Rapport sur le TP de probabilité et statistiques

Wassim Saidane, Aurélien Authier

Table des matières

Introduction

Dans ce TP nous allon modéliser différents problèmes de probabilité et statistique en python

1 Première partie : Régression linéaire

Cette partie correspond au fichier AuthierSaidanePart1.py

1.1 Régression Linéaire simple

Soit le modèle linéire suivant :

$$Y_1 = \beta_0 + \beta_1 x_i$$

Avant de calculer le coeficient de régression nous avons créer une fonction moyenne.

```
        def moyenne(tab):
            return sum(tab)/len(tab)

```

Ainsi nous pouvons faire la moyenne de différentes valeurs. Une fois la fonction moyenne codée on peut faire la fonction les coeficients de régressions. Voici la formule du coeficient de régression :

$$a = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_1 - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

$$b = \bar{y} - a\bar{x}$$

En python celà se traduit par :

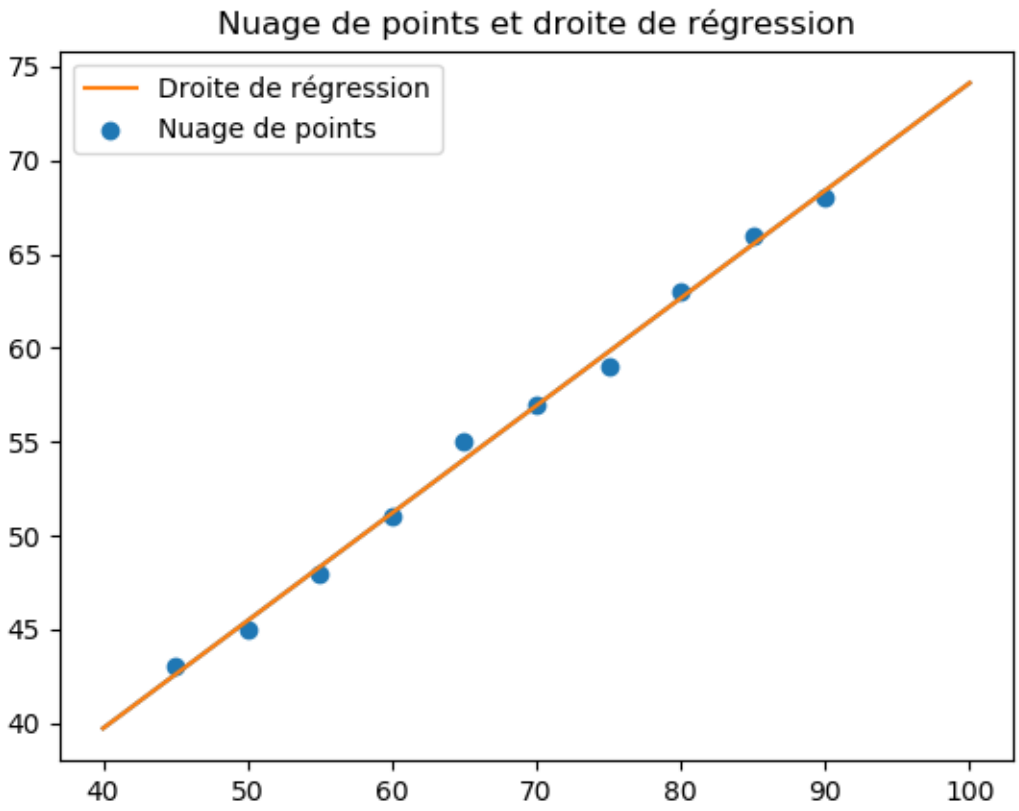
```
        def coef_regression(tab1, tab2):
            beta_1 = 0
            temp1 = 0
            beta_0 = 0
            temp2 = 0
            for i in range(0, len(tab1)):
                temp1 += (tab1[i]-moyenne(tab1))*(tab2[i]-moyenne(
    tab2))
                temp2 += (tab1[i]-moyenne(tab1))**2
            beta_1 = temp1/temp2
            beta_0 = moyenne(tab2)-beta_1*moyenne(tab1)
            return (beta_0, beta_1)

```

Les deux tableaux en paramètres de notre fonction sont les échantillons du sujet. beta_0 correspond à b et beta_1 correspond à a. On obtient :
b=0.5733333333333334
a=16.799999999999997

