Benjamin Rogé / Edouard Poupa

Compte-rendu TP2 Math/Info

Exercice 1 :

Le but de cet exercice est d'écrire une fonction permettant d'approcher au sens des moindres carrés la fonction cos sur l'intervalle [0, π] à l'aide de polynômes, lorsque l'on ne dispose que d'un échantillon bruité de cette fonction.

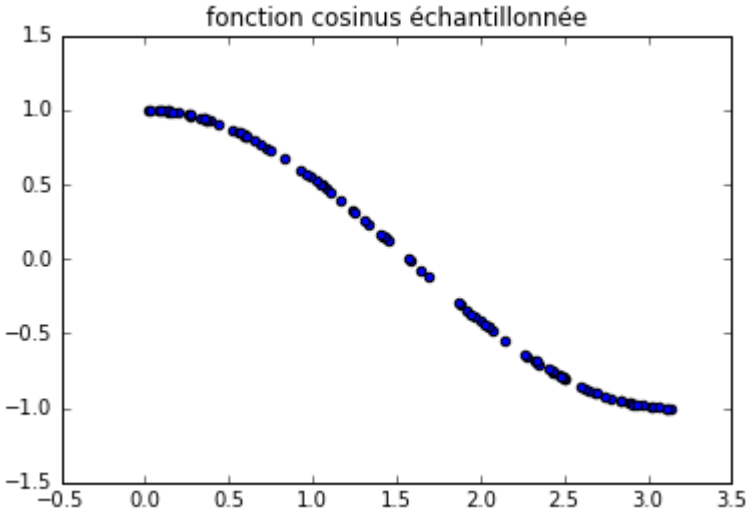
1)

Tout d'abord, on construit un échantillon de n=100 points de la fonction cosinus et on le trace.

On importe aussi au début du programme toutes les bibliothèques que nous utiliserons par la suite.

```
 8 import numpy as np
 9 import math as mp
10 import matplotlib.pyplot as plt
11 from random import uniform
12 from numpy.polynomial import Polynomial
13
14 ######1######
15
16 def graph_cos(n):
17     X=[]
18     Y=[]
19     for i in range(0,n):
20        a=uniform(0,mp.pi)
21        b=mp.cos(a)
22
23        X.append(a)
24        Y.append(b)
25     plt.scatter(X,Y,)
26     plt.title("fonction cosinus échantillonnée")
27     return(X,Y)
```

La fonction graph_cos(100) renvoie alors :



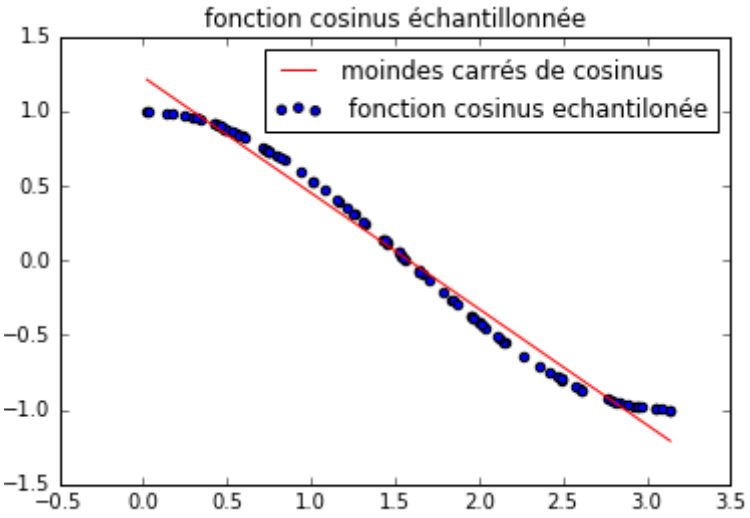
2)

La fonction permettant d'estimer au sens des moindres carrés la fonction cosinus implémentée précédent est :

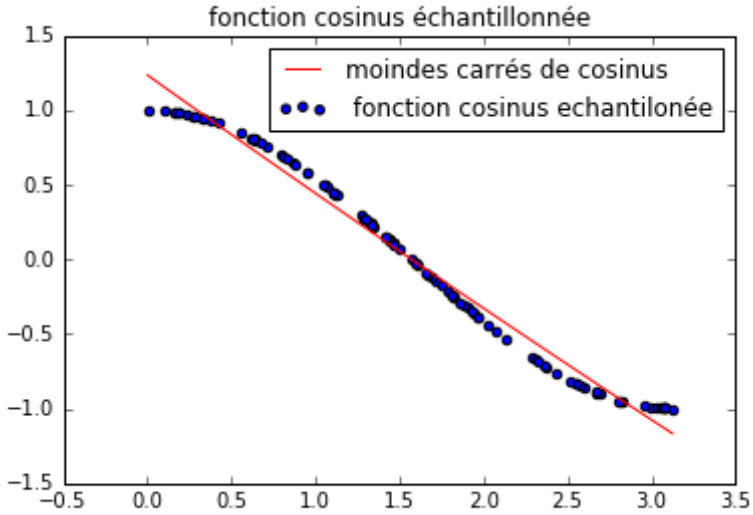
```
#####2#####
def moindres_carres(n,d):
    k=d+1
    X = sorted(graph_cos(n)[0])
    Y = np.cos(X)
    A= np.zeros((k,k))

#Calcul de la matrice A si on prend le probleme sous la forme A.I=B
    for l in range(1,k+1):
        for j in range(1,k+1):
            Somme=0
            for i in range(1,n+1):
                Somme+=X[i-1]**(j+l-2)
            A[l-1,j-1]=Somme

#Calcul de B
    B= np.zeros((k,1))
    for j in range(1,k+1):
        Somme=0
        for i in range(1,n+1):
            Somme+=Y[i-1]*X[i-1]**(j-1)
        B[j-1,0]=Somme

#Calcul de I
    I=np.linalg.solve(A,B)

#Generation du polynome#
    L=I.tolist()
    C=[]
    for i in range(len(L)):
        C.append(L[i][0])
    p=Polynomial(C)

###Graphique#####
    W=[]
    for i in range (n):
        W.append(p(X[i]))
    plt.plot(X,W,'r',label='moindes carrés de cosinus')
    plt.legend()
```

On étudie alors la réponse pour le degré 1,2 et 3.
On entre donc dans la console :

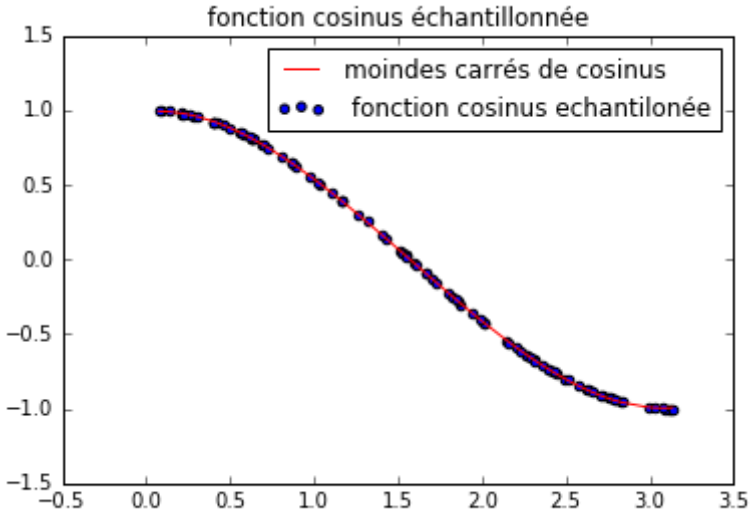
moindres_carres(100,1)



moindres_carres(100,2)



moindres_carres(100,3)



3)

Pour le degré 0, cela renvoie logiquement la moyenne donc une droite d'équation : y=0. Ensuite, on remarque que pour un degré supérieur ou égal à 3, la fonction cosinus est très bien approximée et augmenter le degré du polynôme estimateur n'a pas d'influence visuelle.

Exercice 2 :

Nous cherchons dans cet exemple à approcher une fonction quelconque f à partir d'un échantillon bruité de couples $(x_i, y_i), i = 1, \cdots, n$ avec :

$$y_i = f(x_i) + \varepsilon_i$$

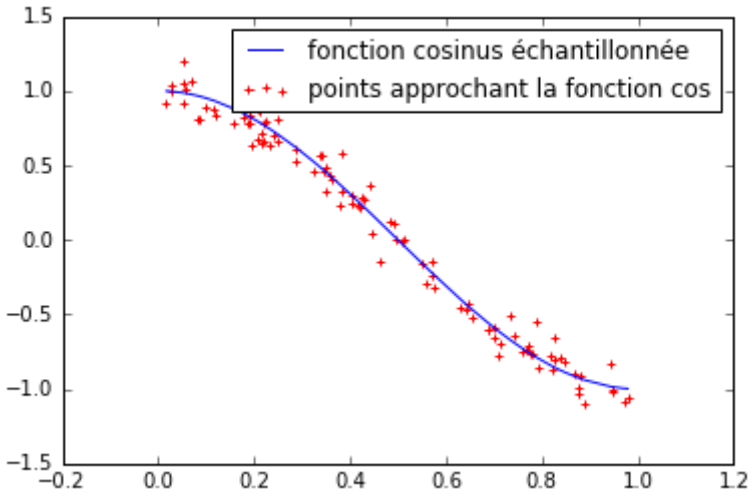
Où ε_i est un bruit gaussien de variance σ^2. Afin d'illustrer la méthode, nous allons prendre la fonction cosinus comme fonction cible en prétendant que nous ne la connaissons pas.

1) 2) 3) 4) 5)

Les questions 1, 2, 3, 4 et 5 consistent à échantillonner sur 100 points la fonction cosinus entre 0 et 1 puis à générer une observation bruitée avec une variance de 0.01 donc un écart-type de 0.1. A la suite de cela, on affiche aussi les points à partir desquels nous allons approcher la fonction cosinus.

```
###Exercice 2####


####1#####
n=100
x=np.asarray(sorted(np.random.rand(n)))

###2###
f=np.cos(np.pi*x)

###3###
sigma=0.1
e=np.random.randn(n)
y=f+sigma*e

####4###
plt.plot(x,f,label='fonction cosinus échantillonnée')

###5####
plt.scatter(x,y,s=20,c='r',marker='+',label='points approchant la fonction cos')
plt.legend()
```

On obtient alors :



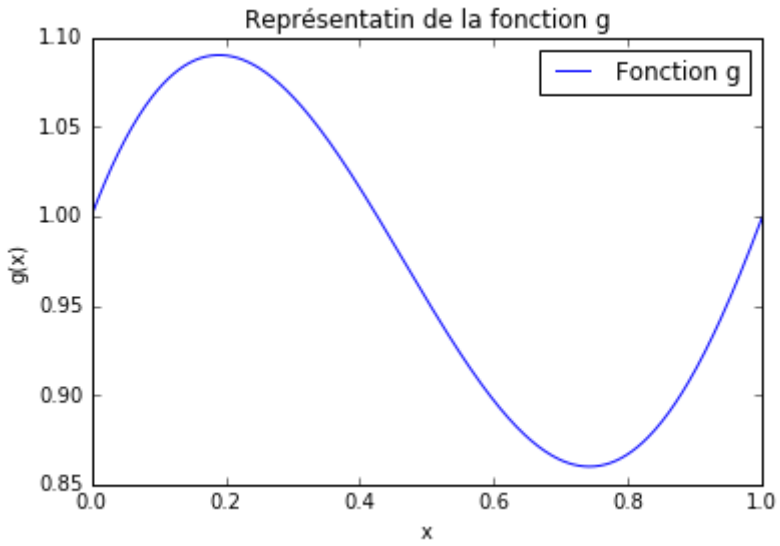
On remarque alors bien l'effet du bruit qui fait que l'on reste autour de la courbe dans un certain intervalle.

6)

