tprenouvellement

July 9, 2020

1 Introduction aux Processus Stochastiques (PRSTO)

P. Carmona

Pour avancer dans le notebook et exécuter les cellules il faut taper Shift+Enter ou utiliser la barre d'outils ci-dessus et choisir Cell, Run Cell and select Below

1.1 Consignes

Vous répondrez aux questions en modifiant ce notebook. En insérant des cellules de type Markdown pour le texte et des cellules de type code pour le code.

Ensuite vous sauvez ce notebook sous le nom Prenom_Nom_tpmarkov.ipynb et vous le déposez sur Moodle

1.2 Simulation d'un processus de Poisson

Nous allons simuler un processus de Poisson d'intensité $\$\lambda = 1/10\$mnsurunintervalledetempsd'unejourneeT = 24 * 60 = 1440$. La commande

[2]:
```
import numpy as np
import matplotlib.pyplot as plt
import numpy.random as rnd
%matplotlib inline


lam=0.1
x=rnd.exponential(scale=1/lam)
x
```

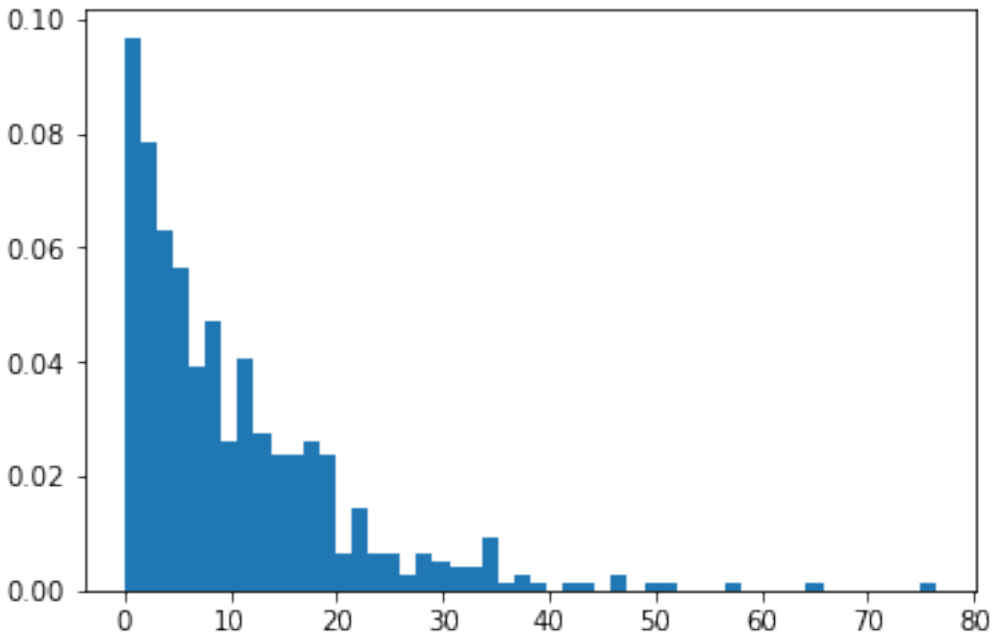
[2]:
```
0.7167649739279774
```

génère une variable exponentielle de paramètre λ et la place dans x. Pour générer un échantillon de $n = 500$ variables, calculer la moyenne et la variance empiriques, et tracer un histogramme normalisé,avec 50 barres, on écrit

[6]:
```
n=500
x=rnd.exponential(scale=1/lam,size=n)
x.mean()
```

```
x.var()
plt.hist(x, 50, normed=1)
```

[6]:
```
(array([0.09681737, 0.07850057, 0.06280045, 0.05625874, 0.03925028,
        0.04710034, 0.02616686, 0.04055863, 0.0274752 , 0.02355017,
        0.02355017, 0.02616686, 0.02355017, 0.00654171, 0.01439177,
        0.00654171, 0.00654171, 0.00261669, 0.00654171, 0.00523337,
        0.00392503, 0.00392503, 0.0091584 , 0.00130834, 0.00261669,
        0.00130834, 0.        , 0.00130834, 0.00130834, 0.        ,
        0.00261669, 0.        , 0.00130834, 0.00130834, 0.        ,
        0.        , 0.        , 0.00130834, 0.        , 0.        ,
        0.        , 0.        , 0.00130834, 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.00130834]),
 array([1.48407080e-02, 1.54349208e+00, 3.07214346e+00, 4.60079483e+00,
        6.12944621e+00, 7.65809758e+00, 9.18674896e+00, 1.07154003e+01,
        1.22440517e+01, 1.37727031e+01, 1.53013545e+01, 1.68300058e+01,
        1.83586572e+01, 1.98873086e+01, 2.14159600e+01, 2.29446113e+01,
        2.44732627e+01, 2.60019141e+01, 2.75305655e+01, 2.90592168e+01,
        3.05878682e+01, 3.21165196e+01, 3.36451710e+01, 3.51738223e+01,
        3.67024737e+01, 3.82311251e+01, 3.97597765e+01, 4.12884278e+01,
        4.28170792e+01, 4.43457306e+01, 4.58743820e+01, 4.74030333e+01,
        4.89316847e+01, 5.04603361e+01, 5.19889875e+01, 5.35176388e+01,
        5.50462902e+01, 5.65749416e+01, 5.81035930e+01, 5.96322443e+01,
        6.11608957e+01, 6.26895471e+01, 6.42181985e+01, 6.57468498e+01,
        6.72755012e+01, 6.88041526e+01, 7.03328040e+01, 7.18614553e+01,
        7.33901067e+01, 7.49187581e+01, 7.64474095e+01]),
 <a list of 50 Patch objects>)
```



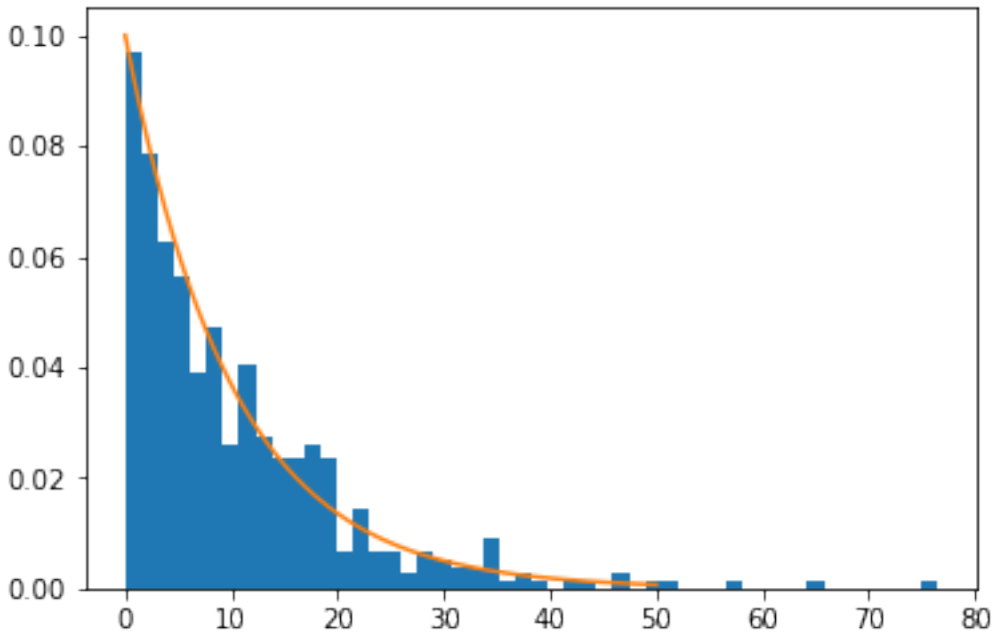
Lorsque l'on désire des informations détaillées sur une fonction on tape

[7]:
```
?plt.hist
```

On améliore le graphique ci-dessus en comparant l'histogramme avec la fonction densité.

[11]:
```
plt.hist(x, 50, normed=1)
tt=np.linspace(0.0, 50,200)
plt.plot(tt,lam*np.exp(-lam*tt))
```

[11]:
```
[<matplotlib.lines.Line2D at 0x7f73f3aec4e0>]
```



1.3 Exercice

En utilisant la structure de contrôle

while True:

    ...

    if cond:

        break

coder une fonction

    ...

[22]:
```
def spoi(lam,T):
```

```
  File "<ipython-input-22-d8cd16ac83cb>", line 1
    def spoi(lam,T):
                   ^
SyntaxError: unexpected EOF while parsing
```

qui prend en paramètre λ et T, et qui renvoie un vecteur des temps de saut du processus de Poisson sur l'intervalle $[0, T]$.

On remarque que

[24]:
```
plt.plot(spoi(0.1,100))
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)

<ipython-input-24-c037ed53cbe2> in <module>()
----> 1 plt.plot(spoi(0.1,100))
      2

NameError: name 'spoi' is not defined
```

n'affiche ansolument pas ce que l'on veut mais la courbe inverse. Réfléchissez bien et affichez la bonne courbe.

Une solution est d'écrire

[25]:
```
s=spoi(0.1,100)
plt.step(s,range(len(s)),where='mid')
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)

<ipython-input-25-32c0464ba3ef> in <module>()
----> 1 s=spoi(0.1,100)
      2 plt.step(s,range(len(s)),where='mid')

NameError: name 'spoi' is not defined
```

1.4 Exercice

Coder une seconde fonction sspoi qui a les mêmes paramètres, mais qui utilise la construction générale d'un processus de Poisson. Donc elle genère une variable N de loi $Pois(\lambda T)$ Puis elle génère N uniformes sur $[0, T]$ qu'elle trie. Utiliser les commandes ,rnd.poisson,rnd.runif et x.sort() pour trier un tableau de valeurs. Et affichez également le processus obtenu.

Pour être pleinement convaincu que cette seconde méthode génère bien un processus de Poisson classique, vous allez illustrer le fait que le premier temps de saut est une variable exponentielle de paramètre λ en générant $n = 500$ telles variables et en traçant l'histogramme et la densité de la loi exponentielle.

Le paradoxe de l'autobus Dans un premier temps on suppose que l'on arrive à $t = 15h00 = 15 * 60$ minutes et que l'on désire déterminer les valeurs moyennes de l'âge $A_t = t - S_{N_t}$ et du temps de vie résiduel $R_t = S_{1+N(t)} - t$, qui sont respectivement l'intervalle depuis le passage du dernier bus et le temps d'attente du prochain bus.

On écrit donc une fonction *attente* qui prend en paramètres λ et t et revoie le couple de valeurs A_t, R_t.

def attente(lam,t): >...

return ((at,rt))

[ ]:

Puis on utilise

[26]:
```
lam=0.1
t=300
res=[attente(lam,t) for i in range(1000)]
```

```
---------------------------------------------------------------------------

NameError                                 Traceback (most recent call last)

<ipython-input-26-55e34e3b5e70> in <module>()
  1 lam=0.1
  2 t=300
----> 3 res=[attente(lam,t) for i in range(1000)]


<ipython-input-26-55e34e3b5e70> in <listcomp>(.0)
  1 lam=0.1
  2 t=300
----> 3 res=[attente(lam,t) for i in range(1000)]


NameError: name 'attente' is not defined
```

qui fabrique un vecteur contenant 1000 réplications (i.e. les valeurs de 1000 appels successifs de la fonction) et on les place dans une matrice, en calculant la moyenne des lignes ce qui par la loi des grands nombres donne des estimations des moyennes recherchées

[27]:
```
m=res.reshape(2,500)
m.mean(axis=1)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)

<ipython-input-27-a012c8b3c9f7> in <module>()
----> 1 m=res.reshape(2,500)
      2 m.mean(axis=1)


NameError: name 'res' is not defined
```

1.5 Exercice

Pour illustrer l'indépendance par rapport à l'instant d'arrivée, faire les mêmes calculs en tirant cette fois au hasard votre temps d'arrivée à l'arrêt, en utilisant par exemple une loi uniforme sur 15h, 15h15.

Vérifier vos simulations en utilisant le calcul matriciel. Sous Python, le produit matriciel des matrices p et q s'écrit np.dot(p,q) ce qui est très différent de $p * q$. Ecrivez une fonction puis(p,n) qui renvoie la matrice p^n. (Vous pouvez utiliser un algorithme de calcul rapide de puissance si vous savez le coder).

1.6 Taille optimale d'un buffer

On fixe les valeurs du coût de vidage $K = 100$ euros et $\lambda = 1$, i.e. on a un message en moyenne par unité de temps. En faisant varier le coût unitaire h de stockage d'un message (valeurs \$0.1,1,10 \$ ou 100), calculer le coût moyen par cycle en effectuant un grand nombre (n=500) de cycles, et ce pour suffisamment de valeurs de T pour identifier la valeur optimale.

On pourra écrire d'abord une fonction

[28]:
```
def coutcycle(T,h,K,lam):
```

```
  File "<ipython-input-28-5ca6ce36db80>", line 3

    ^
SyntaxError: unexpected EOF while parsing
```

qui calcule le coût unitaire observé pour la réalisation d'un cycle.

Commandes Python : np.linspace pour construire un vecteurs de valeurs uniformément réparties

Ensuite il faut une fonction qui donne le coût moyen estime par la loi forte des grands nombres

[29]:
```
def coutmoyenempi(T,h,K,lam,n):
```

```
  File "<ipython-input-29-ea9d9f5965d8>", line 1
    def coutmoyenempi(T,h,K,lam,n):
                                   ^
SyntaxError: unexpected EOF while parsing
```

Puis on tracera, au moins pour une valeur de h, la courbe qui donne le coût moyen estimé en fonction de T. Et la on vérifiera que la valeur T^* calculée en TD est bien optimale.

[ ]:

[ ]: