Lycée Faidherbe, 2020-21

T.P. d'informatique

MP, PC & PSI

Version du 21 février 2021

Table des matières

TP I

Révisions

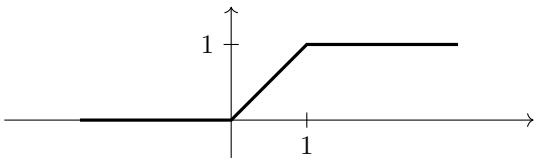
Résumé

Les exercices ici proposés sont classés en fonction de leur niveau de difficulté. Tous les étudiants doivent savoir faire les exercices de niveaux 1 sans problème et parvenir à une solution pour les exercices de niveau 2. Pour chaque algorithme proposé, on précisera sa complexité en fonction des paramètres fournis.

1 Niveau 1

Exercice 1

Écrire une fonction `seuil(x)` telle que $\text{seuil}(x) = \begin{cases} 0 & \text{pour } x \leqslant 0 \\ x & \text{pour } 0 \leqslant x \leqslant 1 \\ 1 & \text{pour } 1 \leqslant x \end{cases}$



Exercice 2

Écrire une fonction `facto(n)` qui calcule $n!$.
En déduire une fonction `binomial(n, p)` qui calcule $\binom{n}{p}$.

Exercice 3

Écrire une fonction `u(n)` qui calcule le terme u_n de la suite définie par $u_0 = 1$ et $u_{p+1} = \dfrac{u_p + 2}{u_p + 1}$ pour $p \in \mathbb{N}$.

Exercice 4

Proposer une fonction `testPres(L, x)` qui teste la présence de l'élément x dans la liste L.
Exemple : `testPres([1,2,27,3],-7) -> False`.

Exercice 5
Écrire une fonction `maxPerso(L)` qui retourne le maximum de la liste L sans utiliser la fonction max de python.
Écrire une fonction `IndiceMax(L)` qui retourne un indice du maximum de la liste L.

On rappelle que l'opérateur `%` *calcule le reste de la division (entière), il permet de tester la divisibilité : p divise n si et seulement si* `n % p == 0`.

Exercice 6
Proposer une fonction `testPrime` qui teste si un nombre fourni en paramètre est premier.
Exemple : `testPrime(47) -> True`

Exercice 7
Proposer une fonction `pgcd(n, p)` qui détermine le plus grand diviseur commun de n et p entiers positifs, c'est-à-dire le plus grand entier qui divise n et p. Il est compris entre 1 et `min(n, p)`.

Exercice 8
Écrire une fonction `est_croissante(liste)` qui renvoie `True` ou `False` selon que la liste est croissante ou non.

2 Niveau 2

Exercice 9
On considère une suite récurrente linéaire d'ordre trois donnée par : u_0, u_1, u_2 puis la relation

$$u_{n+3} = au_{n+2} + bu_{n+1} + cu_n$$

. Proposer une fonction qui prenant en paramètres $u_0, u_1, u_2, a, b, c, n$ retourne u_n.
Exemple : `suiteRec(1,2,3,1,1,1,3) -> 6.`

Exercice 10
Proposer une fonction permettant d'obtenir l'écriture en base 2 de tout nombre entier positif (sous forme de liste de 0 et de 1), puis proposer une fonction réciproque.
On mettra la liste sous forme des puissances de 2 croissantes c'est-à-dire que $L = [a_0, ..., a_{k-1}]$ représente $n = a_0 + 2a_1 + 4a_2 ... + 2^{k-2}a_{k-2} + 2^{k-1}a_{k-1}$.
Exemple : `base10To2(13) ->[1, 0, 1, 1] ; base2To10([1, 0, 0, 1, 1]) -> 25`

Exercice 11
Déterminer tous les triplets pythagoriciens $[a, b, c]$ tels que a et b sont des entiers de $[\![1, 100]\!]$ tels que $a \leqslant b$ et $a^2 + b^2 = c^2$.
Réponse : il y en a 63.

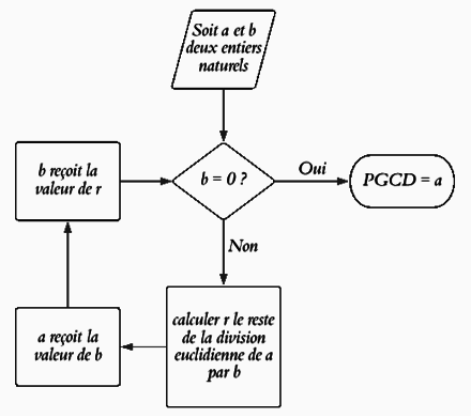
Exercice 12
En utilisant la propriété $\binom{n}{p} = \frac{n}{p}\binom{n-1}{p-1}$ pour $1 \leqslant p \leqslant n$ et la valeur $\binom{n}{0} = 1$, écrire une fonction `binom(n, p)` qui calcule $\binom{n}{p}$ en ne faisant que p multiplication et p divisions entières.

Exercice 13
Proposer une fonction renvoyant une nouvelle liste obtenue en retournant une liste L fournie en paramètre. Exemple : `inverse([2,4,7,13]) -> [13,7,4,2].`
Proposer ensuite une procédure réalisant cela "sur place".

Exercice 14
Écrire une fonction `DeuxMax(L)` qui retourne les deux plus grandes valeurs de la liste L supposées sans doublon.

Exercice 15

L'algorithme d'Euclide permet de calculer le PGCD rapidement. Son principe est symbolisé dans le schéma ci-joint.
Expliciter les valeurs de a et b après un passage lorsque, au départ de la boucle, on a $b > a$.
Avec cette méthode, écrire une fonction `Euclide(a,b)` qui retourne le PGCD des entiers naturels a et b.



Exercice 16
Proposer une fonction qui prenant en paramètre une liste contenant au moins deux entiers, détermine les deux éléments les plus proches.
Exemple : `plusProches([2,13,7,15,28]) -> [13,15]`.
Peut-on écrire une fonction plus efficace si on suppose que la liste est triée ?

Exercice 17
Écrire une fonction `est_modale(liste)` qui renvoie `True` ou `False` selon que la liste est modale ou non. Une liste `l` est modale s'il existe p compris entre 0 et n (n est sa longueur) telle que `l[ : p]` est strictement croissante et `l[p :p]` est strictement décroissante ; pour $p = 0$ (resp. $p = n$) cela signifie que la liste est strictement décroissante (resp. strictement croissante).

3 Niveau 3

Exercice 18
Proposer une fonction `testPresDicho(L,x)` qui teste la présence de l'élément x dans la liste triée dans l'ordre croissant L, en procédant à une dichotomie.

Exercice 19
Proposer une fonction qui calcule les coefficients binomiaux en utilisant le triangle dit de Pascal.
Pour calculer $\binom{n}{p}$ on calculera la liste des $\binom{m}{k}$, k variant de 0 à m, pas-à-pas pour les entiers m allant de 0 à n.

Exercice 20
Une liste L de booléens représente les résultats successifs d'une expérience de Bernoulli. Proposer une fonction qui détermine les longueurs des suites de succès consécutifs.
Exemple : `listeSucces([False,True,True,True,False,True,True]) -> [3,2]`

Exercice 21
Une application de l'ensemble $[\![1, n]\!]$ dans l'ensemble $[\![1, p]\!]$ est modélisée par la donnée de l'entier p ainsi que la liste ordonnée des images des éléments de $[\![1, n]\!]$.
Proposer deux fonctions déterminant si une telle application est injective, puis surjective.
Exemple : `testInj(5,[1,3]) -> True ; testSurj(3,[1,3,1]) -> False.`

Exercice 22
Proposer une fonction qui détermine si un nombre est égal à la somme des cubes de ses chiffres.
Trouver les entiers égaux à la somme des cubes de leurs chiffres (toutes les solutions ont 3 chiffres).

Exercice 23

Définir une fonction qui, prenant en paramètre un entier n, retourne le n-ième nombre entier positif supérieur à 10 dont l'écriture en base 10 est symétrique.

Exercice 24 — Mines-Ponts 2018
Une liste *Lalt* modélise la liste des altitudes d'une succession de points sur un chemin. Proposer une fonction qui retourne les hauteurs des dénivelés successifs des montées et des descentes.
Exemple : `listeDeniveles([0,2,7,13,12,7,15,22,21,25,37]) -> [13,-6,15,-1,16].`

Exercice 25
Proposer une fonction qui détermine la liste de nombres premiers inférieurs ou égaux à un entier n fourni en paramètre, en utilisant la méthode du crible d'Eratosthène.
Exemple : `crible(22) -> [2,3,5,7,11,13,17,19]`

Exercice 26
Écrire une fonction `goldbach(n)` qui retourne un couple de nombres premiers (a, b) tels que $a+b = n$ lorsque n pair.
Exemple : `goldbach(232) -> (3, 229)` et `goldbach(252) -> (11, 241)`
Tester la conjecture selon laquelle tout entier pair supérieur à trois est la somme de deux nombres premiers, jusque 10.000.000 (Conjecture de Golbach).

Exercice 27
On fixe n, un entier supérieur à 100. Tester sur les nombres p premiers inférieurs à 100 le petit théorème de Fermat : si p ne divise pas n, alors l'entier $n^{p-1} - 1$ est divisible par p.

4 Niveau 4

Exercice 28
Proposer une fonction qui détermine si un nombre n fournis en paramètre peut s'écrire comme la somme de cubes d'entiers strictement supérieurs à 1.
On pourra introduire une matrice de booléens B telles que $B[i, j]$ vaut vrai si et seulement si i est somme de j cubes d'entiers strictement supérieurs à 1.
Exemple : `testSommeCubes(24) -> True`

Exercice 29
On se donne un entier n et trois entiers $a < b < c$ non nuls. écrire une fonction qui donne le nombre de listes à valeurs dans $\{a, b, c\}$ dont la somme des termes vaut n. Par exemple, les 7 listes $[1, 1, 1, 1], [1, 1, 2], [1, 2, 1], [2, 1, 1], [2, 2], [1, 3], [3, 1]$ sont à dénombrer lors de l'appel à `nbrSommes(4,[1,2,3])`.

Exercice 30
On appelle permutation de l'ensemble $[\![1, n]\!]$ une bijection de cet ensemble dans lui-même. Proposer une fonction qui, prenant en paramètre n, retourne la liste des permutations de l'ensemble $[\![1, n]\!]$.
Exemple : `listePerm(3) -> [[1,2,3],[1,3,2],[2,1,3],[2,3,1],[3,1,2],[3,2,1]]`

Exercice 31 — Exercice des olympiade US, USACO
La plupart des sujets des Usaco relatent les aventures de John le fermier et de ses vaches, dont Bessie est la star incontestée.
Bessie suit un régime tel qu'elle ne peut pas manger plus de C ($10 \leqslant C \leqslant 35000$) calories par jour. John le fermier la taquine en plaçant B ($1 \leqslant B \leqslant 21$) seaux de fourrage, chacun ayant un nombre (pas nécessairement unique) de calories (dans l'intervalle $[\![1, 35000]\!]$). Bessie ne sait pas se contrôler : une fois qu'elle commence à manger le contenu d'un seau, elle mange tout ce qu'il contient.
Bessie n'est pas très douée pour la combinatoire. Déterminer la combinaison optimale de seaux de fourrage qui donne à Bessie autant de calories que possible sans dépasser la limite C.

Par exemple, considérer une limite de 40 calories et 6 seaux de dimensions 7, 13, 17, 19, 29 et 31. Bessie peut manger $7 + 31 = 38$ calories mais peut en manger encore plus en consommant trois seaux $7 + 13 + 19 = 39$ calories. Elle ne peut pas trouver de meilleure combinaison.

5 Solutions

Solution de l'exercice 1 -

```
def seuil(x):
    if x < 0:
        return 0
    elif x < 1:
        return x
    else:
        return 1
```

Solution de l'exercice 2 -

```
def facto(n):
    f = 1
    for i in range(n):
        f = f*(i+1)
    return f

def binomial(n, p):
    return facto(n)//facto(p)//facto(n-p)
```

Solution de l'exercice 3 -

```
def u(n):
    x = 1
    for i in range(n):
        x = (2 + x)/(1 + x)
    return x
```

Solution de l'exercice 4 -

```
def testPres(L,x):
    n=len(L)
    for k in range(n):
        if L[k] == x:
            return True
    return False
```

Solution de l'exercice 5 -

```
def maxPerso(L):
    indice=0
    max = L[0]
    for j in range(1, len(L)):
        if L[j]> L[indice]:
            indice = j
            max = L[j]
    return max
```

L'indice est en fait plus simple, on n'a pas besoin de garder le maximum.

```
def IndiceMax(L):
    indice=0
    for j in range(1,len(L)):
        if L[j]> L[indice]:
            indice = j
    return indice
```

Solution de l'exercice 6 -

```
def testPrime(n):
    k = 2
    while k*k <= n:
        if n%k == 0:
            return False
        k = k+1
    return(True)
```

Solution de l'exercice 7 -

```
def pgcd(a,b):
    commun = 1
    for k in range(1, min(a,b)+1):
        if (a % k == 0) and (b % k == 0 ):
            commun = k
    return commun
```

Solution de l'exercice 8 -

```
def est_croissante(liste):
    n = len(liste)
    out = True
    for i in range(n-k-1):
        if liste[i+1] < liste[i]:
            out = False
    return out
```

Si on veut s'arrêter le plus tôt possible

```
def est_croissante(liste):
    n = len(liste)
    i = 0
    while i < n-1 and liste[i] <= liste[i+1]:
        i = i + 1
    return i == n - 1
```

Solution de l'exercice 9 -

```
def suiteRec(u0, u1, u2, a, b, c, n):
    v0, v1, v2 = u0, u1, u2
    for k in range(n):
        temp= a*v2 + b*v1 + c*v0
        v0 = v1
        v1 = v2
        v2 = temp
    return v0
```

Solution de l'exercice 10 -

```
def base10To2(n):
    nb = n
    liste = []
    while nb > 0:
        liste.append(nb%2)
        nb = nb//2
    return liste

def base2to10(L):
    nb = 0
    n = len(L)
    for k in range(n):
        nb = nb + (2**k)*L[k]
    return nb
```

Solution de l'exercice 11 -

```
def testCarre(n):
    racine = int(n**0.5)
    return (racine**2 == n, racine)

def triplet():
    Liste=[]
    for a in range(1,100):
        for b in range(a,100):
            (booleen, racine) = testCarre(a*a+b*b)
            if  booleen:
                Liste.append([a,b,racine])
    return(Liste)
```

Solution de l'exercice 12 -

```
def binom(n, p):
    b = 1
    for i in range(p):
        b = b*(n-p+i+i)//(i+1)
    return b
```

Solution de l'exercice 13 -

```
def inverse(L):
    n=len(L)
    miroir =[]
    for k in range(n):
        miroir.append(L[-k-1])
    return miroir
```

```
def inverseSurPlace(L):
    '''L est modifie après appel a la fonction'''
    n=len(L)
    k=0
    while k<n-k-1:
        temp=L[k]
        L[k] = L[-k-1]
        L[n-k-1] = temp
        k=k+1
```

Solution de l'exercice 14 -

```
def DeuxMax(L):
    '''indice1 contient l'indice du max et indice2 contient l'
       indice du 2e max'''
    if L[0] <L[1]:
        indice1=1
        indice2=0
    else:
        indice1=0
        indice2=1
    for k in range(2,len(L)):
        if L[k]>L[indice1]:
            indice2=indice1
            indice1=k
        elif L[k]>L[indice2 ]: # dans ce cas L[indice2]<L[k]<L
           [indice1]
            indice2=k
    return (L[indice2], L[indice1])
```

Solution de l'exercice 15 -

```
def Euclide(a, b):
    while b > 0:
        r = a%b
        a = b
        b = r
    return a
```

Solution de l'exercice 16 -

```
def plusProches(L):
    n = len(L)
    indice1=0
    indice2=1
    distance = abs(L[1] - L[0])
    for i in range(n):
        for j in range(i+1,n):
            if abs(L[i]-L[j])< distance:
                distance=abs(L[i] - L[j])
                indice1=i
                indice2=j
    return (distance, L[indice1], L[indice2])
```

La complexité est quadratique, on peut l'améliorer en complexité linéaire si la liste est triée.

```
def plusProches2(L):
    n = len(L)
    indice=0
    distance = abs(L[1] - L[0])
    for i in range(n-1):
        if abs(L[i] - L[1+i])< distance:
                distance=abs(L[i] - L[1+i])
                indice=i
    return (distance, L[indice], L[1+indice])
```

Solution de l'exercice 17 -

```
def est_modale(liste):
    n = len(liste)
    i = 0
    while i < n-1 and liste[i] < liste[i+1]:
        i = i + 1
    while i < n-1 and liste[i] > liste[i+1]:
        i = i + 1
    return i == n - 1
```

Solution de l'exercice 18 -

```
def dicho(L,x):
    '''teste la presence de x dans L'''
    i = 0
    j = len(L)
    while i < j:       # recherche dans L[i:j] (tranche non
       vide)
        m = (i+j)//2  # indice du milieu de tranche
        if L[m] == x:
            return True
        if x < L[m]:
            j = m      # recherche dans la partie basse de la
               tranche
        else:
            i = m + 1 # recherche dans la partie haute de la
               tranche
    # en sortie i >= j donc la tranche L[i:j] est vide
    return False
```

Solution de l'exercice 19 -

```
def binom(n,p):
    liste=[1,1] # pour n=1
    for k in range(n-1): # ici liste est celle de k+1
        listeTemp = [1]
        for j in range(len(liste)-1):
            listeTemp.append(liste[j]+liste[j+1])
        liste=listeTemp
        liste.append(1)# ici liste est celle de k+2
    return liste[p]
```

On peut n'utiliser qu'une seule liste.

```
def binom(n,p):
    liste=[1]*(n+1)
    for m in range(n): on calcule les k parmi m
        for k in range(k-1, 0, -1):
            liste[k] = liste[k] + liste[k-1]
    return liste[p]
```

Solution de l'exercice 20 -

```
def listeSucces(L):
    liste = []
    lg = 0
    for k in range(len(L)):
        if L[k]:
            lg = lg + 1
        else:
            if lg > 0:
                liste.append(lg)
                lg = 0
    if lg > 0:
        liste.append(lg)
    return liste
```

Solution de l'exercice 21 -

```
def testInj(n,p,L):
    arrivee = [False]*(p+1) # la valeur d'indice 0 ne compte
       pas
    for y in L:
        if arrivee[y]:
            return False   # la valeur y est deja une image
        arrivee[y] = True  # desormais y est une image
    return True
```

```
def testSurj(n,p,L):
    arrivee = [False]*(p+1)
    for y in L:
        arrivee[y]=True # desormais  y est connue comme image
    for k in range(1,len(arrivee)):
        if not(arrivee[k]):
            return False # k n'a pas d'antecedent
    return True
```

Solution de l'exercice 22 -

```
def listeChiffres(n):
    nb = n
    liste = []
    while nb >0:
        liste.append(nb % 10)
        nb = nb//10
    return liste

def sommeCube(L):
    s = 0
    for x in L:
        s = x**3 + s
    return s

def testCube(n):
    return n == sommeCube(listeChiffre(n))

def cherche():
    for k in range(2, 1000):
        if testCube(k):
            print(k)
```

Solution de l'exercice 23 - Exercice 22 pour `listeChiffres`, exercice 13 pour `inverse`.

```
def est_miroir(n):
    chf = listeChiffres(n)
    fhc = inverse(chf)
    return chf == fhc

def sym(n):
    compteur=0
    k=10
    while compteur < n:
        k=k+1
        if est_miroir(k):
            compteur = compteur + 1
    return k
```

Solution de l'exercice 24 -

```
def listeDeniveles(L):
    liste=[]
    n=len(L)
    signe = L[1] - L[0]
    indiceDebut = 0
    for k in range(2, n):
        if signe*(L[k]-L[k-1]) < 0:
            signe =-signe
            liste.append(L[k-1]-L[indiceDebut])
            indiceDebut = k-1
    liste.append(L[n-1]-L[indiceDebut])
    return liste
```

Solution de l'exercice 25 -

```
def crible(n):
    tableau=[True]*(n+1) # l'indice 0 sera inutile
    for k in range(2, n+1):
        if tableau[k]:
            indice = 2*k
            while indice <= n:
                tableau[indice] = False
                indice = indice+k
    liste=[k for k in range(2, n+1) if tableau[k]]
    return liste
```

Solution de l'exercice 26 -

```
def goldbach(n):
    liste = crible(n)
    for j in liste:
        if n-j in liste:
            return j,n-j
```

Solution de l'exercice 27 -

```
def testFermat(n):
    Liste = []
    for k in range(2,100):
        if testPrime(k):
            Liste.append(k)
    for p in Liste:
        if (n%p != 0):
            if not((n**(p-1)-1)% p ==0):
                return False
    return True
```

Solution de l'exercice 28 -

```
def sommeCubes(n):
    somme3 = [False]*(n+1)
    somme3[0] = True
    for i in range(1, n+1):
        k = 2
        while k**3 <= i:
            somme3[i] = somme3[i] or somme3[i - k**3]
            k = k + 1
    return somme3[n]
```

Solution de l'exercice 29 -

```
def nbrSommesIter(n,L):
    [a,b,c]=L
    import numpy as np
    B=np.zeros((n+1,n+1),int)
    B[a][1]=1
    B[b][1]=1
    B[c][1]=1
    for j in range(2,n+1):
        for i in range(a,n+1):
            B[i][j] =   B[max(0,i-a)][j-1]
                      + B[max(0,i-b)][j-1]
                      + B[max(0,i-c)][j-1]
    return sum(B[n])
```

Solution de l'exercice 30 -

```
from copy import deepcopy

def listePerm(n):
    liste=[[1]]
    for k in range(2,n+1):
        newL = []
        for permutation in liste:
            for j in range(len(permutation)+1):
                newL.append(  permutation[:j]
                            + [k]
                            + permutation[j:])
        liste = deepcopy(newL)
    return liste
```

Solution de l'exercice 31 -

```
def bessie(CalMax, nbSeaux, Lprime):
    L = [0] + Lprime
    L.sort()
    B=np.zeros((1+nbSeaux, 1+CalMax))
    for p in range(1, 1+CalMax):
        if L[1] <= p:
            B[1][p] = L[1]
    for k in range(1+nbSeaux):
        for p in range(1,1+CalMax):
            if p<L[k]:
                B[k][p] = B[k-1][p]
            else:
                B[k][p] = max(B[k-1][p], B[k-1][p-L[k]]+L[k])
    return B[nbSeaux][CalMax]
```

TP II

Équations différentielles

- Dans le TP la variable des fonctions sera représentée par t. En particulier les équations différentielles seront sous la forme $y' = \varphi(y, t)$.
- Il arrivera que la variable t n'apparaisse pas directement dans l'équation : on devra néanmoins la faire intervenir dans la fonction python `def phi(y, t)`.
- Les fonctions mathématiques seront importées depuis le module `math`

```
import math as m
```

- Lors de cette étude nous allons régulièrement tracer le graphe des solutions.
  On aura calculé la liste des abscisses : `T` et la liste des valeurs approchées d'une fonction : `Y`.
  Pour tracer le graphe on utilise le module `matplotlib`.

```
import matplotlib.pyplot as plt
plt.plot(X, Y)
plt.show()
```

- Lorsque l'on trace plusieurs graphes dans un même cadre, on pourra donner une étiquette à chacun :

```
plt.plot(T, Y, label = "Ma jolie fonction")
```

  et faire afficher ces étiquette par `plt.legend()`.
- Des fonctions supplémentaires sont données dans le document du concours centrale.

1 Outils de base

Exercice 1
Écrire une fonction `listeT(a, b, n)` *qui calcule une liste de longueur n dont les valeurs sont équiréparties entre a et b.*

`listeT(1, 3, 5)` doit renvoyer `[1.0, 1.5, 2.0, 2.5, 3.0]`.
On remarquera que le pas est $\frac{b-a}{n-1}$.

Exercice 2
Écrire une fonction `appliquer(f, T)` *où f est une fonction et T est une liste et qui renvoie une liste Y de même longueur que T telle que* `Y[i] = f(T[i])`.

Pour la fonction $f : x \mapsto 3x + 1$, `appliquer(f, [0, 1, 2])` doit renvoyer `[1, 4, 7]`.

2 Méthode d'Euler

On rappelle la fonction de calcul de solution par la méthode d'Euler.

```
def euler(phi, y0, T):
    """Entrée : phi : fonction de ExR vers E
                 y0 élément de E, la condition initiale y0
                 T : une liste d'abscisses
       Sortie : une solution approchée de y' = f(t,y)
                 avec les conditions initiales (T[0],y0)
                 sous la forme d'une liste des valeurs
                 aux points de T"""
    n = len(T)
    Y = [0]*n
    Y[0] = y0 # ordonnée initiale
    for i in range(n-1): # il reste n-1 valeurs à calculer
        pas = T[i+1] - T[i]
        pente = phi(Y[i], T[i])
        Y[i+1] = Y[i] + pas*pente
    return Y
```

2.1 Un exemple simple

On s'intéresse à l'équation $y' = y$, sur $[0; 1]$ avec $y(0) = 1$. La solution est $x \mapsto \exp(x)$.

Exercice 3

Approcher les solutions de cette équation en utilisant la méthode d'Euler avec 5, 20 et 100 points. On tracera aussi le graphe associé à `math.exp(t)` *pour une liste de temps avec 100 points.*

Exercice 4

Dans l'exercice précédent, l'écriture naturelle est de répéter 3 fois les calculs (T, la solution, le graphe). Si c'est le cas, ré-écrire la solution en factorisant sous la forme :

```
for n in [5, 20, 100]:
     T =
     Y =
     plt.plot
```

Comme on connaît la solution exacte, on peut mesurer l'erreur commise par la méthode.
On admet que l'erreur est maximale pour $t = 1$.

Exercice 5 — Qualité de l'approximation

Écrire les instructions qui permettent d'afficher à l'écran la valeur absolue entre `m.exp(1)` *et le dernier terme de la liste retournée par la fonction* `euler` *pour des listes de* 10^p *points avec* $p \in \{1, 2, 3, 4, 5, 6, 7\}$.

Exercice 6 — Revenir en arrière

Tracer la solution de l'équation sur $[-1; 0]$

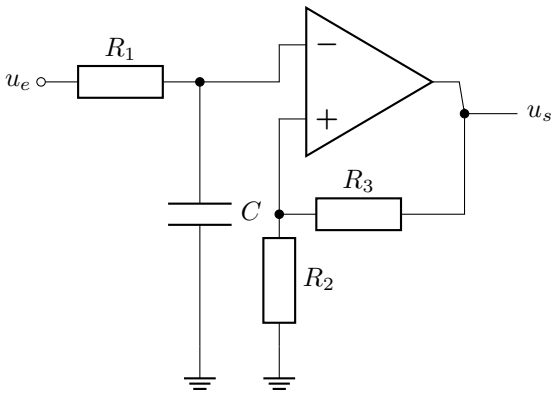
Le calcul est un peu long : on remarque que l'erreur est de l'odre de $\frac{1}{n}$.

2.2 Étude d'un filtre actif passe bas

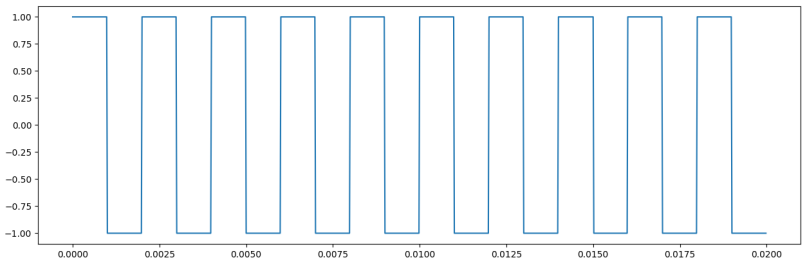
On peut réaliser un filtre actif passe-bas avec un A.O. (Amplificateur opérationnel).
Le fonctionnement de l'amplificateur opérationnel implique que la tension de sortie, u_s, est solution de

$$R_1 C y'(t) + y(t) = \frac{R_2 + R_3}{R_2} u_e(t)$$

où u_e est la tension d'entrée.



Les valeurs numériques sont les suivantes :$R_2 = R_3 = 1000\Omega$ et $C = 1\mu$F.
Différentes valeurs de R_1 seront testées afin de mettre en évidence les différents comportements.
Le signal d'entrée $u_e(t)$ est un signal rectangulaire de fréquence 500 Hz.



Exercice 7
Écrire une fonction `creneau(t,T)` *qui calcule la valeur d'un signal rectangulaire de période T : la valeur devra être 1 sur $[0; \frac{T}{2}[$ et -1 sur $[\frac{T}{2}; T[$.*

On pourra utiliser l'expression `t%T` qui renvoie le réel x appartenant à $[0; T[$ tel que $t - x$ est un multiple entier de T.

Exercice 8
En utilisant la méthode d'Euler déterminer alors les solutions de l'équation : on prendra une liste de temps de 1000 points entre 0 et 0,01s avec une condition initiale nulle.
On calculera les solutions pour $R_1 = 100\Omega$, $R_1 = 200\Omega$, $R_1 = 500\Omega$, $R_1 = 1000\Omega$ et $R_1 = 2000\Omega$.
On pourra tracer les solutions en même temps que le signal d'entrée.

2.3 Limites

On considère l'équation différentielle $y' = y - \dfrac{t+2}{(t+1)^2}$ avec la condition initiale $y(0) = 1$.

Exercice 9
Résoudre l'équation et tracer la solution sur $[0; 3]$, $[0; 10]$ et $[0; 30]$ avec 10 000 points.
Commenter sachant que la solution est $t \mapsto \frac{1}{t+1}$ et que la solution générale est $t \mapsto C.e^t + \frac{1}{t+1}$.

On considère l'équation différentielle $y' = \dfrac{-100.y}{\ln\left(\frac{t}{10} + 1.01\right)}$ avec la condition initiale $y(0) = 1$.

Exercice 10
Résoudre l'équation et tracer la solution sur $[0; 2]$ avec 10 000 points.
On pourra voir les détails à l'origine avec `plt.ylim([0, 0.02])` *; la solution devrait être une fonction qui décroît très rapidement vers 0.*

3 Modifications de la méthode d'Euler

3.1 Schéma de Heun

Le schéma de Heun améliore le schéma d'Euler en s'inspirant du calcul d'une intégrale par la méthode des trapèzes : $\int_a^b g(u)\mathrm{d}u \simeq (b-a)\dfrac{g(a)+g(b)}{2}$. On obtient

$$y(t_{k+1}) - y(t_k) \simeq (t_{k+1}-t_k)\frac{\varphi\big(y(t_k),t_k\big)+\varphi\big(y(t_{k+1}),t_{k+1}\big)}{2}$$

Cependant cela devient une équation implicite en $y(t_{k+1})$ qu'on ne souhaite pas résoudre.
On va donc procéder en approchant la valeur de $y(t_{k+1})$ dans le second membre par celle que calcule la méthode d'Euler.

- On calcule $z_k = y_k + \varphi(y_k,t_k)(t_{k+1}-t_k)$ comme valeur approchée de $y(t_{k+1})$
- On calcule la valeur approchée $y_{k+1} = y_k + \dfrac{\varphi(y_k,t_k)+\varphi(z_k,t_{k+1})}{2}(t_{k+1}-t_k)$.

Exercice 11
Écrire une fonction `heun(phi, y0, T)` *qui implémente le schéma de Heun.*

Exercice 12
Utiliser le schéma de de Heun pour résoudre $(t+1)y'(t)+y(t)=e^t$, $y(0)=1$ *sur* $[0;2]$ *avec 5, 10 et 20 points. La solution exacte est* $t \mapsto \frac{e^t}{t+1}$.

On revient à l'équation $y' = y$ sur $[0;1]$ avec $y(0)=1$.

Exercice 13 — Qualité de l'approximation
Écrire les instructions qui permettent d'afficher à l'écran la valeur absolue entre `m.exp(1)` *et le dernier terme de la liste retournée par la fonction* `heun` *pour des listes de* 10^p *points avec* $p \in \{1,2,3,4,5,6,7\}$.

3.2 Équations d'ordre 2

On peut utiliser le principe du schéma d'Euler pour résoudre des équations d'ordre 2 (ou plus).
Ce sont des équations de la forme $y'' = \psi(y,y',t)$.
Par exemple l'équation $y'' + y' + ty = \sin(t)$ sera définie par

```
def psi0(y, v, t):
    return m.sin(t) - v - t*y
```

Les conditions initiales seront données sous la forme $y(t_0)=y_0$, $y'(t_0)=v_0$.
Le cas des conditions aux limites, $y(t_0)=y_0$ et $y(t_{final})=y_{final}$, plus difficile, ne sera pas traité ici.
On peut résoudre ces équations en suivant le même principe que dans le cas des équations d'ordre 1 en incluant le calcul des dérivées aux temps t_i. On cherchera donc à définir deux listes `Y`, pour les valeurs de la fonctions, et `V`, pour les valeurs de sa dérivée. La valeur de `Y[i]`, y_i, et celle de $V[i]$, v_i seront des valeurs approchées de $f(t_i)$ et $f'(t_i)$ respectivement où f est la solution telle que $f(t_0)=y_0$ et $f'(t_0)=v_0$.
Le schéma d'Euler se décompose alors en deux calculs. On note $\delta_i = t_{i+1}-t_i$:

$$y_{i+1} \simeq f(t_{i+1}) \simeq f(t_i) + \delta_i f'(t_i) \simeq y_i + \delta_i v_i \text{ et}$$

$$v_{i+1} \simeq f'(t_{i+1}) \simeq f'(t_i) + \delta_i f''(t_i) = f'(t_i) + \delta_i \psi\big(f(t_i),f'(t_i),t_i\big) \simeq v_i + \delta_i \psi(y_i,v_i,t_i)$$

Exercice 14
Écrire une fonction `Euler2(phi, y0, v0, T)` *qui résout une équation différentielle d'ordre 2 définie par* ψ *avec les conditions initiales* y_0 *et* v_0 *en* t_0 *aux points définis par la liste des temps* `T`. *La fonction renverra deux listes de même taille que* `T`, *la première contiendra les valeurs approchées de la solution, l'autre les valeurs approchées de sa dérivée.*

3.3 Schéma de Runge-Kutta

La méthode de Runge-Kutta s'inspire de la méthode de Simpson :

$$\int_a^b g(u)\mathrm{d}u \simeq (b-a)\frac{g(a)+4g\big(\frac{a+b}{2}\big)+g(b)}{6}$$

On obtient, pour $m_k = \frac{t_k+t_{k+1}}{2}$ et $h = t_{k+1}-t_k$,

$$y(t_{k+1}) \simeq y(t_k) + h\frac{\varphi\big(y(t_k),t_k\big)+4\varphi\big(y(m_k),m_k\big)+\varphi\big(y(t_{k+1}),t_{k+1}\big)}{6}$$

Comme dans la méthode de Heun, on calcule des valeurs de de $y(m_k)$ et $y(t_{k+1})$ par des approximations simples ; en fait on calcule deux approximations de $y(m_k)$.

- $a = y_k + \frac{h}{2}\varphi\big(y_k,t_k\big)$ est une première valeur approchée de $y(m_k)$,
- $b = y_k + \frac{h}{2}\varphi\big(a,t_k+\frac{h}{2}\big)$ est aussi une valeur approchée de $y(m_k)$,
- $c = y_k + h\varphi\big(b,t_k+\frac{h}{2}\big)$ est une valeur approchée de $y(t_{k+1})$.
-
- On pose alors
  $y_{k+1} = y_k + h\frac{\varphi\big(y_k,t_k\big)+2\varphi\big(a,t_k+\frac{h}{2}\big)+2\varphi\big(b,t_k+\frac{h}{2}\big)+\varphi\big(c,t_k+h\big)}{6}$

Exercice 15
Écrire une fonction `RK(phi, y0, T)` *qui donne une approximation de la solution de* $y' = \varphi(y,t)$ *avec la condition initiale* $y(t_0) = y_0$ *aux points de la liste* T *par la méthode de Runge-Kutta.*

3.4 Schéma d'Euler implicite

La formule d'approximation utilisée : $f(x_{i+1}) \simeq f(x_i) + (x_{i+1}-x_i)f'(x_i)$ peut être lue dans le sens inverse : $f(x_i) \simeq f(x_{i+1}) + (x_i - x_{i+1})f'(x_{i+1}) = f(x_{i+1}) + (x_i - x_{i+1})\varphi\big(f(x_{i+1},x_{i+1}\big)$. En remplaçant les valeurs par leurs valeurs approchées on aboutit au schéma d'Euler implicite

$$y_i == y_{i+1} + (t_i - t_{i+1})\varphi(y_{i+1},t_{i+1})$$

On voit qu'on ne peut pas directement exprimer y_{i+1} en fonction de y_i ; il faut résoudre, à chaque étape, l'équation $g_i(y) = 0$ avec $g_i(y) = y + (t_i - t_{i+1})\varphi(y_{i+1},t_{i+1}) - y_i$.

Dans certains cas, rares, cela est possible simplement.
Par exemple si on veut résoudre l'équation différentielle $y' = y^2 + t^2$ pour $y(t_0) = y_0$ sur $[t_0;t_1]$ par la méthode d'Euler implicite, on doit doit calculer les y_i à partir de y_0 et $y_i = y_{i+1} + (t_i - t_{i+1})\big(y_{i+1}^2 + t - i + 1^2\big)$.
Ainsi y_{i+1} est racine d'un polynôme de degré 2 : on choisira la racine proche de y_i.

Exercice 16
Écrire une fonction `implicite(y0, T)` *qui donne une approximation de la solution de* $y' = y^2+t^2$ *avec la condition initiale* $y(t_0) = y_0$ *aux points de la liste* T *à l'aide du schéma d'Euler implicite. On notera que cette fonction ne sait résoudre qu'une seule équation.*

Dans le cas général on peut utiliser la fonction `fsolve` du module `scipy.optimize` pour résoudre l'équation en y_{i+1}. On devra, à chaque étape, définir une nouvelle fonction qu'il faudra envoyer comme paramètre à `fsolve`. On prendra y_i comme valeur initiale.

Exercice 17
Écrire une fonction `eulerImplicite(phi, y0, T)` *qui donne une approximation de la solution de* $y' = \varphi(y,t)$ *avec la condition initiale* $y(t_0) = y_0$ *aux points de la liste* T *par la méthode d'Euler implicite.*

4 Solutions

Solution de l'exercice 1 -

```
def listeT(a, b, n):
    pas = (b-a)/(n-1)
    T = [0]*n
    for k in range(n):
        T[k] = a + k*pas
    return T
```

Solution de l'exercice 2 -

```
def appliquer(f, T):
    n = len(T)
    Y = [0]*n
    for i in range(n):
        Y[i] = f(T[i])
    return Y
```

Solution de l'exercice 3 -

```
import math as m

def phi1(y, t) :
    return y

y0 = 1

T = listeT(0, 1, 5)
Y = euler(phi1, y0, T)
plt.plot(T, Y, label = "Euler avec 5 points")

T = listeT(0, 1, 20)
Y = euler(phi1, y0, T)
plt.plot(T, Y, label = "Euler avec 20 points")


T = listeT(0, 1, 100)
Y = euler(phi1, y0, T)
plt.plot(T, Y, label = "Euler avec 100 points")


T = listeT(0, 1, 100)
Y = appliquer(m.exp, T)
plt.plot(T, Y, label = "Solution exacte")

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

Solution de l'exercice 4 -

```
import math as m

def phi1(y, t) :
    return y

y0 = 1

for n in [5, 20, 100]:
    T = listeT(0, 1, n)
    Y = euler(phi1, y0, T)
    plt.plot(T, Y, label = "Euler avec {} points".format(n))

T = listeT(0, 1, 100)
Y = appliquer(m.exp, T)
plt.plot(T, Y, label = "Solution exacte")

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

Solution de l'exercice 5 -

```
import math as m

def phi1(y, t) :
    return y

y0 = 1

for n in [5, 20, 100]:
    T = listeT(0, 1, n)
    Y = euler(phi1, y0, T)
    plt.plot(T, Y, label = "Euler avec {} points".format(n))

T = listeT(0, 1, 100)
Y = appliquer(m.exp, T)
plt.plot(T, Y, label = "Solution exacte")

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

Solution de l'exercice 6 -

```
y0 = 1

T = listeT(0, -1, 1000)
Y = euler(phi1, y0, T)
plt.plot(T, Y)

plt.show()
```

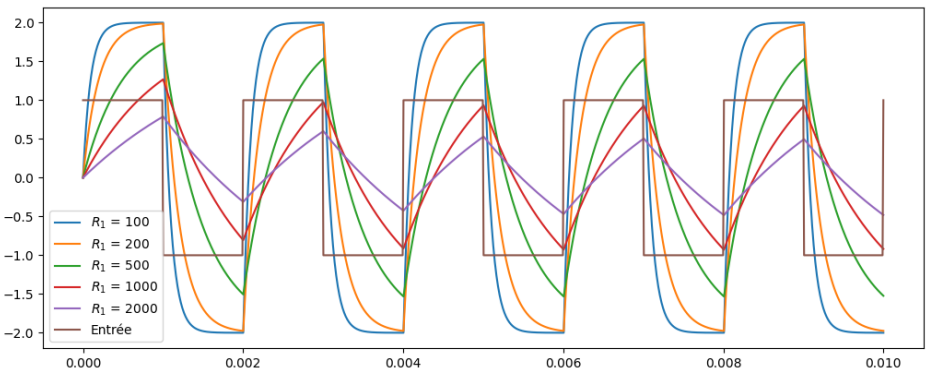
Solution de l'exercice 7 -

```
def creneau(t, T):
    x = t%T
    if x < T/2:
        return 1
    else:
        return -1
```

Solution de l'exercice 8 -

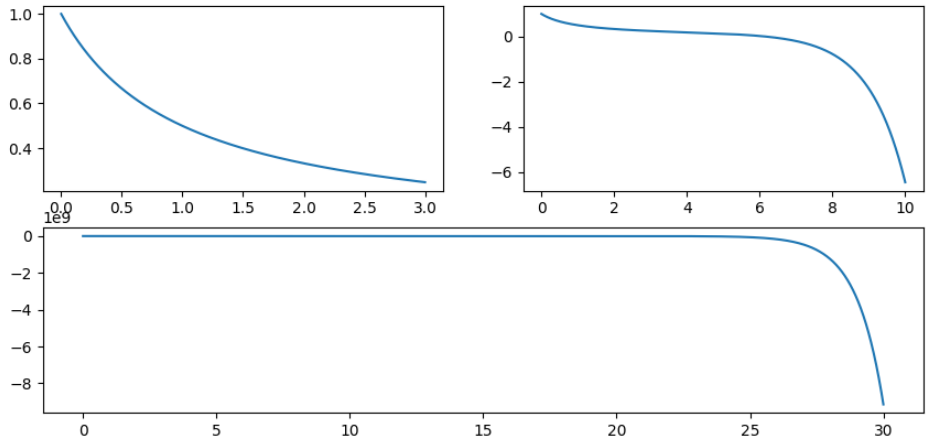
```
def filtre(y, t):
    return ((R2 + R3)/R2*creneau(t, per) - y)/R1/C
```

```
R2 = 1000
R3 = 1000
f = 500
per = 1/f
C = 1e-6
T = listeT(0, 0.01, 1000)
for R1 in [100, 200, 500, 1000, 2000]:
    Y = Euler(filtre, 0, T)
    plt.plot(T, Y, label = "$R_1$ = {}".format(R1))

def cr(t):
    return creneau(t, per)
Y = appliquer(cr, T)
plt.plot(T, Y, label = "Entrée")
plt.legend()
plt.show()
```



Solution de l'exercice 9 -

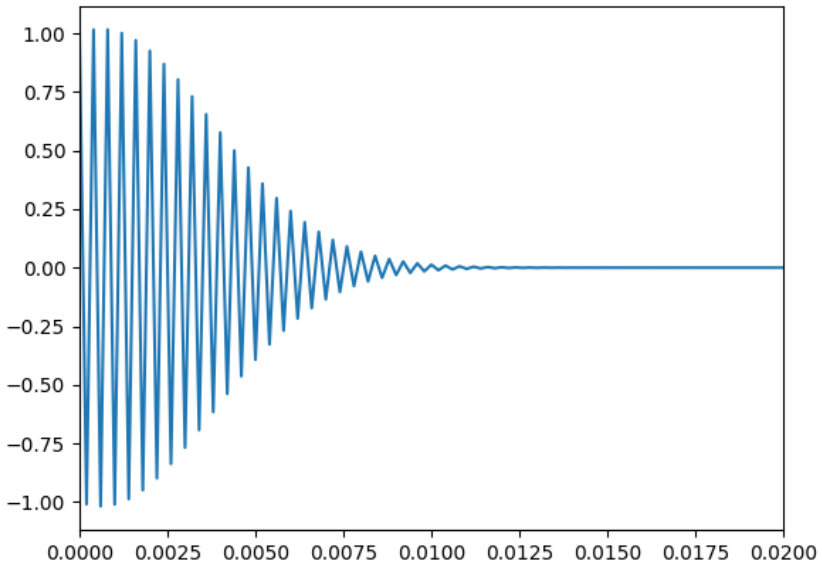
```
y0 = 1
t_max = 30

T = listeT(0, t_max, 10000)
Y = euler(phi2, y0, T)
plt.plot(T, Y)
plt.show()
```



Les erreurs d'arrondis des flottants font qu'on a en fait une solution $t \mapsto C.e^t + \frac{1}{t+1}$ avec C petit mais le terme exponentiel, qui croît rapidement, finit par ne plus être négligeable.

Solution de l'exercice 10 -

```
def phi3(y, t) :
    return - y *100/m.log(t/10+1.01)

y0 = 1
t_max = 2
n = 10000
T = listeT(0, t_max, n)
Y = euler(phi3, y0, T)
plt.plot(T, Y)
plt.xlim([0, 0.02])
plt.show()
```



La dérivée est très grande et engendre des oscillations dues au dépassement des y_i de chaque coté de 0.

Solution de l'exercice 11 -

```
def heun(phi, y0, T):
    n = len(T)
    Y = [0]*n
    Y[0] = y0
    for i in range(n-1):
        y = Y[i]
        pas = T[i+1] - T[i]
        pente_g = phi(y, T[i])
        z = y + pas*pente_g
        pente_d = phi(z, T[i+1])
        pente = (pente_g  + pente_d)/2
        Y[i+1] = y + pas*pente
    return Y
```

Solution de l'exercice 12 -

```
def phi4(y, t) :
    return (m.exp(t) - y )/(t+1)

def f4(t) :
    return m.exp(t)/(t+1)

y0 = 1
t_max = 5

for n in [10, 100]:
    T = listeT(0, t_max, n)
    Y = heun(phi4, 1, T)
    plt.plot(T, Y, label='avec {} points'.format(n))
T = listeT(0, t_max, 1000)
Y = appliquer(f4, T)
plt.plot(T, Y, label = 'Solution exacte')
plt.legend()
plt.show()
```

Solution de l'exercice 13 -

```
y0 = 1
t_max = 1
for k in range(1, 8):
    n = 10**k
    T = listeT(0, t_max, n)
    Y = heun(phi1, y0, T)
    err = abs(m.exp(t_max) - Y[-1])
    print("Erreur : {} pour {} points".format(err, n))
```

L'erreur diminue d'un facteur 100 chaque fois qu'on multiplie par 10, sauf à la fin où les erreur d'arrondis ne permettent plus d'augmenter la précision des calculs.

Solution de l'exercice 14 -

```
def Euler2(psi, y0, v0, T):
    n = len(T)
    Y = [0]*n
    Y[0] = y0
    V = [0]*n
    V[0] = v0
    for i in range(n-1):
        pas = T[i+1] - T[i]
        Y[i+1] = Y[i] + pas*V[i]
        V[i+1] = V[i] + pas*psi(Y[i], V[i], T[i])
    return Y, V
```

Solution de l'exercice 15 -

```
def RK(f, y0, T):
    n = len(T)
    Y = [0]*n
    Y[0] = y0
    for i in range(n-1):
        y = Y[i]
        t = T[i]
        t1 = T[i+1]
        pas = t1 - t
        pente1 = f(y,t)
        a = y + pente1*pas/2   # 1ère valeur au milieu
        pente2 = f(a, t+pas/2) # 1ère pente au milieu
        b = y + pente2*pas/2   # 2ème valeur au milieu
        pente3 = f(b, t+pas/2) # 2ème pente au milieu
        c = y + pente3*pas     # valeur à droite
        pente4 = f(c,t1)       # pente à droite
        pente = (pente1 + 2*pente2 + 2*pente3 + pente4)/6
        Y[i+1] = y + pas*pente
    return Y
```

Solution de l'exercice 16 - Les racines sont $\dfrac{1 \pm \sqrt{1 - 4hy_i - 4h^2t_{i+1}^2}}{2h}$ avec $h = t_{i+1} - t_i$; celle qui est proche de y_i est $\dfrac{1 - \sqrt{1 - 4hy_i - 4h^2t_{i+1}^2}}{2h}$.

```
def implicite(y0, T):
    n = len(T)
    Y = [0]*n
    Y[0] = y0
    for i in range(n-1):
        pas = T[i+1] - T[i]
        delta = 1 - 4*pas*(Y[i]+pas*T[i+1]**2)
        Y[i+1] = (1-delta**(1/2))/(2*h)
    return Y
```

Solution de l'exercice 17 -

```
from scipy.optimize import fsolve

def eulerImplicite(phi, y0, T) :
    n = len(T)
    Y = [0]*n
    Y[0] = y0
    for i in range(n-1):
        # On féfinit une fonction dans la fonction
        def g(x):
            return x - Y[i] - (T[i+1]-T[i])*phi(x, T[i+1])
        Y[i+1] = fsolve(g,Y[i])
    return Y
```

TP III

Systèmes différentiels

Résumé

- *Dans le TP nous allons prolonger l'étude des équations différentielles en quittant le cas des équations scalaires d'ordre 1.*
- *Dans un premier temps nous allons voir qu'il n'y a rien à changer quand on a plusieurs équations différentielles couplées : les dérivées des fonctions varient en fonction des valeurs de* **plusieurs** *fonctions inconnues.*
- *Nous verrons ensuite comment se ramener au cas précédent dans le cas d'équations d'ordre 2 ou plus.*

1 Complément de cours

1.1 Présentation

Il arrivera souvent que les modélisations des phénomènes aboutissent à plusieurs fonctions inconnues solutions différentielles dans lesquelles plusieurs fonctions interviennent, des systèmes différentiels. Par exemple Lorenz, en simplifiant des équations intervenant en météorologie, est parvenu au système

$$\begin{cases} x' &= \sigma(y-x) \\ y' &= \rho x - y - xz \\ z' &= xy - \beta z \end{cases}$$

Dans certains cas (rares) le sens physique ou chimique permet de détecter des symétries qui, après changement de variables, permettent de découpler les équations et d'obtenir plusieurs équations d'ordre 1 simple. Ce sera aussi le cas dans le cas de systèmes linéaires à coefficients constants dont l'étude sera vue en mathématiques et utilise la diagonalisation.

Par exemple $\begin{cases} x' &= 2x+2y \\ y' &= x+3y \end{cases}$ se transforme en $\begin{cases} u' &= 4u \\ v' &= v \end{cases}$ en posant $u = 2+2y$ et $v = x-y$.

1.2 Une idée

Nous allons ici donner une méthode qui permettra de résoudre numériquement, c'est-à-dire de manière approchée, les systèmes différentiels généraux.

L'idée est de considérer les différentes équations dans leur ensemble en formant la fonction **vectorielle** qui rassemble des fonctions inconnues. Dans le cas des équations de Lorenz, on pose $M(t) = \big(x(t), y(t), z(t)\big)$.

La dérivée de la fonction vectorielle est le vecteur dont les composantes sont les dérivées, il s'exprime donc en fonction de la fonction initiale et du temps (le temps n'apparaît pas toujours).

$$M'(t) = \varphi\big(M(t), t\big)$$

On est ramené à ce que l'on connaît déjà, avec la nouveauté que les fonctions inconnues sont à valeurs dans un ensemble $\mathbb{R}^n$ et que la première variable de φ appartient à cet espace $\mathbb{R}^n$.

La méthode d'Euler s'applique toujours : $M(t_{k+1}) \sim M(t_k) + (t_{k+1} - t_k)M'(t_k)$.
Cependant les opérations qui interviennent ici sont l'addition de deux vecteurs et le produit d'un vecteur par un scalaire, ce sont les opérations définie dans tout espace vectoriel.

1.3 Des vecteurs en Python

Un moyen naturel de rassembler plusieurs valeurs en Python est d'utiliser une liste.
Par exemple, dans le cas des équations de Lorenz, on travaille avec des objet des la forme `[x, y, z]`.
Malheureusement l'addition de deux listes et le produit d'une liste (seulement par un entier) ne sont pas les opérations souhaitées. On pourrait les définir "à la main".
Heureusement, il existe le module **`numpy`** :

```
import numpy as np
```

Il définit un type de données, **`array`**, qu'on nommera vecteur.
Si `U` est un vecteur, il ressemble beaucoup à une liste Python :

- on accède et on modifie les éléments par leur indice : `U[i] =` , `x = U[i]`,
- la longueur du tableau est définie par `len(U)`,
- on peut parcourir les éléments par `for u in U`,
- on peut extraire des sous-vecteurs : `U[i : j]`

On définit un vecteur

- soit en convertissant une liste, `U = np.array(L)`
- soit en définissant un tableau de valeurs nulles, `U = np.zeros(n)` puis en le remplissant.
- **On ne peut pas construire un vecteur par adjonctions, par de méthode** `append`.

Ensuite les vecteurs réagissent à l'addition et à la multiplication par un scalaire comme des vecteurs d'un espace vectoriel.

```
>>> a = np.array([1.0, 3.5, -1.2])
>>> b = np.array([2.1, 0.3, 12.4])
>>> a + 1.75*b
array([ 4.675,  4.025, 20.5  ])
```

1.4 Résolution

Pour résoudre un système différentiel, il y a donc peu de choses qui changent.

- On définit la fonction qui décrit le système différentiel `phi(u, t)` mais `u` sera un vecteur de longueur n, le nombre de variables et la fonction renvoie un vecteur de même taille.

```
def phi(u, t):
    x, y, ... = u
    x_prime = ...
    y_prime = ...
    ...
    return np.array([x_prime, y_prime, ...])
```

- La condition initiale est un vecteur dont les composantes sont les valeurs et t_0 de chacune des variables.
- La liste des temps est identique ; on pourra utiliser `np.linspace(a, b, n)` qui construit un vecteur de taille n dont les valeurs vont de a à b compris et sont régulièrement espacées.
- On utilise les fonctions de résolution, `euler` ou autre **sans rien changer.**

1.5 Un exemple

1. On se donne le système différentiel

$$\begin{cases} x' &= \frac{xy}{x^2+y^2} \\ y' &= \frac{y^2+a^2-ax}{x^2+y^2} \end{cases}$$

   On pourra choisir $a = 1$.
2. On commence par définir la fonction φ.

```
a = 1

def phi(u, t):
    x, y = u
    den = x**2 + y**2
    dx = x*y/den
    dy = (y**2 + a**2 - a*x)/den
    return np.array([dx, dy])
```

3. On définit ensuite la condition initiale en $t = 0$: `u0 = np.array([0.5, 0.0])`,
4. puis une liste de temps : 2000 points sur $[0; 5]$.

```
t_max = 5
N = 2000
T = np.linspace(0, t_max, N)
```

5. On peut alors calculer les solutions : `U = euler(phi, u0, T)`
6. `U` est une liste de vecteurs de taille 2, on isole les composantes.

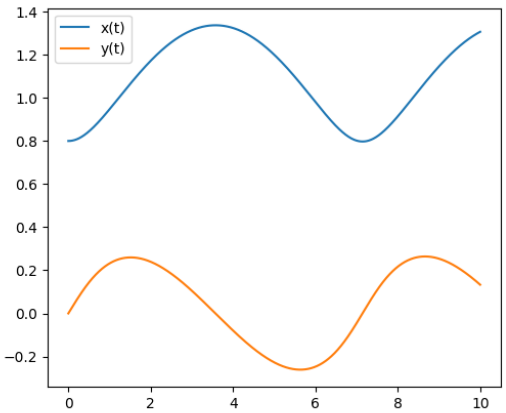
```
X = [u[0] for u in U]
Y = [u[1] for u in U]
```

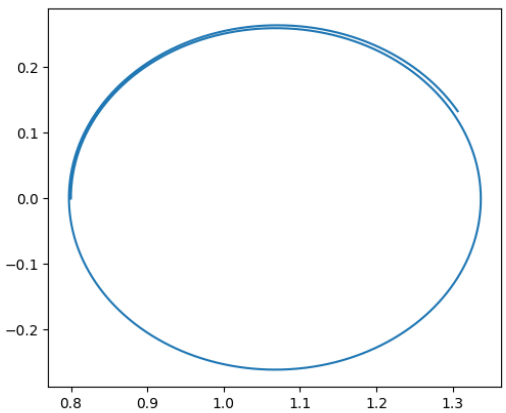
7. On peut alors tracer les solutions

```
plt.plot(T, X, label = "x(t)")
plt.plot(T, Y, label = "y(t)")
plt.legend()
plt.show()
```

8. Comme les variables représentent des coordonnées, on peut utilement ici tracer la trajectoire

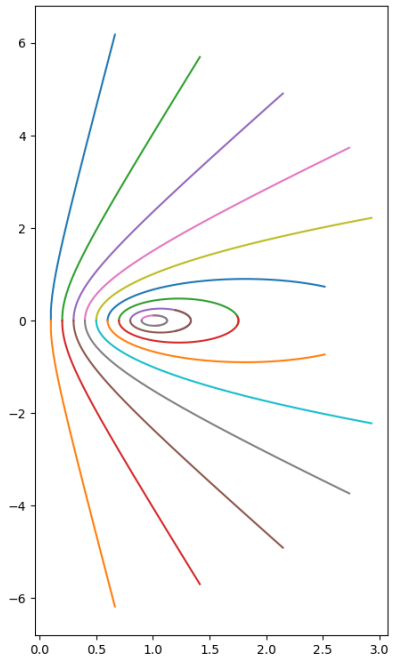
```
plt.plot(X, Y)
plt.show()
```





9. En théorie, les trajectoires bornées se referment.
   Le graphe ci-dessus présente une divergence.
   On peut obtenir une meilleure approximation en augmentant le nombre de points.
   On peut aussi utiliser une méthode plus précise ou, plus simplement, utiliser la fonction `odeint` qui s'emploie exactement de la même façon que `euler`.

```
from scipy.integrate import odeint
```



10. On rappelle qu'on peut tracer les graphes sur $[-t_{Max};0]$ avec la même condition initiale en définissant une liste de temps à l'envers :

```
T = np.linspace(0, -t_max, N)
```

11. Pour calculer plusieurs solution, par exemple avec des conditions initiales distinctes, on peut inclure les calculs dans une boucle.

```
for x0 in (range(1, 10)):
    x0 = i*a/10
    u0 = np.array([x0 , 0])
```

    Dans cette exemple on obtient des coniques de mêmes foyer et directrices.

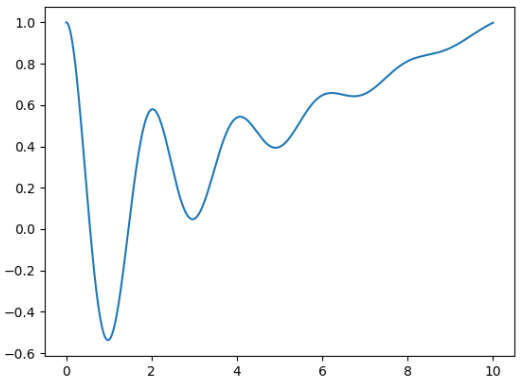
1.6 Équations d'ordre 2

Pour résoudre des équations d'ordre 2, on peut modifier la méthode d'Euler en calculant les valeurs de la fonction et celles de sa dérivée. Pour l'équation $y'' = \psi(y, y', t)$, on note $v(t) = y'(t)$ et on a alors $v'(t) = y''(t) = \psi\big(y(t), y'(t), t\big)$.

On aboutit ainsi à un système $\begin{cases} y' &= & 2v \\ v' &= & \psi(y, y', t) \end{cases}$

Exemple : $y'' + y' + 10y = t$, $y(0) = 1$, $y'(0) = 0$.

```
def phi(u, t):
    y, v = u
    acc = t - 10*y - v
    return np.array([v, acc])

u0 = np.array([1, 0])
T = np.linspace(0, 10, 2000)
U = euler(phi, u0, T)
Y = [u[0] for u in U]
plt.plot(T, Y)
```

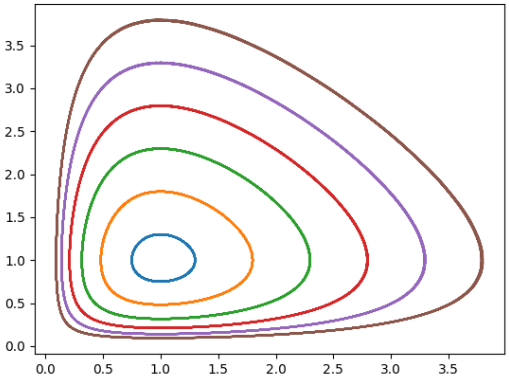


2 Exercices

2.1 Proies-prédateurs

Un système classique pour modéliser les interactions entre proies et prédateurs a été obtenu par Lotka et Volterra qui voulaient comprendre les évolutions des populations de requeins et de thon en Adriatique après la première guerre mondiale. Après simplification des constantes on arrive au système suivant.

$$\begin{cases} x' &= x - xy \\ y' &= xy - y \end{cases}.$$



Exercice 1
Tracer les solutions de solutions du système pour différentes conditions initiales $(a, 1)$ *avec* $a > 1$ *sur* $[0; 20]$

2.2 Simulation d'une épidémie

On modélise la propagation d'une épidémie en séparant la population en 3 :

- les individus **sains** dont la proportion est représentée par une fonction $S(t)$,
- les personnes **infectées** de proportion $I(t)$ et
- les guéris comptés par $R(t)$ (pour **recovered**).

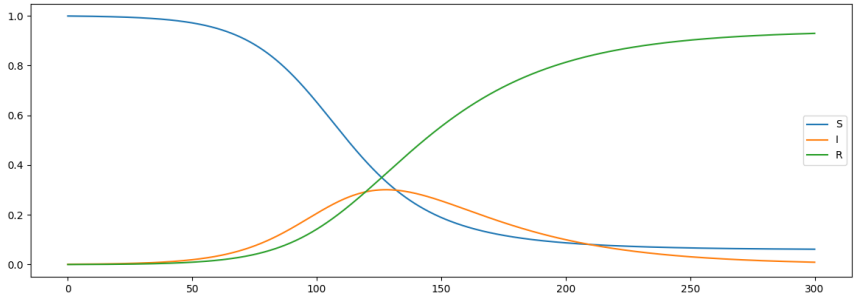
On parle du modèle SIR.
On définit deux constantes β et γ qui représente le taux de transmission pour β, c'est à dire le taux de personnes saines qui deviennent infectées par contact avec les personnes infectées par unité de temps et le taux de guérison pour γ, c'est à dire le taux de personnes infectées qui deviennent guéries par unité de temps.
On suppose que les personnes guéries ne peuvent plus être infectées.

Mathématiquement, le modèle SIR est donné par le système suivant :

$$\begin{cases} \dfrac{\mathrm{d}S}{\mathrm{d}t}(t) &= -\beta S(t)I(t) \\ \dfrac{\mathrm{d}I}{\mathrm{d}t}(t) &= \beta S(t)I(t) - \gamma I(t) \\ \dfrac{\mathrm{d}R}{\mathrm{d}t}(t) &= \gamma I(t) \end{cases}$$

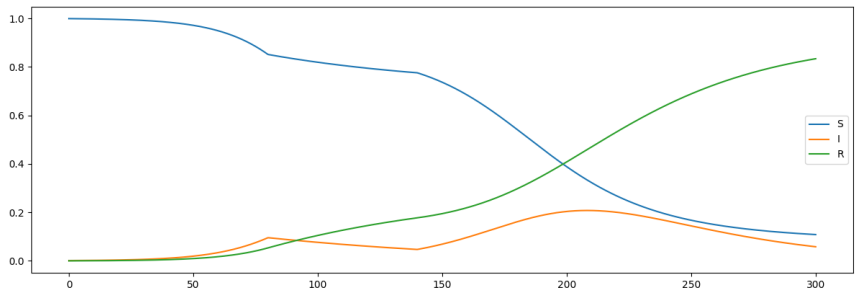
On note R_0 le rapport $\dfrac{\beta}{\gamma}$, il mesure la propagabilité de la maladie.
On prendra comme valeurs $R_0 = 3$ et $\gamma = 0.03$.
On considérera qu'on a $I(0) = 10^{-3}$ et $R(0) = 0$ et on étudiera pour $t \in [0; 300]$

Exercice 2
Tracer, sur un même graphique, les proportions de chaque type de population.



On ajoute une période de confinement entre les temps $t = 80$ et $t = 140$, pendant laquelle le coefficient R_0 est divisé par 4.

Exercice 3
Après avoir défini une nouvelle fonction ψ, tracer, sur un même graphique, les proportions de chaque type de population.

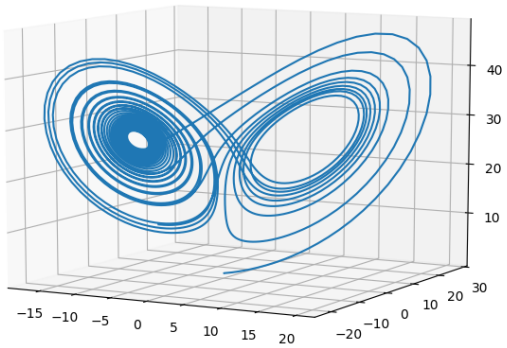


2.3 Équations de Lorenz

Lors de l'étude des phénomènes météorologiques, Edward Lorenz a proposé, en 1963, les équations issues de la mécanique des fluides en un système dynamique tridimensionnel. Ce système engendre un comportement chaotique dans certaines conditions. Il est donné par

$$\begin{cases} x' &= \sigma(y-x) \\ y' &= \rho x - y - xz \\ z' &= xy - \beta z \end{cases}$$

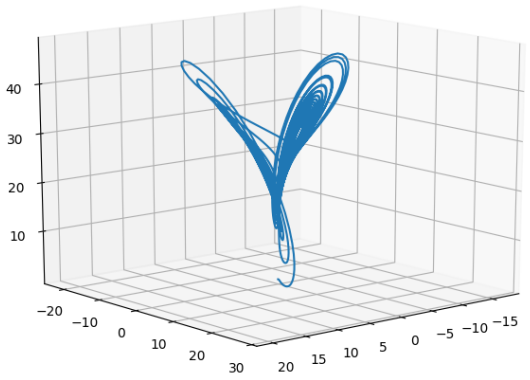
avec $\sigma = 10$, $\rho = 28$ et $\beta = 8/3$.



Exercice 4
Tracer le portrait de phase sur $[0;30]$ pour les conditions initiales $x(0) = 0.1$, $y(0) = 0$ et $z(0) = 0.1$.

On pourra tracer une représentation en 3 dimension par

```
from mpl_toolkits.mplot3d
    import Axes3D
# Un cadre contenant les tracés
dessin = Axes3D(plt.figure())
# On y ajoute un tracé
dessin.plot(x, y, z)
```



2.4 Équations d'ordre 2

Exercice 5
Tracer les solutions de $y'' = -2y' + 3y$ avec $y(0) = 1$ et $y'(0) = -3$.
Remarquez et essayez d'expliquer ce qui se passe pour les grandes valeurs de t.

Exercice 6
Tracer les solutions de l'équation de van der Pol : $y'' = (1-y^2)y' - y$ pour différentes conditions initiales, par exemple $y_0 = \frac{k}{2}$, $y_0' = 0$ pour $k \in \{0, 1, \ldots, 9\}$.
On pourra tracer la trajectoire dans l'espace des phases (y, y'). Que remarque-t-on ?

2.5 Effet Zeeman

L'électron dans l'atome est classiquement modélisé comme «élastiquement lié» : un point matériel de masse m et de charge électrique e, repéré par sa position $\vec{r}$, et soumis de la part du noyau à une force de rappel élastique, anticolinéaire à $\vec{r}$. Il se comporte alors comme un oscillateur harmonique de pulsation ω_0.
Si l'atome est soumis à un champ magnétique $\vec{B}$ colinéaire à $\vec{u}_z$, il subit une seconde force d'origine magnétique ; il se produit alors un phénomène appelé effet Zeeman :

- L'électron garde un mouvement oscillant à la pulsation ω_0 le long du champ magnétique.
- Il développe un mouvement bi-harmonique dans le plan perpendiculaire au champ magnétique, aux pulsations $\omega_0 \pm \Omega$, où Ω est une pulsation proportionnelle au champ magnétique.
- L'atome peut alors émettre un rayonnement sur les trois pulsations ω_0, $\omega_0 - \Omega$ et $\omega_0 + \Omega$.

Le mouvement suivant z est un simple oscillateur harmonique découplé des deux autres, donc il ne sera pas étudié.

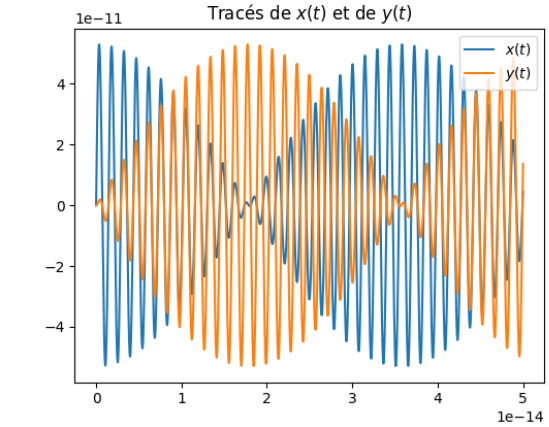
L'équation différentielle gouvernant le vecteur position est $\vec{r}''(t) + e\vec{r}'(r) \wedge \vec{B} + \omega_0^2 \vec{r} = \vec{0}$. Elle se traduit en le système $\begin{cases} \ddot{x} + \omega_0^2 x &= -2\Omega\dot{y} \\ \ddot{y} + \omega_0^2 y &= 2\Omega\dot{x} \end{cases}$ avec $\Omega = \frac{eB}{2m}$.
L'équation différentielle portera donc sur 4 variables :

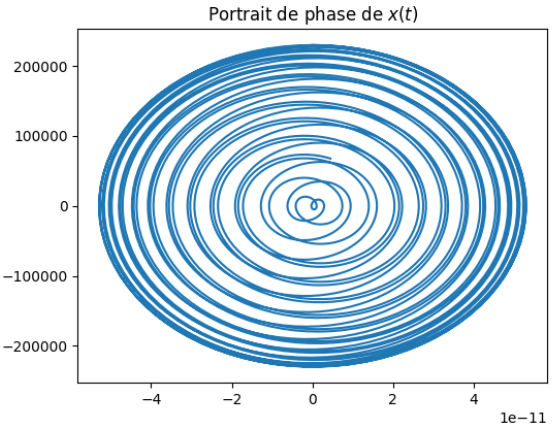
```
def psi4(u, t):
    x, y, vx, vy = u
    ...
```

On choisit, pour conditions initiales, $\vec{r}(0) = -a\cos(\theta_0)\vec{u}_y$ et $\vec{r}'(0) = a\omega_0\vec{u}_x$.
Valeurs numériques : [a]

- Électron : $m = 9,1.10^{-31}$kg,
  $e = 1,6.10^{-19}$C,
  $\omega_0 = 4,34.10^{15}$rads^{-1}
- $a = 5,3.10^{-11}$m, $\theta_0 = \pi/2$
- $\vec{B} = B\,\vec{u}_z$ avec $B = 1000$T

Exercice 7
Tracer le graphe des solutions $x(t)$ et $y(t)$ sur $[0; 5.10^{-14}]$.
Tracer le diagramme des phases pour la solution $x(t)$.





a. La valeur numérique du champ magnétique est énormément amplifiée pour les besoins de cet exercice, sans ça le phénomène est difficile à voir sur un graphique.

3 Sujets d'oraux de l'épreuve Centrale 2

Exercice 8
On considère l'équation différentielle $(1-x)^3 y'' = y$. *f est la solution sur* $]-\infty;1[$.
En utilisant la méthode d'Euler, tracer une approximation du graphe de f sur [0 ; 0,9].

Exercice 9
On pose, pour tout $t \neq 0$,

$$A(t) = \begin{pmatrix} 1-\frac{1}{t} & 1 \\ \frac{1}{2t} & 0 \end{pmatrix} \text{ et } (S) \begin{cases} x'(t) = \left(1-\frac{1}{t}\right) x(t) + y(t) \\ y'(t) = \frac{x(t)}{2t} \end{cases}$$

1. *Résoudre numériquement pour plusieurs valeurs de* $x(1) = a$ *et* $y(1) = b$ *, les tracer pour* $t \in [1,4]$. *Commenter.*
2. **Pour les cubes** *: Pour* $t \in \{0,5;1;1,5;2\}$, *trouver les valeurs propres et des vecteurs propres de* $A(t)$. *Que peut-on conjecturer ?*

Exercice 10
1. *On considère l'équation* $(E) : (1-x)y'' = y$ *sur* $]-\infty,1[$.
   (a) *Montrer qu'il existe une unique solution de* (E) *sur* $]-\infty,1[$ *vérifiant* $f(0) = 0$ *et* $f'(0) = 1$.
   (b) *Représenter graphiquement f sur* $[-2;0,95]$.
2. *Soit* $(a_n) \in \mathbb{R}^{\mathbb{N}}$, *définie par :* $a_0 = 0, a_1 = 1$
   et, pour tout $n \in \mathbb{N} - \{0,1\}$, $a_n = \dfrac{n-2}{n} a_{n-1} + \dfrac{1}{n(n-1)} a_{n-2}$.
   (a) *Représenter graphiquement* a_n *pour* $n \in [\![0,100]\!]$.
   (b) *Montrer que le rayon de convergence de* $\sum a_n x^n$ *est supérieur ou égal à 1.*
   (c) *Représenter graphiquement* $s : x \mapsto \sum_{k=0}^{100} a_k x^k$ *sur* $[-1,1;0,95]$. *Que constate-t-on ?*
   (d) *Démontrer le résultat.*

3.1 Un sujet de physique

Ce sujet discute d'un modèle d'interaction forte, une force attractive s'exerçant seulement à courte distance entre nucléons (protons et neutrons). On étudie en particulier l'interaction d'un neutron projectile de masse m_n dont la trajectoire rencontre celle d'un noyau fixe, sphérique de rayon r_0. La trajectoire initiale du neutron est rectiligne uniforme de vitesse v_0 et de paramètre d'impact y_0. Les données numériques dutilisent pour unités de longueur, d'énergie et de masse respectivement le femtomètre ($1fm = 1,0 \times 10^{-15} m$), le méga-électron volt ($e = 1,60 \times 10^{-19} C$) et la masse d'un nucléon ($m_n = 1,67 \times 10^{-27} kg$)

Exercice 11
Quelles sont les unités de durée et de vitesse correspondante ?

La force exercée par le noyau sur le neutron est centrale, attractive, d'expression (en coordonnées sphériques) $\vec{F} = F(r)\vec{e}_r$ avec $F(r) = -\dfrac{U_0}{d} \exp\left(-\dfrac{(r-r_0)^2}{d^2}\right)$. On donne

```
U0 = 10.0
r0 = 1.0
d = .1
```

Exercice 12
Tracer et interpréter les courbes donnant $F(r)$ et l'énergie potentielle $U(r)$ associée.

4 Solutions

Solution de l'exercice 1 -

```
def proiesp(u,t):
  x, y = u
  dx = x - x*y
  dy = x*y - y
  return np.array([dx, dy])

T = np.linspace(0, 10, 1000)
for i in range(6):
  y0 = np.array([1.3 + 0.5*i,
     1.0])
  U = odeint(proiesp,y0,T)
  X = [u[0] for u in U]
  Y = [u[1] for u in U]
  plt.plot(X, Y)
plt.show()
```

Solution de l'exercice 2 -

```
R0 = 3
gamma = 0.03
beta = R0*gamma

def phi(u, t):
    s, i, r = u
    return np.array([-beta*s*i, beta*s*i - gamma*i, gamma*i])

prop = 1e-3
CI = np.array([1-prop, prop, 0])
t_max = 300
T = np.linspace(0, t_max, 10000)
U = odeint(psi3, CI, T)
S = [u[0] for u in U]
I = [u[1] for u in U]
R = [u[2] for u in U]
plt.plot(T, S, label = "S")
plt.plot(T, I, label = "I")
plt.plot(T, R, label = "R")
plt.legend()
plt.show()
```

Solution de l'exercice 3 -

```
def phi(u, t):
    if 10 < t < 40:
        beta1 = beta/2
    else:
        beta1 = beta
    s, i, r = u
    return np.array([-beta1*s*i, beta1*s*i - gamma*i, gamma*i
       ])

prop = 1e-3
CI = np.array([1-prop, prop, 0])

t_max = 100
T = np.linspace(0, t_max, 10000)
U = odeint(psi3, CI, T)
plt.plot(T,U)
plt.show()
```

Solution de l'exercice 4 -

```
def lorenz(u, t):
    sigma = 10
    rho = 28
    beta = 8/3
    x, y, z = u
    vx = sigma*(y - x)
    vy = rho*x - y - x*z
    vz = x*y - beta*z
    return np.array([vx, vy, vz
       ])

T = np.linspace(0, 30, 3000)
y0 = np.array([0.1, 0.0, 0.1])
U = odeint(lorenz, y0, T)
X = [u[0] for u in U]
Y = [u[1] for u in U]
Z = [u[2] for u in U]
dessin = Axes3D(plt.figure())
dessin.plot(X, Y, Z)
plt.show()
```

Solution de l'exercice 5 - La solution devrait être $y(t) = e^{-3t}$ mais les erreurs d'arrondi et d'imprécision font que la solution calculée comporte un terme en e^t.

Solution de l'exercice 6 -

```
def vdPol(u,t) :
  (x,y) = u
  xPrime = y
  yPrime = y*(1-x*x)-x
  return np.array([xPrime, yPrime])

T = np.linspace(0, 30, 1000)
for i in range(10):
  y0 = np.array([i/2, i/3])
  sol = heun(vdPol, y0, T)
  X = [u[0] for u in sol]
  X = [u[0] for u in sol]
  plt.plot(T, X)
plt.show()
```

Toutes les solutions se rapprochent d'un cycle limite.

Solution de l'exercice 7 -

```
def zeeman(u, t):
    x, y, vx, vy = u
    ax = -2*W*vy-w0**2*x
    ay = 2*W*vx-w0**2*y
    return np.array([vx, vy, ax, ay])

t_max = 5e-14
T = np.linspace(0, t_max, 3000)

a = 5.3e-11
theta0 = np.pi/2
r0x = 0
r0y = -a*np.cos(theta0)
v0x = a*w0
v0y = 0
CI = np.array([r0x, r0y, v0x, v0y])


U = odeint(zeeman, CI, T)
X = [u[0] for u in U]
Y = [u[1] for u in U]
VX = [u[2] for u in U]
VY = [u[3] for u in U]

plt.subplot(2, 1, 1)
plt.plot(T, X, label = "$x(t)$")
plt.plot(T, Y, label = "$y(t)$")
plt.title("Tracés de $x(t)$ et de $y(
   t)$")
plt.legend()

plt.subplot(2, 1, 2)
plt.plot(X, VX)
```

```
plt.title("Portrait de phase de $x(t)
    $")
plt.show()
```

Solution de l'exercice 8 -

Solution de l'exercice 9 -

Solution de l'exercice 11 -

Solution de l'exercice 12 -

TP IV

SQL : RÉVISIONS

1 Rappels

1.1 Structure

Les bases de données relationnelles utilisent des données regroupées en **tables** (ou **relations**).

- Chaque ligne d'une relation représente une **entité** (personne, objet, action, relation, ...).
- Une entité est un n-uplet (tuple) de valeurs.
- L'ensemble des entités est l'**extension** de la relation.
- Chaque élément d'un n-uplet est un **attribut** de l'entité.
- L'ensemble des attributs (les colonnes) est le **schéma** de la relation.
- Chaque attribut prend des valeurs dans un ensemble défini : son **domaine**.
- Une **clé primaire** est un ensemble d'attributs qui identifie chaque entité de la relation de manière unique et minimale.
- Souvent, la clé primaire sera un attribut dédié, un identifiant.

1.2 Requêtes SQL

On interroge les bases de données à l'aide d'un langage standardisé : SQL.
Une requête produit un résultat qui est lui même une table (ou une valeur unique), il sera donc possible de composer ou combiner les requêtes.
L'ordre des instructions dans une requête est fixé :

```
SELECT ...
FROM ...
(WHERE ...)
(GROUP BY ...)
(HAVING ...)
(ORDER BY ...)
(LIMIT ...)
(OFFSET ...)
```

Seules les deux premières instructions sont obligatoires. Les instructions à partir de `from` sont exécutées dans l'ordre, `select` chapeaute la requête.

Select est suivi de l'énumération des valeurs à afficher séparées par des virgules.
- Ce peut être des attributs de la tables, * si on les veut tous.
- Ce peut être des valeurs calculées à partir des attributs, il sera alors utile de donner un nom (un alias) au résultat : *calcul* `as` *nom*.
- Ce peut être le résultat d'un calcul statistique sur les lignes sélectionnées, regroupées ou non (voir **Group by**). On peut utiliser la somme des valeurs d'un attribut `sum(`a`)`, la moyenne `avg(`a`)`, le maximum `max(`a`)`, le minimum `min(`a`)`, le nombre d'entités sélectionnées `count()`.
- Pour éviter les répétitions on peut utiliser `select distinct`.

From est suivi de la table utilisée. Ce peut être
- une table simple de la relation,
- un produit de tables, on les énumère avec des virgules pour séparer,
- une **jointure** de tables.

Where est suivi des conditions que doivent vérifier les attributs (ou les valeurs calculées en utilisant leur alias). On peut les combiner avec `and`, `or` et `not`.

Group by est suivi d'un attribut : il partitionne la table selon les valeurs de cet attribut et permet un calcul statistique par paquet. Quand cette instruction est utilisée on ne peut utiliser dans `select` que l'attribut du regroupement [1] et des valeurs statistiques.

Having n'est utile qu'avec une instruction `Group by`. Il permet de donner des conditions portant sur les valeurs statistiques calculées.

Order by suivi d'un nom d'attribut ou d'alias permet de trier par ordre croissant selon les valeurs de cet attribut. On peut faire suivre par `desc` si on veut un ordre décroissant.

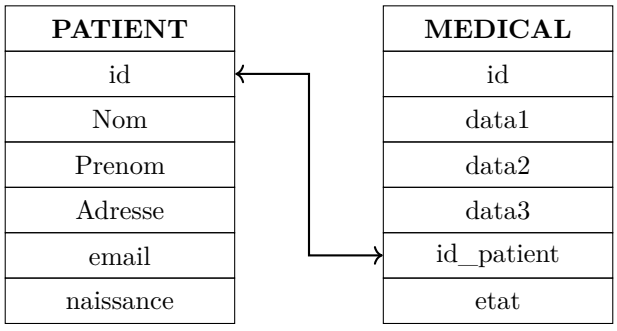
Limit n'est utile qu'avec une instruction `Order by`. Il permet de limiter le nombre de résultats affichés

Offset n'est utile qu'avec une instruction `Limit`. Il permet de décaler les instructions. `Limit 3 Offset 4` affiche les résultats des rangs 4 à 6.

1.3 Jointures

Remarque : quand on utilise plusieurs tables, on accède aux attributs en donnant aussi la table d'où ils proviennent : *table.nom*.
Il sera souvent utile de donner un alias court aux tables : `From` *table* `as` *t*.

Quand on utilise plusieurs table, il y a souvent des liens entre celles-ci : un attribut d'une table réfère à un attribut d'une autre table. Cela sera parfois indiqué dans une représentation des schémas.



Il est recommandé d'utiliser cette relation en améliorant le produit par une jointure. Dans l'exemple cela donnerait

```
FROM patient as p Join medical as m On p.id = m.id_patient
```

Le nom d'un patient sera alors connu par `m.nom`

1. On peut aussi utiliser des attributs qui ont la même valeur par paquet.

1.4 Assemblages de requêtes

On peut combiner des requêtes de plusieurs manières.

1. Quand deux requêtes produisent des tables avec le même schéma, on peut en faire l'union : `union`, l'intersection : `intersection` ou la différence : `except`. Le mot clé est simplement placé entre les deux requêtes.
2. Une requête peut produire une valeur unique (avec une fonction statistique), cette valeur peut être utilisée dans une condition (`where` ou `having`). On ne peut malheureusement pas sauvegarder la valeur dans une variable, il faut l'utiliser en écrivant la requête entre parenthèses.
3. En général une requête produit une nouvelle table. On peut alors effectuer une requête sur cette table. Ici encore on écrit la requête entre parenthèses dans l'instruction `From`.

2 Présentation

Nous allons utiliser une base de données qui contient de nombreuses tables.
Certaines sont des tables qui contiennent des descriptions, par exemple ***Pays****.*
D'autres représentent des associations entre des données, par exemple ***Frontiere****.*

Voici les tables tables utilisées et leurs attributs. La clé primaire est soulignée.

- **Appartenance** *: Pays, Continent, Pourcentage*
- **Continent** *: Nom, Superficie*
- **Frontiere** *: Pays1, Pays2, Longueur*
- **Ile** *: Nom, Iles, Superficie, Altitude, Type, Longitude, Latitude*
- **IleDans** *: Ile, Mer, Lac, Riviere*
- **IlePays** *: Ile, Pays, Province*
- **Langage** *: Pays, Nom, Percentage*
- **MembreOrganisation** *: Organisation, Pays, Type*
- **Montagne** *: Nom, Chaine, Altitude, Type, Longitude, Latitude*
- **MontagnePays** *: Montagne, Pays, Province.*
- **Organisation** *: Abbreviation, Nom, Ville, ...*
- **Pays** *: Nom, Code, Capitale, Province, Superficie, Population*
- **Politique** *: Pays, DateIndependance, AncienneDependance, Dependance, ...*
- **Riviere** *: Nom, Riviere, Lac, Mer, ...*
- **RivierePays** *: Riviere, Pays, Province*

Dans les tables d'associations de nom du pays est le code de la table **Pays***.*
Dans chaque question on a donné le nom de la table ou des tables utilisées.
La table n'est pas complète et comporte des erreurs.

3 Requêtes sur une seule table

Exercice 1
Déterminer la liste des continents et de leur superficie, par ordre croissant.

Continent

Exercice 2
Quels sont les pays de plus de 10^8 habitants ? Il y en a 10.

Pays

Exercice 3
Quels sont les volcans (`type = "volcano"`) de plus de 6000 m ? Il y en a 7.

Montagne

Une rivière se jette dans une autre rivière (elle est un affluent), dans un lac ou dans une mer. C'est ce qu'indiquent les attributs Rivière, Lac ou Mer. En théorie un seul de ces attributs n'est pas vide mais une rivière peut aussi passer dans un lac avant d'atteindre une autre rivière ou une mer.

Exercice 4
Quels sont les affluents du Rhin (**Rhein**) ? Il y en a 4.

Riviere

4 Fonctions d'agrégation

Exercice 5
Quelle est la longueur de la frontière (terrestre) de la France (code "F") ? (2892.4 km)

Frontiere

Exercice 6
Quelles sont les îles faisant partie des Caraïbes (`iles = "Caraibes"`) avec leur superficie ?

Ile

Exercice 7
Combien y a-t-il d'îles dans les Caraïbes ? `count` (22)
Quelle est la superficie totale ? (12165 km^2)
Quelle est la moyenne des points culminants ? (825 m)

Ile

Exercice 8
Quelles sont les mers dans lesquelles se jettent plus 5 rivières ou plus (`having`) ? Il y en a 5.
On excluera les rivières qui ne sont pas des fleuves par la condition : `mer is not null`.

Riviere

Exercice 9
Quelles sont les villes qui sont le siège de plus de 5 organisations ? Il y en a 6.

Organisation

Exercice 10
Quelles sont les langues parlées dans 5 pays au moins par plus de 25 % de la population ?
Il y en a 4.

Langage

5 Jointures

Exercice 11
Quels sont les pays membres de l'UNESCO ? Il y en a 192 (188 vrais membres).

MembreOrganisation, **Pays**

Exercice 12
Quels sont les pays qui ont été des colonies de la France (`AncienneDependance = "F"`) ? (24)

Politique, **Pays**

Exercice 13
Quels sont les pays qui dans lesquels l'anglais est la langue pour plus de 10 % de la population ?

Langage, **Pays**

Exercice 14
Quels sont les pays qui sont traversés par le Danube (**Donau**) ? Il y en a 10.

RivierePays, **Pays**

Exercice 15
Quelles sont les montagnes d'Amérique de plus de 6000 m avec le nom du pays dans lequel elles sont situées et leur altitude ? Il y en a 16.

MontagnePays, **Appartenance**, **Pays**, **Montagne**

Exercice 16
Quels sont les affluents des affluents du Rhin ? Il y en a 3.

Riviere 2 fois

Exercice 17
Quels sont les pays dont la plus grande partie est en Europe ? Il y en a 51.

Pays, **Appartenance**

Exercice 18
Quels sont les organisations qui rassemblent des pays avec un total de plus de 5.10^9 habitants ? Il y en a 27.

Organisation, **Pays**, **MembreOrganisation**

Exercice 19
Dans quels pays existe-t-il une île dans un Lac ? Il y en a 4 (2 îles au Canada).

IleDans, **IlePays**, **Pays**

Exercice 20
Quels sont les pays qui ont une montagne dans la chaîne des Alpes (**"Alps"**) ? Il y en a 5.

Pays, **MontagnePays**, **Montagne**

Exercice 21
Quels sont les fleuves qui passent en France ?

Riviere, **RivierePays**, **Pays**

6 Sous-requêtes et combinaisons

Exercice 22
Quels sont les pays qui font partie de l'ONU (**UN**) mais pas de l'UNESCO ? Il y en a 6.

Pays, **MembreOrganisation**

Exercice 23
Quels sont les pays qui ont une frontière avec la Russie (**Russia**, code **R**) ? Il y en a 14.

Frontiere, **Pays**

Exercice 24
Combien ont d'affluents les fleuves qui passent en France ? (Seuls 3 ont des affluents dans la base)

Riviere deux fois, **RivierePays**, **Pays**
La réponse n'est pas que la Seine a quatre affluents dans la base.

Exercice 25
Quels sont les langues parlées par plus de 100.000.000 locuteurs ? (7)

Pays, **Langage**

Exercice 26
Quels sont les pays qui partagent une même montagne ? Il y en a 69.

Pays, **MontagnePays**, **Montagne**

Exercice 27
Quels sont les pays d'Europe qui ont une montagne dans la base ? On donnera aussi le nom et l'altitude de la plus haute montagne. Il y en a 20.

Pays, **MontagnePays**, **Montagne**, **Appartenance**

7 Complément : championnat de France

Nous allons travailler avec une base de données qui recense les renseignements sur les matchs de division 1 d'une année. La base ne contient qu'une table, nommée `statFoot`.

En voici quelques lignes, la saison est 2018-2019

jour	*equDom*	*equExt*	*butsDom*	*butsExt*
10/08/2018	*Marseille*	*Toulouse*	*4*	*0*
11/08/2018	*Angers*	*Nimes*	*3*	*4*
11/08/2018	*Lille*	*Rennes*	*3*	*1*
11/08/2018	*Montpellier*	*Dijon*	*1*	*2*
11/08/2018	*Nantes*	*Monaco*	*1*	*3*

Il y a 20 équipes ; chaque équipe reçoit toutes les autres une fois et une seule.

- `jour` *est la date du match, sous forme d'une chaîne de caractères.*
- *Les autres attributs vont par paire : le suffixe* `Dom` *indique que l'attribut concerne l'équipe qui joue à domicile, le suffixe* `Ext` *indique que l'attribut concerne l'équipe qui joue à l'extérieur.*
- `equDom` *ou* `equExt` *indique le nom de l'équipe, sous forme d'une chaîne de caractères.*
- `butsDom` *ou* `butsExt` *indique le nombre de buts marqués pendant le match.*

Un match nul vaut 1 point, un match gagné vaut 3 points.

Exercice 28
Écrire une requête qui permet de connaître le nombre de matchs gagnés par chaque équipe, le nombre de matchs nuls et le total des points de l'année.
N.B. Bien que la base ne contienne qu'une table, la requête peut faire intervenir des jointures.

8 Solutions

Solution de l'exercice 1 -

```
select *
from continent
order by Superficie
```

Solution de l'exercice 2 -

```
select nom, population
from Pays
where population > 100000000
order by 2 desc
```

Solution de l'exercice 3 -

```
select nom, altitude
from Montagne
where type = 'volcano'  AND altitude > 6000
```

Solution de l'exercice 4 -

```
select nom
from Riviere
where riviere = "Rhein"
```

Solution de l'exercice 5 -

```
select sum(longueur)
from Frontiere
where Pays1 = "F" or Pays2 = "F"
```

Solution de l'exercice 6 -

```
select nom,superficie
from Ile
where Iles = "Caraibes"
order by 1
```

Solution de l'exercice 7 -

```
select count() as nombre, sum(Superficie) as surface, avg(
   Altitude) as altitude
from ile
where iles = "Caraibes"
```

Solution de l'exercice 8 -

```
select count() as nombre, mer
from Riviere
where mer is not null
group by mer
having nombre >4
```

Solution de l'exercice 9 -

```
select Ville, count() as nbSieges
from Organisation
where Ville is not Null
group by Ville
having nbSieges >= 5
```

Solution de l'exercice 10 -

```
select nom, count() as nb
from langage
where percentage >25
group by nom
having nb >4
```

Solution de l'exercice 11 -

```
select p.nom
from Pays as p join MembreOrganisation as mo on p.Code = mo.
   Pays
where mo.Organisation = "UNESCO"
```

Solution de l'exercice 12 -

```
select p.nom
from Pays as p join Politique as pol on p.code = pol.Pays
where pol.AncienneDependance = "F"
```

Solution de l'exercice 13 -

```
select p.nom
from langage as l join pays as p on p.code = l.pays
where l.percentage > 10 and l.nom = "English"
```

Solution de l'exercice 14 -

```
select distinct p.nom
from Pays as p join RivierePays as r on p.code = r.Pays
where r.Riviere = "Donau"
```

Solution de l'exercice 15 -

```
select distinct m.Nom, p.Nom, m.Altitude
from Appartenance as a join MontagnePays as mp on a.Pays = mp.
   Pays
                       join Montagne as m on m.Nom = mp.
                          Montagne
                       join Pays as p on p.Code = a.Pays
where m.Altitude > 6000 and a.Continent = "America"
```

Solution de l'exercice 16 -

```
select r.nom
from Riviere as r join (select nom, riviere from Riviere) as
   aff
                  on r.riviere = aff.nom
where aff.riviere = "Rhein"
```

On peut aussi composer les requêtes avec **in**.

```
select nom
from Riviere
where riviere in (select nom
                  from Riviere
                  where riviere = "Rhein")
```

Solution de l'exercice 17 -

```
select p.Nom
from Pays as p join Appartenance as a on a.Pays = p.code
where a.Continent = "Europe" and a.Pourcentage > 50
```

Solution de l'exercice 18 -

```
select o.Nom, sum(p.Population) as Pop
from Organisation as o join MembreOrganisation
                            as mo on o.Abbreviation=mo.
                               Organisation
                       join Pays as p on p.Code = mo.Pays
group by mo.Organisation
having pop > 5000000000
```

Solution de l'exercice 19 -

```
select p.Nom, i.Ile as Ile, i.Lac
from Pays as p Join IlePays as ip on p.code=ip.Pays
               Join IleDans as i on ip.Ile=i.Ile
where i.Lac is not Null
```

Solution de l'exercice 20 -

```
select distinct p.Nom
from Pays as p join MontagnePays as mp on p.Code = mp.Pays
               join Montagne as m on m.Nom = mp.Montagne
where m.Chaine = "Alps"
```

Solution de l'exercice 21 -

```
select distinct r.nom
from Riviere as r join RivierePays as rp on r.nom = rp.riviere
                                 join Pays as p on rp.pays = p.
                                    code
where p.nom ="France" and r.mer is not null
```

Solution de l'exercice 22 -

```
select p.Nom
from Pays as p join MembreOrganisation as mo on p.Code = mo.
   Pays
where mo.Organisation = "UN" AND type = "member"

except

select p.Nom
from Pays as p join MembreOrganisation as mo on p.Code = mo.
   Pays
where mo.Organisation = "UNESCO" AND type = "member"
```

Solution de l'exercice 23 -

```
select p.Nom
from Pays as p join Frontiere as f on p.Code = f.Pays1
where f.Pays2 = "R"

union

select p.Nom
from Pays as p join Frontiere as f on p.Code = f.Pays2
where f.Pays1 = "R"
```

Solution de l'exercice 24 - La base `RivierePays` indique les rivière plusieurs fois, pour chaque région parcourue. Il faut donc ne prendre que les fleuves distincts.

```
select f.fleuve, count()
from (select distinct r.nom as fleuve
          from Riviere as r join RivierePays as rp
                                  on r.nom = rp.riviere
                              join Pays as p
                                  on rp.pays = p.code
          where p.nom ="France" and r.mer is not null) as f
          join Riviere as r1 on f.fleuve = r1. riviere
group by f.fleuve
```

Solution de l'exercice 25 -

```
select langue, sum(locuteurs) as total
from (
  select p.nom, l.nom as langue, p.population*l.percentage/100
      as locuteurs
  from langage as l join pays as p on p.code = l.pays)
  group by langue
  having total > 100000000
  order by total desc
```

Solution de l'exercice 26 -

```
select distinct mm.Nom as NomMontagne, p.Nom as NomPays
from (select Montagne as Nom, count() as nb
      from (select distinct Montagne,Pays
            from MontagnePays)
            group by Nom) as mm
           join MontagnePays as mp on  mp.Montagne = mm.Nom
           join Pays as p on p.Code = mp.Pays
where mm.nb > 1
order by 1
```

Solution de l'exercice 27 -

```
select p.Nom, mp.Montagne, max(m.Altitude)
from Pays as p join Appartenance as a on a.Pays = p.code
               join MontagnePays as mp on p.Code = mp.Pays
               join Montagne as m on m.Nom = mp.Montagne
where a.Continent = "Europe" and a.Pourcentage > 50
group by mp.Pays
```

Solution de l'exercice 28 -

```
select d.club, (d.victoiresDom + e.victoiresExt) as victoires,
       n.nuls,
       (38 - n. nuls - d.victoiresDom - e.victoiresExt) as Dé
          faites,
       (3*(d.victoiresDom + e.victoiresExt) + n.nuls) as
          points
from (select equDom as club, count() as victoiresDom
      from stat
      where butsDom > butsExt group by equDom) as d
     join
     (select equExt as club, count() as victoiresExt
      from stat
      where butsDom < butsExt group by equExt) as e
     join
     (select equDom as club, count() as nuls
      from stat
      where butsDom = butsExt group by equDom) as n
     on e.club = d.club and e.club = n.club
order by points desc
```

TP V

Images

1 Présentation

1.1 Introduction

Une image est codée dans un ordinateur sous la forme de petits carrés appelés `pixels`[1]. Ces éléments sont rassemblés dans une matrice dont les lignes et les colonnes correspondent aux positions dans l'image.
Pour commencer de façon basique, on peut coder avec des 0 (pas de lumière) et des 1 (de la lumière). Voici une première image simple :

```
image1 = np.array([[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
                   [1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1],
                   [1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1],
                   [1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1],
                   [1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1],
                   [1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1],
                   [1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1],
                   [1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1],
                   [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]])
```

On remarquera qu'on a utilisé une matrice, c'est-à-dire une liste de listes, convertie en matrice `numpy` avec `np.array` : les calculs seront plus rapides pour les grandes tailles et il est plus facile de sélectionner des morceaux.
On devra donc auparavant charger les bibliothèques

```
import numpy as np
import matplotlib.pyplot as plt
```

Pour voir ce que représente cette image on peut colorier les 0 en noir.
On peut aussi la faire afficher :

```
plt.imshow(image1)
plt.show()
```

Ce n'est pas vraiment ce que l'on attendait car Python choisit une échelle de couleurs par défaut qui n'est pas une échelle de gris.
On peut voir cette échelle en tapant `plt.colorbar()`.
Si on veut simplement une échelle de gris, on peut le spécifier par la commande `plt.gray()`.
On peut aussi changer l'échelle des couleurs avec le paramètre `cmap` (pour Color MAP)

1. **pic**ture **el**ements

`plt.imshow(image1, cmap='gray')`
On peut voir les différentes échelles de couleur à l'adresse

`http://matplotlib.org/examples/color/colormaps_reference.html`

1.2 Niveaux

On a vu qu'il y avait une échelle, on peut choisir des valeurs intermédiaires.
L'échelle s'adapte : l'intervalle des valeurs sera toujours $[a; b]$ où a est le minimum des valeurs de la matrice et b le maximum.

```
image2 = np.zeros((100,100))
for i in range(100):
    for j in range(100):
        x = (i-50)/20
        y = (j-50)/20
        image2[i,j] = np.sin(x*x+y*y)

plt.imshow(image2)
plt.show()
```

Si on affiche l'échelle des couleurs on voit que les valeurs utilisées vont de -1 à 1.
L'image présente un effet de carrelage, on voit les pixels. On peut estomper les différences en ajoutant une interpolation :

```
plt.imshow(image2, interpolation = 'bilinear')
```

1.3 Photos monochromes

Ce qui précède utilise la représentation des matrices pour illustrer des données : la valeur atteinte par une fonction de deux variables (les coordonnées) est visualisée par sa couleur dans une échelle. On peut aussi penser ces valeurs comme la luminosité d'un point et considérer une matrice comme une image, par exemple une photo en noir-et-blanc.
Dans `scipy.misc` on peut trouverun exemple de photo.

```
from scipy.misc import ascent

image4 = ascent()
plt.gray()
plt.imshow(image4)
plt.show()
```

`matplotlib` permet de lire des images en format `png` et les transforme en matrice `numpy`.
Le dossier ßpublic devrait contenir une photo du Taj-Mahal.

```
imgage5 = plt.imread('chemin/vers/le/fichier/taj-NB.png')
plt.gray()
plt.imshow(imgage5)
plt.show()
```

Nous allons dans ce TP modifier une image.
Comme il est souhaitable de garder inchangée l'image originale on devra

- soit copier l'image, `img = np.copy(image)`
- soit créer une image de zéros de même taille

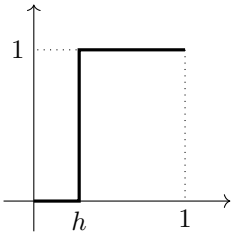
  ```
  p,q = np.shape(image)
  img = np.zeros((p,q))
  ```

  Il existe une astuce pour les matrices `numpy` pour ce dernier cas : `img = 0*image`.

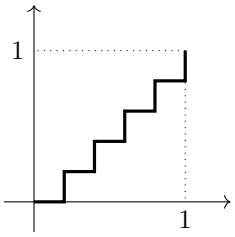
2 Modification de l'échelle

La première action possible sur une image est de modifier globalement la valeur des pixels.
Nous avons vu que `matplotlib` s'adaptait à l'intervalle des valeurs : il détermine les minimum et maximum et distribue la coloration entre ces extrêmes.
Cependant, par cohérence avec les images en couleur, nous allons nous restreindre à une échelle de valeur entre 0 et 1.
On définit donc `image1 = ascent()/255` car l'échelle initiale va de 0 à 255.
Le principe général sera de déterminer une fonction de $[0;1]$ dans lui même que l'on appliquera à tous les pixels.

Exercice 1 :



Exercice 2 :



Exercice 1 — Image en noir et blanc
Écrire une fonction `seuil(x,h)` *qui reçoit deux réels x et h et qui renvoie 0 dans la cas $x < h$ et 1 sinon. En déduire une fonction* `seuil_image(image,h)` *qui reçoit une image sous la forme d'une matrice* `numpy` *et un réel $h \in [0;1]$ et qui renvoie une nouvelle image dont les éléments sont les images par* `seuil(x,h)` *pour tous les pixels x de l'image initiale.*

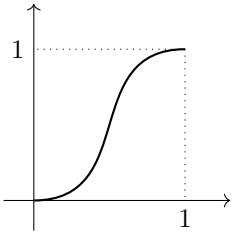
On peut généraliser en calculant plusieurs valeurs de seuil : on obtient un effet de sérigraphie.
La fonction $x \mapsto \frac{1}{n}\lfloor nx \rfloor$ ne donnera que les valeurs 0, $\frac{1}{n}$, $\frac{2}{n}$, …, $\frac{n-1}{n}$, 1

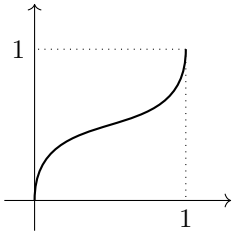
Exercice 2 — Image posterisée
Écrire une fonction `poster(image,k)` *qui reçoit une image et un entier k et qui renvoie une nouvelle image dont les éléments ne prennent que les valeurs de la forme $\frac{j}{k}$.*
On pourra utiliser `cmap = "gist_rainbow"`.

On peut aussi modifier les valeurs en les orientant vers le blanc ou le noir. Il suffit de trouver une fonction de $[0;1]$ dans lui même qui vérifie $f(x) \geqslant x$ pour donner plus de luminosité ou $f(x) \leqslant x$ pour assombrir.
Les fonctions $x \mapsto x^a$ sont des bons candidats : $a > 1$ assombrit, $a < 1$ éclaircit.

Exercice 3 — Luminosité
Écrire une fonction `luminosite(image,a)` *qui reçoit une image et un réel $a > 0$ et qui renvoie une nouvelle image transformée par $x \mapsto x^a$.*

Plutôt que déplacer la luminosité on peut modifier la différentiation entre le noir et le blanc : le contraste. Augmenter le contraste consiste à diminuer les petites valeurs et augmenter les grandes.
Les fonctions que l'on recherche auront un graphe qui ressemble aux graphes suivant





Ces fonctions peuvent être obtenue à l'aide de `np.sin` ou `np.arcsin`.
Par exemple la première fonction est $y = \frac{1}{2}\left(1 + \sin\left(\pi\left(x - \frac{1}{2}\right)\right)\right)$.

Exercice 4 — Contraste
Écrire des fonctions `contrastePlus(image)` *et* `contrasteMoins(image)` *qui reçoivent une image et qui renvoient une nouvelle image dont le contraste a été augmenté ou diminué.*

3 Modifications locales

Dans cette partie nous allons modifier les images de manière locale : un point sera modifié en fonction de son environnement.
L'outil utilisé est la **convolution** : on veut modifier une image représentée par une matrice M de taille $n \times m$. On se donne une matrice c de taille $(2p+1) \times (2p+1)$ et on remplace chaque pixel $M_{i,j}$ par

$$M'_{i,j} = \sum_{k=0}^{2p} \sum_{l=0}^{2p} M_{i-p+k,j-p+l} c_{k,l}$$

pour $p \leqslant i < n-p$ et $p \leqslant j < m-p$. On a utilisé des indices commençant par 0.
On pourra laisser les p lignes ou colonnes des bords avec une valeur nulle.

Exercice 5 — Convolution
Écrire une fonctions `convoler(image,noyau)` *qui reçoit une image et une matrice de coefficients et qui renvoie une nouvelle image dont les coefficients sont calculés par la formule ci-dessus.*

Exercice 6 — Floutage
Pour calculer une image floutée on peut déterminer la convolution avec une matrice $5x5$ *dont les coefficients sont* $\frac{1}{25}$.

Exercice 7 — Netteté
On peut, inversement, augmenter le "piqué" de la photo en augmentant le contraste local.
On pourra utiliser la matrice $\begin{pmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{pmatrix}$.
Cependant il y a des petits défauts dans la photo qui donnent de grandes valeurs indésirables au résultat. On pourra tronquer les valeurs en remplaçant les négatifs par 0 et les valeurs supérieures à 1 par 1. Une fonction `numpy` *fait cela :* `np.clip(mat,a,b)` *renvoie un tableau* `numpy` *dans lequel les valeurs comprises entre a et b sont conservées et les valeurs inférieures à a ou supérieures à b sont ramenées à a ou b respectivement.*

Exercice 8 — Contour 1
La matrice $\begin{pmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{pmatrix}$ *calcule la valeur ci-dessus moins la valeur initiale ; elle permet donc de déterminer les fortes variations, c'est-à-dire le contour, en niveaux de gris.*
Comme ci-dessus il vaut mieux tronquer entre 0 et 1.
Nous sommes plus habitués à un contour noir sur fond blanc : on pourra inverser le résultat.

Exercice 9 — Contour 2
On peut améliorer le calcul du contour en déterminant les variations horizontales et verticales en convolant avec les matrices $\begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}$ *et* $\begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}$. *On obtient les matrices* Δx *et* Δy *et on détermine la matrice des contour en calculant les coefficients* $\sqrt{(\Delta x)^2_{i,j} + (\Delta y)^2_{i,j}}$.
Ici encore on pourra inverser le résultat.

4 Exemples de résultats

Figure V.1 – Image originale

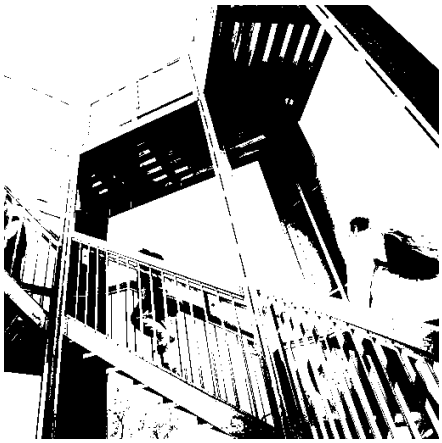


Figure V.2 – Exercice 1

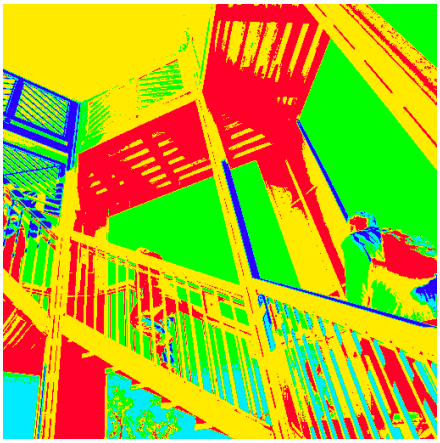


Figure V.3 – Exercice 2, $k = 5$

Figure V.4 – Exercice 3, $a = 0.5$

Figure V.5 – Exercice 4, `contrastePlus`

Figure V.6 – Exercice 4, `contrasteMoins`

Figure V.7 – Image originale

Figure V.8 – Exercice 6

Figure V.9 – Exercice 7

Figure V.10 – Exercice 8

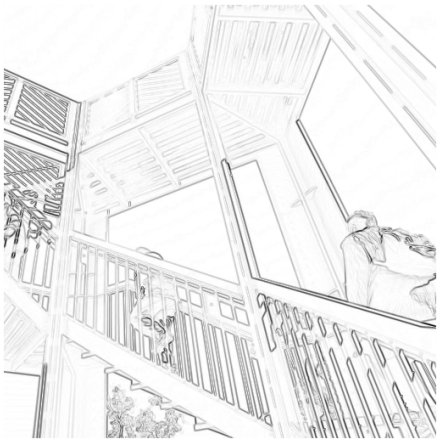


Figure V.11 – Exercice 9

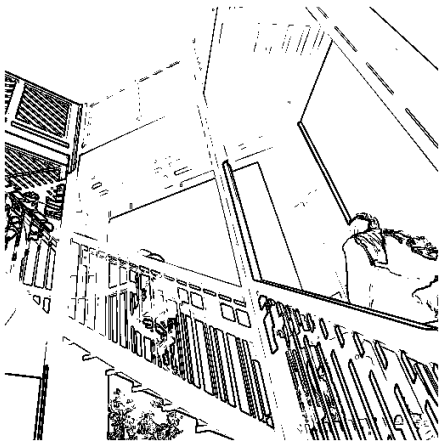


Figure V.12 – Exercice 9 puis seuil

5 Contrôle d'un paramètre à la souris

Dans certaines questions on faisait intervenir un paramètre. Il peut être agréable de le voir agir en direct.
Pour cela on utilisera un "widget" curseur qu'il faut importer.

```
from matplotlib.widgets import Slider
```

Voici les lignes correspondant à l'exercice 3.

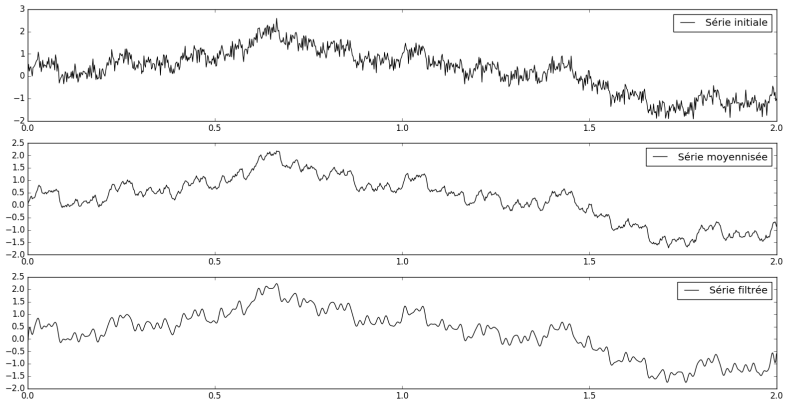
```
1  a0 = 1
2  image4 = luminosite(image1, a0)
3
4  img = plt.imshow(image4, cmap="gray")
5
6  plt.subplots_adjust(top=0.8)
7  place = plt.axes([0.25, 0.92, 0.6, 0.04])
8  curseur = Slider(place, "Valeur de a", 0, 3, valinit = a0)
9
10 def changer(val):
11     a = curseur.val
12     img.set_data(luminosite(image1, a))
13
14 curseur.on_changed(changer)
```

- Les lignes 1 et 2 définissent la valeur de départ du paramètre et l'image correspondante.
- La ligne 4 crée la représentation graphique abstraite, elle sera réalisée avec le `plt.show()`.
- La ligne 6 ajuste la partie correspondant au tracé en lui laissant 80% en hauteur dans le bas (`top = 0.8`). On peut réserver de la place en bas (`bottom =`), à gauche (`left =`) ou à droite (`right =`).
- La ligne 7 crée un rectangle qui contiendra le curseur. Les paramètres sont la position initiale définie en proportion du cadre avec l'origine en bas à gauche puis les dimensions largeur et hauteur, toujours en proportion du cadre. Ici on commence à 1/4 de la largeur et 92% de la hauteur.
- La ligne 8 crée le curseur dans le rectangle `place` défini au-dessus, avec le nom "Valeur de a" qui sera écrit à gauche du rectangle, les valeurs extrêmes du paramètre (ici entre 0 et 3). La valeur initial est optionnelle avec le nom `valint`. La valeur choisie est affichée à droite du rectangle, elle changera en fonction de la place du curseur.
- Les lignes 10 à 12 définissent la fonction de mise-à-jour. Son nom est libre mais son paramètre est le nom de la variable retournée par le curseur. La valeur est calculée par `curseur.val` et on change les données graphiques avec `img.set_data` où `img` et le nom choisi pour recevoir le tracé.
- La ligne 14 permet d'appeler la mise-à-jour chaque fois que le curseur prend une nouvelle valeur.

6 Filtrage

Le floutage de l'exercice 6 a consisté, en fait, à calculer une moyenne locale des valeurs.

Si on illustre cette action sur un tableau uni-dimensionnel on calcule $v_i = \frac{1}{5}\sum_{k=0}^{4} u_{i+2-k}$ pour $2 \leqslant i \leqslant n-3$.



On remarque que cela revient à atténuer les "hautes fréquences".
Il existe un outil qui permet de "décomposer en fréquences" une fonction ; c'est la transformée de Fourier. Ce pourra être un sujet d'étude en mathématiques en seconde année. Il en existe une traduction pour les suites finies : c'est la transformée de Fourier discrète. De plus il existe un algorithme efficace qui permet de calculer cette transformée : la transformation de Fourier rapide (Fast Fourier Transform, FFT, en anglais).
La transformée de Fourier du tableau initial donne des pics qui correspondent aux fréquences lues. Pour filtrer on annule les valeurs supérieures à un seuil puis on revient en arrière (on calcule la transformée de Fourier inverse) ; on obtient le troisième tableau.

La transformation de Fourier existe aussi pour les tableaux à deux dimensions : nous allons l'utiliser pour nos images. Il y a quelques propriétés dont il faut tenir compte.

- La transformée de Fourier et son inverse sont à valeurs dans $\mathbb{C}$.
- Lorsque les valeurs initiales sont réelles la transformée de Fourier présente des symétries (il y a "repliement du spectre"). Les valeurs correspondant aux fréquences basses sont aux quatre coins de la matrice (aux deux bords d'un tableau uni-dimensionnel). On va décaler (**shift** en anglais) les basses fréquences au centre.
- Les fonctions de transformée de Fourier sont dans le module `numpy.fft`.

```
from numpy.fft import *
```

1. `fft` et `ifft` calculent la transformée de Fourier et la transformée de Fourier inverse des tableaux à une dimension.
2. `fft2` et `ifft2` calculent la transformée de Fourier et la transformée de Fourier inverse des matrices.
3. `fftshift` et `ifftshift` décalent et remettent en place l'origine des fréquences.

Pour transformer une image on va donc :

1. la transformer : `im1 = fft2(image)`
2. décaler le centre : `im1 = fftshift(im1)`
3. appliquer une transformation : `im1 = im1*modif`
4. repositionner le centre : `im1 = ifftshift(im1)`
5. appliquer la transformée inverse : `image1 = ifft2(im1)`

On pourra insérer tout cela dans une fonction.

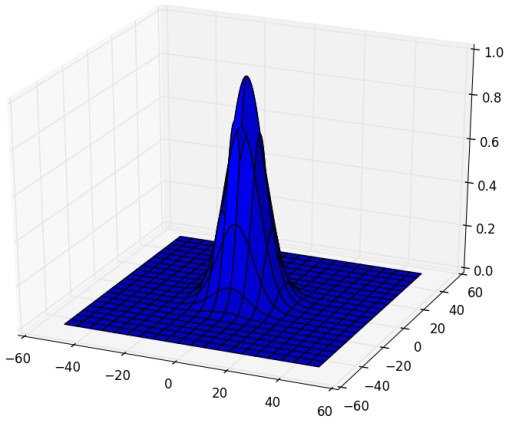
Le résultat est une matrice à coefficients complexes, pour l'afficher on en prend le module avec la fonction `abs`.

On ne va pas filtrer de manière brutale : on va multiplier les coefficients par une matrice dont le terme central vaut 1 et dont les valeurs décroissent vers 0 en s'éloignant du centre.

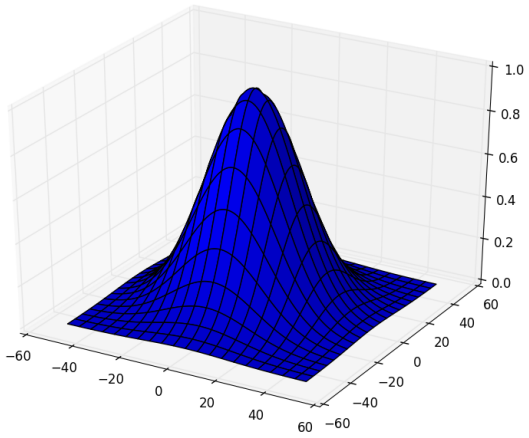
Une fonction usuelle est la gaussienne :

$$g_{x_0,y_0,\sigma}(x,y) = \exp\left(-\frac{(x-x_0)^2+(y-y_0)^2}{\sigma^2}\right)$$

où σ est un paramètre qui indique la largeur utile.



$\sigma = 10$



$\sigma = 25$

Exercice 10 — Gaussienne

Écrire une fonction `gaussienne(a,b,alpha)` *qui reçoit 2 entiers positifs a et b et un réel positif α et qui renvoie une matrice de taille $a \times b$ dont les valeurs sont les $g_{x_0,y_0,\sigma}(i,j)$ pour le terme d'indices i et j avec $x_0 = a//2$, $y_0 = b//2$ et $\sigma = \alpha\frac{a+b}{4}$.*

Exercice 11 — Flou gaussien

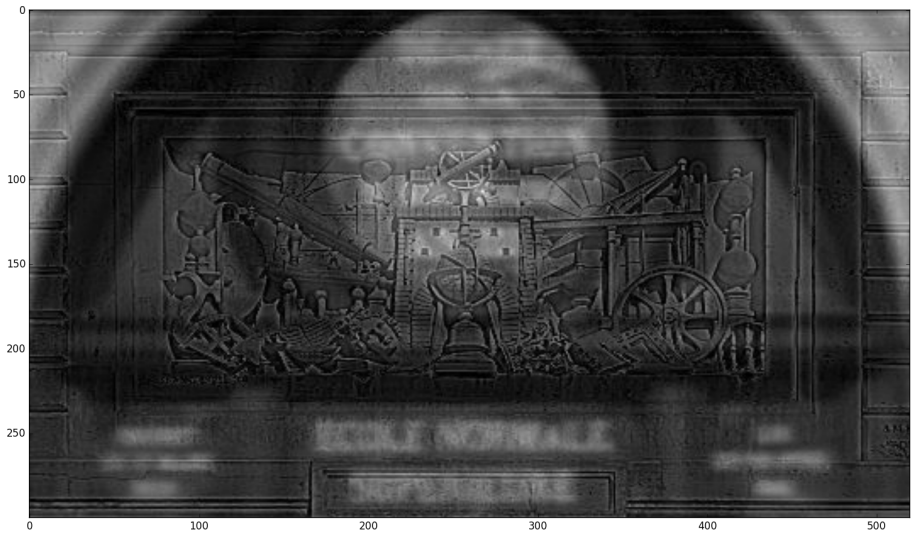
En déduire une fonction de flou `flouGaussien(image,alpha)` *en appliquant l'algorithme décrit ci-dessus qui utilise une matrice calculée dans la question précédente pour la modification.*

7 Images hybrides

Si on a calculé `im1 = flouGaussien(im)` alors l'image `im - im1` représente les parties à "haute fréquence" de `im`, c'est à dire les détails locaux.

IL peut venir alors l'idée de prendre deux photos qui ont une ressemblance et qui ont même taille, de prendre la transformée floue de l'une et de l'additionner à la partie haute-fréquence de l'autre et de faire la somme.

Lorsque les réglages sont adaptés on obtient une image hybride dans laquelle la première image est distinguée de près tandis que la seconde se voit de loin.



Les détails vus de près sont le bas-relief au-dessus de l'entrée de l'ancienne école polytechnique ; la vue de loin est le dessus de l'entrée de l'école normale supérieure de Paris.

Exercice 12 — Projet
Créez vous-même une image hybride.

- On pourra trouver des images depuis internet.
- La conversion en format png monochrome peut se faire en ligne :
  `http://image.online-convert.com/fr/convertir-en-png`
  On choisira Gris comme format.
- On peut découper une partie d'une image par une extraction dans python :
  `image2 = image1[a:b,c:d]`

8 Solutions

Solution de l'exercice 1 -

```
def seuil(h,x):
    if x < h:
        return 0
    else:
        return 1

def seuil_image(image,h):
    img = np.copy(image)
    a, b = np.shape(img)
    for i in range(a):
        for j in range(b):
            img[i,j] = seuil(h,img[i,j])
    return img
```

Solution de l'exercice 2 -

```
def discretiser(x,k):
    return np.floor(k*x)/k

def poster(image,k):
    img = np.copy(image)
    a, b = np.shape(img)
    for i in range(a):
        for j in range(b):
            img[i,j] = discretiser(img[i,j],k)
    return img
```

On peut appliquer la fonction directement à un tableau `numpy`.

```
def poster(image,k):
    return discretiser(image,k)
```

Cela n'a pas été fait dans la question précédente car la fonction avec des tests ne peut pas être distribuée sur une matrice. On peut la transformer pour que cela soit possible

```
seuil_img = np.vectorize(seuil_image)
```

On applique ensuite la nouvelle fonction `seuil_img`.

Solution de l'exercice 3 -

```
def luminosite(image, a):
    return image**a
```

Solution de l'exercice 4 -

```
def contrastePlus(image):
    return (np.sin((2*image-1)*np.pi/2)+1)/2

def contrasteMoins(image):
    return np.arcsin(2*image-1)/np.pi+1/2
```

Solution de l'exercice 5 - La version normale demande de parcourir chaque point en lui faisant calculer un produit de deux matrices, cela donne 4 boucles **for** imbriquées.

```
def convoler(image,noyau):
    m,n = np.shape(image)
    p,q = np.shape(noyau)
    img = np.zeros((m,n))
    p1 = (p-1)//2
    for i in range (p1,m-p1):
        for j in range(p1,n-p1):
            for k in range(p):
                for l in range(p):
                    img[i,j] = img[i,j]
                              +noyau[k,l]*image[i-p1+k,j-p1+l]
    return img
```

Cette solution sera assez lente : on peut l'améliorer en sommant par blocs ; on découpe un rectangle dans la matrice autour de la partie centrale, on le multiplie par le coefficient du noyau et on ajoute.

```
def convoler(image,noyau):
    m,n = np.shape(image)
    p,q = np.shape(noyau)
    img = np.zeros((m,n))
    p1 = (p-1)//2
    for i in range(p):
        for j in range(p):
            img[p1:-p1,p1:-p1] += noyau[i,j]
                                  *image[i:m-p+1+i,j:n-q+1+j]
    return img
```

Le gain de vitesse est net : on passe dans les exemples de 13 secondes à 0,04 secondes.

Solution de l'exercice 6 -

```
flou = np.ones((5,5))/25
image6= convoler(image,flou)
```

Solution de l'exercice 7 -

```
net = np.array([[0.0, -1, 0],[-1, 5, -1],[0, -1, 0]])
image7 = np.clip(convoler(image, net), 0, 1)
```

Solution de l'exercice 8 -

```
contour = np.array([[0.0,-1,0],[-1,4,-1],[0,-1,0]])
image8 = 1-np.clip(convoler(image,contour),0,1)
```

Solution de l'exercice 9 -

```
deltay = np.array([[-1.0,0,1],[-2,0,2],[-1.0,0,1]])
image9x = convoler(image,deltay)
deltay = np.array([[-1.0,-2,-1],[0,0,0],[1,2,1]])
image9y = convoler(image,deltay)
image9 = 1-np.sqrt(image9x**2+image9y**2)
image9a = seuil_image(image9,0.4)
```

Solution de l'exercice 10 -

```
def gaussienne(haut,larg,alpha):
    h0 = haut//2
    l0 = larg//2
    sigma = alpha*(haut+larg)/4
    g  = np.zeros((haut,larg))
    for i in range(haut):
        for j in range(larg):
            r2 = (i-h0)**2+(j-l0)**2
            g[i,j] = np.exp(-r2/(2*sigma**2))
    return g
```

Solution de l'exercice 11 -

```
def flouGaussien(image,alpha):
    m,n = np.shape(image)
    im1 = fft2(image)
    im2 = fftshift(im1)
    im3 = im2*gaussienne(m,n,alpha)
    im4 = ifftshift(im3)
    return ifft2(im4)
```

TP VI

Images couleurs

1 Images en couleur

1.1 Première exploration

`scipy` contient une image en couleurs, un mignon raton-laveur.

```
from scipy.misc import face

image1 = face()
```

Exercice 1
Afficher l'image.
Quelle est sa taille ?

On remarque que la matrice a trois dimensions.
De plus l'image est tout de suite vue en "vraies couleur", `cmap` n'a aucun effet. En effet les outils de `matplotlib` vont gérer le fait que l'image est composée de plusieurs couches.

Le dossier public contient aussi une image aux couleurs plus marquées que celle que fournit `scipy` : `bouquet.png`.

Exercice 2
Charger cette image en mémoire et l'afficher.

1.2 Séparation des couleurs

On a donc affaire à une structure de données à 3 dimensions : la hauteur, la largeur et la profondeur qui peut être 3 [1]. Dans l'interface `matplotlib` les valeurs de ce tableau sont le plus souvent des flottants compris entre 0 et 1 ou des entiers sur 8 bits, compris entre 0 et 255.
Pour voir l'effet de chaque couche nous allons créer des images qui n'utilisent qu'une des couches les deux autres étant mises à 0.
La première couche sera donc isolée par

```
p, q, r = np.shape(image1)
couche0 = np.zeros((p, q, r))
for i in range(p):
    for j in range(q):
        couche0[i, j, 0] = image1[i, j, 0]
```

1. la profondeur peut être 4 quand on ajoute une composante de transparence.

On peut remplacer la double boucle par une extraction généralisée à une liste multiple : les trois dimensions sont séparées par des virgules et on sélectionne les portions par `a:b`. Si `a` est omis, il est remplacé par 0, si `b` est omis, il est remplacé par la taille de la dimension.

```
p, q, r = np.shape(image1)
couche0 = np.zeros((p, q, r))
couche0[ : , : , 0] = image1[ : , : , 0]
```

On peut aussi définir la matrice en annulant les couches à partir de l'image.

```
couche0 = np.copy(image1)
couche0[ : , : , 1] = 0
couche0[ : , : , 2] = 0
```

On voit que `numpy` distribue (CAST en anglais) la valeur 0 à tous les points de l'extraction.

Exercice 3
Déterminer de même les images ne contenant respectivement que la couche 1 et la couche 2 de l'image.

On peut afficher plusieurs images en même temps :

```
plt.subplot(2, 2, 1)
plt.imshow(image1)
plt.subplot(2, 2, 2)
plt.imshow(couche0)
plt.subplot(2, 2, 3)
plt.imshow(couche1)
plt.subplot(2, 2, 4)
plt.imshow(couche2)
```

On voit donc que les couleurs sont séparées en 3 composantes : rouge, vert et bleu.
On notera (r, v, b) les composantes de la couleur d'un point elles sont obtenues par `image[i,j,:]`.

1.3 Création

Exercice 4
Définir une image de taille 300x300 nulle initialement dans laquelle on mettra les valeurs à 1 dans les partie décrites ci-après :

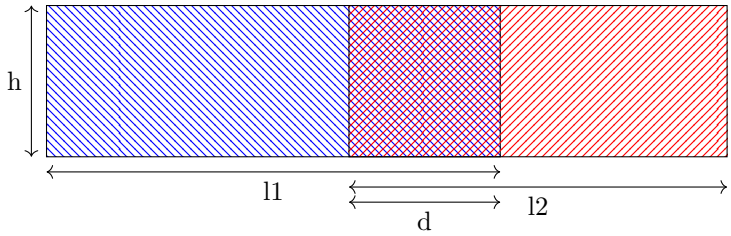
- *le cercle de centre (135,124) de rayon 50 pour la première couche,*
- *le cercle de centre (135,176) de rayon 50 pour la deuxième couche,*
- *le cercle de centre (180,150) de rayon 50 pour la troisième couche.*

On obtient une image classique d'illustration de la synthèse additive.
Par exemple on voit que le jaune est obtenu en ajoutant le rouge et le vert.
Le résultat devrait ressembler à l'image `spots.png` qui se trouve dans le dossier public.

Exercice 5
Créer le drapeau d'un pays de votre choix ; éviter les drapeaux compliqués comme celui du Royaume-Uni, préférer celui de la France, de la Belgique, de l'Italie, ...

2 Recollements d'images

Le but de cette partie est d'assembler deux images en essayant de ne pas faire apparaître la liaison.
Si on applique à deux copies d'une même image on peut prolonger celle-ci ; c'est l'idée à la base du remplissage d'une surface par un motif.
Nous nous plaçons dans le cas particulier d'un placement côte-à-côte de deux images de même hauteur.
On note l1 la largeur de la première image et l2 la largeur de la seconde.



Voici les images qui serviront dans les exemples.


Exercice 6

Écrire une fonction `decoupe(img1,img2,d)` *qui prend l1 -* $\frac{d}{2}$ *pixels de gauche à img1 et l2-*$\frac{d}{2}$ *pixels de droite de img2 et les associe : on découpe au milieu de la zone commune.*

`decoupe(img1,img2,100)` donne 

Exercice 7

Pour obtenir une transition moins brutale on va plutôt mélanger les deux images. Écrire une fonction `melange(img1,img2,d)` *qui calcule les pixel de la bande commune sous la forme* `img[x,y,k]` $=$ $\frac{\texttt{img1[x,y,k] + img2[x,y - l1 + d,k]}}{2}$. *Pourquoi divise-t-on par 2 ?*

`melange(img1,img2,100)` donne 

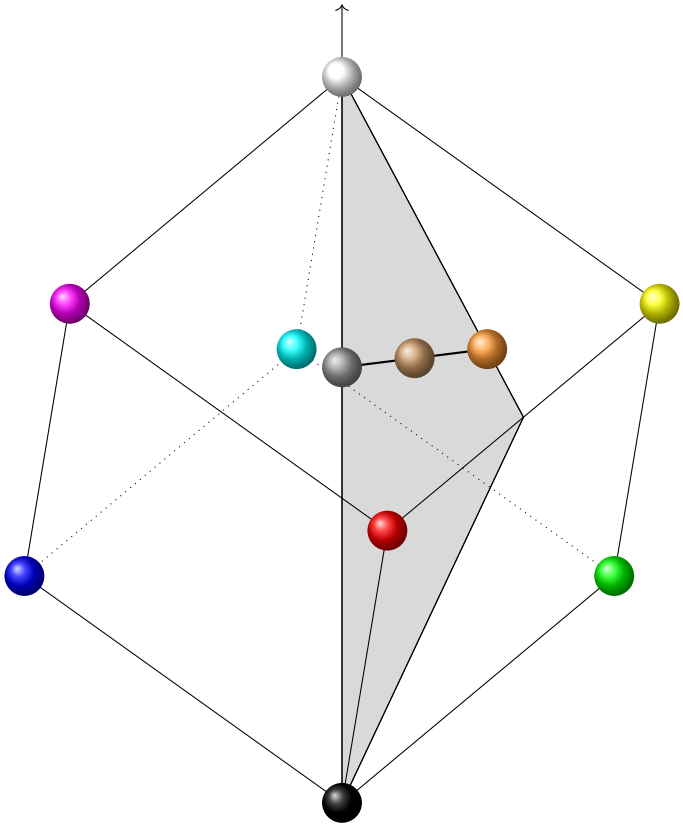
Exercice 8

Le mélange ci-dessus manque encore de transition : on va mélanger progressivement.
Écrire une fonction `fondu(img1,img2,d)` *qui calcule les pixel de la bande commune sous la forme* `img[x,y,k] = (1-t)img1[x,y,k] + t *img2[x,y - l1 + d,k]` *avec* $t = \frac{y-l1+d}{d}$ *et y compris entre l1 - d et l1.*

`fondu(img1,img2,100)` donne 

3 Autre définition des couleurs

Le modèle RVB employé ci-dessus est pratique pour le calcul des images mais rend difficile une modification cohérente des couleurs. On voudrait pouvoir augmenter/diminuer le caractère coloré des images (on parle de **saturation**), augmenter/diminuer la luminosité et même transformer les couleurs ...
On a vu que les 3 composantes des couleurs décrivent un cube : $[0;1]^3$.
On peut remarquer que la diagonale du cube, de (0,0,0) à (1,1,1) représente les tons de gris et que sa direction semble indiquer la luminosité. On va donc privilégier cette direction.



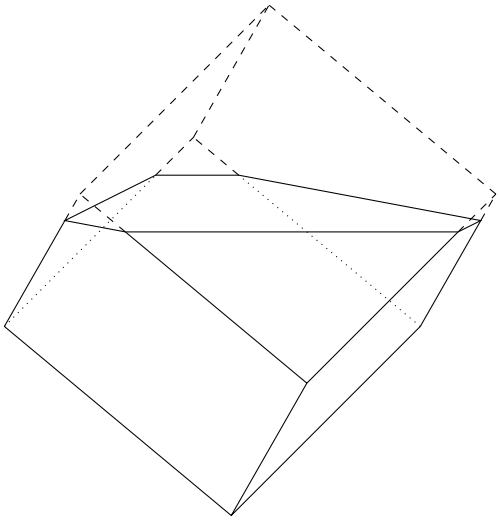
Nous allons choisir une représentation qui ressemble aux représentation HSV et HSL [2] mais dont les calculs sont simplifiés. Pour cela nous utiliserons les coordonnées cylindrique avec un axe porté par $(1,1,1)$. L'angle est déterminé de telle manière que les couleurs rouges, les points (r,0,0) ou (1,s,s), définissent un angle nul.
On commence par définir une nouvelle base orthonormée $(\vec{u}_1, \vec{u}_2, \vec{u}_3)$ dont le troisième vecteur dirige l'axe. Le choix de la couleur rouge comme angle initial impose que le plan engendré par $\vec{u}_1$ et $\vec{u}_3$ contienne le rouge, c'est-à-dire le vecteur $(1,0,0)$.
On aboutit à $\vec{u}_1 = \frac{1}{\sqrt{6}}\begin{pmatrix}2\\-1\\-1\end{pmatrix}$ $\vec{u}_2 = \frac{1}{\sqrt{2}}\begin{pmatrix}0\\1\\-1\end{pmatrix}$ et $\vec{u}_3 = \frac{1}{\sqrt{3}}\begin{pmatrix}1\\1\\1\end{pmatrix}$.
Une couleur de composantes (r,v,b) sera donc définie par les nouvelles coordonnées (X,Y,Z) avec $\begin{pmatrix}r\\v\\b\end{pmatrix} = X\vec{u}_1 + Y\vec{u}_2 + Z\vec{u}_3$ d'où $r = \frac{2X}{\sqrt{6}} + \frac{Z}{\sqrt{3}}$, $v = -\frac{X}{\sqrt{6}} + \frac{Y}{\sqrt{2}} + \frac{Z}{\sqrt{3}}$ et $b = -\frac{X}{\sqrt{6}} - \frac{Y}{\sqrt{2}} + \frac{Z}{\sqrt{3}}$
On pose alors $X = S\cos(H)$ et $Y = S\sin(\varphi)$ en coordonnées polaires donc $S = \sqrt{X^2+Y^2}$ et l'angle H est l'argument du complexe $X + iY$ que l'on peut calculer par une fonction `numpy` : `H = np.arctan2(Y,X)`.

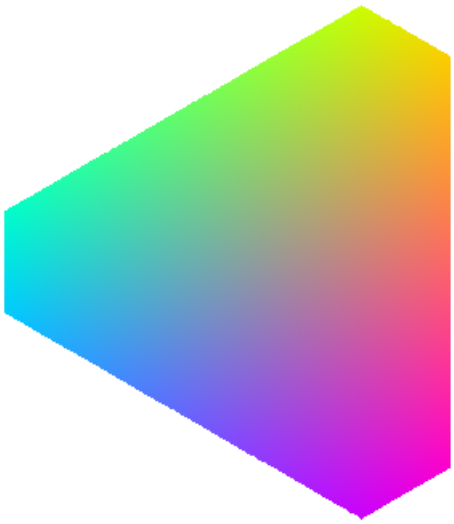
2. Voir `https://en.wikipedia.org/wiki/HSL_and_HSV`

On pose aussi $I = \dfrac{Z}{\sqrt{3}} = \dfrac{r+v+b}{3}$ pour obtenir un réel dans $[0;1]$.

- H (pour **hue**, teinte en anglais) indique la teinte.
- S est l'éloignement du gris, c'est une mesure de la saturation en couleur.
- On a vu que I mesurait la luminosité.

La saturation n'atteint pas la valeur 1 : voici les couleurs qui ont une luminosité $I = 0,6$, cela correspond à la section du cube par un plan perpendiculaire à l'axe de gris.





On remarque toutes les valeurs s'obtiennent à l'aide de fonctions simples donc on pourra les calculer globalement en appliquant les fonctions aux tableaux.
On définira les matrices des couleurs par `R = image[:,:,0]`, de même pour `V` et `B`.
La matrice des X se calcule alors par `X = (R + V + B)/np.sqrt(3)`.

Exercice 9
Écrire une fonction `HSI(image)` *qui renvoie les matrices des 3 composantes des couleurs des points.*

Exercice 10
Écrire une fonction `faireImage(H,S,I)` *qui renvoie l'image reconstituée à partir des composantes* (H, S, I) *des couleurs.*

Des composantes transformées peuvent donner des valeurs de r, g ou b qui ne sont pas dans $[0;1]$.
On tronquera les valeurs pour qu'elles restent entre 0 et 1 à l'aide de la fonction `np.clip(x,0,1)` qui renvoie 0 pour $x < 0$, x pour $0 \leqslant x \leqslant 1$ et 1 pour $x > 1$.

Exercice 11
Transformer les images (raton-laveur ou bouquet). On pourra

- *modifier la teinte en ajoutant une constante à H,*
- *modifier la luminosité en multipliant I par une constante positive,*
- *modifier la saturation en multipliant I par une constante positive.*

Dans le cas des multiplications on expérimentera avec un facteur supérieur à 1 et un facteur inférieur à 1.

4 Solutions

Solution de l'exercice 1 -

```
from scipy.misc import face

image1 = face()
plt.imshow(image1)
plt.show()

print(np.shape(image1))
```

Solution de l'exercice 2 -

```
image2 = plt.imread("/home/ericd13/Travail/2016-2017/IC2/TP/
   TP1 Images/bouquet.png")

plt.imshow(image3)
plt.show()
```

Solution de l'exercice 3 -

```
couche1 = np.copy(image1)
couche1[ : , : , 0] = 0
couche1[ : , : , 2] = 0

couche2 = np.copy(image1)
couche2[ : , : , 0] = 0
couche2[ : , : , 1] = 0
```

Solution de l'exercice 4 -

```
image2 = np.zeros((300,300,3))
for i in range(300):
    for j in range(300):
        if (i -135)**2 + (j-124)**2 < 50**2:
            image2[i,j,0] = 1
        if (i -135)**2 + (j-176)**2 < 50**2:
            image2[i,j,1] = 1
        if (i -180)**2 + (j-150)**2 < 50**2:
            image2[i,j,2] = 1

plt.imshow(image2)
```

Solution de l'exercice 5 -

Solution de l'exercice 6 -

```
def decoupe(img1,img2,d):
    h,l1,n = img1.shape
    h,l2,n = img2.shape
    ll1 = l1 - d//2
    tout = np.zeros((h,l1+l2-d,n))
    tout[:,0:ll1,:] = img1[:,0:ll1,:]
```

```
    tout[:,ll1:,:] = img2[:,d-d//2:,:]
    return tout
```

Solution de l'exercice 7 -

```
def melange(img1,img2,d):
    h,l1,n = img1.shape
    h,l2,n = img2.shape
    tout = np.zeros((h,l1+l2-d,n))
    tout[:,0:l1-d,:] = img1[:,0:l1-d,:]
    tout[:,l1:,:] = img2[:,d:,:]
    tout[:,l1-d:l1]=(img1[:,l1-d:,:]+img2[:,:d,:])/2
    return tout
```

Solution de l'exercice 8 -

```
def fondu(img1,img2,d):
    h,l1,n = img1.shape
    h,l2,n = img2.shape
    tout = np.zeros((h,l1+l2-d,n))
    tout[:,0:l1-d,:] = img1[:,0:l1-d,:]
    tout[:,l1:l1+l2-d,:] = img2[:,d:l2,:]
    for i in range(0,d):
        tout[:,l1-d+i,:] = (d-i)/d*img1[:,l1-d+i,:]+i/d*img2
            [:,i,:]
    return tout
```

Solution de l'exercice 9 -

```
def HSI(image):
    R = image[:,:,0]
    V = image[:,:,1]
    B = image[:,:,2]
    X = (2*R-V-B)/np.sqrt(6)
    Y = (V-B)/np.sqrt(2)
    I = (R+V+B)/3
    H = np.arctan2(Y,X)
    S = np.sqrt(X*X+Y*Y)
    return (H,S,I)
```

Solution de l'exercice 10 -

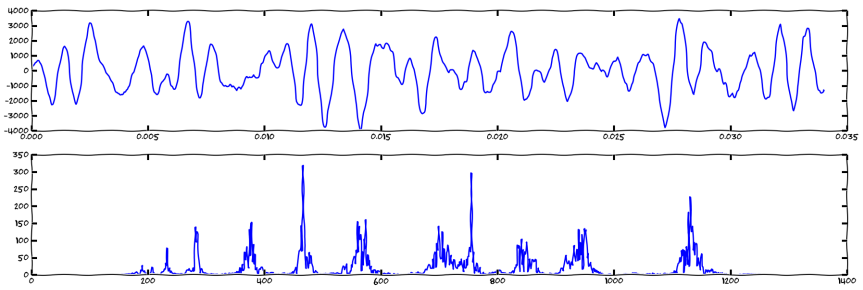
```
def faireImage(H,S,I):
    X = S*np.cos(H)
    Y = S*np.sin(H)
    haut, larg = np.shape(H)
    image = np.zeros((haut,larg,3))
    image[:,:,0] = 2*X/np.sqrt(6) + I
    image[:,:,1] = -X/np.sqrt(6) + Y/np.sqrt(2) + I
    image[:,:,2] = -X/np.sqrt(6) - Y/np.sqrt(2) + I
    return np.clip(image,0,1)
```

Solution de l'exercice 11 -

TP VII

F.F.T.

Transformation de Fourier Discrète



1 Introduction

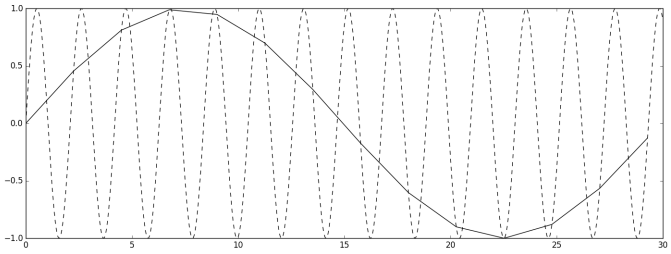
Le but de ce T.P. est d'étudier une transformation réversible de listes.

Les listes utilisées représentent la valeur d'une variable en fonction du temps, on parle de signal, et on souhaite étudier la variable du point de vue des fréquences.

L'étude en fréquence d'une fonction périodique peut se faire à l'aide des séries de Fourier : c'est un outil puissant utilisé utilisé dans de nombreux domaines. Cependant cette étude théorique rencontre des obstacles dans des applications concrètes.

- Une fonction périodique doit être définie sur $\mathbb{R}$ tout entier : nous ne connaissons le signal que sur un intervalle de temps.
- On pourrait ne connaître le signal que sur une période pour en étudier les composantes harmoniques mais la période (ou la fréquence) est en général inconnue, ce sera souvent la première mesure à faire.
- Les mesure ne sont connues qu'en un nombre fini de valeur du temps : le temps est discrétisé. Cela rend invisibles les fréquences au delà d'un seuil.
  Nous supposerons ici que la discrétisation est régulière, il y a une fréquence d'échantillonnage.
- Les mesures elles-mêmes ne sont faites que sur une échelle discrète, on parle de précision de la mesure. Dans cette étude nous ne considérerons pas ce problème, la précision sera supposée assez grande pour le phénomène de crénelage (passage brutal d'une valeur à une autre) soit négligeable.

Pour illustrer un de ces problèmes voici (en trait plein) le signal mesuré lorsque la fréquence d'échantillonnage est petite devant celle du signal (en traits discontinus).



Le principe à retenir (qui est une forme du théorème de Shannon) peut être "*Les fréquences supérieure à la moitié de la fréquence d'échantillonnage sont invisibles dans le signal échantillonné.*

Pour ce qui est de l'intervalle de temps pendant lequel le signal est échantillonné nous supposerons que ce temps est suffisamment long pour contenir un grand nombre de fréquences de base du signal. La limitation de ce temps permet d'avoir des tailles d'échantillons raisonnables et aussi de gérer des signaux dynamique dont les fréquences varient dans le temps.
Un exemple de ce comportement est l'**auto-tune** : le principe est de calculer la fréquence de la voix d'un chanteur et de la caler à la note de musique la plus proche à chaque intervalle de temps. Lorsque la fréquence chantée n'est pas tenue, la note restituée peut faire des sauts d'un demi-ton caractéristiques de ce procédé.
Les outils utilisés, calcul de la fréquence et ajustement (pitch-shifting), sont calculés à l'aide de la transformation étudiée dans ce chapitre.

2 Transformation de Fourier discrète

2.1 Données

On part d'un signal représenté par une fonction $f(t)$.
Il est numérisé pendant un temps T divisé en N intervalles ; on suppose que l'origine de l'échantillonnage correspond à $t = 0$. On obtient donc un tableau de N valeurs, $x_i = f\big(i\frac{T}{N}\big)$, que l'on notera X.
La fréquence d'échantillonnage est $\frac{1}{T/N} = \frac{N}{T}$.
Dans cette étude on considérera X comme un élément de $\mathbb{R}^N$ ou de $\mathbb{C}^N$.

La représentation fréquentielle va consister à représenter X en fonction de signaux associés à des fréquences.

- Un choix naturel est de considérer une expression linéaire : on se donne un nombre fini de tableaux F_k associées à des fréquences fixes et on veut écrire $X = \sum_{k=0}^{p} c_k F_k$.
- Comme le but est de calculer les coefficients c_k il faudra résoudre le système : on choisira de prendre N vecteurs indépendants pour les F_k.
  Mathématiquement cela revient à effectuer un changement de base dans l'espace K^N.
- Pour déterminer les fréquences utilisées on choisit les multiples de la fréquence ayant T pour période : les fréquences seront donc de la forme $f_k = \frac{k}{T}$. Les périodes sont dont $\frac{T}{k}$.
  D'autres choix sont possibles mais celui-ci permet les calculs les plus simples.
- Le théorème de Shannon implique que l'on doit choisir k tel que $f_k = \frac{k}{T} \leqslant \frac{1}{2}\frac{N}{T}$ donc $k \leqslant \frac{N}{2}$.

2.2 Composantes réelles

L'usage est de considérer que les signaux périodiques "simples" de fréquence donnée sont les fonctions sinusoïdales : pour une fréquence f données ce sont les fonctions $t \mapsto A.\sin(2\pi ft + \varphi)$ que l'on peut écrire aussi $t \mapsto a.\cos(2\pi ft) + b\sin(2\pi ft)$.
Lorsque le signal a des propriétés de symétrie on peut n'utiliser que des sinus ou des cosinus.
Cette méthode est tout-à-fait satisfaisante en théorie. Cependant d'un point de vue pratique il faudra gérer des formules différentes pour les sinus et cosinus et, surtout, il n'existe pas de possibilité d'obtenir un calcul rapide comme celui qui sera présenté ici.

2.3 Composantes complexes

La représentation que nous allons utiliser unifie les fonctions en utilisant les exponentielles :

$$\cos(2\pi ft) = \frac{1}{2}\big(e^{2i\pi ft} + e^{-2i\pi ft}\big) \quad \sin(2\pi ft) = \frac{1}{2i}\big(e^{2i\pi ft} - e^{-2i\pi ft}\big)$$

Les fréquences utiles sont les $f_k = \frac{k}{T}$ avec $0 \leqslant k \leqslant \frac{N}{2}$
donc on peut utiliser les fonctions $e_k : t \mapsto e^{2i\pi kt/T}$ avec $|k| \leqslant \frac{N}{2}$.
Le tableau des valeurs de e_k aux points $p\frac{T}{N}$ est $F_k = \big[e_k\big(p\frac{T}{N}\big)\ ;\ 0 \leqslant p < N\big]$
Les composantes sont $e^{2i\pi \frac{k}{T} p \frac{T}{N}} = e^{2i\pi kp/N}$.

$$F_k = \big[1, e^{2ik\pi/N}, e^{2ik2\pi/N}, e^{2ik3\pi/N}, \ldots, e^{2ik(N-1)\pi/N}\big] \quad -\frac{N}{2} \leqslant k \leqslant \frac{N}{2}$$

On remarque qu'on a $e^{2i\pi kp/N} = e^{2i\pi kp/N + 2ip\pi} = e^{2i\pi(k+N)p/N}$ d'où $F_k = F_{k+N}$.
On peut donc remplacer F_{-k} par F_{-k+N} : on choisit la famille $\big(F_k\ ;\ 0 \leqslant k < N\big)$.

Exercice 1
Prouver que la famille $\big(F_k\ ;\ 0 \leqslant k < N\big)$ est une base de $\mathbb{C}^N$.

N.B. le choix est légitime car la famille est libre,

2.4 Calcul direct

L'équation à résoudre, $X = \sum_{k=0}^{p} c_k F_k$, s'écrit donc sous la forme de N équations dont les N inconnues sont les coefficients c_k que l'on obtient en séparant les composantes

$$\begin{array}{rlllll}
x_0 = c_0 & + c_1 & + \cdots & + c_k & + \cdots & + c_{N-1} \\
x_1 = c_0 & + c_1 e^{2i\pi/N} & + \cdots & + c_k e^{2i\pi k/N} & + \cdots & + c_{N-1} e^{2i\pi(N-1)/N} \\
\vdots \quad \vdots & \vdots & \vdots \quad \vdots & \vdots & \vdots & \vdots \\
x_p = c_0 & + c_1 e^{2ip\pi/N} & + \cdots & + c_k e^{2ip\pi k/N} & + \cdots & + c_{N-1} e^{2ip\pi(N-1)/N} \\
\vdots \quad \vdots & \vdots & \vdots \quad \vdots & \vdots & \vdots & \vdots \\
x_{N-1} = c_0 & + c_1 e^{2i(N-1)\pi/N} & + \cdots & + c_k e^{2i(N-1)\pi k/N} & + \cdots & + c_{N-1} e^{2i(N-1)\pi(N-1)/N}
\end{array}$$

On peut aussi l'écrire sous la forme $X = \Omega.C$ où C est le vecteur $(c_0, c_1, \ldots, c_{N-1})$ et $\Omega = (\omega_{a,b})_{1 \leqslant a,b \leqslant N}$ est une matrice de $\mathcal{M}_N(\mathbb{C})$ définie par $\omega_{a,b} = e^{2i(a-1)(b-1)\pi/N}$.

Exercice 2
Prouver qu'on a $\Omega.\overline{\Omega} = \overline{\Omega}.\Omega = NI_N$ avec $\overline{\Omega} = (\overline{\omega_{a,b}})_{1 \leqslant a,b \leqslant N}$

On a donc $C = \Omega^{-1}.X = \frac{1}{N}\overline{\Omega}.X$ donc $c_k = \dfrac{1}{N}\displaystyle\sum_{p=0}^{N-1} e^{-2ipk\pi/N} x_p$.

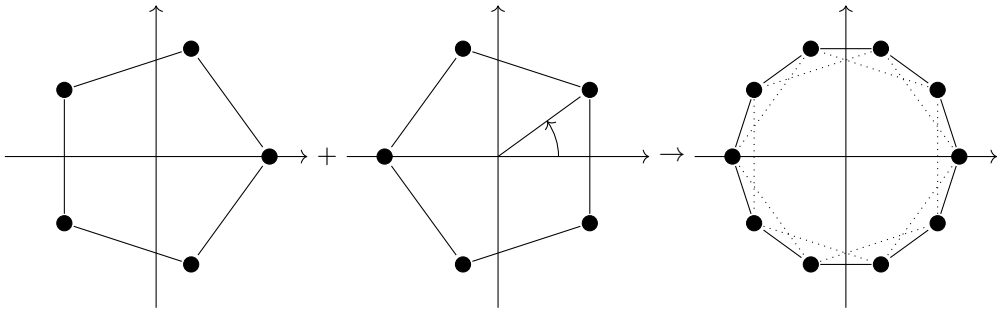
C est la **transformé de Fourier discrète** de X.
En anglais DFT pour Discrete Fourier Transform.

- Même lorsque le signal est réel sa transformée est complexe.
- $c_0 = \dfrac{1}{N}\sum_{p=0}^{N-1} x_p$ est la valeur moyenne des x_p, c'est la composante constante.
- Si X est réel on a $c_{N-k} = \overline{c_k}$ pour $1 \leqslant k \leqslant N-1$. Les seule valeurs utiles si on ne considère que les modules des coefficients sont les valeur entre 0 et $N/2$.
- Cela correspond à la majoration de la fréquence visible par le théorème de Shannon.
- Le nombre de multiplication effectuées est N^2.

Exercice 3
Prouver que le signal est réel si et seulement si $c_{N-k} = \overline{c_k}$ *pour* $1 \leqslant k \leqslant N-1$ *et* $c_0 \in \mathbb{R}$.

3 Transformation de Fourier rapide

Le calcul de la transformée de Fourier vu ci-dessus demande un nombre de multiplication (et d'additions) de l'ordre de N^2 où N est la taille de l'échantillon. On peut remarquer que c'est un progrès par rapport à la résolution simple d'un système de N équations à N inconnues qui demande un nombre de multiplications de l'ordre de N^3. Cependant l'utilisation de l'algorithme est limitée en raison de cette complexité : il arrivera souvent que l'on ait besoin de la décomposition spectrale "en temps réel".
Une utilisation de la structure des coefficients $e^{2ipq\pi/N}$ pour N pair va permettre une amélioration du temps de calcul.
Si on note $U_n = \{e^{2ip\pi/n}\ ;\ 0 \leqslant p < n\}$ on obtient, en séparant les termes $e^{2ip\pi/2n}$ avec p pairs ou p impair, $U_{2n} = U_n \cup e^{2i\pi/2n}.U_n$ avec la notation $a.E = \{ax\ ;\ x \in E\}$.
Dans le dessin ci-après on construit U_{10} à partir de U_5 et de la rotation de U_5 d'angle $\frac{2\pi}{10}$.



$$
\begin{aligned}
c_k &= \frac{1}{N}\sum_{p=0}^{N-1} e^{-2ipk\pi/N}x_p = \frac{1}{2M}\sum_{p=0}^{2M-1} e^{-2ipk\pi/2M}x_p \text{ en posant N=2M} \\
&= \frac{1}{2M}\sum_{p=0,\ p \text{ pair}}^{2M-1} e^{-2ipk\pi/2M}x_p + \frac{1}{2M}\sum_{p=0,\ p \text{ impair}}^{2M-1} e^{-2ipk\pi/2M}x_p \\
&= \frac{1}{2M}\sum_{q=0}^{M-1} e^{-2i(2q)k\pi/2M}x_{2q} + \frac{1}{2M}\sum_{q=0}^{M-1} e^{-2i(2q+1)k\pi/2M}x_{2q+1} \\
&= \frac{1}{2M}\sum_{q=0}^{M-1} e^{-2iqk\pi/M}x_{2q} + \frac{e^{-2ik\pi/2M}}{2M}\sum_{q=0}^{M-1} e^{-2iqk\pi/M}x_{2q+1}
\end{aligned}
$$

Si on note X' la suite $(x_0, x_2, \ldots, x_{2N-2})$ et X'' la suite $(x_1, x_3, \ldots, x_{2N-1})$, les coefficients de la

transformée de Fourier de X' et X'', de longueur M, sont, pour $0 \leqslant k < M$,

$$c'_k = \frac{1}{M}\sum_{q=0}^{M-1} e^{-2iqk\pi/M} x'_q = \frac{1}{M}\sum_{q=0}^{M-1} e^{-2iqk\pi/M} x_{2q}$$
$$c''_k = \frac{1}{M}\sum_{q=0}^{M-1} e^{-2iqk\pi/M} x''_q = \frac{1}{M}\sum_{q=0}^{M-1} e^{-2iqk\pi/M} x_{2q+1}$$

On a donc $c_k = \frac{1}{2}\big(c'_k + e^{-2ik\pi/2M} c''_k\big)$ pour $0 \leqslant k < M$.

Pour k compris entre M et $N-1$ on pose $k = k' + M$.
On trouve alors, en utilisant $e^{-2iqM\pi/M} = 1$ et $e^{-2iM\pi/2M} = -1$,

$$\begin{aligned} c_k &= \frac{1}{2M}\sum_{q=0}^{M-1} e^{-2iq(k'+M)\pi/M} x'_q + \frac{e^{-2i(k'+M)\pi/2M}}{2M}\sum_{q=0}^{M-1} e^{-2iq(k'+M)\pi/M} x''_q \\ &= \frac{1}{2M}\sum_{q=0}^{M-1} e^{-2iqk'\pi/M} x'_q + \frac{-e^{-2ik'\pi/2M}}{2M}\sum_{q=0}^{M-1} e^{-2iql\pi/M} x''_q = \frac{1}{2}\big(c'_l - e^{-2il\pi/N} c''_l\big) \end{aligned}$$

3.1 Algorithme

Pour pouvoir effectuer la division par 2 à chaque étape nous allons supposer que

N est une puissance de 2.

On arrive ainsi à un calcul récursif.
Pour calculer les coefficients à partir d'une suite de longueur N,

1. Si $N = 1$ on a $c_0 = x_0$
2. Sinon on décompose la suite en deux suite de taille $M = N/2$,
   $X' = (x_0, x_2, \ldots, x_{2N-2})$ et $X'' = (x_1, x_3, \ldots, x_{2N-1})$
3. On calcule les coefficients des suites X' et X''
4. On calcule $c_k = \frac{1}{2}\big(c'_k + e^{-2ik\pi/N} c''_k\big)$ et $c_{k+M} = \frac{1}{2}\big(c'_k - e^{-2ik\pi/N} c''_k\big)$ pour $0 \leqslant k < M$.

Complexité

On note $C(N)$ le nombre de multiplications nécessaires pour calculer la suite des c_k.
On a $C(N) = 2C(M) + 4M$ et $C(1) = 0$.
Si on pose $N = 2^p$ (d'où $M = 2^{p-1}$) on a $C(2^p) = 2C(2^{p-1}) + 4.2^{p-1}$.
On pose $C(2^p) = 2^p u_p$, on a $u_p = u_{p-1} + 2$ et $u_0 = 0$ d'où $u_p = 2p$ puis
$C(2^p) = 2p2^p = 2\log_2(N)N$ qui est infiniment plus petit que le N^2 de la méthode ci-dessus.

On calcule ici le produit d'une matrice par un vecteur avec une complexité quasi-linéaire !

Exercice 4
Montrer que le nombre d'additions est $\log_2(N)N$*, le nombre de calculs d'une exponentielle complexe est* $\frac{1}{2}\log_2(N)N$ *et le nombre d'affectations est* $2\log_2(N)N$.

3.2 Transformation inverse

La transformation de Fourier inverse consiste à déterminer les x_p à partir des c_k ; cela correspond aux formules initiales.
On connaît la suite des c_k avec $0 \leqslant k < N = 2M$.
On veut calculer $x_p = \sum_{k=0}^{N-1} c_k e^{2ip\pi k/N}$ pour $0 \leqslant p < N$.
On définit $d_k = c_{2k}$ et $d'_k = c_{2k+1}$ pour $0 \leqslant k < M$.
On suppose qu'on a calculé $y_p = \sum_{k=0}^{M-1} d_k e^{2ip\pi k/M}$ et $y'_p = \sum_{k=0}^{N-1} d'_k e^{2ip\pi k/M}$ pour $0 \leqslant p < M$.

Exercice 5
Prouver qu'on a $x_p = y_p + e^{2ik\pi/N} y'_p$ *et* $x_{p+M} = y_p - e^{2ik\pi/N} y'_p$ *pour* $0 \leqslant p < M$.

4 Programmation

- Pour accélérer les calculs on utilisera des tableaux `numpy` : ils sont créés en convertissant une liste (`np.array(liste)`) ou en affectant les valeurs d'une liste définie par `np.zeros(taille)`.
- On rappelle qu'on peut additionner ou multiplier deux tableaux de même longueur et multiplier un tableau par une constante sans avoir besoin d'écrire une boucle `for`.
- Ces tableaux, comme les listes usuelles, permettent l'extraction sous la forme `liste[a:b:c]` qui extrait les termes `liste[a+kc]` avec $k \geqslant 0$ et $a + kc < b$.
- Les nombres complexes sont utilisables nativement dans python. Le nombre complexe de module 1 et d'argument $\frac{\pi}{2}$ est noté `1j`.
- On peut définir un nombre complexe sous la forme `0.6+0.8j` si ses parties réelles et imaginaires sont explicites mais si les composantes sont des variables il faut employer `a +b*1j` ou `complex(a,b)` qui renvoie $a + ib$.
- Les fonctions `numpy` acceptent des variables complexes. En particulier on peut définir le nombre complexe $e^{i\theta}$ par `np.exp(theta*1j)`
- Si on veut un tableau qui peut contenir des complexes on l'initialise en donnant son type :

