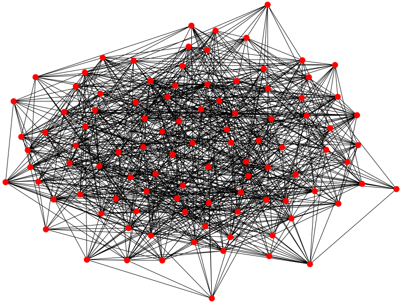
11020

Le PageRank

BOES Rémi

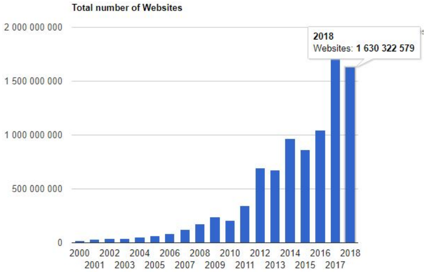
2020-2021



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

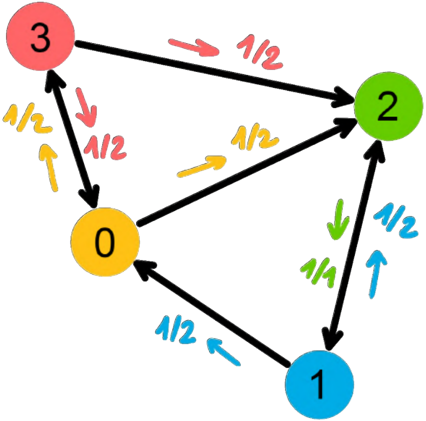


Nombre de sites web selon arobasenet.com

Plan :

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

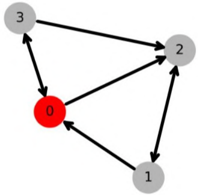
Définition du PageRank



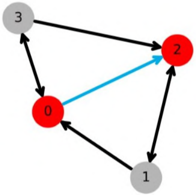
$$H = \begin{pmatrix} 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ 0 & 1 & 0 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 \end{pmatrix}$$

Modèle du surfer aléatoire

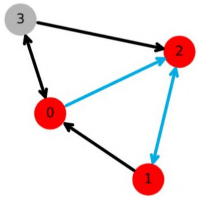
Marche aléatoire :



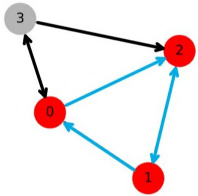
0 pas



1 pas

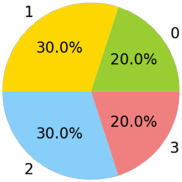


2 pas

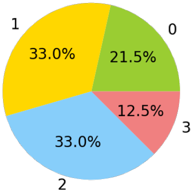


3 pas

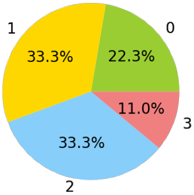
Fréquence de passages après :



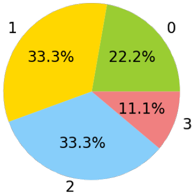
10 pas



10^3 pas



10^5 pas



10^7 pas

Modèle matricielle

Marche aléatoire matricielle :

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

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

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

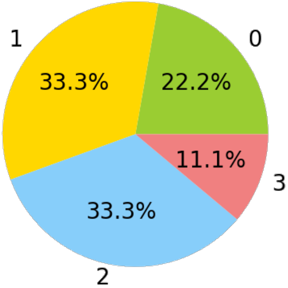
$\vdots$

$\vdots$

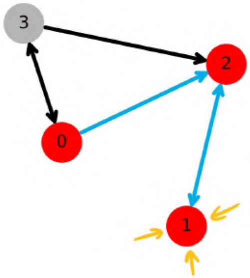
n pas matriciel : $\mu_n = G * \mu_{n-1} = G^n * \mu_0$

Application numérique :

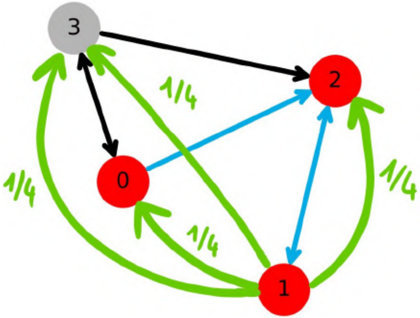
Pour n=10^7 : $\mu_n = \begin{pmatrix} 0.22222222 \\ 0.33333333 \\ 0.33333333 \\ 0.11111111 \end{pmatrix}$



Ajustement du modèle

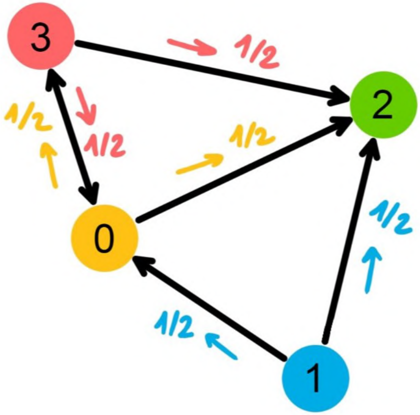


Trou noir



Redémarrage aléatoire

Le modèle de Google



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

$$G = c \left(\underbrace{\begin{pmatrix} 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 \end{pmatrix}}_{H} + \frac{1}{4} \underbrace{\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}}_{S} \right) + \frac{1-c}{4} \underbrace{\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}}_{A}$$

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

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 ≥ 0 (respectivement strictement positive et on écrit A > 0) si :
$\forall\ (i,j) \in [\![1\,;n]\!]^2$ on a $a_{ij} \geq 0$ (respectivement $a_{ij} > 0$)

Matrice primitive :

Soit A $\in \mathcal{M}_n(\mathbb{R})$, A ≥ 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 : $\sum_{j=0}^{n} g_{i,j} = 1$

$\Rightarrow$ 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 μ 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 :

$\mu = G * \mu$ 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 :

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

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

Réecriture du système

Réécriture du problème sous la forme Ax=b :

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

$$
\begin{aligned}
&\mu = H * \mu \\
&\Leftrightarrow \mu = (cS + \frac{1-c}{n}ee^T)\mu \\
&\Leftrightarrow \mu = cS\mu + \frac{1-c}{n}ee^T\mu \\
&\Leftrightarrow (I_n - cS)\mu = \frac{1-c}{n}e \\
&\Leftrightarrow A\mu = b
\end{aligned}
$$

Le calcul de μ revient à résoudre l'équation

$$\boxed{Ax = b}$$

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

Vocabulaire :

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 :

$$
\begin{aligned}
Ax &= b \\
Mx &= Nx + b \\
x &= M^{-1}Nx + M^{-1}b \\
(I_n - M^{-1}N)x &= M^{-1}b \\
M^{-1}Ax &= M^{-1}b
\end{aligned}
$$

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

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

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

Décomposition A = D + U + L

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

$$A = \underbrace{\begin{pmatrix} a_{0,0} & & 0 \\ & \ddots & \\ 0 & & a_{n,n} \end{pmatrix}}_{D} + \underbrace{\begin{pmatrix} 0 & a_{0,1} & \dots & a_{0,n} \\ \vdots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & a_{n-1,n} \\ 0 & \dots & \dots & 0 \end{pmatrix}}_{U} + \underbrace{\begin{pmatrix} 0 & \dots & \dots & 0 \\ a_{1,0} & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ a_{n,0} & \dots & a_{n,n-1} & 0 \end{pmatrix}}_{L}$$

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 ω permettant d'influer sur la convergence

Principe : On approxime périodiquement dans la méthode de la puissance x_n comme combinaison linéaire des <u>deux</u> 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 $\boxed{u_1 = x_n - \dfrac{(x_{n+1} - x_n)^2}{x_n - 2x_{n+1} + x_{n+2}}}$

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 <u>trois</u> premiers vecteurs propres de G :

$$x_n = u_1 + \alpha_2 u_2 + \alpha_3 u_3$$
$$x_{n+1} = u_1 + \alpha_2 \lambda_2 u_2 + \alpha_3 \lambda_3 u_3$$
$$x_{n+2} = u_1 + \alpha_2 \lambda_2^2 u_2 + \alpha_3 \lambda_3^2 u_3$$

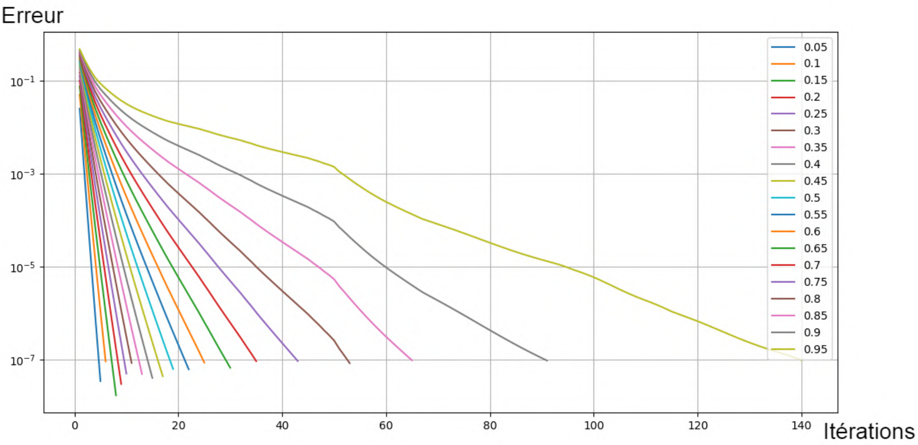
Dans ce cas on montre que $\boxed{u_1 = \beta_0 x_n + \beta_1 x_{n+1} + \beta_2 x_{n+2}}$

Avec β_0, β_1 et β_2 déterminés à l’aide des 3 équations

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

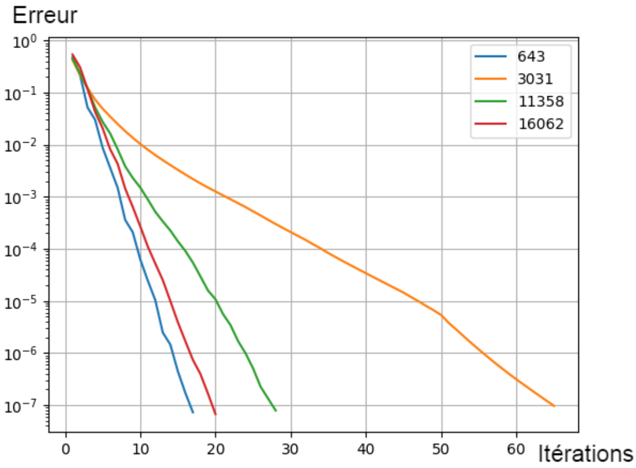
Résultats et comparaisons

Variation selon c



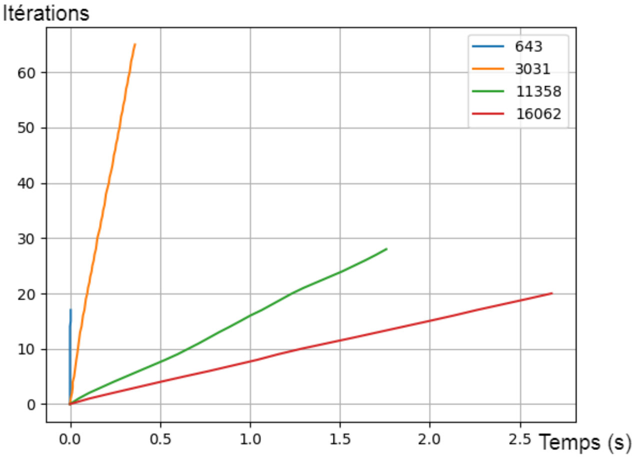
Graphique de l’erreur en fonction du nombre d’itérations selon c (puissance)

Comparaison taille graphe (itérations)



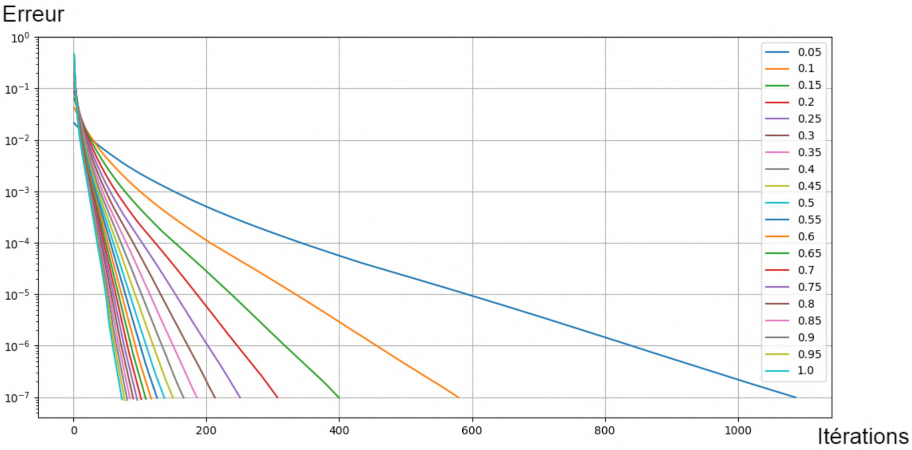
Graphique de l'erreur en fonction du nombre d'itérations selon la taille du graphe (puissance)

Comparaison taille graphe (temporelle)



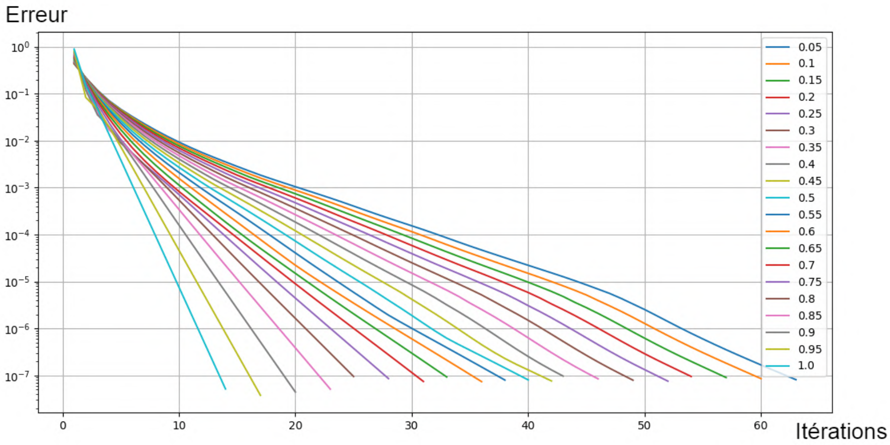
Graphique de l'erreur en fonction du temps selon la taille du graphe (puissance)

Comparaison selon w (SOR)



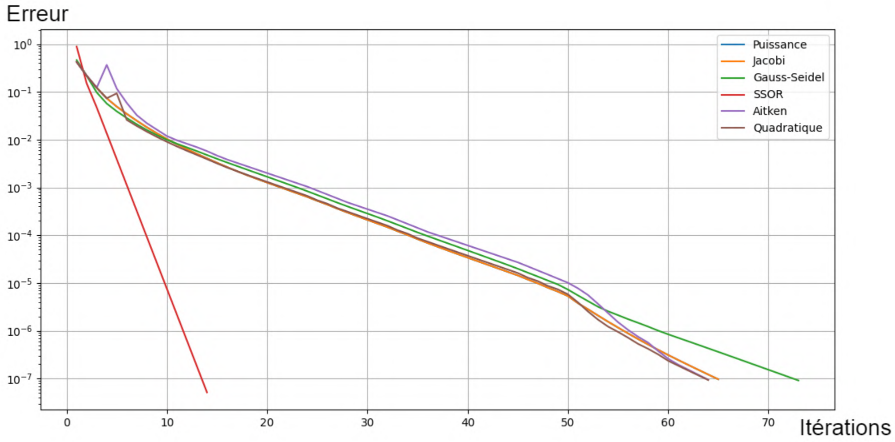
Graphique de l'erreur en fonction du nombre d'itérations selon w (SOR)

Comparaison selon w (SSOR)



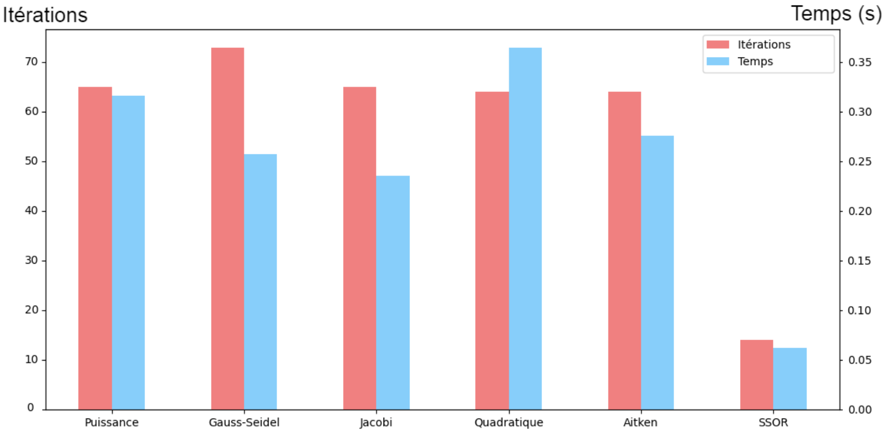
Graphique de l'erreur en fonction du nombre d'itérations selon w (SSOR)

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

```python
import numpy as np
import matplotlib.pyplot as plt
n=3031 #taille du graphe
ext = np.loadtxt('graphtrue.txt')
h=np.zeros((n,n))

for i in range(n):
    ligne=ext[i]
    h[int(ligne[0])][int(ligne[1])]=1
#Methode de la puissance
s=np.empty((n,n))
g=np.empty((n,n))
for i in range(n):
    somme=np.sum(h[i])
    if somme==0:
        s[i]=[1/n]*n
    else:
        s[i]=np.copy(h[i]/somme)
#h matrice avec coeff=1/li si i pointe vers j
#s=h ou on rajoute 1/lj à tous les coeffs d'une ligne nulle

def f(x):
    return np.dot(x,g)

x = np.ones(n)/n
N = 70
def puissance(f,x,result):
    i=0
    residu=10
    while residu>1e-7:
        x0=x
        x=f(x)+((1-alpha)/n)
        residu=np.linalg.norm(x-x0,1)
        result.append(residu)
        i+=1
    return x

resultat=[]
for i in np.arange(0.05,1,0.05):
    alpha=i
    g=alpha*s
    resultat.append([])
    puissance(f,x,resultat[-1])

#affichage
for i in range(len(resultat)):
    l=len(resultat[i])
    x=[resultat[i][j] for j in range(l)]
    y=[j for j in range(1,l+1)]
    plt.plot(y,x,label=int((i*0.05+0.05)*(10**2))/(10**2))

plt.legend()
plt.yscale('log')
plt.grid()
plt.show()
```

Algorithme convergence iterations - taille du graphe

```
import numpy as np
import matplotlib.pyplot as plt

def puissance(g,x0,result):
    x=x0
    i=0
    residu=10
    while residu>1e-7:
        x0=x
        x=np.dot(g,x)
        residu=np.linalg.norm(x-x0,1)
        result.append(residu)
        i+=1
    return x

resultat=[]
for i in range(4):
    if i==0:
        n=643 #taille du graphe
        ext = np.loadtxt('graphtrue3.txt')
    if i==1:
        n=3031 #taille du graphe
        ext = np.loadtxt('graphtrue.txt')
    if i==2:
        n=11358 #taille du graphe
        ext = np.loadtxt('graphtrue4.txt')
    if i==3:
        n=16062 #taille du graphe
        ext = np.loadtxt('graphtrue2.txt')
    h=np.zeros((n,n))
    for i in range(n):
        ligne=ext[i]
        h[int(ligne[0])][int(ligne[1])]=1
    s=np.empty((n,n))
    g=np.empty((n,n))
    alpha=0.85
    for i in range(n):
        somme=np.sum(h[i])
        if somme==0:
            s[i]=[1/n]*n
        else:
            s[i]=np.copy(h[i]/somme)
    g=np.transpose(alpha*s+((1-alpha)/n))
    x=np.ones(n)*1/n
    resultat.append([])
    print(puissance(g,x,resultat[-1]))
#affichage
for i in range(len(resultat)):
    l=len(resultat[i])
    x=[resultat[i][j] for j in range(l)]
    y=[j for j in range(1,l+1)]
    if i==0:
        plt.plot(y,x,label="643")
    if i==1:
        plt.plot(y,x,label="3031")
    if i==2:
        plt.plot(y,x,label="11358")
    if i==3:
        plt.plot(y,x,label="16062")

plt.yscale('log')
plt.legend()
plt.grid()
plt.show()
```

Algorithme convergence temps - taille du graphe

```
import numpy as np
import matplotlib.pyplot as plt
from time import time

def puissance(g,x0,result):
    x=x0
    i=0
    residu=10
    t=time()
    result.append(0)
    while residu>1e-7:
        x0=x
        x=np.dot(g,x)
        residu=np.linalg.norm(x-x0,1)
        t1=time()
        result.append(t1-t)
        i+=1
    return x

resultat=[]
for i in range(4):
    if i==0:
        n=643 #taille du graphe
        ext = np.loadtxt('graphtrue3.txt')
    if i==1:
        n=3031 #taille du graphe
        ext = np.loadtxt('graphtrue.txt')
    if i==2:
        n=11358 #taille du graphe
        ext = np.loadtxt('graphtrue4.txt')
    if i==3:
        n=16062 #taille du graphe
        ext = np.loadtxt('graphtrue2.txt')
    h=np.zeros((n,n))
    for i in range(n):
        ligne=ext[i]
        h[int(ligne[0])][int(ligne[1])]=1
    s=np.empty((n,n))
    g=np.empty((n,n))
    alpha=0.85
    for i in range(n):
        somme=np.sum(h[i])
        if somme==0:
            s[i]=[1/n]*n
        else:
            s[i]=np.copy(h[i]/somme)
    g=np.transpose(alpha*s+((1-alpha)/n))
    x=np.ones(n)*1/n
    resultat.append([])
    print(puissance(g,x,resultat[-1]))

#affichage

for i in range(len(resultat)):
    l=len(resultat[i])
    y=[resultat[i][j] for j in range(l)]
    x=[j for j in range(l)]
    if i==0:
        plt.plot(y,x,label="643")
    if i==1:
        plt.plot(y,x,label="3031")
    if i==2:
        plt.plot(y,x,label="11358")
    if i==3:
```

Algorithme convergence temps - taille du graphe

```
        plt.plot(y,x,label="16062")

plt.legend()
plt.grid()
plt.show()
```

Algorithme comparaison w (SOR et SSOR)

```python
import numpy as np
import matplotlib.pyplot as plt

n=3031 #taille du graphe
ext = np.loadtxt('graphtrue.txt')

h=np.zeros((n,n))

for i in range(n):
    ligne=ext[i]
    h[int(ligne[0])][int(ligne[1])]=1
#Methode de la puissance
s=np.empty((n,n))
g=np.empty((n,n))
alpha=0.85
for i in range(n):
    somme=np.sum(h[i])
    if somme==0:
        s[i]=[1/n]*n
    else:
        s[i]=np.copy(h[i]/somme)
#h matrice avec coeff=1/li si i pointe vers j
#s=h ou on rajoute 1/lj à tous les coeffs d'une ligne nulle
g=np.transpose(alpha*s+((1-alpha)/n))
#on creer g=alpha*s +((1-alpha)/n))*E
#x=np.ones(n)*1/n #vecteur inital pagerank
x=np.ones(n)*1/n

A = (np.eye(n)-alpha*np.transpose(s))
b = (((1-alpha)/n))*np.ones(n)
D = np.diag(np.diag(A))
L=np.tril(A)-D
U=A-L-D
loop=80

def precond(M,N,b,x0,loop,result):
    invM=np.linalg.inv(M)
    K1=np.dot(invM,N)
    K2=np.dot(invM,b)
    x=x0
    residu=10
    while residu>1e-7:
        x0=x
        x=np.dot(K1,x)+K2
        residu=np.linalg.norm(x-x0,1)
        result.append(residu)
    return x

#SSOR
resultat=[]
for i in np.arange(1.0,1.35,0.05):
    w=i
    M=np.dot(np.dot(D+w*L,np.linalg.pinv(D)),D+w*U)
    N=M-A
    resultat.append([])
    print(precond(M,N,b,x,loop,resultat[-1]))
"""
#SOR
M=D/w+L
"""
#affichage

for i in range(len(resultat)):
    l=len(resultat[i])
```

Algorithme comparaison w (SOR et SSOR)

```python
    x=[resultat[i][j] for j in range(l)]
    y=[j for j in range(1,l+1)]
    plt.plot(y,x,label=int((i*0.05+1.0)*(10**2))/(10**2))

plt.yscale('log')
plt.legend()
plt.grid()
plt.show()
```

Algorithme comparaison méthode (itérations)

```python
import numpy as np
import matplotlib.pyplot as plt

n=3031 #taille du graphe
ext = np.loadtxt('graphtrue.txt')
"""
http://networkrepository.com/web.php

n=643 #taille du graphe
ext = np.loadtxt('graphtrue3.txt')
n=3031 #taille du graphe
ext = np.loadtxt('graphtrue.txt')
n=11358 #taille du graphe
ext = np.loadtxt('graphtrue4.txt')
n=16062 #taille du graphe
ext = np.loadtxt('graphtrue2.txt')
"""
h=np.zeros((n,n))

for i in range(n):
    ligne=ext[i]
    h[int(ligne[0])][int(ligne[1])]=1
#Methode de la puissance
s=np.empty((n,n))
g=np.empty((n,n))
alpha=0.85
for i in range(n):
    somme=np.sum(h[i])
    if somme==0:
        s[i]=[1/n]*n
    else:
        s[i]=np.copy(h[i]/somme)
#h matrice avec coeff=1/li si i pointe vers j
#s=h ou on rajoute 1/lj à tous les coeffs d'une ligne nulle
g=np.transpose(alpha*s+((1-alpha)/n))
#on creer g=alpha*s +((1-alpha)/n))*E
#x=np.ones(n)*1/n #vecteur inital pagerank
x=np.ones(n)*1/n

A = (np.eye(n)-alpha*np.transpose(s))
b = (((1-alpha)/n))*np.ones(n)
D = np.diag(np.diag(A))
L=np.tril(A)-D
U=A-L-D
epsilon=1e-7

def puissance(g,x0,result):
    x=x0
    i=0
    residu=10
    while residu>epsilon:
        x0=x
        x=np.dot(g,x)
        residu=np.linalg.norm(x-x0,1)
        result.append(residu)
        i+=1
    return x


def precond(M,b,x0,result):
    invM=np.linalg.inv(M)
    K1=np.dot(invM,N)
    K2=np.dot(invM,b)
    x=x0
```

Algorithme comparaison méthode (itérations)

```python
    i=0
    residu=10
    while residu>epsilon:
        x0=x
        x=np.dot(K1,x)+K2
        residu=np.linalg.norm(x-x0,1)
        result.append(residu)
        i+=1
    return x

def f(x):
    return np.dot(g,x)

def Aitken(x0,x1,x2):
    g = (x1-x0)**2;
    h = x2-2*x1+x0;
    return x2-np.divide(g, h)

def Steffensen(f,x,result):
    x0=np.dot(g,x)
    result.append(np.linalg.norm(x-x0,1))
    x1=np.dot(g,x0)
    result.append(np.linalg.norm(x1-x0,1))
    i=2
    residu=10
    while residu>epsilon:
        x = x0
        x0 = x1
        x1 = np.dot(g,x1)
        if i==3:
            x1 = Aitken(x,x0,x1)
        residu=np.linalg.norm(x1-x0,1)
```

```python
        result.append(residu)
        i+=1
    return x1

def Quadruple_Extrapolation(x0, x1, x2, x3):
    y1 = x1-x0;
    y2 = x2-x0;
    y3 = x3-x0;
    Y = np.transpose(np.array([y1, y2]));
    z = -np.dot(np.linalg.pinv(Y),y3)
    z1=z[0]; z2=z[1]; z3=1;
    z0 = -(z1+z2+z3);
    B0 = z1+z2+z3;
    B1 = z2+z3;
    B2 = z3;
    x = B0*x1 + B1*x2 + B2*x3;
    return x

def quadratique(f,x0,result):
    x1 = f(x0)
    result.append(np.linalg.norm(x1-x0,1))
    x2 = f(x1)
    result.append(np.linalg.norm(x2-x1,1))
    x3 = f(x2)
    result.append(np.linalg.norm(x3-x2,1))
    i=3
    residu=10
    while residu>epsilon:
        x0 = x1;
        x1 = x2;
        x2 = x3;
        x3 = f(x2);
```

Algorithme comparaison méthode (itérations)

```python
        if i == 4:
            x3 = Quadruple_Extrapolation(x0,x1,x2,x3);
        x3=x3/np.linalg.norm(x3,1)
        residu=np.linalg.norm(x3-x2,1)
        result.append(residu)
        i+=1
    return(x3)

#Puissance
result_puissance=[]
print(puissance(g,x,result_puissance))

#Jacobi
result_jacobi=[]
M=D
N=-L-U
print(precond(M,b,x,result_jacobi))

#Gauss-Seidel
result_gauss=[]
M=D+L
N=M-A
print(precond(M,b,x,result_gauss))

#SSOR
result_ssor=[]
w=1
M=np.dot(np.dot(D+w*L,np.linalg.pinv(D)),D+w*U)
N=M-A
print(precond(M,b,x,result_ssor))

#Aitken
result_aitken=[]
print(Steffensen(f,x,result_aitken))

#Quadratique
result_quadratique=[]
print(quadratique(f,x,result_quadratique))

#affichage
l=len(result_puissance)
x=[result_puissance[i] for i in range(l)]
y=[i for i in range(1,l+1)]
plt.plot(y,x,label="Puissance")

l=len(result_jacobi)
x=[result_jacobi[i] for i in range(l)]
y=[i for i in range(1,l+1)]
plt.plot(y,x,label="Jacobi")

l=len(result_gauss)
x=[result_gauss[i] for i in range(l)]
y=[i for i in range(1,l+1)]
plt.plot(y,x,label="Gauss-Seidel")

l=len(result_ssor)
x=[result_ssor[i] for i in range(l)]
y=[i for i in range(1,l+1)]
plt.plot(y,x,label="SSOR")

l=len(result_aitken)
x=[result_aitken[i] for i in range(l)]
y=[i for i in range(1,l+1)]
plt.plot(y,x,label="Aitken")
```

Algorithme comparaison méthode (itérations)

```

l=len(result_quadratique)
x=[result_quadratique[i] for i in range(l)]
y=[i for i in range(1,l+1)]
plt.plot(y,x,label="Quadratique")

plt.yscale('log')
plt.legend()
plt.grid()
plt.show()
```

Algorithme comparaison méthode (itérations et temporelle)

```python
import numpy as np
import matplotlib.pyplot as plt
from time import time
import pandas as pd
from pandas import plotting

n=3031 #taille du graphe
ext = np.loadtxt('/Users/Beaudinard/Documents/mp/tipe/graphtrue.txt')
h=np.zeros((n,n))

for i in range(n):
    ligne=ext[i]
    h[int(ligne[0])][int(ligne[1])]=1
#Methode de la puissance
s=np.empty((n,n))
g=np.empty((n,n))
alpha=0.85
for i in range(n):
    somme=np.sum(h[i])
    if somme==0:
        s[i]=[1/n]*n
    else:
        s[i]=np.copy(h[i]/somme)
#h matrice avec coeff=1/li si i pointe vers j
#s=h ou on rajoute 1/lj à tous les coeffs d'une ligne nulle
g=np.transpose(alpha*s+((1-alpha)/n))
#on creer g=alpha*s +((1-alpha)/n))*E
#x=np.ones(n)*1/n #vecteur inital pagerank
x=np.ones(n)*1/n

A = (np.eye(n)-alpha*np.transpose(s))
```

```python
b = (((1-alpha)/n))*np.ones(n)
D = np.diag(np.diag(A))
L=np.tril(A)-D
U=A-L-D

def puissance(g,x0,result):
    x=x0
    i=0
    t=time()
    residu=10
    while residu>1e-7:
        x0=x
        x=np.dot(g,x)
        residu=np.linalg.norm(x-x0,1)
        result.append(time()-t)
        i+=1
    return x


def precond(M,N,b,x0,result):
    invM=np.linalg.inv(M)
    K1=np.dot(invM,N)
    K2=np.dot(invM,b)
    x=x0
    i=0
    t=time()
    residu=10
    while residu>1e-7:
        x0=x
        x=np.dot(K1,x)+K2
        residu=np.linalg.norm(x-x0,1)
```

Algorithme comparaison méthode (itérations et temporelle)

```python
            result.append(time()-t)
            i+=1
        return x

    def f(x):
        return np.dot(g,x)

    def Aitken(x0,x1,x2):
        g = (x1-x0)**2;
        h = x2-2*x1+x0;
        return x2-np.divide(g, h)

    def Steffensen(f,x,result):
        t=time()
        x0=np.dot(g,x)
        result.append(time()-t)
        x1=np.dot(g,x0)
        result.append(time()-t)
        i=2
        residu=10
        while residu>1e-7:
            x = x0
            x0 = x1
            x1 = np.dot(g,x1)
            if i==3:
                x1 = Aitken(x,x0,x1)
            residu=np.linalg.norm(x1-x0,1)
            result.append(time()-t)
            i+=1
        return x1

```

```python
def Quadruple_Extrapolation(x0, x1, x2, x3):
    y1 = x1-x0;
    y2 = x2-x0;
    y3 = x3-x0;
    Y = np.transpose(np.array([y1, y2]));
    z = -np.dot(np.linalg.pinv(Y),y3)
    z1=z[0]; z2=z[1]; z3=1;
    z0 = -(z1+z2+z3);
    B0 = z1+z2+z3;
    B1 = z2+z3;
    B2 = z3;
    x = B0*x1 + B1*x2 + B2*x3;
    return x

def quadratique(f,x0,result):
    t=time()
    x1 = f(x0)
    result.append(time()-t)
    x2 = f(x1)
    result.append(time()-t)
    x3 = f(x2)
    result.append(time()-t)
    i=3
    residu=10
    while residu>1e-7:
        x0 = x1;
        x1 = x2;
        x2 = x3;
        x3 = f(x2);
        if i%2 == 0:
            x3 = Quadruple_Extrapolation(x0,x1,x2,x3);
```

Algorithme comparaison méthode (itérations et temporelle)

```python
        x3=x3/np.linalg.norm(x3,1)
        residu=np.linalg.norm(x3-x2,1)
        result.append(time()-t)
        i+=1
    return(x3)

result=[]
#Puissance
result_puissance=[0]
print(puissance(g,x,result_puissance))
result.append([result_puissance[-1],len(result_puissance)-1])

#Jacobi
result_jacobi=[0]
M=D
N=-L-U
print(precond(M,N,b,x,result_jacobi))
result.append([result_jacobi[-1],len(result_jacobi)-1])

#Gauss-Seidel
result_gauss=[0]
M=D+L
N=M-A
print(precond(M,N,b,x,result_gauss))
result.append([result_gauss[-1],len(result_gauss)-1])

#SSOR
result_ssor=[0]
w=1
M=np.dot(np.dot(D+w*L,np.linalg.pinv(D)),D+w*U)
N=M-A
```

```python
print(precond(M,N,b,x,result_ssor))
result.append([result_ssor[-1],len(result_ssor)-1])

#Aitken
result_aitken=[0]
print(Steffensen(f,x,result_aitken))
result.append([result_aitken[-1],len(result_aitken)-1])

#Quadratique
result_quadratique=[0]
print(quadratique(f,x,result_quadratique))
result.append([result_quadratique[-1],len(result_quadratique)-1])

print(result)

x=[result[0][0],result[2][0],result[1][0],result[5][0],result[4][0],result[3][0]]
y=[result[0][1],result[2][1],result[1][1],result[5][1],result[4][1],result[3][1]]

color = dict(boxes='DarkGreen', whiskers='DarkOrange',medians='DarkBlue', caps='Gray')
mydata = pd.DataFrame({"Itérations":y,"Temps":x})
mydata.index = ["Puissance","Gauss-Seidel","Jacobi","Quadratique","Aitken","SSOR"]

mydata.plot(kind="bar",secondary_y="Itérations",color = ['lightcoral', 'lightskyblue'])
plt.show()
```

Algorithme comparaison méthode (itérations et temporelle)

```python
import numpy as np
import matplotlib.pyplot as plt
from time import time
import pandas as pd
from pandas import plotting

n=3031 #taille du graphe
ext = np.loadtxt('/Users/Beaudinard/Documents/mp/tipe/graphtrue.txt')
h=np.zeros((n,n))

for i in range(n):
    ligne=ext[i]
    h[int(ligne[0])][int(ligne[1])]=1
#Methode de la puissance
s=np.empty((n,n))
g=np.empty((n,n))
alpha=0.85
for i in range(n):
    somme=np.sum(h[i])
    if somme==0:
        s[i]=[1/n]*n
    else:
        s[i]=np.copy(h[i]/somme)
#h matrice avec coeff=1/li si i pointe vers j
#s=h ou on rajoute 1/lj à tous les coeffs d'une ligne nulle
g=np.transpose(alpha*s+((1-alpha)/n))
#on creer g=alpha*s +((1-alpha)/n))*E
#x=np.ones(n)*1/n #vecteur inital pagerank
x=np.ones(n)*1/n

A = (np.eye(n)-alpha*np.transpose(s))
b = (((1-alpha)/n))*np.ones(n)
D = np.diag(np.diag(A))
L=np.tril(A)-D
U=A-L-D

def puissance(g,x0,result):
    x=x0
    i=0
    t=time()
    residu=10
    while residu>1e-7:
        x0=x
        x=np.dot(g,x)
        residu=np.linalg.norm(x-x0,1)
        result.append(time()-t)
        i+=1
    return x


def precond(M,N,b,x0,result):
    invM=np.linalg.inv(M)
    K1=np.dot(invM,N)
    K2=np.dot(invM,b)
    x=x0
    i=0
    t=time()
    residu=10
    while residu>1e-7:
        x0=x
        x=np.dot(K1,x)+K2
        residu=np.linalg.norm(x-x0,1)
```