

# Simulation Numérique et Applications

Aide-Mémoire pour la préparation de l'épreuve  
d'Informatique du Concours National d'Entrée aux  
Ecoles d'Ingénieurs MP-PC-T 2020



Elaboré par :

***Mohamed Hammami (IPEIEM)***

***Houcemeddine Filali (IPEIM)***

***Laila Hammami (IPEIN)***

***Wafa Meftah (IPEIS)***

***Hayet Latrach (IPEIT)***

***Besma Fayeche (ENSIT)***

## Préambule

*Ce document est destiné aux étudiants de deuxième année MP-PC-T des Instituts Préparatoires aux Etudes d'Ingénieurs.*

*Dans les circonstances exceptionnelles liées à la crise sanitaire mondiale et nationale du Covid19, un ensemble de supports de révision pour le programme d'informatique du concours national d'entrée aux écoles d'ingénieurs ont été élaborés essentiellement par les membres de la commission informatique du concours, suite à une action bénévole.*

*Le but de ces supports est d'accompagner les étudiants de tout le territoire Tunisien dans leur révision en leur fournissant à tous les mêmes documents, afin d'assurer une équité entre eux, qu'ils bénéficient ou non d'un enseignement ou tutorat à distance.*

*Toutefois, ces documents sont considérés comme des aide-mémoires et ne remplacent pas les cours assurés en présentiel.*

*Le document présent comporte une description de quelques bibliothèques utiles à importer, comme numpy, scipy ou matplotlib, pour la résolution, entre autres, de problèmes mathématiques, simulation numérique ou traitement d'images. Des applications associées à des algorithmes de cryptage sont aussi exposées. Plusieurs exercices et leurs corrigés sont intégrés dans ce document.*

*Pour les différentes bibliothèques, quelques fonctions seulement parmi les plus utilisées sont décrites. Aussi, toutes les applications présentées sont données juste à titre d'exemple et ne constituent en aucun cas un ensemble complet.*

## Table des matières

|        |                                                                 |    |
|--------|-----------------------------------------------------------------|----|
| I      | Introduction .....                                              | 3  |
| II     | Les bibliothèques numpy, matplotlib et scipy .....              | 3  |
| II.1   | La bibliothèque numpy .....                                     | 3  |
| II.1.1 | Vecteurs .....                                                  | 4  |
| II.1.2 | Matrices .....                                                  | 5  |
| II.2   | La bibliothèque matplotlib .....                                | 8  |
| II.2.1 | Utilisation de la fonction plot .....                           | 9  |
| II.2.2 | Subdivision régulière d'un intervalle .....                     | 11 |
| II.3   | La bibliothèque scipy.....                                      | 12 |
| II.3.1 | Présentation.....                                               | 12 |
| II.3.2 | Le module optimize .....                                        | 12 |
| II.3.3 | Le module integrate .....                                       | 13 |
| III    | Application au Traitement d'images.....                         | 15 |
| III.1  | Présentation et définitions.....                                | 15 |
| III.2  | Traitement des images sous Python .....                         | 16 |
| III.3  | Exercice pour quelques traitements d'image .....                | 17 |
| IV     | Application à la cryptographie .....                            | 19 |
| IV.1   | Définition .....                                                | 19 |
| IV.2   | Le chiffrement de César.....                                    | 19 |
| IV.2.1 | Principe .....                                                  | 19 |
| IV.2.2 | Exercice : Chiffrement César .....                              | 20 |
| IV.3   | Le chiffrement de Vigenère .....                                | 20 |
| IV.3.1 | Principe .....                                                  | 20 |
| IV.3.2 | Exercice : Chiffrement de Vigenère.....                         | 21 |
| IV.4   | Le chiffrement RSA.....                                         | 21 |
| IV.4.1 | Principe .....                                                  | 21 |
| IV.4.2 | Exercice : Chiffrement RSA .....                                | 21 |
| V      | Autres Exercices d'application.....                             | 22 |
| V.1    | Méthodes de résolution d'un système linéaire .....              | 22 |
| V.2    | Méthodes de recherche de zéros d'une fonction.....              | 26 |
| VI     | Alternatives de corrections pour les exercices .....            | 29 |
| VI.1   | Corrigé des exercices de traitement d'image.....                | 29 |
| VI.2   | Corrigés des exercices sur les algorithmes de cryptage .....    | 32 |
| VI.3   | Corrigé des méthodes de résolution d'un système linéaire .....  | 36 |
| VI.4   | Corrigé des méthodes de recherche de zéros d'une fonction ..... | 38 |
|        | Bibliographie et Netographie .....                              | 42 |

## I Introduction

Python offre une panoplie de bibliothèques intéressantes pour, entre autres, la simulation numérique et le calcul scientifique. Dans ce but, nous exposerons dans ce qui suit les principales fonctionnalités de Numpy, Matplotlib et Scipy. Nous présenterons ensuite des applications liées au traitement d'image et à la cryptographie, en plus de plusieurs exercices d'application avec leurs corrigés.

## II Les bibliothèques numpy, matplotlib et scipy

### II.1 La bibliothèque numpy

**numpy** est une bibliothèque numérique apportant le support efficace de manipulation des tableaux (uni, bi et multi-dimensionnel) ainsi que les opérations mathématiques de haut niveau (fonctions spéciales, algèbre linéaire, statistiques, etc.). La bibliothèque **numpy** doit être importée avant toute utilisation.

Le module **numpy** contient entre autres la classe **numpy.ndarray**, proposant des outils similaires à la classe `list`, toutefois les tableaux modélisés sont de taille fixe et ayant des éléments de même type (entiers, flottants, booléens).

```
>>>import numpy as np
>>>a=np.array([1,4,5,8])
>>>b=np.array([[1,2],[3,4],[5,6]],dtype=float)
>>>c=np.array([[[1,2],[1,2]],[[1,2],[1,2]]])
>>>print(a)
[1 4 5 8]
>>>print(b)
[[ 1.  2.]
 [ 3.  4.]
 [ 5.  6.]]
>>>print(c)
[[[1 2]
 [1 2]]
 [[1 2]
 [1 2]]]
>>>c[1,1,1]
2
```

Nous limiterons notre propos aux tableaux uni et bi-dimensionnel appelés aussi vecteurs et matrices.

### II.1.1 Vecteurs

En Python la fonction **numpy.array** permet de créer un vecteur qui est un tableau à une dimension. Pour créer un vecteur il suffit de passer l'itérable contenant les éléments comme argument.

```
>>>v=np.array([-1,3.,0,-2,5,8,0,22,0,-5]) #l'itérable est une liste
>>>print(v)
[ -1.  3.  0. -2.  5.  8.  0. 22.  0. -5.]
```

Les fonctions **size** retourne le nombre d'éléments d'un vecteur et la fonction **ndim** retourne la dimension qui est 1 dans le cas d'un vecteur (2 pour une matrice).

```
>>>np.size(v); v.size #de façon équivalente
10
10
>>>np.ndim(v); v.ndim #de façon équivalente
1
1
```

L'accès à un élément ou une tranche d'éléments (slicing) se fait comme pour les listes.

```
>>>v[0]
-1.0
>>>v[len(v)-1]
-5.0
>>>v[2:];v[:3];v[4:8]
array([ 0., -2.,  5.,  8.,  0., 22.,  0., -5.])
array([-1.,  3.,  0.])
array([ 5.,  8.,  0., 22.])
>>>v[-1]
-5.0
```

Les fonctions **empty**, **zeros**, **ones**, **arange**, **linspace** permettent aussi la création de vecteurs particuliers.

| Fonctions       | Description                                                                                 | Syntaxe                         |
|-----------------|---------------------------------------------------------------------------------------------|---------------------------------|
| <b>Empty</b>    | Construit un vecteur non initialisé (le contenu est totalement imprédictible ou aléatoire). | <code>np.empty(taille)</code>   |
| <b>Zeros</b>    | Construit un vecteur dont les éléments sont nuls                                            | <code>np.zeros(taille)</code>   |
| <b>Ones</b>     | Construit un vecteur dont les éléments sont tous 1                                          | <code>np.ones(taille)</code>    |
| <b>Arange</b>   | Construit un vecteur à partir d'un intervalle d'entiers                                     | <code>np.arange(d,f,p)</code>   |
| <b>Linspace</b> | Construit un vecteur de points équidistant dans un intervalle                               | <code>np.linspace(a,b,n)</code> |

**Exemple**

```

>>>v1=np.linspace(-1,1,10)
>>>print(v1)
[-1. -0.77777778 -0.55555556 -0.33333333 -0.11111111  0.11111111
 0.33333333  0.55555556  0.77777778  1. ]
>>>v2=np.arange(10)
>>>print(v2)
[0 1 2 3 4 5 6 7 8 9]
>>>type(v2)
<type 'numpy.ndarray'>
>>>v3=np.arange(2,10,2)
>>>print(v3)
[2 4 6 8]
>>>v4=np.zeros(8);v5=np.ones(6,dtype=int)
>>>print(v4);print(v5)
[ 0.  0.  0.  0.  0.  0.  0.  0.]
[1 1 1 1 1 1]
1

```

Les opérations sur les vecteurs sont possibles avec les opérateurs classiques de Python, tels que décrit ci-dessous.

```

>>>u=np.array([1,2,3,4]);v=np.array([5,6,7,8])
>>>u+v
>>>v-u
>>>u*2
>>>v**3
>>>u*v
>>>v/u

```

**II.1.2 Matrices**

Pour créer une matrice en Python, qui est un tableau à deux dimensions, il suffit de passer un itérable d'itérables (liste de listes, liste de tuples, tuple de tuples, etc.) comme argument à la fonction **array**.

```

>>>m=np.array([[1.2,2.5],[3.2,1.8],[1.1,4.3]]) # une liste de listes
>>>print(m)
[[ 1.2  2.5]
 [ 3.2  1.8]
 [ 1.1  4.3]]

```

Les fonctions **size**, **shape** et **ndim** retournent respectivement le nombre d'élément, nombre de lignes/colonnes et la dimension. Par exemple :

```
>>>np.size(m);np.shape(m);np.ndim(m) #ou bien les attributs m.size;
m.shape ; m.ndim
6
(3, 2)
2
1
```

La fonction **reshape** permet de redimensionner un vecteur ou une matrice, comme décrit ci-après :

```
>>>m1=np.reshape(np.arange(10),(2,5))
#ou bien m1 = np.arange(10).reshape((2,5))
>>>print(m1)
[[0 1 2 3 4]
 [5 6 7 8 9]]
>>>m.reshape(2,3)
array([[ 1.2, 2.5, 3.2],
 [ 1.8, 1.1, 4.3]])
```

Les fonctions **empty**, **zeros**, **ones**, **eye** et **diag** permettent de créer des matrices particulières.

