Centrale-Supelec

Oral Maths 2
Sujet type N°3
Corrigé

1. On montre sans problème que les valeurs propres de $M(a,b)$ sont $a+1$ et b (éventuellement confondues). De fait, si on considère, comme le suggère l'énoncé, que *a* et *b* sont entiers, renvoyer la valeur absolue de leur différence à 10^{-2} près ne semble pas avoir beaucoup de sens puisque cette valeur devrait être entière… Cependant en obtenant les valeurs propres grâce à la fonction `eigvals` du module `numpy.linalg`, on constate que celles-ci peuvent ne pas être entières : des problèmes d'arrondi apparaissent… Ainsi, supposant que l'on ne dispose pas des valeurs théoriques, on se donne une tolérance (10^{-2}) en deçà de laquelle on pourra supposer l'égalité des deux valeurs propres.

   La fonction `ecart` reçoit comme argument les entiers *a* et *b* (nous ne vérifions qu'ils le sont effectivement).

   Elle doit passer la matrice $M(a,b)$ comme argument à la fonction `eigvals`. On doit donc, dans un premier temps, construire cette matrice.

   La fonction `eigvals` renvoie un tableau numpy comportant une ligne et deux colonnes puisque la matrice *M* est carrée d'ordre 2. Une remarque : dans le cas où le discriminant du polynôme caractéristique est strictement négatif, la fonction `eigvals` renvoie les deux valeurs propres complexes conjuguées.

   Pour obtenir la valeur approchée de $d=|\lambda_1-\lambda_2|$, il convient de considérer la valeur de la troisième décimale : si celle-ci est strictement inférieure à 5, on doit renvoyer $\mathrm{E}(100d)/100$ mais si elle est supérieure ou égale à 5, on doit renvoyer $\mathrm{E}(100d+1)/100$ (i.e. on ajoute 1 à la deuxième décimale).

   On peut donc proposer le code suivant :

```
from math import floor
from numpy import array
from numpy.linalg import eigvals

def ecart(a,b):
     # Construction de la matrice M
     M = array([[3*a-2*b,-6*a+6*b+3],[a-b,-2*a+3*b+1]])
     # Obtention des valeurs propres de M (appel à la
     fonction "eigvals" du module linalg)
     VP = eigvals(M)
```

```
# Valeur absolue de la différence des valeurs propres
obtenues
d = abs(VP[0]-VP[1])
# Obtention de la troisième décimale
d3 = floor(1000*d)%10
if d3 < 5:
     e = floor(100*d)/100
else:
     e = floor(100*d+1)/100
return(e)
```

2. a) D'après le document d'aide Python relatif aux probabilités, le module `random` de la bibliothèque `numpy` offre la fonction `geometric` permettant de simuler une variable aléatoire suivant une loi géométrique. L'appel à cette fonction requiert deux arguments :

   - le paramètre p de la loi ($p \in]0\,;1[$).
   - Le nombre N de valeurs souhaitées.

   La fonction `geometric` renvoie un tableau comportant une ligne et N colonnes.
   Ici, on souhaite obtenir $N = 500$ valeurs. On fournira donc comme argument à la fonction `hasard` le seul paramètre p.

   Dans un premier temps, on effectue deux appels à la fonction `geometric` afin de disposer des tableaux A et B.
   Dans un deuxième temps, on exécute 500 fois une boucle `for` avec i comme compteur. A chaque exécution, on appelle la fonction `ecart` avec A[i] et B[i] comme arguments.
   Si la valeur renvoyée par `ecart` à 0,1 est supérieur ou égal à 0,1 alors on incrémente une variable S initialisée à 0 avant la boucle.

   En définitive, on peut proposer le code suivant :

```
from numpy import array
from numpy.random import geometric

def hasard(p):
   # Génération des tableaux A et B
   A = geometric(p,500)
   B = geometric(p,500)
   # On compte le nombre de fois où |A[i] - B[i]| est
   supérieur ou égal à 0,1.
   S = 0
   for i in range(500):
        if ecart(A[i],B[i]) >= 0.1:
             S += 1
   return(S)
```

b) Dans cette question, on va écrire un script qui fera appel à la fonction `hasard` écrite précédemment. On n'oubliera pas de rajouter dans les `import` une ligne comme :

```
from matplotlib.pyplot import plot, clf
```

On commence par générer deux listes comportant chacune 99 éléments :

- La première pour les abscisses : $\frac{1}{100}, \frac{2}{100}, \frac{3}{100}, ..., \frac{99}{100}$
- La seconde pour les ordonnées :

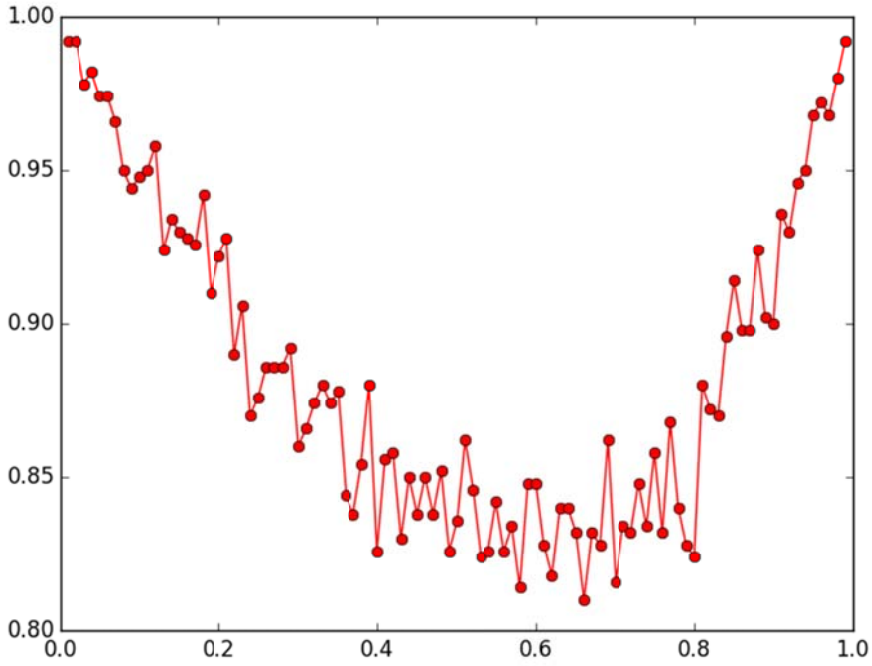
$$\frac{\text{hasard}\left(\frac{1}{100}\right)}{500}, \frac{\text{hasard}\left(\frac{2}{100}\right)}{500}, \frac{\text{hasard}\left(\frac{3}{100}\right)}{500}, ..., \frac{\text{hasard}\left(\frac{99}{100}\right)}{500}$$

Ces listes sont ensuite fournies comme arguments à la fonction `plot`.

On obtient le code suivant :

```
clf()
X, Y = [], []
for i in range(99):
   p = (i+1)/100
   X += [p]
   Y += [hasard(p)/500]
plot(X,Y,color='r',marker='o')
```

On obtient une figure qui ressemble à :



c) On commence par définir une fonction correspondant à : $p \mapsto \dfrac{2-2p+p^2}{2-p}$.

On a facilement le code suivant :

```
def f(p):
  return((2 - 2*p + p**2) / (2 - p))
```

Comme précédemment, il nous faut définir les abscisses et les ordonnées des points de la courbe représentative de la fonction *f*.

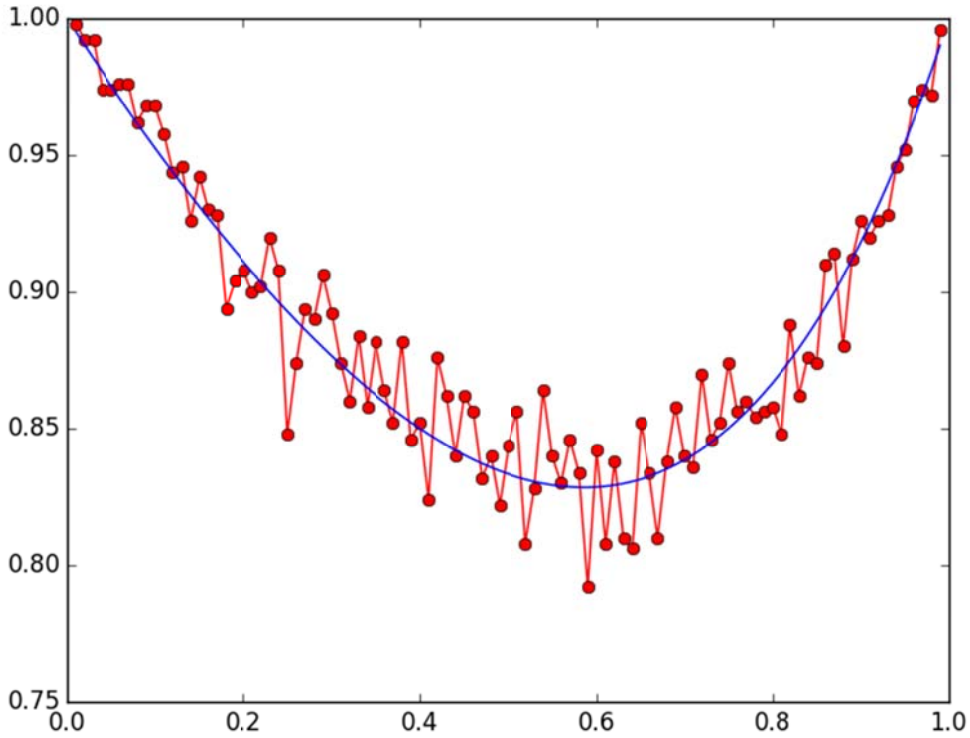
Pour les abscisses, on considère l'intervalle $]0\,;1[$ et un pas de 0,01. On construit en fait un tableau de valeurs grâce à la fonction `arange` de la bibliothèque numpy (on prend donc soin de mettre à jour l'`import` correspondant !) :

```
X = arange(0.01,1,0.01)
```

