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. Un octet est une unité de quantité de mémoire égale à 8 bits. Un octet stocke les valeurs entières de l'intervalle $[0, 256[$, deux octets stockent les valeurs de $[0, 256^2[$, trois octets stockent $[0, 256^3[$ et ainsi de suite.

[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. La formule pour la méthode des trapèzes est $\int_a^b f(t)\ \mathrm{d}t = \sum_{i=0}^{n} \frac{b-a}{n} \times \frac{f\left(a + i\frac{(b-a)}{n}\right) + f\left(a + (i+1)\frac{(b-a)}{n}\right)}{2}$.

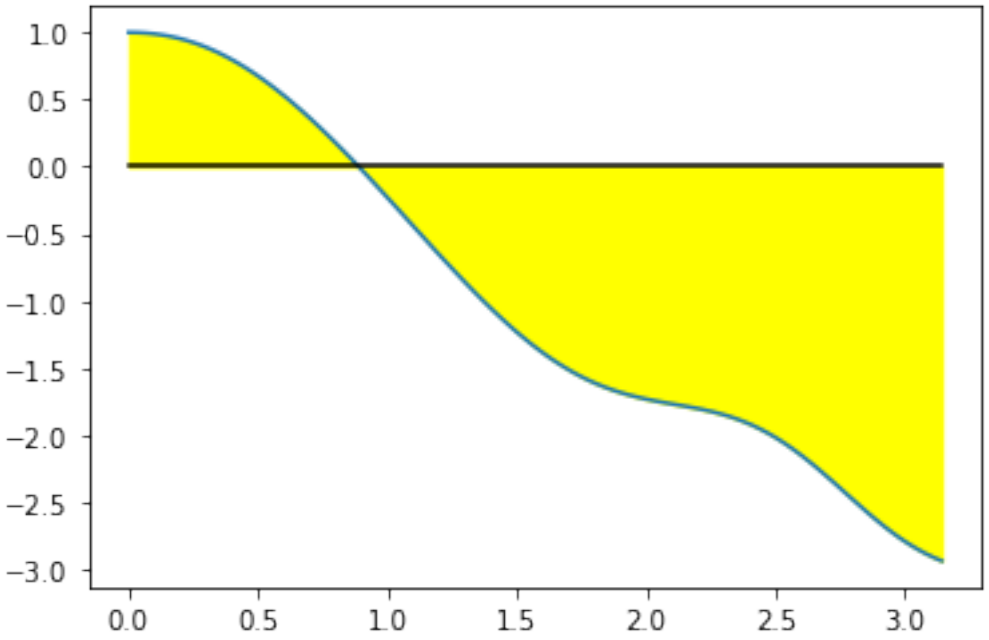
[13]:
```
def trapz(f,a,b,n):
    pas = (b-a)/n
    I = 0
    for i in range(n):
        I += pas *(f(a+i*pas) + f(a+(i+1)*pas))/2
    return I
```

[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)  # vecteur numpy des valeurs de t entre 0 et pi
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 dicho(f,a,b):
    while b-a>1e-5:  # 1e-5 est une précision arbitraire
        m = (a+b)/2
        if f(m)*f(a)>0: # on garantit l'invariant f(a).f(b)<=0
            a = m       # on conserve [m,b] car f(m).f(b)<=0
        else:
            b = m       # on conserve [a,m] car f(a).f(m)<=0
    return (a,b)        # (a,b) est un encadrement d'un zéro de f
```

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

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

Q3. Formule pour la méthode de Newton : $x_{n+1} = x_n - \dfrac{f(x_n)}{f'(x_n)}$

[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 bouche While, on a « `L[g] < val <= L[d]` » pour trouver la première occurence 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)
```

La complexité pour la recherche d'une valeur dans une liste **triée** de taille n est $c(n) = \mathcal{O}(\ln(n))$, c'est à dire **complexité quasi-constante**.

Version récursive (uniquement pour démontrer votre habileté car on ne gagne rien)

[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. On peut écrire `pop = (SELECT ...)` si la sous-requête ne renvoie qu'un résultat (c'est le cas ici pour `MIN`). On peut également utiliser `pop IN (SELECT ...)` si la sous-requête renvoie plusieurs valeurs.

```
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 (`DESC` = décroissant, `ASC` = croissant).

```
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)
    )
```