Pour la représentation graphique nous avons utilisé l'outil pyplot de la bibliothèque matplotlib.

```
        coefs = coef_regression(x_i, y_i)
        x = np.linspace(40, 100)
        plt.plot(x, coefs[0]+coefs[1]*x)
        plt.scatter(x_i, y_i, label="Nuage de points")
        p = np.polyfit(x_i, y_i, 1)
        plt.plot(x, p[1]+p[0]*x, label="Droite de regression")
        plt.legend()
        plt.title("Nuage de points et droite de regression")
        plt.show()

```

On obtient le graphique suivant :



Soit le modèle vectoriel suivant :

$$y = A\beta$$

Pour pouvoir faire des opérations sur les matrices nous avons utilisé la bibliothèque numpy.
Nous avons coder la formule $(A^T A)^{-1} A^T y)$.
Tout d'abord nous avons implémenter deux petites fonctions permettant de determiné A, A^T et Y.

```
        def matA(x):
            vect = np.ones(len(x))
            Atranspose = np.vstack((vect, x))
            A = np.transpose(Atranspose)
            return (A,Atranspose)
        def matY(y):
            return np.matrix(y_i).transpose()

```

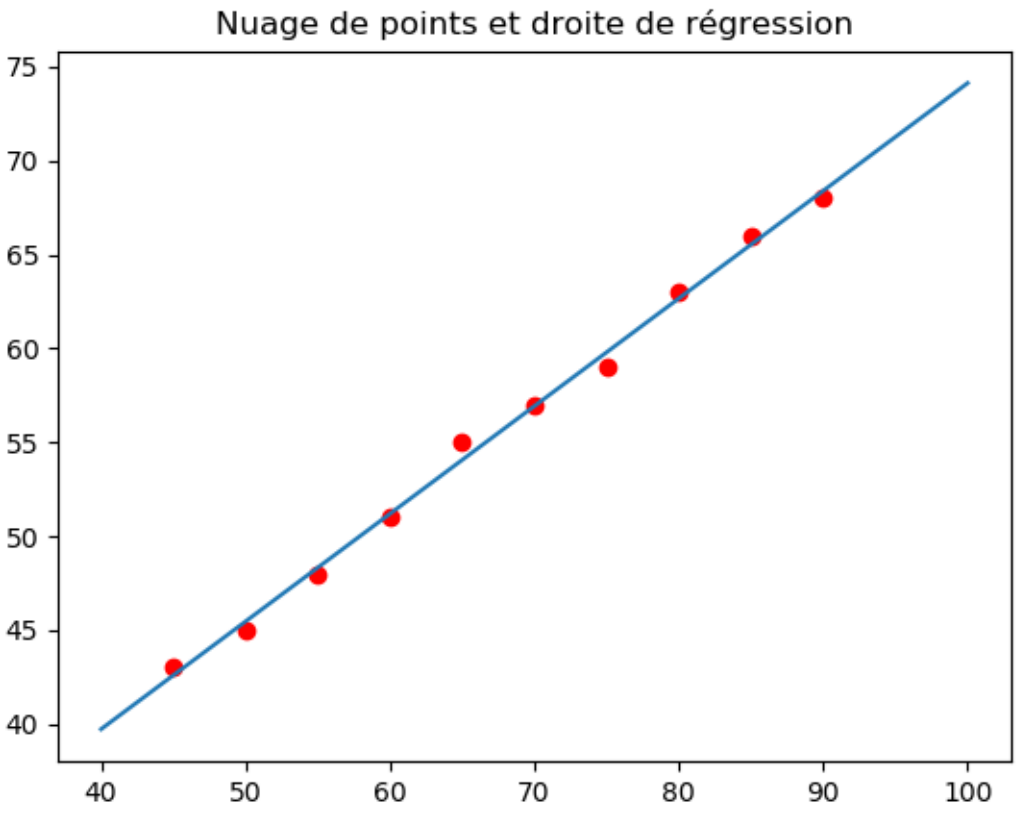
Ensuite en deux opérations nous avons calculer $A^T A$ et $(A^T A)^-1$.

```
        A_transposeA = np.dot(A_transpose, A)
        inv_ATA = np.linalg.inv(A_transposeA)

```

Pour finir nous avons implémenter la formule demandé :

```
        def matBeta(inv,Atranspose,Y):
            B1=np.dot(inv,Atranspose)
            B2=np.dot(B1,Y)
            return np.asarray(B2).reshape(-1)

```

On obtien ainsi Beta : [16.8 0.57333333]
Ansi que la graphique suivant : (voir page suivante)


Nuage de points et droite de régression
75
70
65
60
55
50
45
40
40
50
60
70
80
90
100

1.2 Régression linéaire et descente de gradient

2 Deuxième partie : Etude et manipulation de lois de probailités

2.1 Loi Binomiale

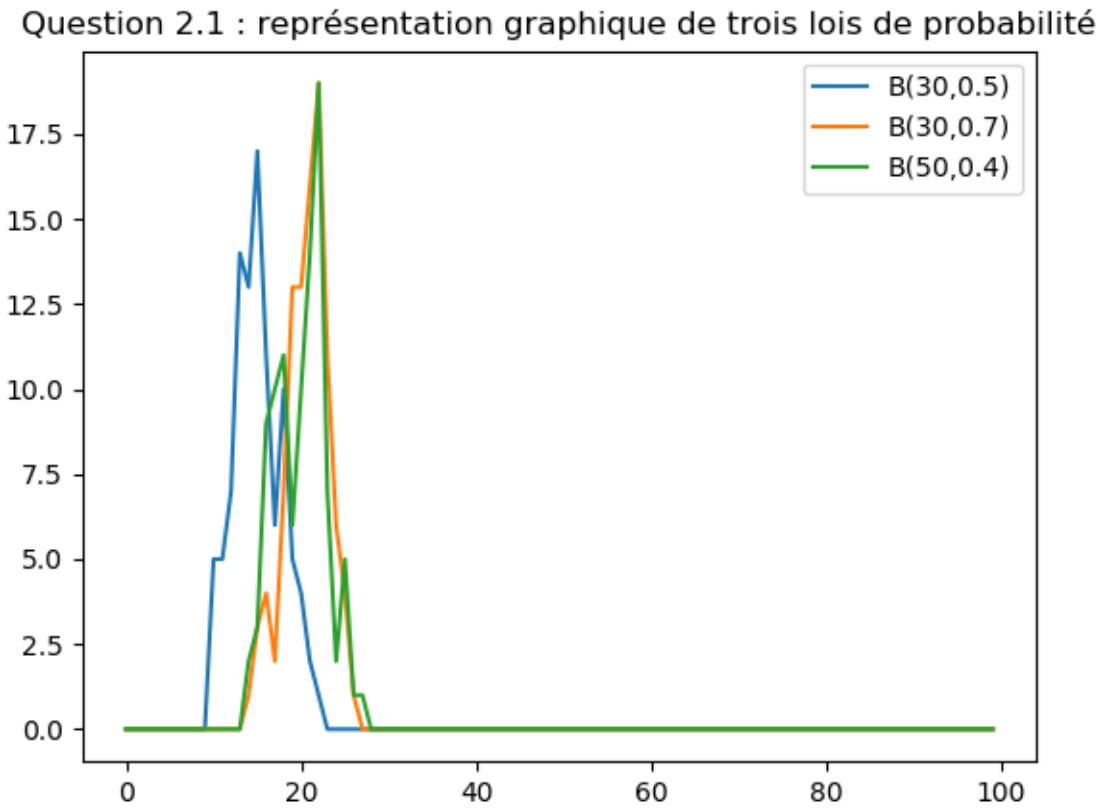
Nous avons modéliser en python une loi binomiale, pour celà nous avons utiliser la fonction python np.random.binomial.

```
        def binomial (n,p):
            valeur = []
            binom = []
            for i in range(100):
                binom.append(np.random.binomial(n, p))

            for i in range(100):
                valeur.append(binom.count(i))

        return valeur

```

Enusuite on ajoute les 3 expériences.

```
        exp1 = binomial(30, 0.5)
        exp2 = binomial(30, 0.7)
        exp3 = binomial(50, 0.4)

```

On obtient ainsi le résultat suivant : (page suivante)


Question 2.1 : représentation graphique de trois lois de probabilité
B(30,0.5)
B(30,0.7)
B(50,0.4)
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
0
20
40
60
80
100

