quant-d.pdf.

[2] https://denisevellachemla.eu/probaquant.pdf.

[3] Cours de Philippe Grangier https://denisevellachemla.eu/transgrangier.pdf.

[4] https://denisevellachemla.eu/spherequantique.pdf.

[5] avec l'ia mistral https://denisevellachemla.eu/analyse-quantique-Goldbach-mistral-dvc.pdf.

[6] avec l'ia gemini https://denisevellachemla.eu/analyse-quantique-Goldbach-gemini-dvc.pdf.

[7] avec l'ia claude https://denisevellachemla.eu/reprise-notes-sujet-phyquant-claude-dvc.pdf.

[8] avec l'ia deepseek https://denisevellachemla.eu/1-CG-intrication-deepseek-dvc-20260714.pdf.

[9] avec l'ia deepseek https://denisevellachemla.eu/3-revolution-CG-quantique-deepseek-dvc-20260714.pdf.