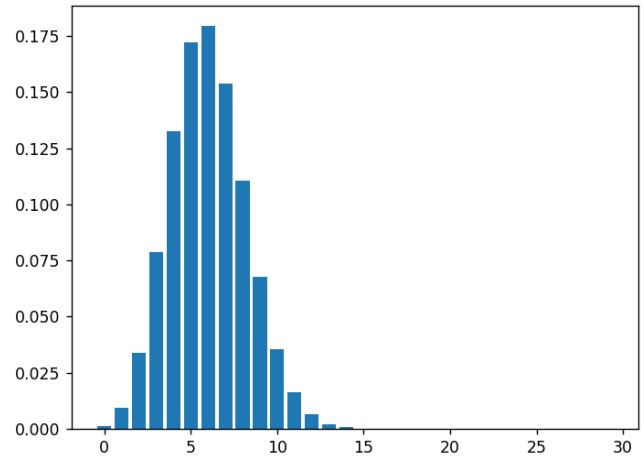
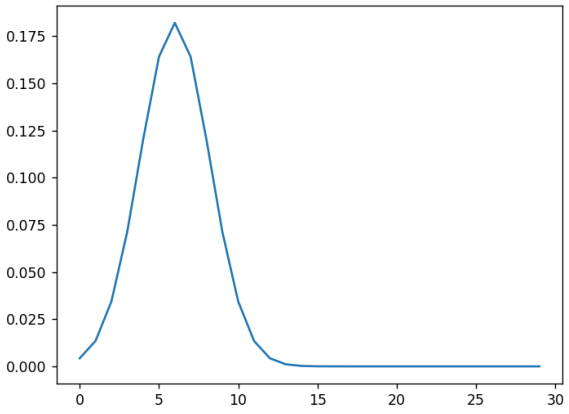
TP Informatique

Approximation d'une loi Binomiale par une loi Normale

L'objectif de ce travail est de produire un code qui permet de valider le fait de pouvoir approcher une loi Binomiale $\mathcal{B}(n\ ,p)$par une loi Normale $\mathcal{N}(\mu\ ,\sigma)$:

> Propriété : Si une variable aléatoire X suit une loi Binomiale $\mathcal{B}(n,p)$ et si $n \geq 30$, $np \geq 5$ et $n(1-p) \geq 5$, cette loi peut être approchée par une loi Normale $\mathcal{N}(\mu\ ,\sigma)$ avec $\mu = n\,p$ et $\sigma = \sqrt{np(1-p)}$

On exécutera ce code sur la situation suivante, déjà étudiée en exercice :
« Un atelier produit des pièces, dont 20 % sont d'excellentes qualité. On prélève successivement au hasard 30 pièces dans la production. On note X le nombre d'excellentes qualités dans le lot. On décide d'approcher X par une loi normale »

Résolution en utilisant la loi Binomiale	Résolution en utilisant la loi Normale
X suit une loi Binomiale $\mathcal{B}(n\ ,p)$ avec $n = 30$ et $p = 0.20$. La probabilité d'obtenir k pièces d'excellentes qualités dans le lot de n pièces est : $p(X=k) = \binom{n}{k}\ p^k\ (1-p)^{n-k}$ Le coefficient $\binom{n}{k}$ peut se calculer en utilisant le factoriel : $\binom{n}{k} = \frac{n!}{k!\,(n-k)!}$ Si on calcule $p(X=k)$ pour toutes les valeurs de k comprises entre 0 et 30, on peut tracer l'histogramme ci-dessous : 	X suit une loi Binomiale $\mathcal{N}(\mu\ ,\sigma)$. On calcule la moyenne μ et l'écart-type σ : $\mu = n\,p = 30 \times 0.20 = 6$ $\sigma = \sqrt{np(1-p)} = \sqrt{30\ \times 0.20 \times (1-0.2)} \approx 2.20$ X suit une loi Normale $\mathcal{N}(6\ ,2.20)$. La fonction de densité f est alors la suivante : $f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2}$ $f(x) = \frac{1}{2.20\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-6}{2.20})^2}$ Si on trace la courbe représentative de cette fonction f, on obtient le tracé suivant : 

Ces 2 diagrammes sont-ils équivalents, permettent-ils d'obtenir les mêmes résultats sur les probabilités ? On se propose dans ce TP, de tracer l'histogramme de la loi Binomiale et la courbe de la loi Normale sur le même graphe, afin de pouvoir les comparer

⇨ Ouvrir un nouveau fichier sur Pyzo et l'enregistrer sous le nom *loiBinomialeNormale_monNom.py* .

1. FONCTION FACT() :

La fonction *fact()* donnée ci-contre est incomplète. Elle a comme paramètre un nombre entier n et renvoie le nombre $n! = n \times (n-1) \times (n-2) \times \ldots \times 2 \times 1$

```
def fact(n) :


    return
```

⇨ Ecrire ce code en le complétant, dans votre fichier *loiBinomiale _monNom.py*.

⇨ Vérifier le bon fonctionnement de votre script en exécutant dans la console les tests suivants :

Tests simples :

```
>>> fact(3)
6
```

```
>>> fact(8)
40320
```

Tests avec des valeurs limites :

```
>>> fact(0)
1
```

```
>>> fact(1)
1
```

```
>>> fact(20)
2432902008176640000
```

CORRIGE

```
def fact(n) :
    f = 1
    for i in range(1,n+1)
        f = f * i
    return f
```

2. : FONCTION COEFBIN() :

La fonction *coefBin()* donnée ci-contre est incomplète. Elle a comme paramètre 2 nombres entiers n et k . Elle renvoie le nombre

```
def coefBin(n,k) :


    return
```

$$\binom{n}{k} = \frac{n!}{k!\ (n-k)!}$$

⇨ Ecrire ce code en le complétant, dans votre fichier *loiBinomialeNormale _monNom.py*. Utiliser bien sur la fonction *fact()* mises au point précédemment0

⇨ Vérifier le bon fonctionnement de votre script en exécutant dans la console les tests suivants :

Tests simples :

```
>>> coefBin(8,4)
70
```

```
>>> coefBin(2,1)
2
```

Tests avec des valeurs limites

```
>>> coefBin(100,1)
100
```

```
>>> coefBin(1,1)
1
```

```
>>> coefBin(100,100)
1
```

```
>>> coefBin(100,50)
100891344545564202071714955264
```

```
def coefBin(n,k) :
    c = fact(n) / (fact(k) * fact(n-k))
    return int(c)
```

CORRIGE

3. : FONCTION LOIBIN() :

```
def loiBin(n,p,k) :


    return
```

La fonction *loiBin()* donnée ci-contre est incomplète. Elle a comme paramètre 2 nombres entiers n , k et un nombre réel p. Elle renvoie le nombre

$$p(X = k) = \binom{n}{k}\ p^k\ (1-p)^{n-k}$$

⇨ Ecrire ce code en le complétant, dans votre fichier *loiBinomialeNormale _monNom.py*. Utiliser bien sur les fonctions *fact()* et *coefBin()* mises au point précédemment. Pour rappel, l'opération puissance en python est repérée par ** . par exemple $0.05^{15} = 0.05 ** 15$

⇨ Vérifier le bon fonctionnement de votre script en exécutant dans la console les tests suivants :

Test simples: Pour calculer $p(X = 1)$ sur la loi Binomiale $\mathcal{B}(5,0.05)$ on exécute :

```
>>> loiBin(5,0.05,1)
0.2036256249999997
```

On a ainsi $p(X = 1) \approx 0{,}204$

```
def loiBin(n,p,k) :
    proba = coefBin(n,k) * p**k * (1-p)**(n-k)
    return proba
```

CORRIGE

4. : FONCTION HISTOGRAMME() :

La fonction *histogramme()* donnée ci-contre est incomplète. Elle a comme paramètre 1 nombre entier n , un nombre réel p et une liste de nombres entiers $[k_0, k_1, k_2, k_3 \dots]$. Elle renvoie une liste contenant les probabilités $[p(X = k_0), p(X = k_1), p(X = k_2), p(X = k_3) \dots]$.

```
def histogramme(n,p,X) :
    listeProba = []



    return listeProba
```

⇨ Ecrire ce code en le complétant, dans votre fichier *loiBinomialeNormale _monNom.py*. Utiliser bien sur la fonctions *loiBin()* miss au point précédemment.

⇨ Vérifier le bon fonctionnement de votre script en exécutant dans la console les tests suivants :

Test simples: Sur la loi Binomiale $\mathcal{B}(3\ ,0.20)$ on exécute :

```
>>> histogramme(3,0.20,[0,1,2,3])
[0.5120000000000001, 0.3840000000000001, 0
.09600000000000003, 0.008000000000000002]
```

```
def histogramme(n,p,X) :
    listeProba = []
    for k in X :
        proba = loiBin(n,p,k)
        listeProba.append(proba)
    return listeProba
```

CORRIGE

5. : FONCTION CALCULMOYSIG() :

La fonction *calculMoySig()* donnée ci-contre est incomplète. Elle a comme paramètre 1 nombre entier n , p et un nombre réel p. Elle renvoie les 2 nombres réels suivants :

$$\mu = n\,p \qquad \sigma = \sqrt{np(1-p)}$$

```
def calculMoySig(n,p) :
    moy =
    sig =
    return moy,sig
```

⇨ Ecrire ce code en le complétant, dans votre fichier *loiBinomialeNormale _monNom.py*.

```
def calculMoySig(n,p) :
    moy = n * p
    sig = sqrt(n * p* (1-p))
    return moy,sig
```

CORRIGE

6. : FONCTION FDENSITE() :

La fonction *fDensite()* donnée ci-contre est incomplète. Elle a comme paramètre 2 nombres réels n , p et une liste de nombres entiers $[x_0, x_1, x_2, x_3 \dots]$. Elle renvoie une liste contenant les images par la fonction $f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2}$: $[f(x_0), f(x_1), f(x_2), f(x_3), \dots]$.

```
def fDensite(moy,sig,X) :
    Y = []
    for x in X :

        Y.append(y)
    return Y
```

⇨ Ecrire ce code en le complétant, dans votre fichier *loiBinomialeNormale _monNom.py*.

CORRIGE

```
def fDensite(moy,sig,X) :
    Y = []
    for x in X :
        y = 1/(sig*sqrt(2*pi))*exp(-0.5*((x-moy)/sig)**2)
        Y.append(y)
    return Y
```

7. : FONCTION TRACE() :

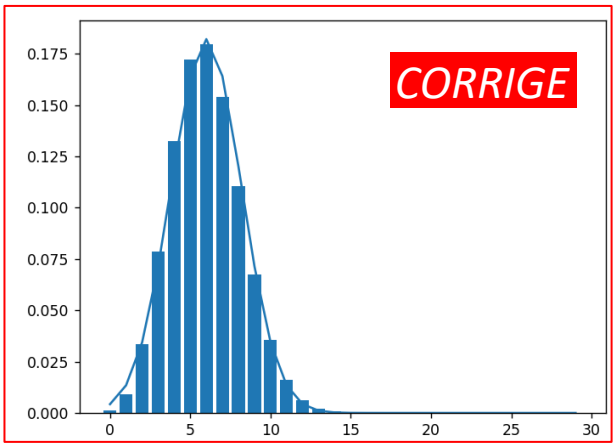
La fonction *trace()* donnée ci-contre a comme paramètre 2 nombres réels n , p. Elle permet de tracer sur le même graphe l'histogramme issue de la loi Binomiale et la courbe représentative de la fonction de densité issue de la loi Normale :

```
def trace(n,p) :
    X = [i for i in range(n)]
    Ybinomiale = histogramme(n,p,X)
    moy,sig = calculMoySig(n,p)
    Ynormale = fDensite(moy,sig,X)
    plt.bar(X, Ybinomiale)
    plt.plot(X,Ynormale)
    plt.show()
```

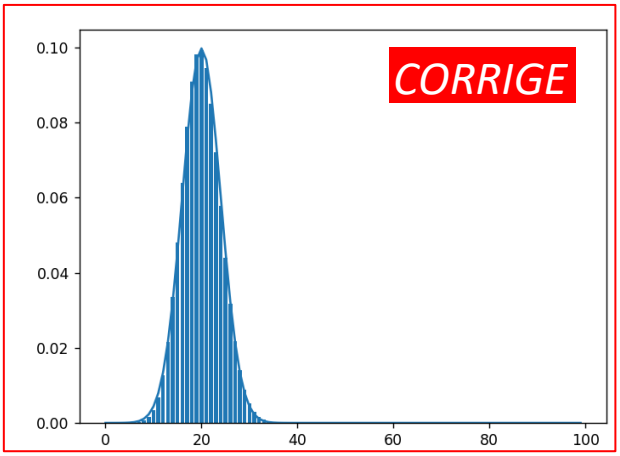
⇨ Ecrire ce code en le complétant, dans votre fichier *loiBinomialeNormale _monNom.py*.

⇨ Tester les exécutions suivantes : `>>> trace(30,0.20)` , `>>> trace(100,0.20)` , etc …

CORRIGE



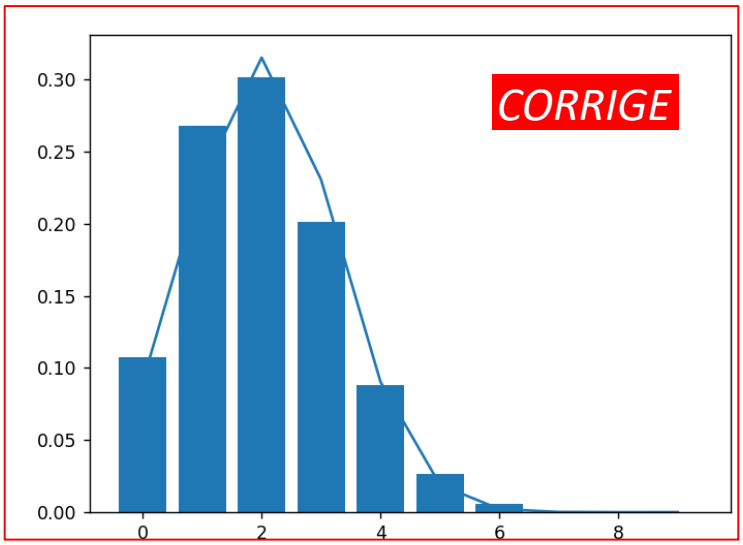
CORRIGE



⇨ Placer les copies d'écran des exécutions dans un fichier word que vous nommerez : *loiBinomialeNormale _monNom.docx*. Tester la correspondance loi Binomiale – Loi Normale en vous plaçant en limite des critère $n \geq 30$, $np \geq 5$, $n(1-p) \geq 5$.

Faire un compte-rendu à rédiger dans ce document word.

Si $p = 0.20$, pour avoir $np = 5$, on doit avoir $n = \frac{5}{0.20} = 10$. La confrontation des résultats calculés par la loi Binomiale $\mathcal{B}(10\ , 0.20)$ avec ceux calculés par une approximation loi Normale donne alors :



Si $p = 0.20$, pour avoir $n(1 - p) = 5$, on doit avoir $n = \frac{5}{1-0.20} \approx 6$. La confrontation des résultats calculés par la loi Binomiale $\mathcal{B}(6\ , 0.20)$ avec ceux calculés par une approximation loi Normale donne alors :