Pour les ordonnées, on utilise la syntaxe classique :

```
Y = [f(x) for x in X]
```

Après ajout de ces lignes à notre script, on obtient une figure comme :



Finalement, voici notre script complet :

```
from math import floor
from numpy import array, arange
from numpy.linalg import eigvals
from numpy.random import geometric
from matplotlib.pyplot import plot, clf

def ecart(a,b):
  # Construction de la matrice M
  M = array([[3*a-2*b,-6*a+6*b+3],[a-b,-2*a+3*b+1]])
  # Obtention des valeurs propres de M (appel à la fonction
  "eigvals" du module linalg)
  VP = eigvals(M)
  # Valeur absolue de la différence des valeurs propres
  obtenues
  d = abs(VP[0]-VP[1])
  # Obtention de la troisième décimale
  d3 = floor(1000*d)%10
  if d3 < 5:
       e = floor(100*d)/100
  else:
       e = floor(100*d+1)/100
  return(e)

def hasard(p):
  # Génération des tableaux A et B
  A = geometric(p,500)
  B = geometric(p,500)
  # On compte le nombre de fois où |A[i] - B[i]| est
  supérieur ou égal à 0,1.
  S = 0
  for i in range(500):
       if ecart(A[i],B[i]) >= 0.1:
            S += 1
  return(S)

def f(p):
  return((2 - 2*p + p**2) / (2 - p))

clf()
X, Y = [], []
for i in range(99):
  p = (i+1)/100
  X += [p]
  Y += [hasard(p)/500]
plot(X,Y,color='r',marker='o')

X = arange(0.01,1,0.01)
Y = [f(x) for x in X]
plot(X,Y)
```

Remarque (personnelle) : de façon générale, je n'importe que ce dont j'ai besoin ! Vous pouvez inclure dans votre script, si vous en avez envie, des instructions `import` plus générales comme « `from matplotlib.pyplot import *` »…

3. a) En effectuant des opérations sur les lignes et les colonnes, on obtient diverses matrices semblables les unes aux autres.

On part de :

$$\begin{pmatrix} 3a-2b & -6a+6b+3 \\ a-b & -2a+3b+1 \end{pmatrix}$$

A la première ligne, on soustrait le double de la seconde. On obtient :

$$\begin{pmatrix} 3a-2b-2(a-b) & -6a+6b+3-2(-2a+3b+1) \\ a-b & -2a+3b+1 \end{pmatrix} = \begin{pmatrix} a & -2a+1 \\ a-b & -2a+3b+1 \end{pmatrix}$$

A la deuxième colonne on ajoute le double de la première. On obtient :

$$\begin{pmatrix} a & -2a+1+2a \\ a-b & -2a+3b+1+2(a-b) \end{pmatrix} = \begin{pmatrix} a & 1 \\ a-b & b+1 \end{pmatrix}$$

On soustrait la première ligne à la deuxième. On obtient :

$$\begin{pmatrix} a & 1 \\ a-b-a & b+1-1 \end{pmatrix} = \begin{pmatrix} a & 1 \\ -b & b \end{pmatrix}$$

Enfin, on ajoute la deuxième colonne à la première. On obtient :

$$\begin{pmatrix} a+1 & 1 \\ -b+b & b \end{pmatrix} = \begin{pmatrix} a+1 & 1 \\ 0 & b \end{pmatrix}$$

En définitive :

La matrice $M(a,b)$ est semblable à la matrice $\begin{pmatrix} a+1 & 1 \\ 0 & b \end{pmatrix}$.

b) Deux matrices semblables admettent le même polynôme caractéristique. D'après la question précédente, il vient immédiatement :

$$\chi_{M(a,b)}(X) = (X-(a+1))(X-b)$$

Comme nous l'avions annoncé dans la première question, la matrice $M(a,b)$ admet donc $a+1$ et b pour valeurs propres avec, éventuellement $a+1=b$.

Si $a+1\neq b$ alors $M(a,b)$ est diagonalisable puisqu'annulée par un polynôme scindé à racines simples (son polynôme caractéristique).

Supposons maintenant que l'on ait $a+1=b$.

La valeur propre *b* est donc de multiplicité égale à 2. Avec $\begin{pmatrix} x \\ y \end{pmatrix} \neq \begin{pmatrix} 0 \\ 0 \end{pmatrix}$, on montre facilement que l'on a : $M(a,b)\times\begin{pmatrix} x \\ y \end{pmatrix} = b\begin{pmatrix} x \\ y \end{pmatrix} \Leftrightarrow y=0$. Ainsi, l'espace propre associé à la valeur propre *b* est dans ce cas la droite vectorielle d'équation $y=0$, c'est-à-dire un espace vectoriel de dimension $1\neq 2$. La matrice $M(a,b)$ n'est donc pas diagonalisable.

Finalement :

$M(a,b)$ est diagonalisable si, et seulement si : $a+1\neq b$.

c) Soit l'événement E :« $M(A,B)$ est diagonalisable ».

On a : $\overline{\mathrm{E}}$: « $M(A,B)$ n'est pas diagonalisable » et, d'après la question précédente :

$P(\mathrm{E}) = 1-P(\overline{\mathrm{E}}) = 1-P(A+1=B)$.

Les variables aléatoires *A* et *B* prennent leurs valeurs dans $\mathbb{N}^*$ et on a alors, en tenant compte de leur indépendance :

$$\begin{aligned} P(A+1=B) &= \sum_{k=1}^{+\infty} P\big((A=k)\cap(B=k+1)\big) \\ &= \sum_{k=1}^{+\infty} P(A=k)\times P(B=k+1) \\ &= \sum_{k=1}^{+\infty} (1-p)^{k-1}\times p\times(1-p)^{k}\times p \\ &= p^2(1-p)\sum_{k=1}^{+\infty}(1-p)^{2k-2} \\ &= p^2(1-p)\sum_{k=0}^{+\infty}(1-p)^{2k} \end{aligned}$$

Comme $p\in\left]0\,;1\right[$, il en va de même pour $1-p$ et $(1-p)^2$ et il vient finalement :

$$P(A+1=B) = p^2(1-p)\sum_{k=0}^{+\infty}(1-p)^{2k} = p^2(1-p)\times\frac{1}{1-(1-p)^2} = \frac{p^2(1-p)}{p(2-p)}$$

$$= \frac{p(1-p)}{2-p}$$

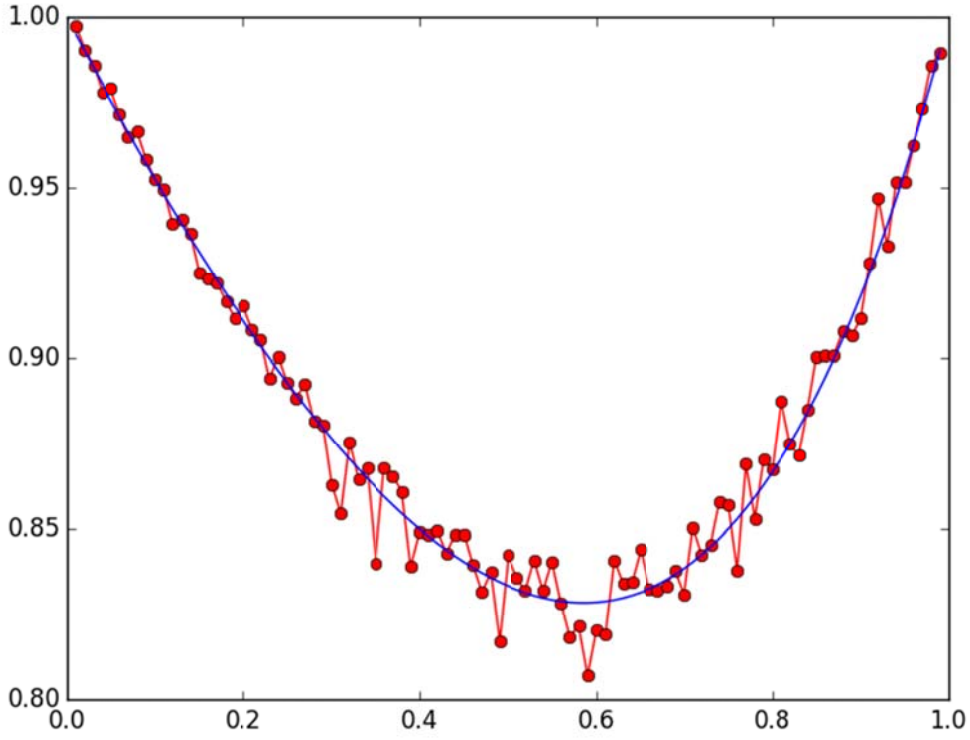
Il vient alors :

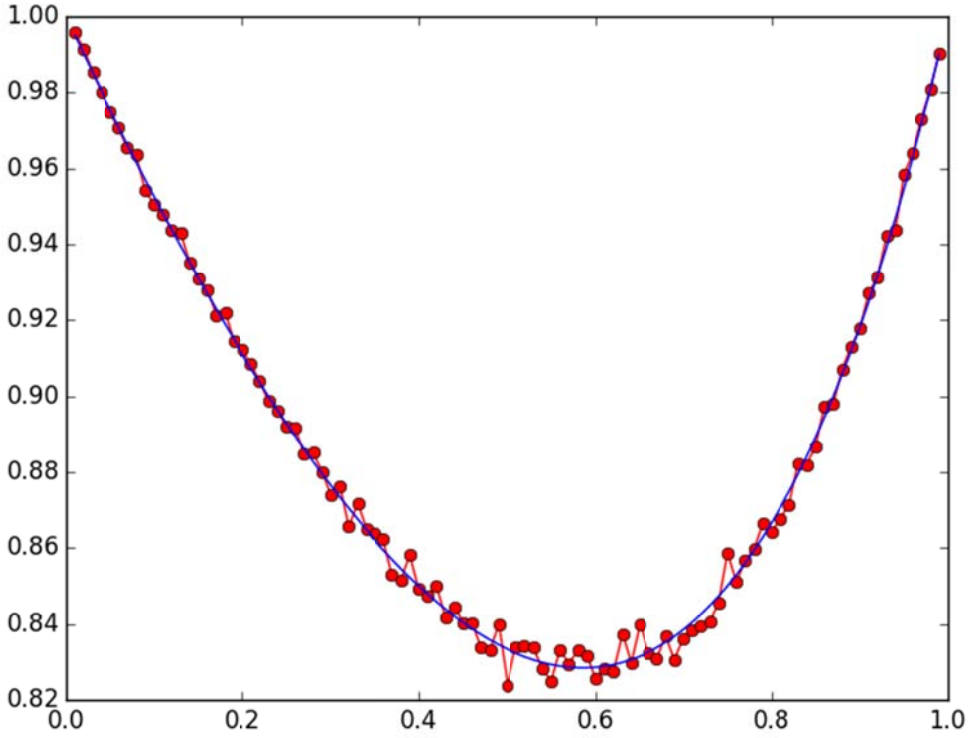
$$P(\mathrm{E}) = 1 - P(\overline{\mathrm{E}}) = 1 - P(A+1=B) = 1 - \frac{p(1-p)}{2-p} = \frac{2-p-p+p^2}{2-p} = \frac{2-2p+p^2}{2-p}$$

La probabilité de l'événement « $M(A, B)$ est diagonalisable » est égale à : $\frac{2-2p+p^2}{2-p}$.

On retrouve exactement l'expression donnée à la question 2. c). On ne doit pas s'en étonner ! A la question 2. on a fondamentalement effectué une simulation et on est parti du principe que les deux valeurs propres de la matrice étaient différentes dès lors que leur écart était, en valeur absolue, supérieur à 10^{-1}. On peut, à posteriori, émettre des réserves sur la valeur 500 qui a finalement entraîné une fluctuation non négligeable autour de la courbe théorique (bleue).

Avec respectivement 2 000 et 10 000valeurs simulées (pour chaque valeur de *p*) au lieu de 500, on obtient :




1.00
0.98
0.96
0.94
0.92
0.90
0.88
0.86
0.84
0.82
0.0
0.2
0.4
0.6
0.8
1.0