TP1 correction : Méthode des moindres carrés

Exercice 1. Régression linéaire en dimension 1.

1.
```
import numpy as np

#choix d'une valeur pour Rtheo
Rtheo = 8.2564 #ohm
a, b, n = 0, 1.5, 10
I = a + (b-a)*np.random.rand(n) # n valeurs aleatoires entre a et b
I.sort()
U = Rtheo*I # valeurs theoriques de U associees a I
# ajout d'un bruit
sigma = 0.1
Ubruit = U + sigma * np.random.randn(n)
```

2. La fonction à minimiser selon la méthode des moindres carrés est : $r \mapsto \|rI - U\|_2^2$, soit

$$r \mapsto \|I\|_2^2 r^2 - 2\langle I, U\rangle r + \|U\|_2^2$$

C'est une forme quadratique (ici en dimension 1).

3. On en déduit l'expression de la résistance R_{opt} optimale selon la méthode des moindres carrés :

$$R_{opt} = \frac{\langle I, U\rangle}{\|I\|_2^2} = \left(\sum_{i=1}^{n} I_i^2\right)^{-1} \left(\sum_{i=1}^{n} I_i U_i\right)$$

4.
```
#Ropt :
Ropt= np.dot(Ubruit.transpose(),I)/(np.linalg.norm(I)**2)

# figure
plt.plot(I, Ubruit, 'ob', I, Ropt*I,'+-r', I, Rtheo*I, '--g')
plt.legend(['U','$R_{opt}I$','$R_{theo}I$'])#
plt.xlabel('intensite (amperes)') ; plt.ylabel('tension (volt)')
plt.grid()

plt.show()

print('R theorique : ',Rtheo)
print('R optimal   : ',Ropt)
```

5. Avec un niveau de bruit plus important on dégrade l'estimation de R.

```
#erreur relative en fonction du bruit
error = []
listbruit = [0.1, 0.5, 0.7, 1., 1.3, 1.5]
for sig in listbruit:
    Ubruit = U + sig * np.random.randn(n)
   #calcul de Ropt pour le bruit choisi
    Ropt= np.dot(Ubruit.transpose(),I)/(np.linalg.norm(I)**2)
```

```
    error.append(np.abs(Ropt - Rtheo)/Rtheo) #sauve erreur dans une liste

plt.figure()
plt.plot(listbruit, error, 'x')
plt.xlabel('niveau bruit')
plt.ylabel('erreur relative')
plt.show()
```

6. Avec peu de points de données, on dégrade également l'estimation de R

Exercice 2. Régression polynomiale.

1.
```
c1,c2=-10,10
p=5
coef=c1+(c2-c1)*np.random.rand(p+1)   # p+1 coefficients entre c1 et c2
                                      # coefficients reels du polynome P
a,b=-2,2
n=10
x=a+(b-a)*np.random.rand(n)           # n  points  entre a et b
x.sort()
Px=np.zeros(n)
for k in range(len(coef)):
    Px=Px+coef[k]*x**k                 # P en ces points

sigma = 0.2
y = Px + sigma * np.random.randn(n) # ajout d'un bruit de niveau sigma
```

2. L'ajustement polynomial des données par une méthode des moindres carrés consiste à résoudre le problème d'optimisation suivant

$$\min_{a\in\mathbb{R}^{p+1}} \sum_{i=1}^{n} \left(\sum_{k=0}^{p} a_k x_i^k - y_i \right)^2$$

Notant que $\sum_{k=0}^{p} a_k x_i^k - y_i = (Xa-y)_i$ où $X \in \mathcal{M}_{n,p+1}(\mathbb{R})$ est définie par $X_{i,k} = x_i^k$, on peut reformuler le problème ainsi :

$$\min_{a\in\mathbb{R}^{p+1}} \sum_{i=1}^{n} \left(\sum_{k=0}^{p} a_k x_i^k - y_i \right)^2 = \min_{a\in\mathbb{R}^{p+1}} \|Xa-y\|_2^2.$$

Les solutions du problème des moindres carrés, si elles existent, sont exactement les solutions de "l'équation normale" : $X^T X a = X^T y$.

