simulationVARaDensite

February 1, 2023

1 Simulation de variables aléatoires à densité

1.1 I. Introduction

Simuler une loi de probabilité $\mathcal{L}$ consiste à écrire un algorithme qui génère une suite finie de nombres réels dont on considère que ce sont des réalisations indépendantes de la loi $\mathcal{L}$.

On notera au préalable que, puisque les algorithmes sont par nature déterministes, les suites engendrées ne sont, dans les faits, ni aléatoires, encore moins indépendantes... On admettra pourtant qu'elles approchent suffisamment ces hypothèses pour que, dans la pratique, on parle de *suites pseudo-aléatoires*.

Python, *via* son module `random` permet de simuler la réalisation de la loi uniforme sur l'intervalle $[0,1[$. Il suffit pour cela d'appeler la fonction `random()`. Cette fonction est fondamentale car elle permet toutes les simulations de lois usuelles imposées par le programme officiel.

On peut lire dans l'aide en lignes `docs.python.org` la description suivante :" Almost all module functions depend on the basic function random(), which generates a random float uniformly in the semi-open range [0.0, 1.0). Python uses the Mersenne Twister as the core generator. It produces 53-bit precision floats and has a period of $2 * *19937 - 1$. [...] The Mersenne Twister is one of the most extensively tested random number generators in existence. However, being completely deterministic, it is not suitable for all purposes."

Dans la pratique, on retiendra que les générateurs de nombres pseudo-aléatoires initialisent leurs calculs à l'aide d'une valeur initiale s_0 appelée *graine*.
Il vous est possible de choisir cette graine grâce à la fonction `random.seed(a)` et les avantages sont multiples : Une fois la graine choisie par l'utilisateur, il est possible de reproduire intégralement la séquence des nombres pseudo-aléatoires produits, ce qui peut être utile pour vérifier des résultats obtenus par simulation. Enfin, "le fait d'employer une procédure standardisée permet de disposer d'informations fiables sur la qualité et les performances des procédures utilisées en provenance de divers chercheurs et utilisateurs" (D'après http://math.univ-lyon1.fr/~jberard/genunif-www.pdf - document complet sur lequel on trouvera quantité d'informations sur les générations possibles de nombres pseudo-aléatoires).

Une bibliothèque complémentaire indispensable :

La bibliothèque `scipy.stats` possède les fonctions `randint`, `bernoulli`, `binom` et `hypergeom` qui permettent le calcul de $\mathbb{P}(X = k)$ pour tout k dans $X(\Omega)$ lorsque X est une variable aléatoire qui suit respectivement les lois *uniforme*, *bernoulli*, *binomiale* et *hypergéométrique* mais aussi (parmis bien d'autres) les fonctions `uniform`, `expon` et `norm` qui permettent d'obtenir une densité, la fonction de

répartition, la réciproque de la fonction de réciproque, ou encore des moments des lois *uniforme*, *exponentielles* et *normal*.

Il suffit pour ça, dans le cas des variables aléatoires à densité, d'appeler :
- `nom_Loi.pdf(x, paramètres)` (`pdf` pour *probability density function*) pour obtenir $f_X(x)$,
- `nom_Loi.cdf(x, paramètres)` (`cdf` pour *cumulative density function*) pour obtenir $F_X(x)$,
- `nom_Loi.ppf(y, paramètres)` (`ppf` pour *percentil point function*) pour obtenir $F_X^{-1}(y)$ pour tout $y \in [0,1[$.

Enfin l'exécution de `mean, var = nom_Loi.stats(paramètres, moments='mv')` permet d'obtenir la moyenne et la variance de la loi dont les paramètres sont fournis en argument.

1.2 II/ Importation préalable des bibliothèques nécessaires :

[1]:
```
import numpy as np
import numpy.random as nrdm
import matplotlib.pyplot as plt
import random as rdm
```

[2]:
```
rdm.seed(1)
```

1.3 III/ Modélisation des lois usuelles.

1.3.1 1. La loi uniforme sur $[0,1[$.

On rappelle que X suit une loi uniforme sur $[0,1[$ si elle admet pour densité :

$$f(x) = \begin{cases} 1 & \text{si } 0 \leq x < 1 \\ 0 & \text{sinon} \end{cases}$$

et pour fonction de répartition :

$$F_X(x) = \begin{cases} 0 & \text{si } x < 0 \\ x & \text{si } 0 \leq x < 1 \\ 1 & \text{sinon} \end{cases}$$

On a par ailleurs :

$$\mathbb{E}(X) = \frac{1}{2}; \mathbb{V}(X) = \frac{1}{12}$$

Q. 1.1 : Écrire une fonction Python `simul_loiUniformeL(m)` qui renvoie une liste L formée de m réels compris entre 0 et 1 considérés comme m réalisations indépendantes de la loi uniforme sur $[0,1]$:

[3]:
```
def simul_loiUniformeL(m: int) -> list:
    return [rdm.random() for k in range(m)]
```

[4]:
```
L = simul_loiUniformeL(10)
print(L, end = "")
```

```
[0.13436424411240122, 0.8474337369372327, 0.763774618976614, 0.2550690257394217,
0.49543508709194095, 0.4494910647887381, 0.651592972722763, 0.7887233511355132,
0.0938595867742349, 0.02834747652200631]
```

Q. 1.2 : Écrire une fonction Python `simul_loiUniformeT(m)` qui renvoie un tableau T de taille (n_l, n_c) formé de $m = n_l \times n_c$ réels compris entre 0 et 1 considérés comme m réalisations indépendantes de la loi uniforme sur $[0,1]$:

[5]:
```
def simul_loiUniformeT(nl:int, nc:int) -> np.ndarray:
    return nrdm.random((nl, nc))
```

[6]:
```
simul_loiUniformeT(5, 3)
```

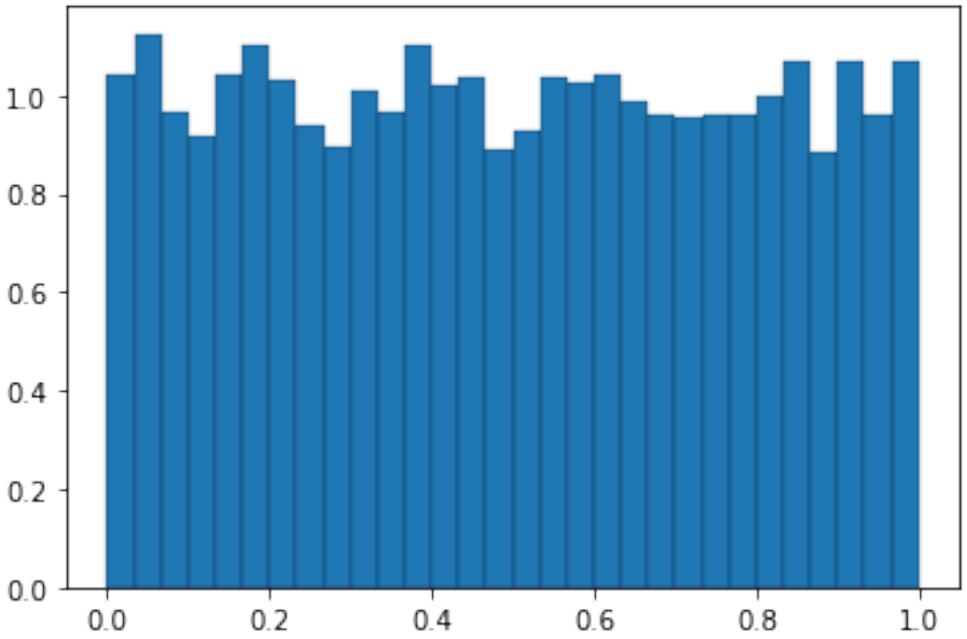
[6]:
```
array([[0.15464867, 0.08526113, 0.90160988],
       [0.57859147, 0.08966609, 0.38029836],
       [0.74667891, 0.28980509, 0.04968183],
       [0.53201461, 0.21118639, 0.55015345],
       [0.01512282, 0.44454913, 0.51046604]])
```

Q. 1.3 : Ecrire une fonction `histo` d'argument une liste de réels et le nombre de barres souhaitées dans l'histogramme, qui trace cet histogramme de telle façon que la somme des surfaces fasse 1, et renvoie la largeur des barres ainsi que leur hauteur.

[7]:
```
def histo(vals: list, nb_barres: int) -> list:
    """ H est un tuple :
    H[0] : liste des fréquences relatives de chaque classe divisées par la␣
 ↪largeur des barres
    H[1] : liste des bornes gauche de chaque classe ainsi que la borne sup de la␣
 ↪dernière classe
    """
    m = min(vals)
    M = max(vals)
    pas = (M - m) / nb_barres # largeur des barres
    H = plt.hist(vals, bins=[m + pas*k for k in range(nb_barres + 1)],
                 density = True, edgecolor='black', linewidth=0.2)
    return pas, H
```

[8]:
```
Lu = simul_loiUniformeL(10000)
pas, H = histo(Lu, 30)
print("fréquences empiriques = ", H[0]*pas)
print("la somme des surfaces vaut :", sum(H[0])*pas)
```

```
fréquences empiriques =  [0.0348 0.0375 0.0322 0.0305 0.0347 0.0368 0.0343
0.0313 0.0298 0.0336
 0.0322 0.0367 0.034  0.0346 0.0296 0.031  0.0346 0.0342 0.0347 0.0329
 0.0321 0.0319 0.032  0.032  0.0333 0.0357 0.0295 0.0357 0.0321 0.0357]
la somme des surfaces vaut : 1.0000000000000002
```

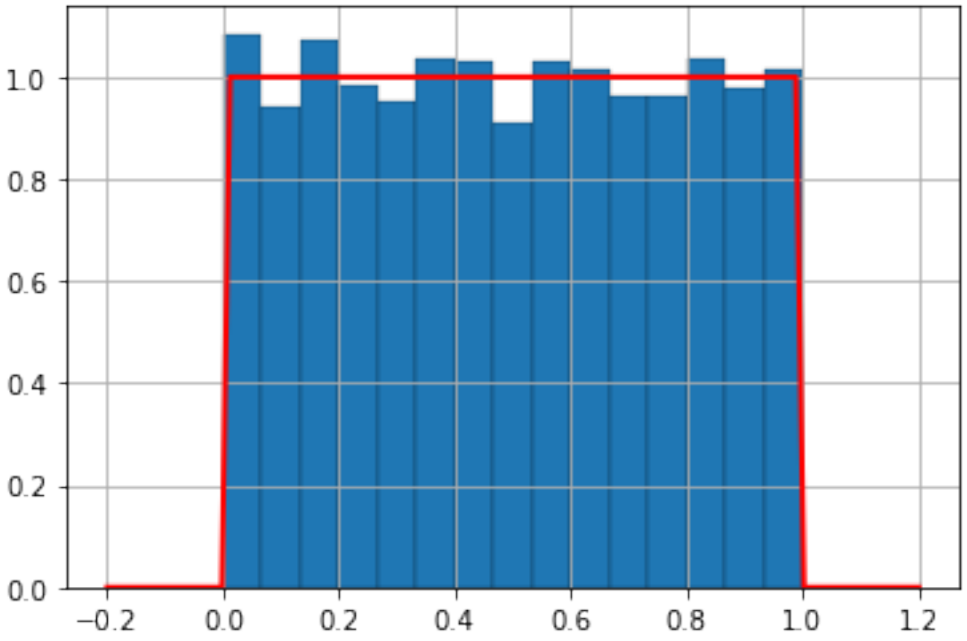


[9]:
```
print("le pas vaut :", pas)
print("on estime P(0 <= X < pas) =", H[0][0]*pas)
print("la valeur exacte vaut F_X(pas), soit :", pas) # puisque F_X(x) = x
```

```
le pas vaut : 0.03332380796947866
on estime P(0 <= X < pas) = 0.034800000000000005
la valeur exacte vaut F_X(pas), soit : 0.03332380796947866
```

Q. 1.4 : Utiliser la bibliothèque `scipy.stats` pour tracer une densité de la loi uniforme sur l'intervalle $[-0.2, 1.2]$ et la confronter à l'histogramme obtenu à la question précédente.

