Université de Versailles
Saint-Quentin-en-Yvelines

Rapport de projet

Calcul du Pagerank par Gauss-Seidel descendant

Alexandre Mihet 22005024
Billal Medour 21924103

Encadrants

Franck Quessette
Sandrine Vial

Date de soumission : 23 mai 2024

Préambule

Ce rapport présente le travail réalisé dans le cadre d'un projet de calcul du Pagerank a l'aide de l'algorithme de Gauss-Seidel descendat. Le projet a été mené sous la supervision de Franck Quessette et Sandrine Vial, et s'inscrit dans le contexte des Masters Secrets et AMIS .

Le PageRank, développé par les fondateurs de Google, Larry Page et Sergey Brin, est une méthode cruciale pour évaluer l'importance des pages web en fonction de leur structure de liens. Modélisant Internet sous forme de graphe orienté, le PageRank calcule l'importance de chaque page à travers la résolution d'un système d'équations linéaires. Deux méthodes candidates pour cette résolution sont l'algorithme des puissances et la méthode de Gauss-Seidel descendant.

Ce projet vise à comparer ces deux approches pour évaluer leur efficacité respective en termes de vitesse de convergence, de complexité computationnelle et de leur temps d'exécution. L'algorithme des puissances est reconnu pour sa simplicité et son efficacité dans l'obtention de l'autovecteur principal d'une matrice, tandis que la méthode de Gauss-Seidel descendant utilise des solutions successives pour améliorer la convergence.

Table des matières

Chapitre 1

Introduction

Le calcul du PageRank est une méthode essentielle pour évaluer l'importance des nœuds dans un graphe, utilisée notamment par les moteurs de recherche pour classer les pages web. Dans ce rapport, nous nous concentrons sur deux algorithmes pour calculer le PageRank : la méthode des puissances et la méthode de Gauss-Seidel descendant. Chaque méthode présente des avantages et des inconvénients en termes de performance et d'utilisation de la mémoire, particulièrement lorsqu'il s'agit de traiter de grandes matrices creuses.

1.1 Méthode des Puissances

La méthode des puissances est un algorithme itératif simple utilisé pour trouver les valeurs propres dominantes et les vecteurs propres d'une matrice. Dans le contexte du PageRank, elle est utilisée pour calculer les rangs des sommets dans un graphe en itérant jusqu'à ce que le vecteur de PageRank converge.

Principe de base : L'algorithme commence par un vecteur de départ aléatoire et multiplie ce vecteur par la matrice de transition du graphe. Cette opération est répétée jusqu'à ce que le vecteur converge, c'est-à-dire que les valeurs du vecteur ne changent plus significativement d'une itération à l'autre.

1.2 Méthode de Gauss-Seidel Descendant

La méthode de Gauss-Seidel est une technique itérative utilisée pour résoudre des systèmes linéaires. Dans le contexte du PageRank, l'algorithme Gauss-Seidel descendant optimise la convergence en utilisant les valeurs calculées les plus récentes à chaque itération.

Principe de base : L'algorithme met à jour les valeurs du vecteur PageRank de manière descendante, en utilisant les nouvelles valeurs dès qu'elles sont calculées pour améliorer la vitesse de convergence. À chaque itération, les éléments du vecteur sont mis à jour en tenant compte des éléments précédemment calculés à la même itération.

Chapitre 2

Conception du projet

2.1 Technologies Utilisées

Au long de ce projet, nous avons utilisé **Python** comme langage de programmation du fait de sa simplicité d'usage et par ce que c'est le langage que nous maîtrisons le mieux. Nous verrons plus tard qu'il ne s'agissait peut-être pas du choix le plus judicieux.
Nous avons également utilisé plusieurs bibliothèques que nous allons maintenant expliciter :

— **numpy** : NumPy est une bibliothèque de calcul en Python. Elle permet la prise en charge et la manipulation des tableaux et des matrices ce qui nous est utile pour gérer les opérations sur les vecteurs et les matrices. Par exemple, **np.random.rand** génère des vecteurs aléatoires, et **np.linalg.norm** calcule la norme de vecteurs.

— **time** : Time a été utilisé pour mesurer les durées d'exécution des différentes parties de notre code, notamment pour évaluer les performances des algorithmes.

— **matplotlib** : Nous utilisons **matplotlib.pyplot** pour tracer les graphes des performances de nos algorithmes en fonction de différents paramètres (par exemple, le facteur de damping alpha).

— **scipy** : Le module sparse de SciPy permet de travailler avec des matrices creuses. Nous utilisons **csr_matrix** pour stocker et manipuler les grandes matrices creuses de notre projet de manière efficace.

— **tracemalloc** : Nous utilisons tracemalloc pour mesurer la consommation de mémoire de nos algorithmes pour évaluer l'efficacité en termes de mémoire.

— **tqdm** : Nous utilisons tqdm pour afficher des barres de progression pendant les itérations des algorithmes. Cela améliore l'expérience utilisateur en fournissant une indication visuelle sur l'avancement des calculs

2.2 Difficultés rencontrés

Nous avons quand même fait face a des problèmes lors de la conception de notre projet. Une des plus grandes difficultés était de manipuler des tres grands fichiers et sets de données avec Python. Bien que ce soit un langage facile à comprendre, il est vraiment

vraiment lent pour faire des traitements quand les données deviennent grandes, même avec un maximum d'optimisations. Il aurait été plus judicieux d'utiliser un autre langage de programmation comme par exemple le **C** qui est réputé pour sa vitesse et sa manipulation de mémoire libre. Le problème c'est qu'on ne connaît pas bien ce langage et qu'une mise en oeuvre du projet aurait été compliqué. Peut-être un autre langage comme le Java aurait été mieux dans notre cas.

1. Gestion des Grandes Matrices Creuses

Une des principaux défis a été la gestion des grandes matrices creuses. Pour résoudre ce problème, nous avons utilisé la structure de données `csr_matrix` de la bibliothèque `scipy.sparse`, qui permet de stocker efficacement les matrices creuses en ne conservant que les éléments non nuls.

2. Optimisation des Algorithmes

L'optimisation des algorithmes de PageRank, en particulier l'algorithme de Gauss-Seidel Descendant, a également posé un défi enorme. Cet algorithme nécessite une mise à jour descendante des solutions, ce qui implique de mélanger les solutions calculées à l'itération $k+1$ avec celles de l'itération k. Assurer la convergence rapide tout en maintenant des performances élevées et un résultat correct a nécessité plusieurs heures de peaufinage et d'optimisations.

Chapitre 3

Résultats obtenus

Comme décrit succinctement dans l'introduction, nous allons désormais faire la comparaison entre la méthode des puissances et la méthode de Gauss-Seidel descendant pour le calcul de PageRank.
Pour cela nous allons comparer **3 métriques différentes** sur plusieurs graphes de tailles différentes en faisant varier la valeur de alpha :

— **Nombre d'itérations avant convergence** : Le nombre total de fois que l'algorithme doit mettre à jour le vecteur PageRank avant de converger.

— **Durée d'exécution avant convergence** : Le temps total nécessaire pour qu'un algorithme converge vers une solution, c'est-à-dire le moment où les valeurs du vecteur PageRank ne changent plus de manière significative d'une itération à l'autre (elles passent sous un seuil donné en entrée de l'algorithme).

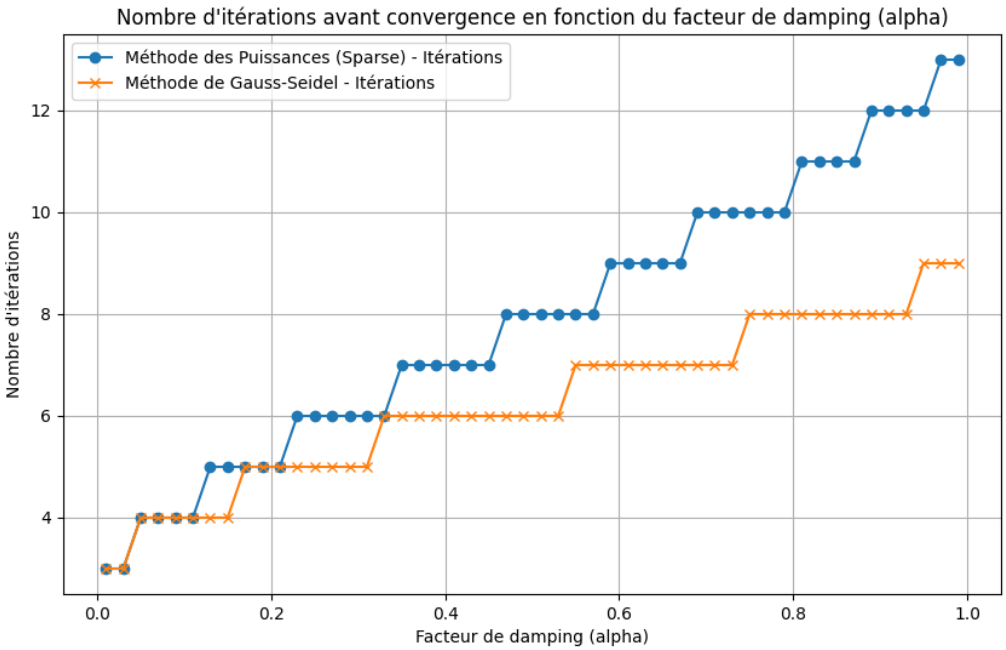
— **Pic de consommation mémoire** : La quantité de mémoire (RAM) utilisée par l'algorithme pendant son exécution.

3.1 Présentation des résultats

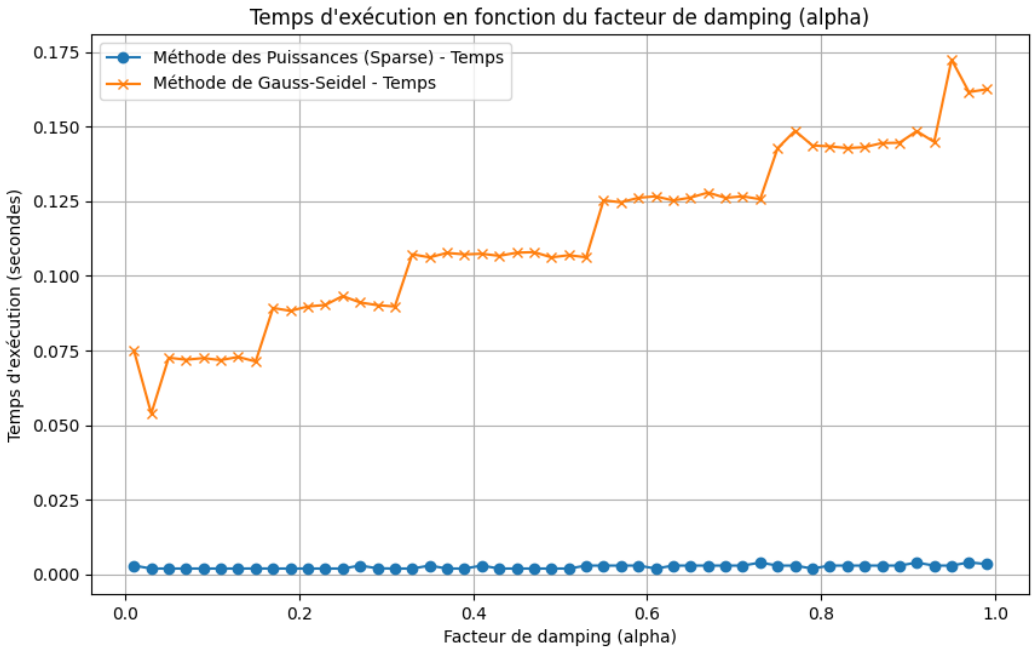
Pour chacun de ces graphes, nous allons donc analyser les 3 métriques cités précédemment en faisant varier alpha. Pour chaque graphe, nous allons faire **50** simulations pour avoir des courbes avec un bon nombre de points.

3.1.1 G1000.txt

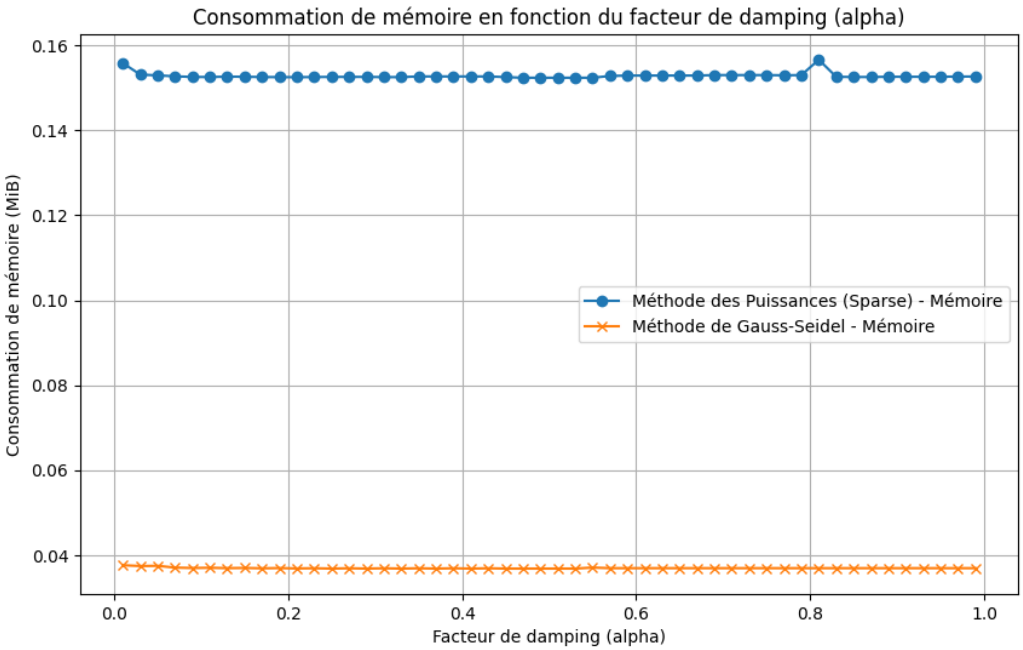
Nombre de sommets : 1000
Nombre d'arêtes : 9958



— **Méthode des Puissances** :
  — Le nombre d'itérations augmente progressivement avec l'augmentation de α.
  — La méthode des puissances nécessite plus d'itérations pour des valeurs de α plus élevées, atteignant 13 itérations pour α proche de 1.
— **Méthode de Gauss-Seidel** :
  — Cette méthode nécessite moins d'itérations que la Méthode des Puissances pour toutes les valeurs de α.
  — L'augmentation du nombre d'itérations est également progressive, mais reste inférieure à 10 itérations même pour α proche de 1.



— **Méthode des Puissances** :
  — Le temps d'exécution reste relativement stable et bas, même avec l'augmentation de α.
  — Les variations sont faibles, et bonne stabilité en termes de temps d'exécution.
— **Méthode de Gauss-Seidel** :
  — Le temps d'exécution est plus élevé que celui de la Méthode des Puissances.
  — Il y a des fluctuations notables pour les petites valeurs de α, avec des pics de temps d'exécution.

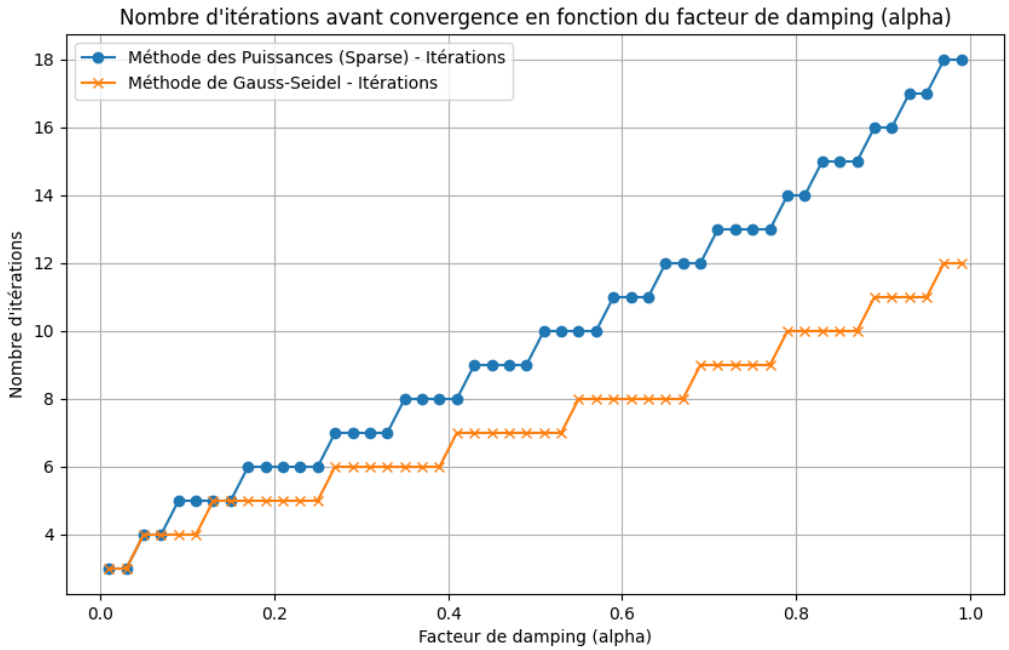


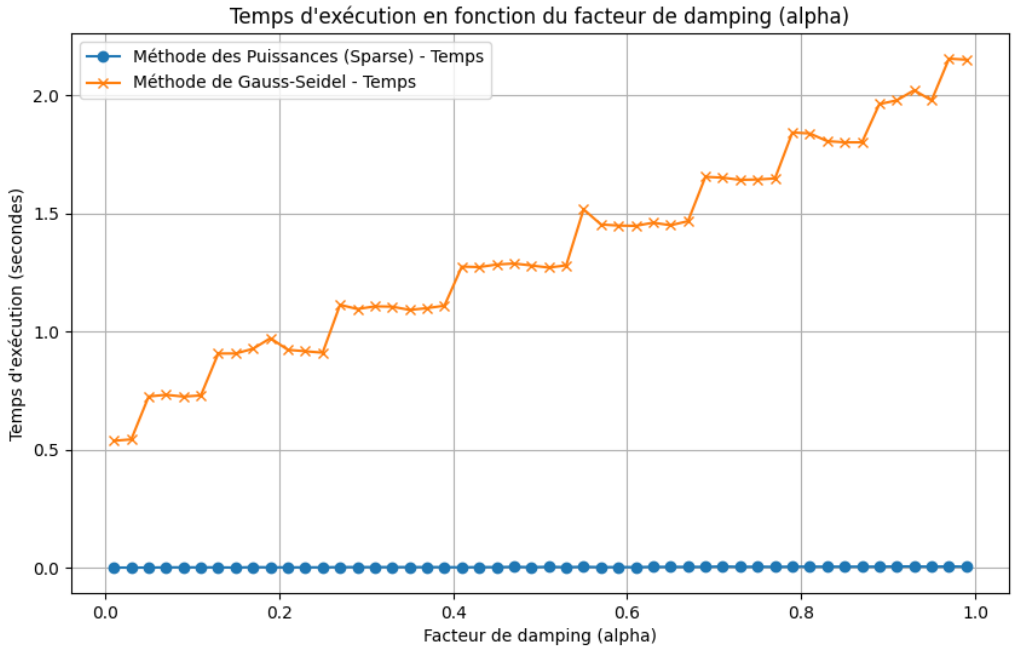
— **Méthode des Puissances** :
  — La consommation est plus élevée que celle de la Méthode de Gauss-Seidel.
— **Méthode de Gauss-Seidel** :
  — La consommation de mémoire est plus faible et plus stable par rapport à la Méthode des Puissances.

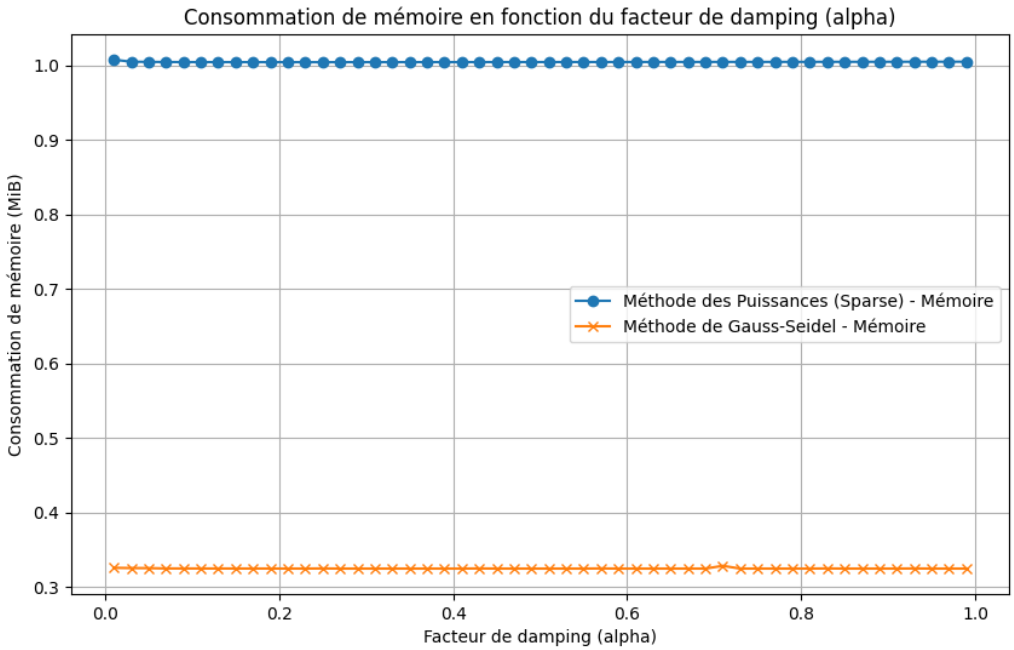
Temps total d'exécution du programme : 5.8726 secondes.

3.1.2 G10001.txt

Nombre de sommets : 10001
Nombre d'arêtes : 59995


Nombre d'itérations avant convergence en fonction du facteur de damping (alpha)
Méthode des Puissances (Sparse) - Itérations
Méthode de Gauss-Seidel - Itérations
Nombre d'itérations
Facteur de damping (alpha)


Temps d'exécution en fonction du facteur de damping (alpha)
Méthode des Puissances (Sparse) - Temps
Méthode de Gauss-Seidel - Temps
Temps d'exécution (secondes)
Facteur de damping (alpha)



Nous pouvons remarquer que les courbes ont une même allure que pour le premier graphe mais avec une autre échelle : La ou le premier graphe allait jusqu'à 13 itérations avant convergence pour la méthode des puissances, ici plus on s'approche de $\alpha = 1$, plus la valeur augmente jusqu'à arriver a 18.
Idem pour le temps en secondes : nous pouvons remarquer que la méthode de Gauss-Seidel Descendante est bien bien plus lente que celle des puissances malgré qu'elle fasse moins d'itérations avant de converger.
Enfin, concernant la consommation en mémoire, nous pouvons voir que la méthode de Gauss-Seidel descendante est un plus optimisée. La consommation en mémoire reste stable

Explications

— **Méthode de Gauss-Seidel** : bénéficie de mises à jour séquentielles, ce qui permet une convergence plus rapide en termes d'itérations.

Temps d'exécution

— **Méthode des Puissances** : est bien optimisée pour les matrices creuses, ce qui maintient un temps d'exécution bas et stable.

— **Méthode de Gauss-Seidel** : a un temps d'exécution plus élevé et variable, probablement en raison des dépendances séquentielles et des variations dans la convergence.

Consommation de mémoire

— **Méthode des Puissances** : utilise plus de mémoire, probablement en raison du stockage de multiples vecteurs intermédiaires.

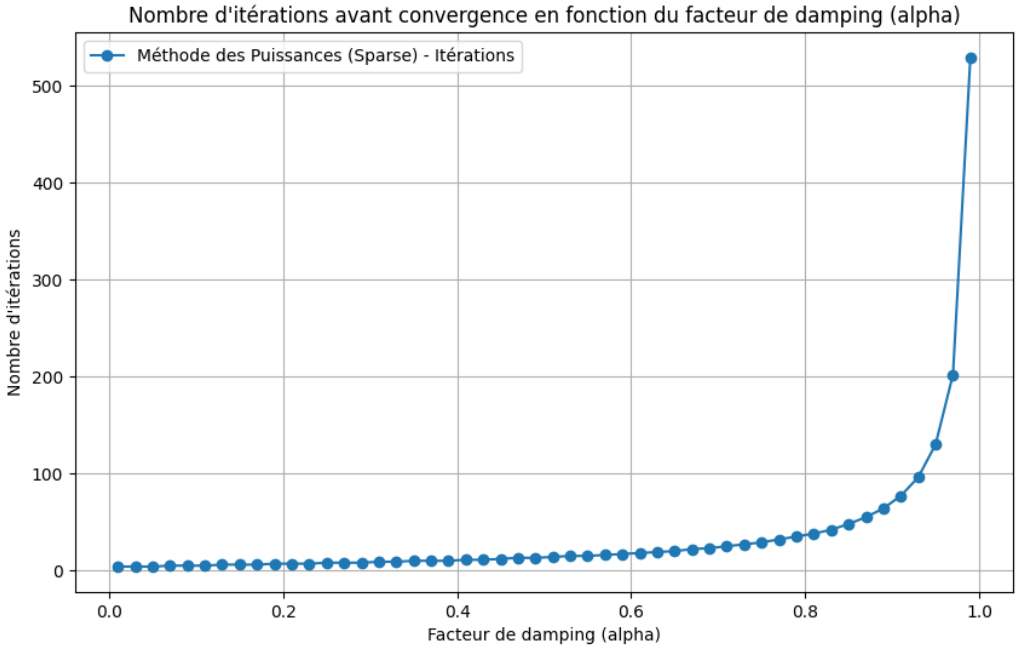
— **Méthode de Gauss-Seidel** : est plus économique en mémoire, grâce à des mises à jour in-place.

Temps total d'exécution du programme : 67.3543 secondes.

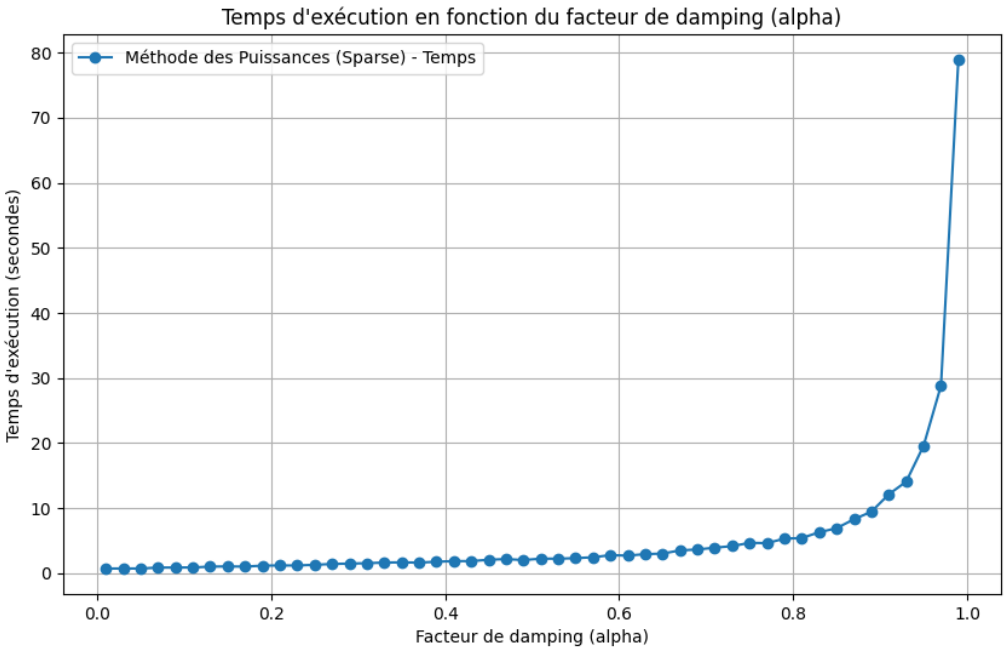
3.1.3 wikipedia.txt

Nombre de sommets : 1634989
Nombre d'arêtes : 19753078

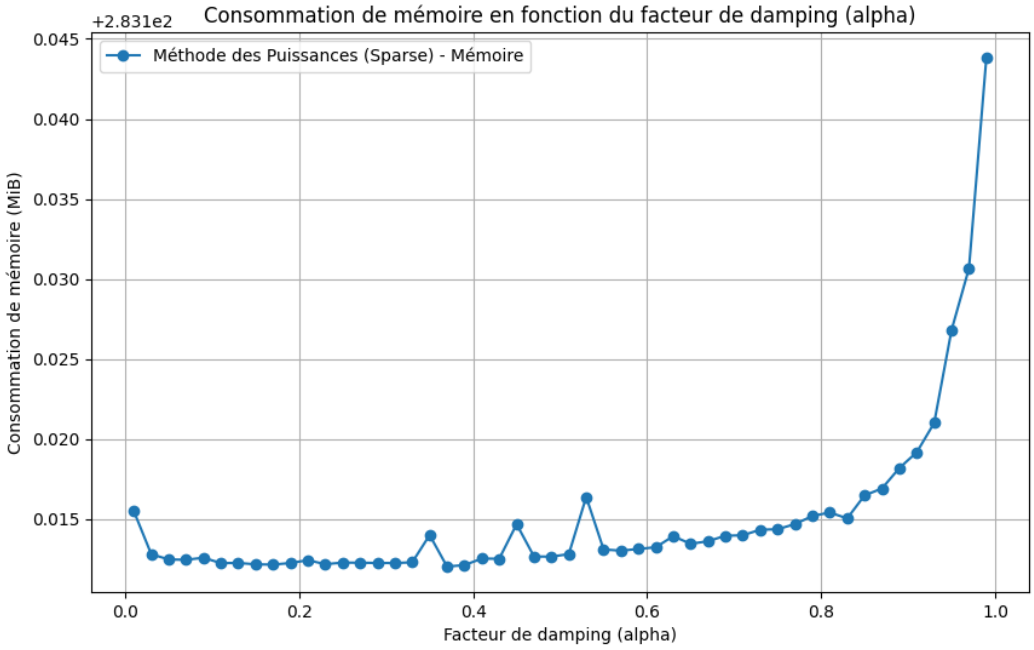
La méthode de Gauss-Seidel Descendate prend beaucoup trop de temps pour pouvoir faire des dizaines de simulations et tracer des graphiques, c'est pourquoi nous allons étudier uniquement la méthode des puissances sur un graphe énorme.



Nous pouvoir remarquer que nombre d'itérations reste bas et stable pour les valeurs de α inférieures à 0.6, puis augmente rapidement pour les valeurs de supérieures à 0.8. À mesure que α se rapproche de 1, le nombre d'itérations augmente de manière exponentielle, atteignant plus de 500 itérations pour un α proche de 1.



Le temps d'exécution suit une tendance similaire au nombre d'itérations, restant bas et stable pour les valeurs de α inférieures à 0.6, puis augmentant rapidement pour les valeurs de α plus élevées. Pour des valeurs de α proches de 1, le temps d'exécution atteint jusqu'à 80 secondes, ce qui reflète directement l'augmentation du nombre d'itérations.



Même constat que pour les autres graphiques mais pas de façon aussi énorme.

Explications :

1. **Nombre d'itérations :**
   — Lorsque α est faible, les effets du damping réduisent les contributions des composants moins dominants, permettant une convergence rapide.
   — À des valeurs élevées de α, les contributions des composants moins dominants deviennent significatives, ralentissant la convergence et augmentant considérablement le nombre d'itérations nécessaires.
2. **Temps d'exécution :**
   — Le temps d'exécution est fortement corrélé au nombre d'itérations. Plus le nombre d'itérations est élevé, plus le temps d'exécution augmente.
   — Le temps d'exécution montre une augmentation exponentielle similaire au nombre d'itérations à mesure que α se rapproche de 1.
3. **Consommation de mémoire :**
   — Pour des valeurs élevées de α la consommation de mémoire augmente, mais de manière moins dramatique que celle du nombre d'itérations et du temps d'exécution. Cela pourrait être dû à la nécessité de stocker plus de vecteurs intermédiaires ou des informations supplémentaires pour les itérations supplémentaires.

Conclusions :
— La Méthode des Puissances performe bien pour des valeurs de α inférieures à 0.6, avec un faible nombre d'itérations, un temps d'exécution rapide et une consommation de mémoire stable.
— Pour des valeurs de α proches de 1, la méthode devient inefficace en termes de nombre d'itérations et de temps d'exécution.
Temps total d'exécution du programme : 304.5026 secondes.

3.2 Vérification des résultats

Au long de ce projet, il a été important pour nous de bien vérifier la cohérence des résultats obtenus pour s'assurer que les algorithmes fonctionnent bien. C'est bien d'avoir un algorithme rapide mais si il ne calcule pas la bonne chose ou que le résultat est faux et incohérent alors on perd tout l'intérêt de la comparaison.

Pour chacun des graphes, nous avons fait plusieurs vérifications dans le terminal :

1. **Somme des éléments du vecteur PageRank est égal a 1** : Plusieurs fois, probablement a cause de problèmes de normalisation a chaque itération, la somme des éléments du vecteur PageRank a été différente de 1. C'est pourquoi nous avons ajouté une vérification qui fonctionne a merveille et qui détecte les erreurs a chaque itération ET verifie egalement la cohérence du vecteur final.
2. **Vérification du ranking final** : L'idée c'est d'utiliser deux méthodes différentes qui ont le même objectif final, de faire de ranking des sommets a partir d'une matrice. Il est donc important a la fin de vérifier le vecteur final et de comparer si il est semblable (a quelques variations mineures) entre les deux algorithmes. Dans notre code on vérifie a la fin de l'exécution le top 5 des sommets avec le meilleur ranking avec leur ranking calculé associé avec les deux methodes.

Chapitre 4

Conclusion

4.1 Bilan

Dans ce projet, nous avons exploré et comparé deux algorithmes principaux pour le calcul du PageRank : la méthode des puissances et la méthode de Gauss-Seidel descendant. Nous avons mis en place une série de vérifications pour garantir la validité et la cohérence des résultats obtenus par ces algorithmes. Voici les principales conclusions que nous pouvons tirer de ce travail :

— **Vitesse de Convergence** : La méthode des puissances est simple à implémenter et efficace pour les matrices creuses, mais elle peut nécessiter un grand nombre d'itérations pour converger. La méthode de Gauss-Seidel descendant, bien que plus complexe, tend à converger en moins d'itérations en utilisant les valeurs mises à jour dès qu'elles sont calculées.

— **Vitesse d'exécution** : La méthode des puissances est cependant plus rapide dans toutes les exécutions que celle de Gauss-Seidel. Cela est peut-être du a notre implémentation de Gauss-Seidel qui n'est pas la plus optimisée mais nous nous sommes renseignés et si on suit la logique de la l'article [1], ce n'est pas si anormal que ça d'obtenir une vitesse d'éxécution plus faible pour l'algorithme de Gauss-Seidel.

— **Utilisation de la Mémoire** : Les deux algorithmes peuvent être optimisés pour utiliser des meilleures structures de données, mais la méthode des puissances consomme a chaque itération plus de mémoire que la méthode de Gauss-Seidel.

En conclusion, ce projet a permis de mieux comprendre les forces et les faiblesses des algorithmes de PageRank, et de mettre en évidence les possibilités d'optimisation pour traiter efficacement des graphes de très grande taille.
De plus cela nous a permis de nous rendre compte que le python n'est peut-être pas le meilleur langage de programmation quand il s'agit de traiter des grandes données et d'optimiser du code.

4.2 Idées d'amélioration

Il y aurait eu pleins de choses que nous aurions pu faire en plus de ce qui a été fait comme par exemple :

— **Optimiser supplémentaire de vitesse d'exécution** : Nous aurions pu, dans la limite de ce que Python nous permet, encore d'optimiser plus la vitesse d'exécution. Nous pouvons par exemple utiliser Multiprocessing pour paralléliser certains calculs. Il existe une librairie (Threading) que nous aurions pu utiliser mais nous n'avons pas eu le temps d'en faire une implémentation satisfaisante.

— **Trouver d'autres métriques a mesurer** : Il aurait également été intéressant de pouvoir comparer d'autres variables a faire varier afin de produire d'autres graphes intéressants.

Bibliographie

[1] Saint-Jean A. O. Djungu, Pierre Manneback, Fabien Mathieu. *Étude comparative des méthodes de calcul de PageRank*. Faculté Polytechnique de Mons, Belgique.
PDF : `https://www.cari-info.org/actes2006/150.pdf`