Gradient conjugué et préconditionnement

Projet S1V2 - GM3

Quentin GAROT & Rojan KAYA

Novembre 2023

Table des matières

1 Introduction **3**

2 Discrétisation du problème **3**

3 Résolution du système linéaire associé à l'EDP $\Delta u = ku$ **4**
3.1 Propriétés le la matrice A 4
3.2 Méthode du gradient conjugué 5
3.2.1 Présentation de la méthode 5
3.2.2 Premiers résultats 5
3.3 Méthode du gradient conjugué avec préconditionnement (SSOR) 8
3.3.1 Présentation générale du préconditionnement 8
3.3.2 Préconditionnement SSOR et algorithme 8
3.3.3 Résultats et influence du paramètre w du préconditionneur 9
3.4 Comparaison des deux méthodes 11

4 Démarche pour l'implémentation en Python **13**

5 Conclusion **13**

1 Introduction

La résolution de système linéaire est un problème qui apparait dans de très nombreux contextes et est à la base de toute simulation numérique. Par exemple, la simulation numérique d'un phénomène physique décrit par une équation aux dérivées partielles (EDP) conduit à la résolution d'un système linéaire. Dans ce projet, intéressons nous à la résolution de l'EDP suivante :

$$\Delta u = ku \tag{1}$$

Après avoir écrit le problème sous forme discrétisée pour obtenir le système linéaire associé, nous étudierons la méthode itérative du gradient conjugué avec et sans préconditionnement. Nous finirons par expliquer la structure du code Python implémentant cette méthode et par montrer quelques domaines d'applications.

2 Discrétisation du problème

Plaçons nous dans le cas unidimensionnel, sur un domaine $\mathcal{D} = [0, 1]$. Fixons $n \in \mathbb{N}$ et posons $h := \frac{1}{n+1}$, $\forall i \in [\![1, n]\!]$, $x_i := ih$, $u_i := u(x_i)$, $u_0 := 1$ et $u_{n+1} = 0$. Cherchons maintenant une approximation de la dérivée seconde de u en chaque point x_i. En utilisant la formule de Taylor-Young, on obtient pour tout $i \in [\![1, n]\!]$:

$$u''(x_i) \approx \frac{u_{i+1} - 2u_i + u_{i-1}}{h^2} \tag{2}$$

Ainsi, l'équation (1) nous conduit à la résolution du système $(\mathcal{S})$ suivant :

$$(\mathcal{S}) : \forall i \in [\![1, n]\!], \ -\frac{u_{i+1} - 2u_i + u_{i-1}}{h^2} + ku_i = 0 \tag{3}$$

A partir de ce résultat, l'objectif est d'écrire le système linéaire de taille n associé. Raisonnons par équivalences :

$$(\mathcal{S}) \Leftrightarrow \forall i \in [\![1, n]\!], \ \left(-\frac{1}{h^2}\right) u_{i-1} + \left(k + \frac{2}{h^2}\right) u_i + \left(-\frac{1}{h^2}\right) u_{i+1} = 0$$

$$\Leftrightarrow \begin{cases} \left(-\frac{1}{h^2}\right) u_0 + \left(k + \frac{2}{h^2}\right) u_1 + \left(-\frac{1}{h^2}\right) u_2 = 0 \\ \left(-\frac{1}{h^2}\right) u_{n-1} + \left(k + \frac{2}{h^2}\right) u_n + \left(-\frac{1}{h^2}\right) u_{n+1} = 0 \\ \forall i \in [\![2, n-1]\!], \ \left(-\frac{1}{h^2}\right) u_{i-1} + \left(k + \frac{2}{h^2}\right) u_i + \left(-\frac{1}{h^2}\right) u_{i+1} = 0 \end{cases}$$

$$\Leftrightarrow \begin{cases} \left(k + \frac{2}{h^2}\right) u_1 + \left(-\frac{1}{h^2}\right) u_2 = \left(\frac{1}{h^2}\right) \underbrace{u_0}_{=1} \\ \left(-\frac{1}{h^2}\right) u_{n-1} + \left(k + \frac{2}{h^2}\right) u_n = \left(\frac{1}{h^2}\right) \underbrace{u_{n+1}}_{=0} \\ \forall i \in [\![2, n-1]\!], \ \left(-\frac{1}{h^2}\right) u_{i-1} + \left(k + \frac{2}{h^2}\right) u_i + \left(-\frac{1}{h^2}\right) u_{i+1} = 0 \end{cases}$$

$$\Leftrightarrow \forall i \in [\![1,n]\!],\ \sum_{j=1}^{n} a_{ij}u_j = b_i$$

où on pose :

$$\forall i,j \in [\![1,n]\!],\ a_{ij} = \begin{cases} k+\dfrac{2}{h^2} & \text{si } j=i \\ -\dfrac{1}{h^2} & \text{si } j=i\pm 1 \\ 0 & \text{sinon} \end{cases} \text{ et } b_i = \begin{cases} \dfrac{1}{h^2} & \text{si } i=1 \\ 0 & \text{sinon} \end{cases}$$

Cela permet de définir la matrice $A = (a_{ij})_{1\leq i,j\leq n}$ et le vecteur colonne $b = (b_i)_{1\leq i\leq n}$. Ainsi, on a :

$$(\mathcal{S}) \Leftrightarrow Au = b, \text{ où } u = (u_i)_{1\leq i\leq n} \tag{4}$$

Remarquons que le vecteur u contient les informations de la fonction recherchée à l'intérieur du domaine $\mathcal{D}$. Lors de l'implémentation informatique des méthodes, nous devrons prendre en compte les conditions aux bords du domaine, aussi appelées conditions de Dirichlet.