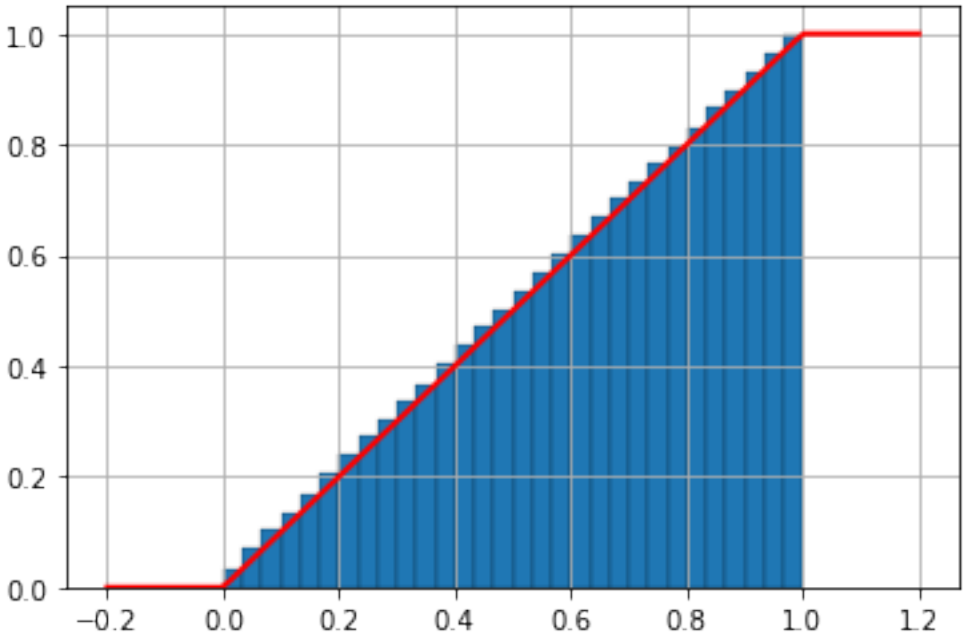
[10]:
```
from scipy.stats import uniform
a, b = -0.2, 1.2
X = np.linspace(a,b,100)
plt.plot(X, uniform.pdf(X),'r',lw=2)
plt.grid()
pas, H = histo(Lu, 15)
```



Q. 1.5 : Modifier la fonction `histo()` pour qu'elle trace l'histogramme des fréquences cumulées et, grâce à la bibliothèque `scipy.stats`, tracez la fonction de répartition de la loi uniforme sur l'intervalle $[-0.2, 1.2]$ et confrontez la à l'histogramme obtenu.

[11]:
```
def histo_FC(vals: list, nb_barres: int) -> list:
    m = min(vals)
    M = max(vals)
    pas = (M - m) / nb_barres # largeur des barres
    Hc = plt.hist(vals, bins=[m + pas*k for k in range(nb_barres + 1)],
                  density = True, cumulative = True, edgecolor='black',
                  linewidth=0.2)
    return pas, Hc

plt.plot(X, uniform.cdf(X),'r',lw=2)
plt.grid()
pas, Hc = histo_FC(Lu, 30)
```



1.3.2 2. Loi uniforme sur $[a,b]$ où $(a,b) \in \mathbb{R}^2$, $a < b$.

On rappelle que X suit une loi uniforme sur $[a,b]$ si elle admet pour densité :

$$f(x) = \begin{cases} \dfrac{1}{b-a} & \text{si } a \leq x \leq b \\ 0 & \text{sinon} \end{cases}$$

et pour fonction de répartition :

$$F_X(x) = \begin{cases} 0 & \text{si } x < a \\ \dfrac{x-a}{b-a} & \text{si } a \leq x \leq b \\ 1 & \text{sinon} \end{cases}$$

On a par ailleurs :

$$\mathbb{E}(X) = \frac{a+b}{2}; \mathbb{V}(X) = \frac{(b-a)^2}{12}$$

Q. 2.1 : Écrire une fonction Python `simul_loiUniforme2L(m, a, b)` qui renvoie une liste L formée de m réels compris entre a et b considérés comme m réalisations indépendantes de la loi uniforme sur $[a,b]$:

[12]:
```
def simul_loiUniforme2L(m: int, a: float, b: float) -> list:
    return [(b-a)*rdm.random()+a for k in range(m)]
```

[13]:
```
L = simul_loiUniforme2L(10, 1, 3)
print(L, end = "")
```

```
[1.2553422438817599, 2.8664580843175305, 1.2105958698103798, 2.4268390595378952,
1.9555136103318058, 1.4498595752146985, 2.5998774311641233, 1.9101357343626735,
2.0992276303540267, 2.7169799473481215]
```

Q. 2.2 : Écrire une fonction Python `simul_loiUniforme2T(nl, nc, a, b)` qui renvoie un tableau T de taille (n_l, n_c) formé de $m = n_l \times n_c$ réels compris entre a et b considérés comme m réalisations indépendantes de la loi uniforme sur $[a, b]$:

[14]:
```
def simul_loiUniforme2T(a:float, b:float, nl:int, nc:int) -> np.ndarray:
    return nrdm.uniform(a, b, (nl, nc))
```

[15]:
```
simul_loiUniforme2T(1, 3, 3, 5)
```

[15]:
```
array([[2.82452095, 2.00727631, 1.74624158, 2.32122208, 1.48393143],
       [2.80034456, 2.70346678, 1.84271831, 2.8517275 , 2.22618887],
       [2.45131862, 2.20151204, 1.83105614, 2.92161267, 1.85502946]])
```

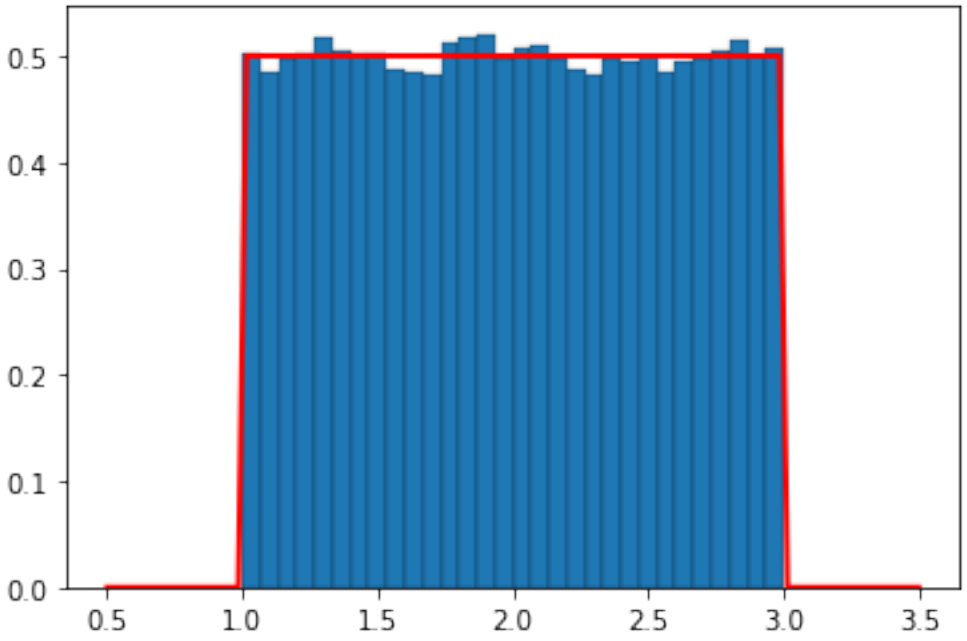
Q. 2.3 : Modéliser sous forme de liste la réalisation d'un nombre m supposé grand d'une variable aléatoire qui suit la loi uniforme sur l'intervalle $[1, 3]$. Calculer la moyenne et la variance expérimentale des valeurs de cette liste et confronter à la moyenne et la variance de la loi uniforme sur $[1, 3]$:

[16]:
```
a, b = 1, 3
Lu = simul_loiUniforme2L(10000, a, b)
print(f'moyenne et variance expérimentale : {np.mean(Lu)} et {np.var(Lu)}')
print(f"l'espérance et la variance de X valent : {(a+b)/2} et {(b-a)**2/12}")
```

```
moyenne et variance expérimentale : 2.00358145075388 et 0.3323135274446052
l'espérance et la variance de X valent : 2.0 et 0.3333333333333333
```

Q. 2.4 : Utiliser la bibliothèque `scipy.stats` pour tracer une densité de la loi uniforme sur l'intervalle $[1, 2]$ et la confronter à l'histogramme obtenu grâce à la fonction `histo`.

[17]:
```
a, b = 1, 3
X = np.linspace(a-0.5, b+0.5, 100)
plt.plot(X, uniform.pdf(X, loc=a, scale=b-a), 'r', lw=2)
Lu = simul_loiUniforme2L(100000, a, b)
pas, H = histo(Lu, 30)
```

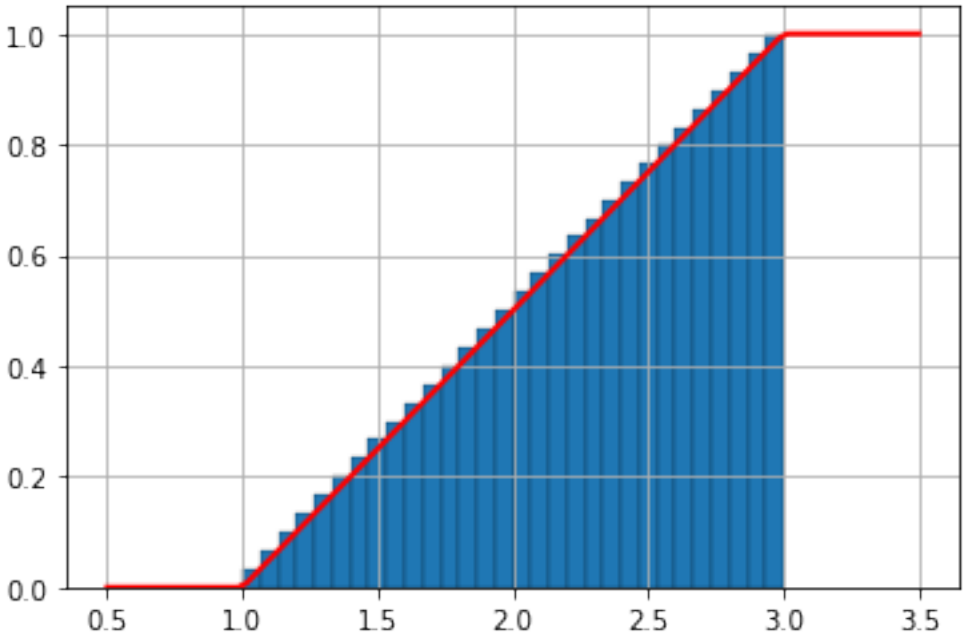


[18]:
```
print("le pas vaut :", pas)
m, M = min(Lu), max(Lu)
k = 5 # on s'intéresse, par exemple, à la 5ème classe de l'histogramme (à␣
 ↪modifier)
ak, bk = m+k*pas, m+(k+1)*pas # kème classe de l'histogramme
print("on estime P(ak <= X < bk) par", H[0][k]*pas)
print("la valeur exacte vaut F_X(bk)-F_X(ak), soit :", (bk-ak)/(b-a))
```

```
le pas vaut : 0.06666490175629604
on estime P(ak <= X < bk) par 0.03364999999999995
la valeur exacte vaut F_X(bk)-F_X(ak), soit : 0.03333245087814807
```

Q. 2.5 : Utiliser la fonction `histo_FC()` et la bibliothèque `scipy.stats` pour confronter la fonction de répartition de la loi uniforme sur l'intervalle $[a, b]$ et l'histogramme des fréquences cumulées.

[19]:
```
plt.plot(X, uniform.cdf(X, loc=a, scale=b-a),'r',lw=2)
plt.grid()
pas, Hc = histo_FC(Lu, 30)
```



1.3.3 3. Loi exponentielle de paramètre λ.

On rappelle que X suit une loi exponentielle de paramètre λ si elle admet pour densité :

$$f(x) = \begin{cases} \lambda \cdot e^{-\lambda x} & \text{si } x \geq 0 \\ 0 & \text{sinon} \end{cases}$$

et pour fonction de répartition :

$$F_X(x) = \begin{cases} 1 - e^{-\lambda x} & \text{si } x \geq 0 \\ 0 & \text{sinon} \end{cases}$$

On a par ailleurs :

$$\mathbb{E}(X) = \frac{1}{\lambda}; \mathbb{V}(X) = \frac{1}{\lambda^2}$$

Q 3.1 : Écrire une fonction Python `simul_loiExponentielleL(m, lbd)` qui renvoie une liste L formée de m réels positifs considérés comme m réalisations indépendantes de la loi exponentielle de paramètre λ :

[20]:
```
def simul_loiExponentielleL(m: int, lbd: float) -> list:
    return [-(1/lbd)*np.log(1-rdm.random()) for k in range(m)]
```

[21]:
```
Le = simul_loiExponentielleL(15, 1/3)
print(Le, end="")
```

```
[0.93309676017633, 0.7106178344479279, 9.223771459190905, 5.473292050759856,
3.8181380573282446, 2.5603754354139334, 0.08431785237555284, 2.9846193780742523,
2.9305311752942558, 3.649258149300054, 20.46639982537671, 1.3827651849215075,
3.9505380043728913, 12.159660810244771, 1.8209183365628274]
```

Q 3.2 : Écrire une fonction Python `simul_loiExponentielleT(nl, nc, lbd)` qui renvoie un tableau T de taille (nl, nc) formé de $m = nl \times nc$ réels positifs considérés comme m réalisations indépendantes de la loi exponentielle de paramètre λ :

[22]:
```
lbd = 1/3
Te = nrdm.exponential(1/lbd, (3, 5)) # Attention : c'est l'espérance en argument
print(Te)
```

```
[[0.67180986 1.69522358 0.48875264 0.9009437  0.89312271]
 [5.61649955 0.81879625 1.44432073 0.11467736 4.89267239]
 [3.48292146 4.27725554 3.5183623  1.10458111 0.84634371]]
```

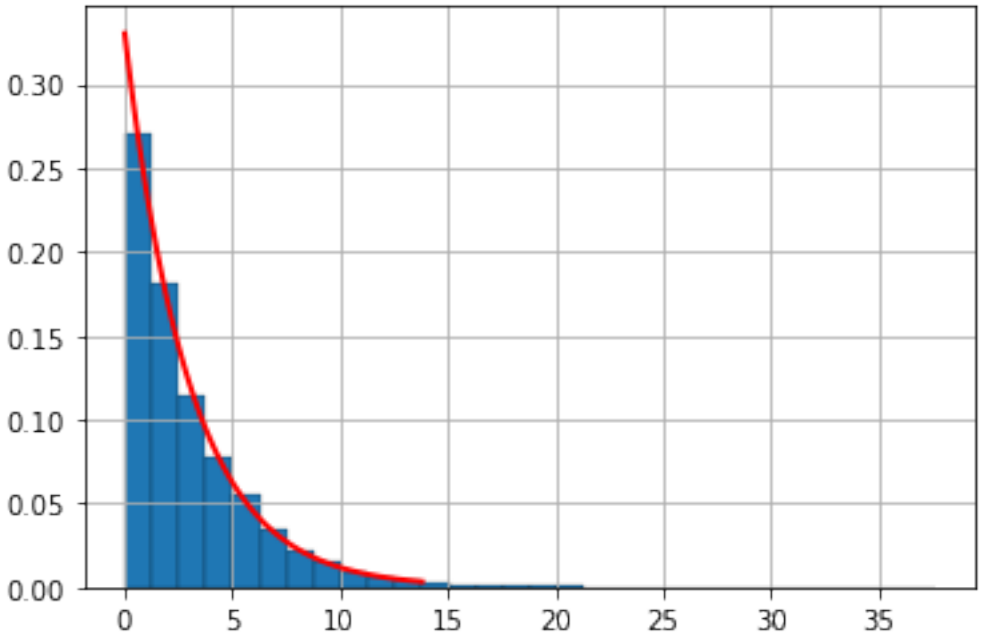
Q. 3.3 : Modéliser sous forme de liste la réalisation d'un nombre m supposé grand d'une variable aléatoire qui suit la loi exponentielle de paramètre λ. Calculer la moyenne et la variance expérimentale des valeurs de cette liste et confronter à la moyenne et la variance de la loi exponentielle de paramètre λ :

[23]:
```
lbd = 1/3
Le = simul_loiExponentielleL(10000, lbd)
print(f'moyenne et variance expérimentale : {np.mean(Le)} et {np.var(Le)}')
print(f"l'espérance et la variance de X valent : {1/lbd} et {1/lbd**2}")
```

```
moyenne et variance expérimentale : 3.0101643649348118 et 8.714274134702078
l'espérance et la variance de X valent : 3.0 et 9.0
```

Q. 3.4 : Utiliser la bibliothèque `scipy.stats` pour tracer une densité de la loi exponentielle de paramètre λ et la confronter à l'histogramme de la série `Le` obtenu grâce à la fonction `histo`.

[24]:
```
from scipy.stats import expon
lbd = 1/3
a = expon.ppf(0.01, scale = 1/lbd) # FX(a) = P(X<=a) = 0.01
b = expon.ppf(0.99, scale = 1/lbd) # Fx(b) = P(X<=b) = 0.99 [ou P(X>b)=0.01]
X = np.linspace(a, b, 100)
plt.plot(X, expon.pdf(X,scale = 1/lbd), 'r', lw=2)
plt.grid()
Le = simul_loiExponentielleL(10000, lbd)
pas, He = histo(Le, 30)
```

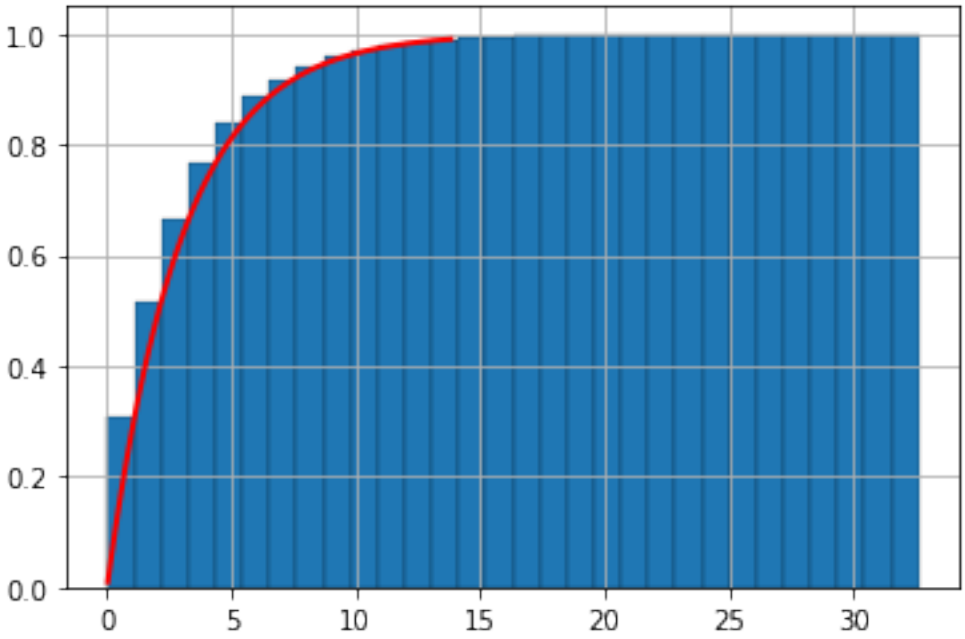


[25]:
```
print("le pas vaut :", pas)
m, M = min(Le), max(Le)
k = 5 # on s'intéresse, par exemple, à la 5ème classe de l'histogramme
      # n'hésitez pas à modifier cette valeur...
ak, bk = m+k*pas, m+(k+1)*pas # kème classe de l'histogramme
print("on estime P(ak <= X < bk) par", He[0][k]*pas)
print("la valeur exacte vaut F_X(bk)-F_X(ak), soit :",
      np.exp(-lbd*ak)-np.exp(-lbd*bk))
```

```
le pas vaut : 1.2540898438677612
on estime P(ak <= X < bk) par 0.04309999999999997
la valeur exacte vaut F_X(bk)-F_X(ak), soit : 0.042243389973465884
```

Q. 3.5 : Utiliser la fonction `histo_FC()` et la bibliothèque `scipy.stats` pour confronter la fonction de répartition de la loi exponentielle de paramètre λ et l'histogramme des fréquences cumulées.

[26]:
```
plt.plot(X,expon.cdf(X, scale = 1/lbd), 'r', lw=2)
plt.grid()
Le = simul_loiExponentielleL(10000, lbd)
pas, He_c = histo_FC(Le, 30)
```



1.3.4 4. Loi normale centrée réduite.

On rappelle que X suit une loi normale de paramètre m et σ^2 si elle admet pour densité :

$$\varphi_{m,\sigma}(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-m)^2}{2\sigma^2}}$$

et pour fonction de répartition :

$$F_X(x) = \int_{-\infty}^{x} \varphi_{m,\sigma}(t)dt$$

On a par ailleurs :

$$\mathbb{E}(X) = m; \mathbb{V}(X) = \sigma^2$$

Q 4.1 : Écrire une fonction Python `simul_loiNormaleL(N)` qui renvoie une liste L formée de N réels considérés comme N réalisations indépendantes de la loi normale centrée réduite :

[27]:
```
from scipy.stats import norm
```

[28]:
```
def simul_loiNormaleL(N: int) -> list:
    return [norm.ppf(rdm.random()) for k in range(N)]
```

[29]:
```
Ln = simul_loiNormaleL(10)
print(Ln, end="")
```

```
[-0.21915527904353796, 0.21241382433231176, -0.3562910718814751,
0.848268387693075, -1.2163901154405319, -0.6510237987712244,
```

```
-0.11841986251199055, -0.24090269364664826, 0.1366820920908907,
-1.4848267723986304]
```

Q 4.2 : Écrire une fonction Python `simul_loiNormaleT(nl, nc)` qui renvoie un tableau T de taille (nl, nc) formé de $N = nl \times nc$ réels considérés comme \$N \$ réalisations indépendantes de la loi normale de paramètre 0 et 1:

[30]:
```
def simul_loiNormale(nl, nc):
    return nrdm.normal(0, 1, (nl, nc))
```

[31]:
```
simul_loiNormale(3, 5)
```

[31]:
```
array([[ 0.48401006, -0.19773506,  0.46567493, -0.29734273,  0.81357671],
       [ 0.87806954, -0.07389912, -0.4764351 ,  0.78323867, -0.1509684 ],
       [-1.41646878,  2.79232232, -0.62794523, -1.49744098,  0.76922112]])
```

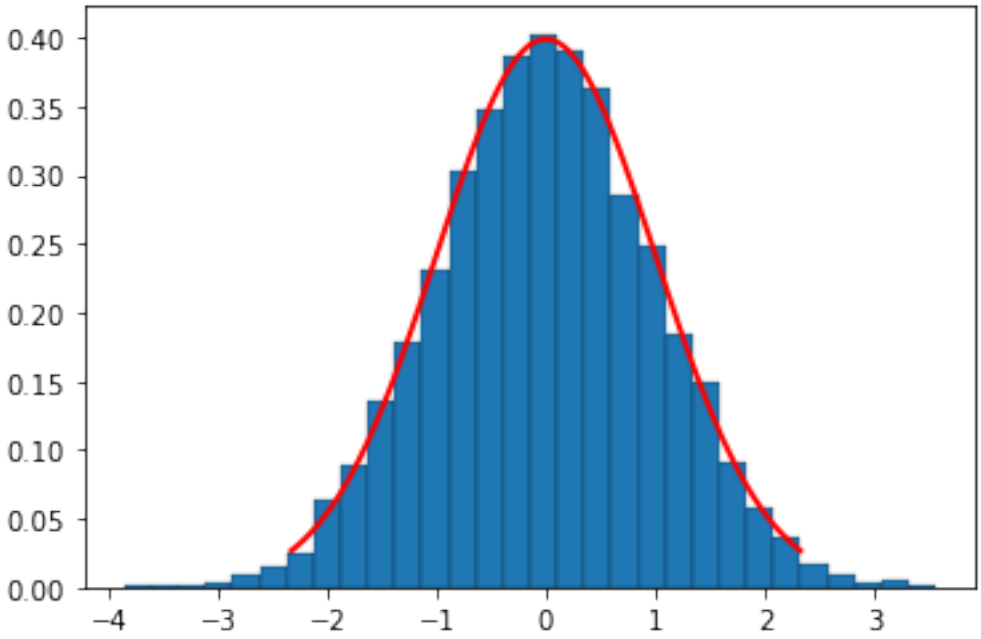
Q. 4.3 : Modéliser sous forme de liste la réalisation d'un nombre m supposé grand d'une variable aléatoire qui suit la loi normale de paramètre 0 et 1. Calculer la moyenne et la variance expérimentale des valeurs de cette liste et confronter à la moyenne et la variance de la loi normale centrée réduite :

[32]:
```
Ln = simul_loiNormaleL(10000)
print(f'moyenne et variance expérimentale : {np.mean(Ln)} et {np.var(Ln)}')
print(f"l'espérance et la variance de X valent : {0} et {1}")
```

```
moyenne et variance expérimentale : 0.005269904549597079 et 0.9988027729461614
l'espérance et la variance de X valent : 0 et 1
```

Q. 4.4 : Utiliser la bibliothèque `scipy.stats` pour tracer une densité de la loi normale centrée réduite et la confronter à l'histogramme de la série `Le` obtenu grâce à la fonction `histo`.

[33]:
```
a = norm.ppf(0.01) # FX(a) = P(X<=a) = 0.01
b = norm.ppf(0.99) # Fx(b) = P(X<=b) = 0.99 ou P(X>b) = 0.01
X = np.linspace(a,b,100)
plt.plot(X, norm.pdf(X), 'r', lw=2)
Ln = simul_loiNormaleL(10000)
pas, Hn = histo(Ln, 30)
```

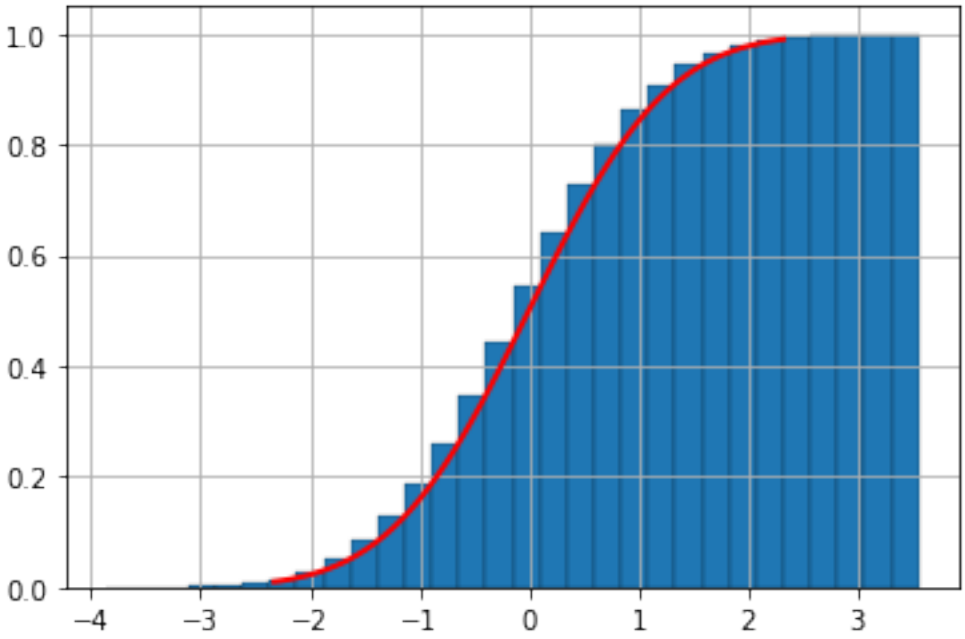


[34]:
```
print("le pas vaut :", pas)
m, M = min(Ln), max(Ln)
k = 15 # on s'intéresse, par exemple, à la 15ème classe de l'histogramme
      # n'hésitez pas à modifier cette valeur...
ak, bk = m+k*pas, m+(k+1)*pas # kème classe de l'histogramme
print("on estime P(ak <= X < bk) par", Hn[0][k]*pas)
print("la valeur exacte vaut F_X(bk)-F_X(ak), soit :",
      norm.cdf(bk)-norm.cdf(ak))
```

```
le pas vaut : 0.24726518374789175
on estime P(ak <= X < bk) par 0.09960000000000004
la valeur exacte vaut F_X(bk)-F_X(ak), soit : 0.09837045271153122
```

Q. 4.5 : Utiliser la fonction `histo_FC()` et la bibliothèque `scipy.stats` pour confronter la fonction de répartition de la loi normale centrée réduite et l'histogramme des fréquences cumulées.

[35]:
```
plt.plot(X,norm.cdf(X),'r',lw=2)
plt.grid()
pas, Hn_c = histo_FC(Ln, 30)
```



Q 4.5 : Écrire une fonction Python `simul_loiNormale2L(N, m, sigma)` qui renvoie une liste L formée de N réels considérés comme N réalisations indépendantes de la loi normale de paramètres m et σ :

[36]:
```
def simul_loiNormaleL(N: int, m: float, sigma: float) -> list:
    return [sigma*norm.ppf(rdm.random())+m for k in range(N)]
```

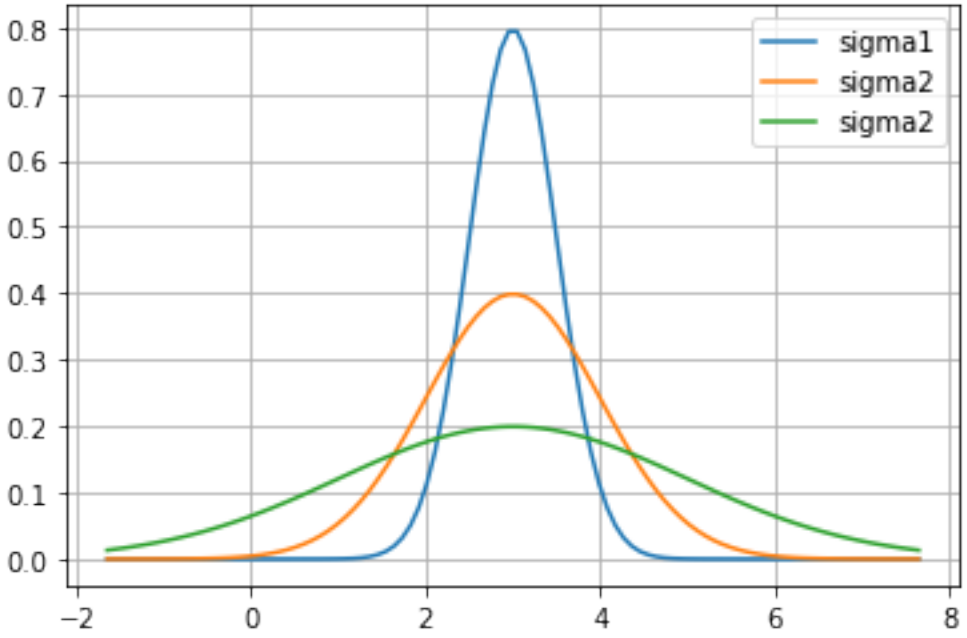
[37]:
```
Ln2 = simul_loiNormaleL(10, 3, 0.5)
print(Ln2, end="")
```

```
[2.8526442741901428, 3.8162527299946674, 2.3731130143490335, 3.352281946888982,
2.408039973558267, 2.8934330868158282, 3.287482960803008, 2.8359799548132942,
3.2731559235143792, 3.202833935646563]
```

Q. 4.6 : Tracer dans une même fenêtre une densité de la loi normale de paramètres $m = 3$ et $\sigma_1 = 0.5$, de la loi normale de paramètres $m = 3$ et $\sigma_2 = 1$ et de la loi normale de paramètres $m = 3$ et $\sigma_3 = 2$.

[38]:
```
m, sig1, sig2, sig3 = 3, 0.5, 1, 2
a = norm.ppf(0.01, loc=m, scale=sig3) #FY(a) = 0.01
b = norm.ppf(0.99, loc=m, scale=sig3) # FY(b) = 0.99
X = np.linspace(a, b, 100)
plt.plot(X, norm.pdf(X, loc=m, scale=sig1), label='sigma1')
plt.plot(X, norm.pdf(X, loc=m, scale=sig2), label='sigma2')
plt.plot(X, norm.pdf(X, loc=m, scale=sig3), label='sigma2')
plt.grid()
plt.legend(loc='best')
```

[38]: <matplotlib.legend.Legend at 0x7a77781ac0>



[ ]: