E C G

Correction DS n°4

Option économique

MATHEMATIQUES

20 Janvier 2024

Exercice 1

1.
```
def appartient(L,x):
    for i in range(len(L)):
        if x==L[i]:
            return('True')
    return("False")
```

2.
```
def comptage(L,x):
    c=0
    for i in range(len(L)):
        if x==L[i]:
            c=c+1
    return(c)
```

Exercice 2

1.

```
def PythonX2(n,a,b):
    N=0
    for i in range(1,n+1):
        if rd.rand()<b/(a+b):
            a=a+1
            b=b-1
            N=N+1
    return(N)
```

2.

```
def Y(a,b):
    N=0
    while b>0:
```

```
        if rd.rand()<b/(a+b):
            a=a+1
            b=b-1
        N=N+1
    return(N)
```

Exercice 3

1. cf. Cours d'informatique.
2.
```
def mat(n):
    A=np.array([[1/3,0,-2/3,0,-2/3],[1/3,0,1/3,1,1/3],[-2/3,0,1/3,0,-2/3]
    ,[1/3,1,1/3,0,1/3],[-2/3,0,-2/3,0,1/3]])
    P=np.array([[-1,1,1,1,1],[1,-1,1,1,1],[1,1,-1,1,1],
    [1,1,1,-1,1],[1,1,1,1,-1]])
    Q=al.inv(P)
    return(Q,P.dot(A).dot(Q),al.matrix_power(A,n))
```

Exercice 4

1.
```
def f(x):
    return(np.sqrt(4+3*x))

x=np.linspace(-1.3,10,10000)
y=f(x)
plt.plot(x,y,x,x)
plt.show()
```

2.
```
def U(n):
    u=1
    for i in range(2,n+1):
        u=f(u)
    return(u)
```

3. On raisonne par récurrence.

   Initialisation :
   Pour $n = 1$, on a d'après l'énoncé $u_1 = 1$ et $u_2 = \sqrt{4 + 3u_1} = \sqrt{7} < \sqrt{16}$, donc
   $\boxed{u_1 < u_2 < 4}$ et l'initialisation est vérifiée.
   Hérédité :
   Soit $n \in \mathbb{N}^*$. On suppose que $u_n < u_{n+1} < 4$. Montrons que $u_{n+1} < u_{n+2} < 4$.

   Comme d'après l'hypothèse de récurrence, $u_n < u_{n+1} < 4$, alors

   $$4 + 3u_n < 4 + 3u_{n+1} < 4 + 3 \times 4$$

   et par stricte croissance de la fonction racine carrée :

   $$\sqrt{4 + 3u_n} < \sqrt{4 + 3u_{n+1}} < \sqrt{16}$$

   c'est à dire

   $$\boxed{u_{n+1} < u_{n+2} < 4}$$

Conclusion :

$\boxed{\forall n \in \mathbb{N}^*,\ u_n < u_{n+1} < 4}$

Ceci montre que la suite (u_n) est croissance et majorée, donc d'après le théorème de la limite monotone, $\boxed{\text{elle converge vers un réel } \ell \leq 4.}$

Par ailleurs, comme $x \mapsto \sqrt{4+3x}$ est continue sur $]-4/3; 4]$, d'après le théorème du point fixe, ℓ est solution de l'équation

$$x = \sqrt{4+3x}$$

On a donc $x^2 = 4 + 3x$ ou encore $x^2 - 3x - 4 = 0$. Ceci est trinôme dont les deux racines sont 4 et -1. Or comme (u_n) démarre de 1 et est croissante, $\boxed{\text{elle converge vers 4.}}$

4.
```
u=1
n=1
while abs(u-4)>10**(-4):
    u=f(u)
    n=n+1
print(n)
```

Exercice n°5

Partie 1

1. (a) **Initialisation :** Par définition F_1 et $F_2 = F_1 + F_0 = 1$ sont tous les deux strictement positifs.

   Hérédité : Soit n entier naturel non nul, supposons que F_n et F_{n+1} sont strictement positifs, alors F_{n+2} est la somme de deux réels strictement positifs, donc

   $$F_{n+2} > 0$$

   Conclusion : Par récurrence

   $$\boxed{\text{Pour tout entier naturel } n,\ F_n > 0}$$

   (b) Soit n un entier plus grand que 2, alors

   $$F_{n+1} = F_n + F_{n-1}$$

   donc

   $$F_{n+1} - F_n = F_{n-1}$$

   donc, en utilisant la question précédente, et comme $n - 1 > 0$.

   $$\boxed{\text{La suite } (F_n)_{n\geqslant 0} \text{ est strictement croissante à partir du rang 2 .}}$$

   (c) On reconnaît une suite récurrente linéaire d'ordre 2 dont l'équation caractéristique est

   $$(E) \qquad X^2 = X + 1$$

   qui à pour solution

   $$x_1 = \frac{1-\sqrt{5}}{2} \qquad x_2 = \frac{1+\sqrt{5}}{2}$$

   D'après le théorème du cours il existe deux réels, α et β tels que

   $$\forall n \in \mathbb{N} \qquad F_n = \alpha x_1^n + \beta x_2^n$$

En considérant les deux premiers termes

$$\begin{cases} F_0 &= 0 \\ F_1 &= 1 \end{cases} \Leftrightarrow \begin{cases} \alpha+\beta &= 0 \\ \alpha x_1+\beta x_2 & \end{cases}$$

$$\Leftrightarrow \begin{cases} \alpha+\beta &= 0 \\ \alpha x_1+\beta x_2 &= 1 \end{cases}$$

$$\Leftrightarrow \begin{cases} \alpha+\beta &= 0 \\ \alpha\left(x_1-\beta x_2\right) &= 1 \end{cases}$$

$$\Leftrightarrow \begin{cases} \alpha+\beta &= 0 \\ -\alpha\sqrt{5} &= 1 \end{cases} \begin{cases} \alpha = -\dfrac{\sqrt{5}}{5} \\ \beta = \dfrac{\sqrt{5}}{5} \end{cases}$$

$$\boxed{\text{Pour } n \text{ entier naturel } F_n = -\frac{\sqrt{5}}{5}\left(\frac{1-\sqrt{5}}{2}\right)^n + \frac{\sqrt{5}}{5}\left(\frac{1+\sqrt{5}}{2}\right)^n}$$

(d) On remarque que

$$\sqrt{4} < \sqrt{5} < \sqrt{9}$$

donc

$$-1 < x_1 < -\frac{1}{2} \qquad \frac{3}{2} < x_2 < 2$$

donc

$$\lim_{n\to+\infty} x_1^n = 0 \qquad \lim_{n\to+\infty} x_2^n = +\infty$$

$$\boxed{\lim_{n\to+\infty} F_n = +\infty.}$$

2.
```
def fibo(n):
    'renvoie la liste des  nombres de fibonacci F_0,...f_n'
    L=[0,1]
    for i in range(2,n+1):
        L.append(L[i-1]+L[i-2])
    return L
```

3.
```
def recherche(x,L):
    i=1
    while i<len(L) and L[i]<=x:
        i=i+1
    return L[i-1]
```

Partie 2

On s'intéresse dans cette partie au théorème suivant, appelé Théorème de Zeckendorf Pour tout entier naturel non nul n, il existe un unique entier k et un unique k-uplet d'entiers $(c_1, \ldots, c_k)$, verifiant :

$\star$ $c_1 \geqslant 2$

$\star\star$ pour tout i appartenant à $[\![1, k-1]\!], c_i + 1 < c_{i+1}$, tels que :

$$n = \sum_{i=1}^{k} F_{c_i}$$

Cette décomposition s'appelle la décomposition de Zeckendorf du nombre n. Par exemple, $n = 4$ se décompose en : $4 = 1 + 3 = F_2 + F_4$. Donc $k = 2$ et $(c_1, c_2) = (2, 4)$. Par ailleurs, $n = 17$ se décompose en : $17 = 1 + 3 + 13 = F_2 + F_4 + F_7$. Donc $k = 3$ et $(c_1, c_2, c_3) = (2, 4, 7)$

4. On rappelle que la liste des premiers termes de la suite $(F_n)_{n\geqslant 0}$ a été donnée dans la question 2.
   (a) La première décomposition vérifie bien le point $\star$ mais pas le point $\star\star$. La deuxième décomposition vérifie les deux points.

   $$\boxed{6 = F_2 + F_5}$$

   (b) $\boxed{35 = 1 + 34 = F_2 + F_9 \text{ qui vérifie bien les deux conditions}}$

   (c) $\boxed{130 = 2 + 5 + 34 + 89 = F_3 + F_5 + F_9 + F_11}$

5. Soit $n \in \mathbb{N}^*$.
   (a) Application directe de la définition de $\lim\limits_{n\to+\infty} F_n = +\infty$

   (b) Comme $F_2 = 1$ et $1 \leqslant n$

   $$\boxed{2 \in A_n}$$

   En reprenant les notations de la question précédentes

   $$\forall i \geqslant J \qquad F_i \notin A_n$$

   donc

   $$A_n \subset [\![1;\ J]\!]$$

   ce qui démontre

   $$\boxed{A_n \text{ contient au plus } J \text{ éléments}}$$

   (c) j est le maximum de J donc appartient à J et donc

   $$F_j \leqslant n$$

   $j + 1 > j$ donc $j + 1$ ne peut pas appartenir à J car sinon j ne serait pas le maximum de J. donc $F_{j+1} > n$

   $$\boxed{F_j \leqslant n < F_{j+1}}$$

   (d) D'après la question précédente $n < F_{j+1}$ or comme $j \leqslant 1$, $F_{j+1} = F_j + F_{j-1}$ donc

   $$n < F_j + F_{j-1}$$

   donc

   $$\boxed{n - F_j < F_{j-1}}$$

   (e) On commence par récupérer la liste des nombres de Fibonacci inférieur à n.
   Puis on prend le plus grand nombre de Fibonacci inférieur à n. que l'on met dans la liste T
   On soustrait ce nombre à n et on recommence, c'est àa dire que l'on cherche le plus grand nombre de Fibonacci inférieur à n-f.
   On recommence et on s'arrête quand le nombre est épuisé.
   D'après ce qui précède T est la décomposition de Zeckendorf de n (donnée dans l'ordre décroissant)

6. On commence par choisir le plus grand nombre de Fibonacci inférieur à l'entier n que l'on veut décomposer donc on optimise chaque état sans jamais revenir en arrière.

Problème

Préliminaires

1. La matrice est symétrique, les coefficients diagonaux sont tous nuls et les autres coefficients ne peuvent avoir pour valeur que 0 ou 1.
2. Il suffit d'écrire

   ```
   np.zeros(n,n)
   ```

3. ```
   def verification(M):
       '''verifie si la matrice M supposé carrée est bien symétrique'''
       n=len(M)
       for i in range(n):
           for j in range(i+1,n):
               if M[i][j]!=M[j][i]:
                   return False
       return True
   ```

   avec des boucles `while` et un drapeau.

   ```
   def verification(M):
       '''verifie si la matrice M supposée carrée est bien symétrique'''
       n=len(M)
       i=0
       drapeau=True
       while i<n and drapeau:
           j=0
           while j<n and drapeau:
               drapeau =(M[i][j]==M[j][i])## == et = sont  des opérateurs distincts!
               j=j+1
           i=i+1

       return drapeau
   ```

4. - **Initialisation :**
     Pour $n = 2, \sum_{k=1}^{n-1} k = \sum_{k=1}^{2-1} k = 2$ et $\frac{2 \times (2-1)}{2} = 1$
   - **Hérédité :** Pour $n \in \mathbb{N}^* \setminus \{1\}$ supposons que

$$\sum_{k=1}^{n-1} k = \frac{n(n-1)}{2}$$
   ```

alors

$$\sum_{k=1}^{n+1-1} k = \sum_{k=1}^{n-1} k + n$$

$$\begin{aligned}
&= \frac{n(n-1)}{2} + n \\
&= \frac{n^2 - n + 2n)}{2} \\
&= \frac{n^2 + n)}{2} \\
&= \frac{(n+1)n)}{2} \\
&= \frac{(n+1)(n+1-1))}{2}
\end{aligned}$$

Da'près le principe de récurrence

$$\boxed{n \in \mathbb{N}^* \setminus \{1\}, \sum_{k=1}^{n-1} k = \frac{n(n-1)}{2}}$$

5. Un graphe peut avoir 0 arête.

Une arête non orienté est définie par ses deux sommets. Il y a $\binom{n}{2} = \frac{n(n-1)}{2}$ façons de choisir une paire de deux sommets (sans répétition et sans ordre).

$$\boxed{\text{Le nombre d'arêtes maximal est } \binom{n}{2} = \frac{n(n-1)}{2}}$$

6. Pour déterminer un graphe il suffit de choisir les coefficients de la matrices d'adjacence. Comme la matrice doit être symétrique à diagonale nulle il faut chosir les coefficients $M_{i,j}$ pour $0 \leqslant i < j \leqslant n-1$. Il y a donc $n-1$ coefficients sur sur la première ligne $n-2$ coefficients sur la deuxième ligne,..., 1 coefficient sur la ligne $n-2$. Au total, il y a $n-1+n-2+\ldots 1 = \sum_{k=1}^{n-1} k = \frac{n(n-1)}{2}$ On retrouve le nombre d'arêtes possibles d'un graphe.
Chaque coefficient à deux valeurs possibles 0 ou 1. Le nombre de graphe possibles est donc

$$\underbrace{2 \times 2 \times \cdots \times 2}_{\frac{n(n-1)}{2} \text{ fois}}$$

$$\boxed{\text{Le nombre graphe est } \binom{n}{2}^{\frac{n(n-1)}{2}}}$$

7.
```
def aretes(M):
    '''nombre d'aretes d'un graphe non orienté'''
    nb=0
    n=len(M)
    for i in range(n):
        for j in range(n):
            nb=nb+M[i][j]
    return nb//2
```

Dans toute la suite n est un entier naturel plus grand que 2 fixé, qui représente le nombre de sommet des graphes étudiés.

Graphe binomiale

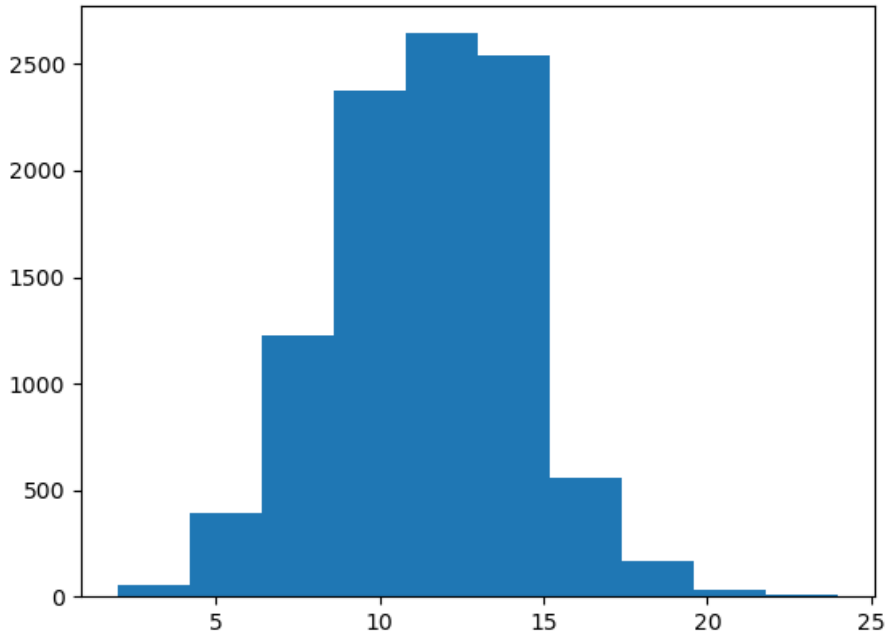
8.
```
def graphe(n,p):
    '''retourne la matrice d'adjacence d'un graphe aléatoire à n sommets'''
    M=np.zeros(n,n)
    for i in range(n):
        for j in range(i+1,n):
            if rd.random()<p:
                M[i][j]=1
                M[j][i]=1
    return M
```

9. X compte le nombre de succès "l'arête (i,j) est présente" lors de la répétition de $\binom{n}{2}$ expérience de Bernoulli indépendantes de probabilité de succès p

$$\boxed{X \hookrightarrow \mathcal{B}(n,p),\ E(X)=np,\ V(X)=np(1-p).}$$

10. On construit des graphes à 5 sommets ou la probabilité de présence de chaque arête est $\frac{1}{2}$. Le nombre d'arête suit donc la loi $\mathcal{B}(10, 0.5)$
POur le vérifier on construit $N = 10000$ graphes, et on compte leur arètes, l'histogramme de la distribution du nombre d'arêtes ressemble bien à celui d'un variable aléatoire suivante cette loi. Il est notemment symétrique autour de 5 la moyenne.

11. Cette fois ci l'histogramme de la distribution du nombre d'arètes doit ressembler à celui de la loi $\mathcal{B}(10.025)$ il va être concentré autour de la moyenne $np = 0.25 \times \frac{10 \times 9}{2} = 11.25$ et s'étend de 0 à 45.



12. Le sommet 0 à $n-1$ voisins possibles (tous les autres). Chaque arêts entre 0 et un autre sommet à une probabilité p d'ètre présente.
N_0 compte le nombre de succès "une arête est présente", de probabilité p lors de la répétition de $n-1$ expériences de Bernoulli indépendantes.

$$\boxed{N_0 \hookrightarrow \mathcal{B}(n-1,p)}$$

Sommets isolés

13.
```
def isoles(M):
    n=len(M)
    nb=0
    for i in range(n):
        s=0
        for j in range(n):
            s=s+M[i][j]
        if s==0:
            nb=nb+1
    return nb
```

14. $\boxed{X_n = Y_0 + Y_1 + \cdots Y_{n-1}}$

15. En utilisant la question 12.
$P(Y_0 = 1) = P(N_0 = 1) = \binom{n-1}{0} p^0 (1-p)^{n-1} = (1-p)^{n-1}$

$$P(Y_0 = 1) = (1-p)^{n-1}$$

Comme Y_0 ne prend que deux valeurs elle suit une loi binomiale, $Y_0 \hookrightarrow \mathcal{B}((1-p)^{n-1})$.
D'après les résultats du cours

$$E(Y_0) = (1-p)^{n-1}$$

16. Par linéarité de l'espérance

$$E(X_n) = E(X_0) + E(X_1) + \cdots + E(X_{n-1})$$

$$E(X_n) = n\left(1 - \frac{c + \ln n}{n}\right)^{n-1}$$

17.

$$\begin{aligned}
E(X_n) &= n\left(1-\frac{c+\ln n}{n}\right)^{n-1}\\
&= n\exp\left((n-1)\ln\left(1-\frac{c+\ln n}{n}\right)\right)\\
&= n\exp\left((n-1)\left(-\frac{c+\ln n}{n}\right)\frac{\ln\left(1-\frac{c+\ln n}{n}\right)}{-\frac{c+\ln n}{n}}\right)\\
&= n\exp\left((n-1)\left(-\frac{c}{n}\right)\frac{\ln\left(1-\frac{c+\ln n}{n}\right)}{-\frac{c+\ln n}{n}}\right)\exp\left((n-1)\left(-\frac{\ln n}{n}\right)\frac{\ln\left(1-\frac{c+\ln n}{n}\right)}{-\frac{c+\ln n}{n}}\right)\\
&= n\exp\left((n-1)\left(-\frac{c}{n}\right)\frac{\ln\left(1-\frac{c+\ln n}{n}\right)}{-\frac{c+\ln n}{n}}\right)\times\exp(-\ln n)\exp\left(\left(-\frac{n-1}{n}\right)\frac{\ln\left(1-\frac{c+\ln n}{n}\right)}{-\frac{c+\ln n}{n}}\right)\\
&= n\exp\left((n-1)\left(-\frac{c}{n}\right)\frac{\ln\left(1-\frac{c+\ln n}{n}\right)}{-\frac{c+\ln n}{n}}\right)\times\frac{1}{n}\exp\left(\left(-\frac{n-1}{n}\right)\frac{\ln\left(1-\frac{c+\ln n}{n}\right)}{-\frac{c+\ln n}{n}}\right)\\
&= \exp\left((n-1)\left(-\frac{c}{n}\right)\frac{\ln\left(1-\frac{c+\ln n}{n}\right)}{-\frac{c+\ln n}{n}}\right)\times\exp\left(\left(-\frac{n-1}{n}\right)\frac{\ln\left(1-\frac{c+\ln n}{n}\right)}{-\frac{c+\ln n}{n}}\right)
\end{aligned}$$

Or $\lim\limits_{n\to+\infty} -\dfrac{c+\ln n}{n} = 0$, théorème des croissances comparées. donc

$$\lim_{n\to+\infty}\frac{\ln\left(1-\frac{c+\ln n}{n}\right)}{-\frac{c+\ln n}{n}} = 1$$

et on trouve le résultat demandé en utilisant la continuité de la fonction exponentielle.

$$\boxed{\lim_{n\to+\infty} E(X_n) = \mathrm{e}^{-c}.}$$

Remarques Il est beaucoup plus simple d'utiliser un DL 1.

18. On peut proposer deux approches.

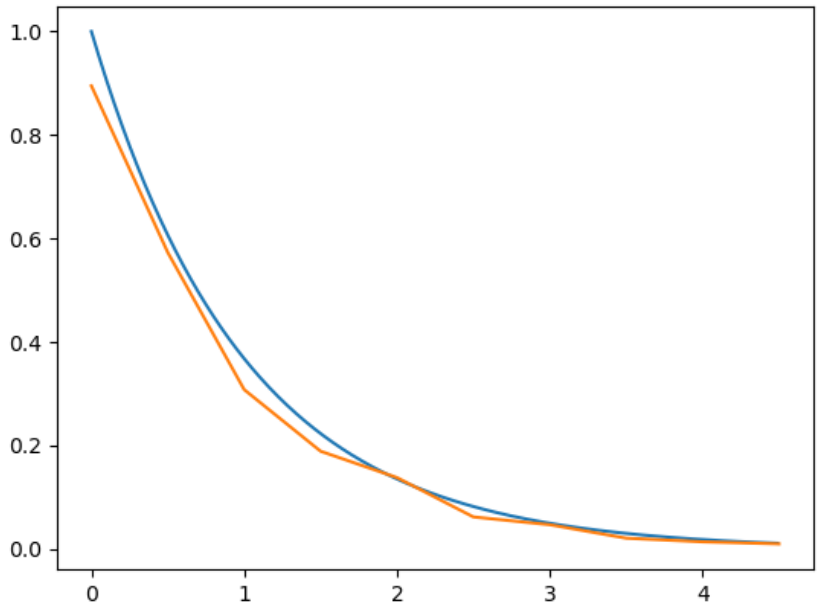
varier c à n fixé On fixe le nombre de sommets à n assez grand. Puis pour plusieurs valeurs de c on construit N (grand) graphes $G(n,p(n))$ et on comptes leur nombre de sommets isolés pour faire une moyenne. On obtient un graphe en fonction de c. Pour comparaison on fait apparaitre le graphe de la fonction $c\mapsto\exp(-c)$

```
n=100  #nombre de sommets des graphes
N=1000 # nombre de graphes générés
C=[x/2 for x in range(10)] # valeurs de c étudiées 0,0.5,1....,4.5

M=[]# liste des moyennes
for c in C:
    s=0
    for i in range(N):
```

```
            s=s+isoles(graphe(n,(np.log(n)+c)/n))
        M.append(s/N)
    # comparaison avec la cour y=exp(-x)

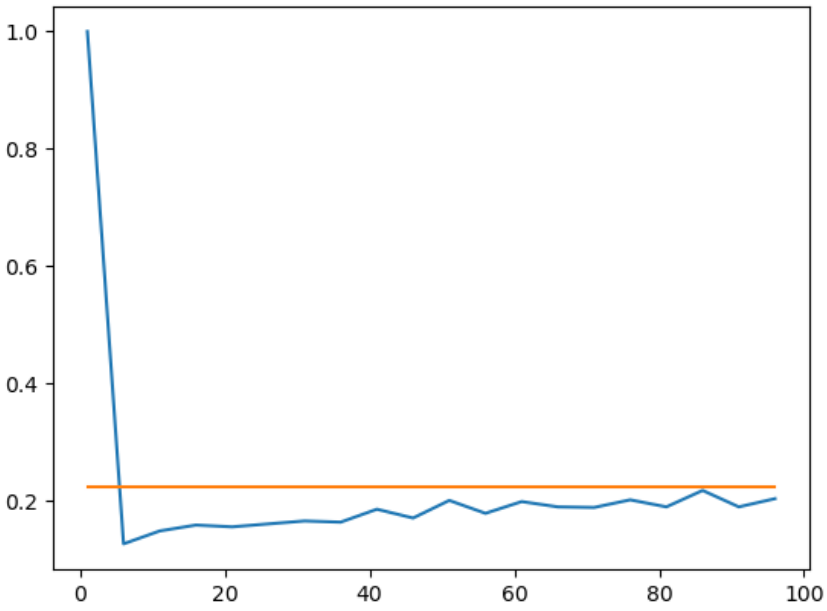
    x=np.arange(0,4.5,0.01)
    y=np.exp(-x)
    plt.plot(x,y)
    plt.plot(C,M)
    plt.show()
```



varier n à c fixé. On fixe la paramètre c, puis on fait varier n. Pour chaque cas on construit N (grand) graphes pour compter leur nombre de sommets isolés. Le graphe obtenu montre une courbe s'approchant de $exp(-c)$ quand n devient grand.

```
c=1.5
N=1000 # nombre d'expériences
nmax=100
M=[]# liste des moyennes
for n in range(1,nmax,5)  :
    s=0
    for i in range(N):
        s=s+isoles(graphe(n,(np.log(n)+c)/n))
    M.append(s/N)

##
x= [n for n in range(1,nmax,5)]
plt.plot(x,M)
const= [np.exp(-c) for n in range(1,nmax,5)]
plt.plot(x,const)
plt.show()
```


1.0
0.8
0.6
0.4
0.2
0
20
40
60
80
100