| Fonctions    | Description                                                    | Syntaxe                      |
|--------------|----------------------------------------------------------------|------------------------------|
| <b>empty</b> | Construit une matrice non initialisée (l lignes et c colonnes) | <code>np.empty((l,c))</code> |
| <b>zeros</b> | Construit une matrice nulle                                    | <code>np.zeros((l,c))</code> |
| <b>ones</b>  | Construit une matrice dont les éléments sont tous 1            | <code>np.ones((l,c))</code>  |
| <b>eye</b>   | Construit une matrice identité d'ordre n                       | <code>np.eye(n)</code>       |
| <b>diag</b>  | Construit une matrice diagonale                                | <code>np.diag(L)</code>      |

### Exemple

```
>>>m2=np.zeros((2,4));m3=np.ones((3,3),dtype=int);
>>>m4=np.eye(4); m5=np.diag([1,2,3])
>>> print('m2=',m2);print('m3=',m3);print('m4=',m4);print('m5=',m5)
m2= [[0. 0. 0. 0.]
 [0. 0. 0. 0.]]
m3= [[1 1 1]
 [1 1 1]
 [1 1 1]]
m4= [[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
```

```
[0. 0. 0. 1.]
m5= [[1 0 0]
      [0 2 0]
      [0 0 3]]
```

L'accès à un élément d'une matrice se fait à l'aide d'indexage  $A[i, j]$  où  $i$  désigne la ligne et  $j$  la colonne. On peut également récupérer une partie d'une matrice : ligne, colonne et sous-matrice. Par exemple :

```
>>>A=np.array([[1,2,3],[4,5,6],[7,8,9]],dtype=int)
>>>A[1,0];A[2,2]
4
9
>>>A[0,:];A[:,1]
array([1, 2, 3])
array([2, 5, 8])
>>>A[0:1,:];A[:,1:2]
array([[1, 2, 3]])
array([[2],
       [5],
       [8]])
```

Les **opérations algébriques** sur les matrices sont possibles avec les opérateurs classiques de Python.

```
>>>A+A
array([[ 2,  4,  6],
       [ 8, 10, 12],
       [14, 16, 18]])
>>>A-A
array([[0, 0, 0],
       [0, 0, 0],
       [0, 0, 0]])
>>>A*2
array([[ 2,  4,  6],
       [ 8, 10, 12],
       [14, 16, 18]])
```

Pour le produit matriciel, il faut utiliser la fonction **dot**, comme l'exemple suivant :

```
>>>np.dot(A,A) # ou bien A.dot(A)
array([[ 30,  36,  42],
       [ 66,  81,  96],
       [102, 126, 150]])
```

La trace et la transposé s'obtiennent avec les fonctions **trace** et **transpose**. Par exemple :

```
>>>np.trace(A) # ou bien A.trace()
15
>>> A.transpose() # ou bien A.T
array([[1, 4, 7],
       [2, 5, 8],
       [3, 6, 9]])
```

Le déterminant, le rang, l'inverse, les valeurs propres et les vecteurs propres s'obtiennent respectivement par les fonctions **det**, **matrix\_rank**, **inv**, **eigvals** et **eig** du module **numpy.linalg**. Par exemple :

```
>>>np.linalg.det(A)
6.66133814775e-16
>>>np.linalg.matrix_rank(A)
2
>>>np.linalg.inv(A)
[[ -4.50359963e+15  9.00719925e+15 -4.50359963e+15]
 [ 9.00719925e+15 -1.80143985e+16  9.00719925e+15]
 [-4.50359963e+15  9.00719925e+15 -4.50359963e+15]]
>>>np.linalg.eigvals(A)
[ 1.61168440e+01 -1.11684397e+00 -1.30367773e-15]
>>>np.linalg.eig(A)
(array([ 1.61168440e+01, -1.11684397e+00, -1.30367773e-15]),
 array([[ -0.23197069, -0.78583024,  0.40824829],
        [-0.52532209, -0.08675134, -0.81649658],
        [-0.8186735 ,  0.61232756,  0.40824829]]))
```

Pour finir, la fonction **solve** du module **numpy.linalg** permet de résoudre le système linéaire  $A \cdot x = b$

```
>>>print(np.linalg.solve(A,np.array([1,0,1])))
[ -9.00719925e+15  1.80143985e+16 -9.00719925e+15]
```

## II.2 La bibliothèque matplotlib

Le module **matplotlib.pyplot** est un sous module de la bibliothèque **matplotlib** qui fournit des outils de traçages graphiques qu'on peut paramétrer, semblables à ceux du logiciel Matlab.

Pour l'utiliser, on procédera toujours de la manière suivante :

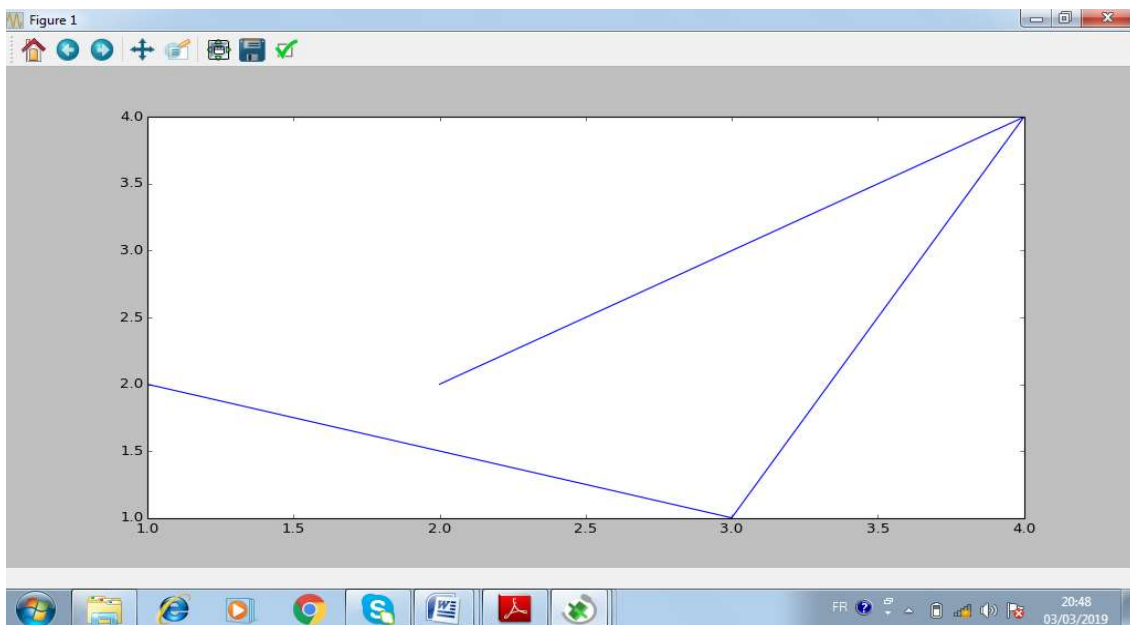
- Appeler la fonction **clf()** (clean figure) pour effacer la figure précédente et en commencer une nouvelle.
- Tracer des courbes, nuages de points, en appelant (éventuellement plusieurs fois) la fonction **plot()**.
- On remarquera que rien n'est affiché, seul un objet est créé par Python, d'où l'intérêt du paramétrage.
- Paramétrer le graphique (axes, titre, légendes etc.) en utilisant les fonctions adéquates.
- Afficher le graphique avec la fonction **show()** ou le sauvegarder avec la fonction **savefig()**. On dispose alors d'une image (différents formats possibles).

### II.2.1 Utilisation de la fonction plot

Cette fonction peut être utilisée de nombreuses manières différentes, pour obtenir des résultats assez fins. L'idée générale est de lui donner en argument deux tableaux de nombres, décrivant respectivement les abscisses et les ordonnées des points à tracer. Sans autre option, ces points sont reliés par des segments bleus, en traits pleins.

#### Exemple 1

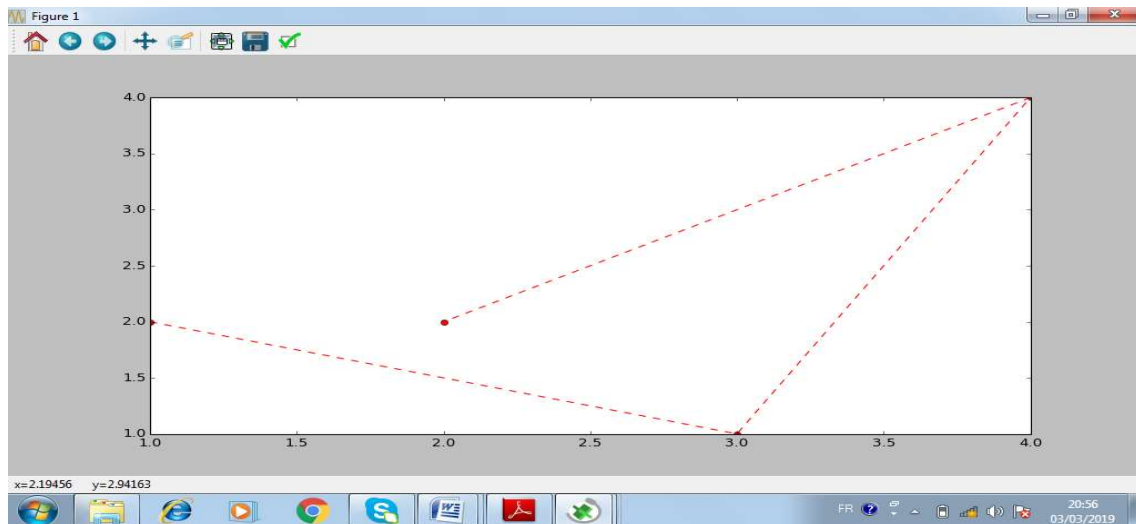
```
import matplotlib.pyplot as plt
x = [1, 3, 4, 2]
y = [2, 1, 4, 2]
plt.clf()
plt.plot(x,y)
plt.show()
plt.savefig("ex_base_01.png")
```



Les options graphiques permettent notamment de modifier le type de segment (pointillés, sans segment), d'afficher ou non un marqueur au niveau de chaque point et de changer la couleur de ces objets.

#### Exemple 2

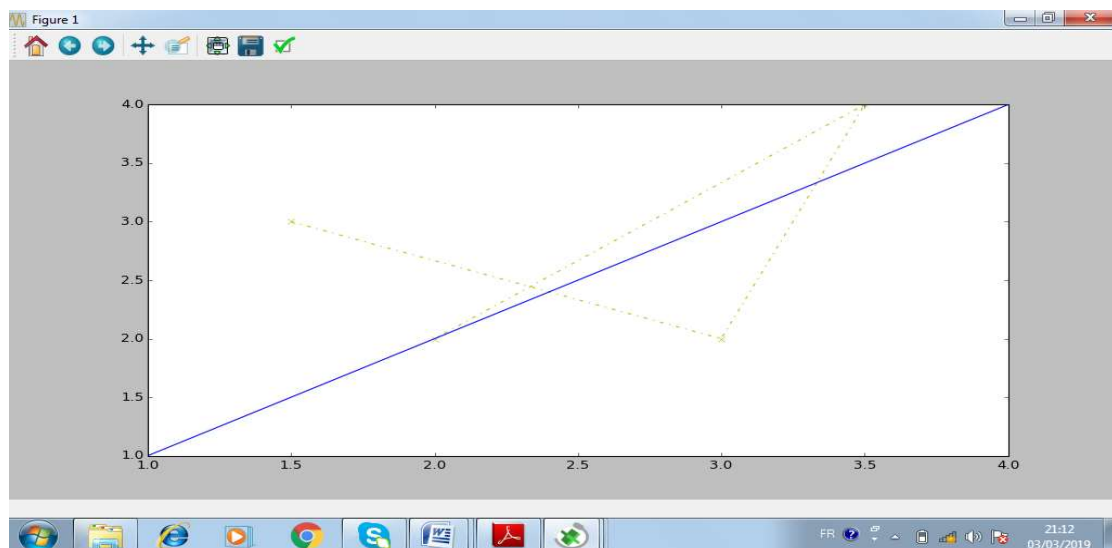
```
import matplotlib.pyplot as plt
x = [1, 3, 4, 2]
y = [2, 1, 4, 2]
plt.clf()
plt.plot(x,y,marker='o',color='r',linestyle='--')
plt.show()
plt.savefig("ex_base_02.png")
```



On peut enfin appeler plusieurs fois la fonction **plot** successivement. Les tracés sont alors superposés.

### Exemple 3

```
import matplotlib.pyplot as plt
x = [1.5, 3, 3.5, 2]
y = [3, 2, 4, 2]
plt.clf()
plt.plot(x,y,marker='x',color='y',linestyle='-.')
plt.plot([1,4],[1,4])
plt.show()
plt.savefig("ex_base_03.png")
```



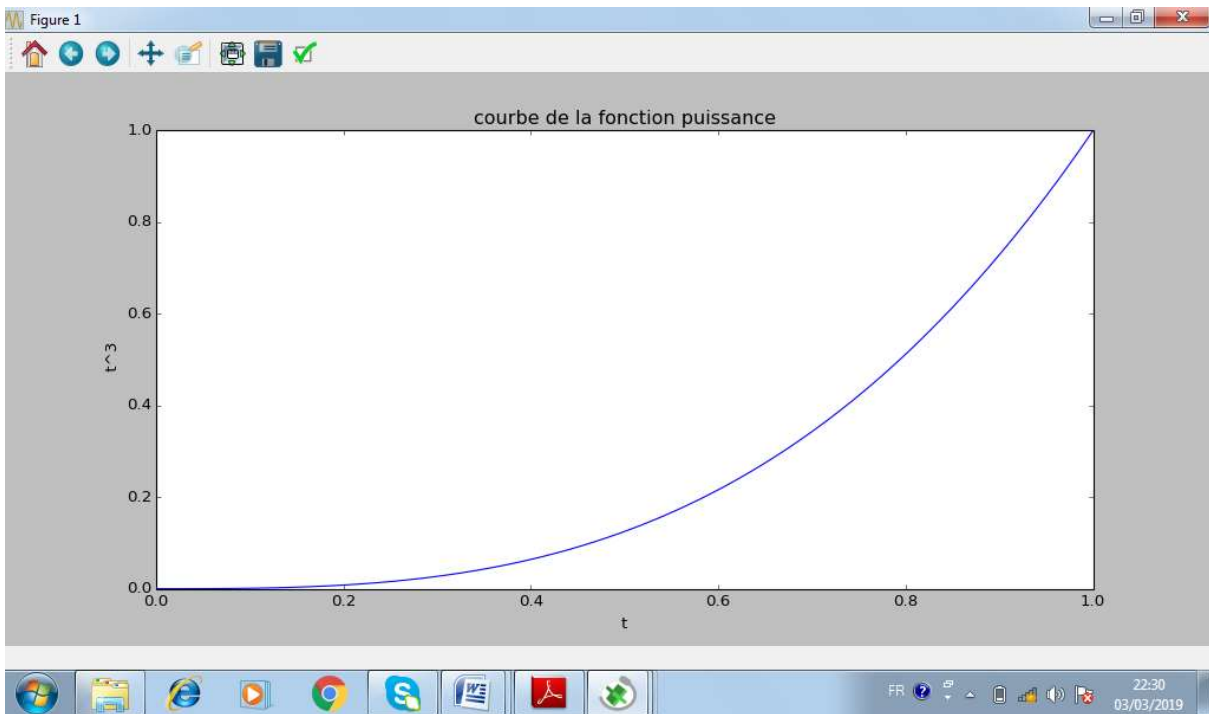
### II.2.2 Subdivision régulière d'un intervalle

Pour tracer une fonction, on trace en fait des cordes de cette fonction, à partir d'une subdivision de l'intervalle considéré. La fonction `linspace()` de la bibliothèque **numpy** permet de créer automatiquement de telles subdivisions, avec un nombre de points à régler manuellement.

#### Exemple

Pour tracer le graphe de la fonction  $x \mapsto x^3$  sur  $[0,1]$  avec une subdivision en 100 segments, donc en utilisant 101 points, on peut utiliser le code suivant :

```
import matplotlib.pyplot as plt
from numpy import linspace
x = linspace(0,1,100)
y = [t**3 for t in x]
plt.clf()
plt.plot(x,y)
plt.xlabel("t")
plt.ylabel(" t^3 ")
plt.title("courbe de la fonction puissance")
plt.show()
plt.savefig('ex_numpy_01.png')
```



## II.3 La bibliothèque scipy

### II.3.1 Présentation

**Scipy** est une bibliothèque numérique d'algorithmes et de fonctions mathématiques, basée sur les tableaux, complétant ou améliorant (en termes de performance) les fonctionnalités de la bibliothèque **numpy**.

La librairie **Scipy** contient de nombreuses boîtes à outils consacrées aux méthodes de calcul scientifique :

- Fonctions spéciales : **scipy.special** (fonctions de Bessel, erf, gamma, etc.)
- Intégration numérique : **scipy.integrate** (intégration numérique ou d'équations différentielles)
- Méthodes d'optimisation : **scipy.optimize** (minimisation, moindres-carrées, zéros d'une fonction)
- Interpolation : **scipy.interpolate**
- Transformées de Fourier : **scipy.fftpack**
- Traitement de signal : **scipy.signal** (convolution, corrélation, filtrage, ondelettes, etc.)
- Algèbre linéaire : **scipy.linalg**
- Statistiques : **scipy.stats** (fonctions et distribution statistiques)
- Traitement d'images multidimensionnelles : **scipy.ndimage**
- Entrées/Sorties : **scipy.io**

### II.3.2 Le module optimize

Ce module propose des fonctions qui permettent de déterminer des valeurs approchées des zéros d'une fonction :

- **fsolve** : déterminer la racine d'une équation scalaire ou vectorielle.
- **bisect** : permet d'implémenter la méthode de recherche par dichotomie.
- **newton** : permet d'implémenter la méthode de recherche newton.

#### Exemple

```
import scipy.optimize as spo
f = lambda x : x**2-2
a,b=1,2
x1=spo.fsolve(f,a)
x2=spo.bisect(f,a,b)
x3=spo.newton(f,a)
>>>x1
array([ 1.41421356])
>>>x2
1.4142135623724243
>>>x3
1.4142135623730947
```

### II.3.3 Le module integrate

L'extension **scipy.integrate** propose des fonctions pour calculer des intégrales simples, doubles ou même triples et aussi de résoudre des équations différentielles de premier ou second ordre.

- **quad** : permet de calculer la valeur d'une intégrale simple.
- **trapz** : implémente la méthode des trapèzes.
- **sims** : implémente la méthode de simpson.
- **odeint** : résout numériquement des équations différentielles.

#### Exemple

```
import math
import scipy.integrate as spi
h = lambda x:math.cos(x)
y=spi.quad(h,0.0,math.pi/2.0)
>>>y
(0.9999999999999999, 1.1102230246251564e-14)
```

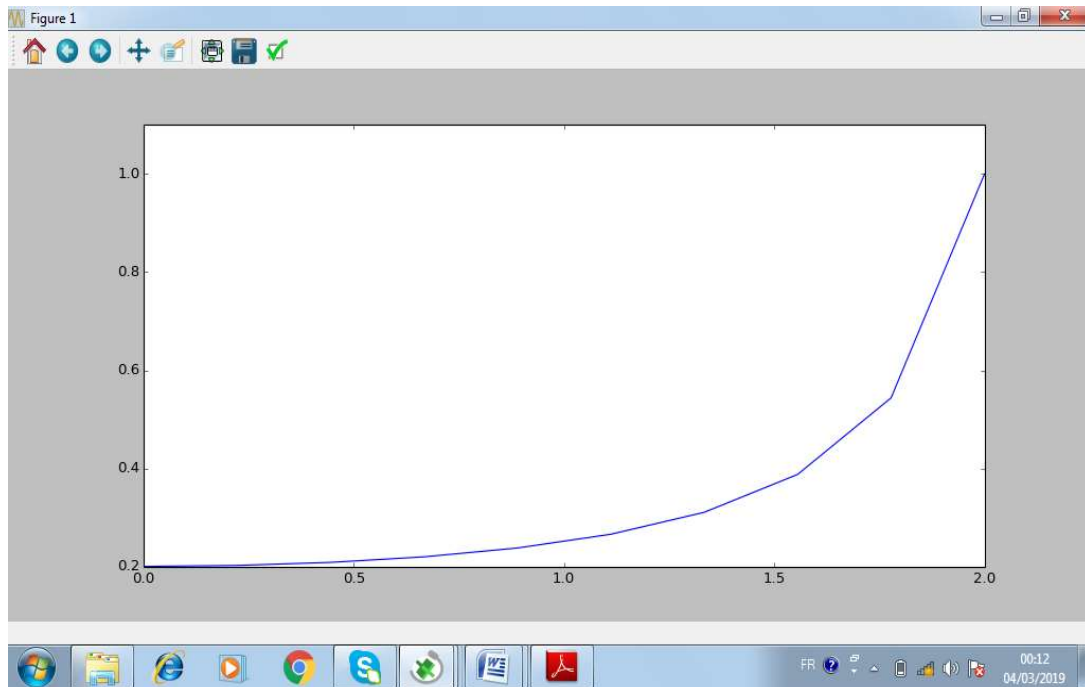
La fonction **odeint** résout numériquement des équations différentielles avec des conditions initiales de la forme :  $\frac{\partial y}{\partial t} = f(y(t), t)$

**Exemple** : on souhaite résoudre l'équation  $\frac{\partial y}{\partial t} = 2ty^2$  avec  $t \in [0,2]$  et  $y(0) = 0,2$

```
from numpy import linspace
import scipy.integrate as spi
k = lambda y,t:2*t*y**2
t = linspace(0,2,10)
y0 = 0.2
y = spi.odeint(k,y0,t)
>>>y
array([[ 0.2          ],
       [ 0.20199509 ],
       [ 0.20822631 ],
       [ 0.2195123  ],
       [ 0.23753682 ],
       [ 0.26557403 ],
       [ 0.31034526 ],
       [ 0.38756059 ],
       [ 0.5436259  ],
       [ 1.00000647 ]])
```

La représentation graphique des solutions de cette résolution peut être donnée par :

```
import matplotlib.pyplot as plt  
plt.plot(t,y)  
plt.show()
```



## III Application au Traitement d'images

### III.1 Présentation et définitions

Les appareils photographiques numériques sont actuellement omniprésents via des supports variés comme les téléphones portables ou les tablettes numériques. Ils permettent le stockage d'un grand nombre d'images et ils proposent également des options de retouches.

La suite présente quelques aspects du traitement mathématique et informatique des images numériques.

#### ❖ Pixel :

Le terme pixel provient de l'abréviation anglaise « PiCture ELement » qui signifie élément d'image. Un pixel est l'unité de base qui permet de mesurer la définition d'une image numérique.

#### ❖ Image numérique :

Une image numérique est une matrice de  $n$  lignes et  $p$  colonnes dont les coefficients sont les valeurs des pixels. Donc, il est possible d'avoir facilement accès à la grille des pixels et de faire des traitements matriciels directs sur les images.

#### ❖ Image en niveau de gris :

Dans le cas des images numériques en niveaux de gris, la valeur du pixel est généralement enregistrée dans l'ordinateur ou l'appareil photo numérique sous forme de nombres entiers entre 0 et 255 soit 256 valeurs possibles pour chaque pixel. La valeur 0 est attribué au noir, 255 correspond au blanc et les pixels entre 0 et 255 représentent les niveaux de gris (ou nuances de gris).

#### ❖ Image couleur :

Une image couleur est en fait la réunion de trois images : une rouge, une verte et une bleue.

Chaque pixel est alors représenté par un triplet (*Red, Green, Blue*), chacun étant un entier codé sur 8 bits.

**Exemple :** (255,0,0) correspond à un pixel rouge, (0,255,0) représente un pixel vert, (0,0,255) est le pixel bleu. Par additivité (255,0,255) devient le pixel magenta.

**Remarque :** Sur une image en niveaux de gris, chaque pixel est ou bien noir (0) ou bien blanc (255) ou bien possède un niveau de gris entre les deux. Cela signifie que les trois composantes R, G, B ont la même valeur.

Si  $R = G = B = 0$ , le pixel est noir.

Si  $R = G = B = 255$  (valeur maximale), le pixel est blanc.

Dans la suite de cette partie, les traitements ne s'appliquent qu'à des images en niveaux de gris, il est possible d'opérer sur des images couleurs en appliquant les traitements proposés sur chacune des trois couleurs du triplet.

## III.2 Traitement des images sous Python

Python propose plusieurs modules qui permettent de faire des traitements sur des images de différents formats (.jpg, .jpeg, .png, bmp ...) comme le module **PIL** ou les module **ndimage** et **misc** de la bibliothèque **scipy** ou le module **pyplot** de la bibliothèque **matplotlib**.

Dans la suite, on propose l'utilisation du module **pyplot** de la bibliothèque **matplotlib** qui permet de charger une image directement sous forme d'un tableau **numpy** (de type **ndarray**).

### ❖ Lecture d'une image :

La fonction **imread** permet d'interpréter un fichier image comme un tableau **numpy**.

```
>>>import numpy as np
>>>import matplotlib.pyplot as plt
>>>M=plt.imread("mon_image.png")
```

### ❖ Affichage d'une image :

La fonction **imshow** permet d'afficher une image (si c'est un tableau **numpy**).

```
>>>M1=M[50:101 , 50:101]
>>>plt.imshow(M1)
>>>plt.show()
```

### ❖ Sauvegarde des images :

La fonction **imsave** permet de sauvegarder une image.

```
>>>plt.imsave("sous_image.png")
```

### ❖ Création d'une image :

Créer une image revient à construire un tableau **numpy** puis le représenter sous forme d'image à l'aide du module **pyplot** de **matplotlib**.

```
>>>import numpy as np
>>>import matplotlib.pyplot as plt
>>>check=np.zeros((8,8)) #exemple pour image en niveau de gris
>>>check[:,1::2]=255
>>>check[1::2,::2]=255
>>>plt.imshow(check)
>>>plt.show()
>>>plt.imsave("echiquier.png")
```

```
>>>import numpy as np
>>>import matplotlib.pyplot as plt
>>>check=np.zeros((8,8,3),dtype='uint8') #exemple pour image couleur
>>>check[:,1::2,::2]=255
>>>plt.imshow(check)
>>>plt.show()
```

### III.3 Exercice pour quelques traitements d'image

#### ❖ Binarisation :

Binariser une image consiste à la partitionner en deux parties : une partie sombre et une partie claire selon un seuil  $s$  de niveau de gris.

1. Ecrire le script de la fonction Python permettant la binarisation d'une image donnée.

#### ❖ Contraste :

Le contraste peut être modifié par l'application de fonctions mathématiques. Par exemple, en remplaçant la valeur de chaque pixel par sa racine carrée, l'image assombrie alors qu'elle s'éclaircit si la valeur du pixel est remplacée par son carré. Ces fonctions ont pour effet de diminuer le contraste.

Une autre idée consiste à faire un réajustement linéaire des valeurs des pixels selon :

$$f: [0,256[ \rightarrow [0,256[$$

$$x \mapsto \begin{cases} y = x + 0,4 \cdot (x - 127) & \text{si } y \in [0,255] \\ 0 & \text{si } y < 0 \\ 255 & \text{si } y > 255 \end{cases}$$

Il s'agit d'augmenter l'écart entre la valeur du pixel et la valeur moyenne (127) pour une image dont les valeurs sont comprises entre 0 et 255. Cette fonction permet d'augmenter le contraste, les "noirs" sont plus noirs et les "blancs" plus blancs.

2. Ecrire le script de la fonction Python permettant de modifier le contraste d'une image donnée.

#### ❖ Négatif d'une image :

Une image négative est une image dont les niveaux de gris sont inversés par rapport à l'originale. La valeur  $x$  de chaque pixel est remplacée par  $255 - x$ .

3. Ecrire le script de la fonction Python permettant de déterminer le négatif d'une image donnée.

#### ❖ Transformer une image couleur en une image en niveau de gris :

Dans les images en niveaux de gris, les trois niveaux de couleur sont égaux. On remplace donc Rouge, Vert et Bleu par la moyenne pondérée des trois couleurs donnée par :  $0,21 \times R + 0,71 \times V + 0,07 \times B$

4. Ecrire le script de la fonction Python permettant de transformer une image couleur donnée en une image en niveau de gris.

#### ❖ Filtrage (enlever le bruit)

Les images sont parfois de mauvaise qualité. Un exemple typique de défaut est le bruit qui apparaît quand une photo est sous-exposée, c'est-à-dire qu'il n'y a pas assez de luminosité. Ce bruit se manifeste par de petites fluctuations aléatoires des niveaux de gris.

Afin d'enlever le bruit dans les images, il convient de faire une modification aux valeurs de pixels. L'opération la plus simple consiste à remplacer la valeur  $a$  de chaque pixel par la moyenne de  $a$  et des 8 valeurs  $b, c, d, e, f, g, h, i$  des 8 pixels qui entourent  $a$ .

On obtient ainsi une image modifiée en remplaçant  $a$  par :  $\frac{a+b+c+d+e+f+g+h+i}{9}$

5. Ecrire le script de la fonction Python permettant de filtrer une image donnée en utilisant la méthode de moyennage.

En effectuant cette opération pour chaque pixel, on supprime une partie du bruit, car ce bruit est constitué de fluctuations aléatoires, qui sont diminuées par un calcul de moyennes.

Le moyennage des pixels est très efficace pour enlever le bruit dans les images, malheureusement il détruit également une grande partie de l'information de l'image. On peut en effet s'apercevoir que les images obtenues par moyennage sont floues. Ceci est en particulier visible près des contours, qui ne sont pas nets.

Afin de réduire ce flou, il faut remplacer le moyennage par une opération un peu plus complexe, que l'on nomme médiane. Etant donné la valeur  $a$  d'un pixel, et les valeurs  $b, c, d, e, f, g, h, i$ , on commence par les classer par ordre croissant.

La médiane des neuf valeurs  $a, b, c, d, e, f, g, h, i$  est la 5<sup>ème</sup> valeur de ce classement (c'est-à-dire la valeur centrale de ce classement).

6. Ecrire le script de la fonction Python permettant de filtrer une image donnée en utilisant la méthode de médiane.

#### ❖ Détection de contours :

Pour localiser des objets dans les images, il est nécessaire de détecter les bords de ces objets. Ces bords correspondent à des zones de l'image où les valeurs des pixels changent rapidement.

Afin de savoir si un pixel avec une valeur  $a$  est le long d'un bord d'un objet, on prend en compte les valeurs  $b, c, d, e$  de ses quatre voisins (deux horizontalement et deux verticalement), qui sont disposés par rapport à  $a$ . Notons que l'on utilise ici seulement les 4 voisins qui ont un coté commun avec le pixel considéré, ce qui est différent du calcul de moyennes et de médianes où l'on utilisait 8 voisins. Ceci est important afin de détecter aussi précisément que possible les bords des objets.

On calcule une valeur  $l$  suivant la formule :  $l = \sqrt{(b - d)^2 + (c - e)^2}$

On peut remarquer que si  $l = 0$ , alors on a :  $b = d$  et  $c = e$ . Au contraire, si  $l$  est grand, ceci signifie que les pixels voisins ont des valeurs très différentes, le pixel considéré est donc probablement sur le bord d'un objet.

On affiche ainsi ces valeurs avec du noir quand  $l = 0$ , du blanc quand  $l$  est grand, et on utilise des niveaux de gris pour les valeurs intermédiaires.

7. Ecrire le script de la fonction Python permettant de détecter les contours des objets d'une image donnée.

## IV Application à la cryptographie

### IV.1 Définition

La cryptographie est une des disciplines de la cryptologie s'attachant à protéger des messages (assurant confidentialité, authenticité et intégrité) en s'aidant souvent de secrets ou clés. Elle se distingue de la stéganographie qui fait passer inaperçu un message dans un autre message alors que la cryptographie rend un message inintelligible à autre que qui-de-droit. Elle est utilisée depuis l'Antiquité, mais certaines de ses méthodes les plus importantes, comme la cryptographie asymétrique, datent de la fin du XXe siècle.

Le chiffrement est la conversion des données d'un format lisible à un format codé qui peut uniquement être lu ou traité qu'après déchiffrement.

Les principes du chiffrement se basent sur la notion d'algorithmes de chiffrement et de « clés ». Lorsque l'information est envoyée, elle est chiffrée à l'aide d'un algorithme et peut être décodée uniquement à l'aide de la clé appropriée.

Quelques méthodes qui sont utilisées pour coder et décoder les informations sont présentées dans la suite.

### IV.2 Le chiffrement de César

#### IV.2.1 Principe

Jules César a-t-il vraiment prononcé la célèbre phrase :

**DOHD MDFWD HVW**

ou bien comme le disent deux célèbres Gaulois : « Ils sont fous ces romains ! ».

En fait César, pour ses communications importantes à son armée, cryptait ses messages (ce que l'on appelle le chiffrement de César). Le chiffrement de César est un décalage des lettres : pour crypter un message, *A* devient *D*, *B* devient *E*, *C* devient *F*,...

$$A \leftrightarrow D \quad B \leftrightarrow E \quad C \leftrightarrow F \quad \dots \quad W \leftrightarrow Z \quad X \leftrightarrow A \quad Y \leftrightarrow B \quad Z \leftrightarrow C$$

Pour déchiffrer le message de César, il suffit de décaler les lettres dans l'autre sens, *D* se déchiffre en *A*, *E* en *B*,...

Pour ce faire, nous associons à chacune des 26 lettres majuscules (ou minuscules) de *A* à *Z* (de *a* à *z*) un entier de 0 à 25.

Le chiffrement de César est simplement une addition dans  $\mathbb{Z}/26\mathbb{Z}$ . Fixons un entier *k* qui est le décalage (par exemple *k* = 3 dans l'exemple de César ci-dessus) et définissons la fonction de chiffrement de César de décalage *k* qui va de l'ensemble  $\mathbb{Z}/26\mathbb{Z}$  dans lui-même :

$$C_k: \begin{cases} \mathbb{Z}/26\mathbb{Z} \rightarrow \mathbb{Z}/26\mathbb{Z} \\ x \mapsto x + k \end{cases}$$

Pour déchiffrer, il suffit d'aller dans l'autre sens, c'est-à-dire ici de soustraire. La fonction de déchiffrement de César de décalage  $k$  est :

$$D_k: \begin{cases} \mathbb{Z}/26\mathbb{Z} \rightarrow \mathbb{Z}/26\mathbb{Z} \\ x \mapsto x - k \end{cases}$$

### Exemple

FLEN veut envoyer des messages secrets à FELTEN. Ils se sont d'abord mis d'accord sur une clé secrète  $k$ , par exemple  $k = 11$ . Alice veut envoyer le message "COUCOU" (ou "coucou") à Bruno. Elle transforme "COUCOU" (ou "coucou") en "2 14 20 2 14 20". Elle applique la fonction de chiffrement  $C_{11}(x) = x + 11$  à chacun des nombres qui donne "13 25 5 13 25 5" ce qui correspond au mot crypté "NZFNZF" (ou "nzfnzf"). Elle transmet le mot crypté à Bruno, qui selon le même principe applique la fonction de déchiffrement  $D_{11}(x) = x - 11$  afin de récupérer le message de départ.

### IV.2.2 Exercice : Chiffrement César

1. Ecrire une fonction nommée **C** qui définit la fonction de chiffrement de César.
2. Ecrire une fonction nommée **D** qui définit la fonction de déchiffrement de César.
3. Ecrire une fonction nommée **chiffreCesar** permettant de chiffrer un **message** clair donné avec une clé **clef** donnée.
4. Ecrire une fonction nommée **dechiffreCesar** permettant de déchiffrer un **message** crypté donné avec une clé **clef** donnée.
5. Ecrire une fonction nommée **attaqueCesar** permettant d'attaquer un **message** chiffré donné en affichant toutes les possibilités de déchiffrement.

## IV.3 Le chiffrement de Vigenère

### IV.3.1 Principe

Le chiffrement de Vigenère est un système de chiffrement poly-alphabétique, c'est un chiffrement par substitution, mais une même lettre du message clair peut, suivant sa position dans celui-ci, être remplacée par des lettres différentes.

On dispose d'une clé qui est une chaîne de caractère (en général un mot), et on décale la première lettre du texte à crypter d'un rang correspondant à celui de la première lettre de la clé dans l'alphabet, on décale la deuxième lettre du texte à crypter d'un rang correspondant à celui de la deuxième lettre de la clé dans l'alphabet et ainsi de suite jusqu'à la dernière lettre du texte. On revient au début de la clé si le texte à crypter est plus long que la clé.

### Exemple

Soit à crypter le texte : J'adore écouter la radio toute la journée, avec la clé suivante : MUSIQUE

Texte en clair : J'adore écouter la radio toute la journée

Clé répétée : M USIQU EMUSIQU EM USIQU EMUSI QU EMUSIQU

Rang : 12 20 18 8 16 20 4 12 20 18 8 16 20 4 12 20 18 8 16 20 4 12 20 18 8 16 20 4 12 20 18 8 16 20

La lettre J est décalée de 12 ce qui donne V

La lettre a est décalée de 20 ce qui donne u Etc.

Texte chiffré : V'uvwhy ioimbul pm lslyi xaolm bu naojvuy

### IV.3.2 Exercice : Chiffrement de Vigenère

Ecrire une fonction nommée **chiffreVigenere** permettant de chiffrer un **message** clair donné avec une clé **clef** donnée en utilisant le chiffrement de Vigenère.

## IV.4 Le chiffrement RSA

### IV.4.1 Principe

Le **chiffrement RSA** (nommé par les initiales de ses trois inventeurs) est un algorithme de cryptographie asymétrique, très utilisé dans le commerce électronique et plus généralement pour échanger des données confidentielles sur Internet. Cet algorithme a été décrit en 1977 par **Rivest, Shamir et Adelman**.

Le cryptage/décryptage RSA est fondé sur l'utilisation d'une paire de clés composée d'une clé publique pour crypter et d'une clé privée pour décrypter des données confidentielles.

Chaque clé est un couple d'entiers déterminé comme suit :

- choisir deux grands nombres premiers  $p$  et  $q$  puis on pose  $n = p * q$  ;

La création des clés s'effectue de la manière suivante. Soient  $p$  et  $q$  deux grands nombres premiers.

- calculer l'indicatrice  $z = (p - 1) * (q - 1)$  ;
- calculer  $n = p * q$  ;
- choisir un grand entier  $c$ , premier avec  $z$  ;
- calculer l'entier  $d = c^{(-1)} \bmod z$ . (notons que l'inverse modulaire est calculé par l'algorithme d'Euclide étendu)

La clé publique est le tuple  $(n, c)$  et la clé privée est le tuple  $(n, d)$

L'entier crypté  $y$  associé à un entier  $x$  est donné par :  $y = x^c \bmod n$ .

L'entier  $x$  est retrouvé en décryptant l'entier  $y$  par :  $x = y^d \bmod n$ .

### IV.4.2 Exercice : Chiffrement RSA

On suppose avoir déjà défini la fonction python **invmod** qui calcule l'inverse modulaire de deux entiers premiers entre eux avec l'algorithme d'Euclide étendu.

```
def invmod(n,p):
    n1,p1,u1,u2)=(n,p,1,0)
    while p1>0:
        q=n1/p1
        (n1,p1,u1,u2)=(p1,n1-q*p1,u2,u1-q*u2)
    return u1%p

>>> invmod(39,53)
34
```

**PARTIE A : cryptage et décryptage d'entier(s)**

1. Ecrire une fonction python, nommée **clefRSA**, qui prend en entrée deux grands nombres **p** et **q** (supposés premiers) et retourne les deux clés (publique et privée).
2. Ecrire une fonction python, nommée **cryptDecryptE**, qui prend un entier **a** et une clé **CL** et retourne l'entier crypté (ou décrypté) associé à **a**.
3. Ecrire une fonction python, nommée **cryptDecryptList**, qui prend une liste d'entiers **L** et une clé **CL** et retourne la liste image de **L** par **cryptDecryptE**.

**PARTIE B : cryptage et décryptage d'une chaîne de caractères**

Le cryptage/décryptage d'une chaîne de caractères consiste à crypter/décrypter les codes ASCII correspondants aux caractères de la chaîne.

4. Ecrire une fonction python, nommée **codage**, qui prend une chaîne **CH** et retourne la liste des codes ASCII correspondants.
5. Ecrire une fonction python, nommée **decodage**, qui prend une liste d'entiers **L** (codes ASCII) et retourne la chaîne de caractères correspondante.
6. Ecrire une fonction python, nommée **cryptageC**, qui prend une chaîne **CH** et une clé **CL** et retourne la liste des entiers cryptés associés aux codes ASCII de **CH**.
7. Ecrire une fonction python, nommée **decryptageC**, qui prend une liste d'entiers cryptés **L** et une clé **CL** et retourne la chaîne claire.

**V Autres Exercices d'application****V.1 Méthodes de résolution d'un système linéaire**

On se propose de programmer quelques méthodes permettant de résoudre les systèmes de **n** équations à **n** inconnues de la forme :

$$\begin{cases} a_{0,0}x_0 + a_{0,1}x_1 + \cdots \cdots \cdots + a_{0,n-1}x_{n-1} = b_0 \\ a_{1,0}x_0 + a_{1,1}x_1 + \cdots \cdots \cdots + a_{1,n-1}x_{n-1} = b_1 \\ \vdots \\ a_{n-1,0}x_0 + a_{n-1,1}x_1 + \cdots \cdots \cdots + a_{n-1,n-1}x_{n-1} = b_{n-1} \end{cases}$$

Nous noterons ce système sous la forme matricielle :  $A \cdot x = b$

Où  $A$  désigne la matrice des éléments  $a_{ij}$  avec  $0 \leq i \leq n-1, 0 \leq j \leq n-1$  ;  $b$  étant le vecteur des composantes  $(b_0, b_1, \dots, b_{n-1})$  et  $x$  le vecteur des composantes  $(x_0, x_1, \dots, x_{n-1})$ .

**Partie I : Utilisation de la formule de CRAMER pour la résolution du système**

Dans cette méthode, et pour chaque composante  $x_i$  du vecteur  $x$ , on construit une nouvelle matrice que l'on obtient en remplaçant les composantes de la colonne  $i$  de la matrice  $A$  par les composantes du vecteur  $b$ . la composante  $x_i$  est calculée par division du déterminant de cette nouvelle matrice par celui de  $A$ . (Il est nécessaire que le déterminant de  $A$  ne soit pas nul)

$$x_i = \frac{\begin{vmatrix} a_{0,0} & \dots & a_{0,j-1} & b_0 & a_{0,j+1} & \dots & a_{0,n-1} \\ \vdots & & \vdots & & \vdots & & \vdots \\ a_{n-1,0} & \dots & a_{n-1,j-1} & b_{n-1} & a_{n-1,j+1} & \dots & a_{n-1,n-1} \end{vmatrix}}{\det(A)}$$

- 1) Ecrire une fonction python, nommée **METHODE1**, qui permet de résoudre le système d'équations ( $Ax = b$ ) par l'utilisation de la formule de **CRAMER**.

## Partie II : Résolution du système par calcul de l'inverse

A partir de l'équation  $Ax = b$  et en supposant que la matrice  $A$  est inversible, on peut calculer le vecteur résultat  $x$  par la relation :

$$x = A^{-1} \cdot b$$

On rappelle que chaque coefficient  $x_i$  du vecteur résultat  $x$  est obtenu par la relation :

$$x_i = \sum_{j=0}^{n-1} a_{ij} b_j$$

avec  $a_{ij}$  le coefficient de  $A^{-1}$  d'indices  $i, j$  et  $b_j$  le coefficient d'indices  $j$  du vecteur  $b$ .

- 2) Ecrire une fonction python, nommée **METHODE2**, qui permet de résoudre le système d'équations ( $Ax = b$ ) par calcul de l'inverse.

## Partie III : Résolution du système par la méthode dite directe

Nous pouvons remarquer que si le système se présente sous la forme suivante :

$$\begin{cases} a_{0,0}x_0 + a_{0,1}x_1 + \dots + a_{0,n-1}x_{n-1} = b_0 \\ a_{1,1}x_1 + \dots + a_{1,n-1}x_{n-1} = b_1 \\ \vdots \\ a_{n-1,n-1}x_{n-1} = b_{n-1} \end{cases}$$

et en supposant qu'aucun des termes  $a_{ii}$ ,  $0 \leq i \leq n-1$ , n'est nul, on peut résoudre le système comme suit :

$$\begin{cases} x_{n-1} = \frac{b_{n-1}}{a_{n-1,n-1}} \\ x_i = \frac{b_i - \sum_{j=i+1}^{n-1} a_{ij}x_j}{a_{ii}} \quad i = n-2, n-1, \dots, 0 \end{cases}$$

Pour pouvoir utiliser cette méthode, il faut rendre le système triangulaire supérieur. Pour cela, on procède comme suit :

### Etape 1

- On vérifie que l'élément  $a_{0,0}$  est non nul sinon on permute la ligne d'indice 0 avec la première ligne du système à premier coefficient non nul.
- On applique la relation  $L_i \leftarrow L_i - \frac{a_{i,0}}{a_{0,0}} L_0$  sur les coefficients des lignes d'indices 1 à  $n-1$  ( $L_i$  représente les coefficients de la ligne  $i$  de  $A$ ).

Etape k

- On vérifie que l'élément  $a_{kk}$  est non nul sinon on permute la ligne d'indice  $k$  avec la première ligne (parmi les lignes de  $k+1$  à  $n$  du système) qui présente un coefficient d'indice colonne  $k$  non nul.
- On applique la relation  $L_i \leftarrow L_i - \frac{a_{ik}}{a_{kk}} L_k$  sur les coefficients des lignes de  $k+1$  à  $n-1$  ( $L_i$  représente les coefficients de la ligne  $i$  de  $A$ ).

L'étape 1 permet de rendre nul les coefficients de  $a_{1,0}$  à  $a_{n-1,0}$  de la première colonne de la matrice. Il faut donc ré-appliquer cette démarche, au reste de la matrice, étape par étape, jusqu'à obtenir une matrice triangulaire supérieure.

- 3) En supposant qu'aucun des termes  $a_{ii}$ ,  $0 \leq i \leq n-1$ , n'est nul, Ecrire une fonction python, nommée **TRIANGLE**, qui permet de rendre le système triangulaire supérieure.
- 4) Ecrire une fonction python, nommée **METHODE3**, qui permet de résoudre le système d'équations ( $Ax = b$ ) par utilisation de la méthode dite directe.

**Partie IV : Résolution du système par la décomposition LU**

La factorisation **LU** d'une matrice  $A_{n \times n}$  est une méthode de décomposition très importante dans le domaine de l'analyse numérique. Sa définition est simple, mais ses applications sont très nombreuses et très utiles.

La factorisation **LU** consiste à écrire une matrice inversible  $A$  comme le produit de deux autres matrices  $L$  et  $U$  :  $A = L \times U$

$L$  est une matrice triangulaire **inférieure** ayant des 1 sur la diagonale et  $U$  une matrice triangulaire **supérieure**.

$$L = \begin{pmatrix} 1 & 0 & 0 & \dots & 0 \\ l_{1,0} & 1 & 0 & \dots & 0 \\ l_{2,0} & l_{2,1} & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ l_{n-1,0} & l_{n-1,1} & l_{n-1,2} & \dots & 1 \end{pmatrix} U = \begin{pmatrix} u_{0,0} & u_{0,1} & \dots & u_{0,n-2} & u_{0,n-1} \\ 0 & u_{1,1} & \dots & u_{1,n-2} & u_{1,n-1} \\ 0 & 0 & \dots & u_{2,n-2} & u_{2,n-1} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & u_{n-1,n-1} \end{pmatrix}$$

Les déterminations des coefficients de  $L$  et de  $U$  sont liées et sont donnés par l'algorithme suivant :

- Le coefficient  $u_{0,0} = a_{0,0}$  ;
- La première ligne de  $U$  est la première ligne de  $A$  et la première colonne de  $L$  est formée par les pivots  $l_{i,0} = \frac{a_{i,0}}{a_{0,0}}$ .
- Les coefficients de la diagonale de  $U$  sauf  $u_{n-1,n-1}$  sont calculés par :  $u_{i,i} = a_{i,i} - \sum_{k=0}^{i-1} l_{i,k} u_{k,i}$  ;
- Les coefficients de  $U$  et  $L$  en dehors de la diagonale sont calculés par :  $u_{i,j} = a_{i,j} - \sum_{k=0}^{i-1} l_{i,k} u_{k,i}$   
 $l_{j,i} = \frac{1}{u_{i,i}} (a_{j,i} - \sum_{k=0}^{i-1} l_{j,k} u_{k,i})$  ;

Le coefficient  $u_{n-1,n-1} = a_{n-1,n-1} - \sum_{k=0}^{n-2} l_{n-1,k} u_{k,n-1}$ .

- 5) En supposant que la matrice  $A$  du système est inversible, écrire une fonction python, nommée **decomp\_LU**, qui permet de décomposer une matrice **A** en deux matrices **L** et **U**.

Supposons maintenant que nous connaissons la factorisation **LU** de la matrice  $A$ . Alors, le système  $A \cdot x = b$  peut se réécrire  $L \cdot U \cdot x = b$ .

Si l'on pose  $z = U \cdot x$ , on doit résoudre les équations  $L \cdot z = b$  où l'inconnue est  $z$  puis  $U \cdot x = z$  où l'inconnue est  $x$ .

Deux équations au lieu d'une, cela paraît plus compliqué mais ici, les matrices  $L$  et  $U$  sont triangulaires et dans ce cas la résolution est rapide.

L'algorithme de détermination de  $z$  est : 
$$\begin{cases} z_0 = b_0 \\ z_i = b_i - \sum_{j=0}^{i-1} L_{i,j} z_j \end{cases} 1 \leq i \leq n-1$$

Pour  $U.x = z$ , on retrouve l'algorithme de Gauss puisque  $U$  est triangulaire supérieure.

$$\begin{cases} x_{n-1} = z_{n-1} \\ x_i = \frac{(z_i - \sum_{j=i+1}^{n-1} U_{i,j} x_j)}{U_{i,i}} \end{cases} n-2 \leq i \leq 0$$

- 6) Ecrire une fonction python, nommée **METHODE4**, qui permet de résoudre le système d'équations ( $Ax = b$ ) par utilisation de la décomposition LU.

### Partie V : Résolution en PYTHON et comparaison des différentes méthodes

Nous voulons comparer les différentes méthodes de résolution du système présentées dans les parties **I**, **II**, **III**, **IV** et celle de Python et cela par calcul du temps d'exécution de chacune des méthodes. Le temps d'exécution en Python pourrait être obtenu en faisant appel à la fonction `time()` de la bibliothèque `time` avant et après l'appel de la fonction dont on désire calculer le temps d'exécution. La commande `time()` retourne en effet, une valeur représentant le temps **CPU**. Il suffit donc de calculer la différence entre les deux valeurs retournées par `time()` pour connaître la durée d'exécution d'une fonction.

- 7) Ecrire une fonction python, nommée **METHODE5**, qui permet de résoudre le système d'équations ( $Ax = b$ ) par utilisation de la fonction prédéfinie de Python.

Soit le système **S** défini par :

$$S = \begin{cases} 2.x_0 + -1.x_1 = 1 \\ -1.x_0 + 2.x_1 + -1.x_2 = 2 \\ -1.x_1 + 2.x_2 = 3 \end{cases}$$

- 8) Définir la matrice **A** du système **S**.
- 9) Définir le vecteur **b** du système **S**.
- 10) Ecrire les instructions Python permettant la résolution du système **S** par les différentes méthodes de résolutions présentées en **I**, **II**, et **III**, **IV** et celle de Python et d'indiquer la méthode la plus rapide pour la résolution de ce système.

## V.2 Méthodes de recherche de zéros d'une fonction

### PARTIE A

On désire implémenter un module de fonctions python permettant la recherche approchée d'un zéro d'une fonction dans un intervalle fixé, par plusieurs méthodes avec une précision de calcul  $\varepsilon$ .

#### Description des méthodes

##### ● *Méthode des dichotomies*

Soit une fonction  $f$  donnée. En supposant que l'on connaisse un intervalle  $[a, b]$  sur lequel la fonction change de signe, c'est à dire que  $f(a)f(b) < 0$ . Si la fonction  $f$  est continue et strictement monotone, nous savons que cet intervalle contient alors une racine  $\alpha$  de  $f$ .

Découpons cet intervalle en deux en posant :  $c = \frac{(a+b)}{2}$ . Trois cas sont possibles :

- Si  $f(c)$  est du signe de  $f(a)$ , la racine appartient à l'intervalle  $[c, b]$ ,
- Si  $f(c)$  est du signe opposé à  $f(a)$ , la racine appartient à l'intervalle  $[a, c]$ ,
- Si  $f(c) = 0$ , la racine de  $f$  est  $c$ .

Il suffit de répéter le processus jusqu'à ce que la précision  $\varepsilon$  demandée soit atteinte. Cette précision est atteinte lorsque  $|f(c)| \leq \varepsilon$ .

##### ● *Méthode de Newton*

La méthode de Newton consiste à calculer une suite de valeurs  $(x_n)$  convergeant vers la racine  $\alpha$  de l'équation  $f(x) = 0$ . Cette méthode est applicable à la double condition : la fonction doit être dérivable et la dérivée ne doit pas s'annuler au voisinage de  $\alpha$ .

Partant d'un point  $x_0$ , on calcule successivement :

$$\begin{aligned} x_1 &= x_0 - \frac{(f(x_0))}{(f'(x_0))} \\ x_2 &= x_1 - \frac{(f(x_1))}{(f'(x_1))} \\ &\vdots \\ x_n &= x_{n-1} - \frac{(f(x_{n-1}))}{(f'(x_{n-1}))} \end{aligned}$$

Le processus s'arrête lorsque la différence  $|x_n - x_{n-1}| \leq \varepsilon$ ,  $x_n$  est alors une valeur approchée de  $\alpha$ .

##### ● *Méthode d'approximation successive*

En posant par exemple  $g(x) = x + f(x)$ , l'équation  $f(x) = 0$  peut s'écrire sous la forme  $x = g(x)$ .

En supposant que la solution appartienne à un intervalle  $[a, b]$  fixé et que  $\forall x \in [a, b]$ , alors la suite définie par  $x_{n+1} = g(x_n) \forall x_0 \in [a, b]$  converge vers la racine  $\alpha$  quand  $n$  tend vers l'infini.

Le processus s'arrête lorsque la différence  $|x_n - x_{n-1}| \leq \varepsilon$ ,  $x_n$  est alors une valeur approchée de  $\alpha$ .

### ● Méthode de Regula Falsi

On peut améliorer la convergence de la méthode des dichotomies.

On définit l'abscisse  $c$  comme intersection de l'axe des  $x$  et de la corde joignant les points  $M_a$  et  $M_b$  tels que  $M_a = (a, f(a))$  et  $M_b = (b, f(b))$

$$c = \frac{(af(b) - bf(a))}{(f(b) - f(a))}$$

- Si  $f(c) = 0$ , on a évidemment trouvé une racine de  $f$  ;
- Si  $f(c)$  est du signe de  $f(a)$ , la racine appartient à l'intervalle  $[c, b]$  ;
- Si  $f(c)$  est du signe opposé à  $f(a)$ , la racine appartient à l'intervalle  $[a, c]$ .

Il suffit de répéter le processus jusqu'à ce que la précision  $\varepsilon$  demandée soit atteinte. Cette précision est atteinte lorsque  $|f(c)| \leq \varepsilon$ .

### Travail demandé

**NB :** Pour toutes les questions, les vérifications de continuité, de dérivabilité et de monotonie des fonctions ne sont pas demandées.

1. On souhaite chercher une racine de l'équation  $f$  définie par :  $f(x) = tg(x) - x$ .

Ecrire une fonction **f** définissant la fonction  $f$ .

2. L'utilisation de certaines méthodes nécessite le calcul de la fonction dérivée. Une approximation numérique de la dérivée d'une fonction  $f$  en un point  $x$  est donnée par :  $f'(x) = \frac{f(x+h) - f(x-h)}{2h}$ , avec  $h$  constante égale à 0.001.

Ecrire une fonction **fprime** permettant de calculer puis retourner la dérivée de  $f$  en un point  $x$  donné.

Dans la suite, **a**, **b**, **eps**, **xn**, **kmax**, **nb** et **ibf** désignent :

**a** et **b** : les extrémités de l'intervalle où se trouve la racine de  $f$  que l'on cherche,

**eps** : la précision  $\varepsilon$  voulue,

**xn** : approximation de la racine recherchée,

**kmax** : le nombre maximum d'itérations que l'on accepte d'effectuer,

**nb** : nombre d'itérations réellement effectuées,

**ibf** : indicateur de bonne fin. Si le résultat de recherche de la racine est obtenu avec la précision voulue avant que l'on ait effectué les **kmax** itérations, **ibf=0** et **xn** contient l'approximation de la racine recherchée ; sinon **ibf=1**.

Chacune des fonctions de calcul de zéro d'une fonction  $f$  prend **a**, **b**, **eps** et **kmax** comme données (paramètres en entrée) et retourne **xn**, **nb** et **ibf**.

3. Ecrire une fonction nommée **dicho** qui permet de calculer, par la méthode des dichotomies, une valeur

approximative **xn** de la racine  $\alpha$  pour laquelle la fonction  $f$  s'annule.

4. Ecrire une fonction nommée **newton** qui permet de calculer, par la méthode de Newton, une valeur approximative **xn** de la racine  $\alpha$ . On prendra comme point de départ la valeur  $x_0 = \frac{a+b}{2}$ .
5. L'utilisation de la méthode d'approximations successives nécessite la réécriture de l'équation  $f(x) = 0$  sous la forme  $x = g(x)$ .
  - 5.1. Ecrire une fonction **g** définissant la fonction  $g(x) = x + f(x)$
  - 5.2. Ecrire une fonction nommée **success** qui permet de calculer, par la méthode d'approximations successives, une valeur approximative **xn** de la racine  $\alpha$  pour laquelle la fonction  $f$  s'annule en utilisant la fonction  $g$  définie en 5.1. On prendra comme point de départ la valeur  $x_0 = \frac{a+b}{2}$ .
6. Ecrire une fonction nommée **falsi** qui permet de calculer, par la méthode de Regula Falsi, une valeur approximative **xn** de la racine  $\alpha$  pour laquelle la fonction  $f$  s'annule.

## **PARTIE B**

On veut conserver les résultats d'exécution de ces fonctions dans une matrice. Pour cela, on associe à chacune des méthodes un numéro tel que :

- 1 : pour la méthode des dichotomies,
- 2 : pour la méthode de Newton,
- 3 : pour la méthode d'approximations successives,
- 4 : pour la méthode de Regula Falsi.

7. Ecrire une fonction nommée **creer** qui retourne une matrice **M** contenant un nombre de lignes égal au nombre de méthodes de recherche. Chacune de ces lignes contient dans l'ordre : le numéro de la méthode et les valeurs **nb**, **xn** et **ibf**. Ces valeurs seront extraites à partir du résultat obtenu par appel des fonctions associées à chacune des méthodes.

On désire réorganiser le contenu de la matrice **M** obtenu par l'appel de la fonction **creer** de la question 7 de telle sorte que les lignes dont **ibf** vaut 0 (c'est-à-dire recherche fructueuse) seront placées au début de la matrice. Ces lignes seront par la suite triées selon la valeur associée à **nb** dans **M**, de la plus petite vers la plus grande.

8. Ecrire une fonction nommée **organiser** qui permet de retourner une matrice **M** quelconque réorganisée selon le principe présenté ci-dessus.

On demande d'utiliser, pour la programmation du tri, la méthode dite par sélection et qui consiste dans une première étape à chercher la valeur la plus petite associée à **nb** dans la partie de **M** à trier et à permuter la ligne correspondante avec la première ligne. L'étape suivante consiste à trier la matrice **M** diminuée de sa première ligne c'est-à-dire de chercher dans la matrice diminuée la valeur associée à **nb** la plus petite puis de permuter la ligne correspondante avec la première ligne de la matrice diminuée. Cette même opération sera répétée pour le reste de la matrice jusqu'à ce que le tri souhaité soit effectué.

9. En supposant que les valeurs de **a**, **b**, **eps** et **kmax** sont déjà saisies, écrire le script python permettant d'afficher le nom de la méthode (si elle existe) ayant effectué le minimum d'itérations pour le calcul du zéro de  $f$  ainsi que le résultat de la recherche.

## VI Alternatives de corrections pour les exercices

### VI.1 Corrigé des exercices de traitement d'image

#### 1. Binarisation

```
import numpy as np

import matplotlib.pyplot as plt

def Binariser(M,s):

    n,p=np.shape(M)

    M1=np.zeros((n,p))

    for i in range(n):

        for j in range(p):

            if M[i,j]>s:

                M1[i,j]=255

    return M1

A=plt.imread("source.png")

B=Binariser(A,127)

plt.imshow(B)

plt.show()

plt.imsave("cible.png")
```

#### 2. Contraste

```
f=lambda x:x-0.4*(x-127)

def Contraste(M):

    n,p=np.shape(M)

    M1=np.empty((n,p))

    for i in range(n):

        for j in range(p):

            y=f(M[i,j])

            if y>255:
```

```
        M1[i,j]=255

    elif y<0:

        M1[i,j]=0

    else:

        M1[i,j]=y

return M1
```

### 3. Négatif d'une image

```
def Negatif(M):

    n,p=np.shape(M)

    M1=np.empty((n,p))

    for i in range(n):

        for j in range(p):

            M1[i,j]=255-M[i,j]

    return M1
```

### 4. Transformation d'une image couleur en une image en niveau de gris

```
def Transformer(M):

    R,V,B=M[:, :, 0], M[:, :, 1], M[:, :, 2]

    M1=0.21*R + 0.71*V + 0.07*B

    return M1
```

### 5. Filtrage avec moyennage

```
def Filtrerer_moy(M):

    n,p=np.shape(M)

    M1=M[:, :]

    for i in range(1,n-1):

        for j in range(1,p-1):
```

```
I=M[i-1:i+2,j-1,j+2]

M1[i,j]=sum(I[x,y] for y in range(3) for x in range(3))/9

return M1
```

## 6. Filtrage avec médiane

```
def Filtrer_mediane(M):

    n,p=np.shape(M)

    M1=M[:, :]

    for i in range(1,n-1):

        for j in range(1,p-1):

            I=M[i-1:i+2,j-1,j+2]

            L=[I[x,y] for y in range(3) for x in range(3)]

            L.sort()

            M1[i,j]=L[4]

    return M1
```

## 7. Détection de contours

```
def Detecter_contours(M):

    n,p=np.shape(M)

    M1=M[:, :]

    for i in range(1,n-1):

        for j in range(1,p-1):

            I=M[i-1:i+2,j-1,j+2]

            l=int(np.sqrt((I[1,0]-I[0,1])**2+(I[1,2]-I[2,1])**2))

            if l==0:

                M1[i,j]=0

            elif l>200:

                M1[i,j]=255

    return M1
```

## VI.2 Corrigés des exercices sur les algorithmes de cryptage

### #Chiffrement Cesar

```

Lmajus=['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O',
'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z']
Lminus=['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o',
'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z']
def C(x,k):
    y=x+k
    if y>26:
        y-=26
    return y
def D(x,k):
    y=x-k
    if y<0:
        y+=26
    return y
def Chiffre_cesar(msg,clef):
    msg1=""
    for c in msg:
        if c in Lmajus:
            e=Lmajus.index(c)
            e1=C(e,clef)
            msg1+=Lmajus[e1]
        elif c in Lminus:
            e=Lminus.index(c)
            e1=C(e,clef)
            msg1+=Lminus[e1]
        else:
            msg1+=c
    return msg1
def Dechiffre_cesar(msg,clef):
    msg1=""
    for c in msg:
        if c in Lmajus:
            e=Lmajus.index(c)
            e1=D(e,clef)
            msg1+=Lmajus[e1]
        elif c in Lminus:
            e=Lminus.index(c)

```

```
        e1=D(e,clef)
        msg1+=Lminus[e1]
    else:
        msg1+=c
    return msg1

>>> Chiffre_cesar("COUCOU",11)
'NZFNZF'
>>> Dechiffre_cesar("NZFNZF",11)
'COUCOU'
>>> Chiffre_cesar("J'aime écouter la radio toute la journée",20)
"D'ucgy ywionyl fu luxci niony fu diolhyy"
>>> Dechiffre_cesar("D'ucgy ywionyl fu luxci niony fu diolhyy",20)
"J'aime écouter la radio toute la journée"
```

```
def Attaque_cesar(msg):
    for k in range(1,26):
        print(Dechiffre_cesar(msg,k))
```

```
>>> Attaque_cesar("NZFNZF")
```

```
MYEMYE
LXDLXD
KWCKWC
JVBJVB
IUAIUA
HTZHTZ
GSYGSY
FRXFRX
EQWEQW
DPVDPV
COUCOU
BNTBNT
AMSAMS
ZLRZLR
YKQYKQ
XJPXJP
WIOWIO
VHNVHN
UGMUGM
TFLTFL
SEKSEK
RDJRDJ
```

QCIQCI  
PBHPBH  
OAGOAG

### #Chiffrement Vigenère

```
def Chiffre_vigenere(msg,mot):  
    msg1=""  
    mot=mot.upper()  
    i=0  
    for c in msg:  
        if c in Lmajus:  
            e=Lmajus.index(c)  
            e1=C(e,Lmajus.index(mot[i]))  
            msg1+=Lmajus[e1]  
        elif c in Lminus:  
            e=Lminus.index(c)  
            e1=C(e,Lmajus.index(mot[i]))  
            msg1+=Lminus[e1]  
        else:  
            msg1+=c  
        i+=1  
        if i>len(mot)-1:  
            i=0  
    return msg1  
  
def Dechiffre_vigenere(msg,mot):  
    msg1=""  
    mot=mot.upper()  
    i=0  
    for c in msg:  
        if c in Lmajus:  
            e=Lmajus.index(c)  
            e1=D(e,Lmajus.index(mot[i]))
```

```
        msg1+=Lmajus[e1]

    elif c in Lminus:

        e=Lminus.index(c)

        e1=D(e,Lmajus.index(mot[i]))

        msg1+=Lminus[e1]

    else:

        msg1+=c

    i+=1

    if i>len(mot)-1:

        i=0

return msg1
```

```
>>> Chiffre_vigenere("J'aime écouter la radio toute la journée","MUSIQUE")
"V'sqcy qwgcjyv fs huhui beoxq di ds glfmu"
>>> Dechiffre_vigenere("V'sqcy qwgcjyv fs huhui beoxq di ds glfmu","MUSIQUE")
"J'aime écouter la radio toute la journée"
```

### **#Chiffrement RSA**

```
def invmod(n,p):

    (n1,p1,u1,u2)=(n,p,1,0)

    while p1>0:

        q=n1/p1

        (n1,p1,u1,u2)=(p1,n1-q*p1,u2,u1-q*u2)

    return u1%p

def clef_RSA(p,q):

    z=(p-1)*(q-1)

    n=p*q

    c=z+1

    d=invmod(c,z)

    return (n,c),(n,d)

def CryptDecryptE(e,CL):
```

```
        return (e**CL[1])%CL[0]
def CryptDecryptList(L,CL):
    return [CryptDecryptE(e,CL) for e in L]
def Codage(ch):
    return [ord(c) for c in ch]
def Decodage(L):
    ch=""
    for e in L:
        ch+=chr(e)
    return ch
def CryptageC(ch,CL):
    return CryptDecryptList(Codage(ch),CL)
def DecryptageC(L,CL):
    return Decodage(CryptDecryptList(L,CL))
```

### VI.3 Corrigé des méthodes de résolution d'un système linéaire

```
import numpy as np
def METHODE1(A,b):
    n=len(b)
    x=np.empty(n)
    for i in range(n):
        A1=A[:,:]
        for k in range(n):
            A1[k,i]=b[k]
        x[i]=np.linalg.det(A1)/ np.linalg.det(A)
    return x
def METHODE2(A,b):
    n=len(b)
    x=np.empty(n)
    IA=np.linalg.inv(A)
    for i in range(n):
        x[i]=sum(IA[i,j]*b[j] for j in range(n))
    return x
def TRIANGLE(A):
    n=len(A)
    for k in range(1,n):
```

```

        for i in range(k,n):
            pivot=A[i,k-1]/ A[k-1,k-1]
            for j in range(k-1,n)
                A[i,j]=A[i,j] - pivot*A[k-1,j]

    return A
def METHODE3 (A,b):
    n=len(b)
    x=np.empty(n)
    x[n-1]=b[n-1]/A[n-1,n-1]
    for i in range(n-2,-1,-1):
        x[i]=(b[i]-sum(A[i,j]*x[j] for j in range(i+1,n)))/A[i,i]
    return x
def decomp_LU(A):
    n=len(b)
    L=np.eye(n)
    U=np.zeros((n,n))
    U[0,0]=A[0,0]
    for i in range(1,n):
        U[0,i]=A[0,i]
        L[i,0]=A[i,0]/ A[0,0]
        U[i,i]= A[i,i]-sum(L[i,k]*U[k,i] for k in range(i))
        for j in range(i+1,n):
            U[i,j]= A[i,j]-sum(L[i,k]*U[k,j] for k in range(i))
            L[j,i]= (A[j,i]-sum(L[k,i]*U[j,k] for k in range(i)))/U[i,i]
    U[n-1,n-1]= A[n-1,n-1]-sum(L[n-1,k]*U[k,n-1] for k in range(n-1))
    return L,U
def METHODE4 (A,b):
    n=len(b)
    x=np.empty(n)
    L,U=decomp_LU(A)
    z=np.empty(n)
    z[0]=b[0]
    for i in range(1,n):
        z[i]=b[i]-sum(L[i,j]*z[j] for j in range(i))
    x[n-1]=z[n-1]
    for i in range(n-2,-1,-1):
        x[i]=(z[i]-sum(U[i,j]*x[j] for j in range(i+1,n)))/U[i,i]
    return x
def METHODE5 (A,b):
    return np.linalg.solve(A,b)
A=np.array([[2,-1,0],[-1,2,-1],[0,-1,2]])
b=np.array([1,2,3])

```

```
from time import time
t=time()
x=METHODE1(A,b)
tm1=time()-t
t=time()
x=METHODE2(A,b)
tm2=time()-t
t=time()
x=METHODE3(A,b)
tm3=time()-t
t=time()
x=METHODE4(A,b)
tm4=time()-t
t=time()
x=METHODE5(A,b)
tm5=time()-t
L=[tm1,tm2,tm3,tm4,tm5]
Tmin=min(L)
print("la methode{} est la méthode la plus rapide".format(L.index(Tmin)+1))
```

## VI.4 Corrigé des méthodes de recherche de zéros d'une fonction

```
from math import tan
f=lambda x:tan(x)-x
def fprime(x):
    h=0.001
    return (f(x+h)-f(x-h))/(2*h)
def dicho(a,b,eps,kmax):
    nb=0
    while 1:
        nb+=1
        xn=(a+b)/2
        if f(a)*f(xn)<0:
            b=xn
        elif f(a)*f(xn):
            a=xn
        if abs(f(xn))<=eps:
            break
    if nb<kmax:
        ibf=0
    else:
```

```

        ibf=1
    return nb,xn,ibf
def newton(a,b,eps,kmax):
    nb=0
    x0=(a+b)/2
    while 1:
        nb+=1
        xn=x0-f(x0)/fprime(x0)
        if abs(xn-x0)<=eps:
            break
        else:
            x0=xn
    if nb<kmax:
        ibf=0
    else:
        ibf=1
    return nb,xn,ibf
g=lambda x:f(x)-x
def success(a,b,eps,kmax):
    nb=0
    x0=(a+b)/2
    while 1:
        nb+=1
        xn=g(x0)
        if abs(xn-x0)<=eps:
            break
        else:
            x0=xn
    if nb<kmax:
        ibf=0
    else:
        ibf=1
    return nb,xn,ibf
def falsi(a,b,eps,kmax):
    nb=0
    while 1:
        nb+=1
        xn=(a*f(b)-b*f(a))/(f(b)-f(a))
        if f(a)*f(xn)<0:
            b=xn
        elif f(a)*f(xn):
            a=xn

```

```

        if abs(f(xn)) <= eps:
            break
    if nb < kmax:
        ibf = 0
    else:
        ibf = 1
    return nb, xn, ibf
import numpy as np
def creer(a, b, eps, kmax):
    M = np.empty((4, 4))
    t1 = dicho(a, b, eps, kmax)
    t2 = newton(a, b, eps, kmax)
    t3 = success(a, b, eps, kmax)
    t4 = falsi(a, b, eps, kmax)
    for i in range(4):
        M[i, 0] = i + 1
    for j in range(1, 4):
        M[0, j] = t1[j - 1]
        M[1, j] = t2[j - 1]
        M[2, j] = t3[j - 1]
        M[3, j] = t4[j - 1]
    return M
def organiser(M):
    nl = np.shape(M)[0]
    i, j = 0, nl - 1
    while 1:
        while M[i, 3] == 0 and i <= nl - 1:
            i += 1
        while M[j, 3] == 1 and j >= 0:
            j -= 1
        if i < j:
            M[i, :], M[j, :] = M[j, :], M[i, :]
            i += 1
            j -= 1
        else:
            break
    nlz = i - 1
    for i in range(nlz - 1):
        im = i
        for j in range(i + 1, nlz):
            if M[j, 1] < M[im, 1]:
                im = j

```

```
        M[i,:],M[im,:]= M[im,:],M[i,:]
    return M
M=creer(a,b,eps,kmax)
M=organiser(M)
d={1:"dichotomie", 2:"newton", 3:"approximation successive", 4:"regula falsi"}
if M[0,3]==1:
    print("aucune méthode")
else :
    print("la méthode{} a effectué le minimum d'itérations avec un nombre
d'itéartions égal à {}".format(d[M[0,0]],M[0,1]))
```

## Bibliographie et Netographie

### Ouvrages :

Learning Python, Mark Lutz, Edition O'Reilly, Juin 2013, ISBN: 9781449355722.

Informatique pour tous en classes préparatoires aux grandes écoles, Benjamin Wack & al., Edition Eyrolles, Août 2013, ISBN: 978-2-212-13700-2.

### Supports de Cours

Support de Cours de M.Hammami, Simulation numérique, pour 2<sup>ème</sup> année préparatoire des sections MP-PC-T, IPEIEM, Année universitaire 2019-2020.

### Sites Web :

<https://www.python-course.eu/>

<https://www.ics.uci.edu/~brgallar/index.html>

<https://www.scipy.org/>

<https://numpy.org/>

<https://matplotlib.org/>