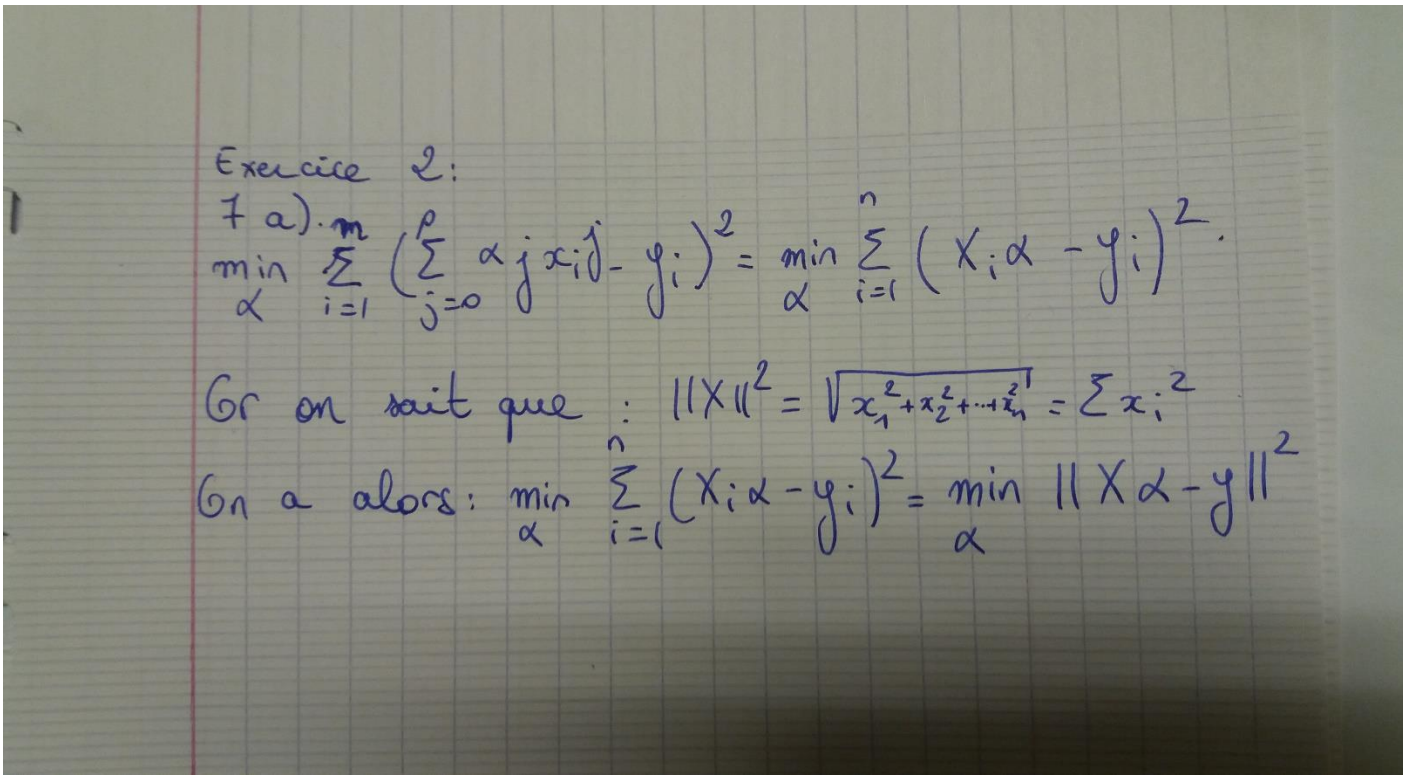
Ensuite, il nous est demandé de tracer la fonction : $g(x)=1+x-3x^2+x^3+x^4+x^5-x^6$ pour $x\in[0,1]$ sur une deuxième figure

```
92 ###6###
93 def g (x) :
94     return (1+x-3*x**2+x**3+x**4+x**5-x**6)
95
96 # Affichage
97
98 B = np.linspace(0,1,500)
99 C = g(B)
100 plt.figure()
101 plt.plot(B,C, label = 'Fonction g')
102 plt.xlabel('x')
103 plt.ylabel('g(x)')
104 plt.title('Représentatin de la fonction g')
105 plt.legend()
106 plt.show()
```

Par conséquent on obtient simplement :



7) a)



X a donc pour dimension (p,1)

On calcule alors X, pour cela on adapte la fonction moindres_carres du premier exercice :

```
113 def moindres_carres2(n,p):
114 
115     X = x
116     Y = y
117     A= np.zeros((p,p))
118 
119 #Calcul de la matrice A si on prend le probleme sous la forme A.I=B
120     for l in range(1,p+1):
121         for j in range(1,p+1):
122             Somme=0
123             for i in range(1,n+1):
124                 Somme+=X[i-1]**(j+l-2)
125 
126             A[l-1,j-1]=Somme
127 
128 #Calcul de B
129 
130     B= np.zeros((p,1))
131     for j in range(1,p+1):
132         Somme=0
133         for i in range(1,n+1):
134             Somme+=Y[i-1]*X[i-1]**(j-1)
135         B[j-1,0]=Somme
136 
137 #Calcul de I
138 
139     I=np.linalg.solve(A,B)
140 
141     return(I) #on retourne I qui est l'equivalent du X dans l'énoncé
142 
```

Par exemple pour p=6 et n=100 on obtient :

X= array([[1.02095539],
[-0.47046073],
[-2.36733986],
[-3.80361408],
[5.24101202],
[-0.54266945]])

b)

On créer une fonction permettant de calculer la fonction cout d'optimisation avec α allant de 1 à p.

```
###b####

def cout_optimisation(n,p):

    X=moindres_carres2(n,p)
    Xt=np.transpose(X)
    X_alpha=np.zeros((p,1))

    for i in range(0,p):

        X_alpha[i][0]=X[i][0]+(i+1)


    return(Xt.dot(X_alpha-y))

```

c) d) e)

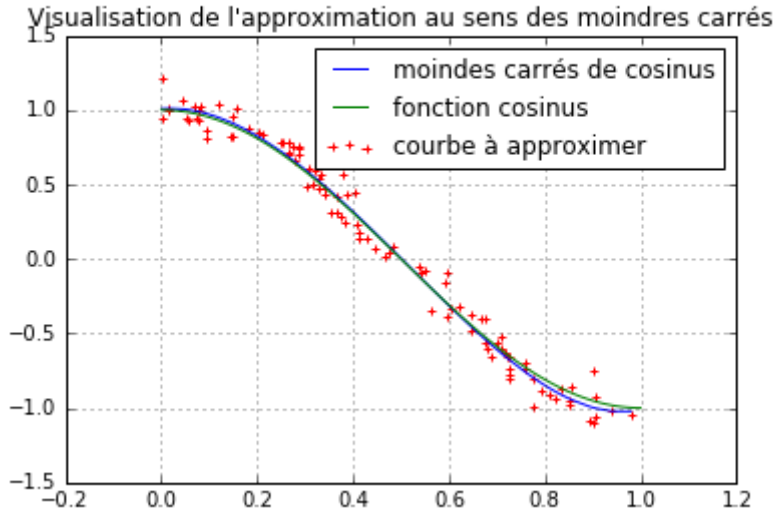
On regroupe les questions c), d) et e) car elles vont être regroupées dans une même fonction adaptée de l'exercice 1 nommée moindres_carres3.

```
###c/d/e###
def moindres_carres3(n,p):

    X = x
    Y = y
    A= np.zeros((p,p))

#Calcul de la matrice A si on prend le probleme sous la forme A.I=B
    for l in range(1,p+1):
        for j in range(1,p+1):
            Somme=0
            for i in range(1,n+1):
                Somme+=X[i-1]**(j+l-2)
            A[l-1,j-1]=Somme

#Calcul de B
    B= np.zeros((p,1))
    for j in range(1,p+1):
        Somme=0
        for i in range(1,n+1):
            Somme+=Y[i-1]*X[i-1]**(j-1)
        B[j-1,0]=Somme

#Calcul de I
    I=np.linalg.solve(A,B)

#Generation du polynome#
    L=I.tolist()
    C=[]
    for i in range(len(L)):
        C.append(L[i][0])
    p=Polynomial(C)
```

```
###Graphique#####
    W=[]
    for i in range (n):
        W.append(p(X[i]))

    plt.figure()
    plt.plot(X,W,label='moindes carrés de cosinus') # On trace l'approximation au sens des moindres carrés de la fonction cosinus
    plt.scatter(x,y,s=20,marker='+',label='courbe à approximer') # On ajoute la courbe que l'on souhaite approximer

    w=np.linspace(0,1,100)


    plt.plot(w,np.cos(np.pi*w),label='fonction cosinus')# On ajoute la fonction cosinus réelle pour vérifier la concordance avec
    #notre approximation
    plt.title("Visualisation de l'approximation au sens des moindres carrés")
    plt.grid()
    plt.legend()
    plt.show()
```

On obtient donc dans la console en lançant la fonction pour n=100 et p=6 :

```
In [160]: moindres_carres3(100,6)
```



Le résultat est satisfaisant car malgré le bruit généré sur la fonction cosinus, l'approximation au sens des moindres carrés se trouve presque confondue avec la courbe de la fonction cosinus.