Indexation du Web : l'agorithme de PageRank

Danil Garmaev, Lisa Ceresola

Table des matières

1 Introduction

La recherche d'informations pertinentes sur le Web est un des problèmes les plus cruciaux pour l'utilisation de ce dernier. Des enjeux économiques colossaux sont en jeu, et diverses multinationales se livrent à de grandes manœuvres. Le leader actuel de ce marché, Google, utilise pour déterminer la pertinence des références fournies, un certain nombre d'algorithmes dont certains sont des secrets industriels jalousement gardés, mais d'autres sont publics. On va s'intéresser ici à l'algorithme PageRank, lequel fait intervenir des valeurs propres et vecteurs propres d'une énorme matrice.

2 La matrice de Google

L'algorithme de PageRank de Google, développé en 1998, classe ces pages en leur attribuant des poids afin d'en mesurer l'importance. La plupart de ces pages incluent des liens vers d'autres pages. L'idée utilisée par les moteurs de recherche pour attribuer à chaque page un poids suivant leur importance, consiste à considérer que plus une page est la cible d'autres pages, plus elle est importante. Il s'agit donc d'attribuer à chaque page un rang numérique. Plus une page sera importante, plus son rang sera grand.

2.1 Construction

On se donne un nombre arbitraire sur l'ensemble des pages que l'on numérote de $i = 1$ à $i = N$. La matrice représentant la structure de connectivité du Web sera notée $L = (l_{ij})_{1,\leq i,j\leq N}$ où :

$$l_{ij} = \begin{cases} 1 \text{ si la page } j \text{ pointe sur la page } i \\ 0 \text{ sinon} \end{cases} \tag{1}$$

On souhaite ainsi attribuer un score $r_j \in \mathbb{R}_+^*$ afin de pouvoir classer l'ensemble de nos pages par ordre décroissant et présenter à l'utilisateur une liste classée des pages répondant à sa requête.
On part du principe que plus il y a de pages qui pointent vers une page i, plus son score sera élevé. On introduit une nouvelle matrice $Q = (q_{ij})_{1,\leq i,j\leq N}$ définie par :

$$q_{ij} = \begin{cases} 1/N_j \text{ si } N_j \neq 0 \\ 0 \text{ sinon} \end{cases} \tag{2}$$

où N_j est le nombre total de liens présents sur la page j. L'application des principes ci-dessus conduit donc à une équation pour le vecteur $r\in \mathbb{R}^N$ des scores des pages :

$$r_i = \sum_{j=1}^{N} q_{ij} r_j \text{ c'est-à-dire } r = Qr \tag{3}$$

Ainsi, le classement des pages du Web se ramène à la recherche d'un vecteur propre associé à la valeur propre 1. Cependant, il peut arriver que notre matrice Q n'admette pas la valeur propre 1. Pour y remédier, on considère alors un vecteur $e = {}^t(1,\ldots,1) \in \mathbb{R}^N$ et d $\in \mathbb{R}^N$ tel que $d_j = 1$ si $N_j = 0$, $d_j = 0$ sinon. On pose ainsi la matrice :

$$P = Q + \frac{1}{N} e\ {}^t d \tag{4}$$

qui est la transposée d'une matrice stochastique. Comme ${}^t eP = {}^t e$, P admet bien 1 comme valeur propre. Cette valeur est en général multiple, or, nous cherchons un unique vecteur. On introduit alors la matrice de Google, avec $0 < \alpha < 1$ dit "facteur d attenuation" :

$$A = \alpha P + (1-\alpha)\frac{1}{N} e\ {}^t e \tag{5}$$

La première partie de l'équation : αP représente la probabilité qu'un utilisateur suive un lien depuis une page pour aller vers d'autres pages. La deuxième partie de l'équation $(1-\alpha)\frac{1}{N} e\ {}^t e$ représente la probabilité qu'un utilisateur arrive sur une page aléatoirement. Ce facteur d'atténuation α joue un rôle crucial dans l'algorithme de PageRank car il modélise le fait qu'un utilisateur lorsqu'il parcourt les pages du web ne va pas forcément suivre les liens sur une page. Il peut décider de surfer aléatoirement vers n'importe quelle page du web.

2.2 Théorème de Perron-Frobenius

On peut admettre que le rayon spectral de notre matrice A est égal à 1. Cela vient du fait que l'algorithme de PageRank a été conçu pour que la probabilité totale de se trouver sur n'importe quelle page du web doit être égale à 1. Montrons alors que 1 est valeur propre dominante et simple de notre matrice A.

Définition 1. *On note $A > 0$ une matrice dont tous les coefficients sont strictements positifs.*

Lemme 1. *Pour une matrice $A = (a_{ij})_{1 \leq i,j \leq n} \in M_n(\mathbb{R})$, on a :*
$A > 0$ si et seulement si $Ax > 0$ pour tout $x \in \mathbb{R}\{0\}$ tel que $x \geq 0$.

Définition 2. *Soit $\mathbb{K} = \mathbb{R}$ ou $\mathbb{C}$ et $A \in M_n(\mathbb{K})$, on note :*
(i) $\sigma(A)$ le spectre de A ;
(ii) $r(A) = max_{\lambda \in \sigma(A)} |\lambda|$ le rayon spectral de A ;
(iii) Une valeur propre λ est dite dominante si, $\forall \alpha \in \sigma(A)$ λ on a $|\lambda| > |\alpha|$.

Définition 3. *Soit $A \in M_n((R))$, $A \geq 0$.*
A est dite primitive s'il existe $k \in \mathbb{N}$ tel que $A^k > 0$.

Théorème 1 (Perron-Frobenius)**.** *Soit $A = (a_{ij})_{1 \leq i,j \leq n} \in M_n(\mathbb{R})$ une matrice positive et $r = r(A)$ son rayon spectral. Supposons que A est primitive. Alors $r > 0$, r est une valeur propre dominante et simple de A et il existe un unique vecteur x^+ strictement positif tel que $Ax^+ = rx^+$ et $\|x^+\|_1 = 1$*

Démonstration. La démonstration se fait en plusieurs étapes, que nous allons détailler.

1ère étape : Montrons que l'on peut supposer que $A > 0$.
Comme A est primitive, alors $\exists k \in \mathbb{N}$, tel que $A^k > 0$. Supposons le théorème démontré pour $B = A^k$. Le spectre de B est $\sigma(B) = \left\{\lambda^k : \lambda \in \sigma(A)\right\}$. Notons qu'un vecteur propre de A pour la valeur propre λ est un vecteur propre de B pour la valeur propre λ^k. On note $s = r^k$ le rayon spectral de B.
Donc $r > 0$ et r est une valeur propre de A. Si pour $x, y \in \mathbb{C}^n$ on a $Ax = rx$ et $Ay = ry$ tel que $Bx = r^k x$ et $By = r^k y$ alors r est une valeur propre simple de A. r est une valeur propre dominante de A : si λ est une valeur propre de A de module r alors λ^k est une valeur propre de B de module $r^k = s$ ainsi $\lambda^k = s$. Et donc, $\lambda = r$. Il s'ensuit que l'unique vecteur propre $x^+ > 0$ de B tel que $\|x^+\|_1 = 1$ pour la valeur propre r^k est l'unique vecteur propre $x^+ > 0$ de A tel que $\|x^+\|_1 = 1$ pour la valeur propre r. Ainsi, $A > 0$.

On suppose alors pour la suite de la démontration que $A > 0$. On a ainsi par le Lemme 1 que $Ax > 0$ pour tout $x > 0$. On peut observer que cela implique que pour tout $x \in \mathbb{R}^n, x \geq 0$, si x est un vecteur propre de A ; alors $x > 0$.

Posons $X = \{x \in \mathbb{R}^n | x \geq 0, \|x\|_1 = 1\}$ un compact de $\mathbb{R}^n$. Pour tout $x \in X$, on considère l'ensemble $\{t \in \mathbb{R}^+ | tx \leq Ax\}$ non vide, bornée par $\|A\|$ et fermé.
On définit une fonction $\theta : X \to \mathbb{R}^+$ par :

$$\theta(x) = max\left\{t \in \mathbb{R}^+ | tx \leq Ax\right\}, x \in X$$

On observera que $\theta(x) > 0$ puisque $Ax > 0$ et que la fonction θ est bornée. Soit $\boxed{\mathrm{r_0} = \sup_{x \in X} \theta(x) \in]0, +\infty[}$.

2ème étape : Montrons par l'absurde que pour tout $x \in X$ tel que $\theta(x) = r_0$, x est un vecteur propre de A pour la valeur propre r_0.
Supposons $Ax \neq r_0 x$. Par définition de la fonction θ, on a $Ax - r_0 x = Ax - \theta(x)x > 0$ car $Ax \geq \theta(x)x$. Ainsi, $A(Ax - r_0 x) > 0$ puisque $A > 0$. On a $A(Ax - (r_0 + \epsilon)x) \geq 0$ pour $\epsilon > 0$ assez petit. Soit $y = \frac{Ax}{\|Ax\|_1} \in X$. Par définition de la fonction θ on a, $Ay \geq (r_0 + \epsilon)y$. Alors :

$$Ay - (r_0 + \epsilon)y = \frac{A}{\|Ax\|_1}(Ax - (r_0 + \epsilon)x) \geq 0$$

Comme $\theta(y) = \max\{t \in \mathbb{R} | ty \leq Ay\}$, on a $\theta(y) \geq r_0 + \epsilon$ ce qui contredit la définition de r_0.

3ème étape : Montrons qu'il existe un vecteur $x^+ \in X$ tel que $\theta(x^+) = r_0$.
Soit $(x_n)_{n\in\mathbb{N}} \in X^{\mathbb{N}}$ tel que $\lim\limits_{n\to+\infty} \theta(x_n) = r_0$. Comme X est compact, il existe une sous-suite convergente de $(x_n)_{n\in\mathbb{N}}$ notée $(x_{\phi(n)})_{n\in\mathbb{N}}$ avec $\phi : \mathbb{N} \to \mathbb{N}$ une application strictement croissante.
Soit $\lim\limits_{n\to+\infty} x_{\phi(n)} = x^+ \in X$. On a $\theta(x_{\phi(n)})x_{\phi(n)} \leq Ax_{\phi(n)}$. Par passage à la limite, on a $x^+ r_0 \leq Ax^+$ et donc $r_0 \leq \theta(x^+)$ car $x^+ \geq 0$. Or, $r_0 \geq \theta(x^+)$ par définition, ainsi $r_0 = \theta(x^+)$.
Donc x^+ est un vecteur propre de A de la valeur propre r_0 d'après l'étape 2.

4ème étape : Montrons que r_0 est égal au rayon spectral r de A.
Soit $v = {}^t(v_1 \dots v_n) \in \mathbb{C}^n$, vecteur propre de λ une valeur propre de A, avec $\|v\|_1 = 1$.
Alors, $\forall i \in \{1, \dots, n\}$ on a $\lambda v_i = \sum_{j=1}^N a_{ij} v_j$. Par inégalité triangulaire on a :

$$|\lambda||v_i| \leq \sum_{j=1}^N |a_{ij}||v_j| = \sum_{j=1}^N a_{ij}|v_j|.$$

On a donc $|\lambda||v| \leq A|v|$ avec $|v| = {}^t(|v_1|, \dots, |v_n|) \in X$. Ceci implique que $|\lambda| \leq \theta(|v|)$. Or par définition de r_0, on a $|\lambda| \leq r_0$.

5ème étape : Montrons que r est une valeur propre dominante et simple de A.
Soit $\lambda \in \mathbb{C}$ une valeur propre de A avec $|\lambda| = r$. Soit $v = {}^t(v_1, \dots, v_n) \in \mathbb{C}^n$ un vecteur propre pour λ avec $\|v\|_1 = 1$.

• r valeur propre dominante :
Il suffit de montrer que $\lambda = r$.
Comme plus haut, pour $|v| = {}^t(|v_1|, \dots, |v_n|) \in X$ on a $r = \theta(|v|)$. Or d'après nos conclusions précédentes, on a que $|v|$ est vecteur propre de la valeur propre r. Donc, $\forall i \in \{1, \dots, n\}, r|v_i| = \sum_{j=1}^N a_{ij}|v_j|$. D'autre part, comme $Av = \lambda v$, $|\sum_{j=1}^N a_{ij} v_j| = r|v_i|$; il s'ensuit que :

$$|\sum_{j=1}^N a_{ij} v_j| = \sum_{j=1}^N a_{ij}|v_j|, 1 \leq i \leq n.$$

Comme $a_{ij} > 0$, on peut écrire $v = e^{i\theta}|v|$ avec $\theta \in [0, 2\pi[$. On a donc $Av = e^{i\theta}A|v| = e^{i\theta}r|v|$. Par ailleurs, on a $Av = \lambda v = e^{i\theta}\lambda|v|$. Donc pas identification, on a $\lambda = r$.

• r valeur propre simple :
Montrons désormais que v et $x^+ = {}^t(x_1, \dots, x_n)$ sont colinéaires.
Comme $|v|$ est un vecteur propre de A alors $|v| > 0$. Ainsi, on peut supposer $v > 0$ et on a $x^+ > 0$. On définit une fonction :

$$\begin{array}{rccc} \Psi_i & : & \mathbb{R} & \to & \mathbb{R} \\ & & t & \mapsto & x_i + tv_i \end{array}$$

Si $t \geq 0, \Psi_i(t) \geq 0$.
On a : $x_i, v_i \geq 0$ donc $\exists! t_i$ tel que $\Psi_i(t_i) = 0$. Ainsi, $\forall j, \Psi_j(t_i) \geq 0, \Psi_i(t_i) = 0$. Comme $x^+ + t_i v$ est un vecteur propre de A, on a :

$$A(x^+ + t_i v) = r(x^+ + t_i v) = r(x_1 + t_i v_1, \dots, x_n + t_i v_n).$$

Or, comme $x^+ + t_i v \geq 0$ et $A > 0$, on devrait avoir $A(x^+ + t_i v) > 0$. On obtient alors une contradiction comme $x_i + t_i|v_i| = 0$ les composantes ne sont pas toutes > 0. Donc $x^+ + tv = 0$, ainsi, x^+ et v son colinéaires. □

Ainsi, comme $r(A) = 1$ et que A est positive, alors d'après le théorème de Perron-Frobenius, 1 est valeur propre simple et dominante de A. On peut donc choisir un vecteur propre correspondant aux composantes toutes postives.

3 La méthode de la puissance

Lors du calcul du vecteur propre, on vise à trouver celui associé à la plus grande valeur propre.

Dans le contexte de l'indexation web de Google, cette approche est cruciale. Dans l'algorithme PageRank, le vecteur propre lié à la plus grande valeur propre (égale à 1) est essentiel. Il représente la distribution stationnaire des probabilités pour chaque page web, identifiant ainsi les pages les plus influentes et centrales du réseau.

3.1 Algorithme

Étant donnée la taille de la matrice A, la seule méthode numérique envisageable pour calculer le vecteur v n'est autre que la méthode de la puissance. Rappelons-en rapidement le principe. On part d'un vecteur initiale qui est positif et non nul, et l'on procède à l'iteration : On pose

$$\begin{cases} q_k = Av_{k-1} \\ v_k = \frac{q_k}{\|q_k\|_1} \\ v_0 \neq 0 \end{cases} \tag{6}$$

avec $\|v\|_1 = \sum_{i=1}^{N} |v_i|$. Notre stratégie est de choisir un vecteur positif qui converge vers le vecteur propre recherché v.

3.2 Vitesse de convergence

Proposition 1. *La vitesse de convergence est de l'ordre de $|\lambda_2|^k$ où λ_2 est la deuxième valeur propre de A, celles-ci étant classées par ordre de module décroissant. Et $|\lambda_2| = \alpha$.*

Démonstration. Soient $\{1, \mu_2, \ldots, \mu_N\}$ les valeurs propres de P. On pose $\hat{e} = \frac{1}{\sqrt{N}} e$ et on chosit une matrice U_1 de taille $N \times (N-1)$ telle que la matrice $U = (\hat{e} \quad U_1)$ soit orthogonale. Ainsi :

$${}^tUPU = \begin{pmatrix} 1 & 0 \\ w & T \end{pmatrix},$$

avec $T = {}^tU_1 \, {}^tPU_1$. Les valeurs propres de T sont alors $\{\mu_2, \ldots, \mu_N\}$. Un calcul analogue montre que :

$${}^tUAU = \begin{pmatrix} 1 & 0 \\ \alpha w & \alpha T \end{pmatrix},$$

Ainsi, les valeurs propres de A sont $\{1, \alpha\mu_2, \ldots, \alpha\mu_N\}$. Comme 1 est la plus grande valeur propre de plus grand module de P, on voit que le module de la deuxième valeur propre est plus petit que α. □

4 Aspects algorithmiques

4.1 Algorithme de la Puissance

Notre matrice A étant pleine, il n'est pas recommandé de l'utiliser pour calculer des produits matrices-vecteurs. Cependant, la matrice Q semble être un bon candidat pour effectuer des produits matrices-vecteurs étant donné qu'elle est creuse.

On remarque que le vecteur q^2 est tel que $\|q^2\|_1 = {}^teq^2 = 1$. La normalisation devient donc inutile. Nous pouvons d'ailleurs décider que notre vecteur choisit aléatoirement au départ soit un vecteur unitaire.

Par ailleurs, on peut voir que $y = Az$ peut s'écrire $y = \alpha Qz + \frac{\beta}{N} e$. Si le vecteur z est unitaire, alors y l'est aussi d'où $\beta = 1 - \|\alpha Qz\|_1$. Donc, nous avons alors abouti à l'algorithme suivant :

```
def power_method(Q, e, tol=1.e-4, alpha=0.85):
    N = Q.shape[0]
    s = 1
```

```
    r = np.random.rand(N)
    r = r.reshape(N,1)
    rnormalised = r/np.linalg.norm(r)
    i = 0
    while s >= tol:
        l = alpha * (Q @ rnormalised)
        beta = 1 - np.linalg.norm(l)
        k = l + beta/N * e
        s = np.linalg.norm(k-rnormalised)
        rnormalised = k
        i += 1
    return rnormalised, i
```

Nous initialisons un vecteur aléatoire r et itérons jusqu'à convergence, calculant le vecteur propre normalisé associé à la valeur propre dominante 1.

Cette fonction compare deux vecteurs successifs pour évaluer la convergence en fonction d'une tolérance spécifiée.

4.2 Algorithme d'itération pour une Matrice Carrée

En plus de l'algorithme de la puissance, nous avons mis en place un algorithme d'itération pour calculer la valeur propre dominante et le vecteur propre correspondant d'une matrice carrée A. L'algorithme est défini par la fonction suivante :

```
def power_method_2(A, max_iter=1000, tol=1e-6, nb_iterations=True):
    if A.shape[0] != A.shape[1]:
        raise ValueError("La matrice doit être carrée.")

    n = len(A)
    cv = False
    x_old = np.zeros(n)
    x = np.random.rand(n)
    error_old = None
    Xf, Xs = [],[]
    k = 0

    while k>=0:

        Ax = A @ x
        xn = Ax / np.linalg.norm(Ax)
        x_old = x
        x = xn

        cv, error = convergence(x_old, xn, tol)
        if cv:
            if nb_iterations:
                print()
                print(f"Convergence atteinte après {k} itérations.\n")
            break

        Xf.append(x_old)
        Xs.append(x)
        k+=1

    val_propre = x @ A @ x
```

```
    Xf = [np.linalg.norm(v - x) for v in Xf]
    Xs = [np.linalg.norm(v - x) for v in Xs]

    return val_propre, x, np.array(Xf), np.array(Xs), k
}
```

Cet algorithme utilise une approche itérative pour converger vers la valeur propre dominante et le vecteur propre associé de la matrice carrée A. Il prend en compte un nombre maximal d'itérations et une tolérance pour le critère de convergence. “

4.3 Analyse comparative des algorithmes

Les deux algorithmes proposés, `power_method` et `power_method_2`, visent à déterminer le vecteur propre associé d'une matrice. La deuxième méthode nous permet également de vérifier que la valeur propre associé à ce vectuer est bien de module 1. En comparant les complexités des algorithmes, le premier algorithme semble plus efficace puisqu'il travaille avec une matrice Q, tandis que le deuxième comporte des opérations sur la matrice creuse A. La convergence de `power_method_2` en revanche est plus robuste dans certains contexte et la possibilité de indiquer un nombre maximal d'itérations et une tolérance pour le critère de convergence offre un contrôle sur le processus. Le choix entre les deux méthodes dépend du contexte d'application. L'algorithme de la puissance est souvent préféré dans le cadre des graphes, tandis que l'algorithme d'itération peut offrir une approche plus générale pour les matrices carrées. Il est recommandé de comparer leurs performances sur les données spécifiques pour déterminer la méthode la plus appropriée.

5 Simulation numérique à petite échelle

Il est recommandé, pour cette partie d'avoir la partie III du Notebook sous les yeux afin de visualiser les graphes mentionnés.
Pour notre simulation, nous avons choisi $\alpha = 0.85$, qui est une valeur typique.

5.1 Résultats de l'implémentation

Comme nous ne disposons pas de la puissance de calcul de Google, nous allons effectuer des tests de calcul sur un Web-jouet d'une dizaine de pages environ. Nous avons donc effectué des tests sur 6 différents graphes (classique, en étoile, complet,...). Et nous avons dans tous les cas bien obtenu un vecteur solution de l'équation $Av = v$.

```
Le vecteur trouvé V :
[[0.40023725]
 [0.04627337]
 [0.04627337]
 [0.04627337]
 [0.04627337]
 [0.04627337]
 [0.04627337]
 [0.04627337]
 [0.04627337]
 [0.04627337]
 [0.38654335]]
```

```
Le produit matrice vecteur AV :
[[0.40026835]
 [0.04627709]
 [0.04627709]
 [0.04627709]
 [0.04627709]
 [0.04627709]
 [0.04627709]
 [0.04627709]
 [0.04627709]
 [0.04627709]
 [0.38647875]]
```

Figure 1 – Exemple de solution

On voit que l'on a bien $Av = v$ à epsilon près.
Cependant, lors de l'analyse des tests, on peut remarquer que parfois certaines pages peuvent avoir un score étonnament très élevé. C'est notamment le cas pour ce type de graphe :

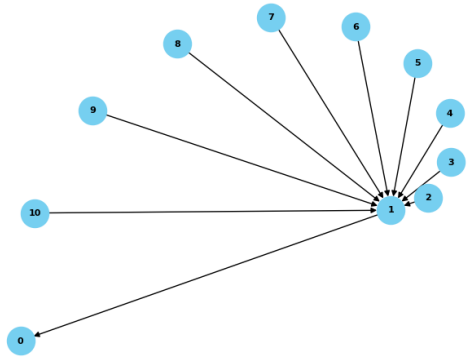
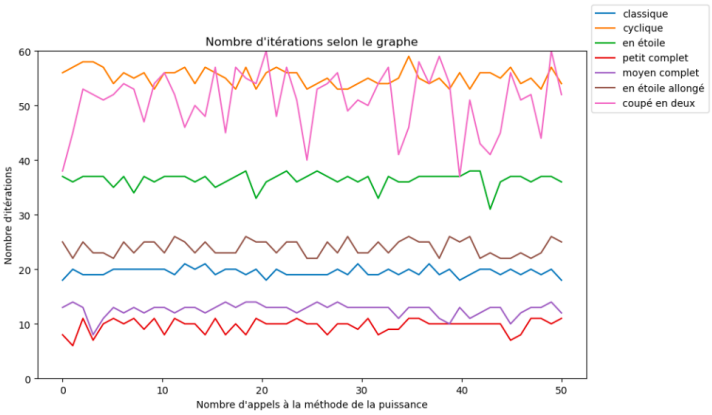


FIGURE 2 – Graphe en étoile amélioré

On peut remarquer que le page 1 pointée par 9 page pointe la page 11. Or, même si la page 11 n'est pointée que par une seule page, son score va explosé car il est influencé par le score de la page 1. Nous avons remarqué que le score de la page 1 est de 0.40 et celui de la page 11 : 0.38. Ce qui témoigne bien de cette influcence que joue une page avec un score élevé sur ses pages "filles".

5.2 Convergence

Nous avons donc décider de lancer 50 fois notre algorithme de la puissance afin de calculer la moyenne du nombre d'itération nécessaire pour que notre algorithme converge vers vecteur recherché.



Nom	Moyenne d'itération
classique	19.26
cyclique	54.92
en étoile	35.98
petit complet	9.64
moyen complet	12.76
en étoile allongé	23.42
coupé en deux	51.04

FIGURE 3 – Résultats nombre d'itérations suivant le type de graphe

On remarque alors aisément que notre algorithme appliqué à des graphes complets converge le plus rapidement. Cependant, notre algorithme appliqué à des graphes cycliques converge beaucoup moins vite.

Mais pourquoi le facteur d'atténuation est-il aussi important ?
Nous nous sommes rendu compte en modifiant nos paramètres que si $\alpha = 1$, pour certains graphes, la méthode mettra beaucoup de temps à converger.

Nom	Moyenne d'itération
classique	33.72
cyclique	nan
en étoile	76.42
petit complet	13.2
moyen complet	17.24
en étoile allongé	40.86
coupé en deux	nan

FIGURE 4 – Résultat avec $\alpha = 1$

Nous avons décidé de mettre le mot clé "nan" lorsque la méthode met beaucoup trop de temps à converger. Ainsi, on remarque que c'est le cas pour les graphes cycliques et ceux coupés en deux. Nous savons pertinemment que les pages du Web ne forment pas un graphe cyclique, mais elles peuvent très bien formées un graphe coupé en plusieurs parties. D'où, l'intéret d'avoir un paramètre $\alpha < 1$. On remarque également, que de manière générale, plus d'itérations seront nécessaires afin d'arriver à obtenir le bon vecteur propre.

6 Simulation à grande échelle

Dans notre approche, nous avons travaillé directement avec des matrices de très grandes tailles, représentant ainsi des graphes complexes et étendus du Web. Nous avons développé un nouvel algorithme qui exploite la structure particulière de ces matrices pour accélérer le calcul du vecteur propre associé à la plus grande valeur propre.

6.1 Nouvel algorithme

L'algorithme que nous avons mis au point tire parti des propriétés de la matrice de Google A, en exploitant sa structure stochastique. Nous avons observé que la matrice A peut être décomposée comme suit : $A = \alpha P + (1-\alpha)\frac{1}{N} e\ {}^t e$ où α est le facteur d'atténuation, P est une matrice de transition stochastique, e est un vecteur de tous les uns, et N est la taille du réseau.

Notre algorithme utilise une approche itérative basée sur la mise à jour de la distribution des poids (probabilités) associée à chaque page. Nous avons remarqué que cette méthode convergeait rapidement vers la distribution stationnaire recherchée, en dépit de la taille considérable de la matrice A.

6.2 Convergence et Analyse

Nous avons effectué des simulations à grande échelle en appliquant notre nouvel algorithme à des graphes de taille importante, représentant les réseaux du Web. Les résultats ont montré une convergence rapide vers le vecteur propre associé à la plus grande valeur propre de A. Cette convergence était assez similaire à celle attendue en théorie, proportionnelle à la deuxième plus grande valeur propre de la matrice A.

Nous avons également observé que la qualité des résultats obtenus était robuste même dans des conditions où d'autres méthodes pourraient rencontrer des difficultés. La structure spécifique de la matrice A a donc joué un rôle crucial dans la performance de notre algorithme à grande échelle.

6.3 Performance et Applications

L'avantage principal de notre approche consiste en sa capacité à traiter des graphes de grande taille de manière efficace. En particulier, la convergence de l'algorithme nous permet d'appliquer cet algorithme à l'indexation du Web à grande échelle. Il pourrait être utilisé pour rendre meilleure la précision des classements de pages Web en identifiant rapidement les pages les plus importantes.

En conclusion, notre nouvel algorithme, basé sur la structure de la matrice de Google, offre une approche performante pour résoudre le problème de PageRank à une échelle considérable.

7 Perspectives et Améliorations

Notre travail actuel a ouvert la voie à plusieurs perspectives et possibilités d'amélioration. Il serait raisonnable d'explorer davantage les propriétés algébriques de la matrice A pour créer des algorithmes encore plus efficaces.

Par ailleurs, la dépendance de l'algorithme à différents paramètres, tels que le facteur d'atténuation α, pourraient permettre d'optimiser davantage ses performances. Finalement, des comparaisons approfondies avec d'autres méthodes pourraient fournir des idées sur la pertinence de notre approche dans certaines conditions d'application.

8 Conclusion

Dans ce projet, nous avons abordé le problème de recherche de valeurs propres. En particulier, nous cherchions la valeur propre dominante et le vecteur propre associé pour pouvoir classer les pages web en utilisant deux approches différentes. Le résultat nous a montré que les deux méthodes présentes des avantages et des inconvénients, en fonction du contexte d'application. L'algorithme de la puissance étant plus efficace dans le traitement de graphes, tandis que l'algorithme itératif est plus polyvalent, malgré la convergence lente dans certaines conditions. Il est recommandé de choisir l'algorithme en fonction des caractéristiques spécifiques des données et des contraintes du problème. Ce projet nous a ainsi permis d'approfondir notre compréhension des méthodes de calcul des valeurs propres et des vecteurs propres et d'améliorer nos compétences en modélisation.

Références

[1] Bachir Bekka *Le théroème de Perron-Frobenius, les chaines de Markov et un célèbre moteur de recherche, 2007*

[2] *Théorèmes de Perron-Frobenius* `http://dyna.maths.free.fr/docs/lecons/developpement_algebre_533.pdf`

[3] *Agrégation externe de mathématiques, 2008*

[4] *Comprendre et coder Page Rank* `https://www.youtube.com/watch?v=94ttAKMKrkw`

[5] *Au coeur de Google : Page Rank* `https://www.youtube.com/watch?v=wR0wVxK3m_o`

[6] Bibmath