3. Les solutions du problème sont les vecteurs $a \in \mathbb{R}^{p+1}$ solutions du système linéaire $X^T X a = X^T y$. Ce système admet toujours une solution (montrer que $Im(X^T) \subset Im(X^T X)$). Si on suppose que $n \geq p+1$ (ce qui est le cas pratique le plus courant), cette solution est unique car la matrice X est alors injective (puisque les x_i sont tous distincts) et donc la matrice $X^T X$ est inversible. D'où l'expression de la solution

$$a = (X^T X)^{-1} X^T y$$

4.
```
# On forme la matrice X
X=np.zeros((n,p+1))
for k in range(len(coef)):
    X[:,k]=x**k

# On resout les equations normales
Legende=[]                  # Liste contenant les legendes des graphiques
x_precis = np.linspace(-2, 2, 100)# pour tracer les poly sur plus de points
for pp in range(1,p+2):
    Xp=X[:,0:pp]                  # premieres colonnes de la marice X

    # On forme les matrices X^T X et le vecteur X^T y
```

```
    XTX=np.dot(Xp.transpose(),Xp)
    XTy=np.dot(Xp.transpose(),y)

    cf=np.linalg.solve(XTX,XTy) # solution du systeme lineaire

    sol=np.zeros(len(x_precis))
    for k in range(len(cf)):
        sol+=cf[k]*x_precis**k
    plt.plot(x_precis,sol)
    Legende.append('deg = '+str(pp))

plt.plot(x,y,'o') # nuage de points
Legende.append('donnees')

#P theorique
Ptheo=np.zeros(len(x_precis))
for k in range(len(coef)):
    Ptheo=Ptheo+coef[k]*x_precis**k

plt.plot(x_precis, Ptheo, '--', color = 'r')
Legende.append('P theorique')

plt.legend(Legende)
plt.show()
```

5.

```
# on recupere les donnees
donnee_x = np.array([-2.6605, -2.5024, -2.4075, -2.0656, -1.2418,
      -1.1105, -1.0194, -0.8459, -0.8403, -0.4906,
       0.7695,  0.8933,  1.0023,  1.5085, 2.3193])

donnee_y =np.array([ 97.1414, 80.8250, 72.0406, 46.0602, 12.0776,
      9.4146, 7.8925, 5.6353, 5.5751, 3.0525, -0.3093,
      -1.3142, -2.4258, -11.0032, -39.9284])

n = len(donnee_x)
err = 1.0
p = 1
savecoef = []
while (err > 1e-6 and p < 10) : # on choisi d'approcher a 1e-6,
                          # et on fait varier le degre du polynome
    # On forme la matrice X
    X=np.zeros((n,p+1))
    for k in range(p+1):
        X[:,k]=donnee_x**k

    # On forme les matrices X^T X et le vecteur X^T y
    XTX=np.dot(X.transpose(),X)
    XTy=np.dot(X.transpose(),donnee_y)
    cf=np.linalg.solve(XTX,XTy) # solution du systeme lineaire
    savecoef.append(cf) # on sauve les coeff pour chaque degre de polynome

    #calcul de l'erreur L2
    P = 0
    for k in range(len(cf)) :
        P = P+cf[k] * donnee_x**k
    err = np.sqrt(np.sum( (P - donnee_y)**2 ))

    print('p = ', p,', error = ', err)
    p = p+1 # augmente le degre du polynome

print("deg poly =", p-1) # degre du polynome choisi
print("coef =", cf) # ses coefficients
```

```
print("erreur =", err) # erreur L2 entre les donnees
                       # et le polynome qui les approche

# on trace la comparaison
plt.figure()
plt.plot(donnee_x, donnee_y, '.')
plt.plot(donnee_x, P)
plt.show()
```

Exercice 3.

```
# comparaison avec np.polyfit fonction de numpy :
# Least squares polynomial fit i.e la meme chose que votre algo
# Fit a polynomial p(x) = p[0] * x**deg + ... + p[deg]
# of degree deg to points (x, y).
# Returns a vector of coefficients p that minimises
# the squared error in the order deg, deg-1, ..., 0.

polfit = []
for pp in range(1,p) :
    polfit.append(np.polyfit(donnee_x, donnee_y, pp))

for k, aa in enumerate(polfit):
    print("degre P = ", len(aa) -1)
    print("coeff polyfit: ", aa)
    coef = savecoef[k]
    print("coeff moindre carres : ", coef[::-1])# polyfit range les coef
                                        # dans "l'autre sens"
```

1. Un polynôme de degré 4 approche les données à 10^{-8} près.
2. Le résultat de `polyfit` est le même que celui des moindres carrés, en effet c'est cette méthode qui est utilisée par la fonction `polyfit`.