PROBLÈME DU CAVALIER

Mathématiques Expérimentales II

30 décembre 2022

Couto Melo Diana	0210952330
Rafael Ferreira Simoes Christophe	019141964F
Tahireddine Camal	0211658018

Table des matières

1 Introduction **1**

1.1 Question principale 1

1.2 Échiquier de taille infinie 2

2 Processus **3**

2.1 Étapes initiales 3

3 Résultats **5**

3.1 Cases atteignables pour un pas donné 5

3.2 Taille minimale pour couvrir l'échiquier 10

4 Application : Programmes Python **14**

4.1 Cases atteignables pour un pas et une taille donnés 14

4.2 Temps de parcours 16

5 Conclusion **19**

6 Annexe **20**

6.1 Réduction de pas 20

6.2 Programme Python : Dessin des orbites 21

1 | Introduction

1.1 Question principale

Le problème du cavalier est un problème des mathématiques et de la logique, qui peut être rencontré sous diverses formes et variantes. En effet, le problème du cavalier peut être retracé au millénaire précédent, surtout dans le monde oriental, puis dans le monde occidental plus tard. Le but reste principalement le même : parcourir l'échiquier classique de taille 8×8 en entier. Mais il est possible de modifier un grand nombre de paramètres et de restriction pour ce problème : la taille et la forme de l'échiquier, la manière de déplacer le cavalier, la possibilité de repasser par des cases ou non, la fermeture ou l'ouverture du parcours, la possibilité de croisement dans le parcours. On peut également trouver des parcours dont l'ordre des cases forme un carré semi-magique ! L'étendue de ce sujet est donc vaste et malléable.

La définition du sujet pour ce projet se base sur la question principale suivante :

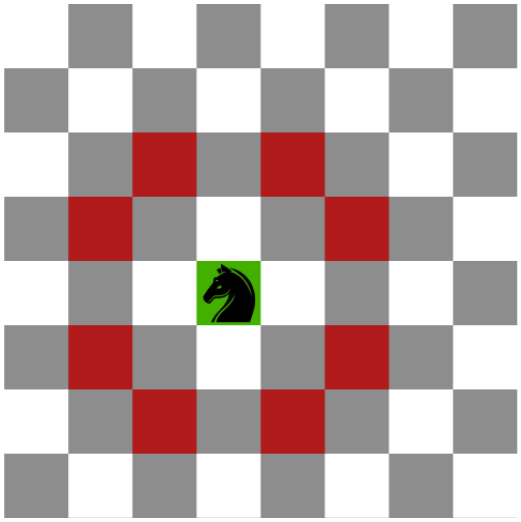
Quelles cases d'un échiquier de taille $\mathbf{n}$ est-ce qu'un cavalier qui se déplace de pas $[\mathbf{a}, \mathbf{b}]$ peut atteindre ?

Il est utile d'ajouter que la possibilité de repasser par une case est permise et que la forme de l'échiquier est fixée comme un carré. On note que la manière dont le cavalier se déplace n'est pas nécessairement celle du jeu de l'échec, mais peut varier.

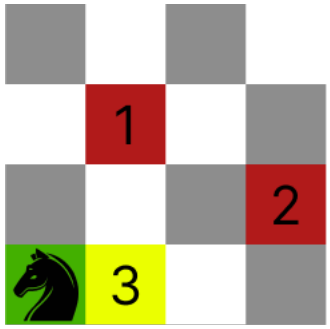
Dans ce rapport, on retrace le processus de nos recherches expérimentales, puis nos résultats. Ensuite, on met en œuvre nos observations dans quelques applications numériques. Finalement, on délimite nos conclusions dans une synthèse.

1.2 Échiquier de taille infinie

Le cavalier peut-il atteindre toutes les cases dans un échiquier infini ?
Soit un échiquier de taille infinie et un cavalier classique des échecs, qui se déplace donc d'une-par-deux cases, comme illustré ci-dessous :



Le cavalier fait alors ce pas vers le haut à partir de la position de départ, puis vers la droite et enfin vers l'arrière, comme le montre la figure suivante :



Il a alors quitté la position de départ (en vert) pour atterrir sur la case voisine (en jaune) et a ainsi fait le plus petit pas possible. Comme il se trouve dans un champ infini, ces 3 pas sont toujours possibles et il peut se déplacer dans les 4 directions. Il peut donc atteindre le carré du haut, tout comme le carré du bas et celui à gauche. Comme le cavalier peut maintenant se déplacer case par case, il peut atteindre toutes les cases d'un échiquier infini.

2 | Processus

2.1 Étapes initiales

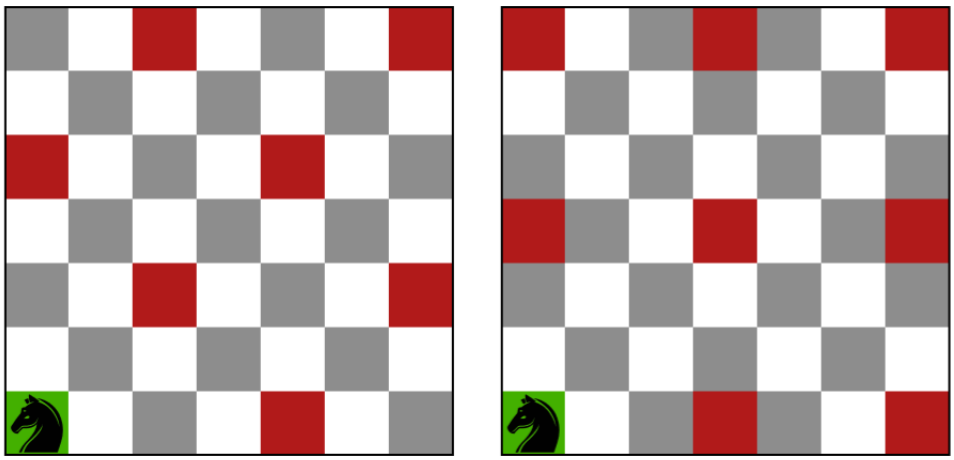
Au début de notre travail, nous ne savions pas comment nous y prendre pour résoudre ce problème, nous avons donc créé un échiquier sur lequel nous pouvions changer la taille de l'échiquier à tout moment et nous avons essayé de voir comment notre cavalier se déplaçait.

Nous avons commencé par des petits pas comme [1,2] , [1,3] ou même un pas de [1,1] .

Mais nous ne pouvons pas en conclure, alors nous avons continué à essayer et nous n'avons pas abandonné. Mais lorsque nous sommes passés à des pas plus grands, nous avons dû agrandir notre échiquier, car sinon nous l'aurions dépassé. Nous avons donc dû revoir les conditions sur la taille de l'échiquier.

Maintenant, avec un échiquier plus grand, nous avons essayé des pas comme [3,4] , [6,7], [1,12] ou encore [5,5].

Lorsque nous avons rempli le champ avec nos pas, nous avons réalisé qu'il s'agissait toujours d'un motif. Le motif ressemblait au cas à gauche ou à droite dans l'illustration suivante :



Ensuite, on a constaté que nous pouvions réduire les différents pas à un motif plus petit et nous avons remarqué que notre cavalier ne pouvait pas atteindre les autres cases, il devait donc partir de la case d'à côté. Cela nous a fait comprendre que nous avions des chemins différents. Par la suite, nous appellerons "une orbite" un des chemins parcourus par le cavalier.

Après nous avons remarqué que si nous commençons sur une case noire et que la parité est la même, nous atterrissons à nouveau sur une case noire et ne pouvons donc jamais tomber sur une case blanche, c'est le [1,1] pas.

Nous avons alors commencé à faire des conclusions, par exemple si a et b sont de la même parité ou non et si le pgcd est égal ou supérieur à 1 et nous avons commencé à obtenir différents cas.

3 | Résultats

Dans cette section, on expose nos observations finaux sous forme de théorème que l'on prouve. Ces théorèmes indiquent quelles cases sont atteignables en fonction du pas, puis la taille minimale pour atteindre toutes les cases. Par la suite, nous appellerons "une orbite" un des chemins parcourus par le cavalier. On note Ω le nombre d'orbites.

3.1 Cases atteignables pour un pas donné

Théorème 1. *Si le chevalier effectue un pas* $[a, b]$ *dans une grille de taille* $n \geq 2b$ *avec* $a \equiv b\ [2]$ *alors le cavalier peut, depuis l'origine* $(0, 0)$*, atteindre toutes les cases de coordonnées* $(p \cdot (a \wedge b), q \cdot (a \wedge b))$ *avec* $p, q \in N$*, tels que* p *et* q *ont la même parité.*
Le nombre d'orbites est : $\Omega = 2 \cdot (a \wedge b)^2$.

Démonstration. Faisons la preuve des théorèmes en montrant pour chaque cas, quelles coordonnées le cavalier peut atteindre :
Soit $[a, b]$ le pas du cavalier. On cherche les solutions du système $ax + by = \gamma$ pour x, y entiers relatifs. Ce système permet de trouver la coordonnée horizontale γ des cases atteignables. On sait que cela admet une solution lorsque le pgcd de a et b divise γ. On divise alors les deux membres de l'équation par $a \wedge b$ et on obtient, en posant $A := \frac{a}{a \wedge b}$ et $B := \frac{b}{a \wedge b}$ le système $Ax + By = \Gamma$ avec $\Gamma = \frac{\gamma}{a \wedge b}$. On a donc que $A \wedge B = 1$. On a un système similaire $ax' + by' = \delta$ avec x', y' des entiers relatifs. En divisant ici aussi par $a \wedge b$, on obtient $Ax' + By' = \Delta$ Ce système est pour la coordonnée verticale que le cavalier peut atteindre. On note E l'ensemble des cases atteignables de coordonnées (Γ, Δ).
Supposons maintenant que $A \equiv B\ [2]$. On étudie la parité entre les points de E. On doit avoir :

$$(\Gamma, \Delta) := (Ax + By, Ax' + By') \quad \text{avec} \begin{cases} x \equiv y'\ [2] \\ x' \equiv y\ [2] \end{cases}$$

Les deux conditions de parité sur x, y' et x', y proviennent du fait que le cavalier doit effectuer des pas de $[\pm A, \pm B]$. En effet, lorsqu'on rajoute A dans la première coordonnée, on doit soit rajouter ou retrancher B dans la deuxième coordonnée, et vice-versa. On obtient donc, pour que ça reste un pas de cavalier valable, que le nombre de fois x où on ajoute ou retranche A de la première coordonnée doit avoir la même parité que le nombre de fois y' où on ajoute ou retranche B de la deuxième coordonnée. De même pour x' le nombre de B ajoutés à la première coordonnée qui a la même parité que le nombre y le nombre d' A ajoutés à la deuxième coordonnée. Ceci nous ramène au calcul suivant :

$$\begin{aligned}
&(Ax + By) - (Ax' + By') \ [2] \\
&\equiv Ax + By - Ax' - By' \ [2] \\
&\equiv A(x + y - x' - y') \ [2] \qquad \text{par hypothèse} \\
&\equiv 0 \ [2]
\end{aligned}$$

Ainsi les cases atteignables ont la même parité. Autrement dit, les cases atteignables (Γ,Δ) doivent vérifier $\Gamma \equiv \Delta \ [2]$. On a donc trouvé une condition nécessaire. Mais est-elle suffisante ? A-t-on : " $(\Gamma, \Delta) = (Ax + By, Ax' + By')$ tels que $x \equiv y' \ [2]$ et $x' \equiv y \ [2]$ " ?

$$\begin{aligned}
&(Ax + By) \equiv (Ax' + By') \ [2] \\
&\iff A(x + y) \equiv A(x' + y') \ [2] \\
&\iff x + y \equiv x' + y' \ [2] \\
&\iff x - y' \equiv x' - y \ [2]
\end{aligned}$$

Cette situation fait discerner deux cas :
Soit $x \equiv y' \ [2]$ et donc $x' \equiv y \ [2]$ et on a trouvé le résultat escompté.
Soit $x \not\equiv y' \ [2]$. Dans ce cas, on pose : $\begin{cases} \tilde{x} := x + A \\ \tilde{y} := y - B \end{cases}$

On peut effectuer ce changement de variables, car le système $A\tilde{x} + B\tilde{y} = \Gamma$ reste vrai et on ne modifie donc pas les conditions sur la parité de $\tilde{x}$ et $\tilde{y}$. Sachant que $A \wedge B = 1$ et $A \equiv B \ [2]$, on a nécessairement que $A \equiv 1 \ [2]$. Donc :

$$\begin{aligned}
\tilde{x} - y' &\equiv x + A - y' \ [2] \\
&\equiv x - y' + 1 \ [2] \qquad \text{car} x \not\equiv y' \ [2] \\
&\equiv 0 \ [2]
\end{aligned}$$

Ainsi on a trouvé que $\tilde{x} \equiv y' \ [2]$ et on a prouvé le résultat voulu. □

Théorème 2. *Si le chevalier effectue un pas* $[a, b]$ *dans une grille de taille* $n \geq 2b$ *avec* $a \not\equiv b\ [2]$ *alors, il peut atteindre toutes les cases de coordonnées* $(p \cdot (a \wedge b), q \cdot (a \wedge b))$ *avec* $p, q \in N$ *depuis l'origine* $(0, 0)$*. Le nombre d'orbites est donc :* $\Omega = (a \wedge b)^2$.

Démonstration. Supposons, en posant ici aussi $A := \frac{a}{a \wedge b}$ et $B := \frac{b}{a \wedge b}$ que $A \not\equiv B\ [2]$ et donc que $A \wedge B = 1$. Ceci est équivalent à dire que $A + 1 \equiv B\ [2]$
On a, comme dans la première preuve que :

$$(\Gamma, \Delta) := (Ax + By, Ax' + By')\ \ [2] \qquad \text{avec} \begin{cases} x \equiv y'\ [2] \\ x' \equiv y\ [2] \end{cases}$$

On étudie ici aussi la parité entre les coordonnées horizontaux et verticaux que le cavalier peut atteindre. On a donc :

$$\begin{aligned} &(Ax + By) - (Ax' + By')\ [2] \\ &\equiv Ax + By - Ax' - By'\ [2] \\ &\equiv Ax + (A+1)y - Ax' - (A+1)y'\ [2] \\ &\equiv A(x + y - x' - y') + y - y'\ [2] \\ &\equiv y - y'\ [2] \\ &\equiv x' - x\ [2] \end{aligned}$$

Ceci montre que la parité des coordonnées des cases atteignables dépend de la parité des nombres de pas effectués. En effet, lorsqu'on effectue un nombre pair de pas, on atterrit sur la même couleur que celle où on a commencé. Et lorsqu'on effectue un nombre impair de pas, on change de couleur. Montrons maintenant qu'on peut effectivement atteindre toutes les cases :

$$\text{Donc si} \begin{cases} y \equiv y' \equiv x \equiv x' \equiv 0\ [2] \implies \Gamma \equiv \Delta \\ y \equiv y' \equiv x \equiv x' \equiv 1\ [2] \implies \Gamma \not\equiv \Delta \end{cases}$$

Pour A et B de parité différente, le cavalier peut sauter sur les cases noires et blanches. Supposons que A est pair et B donc impair et montrons que tous les couples (Γ, Δ) peuvent être atteints. On a :

$$\begin{cases} \Gamma = Ax + By \\ \Delta = Ax' + By' \end{cases}$$

On doit alors distinguer quatre cas :

Premier cas

Si on a :

$$\begin{cases} x = y' \ [2] \\ x' = y \ [2] \end{cases}$$

On obtient directement le résultat voulu.

Deuxième cas

Si on a :

$$\begin{cases} x = y' + 1 \ [2] \\ x' = y \ [2] \end{cases}$$

Dans ce cas, on effectue un changement de variable qui modifie x sans modifier la parité de y.

Troisième cas

Si on a :

$$\begin{cases} x = y' \ [2] \\ x' = y + 1 \ [2] \end{cases}$$

Pour pouvoir trouver que x et y' aient la même parité il faut additionner x' par B et soustraire A de y'.

Quatrième cas

Si on a :

$$\begin{cases} x = y' + 1 \ [2] \\ x' = y + 1 \ [2] \end{cases}$$

On effectue les changements de variables suivants :

$$\begin{cases} \tilde{x} := x + B \\ \tilde{y} := y - A \\ \tilde{x}' := x' + B \\ \tilde{y}' := y' - A \end{cases}$$

Donc ça nous raméne a un cas précédent.

On a donc montré que dans tous les cas, on peut atteindre toutes les cases lorsque A et B n'ont pas la même parité. Finalement le nombre d'orbites est $\Omega = (A \wedge B)^2 = 1$. □

3.2 Taille minimale pour couvrir l'échiquier

Théorème 3. *Lorsque le cavalier effectue un pas* $[a, b]$ *avec* $a \not\equiv b\ [2]$ *et* $a \wedge b = 1$*, alors la taille minimale de l'échiquier pour qu'il puisse atteindre toutes les cases est de* $n = 2 \cdot \max(a, b)$.

Démonstration. Soit n la taille de l'échiquier et $[a, b]$ le pas du cavalier, avec $a \not\equiv b\ [2]$. On suppose, sans perte de généralité que $a < b$. On peut déjà observer que $n \geq 2b$ car sinon le cavalier ne peut pas faire de pas lorsqu'il se trouve sur une case centrale et cette case devient ainsi inatteignable.

On montre le résultat par récurrence.

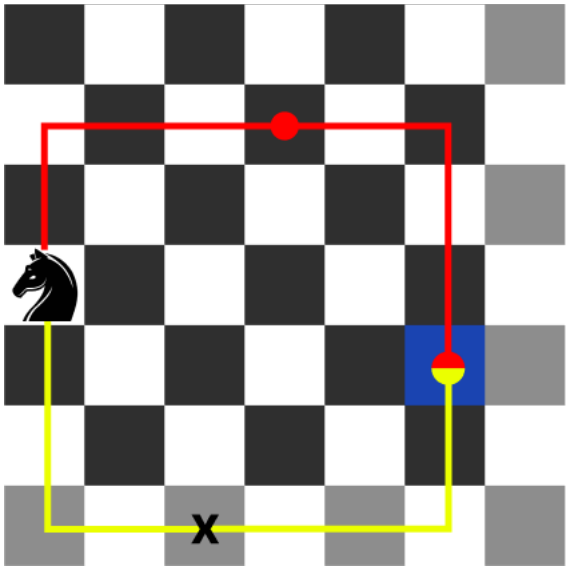
Pour une grille de taille n, tous les points sont atteints en considérant que le cavalier peut sortir de la grille. Puisqu'il y a un nombre fini de points, on suppose que le cavalier se déplace dans une grille de taille $n + k$, pour un k suffisamment grand (Par théorème 1 de partie 3).

Pour cela, on énumère les différentes parties de chemin que le cavalier puisse faire pour atteindre une case voulue. Il y a, à symétrie près, 5 cas de suites de pas à considérer. On illustre comment confiner un chemin dans une grille de taille $(n + 1)$ à une grille de taille n. On prendra ici $b = 3$, $a = 2$ et donc $n = 6$. La preuve pour a, b quelconque suit de manière similaire. On remarque que le cas général pour passer de la taille $n + k$ à $n + k - 1$ et en tout point identique.

Pour les cinq prochains cas, le segment jaune est une partie du chemin qui relie la case de départ à une case quelconque de notre grille et on illustre le chemin alternatif qui est atteignable sans sortir de la grille.

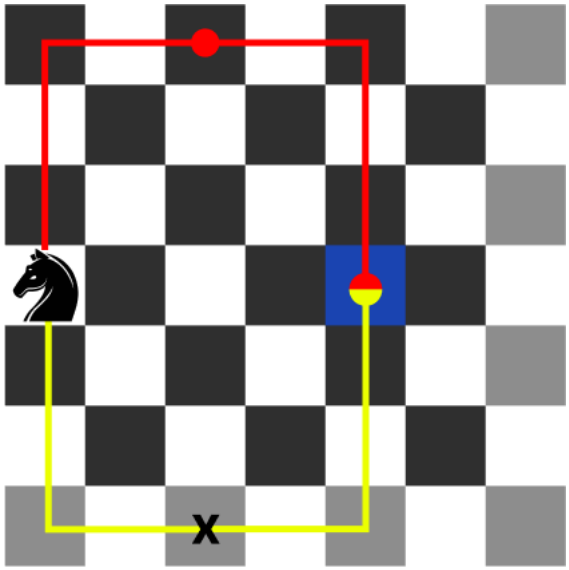
Premier cas

Sur l'illustration, le cavalier veut atteindre la case bleue à l'aide du chemin jaune qui passe par l'échiquier de taille $n + 1$. On montre alors qu'il y a un chemin alternatif rouge qui reste dans l'échiquier de taille n. Ainsi, cette suite de pas peut être effectué sans quitter l'échiquier voulu.



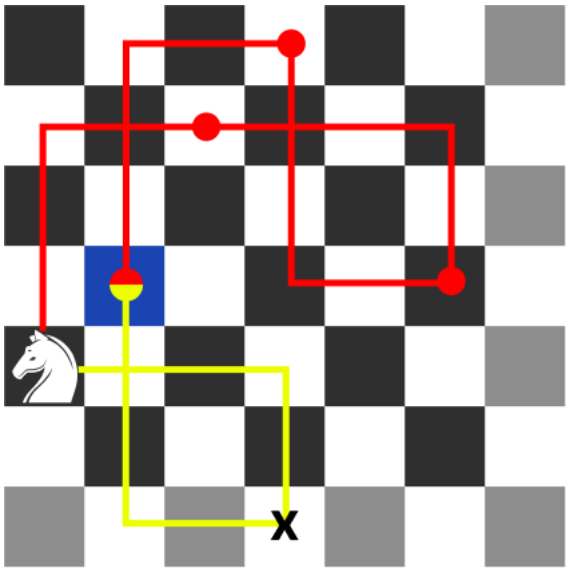
Deuxième cas

Similairement, la suite de pas suivante peut être effectué sans quitter l'échiquier voulu.



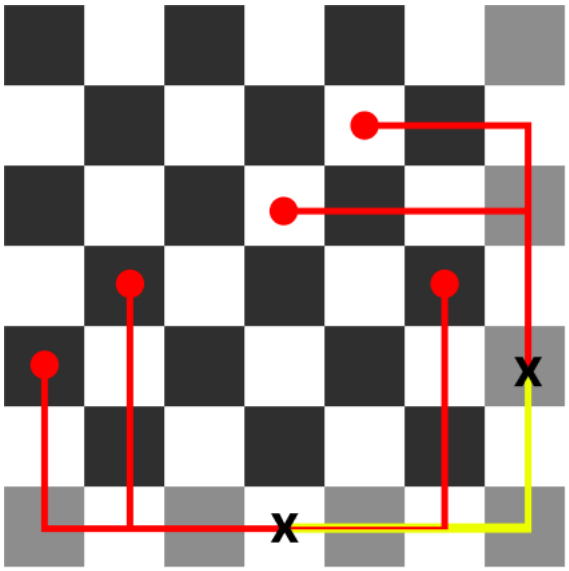
Troisième cas

Le chemin jaune dans ce cas doit être remplacé par la même suite de pas, contrairement avec les cas précédents on ajoute ici deux pas en plus ce qui rallonge le temps pour que le cavalier puisse atteindre cette case d'arrivée bleue sans quitter l'échiquier de taille n.



Quatrième cas

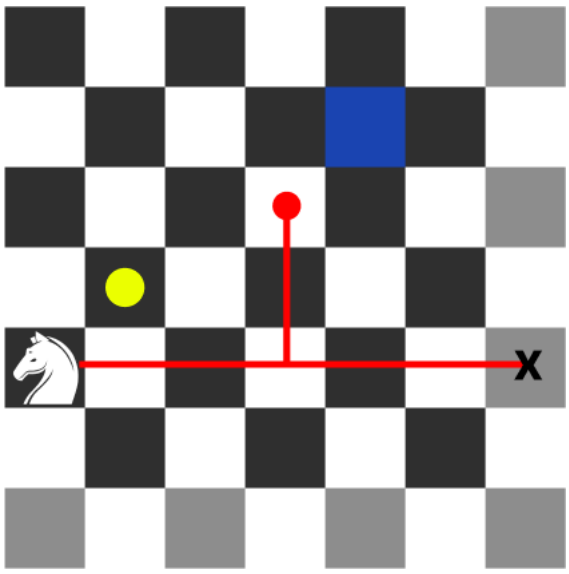
Lorsque le cavalier doit effectuer une partie de chemin (en jaune) contenue dans l'échiquier de taille $n+1$, il peut par la suite atteindre cinq cases dans l'échiquier de taille n, à savoir *a3, b4, d5, e6* et *f4*. (avec le système de coordonnées standard des échecs). Supposons que le cavalier commence sur *a3*. Il peut alors atteindre la case *b4* par la méthode du 3^e cas. Il peut atteindre la case *d5* directement, et la case *e6* en allant par *b4* et puis directement sur *e6*. Finalement, il peut atteindre la case *f4* par la méthode du 1^{er} cas.



Cinquième cas

Lorsque le cavalier nécessite cette suite de pas pour atteindre la case avec la croix, c'est pour atteindre trois cases par la suite : *d1, d5* et *e6*. La case *d1* est traité dans le cas précédent. La case *d5* est atteinte directement. Pour la case bleue *e6*, on peut atteindre la case avec

le point jaune à partir de la case départ à l'aide du 3^e cas et il ne manque qu'un pas pour atterrir sur la case finale.



On a donc montré que les différentes suites de pas possibles, à symétrie près, dans un échiquier de taille $n+k$, avec k suffisamment grand pour l'initialisation, peuvent être effectuées dans la taille $n+k-1$. Ainsi par récurrence descendante et sachant que $n \geq 2b$, on trouve finalement $n = 2b$ comme taille minimale pour que le cavalier puisse atteindre toutes les cases de l'échiquier. □

4 Application : Programmes Python

4.1 Cases atteignables pour un pas et une taille donnés

La première application consiste à déterminer, pour une taille de l'échiquier, un pas, et une position initiale donnés, si le cavalier peut atteindre toutes les cases. Pour cela, la méthode de **parcours de graphe en largeur** a été utilisée. Dans ce cas, cela implique que pour chaque position que le cavalier atteint, on rassemble les cases atteignables, ou « voisines », de cette case dans un ensemble. Puis, ces cases voisines deviennent des cases atteintes et on itère le processus jusqu'à ce que le cavalier n'ait plus de voisins non atteints.
Ci-dessous le programme python :

```
def onBoard(x, y, n):
    return 0 <= x < n and 0 <= y < n
def getNeighbours(x, y, n, a, b):
    neighbours = set()
    if onBoard(x + a, y + b, n):
        neighbours.add((x + a, y + b))
    if onBoard(x + a, y - b, n):
        neighbours.add((x + a, y - b))
    if onBoard(x - a, y + b, n):
        neighbours.add((x - a, y + b))
    if onBoard(x - a, y - b, n):
        neighbours.add((x - a, y - b))
    if onBoard(x + b, y + a, n):
        neighbours.add((x + b, y + a))
    if onBoard(x + b, y - a, n):
        neighbours.add((x + b, y - a))
    if onBoard(x - b, y + a, n):
        neighbours.add((x - b, y + a))
```

```
    if onBoard(x - b, y - a, n):
        neighbours.add((x - b, y - a))
    return neighbours

#print("Le cavalier peut-il atteindre toutes les cases pour une taille d'échiquier,
#un pas du cavalier et sa position initiale donnés?")

n = int(input("Entrez la taille de l'échiquier:"))
a = int(input('Entrez le petit pas:'))
b = int(input('Entrez le grand pas:'))
x = int(input('Entrez la position initiale horizontale:'))
y = int(input('Entrez la position initiale verticale:'))
i = 0

queue = set()
queue.add((x, y))
visited = set()
neighbours = getNeighbours(x,y,n,a,b)

while not len(neighbours) == 0:
    visited = (visited | queue)
    queue = neighbours
    neighbours = set()
    for point in queue:
        neighbours = (neighbours | getNeighbours(point[0], point[1], n, a, b))
    neighbours = neighbours-visited
    i+=1
visited = (visited | queue)

print('\n')
print("Le cavalier a visité au total",len(visited), "cases")

if len(visited) == n**2:
    print('Donc oui, il peut atteindre toutes les cases.')
else:
    print('Donc non, il ne peut pas atteindre toutes les cases.')
print('Le temps est:',i)
```

Voici un exemple d'output du code :

```
Le cavalier peut-il atteindre toutes les cases pour une taille d'échiquier,
un pas du cavalier et sa position initiale donnés ?
Entrez la taille de l'échiquier :6
Entrez le petit pas :2
Entrez le grand pas :3
Entrez la position initiale horizontale :0
Entrez la position initiale verticale :0
Le cavalier a visité au total 36 cases
Donc oui, il peut atteindre toutes les cases.
Le temps est : 9
```

4.2 Temps de parcours

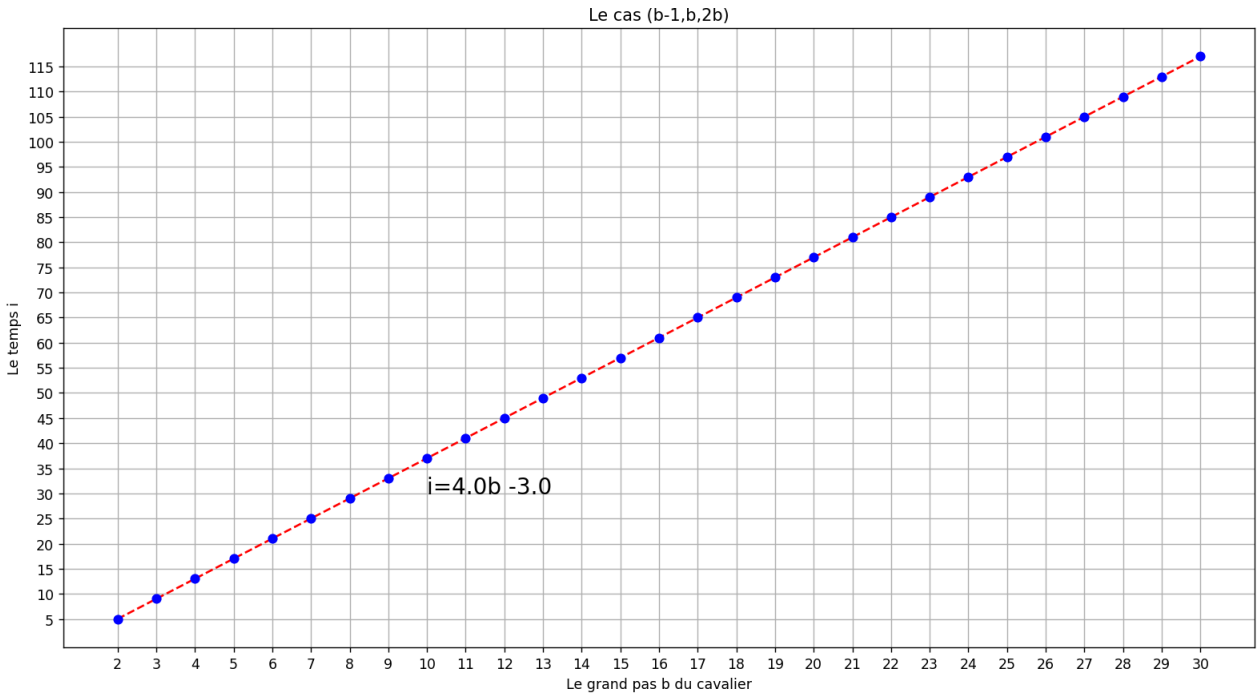
Le programme compte également le nombre d'itérations pour obtenir tous les voisins de toutes les cases atteignables à chaque étape, appelé « temps » et dénoté par la lettre i.

On peut alors s'interroger sur la relation entre le pas $[a, b]$ et le temps i lorsqu'on impose certaines conditions sur le pas. Lorsqu'on calcule le temps pour le pas $[b-1, b]$ dans un échiquier de taille $2b$, avec $b > 1$, et qu'on trace le temps en fonction de b, on obtient une relation entre les deux. Voici le code qui permet de tracer le graphique, en ayant transformé le code précédant en tant que fonction qui retourne i pour un b donné, nommée *temps(b)*.

```
abscisses = []
ordonnees = []
for j in range(2,31):
    abscisses.append(j)
    ordonnees.append(temps(j))

z = np.polyfit(abscisses, ordonnees, 1)
p = np.poly1d(z)

plt.plot(abscisses, p(abscisses), "r--")
plt.plot(abscisses, ordonnees, 'bo')
plt.grid()
```

```
plt.title('Le cas (b-1,b,2b)')

plt.text(10, 30, 'i='+ str(np.round(z[0]))+'b ' + str(np.round(z[1])) , fontsize = 16)

plt.xlabel('Le grand pas b du cavalier')
plt.ylabel('Le temps i')

plt.xticks(np.arange(min(abscisses), max(abscisses)+1, 1.0))
plt.yticks(np.arange(min(ordonnees), max(ordonnees)+1, 5.0))

plt.show()
```

On obtient alors le graphique suivant en tant qu'output :

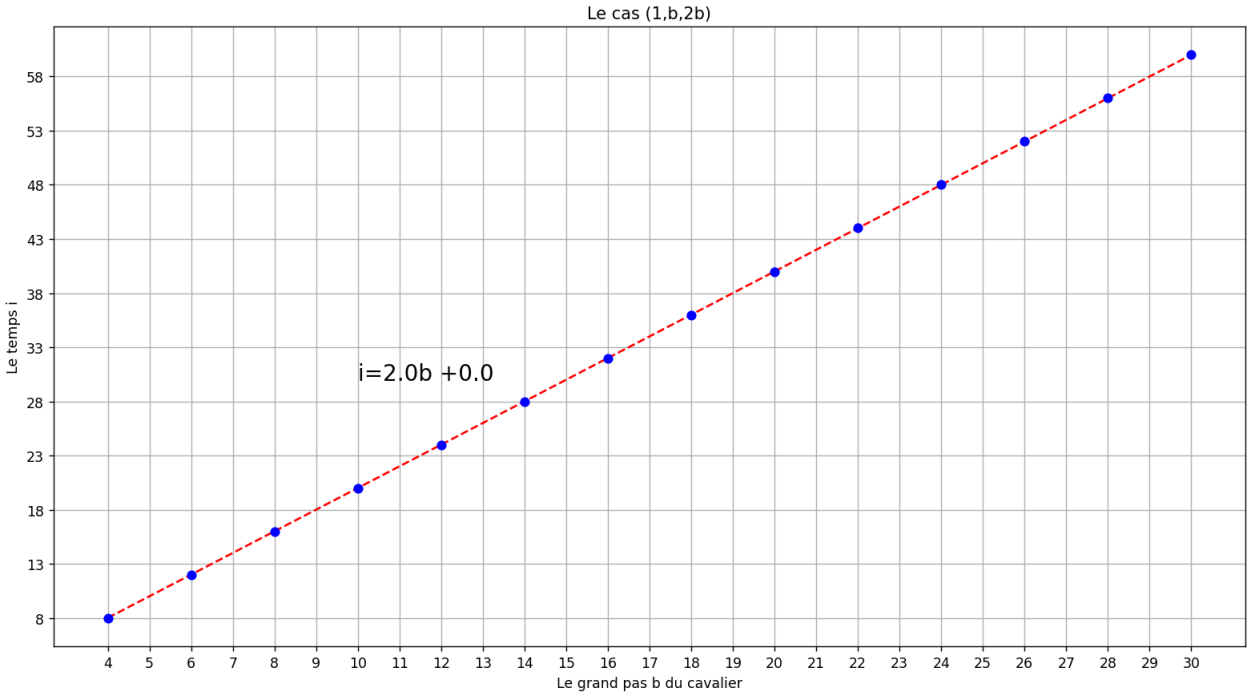


On peut alors formuler la conjecture suivante :

