La loi binomiale

Marc Lorenzi - 31 mai 2018

In [1]:
```
import matplotlib.pyplot as plt
import random
import loi_binomiale
from math import sqrt, exp, pi
%matplotlib inline
```

1. Somme de variables de Bernoulli indépendantes

Soit Ω un univers muni d'une probabilité P. Soit $X : \Omega \to \{0, 1\}$ une variable aléatoire à valeurs dans $\{0, 1\}$. Soit $p \in [0, 1]$. On dit que X suit une loi de Bernoulli de paramètre t lorsque $P(X = 1) = p$. On a alors évidemment $P(X = 0) = 1 - p$.

1.1 Simuler des variables de Bernoulli

La fonction `rand_bernoulli` prend un réel $p \in [0, 1]$ en paramètre. Elle appelle la fonction `random` pour obtenir un réel aléatoire x entre 0 et 1. Si $0 \leq x \leq p$, la fonction renvoie 1. Sinon elle renvoie 0. Le générateur de nombres (pseudo-) aléatoires de Python est censé renvoyer des réels entre 0 et 1 suivant une loi uniforme. Du moins nous allons le croire. Ainsi, $P(0 \leq x \leq p) = \frac{\mu([0,p])}{\mu([0,1])} = p$, où $\mu(A)$ désigne la mesure de l'ensemble A (sa longueur). Bref, `rand_bernoulli` renvoie 1 avec une probabilité p et 0 sinon. C'est exactement ce qu'il nous fallait.

In [2]:
```
def rand_bernoulli(p):
    x = random.random()
    if x <= p: return 1
    else: return 0
```

In [3]:
```
print([rand_bernoulli(0.3) for k in range(100)])
```

```
[1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0
, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0,
1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
0, 1, 1, 0, 1, 0, 0, 0, 1]
```

1.2 Additionner des variables de Bernoulli indépendantes

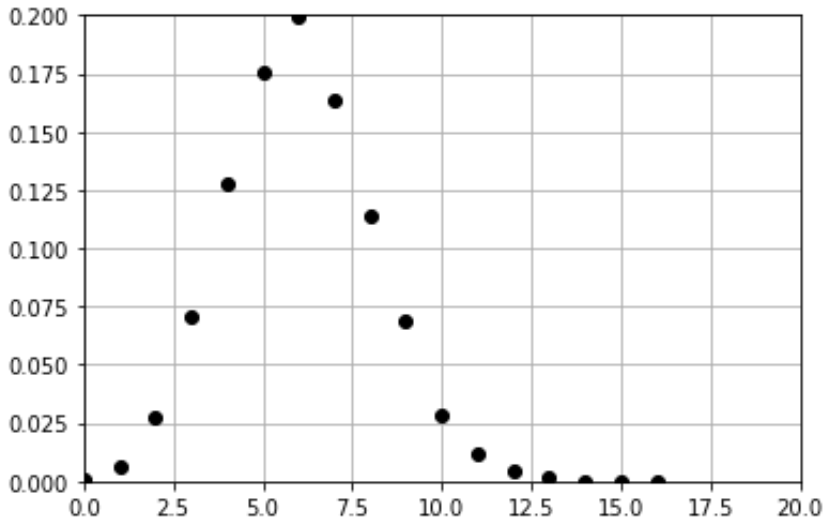
C'est bien connu, la somme de n variables aléatoires suivant une loi de Bernoulli de paramètre p est une v.a. suivant une loi binomiale de paramètres n, p. Testons.

In [4]:
```
def rand_binomial(n, p):
    s = 0
    for k in range(n):
        s = s + rand_bernoulli(p)
    return s
```

On écrit une petite fonction qui trace l'histogramme d'une liste de nombres, puis on trace.

In [5]:
```
def histogramme(s):
    m = max(s)
    h = (m + 1) * [0]
    c = 0
    for k in s:
        h[k] = h[k] + 1
        c = c + 1
    for i in range(len(h)): h[i] = h[i] / c
    return h
```

In [6]:
```
n = 20
p = 0.3
s = [rand_binomial(n, p) for k in range(10000)]
plt.plot(histogramme(s), 'ko')
plt.axis([0,n,0,0.2])
plt.grid()
plt.show()
```



1.3 La loi binomiale

1.3.1 Coefficients binomiaux

La fonction `power_down` prend en paramètres deux entiers n et k. Elle renvoie $(n)_k = n(n-1)\dots(n-k+1)$.

In [7]:
```
def power_down(n, k):
    p = 1
    for j in range(k): p = p * (n - j)
    return p
```

In [8]:
```
power_down(5, 3)
```

Out[8]:
```
60
```

Cas particulier : $k = n$. On obtient la factorielle.

In [9]:
```
def factorial(n):
    return power_down(n, n)
```

In [10]:
```
factorial(10)
```

Out[10]:
```
3628800
```

La fonction `binomial` prend en paramètres deux entiers n et k Elle renvoie le coefficient binomial $\binom{n}{k}$.

In [11]:
```
def binomial(n, k):
    return power_down(n, k) // factorial(k)
```

In [12]:
```
[binomial(5, k) for k in range(6)]
```

Out[12]:
```
[1, 5, 10, 10, 5, 1]
```

Si X est une variable aléatoire suivant une loi binomiale de paramètres n et p, on a pour $0 \leq k \leq n$, $P(X = k) = \binom{n}{k} p^k (1-p)^{n-k}$. La fonction `binom` renvoie précisément cette valeur.

In [13]:
```
def binom_law(n, p, k):
    return binomial(n, k) * p ** k * (1 - p) ** (n - k)
```

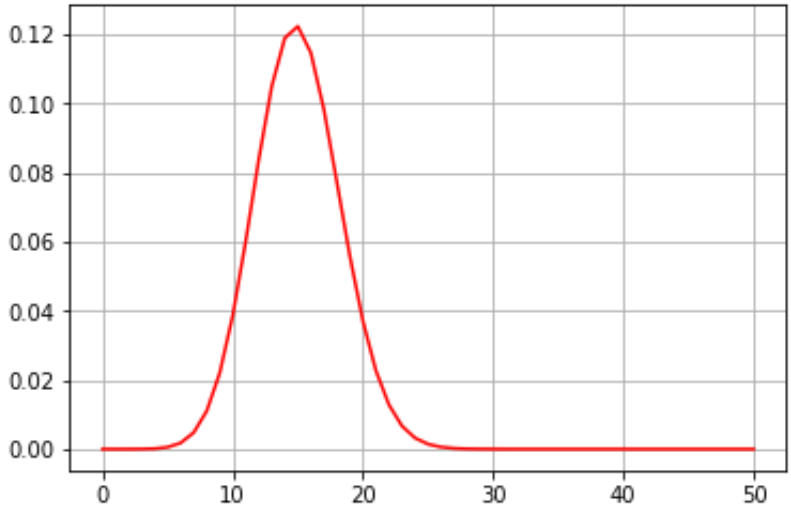
In [14]:
```
[binom_law(5, 0.5, k) for k in range(6)]
```

Out[14]:
```
[0.03125, 0.15625, 0.3125, 0.3125, 0.15625, 0.03125]
```

1.3.2 Histogramme

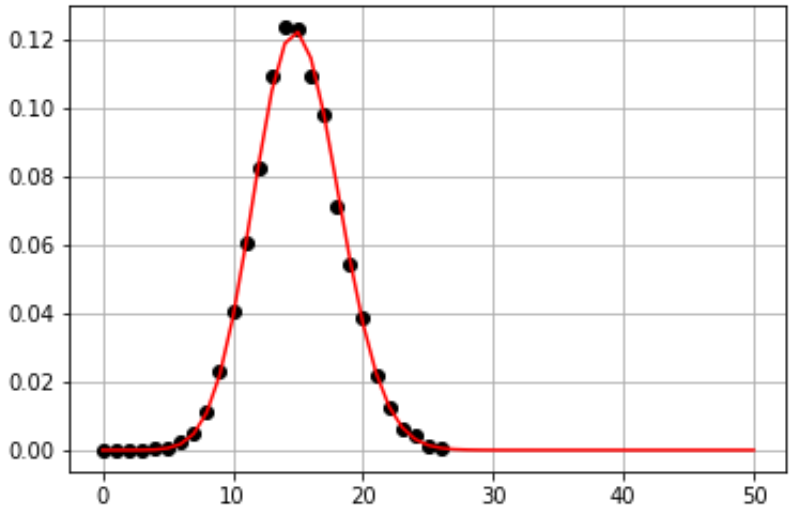
Voici l'histogramme "théorique" pour la loi binomiale.

In [16]:
```
n, p = 50, 0.3
s = [binom_law(n, p, k) for k in range(0, n + 1)]
plt.grid()
plt.plot(s, 'r-')
plt.show()
```



On le superpose à l'histogramme de la somme de variables de Bernoulli obtenues avec la fonction `random` de Python ? Cela devrait coller.

In [17]:
```
n, p = 50, 0.3
s = [rand_binomial(n, p) for k in range(10000)]
#plt.axis([0,n,0,0.2])
plt.plot(histogramme(s), 'ko')
s1 = [binom_law(n, p, k) for k in range(0, n + 1)]
plt.grid()
plt.plot(s1, 'r-')
plt.show()
```



Eh oui, ça colle. Les maths ça marche :-).

Le fichier `loi.binomiale.py` contient une simulation interactive de la loi binomiale. La cellule ci-dessous l'exécute mais vous devez refermer la fenêtre pour poursuivre le notebook. Vous pouvez bien sûr lancer cette simulation séparément en tapant dans un terminal la ligne de commande `python loi_binomiale.py` .

In [18]:
```
loi_binomiale.run_gui()
```

En noir, l'histogramme de la loi binomiale. Vous pouvez régler les valeurs de n (entre 1 et 100) et p (entre 0 et 1). Mais que représentent les traits rouges, les traits bleus et la courbe verte ?

- Les traits rouges représentent la fonction de répartition de la loi binomiale : si X suit une loi $\mathcal{B}(n,p)$, le trait rouge à l'abscisse k a pour ordonnée $P(X \leq k)$.
- Les traits bleus représentent l'histogramme d'une loi de Poisson de paramètre $\lambda = np$.
- La courbe verte est la courbe de densité d'une loi de Gauss d'espérance $m = np$ et d'écart-type $\sigma = \sqrt{np(1-p)}$.

Nous en reparlons un peu plus loin.

1.4 Espérance et variance

L'espérance et l'écart-type d'une variable aléatoire suivant une loi binomiale de paramètres n et p sont respectivement $m = np$ et $\sigma = \sqrt{np(1-p)}$. Faisons une petite simulation.

In [19]:
```
def esperance_simulee(X, n=10000):
    s = 0
    for k in range(n):
        s = s + X()
    return s / n
```

In [20]:
```
def esperance(n, p): return n * p
```

In [21]:
```
n, p = 20, 0.3
print(esperance_simulee(lambda : rand_binomial(n, p)))
print(esperance(n, p))
```

```
6.0057
6.0
```

In [22]:
```
def ecart_type_simule(X, n=10000):
    return sqrt(esperance_simulee(lambda : X() ** 2, n) - esperance_si
```

In [23]:
```
def ecart_type(n, p):
    return sqrt(n * p * (1 - p))
```

In [24]:
```
n, p = 20, 0.3
print(ecart_type_simule(lambda: rand_binomial(20, 0.3)))
print(ecart_type(n, p))
```

```
2.0459315726582847
2.0493901531919194
```

2 Convergence vers la loi de Poisson

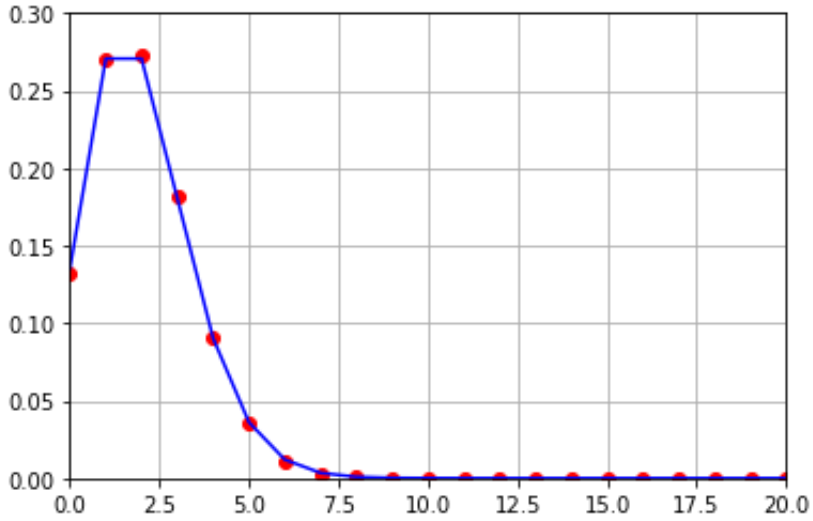
Soit $X : \Omega \to \mathbb{N}$ une variable aléatoire. Soit $\mu > 0$. On dit que X suit une loi de Poisson de paramètre μ lorsque, pour tout entier k, $P(X = k) = e^{-\mu}\frac{\mu^k}{k!}$.

In [25]:
```
def poisson(mu, k):
    return exp(-mu) * mu ** k / factorial(k)
```

Soit $\mu > 0$. pour tout $n \in \mathbb{N}$, soit $p_n = \frac{\mu}{n}$. On peut montrer que, dans un certain sens, la loi binomiale $\mathcal{B}(n, p_n)$ approche la loi de Poisson de paramètre μ.

Ci-dessous, l'histogramme d'une loi de Poisson de paramètre μ, superposée aux valeurs des probabilités pour une loi binomiale de paramètres n et p_n .

In [26]:
```
mu = 2
n = 100
p = mu / n
s = [binom_law(n, p, k) for k in range(0, n + 1)]
s1 = [poisson(mu, k) for k in range(0, n + 1)]
plt.grid()
plt.axis([0,20,0,0.3])
plt.plot(s, 'ro')
plt.plot(s1, 'b-')
plt.show()
```



3 Convergence vers la loi normale

Soient $m \in \mathbb{R}$ et $\sigma > 0$. La loi normale de paramètres m et σ est une loi de probabilité P définie sur un ensemble $\mathcal{T}$ de parties de $\mathbb{R}$ que je ne décrirai pas ici. Disons seulement que $\mathcal{T}$ contient tous les intervalles et qu'il est stable par réunions et intersections dénombrables (mais pas quelconques !). Si X est une variable aléatoire réelle suivant la loi normale $\mathcal{N}(m, \sigma)$, on a en particulier, pour tout réel x :

$$P(X \leq x) = \frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^{x} \exp(-\frac{(t-m)^2}{2\sigma^2})dt$$

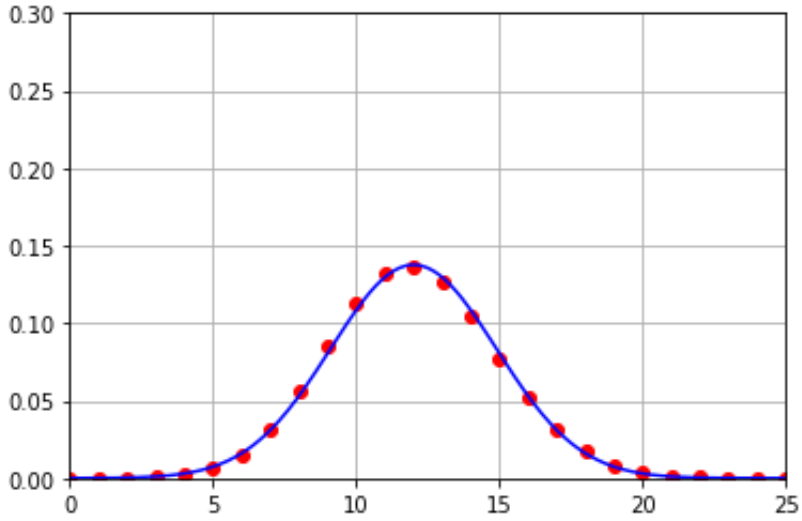
Plus généralement, soit la fonction $f_{m,\sigma} : x \mapsto \frac{1}{\sigma\sqrt{2\pi}} \exp(-\frac{(x-m)^2}{2\sigma^2})$. Cette fonction est appelée la fonction de densité de la loi normale. La probabilité d'une partie A adéquate de $\mathbb{R}$ s'obtient en intégrant la fonction de densité sur l'ensemble A :

$$P(A) = \int_A f_{m,\sigma}$$

Je n'entrerai pas dans les détails ici mais pour de grandes valeurs de n et des valeurs de k pas trop petites, mais petites devant n, la la loi normale $\mathcal{N}(np, \sqrt{np(1-p)})$ est "approchée" par la loi binomiale $\mathcal{B}(n, p)$. L'énoncé précis du résultat n'est pas simple, contentons nous d'une simulation.

In [27]:
```
def normal(x, m, sigma):
    return 1 / (sigma * sqrt(2 * pi)) * exp(-(x - m) ** 2 / (2 * sigma
```

In [28]:
```
n = 40
p = 0.3
m = n * p
sigma = sqrt(n * p * (1 - p))
s = [binom_law(n, p, k) for k in range(0, n + 1)]
x = [k / 10 for k in range(10 * (n + 1))]
s1 = [normal(k / 10, m, sigma) for k in range(0, 10 * (n + 1))]
plt.grid()
plt.axis([0,25,0,0.3])
plt.plot(s, 'ro')
plt.plot(x, s1, 'b-')
plt.show()
```



In [ ]:
```

```