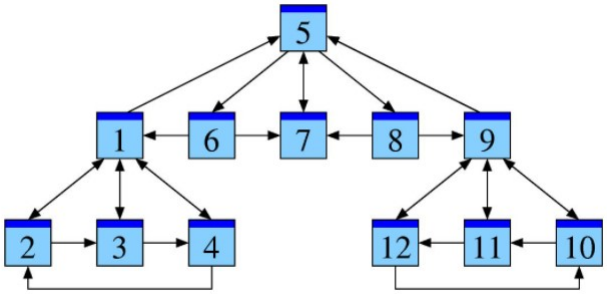
Google Ranking

Processus stochastiques

Ayika Ayoko
Deprun Alexandre
Edjinedja Kokou Laris

Présentation du modèle

Google est un moteur de recherches. Lorsqu'un utilisateur effectue une recherche, plusieurs pages web lui sont proposées. La plupart du temps les pages web mentionnent d'autres pages pour préciser les sources de leur contenu ou apporter d'autres informations et ce sont ces liens qui vont nous intéresser dans notre modèle. Nous modélisons donc le web par un graphe dont les N sommets sont les pages web reliés par les liens existants entre les pages. Prenons l'exemple d'un graphe de $N = 12$ pages:



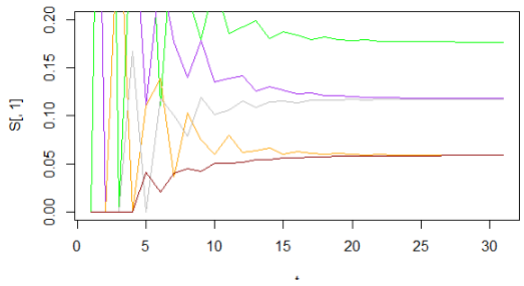
Un lien d'une page P_i vers une page P_j peut être vu comme un vote de P_i en faveur de P_j. Dans notre exemple, la structure du graphe permet de voir que la page P_5 peut être considérée comme la plus importante. Nous allons nous intéresser à la manière d'établir un classement des pages des plus pertinentes au moins pertinentes.

Modèle naïf

Pour une page P_j, nous pouvons définir une mesure de pertinence m_j correspondant à la somme des liens pointant vers P_j, $m_j = \sum_{i \to j} 1$.
En reprenant notre exemple, on remarque que dans ce cas, $m_1 = m_9 = 4$ et $m_5 = 3$ ce qui n'est pas en accord avec le fait que P_5 est la plus importante. Pour palier à ce problème, l'idée va être d'une part de pondérer les votes de la page P_i vers P_j par le nombre de liens émis de P_i puis de multiplier par la mesure d'importance de P_i. En effet, si beaucoup de pages importantes citent P_j alors P_j peut être considérée comme importante. On obtient alors la mesure d'importance de manière récursive $m_j = \sum_{i \to j} \frac{1}{l_i} m_i$ où l_i est le nombre de liens émis par P_i.
Cette nouvelle mesure d'importance peut s'écrire de manière matricielle grâce à la matrice A de taille $N * N$ telle que pour tout $i,j \in \{1,\ldots,N\}$, $a_{i,j} = \frac{1}{l_i}$ si $i \to j$ et $a_{i,j} = 0$ sinon. L'équation de la mesure d'importance devient alors $m = mA$.
On remarque que la matrice A vérifie pour tout $i,j \in \{1,\ldots,N\}$, $a_{i,j} \geq 0$ et $\sum_j a_{i,j} = 1$. A est alors une matrice de transition et notre modèle revient à

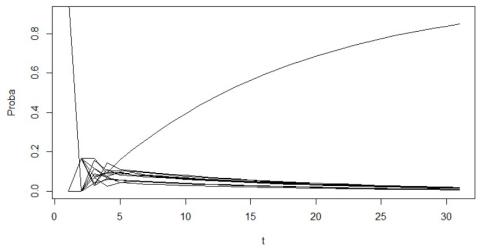
étudier le comportement d'une chaîne de Markov. La mesure d'importance ainsi définie nous donne les probabilités de se trouver sur les pages après un pas dans la marche aléatoire dans le graphe. L'information qu'il nous faut pour établir le classement est alors la probabilité après plusieurs pas dans cette marche aléatoire. En effet, si l'on obtient une grande probabilité de se trouver sur une des pages après avoir suivi plusieurs liens, on peut alors se dire que cette page est importante.
Nous avons codé en R cette méthode, appliquons maintenant notre algorithme au graphe ci-dessus:

```
[[2]]
[1] "le graphe ne contient aucun trou noir"

$`distribution initiale`
 [1] 0 0 0 0 0 0 0 1 0 0 0 0 0

$`distribution finale`
 [1] 0.11758772 0.05876707 0.05876707 0.05876707 0.17663471 0.05888864 0.11781017
 [8] 0.05888864 0.11758772 0.05876707 0.05876707 0.05876707
```



Nous définirons dans la suite ce que sont les trous noirs. La distribution initiale nous dit que nous partons de la page P_7. Nous voyons sur le graphe que les probabilités convergent pour t entre 20 et 25 et pour $t = 30$ on a la distribution finale dans laquelle la probabilité de se trouver sur la page P_5 est $m_5 = 0.18$. Et pour tout $i \in \{1, \ldots, 12\}$,$m_i \leq m_5 = 0.18$. P_5 est donc bien la page la plus importante de ce graphe.
Sur notre graphe exemple cette méthode fonctionne bien mais peut-on être sûr que cette méthode de classement marche tout le temps et aussi bien pour n'importe quel graphe ? Simulons un graphe aléatoirement de même taille $N = 12$ et regardons ce que ça donne.

```
$`les pages suivantes sont des trous noirs`
[1] 3

$`distribution initiale`
 [1] 0 0 0 0 0 0 0 0 0 0 1 0 0

$`distribution finale`
 [1] 0.010392981 0.016674771 0.847263376 0.009654241 0.016232955 0.017413640 0.019449586 0.009969390
 [9] 0.020353822 0.007588024 0.009741787 0.015265426
```

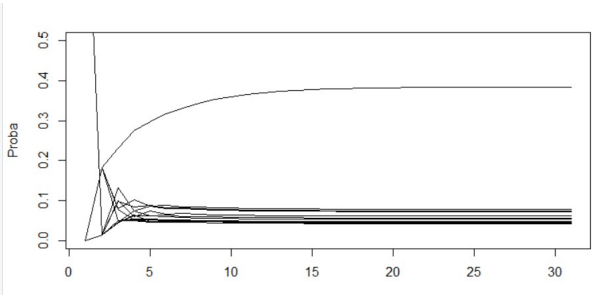


Pour ce nouvel exemple, on a toujours une convergence des probabilités cependant elles convergent toutes vers 0 sauf m_3 qui converge vers 1. On remarque également que P_3 est qualifiée de 'trou noir' c'est-à-dire que c'est une page sans issue, elle reçoit des liens mais elle n'en émet pas, elle se cite elle-même. Dans la matrice de transition A de cet exemple on a donc $a_{3,3} = 1$. On comprend alors pourquoi on observe ce comportement, la promenade aléatoire dans le graphe nous conduit forcément à un instant sur P_3 et comme nous ne pouvons pas sortir de cette page, nous restons bloqués indéfiniment dessus. Les trous noirs absorbent donc les probabilités.
Cette méthode n'est donc pas efficace pour les graphes contenant des pages sans issues, c'est le modèle PageRank de Google qui va nous donner la solution de ce problème.

Modèle PageRank

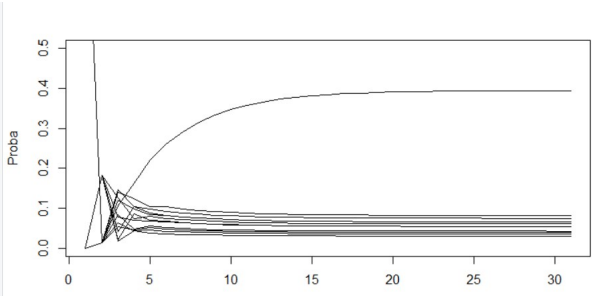
Pour ne pas être pris au piège par une page sans issue, l'astuce de Google va être d'introduire une probabilité c de téléportation avec laquelle on va abandonner la page sur laquelle on est et repartir sur une page du graphe choisie de manière équiprobable, et avec une probabilité $1 - c$, continuer normalement notre promenade aléatoire. La mesure d'importance devient donc: $m_j = \frac{c}{N} + (1 - c) \sum_{i \to j} \frac{1}{l_i} m_i$. On peut donc écrire cette mesure d'importance de manière matricielle avec M de taille $N * N$ telle que $M = cE + (1 - c)A$ avec E la matrice $N * N$ dont toutes les composantes valent $\frac{1}{N}$ et A la matrice de transition associée au graphe (on rappelle: pour tout i,$j \in \{1, \ldots, N\}$, $a_{i,j} = \frac{1}{l_i}$ si $i \to j$ et $a_{i,j} = 0$ sinon).
Le paramètre c doit être petit mais non nul, de nombreux articles fixent $c = 0.15$ nous allons donc en faire de même. Nous avons codé cette méthode, regardons ce qu'elle donne sur deux exemples de graphes générés aléatoirement.
Simulation 1:

```
$`les pages suivantes sont des trous noirs`
[1] 5

$`distribution initiale`
 [1] 0 0 0 0 0 0 0 1 0 0 0 0

$`distribution finale`
 [1] 0.04595547 0.07791638 0.04713239 0.04207495 0.38423868 0.07216316 0.04446092 0.06093960
 0.04255060 0.05586263 0.07391908
[12] 0.05278615
```



Simulation 2:

```
$`les pages suivantes sont des trous noirs`
[1] 10

$`distribution initiale`
 [1] 0 0 0 0 0 0 0 0 1 0 0 0 0

$`distribution finale`
 [1] 0.05434679 0.07464110 0.06100976 0.06584958 0.03016610 0.03931783 0.08204829
 [8] 0.03553450 0.04253175 0.39416420 0.05438478 0.06600532
```



Les deux simulations nous montrent que les probabilités convergent et que les pages sans issues n'absorbent plus totalement les probabilités.
Nous avons vu qu'on pouvait établir un classement des pages pour des graphes de petite taille mais est-ce toujours le cas pour n'importe quel graphe de n'importe quelle taille ? La réponse est évidement oui car Google classe des millions de pages pour chaque recherche. L'explication se trouve dans la théorie des chaînes de Markov. La recherche du classement des pages d'un graphe revient à l'étude du comportement d'une chaîne de Markov comme nous l'avons dit précédemment. La matrice de transition étant définie comme $M = cE + (1-c)A$ avec E la matrice $N * N$ dont toutes les composantes valent $\frac{1}{N}$ et A la matrice de transition du graphe, on a alors la chaîne de Markov associée à M irréductible et récurrente positive car à chaque pas on a la possibilité (c) de se téléporter sur n'importe quelle page. La chaîne admet alors une unique probabilité stationnaire m. De plus, la chaîne étant apériodique grâce à la téléportation, on a la convergence (en loi) vers cette probabilité stationnaire indépendamment de la distribution initiale.

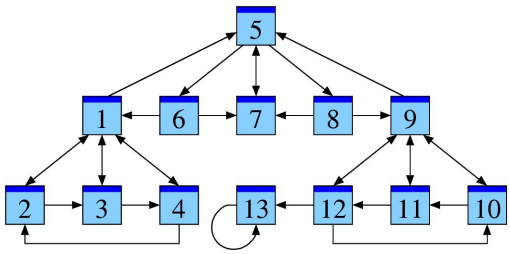
Du point de vue analyse, cela est équivalent au théorème du point fixe (qui nous sera utile dans la dernière partie) ci-dessous.

Théorème du point fixe. *: Considérons un graphe fini quelconque et fixons le paramètre c tel que* $0 < c \leq 1$*. Alors, l'équation* $m_j = \frac{c}{N} + (1-c)\sum_{i \to j} \frac{1}{l_i} m_i$ *admet une unique solution vérifiant* $\sum_{j \in E} m_j = 1$ *. Dans cette solution* $m_1, \ldots, m_n$ *sont tous strictement positifs. Pour toute distribution de probabilité initiale sur le graphe, le processus* $m_j = \frac{c}{N} + (1-c)\sum_{i \to j} \frac{1}{l_i} m_i$ *converge vers cette unique mesure stationnaire* m*.*
La convergence est au moins aussi rapide que celle de la suite géométrique $(1-c)^n$ *vers 0.*

Pour n'importe quel graphe de n'importe quel taille, notre classement des pages correspond donc à cette probabilité stationnaire m où pour tout i, m_i correspond à la mesure d'importance de la page P_i.

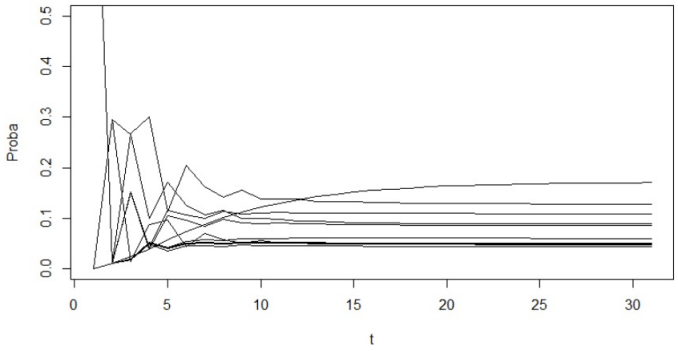
Optimisation de notre algorithme

Nous avons vu dans la partie précédente que notre modèle basé sur l'algorithme PageRank de Google donne de meilleurs résultats que l'approche naïve mais nous allons voir qu'il n'est pas totalement satisfaisant. En effet, pour un graphe contenant un ou des trous noirs, l'astuce de la possible téléportation pendant la marche aléatoire permet d'éviter que les probabilités de la mesure stationnaire ne soient 0 pour toutes les pages autres que les trous noirs. Reprenons le graphe qui nous sert d'exemple, ajoutons lui une page sans issue et regardons ce que devient la mesure stationnaire.



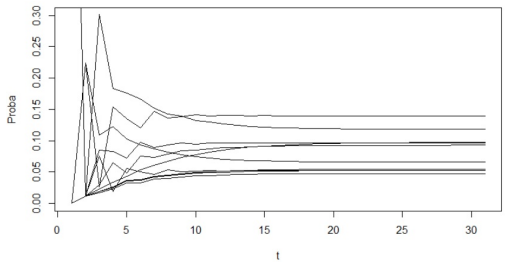
En appliquant notre algorithme à ce graphe, on obtient les résultats et le graphe suivants:

```
$`les pages suivantes sont des trous noirs`
[1] 13

$`distribution initiale`
 [1] 0 0 0 0 1 0 0 0 0 0 0 0 0

$`distribution finale`
 [1] 0.10873664 0.06027365 0.06027365 0.06027365 0.12802005 0.04782279 0.08848437 0.04782279 0.08530798
[10] 0.04390180 0.04833288 0.05021652 0.17053326
```

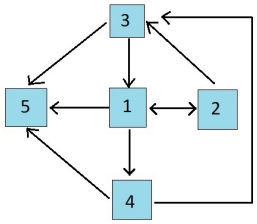


La mesure stationnaire est atteinte à environ $t = 25$. La page 13 semble être la plus importante avec une probabilité 0.17 de s'y trouver, vient ensuite la page 5 avec une probabilité 0.13. Cependant la page 13 ,n'ayant que le vote de la page 12, ne peut pas du tout être considérée comme une page importante, c'est le fait qu'elle soit sans issue qui lui permet de toujours finir en tête du classement. Nous devons alors réfléchir à un moyen de faire diminuer le poids des trous noirs dans la mesure stationnaire.
L'idée va être de réutiliser l'astuce de la téléportation. Nous allons toujours pouvoir nous téléporter à chaque pas de la marche aléatoire comme c'était le cas jusqu'à présent, et nous allons y ajouter une autre possibilité de téléportation si l'on se trouve bloqué sur une page sans issue.
Soit une page i sans issue. On introduit alors une probabilité c' telle que si l'on se trouve sur la page i, on ait une probabilité $1-c'$ de rester sur i et une probabilité $\frac{c'}{N-1}$ de se téléporter sur n'importe quelle page $j \neq i$. Cela se traduit par la modification de la matrice de transition A associée au graphe sur lequel on travaille. Il faut alors identifier les trous noirs de A et modifier les probabilités de transition telles que pour tout i trou noir, $A_{i,i} = 1 - c'$ et pour tout $j \neq i$, $A_{i,j} = \frac{c'}{N-1}$.
Il ne reste maintenant qu'à choisir c'. Afin de ne pas fausser nos résultats précédents il faut choisir c' petit et proche de 0, on peut donc fixer $c' = c = 0.15$. Appliquons ce nouvel algorithme à notre graphe.

```
$`les pages suivantes sont des trous noirs`
[1] 13

$`distribution initiale`
 [1] 1 0 0 0 0 0 0 0 0 0 0 0 0

$`distribution finale`
 [1] 0.11835789 0.06561577 0.06561577 0.06561577 0.13928224 0.05203661 0.09626885 0.05203661 0.09283171
[10] 0.04778112 0.05260300 0.05465213 0.09730253
```

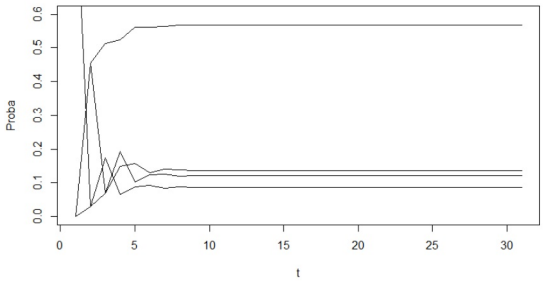


La mesure stationnaire est atteinte à environ $t = 20$. La page 5 arrive en tête avec une probabilité de 0.14 suivie de la page 1 avec une probabilité 0.12. Ce nouvel algorithme est alors plus satisfaisant que le précédent, les pages sans issues absorbent moins les probabilités de la mesure stationnaire, le classement des pages peut donc être considéré comme plus fiable.
Nous pouvons penser que cette astuce est un peu lourde et qu'on aurait pu seulement ne pas tenir compte des trous noirs et donc ne pas regarder leur poids dans le classement final mais nous pouvons imaginer un graphe dans lequel beaucoup de pages votent pour une page i sans issue. Dans ce cas, cette page i est une page pertinente bien qu'elle soit sans issue. Par exemple:



Ici la page 5, sans issue, est clairement la plus importante. Notre algorithme donne alors:

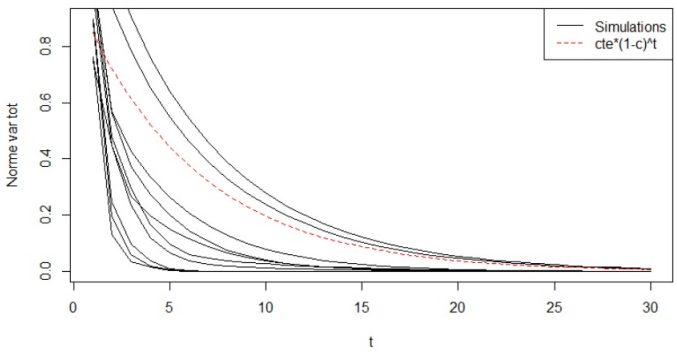
```
$`les pages suivantes sont des trous noirs`
[1] 5

$`distribution initiale`
[1] 0 1 0 0 0

$`distribution finale`
[1] 0.13681988 0.08685626 0.12191845 0.08685626 0.56754914
```



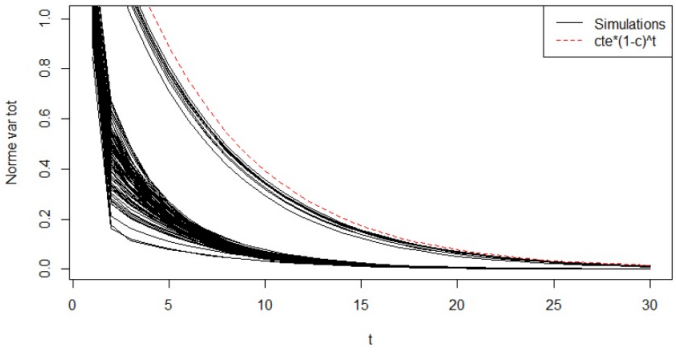
La page 5 arrive largement en tête dans la mesure stationnaire, ce qui est pertinent.
Nous pouvons maintenant nous intéresser à la vitesse de convergence de notre modèle.

Vitesse de convergence

Nous avons pu remarquer dans les graphes des exemples précédents que la mesure stationnaire est atteinte assez rapidement, entre $t = 20$ et $t = 25$. C'est le théorème du point fixe vu précédemment qui va garantir cette bonne vitesse de convergence.
En effet, le théorème affirme que la vitesse de convergence de notre modèle est au moins en $(1-c)^t$ c'est-à-dire que pour K constante on a:
$\|m_0 A_c^t - m_\infty\|_{vt} \leq K(1-c)^t$
avec $A_c = cE + (1-c)A$ la matrice de transition du modèle Google Ranking, E étant la matrice $N * N$ dont tous les coefficients valent $\frac{1}{N}$ (équiprobabilité de la téléportation sur n'importe quelle page), m_0 la distribution initiale, m_∞ la mesure stationnaire, et $\|m_1 - m_2\|_{vt}$ la norme en variation totale définie par $\|m_1 - m_2\|_{vt} = \sum_{k=1}^{N} |m_1(k) - m_2(k)|$. Afin de vérifier la vitesse de notre modèle nous allons donc tracer le graphe de $\|m_0 A_c^t - m_\infty\|_{vt}$ en fonction de t en prenant comme approximation de la mesure stationnaire $m_0 A_c^t$ avec t grand, on choisit $m_\infty = m_0 A_c^{100}$. On tracera également sur le même graphe $K(1-c)^t$ en fonction de t pour différentes valeurs de K.
Simulons 10 graphes aléatoires de taille $N = 12$ et fixons $K = 1$:



On remarque que la plupart des graphes convergent plus rapidement que $(1-c)^t$ mais certains convergent plus lentement, fixons alors $K = 2$ et lançons la simulation de 100 graphes de taille $N = 50$:



Les 100 graphes convergent tous plus rapidement que $2(1-c)^t$, nous avons donc vérifié que notre modèle est bien en accord avec le théorème du point fixe.

Conclusion

Le classement des pages web revient à la recherche de la probabilité invariante d'une chaîne de Markov, l'existence et l'unicité de cette probabilité étant assurées par la théorie des chaîne de Markov. De plus, le théorème du point fixe garantit une bonne vitesse de convergence ce qui permet à Google de traiter des millions de pages à chaque recherche.

Bibliographie

Article de Michael Eisermann (2009): 'Comment Google classe les pages web'
http://images.math.cnrs.fr/Comment-Google-classe-les-pages-web

Annexe: code R

```
graphe = function(N){
  L = matrix(rbinom(N*N,1,prob = 0.5), ncol = N, nrow = N)
  for (i in 1:N){
    L[i,i] = rbinom(1,1,prob = 0.1)
  }
  for (i in 1:N){
    if (L[i,i]==1){
      for (k in 1:N){
        L[i,k] = 0
      }
      L[i,i]=1
    }
  }
  return(L)
}


#cette fonction nous renvoie notre structure en graphe avec les liens entre les pages, et
#avec possibilit  de tomber sur une page sans issue


ponderation = function(A){
  N = length(A[1,])
  for (i in 1:N){
    A[i,] = A[i,]/sum(A[i,])
  }
  return(A)
}


#cette fonction nous renvoie une matrice a_i,j dont les coef correspondent
#   1/(nombre de liens  mis  de la page i)

equi = function(N){
  matrix(rep(1/N,N),ncol = N, nrow = 1)
}

#vecteur d'equi probabilit  entre les pages

vect_init = function(N){              #distribution au temps 0
  i = sample(1:N,1)
  v = rep(0,N)
  v[i] = 1
  return(v)
}



transition_gr = function(N,A,x,cte){          #transition du Google Ranking
  return(cte*equi(N)+(1-cte)*(x %*% A))
}



transition = function(A,x){            #transition basique
```

```
  return(x %*% A)
}


convergence = function(A,init,t){          #promenade aleatoire basique
  for (k in 1:t){
    X = transition(A,init)
    init = X
  }
  return(X)
}


convergence_gr = function(N,A,init,cte,t){          #promenade Google Ranking
  for (k in 1:t){
    X = transition_gr(N,A,init,cte)
    init = X
  }
  return(X)
}


#######################################################
#CAS BASIQUE#


cas_basique = function(N,A,init,n,ylim){
  t = seq(1,n+1,1)
  S = matrix(rep(0,(n+1)*N),ncol = N)
  S[1,] = init
  for (k in 1:n){
    for (i in 1:N){
      S[k+1,i] = convergence(A,init,k)[i]
    }
  }
  plot(t,S[,1],type = "l",ylim =c(0,ylim), ylab = "Proba")
  for (k in 2:N){
    lines(t,S[,k])
  }
  trou_noir = c()
  for (i in 1:N){
    if (A[i,i] == 1){
      trou_noir = append(trou_noir,c(i))
    }
  }
  if (is.null(trou_noir)){
    liste = list("matrice_de_transition" = A, "le_graphe_ne_contient_aucun_trou_noir" ,
                 "distribution_initiale"= init, "distribution_finale" = S[n+1,])
    return(liste)
  }
  else{
    liste = list("matrice_de_transition" = A,
                 "les_pages_suivantes_sont_des_trous_noirs" = trou_noir,
                 "distribution_initiale"= init, "distribution_finale" = S[n+1,])
```

```
    return(liste)
  }

}


cas_basique(12,ponderation(graphe(12)),vect_init(12),30,ylim = 0.9)


cas_basique(100,ponderation(graphe(100)),vect_init(100),30,ylim = 0.5)


######################################################################
#CAS GOOGLE RANKING#


cas_gr = function(N,A,init,cte,n,ylim){
  t = seq(1,n+1,1)
  S = matrix(rep(0,(n+1)*N),ncol = N)
  S[1,] = init
  for (k in 1:n){
    for (i in 1:N){
      S[k+1,i] = convergence_gr(N,A,init,cte,k)[i]
    }
  }
  plot(t,S[,1],type = "l",ylim =c(0,ylim), ylab = "Proba")
  for (k in 2:N){
    lines(t,S[,k])
  }
  trou_noir = c()
  for (i in 1:N){
    if (A[i,i] == 1){
      trou_noir = append(trou_noir,c(i))
    }
  }
  if (is.null(trou_noir)){
    liste = list("matrice de transition" = A, "le graphe ne contient aucun trou noir" ,
                 "distribution initiale"= init, "distribution finale" = S[n+1,])
    return(liste)
  }
  else{
    liste = list("matrice de transition" = A,
                 "les pages suivantes sont des trous noirs" = trou_noir,
                 "distribution initiale"= init, "distribution finale" = S[n+1,])
    return(liste)
  }

}


cas_gr(12,ponderation(graphe(12)),vect_init(12),0.15,30,ylim = 0.5)
```

```
##########################################

#EXEMPLE DU SITE CAS BASIQUE#

N =12
A = matrix(c(0,1,1,1,1,0,0,0,0,0,0,0, 1,0,1,0,0,0,0,0,0,0,0,0, 1,0,0,1,0,0,0,0,0,0,0,0,
             1,1,0,0,0,0,0,0,0,0,0,0, 0,0,0,0,0,1,1,1,0,0,0,0, 1,0,0,0,0,0,1,0,0,0,0,0,
             0,0,0,0,1,0,0,0,0,0,0,0, 0,0,0,0,0,0,1,0,1,0,0,0, 0,0,0,0,1,0,0,0,0,1,1,1,
             0,0,0,0,0,0,0,0,1,0,1,0, 0,0,0,0,0,0,0,0,1,0,0,1, 0,0,0,0,0,0,0,0,1,1,0,0),
           ncol = 12, byrow = TRUE)
A
B = ponderation(A)
B
init = c(0,0,0,0,0,0,1,0,0,0,0,0)
init

t = seq(1,31,1)
S = matrix(rep(0,372),ncol = 12)
S[1,] = init
for (k in 1:30){
  for (i in 1:12){
    S[k+1,i] = convergence(B,init,k)[i]
  }
}

plot(t,S[,1],type = "l",col="blue", ylim = c(0,0.2))
for (k in 1:12){
  lines(t,S[,k])
}
plot(t,S[,1],type = "l",col="blue", ylim = c(0,0.2))
lines(t,S[,2], col="red")
lines(t,S[,3], col="black")
lines(t,S[,4], col="yellow")
lines(t,S[,5], col="green")
lines(t,S[,6], col="cyan")
lines(t,S[,7], col="purple")
lines(t,S[,8], col="orange")
lines(t,S[,9], col="grey")
lines(t,S[,10], col="brown")
lines(t,S[,11], col="darkgreen")
lines(t,S[,12], col="darkred")


cas_basique(12,B,init,30,ylim = 0.5)



##########################################

#EXEMPLE DU SITE CAS GR#

N =12
cte=0.15
A = matrix(c(0,1,1,1,1,0,0,0,0,0,0,0, 1,0,1,0,0,0,0,0,0,0,0,0, 1,0,0,1,0,0,0,0,0,0,0,0,
             1,1,0,0,0,0,0,0,0,0,0,0, 0,0,0,0,0,1,1,1,0,0,0,0, 1,0,0,0,0,0,1,0,0,0,0,0,
             0,0,0,0,1,0,0,0,0,0,0,0, 0,0,0,0,0,0,1,0,1,0,0,0, 0,0,0,0,1,0,0,0,0,1,1,1,
```

```
                0,0,0,0,0,0,0,0,1,0,1,0, 0,0,0,0,0,0,0,0,1,0,0,1, 0,0,0,0,0,0,0,0,1,1,0,0),
              ncol = 12, byrow = TRUE)
A
B = ponderation(A)
B
init = c(1,0,0,0,0,0,0,0,0,0,0,0)
init

t = seq(1,31,1)
S = matrix(rep(0,372),ncol = 12)
S[1,] = init
for (k in 1:30){
  for (i in 1:12){
    S[k+1,i] = convergence_gr(N,B,init,cte,k)[i]
  }
}

plot(t,S[,1],type = "l",col="blue", ylim = c(0,0.2))
for (k in 1:12){
  lines(t,S[,k])
}
plot(t,S[,1],type = "l",col="blue", ylim = c(0,0.2))
lines(t,S[,2], col="red")
lines(t,S[,3], col="black")
lines(t,S[,4], col="yellow")
lines(t,S[,5], col="green")
lines(t,S[,6], col="cyan")
lines(t,S[,7], col="purple")
lines(t,S[,8], col="orange")
lines(t,S[,9], col="grey")
lines(t,S[,10], col="brown")
lines(t,S[,11], col="darkgreen")
lines(t,S[,12], col="darkred")


cas_gr(12,B,init,0.15,30,ylim = 0.5)



################################################################################

#VITESSE DE CONVERGENCE#


vitesse = function(N,n,cte,nbr_test,cte_vitesse){
  S = matrix(rep(0,n*nbr_test),ncol = nbr_test)
  for (k in 1:nbr_test){
    A = ponderation(graphe(N))
    init = vect_init(N)
    mes_statio = convergence_gr(N,A,init,cte,t = 100)
    for (i in 1:n){
      norme = 0
      for (l in 1:N){
        norme = norme + abs(convergence_gr(N,A,init,cte,i)[l]-mes_statio[l])
      }
```

```
      S[i,k] = norme
    }
  }
  t = seq(1,n,1)
  if (nbr_test==1){
    plot(t,S[,1],type = "l",ylab = "Norme var tot", col = "black")
    y = (1-cte)^t
    lines(t,y,type = "l",col = "red", lty = 2)
    legend("topright",legend = c("simulation","(1-c)^t"), col = c("black","red"), lty = 1:2)
  }
  else{
    plot(t,S[,1],type = "l", ylab = "Norme var tot")
    for (k in 2:nbr_test){
      lines(t,S[,k])
    }
    y = cte_vitesse*(1-cte)^t
    lines(t,y,type = "l",col = "red", lty=2)
    legend("topright",legend = c("Simulations","cte*(1-c)^t"), col = c("black","red"),
           lty = 1:2)
  }
}

vitesse(50,30,0.15,100,2)

##############################################################################

#CAS OPTIMAL#


cas_teleport = function(N,A,const,n,init,ylim){
  trou_noir = c()
  for (i in 1:ncol(A)){
    if (A[i,i] == 1){
      A[i,] = rep(const/(ncol(A)-1),ncol(A))
      A[i,i] = 1-const
      trou_noir = append(trou_noir,c(i))
    }
  }
  t = seq(1,n+1,1)
  S = matrix(rep(0,(n+1)*N),ncol = N)
  S[1,] = init
  for (k in 1:n){
    for (i in 1:N){
      S[k+1,i] = convergence_gr(N,A,init,0.15,k)[i]
    }
  }
  plot(t,S[,1],type = "l",ylim =c(0,ylim), ylab = "Proba")
  for (k in 2:N){
    lines(t,S[,k])
  }
  if (is.null(trou_noir)){
    liste = list("matrice de transition" = A, "le graphe ne contient aucun trou noir" ,
                 "distribution initiale"= init, "distribution finale" = S[n+1,])
    return(liste)
  }
  else{
    liste = list("matrice de transition" = A,
```

```
                "les pages suivantes sont des trous noirs" = trou_noir,
                "distribution initiale"= init, "distribution finale" = S[n+1,])
    return(liste)
  }
}


cas_teleport(12,ponderation(graphe(12)),0.7,30,vect_init(12),ylim = 0.2)


##############################################################################################
#CAS OPTIMAL EX CNRS#

A = matrix(c(0,1,1,1,1,0,0,0,0,0,0,0,0, 1,0,1,0,0,0,0,0,0,0,0,0, 1,0,0,1,0,0,0,0,0,0,0,0,
             1,1,0,0,0,0,0,0,0,0,0,0,0, 0,0,0,0,0,1,1,1,0,0,0,0,0, 1,0,0,0,0,0,1,0,0,0,0,0,
             0,0,0,0,1,0,0,0,0,0,0,0,0, 0,0,0,0,0,0,1,0,1,0,0,0,0, 0,0,0,0,1,0,0,0,0,1,1,1,0,
             0,0,0,0,0,0,0,0,1,0,1,0,0, 0,0,0,0,0,0,0,0,1,0,0,1,0, 0,0,0,0,0,0,0,0,0,1,1,0,0,1,
             0,0,0,0,0,0,0,0,0,0,0,0,1), ncol = 13, byrow = TRUE)
A
B = ponderation(A)
B


cas_gr(13,B,vect_init(13),0.15,30,ylim = 0.5)
cas_basique(13,B,vect_init(13),30,ylim = 0.5)
cas_teleport(13,B,0.5,30,vect_init(13),ylim = 0.5)

##############################################################################################

vitesse_teleport = function(N,n,cte,nbr_test,cte_vitesse,const){
  S = matrix(rep(0,n*nbr_test),ncol = nbr_test)
  for (k in 1:nbr_test){
    A = ponderation(graphe(N))
    for (i in 1:ncol(A)){
      if (A[i,i] == 1){
        A[i,] = rep(const/(ncol(A)-1),ncol(A))
        A[i,i] = 1-const
      }
    }
    init = vect_init(N)
    mes_statio = convergence_gr(N,A,init,cte,t = 100)
    for (i in 1:n){
      norme = 0
      for (l in 1:N){
        norme = norme + abs(convergence_gr(N,A,init,cte,i)[l]-mes_statio[l])
      }
      S[i,k] = norme
    }
  }
  t = seq(1,n,1)
  if (nbr_test==1){
    plot(t,S[,1],type = "l",ylab = "Norme var tot", col = "black")
    y = (1-cte)^t
    lines(t,y,type = "l",col = "red", lty = 2)
    legend("topright",legend = c("simulation","(1-c)^t"), col = c("black","red"), lty = 1:2)
```

```
  }
  else{
    plot(t,S[,1],type = "l", ylab = "Norme var tot")
    for (k in 2:nbr_test){
      lines(t,S[,k])
    }
    y = cte_vitesse*(1-cte)^t
    lines(t,y,type = "l",col = "red", lty=2)
    legend("topright",legend = c("Simulations","cte*(1-c)^t"), col = c("black","red"),
           lty = 1:2)
  }
}

vitesse_teleport(20,30,0.15,10,1,0.15)
}
```