Conjecture 1. *Si le cavalier se déplace d'un pas* $[b-1, b]$ *dans un échiquier de taille* $2b$*, pour* $b > 1$*, alors le temps pour atteindre toutes les cases est donné par* $i = 4b - 3$

Cette relation a été observée à partir de la relation par récurrence $i_b = 4 \cdot i_{b-1} + i_0$ lors de quelques essais sur le temps en faisant varier b manuellement. Il a par ailleurs été observé que la relation n'est plus valide lorsque $b = 1$, c'est-à-dire lorsque le cavalier effectue un pas $[0, 1]$ dans un échiquier de taille 2.

Similairement, lorsqu'on calcule le temps pour le pas $[1, b]$ dans un échiquier de taille $2b$, avec $b > 1$ pair (pour que le cavalier puisse atteindre toutes les cases), et qu'on trace le temps en fonction de b on obtient la relation suivante :



On peut alors formuler la conjecture suivante :

Conjecture 2. *Si le cavalier se déplace d'un pas* $[1, b]$ *dans un échiquier de taille* $2b$*, pour* $b > 2$ *pair, alors le temps pour atteindre toutes les cases est donné par* $i = 2b$.

Cette relation a été observée en remarquant que le temps double par rapport au pas. Mais la relation ne tient plus pour $b = 2$.

En outre de prouver ces conjectures, la portée de l'application peut être étendue en identifiant d'autres relations entre le temps et d'autres combinaisons de pas, afin d'établir des relations permettant de mieux comprendre le parcours en largeur de ce graphe. Il est également intéressant de considérer un algorithme de parcours en profondeur.

5 | Conclusion

Les deux résultats principaux lors de ce projet sont énoncés ci-dessous :

Si $a \not\equiv b\ [2]$ et si $a \wedge b = 1$ alors le cavalier peut parcourir toutes les cases de l'échiquier.

Lorsque le cavalier effectue un pas $[a, b]$ avec $a \not\equiv b\ [2]$ et $a \wedge b = 1$, alors la taille minimale de l'échiquier pour qu'il puisse atteindre toutes les cases est de $n = 2 \cdot \max(a, b)$.

Il est remarquable que lors de ce projet, peu de notions mathématiques avancées ont été utilisées pour formuler et démontrer les théorèmes. Il s'agissait surtout de parité, d'arithmétique et de divisibilité.
Une possible addition à ce projet est de s'intéresser plus en détail sur le temps et de ses relations avec le pas. Cela permettrait possiblement de déterminer un parcours du cavalier minimal, ou sans repasser par une case visitée.
Le sujet se prête bien comme introduction à la théorie des graphes. En effet, le cadre des applications python peut être étendu sur des graphiques plus poussés. En particulier, des algorithmes plus complexes peuvent suggérer des résultats supplémentaires concernant ce problème, donnant lieu à plus de conjectures, voire des idées de preuves.

Le sujet du projet se révèle être en thème avec la matière des « mathématiques expérimentales », puisque nos recherches ont commencé par « trial and error » sur des feuilles à carreaux. Lentement, mais sûrement on a pu développer des théories et observer des motifs dans le parcours du cavalier.

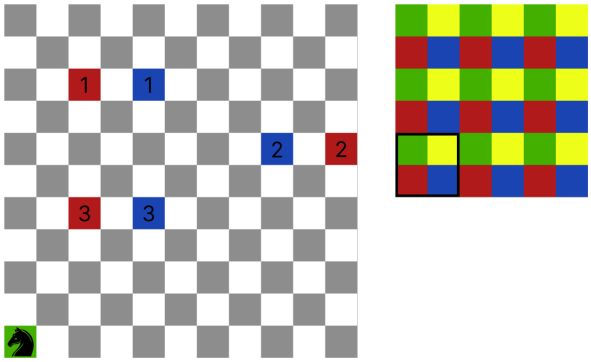
On adresse nos sincères remerciements à notre superviseur Louis Gass pour son soutien et ses conseils lors de ce projet.

6 | Annexe

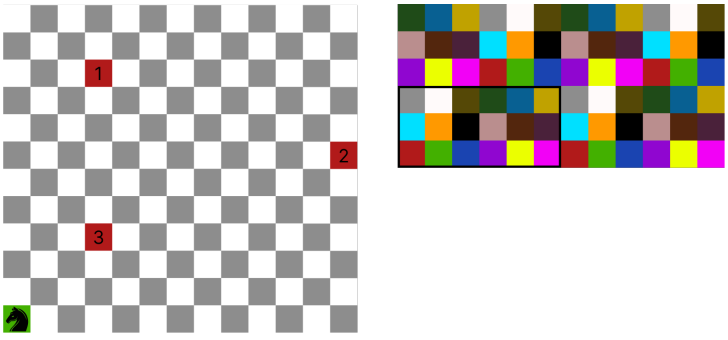
6.1 Réduction de pas

On a découvert qu'on peut réduire, c'est-à-dire rendre plus petit, le pas $[a, b]$ à $[a, b \ mod \ 2a]$ avec les 3 pas suivants : Le premier pas est $[a, b]$, le deuxième pas est $[b, -a]$ et le troisième pas est $[-b, -a]$. Ceci nous a permis de comprendre comment le pas et les cases atteignables sont reliés, ce qui nous a mené à l'algorithme d'Euclide et à l'utilisation du pgcd.

À gauche se trouve l'algorithme de 3 pas, appliqué deux fois, pour réduire le pas $[2, 8]$ au pas $[2, 4]$ puis au pas $[0, 2]$. À droite, les 4 orbites possibles :



À gauche l'algorithme de 3 pas, appliqué une fois, pour réduire le pas $[3, 9]$ au pas $[3, 3]$. À droite, les 18 orbites possibles, en remarquant que chaque couleur représente une orbite et on a comme motif un rectangle de taille 6×3 :



6.2 Programme Python : Dessin des orbites

Ci-dessous se trouvent notre premier Python code, on s'est basés sur les cinq théorèmes qu'on avait assemblés tout au début. Le code commence en nous demandons un a et un b avec la condition que $a \leq b$. À l'aide de ses données, le code construit l'échiquier le plus petit pour que le cavalier puisse encore se déplacer sans sauter en dehors du échiquier. Après avec ce a et b le code passe par les 5 conditions de nos conjectures et pour celui qui correspond ,il trace la première orbite, c'est-à-dire toutes les cases que le cavalier peut atteindre à partir de la case initiale.

```
import turtle
import math

print("Notez que a =< b")
a = int(input("Donnez a non nul: "))
b = int(input("Donnez b: "))
o = 0

#size of board
n = 2*(a+b) + 1
#size of square
x = 30
screen= turtle.Screen()
pen = turtle.Turtle()
def square():
  for i in range(4): #4 sides for a square, number of sides
    pen.forward(x)
    pen.left(90) #90 to draw squares, it's the angle
  pen.forward(x)

def thm134_impair():
        for i in range(n):
          pen.up()
          pen.setpos(-600, -300+ x * i)
          pen.down()
          for j in range(n):
            if (i+j)%2 == 0:
                color = 'black'
```

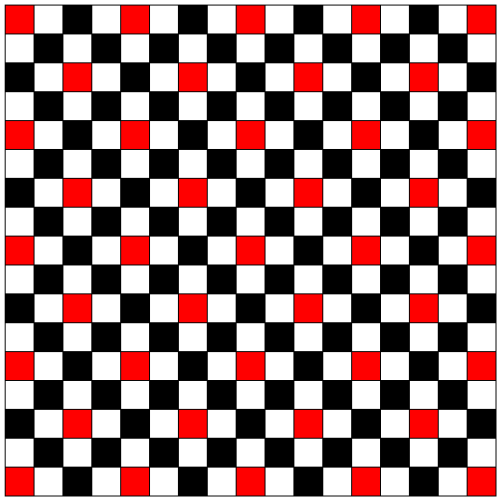
```
                if i%a == 0 and j%a ==0:           #these conditions for a (a,a)
                      color ='red'
            else:
                color ='white'
                if i%a == 0 and j%a ==0:           #these conditions for (a,0)
                    color ='red'
            pen.fillcolor(color)
            pen.begin_fill()
            square()
            pen.end_fill()

def thm25_impair():
        for i in range(n):
          pen.up()
          pen.setpos(-600, -300+ x * i)
          pen.down()
          for j in range(n):
            if (i+j)%2 == 0:
                color = 'black'
                if i%a == 0 and j%a ==0:           #these conditions for a (a,a)
                      color ='red'
            else:
                color ='white'
            pen.fillcolor(color)
            pen.begin_fill()
            square()
            pen.end_fill()

def thm134_pair():
        for i in range(n):
          pen.up()
          pen.setpos(-600, -300+ x * i)
          pen.down()
          for j in range(n):
            if (i+j)%2 == 0:
                color = 'black'
                if i%a == 0 and j%a ==0:    #the first two condition to have a (a,0)
                      color ='red'
```

```
            else:
              color ='white'
              if i%a == 0 and j%a ==0:
                  color ='red'
            pen.fillcolor(color)
            pen.begin_fill()
            square()
            pen.end_fill()

def thm25_pair():
        for i in range(n):
          pen.up()
          pen.setpos(-600, -300+ x * i)
          pen.down()
          for j in range(n):
          #the first two condition to have a (a,0) and the third for (a,a)
            if (i+j)%2 == 0:
                color = 'black'
                if i%a == 0 and j%a ==0 and (i+j)%(2*a) ==0:
                      color ='red'
            else:
              color ='white'
            pen.fillcolor(color)
            pen.begin_fill()
            square()
            pen.end_fill()
if __name__ == "__main__" :
    screen.setup(1920, 1080)
    pen.speed(100)
#tous les cas impairs
    if math.gcd(a,b) == 1 and (a+b)%2 == 1 and a%2 ==1:
        a = 1
        thm134_impair()
        o = 1
    elif math.gcd(a,b) == 1 and (a+b)%2 == 0 and a%2 ==1:
        a = 1
        thm25_impair()
        o = 2
```

```
    elif math.gcd(a,b) > 1 and b%a != 0 and a%2 ==1:
        a = math.gcd(a,b)
        thm134_impair()
        o = a**2
    elif b%a == 0 and (b/a)%2 == 0 and a%2 ==1:
        thm134_impair()
        o = a**2
    elif b%a == 0 and (b/a)%2 == 1 and a%2 ==1:
        thm25_pair()
        o = 2*a**2
#tous les cas pairs
    elif math.gcd(a,b) >= 1 and b%a != 0 and a%2 ==0:
        a = math.gcd(a,b)
        thm134_pair()
        o = math.gcd(a,b)**2
    elif b%a == 0 and (b/a)%2 == 0 and a%2 ==0:
        thm134_pair()
        o = a**2
    elif b%a == 0 and (b/a)%2 == 1 and a%2 ==0:
        thm25_pair()
        o = 2*a**2
    pen.hideturtle()

print("Le nombre d'orbites est:", o)
```

Voici quelques outputs :

```
Notez que a =< b
Donnez a non nul : 2
Donnez b : 6
Le nombre d’orbites est : 8
```



```
Notez que a =< b
Donnez a non nul : 3
Donnez b : 9
Le nombre d’orbites est : 18
```