3 Résolution du système linéaire associé à l'EDP $\Delta u = ku$

3.1 Propriétés le la matrice A

Pour cette sous-partie, considérons $k \geq 0$. Notons alors que la matrice A est évidemment symétrique (par sa définition). De plus, pour $x \in \mathbb{R}^n$, on a :

$$\begin{aligned}
(Ax,x) &= \sum_{i=1}^{n} (Ax)_i x_i \\
&= \sum_{i=1}^{n} \left(-\frac{1}{h^2}x_{i-1} + \left(k+\frac{2}{h^2}\right)x_i + \frac{1}{h^2}x_{i+1} \right) x_i \\
&= \frac{1}{h^2}\sum_{i=1}^{n} \left(-x_{i-1}x_i + 2x_i^2 + x_{i+1}x_i \right) + k\sum_{i=1}^{n} x_i^2 \\
&= \frac{1}{h^2}\sum_{i=1}^{n} \left[\left(x_i^2 - 2x_ix_{i+1} + x_{i+1}^2\right) + \left(x_i^2 - x_{i+1}^2\right) + \left(x_ix_{i+1} - x_{i-1}x_i\right) \right] + k\sum_{i=1}^{n} x_i^2 \\
&= \frac{1}{h^2}\left[\sum_{i=1}^{n} (x_i - x_{i+1})^2 + \sum_{i=1}^{n} \left(x_i^2 - x_{i+1}^2\right) + \sum_{i=1}^{n} \left(x_ix_{i+1} - x_{i-1}x_i\right) \right] + k\sum_{i=1}^{n} x_i^2 \\
&= \frac{1}{h^2}\left(x_1^2 + \sum_{i=1}^{n-1} (x_i - x_{i+1})^2 + x_n^2 \right) + k\sum_{i=1}^{n} x_i^2 \geq 0 \text{ car } k \geq 0
\end{aligned}$$

Soit maintenant $x \in \mathbb{R}^n$ tel que $(Ax,x) = 0$. Il s'agit d'une somme nulle de termes positifs, donc en particulier :

$$x_1^2 = 0 \text{ et } \sum_{i=1}^{n-1} (x_i - x_{i+1})^2 = 0 \text{ et } x_n^2 = 0$$

Même argument pour la somme $\sum_{i=1}^{n-1}(x_i - x_{i+1})^2$:

$$x_1^2 = 0 \text{ et } x_n^2 = 0 \text{ et } \forall i \in [\![1,n-1]\!], \underbrace{(x_i - x_{i+1})^2 = 0}_{\text{donc } x_i = x_{i+1}} \text{ d'où } x = 0_{\mathbb{R}^n}$$

Ainsi, $\forall x \in \mathbb{R}^n \backslash \{0_{\mathbb{R}^n}\}$, $(Ax,x) > 0$, donc A est symétrique définie positive.

3.2 Méthode du gradient conjugué

3.2.1 Présentation de la méthode

La méthode du gradient conjugué est un algorithme permettant de résoudre des systèmes linéaires dont la matrice est symétrique définie positive. Expliquons le principe de cette méthode : soit le potentiel scalaire :

$$\begin{aligned} J : \mathbb{R}^n &\longrightarrow \mathbb{R} \\ x &\longmapsto \frac{1}{2}(Ax, x) - (b, x) \end{aligned}$$

J admet un unique minimum noté u qui est la solution du système linéaire (4). En effet, pour tout vecteur x de $\mathbb{R}^n$, $\nabla J(x) = Ax - b$. Or, u est le minimum de J donc $\nabla J(u) = 0$ ou encore $Au = b$. En partant d'un premier vecteur noté x_0 supposé nul, l'objectif est de se rapprocher petit à petit de la solution u, c'est-à-dire de diminuer la valeur prise par J à chaque itération. Cela suggère de se déplacer selon une certaine direction p_k et selon un certain pas α_k. Les vecteurs p_k ($k \geq 1$) seront conjugués au gradient, d'où le nom de la méthode. On construit alors la suite (x_k) à l'aide de son premier terme x_0 et de la relation de récurrence :

$$x_{k+1} = x_k + \alpha_k \rho_k$$

Algorithm 1 Pseudo-code de la méthode du gradient conjugué

$x_0 \leftarrow 0_{\mathbb{R}^n}$
$r_0 \leftarrow b - Ax_0$
$p_0 \leftarrow r_0$
$k \leftarrow 0$
repeat
  $\alpha_k \leftarrow \dfrac{r_k^T r_k}{p_k^T A p_k}$
  $x_{k+1} \leftarrow x_k + \alpha_k p_k$
  $r_{k+1} = r_k - \alpha_k A p_k$
  if r_{k+1} est suffisamment petit **then**
    Sortir de la boucle
  end if
  $\beta_k \leftarrow \dfrac{r_{k+1}^T r_{k+1}}{r_k^T r_k}$
  $p_{k+1} \leftarrow r_{k+1} + \beta_{k+1} p_k$
  $k \leftarrow k + 1$
Le résultat est x_{k+1}

3.2.2 Premiers résultats

L'implémentation de cet algorithme en Python nous permet par exemple d'obtenir le résultat suivant :

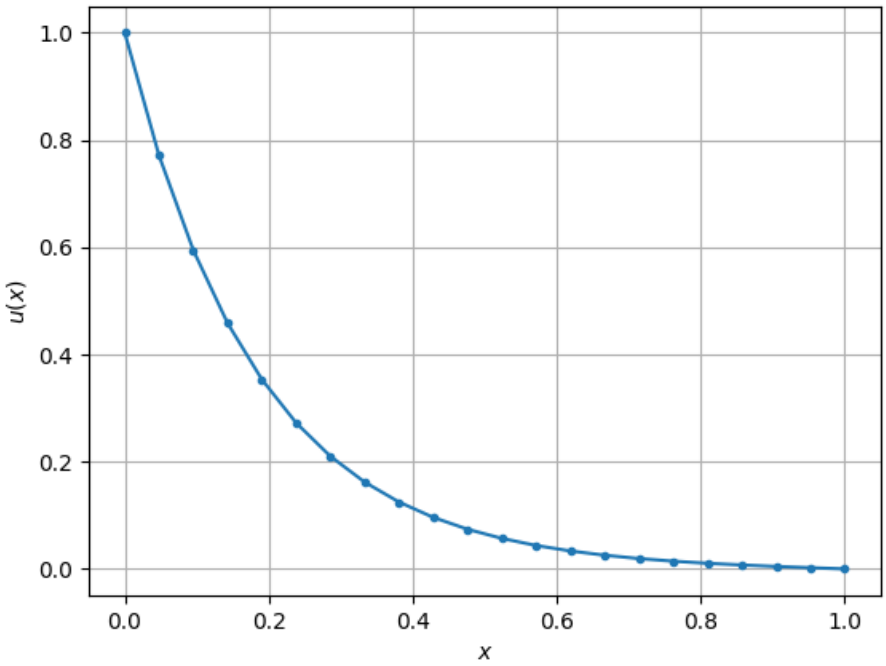

FIGURE 1 – Graphe de la solution u en fonction de x ($n = 20$, $k = 30$)

Comme nous pouvons le remarquer sur la figure ci-après, la dimension du système linéaire est un paramètre qui nous permet d'obtenir une précision plus ou moins correcte : plus elle est grande, plus l'approximation de la solution semble fiable.

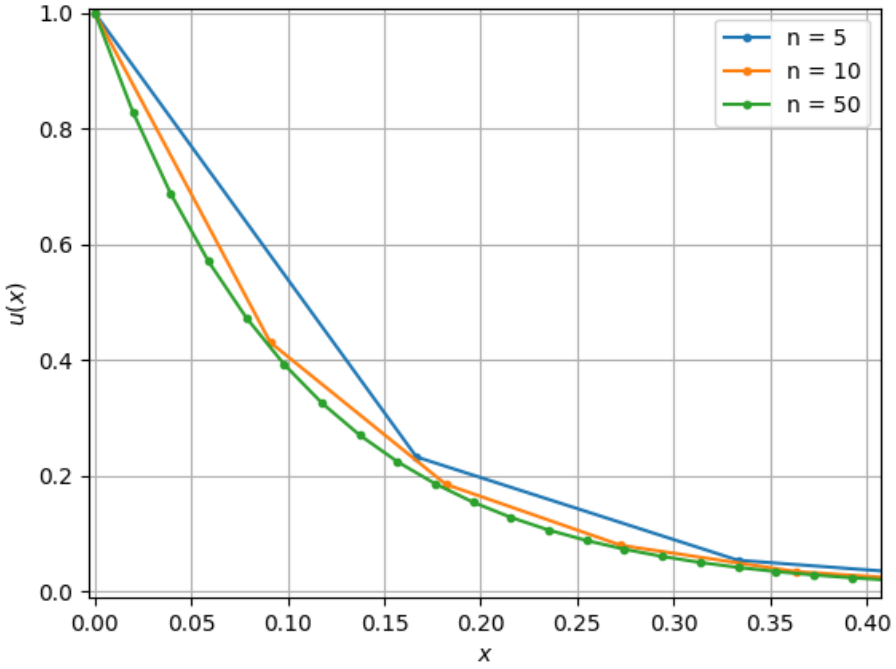

FIGURE 2 – Comparaison des solutions sur une partie de $\mathcal{D}$ en fonction de la taille du système linéaire ($k = 91$)

Pour rappel, nous avons établi dans la partie **3.1** que si le paramètre k est positif ou nul, alors la matrice A du système linéaire (4) est symétrique définie positive. Mais *a priori*, rien ne nous assure que cette propriété est toujours vérifiée lorsque k est strictement négatif. Testons alors l'algorithme avec de telles valeurs :

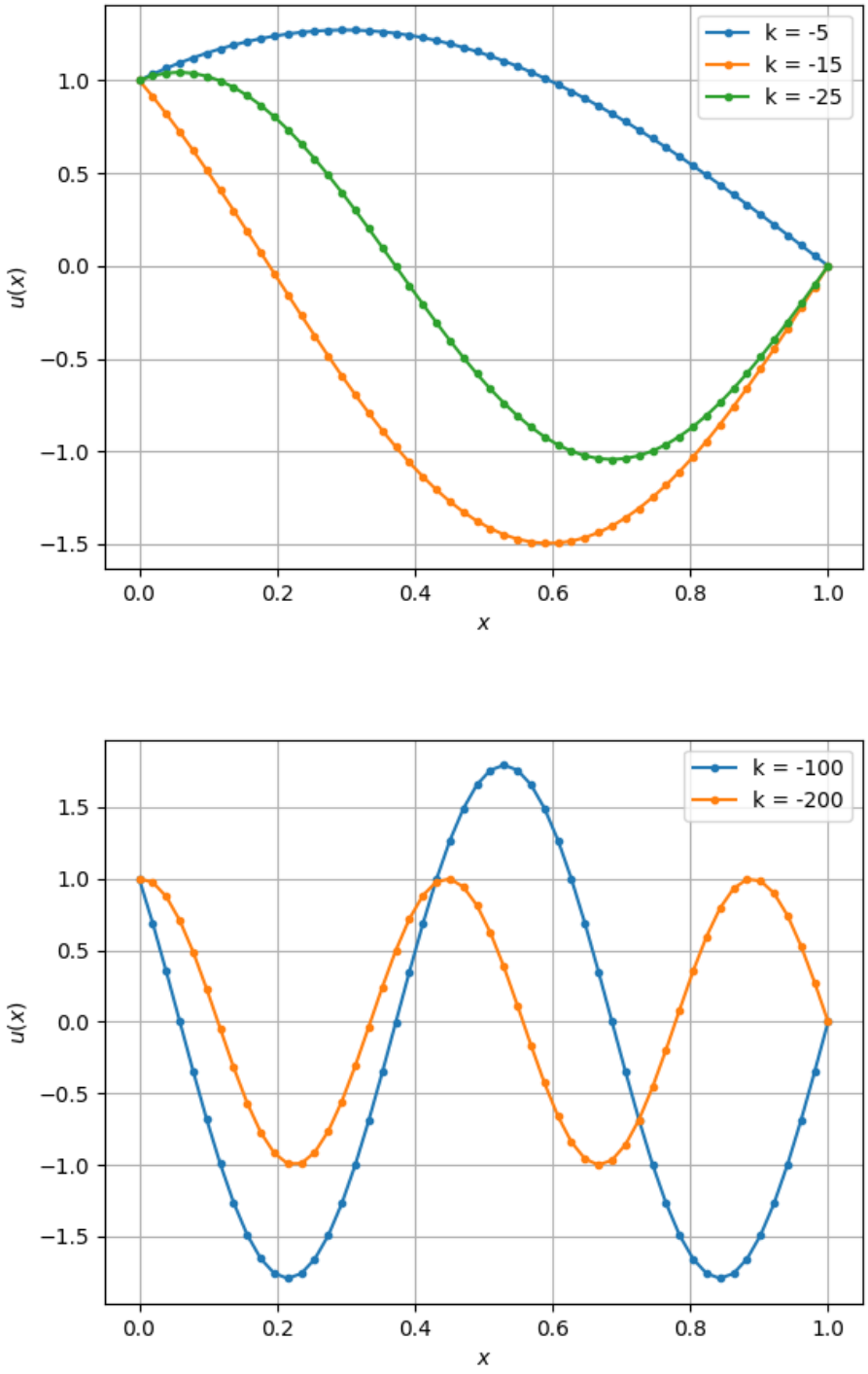

FIGURE 3 – Comparaison des solutions en fonction du paramètre $k < 0$

Pour ces valeurs de k testées, on remarque d'une part que la méthode du gradient conjugué converge toujours et d'autre part que les solutions se mettent à osciller.

3.3 Méthode du gradient conjugué avec préconditionnement (SSOR)

3.3.1 Présentation générale du préconditionnement

Commençons par définir ce qu'est un préconditionneur. On dit qu'une matrice P est un préconditionneur d'une matrice A lorsque le conditionnement de $P^{-1}A$ est plus petit que celui de A :

$$cond(P^{-1}A) < cond(A)$$

Ainsi, au lieu de résoudre $Au = b$, on préfère résoudre le système $P^{-1}Au = P^{-1}b$ puisque ce dernier est mieux conditionné. Nous verrons que cela permet de diminuer considérablement le nombre d'itérations et de ce fait, d'accélérer la convergence vers la solution du problème.

3.3.2 Préconditionnement SSOR et algorithme

Décomposons la matrice A du système (4) de la façon suivante :

$$A = D - E - U \tag{5}$$

où D correspond à la matrice diagonale comportant les éléments diagonaux de A, E une matrice triangulaire strictement inférieure et U une matrice triangulaire strictement supérieure. Comme A est symétrique, on a $U = E^T$. Le préconditionneur P SSOR est défini par :

$$P = \frac{1}{w\,(w-2)}(D - wE)D^{-1}(D - wE^T) \tag{6}$$

où w est un paramètre réel. Un point clé de la version préconditionnée de la méthode du gradient conjugué est la décomposition suivante :

$$P = TT^T \tag{7}$$

où $T = \dfrac{(D - wE)\,D^{-1/2}}{\sqrt{w\,(2-w)}}$. Notons que la matrice T est triangulaire inférieure.

La méthode du gradient conjugué avec préconditionnement SSOR peut être implémentée avec cet algorithme :

Algorithm 2 Pseudo-code de la méthode du gradient conjugué avec préconditionnement SSOR

$x_0 \leftarrow 0_{\mathbb{R}^n}$
$r_0 \leftarrow b - Ax_0$
$Pp_0 = r_0$ ▷ cf. explications ci-après
$z_0 \leftarrow p_0$
$k \leftarrow 0$
repeat
  $\alpha_k \leftarrow \dfrac{(r_k, z_k)}{(Ap_k, p_k)}$
  $x_{k+1} \leftarrow x_k + \alpha_k p_k$
  $r_{k+1} = r_k - \alpha_k Ap_k$
  if r_{k+1} est suffisamment petit **then**
    Sortir de la boucle
  end if
  $Pz_{k+1} = r_{k+1}$ ▷ cf. explications ci-après
  $\beta_{k+1} \leftarrow \dfrac{(r_{k+1}, z_{k+1})}{(r_k, z_k)}$
  $p_{k+1} \leftarrow z_{k+1} + \beta_{k+1} p_k$
  $k \leftarrow k + 1$
Le résultat est x_{k+1}

Expliquons en détail les expressions `Pp0 = r0` et `Pz(k+1) = r(k+1)` de l'algorithme. De façon plus formelle, écrire `Pz = r` dans ce contexte revient à assigner à la variable `z` la solution du système linéaire $Px = r$. C'est à ce stade qu'intervient la décomposition (7) :

$$Px = r \Leftrightarrow TT^T x = r \Leftrightarrow \begin{cases} Ty & = r \\ T^T x & = y \end{cases}$$

Le système $Ty = r$ se résout avec l'algorithme de descente et le système $T^T x = y$ se résout avec l'algorithme de remontée. Après avoir implémenté cet algorithme en Python, analysons les résultats obtenus.

3.3.3 Résultats et influence du paramètre w du préconditionneur

Une première vérification que nous pouvons faire est de vérifier que cette méthode retourne bien le même résultat que celui retourné par la méthode de gradient conjugué classique (**Figure 2**) :

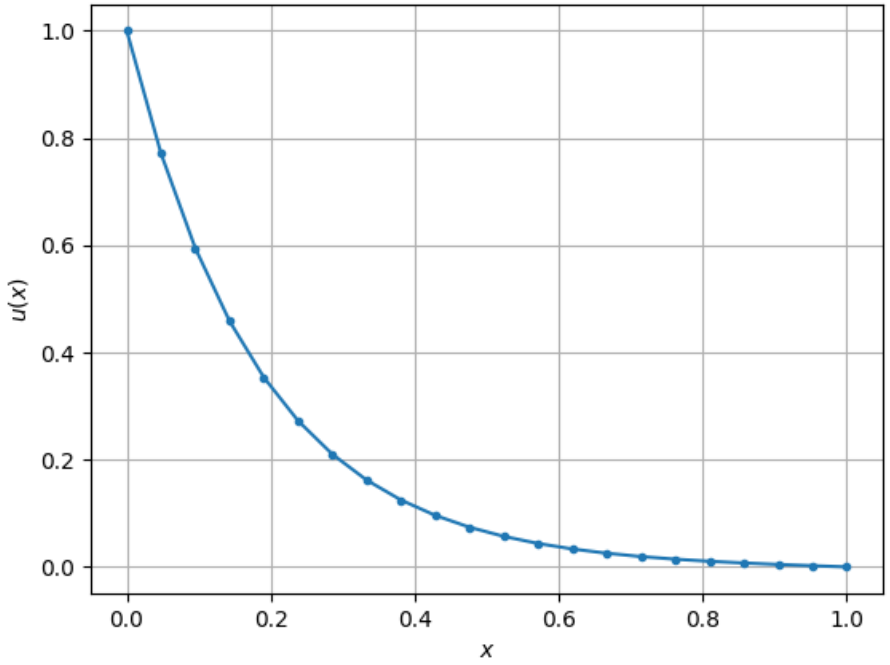

FIGURE 4 – Graphe de la solution u en fonction de x ($n = 20$, $k = 30$, $w = 1.0$)

Analysons maintenant la convergence vers la solution du système linéaire en fonction du paramètre w. Nous remarquons, d'après le graphique ci-après, que plus w est proche de 2, plus la méthode converge vite. Le nombre d'itérations diminue lorsque w augmente (tout en étant strictement plus petit que 2).

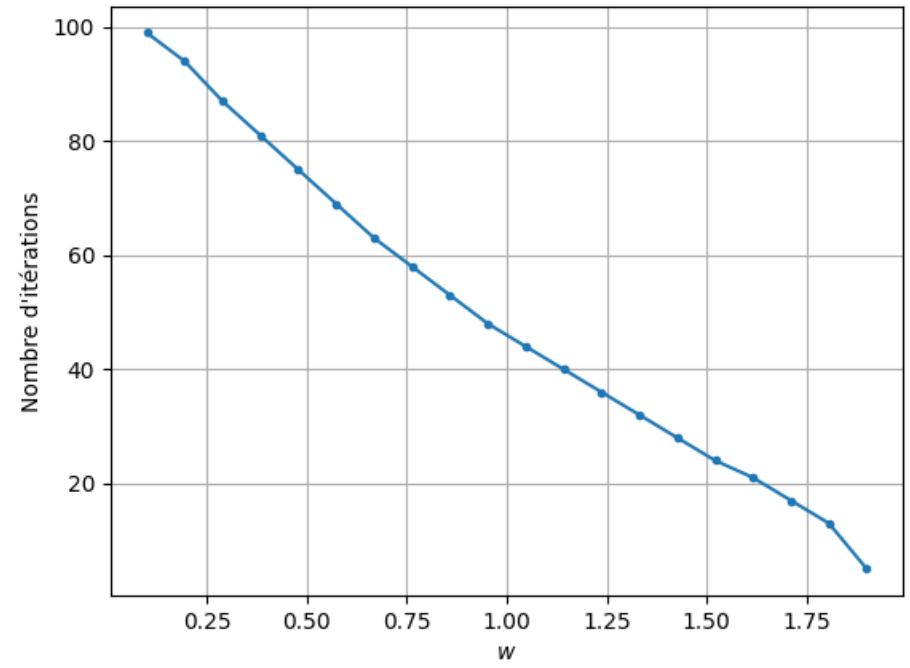

FIGURE 5 – Nombre d'itérations nécessaires à la résolution du système en fonction de w

Il est pertinent d'afficher, pour certaines valeurs w telles que $0 < w < 2$, la quantité $\|Ax_k - b\|$ en fonction de l'étape k de l'algorithme :

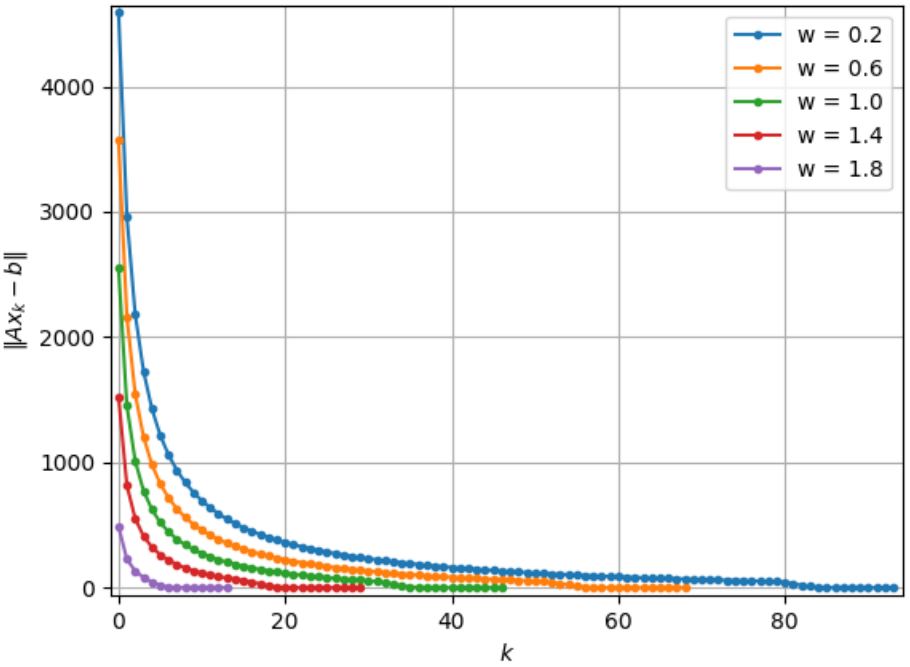

FIGURE 6 – Évolution de la norme du résidu en fonction du paramètre w

Comparons par exemple la courbe bleue et la courbe violette : on voit que l'algorithme est beaucoup plus efficace pour $w = 1.8$ que pour $w = 0.2$. Il trouve la solution en moins de 20 itérations

pour $w = 1.8$ contre plus de 80 pour $w = 0.2$. On remarque également que le résidu est beaucoup plus proche de 0 pour $w = 1.8$, ce qui signifie que dès le début de l'algorithme, on a une bonne approximation de la solution du système linéaire (4).

Enfin, pour mieux comprendre le comportement de l'algorithme lorsque le paramètre du système k est strictement négatif, on peut afficher la quantité $\|Ax_k - b\|$ en fonction de l'étape k, en paramétrant bien le préconditionneur ($w = 1.8$ par exemple) :

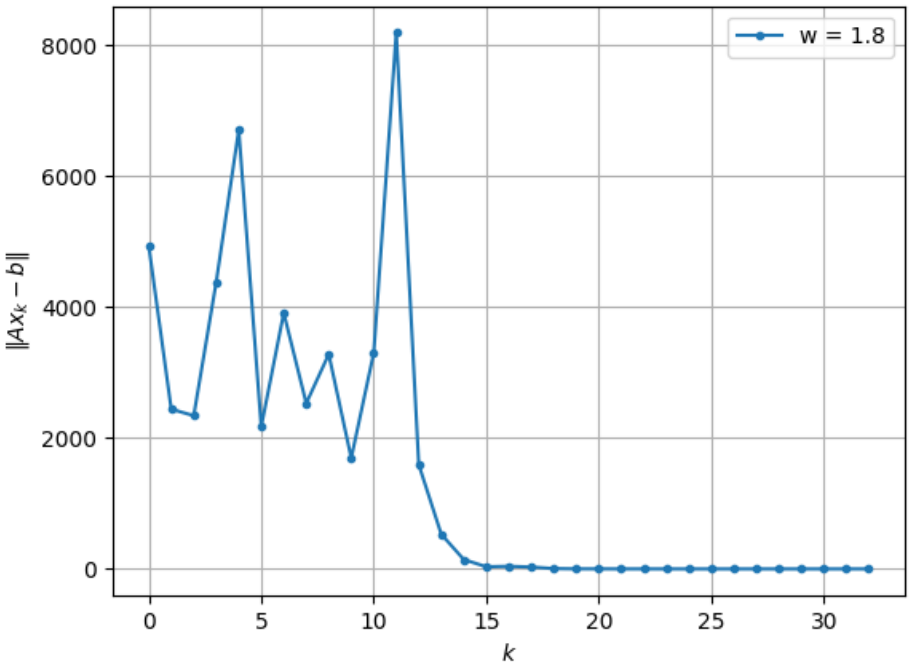

FIGURE 7 – Évolution de la norme du résidu au cours de l'algorithme lorsque le paramètre du système linéaire est fixé à une valeur strictement négative

Bien que l'algorithme converge, nous observons qu'au début de son éxécution, la norme du résidu subit des variations assez brutales. La convergence n'est pas monotone.

3.4 Comparaison des deux méthodes

Afin de comparer l'efficacité des deux méthodes, affichons l'évolution de la norme du résidu pour la méthode du gradient conjugué avec et sans préconditionnement SSOR. La figure ci-après montre que la résolution du système avec les paramètres indiqués est plus efficace lorsque la méthode est préconditionnée. L'algorithme préconditionné retourne la solution en moins de 50 itérations contre le double avec la méthode classique.

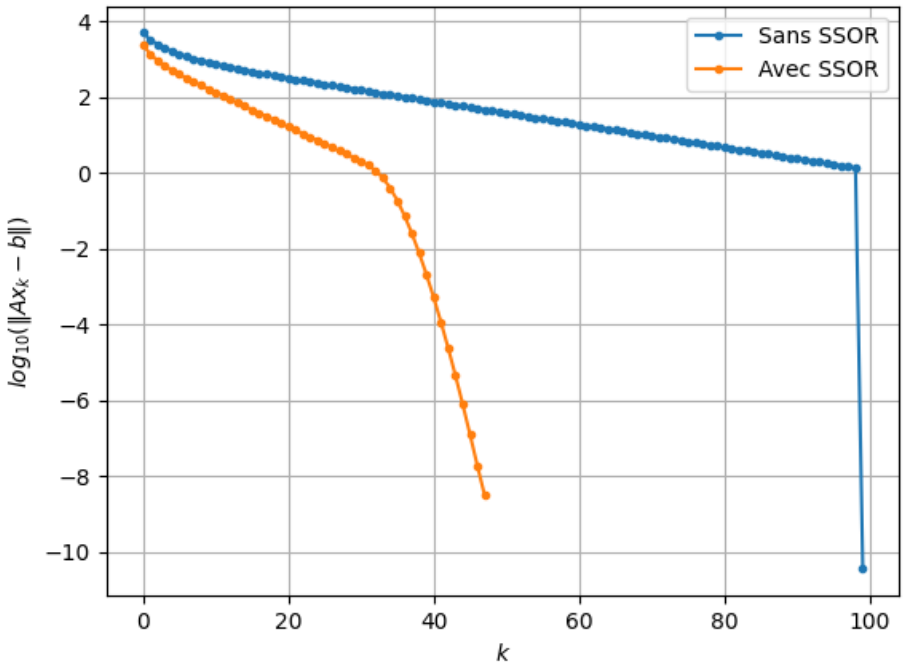

FIGURE 8 – Évolution de la norme du résidu (en log_{10}) au cours de l'algorithme pour les méthodes avec et sans préconditionnement ($n = 100$, $k = 49$, SSOR : $w = 1$)

Comme le montre la figure suivante, l'analyse est globalement la même pour un système dont le paramètre est strictement négatif même si la norme du résidu ne diminue pas de façon monotone au cours de l'algorithme.

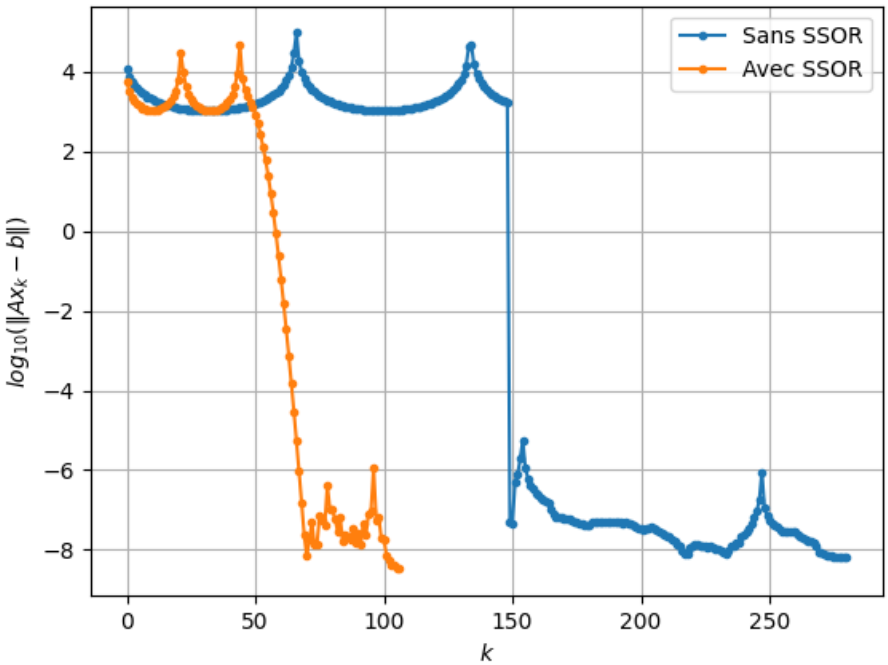

FIGURE 9 – Évolution de la norme du résidu (en log_{10}) au cours de l'algorithme pour les méthodes avec et sans préconditionnement ($n = 150$, $k = -49$, SSOR : $w = 1$)

On montre aussi que les deux algorithmes sont en $O(kmax \times n^2)$, où `kmax` correspond au nombre maximal d'itérations à faire. Cependant, il y a deux points à prendre en compte :

- Réduction du conditionnement : comme souligné dans partie **3.3.1**, le préconditionnement SSOR permet de « simplifier » la résolution du problème puisqu'il conduit à la résolution d'un système linéaire mieux conditionné.
- Stabilité numérique : en réduisant le conditionnement, le préconditionnement SSOR peut améliorer la stabilité numérique de la méthode itérative. Une matrice mieux conditionnée est moins sujette à des problèmes numériques, tels que la perte de chiffres significatifs, qui peuvent ralentir la convergence.

4 Démarche pour l'implémentation en Python

Nous avons structuré notre code en trois fichiers :

- `main.py` : ce module contient l'ensemble des appels des fonctions permettant d'afficher les graphiques présents dans ce rapport.
- `functions.py` : ce module contient l'ensemble des fonctions permettant d'implémenter la méthode du gradient conjugué. Elle contient également ses deux versions (avec et sans préconditionneur SSOR) ainsi qu'une fonction permettant d'afficher sur un graphique une solution.
- `analysis.py` : ce module contient l'ensemble des fonctions permettant de comparer les méthodes, d'étudier le comportement des solutions en fonction de certains paramètres (k, n) et d'étudier la convergence des algorithmes.

Pour plus de clarté, chaque fonction possède une documentation et une signature typée. En d'autres termes, les types des variables de retour et des arguments sont précisés.

5 Conclusion

En conclusion, ce projet nous a permis de découvrir une nouvelle méthode itérative : la méthode du gradient conjugué. Au regard des résultats obtenus, nous pouvons affirmer que la méthode du gradient conjugué avec préconditionnement (SSOR) est très pertinente pour la résolution du problème puisque d'une part, elle est adaptée au caractère symétrique définie positive de la matrice du système, et d'autre part, elle converge bien plus rapidement que la méthode classique.

La méthode du gradient conjugué est adaptée pour la résolution de l'équation de Helmotz, s'écrivant $(\Delta + w^2)u = 0$, qui permet de décrire de nombreux phénomènes ondulatoires. Elle est utilisée en météorologie pour décrire les variations de pression atmosphérique (en raison du relief par exemple). On parle d'ondes de gravité météorologiques ou encore d'ondes orographiques. L'équation de Scorer, qui est une équation de Helmotz, permet de décrire leur comportement en aval d'un relief montagneux. Cette dernière présente alors un intérêt dans le domaine de l'aviation.