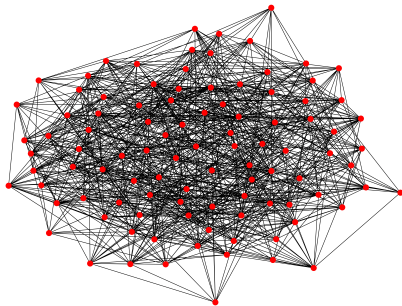


# Le PageRank

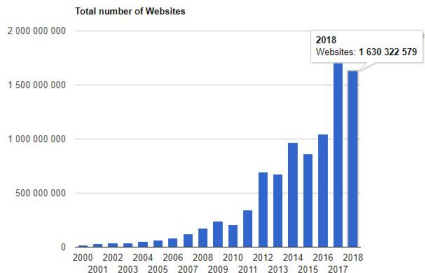
---

BOES Rémi  
2020-2021



# Introduction :

- Apparition du World Wide Web en 1989-1990
- Expansion du web avec l'arrivée de Google en 1995 et explosion à partir des années 2000

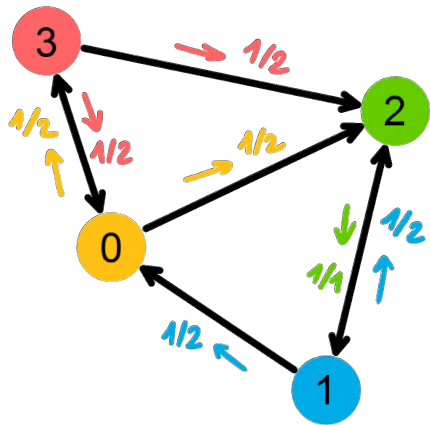


## Plan :

1. Définition du PageRank
2. Méthodes de calcul du PageRank
3. Résultats et comparaisons
4. Annexes

## Définition du PageRank

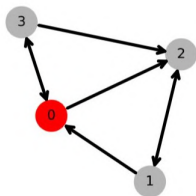
---



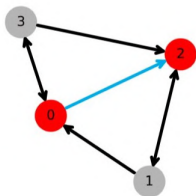
$$H = \begin{pmatrix} 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ 0 & 1 & 0 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 \end{pmatrix}$$

# Modèle du surfer aléatoire

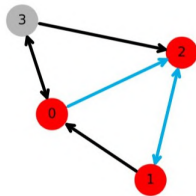
Marche aléatoire :



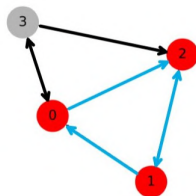
0 pas



1 pas

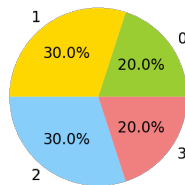


2 pas

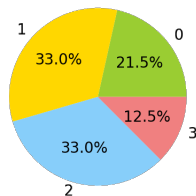


3 pas

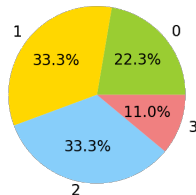
Fréquence de passages après :



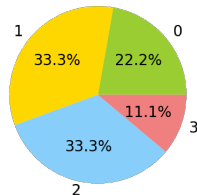
10 pas



$10^3$  pas



$10^5$  pas



$10^7$  pas <sup>5/41</sup>

# Modèle matricielle

Marche aléatoire matricielle :

vecteur initial :  $\mu_0 = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$

1 pas matriciel :  $\mu_1 = G * \mu_0$

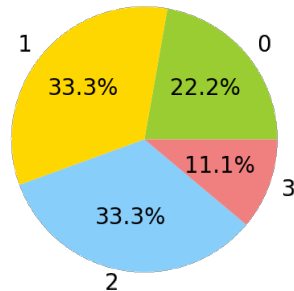
2 pas matriciel :  $\mu_2 = G * \mu_1 = G^2 * \mu_0$

$\vdots$

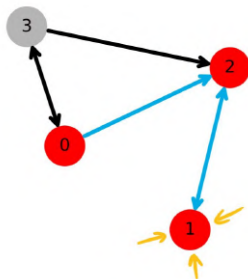
n pas matriciel :  $\mu_n = G * \mu_{n-1} = G^n * \mu_0$

Application numérique :

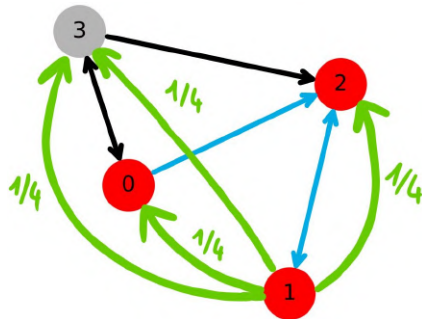
Pour  $n=10^7$  :  $\mu_n = \begin{pmatrix} 0.22222222 \\ 0.33333333 \\ 0.33333333 \\ 0.11111111 \end{pmatrix}$



# Ajustement du modèle



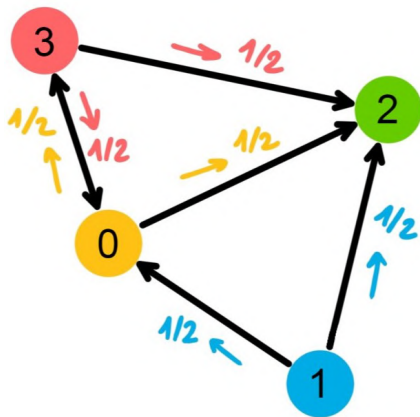
Trou noir



Redémarrage aléatoire



## Le modèle de Google



$$H = \begin{pmatrix} 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 \end{pmatrix}$$

## Le modèle de Google

$$G = c \left( \underbrace{\begin{pmatrix} 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 \end{pmatrix}}_H + \frac{1}{4} \underbrace{\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}}_S \right) + \frac{1-c}{4} \underbrace{\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}}_A$$

La matrice de transition de Google :  $G = c * (H + \frac{1}{n} * S) + \frac{1-c}{n} * A$

Introduction d'une probabilité  $c$  de suivre le modèle

## Vocabulaire :

---

### Matrice positive (strictement) :

Une matrice  $A = (a_{ij})_{1 \leq i, j \leq n} \in \mathcal{M}_n(\mathbb{R})$  est dite positive et on écrit  $A \geq 0$  (respectivement strictement positive et on écrit  $A > 0$ ) si :

$\forall (i, j) \in \llbracket 1 ; n \rrbracket^2$  on a  $a_{ij} \geq 0$  (respectivement  $a_{ij} > 0$ )

### Matrice primitive :

Soit  $A \in \mathcal{M}_n(\mathbb{R})$ ,  $A \geq 0$

A est dite primitive s'il existe  $k \in \mathbb{N}$  tel que  $A^k > 0$

## Théorème de Perron-Frobenius :

### Théorème de Perron-Frobenius :

Si  $A$  est une matrice primitive alors elle admet une valeur propre réelle strictement positive  $r > 0$  telle que :

- Pour toute autre valeur propre  $s$  de  $A$ , on a  $|s| < r$ ;
- L'espace propre associé à  $r$  est de dimension 1;
- Si  $s$  et  $S$  sont respectivement le minimum et le maximum des sommes des éléments de chaque ligne de  $A$ , on a  $s \leq r \leq S$
- Il existe un unique vecteur  $x^+$  de norme 1 à coordonnées strictement positives tel que  $Ax^+ = rx^+$

La valeur propre  $r$  s'appelle la valeur propre de Perron de  $A$  et  $x^+$  est le vecteur propre de Perron associé.

## Lien avec le PageRank :

---

Notons que  $\forall j$  on a :  $\sum_{i=0}^n g_{i,j} = 1$

$\Rightarrow$  la transposée de  $G$  a 1 pour valeur propre

$\Rightarrow 1$  est valeur propre de  $G$

Donc vu que  $G$  est primitive :  $r \leq 1$  et  $r$  est la plus grande valeur propre

$\Rightarrow r=1$

En notant  $\mu$  la limite de la suite  $(\mu_n)_{n \in \mathbb{N}}$  et en passant à la limite dans la récurrence  $\mu_n = G * \mu_{n-1}$  on obtient :

$$\mu = G * \mu \text{ donc } \mu = r$$

## Méthodes de calcul du PageRank

---

# Méthode de la puissance

---

**Principe :** calculer les éléments de la suite  $(\mu_n)_{n \in \mathbb{N}^*}$  telle que :

$$\mu_0 = \begin{pmatrix} \frac{1}{N} \\ \vdots \\ \frac{1}{N} \end{pmatrix} \text{ et } \forall n \in \mathbb{N}^* \mu_n = G * \mu_{n-1}$$

On effectue les itérations successives de la suite jusqu'à que l'écart entre 2 termes consécutifs soit suffisamment faible pour qu'on puisse considérer que la suite a convergé.

## Réécriture du système

Réécriture du problème sous la forme  $Ax=b$  :

On note  $\mu$  la valeur de convergence de la suite  $(\mu_n)_{n \in \mathbb{N}^*}$

$$\mu = H * \mu$$

$$\Leftrightarrow \mu = (cS + \frac{1-c}{n}ee^T)\mu$$

$$\Leftrightarrow \mu = cS\mu + \frac{1-c}{n}ee^T\mu$$

$$\Leftrightarrow (I_n - cS)\mu = \frac{1-c}{n}e$$

$$\Leftrightarrow A\mu = b$$

Le calcul de  $\mu$  revient à résoudre l'équation

$$\boxed{Ax = b}$$

telle que  $A = (I_n - cS)$  et  $b = \frac{1-c}{n}e$



### Matrice de preconditionnement :

Soit  $A \in \mathcal{M}_n(\mathbb{R})$ ,  $P \in \mathcal{GL}_n(\mathbb{R})$

$P$  est un preconditionneur de la matrice  $A$ , si il permet de diminuer le nombre d'itérations dans la méthode de résolution (itérative) du système  $Ax = b$  remplacé par  $P^{-1}Ax = P^{-1}b$

# Matrice de Préconditionnement

---

Supposons que  $A = M - N$  où  $M$  est inversible

Le système  $Ax=b$  peut se réécrire :

$$Ax = b$$

$$Mx = Nx + b$$

$$x = M^{-1}Nx + M^{-1}b$$

$$(I_n - M^{-1}N)x = M^{-1}b$$

$$M^{-1}Ax = M^{-1}b$$

La matrice  $M$  est une matrice de preconditionnement du système linéaire  $Ax = b$

On introduit la suite  $(x_k)_{k \in \mathbb{N}}$  telle que  $\forall k \in \mathbb{N}$  :

$$x_{k+1} = M^{-1}Nx_k + M^{-1}b$$

## Décomposition $A = D + U + L$

On décompose  $A = \begin{pmatrix} a_{0,0} & \dots & a_{1,n} \\ \vdots & & \vdots \\ a_{n,1} & \dots & a_{n,n} \end{pmatrix}$  telle que :

$$A = \underbrace{\begin{pmatrix} a_{0,0} & & 0 \\ & \ddots & \\ 0 & & a_{n,n} \end{pmatrix}}_D + \underbrace{\begin{pmatrix} 0 & a_{0,1} & \dots & a_{0,n} \\ \vdots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & a_{n-1,n} \\ 0 & \dots & \dots & 0 \end{pmatrix}}_U + \underbrace{\begin{pmatrix} 0 & \dots & \dots & 0 \\ a_{1,0} & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ a_{n,0} & \dots & a_{n,n-1} & 0 \end{pmatrix}}_L$$

# Matrice de Préconditionnement

Méthode	Matrice de préconditionnement
Jacobi	$M = D$
Gauss-Seidel (1)	$M = D + L$
Gauss-Seidel (2)	$M = D + U$
SOR (1)	$M = D + \omega L$
SOR (2)	$M = D + \omega U$
SSOR	$M = (D + \omega L)D^{-1}(D + \omega U)$

Tableau regroupant les matrices de préconditionnement selon la méthode de résolution employée

Introduction d'un facteur de relaxation  $\omega$  permettant d'influer sur la convergence

## Méthode d'extrapolation d'Aitken

---

**Principe :** On approxime périodiquement dans la méthode de la puissance  $x_n$  comme combinaison linéaire des deux premiers vecteurs propres de  $G$  :

$$x_n = u_1 + \alpha_2 u_2$$

$$x_{n+1} = u_1 + \alpha_2 \lambda_2 u_2$$

$$x_{n+2} = u_1 + \alpha_2 \lambda_2^2 u_2$$

Dans ce cas on montre que 
$$u_1 = x_n - \frac{(x_{n+1} - x_n)^2}{x_n - 2x_{n+1} + x_{n+2}}$$

On remplace la valeur de  $x_n$  par celle de  $u_1$  ainsi obtenue

## Méthode d'extrapolation quadratique

**Principe :** On approxime périodiquement dans la méthode de la puissance  $x_n$  comme combinaison linéaire des trois premiers vecteurs propres de  $G$  :

$$x_n = u_1 + \alpha_2 u_2 + \alpha_3 u_3$$

$$x_{n+1} = u_1 + \alpha_2 \lambda_2 u_2 + \alpha_3 \lambda_3 u_3$$

$$x_{n+2} = u_1 + \alpha_2 \lambda_2^2 u_2 + \alpha_3 \lambda_3^2 u_3$$

Dans ce cas on montre que  $u_1 = \beta_0 x_n + \beta_1 x_{n+1} + \beta_2 x_{n+2}$

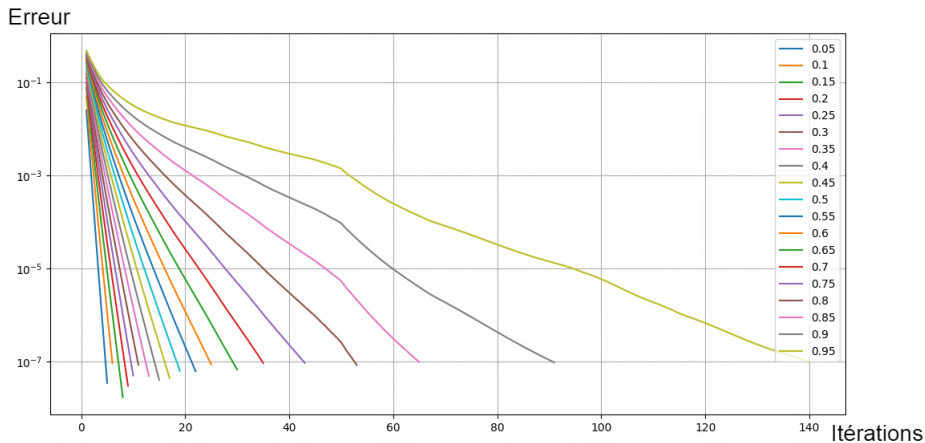
Avec  $\beta_0$ ,  $\beta_1$  et  $\beta_2$  déterminés à l'aide des 3 équations

On remplace la valeur de  $x_n$  par celle de  $u_1$  ainsi obtenue

## Résultats et comparaisons

---

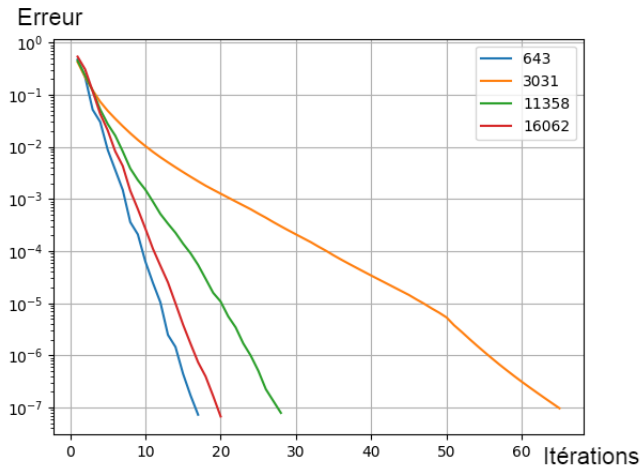
# Variation selon c



Graphique de l'erreur en fonction du nombre d'itérations selon c (puissance)

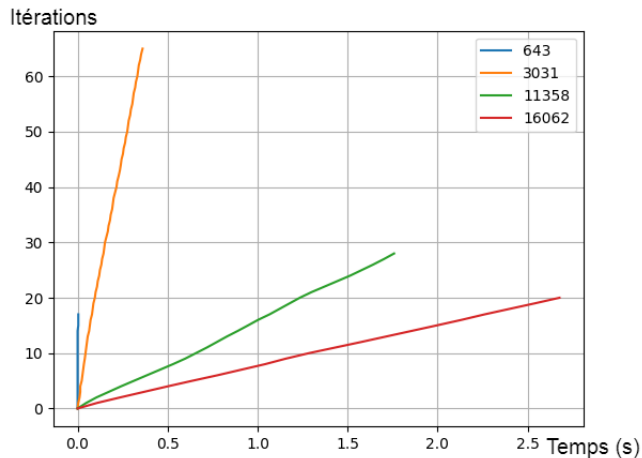


## Comparaison taille graphe (itérations)

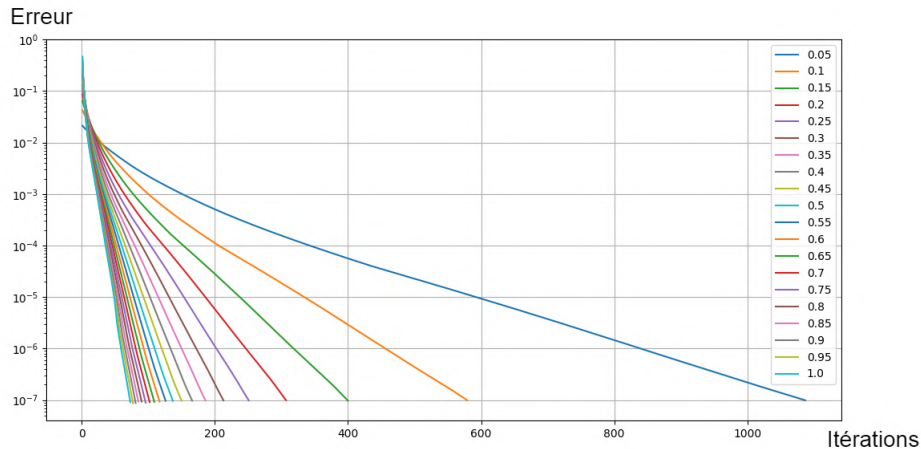


Graphique de l'erreur en fonction du nombre d'itérations selon la taille du graphe (puissance)<sup>22/41</sup>

## Comparaison taille graphe (temporelle)

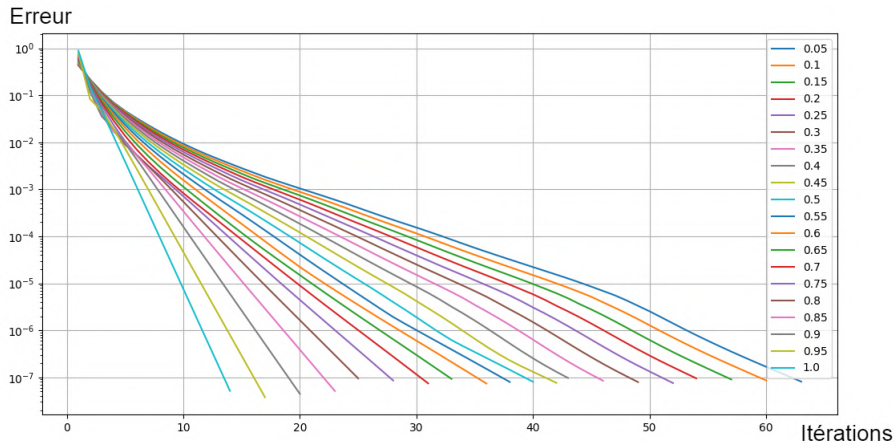


# Comparaison selon w (SOR)



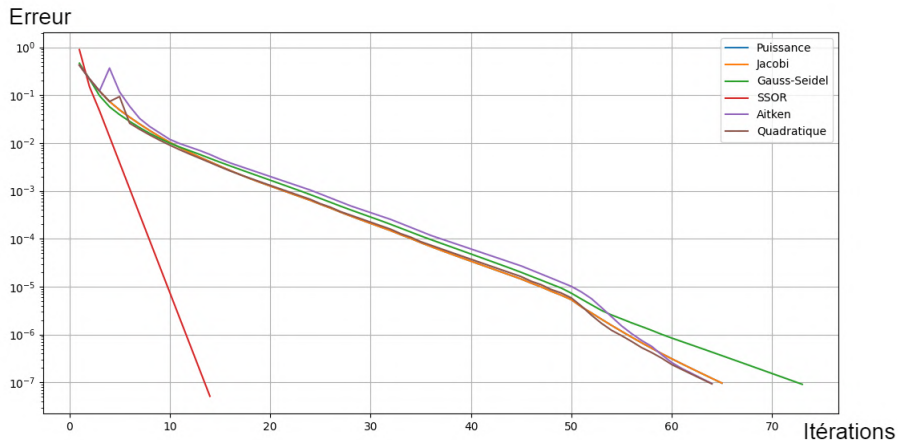
Graphique de l'erreur en fonction du nombre d'itérations selon w (SOR)

## Comparaison selon w (SSOR)



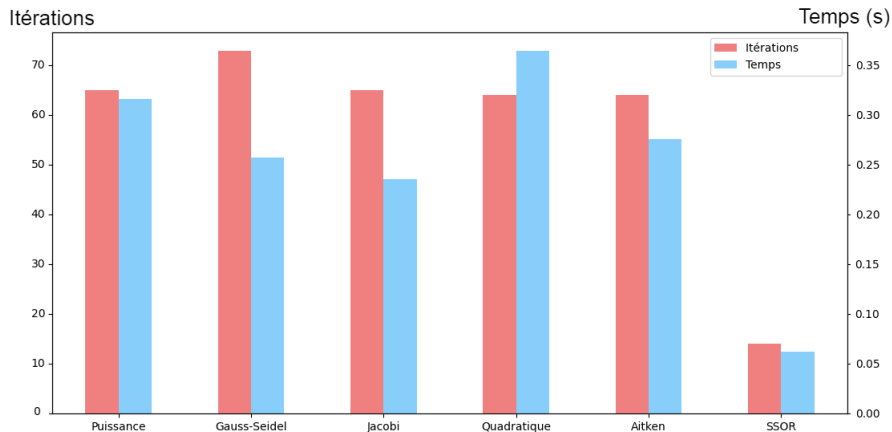
Graphique de l'erreur en fonction du nombre d'itérations selon w (SSOR)

# Comparaison des méthodes (itératifs)



Graphique de l'erreur en fonction du nombre d'itérations

# Comparaison des méthodes (itératifs et temporelle)



Graphique du nombre d'itérations et du temps de convergence

## Annexes

---

# Algorithme comparaison c

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 n=3031 #taille du graphe
4 ext = np.loadtxt('graphtrue.txt')
5 h=np.zeros((n,n))
6
7 for i in range(n):
8     ligne=ext[i]
9     h[int(ligne[0])][int(ligne[1])]=1
10 #Methode de la puissance
11 s=np.empty((n,n))
12 g=np.empty((n,n))
13 for i in range(n):
14     somme=np.sum(h[i])
15     if somme==0:
16         s[i]=[1/n]*n
17     else:
18         s[i]=np.copy(h[i]/somme)
19 #h matrice avec coeff=1/li si i pointe vers j
20 #s=h ou on rajoute 1/lj à tous les coeffs d'une ligne nulle
21
22 def f(x):
23     return np.dot(x,g)
24
25 x = np.ones(n)/n
26 N = 70
27 def puissance(f,x,result):
28     i=0
29     residu=10
30     while residu>1e-7:
31         x0=x
32         x=f(x)+((1-alpha)/n)
```

```
33         residu=np.linalg.norm(x-x0,1)
34         result.append(residu)
35         i+=1
36     return x
37
38 resultat=[]
39 for i in np.arange(0.05,1,0.05):
40     alpha=i
41     g=alpha*s
42     resultat.append([])
43     puissance(f,x,resultat[-1])
44
45 #affichage
46 for i in range(len(resultat)):
47     l=len(resultat[i])
48     x=[resultat[i][j] for j in range(l)]
49     y=[j for j in range(1,l+1)]
50     plt.plot(y,x,label=int((i*0.05+0.05)*(10**2))/(10**2))
51
52 plt.legend()
53 plt.yscale('log')
54 plt.grid()
55 plt.show()
```



# Algorithme convergence iterations - taille du graphe

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 def puissance(g,x0,result):
5     x=x0
6     i=0
7     residu=10
8     while residu>1e-7:
9         x0=x
10        x=np.dot(g,x)
11        residu=np.linalg.norm(x-x0,1)
12        result.append(residu)
13        i+=1
14    return x
15
16 resultat=[]
17 for i in range(4):
18     if i==0:
19         n=643 #taille du graphe
20         ext = np.loadtxt('graphtrue3.txt')
21     if i==1:
22         n=3031 #taille du graphe
23         ext = np.loadtxt('graphtrue.txt')
24     if i==2:
25         n=11358 #taille du graphe
26         ext = np.loadtxt('graphtrue4.txt')
27     if i==3:
28         n=16062 #taille du graphe
29         ext = np.loadtxt('graphtrue2.txt')
30     h=np.zeros((n,n))
31     for i in range(n):
32         ligne=ext[i]
```

```
33         h[int(ligne[0])][int(ligne[1])]=1
34     s=np.empty((n,n))
35     g=np.empty((n,n))
36     alpha=0.85
37     for i in range(n):
38         somme=np.sum(h[i])
39         if somme==0:
40             s[i]=[1/n]*n
41         else:
42             s[i]=np.copy(h[i]/somme)
43     g=np.transpose(alpha*s+((1-alpha)/n))
44     x=np.ones(n)*1/n
45     resultat.append([])
46     print(puissance(g,x,resultat[-1]))
47 #affichage
48 for i in range(len(resultat)):
49     l=len(resultat[i])
50     x=[resultat[i][j] for j in range(l)]
51     y=[j for j in range(1,l+1)]
52     if i==0:
53         plt.plot(y,x,label="643")
54     if i==1:
55         plt.plot(y,x,label="3031")
56     if i==2:
57         plt.plot(y,x,label="11358")
58     if i==3:
59         plt.plot(y,x,label="16062")
60
61 plt.yscale('log')
62 plt.legend()
63 plt.grid()
64 plt.show()
```

# Algorithme convergence temps - taille du graphe

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 from time import time
4
5 def puissance(g,x0,result):
6     x=x0
7     i=0
8     residu=10
9     t=time()
10    result.append(0)
11    while residu>1e-7:
12        x0=x
13        x=np.dot(g,x)
14        residu=np.linalg.norm(x-x0,1)
15        t1=time()
16        result.append(t1-t)
17        i+=1
18    return x
19
20 resultat=[]
21 for i in range(4):
22     if i==0:
23         n=643 #taille du graphe
24         ext = np.loadtxt('graphtrue3.txt')
25     if i==1:
26         n=3031 #taille du graphe
27         ext = np.loadtxt('graphtrue.txt')
28     if i==2:
29         n=11358 #taille du graphe
30         ext = np.loadtxt('graphtrue4.txt')
31     if i==3:
32         n=16062 #taille du graphe
```

```
33         ext = np.loadtxt('graphtrue2.txt')
34         h=np.zeros((n,n))
35         for i in range(n):
36             ligne=ext[i]
37             h[int(ligne[0])][int(ligne[1])]=1
38         s=np.empty((n,n))
39         g=np.empty((n,n))
40         alpha=0.85
41         for i in range(n):
42             somme=np.sum(h[i])
43             if somme==0:
44                 s[i]=[1/n]*n
45             else:
46                 s[i]=np.copy(h[i]/somme)
47         g=np.transpose(alpha*s+((1-alpha)/n))
48         x=np.ones(n)*1/n
49         resultat.append([])
50         print(puissance(g,x,resultat[-1]))
51
52 #affichage
53
54 for i in range(len(resultat)):
55     l=len(resultat[i])
56     y=[resultat[i][j] for j in range(l)]
57     x=[j for j in range(l)]
58     if i==0:
59         plt.plot(y,x,label="643")
60     if i==1:
61         plt.plot(y,x,label="3031")
62     if i==2:
63         plt.plot(y,x,label="11358")
64     if i==3:
```

```
65     plt.plot(y,x,label="16062")
66
67     plt.legend()
68     plt.grid()
69     plt.show()
```

# Algorithme comparaison w (SOR et SSOR)

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 n=3031 #taille du graphe
5 ext = np.loadtxt('graphtrue.txt')
6
7 h=np.zeros((n,n))
8
9 for i in range(n):
10     ligne=ext[i]
11     h[int(ligne[0])][int(ligne[1])]=1
12 #Methode de la puissance
13 s=np.empty((n,n))
14 g=np.empty((n,n))
15 alpha=0.85
16 for i in range(n):
17     somme=np.sum(h[i])
18     if somme==0:
19         s[i]=[1/n]*n
20     else:
21         s[i]=np.copy(h[i]/somme)
22 #h matrice avec coeff=1/li si i pointe vers j
23 #s=h ou on rajoute 1/Lj à tous les coeffs d'une ligne nulle
24 g=np.transpose(alpha*s+((1-alpha)/n))
25 #on creer g=alpha*s +((1-alpha)/n)*E
26 #x=np.ones(n)*1/n #vecteur initial pagerank
27 x=np.ones(n)*1/n
28
29 A = (np.eye(n)-alpha*np.transpose(s))
30 b = (((1-alpha)/n)*np.ones(n))
31 D = np.diag(np.diag(A))
32 L=np.tril(A)-D
```

```
33 U=A-L-D
34 loop=80
35
36 def precondition(M,N,b,x0,loop,result):
37     invM=np.linalg.inv(M)
38     K1=np.dot(invM,N)
39     K2=np.dot(invM,b)
40     x=x0
41     residu=10
42     while residu>1e-7:
43         x0=x
44         x=np.dot(K1,x)+K2
45         residu=np.linalg.norm(x-x0,1)
46         result.append(residu)
47     return x
48
49 #SSOR
50 resultat=[]
51 for i in np.arange(1.0,1.35,0.05):
52     w=i
53     M=np.dot(np.dot(D+w*L,np.linalg.pinv(D)),D+w*U)
54     N=M-A
55     resultat.append([])
56     print(precond(M,N,b,x,loop,resultat[-1]))
57     ""
58 #SOR
59 M=D/w+L
60 ""
61 #affichage
62
63 for i in range(len(resultat)):
64     l=len(resultat[i])
```

# Algorithme comparaison w (SOR et SSOR)

---

```
65     x=[resultat[i][j] for j in range(1)]
66     y=[j for j in range(1,l+1)]
67     plt.plot(y,x,label=int((i*0.05+1.0)*(10**2))/(10**2))
68
69     plt.yscale('log')
70     plt.legend()
71     plt.grid()
72     plt.show()
```

# Algorithme comparaison méthode (itérations)

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 n=3031 #taille du graphe
5 ext = np.loadtxt('graphtrue.txt')
6 """
7 http://networkrepository.com/web.php
8
9 n=643 #taille du graphe
10 ext = np.loadtxt('graphtrue3.txt')
11 n=3031 #taille du graphe
12 ext = np.loadtxt('graphtrue.txt')
13 n=11358 #taille du graphe
14 ext = np.loadtxt('graphtrue4.txt')
15 n=16062 #taille du graphe
16 ext = np.loadtxt('graphtrue2.txt')
17 """
18 h=np.zeros((n,n))
19
20 for i in range(n):
21     ligne=ext[i]
22     h[int(ligne[0])][int(ligne[1])]=1
23 #Methode de la puissance
24 s=np.empty((n,n))
25 g=np.empty((n,n))
26 alpha=0.85
27 for i in range(n):
28     somme=np.sum(h[i])
29     if somme==0:
30         s[i]=[1/n]*n
31     else:
32         s[i]=np.copy(h[i]/somme)
```

```
33 #h matrice avec coeff=1/li si i pointe vers j
34 #s=h ou on rajoute 1/lj à tous les coeffs d'une ligne nulle
35 g=np.transpose(alpha*s+((1-alpha)/n))
36 #on creer g=alpha*s +((1-alpha)/n)*E
37 #x=np.ones(n)*1/n #vecteur initial pagerank
38 x=np.ones(n)*1/n
39
40 A = (np.eye(n)-alpha*np.transpose(s))
41 b = (((1-alpha)/n))*np.ones(n)
42 D = np.diag(np.diag(A))
43 L=np.tril(A)-D
44 U=A-L-D
45 epsilon=1e-7
46
47 def puissance(g,x0,result):
48     x=x0
49     i=0
50     residu=10
51     while residu>epsilon:
52         x0=x
53         x=np.dot(g,x)
54         residu=np.linalg.norm(x-x0,1)
55         result.append(residu)
56         i+=1
57     return x
58
59
60 def precondition(M,b,x0,result):
61     invM=np.linalg.inv(M)
62     K1=np.dot(invM,N)
63     K2=np.dot(invM,b)
64     x=x0
```

# Algorithme comparaison méthode (itérations)

```
65     i=0
66     residu=10
67     while residu>epsilon:
68         x0=x
69         x=np.dot(K1,x)+K2
70         residu=np.linalg.norm(x-x0,1)
71         result.append(residu)
72         i+=1
73     return x
74
75 def f(x):
76     return np.dot(g,x)
77
78 def Aitken(x0,x1,x2):
79     g = (x1-x0)**2;
80     h = x2-2*x1+x0;
81     return x2-np.divide(g, h)
82
83 def Steffensen(f,x,result):
84     x0=np.dot(g,x)
85     result.append(np.linalg.norm(x-x0,1))
86     x1=np.dot(g,x0)
87     result.append(np.linalg.norm(x1-x0,1))
88     i=2
89     residu=10
90     while residu>epsilon:
91         x = x0
92         x0 = x1
93         x1 = np.dot(g,x1)
94         if i==3:
95             x1 = Aitken(x,x0,x1)
96         residu=np.linalg.norm(x1-x0,1)
```

```
97         result.append(residu)
98         i+=1
99     return x1
100
101 def Quadruple_Extrapolation(x0, x1, x2, x3):
102     y1 = x1-x0;
103     y2 = x2-x0;
104     y3 = x3-x0;
105     Y = np.transpose(np.array([y1, y2]));
106     z = -np.dot(np.linalg.pinv(Y),y3)
107     z1=z[0]; z2=z[1]; z3=1;
108     z0 = -(z1+z2+z3);
109     B0 = z1+z2+z3;
110     B1 = z2+z3;
111     B2 = z3;
112     x = B0*x1 + B1*x2 + B2*x3;
113     return x
114
115 def quadratique(f,x0,result):
116     x1 = f(x0)
117     result.append(np.linalg.norm(x1-x0,1))
118     x2 = f(x1)
119     result.append(np.linalg.norm(x2-x1,1))
120     x3 = f(x2)
121     result.append(np.linalg.norm(x3-x2,1))
122     i=3
123     residu=10
124     while residu>epsilon:
125         x0 = x1;
126         x1 = x2;
127         x2 = x3;
128         x3 = f(x2);
```

# Algorithme comparaison méthode (itérations)

```
129         if i == 4:
130             x3 = Quadruple_Extrapolation(x0,x1,x2,x3);
131             x3=x3/np.linalg.norm(x3,1)
132             residu=np.linalg.norm(x3-x2,1)
133             result.append(residu)
134             i+=1
135         return(x3)
136
137     #Puissance
138     result_puissance=[]
139     print(puissance(g,x,result_puissance))
140
141     #Jacobi
142     result_jacobi=[]
143     M=D
144     N=-L-U
145     print(precond(M,b,x,result_jacobi))
146
147     #Gauss-Seidel
148     result_gauss=[]
149     M=D+L
150     N=M-A
151     print(precond(M,b,x,result_gauss))
152
153     #SSOR
154     result_ssor=[]
155     w=1
156     M=np.dot(np.dot(D+w*L,np.linalg.pinv(D)),D+w*U)
157     N=M-A
158     print(precond(M,b,x,result_ssor))
159
160     #Aitken
```

```
161     result_aitken=[]
162     print(Steffensen(f,x,result_aitken))
163
164     #Quadratique
165     result_quadratique=[]
166     print(quadratique(f,x,result_quadratique))
167
168     #affichage
169     l=len(result_puissance)
170     x=[result_puissance[i] for i in range(l)]
171     y=[i for i in range(1,l+1)]
172     plt.plot(y,x,label="Puissance")
173
174     l=len(result_jacobi)
175     x=[result_jacobi[i] for i in range(l)]
176     y=[i for i in range(1,l+1)]
177     plt.plot(y,x,label="Jacobi")
178
179     l=len(result_gauss)
180     x=[result_gauss[i] for i in range(l)]
181     y=[i for i in range(1,l+1)]
182     plt.plot(y,x,label="Gauss-Seidel")
183
184     l=len(result_ssor)
185     x=[result_ssor[i] for i in range(l)]
186     y=[i for i in range(1,l+1)]
187     plt.plot(y,x,label="SSOR")
188
189     l=len(result_aitken)
190     x=[result_aitken[i] for i in range(l)]
191     y=[i for i in range(1,l+1)]
192     plt.plot(y,x,label="Aitken")
```



# Algorithme comparaison méthode (itérations)

---

```
193
194 l=len(result_quadratique)
195 x=[result_quadratique[i] for i in range(l)]
196 y=[i for i in range(1,l+1)]
197 plt.plot(y,x,label="Quadratique")
198
199 plt.yscale('log')
200 plt.legend()
201 plt.grid()
202 plt.show()
```

# Algorithme comparaison méthode (itérations et temporelle)

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 from time import time
4 import pandas as pd
5 from pandas import plotting
6
7 n=3031 #taille du graphe
8 ext = np.loadtxt('/Users/Beaudinard/Documents/mp/tipe/graphtrue.txt')
9 h=np.zeros((n,n))
10
11 for i in range(n):
12     ligne=ext[i]
13     h[int(ligne[0])][int(ligne[1])]=1
14 #Methode de la puissance
15 s=np.empty((n,n))
16 g=np.empty((n,n))
17 alpha=0.85
18 for i in range(n):
19     somme=np.sum(h[i])
20     if somme==0:
21         s[i]=[1/n]*n
22     else:
23         s[i]=np.copy(h[i]/somme)
24 #h matrice avec coeff=1/li si i pointe vers j
25 #s=h ou on rajoute 1/Lj à tous les coeffs d'une ligne nulle
26 g=np.transpose(alpha*s+((1-alpha)/n))
27 #on creer g=alpha*s +((1-alpha)/n))*E
28 #x=np.ones(n)*1/n #vecteur initial pagerank
29 x=np.ones(n)*1/n
30
31 A = (np.eye(n)-alpha*np.transpose(s))
```

```
32 b = (((1-alpha)/n))*np.ones(n)
33 D = np.diag(np.diag(A))
34 L=np.tril(A)-D
35 U=A-L-D
36
37 def puissance(g,x0,result):
38     x=x0
39     i=0
40     t=time()
41     residu=10
42     while residu>1e-7:
43         x0=x
44         x=np.dot(g,x)
45         residu=np.linalg.norm(x-x0,1)
46         result.append(time()-t)
47         i+=1
48     return x
49
50
51 def precondition(M,N,b,x0,result):
52     invM=np.linalg.inv(M)
53     K1=np.dot(invM,N)
54     K2=np.dot(invM,b)
55     x=x0
56     i=0
57     t=time()
58     residu=10
59     while residu>1e-7:
60         x0=x
61         x=np.dot(K1,x)+K2
62         residu=np.linalg.norm(x-x0,1)
```

# Algorithme comparaison méthode (itérations et temporelle)

```
63         result.append(time()-t)
64         i+=1
65     return x
66
67 def f(x):
68     return np.dot(g,x)
69
70 def Aitken(x0,x1,x2):
71     g = (x1-x0)**2;
72     h = x2-2*x1+x0;
73     return x2-np.divide(g, h)
74
75 def Steffensen(f,x,result):
76     t=time()
77     x0=np.dot(g,x)
78     result.append(time()-t)
79     x1=np.dot(g,x0)
80     result.append(time()-t)
81     i=2
82     residu=10
83     while residu>1e-7:
84         x = x0
85         x0 = x1
86         x1 = np.dot(g,x1)
87         if i==3:
88             x1 = Aitken(x,x0,x1)
89         residu=np.linalg.norm(x1-x0,1)
90         result.append(time()-t)
91         i+=1
92     return x1
93
```

```
94 def Quadruple_Extrapolation(x0, x1, x2, x3):
95     y1 = x1-x0;
96     y2 = x2-x0;
97     y3 = x3-x0;
98     Y = np.transpose(np.array([y1, y2]));
99     z = -np.dot(np.linalg.pinv(Y),y3)
100     z1=z[0]; z2=z[1]; z3=1;
101     z0 = -(z1+z2+z3);
102     B0 = z1+z2+z3;
103     B1 = z2+z3;
104     B2 = z3;
105     x = B0*x1 + B1*x2 + B2*x3;
106     return x
107
108 def quadratique(f,x0,result):
109     t=time()
110     x1 = f(x0)
111     result.append(time()-t)
112     x2 = f(x1)
113     result.append(time()-t)
114     x3 = f(x2)
115     result.append(time()-t)
116     i=3
117     residu=10
118     while residu>1e-7:
119         x0 = x1;
120         x1 = x2;
121         x2 = x3;
122         x3 = f(x2);
123         if i%2 == 0:
124             x3 = Quadruple_Extrapolation(x0,x1,x2,x3);
```

# Algorithme comparaison méthode (itérations et temporelle)

```
125     x3=x3/np.linalg.norm(x3,1)
126     residu=np.linalg.norm(x3-x2,1)
127     result.append(time()-t)
128     i+=1
129     return(x3)
130
131 result=[]
132 #Puissance
133 result_puissance=[0]
134 print(puissance(g,x,result_puissance))
135 result.append([result_puissance[-1],len(result_puissance)-1])
136
137 #Jacobi
138 result_jacobi=[0]
139 M=D
140 N=-L-U
141 print(precond(M,N,b,x,result_jacobi))
142 result.append([result_jacobi[-1],len(result_jacobi)-1])
143
144 #Gauss-Seidel
145 result_gauss=[0]
146 M=D+L
147 N=M-A
148 print(precond(M,N,b,x,result_gauss))
149 result.append([result_gauss[-1],len(result_gauss)-1])
150
151 #SSOR
152 result_ssor=[0]
153 w=1
154 M=np.dot(np.dot(D+w*L,np.linalg.pinv(D)),D+w*U)
155 N=M-A
```

```
156 print(precond(M,N,b,x,result_ssor))
157 result.append([result_ssor[-1],len(result_ssor)-1])
158
159 #Aitken
160 result_aitken=[0]
161 print(steffensen(f,x,result_aitken))
162 result.append([result_aitken[-1],len(result_aitken)-1])
163
164 #Quadratique
165 result_quadratique=[0]
166 print(quadratique(f,x,result_quadratique))
167 result.append([result_quadratique[-1],len(result_quadratique)-1])
168
169 print(result)
170
171 x=[result[0][0],result[2][0],result[1][0],result[5][0],result[4][0],result[3][0]]
172 y=[result[0][1],result[2][1],result[1][1],result[5][1],result[4][1],result[3][1]]
173
174 color = dict(boxes='DarkGreen', whiskers='DarkOrange', medians='DarkBlue', caps='Gray')
175 mydata = pd.DataFrame({"Itérations":y,"Temps":x})
176 mydata.index = ["Puissance", "Gauss-Seidel", "Jacobi", "Quadratique", "Aitken", "SSOR"]
177
178 mydata.plot(kind="bar",secondary_y="Itérations",color = ['lightcoral', 'lightskyblue'])
179 plt.show()
```

# Algorithme comparaison méthode (itérations et temporelle)

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 from time import time
4 import pandas as pd
5 from pandas import plotting
6
7 n=3031 #taille du graphe
8 ext = np.loadtxt('/Users/Beaudinard/Documents/mp/tipe/graphtrue.txt')
9 h=np.zeros((n,n))
10
11 for i in range(n):
12     ligne=ext[i]
13     h[int(ligne[0])][int(ligne[1])]=1
14 #Methode de la puissance
15 s=np.empty((n,n))
16 g=np.empty((n,n))
17 alpha=0.85
18 for i in range(n):
19     somme=np.sum(h[i])
20     if somme==0:
21         s[i]=[1/n]*n
22     else:
23         s[i]=np.copy(h[i]/somme)
24 #h matrice avec coeff=1/li si i pointe vers j
25 #s=h ou on rajoute 1/Lj à tous les coeffs d'une ligne nulle
26 g=np.transpose(alpha*s+((1-alpha)/n))
27 #on creer g=alpha*s +((1-alpha)/n))*E
28 #x=np.ones(n)*1/n #vecteur initial pagerank
29 x=np.ones(n)*1/n
30
31 A = (np.eye(n)-alpha*np.transpose(s))
```

```
32 b = (((1-alpha)/n))*np.ones(n)
33 D = np.diag(np.diag(A))
34 L=np.tril(A)-D
35 U=A-L-D
36
37 def puissance(g,x0,result):
38     x=x0
39     i=0
40     t=time()
41     residu=10
42     while residu>1e-7:
43         x0=x
44         x=np.dot(g,x)
45         residu=np.linalg.norm(x-x0,1)
46         result.append(time()-t)
47         i+=1
48     return x
49
50
51 def precondition(M,N,b,x0,result):
52     invM=np.linalg.inv(M)
53     K1=np.dot(invM,N)
54     K2=np.dot(invM,b)
55     x=x0
56     i=0
57     t=time()
58     residu=10
59     while residu>1e-7:
60         x0=x
61         x=np.dot(K1,x)+K2
62         residu=np.linalg.norm(x-x0,1)
```