TIPE : Le Sudoku

TARDY Zaccharie

June 6, 2024

Règle du jeu, Problématisation

- **But :** remplir la grille avec des chiffres de 1 à 9 selon certaines règles

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

Figure: Exemple de grille avec une unique solution

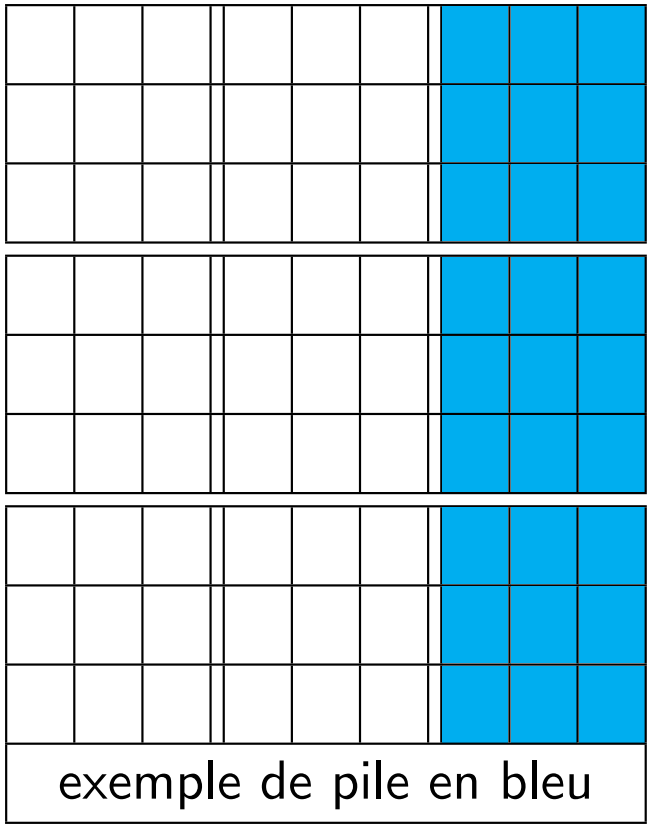
Problématiques

1. Combien de grilles de sudoku complètes existe-il ? Comment classer la difficulté de résolution d'une grille ?
2. Comment résoudre efficacement une grille de sudoku ?

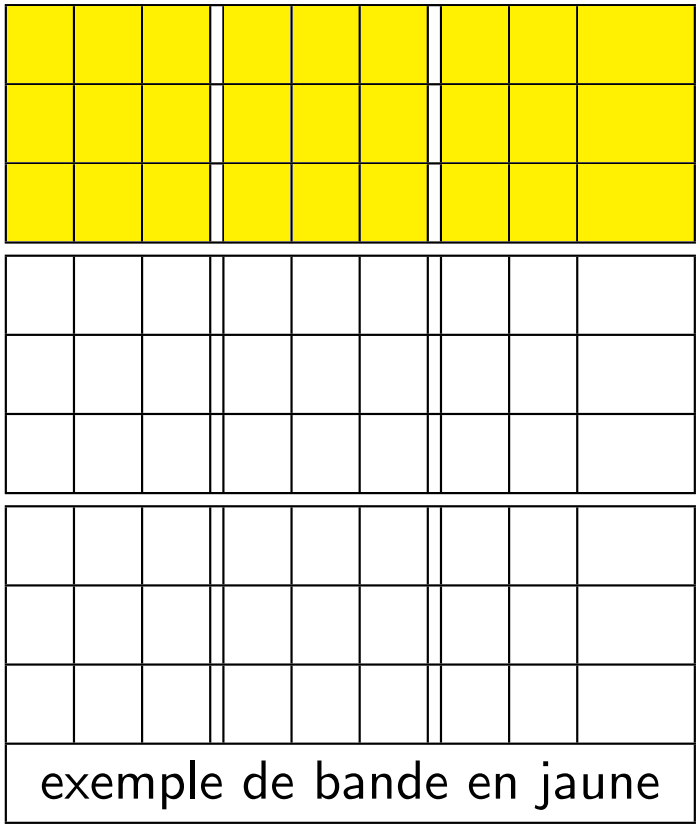
Les Notations

Definition

- Une pile est une suite de blocs adjacents sur l'axe vertical.
- Une bande est une suite de blocs adjacents sur l'axe horizontal.



exemple de pile en bleu



exemple de bande en jaune

Dénombrement des grilles complètes

Il existe 6 670 903 752 021 072 936 960 grilles complètes.

2	8	7	3	1	6	4	5	9
1	4	6	2	9	5	8	3	7
9	3	5	4	7	8	2	6	1
8	5	2	1	3	4	7	9	6
7	9	4	6	5	2	3	1	8
6	1	3	7	8	9	5	4	2
4	2	8	9	6	3	1	7	5
3	7	9	5	2	1	6	8	4
5	6	1	8	4	7	9	2	3

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

Figure: Grille tournée de $-\pi/2$

Dénombrement des grilles complètes distinctes

Definition

Soit $(G, \cdot)$ un groupe, de neutre e, et X un ensemble fini. Une action de groupe de G sur X est une application :

$$\star : G \times X \to X, \quad (g, x) \mapsto g \star x$$

vérifiant :

$$\forall (g, g') \in G^2, \forall x \in X, g \star (g' \star x) = (g \cdot g') \star x$$

$$\forall x \in X, e \star x = x$$

Definition

L'orbite de x de X est :

$$O_x = \{y \in X \mid \exists g \in G : y = g \star x\}$$

Un exemple simple

Théorème (Lemme de Burnside)

Pour $g \in G$, $\mathrm{Fix}(g) = \{x \in X; g \star x = x\}$. Le nombre d'orbites de X distinctes sous l'action de G vaut :

$$\frac{1}{|G|} \sum_{g \in G} |\mathrm{Fix}(g)|$$

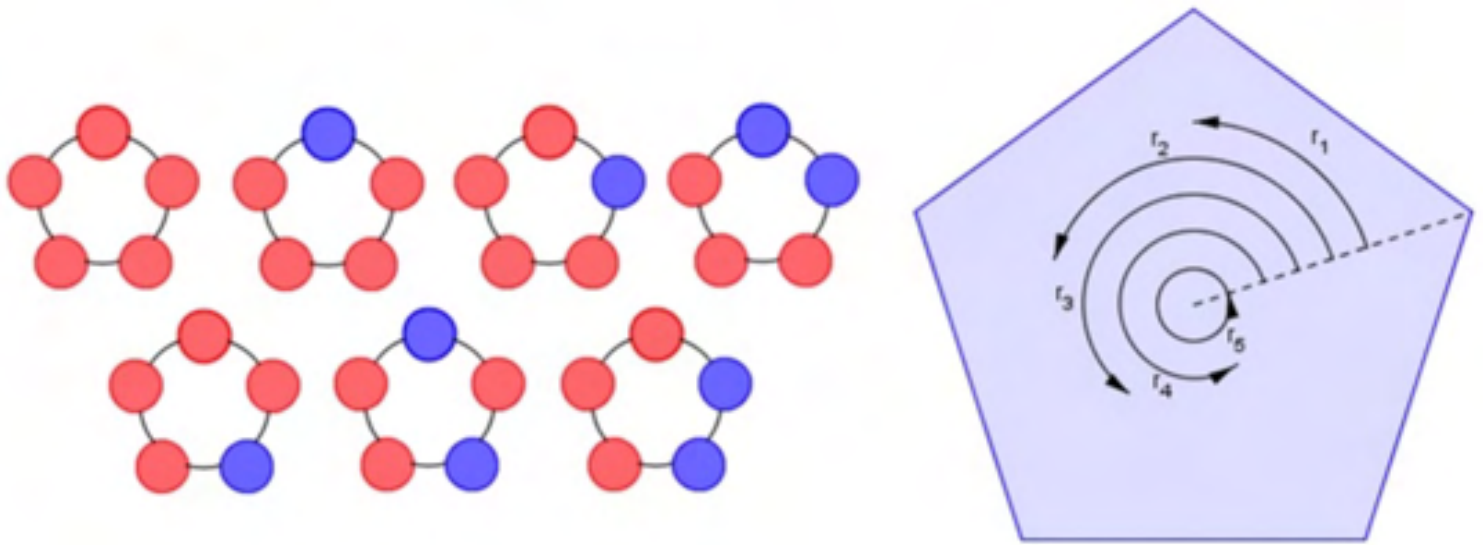


Figure: Exemples des colliers

Application au sudoku

Definition

On construit le groupe $G \subset S_{81}$ engendré par :

1. Permutations des colonnes dans une pile
2. Permutations des piles
3. Permutations des lignes d'une bande
4. Permutations des bandes
5. Transposition (et rotations)

1	2	3	4	5	6	7	8	9
10	11	12	13	14	15	16	17	18
19	20	21	22	23	24	25	26	27
28	29	30	31	32	33	34	35	36
37	38	39	40	41	42	43	44	45
46	47	48	49	50	51	52	53	54
55	56	57	58	59	60	61	62	63
64	65	66	67	68	69	70	71	72
73	74	75	76	77	78	79	80	81

```
gap> Size(sudoku);
3359232
gap> NrConjugacyClasses(sudoku);
275
```

Dénombrement des grilles fixées pour ces opérations

Un exemple en étapes:

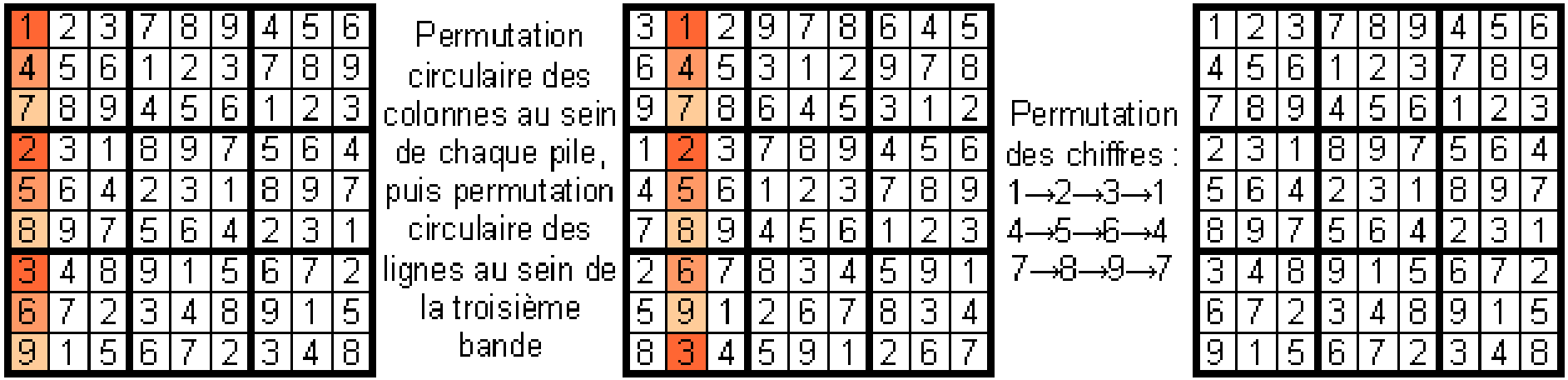


Figure: Cette grille est invariante (à une similitude près) sous l'opération "permutation colonne au sein de chaque pile"

Tableau des classes de conjugaisons

Nature	Taille	Représentant	Nombre de grilles invariantes
Identité	1	()	1838322242069299 2
Permutations des colonnes dans les piles et permutations lignes dans la dernière bande	96	(1 2 3)(4 5 6)(7 8 9)(10 11 12)(13 14 15)(16 17 18)(19 20 21)(22 23 24)(25 26 27)(28 29 30)(31 32 33)(34 35 36)(37 38 39)(40 41 42)(43 44 45)(46 47 48)(49 50 51)(52 53 54)(55 74 66)(56 75 64)(57 73 65)(58 77 69)(59 78 67)(60 76 68)(61 80 72)(62 81 70)(63 79 71)	21233664
Permutations des bandes et permutations des colonnes dans la dernière pile	3888	(1 55 28)(2 56 29)(3 57 30)(4 58 31)(5 59 32)(6 60 33)(7 61 34)(8 63 35 9 62 36)(10 64 46 19 73 37)(11 65 47 20 74 38)(12 66 48 21 75 39)(13 67 49 22 76 40)(14 68 50 23 77 41)(15 69 51 24 78 42)(16 70 52 25 79 43)(17 72 53 27 80 45)(18 71 54 26 81 44)	0

Résultats finaux

5472730538 grilles de sudokus sont réellement différentes !

17 est le nombre minimum de données nécessaire pour avoir une grille valide

Grille Sudoku à 17 données

7			1		8			
	9						3	2
					5			
						1		
9	6			2				
						8		
		5			1			
3	2							6

La solution

7	5	2	1	3	8	6	9	4
1	9	8	7	4	6	5	3	2
4	3	6	2	9	5	7	8	1
2	8	3	4	5	9	1	6	7
9	6	1	8	2	7	3	4	5
5	7	4	6	1	3	8	2	9
6	1	9	3	7	2	4	5	8
8	4	5	9	6	1	2	7	3
3	2	7	5	8	4	9	1	6

Méthode de résolution intuitive

Recherche d'une case vide

3	6	9	8	7	2	5	1	0
4	1	8	0	5	6	2	0	0
2	0	0	0	9	0	0	0	6
1	7	5	0	3	8	6	0	0
8	3	0	5	0	0	1	7	9
0	0	6	1	2	0	8	5	3
0	9	0	6	1	3	0	0	8
6	2	0	7	8	9	0	3	5
0	8	0	0	4	0	9	6	0

Remplissage

3	6	9	8	7	2	5	1	4
4	1	8	0	5	6	2	0	0
2	0	0	0	9	0	0	0	6
1	7	5	0	3	8	6	0	0
8	3	0	5	0	0	1	7	9
0	0	6	1	2	0	8	5	3
0	9	0	6	1	3	0	0	8
6	2	0	7	8	9	0	3	5
0	8	0	0	4	0	9	6	0

Backtracking

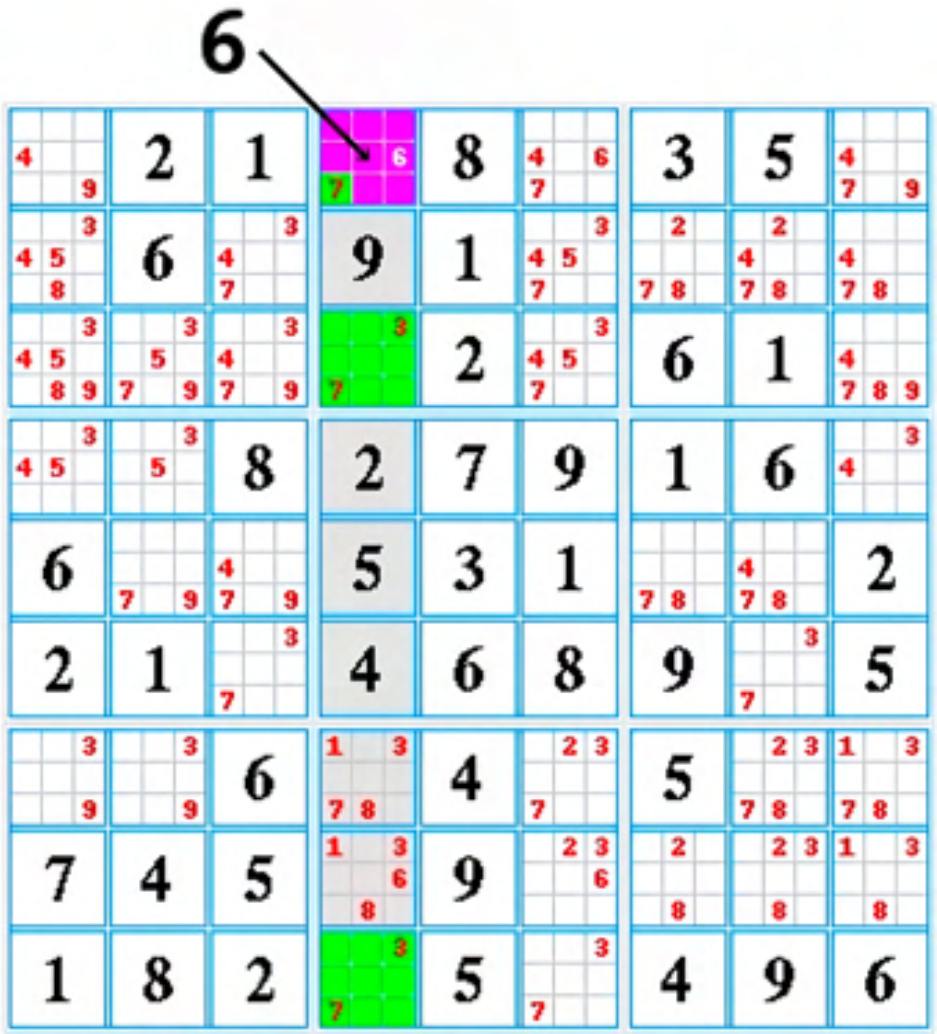


Figure: Exemple de grilles nécessitant une méthode de résolution plus avancée

Méthode des projections alternées

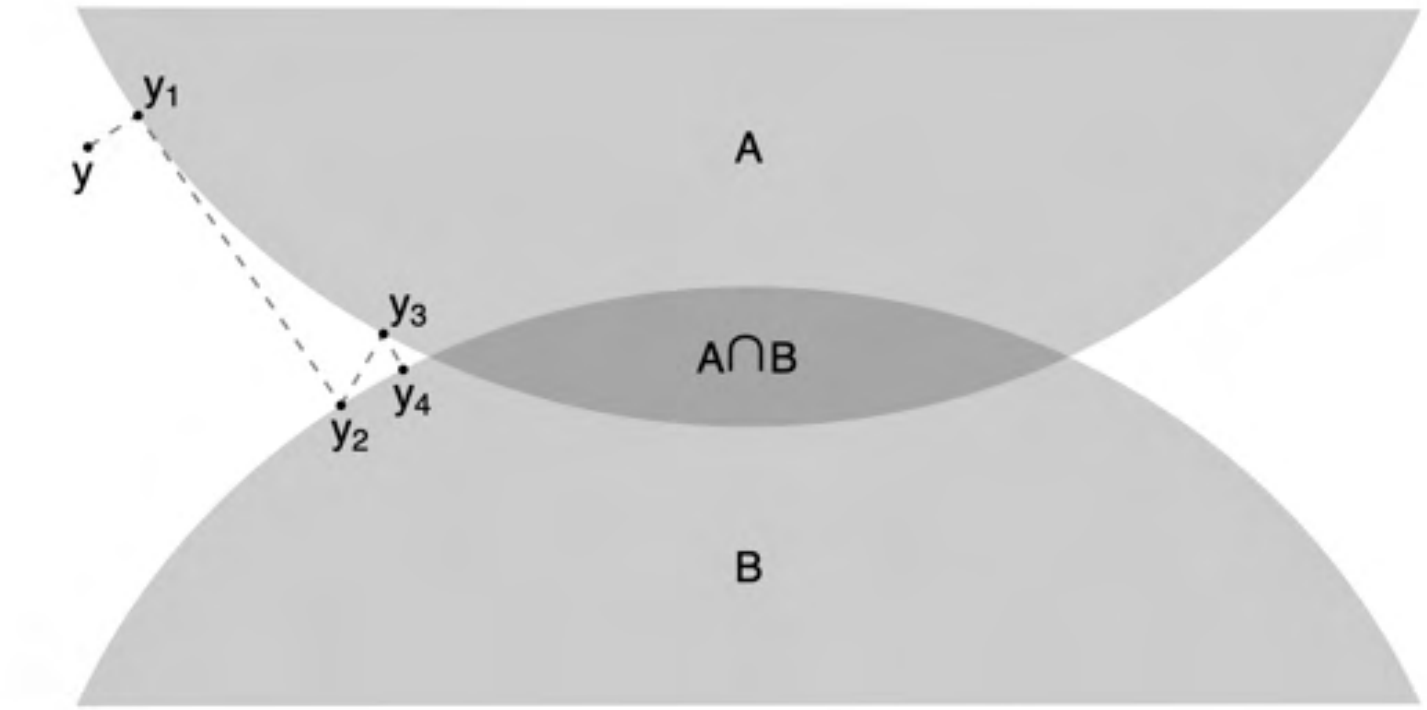


Figure: Projections alternées sur des convexes

Méthode des projections alternées

7	1	9		5		4	3	6
8			3	4	6		1	9
3		6	7		1	8	2	
9	3	1	8		4	6		
		8	9	6	2	1		
6	7							
		7		3	9	2	8	
1	8	4	6	2		3	9	7
	9		4			5	6	1

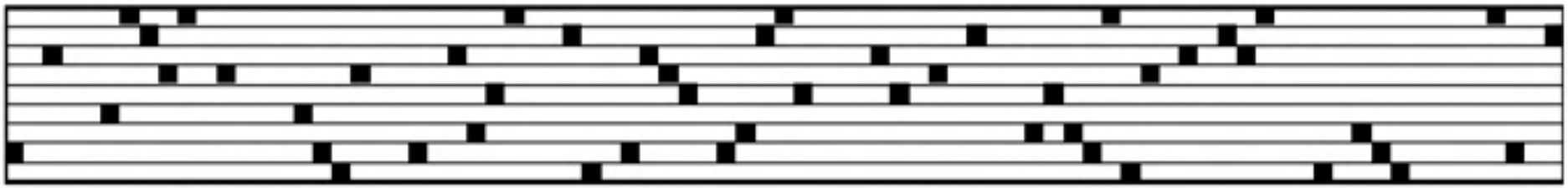


Figure: Vecteur découpé selon les contraintes (1 = noir, 0 = blanc)

Méthode des projections alternées

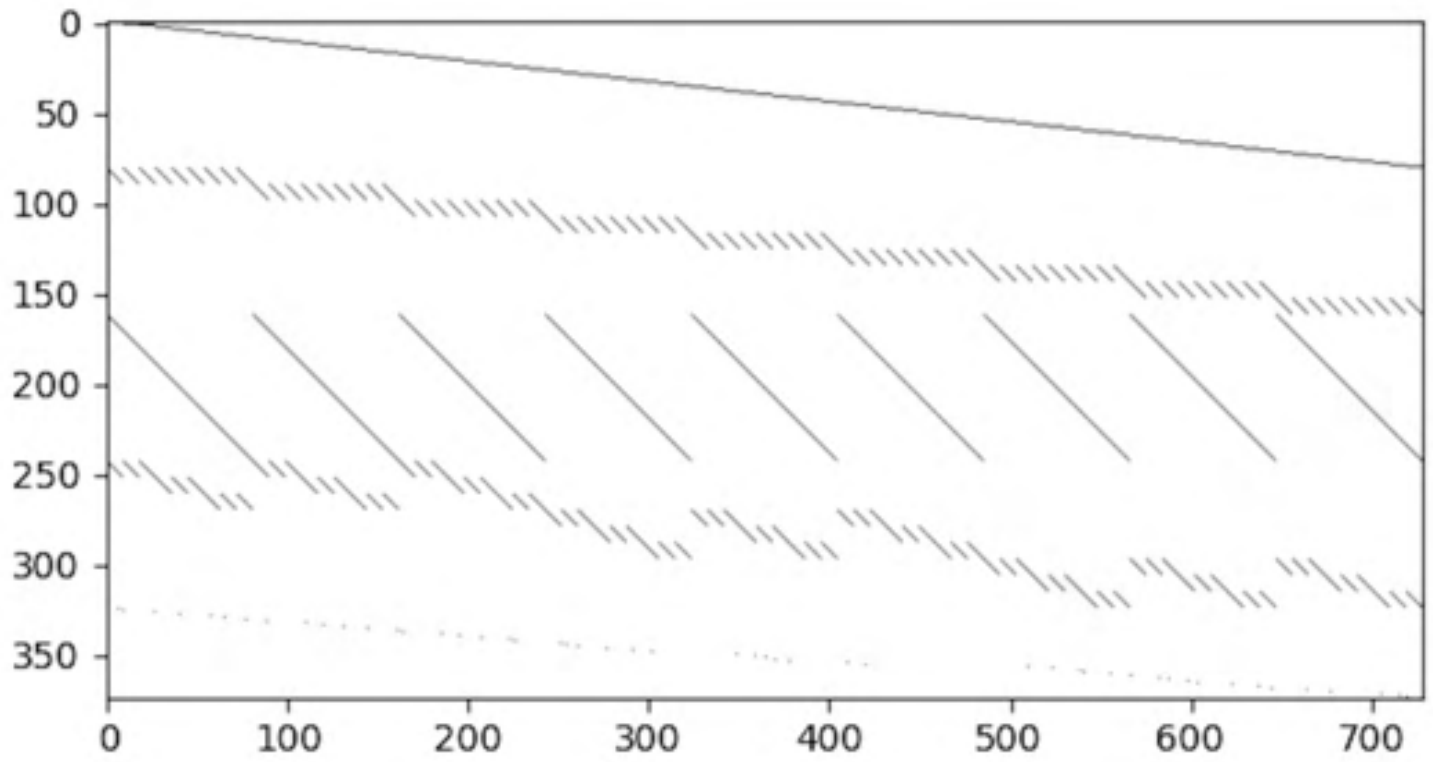


Figure: Matrices des contraintes pour l'exemple $A_{\text{contraintes}}$

$$V \text{ est solution} \Leftrightarrow A_{\text{contraintes}} V = 1 \text{ et } V \in \{0,1\}^{729}.$$

Méthode des projections alternées

$$S_{Ax=b} = \left\{ \underbrace{A^+ b}_{\text{Solution de norme minimale}} + \underbrace{\left(I - A^+A\right) V}_{\in \text{Ker}(A)} \mid V \in \mathbb{R}^{729} \right\}$$

Pseudo-inverse A^+

$$AA^+A = A \qquad A^+AA^+ = A^+ \qquad (AA^+)^* = AA^+ \qquad (A^+A)^* = A^+A$$
$$A^+ = \lim_{\delta \to 0}(A^*A + \delta I)^{-1}A^* = \lim_{\delta \to 0} A^*(AA^* + \delta I)^{-1}$$

Projections

Projection sur le segment [0,1]

$$PK_1(x_i) = \begin{cases} 0 & \text{si } x_i < 0 \\ x_i & \text{si } 0 \leq x_i \leq 1 \\ 1 & \text{si } x_i > 1 \end{cases}$$

Projection affine à l'aide du pseudo-inverse

$$PA(A, A^+, x) = A^+1 + (I - A^+A)x$$

Méthode des projections alternées

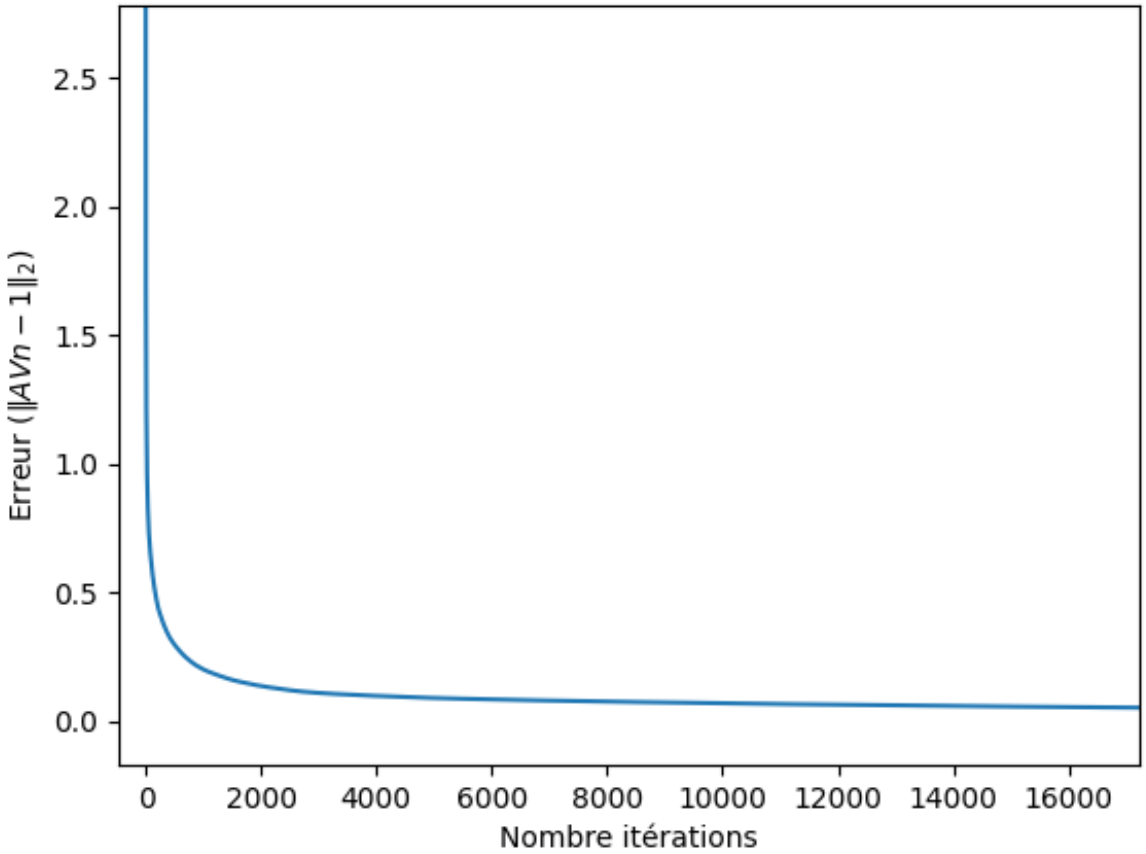


Figure: Evolution de l'erreur au cours des itérations

Les Résultats

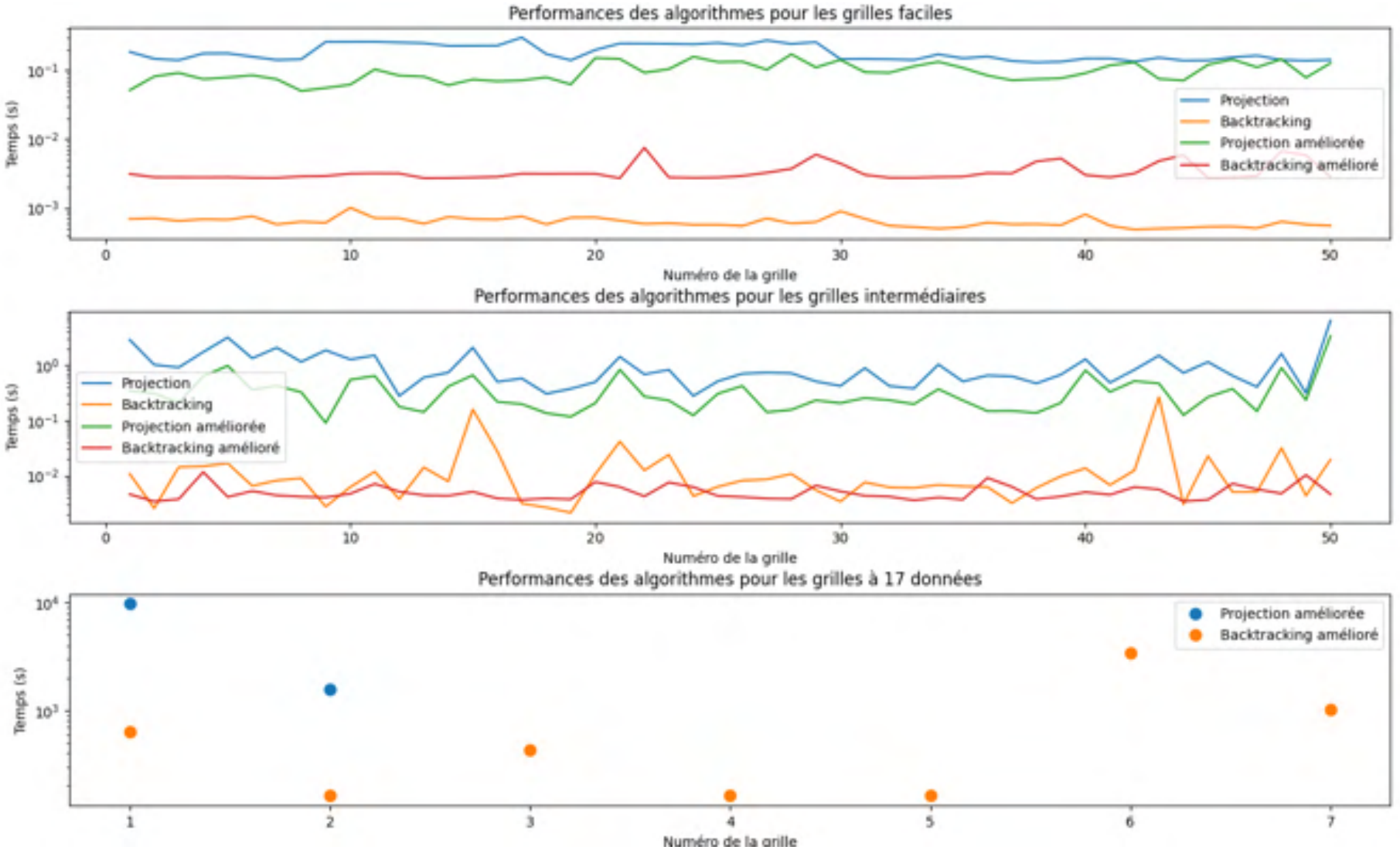


Figure: Temps de la résolution

Les Résultats

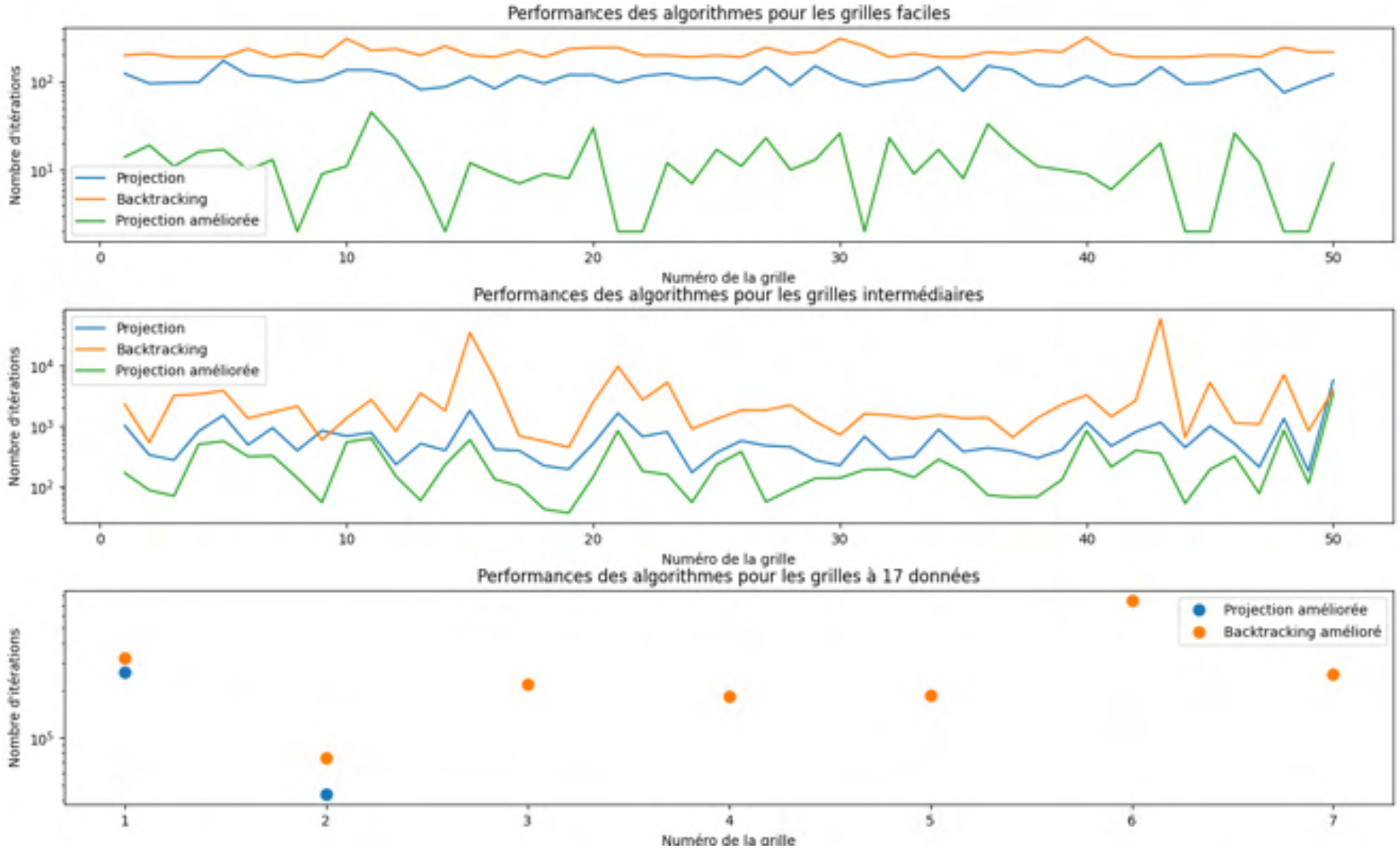


Figure: Nombre d'itérations pour la résolution

Conclusion

1. Des variantes du Sudoku : le Sudoku X

	3		8	4				
			9					
		5						
2	5				7	4	8	
		1					3	
	7	3						1
	4							
		8	6			9		
9								

6	3	9	8	4	1	2	7	5
7	2	4	9	5	3	1	6	8
1	8	5	7	2	6	3	9	4
2	5	6	1	3	7	4	8	
4	9	1	5	8	2	6	3	7
8	7	3	4	6	9	5	2	1
5	4	2	3	9	8	7	1	6
3	1	8	6	7	5	9	4	2
9	6	7	2	1	4	8	5	3

2. D'autres algorithmes
3. Recherche du nombre minimal de données nécessaires à la validité d'une grille

Annexe

```
def candidat_cache(grille):
    for i in range(9):
        for j in range(9):
            if grille[i][j] == 0:
                candidats = [1, 2, 3, 4, 5, 6, 7, 8, 9]
                for num in grille[i]:
                    if num in candidats:
                        candidats.remove(num)
                for num in [grille[x][j] for x in range(9)]:
                    if num in candidats:
                        candidats.remove(num)
                groupe_ligne = (i // 3) * 3
                groupe_col = (j // 3) * 3
                for ligne in range(groupe_ligne, groupe_ligne + 3):
                    for colonne in range(groupe_col, groupe_col + 3):
                        num = grille[ligne][colonne]
                        if num in candidats:
                            candidats.remove(num)
                if len(candidats) == 1:
                    grille[i][j] = candidats[0]
    return grille
```

Annexe

```
def encode(P):
    """
    encoder une grille de Sudoku en vecteur de taille 729
    """
    V = np.zeros((n**3))
    for i in range(n):
        for j in range(n):
            k = P[i,j]
            if k != 0:
                V[(n**2)*i+n*j+k-1] = 1
    return V

def decode(V):
    """
    decode un vecteur en grille de Sudoku
    """
    P = np.zeros((n,n), dtype='int')
    for i in range(n):
        for j in range(n):
            for k in range(n):
                if V[(n**2)*i+n*j+k] > 1/2:
                    P[i,j] = k+1
    return P
```

Annexe

```
# Initialisation des matrices de contraintes pour les cellules, lignes, colonnes et blocs
Acellules = np.zeros((n**2, n**3))
Acellules = np.zeros((n**2, n**3))
for i in range(n):
    for j in range(n):
        for k in range(n):
            Acellules[n*i+j, (n**2)*i+n*j+k] = 1
Alignes = np.zeros((n**2, n**3))
for i in range(n):
    for j in range(n):
        for k in range(n):
            Alignes[n*i+k, (n**2)*i+n*j+k] = 1

Acolonnes = np.zeros((n**2, n**3))
for i in range(n):
    for j in range(n):
        for k in range(n):
            Acolonnes[n*j+k, (n**2)*i+n*j+k] = 1


Ablocs = np.zeros((n**2, n**3))
for I in range(int(math.sqrt(n))):
    for J in range(int(math.sqrt(n))):
        numBloc = int(math.sqrt(n))*I+J
        for k in range(n):
            for i in range(int(math.sqrt(n))):
                for j in range(int(math.sqrt(n))):
                    Ablocs[n*numBloc +
                           k,(n**2)*(int(math.sqrt(n))*I+i)+n*(int(math.sqrt(n))*J+j)+k] =
                           1

Aregles = np.vstack((Acellules, Alignes, Acolonnes, Ablocs))
```

Annexe

```
# Combinaison des differentes matrices de contraintes en une seule
Aregles = np.vstack((Acellules, Alignes, Acolonnes, Ablocs))


def getApuzzle(P):
    """
    Construction de la matrice de contraintes pour une grille donnee
    """
    Apuzzle = Aregles
    for i in range(n):
        for j in range(n):
            k = P[i,j]
            if k != 0:
                indice = np.zeros(n**3)
                indice[n**2*i+n*j+k-1] = 1
                Apuzzle = np.vstack((Apuzzle, indice))
    return Apuzzle


def PK(x):
    """
    Projection sur le segment [0,1]
    """
    return np.minimum(1, np.maximum(0, x))

def PA(A,Ap,x):
    """
    Projection affine a l'aide du pseudo-inverse
    """
    return x - np.dot(Ap, np.dot(A,x) - 1)
```

Annexe

```
def est_bon(grille):
    """
    Verifie si une grille de Sudoku est correcte
    """
    for i in range(n):
        if len(set(grille[i])) != n or 0 in set(grille[i]):
            return False
    return True
def resolutionParProjections(P, tol, Niter):
    """
    Resolution du Sudoku par la methode des projections
    """
    Y=[]
    A = getApuzzle(P)
    Ap = np.linalg.pinv(A, rcond=1e-10)  #pseudo inverse
    Vn,L = np.zeros(n**3), []
    N=0
    for k in range(Niter):
        error = np.linalg.norm(np.dot(A, Vn) - 1)
        N = N+1
        if error < tol:
            return Vn,L
        Vn = PK(PA(A, Ap,Vn))
    print("Probleme")
    return Vn,L
```

Annexe

```
def resolutionParProjectionsameliore(P, tol, Niter):
    """
    Resolution du Sudoku par la methode des projections
    """
    Y=[]
    A = getApuzzle(P)
    Ap = np.linalg.pinv(A, rcond=1e-10)  #pseudo inverse
    Vn,L = np.zeros(729), []
    N=0
    for k in range(Niter):
        error = np.linalg.norm(np.dot(A, Vn) - 1)
        N=N+1
        a=candidat_cache(decode(Vn))
        if error < tol or est_bon(a):
            return encode(a),L
        Vn = PK(PA(A, Ap,Vn))
    print("Probleme")
    return Vn,L
```

Annexe

```
def verif_hypothese(grille, candidatures) :
    """fonction utilisant le backtracking et la resolution rapide"""
    copie_grille = copy.deepcopy(grille)
    copie_candidats = copy.deepcopy(candidatures)
    while True:
        if not candidat_cache(grille) : break
    candidatures = candidature(grille)
    copie_can = copy.deepcopy(candidatures)
    for nb_candidats in range(2, 9) :
        for i in range(8, -1, -1) :
            for j in range (9) :
                if len(candidatures[i][j]) == nb_candidats : #Si bon nombre de
                     candidats, on les essaye
                    for num in range (nb_candidats) :
                        grille[i][j] = candidatures[i][j][num]
                        candidatures[i][j] = []
                        if verif_hypothese(grille, candidatures,i) : #On regarde si
                             l'hypothese permet de finir de remplir la grille
                              return True
                        grille[i][j] = 0 #Si non, on reinitialise la grille et les
                             candidats
                        candidatures = copy.deepcopy(copie_can)
                    copy_grille(copie_grille, grille)
                    copy_cand(copie_candidats, candidatures)
                    return(False)
            #Si une ligne est fausse, une hypothese precedente etait fausse
            if test_ligne(grille, candidatures, i) == False :
                copy_grille(copie_grille, grille)
                copy_cand(copie_candidats, candidatures)
                return False
    global solution #On renvoie la grille resolue dans la variable globale solution
    solution = copy.deepcopy(grille)
    return(True)
```

Annexe

```
def copy_grille (source, dest) :
    for i in range (9) :
        for j in range (9):
            dest[i][j] = source[i][j]

def copy_cand (source, dest) :
    for i in range (9) :
        for j in range (9) :
            for k in range (len(source[i][j])) :
                dest[i][j].append(source[i][j][k])
def test_ligne(grille, candidatures, i):
    """
    Verifie si une ligne est correctement remplie
    Renvoie:
        False si il y a une case vide sans candidats sur la ligne
    """
    liste = []
    #Parcourt toutes les cases de la ligne
    for j in range (9) :

        if grille[i][j] == 0 and len(candidatures[i][j]) == 0 :
            return(False)
        if grille[i][j] != 0 and grille[i][j] in liste :
            return(False)
        liste.append(grille[i][j])
    return(True)
```

Annexe

```
def resoudre_sudoku(grid,i):
# Trouver la prochaine case vide
    for ligne in range(9):
        for col in range(9):
            if grid[ligne][col] == 0:
                for num in range(1, 10):
                    N[i]=N[i]+1
                    if est_valide(grid, ligne, col, num):
                        grid[ligne][col] = num
                        if resoudre_sudoku(grid,i):
                            return True
                        grid[ligne][col] = 0
                return False
def est_valide(grid, ligne, col, num):
    for i in range(9):
        if grid[ligne][i] == num:
            return False
    for j in range(9):
        if grid[j][col] == num:
            return False
    region_ligne = (ligne // 3) * 3
    region_col = (col // 3) * 3
    for i in range(region_ligne, region_ligne + 3):
        for j in range(region_col, region_col + 3):
            if grid[i][j] == num:
                return False
    return True
```

Puisque A est de rang p, l'application $X \mapsto AX$ qui va de $M_{p,1}(\mathbb{R})$ dans $\mathrm{Im}(A)$ est injective. Or, $\inf\{\|AX - B\|; X \in M_{p,1}(\mathbb{R})\}$ est la distance de B à $\mathrm{Im}(A)$. Cette distance est atteinte uniquement au projeté orthogonal sur $\mathrm{Im}(A)$ (qui est de dimension finie) de B. Ce projeté orthogonal s'écrit de façon unique AX_0. On a

$$AX_0 = p_{\mathrm{Im}(A)}(B) \iff \forall Z \in \mathrm{Im}(A), AX_0 - B \perp Z$$

$$\iff \forall X \in M_{p,1}(\mathbb{R}), AX_0 - B \perp AX$$

$$\iff \forall X \in M_{p,1}(\mathbb{R}), (AX)^T(AX_0 - B) = 0$$

$$\iff \forall X \in M_{p,1}(\mathbb{R}), X^T(A^TAX_0 - A^TB) = 0 \iff A^TAX_0 = A^TB.$$

X_0 est donc bien l'unique solution de $A^TAX = A^TB$.

Annexe: Propriétés du Pseudo-inverse A^+ I

Dans le cas matriciel,

$$Q = A^+A \quad Q = Q^T \quad Q^2 = Q, \quad AQ = A \quad \text{et} \quad QA^+ = A^+;$$

On a :

$$AQ = A \quad \text{et} \quad QA^+ = A^+;$$

- Q est le projecteur orthogonal sur l'image de A^* (égal à l'orthogonal du noyau de A) ; - $I - Q$ est le projecteur orthogonal sur le noyau de A.
Le système

$$Ax = b$$

a des solutions si et seulement si

$$AA^+b = b,$$

Annexe: Propriétés du Pseudo-inverse A^+ II

et les solutions sont alors tous les vecteurs de la forme

$$x = A^+ b + \left(I - A^+ A\right) w,$$

où le vecteur w est arbitraire (si ce n'est sa dimension).
En effet (voir supra), l'image de A est égale au noyau de $I - AA^+$, et le noyau de A est égal à l'image de $I - A^+ A$.

Le système

$$Ax = b$$

a des solutions si et seulement si

$$AA^{+}b = b,$$

et les solutions sont alors tous les vecteurs de la forme

$$x = \underbrace{A^{+}b}_{\text{Solution de norme minimale}} + \underbrace{\left(I - A^{+}A\right)w}_{\in \text{Ker}(A)}$$

où le vecteur w est arbitraire (si ce n'est sa dimension).

Théorème

Soit X un ensemble fini et G un groupe fini agissant sur X. Pour tout $g \in G$, on pose

$$\mathrm{Fix}_X(g) = \{x \in X \mid g(x) = x\}$$

On note $\mathrm{Orb}_X(G)$ l'ensemble des G-orbites dans X. Alors

$$|\mathrm{Orb}_X(G)| = \frac{1}{|G|} \sum_{g \in G} |\mathrm{Fix}_X(g)|$$

Preuve

Supposons dans un premier temps que G agisse transitivement sur X (

$$\forall (x, y) \in X, \exists g \in G \text{ tel que } g \cdot x = y.$$

). Posons

$$S = \{(x, g) \in X \times G \mid g(x) = x\}$$

D'une part, on a

$$|S| = \sum_{g \in G} |\mathrm{Fix}_X(g)|$$

D'autre part, on a

$$|S| = \sum_{x \in X} |G_x|$$

où $G_x = \{g \in G \mid g(x) = x\}$ est le stabilisateur de x.

D'après le Théorème de Lagrange, comme X est un espace G-transitif, on a

$$|G_x| = \frac{|G|}{|X|}$$

D'où

$$|S| = \sum_{x \in X} |G_x| = |X| \cdot \frac{|G|}{|X|} = |G|$$

Ainsi

$$1 = |\mathrm{Orb}_X(G)| = \frac{1}{|G|} \sum_{g \in G} |\mathrm{Fix}_X(g)|$$

Supposons à présent que les G-orbites de X soient $X_1, \ldots, X_k$. Alors

$$\mathrm{Fix}_X(g) = \bigcup_{i=1}^{k} \mathrm{Fix}_{X_i}(g) \quad \text{et} \quad |\mathrm{Fix}_X(g)| = \sum_{i=1}^{k} |\mathrm{Fix}_{X_i}(g)|$$

D'où

$$\frac{1}{|G|} \sum_{g \in G} |\mathrm{Fix}_X(g)| = \frac{1}{|G|} \sum_{g \in G} \sum_{i=1}^{k} |\mathrm{Fix}_{X_i}(g)| = \sum_{i=1}^{k} \frac{1}{|G|} \sum_{g \in G} |\mathrm{Fix}_{X_i}(g)| = \sum_{i=1}^{k} 1 = k = |$$