

Corrigé

Corrigé feuille exercices révision

```
[1]: import numpy as np
import matplotlib.pyplot as plt
```

Exercice 1 : Représentation binaire

Q1.

```
[2]: bin(213)
```

```
[2]: '0b11010101'
```

Conversion du binaire en décimal

```
[3]: int('0b11101',2)
```

```
[3]: 29
```

Conversion de l'hexadécimal vers le décimal

```
[4]: int('0xCA',16)
```

```
[4]: 202
```

Q2.

0,375 est représenté par 0.011 car $0,375 = 0 + 0 \times 1/2 + 1 \times 1/4 + 1 \times 1/8$

Q3.

```
[5]: def binaire(n):
    L = []
    while n>0:
        L.append(n%2)
        n = n//2
    return L
```

```
[6]: binaire(213),bin(213)
```

```
[6]: ([1, 0, 1, 0, 1, 0, 1, 1], '0b11010101')
```

Q4.

```
[7]: def octet(n):
    from math import log
    return int(log(n)/log(256))+1
```

```
[8]: n = 21354654
p = octet(n)
p, n<256**p, 256**(p-1)<=n
```

[8]: (4, True, True)

```
[9]: def octet(n):  
    p = 1  
    while n>256:  
        n = n//256  
        p += 1  
    return p
```

```
[10]: n = 21354654  
p = octet(n)  
p, n<256**p, 256**(p-1)<=n
```

[10]: (4, True, True)

Exercice 2 - Échantillonnage

Q1.

Une résolution de 16 bits donne $2^{16} = 65536$ valeurs ce qui donne une précision de $\frac{5}{2^{16}} = \frac{5}{65536}$.

```
[11]: 5/2**16
```

[11]: 7.62939453125e-05

Q2.

On a 1000 mesures par seconde \times 16 bits \times 5 secondes = 80 000 bits soit 10 000 octets.

Q3.

L'échantillonnage correspond à 16 000 bits par seconde ce qui est compatible avec 127 000 bits par seconde.

Q4.

```
[12]: 127000/16
```

[12]: 7937.5

La liaison permet 7937 échantillons par seconde soit une fréquence d'échantillonnage de 7937 Hz.

Exercice 3 - Calcul numérique

Q1.

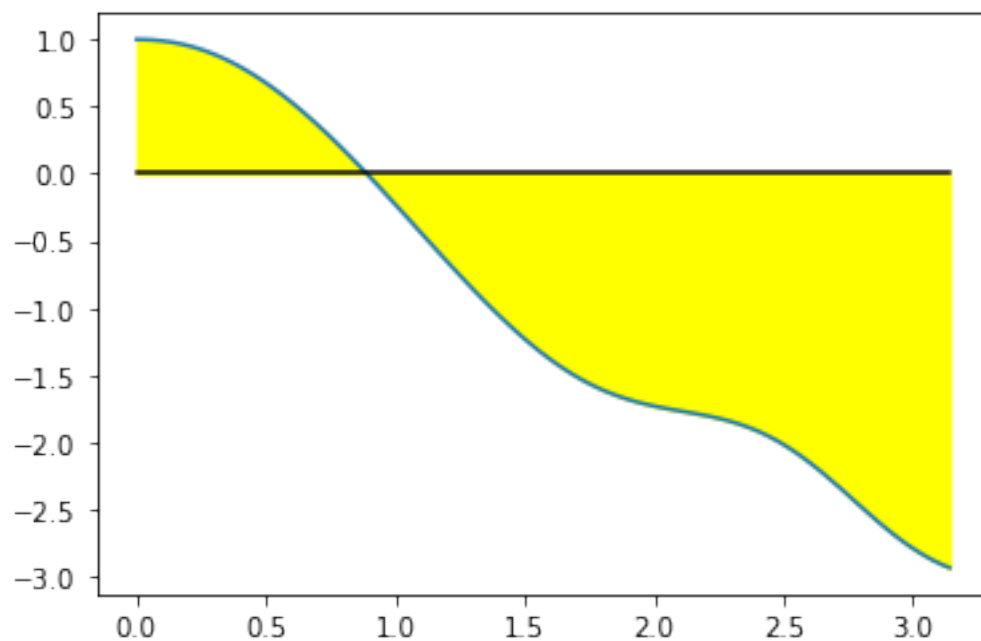
```
[13]: def trapz(f,a,b,n):  
    pas = (b-a)/n  
    I = 0  
    for i in range(n):  
        I += pas *(f(a+i*pas) + f(a+(i+1)*pas))/2  
    return I
```

```
[14]: def f(t):  
       return (np.cos(t**2)-t**3)/(1+t**2)
```

```
[15]: trapz(f,0,np.pi,20)
```

```
[15]: -3.0949580551501663
```

```
[16]: Lt = np.linspace(0,np.pi,100)  
      Ly = f(Lt)  
      plt.plot(Lt,Ly)  
      plt.plot(Lt,0*Lt,"k")  
      plt.fill_between(Lt, 0*Lt, Ly, color = 'yellow')  
      pass
```



Q2.

```
[17]: def dichot(f,a,b):  
       while b-a>1e-5:  
           m = (a+b)/2  
           if f(m)*f(a)>0:  
               a = m  
           else:  
               b = m  
       return (a,b)
```

```
[18]: dichot(f,0,3)
```

```
[18]: (0.8892803192138672, 0.8892860412597656)
```

Q3.

```
[19]: def newton(f,fp,x0):
      x1 = x0 + 1e-3
      while abs(x1-x0)>1e-7:
          x1 = x0
          x0 = x0 - f(x0)/fp(x0)
      return x1
```

```
[20]: def fp(t):
      return -2*t*np.sin(t**2)/(1+t**2) - 3*t**2/(1+t**2) - 2*t*f(t)/(1+t**2)
```

```
[21]: newton(f,fp,1.0)
```

```
[21]: 0.8892810752556453
```

Exercice 4 : Recherche dichotomique

Invariant de boucle : à chaque itération de la boucle While, on a « $L[g] < val \leq L[d]$ » pour trouver la première occurrence de val.

```
[22]: def rechDicho(L,val):
      d = len(L)-1
      g = -1
      while d-g>1:
          m = (g+d)//2 # on veut obtenir un entier
          if L[m] < val :
              g = m # on garantit ainsi L[g]<val<= L[d]
          else:
              d = m
      if L[d] == val:
          return True,d
      else :
          return False,-1
```

```
[23]: rechDicho([0,0,1,1,1,1,2,3,3,4,5,6,8,9],1)
```

```
[23]: (True, 2)
```

```
[ ]:
```

Version récursive

```
[24]: def rechDichoRec(val,L,g,d):
      if d-g<=1:
          if L[d] == val:
              return True, d
          else:
              return False, -1
      m = (g+d)//2 # on veut obtenir un entier
      if L[m] < val :
          g = m # on garantit ainsi L[g]<val<= L[d]
      else:
```

```

        d = m
    return rechDichoRec(val,L,g,d)

```

```

[25]: L = [k for k in range(200)]
      val = 35
      rechDichoRec(val,L,0,len(L)),rechDichoRec(val+.5,L,0,len(L))

```

```

[25]: ((True, 35), (False, -1))

```

Exercice 5 - SQL

Q1. La moyenne des populations des communes de France :

```
SELECT AVG(pop) FROM communes
```

Q2. Les communes de Loire-Atlantique :

```
SELECT * FROM communes AS C JOIN departements AS D ON C.dep_id = D.id WHERE D.nom = "Loire-Atlantique"
```

Les communes des Pays de la Loire :

```
SELECT * FROM communes AS C JOIN departements AS D ON C.dep_id = D.id JOIN regions AS R ON D.reg_id = R.id WHERE R.nom = "Pays de la Loire"
```

Q3. Pour la (ou les) commune (s) la (les) moins peuplée(s), on utilise une sous-requête :

```
SELECT nom FROM communes WHERE pop = (SELECT MIN(pop) FROM communes)
```

Q4. Pour les 100 communes les plus peuplées, on ordonne les communes par population décroissante et on ne conserve que les 100 premières.

```
SELECT nom,pop FROM communes ORDER BY pop DESC LIMIT 100
```

Q5. Pour la moyenne de population des communes pour chaque département, on groupe les tuples par département pour calculer les moyennes :

```
SELECT D.nom, AVG(C.pop) AS Moyenne FROM communes AS C JOIN departements AS D ON C.dep_id = D.id GROUP BY D.id
```

Q6. Pour les départements ayant une population totale supérieure à 1 million d'habitants, on ne conserve que les départements (groupes) ayant une population supérieure à 1 million :

```
SELECT D.nom, SUM(C.pop) AS Total FROM communes AS C JOIN departements AS D ON C.dep_id = D.id GROUP BY D.id HAVING Total>1E6
```

Q7. Pour les départements ayant une population totale supérieure à la moyenne, on utilise deux sous-requêtes : une pour calculer les populations totales et une pour calculer la moyenne.

```
SELECT D.nom, SUM(C.pop) AS Total FROM communes AS C JOIN departements AS D
ON C.dep_id = D.id GROUP BY D.id HAVING Total> (SELECT AVG(Total) FROM (
SELECT D.nom, SUM(C.pop) AS Total FROM communes AS C JOIN departements AS
D ON C.dep_id = D.id GROUP BY D.id))
```

Exercice 6 : Algorithme de Gauss

```
[26]: def recherchePivot(M,j):
    maxi = j
    n = len(M)
    for i in range(j,n):
        if abs(M[i,j])>abs(M[maxi,j]):
            maxi = i
    return maxi

def transvection(M,i,piv,alpha):
    p = len(M[0])
    for j in range(p):
        M[i,j] = M[i,j] + alpha*M[piv,j]

def echange(M,i,k):
    p = len(M[0])
    for j in range(p):
        M[i,j], M[k,j] = M[k,j], M[i,j]

def dilatation(M,i,alpha):
    p = len(M[0])
    for j in range(p):
        M[i,j] = alpha*M[i,j]

def echelonne(M):
    n = len(M)
    for j in range(n):
        piv = recherchePivot(M,j)  # plus grand pivot dans la colonne j
        echange(M,j,piv)           # échange pivot et ligne courante
        for i in range(n):
            if i != j:             # transvections sur toutes les lignes
                transvection (M,i,j,-M[i,j]/M[j,j])
            dilatation(M,j,1/M[j,j])
    return M[:,-1]
```

```
[27]: M = [[1.0,2,1,3,4 ],[2,1,1,-1,1],[1,2,1,1,0],[3,1,2,3,1]]
M = np.array(M)
```

```
[28]: echelonne(M)
```

```
[28]: array([ 8.,  3., -16.,  2.]
```

```
[29]: M
```

```
[29]: array([[ 1.,  0.,  0.,  0.,  8.],
            [ 0.,  1.,  0.,  0.,  3.],
            [-0., -0.,  1.,  0., -16.],
            [ 0.,  0.,  0.,  1.,  2.]])
```

```
[30]: def transvection(M,i,piv,alpha):
        M[i] = M[i] + alpha*M[piv]

    def echange(M,i,k):
        # M[i], M[k] = M[k], M[i] # NE MARCHE PAS
        M[[i,k]] = M[[k,i]] # MARCHE !!! Notation spécifique de numpy appelée
        ↪ «advanced slicing».

    def dilatation(M,i,alpha):
        M[i] = alpha*M[i]
```

```
[31]: M = [[1.0,2,1,3,4 ],[2,1,1,-1,1],[1,2,1,1,0],[3,1,2,3,1]]
M = np.array(M)
M
```

```
[31]: array([[ 1.,  2.,  1.,  3.,  4.],
           [ 2.,  1.,  1., -1.,  1.],
           [ 1.,  2.,  1.,  1.,  0.],
           [ 3.,  1.,  2.,  3.,  1.]])
```

```
[32]: transvection(M,1,0,-2)
M
```

```
[32]: array([[ 1.,  2.,  1.,  3.,  4.],
           [ 0., -3., -1., -7., -7.],
           [ 1.,  2.,  1.,  1.,  0.],
           [ 3.,  1.,  2.,  3.,  1.]])
```

```
[33]: dilatation(M,1,-1/3)
M
```

```
[33]: array([[ 1.          ,  2.          ,  1.          ,  3.          ,  4.          ],
           [-0.          ,  1.          ,  0.33333333,  2.33333333,  2.33333333],
           [ 1.          ,  2.          ,  1.          ,  1.          ,  0.          ],
           [ 3.          ,  1.          ,  2.          ,  3.          ,  1.          ]])
```

```
[34]: echange(M,0,1)
M
```

```
[34]: array([[ -0.          ,  1.          ,  0.33333333,  2.33333333,  2.33333333],
           [ 1.          ,  2.          ,  1.          ,  3.          ,  4.          ],
           [ 1.          ,  2.          ,  1.          ,  1.          ,  0.          ],
           [ 3.          ,  1.          ,  2.          ,  3.          ,  1.          ]])
```

```
[35]: echelonne(M)
```

```
[35]: array([ 8.,  3., -16.,  2.] )
```

```
[ ]:
```

Exercice 7 : Tri Fusion

```
[36]: def fusion(L1,L2):
        """ on fusionne L1 et L2 en respectant l'ordre """
        nL = []
        i1, i2 = 0,0
        for k in range(len(L1)+len(L2)):
            if i2>= len(L2) or ( i1<len(L1) and L1[i1] < L2[i2] ):
                nL.append(L1[i1])
                i1 += 1
            else:
                nL.append(L2[i2])
                i2 += 1
        return nL

[37]: def tri_fusion(L):

        if len(L) <=1: # cas de base si la liste contient un seul élément
            return L
        # sinon on trie récursivement les deux moitiés et on les fusionne
        L1 = tri_fusion(L[0:len(L)//2])
        L2 = tri_fusion(L[len(L)//2:])
        return fusion(L1,L2)

[38]: L=[1,5,4,65,4,2,6,4,2,3,4,5,65,654,47,532,1,351,35,4]
        tri_fusion(L), len(L) == len(tri_fusion(L))

[38]: ([1, 1, 2, 2, 3, 4, 4, 4, 4, 4, 5, 5, 6, 35, 47, 65, 65, 351, 532, 654], True)

[ ]:
```

Exercice 8 : Manipulation d'échantillons

```
[39]: Lt = np.linspace(0,10,180)
        Ly = f(Lt)

Q1.

[40]: def valeurMoyenne(Lt,Ly):
        n = len(Lt) # nombre de points
        val = 0
        for i in range(n-1): # pour chaque intervalle
            val += (Lt[i+1] - Lt[i]) * (Ly[i] + Ly[i+1])/2
        return val/(Lt[-1]-Lt[0])

[41]: valeurMoyenne(Lt,Ly)

[41]: -4.704012893040506
```

Q2.


```
[42]: def ecrire(nom_fichier,Lt, Ly):
        f = open(nom_fichier,"wt")
        for i in range(len(Lt)):
            f.write(str(Lt[i])+" ; "+str(Ly[i]) + "\n")
        f.close()
```

```
[43]: ecrire("listes.txt",Lt,Ly )
```

Q3.

```
[44]: def lire(nom_fichier):
        f = open(nom_fichier,"rt")
        Lt, Ly = [], []
        for ligne in f:
            L=ligne.split(";")
            Lt.append(float(L[0]))
            Ly.append(float(L[1]))
        return Lt, Ly
```

```
[45]: Lt1,Ly1 = lire("listes.txt")
```

Q4. Moyenne glissante

```
[46]: def moyenne(p,Lt,Ly):
        n = len(Ly)
        Lu=[]
        Lz=[]
        somme = 0
        for i in range(p-1):
            somme += Ly[i]
        for i in range(p,n):
            somme = somme + Ly[i]
            Lz.append(somme)
            somme -= Ly[i-p]
            Lu.append(Lt[i])
        return Lu,Lz
```

```
[47]: Lu,Lz = moyenne(4,Lt,Ly)
```

```
[48]: len(Lu), len(Lt)
```

```
[48]: (176, 180)
```

Exercice 9 : Fonction récursive

```
[65]: def permut(n):
        if n == 1:                                # cas de base
            return [[1]]
        else:
            L = permut(n-1)                        # on calcule les permutations des premiers
            nL = []
```

```

    for p in L:                # pour chaque permutation des n-1 premiers
        for k in range(n-1): # on place n en position k
            nL.append( p[:k]+[n]+p[k:])
            nL.append(p + [n])  # et on place n en dernière position
    return nL

```

```
[66]: len(permut(4))
```

```
[66]: 24
```

```
[67]: L = [0,1,2]
      L[:0]
```

```
[67]: []
```

```
[ ]:
```

Exercice 10 : Manipulation de listes

Q1.

```
[68]: def zip(L1,L2):
      assert len(L1)==len(L2)
      L = []
      for k in range(len(L1)):
          L.append( [L1[k], L2[k]] )
      return L

```

```
[73]: L1 = [1,2,3,4,5,6]
      L2 = ["a","b","c","d","e","f"]
      zip(L1,L2)
```

```
[73]: [[1, 'a'], [2, 'b'], [3, 'c'], [4, 'd'], [5, 'e'], [6, 'f']]
```

Q2.

```
[74]: def seuil1(L,s):
      i = 0
      while L[i]<s:
          i += 1
      return i

```

```
[75]: seuil1(L1,3)
```

```
[75]: 2
```

Q3.

```
[76]: def seuil2(L,s):
      i = len(L)-1
      while L[i]>=s:
```

```
    i -= 1
    return i
```

Q4.

```
[77]: def seuilx(LP,s):
        L = []
        for k in range(len(LP)):
            if LP[k][0] >= s:
                L.append(LP[k])
        return L
```

```
[78]: LP = [ [1,2], [3,4], [1,3], [2,5]]
        seuilx(LP,2)
```

```
[78]: [[3, 4], [2, 5]]
```

Q5.

```
[79]: def passage(L,s):
        res = []
        for k in range(len(L)-1):
            if L[k] < s <= L[k+1]:
                res.append(k)
        return res
```

```
[80]: L3 = [1,2,3,4,3,2,1,2,3,4,5,6,5,4,3,2,1]
        passage(L3,3)
```

```
[80]: [1, 7]
```

Q6.

```
[81]: def monotonie(L):
        croissant = L[0] < L[1]
        res = []
        i = 0
        while i < len(L) - 1:
            if croissant and L[i] > L[i+1]: # ça devient décroissant
                croissant = False
                res.append(i)
            if not croissant and L[i] < L[i+1]: # ça devient croissant
                croissant = True
                res.append(i)
            i += 1
        return res
```

```
[82]: monotonie(L3)
```

```
[82]: [3, 6, 11]
```

```
[ ]:
```

[]: