Algorithme de recherche par marche quantique

Simon Flory-Bardin, Hanwen Liu, Raphaël Misme, Yohan Nunes
Licence de Mathématiques
Sorbonne Université

24 avril 2023

Résumé

Ce projet a pour but de découvrir les rudiments des algorithmes quantiques, notamment leurs intérêts par rapport aux algorithmes classiques dans le cas de marches ou de recherche.

Table des matières

Introduction

Nous avons fait le choix de ne pas utiliser les notations du document de référence ("à la physicienne", notation de Dirac, ou "bra-ket") pour la rédaction de notre rapport. Quoique parfois moins encombrante visuellement, nous avons préféré garder des notations que nous avons l'habitude d'utiliser.

Notations :

- par convention, les vecteurs sont colonnes
- base canonique : $(e_i)_{i\in[\![0,n-1]\!]}$
- vecteur : v
- matrice : A
- transconjuguée : $v^\dagger$
- produit scalaire : $\langle x, y\rangle$
- produit matriciel : Av, $A \cdot v$, $v \cdot v^\dagger$, $vv^\dagger$
- produit tensoriel : $v \otimes w$, $A \otimes B$
- $\mathcal{H} = \mathbb{C}^2$, espace Hilbertien de dimension 2

Tout d'abord, nous présenterons le principe d'une marche dans le cas le plus simple, sur une ligne infinie ($\mathbb{Z}$), dans le cas classique avec lequel nous avons l'habitude , puis dans le cas quantique, qui nous intéresse ici. Nous présenterons ensuite l'algorithme de Grover, un algorithme de recherche quantique, plus efficace que l'algorithme classique pour rechercher un élément dans une liste finie. L'idée est ensuite de complexifier les objets dans lequels nous allons effectuer des marches ou des recherches : cycles, grilles, hypercubes.

1 Marche discrète

1.1 Marche discrète classique

Une marche aléatoire est un modèle mathématique representant le déplacement aléatoire d'une particule le long d'un graphe au cours d'un temps discrétisé en pas.

Soit q une particule, elle est initialement en un point i de $\mathbb{Z}$. Elle peut se déplacer sur le point suivant $i+1$ avec une probabilitée p, ou se déplacer sur le point précédent $i-1$ avec une probabilitée $1-p$. Ses déplacements sont indépendants les uns des autres. Intéressons nous à la positions de la particule q dans le temps. Soit n la position de la particule à l'instant t et $\mathcal{X}_k$ qui décrit le $k^{\text{ème}}$ pas.

$$\mathcal{X}_k = \begin{cases} +1 \text{ avec une probabilitée p} \\ -1 \text{ avec une probabilitée 1-p} \end{cases}$$

On peut remarquer que la probabilité d'être à la n-ième position se rapproche de la loi binomiale. On a donc pour $p = \frac{1}{2}$:

$$p(t,n) = \frac{1}{2^t} \begin{pmatrix} n \\ \frac{t+n}{2} \end{pmatrix}$$

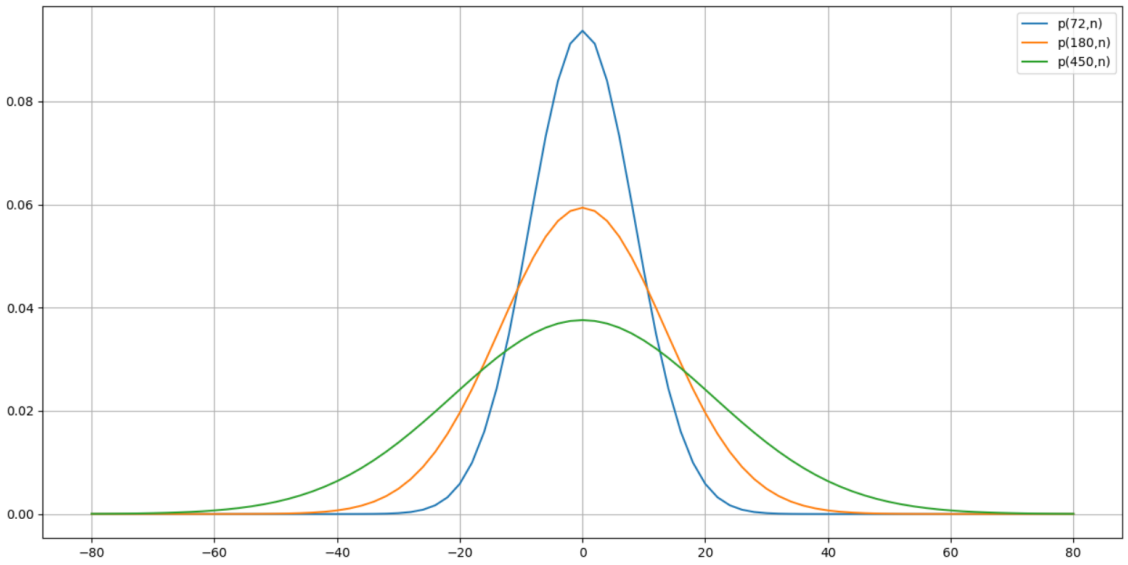

FIGURE 1 – Probabilité de distribution de la marche classique

1.2 Description de l'algorithme de la marche classique

La fonction `pas` sert à déplacer la particule d'une cellule de la matrice à une autre en la déplaçant sur la cellule suivante ou la précédente.
La fonction `marche` sert à répéter N fois l'algorithme `pas` afin de pouvoir observer sur quelles celulles de la matrice, la particule a le plus de chances d'être.
Le troisème algorithme `prob_classic` lui calcule la probabilité p dont nous avons parlé précédemment qui se rapproche d'une probabilitée d'une loi binomiale.
Le dernier algorithme de la marche discrète classique lui permet de représenter le

graphique ci-dessus, qui permet d'observer la probabilitée sur un échantillon t en fonction de n.

1.3 Marche discrète quantique

La mécanique quantique a modifié la perception du monde physique microscopique. Il est désormais difficile de croire que, selon les lois de la mécanique quantique, des possibilités contradictoires peuvent se produire simultanément (superposition, interférence, intrication).

C'est ainsi qu'est née l'informatique quantique. Contrairement à la marche classique, la marche quantique superpose les possibilités les unes aux autres à chaque étape, au lieu de ne présenter qu'une seule possibilité. Une telle caractéristique peut permettre d'élaborer des algorithmes efficaces pour des problèmes spécifiques.

Dans un ordinateur classique, l'information est stockée dans les bits, dont la valeur est soit 0, soit 1. Un bit quantique appelé aussi qubit a, quand à lui, deux états quantiques $|0>$ et $|1>$, ayant une différence d'énergie, et peut être à la fois dans ces deux états. Au cours d'un algorithme il y a une succession d'opérations dites de "portes logiques".

La marche classique permet de vérifier une valeur à la fois alors que la marche quantique permet d'en vérifier un ou plusieurs éléments à la fois ce qui permet d'obtenir l'élément recherché bien plus rapidement.

Un système quantique est composé de N qubits, est une base de Hilbert où on utilise une base orthonormale $\{|0>, |1>\}$ de $\mathcal{H}^2$. Un circuit quantique est un graphe représentant une composition d'opérateurs unitaires. Un calcul quantique est un circuit quantique appliqué à un état quantique initial d'un système quantique.

La porte quantique que nous allons utiliser par la suite est une des portes quantiques les plus utilisées. Il s'agit de la porte de Hadamard, c'est une porte qui agit sur un seul qubit. Cette porte est d'abord définie sur l'espace $\mathcal{H}$ puis étendue sur tout l'espace $\mathcal{H}^N$ par produit tensoriel.

Là où l'état classique de notre particule qui se déplace est simplement un entier $n \in \mathbb{Z}$ qui indique sa position, notre état quantique est le produit tensoriel de deux vecteurs, deux composantes : celle de la position et celle de la « pièce ». On l'appelle ainsi car c'est cet opérateur qui va définir le sens de la marche. On se place donc sur l'espace $\mathcal{H}^2 \otimes \mathcal{H}^{\mathbb{Z}}$, les bases canoniques sont $(a_i)_{i\in\{0,1\}}$ et $(e_n)_{n\in\mathbb{Z}}$. Là où, avec la pièce classique, si la prticule est à l'état n, elle devient $n+1$ si la pièce donne face, $n-1$ si elle donne pile. Ici, on a un opérateur S qui va faire office de pièce qui agit comme suit :

$$\begin{cases} S(a_0 \otimes e_n) = a_0 \otimes e_{n+1} \\ S(a_1 \otimes e_n) = a_1 \otimes e_{n-1} \end{cases}$$

On commence généralement avec l'état $\psi(0) = a_0 \otimes e_0$. Ensuite, on va définir l'opérateur qui caractérise la marche. Il va d'abord agir sur la pièce, comme si on la lançait, puis déplacer la particule en fonction. On définit l'opérateur de Hadamard

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

Pour passer de l'étape t, où notre particule est dans l'état $\psi(t)$, à l'étape $t+1$, on a donc :

$$\psi(t+1) = U\psi(t) = S(H \otimes I_2)\psi(t)$$

et alors pour tout t, $\psi(t) = U^t\psi(0)$.

De façon plus générale, on a pour tout t,

$$\psi(t) = \sum_{n \in \mathbb{Z}} (A_n(t)a_0 + B_n(t)a_1) \otimes e_n$$

On peut alors calculer la probabilité de se trouver en tout point n à tout instant t :

$$p(t, n) = |A_n(t)|^2 + |B_n(t)|^2$$

Par exemple, la probabilité de distribution pour $t = 100$ donne Fig.2

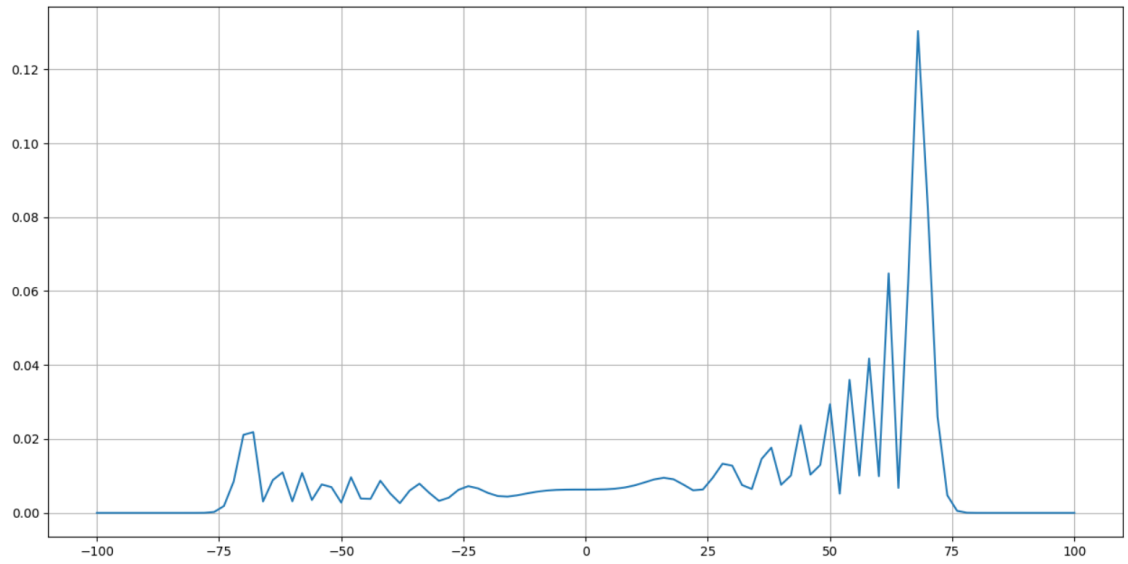

FIGURE 2 – Probabilité de distribution pour la marche quantique, avec état initial $\psi = |0\rangle|0\rangle$

2 Algorithme de Grover

Notations : Tout au long de cette partie, on utilise un certain $N \in \mathbb{N}$, $N \geq 2$, et les notations suivantes seront utilisées :

— $(e_i)_{i\in[\![0,N-1]\!]}$ la base canonique de $\mathcal{H}^N$

— $t = \left\lfloor \frac{\pi}{4}\sqrt{N} \right\rfloor$

— $d = \frac{1}{\sqrt{N}} \sum_{i=0}^{N-1} e_i$

— Base de $\mathcal{H}^2$: (a_0, a_1)

— Avec les notations précédentes, dans $\mathcal{H}^2$, on note $v^- = \frac{a_0 - a_1}{\sqrt{2}}$

Dans cette partie, nous allons nous concentrer sur un algorithme quantique spécifique : l'algorithme de Grover, conçu pour effectuer une recherche dans une base de donnée non ordonné de taille N.

Avec un ordinateur classique il faudrait parcourir la base de donnée terme à terme jusqu'à trouver le bon élément. On a donc un algorithme de complexité $O(N)$.

L'algorithme de Grover est quand à lui un algorithme probabiliste de complexité $O\left(\sqrt{N}\right)$ renvoyant le bon résultat avec une probabilité supérieure à $1 - \frac{1}{N}$.

Nous modéliserons notre base de donnée de taille N comprenant l'élément à trouver i_0 par la suite des entiers allant de 0 à $N-1$. Par abus de notation nous confondrons i_0 et son indice (inconnu) dans $[\![0, N-1]\!]$.

2.1 Description de l'algorithme de Grover

L'idée de l'algorithme est de se placer dans une base orthonormée $(e_i)_{i\in[\![0,N-1]\!]} \in \mathcal{H}^N$, chaque vecteur représentant un élément de notre base de données. Pour pouvoir travailler nous tensorisons ensuite $\mathcal{H}^N$ avec $\mathcal{H}^2$. Nous appelerons (a_0, a_1) la base de $\mathcal{H}^2$. Dans les fait les vecteurs $\psi \in \mathcal{H}^N \otimes \mathcal{H}^2$ que nous utiliserons resterons toujours des tenseurs fondamentaux (voir partie Analyse de l'Algorithme). On introduit donc ici une notation :

$$(\psi^0, \psi^1) \in \mathcal{H}^N \times \mathcal{H}^2 \text{ sont défini comme l'unique couple tel que : } \psi = \psi^0 \otimes \psi^1$$

Nous disposons (voir la partie Marche discrete quantique) d'une fonction probabiliste, appelé mesure, noté $\mathcal{M}$ tel que : $\forall \psi \in \mathcal{H}^N \otimes \mathcal{H}^2$ unitaire :

$$\mathcal{M}(\psi^0) = i \text{ avec probabilité } \langle \psi^0, e_i \rangle^2 \quad \forall i \in [\![0, N-1]\!]$$

Notre objectif est donc, en partant d'un vecteur unitaire ψ_0, d'obtenir un vecteur unitaire ψ_n avec ψ_n^0 aussi colinéaire que possible avec e_{i_0}.

Pour ce faire nous commençons par un vecteur unitaire de départ :

$$\psi_0 := d \otimes v^- := \left(\frac{1}{\sqrt{N}} \sum_{i=0}^{N-1} e_i \right) \otimes \left(\frac{a_0 - a_1}{\sqrt{2}} \right)$$

On construit ensuite une suite de vecteurs $(\psi_i)_{i \in [\![0,t]\!]}$ par :

$$\psi_{n+1} := \mathcal{U}\psi_n := \mathcal{R}_d \mathcal{R}_f \psi_n$$

où $\mathcal{U}$ est défini comme la composée de deux opérateurs unitaires $\mathcal{R}_f$ et $\mathcal{R}_d$, où

$$\mathcal{R}_f(e_i \otimes a_{j \in \{0,1\}}) := \begin{cases} e_{i_0} \otimes a_j, & \text{si } i \neq i_0 \\ e_i \otimes a_{1-j}, & \text{sinon} \end{cases}$$

$$\mathcal{R}_d(e_i \otimes a_j) := (2dd^\dagger e_i - e_i) \otimes a_j$$

Notre vecteur final, celui sur lequel nous pratiquerons la mesure se trouve être ψ_t pour $t = \left\lfloor \frac{\pi}{4}\sqrt{N} \right\rfloor$

2.2 Analyse de l'algorithme

Dans cette partie, nous expliquons l'origine de la constante $t = \left\lfloor \frac{\pi}{4}\sqrt{N} \right\rfloor$.

On observe d'abord que les opérateurs $\mathcal{R}_f$ et $\mathcal{R}_d$ sont unitaires, ce qui fait de $\mathcal{U}$ un opérateur unitaire.

On vérifie que depuis l'état initial ψ_0 jusqu'à la fin de l'algorithme, on évolue dans un espace de dimension deux : $\text{Vect}(x_0, d) =: V$. Rappelons que $\mathcal{R}_d = (2dd^\dagger - I_N) \otimes I_2$. Soit $i \in [\![0, N-1]\!]$. Si $i \neq i_0$:

$$\mathcal{R}_f(e_i \otimes v^-) = (e_i \otimes v^-)$$

Si $i = i_0$:

$$\begin{aligned} \mathcal{R}_f(e_i \otimes v^-) &= \frac{\mathcal{R}_f(e_{i_0} \otimes a_0) - \mathcal{R}_f(e_{i_0} \otimes a_1)}{\sqrt{2}} \\ &= \frac{e_{i_0} \otimes a_1 - e_{i_0} \otimes a_0}{\sqrt{2}} \\ &= -e_i \otimes v^- \end{aligned}$$

De plus, $\forall i \in [\![0, N-1]\!]$

$$dd^\dagger e_i = \frac{1}{\sqrt{N}} d$$

Alors :

$$\begin{aligned}
\mathcal{U}(d \otimes v^-) &= \frac{1}{\sqrt{N}} \cdot \mathcal{U}\left(e_{i_0} \otimes v^- + \sum_{i \neq i_0} e_i \otimes v^- \right) \\
&= \frac{1}{\sqrt{N}} \cdot \mathcal{R}_d \mathcal{R}_f \left(e_{i_0} \otimes v^- + \sum_{i \neq i_0} e_i \otimes v^- \right) \\
&= \frac{1}{\sqrt{N}} \cdot \mathcal{R}_d \left(-e_{i_0} \otimes v^- + \sum_{i \neq i_0} e_i \otimes v^- \right) \\
&= \frac{1}{\sqrt{N}} \cdot \mathcal{R}_d \left(\left(-e_{i_0} + \sum_{i \neq i_0} e_i \right) \otimes v^- \right) \\
&= \frac{1}{\sqrt{N}} \cdot \left(\left(-2dd^\dagger e_{i_0} + I_N e_{i_0} + \sum_{i \neq i_0} \left(2dd^\dagger e_i - I_N e_i \right) \right) \otimes v^- \right) \\
&= \frac{1}{\sqrt{N}} \cdot \left(\left(\frac{-2}{\sqrt{N}} d + e_{i_0} + \sum_{i \neq i_0} \left(\frac{2}{\sqrt{N}} d - e_i \right) \right) \otimes v^- \right) \\
&= \frac{1}{\sqrt{N}} \cdot \left(\left(\frac{-2}{\sqrt{N}} d + e_{i_0} + (N-1) \frac{2}{\sqrt{N}} d - (\sqrt{N} d - e_{i_0}) \right) \otimes v^- \right) \\
&= \left(\left(\frac{1}{N} - 1 + (N-2) \frac{2}{N} \right) d + \frac{2}{\sqrt{N}} e_{i_0} \right) \otimes v^- \\
&= \left(\frac{N-4}{N} d + \frac{2}{\sqrt{N}} e_{i_0} \right) \otimes v^-
\end{aligned}$$

Et

$$\begin{aligned}
\mathcal{U}(e_{i_0} \otimes v^-) &= \mathcal{R}_d \mathcal{R}_f (e_{i_0} \otimes v^-) \\
&= \mathcal{R}_d (-e_{i_0} \otimes v^-) \\
&= \left(-2dd^\dagger e_{i_0} + I_N e_{i_0} \right) \otimes v^- \\
&= \left(\frac{-2}{\sqrt{N}} d + e_{i_0} \right) \otimes v^-
\end{aligned}$$

On remarque alors que sous l'action de $\mathcal{U}$, la deuxième composante v^- n'est jamais modifiée. On peut donc se restreindre à l'action de $\mathcal{U}$ sur la première composante, sur

l'espace $\mathcal{H}^N$, qui se traduit par une matrice. Cette matrice est unitaire et diagonalisable, et le sous espace $V = \text{Vect}(d, e_{i_0})$ est stable par l'action de $\mathcal{U}$. On note alors U le bloc correspondant à ce sous-espace, et on cherche à diagonaliser U. Dans la base (d, e_{i_0}), U est de la forme :

$$U \sim \begin{pmatrix} \dfrac{N-4}{N} & \dfrac{2}{\sqrt{N}} \\ \dfrac{-2}{\sqrt{N}} & 1 \end{pmatrix}, \text{ et alors } \begin{cases} \text{tr}(U) = \dfrac{N-4}{N} + 1 = 2 - \dfrac{4}{N} \\ \det(U) = \dfrac{N-4}{N} + \dfrac{4}{N} = 1 \end{cases}$$

Le polynôme caractéristique de U est alors :

$$\chi_U(X) = X^2 - \text{tr}(U)X + \det(U) = X^2 - \left(2 - \frac{4}{N}\right) X + 1$$

Le déterminant est $\Delta = \left(2 - \dfrac{4}{N}\right)^2 - 4 = \dfrac{16}{N^2} - \dfrac{8}{N} = \dfrac{8}{N}\left(\dfrac{2}{N} - 1\right)$. Comme $N \geq 2$, $\Delta < 0$, donc χ_U admet deux racines complexes non réelles conjugués α et $\overline{\alpha}$. Elles sont de module 1, car $\det(U) = 1 = \alpha\overline{\alpha} = |\alpha|^2$. On peut alors noter ces racines $e^{i\theta_N}$ et $e^{-i\theta_N}$.

$$U \sim \begin{pmatrix} e^{i\theta_N} & 0 \\ 0 & e^{-i\theta_N} \end{pmatrix}$$

On essaie maintenant d'extraire θ_N. $\text{tr}(U) = 2 - \dfrac{4}{N} = e^{i\theta_N} + e^{-i\theta_N} = 2\cos(\theta_N)$. Alors $\cos(\theta_N) = 1 - \dfrac{2}{N}$, donc $\theta_N = \arccos\left(1 - \dfrac{2}{N}\right)$. On donne le développement limité de $x \mapsto \arccos(1-x)$ en 0^+ : $\arccos(1-x) \underset{x\to 0^+}{\sim} \sqrt{2x}$. En effet, en posant $g\colon [0,1] \to \mathbb{R}, u \mapsto \arccos(1-u^2)$, on a $g'\colon u \mapsto \dfrac{2}{\sqrt{2-u^2}}$. En remarquant que g est dérivable en 0, et que g est $\mathcal{C}^1$, on peut écrire le développement de Taylor en 0 : $g(u) = g(0) + g'(0)u + o(u) = \arccos(1) + \sqrt{2}u + o(u) = \sqrt{2}u + o(u)$, et donc $\arccos(1-u^2) \underset{u\to 0^+}{\sim} \sqrt{2}u$, d'où le résultat. On en déduit que

$$\theta_N \underset{N\to\infty}{\sim} \frac{2}{\sqrt{N}} \tag{2.1}$$

Ce graphe est obtenu à partir de la fonction `trace_grover2`. Ici, $N = 1000$ et comme indiqué sur le graphe, $t = \left\lfloor \dfrac{\pi}{4}\sqrt{N} \right\rfloor = 24$. On remarque qu'à l'étape t, $\|\psi^t - e_{i_0}\|_\infty$ est minimigne, et réaugmente si l'on continue à calculer après l'étape t.

Mais à quoi correspond ce θ_N ?

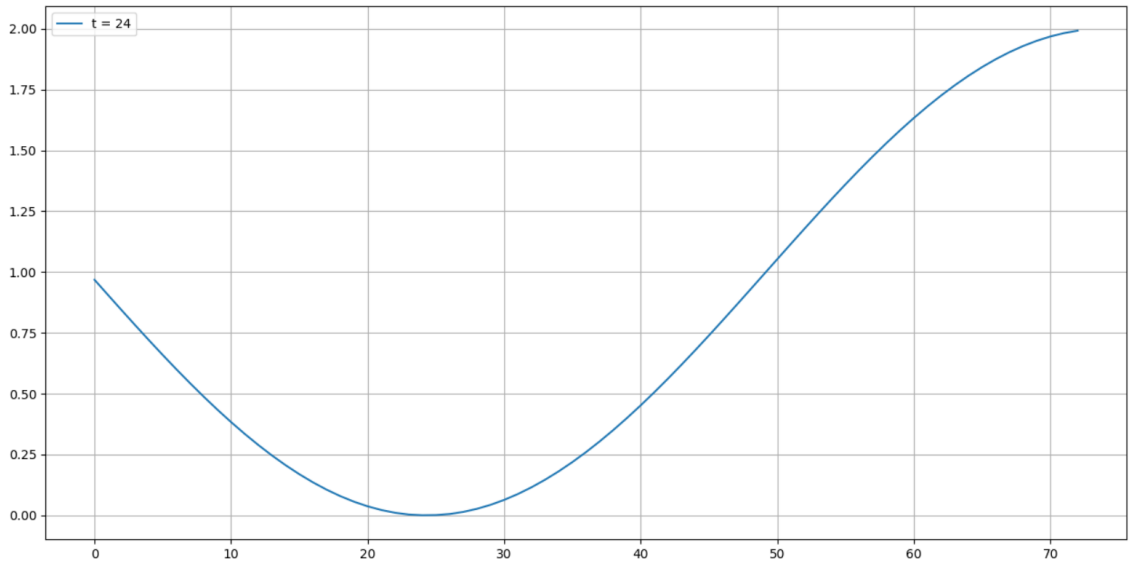

FIGURE 3 – Évolution de $\|\psi^n - e_{i_0}\|_\infty$ en fonction du nombre n d'étapes

2.3 Analyse de l'algorithme dans l'espace réel bidimensionnel

Dans cette partie, nous expliquons les implications géométriques de l'algorithme dans la base orthonormée $Vect(\boldsymbol{e_{i_0}}, \boldsymbol{f})$ où $\boldsymbol{f} = \frac{1}{\sqrt{N-1}} \sum_{i \neq i_0} e_i$ et e_{i_0} est le vecteur cible.

Le vecteur unitaire dans cette base est donc :

$$\boldsymbol{d} = \frac{\sqrt{N-1}}{\sqrt{N}} \cdot \boldsymbol{f} + \frac{1}{\sqrt{N}} \cdot \boldsymbol{e_{i_0}}$$

Soit $\dfrac{\theta}{2}$ l'angle entre les vecteurs $\boldsymbol{f}$ et $\boldsymbol{d}$, on obtient :

$$\boldsymbol{d} = \cos\frac{\theta}{2} \cdot \boldsymbol{f} + \sin\frac{\theta}{2} \cdot \boldsymbol{e_{i_0}}$$

Donc en utilisant la définition de $\mathcal{R}_d = 2\,\boldsymbol{d}\boldsymbol{d}^\dagger - I_N$ et $\mathcal{R}_f = I_N - 2\boldsymbol{e_{i_0}}\boldsymbol{e_{i_0}}^\dagger$, on obtient

$$r_d = \begin{pmatrix} \cos\theta & \sin\theta \\ \sin\theta & -\cos\theta \end{pmatrix} \text{ et } r_f = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

L'essence de r_f et r_d est le changement de symétrie dû à l'addition et à la projection entre les vecteurs. La composition des deux produit la rotation U, où $U = r_d r_f$.

Chaque application de U sur $\boldsymbol{d}$ produit une rotation d'angle θ qui converge finalement vers le vecteur cible $\boldsymbol{e_{i_0}}$ après un nombre t_{run} suffisant d'applications :

$$\theta t_{run} + \frac{\theta}{2} = \frac{\pi}{2}$$

$$t_{run} = \frac{\pi}{2\theta} - \frac{1}{2}$$

Nous avons trouvé également $t = \left\lfloor \frac{\pi}{4}\sqrt{N} \right\rfloor$ (en utilisant 2.1) et donc $\theta = \theta_N$. Ainsi, θ_N dans la section précédente correspond à un angle de rotation θ.

Nous allons nous servir de ce point de vue géométrique pour calculer p_N la probabilité d'obtenir le bon i_0 a la fin de l'algorithme. Par définition

$$p_N = \langle \psi^t, e_{i_0} \rangle^2 = \langle U^t d, e_{i_0} \rangle^2$$

On utilise ensuite la formule classique :

$$\langle u, v \rangle = \|u\| \|v\| \cos(\widehat{u, v})$$

Or $\widehat{U^t d, e_{i_0}} + \widehat{U^t d, f} = \widehat{e_{i_0}, f} = \frac{\pi}{2}$ et $\|U^t d\| = \|_{i_0}\| = 1$ Donc :

$$p_n = \cos(\frac{\pi}{2} - \widehat{U^t d, f}) = \sin(\widehat{U^t d, f})$$

Comme U équivaut à une rotation d'angle θ_N et que l'angle initial de d avec f vaut $\frac{\theta}{2}$ on obtient in fine cette formule :

$$p_N = \sin(t\theta + \frac{\theta}{2})^2$$

Comme t est optimal et sin croissant sur $[0, \frac{\pi}{2}]$:

$$\frac{\pi}{2} \geq t\theta + \frac{\theta}{2} \geq \frac{\pi}{2} - \theta$$

$$p_N \geq \sin(\frac{\pi}{2} - \theta)^2$$

$$p_N \geq \cos(\theta)^2$$

$$p_N \geq 1 - \sin(\theta)^2$$

On utilise maintenant que $\theta \approx \frac{2}{\sqrt{N}}$ et $\sin(x) \simeq x$ quand $x \to 0$ donc pour N assez grand :

$$p_N \geq 1 - \frac{4}{N}$$

3 Algorithmes de recherche sur un cycle ou une grille

3.1 Marche sur un cycle

On étudie maintenant la marche quantique sur un cycle, c'est-à-dire une ligne finie, dont les extrémités sont reliées. Sur ce cycle sont placés N nœuds ($N \in \mathbb{N}^*$), on travaillera donc dans l'espace $\mathcal{H}^2 \otimes \mathcal{H}^N$, le premier étant l'espace de la pièce et le second de l'espace. On utilise l'opérateur de Hadamard

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

et on définit l'opérateur S qui agit sur $\mathcal{H}^2 \otimes \mathcal{H}^N$ par (en utilisant les notations de la section précédente) :

$$\forall (s,j) \in \{0,1\} \times [\![0, N-1]\!],\ S(a_s \otimes e_j) = a_s \otimes e_{j+(-1)^s}$$

où les calculs sur la variable j sont effectués modulo N. Enfin, on définit

$$U = S(H \otimes I_N)$$

Comme précédemment, on identifie le i^e nœud au vecteur e_i de la base canonique de $\mathcal{H}^N$. Comment obtenir la probabilité de se trouver en un point après un certain nombre d'itérations ? L'état à l'instant t s'exprime sous la forme :

$$\psi(t) = \sum_{j=0}^{N-1} (\psi_{0,j}(t) a_0 + \psi_{1,j}(t) a_1) \otimes e_j$$

La probabilité de se trouver en j après t étapes est donc

$$p_j(t) = |\psi_{0,j}(t)|^2 + |\psi_{1,j}(t)|^2$$

et pour tout t on a

$$\sum_{j=0}^{N-1} p_j(t) = 1$$

On peut déjà visualiser ce à quoi ressemble la marche sur un tel cycle, par itérations de l'opérateur U sur un état initial ψ_0. Par exemple, sur un cycle avec 200 nœuds, après 100, 200 et 300 itérations, avec un état initial $\psi_0 = a_0 \otimes e_0$, on obtient Fig. 4.

On remarque les similitudes avec la marche classique sur la ligne lorsque le nombre d'étapes est petit.

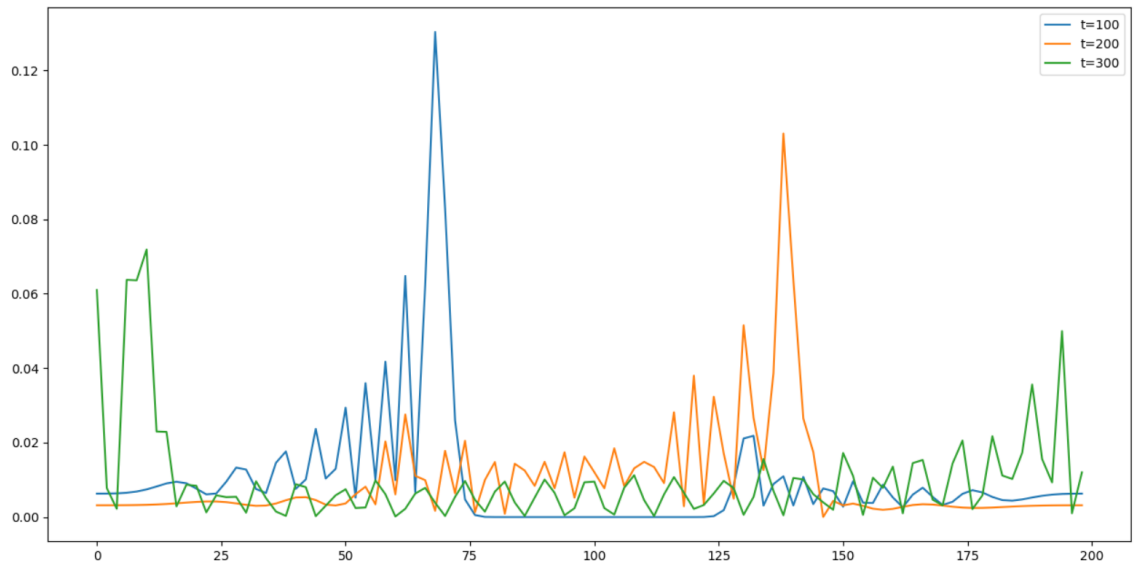

FIGURE 4 – Probabilité de distribution sur un cycle

3.2 Recherche sur un cycle

On veut maintenant appliquer un algorithme de recherche, comme on l'a fait précédemment avec l'algorithme de Grover sur une liste. On peut considérer sans perte de généralité que l'on cherche le vecteur e_0. On définit alors les opérateurs

$$R = I_N - 2e_0e_0^\dagger \text{ et } U' = UR$$

On cherche alors la probabilité de trouver le vecteur e_0 et le nombre d'étapes nécessaires pour obtenir la meilleure probabilité de réussite ; car comme dans l'algorithme de Grover, il existera toujours une probabilité que le vecteur trouvé par l'algorithme ne soit pas celui cherché.

L'état initial est

$$\psi(0) = a_0 \otimes D_P$$

où D_P est l'état diagonal de l'espace de position, c'est-à-dire

$$D_P = \frac{1}{\sqrt{N}} \sum_{j=0}^{N} e_j e_j^\dagger$$

On essaie alors de trouver le nombre optimal d'itérations à effectuer pour trouver le vecteur cherché avec une probabilité maximale. Pour la section précédente, avec l'algorithme de Grover, on avait $t_{opt} = \lfloor \frac{\pi}{4}\sqrt{N} \rfloor$.

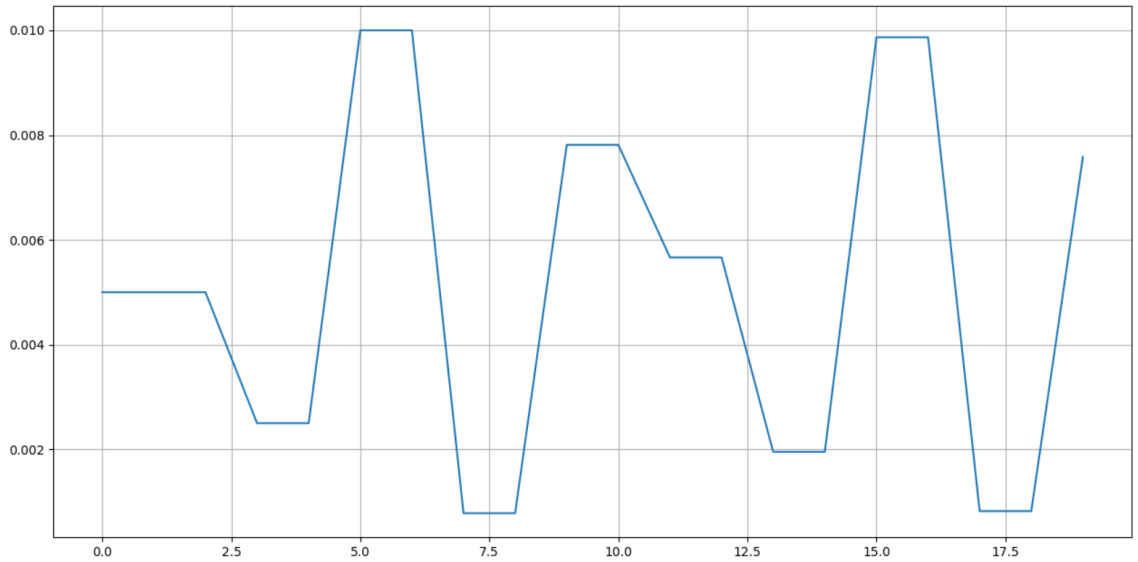

FIGURE 5 – $p_0(t)$ sur un cycle avec $N = 200$ nœuds

3.3 Marche sur une grille finie

On suppose maintenant que N est un carré parfait, et on se place sur une grille de taille $\sqrt{N} \times \sqrt{N}$, avec des conditions cycliques (*ie.* si l'on se déplace de $\sqrt{N}$ dans une direction, on se retrouve au point de départ). Notre espace de position a donc 2 dimensions désormais, et sa base canonique est $(e_i \otimes f_j)_{i,j \in [\![0,\sqrt{N}-1]\!]}$. L'espace de la pièce a aussi une dimension supplémentaire, et il a pour base $(a_d \otimes b_s)_{d,s \in \{0,1\}}$.

On définit l'opérateur S par :

$$S((a_d \otimes b_s) \otimes (e_i \otimes f_j)) = (a_d \otimes b_{s \oplus 1}) \otimes (e_{i+(-1)^s \delta_{d,0}} \otimes f_{j+(-1)^s \delta_{d,1}})$$

où $s \oplus 1 = \begin{cases} 0 \text{ si } s = 1 \\ 1 \text{ si } s = 1 \end{cases}$, et où $\delta_{d,s}$ est le delta de Kronecker.

L'opérateur pièce que l'on utilise n'est plus l'opérateur de Hadamard, mais la "pièce de Grover" :

$$G = \frac{1}{2} \begin{pmatrix} -1 & 1 & 1 & 1 \\ 1 & -1 & 1 & 1 \\ 1 & 1 & -1 & 1 \\ 1 & 1 & 1 & -1 \end{pmatrix}$$

On définit alors de la même façon l'opérateur $U = S(G \otimes I_N)$ qui caractérise l'évolution de la marche, et de façon similaire au cas du cycle, on peut trouver la probabilité de trouver en un nœud après un certain nombre d'étapes.

3.4 Recherche sur une grille finie

De la même façon que précédemment, on va définir l'opérateur R' qui exprime le fait de "marquer" le point de coordonnées $(0, 0)$, celui que l'on recherche.

$$R' = I - 2zz^\dagger$$

où

$$z = D_C \otimes (e_0 \otimes f_0)$$

avec

$$D_C = \frac{1}{2} \sum_{d,s=0}^{1} (a_d \otimes b_s)(a_d \otimes b_s)^\dagger$$

L'état initial est

$$\psi(0) = D_C \otimes D_P$$

où D_C est l'état diagonal de l'espace de la pièce, et D_P l'état diagonal de l'espace de position, c'est-à-dire

$$D_P = \frac{1}{\sqrt{N}} \sum_{i,j=0}^{\sqrt{N}} (e_i \otimes f_j)(e_i \otimes f_j)^\dagger$$

On peut encore une fois chercher le nombre d'itérations optimal.

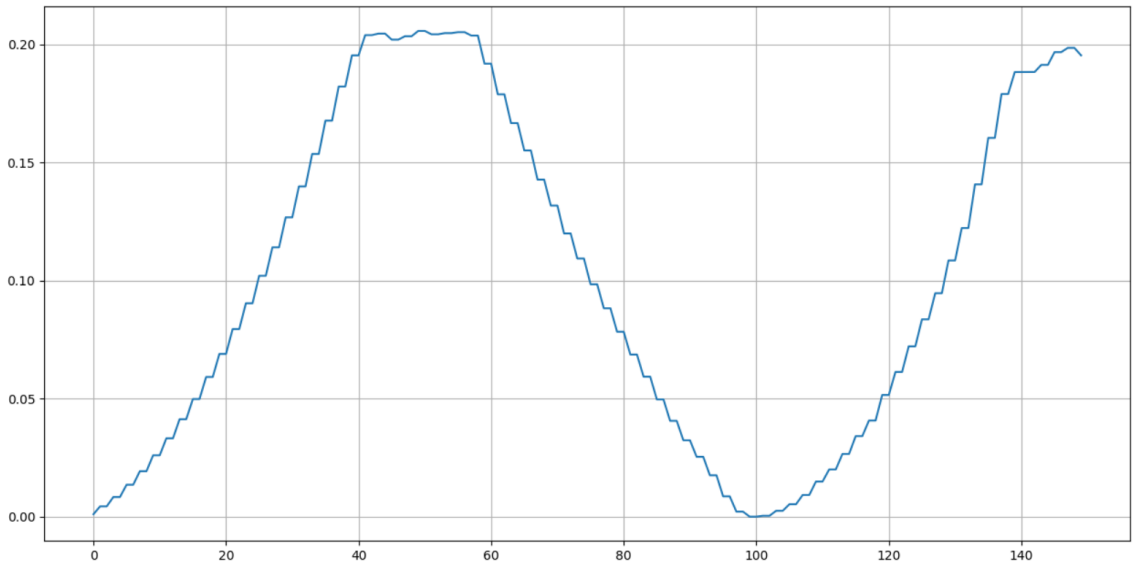

FIGURE 6 – $p_0(t)$ sur une grille de taille 30×30

4 Algorithmes de recherche spatiale pour un hypercube à n dimensions

4.1 Analyse de l'algorithme

Sans perte de généralité, nous prenons le vecteur $\mathbf{0}$ comme vecteur à rechercher.
L'expression de la probabilité de trouver ce vecteur $\mathbf{0}$ après t étapes est la suivante :

$$\begin{aligned} p(t) &= \langle \mathbf{0}, \boldsymbol{\psi}(t) \rangle^2 \\ &= \langle \mathbf{0}, (U')^t \boldsymbol{\psi}(0) \rangle^2, \end{aligned} \tag{4.1}$$

En se concentrant sur les vecteurs propres λ et λ', la décomposition spectrale de U' [1] serait :

$$U' = e^{i\lambda}\, \boldsymbol{\lambda}\boldsymbol{\lambda}^\dagger + e^{i\lambda'}\, \boldsymbol{\lambda'}\boldsymbol{\lambda'}^\dagger + U_{tiny}$$

$$(U')^t = e^{it\lambda}\, \boldsymbol{\lambda}\boldsymbol{\lambda}^\dagger + e^{it\lambda'}\, \boldsymbol{\lambda'}\boldsymbol{\lambda'}^\dagger + U_{tiny}^t, \tag{4.2}$$

Par conséquent, en utilisant l'équation 4.2 sur 4.1, on obtient :

$$p(t) = |e^{it\lambda} \langle \mathbf{0}, \boldsymbol{\lambda} \rangle \langle \boldsymbol{\lambda}, \boldsymbol{\psi}(t) \rangle + |e^{it\lambda'} \langle \mathbf{0}, \boldsymbol{\lambda'} \rangle \langle \boldsymbol{\lambda'}, \boldsymbol{\psi}(t) \rangle + \epsilon|^2, \tag{4.3}$$

Notre but est maintenant de trouver λ (λ') en supposant que nous avons déjà obtenu la décomposition spectrale de U , c'est-à-dire que nous supposons que l'ensemble des vecteurs $\boldsymbol{\psi_k}$ est une base orthonormée de vecteurs propres de U et que $e^{it\phi_k}$ sont les valeurs propres correspondantes.
Nous avons alors $I = \sum_k \boldsymbol{\psi_k}\boldsymbol{\psi_k}^\dagger$ [2], donc

$$\langle \mathbf{0}, \boldsymbol{\lambda} \rangle = \sum_k \langle \mathbf{0}, \boldsymbol{\psi_k} \rangle \langle \boldsymbol{\psi_k}, \boldsymbol{\lambda} \rangle \tag{4.4}$$

et

$$\langle \boldsymbol{\psi_k}, \boldsymbol{\lambda} \rangle = \frac{2\langle \mathbf{0}, \boldsymbol{\lambda} \rangle \langle \boldsymbol{\psi_k}, \mathbf{0} \rangle}{1 - e^{i(\lambda - \phi_k)}} \text{ [3]} \tag{4.5}$$

En utilisant l'équation 4.4 sur 4.5, et après avoir simplifié ces équations, pour tous $\phi_k \neq \lambda$, nous obtenons :

$$\sum_k \frac{2\langle \mathbf{0}, \boldsymbol{\psi_k} \rangle^2}{1 - e^{i(\lambda - \phi_k)}} = 1 \tag{4.6}$$

1. $U' = UR$
2. Completeness relation
3. Le résultat de l'ex 9.1 du livre "Quantum Science and Technology by Renato Portugal "

Étudions la partie imaginaire de l'équation (4.6), on a :

$$\sum_k 2\langle \mathbf{0}, \boldsymbol{\psi_k}\rangle^2 \frac{\sin\lambda - \phi_k}{1-\cos\lambda - \phi_k} = 0 \,^4 \tag{4.7}$$

Divisons la somme (4.7) en deux parties : la somme des termes tels que $\phi_k = 0$ et $\phi_k \neq 0$:

$$\sum_{\phi_k=0} \langle \mathbf{0}, \boldsymbol{\psi_k}\rangle^2 \frac{\sin\lambda}{1-\cos\lambda} + \sum_{\phi_k\neq 0} \langle \mathbf{0}, \boldsymbol{\psi_k}\rangle^2 \frac{\sin\lambda - \phi_k}{1-\cos\lambda - \phi_k} = 0 \tag{4.8}$$

Finalement, utilisons la formule de Taylor [5], qui nous donne :

$$A - B\lambda - C\lambda^2 = O(\lambda^3) \tag{4.9}$$

Où

$$\begin{aligned} A &= 2\sum_{\phi_k=0} \langle \mathbf{0}, \boldsymbol{\psi_k}\rangle^2 \\ B &= \sum_{\phi_k\neq 0} \langle \mathbf{0}, \boldsymbol{\psi_k}\rangle^2 \frac{\sin\phi_k}{1-\cos\phi_k} \\ C &= \sum_{\phi_k\neq 0} \frac{\langle \mathbf{0}, \boldsymbol{\psi_k}\rangle^2}{1-\cos\phi_k} \end{aligned} \tag{4.10}$$

Appliquons maintenant le résultat du coefficient trouvé dans le cas d'un hypercube. Pour calculer des valeurs spécifiques, nous devons connaître les valeurs propres et les vecteurs propres de U. Nous utiliserons ici directement un résultat du livre [6]. Nous ferons tous les calculs dans cette base de Fourier : $\{\boldsymbol{a}\otimes\boldsymbol{\beta_k}, 1\leq a\leq n, 0\leq k\leq 2n-1\}$, et $\boldsymbol{D}\otimes\boldsymbol{\beta_0}$, $\boldsymbol{D}\otimes\boldsymbol{\beta_1}$ [7] est le vecteur propre correspondant de 1, -1. Conversion de la notation précédente en notation de l'hypercube à n dimensions : $\boldsymbol{\phi_k}\to\omega_k, \boldsymbol{\psi_k}\to\boldsymbol{\beta_k}$.

On obtient :

$$\begin{aligned} A &= \frac{2}{N} \\ B &= 0 \\ C &= \sum_{k=1}^{N-2} \frac{\left(\left|\langle \boldsymbol{D}, \boldsymbol{\alpha_1^k}\rangle\right|^2 + \left|\langle \boldsymbol{D}, \boldsymbol{\alpha_n^k}\rangle\right|^2\right)\left|\langle \mathbf{0}, \boldsymbol{\beta_k}\rangle\right|^2}{1-\cos\omega_k} + \frac{\left|\langle \mathbf{0}, \boldsymbol{\beta_1}\rangle\right|^2}{2} \end{aligned} \tag{4.11}$$

4. En utilisant $\frac{2}{1-e^{ia}} = 1 + \frac{i\sin a}{1-\cos a}$

5. $\frac{\sin\lambda-\phi_k}{1-\cos\lambda-\phi_k} = -\frac{\sin\phi_k}{1-\cos\phi_k} - \frac{\lambda}{1-\cos\phi_k} + O(\lambda^2)$, $\frac{\sin\lambda}{1-\cos\lambda} = -\frac{2}{\lambda} + O(\lambda)$

6. Sect 6.3.1 of du livre "Quantum Science and Technology by Renato Portugal "

7. $\boldsymbol{\beta_k} = \frac{1}{\sqrt{2^n}}\sum_v (-1)^{\boldsymbol{kv}}\boldsymbol{v}$

En simplifiant C [8], on obtient :

$$\begin{aligned} C &= \frac{n}{2N}\sum_{k=1}^{n}\frac{1}{k}\binom{n}{k} \\ &= \frac{n}{2}\mathbb{E}\left[\frac{1}{X}\mathbf{1}_{X>0}\right] && (\text{Où } X \sim \text{Bin}(n,\frac{1}{2})) \\ &= \mathbb{E}\left[\hat{X}\mathbf{1}_{\hat{X}>0}\right] && (\text{Où } \hat{X} \sim \frac{2}{n}\text{Bin}(n,\frac{1}{2})) \\ &= 1 \end{aligned}$$

En ramenant le résultat à (4.3), on obtient :

$$p(t) = \frac{1}{2}(\sin\frac{2t}{\sqrt{2N}})^2$$

Donc $p_{opt}(t) \approx \frac{1}{2}$, avec $t_{opt} = \left\lfloor \frac{\pi\sqrt{cN}}{4} \right\rfloor$ [9]

4.2 Étude statistique de l'algorithme

Comme l'algorithme a une chance sur deux de trouver le bon élément de la base de données, il est nécessaire d'établir un test afin de connaître avec une plus grande probabilité l'élément recherché. L'idée de ce test sera de répéter l'algorithme n fois et de prendre l'élément revenant le plus souvent.

4.2.1 Modélisation et convergence asymptotique

Formellement, nous modéliserons notre problème comme suit :
Soit $(X_i)_{1\leqslant i\leqslant n}$ un échantillon i.i.d. à valeur dans $\{0,...,N-1\}$ tel que

$$\exists\Delta \in \{0,...,N-1\}, \mathbb{P}(X_1 = \Delta) = \frac{1}{2}$$ [10]

8. En utilisant $\boldsymbol{\beta_k} = \frac{1}{\sqrt{2^n}}\sum_v(-1)^{\boldsymbol{kv}}\boldsymbol{v}$ et $\langle \mathbf{0}, \boldsymbol{\beta_k}\rangle = \frac{1}{\sqrt{2}}$

9. $\sin\frac{2t_{opt}}{\sqrt{2N}} \approx 1$

10. Nous pourrions faire l'ensemble de notre raisonnement avec la probabilité exacte trouver dans la partie précédente mais cela ne ferait que fortement alourdir les calculs pour un gain de précision peu important dès lors que N est grand.

On suppose que :

$$\exists C \in \mathbb{R} \text{ connue tq, } \forall j \neq \Delta,\ \mathbb{P}(X_1 = j) \leqslant C < \frac{1}{2} \tag{4.1}$$

On va s'intéresser, pour un j quelconque différent de Δ et pour tous les $i \leqslant n$, à la loi de $\mathbb{1}_{X_i=\Delta} - \mathbb{1}_{X_i=j}$. On commence par remarquer que :

$$\mathbb{E}[\mathbb{1}_{X_i=\Delta} - \mathbb{1}_{X_i=j}] = \mathbb{P}(X_1 = \Delta) - \mathbb{P}(X_1 = j) \in]0, \frac{1}{2}] \tag{4.2}$$

De plus, en prenant en compte que $\mathbb{1}_{X_1=\Delta}\mathbb{1}_{X_1=j} = 0$ pour tout $j \neq \Delta$ et que le carré d'une indicatrice est égale à l'indicatrice on obtient en developpant le carré :

$$\mathbb{E}[(\mathbb{1}_{X_i=\Delta} - \mathbb{1}_{X_i=j})^2] = \mathbb{E}[\mathbb{1}_{X_i=\Delta} - \mathbb{1}_{X_i=j}]$$

On s'intéresse maintenant à la variance :

$$\begin{aligned}
\mathbb{V}[\mathbb{1}_{X_i=\Delta} - \mathbb{1}_{X_i=j}] &= \mathbb{E}[(\mathbb{1}_{X_i=\Delta} - \mathbb{1}_{X_i=j})^2] - \mathbb{E}^2[\mathbb{1}_{X_i=\Delta} - \mathbb{1}_{X_i=j}] \\
\mathbb{V}[\mathbb{1}_{X_i=\Delta} - \mathbb{1}_{X_i=j}] &= \mathbb{E}[\mathbb{1}_{X_i=\Delta} - \mathbb{1}_{X_i=j}] - \mathbb{E}^2[\mathbb{1}_{X_i=\Delta} - \mathbb{1}_{X_i=j}] \\
\mathbb{V}[\mathbb{1}_{X_i=\Delta} - \mathbb{1}_{X_i=j}] &= \mathbb{E}[\mathbb{1}_{X_i=\Delta} - \mathbb{1}_{X_i=j}](1 - \mathbb{E}[\mathbb{1}_{X_i=\Delta} - \mathbb{1}_{X_i=j}])
\end{aligned}$$

en utilisant (4.2) on obtient trivialement :

$$\mathbb{V}[\mathbb{1}_{X_i=\Delta} - \mathbb{1}_{X_i=j}] < \inf$$

On peut donc appliquer le TCL aux $(\mathbb{1}_{X_i=\Delta} - \mathbb{1}_{X_i=j})_{i \leqslant n}$:

$$\sqrt{n}(\frac{\sum_{i=1}^n \mathbb{1}_{X_i=\Delta} - \mathbb{1}_{X_i=j}}{n} - \mathbb{E}[\mathbb{1}_{X_i=\Delta} - \mathbb{1}_{X_i=j}]) \xrightarrow[n \to +\inf]{\mathcal{L}} \mathcal{N}(0, \mathbb{V}[\mathbb{1}_{X_i=\Delta} - \mathbb{1}_{X_i=j}])$$

Comme on ne connait pas explicitement $\mathbb{V}[\mathbb{1}_{X_i=\Delta} - \mathbb{1}_{X_i=j]}$ et $\mathbb{E}[\mathbb{1}_{X_i=\Delta} - \mathbb{1}_{X_i=j]}$ nous utilisons que $\mathbb{V}[\mathbb{1}_{X_i=\Delta} - \mathbb{1}_{X_i=j}] = \mathbb{E}[\mathbb{1}_{X_i=\Delta} - \mathbb{1}_{X_i=j}](1 - \mathbb{E}[\mathbb{1}_{X_i=\Delta} - \mathbb{1}_{X_i=j}])$.
Puis nous utilisons (4.2) pour remplacer $\mathbb{E}[\mathbb{1}_{X_i=\Delta} - \mathbb{1}_{X_i=j}]$ par $\mathbb{P}(X_1 = \Delta) - \mathbb{P}(X_1 = j)$.
En dernier nous utilisons le fait que $\mathbb{P}(X_1 = \Delta) = \frac{1}{2}$ pour simplifier l'équation.

Nous n'écrivons pas les étapes intermédiaires car elles sont trop longues pour être écrites de manière lisible ici mais on obtient finalement :

$$\sqrt{n}(\frac{\sum_{i=1}^n \mathbb{1}_{X_i=\Delta} - \mathbb{1}_{X_i=j}}{n} - \frac{1}{2} + \mathbb{P}(X_1 = j)) \xrightarrow[n \to +\inf]{\mathcal{L}} \mathcal{N}(0, \mathbb{P}(X_1 = j)^2 + \frac{1}{4}) \tag{4.3}$$

Á noter que nous n'avons utilisé pour l'instant que des égalités.

4.2.2 Remplacement des données inconnues par des statistiques

Nous allons maintenant poser : $S_n^{(j)} := \frac{\sum_{i=1}^n \mathbb{1}_{X_i=j}}{n}$. Par la Loi Forte des Grands Nombres on sait que :

$$S_n^{(j)} \xrightarrow[n\to+\inf]{\text{p.s}} \mathbb{E}(\mathbb{1}_{X_1=j}) = \mathbb{P}(X_1 = j)$$
$$\text{Donc } S_n^{(j)} - \mathbb{P}(X_1 = j) \xrightarrow[n\to+\inf]{\text{p.s}} 0$$
$$\sqrt{n}(S_n^{(j)} - \mathbb{P}(X_1 = j)) \xrightarrow[n\to+\inf]{\text{p.s}} 0$$

On utilise Slutsky avec ce résultat et 4.3 et on obtient :

$$\sqrt{n}(\frac{\sum_{i=1}^n \mathbb{1}_{X_i=\Delta} - \mathbb{1}_{X_i=j}}{n} - \frac{1}{2} + S_n^{(j)}) \xrightarrow[n\to+\inf]{\mathcal{L}} \mathcal{N}(0, \mathbb{P}(X_1 = j)^2 + \frac{1}{4})$$
$$\text{Donc : } \frac{\sqrt{n}}{\sqrt{\mathbb{P}(X_1 = j)^2 + \frac{1}{4}}}(\frac{\sum_{i=1}^n \mathbb{1}_{X_i=\Delta} - \mathbb{1}_{X_i=j}}{n} - \frac{1}{2} + S_n^{(j)}) \xrightarrow[n\to+\inf]{\mathcal{L}} \mathcal{N}(0,1)$$

Comme $g(x) := \sqrt{x^2 + \frac{1}{4}}$ est continue sur [0,1]. Par les théoremes de continuité on en déduit que :

$$g(S_n^{(j)}) \xrightarrow[n\to+\inf]{\text{p.s}} \sqrt{\mathbb{P}(X_1 = j)^2 + \frac{1}{4}}$$
$$\frac{g(S_n^j)}{\sqrt{\mathbb{P}(X_1 = j)^2 + \frac{1}{4}}} \xrightarrow[n\to+\inf]{\text{p.s}} 1$$

On utilise désormais Slutsky pour obtenir :

$$\frac{\sqrt{n}}{\sqrt{S_n^{(j)\ 2} + \frac{1}{4}}}(\frac{\sum_{i=1}^n \mathbb{1}_{X_i=\Delta} - \mathbb{1}_{X_i=j}}{n} - \frac{1}{2} + S_n^{(j)}) \xrightarrow[n\to+\inf]{\mathcal{L}} \mathcal{N}(0,1) \tag{4.4}$$

4.2.3 Intervalle de confiance et test

Pour tout $\alpha < 1$, on peut donc utiliser q_α le quantile de la loi normale centrée réduite pour obtenir la probabilité asymptotique suivante :

$$\mathbb{P}(q_\alpha < \frac{\sqrt{n}}{\sqrt{S_n^{(j)\ 2} + \frac{1}{4}}}(\frac{\sum_{i=1}^n \mathbb{1}_{X_i=\Delta} - \mathbb{1}_{X_i=j}}{n} - \frac{1}{2} + S_n^{(j)})) \xrightarrow{n\to+\inf} 1-\alpha$$

$$\mathbb{P}(q_\alpha \frac{\sqrt{S_n^{(j)\ 2} + \frac{1}{4}}}{\sqrt{n}} + \frac{1}{2} - S_n^{(j)} < \frac{\sum_{i=1}^n \mathbb{1}_{X_i=\Delta} - \mathbb{1}_{X_i=j}}{n}) \xrightarrow{n\to+\inf} 1-\alpha \tag{4.5}$$

Ce résultat va nous permettre de construire un estimateur $\hat{p}$ et un test pour trouver Δ à partir d'un échentillon de $(X_i)_{i\leqslant n}$ (ie. trouver l'élément que l'on recherche grâce à n itérations de notre algorithme). Nous allons prendre comme estimateur de Δ la valeur la plus fréquente de notre échantillon. Formellement cela donne,

$$\hat{p} = \underset{k\in\{0,\dots,N-1\}}{arg\,max}\ (\sum_{i=1}^n \mathbb{1}_{X_i=k})$$

Dans le cas ou il y aurait plusieurs valeurs possibles pour arg max on considère que notre échantillon ne nous permet pas de trouver Δ.

On souhaite éviter autant que possible de considerer $\hat{p} = \Delta$ lorsque cela n'est pas le cas. On prend donc : $H_0 = \{\Delta \neq \hat{p}\}, H_1 = \{\Delta = \hat{p}\}$. On se fixe un niveau de confiance α ainsi que le quantile q_α associé. Notre test reposera sur l'idée que si nous sommes sous H_0, $\Delta \neq \hat{p}$ donc :

$$\sum_{i=1}^n \mathbb{1}_{X_i=\hat{p}} - \mathbb{1}_{X_i=\Delta} \geqslant 1$$

$$\frac{\sum_{i=1}^n \mathbb{1}_{X_i=\Delta} - \mathbb{1}_{X_i=\hat{p}}}{n} \leqslant -\frac{1}{n}$$

Si $q_\alpha \frac{\sqrt{S_n^{(\hat{p})\ 2} + \frac{1}{4}}}{\sqrt{n}} + \frac{1}{2} - S_n^{(\hat{p})} \leqslant -\frac{1}{n}$ on ne peut rien dire. Sinon on peut utiliser (4.5) pour rejeter l'hypothése H_0 avec un niveau de confiance asymptotique de $1-\alpha$ et donc en conclure que $\hat{p} = \Delta$.

4.3 Etude statistique ratée de l'algorithme

Ce paragraphe traite d'une tentative ratée et non aboutie d'obtenir un test au niveau de confiance $1-\alpha$ **non-asymptotique** permettant de définir si $\hat{p} = \underset{k\in\{0,\dots,N-1\}}{arg\,max}\ (\sum_{i=1}^n \mathbb{1}_{X_i=k})$ donne bien Δ.

Ce désir de test non-asymptotique est justifié par la très forte probabilité que notre algorithme donne Δ. En effet il y a une chance sur deux [11] que $\sum_{i=1}^{n} \mathbb{1}_{X_i=\Delta} \geqslant \sum_{j\neq\Delta} \sum_{i=1}^{n} \mathbb{1}_{X_i=j}$. Devoir effectuer un grand nombre de fois notre algorithme pour donner du sens à l'intervalle asymptotique que nous avons trouvé en 4.5 semble donc exagéré.

Cet essai étant non abouti, nous signalons qu'il y a certainement des coquilles ou raccourcis ayant pu se glisser dans notre rédaction.

4.3.1 Nouvelle modélisation et notation

Pour cette étude nous conservons le même formalisme que précédemment :
Soit $(X_i)_{1\leqslant i\leqslant n}$ un échantillon i.i.d. à valeur dans $\{0, ..., N-1\}$ tel que,
$\exists\Delta \in \{0, ..., N-1\}, \mathbb{P}(X_1 = \Delta) = \frac{1}{2}$.
De plus : $\exists C \in \mathbb{R}$ connu tel que, $\forall j \neq \Delta$, $\mathbb{P}(X_1 = j) \leqslant C < \frac{1}{2}$.
Nous allons ici aussi nous intéresser à $\mathbb{1}_{X_i=\Delta} - \mathbb{1}_{X_i=j}$ pour $j \neq \Delta$.
Cependant pour ce faire nous allons poser plusieurs éléments :

$\tilde{Y}_i^{(j)} := \mathbb{1}_{X_i=\Delta} - \mathbb{1}_{X_i=j}$
$\varphi_j := \mathbb{P}(X_1 = j)$
$(Y_i^{(j)})_{i\leqslant n}$ v.a. i.i.d. à valeur dans $\{-1, 1\}$ tel que $\mathbb{P}(Y_1^{(j)} = 1) := \mathbb{P}(X_1 = j | X_1 \neq 0)$

Il n'est pas compliqué de voir avec un peu de calcul que $Y_1^{(j)}$ suit une loi de Rademacher de paramètre $(\frac{1}{2\varphi_j+1})$:
$\mathbb{P}(Y_1^j = 1) = \frac{1}{2\varphi_j+1}$ et $\mathbb{P}(Y_1^j = -1) = \frac{2\varphi_j}{2\varphi_j+1}$

4.3.2 Idée et plan de la tentative

Nous allons chercher à calculer explicitement $\mathbb{P}(\hat{p} = \Delta)$. Pour ce faire nous remarquons que :

$$\hat{p} = \Delta \Leftrightarrow \forall j, \sum_{i=1}^{n} \tilde{Y}_i^{(j)} > 0$$

Dans la suite nous allons donc chercher à calculer $\mathbb{P}(\sum_{i=1}^{n} \tilde{Y}_i^{(j)} > 0)$ pour un j quelconque (toujours different de Δ évidemment). [12] Pour ce faire nous allons décomposer notre probabilité en une somme de probabilité plus simple et calculable explicitement

11. Calcul d'une loi binomiale de paramètre ($\frac{1}{2}$,n)

12. Les moyens de passer de $\mathbb{P}(\sum_{i=1}^{n} \tilde{Y}_i^{(j)} > 0)$ à $\mathbb{P}(\cap_{j\neq\Delta} \sum_{i=1}^{n} \tilde{Y}_i^{(j)} > 0)$ seront discutés à la toute fin de la tentative.

car suivant une loi binomiale.

$$\mathbb{P}(\sum_{i=1}^{n}\tilde{Y}_i^{(j)}>0)=\sum_{a=1}^{n}\mathbb{P}(\sum_{i=1}^{n}\tilde{Y}_i^{(j)}=a)$$

$$\mathbb{P}(\sum_{i=1}^{n}\tilde{Y}_i^{(j)}>0)=\sum_{a=1}^{n}\sum_{b=0}^{n-a}\mathbb{P}(\sum_{i=1}^{n}\tilde{Y}_i^{(j)}=a|\#\{i:\tilde{Y}_i^{(j)}=0\}=b)\mathbb{P}(\#\{i:\tilde{Y}_i^{(j)}=0\}=b) \tag{4.6}$$

4.3.3 Étude terme à terme de (4.6)

'Etude de $\mathbb{P}(\sum_{i=1}^{n}\tilde{Y}_i^{(j)}=a|\#\{i:\tilde{Y}_i^{(j)}=0\}=b)$

On peut facilement prouver avec une réorganisation des termes de la somme des $\tilde{Y}_i^{(j)}$ que pour tout b :

$$\mathbb{P}(\sum_{i=1}^{n}\tilde{Y}_i^{(j)}=a|\#\{i:\tilde{Y}_i^{(j)}=0\}=b)=\mathbb{P}(\sum_{i=1}^{n-b}Y_i^{(j)}=a)$$

Or on sait que les $Y_i^{(j)}$ suivent des lois de Rademacher de parametre $(\frac{1}{2\varphi_j+1})$ ce qui revient à dire que pour tout i, $\frac{Y_i^{(j)}+1}{2}$ suit une loi binomiale de paramètres $(\frac{1}{2\varphi_j+1})$. On note $\overset{L}{\simeq}$ l'égalité en loi et $\mathcal{B}(x,y)$ la loi binomiale de paramètre x, y :

$$\frac{Y_i^{(j)}+1}{2}\overset{L}{\simeq}\mathcal{B}(1,\frac{1}{2\varphi_j+1})$$

$$\sum_{i=1}^{n-b}\frac{Y_i^{(j)}+1}{2}\overset{L}{\simeq}\mathcal{B}(n-b,\frac{1}{2\varphi_j+1})$$

$$\sum_{i=1}^{n-b}Y_i^{(j)}\overset{L}{\simeq}2\mathcal{B}(n-b,\frac{1}{2\varphi_j+1})-(n-b)$$

Par définition de la convergence de l'égalité en loi on a donc :

$$\mathbb{P}(\sum_{i=1}^{n-b}Y_i^{(j)}=a)=\mathbb{P}(2\mathcal{B}(n-b,\frac{1}{2\varphi_j+1})-(n-b)=a)$$

$$\mathbb{P}(\sum_{i=1}^{n-b}Y_i^{(j)}=a)=\mathbb{P}(\mathcal{B}(n-b,\frac{1}{2\varphi_j+1})=\frac{n+a-b}{2})$$

$$\mathbb{P}(\sum_{i=1}^{n-b}Y_i^{(j)}=a)=\mathbb{1}_{\frac{n+a-b}{2}\in\mathbb{N}}\binom{n-b}{\frac{n+a-b}{2}}\frac{1}{(2\varphi_j+1)^{\frac{n+a-b}{2}}}\frac{(2\varphi)^{\frac{n-a-b}{2}}}{(2\varphi_j+1)^{\frac{n-a-b}{2}}}$$

Notre terme étudié vaut donc :

$$\mathbb{P}(\sum_{i=1}^{n} \tilde{Y}_i^{(j)} = a | \#\{i : \tilde{Y}_i^{(j)} = 0\} = b) = \mathbb{1}_{\frac{n+a-b}{2} \in \mathbb{N}} \binom{n-b}{\frac{n+a-b}{2}} \frac{1}{(1+2\varphi_j)^{\frac{n+a-b}{2}}} \frac{(2\varphi)^{\frac{n-a-b}{2}}}{(1+2\varphi_j)^{\frac{n-a-b}{2}}}$$

$$\mathbb{P}(\sum_{i=1}^{n} \tilde{Y}_i^{(j)} = a | \#\{i : \tilde{Y}_i^{(j)} = 0\} = b) = \mathbb{1}_{\frac{n+a-b}{2} \in \mathbb{N}} \binom{n-b}{\frac{n+a-b}{2}} \frac{(2\varphi)^{\frac{n-a-b}{2}}}{(1+2\varphi_j)^{n-b}} \tag{4.7}$$

Étude de $\mathbb{P}(\#\{i : \tilde{Y}_i^{(j)} = 0\} = b)$
Il est trivial de voir que le terme que nous étudions suit une loi binomiale de paramètre : $\mathcal{B}(n, \mathbb{P}(\tilde{Y}_i^{(j)} = 0))$, or $\mathbb{P}(\tilde{Y}_i^{(j)} = 0)$ est facilement calculable et vaut : $\frac{1-2\varphi}{2}$. On a donc :

$$\mathbb{P}(\#\{i : \tilde{Y}_i^{(j)} = 0\} = b) = \binom{n}{b} (\frac{1-2\varphi}{2})^b (\frac{1+2\varphi}{2})^{n-b}$$

$$\mathbb{P}(\#\{i : \tilde{Y}_i^{(j)} = 0\} = b) = \frac{1}{2^n} \binom{n}{b} (1-2\varphi)^b (1+2\varphi)^{n-b} \tag{4.8}$$

4.3.4 Optimisation de (4.6) et erreur

Dans cette section nous injectons les termes trouvés en (4.7) et (4.8) dans l'équation (4.6). Nous obtenons ainsi une formule explicite pour $\mathbb{P}(\sum_{i=1}^{n} \tilde{Y}_i^{(j)} > 0)$.
Cependant cette formule n'étant pas vraiment élégante (faisant notamment intervenir des sommes de produit de factorielles) nous avons utilisé la formule de Stirling [13] afin d'obtenir une équation plus facile a calculer pour nous et l'ordinateur.

Une implémentation de la formule ainsi obtenue donne toutefois une probabilité $\mathbb{P}(\sum_{i=1}^{n} \tilde{Y}_i^{(j)} > 0) \approx 10^{34}$. Nous ignorons à quel endroit nous avons commis le ou les erreurs aboutissant a ce résultat mais il nous semble qu'elles peuvent être de trois types :

- une simple erreur de calcul telle qu'une erreur de signe. Cela n'est pas du tout impossible vue la lourdeur des équations.
- une erreur dans notre justification de la convergence du produit des formules de Stirling.
- une erreur dans notre implementation de la formule finale. Cette dernière erreur nous semble de loin la moins probable.

Suite a l'échec manifeste de notre calcul de $\mathbb{P}(\sum_{i=1}^{n} \tilde{Y}_i^{(j)} > 0)$, et au vue du temps qu'il nous restait, nous n'avons pas poursuivi cette tentative et avons calculé le test asymptotique détaillé dans la section précédente. Il se peut donc que l'erreur vous paraisse évidente.

13. Nous expliquerons à la toute fin pourquoi la perte de précision engendré par le passage à cette formule ne semble pas poser de problème.

Pour revenir à notre calcul : nous remplacons dans (4.6) les termes que nous avons trouvé :

$$\mathbb{P}(\sum_{i=1}^{n} \tilde{Y}_i^{(j)} > 0) = \sum_{a=1}^{n} \sum_{b=0}^{n-a} \mathbb{1}_{\frac{n+a-b}{2} \in \mathbb{N}} \binom{n-b}{\frac{n+a-b}{2}} \frac{(2\varphi)^{\frac{n-a-b}{2}}}{(1+2\varphi_j)^{n-b}} \cdot \frac{1}{2^n} \binom{n}{b} (1-2\varphi)^b (1+2\varphi)^{n-b}$$

$$\mathbb{P}(\sum_{i=1}^{n} \tilde{Y}_i^{(j)} > 0) = \sum_{a=1}^{n} \sum_{b=0}^{n-a} \mathbb{1}_{\frac{n+a-b}{2} \in \mathbb{N}} \binom{n-b}{\frac{n+a-b}{2}} \binom{n}{b} \frac{(2\varphi)^{\frac{n-a-b}{2}} (1-2\varphi)^b}{2^n}$$

Cette formule bien que parfaitement valide est trop lourde pour être correctement exploitée. Nous la séparons donc en une somme de termes pouvant chacun individuellement être simplifié.

Pour ce faire nous développons les k parmis n en des fractions de factorielle que nous réduisons puis dans l'objectif d'appliquer la formule de Stirling nous isolons tous les termes faisant émerger des factorielles de zéros. [14]

Une fois ceci fait nous n'avons plus qu'à implémenter la formule dans un pc et constater que nous obtenons une probabilité en 10^{34}.

14. Nous n'expliquons les calculs que par des phrases car une fois les k parmis n developpés, l'équation est trop grosse pour rentrer lisiblement dans le document. C'est encore plus vrai une fois la formule de Stirling appliqué.

Références

[1] Renato Portugal, *Quantum Walks and Search Algorithms*, Springer Editions, 2019.

5 Annexe

Programmes Python

Algorithmes de marche sur la ligne :

```
import matplotlib.pyplot as plt
import numpy as np
import random as rd
import scipy.special as sp

## Marche discrete classique

size = 51
N = 50
cell = np.zeros(size, dtype=int)
cell[size//2] += 1
particle = size//2

def pas():
    global particle
    v = rd.choice([1, -1])
    particle += v
    cell[particle] += 1

def marche(N):
    for _ in range(N):
        pas()
    return cell

def prob_classic(t,n):
    if (t + n)%2 == 1 or n > t:
        return 0
    else:
        return 1/(2**t) * sp.binom(t,(t+n)/2)

def prob_distrib_classic(T=[72, 180, 450]):
    X = [i for i in range(-80,81,2)]
    for t in T:
        Y = [prob_classic(t,n) for n in X]
        plt.plot(X,Y, label="p("+str(t)+",n)")
        plt.legend()
    plt.grid()
    plt.show()


## Marche discrete quantique

#psi = [1/np.sqrt(2), -1j/np.sqrt(2)]
```

```
psi = [1, 0]
#Si psi = [a, b], l'etat initial est a*|0>|0> + b*|1>|0>
H = 1/np.sqrt(2) * np.array([[1, 1], [1, -1]])
N = 100

def amplitudes(psi, H):
    A = np.zeros((2*N+1, N+1), dtype=complex)
    B = np.zeros((2*N+1, N+1), dtype=complex)
    for n in range(2*N+1):
        if n == N:
            A[n,0] = psi[0]
            B[n,0] = psi[1]
        else:
            A[n,0] = 0
            B[n,0] = 0

    for t in range(0,N):
        for n in range(2*N):
            A[n,t+1] = H[0,0]*A[n-1,t] + H[0,1]*B[n-1,t]
            B[n,t+1] = H[1,0]*A[n+1,t] + H[1,1]*B[n+1,t]
    return A,B

A,B = amplitudes(psi, H)

def prob_quantum(t,n):
    return abs(A[n+N,t])**2 + abs(B[n+N,t])**2

def prob_distrib_quantum():
    X = [i for i in range(-N,N+1,2)]
    Y = [prob_quantum(N,n) for n in X]
    plt.plot(X,Y)
    plt.grid()
    plt.show()
```

Algorithme de Grover :

```
import matplotlib.pyplot as plt
import numpy as np
import random as rd
import time


## Algorithme classique

def classic(n):
    x0 = rd.randrange(n)
    def f(x):
        return int(x == x0)
```

```
    t0 = time.time()
    for i in range(n):
        if f(i) == 1:
            t1 = time.time()
            return t1-t0

def trace_classic(n, p):
    X = [i for i in range(1,n+1)]
    Y = np.zeros((p, n))
    for i in range(p):
            Y[i] = [classic(t)[1] for t in X]
    L = [sum([Y[i,j] for i in range(p)])/p for j in range(n)]
    plt.plot(X, L)
    plt.show()


## Fonctions preliminaires

def mult(x):
    N = len(x)
    A = np.zeros((N,N))
    for i in range(N):
        for j in range(N):
            A[i,j] = x[i]*x[j]
    return A


## Algorithme de Grover

def grover(N):
    t0 = time.time()

    # definitions des operateurs utilises dans l'algorithme
    x0 = rd.randrange(N)
    b = np.zeros(N)
    Rf = np.zeros((N,N))
    D = np.array([1/np.sqrt(N) for _ in range(N)])
    RD = 2*mult(D) - np.eye(N)
    for i in range(N):
        if i == x0:
            Rf[i,i] = -1
            b[i] = 1
        else:
            Rf[i,i] = 1
    U = np.dot(RD, Rf)
    t = int(np.floor(np.pi/4 * np.sqrt(N)))
    psi = D
    M = [max(np.abs(psi-b))]
```

```
    t1 = time.time()

    # application de l'algorithme
    for _ in range(t):
        psi = np.dot(U, psi)
        M.append(max(np.abs(psi-b)))

    t2 = time.time()

    # on trouve ou se trouve le "1" de psi
    ind = np.argmax(psi)

    for _ in range(2*t):
        psi = np.dot(U, psi)
        M.append(max(np.abs(psi-b)))

    t3 = time.time()
    #print(psi)
    #print("x0 = " + str(x0) + "\n" + "indice trouve : " + str(ind) + "\n" + "en "
    return t2-t1, M

def trace_grover(n, p):
    X = [i for i in range(1,n+1)]
    Y = np.zeros((p, n))
    for i in range(p):
            Y[i] = [grover(t)[0] for t in X]
    L = [sum([Y[i,j] for i in range(p)])/p for j in range(n)]
    plt.plot(X, L)
    plt.show()

def trace_grover2(n):
    t = int(np.floor(np.pi/4 * np.sqrt(n)))
    X = [i for i in range(3*t+1)]
    M = grover(n)[1]
    plt.plot(X, M, label='t = '+str(t))
    plt.legend()
    plt.grid()
    plt.show()
```

Cycles :

```
import matplotlib.pyplot as plt
import numpy as np
from math import *


N=200
```

```
H = 1/sqrt(2)*np.array([[1, 1], [1, -1]])
I = np.eye(N)

def mult(x):
    N = len(x)
    A = np.zeros((N,N))
    for i in range(N):
        for j in range(N):
            A[i,j] = x[i]*x[j]
    return A

def prod_scal(x, y):
    return sum([np.conjugate(x[i])*y[i] for i in range(len(x))])

R = np.eye(N)
R[0,0] = -1

S = np.zeros((2*N,2*N))
for i in range(N):
    S[(i+1)%N, i] = 1
    S[(i-1)%N + N, i+N] = 1

HxI = np.tensordot(H,I,axes=0)

def tens2mat(A):
    r, c, rp, cp = A.shape
    A_mat = np.zeros((2*N,2*N))
    for i in range(r):
        for j in range(c):
            for k in range(rp):
                for l in range(cp):
                    A_mat[i*N+k,j*N+l] = A[i,j,k,l]
    return A_mat

def tens2vect(a):
    r, rp = a.shape
    a_vect = np.zeros(2*N)
    for i in range(r):
        for j in range(rp):
            a_vect[i*N+j] = a[i,j]
    return a_vect

HxI_mat = tens2mat(HxI)

U = S.dot(HxI_mat)

def avancement_0(t):
    V = np.linalg.matrix_power(U, t)
```

```
    psi_0 = np.tensordot([1, 0], [int(i==0) for i in range(N)], axes=0)
    return V.dot(tens2vect(psi_0))


'''def avance(psi):
    psi_1 = np.zeros(2*N)
    for i in range(N):
        psi_1[i] = (psi[(i-1)%N] + psi[(i-1)%N + N])/sqrt(2)
        psi_1[i+N] = (psi[(i+1)%N] - psi[(i+1)%N + N])/sqrt(2)
    return psi_1'''

def prob(j, psi):
    return abs(psi[j])**2 + abs(psi[j+N])**2

def prob_distrib(T):
    X = [i for i in range(0, N, 2)]
    for t in T:
        Y = [prob(x, avancement_0(t)) for x in X]
        plt.plot(X, Y, label='t='+str(t))
        plt.legend()
    plt.show()

Up = U.dot(tens2mat(np.tensordot(np.eye(2), R, axes=0)))

def avancement_N(t):
    V = np.linalg.matrix_power(Up, t)
    psi_0 = np.tensordot([1, 0], [1/sqrt(N) for i in range(N)], axes=0)
    return V.dot(tens2vect(psi_0))

def avance(psi, t):
    psi1 = psi.copy()
    for _ in range(t):
        psi1 = Up.dot(psi1)
    return psi1

def prob_0(t):
    psi = avancement_N(t)
    return abs(prod_scal([int(i==0) for i in range(N)], psi))**2

def cherche_t(n,m,d):
    Psi = np.zeros((m-n)//d)
    psi_0 = tens2vect(np.tensordot([1, 0], [1/sqrt(N) for i in range(N)], axes=0))
    psi = avance(psi_0,n)
    j = 0
    for i in range(n,m,d):
        psi = avance(psi, d)
        Psi[j] = prob(0, psi)
        j += 1
```

```
    X = [i for i in range(n, m, d)]
    plt.plot(X, Psi)
    plt.show()

def cherche(t):
    psi_0 = tens2vect(np.tensordot([1, 0], [1/sqrt(N) for i in range(N)], axes=0))
    proba = []
    for i in range(t):
        psi_0 = Up.dot(psi_0)
        proba.append(sum([psi_0[i*N]**2 for i in range(2)]))
    plt.plot(proba)
    plt.grid()
    plt.show()
```

Grilles :

```
import matplotlib.pyplot as plt
import numpy as np
from math import *


VN = 30
N = VN**2


## Definition de fonctions utiles

def mult(x):
    N = len(x)
    A = np.zeros((N,N))
    for i in range(N):
        for j in range(N):
            A[i,j] = x[i]*x[j]
    return A

def prod_scal(x, y):
    return sum([np.conjugate(x[i])*y[i] for i in range(len(x))])

def tens2mat(A):
    r, c, rp, cp = A.shape
    A_mat = np.zeros((4*N,4*N))
    for i in range(r):
        for j in range(c):
            for k in range(rp):
                for l in range(cp):
                    A_mat[i*N+k,j*N+l] = A[i,j,k,l]
    return A_mat

def tens2vect(a):
```

```
    r, rp = a.shape
    a_vect = np.zeros(4*N)
    for i in range(r):
        for j in range(rp):
            a_vect[i*N+j] = a[i,j]
    return a_vect

def basis(i,n):
    return np.array([int(j==i) for j in range(n)])

def big_basis(d,s,x,y):
    ds = basis(2*d+s,4)
    xy = basis(VN*x+y,N)
    return tens2vect(np.tensordot(ds,xy,axes=0))

def elements_base(i):
    m = i//N
    d = m//2
    s = m%2
    n = i - m*N
    x = n//VN
    y = n%VN
    return d, s, x, y


## Construction des operateurs

G = 1/2 * np.array([[-1, 1, 1, 1],
                    [1, -1, 1, 1],
                    [1, 1, -1, 1],
                    [1, 1, 1, -1]])

S = np.zeros((4*N,4*N))
for i in range(4*N):
    d, s, x, y = elements_base(i)
    S[i] = big_basis(d, np.bitwise_xor(s, 1), (x+int(d==0)*(-1)**s)%VN, (y+int(d==

GxI = tens2mat(np.tensordot(G,np.eye(N),axes=0))

U = S.dot(GxI)

R = np.eye(4*N) - 2*mult(tens2vect(np.tensordot([1/sqrt(4) for i in range(4)], bas

Up = U.dot(R)


## Calcul des probabilites
```

```
def avance(psi):
    return np.dot(U,psi)

def avancement(t, psi_0):
    psi = psi_0.copy()
    for _ in range(t):
        psi = np.dot(U,psi)
    return psi

def prob(j, psi):
    return sum([abs(psi[j+N*i])**2 for i in range(4)])

def prob_distrib(t):
    X = np.zeros((VN,VN))
    psi_0 = tens2vect(np.tensordot([1/2 for i in range(4)], [1/VN for i in range(N
    psi = avancement(t, psi_0)
    for i in range(VN):
        for j in range(VN):
            X[i,j] = prob(VN*i+j, psi)
    plt.matshow(X, cmap='hot')
    plt.colorbar()
    plt.show()


## Algorithme de recherche

def prob_0(t):
    psi_0 = tens2vect(np.tensordot([1/2 for i in range(4)], [1/VN for i in range(N
    proba = []
    for _ in range(t):
        psi_0 = Up.dot(psi_0)
        proba.append(sum([psi_0[i*N]**2 for i in range(4)]))
    return proba

def cherche_t(t):
    plt.plot(prob_0(t))
    plt.grid()
    plt.show()
```