```
np.zeros(taille, dtype=complex)
```

4.1 Implémentation de l'algorithme

Exercice 6
Écrire une fonction `separer(liste)` *qui reçoit une liste de taille* $N = 2M$ *et qui renvoie les deux listes de taille* M *formées des éléments d'indices respectivement pairs et impairs de la liste initiale dans le même ordre.*

Exercice 7
Écrire une fonction `listeExp(N)` *qui reçoit un entier pair* N *et qui renvoie le tableau* `numpy` *de taille* $N/2$ *dont chaque composante d'indice* k *est* $e^{-2ik\pi/N}$.

Exercice 8
Écrire une fonction `TFR(liste)` *qui reçoit une liste de taille* 2^p *et qui renvoie la liste (complexe) de ses coefficients de Fourier en utilisant l'algorithme rapide vu ci-dessus.*
On ne vérifiera pas que la taille est une puissance de 2.

4.2 Quelques spectres

Dans cette partie nous allons construire quelques signaux et les analyser.
Pour que les calculs soient rapides les échantillons seront de taille $N = 1024 = 2^{10}$.
Si on veut une valeur réelle des fréquences calculées il faut connaître la durée de l'échantillon donc la fréquence d'échantillonnage : on choisit $f_e = 1000\ Hz$. La durée totale est donc $\frac{N}{f_e} = 1,024\ s$, le pas de temps est $dt = \frac{1}{f_e} = 1\ ms$, le pas des fréquences, df, est l'inverse de la durée.

```
N = 1024      # nombre de pas
fe = 1000     # fréquence d'échantillonnage
duree = N/fe  # durée de l'échantillonnage
dt = 1/fe     # temps entre deux mesures
df = 1/duree  # fréquence de base
```

Pour tracer les représentations on pourra utiliser des tableaux `numpy` pour les abscisses : `temps` pour les signaux et `freq` pour les composantes de la transformée.
On affichera, les fréquences supérieures à la fréquence limite sous la forme de fréquences négatives.

```
U = np.arange(N)     # tableau de taille N, de 0 à N-1
temps = U*dt
freq = (U-N//2)*df
```

On définit une fonction pour le signal, par exemple

```
f = 25
def fonction(t):
    return np.sin(2*np.pi*f*t)
```

Certaines fonctions ne sont pas directement applicables à un tableau `numpy`, par exemple celles qui effectuent un test (`if`). On peut les transformer pour qu'elles le deviennent avec l'instruction `fonction = np.vectorize(fonction1)`.
Voici un modèle simple pour dessiner le signal et sa transformée.
On notera que l'on calcule les modules des coefficients pour l'affichage.

```
signal = fonction(temps)
coef = TFR(signal)
coefR = np.zeros(N)
coefR[:N//2] = abs(coef)[N//2:]
coefR[N//2:] = abs(coef)[:N//2]

plt.clf()  # On efface

plt.subplot(2,1,1)        # Premier dessin, le signal
plt.plot(temps,signal)    # Tracé
plt.xlim((0,1.05))        # On ajuste l'axe horizontal
plt.xlabel("Temps [s]")   # Label pour les abscisses
plt.ylabel("Signal")      # Label pour les ordonnées

plt.subplot(2,1,2)        # Second dessin, la transformée
plt.plot(freq,coefR)
plt.xlabel("Fréquence [Hz]")
plt.ylabel("Coefficients")

plt.show()                # On fait dessiner
```

La dernière instruction n'est utile que si on n'est pas en mode interactif. On se place en mode interactif avec la commande `plt.ion()`, on en sort avec la commande `plt.ioff()`.

Exercice 9
Déterminer un signal et sa transformée dans les cas suivants.

1. $\sin(2\pi ft)$, *on pourra choisir* $f = 62,5$ *puis* $f = 50$.
   Donner une explication à la différence des spectres.
2. $\sin(2\pi f_1 t) + \sin(2\pi f_2 t)$ *puis* $\sin(2\pi f_1 t).\sin(2\pi f_2 t)$.
3. $|\sin(2\pi ft)|$.
4. *Un signal en dents-de-scie*[1].
5. *Un signal triangulaire.*
6. *Un signal carré.*

1. `x%y` donne la partie fractionnaire de $\frac{x}{y}$ quand x ou y est un réel.

4.3 Transformation inverse

La transformation inverse a été vue à la partie 3.2.
Elle se calcule de la même façon que la transformation directe à l'exception de 2 changements.

- On ne divise plus par 2 les reconstitutions des coefficients.
- On multiplie par une exponentielle de la forme $e^{2ik\pi/N}$ au lieu de $e^{-2ik\pi/N}$

Exercice 10
Écrire une fonction `listeExpInv(N)` *qui reçoit un entier pair* N *et qui renvoie le tableau* `numpy` *de taille* $N/2$ *dont chaque composante d'indice* k *est* $e^{2ik\pi/N}$.

Exercice 11
Écrire une fonction `invTFR(liste)` *qui reçoit une liste de taille* 2^p *représentant les coefficients de Fourier et qui renvoie la liste des valeurs en utilisant l'algorithme rapide vu ci-dessus.*

4.4 Filtrage

Une application de la transformation de Fourier est le filtrage :

- on transforme un signal,
- on modifie les coefficients selon la fréquence,
- on calcule la transformée inverse.

On propose deux sortes de filtrages passe-bas :

1. on enleve les fréquences hautes, par exemple entre `N//6` et `N-N//6`,
2. on atténue les fréquences hautes, par exemple en multipliant par une liste qui décroit de 1 à 0 entre 0 et `N//2` puis croît de 0 à 1 entre `N//2` et `N`.

Exercice 12
Expérimenter ces deux types de filtres avec des signaux carrés, triangulaires ou en dents-de-scie.

5 Solutions

Solution de l'exercice 1 - Le déterminant de cette famille de vecteurs est le déterminant de Vandermonde $V(1, e^{2i\pi/N}, \cdots, e^{2i\pi(N-1)/N})$. Il est non nul car les N racines de l'unité sont distinctes.

Solution de l'exercice 2 -

$\Omega.\overline{\Omega} = (p_{a,b})_{1\leqslant a,b\leqslant N}$ avec $p_{a,b} = \sum_{c=1}^{N} \omega_{a,c}\overline{\omega_{c,b}}$. Ainsi

$$\begin{aligned} p_{a,b} &= \sum_{c=1}^{N} e^{2i(a-1)(c-1)\pi/N} e^{-2i(c-1)(b-1)\pi/N} = \sum_{c=1}^{N} e^{2i(a-b)(c-1)\pi/N} = \sum_{q=0}^{N-1} e^{2i(a-b)q\pi/N} \\ &= \sum_{q=0}^{N-1} \left(e^{2i(a-b)\pi/N}\right)^q = \frac{1-\left(e^{2i(a-b)\pi/N}\right)^N}{1-e^{2i(a-b)\pi/N}} = \frac{1-e^{2i(a-b)\pi}}{1-e^{2i(a-b)\pi/N}} = 0 \text{ si } e^{2i(a-b)\pi/N} \neq 1 \\ &= \sum_{q=0}^{N-1} 1 = N \text{ si } e^{2i(a-b)\pi/N} = 1 \end{aligned}$$

Or $e^{2i(a-b)\pi/N} = 1$ si et seulement si $a-b$ est un multiple de N c'est-à-dire si et seulement si $a = b$ pour $0 \leqslant a, b < N$.

Solution de l'exercice 3 - On a $e^{-2ip(N-k)\pi/N} = e^{2ip\pi}e^{2ipk\pi/N} = e^{2ipk\pi/N} = \overline{e^{-2ipk\pi/N}}$.
Si X est réel alors, pour $1 \leqslant k \leqslant N-1$,
$c_{N-k} = \frac{1}{N}\sum_{p=0}^{N-1} e^{-2ip(N-k)\pi/N}x_p = \frac{1}{N}\sum_{p=0}^{N-1} \overline{e^{-2ipk\pi/N}}x_p = \overline{\frac{1}{N}\sum_{p=0}^{N-1} e^{-2ipk\pi/N}x_p} = \overline{c_k}$.
On a vu qu'on avait c_0 réel dans ce cas.

Inversement, si c_0 est réel et $c_{N-k} = \overline{c_k}$ pour $1 \leqslant k \leqslant N-1$ alors

$$\begin{aligned} \overline{x_p} &= \overline{\sum_{k=0}^{N-1} e^{2ipk\pi/N}c_k} = \overline{c_0 + \sum_{k=1}^{N-1} e^{2ipk\pi/N}c_k} = c_0 + \sum_{k=1}^{N-1} e^{-2ipk\pi/N}c_{N-k} = c_0 + \sum_{k=1}^{N-1} e^{2ip(N-k)\pi/N}c_{N-k} \\ &= c_0 + \sum_{q=1}^{N-1} e^{2ipq\pi/N}c_q = x_p \end{aligned}$$

Solution de l'exercice 4 - On a $C_{add}(2M) = 2C_{add}(M) + 2M$, $C_{exp}(2M) = 2C_{exp}(M) + M$, $C_{aff}(2M) = 2C_{aff}(M) + 2M$ d'où les résultats avec la même méthode que ci-dessus.
On remarquera que, dans le cas de l'algorithme simple, on a $C_{add}(N)) = N^2$, $C_{exp}(N)) = N^2$ et $C_{aff}(N)) = N$, seule la complexité en nombre d'affectation a (légèrement) augmenté.

Solution de l'exercice 5 -

$$\begin{aligned} x_p &= \sum_{k=0}^{N-1} e^{2ipk\pi/N}c_k = \sum_{k=0}^{2M-1} e^{2ipk\pi/2M}c_k \\ &= \sum_{l=0}^{M-1} e^{2ip(2l)\pi/2M}c_{2l} + \sum_{q=0}^{M-1} e^{2ip(2l+1)\pi/2M}c_{2l+1} \\ &= \sum_{l=0}^{M-1} e^{2ipl\pi/M}d_l + e^{2ip\pi/N}\sum_{l=0}^{M-1} e^{2ipl\pi/M}d'_l \end{aligned}$$

Solution de l'exercice 6 -

```
def separer(liste):
    lPairs = liste[::2]
    lImpairs = liste[1::2]
    return lPairs, lImpairs
```

Solution de l'exercice 7 -

```
def listeExp(N):
    M = N//2
    expo = np.zeros(M,dtype=complex)
    for k in range(M):
        theta = -k*np.pi/M
        expo[k] = np.exp(theta*1j)
    return expo
```

On peut, ici aussi, éviter les boucles.

```
def listeExp(N):
    M = N//2
    A = np.linspace(0,M-1,M)
    Theta = -A*np.pi/M
    return np.exp(Theta*1j)
```

Solution de l'exercice 8 -

```
def TFR(liste):
    N = len(liste)
    if N == 1:
        return np.array([liste[0]+0j])
    else:
        M = N//2
        C= np.zeros(N,dtype=complex)
        X1, X2 = separer(liste)
        C1 = TFR(X1)
        C2 = TFR(X2)
        E = listeExp(N)
        C[:M] = (C1 + E*C2)/2
        C[M:] = (C1 - E*C2)/2
        return C
```

Solution de l'exercice 9 -

1. Pour $f = 62,5$ la longueur est un multiple entier de la fréquence : il n'y a qu'une paire de coefficients non nuls. Dans le cas général il apparaît des coefficients non négligeables autour de la fréquence.
2. La linéarisation du produit fait apparaître des signaux de fréquence $f_1 + f_2$ et $f_1 - f_2$
3. On notera que c_0 est non nul, le signal a une composante constante.
4.

```
f = 10
def fn(t):
    T = 1/f
    u = t%T
    return 2*f*u-1 # de -1 à 1

fonction = np.vectorize(fn)
```

5.
```
f = 100
def fn(t):
    T = 1/f
    u = t%T
    return abs(2*f*u-1)

fonction = np.vectorize(fn)
```

On remarque que seuls les multiples impairs de la fréquence apparaissent.

```
f = 10
def fn(t):
    T = 1/f
    u = t%T
    if u < T/2:
        return 1
    else:
        return -1

fonction = np.vectorize(fn)
```

On peut aussi utiliser `np.sign(np.sin(2*np.pi*f*t))`

Solution de l'exercice 10 -

```
def listeExpInv(N):
    M = N//2
    A = np.linspace(0,M-1,M)
    Theta = A*np.pi/M
    return np.exp(Theta*1j)
```

Solution de l'exercice 11 -

```
def invTFR(liste):
    N = len(liste)
    if N == 1:
        return np.array([liste[0]+0j])
    else:
        M = N//2
        X= np.zeros(N,dtype=complex)
        C1, C2 = separer(liste)
        X1 = invTFR(C1)
        X2 = invTFR(C2)
        E = listeExp2(N)
        X[:M] = X1 + E*X2
        X[M:] = X1 - E*X2
        return X
```

Solution de l'exercice 12 -

```
N = 1024    # nombre de pas
fe = 1000    # frequence d'échantillonnage
duree = N/fe # durée de l'échantillonnage
dt = 1/fe    # temps entre deux mesures
df = 1/duree # fréquence de base

U = np.arange(N)    # tableau de taille N, de 0 à N-1
temps = U*dt
freq = (U-N//2)*df

f = 10
# Carré
# def fn(t):
#     T = 1/f
#     u = t%T
#     if u < T/2:
#         return 1
#     else:
#        return -1

# Dents de scie
def fn(t):
    T = 1/f
    u = t%T
    return 2*f*u-1 # de -1 à 1

# Triangle
# def fn(t):
#     T = 1/f
#     u = t%T
#     return abs(2*f*u-1)

fonction = np.vectorize(fn)

signal = fonction(temps)
coef = TFR(signal)

# Filtrage doux
# coef = coef*(abs(2*U-N)/N)**4

# Filtrage raide
d = N//6
coef[d:N-d]=0

signal1 = invTFR(coef).real
signal2 = invTFR(coef).imag
```