

# **ALGORITHMES QUANTIQUES MAJEURS**

## **DE LA THÉORIE À LA PRATIQUE**



## PLAN DE LA PRÉSENTATION

- Introduction aux algorithmes quantiques
- Algorithme de Deutsch-Jozsa
- **Algorithme de Grover** (recherche)
- **Algorithme de Shor** (factorisation)
- Variational Quantum Eigensolver (VQE)
- Quantum Approximate Optimization (QAOA)
- Applications pratiques
- Ressources et perspectives



# **PARTIE 1 : INTRODUCTION**

## **POURQUOI LES ALGORITHMES QUANTIQUES ?**



# AVANTAGE QUANTIQUE

**Problèmes où le quantique excelle :**

- Recherche non structurée → Grover
- Factorisation de nombres → Shor
- Simulation quantique → VQE
- Optimisation → QAOA
- Machine Learning → QML

**Speedup quantique :**

- Exponentiel (Shor)
- Quadratique (Grover)
- Variable (QAOA, VQE)



# CONCEPTS FONDAMENTAUX

## 1. Superposition

- Un qubit = 0 ET 1 simultanément
- N qubits = 2 états parallèles

## 2. Intrication

- Corrélation quantique
- Information partagée

## 3. Interférence

- Amplifier bonnes solutions
- Annuler mauvaises solutions

```
# Exemple : superposition
from qiskit import
QuantumCircuit

qc = QuantumCircuit(1)
qc.h(0) # Hadamard =
superposition
# État = ( $|0\rangle + |1\rangle$ ) /  $\sqrt{2}$ 

# Mesure → 50% chance 0 ou 1
```



```
qc.measure_all()
```

## **PARTIE 2 : DEUTSCH-JOZSA**

### **LE PREMIER ALGORITHME QUANTIQUE**



## PROBLÈME DE DEUTSCH-JOZSA

**Question :** Une fonction  $f(x)$  est-elle constante ou équilibrée ?

**Constante :**  $f(x) = 0$  pour tous  $x$  OU  $f(x) = 1$  pour tous  $x$

**Équilibrée :**  $f(x) = 0$  pour exactement 50% des  $x$

**Classique :** besoin de  $2^{(n-1)} + 1$  évaluations

**Quantique :** 1 seule évaluation !



oracle: fonction quantique à tester

n: nombre de qubits

"""

```
qr = QuantumRegister(n+1, 'q')
```

```
cr = ClassicalRegister(n, 'c')
```

```
qc = QuantumCircuit(qr, cr)
```

```
# Préparation
```

```
qc.x(n) # Dernier qubit en  $|1\rangle$ 
```

```
qc.barrier()
```

```
# Superposition
```

```
for i in range(n+1):
```

```
    qc.h(i)
```

```
qc.barrier()
```

```
# Oracle
```

```
oracle(qc, qr)
```

```
qc.barrier()
```

```
# Interférence
```

```
for i in range(n):
```

```
    qc.h(i)
```





```
# Mesure
qc.measure(qr[:n])

return qc

# Si mesure = 0...0
# Si mesure ≠ 0...0
```

```
def f(x):
    # Ne fait rien !
    # f(x) = 0 pour tous x
    pass

# Créer le circuit
circuit = deutsch_jozsa(
    constant_oracle,
    n=3
)

# Simuler
from qiskit import Aer,
execute
simulator =
Aer.get_backend(
    'qasm_simulator'
)
job = execute(
    circuit,
    simulator,
    shots=1024
)
result = job.result()
counts =
```



```
result.get_counts()
```

```
# Résultat : 000 (100%)
```

```
# → Fonction constante !
```

## **PARTIE 3 : ALGORITHME DE GROVER**

### **RECHERCHE DANS UNE BASE DE DONNÉES**



# LE PROBLÈME DE RECHERCHE

## Contexte :

- Base de données de  $N$  éléments
- Un seul élément marqué (solution)
- Trouver cet élément

## Classique :

- Recherche aléatoire :  $O(N)$
- En moyenne  $N/2$  tentatives

## Grover :

- $O(\sqrt{N})$  tentatives
- Speedup quadratique !

## Exemple :

- $N = 1$  million
- Classique : 500,000 essais
- Grover : 1,000 essais
- **Gain : 500x plus rapide !**



# PRINCIPE DE GROVER

## AMPLIFICATION D'AMPLITUDE

Étapes :

1. **Initialisation** : superposition uniforme
  - Tous les états équiprobables
2. **Oracle** : marquer la solution
  - Inverser le signe de la solution
3. **Diffusion** : amplifier l'amplitude
  - Inversion par rapport à la moyenne
4. **Répéter**  $\sqrt{N}$  fois
5. **Mesurer** → solution !

**Analogie** : Comme chercher une personne dans une foule en la faisant sauter légèrement à chaque itération



- Dépend du problème spécifique
- Utilise des qubits auxiliaires
- Opérations réversibles

```
def simple_oracle(circuit, qubits,
                  target):
    """
    Oracle pour marquer un état
    target: état à trouver (binaire)
    """
    n = len(qubits)

    # Inverser les qubits à 0
    for i, bit in enumerate(target):
        if bit == '0':
            circuit.x(qubits[i])

    # Multi-controlled Z
    circuit.h(qubits[-1])
    circuit.mcx(
        qubits[:-1],
        qubits[-1]
    )
```



ou  $|s\rangle$  – superposition uniforme  
`circuit.h(qubits[-1])`

Effet :

- Amplitudes proches de la moyenne → diminuent
- Amplitudes éloignées → augmentent

```
def diffusion_operator(  
    circuit,  
    qubits  
):  
    """  
    Opérateur de diffusion  
    (inversion par rapport à moyenne)  
    """  
    n = len(qubits)  
  
    # H sur tous  
    circuit.h(qubits)  
  
    # Inverser tous  
    circuit.x(qubits)  
  
    # Multi-controlled Z  
    circuit.h(qubits[-1])  
    circuit.mcx(  
        qubits, qubits[-1])
```



```

        qubits[:-1],
        qubits[-1]
    )
    circuit.h(qubits[-1])

```

## GROVER : NOMBRE D'ITÉRATIONS

# Restaurer

```

circuit.x(qubits)
circuit.h(qubits)

```

Formule :  
Nombre optimal d'itérations :  $R \approx (\pi/4) \times \sqrt{N}$

### Exemples :

- $N = 4 \rightarrow R = 1$
- $N = 16 \rightarrow R = 3$
- $N = 256 \rightarrow R = 12$
- $N = 1024 \rightarrow R = 25$

**Important :** Trop d'itérations  $\rightarrow$  performance décroît !

### Probabilité de succès :

$$P(\text{success}) = \sin^2((2R+1)\theta)$$

$$\text{où } \sin(\theta) = 1/\sqrt{N}$$



```
# Oracle
oracle(qc, qr, target)
qc.barrier()

# Diffusion
diffusion_operator(qc, qr)
qc.barrier()

# 3. Measure
qc.measure(qr, cr)
```

```
return qc
```

```
def oracle(circuit, qubits, target):
    """Oracle qui marque l'état target"""
    # Inverser les qubits qui doivent être 0
    for i, bit in enumerate(target):
        if bit == '0':
            circuit.x(qubits[i])

    # Multi-controlled Z
    circuit.h(qubits[-1])
    circuit.mcx(qubits[:-1], qubits[-1])
    circuit.h(qubits[-1])
```





# Restaurer **1. Recherche en base de données**

```
for i, bit in enumerate(target):  
    if bit == '0':  
        circuit.x(qubits[i])
```

- Recherche non structurée
- Optimisation de requêtes

## 2. Cryptographie

```
def diffusion_operator(circuit, qubits):  
    """Opérateur de diffusion"""  
    circuit.h(qubits)
```

- Attaque sur clés symétriques
- Nécessite de doubler la taille des clés

## 3. Satisfiabilité (SAT)

```
circuit.x(qubits)  
circuit.h(qubits[0])  
circuit.mcx(qubits[:-1], qubits[-1])  
circuit.h(qubits[-1])  
circuit.x(qubits)  
circuit.h(qubits)
```

- Problèmes NP-complets
- Combine avec d'autres techniques

## 4. Optimisation

- Trouver minimum/maximum

- Problèmes combinatoires

```
# Exemple: chercher '101' parmi 8 états  
circuit = grover_search(3, '101')
```

## 5. Machine Learning

- Recherche de patterns

```
# Simuler  
simulator = Aer.get_backend('qasm_simulator')
```

## 6. Jeux et puzzles

```
job = execute(circuit, simulator, shots=1024)  
result = job.result()  
counts = result.get_counts()
```

- Sudoku
- N-Queens



```
print("Résultats: " + pathfinding)
# Output attendu: {'101': ~1000, autres: ~24}
```

**Objectif :** Implémenter Grover pour chercher dans 16 éléments

**Étapes :**

1. Créer un circuit pour  $n=4$  qubits
2. Choisir un état cible (ex: '1010')
3. Calculer  $R = \lfloor (\pi/4)\sqrt{16} \rfloor = 3$
4. Implémenter l'oracle
5. Implémenter la diffusion
6. Exécuter et vérifier

**Défi :** Essayer avec différentes cibles

```
# Votre code ici
def mon_grover():
    # À compléter
    pass

# Test
target = '1010'
circuit = mon_grover()
# ...
```



**Résultat attendu :**

- Probabilité ~95% pour l'état cible
- ~0.3% pour les autres états

**PARTIE 4 : ALGORITHME DE SHOR**  
**FACTORISATION EN TEMPS POLYNOMIAL**



# LE PROBLÈME DE FACTORISATION

**Contexte :** Factoriser  $N = p \times q$  où  $p, q$  premiers

**Exemple :**

- $N = 15 = 3 \times 5$  (facile)
- $N = 2048$  bits (impossible classiquement)




**Classique :**

- Meilleur algorithme : GNFS
- Temps :  $\exp(O(\sqrt[3]{\log N}))$
- RSA-2048 : plusieurs années

**Shor :**

- Temps :  $O((\log N)^3)$
- Polynomial !
- RSA cassé en quelques heures

**Impact :**

-  RSA vulnérable
-  Diffie-Hellman vulnérable
-  ECC partiellement vulnérable



1. **Reduction classique :**  
Mais : nécessite  $\sim 2000-4000$  qubits logiques

- Choisir  $a < N$  aléatoire
- Si  $\text{pgcd}(a, N) \neq 1 \rightarrow$  facteur trouvé !

2. **Trouver la période  $r$  (quantique) :**

- $f(x) = a \bmod N$
- $r =$  période de  $f$

3. **Extraction classique :**

- Si  $r$  pair et  $a^{(r/2)} \neq -1 \bmod N$
- Facteurs =  $\text{pgcd}(a^{(r/2)} \pm 1, N)$

**Exemple :**  $N = 15, a = 7$

$$f(x) = 7 \bmod 15$$

$$x=0: 7 = 1$$

$$x=1: 7 = 7$$

$$x=2: 7 = 49 \bmod 15 = 4$$

$$x=3: 7 = 343 \bmod 15 = 13$$

$$x=4: 7 = 2401 \bmod 15 = 1 \leftarrow \text{répétition!}$$

$$\text{Période } r = 4$$

Facteurs:



$$\gcd(7-1, 15) = \gcd(48, 15) = 3$$

$$\gcd(7+1, 15) = \gcd(50, 15) = 5$$

## 15 = 3 x 5 ✓ **SHOR: QUANTUM PHASE ESTIMATION**

### Composant quantique :

- Estimation de phase (QPE)
- Transformation de Fourier quantique (QFT)

QPE trouve la phase  $\phi$  telle que :  $U|\psi\rangle = e^{(2\pi i\phi)}|\psi\rangle$

### Pour Shor :

- $U$  = multiplication modulaire par  $a$
- $\phi = k/r$  où  $r$  = période
- QFT inverse pour extraire  $r$

### Architecture :

- Registre 1 : comptage ( $n$  qubits)
- Registre 2 : travail ( $n$  qubits)
- QFT inverse sur registre 1
- Mesure  $\rightarrow$  approximation de  $k/r$



- Speedup exponentiel !

### Utilité :

- Détection de périodicité
- Algorithme de Shor
- Simulation quantique

```
def qft(circuit, qubits):  
    """  
    Quantum Fourier Transform  
    """  
    n = len(qubits)  
  
    for i in range(n):  
        # Hadamard sur qubit i  
        circuit.h(qubits[i])  
  
        # Rotations contrôlées  
        for j in range(i+1, n):  
            angle = np.pi / (2**(j-i))  
            circuit.cp(  
                angle,  
                qubits[j],  
                qubits[i])
```



## ) QFT INVERSE

# Swap pour inverser ordre

```
def inverse_qft(circuit, qubits):  
    """  
    Inverse Quantum Fourier Transform  
    Utilisé dans l'algorithme de Shor  
    """  
    n = len(qubits)  
  
    # Swap inverse (d'abord)  
    for i in range(n//2):  
        circuit.swap(qubits[i], qubits[n-1-i])  
  
    # Opérations inverses dans l'ordre inverse  
    for i in range(n-1, -1, -1):  
        # Rotations contrôlées inverses  
        for j in range(n-1, i, -1):  
            angle = -np.pi / (2**(j-i))  
            circuit.cp(angle, qubits[j], qubits[i])  
  
        # Hadamard sur qubit i  
        circuit.h(qubits[i])
```





- Decomposition en multiplications
- Multiplications modulaires successives
- Utilisation de registres auxiliaires

```
def modular_exponentiation(  
    circuit,  
    control_qubits,  
    target_qubits,  
    a,  
    N  
):  
    """  
     $U|x\rangle|y\rangle = |x\rangle|y \cdot a \bmod N\rangle$   
    """  
    n = len(control_qubits)  
  
    for i in range(n):  
        #  $a^{(2^i)} \bmod N$   
        power = pow(  
            a,  
            2**i,  
            N  
        )
```



```
return qc
```

```
def controlled_U_power(circuit, control, target_qubits, a, power, N):
```

```
    """
```

```
    Applique Ude manière contrôlée
```

```
     $U|y\rangle = |ay \bmod N\rangle$ 
```

```
    """
```

```
    # Calcul classique de  $a \bmod N$ 
```

```
    a_power = pow(a, power, N)
```

```
    # Implémentation simplifiée
```

```
    # En réalité, ceci nécessite un circuit complexe
```

```
    # avec des opérations arithmétiques quantiques
```

```
    # Ici on simule avec une porte personnalisée
```

```
    # (version pédagogique)
```

```
    pass
```

```
# Exemple d'utilisation
```

```
N = 15 # Nombre à factoriser
```

```
a = 7  # Base (pgcd(7,15) = 1)
```

```
circuit = shors_algorithm_simple(N, a)
```



```

# Post-traitement classique
def find_period(measurement_results):
    """
    Extraire la période des résultats de mesure
    """
    from fractions import Fraction
    from math import gcd

    # Convertir le résultat binaire en nombre
    measured = int(measurement_results, 2)
    n_count = len(measurement_results)

    # Approximation
    phase = measured / n_count
    frac = Fraction(phase)
    r = frac.denominator

    return r

def factor_from_period(r):
    """
    Extraire les facteurs de r
    """
    if r % 2 != 0:
        return None

```

- 8 qubits de comptage

- 4 qubits auxiliaires

- QPE + QFT inverse

Étape 3 : Mesure

- Résultat : 64 (binaire: 01000000)

- Phase :  $64/256 = 1/4$

Étape 4 : Période

- $1/4 = 1/r \Rightarrow r = 4$

Étape 5 : Factorisation

```

r = 4
a = 7
N = 15

```

```

# r est pair ✓
# a^(r/2) mod N
x = 7 mod 15 = 49 mod 15 = 4

# x ≠ 14 ✓

# Facteurs
p = gcd(4+1, 15) = gcd(5, 15)

```



## 1. Nombre de qubits

```
x = pow(a, r//2, q) = gcd(4-1, 15) = gcd(3, 15)
```

- RSA-2048 : ~4000 qubits logiques

```
if x == 1: N = 20,000+ qubits physiques (avec correction d'erreurs)
```

```
return None # Résultat: 15 = 3 x 5 ✓
```

## 2. Fidélité des portes

- Erreur < 0,01% par porte

```
factor1 = gcd(x + 1, N)
```

```
factor2 = gcd(x - 1, N)
```

- Correction d'erreurs quantiques nécessaire

## 3. Connectivité

```
if factor1 > 1 and factor1 < N:
```

- Portes à 2 qubits entre qubits distants

```
return factor1, N // factor1
```

- SWAP chains

```
if factor2 > 1 and factor2 < N:
```

## 4. Profondeur du circuit

- Millions de portes

```
return None, None
```

- Cohérence limitée

## Progrès actuels :

```
# Exemple de code actuellement
```

```
# (après simulation du circuit)
```

- 2001 : Shor avec 7 qubits (N=15)

```
# measurement = '01000000' # Exemple de résultat
```

- 2012 : N=21 avec photons

```
# r = find_period(measurement, N)
```

- 2019 : N=35 (compilation optimisée)

```
# p, q = factor_from_period(N, r)
```

```
# print(f"N={N} p={p} q={q}")
```

- 2024 : N=48 (nouveau record)

## Estimation :

- RSA cassé d'ici 10-15 ans ?

- Nécessite une rupture technologique

## 1. Cryptographie

- ⚠ Menace sur RSA
- ⚠ Menace sur Diffie-Hellman
- Transition vers post-quantique

## 2. Logarithme discret

- Problème DLP
- Courbes elliptiques
- Protocoles de signature

## 3. Théorie des nombres

- Recherche mathématique
- Propriétés des nombres premiers
- Conjectures

## 4. Chimie

- Structure électronique
- (Shor généralisé)

## 5. Optimisation

- Problèmes combinatoires
- (Variantes de Shor)



**Note :** Migration vers la cryptographie post-quantique en cours (NIST PQC)

## **PARTIE 5 : VQE**

### **VARIATIONAL QUANTUM EIGENSOLVER**



# VQE : PRINCIPE

**Objectif :** Trouver l'état fondamental (plus basse énergie) d'un Hamiltonien  $H$

**Principe :**

- Algorithme hybride quantique-classique
- Circuit paramétré quantique (ansatz)
- Optimisation classique des paramètres

**Applications :**

- Chimie quantique
- Science des matériaux
- Optimisation

**Boucle VQE :**

1. Préparer état  $|\psi(\theta)\rangle$  (quantique)
2. Mesurer  $\langle H \rangle$  (quantique)
3. Optimiser  $\theta$  (classique)
4. Répéter jusqu'à convergence



- Adapté au matériel
- Portes disponibles

## 2. Unitary Coupled Cluster (UCC)

- Inspiré de chimie quantique
- Physiquement motivé

## 3. Alternating Layered

- Couches répétées
- Entanglement + rotations

```
from qiskit.circuit.library
import \
    TwoLocal

# Ansatz simple
ansatz = TwoLocal(
    num_qubits=4,
    rotation_blocks=['ry',
'rz'],
    entanglement_blocks='cx',
    entanglement='linear',
    reps=3
)
```





```
# Exemple:  $H = Z_0Z_1 + 0.5 \cdot X_0X_1$   
 $H = (Z \wedge Z) + 0.5 * (X \wedge X)$ 
```

```
# 2. Créer l'ansatz
```

```
ansatz = TwoLocal(  
    num_qubits=2,  
    rotation_blocks='ry',  
    entanglement_blocks='cx',  
    reps=1  
)
```

```
# 3. Choisir l'optimiseur
```

```
optimizer = COBYLA(maxiter=200)
```

```
# 4. Backend quantique
```

```
backend = Aer.get_backend('qasm_simulator')
```

```
# 5. Créer l'instance VQE
```

```
vqe = VQE(  
    ansatz=ansatz,  
    optimizer=optimizer,  
    quantum_instance=backend  
)
```



```
# 6. Calculer l'énergie de liaison
H = -1.04·I - 0.18·Z0
result = vqe.compute_eigenstate(
    -0.18·Z1 + 0.17·Z0Z1
    + 0.04·X0X1 + 0.04·Y0Y1)

print(f"Énergie fondamental: {result.eigenvalue}")
print(f"État optimal: {result.eigenstate}")
print(f"Paramètres optimaux: {result.optimal_parameters}")
```

**Objectif: Trouver l'énergie de liaison**

```
# Visualiser le circuit optimal
optimal_circuit = a
print(optimal_circuit)

from qiskit_nature.units
import \
    DistanceUnit
from qiskit_nature.second_q.\
    drivers import
PySCFDriver

# Définir la molécule
driver = PySCFDriver(
    atom='H 0 0 0; H 0 0
0.735',

    unit=DistanceUnit.ANGSTROM,
    basis='sto3g'
)

# Obtenir le problème
problem = driver.run()
```



```
# Hamiltonien
hamiltonian = problem.\
    hamiltonian.second_q_op()

# Résoudre avec VQE
# (comme avant)
```

## **PARTIE 6 : QAOA**

### **QUANTUM APPROXIMATE OPTIMIZATION ALGORITHM**



# QAOA : PRINCIPE

**Objectif :** Résoudre des problèmes d'optimisation combinatoire

**Exemples :**

- Max-Cut
- Traveling Salesman
- Portfolio optimization
- Scheduling

**Approche :**

- Algorithme variationnel
- Alternance entre deux Hamiltoniens
- Paramètres  $\beta$  et  $\gamma$

**Structure :**  $|\psi(\beta, \gamma)\rangle = U(\beta_p, \gamma_p) \dots U(\beta_1, \gamma_1) |+\rangle$

où:

- $U(\beta, \gamma) = e^{(-i\beta H_M)} e^{(-i\gamma H_C)}$
- $H_C$  = Hamiltonien de coût
- $H_M$  = Hamiltonien de mélange



## Problème Max-Cut :

- Graphe  $G = (V, E)$
- Partitionner  $V$  en 2 ensembles
- Maximiser arêtes entre ensembles

## Encodage :

- Sommet  $i \rightarrow$  qubit  $i$
- $|0\rangle =$  ensemble A
- $|1\rangle =$  ensemble B

**Hamiltonien de coût :**  $H_C = -\frac{1}{2} \sum_{(i,j) \in E} (Z_i Z_j)$

## Exemple :

Graphe triangle:

0---1  
 \ /  
 2

$$H_C = -\frac{1}{2}(Z_0 Z_1 + Z_1 Z_2 + Z_0 Z_2)$$

Solution optimale:  $|101\rangle$  ou  $|010\rangle$



```
H = 0
```

```
for edge in G.edges():
    i, j = edge
    #  $Z_i \otimes Z_j$  sur les qubits i et j
    pauli_str = ['I'] * n
    pauli_str[i] = 'Z'
    pauli_str[j] = 'Z'

    # Ajouter au Hamiltonien
    H += -0.5 * eval(' ^ '.join([c + str(i)
                                  for i, c in enumerate(pauli_str)]))

return H
```

```
H_C = cost_hamiltonian(G)
```

```
# 3. QAOA
```

```
optimizer = COBYLA()
```

```
backend = Aer.get_backend('qasm_simulator')
```

```
qaoa = QAOA(
    optimizer=optimizer,
    reps=2, # Nombre de couches p
```



```

quantum_instance=backend
)
# 4. Résoudre
result = qaoa.compute_minimum_eigenvalue(H_C)

print(f"Coût optimal: {result.eigenvalue.real}")
print(f"Solution: {result.eigenstate}")

# 5. Interpréter le résultat
from qiskit.result import distributions

# Obtenir les probabilités
probs = result.eigenstate.probabilities_dict()

print("\nSolutions probables:")
for state, prob in sorted(probs.items(), key=lambda x: -x[1][:5]):
    print(f"{state}: {prob:.3f}")

```

**Profondeur p :**

- $p = 1$  : approximation grossière
- $p = 2-5$  : bon compromis
- $p \rightarrow \infty$  : solution exacte

**Performance :**

- $p=1$  : ~75% de l'optimal (Max-Cut)
- $p=2$  : ~85%
- $p=5$  : ~95% distributions

**Trade-off :**

- Plus de couches  $\rightarrow$  meilleure solution
- Plus de couches  $\rightarrow$  plus de bruit

**Exemple : Max-Cut sur graphe 3-régulier**

## p Ratio approx.

|   |        |
|---|--------|
| 1 | 0.6924 |
| 2 | 0.7559 |
| 5 | 0.8621 |



# 1. Optimisation de portefeuille Ratio approx.

- Sélection d'actifs
- Gestion de risque

10

0.9468

## 2. Logistique

- Tournées de véhicules
- Planification de routes

## 3. Machine Learning

- Sélection de features
- Clustering

## 4. Télécommunications

- Allocation de fréquences
- Routage de réseaux

## 5. Planification

- Scheduling
- Allocation de ressources

## 6. Bioinformatique

- Protein folding
- Alignement de séquences





**Note :** QAOA fonctionne bien sur les ordinateurs quantiques NISQ actuels

## **PARTIE 7 : COMPARAISON DES ALGORITHMES**

**QUAND UTILISER QUEL ALGORITHME ?**







# TABLEAU COMPARATIF





| thme | Speedup     | Qubits   | Profondeur    | Maturité    | Appl     |
|------|-------------|----------|---------------|-------------|----------|
| ozsa | Exponentiel | $n+1$    | $O(1)$        | ✓ Prouvé    | Pédagog  |
|      | Quadratique | $\log N$ | $O(\sqrt{N})$ | ✓ Prouvé    | Recher   |
|      | Exponentiel | $2n$     | $O(n^3)$      | ✓ Prouvé    | Factoris |
|      | Variable    | $n$      | Variable      | 🔬 Recherche | Chimie   |
|      | Variable    | $n$      | Variable      | 🔬 Recherche | Optimis  |







### Grover :

-  Speedup garanti ( $\sqrt{N}$ )
-  Simple à implémenter
-  Peu d'applications pratiques
-  Nécessite oracle efficace



### Shor :

-  Speedup exponentiel
-  Applications révolutionnaires
-  Besoin de milliers de qubits
-  Correction d'erreurs nécessaire

### VQE :

-  Fonctionne sur NISQ
-  Applications en chimie
-  Pas de garantie d'optimalité
-  Convergence difficile

### QAOA :

-  Fonctionne sur NISQ
-  Large éventail de problèmes



- ✗ Pas de speedup prouvé
- ✗ Nécessite optimisation classique

## **PARTIE 8 : AUTRES ALGORITHMES IMPORTANTS**

### **AU-DELÀ DES CLASSIQUES**



## Similaire a Deutsch-Jozsa mais plus pratique

```
def bernstein_vazirani(n, secret):  
    """  
    secret: string binaire secret  
    """  
    qc = QuantumCircuit(n+1, n)  
  
    # Ancilla à  $|1\rangle$   
    qc.x(n)  
  
    # Superposition  
    for i in range(n+1):  
        qc.h(i)  
  
    # Oracle  
    for i, bit in enumerate(secret):  
        if bit == '1':  
            qc.cx(i, n)  
  
    # Hadamard  
    for i in range(n):  
        qc.h(i)
```



```
# Measure
qc.measure(range(n), range(n))

return qc
```

**SIMON'S ALGORITHM**  
 # Résultat = secret directement !

**Problème :** Trouver la période  $s$  d'une fonction 2-à-1

**Propriété :**  $f(x) = f(y) \iff x \oplus y = s$

**Classique :**  $O(2^{(n/2)})$  requêtes **Quantique :**  $O(n)$  requêtes

**Impact :**

- Précurseur de Shor
- Speedup exponentiel prouvé

**Applications :**

- Cryptanalyse
- Structures périodiques cachées
- Preuve de concept pour avantage quantique



**Classique** :  $O(N)$  ou  $O(N \log N)$  **Quantique** :  $O(\log N)$

**Conditions :**

- A hermitienne
- A bien conditionnée
- Accès QRAM aux données

**Défi** : La solution est un état quantique, pas un vecteur classique

**Applications :**

- Machine Learning quantique
- Analyse de données
- Simulation physique

```
# HHL est complexe
# Voir qiskit.algorithms.HHL

from qiskit.algorithms import HHL

# Définir matrice A
A = [[1, 0], [0, 2]]

# Vecteur b
b = [1, 2]
```



# Résoudre

hh1 **Concept** : Analogue quantique de la marche aléatoire

solution = hh1.solve(A, b)

**Types :**

- Discrete-time quantum walks
- Continuous-time quantum walks

**Propriétés :**

- Propagation quadratiquement plus rapide
- Interférence quantique
- Localisation

**Applications :**

- Recherche sur graphes
- Algorithmes de graphes
- Simulation de systèmes physiques

**Exemple** : Recherche sur graphe

- Classique :  $O(N)$
- Quantum Walk :  $O(\sqrt{N})$

**Variantes :**

- Search by quantum walk





- 1. Quantum Support Vector Machine
  - Element distinctness
  - Classification quantique
- Triangle finding
  - Kernel quantique

## 2. Quantum Neural Networks

- VQC (Variational Quantum Classifier)
- QNN avec ansatz paramétré

## 3. Quantum PCA

- Analyse en composantes principales
- Speedup exponentiel (théorique)

```
from qiskit_machine_learning.\
    algorithms import VQC

# Créer un classificateur
vqc = VQC(
    feature_map=feature_map,
    ansatz=ansatz,
    optimizer=optimizer
)

# Entraîner
vqc.fit(X_train, y_train)
```



```
# Prédire  
predictions = vqc.predict(X_test)
```

## **PARTIE 9 : APPLICATIONS PRATIQUES**

### **DU LABORATOIRE AU MONDE RÉEL**



## **1. Simulation moléculaire**

- Structure électronique
- États excités
- Réactions chimiques

## **2. Découverte de médicaments**

- Binding affinity
- Optimisation de molécules
- Criblage virtuel

## **3. Matériaux**

- Supraconducteurs
- Batteries
- Catalyseurs

**Algorithmes : VQE, QPE, Trotter**

**Exemples concrets :**

- $\text{LiH}$  : molécule simple, 12 qubits
- $\text{H}_2\text{O}$  : eau, ~14 qubits
- **Enzymes** : objectif à moyen terme

**Entreprises actives :**



- Zapata Computing
- Cambridge Quantum
- QC Ware

## 2. Pricing d'options

- Monte Carlo quantique
- Speedup quadratique

## 3. Détection de fraude

- Pattern recognition
- Anomaly detection

## 4. Risk analysis

- Simulation de scénarios
- Stress testing

## Algorithmes utilisés :

- QAOA (optimisation)
- Quantum Monte Carlo
- Grover (recherche)
- QML (apprentissage)

## Partenariats :



### 1. Optimisation de routes

- JP Morgan + IBM
- Traveling Salesman Problem
  - Goldman Sachs + QC Ware
- Vehicle Routing Problem
  - Citigroup + multiple
- Fleet management

### 2. Supply chain

- Gestion d'inventaire
- Planification de production
- Distribution optimale

### 3. Scheduling

- Planification d'horaires
- Allocation de ressources
- Job shop scheduling

**Algorithmes :** QAOA, Grover

### Cas d'usage :

- **DHL** : optimisation de routes de livraison
- **Volkswagen** : gestion de flotte de taxis
- **Airbus** : optimisation de maintenance

**Bénéfices :**



## 1. Cryptanalyse de coûts

- Diminution des émissions de CO<sub>2</sub>
- Efficacité accrue de la cryptographie symétrique
- Besoin de crypto post-quantique

## 2. Distribution quantique de clés (QKD)

- Sécurité prouvée
- BB84 protocol
- Réseaux quantiques

## 3. Génération de nombres aléatoires

- True randomness
- Certifiable
- Applications en sécurité

## Standardisation :

- **NIST PQC** : algorithmes post-quantiques
  - CRYSTALS-Kyber (chiffrement)
  - CRYSTALS-Dilithium (signature)
  - Falcon, SPHINCS+

## Transition :




- Commencer la migration maintenant
- Hybrid crypto (classique + PQC)

## **PARTIE 10 : RESSOURCES VIDÉO**

### **APPRENDRE AVEC YOUTUBE**



## 1. Qiskit

-  Qiskit Official
- Tutoriels officiels IBM
- Coding with Qiskit series
- Séminaires hebdomadaires


## 2. Microsoft Quantum

-  Microsoft Research
- Azure Quantum tutorials
- Q# programming

## 3. The Coding Train

-  Quantum Computing Playlist
- Approche ludique et accessible

## 4. MinutePhysics

-  Quantum Computing Explained
- Vulgarisation excellent
- Animations claires

## 5. Looking Glass Universe

-  LGU Channel





- Maths et physique quantique
- Très pédagogique

## GROVER : RESSOURCES VIDÉO

### Théorie et Explication :

- 🎥 Qiskit : Grover's Algorithm Explained
  - 15 min, excellent overview
- 🎥 PBS Infinite Series : Grover's Algorithm
  - Approche mathématique accessible

### Implémentation :

- 🎥 Qiskit : Coding Grover's Algorithm
  - Coding tutorial step-by-step
- 🎥 IBM Quantum : Grover's Search Lab
  - Exercice pratique guidé



## SHOR : RESSOURCES VIDÉO

### Théorie :

- 🎥 Veritasium : How Quantum Computers Break Encryption
  - Vulgarisation grand public, 20 min
- 🎥 3Blue1Brown : Quantum Fourier Transform
  - Mathématiques du QFT

### Implémentation :

- 🎥 Qiskit : Shor's Algorithm Tutorial
  - Implémentation complète
- 🎥 Microsoft Research : Shor's Algorithm in Q#
  - Avec Q# language



## VQE ET QAOA : RESSOURCES VIDÉO

### VQE :

- 🎥 Qiskit : VQE for Chemistry
  - Application en chimie quantique
- 🎥 Zapata Computing : Hybrid Quantum-Classical
  - Concept d'algorithmes hybrides

### QAOA :

- 🎥 Qiskit : QAOA Explained
  - Théorie et implémentation
- 🎥 IBM Research : QAOA for Optimization
  - Applications pratiques



## 1. Scott Aaronson

- 🎥 Lectures sur théorie
- Complexité quantique
- Niveau universitaire

## 2. John Preskill

- 🎥 Caltech Lectures
- Information quantique
- NISQ algorithms

## 3. Ronald de Wolf

- 🎥 CWI Lectures
- Algorithmes quantiques
- Niveau master

## 4. Umesh Vazirani

- 🎥 Berkeley Lectures
- Théorie de la complexité
- Très rigoureux

## 5. MIT OpenCourseWare

- 🎥 Quantum Computation



## RESSOURCES EN FRANÇAIS

- Cours complets

### 1. Science4All

- 🎬 Ordinateurs Quantiques
- Vulgarisation en français
- Très accessible

### 2. ScienceEtonnante

- 🎬 Physique Quantique
- Explications claires
- Grand public

### 3. e-penser

- 🎬 Quantique
- Approche ludique

### 4. Unisciel

- 🎬 Cours Quantique
- Niveau universitaire
- Ressources pédagogiques

### 5. CEA Recherche

- 🎬 Quantique CEA
- Recherche française



- Actualités

## CONFÉRENCES ET TALKS

### Grandes conférences :

- 🎥 QIP (Quantum Information Processing)
  - Conference annuelle majeure
- 🎥 Q2B (Quantum 2 Business)
  - Applications industrielles
- 🎥 APS March Meeting
  - Physique quantique

### TEDx Talks :

- 🎥 Michelle Simmons : "Quantum Computing"
- 🎥 Shohini Ghose : "Quantum Computing Explained"
- 🎥 Talia Gershon : "How Quantum Computers Work"



# **PARTIE 11 : EXERCICES PRATIQUES**

## **PROJETS GUIDÉS**



3. Créer l'oracle
  4. Créer l'opérateur de diffusion
  5. Assembler le circuit
  6. Simuler et analyser
- Temps estimé : 2 heures**

```
# Template de départ
from qiskit import *

def mon_grover(target):
    # Nombre de qubits
    n = len(target)

    # Créer circuit
    qc = QuantumCircuit(n, n)

    # 1. Superposition
    # À COMPLÉTER

    # 2. Itérations Grover
    # À COMPLÉTER

    # 3. Mesure
```





## Tâches # À COMPLÉTER

1. Oracle constant  $\rightarrow 0$
2. Oracle constant  $\rightarrow 1$
3. Oracle équilibré (première moitié 0)
4. Oracle équilibré (xor)
5. Tester avec Deutsch-Jozsa

**Challenge :** Créer un oracle équilibré original

**Temps estimé :** 1.5 heures

```
def oracle_constant_0(qc, qubits):  
    # Ne rien faire  
    pass  
  
def oracle_constant_1(qc, qubits):  
    # Inverser le dernier qubit  
    # À COMPLÉTER  
    pass  
  
def oracle_balanced_xor(qc, qubits):  
    # XOR de tous les qubits  
    # À COMPLÉTER  
    pass
```



- 2. Implémenter pour  $n=2$
- # Tester
- 3. Généraliser pour  $n$  quelconque
- # ...
- 4. Vérifier avec la QFT de Qiskit
- 5. Implémenter la QFT inverse
- 6. Appliquer sur différents états

**Temps estimé : 3 heures**

```
def my_qft(qc, qubits):  
    """  
    Quantum Fourier Transform  
    """  
  
    n = len(qubits)  
  
    # Pour chaque qubit  
    for i in range(n):  
        # Hadamard  
        # À COMPLÉTER  
  
        # Rotations contrôlées  
        for j in range(i+1,  
n):  
            # Phase gate  
            # À COMPLÉTER
```



3. Choisir un optimiseur pass
4. Exécuter VQE
5. Comparer avec l'énergie exacte # Swap des qubits
6. Varier la distance inter-atomique # À COMPLÉTER

**Bonus :** Tracer la courbe d'énergie vs distance return qc

**Temps estimé :** 4 heures

```
# Hamiltonien simplifié H2
# (pour R = 0.735 Angstrom)
H = (-1.052 * (I ^ I) +
      0.398 * (Z ^ I) +
      -0.398 * (I ^ Z) +
      -0.011 * (Z ^ Z) +
      0.181 * (X ^ X))
```

```
# Ansatz
ansatz = TwoLocal(
    # À COMPLÉTER
)
```

```
# VQE
vqe = VQE(
    # À COMPLÉTER
```



4. Optimiser les paramètres

5. Analyser les solutions

6. Essayer  $p=2$ ,  $p=3$

7. Comparer avec solution classique

Temps estimé: 5 heures

```
import networkx as nx
```

```
# Créer graphe
```

```
G = nx.Graph()
```

```
G.add_edges_from([
```

```
    # À COMPLÉTER
```

```
])
```

```
# Visualiser
```

```
nx.draw(G, with_labels=True)
```

```
# Hamiltonien
```

```
H_C = # À COMPLÉTER
```

```
# QAOA
```

```
qaoa = QAOA(
```

```
    optimizer=COBYLA(),
```

```
    reps=1
```



**Objectif :** Comparer Grover, classique, et exhaustif

**Problème :** Recherche d'un élément dans une liste

**Tâches :**

1. Implémenter recherche classique

2. Implémenter Grover

3. Implémenter exhaustif

4. Varier N (4, 8, 16, 32, 64, 128)

5. Mesurer le temps d'exécution

6. Compter les requêtes

7. Tracer les courbes

8. Analyser le speedup

**Livrables :**

- Code complet et commenté
- Rapport avec graphiques
- Analyse du speedup
- Limitations observées

**Extensions :**

- Ajouter du bruit
- Tester sur vraie machine



- Implémenter variantes de Grover
- Temps estimé : 8-10 heures**

## **PARTIE 12 : DÉFIS ACTUELS**

### **OBSTACLES ET PERSPECTIVES**



## **1. Correction d'erreurs**

- Erreur  $\sim 0.1-1\%$  par porte
- Besoin  $< 0.01\%$  pour Shor
- Code de correction nécessaire
- Overhead 10-1000x en qubits

## **2. Décohérence**

- Temps de cohérence :  $\mu\text{s}$  à  $\text{ms}$
- Besoin de minutes/heures
- Isolation difficile

## **3. Connectivité**

- Qubits pas tous connectés
- Besoin de SWAP gates
- Augmente profondeur

## **4. Scalabilité**

- Systèmes actuels :  $< 1000$  qubits
- Besoin : millions pour applications
- Cooling et contrôle complexes

## **5. Calibration**

- Doit être refaite régulièrement








**NISQ** = Noisy Intermediate-Scale Quantum

- Paramètres dérivent

### Caractéristiques :

- Automatisation nécessaire
- 50-1000 qubits
- Pas de correction d'erreurs
- Profondeur limitée (~100 portes)
- Applications restreintes

### Algorithmes adaptés :

- VQE 
- QAOA 
- Certains QML 
- Shor 
- Grover (grande échelle) 

### Stratégies :

- Algorithmes variationnels
- Circuits peu profonds
- Techniques de mitigation d'erreurs
- Hybrid quantum-classical

### Timeline :





### 3. Measurement Error Mitigation

- 2024: NISQ mature
- 2030: Early fault-tolerant
- 2040: Error-corrected quantum
- 2045: Large-scale quantum

### 4. Dynamical Decoupling

- Séquences de pulses
- Protéger la cohérence

```
from qiskit.ignis.mitigation \
    import CompleteMeasFitter

# Créer les circuits de calibration
cal_circuits, state_labels = \
    meas_calibration_circuits(
        qr=qr,
        circlabel='mcal'
    )

# Exécuter
cal_results = execute(
    cal_circuits,
    backend,
    shots=1000
).result()
```



```
# Fitter
meas_fitter = CompleteMeasFitter(
    cal_results,
    state_labels
)

# Appliquer
mitigated_results = \
    meas_fitter.filter.apply(results)
```

## PARTIE 13 : PERSPECTIVES FUTURES

### L'AVENIR DU CALCUL QUANTIQUE



### **Court terme (2024-2027) :**

- 1000-5000 qubits NISQ
- VQE/QAOA production
- Premières applications industrielles
- Amélioration des taux d'erreur

### **Moyen terme (2027-2035) :**

- Correction d'erreurs efficace
- 10k-100k qubits logiques
- Shor pour petits nombres
- Applications en chimie mature





### **Long terme (2035+) :**

- Millions de qubits
- RSA-2048 cassé
- Simulation moléculaire complexe
- IA quantique





### **Incertitudes :**

- Technologie gagnante ?







-  Maturité actuelle
  - Temps de développement
-  Bonnes portes 2-qubits
  - Découvertes imprévues
-  Nécessite  $< 20$  mK
-  Cohérence limitée




## 2. Ions piégés (IonQ, Quantinuum)

-  Excellente fidélité
-  Connectivité totale
-  Portes lentes
-  Scalabilité difficile

## 3. Photonique (Xanadu, PsiQuantum)



-  Température ambiante
-  Networking facile
-  Pertes photoniques
-  Détection complexe

## 4. Atomes neutres (Pasqal, QuEra)

-  Scalabilité prometteuse
-  Géométries flexibles
-  Technologie jeune



## 5. Topologique (Microsoft)

-  Correction d'erreurs intrinsèque
-  Pas encore démontré
- Quantum neural networks
- Quantum GANs
- Feature maps quantiques

**Verdict :** Probablement plusieurs technologies pour différentes applications

## 2. Météorologie

- Simulation de systèmes chaotiques
- Prévisions à long terme
- Modèles climatiques

## 3. Biologie

- Protein folding
- Drug design
- Génomique
- Dynamique moléculaire

## 4. Énergie

- Matériaux pour batteries
- Catalyseurs
- Fusion nucléaire
- Supraconducteurs haute température

## 5. Sécurité



- Cryptographie post-quantique
- QKD networks
- Blockchain quantique

## **PARTIE 14 : RESSOURCES D'APPRENTISSAGE**

**CONTINUER À APPRENDRE**



- Auteur: Jack Hidary
- Pratique avec code
- Excellente introduction

### **"Dancing with Qubits"**

- Auteur: Robert Sutor
- Très accessible
- Peu de maths

### **"Quantum Computing for Everyone"**

- Auteur: Chris Bernhardt
- Grand public
- Bien illustré

### **Avancés :**

### **"Quantum Computation and Quantum Information"**

- Nielsen & Chuang
- Bible du domaine
- Très complet

### **"Quantum Computing: A Gentle Introduction"**

- Rieffel & Polak
- Approche pédagogique



- Niveau universitaire
- "Quantum Computing" (MIT)
- "An Introduction to Quantum Computing"
- "The Quantum Internet" (TU Delft)
  - Kaye, Laflamme, Mosca
- "Quantum Cryptography" (Caltech)
  - Focus algorithmes
- 🎓 **Coursera :**
  - Rigueur mathématique
  - "Quantum Mechanics and Quantum Computation" (Berkeley)
  - "Physical Basics of Quantum Computing" (St Petersburg)
- 🎓 **Qiskit Textbook :**
  - Gratuit et interactif
  - Excellente ressource
  - [qiskit.org/textbook](https://qiskit.org/textbook)

## Plateformes :

- 🎓 **IBM Quantum Learning :**
  - Cours structurés
  - Certificats
  - Accès aux machines
- 🎓 **Microsoft Learn :**
  - Q# tutorials
  - Azure Quantum





🎓 **Brilliant.org** • [qiskit.org/documentation](https://qiskit.org/documentation)

- Cours interactifs
- Très complet
- Gamification
- API reference
- Très pédagogique



**Cirq Docs**

- [quantumai.google/cirq](https://quantumai.google/cirq)
- Google's framework
- TensorFlow Quantum



**PennyLane**

- [pennylane.ai](https://pennylane.ai)
- Quantum ML
- Excellents tutorials

**Papers :**



**arXiv.org :**

- Section quant-ph
- Dernières recherches
- Preprints



**Quantum Journal**

- Open access
- Peer-reviewed



- Très bonne qualité
- Support Nature Quantum Information
- Challenges réguliers
- 💡 **Quantum Computing Stack Exchange**

- Questions/réponses
- Experts
- Archives précieuses

#### 💡 **Reddit :**

- r/QuantumComputing
- r/Qiskit
- Discussions variées

#### **Events :**

##### 🎪 **Qiskit Hackathons**

- 2-3 par an
- Projets en équipe
- Prizes

##### 🎪 **Q2B Conference**

- Industrie
- Networking



- Talks d'experts
  - Intègre a QISKit
  - Plusieurs backends
- Académique
- Physics focus
  - C++ backend



## APS March Meeting



## ProjectQ

- Très rapide
- Emulation



## QuTiP

- Dynamique quantique
- Systèmes ouverts
- Visualisation

## Cloud :



## IBM Quantum

- Machines réelles
- Gratuit (limité)
- Queue system



## Azure Quantum

- Multiple providers
- Crédits gratuits



## Certifications :



### IBM Quantum Developer

- Certification officielle
- Exam-based
- Reconnu



### Amazon Braket

- Q#
- IonQ, Rigetti, D-Wave
- Pay-per-shot
- Notebooks



### Microsoft Quantum :

- Q# certification
- Azure Quantum

## Métiers :



Quantum Software Engineer



Quantum Algorithm Researcher



Quantum

Hardware Engineer



Quantum Application Specialist

## Entreprises qui recrutent :

- IBM Quantum
- Google Quantum AI
- Microsoft Quantum
- Amazon Braket
- IonQ, Rigetti, D-Wave
- ☐ Startups (Zapata, QC Ware, etc.)
- ☒ Finance (JPMorgan, Goldman)

- Pharma (Roche, Biogen)

**Salaires : 80k—200k+ selon expérience**

## **PARTIE 15 : CONCLUSION**

**L'AVENIR EST QUANTIQUE**



- Superposition
- Intrication
- Interférence

## **2. Speedup varie selon l'algorithme :**

- Exponentiel (Shor)
- Quadratique (Grover)
- Variable (VQE, QAOA)

## **3. Applications réelles émergent :**

- Chimie (VQE)
- Optimisation (QAOA)
- Finance, Logistique

## **4. Défis restants :**





- Correction d'erreurs
- Scalabilité
- Décohérence

## **5. L'ère NISQ est là :**






- Algorithmes adaptés
- Mitigation d'erreurs
- Applications limitées mais réelles








## 6. Mécanisme quantique de base

-  Qubits et portes
-  États simples
-  Deutsch à voir
-  Bernstein-Vazirani

## Niveau 2 : Algorithmes Classiques (2-3 mois)

-  Grover (détail)
-  Shor (détail)
-  QPE et QFT
-  Simulations
-  Exercices pratiques

## Niveau 3 : NISQ (3-6 mois)

-  VQE
-  QAOA
-  Quantum ML
-  Error mitigation
-  Projets réels

## Niveau 4 : Spécialisation (6+ mois)

- Recherche ou industrie



## Immédiatement :

- Domaine spécifique
- 1. ⚡ Installer Qiskit
- 2. ⚡ Lire Qiskit Textbook (ch. 1-3)
- Contribution communauté
- 3. ⚡ Créer compte IBM Quantum
- 4. ⚡ Implémenter Deutsch-Jozsa
- 5. ⚡ Rejoindre Qiskit Slack

## Cette semaine :

1. 📅 Grover complet (exercice 1)
2. 📅 QFT implementation (exercice 3)
3. 📅 Regarder 3 vidéos recommandées
4. 📅 Lire 1 paper sur arXiv

## Ce mois :

1. 📅 VQE pour  $H_2$  (exercice 4)
2. 📅 QAOA Max-Cut (exercice 5)
3. 📅 Projet final comparatif
4. 📅 Contribuer à open source
5. 📅 Assister à 1 webinar

## Long terme :

- Specialization choisie
- Certification













- Networking
- Carrière quantique !

## RESSOURCES DE CETTE PRÉSENTATION

### Fichiers et Code :

-  Tous les exemples de code
-  Notebooks Jupyter
-  Solutions des exercices
-  Slides en PDF

### Liens Importants :

-  [Qiskit Textbook](#)
-  [IBM Quantum Lab](#)
-  [Qiskit Slack](#)
-  [Quantum Computing Stack Exchange](#)



**QUESTIONS ?**

**MERCI DE VOTRE ATTENTION !**

**Contact et Support :**

- Email : [alainlioret@gmail.com](mailto:alainlioret@gmail.com)

**Continuons l'aventure quantique ensemble ! 🚀**