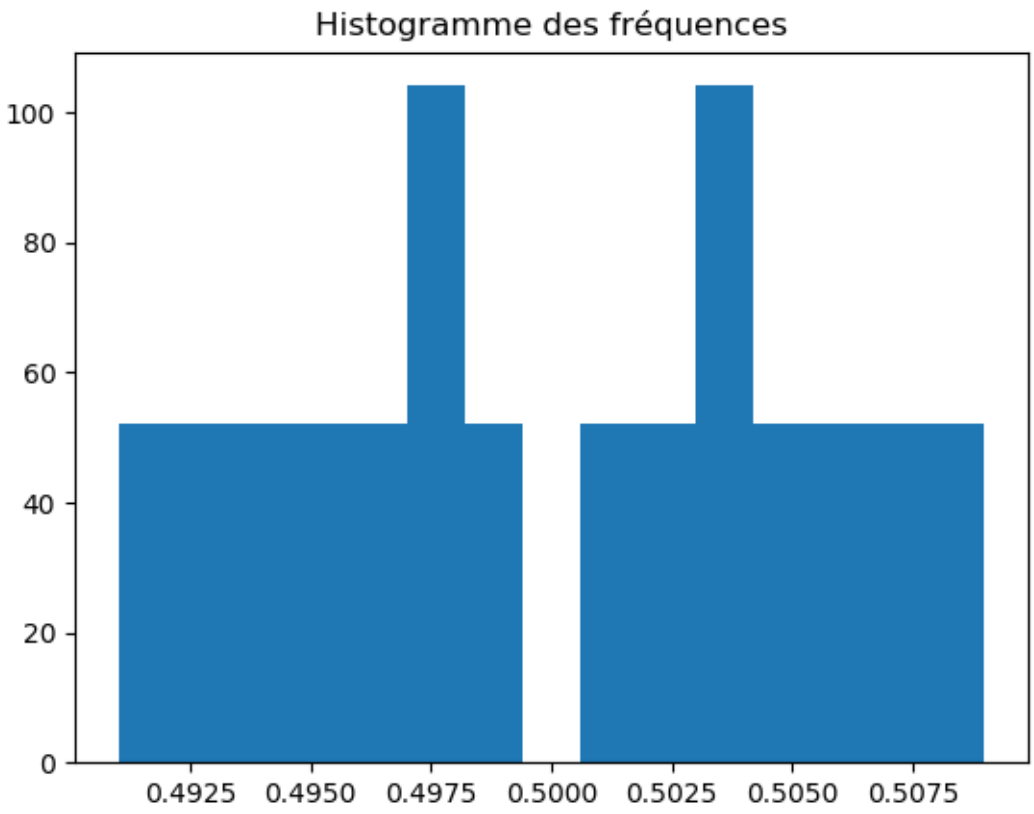
3 Troisième partie : Intervalle de confiance

3.1 Probléme 1

On calcul la moyenne empirique :

```
1         moyenne = sum(poids)/len(poids)
2
```

Ensuite on affiche l'histogramme grace aux fonctions python et on obtien :



On calcul l'écart type :

```
        def ecart_type_emp(data):
            moyenne = sum(data)/len(data)
            somme = 0
            for valeur in data:
                somme += (valeur-moyenne)**2
            return sqrt(somme/(len(data)-1))

```

L'intervalle à 95% et 99% on utilise les formules classisques. Par exemple, pour l'intervalle à 95%,

```
        intervalC95 = [moyenne - interval95*et/sqrt(len(poids)),
        moyenne + interval95*et/sqrt(len(poids))]

```

3.2 Probléme 2

Pour calculer l'intervalle de confiance :

```
        n = 500
        proba = 95/n
        ecart_type = sqrt(proba*(1-proba)/n)
        quantile = 2.576

        intervalC99 = [proba-quantile*ecart_type/sqrt(n), proba+
    quantile*ecart_type/sqrt(n)]
        print(f"Compagnie : Interval a 99% = {intervalC99}")

```

La proportion d'individus satisfaits de la compagnie aérienne est comprise entre 18.8% et 19.2%.

3.3 Probléme 3

Nous avons pris un effectif de 100.

```
        n = 100
        binomiale = np.random.binomial(n, 0.5, 500)
        ecart_type = sqrt(100*0.5*(1-0.5))
        moyenne = sum(binomiale)/len(binomiale)

        intervalC95 = [moyenne - 1.96*ecart_type/sqrt(n), moyenne +
     1.96*ecart_type/sqrt(n)]
        print(f"Binomiale : interval a 95% = {intervalC95}")

```