

# **Programmation Python pour les mathématiques**

Julien Guillod  
Sorbonne Université

2020 – 2021

Le but de ce recueil d'exercices est de fournir une introduction à l'usage de la programmation dans divers domaines des mathématiques. Le langage de programmation utilisé est Python 3.6. Ces exercices servent de base aux travaux pratiques LU2MA100 donnés à Sorbonne Université en licence.

Ce document est aussi disponible au format [HTML](#) et [Jupyter Notebook](#). Les notebooks peuvent également être exécutés sur [GESIS](#).

Les erreurs et remarques concernant ce document sont les bienvenues à l'adresse *julien.guillod* *CHEZ sorbonne-universite.fr* en commençant le sujet du message par *[python]*.

# Table des matières

---

<b>1</b>	<b>Introduction</b>	<b>5</b>
1.1	Pourquoi Python ?	5
1.2	Prérequis	5
1.3	Documentation	6
1.4	Installation	6
1.5	Lancement de Jupyter Lab	8
1.6	Utilisation de Jupyter Lab	9
<b>2</b>	<b>Structures de données</b>	<b>12</b>
	Exercice 2.1 : Listes	12
	Exercice 2.2 : Tuples	14
	Exercice 2.3 : Ensembles	14
	Exercice 2.4 : Dictionnaires	15
<b>3</b>	<b>Structures homogènes</b>	<b>18</b>
	Exercice 3.1 : Introduction à Numpy	18
	Exercice 3.2 : Opérations sur les tableaux	20
	Exercice 3.3 : Représentations graphiques	20
	Exercice 3.4 : ! Indexage de tableaux	23
<b>4</b>	<b>Calcul symbolique</b>	<b>25</b>
	Exercice 4.1 : Introduction à Sympy	26
	Exercice 4.2 : Applications	27
	Exercice 4.3 : Fonction pathologique	28
	Exercice 4.4 : ! Fonction de Green du laplacien	28
<b>5</b>	<b>Intégration</b>	<b>31</b>
	Exercice 5.1 : Méthode des rectangles	31
	Exercice 5.2 : Méthode des trapèzes	32
	Exercice 5.3 : Méthode de Monte-Carlo	33
	Exercice 5.4 : ! Méthode de Simpson	34
	Exercice 5.5 : !! Intégration avec Scipy	35
<b>6</b>	<b>Algèbre</b>	<b>36</b>
	Exercice 6.1 : Décomposition LU	36
	Exercice 6.2 : Méthode de la puissance itérée	37
	Exercice 6.3 : Groupes de permutations	39

<b>7</b>	<b>Zéro de fonctions</b>	<b>40</b>
	Exercice 7.1 : Méthode de Newton en une dimension . . . . .	40
	Exercice 7.2 : Méthode de Newton en plusieurs dimensions . . . . .	41
	Exercice 7.3 : Attracteur de la méthode de Newton . . . . .	41
	Exercice 7.4 : <b>!!</b> Équation différentielle non linéaire . . . . .	42
<b>8</b>	<b>Théorie des graphes</b>	<b>44</b>
	Exercice 8.1 : Graphes comme dictionnaires . . . . .	44
	Exercice 8.2 : Triangles dans un graphe . . . . .	45
	Exercice 8.3 : <b>!!</b> Module NetworkX . . . . .	46
<b>9</b>	<b>Probabilités et statistiques</b>	<b>47</b>
	Exercice 9.1 : Série harmonique de signe aléatoire . . . . .	47
	Exercice 9.2 : Loi de Benford . . . . .	48
	Exercice 9.3 : Ruine du joueur . . . . .	49
	Exercice 9.4 : <b>!!</b> Percolation . . . . .	49
<b>10</b>	<b>Équations différentielles</b>	<b>52</b>
	Exercice 10.1 : Méthodes d'Euler . . . . .	52
	Exercice 10.2 : Méthodes de Runge-Kutta . . . . .	54
	Exercice 10.3 : Mouvement d'une planète . . . . .	55
	Exercice 10.4 : <b>!!</b> Équation des ondes cubique . . . . .	56
	Exercice 10.5 : <b>!!!</b> Méthodes de Bogacki-Shampine . . . . .	57
<b>11</b>	<b>Cryptographie</b>	<b>59</b>
	Exercice 11.1 : Code de Vigenère . . . . .	59
	Exercice 11.2 : <b>!</b> Casser le code de Vigenère . . . . .	60
	Exercice 11.3 : Générer des nombres premiers . . . . .	61
	Exercice 11.4 : Générer des nombres pseudo-premiers . . . . .	61
	Exercice 11.5 : Chiffrement RSA . . . . .	62
	Exercice 11.6 : <b>!!!</b> Casser le chiffrement RSA . . . . .	63

# Introduction 1

---

Le but des exercices de ce recueil n'est pas d'apprendre la syntaxe du langage Python ni ses subtilités, mais de se focaliser sur son utilisation pratique dans différents domaines des mathématiques : les suites, l'algèbre linéaire, l'intégration, la théorie des graphes, la recherche de zéros de fonctions, les probabilités, les statistiques, les équations différentielles, le calcul symbolique, et la théorie des nombres. La résolution des exercices proposés devrait permettre d'avoir une bonne vision d'ensemble des possibilités d'utilisation de la programmation dans les mathématiques et d'être à même de résoudre des problèmes mathématiques complexes avec l'aide de la programmation. Les exercices plus difficiles sont indiqués par des points d'exclamation :

- **!** : plus long ou plus difficile ;
- **!!** : passablement plus long et complexe ;
- **!!!** : défi.

Le séparateur décimal utilisé est le point et non pas la virgule, afin de se conformer avec l'usage international employé par Python et éviter les confusions.

## 1.1 Pourquoi Python ?

Python est un langage généraliste de programmation interprété qui a la particularité d'être très lisible et pragmatique. Il dispose d'une très grosse base de modules externes, notamment scientifiques, qui le rend particulièrement attractif pour programmer des problèmes mathématiques. Le fait que Python soit un langage interprété le rend plus lent que les langages compilés, mais il assure en revanche une grande rapidité de développement qui permet à l'humain de travailler un peu moins tandis que l'ordinateur devra travailler un peu plus. Cette particularité a fait que Python est devenu l'un des principaux langages de programmation utilisés par les scientifiques.

## 1.2 Prérequis

Les prérequis sont de connaître les bases du langage Python, par exemple telles qu'enseignées dans le cours [1IN001](#) dispensé à Sorbonne Université. Les personnes ne connaissant pas bien les bases du langage Python sont fortement incitées à suivre le MOOC (Massive Open Online Course) *Python 3 : des fondamentaux aux concepts avancés du langage* disponible sur le site de [France Université Numérique](#). En dehors des périodes d'ouverture du MOOC, les vidéos du MOOC sont disponibles sur [YouTube](#).

Par ailleurs la réalisation des exercices demande d'avoir accès à un ordinateur ou un service en ligne disposant de Python 3.6 (ou plus récent) complété par les modules suivants : `numpy`,

`scipy`, `sympy`, `matplotlib`, `numba`, `networkx` et `pandas`. L'emploi d'un éditeur de code permettant l'écriture en Python est aussi vivement conseillé. Il est ici suggéré d'utiliser [Jupyter Lab](#), qui permet à la fois l'écriture des notebooks interactifs et des scripts et également l'ajout de ses propres solutions en dessous des énoncés, ce qui est très pratique. Il n'est pas indispensable d'utiliser Jupyter Lab, d'autres environnements sont aussi adaptés, notamment [Spyder](#) ou [Jupyter Notebook](#).

Les sections suivantes décrivent comment installer et lancer l'environnement Python ou l'utiliser en ligne sans installation.

### 1.3 Documentation

Il n'est généralement pas utile (ni souhaitable) de connaître toutes les fonctions et subtilités du langage Python lors d'une utilisation occasionnelle. Par contre il est indispensable de savoir utiliser la documentation de manière efficace. La documentation officielle est disponible à l'adresse <https://docs.python.org/>. La langue et la version peuvent être sélectionnées en haut à gauche. Il est fortement conseillé de regarder comment la documentation est écrite et d'apprendre à l'utiliser.

### 1.4 Installation

Les personnes ne pouvant ou ne voulant pas installer Python peuvent directement se rendre à la section 1.5 pour des alternatives disponibles en ligne sans installation.

Il existe essentiellement quatre façons d'installer Python et les modules requis pour réaliser les exercices :

- **Anaconda** est une distribution Python complète, c'est-à-dire qu'elle installe directement une très grande quantité de modules (beaucoup plus que nécessaire pour faire les exercices suivants). L'avantage de cette installation est qu'elle est très simple ; son désavantage est qu'elle prend beaucoup d'espace disque. C'est la méthode à privilégier si vous êtes sous Windows ou MacOS et que vous n'avez pas de problème d'espace disque.
- **Miniconda** est une version légère d'Anaconda, qui installe par défaut uniquement la base. L'avantage est qu'elle prend peu d'espace disque, mais elle requiert une action supplémentaire pour installer les modules requis pour faire les exercices. C'est la méthode à privilégier si vous êtes sous Windows ou MacOS et que vous avez peu d'espace disque disponible.
- **Dépôts Linux** : la plupart des distributions Linux permettent d'installer Python et les modules de base directement à partir des dépôts de paquets qui les accompagnent. C'est la méthode privilégiée sous Linux.
- **Pip** est un gestionnaire de paquets pour Python. C'est la méthode à privilégier pour ajouter un module si Python est déjà installé par votre système d'exploitation, et que ce module n'est pas inclus dans les paquets de votre distribution. Cette méthode permet une gestion plus fine et avancée des modules installés que ce qui est proposé avec les méthodes précédentes.

**Installation avec Anaconda.** La façon la plus simple d'installer Python 3 et toutes les dépendances nécessaires sous Windows et MacOS est d'installer [Anaconda](#). Le désavantage d'Anaconda est que son installation prend beaucoup d'espace disque car énormément de modules sont installés par défaut. Les procédures d'installation détaillées selon chaque système d'exploitation sont décrites à l'adresse : <https://docs.anaconda.com/anaconda/install/>. En résumé la procédure d'installation est la suivante :

## 1 Introduction

1. Télécharger Anaconda pour Python 3 à l'adresse : <https://www.anaconda.com/products/individual#Downloads>.
2. Double-cliquer sur le fichier téléchargé pour lancer l'installation d'Anaconda, puis suivre la procédure d'installation (il n'est pas nécessaire d'installer VS Code).
3. Une fois l'installation terminée, lancer Anaconda Navigator à partir du menu démarrer ou de la liste des applications.

**Installation avec Miniconda.** La distribution [Miniconda](#) présente l'avantage sur Anaconda de prendre peu d'espace disque, au prix de devoir installer les modules nécessaires manuellement. La procédure d'installation rapide est la suivante :

1. Télécharger Miniconda pour Python 3 à l'adresse : <https://docs.conda.io/miniconda.html>.
2. Double-cliquer sur le fichier téléchargé pour lancer l'installation de Miniconda, puis suivre la procédure d'installation.
3. Une fois l'installation terminée, lancer Anaconda Prompt à partir du menu Démarrer ou de la liste des applications.
4. Dans le terminal, taper la commande :

```
Terminal  
conda install anaconda-navigator numpy scipy sympy matplotlib numba  
↪ networkx pandas jupyterlab
```

et confirmer l'installation des dépendances.

**Installation à partir des dépôts.** La plupart des distributions Linux permettent d'installer facilement Python et les modules les plus standard directement à partir des dépôts qu'elles incluent. La procédure suivante concerne Ubuntu, mais s'adapte facilement aux autres distributions.

1. Installer Python 3 :

```
Terminal  
sudo apt install python3 python-pip3
```

2. Installer les modules `numpy`, `scipy`, `sympy`, `matplotlib`, `numba`, `networkx` et `pandas` :

```
Terminal  
sudo apt install python3-numpy python3-scipy python3-sympy  
↪ python3-matplotlib python3-numba python3-networkx python3-pandas
```

3. Jupyter Lab n'étant pas disponible dans les paquets d'Ubuntu, il faut l'installer avec Pip :

```
Terminal  
pip3 install jupyterlab
```

**Remarque :** Suivant les systèmes d'exploitation il faut parfois remplacer la commande `pip3` par `pip`. Si vous rencontrez un problème de permissions lors de l'exécution de ces commandes, il faut probablement rajouter `--user` à la fin de la commande précédente.

## 1 Introduction

**Installation avancée avec Pip.** La procédure suivante décrit l'installation manuelle de modules avec le gestionnaire `Pip`.

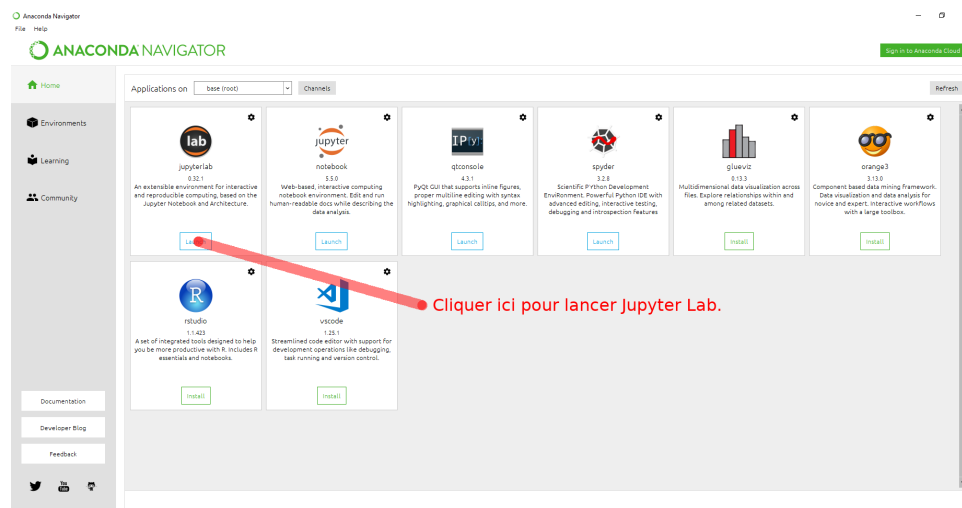
1. Installer Python depuis l'adresse : <https://www.python.org/downloads/>.
2. Installer Pip depuis l'adresse : <https://pip.pypa.io/en/stable/installing/>
3. Installer les modules requis en tapant la ligne de commande suivante dans un terminal :

```
Terminal
pip3 install numpy scipy sympy matplotlib numba networkx pandas
↵ jupyterlab
```

**Remarque :** Suivant les systèmes d'exploitation il faut remplacer la commande `pip3` par `pip`. Si vous rencontrez un problème de permissions lors de l'exécution de ces commandes, il faut probablement ajouter `--user` à la fin de la commande précédente.

### 1.5 Lancement de Jupyter Lab

**Avec Anaconda Navigator.** Si Anaconda ou Miniconda a été installé, il suffit de lancer Anaconda Navigator et de cliquer sur l'icône «jupyterlab» :



**En ligne de commande.** Pour lancer Jupyter Lab en ligne de commande, il faut taper `jupyter lab` dans un terminal. Pour quitter, il faut cliquer sur «Shutdown» dans le menu «File» de la fenêtre Jupyter Lab. Il est aussi possible de taper `Ctrl+C` suivi de `y` (en anglais) ou `o` (en français) dans le terminal où la commande `jupyter lab` a été exécutée.

**Sur le bureau de l'UTES.**

#### Avertissement

Disponible seulement pour les personnes disposant d'un accès informatique à Sorbonne Université.



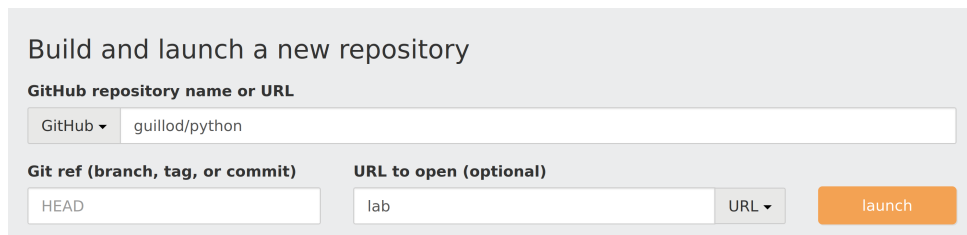
## 1 Introduction

Aller sur le site de l'[UTES](#) puis cliquer sur l'icône correspondante à votre système d'exploitation et suivez les instructions pour vous connecter au bureau à distance.

Pour lancer Jupyter Lab une fois connecté au bureau de l'UTES :

- cliquer sur la loupe située en haut ;
- taper `jupyter lab` dans le champs de rechercher suivi de la touche ENTRÉE ;
- cliquer sur «JUPYTER LAB POUR PYTHON 3.6» ;
- ne pas fermer la fenêtre noire avant d'avoir terminé de travailler sous Jupyter Lab.

**En ligne sans installation.** Pour les personnes ne pouvant ou ne voulant pas installer Python et les dépendances nécessaires sur leur propre ordinateur et n'ayant pas accès au bureau de l'UTES, il est possible d'utiliser Jupyter Lab en ligne avec GESIS en cliquant [ici](#). Aucun compte n'est nécessaire mais les documents stockés sans compte utilisateur sont automatiquement effacés donc il faut impérativement les sauvegarder sur votre propre ordinateur avant de quitter. Il est également possible de créer un compte utilisateur sur GESIS à l'adresse <https://notebooks.gesis.org/hub/>, ce qui permet ensuite de sauvegarder vos documents. Une fois authentifié sur GESIS, entrer les paramètres suivants pour avoir accès aux énoncés des exercices :



The screenshot shows a web form titled "Build and launch a new repository". It contains the following fields and controls:

- A dropdown menu for "GitHub repository name or URL" with "GitHub" selected.
- A text input field containing "guillod/python".
- A dropdown menu for "Git ref (branch, tag, or commit)" with "HEAD" selected.
- A text input field for "URL to open (optional)" containing "lab".
- A dropdown menu for "URL" with a downward arrow.
- An orange "launch" button.

Pour utiliser Jupyter Lab plutôt que Jupyter Notebook sur GESIS, il suffit de remplacer `tree` par `lab` à la fin de l'URL.

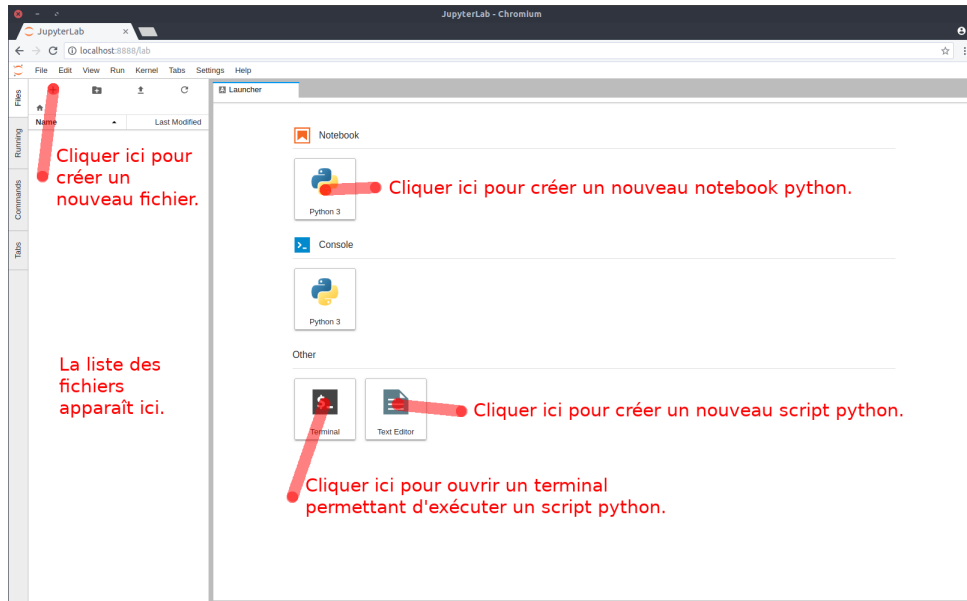
Sinon différents services offrent la possibilité d'utiliser gratuitement Jupyter Lab après création d'un compte :

- [CoCalc](#)
- [Google Colaboratory](#)

## 1.6 Utilisation de Jupyter Lab

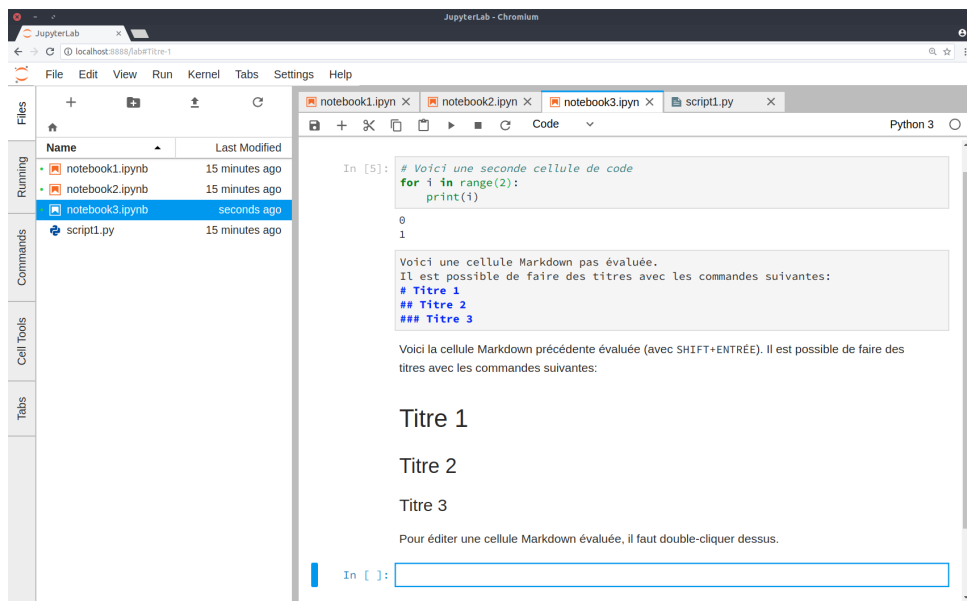
Une fois Jupyter Lab lancé, la fenêtre suivante doit apparaître dans un navigateur :

## 1 Introduction



Jupyter Lab permet essentiellement de traiter trois types de documents : les **notebooks**, les **scripts** et les **terminaux**. Un notebook est constitué de cellules qui peuvent contenir soit du code soit du texte au format Markdown. Les cellules peuvent être évaluées de manière interactive à la demande, ce qui permet une grande flexibilité. Un script Python est simplement un fichier texte contenant des instructions Python. Il s'exécute en entier de A à Z et il n'est pas possible d'interagir interactivement avec lui pendant son exécution (à moins que cela n'ait été explicitement programmé). Pour exécuter un script python il est nécessaire d'ouvrir un terminal.

Un notebook est composé de cellules, ces dernières pouvant être principalement de deux types : des cellules de code et des cellules de texte, comme représenté sur la figure suivante. Les cellules de texte peuvent contenir des formules  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ .



La documentation détaillée de Jupyter Lab est disponible [ici](#).

### Commandes de base :

- Créer un nouveau fichier : cliquer sur le bouton **PLUS** situé en haut à gauche, puis choisir le type de fichier à créer.
- Renommer un fichier : cliquer avec le second bouton de la souris sur le titre du notebook (soit dans l'onglet, soit dans la liste des fichiers).
- Changer le type de cellules : menu déroulant permettant de choisir entre «Code» et «Markdown».
- Exécuter une cellule : combinaison des touches **SHIFT+ENTRÉE**.
- Exécuter un script : taper `python nomduscript.py` dans un terminal pour exécuter le script `nomduscript.py`.
- Réorganiser les cellules : cliquer-déposer.
- Juxtaposer des onglets : cliquer-déposer.

**Remerciements :** Merci à Marie Postel et Nicolas Lantos pour leurs relectures attentives de ce recueil et pour les nombreuses corrections et suggestions. Merci également à Cédric Boutillier, Cindy Guichard et Raphaël Zanella pour avoir pointé différentes erreurs.

# Structures de données 2

Pour représenter des structures de données, Python propose quatre types de base : les listes (type `list`), les tuples (type `tuple`), les ensembles (type `set`) et les dictionnaires (type `dict`). Le but de ce chapitre est de montrer les différences fondamentales entre ces structures de données et d'expliquer à quoi elles sont le plus adaptées. La documentation détaillée sur les structures de données est disponible [ici](#).

## Concepts abordés

- structures de données (liste, tuple, ensemble, dictionnaire)
- types mutable et immutable
- type hashable
- compréhensions de liste, ensemble et dictionnaire
- suites numériques

## Exercice 2.1 : Listes

Une liste est une structure permettant de stocker des éléments hétérogènes :

```
list0 = [0, 5.4, "chaîne", True]
```

Les listes sont mutables, c'est-à-dire qu'il est possible d'y modifier un élément, d'en rajouter un ou d'en supprimer un sans avoir à redéfinir toute la liste.

```
list0[3] = False # remplace True par False
list0.append("nouveau") # ajoute la chaîne de caractères "nouveau" à la liste
list0.insert(2, 34) # insère 34 à la place 2
list0.remove(0) # enlève 0
```

En particulier, il faut faire attention en copiant une liste. Si l'on exécute le code suivant :

```
list1 = list0
list1[2] = "change"
list0
```

alors `list0` est aussi modifié et est égal à `list1`. Pour créer une vraie copie, il faut utiliser le code suivant :

```
list2 = list0.copy()
list2[2] = "rechange"
list0
```

qui ne modifie pas `list0`. À noter qu'il est possible de modifier les éléments d'une liste à l'intérieur d'une fonction :

```
def f(l):
    l[0] = 0
    f(list0)
```

Enfin il est possible de créer des listes à l'aide de la compréhension de listes :

```
list1 = [2*i+1 for i in range(10)]
```

a) Chercher dans la documentation la syntaxe pour concaténer deux listes.

**Indication :** Voir la documentation [ici](#).

b) Chercher dans la documentation la syntaxe pour extraire une tranche d'une liste, c'est-à-dire : si `a` est par exemple une liste de longueur 10, retourner les éléments de 6 à 9.

**Indication :** Voir la documentation [ici](#).

c) Chercher dans la documentation la syntaxe pour retourner la longueur d'une liste.

d) Écrire une fonction `fibonacci(N)` qui retourne la liste des  $N \geq 2$  premiers termes de la suite de Fibonacci définie par  $u_{n+2} = u_{n+1} + u_n$  avec  $u_0 = 0$  et  $u_1 = 1$ .

e) Écrire une fonction `pascal(N)` qui retourne la  $N$ -ième ligne du triangle de Pascal :

```

      1
     1 1
    1 2 1
   1 3 3 1
  1 4 6 4 1
 1 5 10 10 5 1
```

f) Soit les suites  $(u_n)_{n \in \mathbb{N}}$  et  $(v_n)_{n \in \mathbb{N}}$  définies par  $u_0 = 1$ ,  $v_0 = 1$ , et

$$u_{n+1} = u_n + v_n, \quad v_{n+1} = 2u_n - v_n,$$

pour  $n \geq 0$ . Calculer  $u_{100}$  et  $v_{100}$ .

**Réponse :**  $u_{100} = v_{100} = 717897987691852588770249$

g) Écrire une fonction `vk(n0,K)`, qui pour deux entiers  $n_0$  et  $K \geq 1$  calcule la suite des valeurs  $v_k$  définies par  $v_0 = n_0$  et

$$v_{k+1} = \begin{cases} 3v_k + 1 & \text{si } v_k \text{ est impair,} \\ \frac{v_k}{2} & \text{si } v_k \text{ est pair,} \end{cases}$$

pour  $0 \leq k < K$ . Pour  $K = 1\,000$  et diverses valeurs de  $n_0 \in \{10, 100, 1\,000, 10\,000\}$ , afficher les cinq dernières valeurs calculées, c'est-à-dire  $(v_{K-4}, v_{K-3}, v_{K-2}, v_{K-1}, v_K)$ .

**Réponse :** Les assertions suivantes sont vraies :

```
vk(10,1000) == [1, 4, 2, 1, 4]
vk(100,1000) == [2, 1, 4, 2, 1]
vk(1000,1000) == [1, 4, 2, 1, 4]
vk(10000,1000) == [4, 2, 1, 4, 2]
```

## Exercice 2.2 : Tuples

Les tuples permettent tout comme les listes de stocker des éléments hétérogènes :

```
tuple0 = (0, 5.4, "chaîne", True)
```

Mais au contraire des listes, les tuples ne sont pas mutables. Il n'est pas possible d'y modifier un élément, d'en rajouter un ou d'en supprimer un sans redéfinir tout le tuple. L'avantage d'un tuple sur une liste est qu'il est hashable, c'est-à-dire qu'il peut être utilisé comme clef dans un dictionnaire.

Enfin il est possible d'affecter des variables à l'intérieur d'un tuple, par exemple :

```
(a,b) = (1,9)
```

Cela est en particulier très utile pour échanger deux variables sans avoir à utiliser une variable supplémentaire :

```
(a,b) = (b,a)
```

a) Vérifier qu'un tuple est bien immuable.

b) Définir une fonction `mdlast(lst,val)` ayant pour argument une liste de tuples d'entiers `lst` et un entier `val` et retourner la liste de tuples avec le dernier élément de chaque tuple remplacé par `val`. Par exemple si `lst = [(10, 20), (30, 40, 50, 60), (70, 80, 90)]` alors `mdlast(lst,100)` doit retourner `[(10, 100), (30, 40, 50, 100), (70, 80, 100)]`.

c) Comment convertir un tuple en liste et réciproquement ?

## Exercice 2.3 : Ensembles

Les ensembles permettent de stocker des éléments hétérogènes au sens mathématique de la théorie des ensembles :

```
set0 = {0, 5.4, "chaîne", True}
```

Il est possible de tester si un élément appartient à un ensemble :

```
if "chaîne" in set0:
    print("dedans")
```

Les ensembles sont mutables, il est donc possible de rajouter ou retirer un élément d'un ensemble :

```
set0.add(18) # ajoute 18 à l'ensemble
set0.add(0) # ajoute 0 à l'ensemble (ne fait rien car 0 est déjà dedans)
set0.remove("chaîne") # retire "chaîne" de l'ensemble
```

En revanche les ensembles ne peuvent contenir que des éléments hashables, c'est-à-dire immutables. En particulier un ensemble ne peut pas contenir un autre ensemble :

```
set1 = {{1,2},{3},{4}}
TypeError: unhashable type: 'set'
```

À noter qu'il existe également en Python des ensembles immutables `frozenset` :

```
frozenset0 = frozenset([0, 5.4, "chaîne", True])
```

Une chaîne de caractères peut être transformée en ensemble :

```
set1 = set('abracadabra')
```

Comme pour les listes, il est possible de faire des compréhensions d'ensembles :

```
set2 = {x for x in 'abracadabra' if x not in 'abc'}
```

Dans cet exemple les chaînes de caractères sont automatiquement transformées en ensemble. À noter que l'ensemble vide est défini par `set()`.

- a) Définir une fonction `divisible(n)` qui retourne l'ensemble des nombres entiers divisibles par `n` inférieurs ou égaux à 100.
- b) Chercher dans la documentation comment faire l'intersection, l'union, et la différence de deux ensembles. Déterminer les nombres inférieurs ou égaux à 100 qui sont non divisibles par 2 mais divisibles par 3 et 5.

**Indication :** Voir la documentation de `set` [ici](#).

### Exercice 2.4 : Dictionnaires

Les dictionnaires sont une structure permettant de stocker des éléments hétérogènes indexés par des clefs (elles aussi hétérogènes) :

```
dict0 = {"pommes": 0, "poires": 4, 12: 2}
```

Les éléments d'un dictionnaire sont accessibles par les clefs :

```
dict0["pommes"]
dict0[12]
```

Un dictionnaire peut être vu comme un tableau associatif associant à chaque clef une valeur. La liste des clefs et celle des valeurs sont accessibles respectivement avec `dict0.keys()` et `dict0.values()`. Les dictionnaires sont mutables, il est donc possible de modifier une association clef-valeur et d'en rajouter ou supprimer une :

## 2 Structures de données

```
dict0["pommes"] = 3 # modifie la valeur associée à pommes
dict0["oranges"] = "beaucoup" # rajoute orange comme clef avec la valeur
↪ "beaucoup"
del dict0["poires"] # supprime le couple clef-valeur associé à poires
dict0.pop("pommes") # supprime le couple clef-valeur associé à pommes
```

Bien qu'un dictionnaire soit mutable, les clefs qui le composent doivent être des objets hashables, c'est-à-dire immutables. Ainsi une liste ou un ensemble ne peuvent pas servir de clefs dans un dictionnaire :

```
dict0[list0] = "test"
TypeError: unhashable type: 'list'
dict0[set0] = "retest"
TypeError: unhashable type: 'set'
```

En revanche il est possible d'avoir un tuple ou un frozenset comme clef :

```
dict0[tuple0] = "test"
dict0[frozenset0] = "rest"
```

d'où l'intérêt des frozensets. Comme pour les listes et les ensembles, il est possible de faire des compréhensions de dictionnaires :

```
dict1 = {x: x**2 for x in range(5)}
```

Finalement une chose intéressante avec les dictionnaires est l'unpacking illustré par l'exemple suivant :

```
def add(a=0, b=0):
    return a + b
d = {'a': 2, 'b': 3}
add(**d)
```

- a) Comment définir un dictionnaire vide ?
- b) Comment concaténer plusieurs dictionnaires entre eux ?
- c) On considère une liste de mots :

```
mots = ['Abricot', 'Airelle', 'Ananas', 'Banane', 'Cassis', 'Cerise', 'Citron', \
        'Clémentine', 'Coing', 'Datte', 'Fraise', 'Framboise', 'Grenade', 'Groseille', \
        'Kaki', 'Kiwi', 'Litchi', 'Mandarine', 'Mangue', 'Melon', 'Mirabelle',
        ↪ 'Nectarine', \
        'Orange', 'Pamplemousse', 'Papaye', 'Pêche', 'Poire', 'Pomme', 'Prune',
        ↪ 'Raisin']
```

Écrire une fonction `position(mots, x, n)` qui retourne la liste des mots ayant le caractère `x` comme `n`-ième lettre.

**Réponse :** Par exemple `position(mots, 'e', 4)` doit retourner la liste :

```
['Clémentine', 'Datte', 'Groseille', 'Pêche', 'Poire', 'Pomme', 'Prune']
```



**d)** En imaginant que la liste des mots soit très longue, alors à chaque évaluation de la fonction `position` l'ensemble des mots est parcouru, ce qui prend pas mal de temps. Pour améliorer cela, construire un dictionnaire `mots_dict` ayant pour clefs les tuples `(x,n)` et comme valeurs la liste des mots ayant le caractère `x` comme `n`-ième lettre, c'est-à-dire tel que `mots_dict[x,n]` retourne la même chose que `position(mots, x , n)` à l'ordre près. Ainsi la liste `mots` n'est parcourue qu'une seule fois lors de la construction du dictionnaire et ensuite l'évaluation du dictionnaire est extrêmement rapide pour n'importe quelle requête.

# Structures homogènes 3

Les structures de données par défaut de Python permettent de gérer des données hétérogènes (par exemple des entiers et des chaînes de caractères). Cette particularité fait que les structures de données Python sont extrêmement flexibles, au détriment de la performance. En effet vu que des données hétérogènes doivent pouvoir être supportées, il n'est pas possible d'allouer une plage de mémoire fixe pour une structure de données, ce qui ralentit son utilisation. Particulièrement en mathématiques, il apparaît très régulièrement des ensembles de données homogènes de tailles fixes (liste d'entiers, vecteurs réels ou complexes, matrices...). Le module Numpy définit le type `ndarray` qui est optimisé pour de telles structures de données homogènes de tailles fixes. La documentation de Numpy est disponible [ici](#).

Pour charger le module Numpy, il est d'usage de procéder ainsi :

```
import numpy as np
```

## Concepts abordés

- tableau de données homogènes
- slicing
- opérations vectorielles
- indexage et sélection
- représentations graphiques
- optimisation par parallélisation

## Exercice 3.1 : Introduction à Numpy

**Création.** La taille et le type des éléments d'un tableau Numpy doivent être connus à l'avance. La première façon de créer un tableau Numpy est de construire un tableau rempli de zéros en spécifiant la taille et le type :

```
array0 = np.zeros(3, dtype=int) # vecteur de 3 entiers
array1 = np.zeros((2,4), dtype=float) # tableau de flottants de taille 2x4
array2 = np.zeros((2,2), dtype=complex) # matrice carrée complexe de taille 2x2
array3 = np.zeros((5,6,4)) # tableau tridimensionnel de flottants
```

La seconde façon est de passer directement les données :

### 3 Structures homogènes

```
array4 = np.array([1,4,5]) # vecteur d'entiers (1,4,5)
array5 = np.array([[1.1,2.2,3.3,4.4],[1,2,3,4]]) # matrice de taille 2x4 de
↪ flottants
array6 = np.array([[1+1j,0.4],[3,1.5]]) # matrice complexe de taille 2x2
```

Numpy va alors déterminer lui-même le type et la taille du tableau. À noter qu'il est possible de forcer le type :

```
array0 = np.array([1,4,5], dtype=complex) # vecteur de complexes
```

Le type des éléments du tableau Numpy `array1` peut être déterminé par `array1.dtype`. La taille de ce tableau est donnée par `array1.shape`. Les commandes suivantes permettent d'accéder aux éléments des tableaux :

```
array4[1] # retourne 4
array5[1,3] # retourne 4.0
```

À noter que les indices commencent à 0 et non pas à 1. Les tableaux Numpy sont mutables dans le sens où les données peuvent être modifiées mais en conservant le même type et la même taille :

```
array0[1] = 4
array1[1,3] = 3.3
array3[3,4,2] = 3
```

**Slicing.** Le slicing permet d'accéder à certaines parties d'un tableau :

```
array4[2:3] # retourne les éléments d'indices compris entre 2 et 3
array1[0,:] # retourne la première ligne de array1
array1[:,-1] # retourne la dernière colonne de array1
array3[3,3:5,1:4] # retourne la sous-matrice correspondante
```

**Itération.** Il est possible d'itérer un tableau sur sa première dimension, par exemple pour retourner la somme des lignes :

```
for i in array5:
    print(np.sum(i))
```

a) Étudier la documentation de la fonction `arange` et utiliser cette fonction pour générer les vecteurs (5,6,7,8,9) et (3,5,7,9).

**Indication :** La documentation de la fonction `arange` est disponible [ici](#).

b) Étudier la documentation de la fonction `linspace` et l'utiliser pour générer 10 points équidistribués dans l'intervalle [2, 5].

c) Lire la documentation de la fonction `reshape` et effectuer successivement les transformations suivantes :

$$(1, 2, 3, 4, 5, 6) \rightarrow \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix} \rightarrow \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} \rightarrow \begin{pmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{pmatrix}$$

## Exercice 3.2 : Opérations sur les tableaux

Les opérations arithmétiques de base sur les tableaux Numpy sont effectuées éléments par éléments :

```
mat1 = np.array([[1,2.5,3],[5,6.1,8],[3,2,5]])
mat2 = np.array([[1,0.5,0],[0,0.9,8],[2,0,0]])
mat1 + mat2 # retourne la somme élément par élément
mat1 * mat2 # retourne le produit élément par élément (pas le produit
             ↪ matriciel)
10*mat1**2 # retourne 10 fois le carré des éléments de mat1
```

La plupart des fonctions mathématiques définies par Numpy (voir [ici](#)) sont aussi effectuées éléments par éléments :

```
np.cos(mat1) # retourne le cosinus élément par élément de mat1
np.exp(mat1) # retourne l'exponentielle élément par élément de mat1
```

Le produit matriciel peut être effectué d'une des trois façons suivantes :

```
np.dot(mat1,mat2)
mat1.dot(mat2)
mat1 @ mat2
```

a) Donné un vecteur  $(v_0, v_1, \dots, v_{n-1}) \in \mathbb{R}^n$  la dérivée discrète de ce vecteur est définie par le vecteur  $(d_0, d_1, \dots, d_{n-2}) \in \mathbb{R}^{n-1}$  donné par  $d_i = v_{i+1} - v_i$  pour  $i = 0, 1, \dots, n-2$ .

Écrire une fonction `diff_list` qui calcule la dérivée discrète d'une liste et une fonction `diff_np` qui fait la même opération mais sur des vecteurs Numpy en utilisant le slicing.

b) Soit `a_list` et `a_np` respectivement une liste et un tableau de 1000 éléments tirés au hasard dans l'intervalle  $[0,1]$  :

```
a_list = [np.random.random() for _ in range(1000)]
a_np = np.random.random(1000)
```

Comparer le temps d'exécution de `diff_list(a_list)` et de `diff_np(a_np)`.

**Indication :** Dans Jupyter Lab, il est très facile de déterminer le temps pris par une cellule pour s'évaluer, il suffit de commencer la cellule par `%%time`, par exemple :

```
%%time
result = diff_list(a_list)
```

Pour évaluer la cellule à de multiples reprises et faire une moyenne sur le temps d'exécution afin d'obtenir un résultat plus précis, remplacer `%%time` par `%%timeit`. La documentation est disponible [ici](#).

**Réponse :** Le temps d'exécution avec les tableaux Numpy devrait être approximativement 50 à 100 fois plus rapide qu'avec les listes !

## Exercice 3.3 : Représentations graphiques

Le module `matplotlib` permet de faire des représentations graphiques très variées. Pour l'utiliser, il est d'usage de l'importer ainsi :

### 3 Structures homogènes

```
%matplotlib inline
import matplotlib.pyplot as plt
```

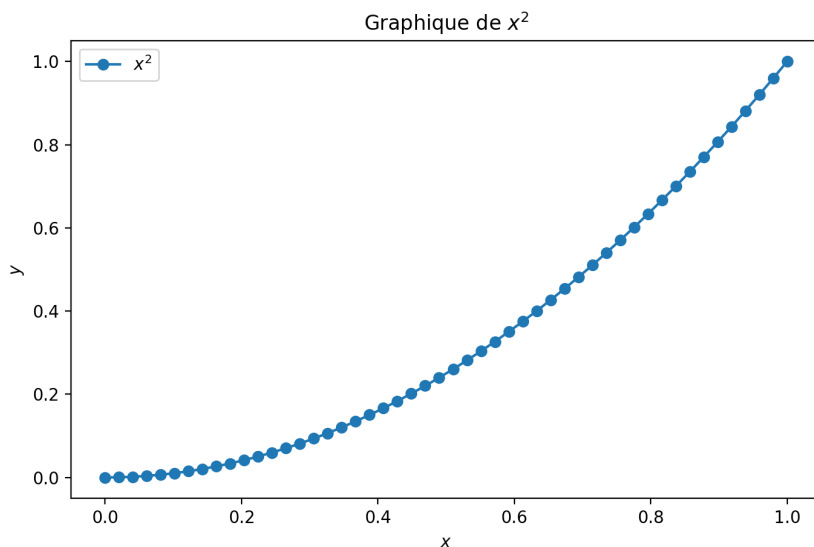
A noter que la première ligne permet de représenter les graphiques directement dans Jupyter Lab, mais n'est pas indispensable.

Par exemple la fonction `plot` peut être utilisée pour représenter la fonction  $x^2$  :

```
x = np.linspace(0,1,50)
y = x**2
plt.plot(x,y)
plt.show()
```

Afin de définir une jolie figure pouvant être exportée, la syntaxe est la suivante :

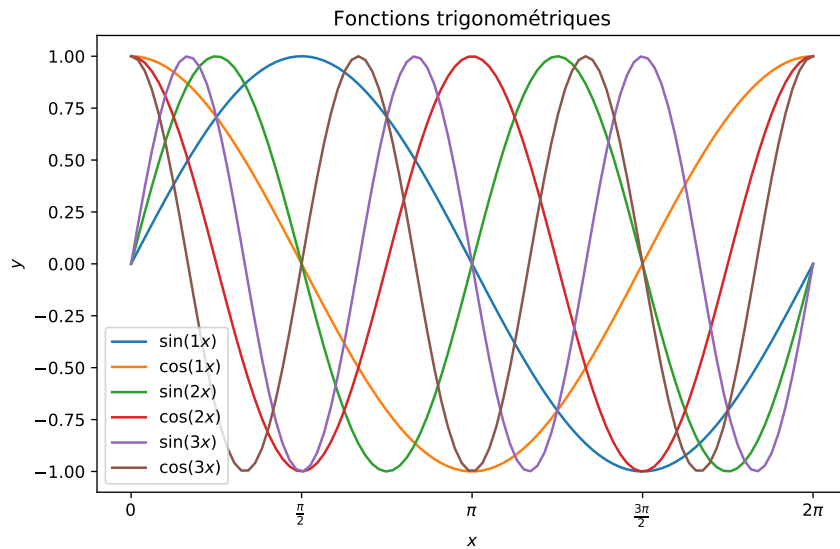
```
plt.figure(figsize=(8,5)) # taille de la figure (en inches)
plt.title(r'Graphique de  $x^2$ ') # titre de la figure (du code LaTeX peut être
↪ inclus)
plt.xlabel(r'$x$') # titre de l'axe horizontal
plt.ylabel(r'$y$') # titre de l'axe vertical
plt.plot(x, y, marker='o', label=r"$x^2$") # légende
plt.legend() # affiche la légende
plt.savefig("test.pdf") # exporte la figure en PDF
plt.savefig("test.png", dpi=100) # exporte la figure en PNG
plt.show()
```



La documentation de Matplotlib est disponible [ici](#).

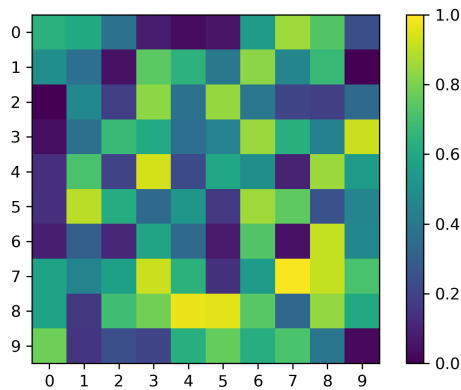
**a)** Représenter graphiquement sur la même figure les fonctions  $\sin(kx)$  et  $\cos(kx)$  pour  $k = 1, 2, 3$  pour  $x \in [0, 2\pi]$ . Faire en sorte que les graduations sur l'axe horizontal soient tous les  $\frac{\pi}{2}$  :

### 3 Structures homogènes



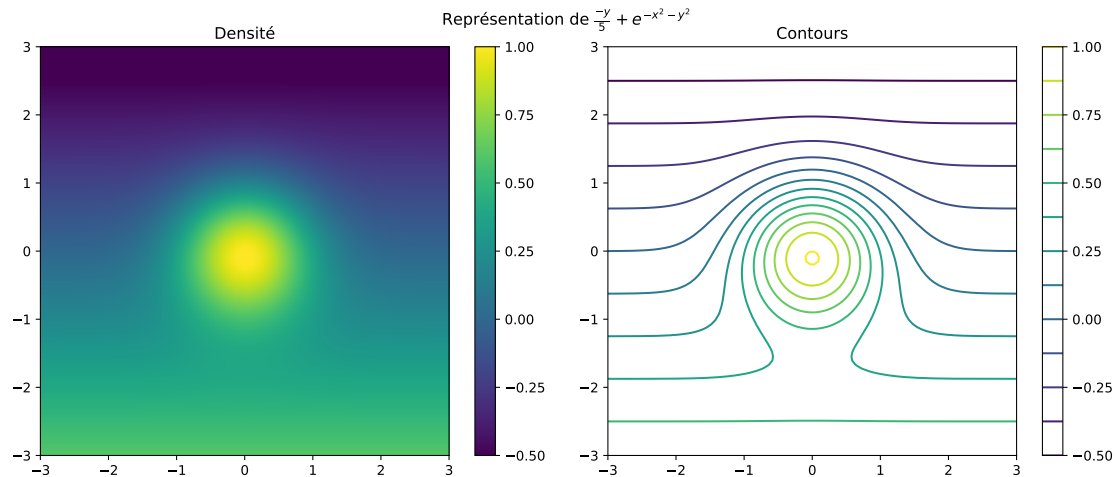
**Indication :** Utiliser la fonction `xticks` ou les fonctions `set_xticks` et `set_xticklabels` décrites [ici](#).

**b)** Regarder l'aide de la fonction `imshow` et l'utiliser pour représenter graphiquement une matrice de nombres aléatoires dans  $[0, 1]$  de taille  $10 \times 10$  :



**c)** ! Représenter graphiquement la fonction  $f(x, y) = \frac{-y}{5} + e^{-x^2 - y^2}$  pour  $x \in [-3, 3]$  et  $y \in [-3, 3]$  en densité et avec des courbes de niveau :

### 3 Structures homogènes



d) !! Regarder les exemples disponibles [ici](#) et en choisir deux à comprendre et à modifier.

### Exercice 3.4 : ! Indexage de tableaux

Le slicing permet de sélectionner des blocs dans un tableau, mais il est également possible de sélectionner des éléments disparates en utilisant un tableau comme indexage :

```
a = np.arange(12)**2 # tableau des carrés parfaits
i = np.array([1,3,8,5]) # tableau d'indices
a[i] # tableau des éléments de a aux places i
```

À noter qu'il est également possible d'indexer par un tableau de dimension supérieure. Le résultat est alors un tableau de la même forme que l'index :

```
j = np.array([[3,4],[9,7]]) # tableau bidimensionnel d'indices
a[j] # sélectionne les éléments de a avec les indices j
```

Pour un tableau à plusieurs dimensions :

```
b = np.array([[0,1,2,3],[4,5,6,7],[8,9,10,11]])
i = np.array([0,1,2,2]) # tableau des premiers indices
j = np.array([1,0,3,1]) # tableau des seconds indices
b[i,j] # sélectionne les éléments d'indices ij
```

Enfin il est possible d'indexer un tableau pour un tableau de booléens :

```
c = np.array([[0,1,2,3],[4,5,6,7],[8,9,10,11]])
cond = (c >= 5) # tableau de booléens valant True si >= 5 et False sinon
c[cond] = 5 # assigne la valeur 5 à toutes les entrées de c plus grandes que 5
```

Pour la suite, on considère les nombres :

```
[0.9602, -0.99, 0.2837, 0.9602, 0.7539, -0.1455, -0.99, -0.9111, 0.9602,
 -0.1455, -0.99, 0.5403, -0.99, 0.9602, 0.2837, -0.99, 0.2837, 0.9602]
```

comme étant les résultats d'une mesure effectuée toutes les 0.1 seconde aux temps compris entre 2 et 3.7 secondes.

### 3 Structures homogènes

- a)** Les mesures étant censées être positives, modifier les données pour mettre 0 lorsque les valeurs sont négatives.
- b)** Calculer les temps pour lesquels les mesures précédentes sont maximales.
- c)** Pour chaque mesure maximale retourner la mesure précédente, la mesure maximale et la mesure suivante. Si la mesure précédente ou la suivante n'existent pas, les remplacer par `np.nan`.



# Calcul symbolique 4

En tant que langage généraliste, Python n'inclut pas par défaut certains concepts mathématiques. Un exemple déjà vu concerne les vecteurs et les matrices numériques qui sont implémentées dans le module Numpy. Le but ici est d'introduire le module Sympy qui permet de faire du calcul symbolique.

## Concepts abordés

- symboles
- fonctions
- simplification
- analyse infinitésimale (limite, dérivation, intégration, développement limité)
- preuve assistée par ordinateur
- fonction pathologique
- fonction de Green
- coordonnées sphériques

Par exemple, le nombre  $\sqrt{8}$  est représenté par Python comme un flottant :

```
import math
math.sqrt(8)
```

L'avantage de Sympy est que  $\sqrt{8}$  est gardé en tant que racine et même automatiquement simplifié :

```
import sympy as sp
sp.init_printing()
sp.sqrt(8)
```

À noter que la deuxième instruction n'est pas nécessaire, mais permet de présenter les résultats de manière plus élégante dans Jupyter Lab. Il est également assez courant d'importer toutes les fonctions de Sympy afin de ne pas avoir à mettre `sp.` devant chaque fonction :

```
from sympy import *
init_printing()
sqrt(8)
```

mais il faut alors faire attention aux éventuels conflits entre fonctions. La documentation de Sympy est disponible [ici](#).

## Exercice 4.1 : Introduction à Sympy

Avant de pouvoir utiliser des variables symboliques, il faut les déclarer comme symboles :

```
x = sp.Symbol("x") # définit le symbole x
y = sp.Symbol("y", real=True) # définit la variable réelle y
e = sp.Symbol(r"\varepsilon", real=True, positive=True) # définit epsilon
↪ positif
```

Ensuite il est possible de faire des opérations entre symboles :

```
x + 2*y + e/4 + x**2 + 3*x + 2*y
```

La plupart des fonctions mathématiques sont implémentées symboliquement dans Sympy et il est également possible de les simplifier :

```
expr = sp.cos(x)**2 + sp.sin(x)**2 + (y**3 + y**2 - y - 1)/(y**2 + 2*y + 1) +
↪ sp.exp(-e)
sp.simplify(expr)
```

Finalement il est possible de faire des substitutions :

```
expr.subs(x,y) # remplace x par y
expr.subs({y:x, e:y}) # remplace y par x et e par y
```

puis par exemple de simplifier l'expression et de tracer son graphe en fonction de  $x$  comme sur la 4.1 :

```
f = sp.simplify(expr.subs({y:x, e:y}))
sp.plot(f,(x,-2,6), title=f"Graphique de ${sp.latex(f)}$")
```

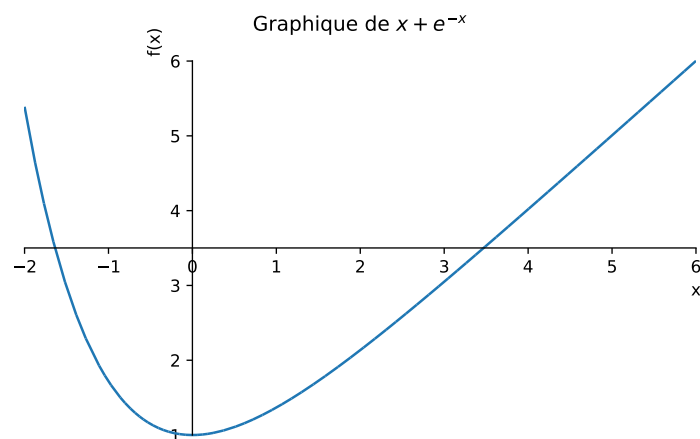


FIGURE 4.1 – Représentation graphique d'une fonction Sympy.

## 4 Calcul symbolique

a) Lire la documentation de la fonction `solve` et l'utiliser pour calculer les racines d'un polynôme général de degré deux, puis de degré trois.

**Indication :** La documentation sur la résolution d'équations algébriques est disponible [ici](#).

b) Lire la documentation des fonctions `evalf` et `N` pour évaluer l'expression  $\frac{\pi^2}{4}$ .

**Indication :** La documentation sur l'évaluation numérique est disponible [ici](#).

c) Déterminer la partie réelle et imaginaire de l'expression :

$$\left( \frac{1 + i\sqrt{3}}{1 + i} \right)^{20}.$$

**Indication :** Voir la documentation [ici](#).

d) Lire la documentation de la fonction `diff` et calculer la dérivée de  $xe^{x^x}$  par rapport à  $x$ .

**Indication :** La documentation sur les dérivées est disponible [ici](#).

e) Lire la documentation de la fonction `integrate` et calculer les intégrales suivantes :

$$I_1 = \int x^5 \sin(x) \, dx, \quad I_2 = \int_0^\infty \sin(x^2) \, dx.$$

f) Calculer avec Sympy les limites suivantes :

$$L_1 = \lim_{x \rightarrow 0} \frac{\sin(x)}{x}, \quad L_2 = \lim_{x \rightarrow 0} \sin\left(\frac{1}{x}\right), \quad L_3 = \lim_{x \rightarrow \infty} \frac{5x^2 + 3x + 2y}{y(x-4)(x-y)}.$$

g) Calculer le développement limité de  $\tan(x)$  en  $x = 0$  à l'ordre 10 et le développement asymptotique de  $\left(1 + \frac{1}{n}\right)^n$  pour  $n \rightarrow \infty$  à l'ordre 5.

h) Déterminer les valeurs propres de la matrice :

$$\begin{pmatrix} 1 & a & 0 \\ a & 2 & a \\ 0 & a & 3 \end{pmatrix}$$

**Indication :** La documentation sur les matrices symboliques est disponible [ici](#).

### Exercice 4.2 : Applications

Le but est d'utiliser Sympy pour résoudre symboliquement différents problèmes mathématiques en calculant le moins de choses possibles à la main.

a) Déterminer le nombre de zéros que contient l'entier 123!.

b) Déterminer le rapport entre la hauteur et le rayon d'un cylindre de manière à minimiser son aire à volume fixé.

c) Pour  $x, y \in \mathbb{R}$  tels que  $xy < 1$ , démontrer que :

$$\arctan(x) + \arctan(y) = \arctan\left(\frac{x+y}{1-xy}\right).$$

## 4 Calcul symbolique

**Indication :** Dériver l'équation par rapport à  $x$  et justifier.

d) Démontrer la formule suivante due à Gauss :

$$\frac{\pi}{4} = 12 \arctan\left(\frac{1}{38}\right) + 20 \arctan\left(\frac{1}{57}\right) + 7 \arctan\left(\frac{1}{239}\right) + 24 \arctan\left(\frac{1}{268}\right).$$

Il est impératif d'utiliser Sympy, la démonstration originale de Gauss faisant 25 pages, voir les pages 477 à 502 du deuxième volume de ses œuvres complètes disponible [ici](#).

**Indication :** Appliquer la fonction tangente de chaque côté de l'équation puis simplifier. La documentation sur les différentes fonctions de simplifications est disponible [ici](#).

e) Déterminer le volume de la région :

$$\{(x, y, z) \in \mathbb{R}^3 : x^2 + y^2 < z < 2x^2 + 4xy + 6y^2, |y| < 5, |x| < 4\}.$$

f) Déterminer l'expression des coefficients de Fourier réels de la fonction  $2\pi$ -périodique  $f$  définie par  $f(x) = |\sin(x)|$ .

### Exercice 4.3 : Fonction pathologique

Le but est de construire une fonction qui visuellement semble régulière, mais qui en fait ne l'est pas. Soit la fonction  $f : \mathbb{R} \rightarrow \mathbb{R}$  définie par :

$$f(x) = \sum_{k=1}^{\infty} \frac{\sin(k^2 x)}{k^5}.$$

Vu que la série converge absolument, la fonction  $f$  est bien définie.

a) À l'aide de Sympy calculer la fonction  $g : \mathbb{R} \rightarrow \mathbb{R}$  définie en gardant les cent premiers termes de la série :

$$g(x) = \sum_{k=1}^{100} \frac{\sin(k^2 x)}{k^5},$$

et représenter la fonction  $g$  graphiquement.

b) Estimer à la main l'erreur entre les fonctions  $f$  et  $g$  en valeur absolue.

c) Calculer la dérivée première et la dérivée seconde de  $g$  et représenter graphiquement ces deux dérivées. Que pouvez-vous conclure ?

d) Expliquer mathématiquement ce qui se passe.

### Exercice 4.4 : ! Fonction de Green du laplacien

Le but de cet exercice est de calculer entièrement automatiquement la fonction de Green du laplacien dans  $\mathbb{R}^3$ , *i.e.* la solution satisfaisant :

$$\Delta G(\mathbf{x}) = \delta(\mathbf{x}),$$

dans  $\mathbb{R}^3$  où  $\delta(\mathbf{x})$  est la distribution de Dirac.

Pour cela, nous introduisons les coordonnées sphériques  $\mathbf{x}' = (r, \theta, \varphi)$  avec  $r > 0$ ,  $0 \leq \theta \leq \pi$ , et  $0 \leq \varphi < 2\pi$  caractérisées par :

$$x_1 = r \cos \varphi \sin \theta$$

$$x_2 = r \sin \varphi \sin \theta$$

$$x_3 = r \cos \theta.$$

#### 4 Calcul symbolique

a) Définir une fonction `to_spherical(expr)` permettant de convertir en coordonnées sphériques une expression donnée en coordonnées cartésiennes.

b) Définir une fonction `to_cartesian(expr)` permettant de convertir en coordonnées cartésiennes une expression donnée en coordonnées sphériques.

c) Calculer les facteurs d'échelle des coordonnées sphériques :

$$h_i = \left\| \frac{\partial \mathbf{x}}{\partial x'_i} \right\|.$$

d) Définir une fonction `gradient(f)` permettant de calculer le gradient d'une fonction  $f : \mathbb{R}^3 \rightarrow \mathbb{R}$  en coordonnées sphériques :

$$\nabla f = \left( \frac{1}{h_i} \frac{\partial f}{\partial x'_i} \right)_{i=1}^3.$$

e) Faire de même pour définir le laplacien en coordonnées sphériques :

$$\Delta f = \sum_{i=1}^3 \frac{1}{J} \frac{\partial}{\partial x'_i} \left( \frac{J}{h_i^2} \frac{\partial f}{\partial x'_i} \right) \quad \text{où} \quad J = \prod_{i=1}^3 h_i.$$

f) Trouver les solutions radiales (*i.e.* ne dépendant que de la variable  $r$ ) de l'équation  $\Delta G = 0$  dans  $\mathbb{R}^3 \setminus \{\mathbf{0}\}$ .

**Indication :** Regarder la documentation de la fonction `dsolve` pour résoudre une équation différentielle.

g) Déterminer les équations que doivent satisfaire les constantes d'intégration pour que la solution précédente satisfasse en coordonnées cartésiennes :

$$\lim_{|\mathbf{x}| \rightarrow \infty} G(\mathbf{x}) = 0 \quad \text{et} \quad \Delta G(\mathbf{x}) = \delta(\mathbf{x}).$$

**Indication :** Il faut transformer les deux conditions en coordonnées sphériques. La première condition s'exprime en coordonnées sphériques par :

$$\lim_{r \rightarrow \infty} G(r) = 0,$$

et la seconde est équivalente à :

$$\lim_{r \rightarrow 0} \int_0^\pi \int_0^{2\pi} (\nabla G(r) \cdot \mathbf{e}_r) J(r, \theta, \varphi) d\varphi d\theta = 1.$$

h) Résoudre les équations sur les constantes d'intégrations et substituer dans la solution radial de  $\Delta G = 0$  pour obtenir l'expression de la fonction de Green du laplacien en coordonnées sphériques. Finalement déterminer la fonction de Green  $G$  du laplacien dans  $\mathbb{R}^3$  en coordonnées cartésiennes.

i) **!!** Soit  $g : \mathbb{R}^3 \rightarrow \mathbb{R}$  une fonction lisse à support compact invariante par rotations selon l'axe vertical. Déterminer le comportement asymptotique à grandes distances de la solution de l'équation :

$$\Delta f(\mathbf{x}) = g(\mathbf{x})$$

jusqu'à l'ordre deux, *i.e.* les termes décroissant comme  $|\mathbf{x}|^{-1}$  et comme  $|\mathbf{x}|^{-2}$ .

#### 4 Calcul symbolique

**Indication :** La solution est donnée par la formule de Green :

$$f(\mathbf{x}) = \int_{\mathbb{R}^3} G(\mathbf{x} - \mathbf{x}_0)g(\mathbf{x}_0) d^3\mathbf{x}_0 = \int_{B_R} G(\mathbf{x} - \mathbf{x}_0)g(\mathbf{x}_0) d^3\mathbf{x}_0 ,$$

où  $R$  est choisi de sorte que le support de  $g$  soit contenu dans  $B_R$ . Pour calculer le développement asymptotique de cette intégrale, la première étape est de convertir  $G(\mathbf{x} - \mathbf{x}_0)$  en coordonnées sphériques pour  $\mathbf{x}$  et  $\mathbf{x}_0$ . Vu que  $g$  est invariante par rotations selon l'axe vertical, alors en coordonnées sphériques  $g$  est indépendante de  $\varphi$  et est donnée par  $g(r, \theta)$ . La deuxième étape est de transformer l'intégrale en coordonnées sphériques avec une intégrale triple sur  $r_0$ ,  $\theta_0$  et  $\varphi_0$ . La troisième étape est de calculer le développement asymptotique de l'intégrand lorsque  $r \rightarrow \infty$ , jusqu'à l'ordre deux. Finalement, vu que  $r_0$ ,  $\theta_0$  et  $\varphi_0$  sont bornés, alors l'intégration commute avec le développement asymptotique et le résultat final est donné par l'intégration individuelle des deux termes du développement asymptotique.

# Intégration 5

Le but est d'obtenir une approximation d'une intégrale définie du type

$$J = \int_a^b f(x)dx$$

pour une certaine fonction  $f : [a, b] \rightarrow \mathbb{R}$  trop compliquée pour *a priori* déterminer la valeur de  $J$  à la main. Des méthodes d'approximations déterministes et probabilistes seront introduites pour obtenir une approximation de  $J$ .

## Concepts abordés

- méthodes classiques (rectangles, trapèzes et Simpson)
- méthode de Monte-Carlo
- vitesse convergence
- statistiques

## Exercice 5.1 : Méthode des rectangles

La méthode des rectangles est basée sur la définition de l'intégrale au sens de Riemann. La première étape est de découper l'intervalle  $[a, b]$  en  $N$  intervalles  $[x_n, x_{n+1}]$  de même taille  $\delta = \frac{b-a}{N}$ , *i.e.*  $x_n = a + n\delta$  pour  $n \in \{0, 1, \dots, N-1\}$ . La seconde étape consiste à supposer que la fonction  $f$  est constante sur chaque intervalle  $[x_n, x_{n+1}]$ , donc à faire l'approximation

$$J_n = \int_{x_n}^{x_{n+1}} f(x)dx \approx \delta f(\tilde{x}_n)$$

pour  $\tilde{x}_n$  une certaine valeur à choisir dans l'intervalle  $[x_n, x_{n+1}]$ . Le choix de  $\tilde{x}_n$  peut par exemple être fait par  $\tilde{x}_n = x_n + \alpha\delta$  avec  $\alpha \in [0, 1]$ . Finalement l'approximation de  $J$  est donnée par la somme des approximations de  $J_n$ ,

$$\tilde{J} = \sum_{n=0}^{N-1} \delta f(\tilde{x}_n).$$

En supposant que  $f \in C^1([a, b])$ , alors il est possible de montrer que la méthode des rectangles converge et que sa vitesse de convergence est d'ordre un. Une méthode numérique est dite d'ordre  $k$  si l'erreur entre l'approximation numérique et le résultat exact est de l'ordre de  $\frac{1}{N^k}$ .

## 5 Intégration

**Indication :** Pour chaque valeur de  $n$ , et  $x \in [x_n, x_{n+1}]$ , par le théorème des accroissements finis, il existe  $c_n$  tel que :

$$f(x) - f(\tilde{x}_n) = (x - \tilde{x}_n)f'(c_n).$$

Comme  $f'$  est continue sur  $[a, b]$ , alors

$$\sup_{x \in [a, b]} |f'(x)| \leq M,$$

et donc

$$|f(x) - f(\tilde{x}_n)| \leq M|x - \tilde{x}_n| \leq M\delta \leq \frac{M(b-a)}{N}.$$

Ainsi

$$\begin{aligned} E_N &= \left| J - \tilde{J} \right| = \left| \sum_{n=0}^{N-1} \left( \int_{x_n}^{x_{n+1}} f(x) \, dx - \delta f(\tilde{x}_n) \right) \right| \\ &\leq \sum_{n=0}^{N-1} \left| \int_{x_n}^{x_{n+1}} (f(x) - f(\tilde{x}_n)) \, dx \right| \leq \sum_{n=0}^{N-1} \int_{x_n}^{x_{n+1}} |f(x) - f(\tilde{x}_n)| \, dx \\ &\leq \sum_{n=0}^{N-1} \int_{x_n}^{x_{n+1}} \frac{M(b-a)}{N} \, dx \leq \frac{M(b-a)^2}{N}. \end{aligned}$$

Par conséquent si  $f \in C^1([a, b])$ , alors la méthode des rectangles est d'ordre un.

**a)** Choisir une fonction continue  $f : [a, b] \rightarrow \mathbb{R}$  et définir la fonction Python `f(x)` correspondante. Pour tester le code, il est judicieux de choisir une fonction  $f$  dont l'intégrale peut être facilement calculable à la main.

**Indication :** La liste des fonctions mathématiques de base disponibles en Python dans le module `math` est disponible dans la documentation [ici](#). À noter que Numpy définit également des fonctions mathématiques, voir la documentation [ici](#).

**b)** Écrire une fonction `rectangles(f, a, b, N)` qui retourne l'approximation de l'intégrale  $J$  par la méthode des rectangles par exemple en choisissant  $\tilde{x}_n = x_n$ , *i.e.* le bord gauche de l'intervalle  $[x_n, x_{n+1}]$ .

**Indication :** Il n'est pas nécessaire de stocker toutes les valeurs des approximations de  $J_n$ , mais il est possible d'incrémenter une variable pour chaque approximation de  $J_n$ .

**c)** Modifier la fonction précédente pour que celle-ci prenne un paramètre optionnel `alpha` déterminant le choix du paramètre  $\alpha \in [0, 1]$ .

**d)** Écrire une fonction `plot_rectangles(f, a, b, N, alpha=0.5)` qui représente graphiquement l'approximation par la méthode des rectangles.

**e)** Déterminer empiriquement la vitesse de convergence de la méthode des rectangles en fonction de  $N$ .

### Exercice 5.2 : Méthode des trapèzes

La méthode des trapèzes est basée sur une approximation linéaire sur chaque intervalle  $[x_n, x_{n+1}]$ , plus spécifiquement :

$$J_n = \int_{x_n}^{x_{n+1}} f(x) \, dx \approx \delta \frac{f(x_n) + f(x_{n+1})}{2}.$$



- a) Écrire une fonction python `trapezes(f,a,b,N)` qui retourne l'approximation de l'intégrale  $J$  par la méthode des trapèzes. Tester la fonction `trapezes(f,a,b,N)` pour différentes fonctions  $f$ .
- b) L'implémentation de votre fonction `trapezes(f,a,b,N)` est-elle optimale quant au nombre d'évaluations de  $f$  effectuées par rapport au nombre d'évaluations nécessaires ? Une implémentation optimale de la fonction `trapezes(f,a,b,N)` devrait effectuer  $N + 1$  évaluations de  $f$ .
- c) Déterminer empiriquement la vitesse de convergence de la méthode des trapèzes en fonction de  $N$ .
- d) ! Déterminer analytiquement la convergence de la méthode des trapèzes. Quelles sont les hypothèses nécessaires sur  $f$  ?

### Exercice 5.3 : Méthode de Monte-Carlo

La méthode de Monte-Carlo (du nom des casinos, pas d'une personne) est une approche probabiliste permettant d'approximer la valeur d'une intégrale. L'idée de base est que l'intégrale  $J$  peut être vue comme l'espérance d'une variable aléatoire uniforme  $X$  sur l'intervalle  $[a, b]$  :

$$J = \int_a^b f(x) dx = (b - a) \mathbb{E}(f(X)).$$

Par la loi des grands nombres cette espérance peut être approximée par la moyenne empirique :

$$\tilde{J} = \frac{b - a}{N} \sum_{i=0}^{N-1} f(x_i),$$

où les  $x_i$  sont tirés aléatoirement dans l'intervalle  $[a, b]$  avec une loi de probabilité uniforme.

Il est possible de montrer que  $\tilde{J}$  converge vers  $J$  comme  $N^{-1/2}$  et cela indépendamment de la dimension et de la régularité de  $f$ .

**Indication :** Selon le théorème central limite, si  $Y_i$  est une suite de variables aléatoires indépendantes d'espérance  $\mu$  et de variance  $\sigma^2$ , alors la variable aléatoire :

$$S_N = \frac{1}{N} \sum_{i=0}^{N-1} Y_i,$$

a une espérance  $\mu$  et une variance :

$$\text{Var}(S_N) = \frac{\sigma^2}{N}.$$

En prenant  $Y_i = f(X_i)$  avec  $X_i$  une suite de variables aléatoires indépendantes uniformément distribuées sur  $[a, b]$ , alors l'espérance de  $Y_i$  est la moyenne de  $f$  et donc l'espérance de  $S_N$  est donnée par :

$$\mathbb{E}(S_N) = \frac{1}{b - a} \int_a^b f(x) dx.$$

La variance de  $Y_i$  est également celle de  $f$ ,  $\sigma^2 = \text{Var}(f(X))$  et donc la variance de  $S_N$  est :

$$\text{Var}(S_N) = \frac{\text{Var}(f(X))}{N}.$$

## 5 Intégration

Par conséquent, cela montre que  $\tilde{J}$  converge vers  $J$  comme  $N^{-1/2}$  vu que la variance est proportionnelle à  $N^{-1}$ .

A remarquer que pour établir ce résultat, aucune condition de régularité sur  $f$  n'est nécessaire, de l'intégrabilité suffit.

En pratique la variance de  $\tilde{J}$  peut être estimée par la variance empirique.

a) Écrire une fonction `montecarlo(f,a,b,N)` qui détermine une approximation  $\tilde{J}$  de  $J$  par la méthode de Monte-Carlo.

**Indication :** Pour générer un vecteur de nombres aléatoires, le sous-module `random` de Numpy peut être utile, voir la documentation [ici](#).

b) Modifier la fonction précédente, pour qu'elle retourne en plus de la moyenne  $\tilde{J}$  également la variance empirique :

$$\tilde{V} = \frac{(b-a)^2}{N} \sum_{i=0}^{N-1} \left( f(x_i) - \frac{\tilde{J}}{b-a} \right)^2.$$

c) Étudier empiriquement la convergence de la méthode de Monte-Carlo en fonction de  $N$  en faisant pour chaque valeur de  $N$  une statistique sur  $k$  exécutions. Plus précisément cela consiste à faire  $k$  évaluations de  $\tilde{J}$  par la méthode de Monte-Carlo et de calculer la moyenne et la variance des  $k$  résultats obtenus.

### Exercice 5.4 : ! Méthode de Simpson

La méthode de Simpson consiste à approximer la fonction  $f$  sur chaque intervalle  $[x_n, x_{n+1}]$  par un polynôme de degré deux. Le choix le plus naturel est le polynôme  $p_n$  de degré deux passant par les points  $(x_n, f(x_n))$ ,  $(\frac{x_n+x_{n+1}}{2}, f(\frac{x_n+x_{n+1}}{2}))$  et  $(x_{n+1}, f(x_{n+1}))$ .

a) Déterminer la forme explicite du polynôme  $p_n$ .

**Indication :** Le polynôme  $L(x) = \frac{(x-c)(x-b)}{(a-c)(a-b)}$  prend la valeur 1 en  $x = a$  et la valeur 0 en  $x = b$  et  $x = c$ . Faire une combinaison linéaire de trois polynômes de ce type.

b) Calculer l'approximation donnée par  $J_n \approx \int_{x_n}^{x_{n+1}} p_n(x) dx$ .

**Indication :** Il est possible de calculer cette intégrale à la main ou bien de le faire avec le module Sympy, voir la documentation [ici](#).

c) Simplifier à la main la somme  $\tilde{J}$  des approximations de  $J_n$ .

**Réponse :** Le résultat est :

$$\tilde{J} = \frac{\delta}{3} \left[ \frac{f(b) - f(a)}{2} + \sum_{n=0}^{N-1} \left( f(x_n) + 2f\left(\frac{x_n + x_{n+1}}{2}\right) \right) \right].$$

d) Écrire une fonction `simpson(f,a,b,N)` permettant de calculer une approximation de  $J$  avec la méthode de Simpson.

e) Comparer la précision des méthodes des rectangles, des trapèzes et de Simpson en fonction de  $N$ .

**Exercice 5.5 : !! Intégration avec Scipy**

Les méthodes d'intégrations précédentes et d'autres sont définies dans le module `integrate` de Scipy. Ce module permet en particulier de traiter des cas plus compliqués, par exemple calculer numériquement

$$E_n(x) = \int_1^\infty \frac{e^{-xt}}{t^n} dt$$

à l'aide du code suivant :

```
import math, scipy.integrate
def integrand(t, n, x):
    return math.exp(-x*t)/t**n
def E(n, x):
    return scipy.integrate.quad(integrand, 1, math.inf, args=(n, x))[0]
E(4,2)
```

- a) Lire la documentation de ce module disponible [ici](#).
- b) À l'aide de ce module, déterminer une approximation de l'intégrale double :

$$I = \int_0^\pi \left( \int_0^y x \sin(xy) dx \right) dy.$$

# Algèbre 6

Tout d'abord une méthode de résolution de système linéaire par un algorithme direct est étudiée, puis une méthode itérative sera utilisée pour déterminer le vecteur propre associé à la plus grande valeur propre d'une matrice. Enfin les groupes générés par un ensemble de permutations seront étudiés.

## Concepts abordés

- méthode de résolution directe (décomposition LU)
- algorithme *in place*
- méthode itérative (puissance itérée)
- groupes de permutations
- orbite et stabilisateur

## Exercice 6.1 : Décomposition LU

La décomposition LU consiste à décomposer une matrice  $A$  de taille  $n \times n$  sous la forme  $A = LU$  où  $L$  est une matrice triangulaire inférieure avec des 1 sur la diagonale et  $U$  une matrice triangulaire supérieure. Une fois la décomposition  $A = LU$  d'une matrice connue, il est alors très facile de résoudre le problème linéaire  $A\mathbf{x} = \mathbf{b}$  en résolvant d'abord  $L\mathbf{y} = \mathbf{b}$  puis  $U\mathbf{x} = \mathbf{y}$ . L'avantage de la décomposition LU sur l'algorithme de Gauss, par exemple, est qu'une fois la décomposition LU connue, il est possible de résoudre le système linéaire rapidement avec des seconds membres différents.

Vu que  $l_{ik} = 0$  si  $k > i$ , nous avons :

$$a_{ij} = \sum_{k=1}^n l_{ik}u_{kj} = l_{ii}u_{ij} + \sum_{k=1}^{i-1} l_{ik}u_{kj},$$

et donc comme  $l_{ii} = 1$  :

$$u_{ij} = a_{ij} - \sum_{k=1}^{i-1} l_{ik}u_{kj}.$$

D'un autre côté, vu que  $u_{kj} = 0$  si  $k > j$ , alors :

$$a_{ij} = \sum_{k=1}^n l_{ik}u_{kj} = l_{ij}u_{jj} + \sum_{k=1}^{j-1} l_{ik}u_{kj},$$

et donc si  $u_{jj} \neq 0$  :

$$l_{ij} = \frac{1}{u_{jj}} \left( a_{ij} - \sum_{k=1}^{j-1} l_{ik} u_{kj} \right).$$

Ainsi, si les  $(i-1)$  premières lignes de  $U$  et les  $(i-1)$  premières colonnes de  $L$  sont connues, il est possible de déterminer la  $i$ -ème ligne de  $U$  par :

$$u_{ij} = a_{ij} - \sum_{k=1}^{i-1} l_{ik} u_{kj}, \quad j \geq i,$$

puis la  $i$ -ème colonne de  $L$  par :

$$l_{ji} = \frac{1}{u_{ii}} \left( a_{ji} - \sum_{k=1}^{i-1} l_{jk} u_{ki} \right), \quad j > i.$$

Pour que la décomposition LU d'une matrice  $A$  soit possible il faut que les  $u_{ii}$  ne soient jamais nuls. Cela est effectivement le cas lorsque la matrice  $A$  et toutes ses sous-matrices principales sont inversibles.

- a) Écrire une fonction `LU(A)` qui retourne le résultat de la décomposition LU d'une matrice.
- b) Donnée la décomposition LU d'une matrice  $A$ , écrire une fonction `solve(L,U,b)` qui résout le système linéaire  $A\mathbf{x} = \mathbf{b}$ .
- c) Écrire une fonction `LU_inplace(A)` qui ne crée pas de nouveaux tableaux pour  $L$  et  $U$  mais modifie  $A$  de sorte que sa partie triangulaire inférieure (sans la diagonale) corresponde à  $L$  et sa partie triangulaire supérieure (avec la diagonale) corresponde à  $U$ . Faire également une version `solve_inplace` prenant en entrée la sortie de `LU_inplace` et retournant la solution  $\mathbf{x}$ , sans utiliser de nouveaux tableaux.
- d) En utilisant la décomposition LU de la matrice  $A$ , écrire une fonction `det(A)` qui retourne son déterminant.

## Exercice 6.2 : Méthode de la puissance itérée

Le but de cet exercice est de déterminer le vecteur propre d'une matrice associé à la valeur propre de plus grand module, en supposant que celle-ci est unique. Étant donné une matrice réelle  $A$  de taille  $n \times n$  et un vecteur  $\mathbf{x}_0 \in \mathbb{R}^n$ , la suite de vecteurs  $(\mathbf{x}_k)_{k \in \mathbb{N}}$  est définie par :

$$\mathbf{x}_{k+1} = \frac{A\mathbf{x}_k}{\|A\mathbf{x}_k\|}.$$

En supposant par exemple que la matrice  $A$  soit diagonalisable, alors il est possible de montrer que la suite  $(\mathbf{x}_k)_{k \in \mathbb{N}}$  converge à un signe près vers le vecteur propre associé à la plus grande valeur propre de  $A$  en valeur absolue.

**Indication :** La première étape est de remarquer que :

$$\mathbf{x}_k = \frac{A^2 \mathbf{x}_{k-2}}{\|A^2 \mathbf{x}_{k-2}\|} = \dots = \frac{A^k \mathbf{x}_0}{\|A^k \mathbf{x}_0\|}.$$

Vu que  $A$  est diagonalisable, soit  $(\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_n)$  une base de vecteurs propres de  $A$  associés aux valeurs propres  $\lambda_1, \lambda_2, \dots, \lambda_n$ . Sans perte de généralité, on suppose que  $\lambda_1$  est la valeur propre

## 6 Algèbre

de plus grand module, c'est-à-dire  $|\lambda_1| > \max(|\lambda_2|, |\lambda_3|, \dots, |\lambda_n|)$ . À noter que cela implique que  $\lambda_1$  est réelle. Le vecteur  $\mathbf{x}_0$  se décompose dans la base :

$$\mathbf{x}_0 = \sum_{i=1}^n c_i \mathbf{v}_i,$$

ainsi en supposant que  $c_1 \neq 0$  :

$$A^k \mathbf{x}_0 = \sum_{i=1}^n c_i \lambda_i^k \mathbf{v}_i = c_1 \lambda_1^k \left( \mathbf{v}_1 + \sum_{i=2}^n \frac{c_i}{c_1} \left( \frac{\lambda_i}{\lambda_1} \right)^k \mathbf{v}_i \right).$$

Vu que  $|\lambda_1| > |\lambda_i|$  pour  $i \geq 2$ , alors :

$$\lim_{k \rightarrow \infty} \left( \mathbf{v}_1 + \sum_{i=2}^n \frac{c_i}{c_1} \left( \frac{\lambda_i}{\lambda_1} \right)^k \mathbf{v}_i \right) = \mathbf{v}_1,$$

puisque  $\left| \frac{\lambda_i}{\lambda_1} \right| < 1$ . Par conséquent,

$$\lim_{k \rightarrow \infty} \text{sign}(\lambda_1)^k \mathbf{x}_k = \lim_{k \rightarrow \infty} \left( \frac{\lambda_1}{|\lambda_1|} \right)^k \frac{A^k \mathbf{x}_0}{\|A^k \mathbf{x}_0\|} = \text{sign } c_1 \frac{\mathbf{v}_1}{\|\mathbf{v}_1\|}.$$

En choisissant  $\mathbf{x}_0$  aléatoirement, alors avec probabilité un  $c_1 \neq 0$  et donc la suite  $(\mathbf{x}_k)_{k \in \mathbb{N}}$  converge à un signe près vers un vecteur propre normalisé associé à la valeur propre de plus grand module.

**a)** Définir une fonction `puissance(A, x0, k)` qui retourne  $\mathbf{x}_k$ . Avec cette fonction, déterminer le plus grand vecteur propre de la matrice :

$$A = \begin{pmatrix} 0.5 & 0.5 \\ 0.2 & 0.8 \end{pmatrix}.$$

**Réponse :** Le plus grand vecteur propre est  $\pm(0.70710678, 0.70710678)$ .

**b)** Déterminer la valeur propre associée au vecteur propre précédent.

**Indication :** Si  $\mathbf{v}$  est un vecteur propre normalisé de  $A$ , alors la valeur propre associée est donnée par  $\lambda = A\mathbf{v} \cdot \mathbf{v}$ .

**c)** Écrire un programme permettant de calculer automatiquement la valeur propre de plus grand module et le vecteur propre associé d'une matrice carrée avec une certaine précision donnée.

**d)** Regarder la documentation de Numpy pour trouver les fonctions permettant de calculer les vecteurs propres et valeurs propres d'une matrice.

**e)** Comparer la rapidité du code précédent et des fonctions Numpy pour des matrices de tailles  $n \times n$  avec  $n = 10, 100, 1\,000$ .

**Indication :** En prenant par exemple des matrices dont les coefficients sont générés aléatoirement dans l'intervalle  $(0, 1)$ , le théorème de Perron-Frobenius assure l'existence d'une unique valeur propre de module maximal, et celle-ci est positive.

### Exercice 6.3 : Groupes de permutations

Le but est d'étudier les groupes de permutations en développant des algorithmes pour les caractériser. Un groupe de permutations  $G \subset S_n$  est défini comme étant généré par un certain nombre de permutations :  $G = \langle g_1, g_2, \dots, g_k \rangle$ , avec  $g_i \in S_n$  une permutation de l'ensemble  $\{1, 2, \dots, n\}$ . Une permutation

$$g = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 4 & 1 & 2 & 3 \end{pmatrix},$$

peut être représentée en python par le tuple  $\mathbf{g} = (0, 4, 1, 2, 3)$ . Le zéro est ajouté afin de se conformer avec l'indexation à partir de zéro de Python et ainsi de simplifier un peu les implémentations. Cela veut simplement dire que le sommet 0 va sur le sommet 0. À noter que cet exercice se prête particulièrement bien à une implémentation orientée objet, et dans ce cas les questions peuvent être adaptées en conséquence.

- a) Écrire une fonction `product(g1,g2)` qui calcule le produit de deux permutations.
- b) Écrire une fonction `inverse(g)` qui calcule l'inverse d'une permutation.
- c) Écrire une fonction qui retourne la décomposition d'une permutation sous forme de cycles représentés par une liste de tuples.
- d) Écrire une fonction qui retourne la permutation correspondant à une liste de cycles, c'est-à-dire qui fait l'inverse de la fonction précédente.
- e) ! En python un groupe  $G = \langle g_1, g_2, \dots, g_k \rangle$  engendré par une famille de permutations, peut être représenté par une liste de permutations  $\mathbf{G} = [\mathbf{g1}, \mathbf{g2}, \dots, \mathbf{gk}]$ . Écrire une fonction `orbit(G,x)` qui retourne l'orbite d'un point  $x \in \{1, 2, \dots, n\}$  :

$$O_x = Gx = \{gx, g \in G\}.$$

**Indication :** Ne pas calculer l'ensemble des éléments du groupe, cela fait une liste beaucoup trop longue. Construire la liste  $(X^0, X^1, \dots, X^N)$  d'ensembles disjoints définie récursivement par  $X^0 = \{x\}$  et  $X^n$  comme l'ensemble des éléments nouveaux de la forme  $g_i y$  avec  $1 \leq i \leq k$  et  $y \in X^{n-1}$  :

$$X^n = \left( \bigcup_{i=1}^k g_i X^{n-1} \right) \setminus \left( \bigcup_{i=1}^{n-1} X^i \right).$$

- f) !! Définir une fonction `stabilizer(G,x)` qui retourne le stabilisateur d'un point  $x \in \{1, 2, \dots, n\}$  :

$$G_x = \{g \in G : gx = x\},$$

sous la forme d'un ensemble de générateurs.

**Indication :** Utiliser le théorème ou lemme de Schreier.

# Zéro de fonctions 7

Le but de cette série d'exercices est de déterminer les zéros de manière approchée d'une fonction notamment par la méthode de Newton. Cela permet en particulier de trouver des solutions approchées d'équations non linéaires. Cette méthode est fondamentale autant d'un point de vue numérique qu'analytique.

## Concepts abordés

- méthode de Newton en une et plusieurs dimensions
- matrice jacobienne
- attracteur de la méthode de Newton
- ensemble fractal
- optimisation par parallélisation
- équation différentielle non linéaire
- différences finies

## Exercice 7.1 : Méthode de Newton en une dimension

En une dimension, la méthode de Newton consiste à trouver une solution approchée d'une seule équation. Cette équation peut être mise sous la forme générale  $F(x) = 0$  où  $F : \mathbb{R} \rightarrow \mathbb{R}$  est une fonction assez régulière, donc le but est de trouver un zéro de la fonction  $F$ . L'équation  $F(x) = 0$  est équivalente (si  $F'(x) \neq 0$ ) à l'équation  $G(x) = x$  où  $G$  est la fonction définie par :

$$G(x) = x - \frac{F(x)}{F'(x)}.$$

La méthode de Newton consiste à trouver un point fixe de  $G$ , *i.e.* résoudre  $G(x) = x$  par itérations successives :

$$x_{i+1} = G(x_i) = x_i - \frac{F(x_i)}{F'(x_i)}$$

à partir d'une valeur initiale  $x_0 \in \mathbb{R}$ . Lorsque la suite  $(x_i)_{i \in \mathbb{N}}$  converge, alors la limite  $x$  est une solution de  $G(x) = x$  donc de  $F(x) = 0$ .

a) Écrire une fonction `newton1d(F, DF, x0, eps=1e-10, N=1000)` qui pour une fonction  $F$ , sa dérivée  $F'$  et une valeur initiale  $x_0$  données calcule les itérations de Newton jusqu'à ce que  $|F(x_i)| < \varepsilon$  et retourne  $x_i$ . Si  $N$  itérations n'ont pas suffi à atteindre ce critère de convergence, alors retourner une erreur.



b) En utilisant la fonction définie précédemment, trouver une solution approchée de l'équation  $e^{-x} = x$ .

**Réponse :** La solution est approximativement donnée par  $x = 0.56714$ .

c) Sans utiliser la fonction `sqrt` ni les puissances fractionnaires, définir une fonction `racine(x,n)` qui calcule  $\sqrt[n]{x}$ .

Parfois, la dérivée de la fonction  $F$  n'étant pas calculable analytiquement, il est alors nécessaire de l'approcher numériquement :

$$F'(x_i) \approx \frac{F(x_i) - F(x_{i-1})}{x_i - x_{i-1}}$$

ce qui mène à la méthode de la sécante :

$$x_{i+1} = x_i - F(x_i) \frac{x_i - x_{i-1}}{F(x_i) - F(x_{i-1})} = \frac{x_{i-1}F(x_i) - x_iF(x_{i-1})}{F(x_i) - F(x_{i-1})}$$

où  $x_0$  et  $x_1$  doivent être choisis.

d) Écrire une méthode `secant1d(F, x0, x1, eps=1e-10, N=1000)` implémentant la méthode de la sécante et la tester sur l'exemple précédent.

## Exercice 7.2 : Méthode de Newton en plusieurs dimensions

La méthode de Newton en une dimension est facilement généralisable à plusieurs dimensions pour résoudre des équations de la forme  $F(\mathbf{x}) = \mathbf{0}$  où  $F : \mathbb{R}^n \rightarrow \mathbb{R}^n$  est une fonction assez régulière. Conceptuellement la méthode est identique : l'équation  $F(\mathbf{x}) = \mathbf{0}$  est équivalente à  $G(\mathbf{x}) = \mathbf{x}$  avec la fonction  $G$  définie par :

$$G(\mathbf{x}) = \mathbf{x} - (F'(\mathbf{x}))^{-1}F(\mathbf{x})$$

où  $F'(\mathbf{x})$  désigne la matrice jacobienne de taille  $n \times n$  de  $F$  en  $x$ . Ainsi les itérations de Newton s'écrivent :

$$\mathbf{x}_{i+1} = \mathbf{x}_i - (F'(\mathbf{x}_i))^{-1}F(\mathbf{x}_i)$$

a) Écrire une fonction `newton(F, DF, x0, eps=1e-12, N=10000)` implémentant la méthode de Newton en plus d'une dimension.

**Indication :** Faire attention que les listes et les tableaux sont mutables. Pour avoir une performance optimale, il ne faut pas inverser la matrice jacobienne mais résoudre un système linéaire avec  $F(\mathbf{x}_i)$  comme second membre.

b) Utiliser la fonction précédente pour résoudre le système suivant :

$$\cos(x) = \sin(y), \quad e^{-x} = \cos(y).$$

**Réponse :** Une solution est approximativement donnée par  $x = 0.58853$  et  $y = 0.98226$ .

## Exercice 7.3 : Attracteur de la méthode de Newton

Le but de cet exercice est de résoudre l'équation  $z^3 = 1$  dans le plan complexe à l'aide de la méthode de Newton et d'analyser vers laquelle des trois racines de l'unité la méthode va converger suivant le choix du point initial  $z_0$ .

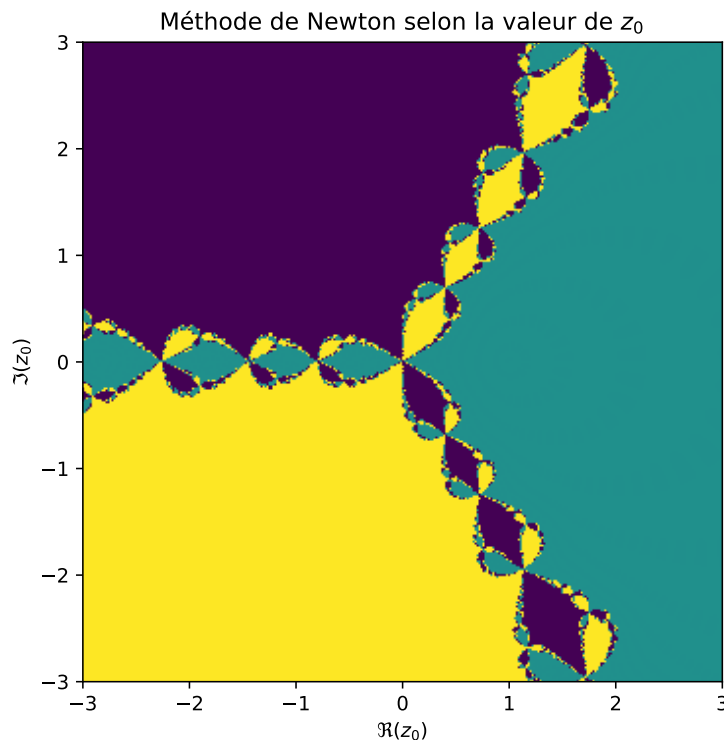
## 7 Zéro de fonctions

a) Si nécessaire adapter la fonction `newton1d` pour que celle-ci s'applique également aux nombres complexes et la tester pour résoudre  $z^3 = 1$  à partir de différentes valeurs de  $z_0$ .

b) Déterminer pour chaque  $z_0 \in \{x_0 + iy_0 : x_0 \in [-3, 3] \text{ et } y_0 \in [-3, 3]\}$  vers quelle racine de l'unité la méthode de Newton va converger. Représenter graphiquement cet ensemble.

**Indication :** La fonction `meshgrid` de Numpy peut être utile pour construire la matrice correspondant à l'ensemble des  $z_0$ .

**Réponse :** Le graphique donnant la convergence de  $z_0$  vers une des trois racines est le suivant :



c) ! La méthode précédente a le désavantage de procéder séquentiellement au calcul pour chaque valeur de  $z_0$ , ce qui rend cette évaluation assez lente. Proposer une nouvelle implémentation permettant de calculer parallèlement toutes les valeurs de  $z_0$  en utilisant les indexages Numpy.

**Indication :** Pour encore plus de rapidité, les itérations de Newton de  $F(z) = z^3 - 1$  peuvent être calculées à la main :

$$z_{n+1} = \frac{1}{3z_n^2} + \frac{2z_n}{3}.$$

### Exercice 7.4 : !! Équation différentielle non linéaire

Le but est de résoudre l'équation différentielle suivante avec conditions de valeurs limites :

$$u''(x) + u^3(x) = \sin(x), \quad u(0) = u(2\pi) = 0,$$

sur l'intervalle  $[0, 2\pi]$ . Cette équation est un modèle simplifié pour une équation de Schrödinger non linéaire.

La méthode employée est celle des différences finies qui consistent à chercher les valeurs de  $u$  aux points  $x_n = \frac{2\pi n}{N}$  pour  $n = 0, 1, \dots, N$ . Les inconnues sont alors les nombres  $u_n = u(x_n)$  et forment un vecteur de dimension  $N + 1$ . La méthode des différences finies consiste à approximer la dérivée seconde par :

$$u''(x) \approx \frac{u(x+h) - 2u(x) + u(x-h)}{h^2},$$

lorsque  $h$  est petit. En prenant  $h = \frac{2\pi}{N}$ , alors :

$$u''(x_n) \approx \frac{u_{n+1} - 2u_n + u_{n-1}}{h^2},$$

et donc l'équation initiale s'approxime par :

$$\frac{u_{n+1} - 2u_n + u_{n-1}}{h^2} + u_n^3 = \sin(x_n), \quad u_0 = u_N = 0,$$

pour  $n = 1, 2, \dots, N-1$ . Cette équation peut être vue comme une équation du type  $F(\mathbf{u}) = \mathbf{0}$  pour  $\mathbf{u} = (u_n)_{n=0}^{N+1}$  et donc être résolue par la méthode de Newton.

a) Montrer l'approximation suivante :

$$u''(x) = \frac{u(x+h) - 2u(x) + u(x-h)}{h^2} + O(h^2) \quad \text{lorsque } h \rightarrow 0.$$

**Indication :** Utiliser le théorème de Taylor.

b) Définir un vecteur  $\mathbf{x}$  représentant les  $N + 1$  points équidistribués dans  $[0, 2\pi]$  et  $h$  la distance entre les points, avec par exemple  $N = 200$ .

c) Définir une fonction  $F(\mathbf{u})$  représentant la fonction  $F : \mathbb{R}^{N+1} \rightarrow \mathbb{R}^{N+1}$  permettant de mettre l'équation approchée sous la forme  $F(\mathbf{u}) = \mathbf{0}$ .

**Indication :** Pour avoir une implémentation rapide, il est impératif pour construire  $F$  d'utiliser non pas une boucle, mais le slicing Numpy.

d) Définir une fonction  $DF(\mathbf{u})$  représentant la jacobienne de la fonction précédente.

**Indication :** La jacobienne est la dérivée de  $F(\mathbf{u}) = F(u_0, u_1, \dots, u_N)$  par rapport à  $\mathbf{u} = (u_0, u_1, \dots, u_N)$ , c'est-à-dire :

$$F'(\mathbf{u}) = \begin{pmatrix} \partial_0 F(\mathbf{u}) & \partial_1 F(\mathbf{u}) & \partial_2 F(\mathbf{u}) & \cdots & \partial_{N-1} F(\mathbf{u}) & \partial_N F(\mathbf{u}) \end{pmatrix},$$

et peut se calculer explicitement à la main.

e) Utiliser la fonction `newton` définie précédemment pour calculer une solution approchée de l'équation. En changeant de valeurs initiales, est-il possible de trouver d'autres solutions ?

**Indication :** Essayer avec la donnée initiale  $u_0(x) = (1 + k) \sin(kx)$  pour  $k = 1, 2, 3, 4$  comme point de départ de la méthode de Newton.

# Théorie des graphes 8

---

Un graphe est un couple  $G = (X, E)$  constitué d'un ensemble  $X$ , non vide et fini, et d'un ensemble  $E$  de paires d'éléments de  $X$ . Les éléments de  $X$  sont les sommets du graphe  $G$ , ceux de  $E$  sont les arêtes du graphe  $G$ . Un graphe est orienté si les arêtes ont une direction, c'est-à-dire si les couples d'éléments de  $E$  sont des listes ordonnées telles que  $(i, j) \in E$  n'est pas équivalent à  $(j, i) \in E$ . Ici seuls des graphes non orientés, c'est-à-dire dont les paires d'éléments de  $X$  sont des ensembles non ordonnés ( $\{i, j\} \in E$ ), sont considérés.

Par exemple, le graphe complet à  $n$  sommets  $K_n$  est le graphe de sommets  $X = 1, 2, \dots, n$  ayant comme arêtes les parties à deux éléments de  $X$ . En particulier,  $K_4 = (X, E)$  où  $X = \{1, 2, 3, 4\}$  et  $E = \{\{1, 2\}, \{1, 3\}, \{1, 4\}, \{2, 3\}, \{2, 4\}, \{3, 4\}\}$ .

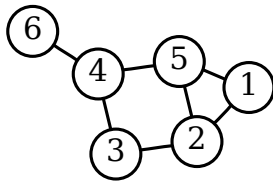
## Concepts abordés

- graphes non orientés
- représentation comme dictionnaire
- utilisation des frozensets
- matrice d'adjacence
- recherche de chemins et de triangles
- fonction récursive

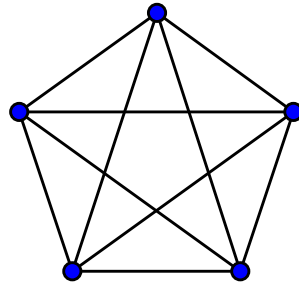
## Exercice 8.1 : Graphes comme dictionnaires

Une façon de représenter un graphe  $G$  est de le faire avec un dictionnaire dont les clefs sont les sommets, et la valeur associée à chaque clef  $x \in X$  est un ensemble contenant les voisins de  $x$ .

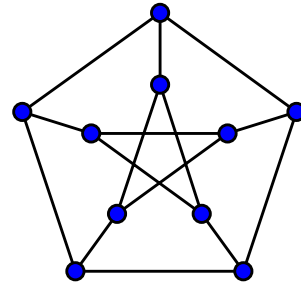
a) Construire sous forme de dictionnaire les graphes suivants :



(a) Graphe à six sommets



(b) Graphe complet



(c) Graphe de Petersen

- b) Écrire une fonction `complet(n)` qui construit comme dictionnaire le graphe complet  $K_n$ .
- c) Un graphe donné sous forme de dictionnaire contient l'information plusieurs fois. Écrire une fonction `corriger(graphe)` permettant de rajouter les éléments manquants de sorte que pour tout sommet  $x$ , si  $y$  appartient à `graphe[x]`, alors  $y$  est aussi une clef et  $x$  appartient à `graphe[y]`. Tester cette fonction.
- d) Écrire une fonction qui retourne l'ensemble (type `set`) de toutes les arêtes d'un graphe représenté par un dictionnaire.

**Indication :** Les ensembles sont mutables et donc pas hashables.

- e) ! Écrire une fonction permettant de déterminer si deux sommets sont connectés par un chemin ou non et qui retourne le chemin si oui.

**Indication :** Utiliser une fonction récursive.

- f) ! Écrire une fonction qui retourne tous les chemins entre deux sommets (sans les cycles).

## Exercice 8.2 : Triangles dans un graphe

Un triangle dans un graphe est un ensemble de trois sommets reliés entre eux par trois arêtes. La recherche et l'analyse des triangles dans un graphe sont importantes pour comprendre sa structure.

- a) Déterminer mathématiquement le nombre de sous-ensembles de cardinal trois que possède un ensemble de sommets  $X$ .

- b) Écrire une fonction retournant l'ensemble des triangles d'un graphe.

À chaque graphe  $G = (X, E)$  correspond une unique matrice  $A$  symétrique de taille  $n \times n$  avec  $n = |X|$  définie par :

$$A_{ij} = \begin{cases} 1, & \text{si } \{i, j\} \in E, \\ 0, & \text{si } \{i, j\} \notin E. \end{cases}$$

Cette matrice est appelée la matrice d'adjacence du graphe  $G$ .

- c) Définir une fonction retournant la matrice d'adjacence d'un graphe.

**Indication :** Faire attention que les sommets ne sont pas forcément indexés par des entiers compris entre 0 et  $n$  dans le dictionnaire.

- d) Définir une fonction ayant pour argument une matrice d'adjacence et retournant le graphe correspondant sous forme d'un dictionnaire.
- e) En utilisant la matrice d'adjacence  $A$  et la matrice  $B = A^2$ , écrire une fonction retournant l'ensemble des triangles d'un graphe.
- f) En utilisant la matrice d'adjacence  $A$ , écrire une fonction calculant le nombre de triangles.

**Indication :** Interpréter les entrées de la matrice  $A^n$ .

### Exercice 8.3 : !! Module NetworkX

De nombreux algorithmes de la théorie des graphes sont implémentés dans le module networkX, voir la documentation [ici](#).

- a) Suivre le tutoriel de networkX disponible [ici](#).
- b) Analyser un des graphes téléchargeables à l'adresse : <https://github.com/gephi/gephi/wiki/Datasets> ou <https://snap.stanford.edu/data/>.

# Probabilités et statistiques 9

Dans un premier temps la statistique de la proportion de nombres commençant par un certain chiffre sera étudiée. Puis dans un second des modèles probabilistes seront introduits et simulés.

## Concepts abordés

- statistiques et probabilités
- loi de Benford
- série harmonique aléatoire
- marche aléatoire
- histogrammes
- importation de données
- optimisation par compilation
- percolation
- transition de phase

## Exercice 9.1 : Série harmonique de signe aléatoire

Le but de cet exercice est de simuler la convergence d'une série harmonique donc le signe est tiré aléatoirement. Plus précisément si  $(X_i)_{i \in \mathbb{N}}$  est une suite de variables aléatoires indépendantes valant  $-1$  ou  $1$  avec probabilité  $\frac{1}{2}$ , alors on définit la somme partielle :

$$W_0 = 0, \quad W_n = \sum_{i=1}^n \frac{X_i}{i},$$

et la question est de déterminer si la suite  $(W_n)_{n \in \mathbb{N}}$  converge et si oui vers quoi.

- Écrire une fonction `sign()` qui simule la variable aléatoire  $X_i$ .
- Écrire une fonction `simulate(n)` qui retourne une liste avec une réalisation de  $(W_1, W_2, \dots, W_n)$ .
- Faire la représentation graphique de la fonction  $n \mapsto W_n$  pour différentes réalisations par exemple pour  $0 \leq n \leq 1000$  et émettre une conjecture quant à la convergence de la suite  $(W_n)_{n \in \mathbb{N}}$ .
- ! Déterminer l'histogramme de  $W_{1000}$  pour  $10^4$  ou  $10^5$  réalisations pour avoir une idée de la loi de la variable aléatoire limite.

## Exercice 9.2 : Loi de Benford

La loi de Benford prédit que statistiquement dans une liste de nombres donnés, la probabilité qu'un de ces nombres commence par le chiffre 1 est plus importante que celle qu'un nombre commence par le chiffre 9. Plus précisément la loi de Benford prédit que la probabilité qu'un nombre commence par le chiffre  $d$  est :

$$p_d = \log_{10} \left( 1 + \frac{1}{d} \right),$$

où  $\log_{10}$  désigne le logarithme en base 10. Il est possible de vérifier que la loi de Benford est la seule qui reste invariante par changement d'unités, *i.e.* en multipliant les nombres de la liste par une constante les probabilités précédentes restent inchangées.

**a)** Écrire une fonction `firstdigit(n)` qui pour un nombre  $n$  donné retourne son premier chiffre et une fonction `occurrences(liste)` qui retourne le nombre d'occurrences des premiers chiffres de `liste`.

**Indication :** Faire en sorte que la fonction `occurrences` fonctionne même si la liste contient des zéros en les ignorant.

**b)** Vérifier si la loi de Benford semble satisfaite pour la suite des nombres  $(2^n)_{n \in \mathbb{N}}$  en comparant l'histogramme empirique avec la loi de Benford.

**c)** Vérifier si la loi de Benford semble satisfaite pour la suite des nombres  $(3n + 1)_{n \in \mathbb{N}}$ .

**d)** En allant sur le site de l'INSEE à l'adresse : <https://insee.fr/fr/statistiques/4171341?sommaire=4171351>, télécharger le fichier au format TXT contenant les données de la population par sexe et âge regroupé (POP1A). Importer ces données pour avoir la population par code postal, sexe et tranche d'âge.

**Indication :** La documentation sur comment lire un fichier est disponible [ici](#). À noter que les fichiers TXT de l'INSEE sont encodés au format ISO-8859-1.

**e)** Déterminer si la liste de toutes les populations par commune, sexe et âge suit la loi de Benford.

**f)** Sommer les données précédentes pour obtenir la liste des populations par commune et déterminer si elle suit la loi de Benford.

**g)** !! Lire la documentation du module Pandas disponible [ici](#) et l'utiliser pour refaire les deux questions précédentes mais avec le fichier de population POP1B avec les âges non regroupés.

**Indication :** Utiliser la fonction `read_csv` de Pandas.

**h)** !! En allant sur le site de l'INSEE ou autre télécharger votre jeu de données préféré, et tester s'il suit la loi de Benford.

**Indication :** Utiliser par exemple les comptes détaillés de l'État disponibles [ici](#).



### Exercice 9.3 : Ruine du joueur

Le but est de simuler l'évolution de la somme d'argent d'un joueur jouant à pile ou face. À chaque lancer le joueur gagne un euro si c'est pile et en perd un si c'est face. La probabilité d'obtenir pile est notée  $p$ , celle d'obtenir face  $q$ . En particulier  $p = q = \frac{1}{2}$  si la pièce est équilibrée.

Mathématiquement, la somme  $S_i$  possédée par le joueur au temps  $i$  est donnée par une marche aléatoire :

$$S_i = \begin{cases} 0, & \text{si } S_{i-1} = 0, \\ S_{i-1} + X_i, & \text{si } S_{i-1} \geq 1, \end{cases}$$

où les  $(X_i)_{i \geq 1}$  sont des variables aléatoires indépendantes de loi  $\mathbb{P}(X_i = 1) = p$  et  $\mathbb{P}(X_i = -1) = q$ .

**a)** Écrire une fonction `simulate(p,k,N)` qui génère une réalisation de longueur  $N$  du processus à partir de  $S_0 = k$ , c'est-à-dire qui retourne  $(S_0, S_1, S_2, \dots, S_N)$ . Représenter graphiquement plusieurs réalisations.

**b)** Simuler un joueur qui, commençant avec une somme  $k$ , joue jusqu'à tout perdre ou avoir la somme  $n \geq k$ .

**c)** Si  $T$  désigne le temps auquel le jeu s'arrête, *i.e.* lorsque  $S_T = 0$  ou  $S_T = n$ , retrouver par simulation les résultats théoriques sur le temps moyen :

$$\mathbb{E}(T) = \begin{cases} k(n-k), & \text{si } p = q, \\ \frac{n}{p-q} \frac{1-\rho^k}{1-\rho^n} - \frac{k}{p-q}, & \text{si } p \neq q, \end{cases}$$

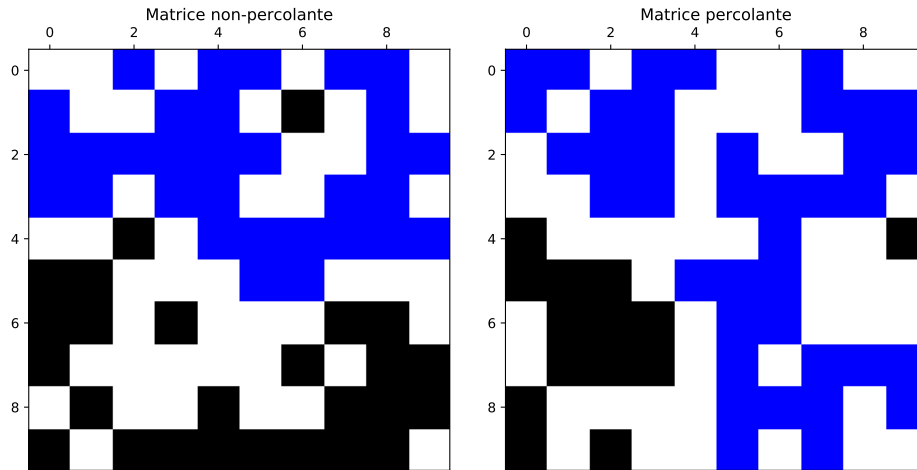
et le lieu de sortie :

$$\mathbb{P}(S_T = 0) = \begin{cases} \frac{n-k}{n}, & \text{si } p = q, \\ \frac{\rho^k - \rho^n}{1 - \rho^n}, & \text{si } p \neq q, \end{cases}$$

où  $\rho = q/p$ . Pour cela on pourra faire un graphique de ces quantités en fonction de  $p$  ou se contenter de considérer le cas  $p = q = \frac{1}{2}$ .

### Exercice 9.4 : !! Percolation

Le but est d'étudier un modèle de percolation dans un milieu poreux. Le milieu est modélisé par une matrice aléatoire de booléens qui détermine les sites qui peuvent être envahis par l'eau et ceux qui sont imperméables. Une matrice percole s'il existe un chemin d'eau allant de la ligne supérieure vers la ligne inférieure. Dans les exemples suivants, les entrées d'une matrice pouvant être envahies par l'eau sont colorées et les entrées effectivement remplies d'eau sont en bleu. La première matrice ne percole pas alors que la seconde oui :



a) Écrire une fonction `generate(n,p)` qui génère une matrice de booléens de taille  $n \times n$  telle que chaque entrée ait probabilité  $p$  d'être juste et  $1 - p$  d'être fausse.

**Indication :** La fonction `numpy.random.binomial` peut être utile.

b) Définir une fonction `fill(isopen)` qui pour une matrice de booléens donnée renvoie une autre matrice de booléens avec les entrées envahies par l'eau.

**Indication :** Définir une matrice de booléens `isfull` pour stocker si une entrée est remplie par l'eau ou pas, puis définir une fonction récursive `flow(isopen, isfull, i, j)` permettant d'envahir toutes les entrées possibles à partir de  $(i, j)$ .

c) À l'aide de Matplotlib représenter le remplissage de différentes matrices générées aléatoirement.

d) Définir une fonction `percolate(isopen)` permettant de déterminer si une matrice de booléens percole ou non.

e) !! Calculer le temps nécessaire pour déterminer si une matrice de taille  $50 \times 50$  avec  $p = 0.9$  percole ou non. Lire la documentation du module `numba` pour réduire le temps de calcul en compilant une des fonctions : <https://numba.pydata.org/>.

**Indication :** La fonction qui est la plus utilisée est la fonction récursive, donc c'est celle qu'il faut optimiser en la compilant.

f) En faisant des statistiques, déterminer la probabilité qu'une matrice aléatoire booléenne de taille  $n \times n$  avec probabilité  $p$  percole. Étudier cette probabilité en fonction de  $p$  et de  $n$ .

**Indication :** Faire le graphique de cette probabilité de percolation en fonction de  $p$  pour différentes valeurs de  $n$ .

**Réponse :** Dans la limite des  $n$  très grands, une matrice percole presque sûrement si  $p > 0.592746$  et presque jamais sinon.

g) !!! Les statistiques effectuées au point précédent sont un exemple typique de calculs pouvant être facilement exécutés en parallèle, car chaque cas est indépendant des autres. Paralléliser l'algorithme précédent de manière à utiliser tous les cœurs de votre processeur, par exemple à l'aide du module `mpi4py`.

**Indication :** L'utilisation de Jupyter Lab pour faire du calcul parallèle est assez complexe à mettre en œuvre, il vaut mieux utiliser la ligne de commande pour exécuter un script en parallèle, par exemple pour quatre cœurs : `mpirun -n 4 script.py`. À noter que [Open MPI](#) ou [MPICH](#) doit être installé sur votre ordinateur.

# Équations différentielles 10

Le but est d'introduire les méthodes de base permettant de résoudre des équations différentielles ordinaires du premier ordre du type :

$$\dot{\mathbf{x}}(t) = f(t, \mathbf{x}(t)), \quad \mathbf{x}(0) = \mathbf{x}_0,$$

où  $f : \mathbb{R}^+ \times \mathbb{R}^n \rightarrow \mathbb{R}^n$  est une fonction assez régulière et  $\mathbf{x}_0 \in \mathbb{R}^n$  une donnée initiale. À noter que les équations différentielles ordinaires d'ordre supérieur peuvent être mises sous la forme précédente du premier ordre.

## Concepts abordés

- méthodes d'Euler
- méthodes de Runge-Kutta
- équation aux dérivées partielles non linéaire
- différences finies
- méthodes adaptatives

## Exercice 10.1 : Méthodes d'Euler

L'idée la plus simple pour résoudre de manière approchée une équation différentielle ordinaire est de discrétiser le temps avec un pas  $h$  et d'approximer la dérivée temporelle sur chaque intervalle de longueur  $h$ . Il y a deux façons simples d'approximer la dérivée en temps. La première est l'approximation par différence finie avant :

$$\dot{\mathbf{x}}(t) \approx \frac{\mathbf{x}(t+h) - \mathbf{x}(t)}{h},$$

la seconde par différence finie arrière :

$$\dot{\mathbf{x}}(t) \approx \frac{\mathbf{x}(t) - \mathbf{x}(t-h)}{h}.$$

Les inconnues étant les évaluations de la solution  $\mathbf{x}$  aux temps  $t_i = ih$  pour  $i \geq 0$ , c'est-à-dire  $\mathbf{x}_i = \mathbf{x}(t_i)$ . L'équation différentielle peut ainsi être approchée à l'aide des différences finies avant par :

$$\frac{\mathbf{x}_{i+1} - \mathbf{x}_i}{t_{i+1} - t_i} = f(t_i, \mathbf{x}_i),$$

## 10 Équations différentielles

ce qui donne la formule d'Euler explicite :

$$\mathbf{x}_{i+1} = \mathbf{x}_i + (t_{i+1} - t_i)f(t_i, \mathbf{x}_i).$$

Avec l'approximation par différences finies arrière, on obtient la méthode d'Euler implicite :

$$\mathbf{x}_i = \mathbf{x}_{i-1} + (t_i - t_{i-1})f(t_i, \mathbf{x}_i).$$

La formule d'Euler explicite permet de calculer directement tous les  $\mathbf{x}_i$  par récurrence en connaissant  $\mathbf{x}_0$ . En revanche la formule d'Euler implicite nécessite à chaque pas de temps la résolution d'une équation non linéaire pour  $\mathbf{x}_i$  par exemple avec la méthode de Newton.

**a)** Écrire une fonction `euler_explicit(f,x0,t)` qui pour une donnée initiale `x0` retourne les valeurs  $\mathbf{x}_0, \mathbf{x}_1, \dots, \mathbf{x}_m$  calculées avec la méthode d'Euler explicite aux temps  $(t_i)_{i=0}^m$  représentés par le vecteur `t`.

**b)** Utiliser la méthode d'Euler explicite pour résoudre l'équation différentielle :

$$\dot{x}(t) + x(t) = \sin(t), \quad x(0) = 1,$$

pour  $t \in [0, 10]$ . Comparer les résultats avec la solution exacte :

$$x(t) = \frac{1}{2}(\sin(t) - \cos(t) + 3e^{-t}),$$

pour différentes discrétisations du temps.

**c)** Résoudre le problème précédent avec la méthode d'Euler implicite.

**Indication :** Vu que l'équation précédente est linéaire, on peut en fait rendre explicite la méthode d'Euler implicite en résolvant l'équation implicite à la main.

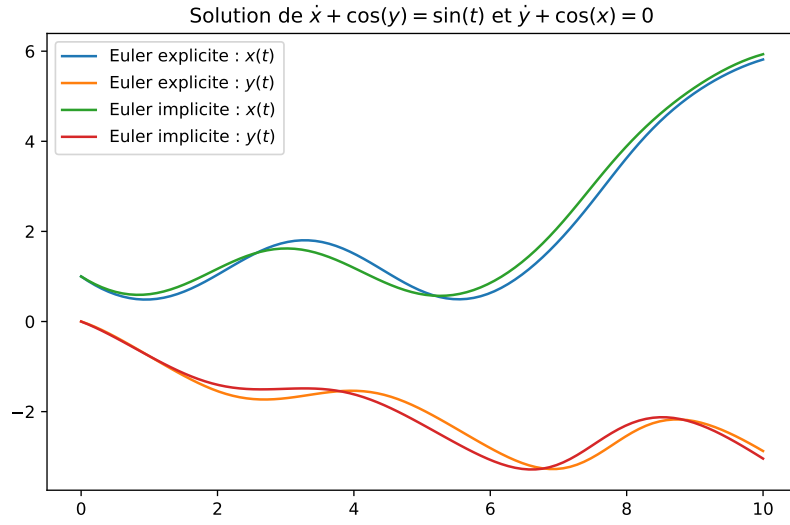
**d) !!** Définir une fonction `euler_implicit(f, Dxf, x0, t)` implémentant la méthode d'Euler implicite pour des équations non linéaires. À noter que pour résoudre le problème non linéaire avec la méthode de Newton, la dérivée de  $f$  selon  $\mathbf{x}$  est nécessaire.

**Indication :** Il est également possible d'utiliser l'algorithme de recherche de zéro de fonction `scipy.optimize.fsolve` qui ne nécessite pas de connaître la dérivée de  $f$ .

**e) !** Utiliser les méthodes précédentes pour trouver une solution approchée du système :

$$\begin{aligned} \dot{x}(t) + \cos(y(t)) &= \sin(t), & x(0) &= 1, \\ \dot{y}(t) + \cos(x(t)) &= 0, & y(0) &= 0. \end{aligned}$$

**Réponse :** Voici le graphique de la solution de l'équation différentielle avec un pas de temps  $h = 0.1$  :



## Exercice 10.2 : Méthodes de Runge-Kutta

Le but de cet exercice est d'introduire une classe de méthodes plus précises que les méthodes d'Euler pour résoudre des équations différentielles ordinaires. Au lieu de faire une approximation au premier ordre en  $h$  l'idée est de faire une approximation d'un ordre supérieur.

L'idée de base est de construire une suite  $\mathbf{x}_i$  donnant une approximation de la solution de  $\dot{\mathbf{x}}(t) = f(t, \mathbf{x})$  au temps  $t_i$  pour  $i \in \mathbb{N}$ . Cette suite est définie par :

$$\mathbf{x}_{i+1} = \mathbf{x}_i + M(t_i, \mathbf{x}_i, t_{i+1} - t_i),$$

pour une certaine fonction  $M$  appelée méthode. Par exemple pour la méthode d'Euler explicite, la fonction  $M$  est donnée par :

$$M(t, \mathbf{x}, h) = hf(t, \mathbf{x})$$

Une méthode de Runge-Kutta d'ordre deux est donnée par :

$$M(t, \mathbf{x}, h) = hf\left(t + \frac{h}{2}, \mathbf{x} + \frac{h}{2}f(t, \mathbf{x})\right).$$

Une méthode de Runge-Kutta d'ordre quatre est donnée par :

$$M(t, \mathbf{x}, h) = \frac{h}{6}(\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4),$$

où

$$\begin{aligned} \mathbf{k}_1 &= f(t, \mathbf{x}), \\ \mathbf{k}_2 &= f\left(t + \frac{h}{2}, \mathbf{x} + \frac{h}{2}\mathbf{k}_1\right), \\ \mathbf{k}_3 &= f\left(t + \frac{h}{2}, \mathbf{x} + \frac{h}{2}\mathbf{k}_2\right), \\ \mathbf{k}_4 &= f(t + h, \mathbf{x} + h\mathbf{k}_3). \end{aligned}$$

## 10 Équations différentielles

À noter que plus généralement, une méthode de Runge-Kutta d'ordre  $s$  est donnée par :

$$M(t, \mathbf{x}, h) = h \sum_{i=1}^s b_i \mathbf{k}_i,$$

où

$$\begin{aligned} \mathbf{k}_1 &= f(t, \mathbf{x}), \\ \mathbf{k}_2 &= f(t + c_2 h, \mathbf{x} + h a_{21} \mathbf{k}_1), \\ \mathbf{k}_3 &= f(t + c_3 h, \mathbf{x} + h(a_{31} \mathbf{k}_1 + a_{32} \mathbf{k}_2)), \\ &\vdots \\ \mathbf{k}_s &= f(t + c_s h, \mathbf{x} + h(a_{s1} \mathbf{k}_1 + a_{s2} \mathbf{k}_2 + \cdots + a_{s,s-1} \mathbf{k}_{s-1})). \end{aligned}$$

Les coefficients  $a_{ij}$  (pour  $1 \leq j < i \leq s$ ),  $c_i$  (pour  $2 \leq i \leq s$ ), et  $b_i$  (pour  $1 \leq i \leq s$ ), sont souvent représentés dans un tableau de Butcher :

$$\begin{array}{c|cccc} 0 & & & & \\ c_2 & a_{21} & & & \\ c_3 & a_{31} & a_{32} & & \\ \vdots & \vdots & & \ddots & \\ c_s & a_{s1} & a_{s2} & \cdots & a_{s,s-1} \\ \hline & b_1 & b_2 & \cdots & b_{s-1} & b_s \end{array}$$

Par exemple, le tableau de Butcher de la méthode précédente d'ordre deux est :

$$\begin{array}{c|c} 0 & \\ \frac{1}{2} & \frac{1}{2} \\ \hline & 0 & 1 \end{array}$$

et celui de la méthode d'ordre quatre :

$$\begin{array}{c|cccc} 0 & & & & \\ \frac{1}{2} & \frac{1}{2} & & & \\ \frac{1}{2} & 0 & \frac{1}{2} & & \\ 1 & 0 & 0 & 1 & \\ \hline & \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6} \end{array}$$

- Définir une fonction `integrate(f, x0, t, M)` qui pour une liste de temps  $(t_i)_{i=0}^N$  donnée retourne les valeurs correspondantes  $\mathbf{x}_0, \mathbf{x}_1, \dots, \mathbf{x}_N$  avec la méthode  $M$ .
- Implémenter les fonctions `M(f, t, x, h)` pour la méthode d'Euler explicite et la méthode de Runge-Kutta d'ordre deux. Comparer les deux méthodes.
- Implémenter la fonction `M(f, t, x, h)` pour la méthode de Runge-Kutta d'ordre quatre. Comparer avec la méthode d'ordre deux.

### Exercice 10.3 : Mouvement d'une planète

Le but est de simuler le mouvement bidimensionnel d'une planète gravitant autour d'une étoile fixe. L'étoile est supposée fixée à l'origine et la position de la planète dans le plan est décrite par le vecteur  $\mathbf{x} \in \mathbb{R}^2$ . L'étoile est supposée interagir avec la planète avec le potentiel :

$$V(\mathbf{x}) = \frac{1}{\alpha} |\mathbf{x}|^\alpha,$$

## 10 Équations différentielles

pour un certain  $\alpha \in \mathbb{R}$  où  $|\mathbf{x}|$  désigne la norme euclidienne du vecteur  $\mathbf{x}$ . À noter que le potentiel gravitationnel correspond à  $\alpha = -1$ . L'équation de la planète dans ce champ de force est donnée par :

$$\ddot{\mathbf{x}} = -\nabla V(\mathbf{x}) = -\mathbf{x}|\mathbf{x}|^{\alpha-2}.$$

- a) Réécrire l'équation différentielle d'ordre deux comme une équation différentielle d'ordre un pour  $\mathbf{x}$  et  $\mathbf{p} = \dot{\mathbf{x}}$ .
- b) Implémenter la fonction `f(t, xp)` correspondant à l'équation trouvée au point précédent.
- c) À l'aide de la méthode de Runge-Kutta d'ordre quatre, résoudre l'équation différentielle pour différentes données initiales et différentes valeurs de  $\alpha$  et tracer les trajectoires  $\mathbf{x}(t)$  dans le plan. Interpréter les résultats et expliquer en particulier pourquoi les cas  $\alpha = -1$  et  $\alpha = 2$  sont différents des autres.

### Exercice 10.4 : !! Équation des ondes cubique

Le but est de résoudre numériquement l'équation des ondes non linéaire sur  $\mathbb{R}$  :

$$-\frac{\partial^2 u}{\partial t^2} + \frac{\partial^2 u}{\partial x^2} = u^3, \quad u(0, \cdot) = u_0, \quad \frac{\partial u}{\partial t}(0, \cdot) = v_0,$$

pour  $u : \mathbb{R}^+ \times \mathbb{R} \rightarrow \mathbb{R}$  avec  $u_0, v_0 : \mathbb{R} \rightarrow \mathbb{R}$  deux fonctions données.

#### Remarque

Les propriétés de cette équation d'apparence simple sont très mal comprises mathématiquement, voir l'article de recherche suivant pour plus de détails : [doi:10.2140/apde.2012.5.411](https://doi.org/10.2140/apde.2012.5.411).

- a) Réécrire l'équation précédente sous la forme de deux équations du premier ordre en temps pour  $u$  et  $v = \frac{\partial u}{\partial t}$ .
- b) En approximant la seconde dérivée en espace par différences finies comme à l'Exercice 7.4, montrer que l'équation peut s'approximer de la manière suivante :

$$\begin{aligned} \frac{\partial u_n}{\partial t} &= v_n, & u_n(0) &= u_0(x_n), \\ \frac{\partial v_n}{\partial t} &= \frac{u_{n-1} - 2u_n + u_{n+1}}{h^2} - u_n^3, & v_n(0) &= v_0(x_n), \end{aligned}$$

où  $(x_n)_{n=0}^N$  représente  $N+1$  points équidistants de  $h$  dans l'intervalle  $[-L, L]$  et  $u_n(t) = u(t, x_n)$  et  $v_n(t) = v(t, x_n)$ . Pour les conditions au bord du domaine, *i.e.* lorsque  $n = 0$  ou  $n = N$ , on prendra :

$$\frac{\partial v_0}{\partial t} = 0, \quad \frac{\partial v_N}{\partial t} = 0.$$

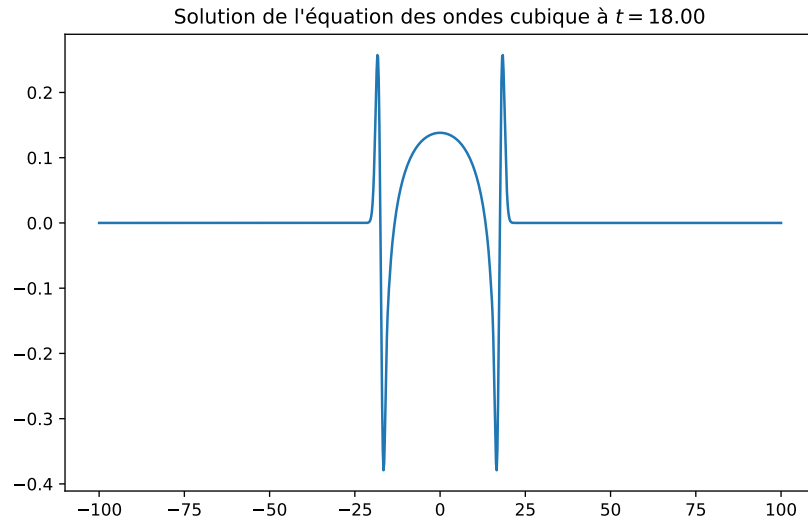
- c) Déterminer la fonction  $f : \mathbb{R} \times \mathbb{R}^{2N+2} \rightarrow \mathbb{R}^{2N+2}$  permettant de mettre l'approximation précédente sous la forme  $\dot{\mathbf{u}} = f(t, \mathbf{u})$  pour  $\mathbf{u} = (u, v)$  et implémenter cette fonction.
- d) Résoudre l'équation différentielle donnée par  $\dot{\mathbf{u}} = f(t, \mathbf{u})$  par exemple avec la méthode de Runge-Kutta d'ordre quatre. Un bon choix de paramètre est  $L = 100$ ,  $N = 1000$ , pour la donnée initiale  $u_0(x) = e^{-x^2}$  et  $v_0(x) = 0$ . La vitesse de propagation de l'onde est un et donc, après un temps plus grand que  $L$ , l'onde sort de la boîte  $[-L, L]$  et ne correspond plus à une bonne approximation de l'équation initiale.



e) À l'aide du module `animation` de Matplotlib réaliser une vidéo montrant l'évolution de l'onde en fonction du temps.

**Indication :** Utiliser par exemple la fonction `FFMpegWriter`.

**Réponse :** Voici la forme de la solution :



### Exercice 10.5 : !!! Méthodes de Bogacki-Shampine

En combinant deux méthodes de Runge-Kutta d'ordres différents (par exemple (2,3) ou (4,5)), on obtiendra une estimation empirique de l'erreur sur un pas de temps. En utilisant cette estimation d'erreur, il est possible d'adapter le pas de temps, soit en l'augmentant soit en le diminuant et ainsi de s'adapter à l'équation.

Pour une méthode de Runge-Kutta d'ordre  $s$ , une méthode interne d'ordre moins élevé (généralement  $s - 1$ ) est donnée par :

$$M^*(t, \mathbf{x}, h) = h \sum_{i=1}^s b_i^* \mathbf{k}_i,$$

où les  $\mathbf{k}_i$  sont identiques à ceux de la méthode d'ordre  $s$ . Une estimation de l'erreur est alors donnée par :

$$E(t, \mathbf{x}, h) = M(t, \mathbf{x}, h) - M^*(t, \mathbf{x}, h) = h \sum_{i=1}^s (b_i - b_i^*) \mathbf{k}_i.$$

Une telle méthode est donnée par un tableau de Butcher étendu :

0					
$c_2$	$a_{21}$				
$c_3$	$a_{31}$	$a_{32}$			
$\vdots$	$\vdots$		$\ddots$		
$c_s$	$a_{s1}$	$a_{s2}$	$\cdots$	$a_{s,s-1}$	
<hr/>					
	$b_1$	$b_2$	$\cdots$	$b_{s-1}$	$b_s$
	$b_1^*$	$b_2^*$	$\cdots$	$b_{s-1}^*$	$b_s^*$

**a)** Implémenter la méthode de Bogacki-Shampine d'ordre (4,5). L'article original est disponible à l'adresse [doi:10.1016/0898-1221\(96\)00141-1](https://doi.org/10.1016/0898-1221(96)00141-1).

**Indication :** Les coefficients des tables de Butcher sont implémentés dans un module `nodepy` donc la documentation est disponible [ici](#). Le nom de la méthode dans ce package est «BS5».

# Cryptographie 11

---

Depuis le code de César, les méthodes cryptographiques permettant de transmettre des messages secrets ont évoluées en suivant les progrès permettant de les casser. La méthode Vigenère qui est une amélioration du code de César sera étudiée, et nous verrons comment il est possible de casser cette méthode de chiffrement. Ensuite, la méthode de chiffrement RSA qui est une des méthodes de cryptographie asymétrique les plus utilisées actuellement sera introduite.

## Concepts abordés

- code Vigenère
- plus grand diviseur commun
- importation de texte
- nombre premier et pseudo-premier
- petit théorème de Fermat
- algorithme d'Euclide
- algorithme de Miller-Rabin
- optimisation par décorateur
- chiffrement asymétrique RSA

## Exercice 11.1 : Code de Vigenère

Le code de Vigenère consiste à choisir une clef formée par un mot secret (en majuscules) et à le transformer en un vecteur dont les éléments sont les positions de ces lettres dans l'alphabet. Par exemple, «ASECRET» correspond à (0, 18, 4, 2, 17, 4, 19). Pour coder un texte (en majuscules, sans espace ni ponctuation) avec la clef «ASECRET» il suffit de décaler la première lettre par 0, la deuxième par 18, la troisième par 4, et ainsi de suite et de recommencer en boucle. Les détails en particulier historiques sont disponibles sur [Wikipédia](#).

a) Écrire une fonction `to_int(s)` qui transforme un caractère en sa place dans l'alphabet et écrire la fonction inverse `to_chr(i)`.

**Indication :** Voir la documentation des fonctions `ord` et `chr`.

b) Écrire une fonction `crypt(text, key)` qui chiffre `text` avec le mot secret `key`.

c) Écrire une fonction permettant de déchiffrer un texte en connaissant la clef.

## Exercice 11.2 : ! Casser le code de Vigenère

Charles Babbage a été le premier à casser le code de Vigenère. L'idée est que trois lettres consécutives apparaissant plusieurs fois dans le texte chiffré ont toutes les chances d'être la conséquence du chiffrement des mêmes lettres du message avec les mêmes lettres de la clef. Cela est encore plus probable avec un groupe de quatre lettres. Ainsi l'espacement entre deux mêmes groupes de lettres chiffrées est un multiple de la longueur de la clef. Par exemple, si la répétition d'un groupe est espacée de 28 lettres, puis celle d'un autre de 91, le plus grand diviseur commun de 28 et 91 est 7. Ainsi il est probable que la clef possède 7 lettres. Ensuite connaissant la taille de la clef, il suffit de se baser sur le fait que la lettre «E» est la plus courante en français. Pour que cette stratégie ait une chance de réussir, il faut que la taille du texte soit assez importante.

- a) Écrire une fonction permettant de calculer le plus grand diviseur commun (PGCD) entre deux nombres. Écrire une autre fonction permettant de calculer le PGCD entre une liste de nombres.
- b) Visiter le site du projet [Gutenberg](#), choisir son texte français préféré et le télécharger au format «Plain Text». Écrire une fonction qui convertisse le texte en majuscules et le débarrasse de toute la ponctuation et autres caractères spéciaux.

**Indication :** Pour convertir un texte en majuscules (en convertissant les accents) et supprimer tous les caractères de ponctuation et autres, il est possible d'utiliser la fonction suivante :

```
import unicodedata, re
def convert_upper(text):
    # convertir en majuscules
    text = text.upper()
    # convertir les accents
    text = unicodedata.normalize('NFKD', text)
    # supprimer les caractères spéciaux
    regex = re.compile('[^a-zA-Z]')
    text = regex.sub('', text)
    return text
```

- c) Garder environ un millier de caractères au milieu du texte choisi et le chiffrer avec une clef. Écrire ensuite une fonction permettant de déterminer la longueur de la clef en regardant dans le message chiffré les chaînes identiques de trois caractères ou plus.

**Indication :** Former tout d'abord un dictionnaire avec comme clef toutes les occurrences de trois lettres et comme valeur les positions des occurrences. Ensuite déterminer la liste des distances entre les occurrences de trois lettres, puis calculer le PGCD de ces distances. Si ce PGCD est égal à 1 ou est trop petit, alors réessayer mais avec des chaînes de plus que trois caractères.

- d) Écrire une fonction permettant de décrypter un message chiffré en retournant la clef. Essayer de décrypter le texte de son auteur favori avec cette fonction.

**Indication :** Pour trouver la première lettre de la clef, il faut calculer le nombre d'occurrences des 26 lettres de l'alphabet dans le message chiffré qui ont été chiffrées avec le premier caractère de la clef. En principe, la lettre dont l'occurrence est maximale correspond à la lettre «E». Il suffit ensuite de faire la même chose pour trouver les autres lettres de la clef.

### Exercice 11.3 : Générer des nombres premiers

La plupart des algorithmes de cryptage actuels sont fondés sur l'utilisation de grands nombres premiers. Le but est d'écrire une fonction permettant de générer relativement rapidement des nombres dits pseudo-premiers, c'est-à-dire qui sont premiers avec une probabilité extrêmement grande. La première étape est de générer un grand nombre aléatoire, c'est-à-dire ayant un certain nombre de bits. Ensuite un test de primalité permet de décider si ce nombre est premier ou pas. Si  $\pi(n)$  désigne le nombre de nombres premiers inférieurs ou égaux à  $n$ , alors asymptotiquement  $\pi(n) \approx \frac{n}{\ln n}$ . Pour un nombre inférieur à  $n$  tiré au hasard la probabilité qu'il soit premier est d'environ  $1/\ln(n)$ . Par exemple pour générer un nombre premier de 1 024 bits (le minimum garantissant une sécurité raisonnable actuellement), c'est-à-dire de l'ordre de  $2^{1024}$ , il faut essayer  $\ln(2^{1024}) \approx 710$  nombres aléatoires avant d'en trouver un qui soit premier. Vu que tous les nombres pairs ne sont clairement pas premiers, il suffit d'en tester en moyenne 355.

a) Écrire un programme permettant de générer un nombre aléatoire impair de  $k$  bits, c'est-à-dire compris entre  $2^{k-1}$  et  $2^k$ .

**Indication :** La façon la plus rapide d'implémenter cela est d'utiliser les opérations sur les bits expliquées [ici](#).

La façon la plus simple de déterminer si un nombre  $n$  est premier est d'essayer de le diviser par tous les nombres entiers  $1 < d < n$ . Il y a deux raisons qui permettent de ne pas tester tous les  $d$  entre 2 et  $n - 1$ . La première est qu'il est inutile d'essayer les  $d$  pairs plus grands que 2. La seconde est qu'il est inutile de tester des nombres plus grands que  $\sqrt{n}$ .

b) Écrire un algorithme `isprime(n)` permettant de déterminer si un nombre est premier ou pas.

c) Écrire une fonction `generate(k,primalité)` permettant de générer un nombre premier aléatoire de  $k$  bits avec le test de primalité `primalité`. Tester avec le test de primalité `isprime`. Est-il raisonnable de pouvoir espérer générer un nombre premier de 1 024 bits avec cet algorithme ?

### Exercice 11.4 : Générer des nombres pseudo-premiers

L'algorithme précédent permettant de générer des nombres premiers étant inutilisable pour générer des grands nombres premiers, une autre approche, probabiliste, est à préconiser. Un test de primalité probabiliste décide qu'un nombre est premier s'il est premier avec une probabilité très grande. Un tel nombre est dit pseudo-premier. Ainsi un test probabiliste peut se tromper et supposer qu'un nombre est premier alors qu'en fait il ne l'est pas.

Le test de primalité le plus simple est fondé sur le petit théorème de Fermat : si  $n$  est premier, alors  $a^{n-1} = 1 \pmod{n}$  pour tout  $1 \leq a \leq n - 1$ . Ainsi si on trouve un  $a$  tel que  $a^{n-1} \neq 1 \pmod{n}$ , alors  $n$  n'est pas premier. Le test de primalité de Fermat teste  $N$  valeurs de  $a$  choisies aléatoirement et si  $a^{n-1} = 1 \pmod{n}$  pour ces  $N$  valeurs, alors il déclare que  $n$  est probablement premier. Les nombres de Carmichael ne sont pas premiers, mais satisfont  $a^{n-1} = 1 \pmod{n}$  pour tout  $a$  premier avec  $n$ . Les premiers nombres de Carmichael sont 561, 1 105 et 1 729. Si  $n$  n'est pas un nombre de Carmichael, alors la probabilité que le test de Fermat se trompe est de  $2^{-N}$ . En choisissant par exemple  $N = 128$ , on obtient une probabilité de se tromper inférieure à  $3 \times 10^{-39}$ .

a) Écrire une fonction implémentant le test de primalité de Fermat. Utiliser ce test pour générer des nombres premiers aléatoires.

**Indication :** Voir la documentation de la fonction `pow` pour une implémentation rapide. Si OpenSSL est installé sur votre ordinateur, il est facile de vérifier si un nombre est premier avec la commande `openssl prime 11` par exemple pour 11.

b) ! Améliorer la rapidité de l'algorithme précédent en testant d'abord si  $n$  est divisible par les nombres premiers inférieurs à 1 000 avant d'appliquer le test de Fermat.

Le test de primalité de Fermat permet de générer de grands nombres pseudo-premiers avec une bonne probabilité de tomber juste. Le problème principal vient du fait de l'existence des nombres de Carmichael qui sont exclus de cette probabilité. Le test de primalité de Miller-Rabin permet d'éviter ce problème.

c) !! Comprendre et implémenter la méthode Miller-Rabin expliquée en détails sur [Wikipédia](#).

## Exercice 11.5 : Chiffrement RSA

L'algorithme RSA, des initiales de Ronald Rivest, Adi Shamir et Leonard Adleman qui l'ont inventé en 1983, est un des algorithmes de cryptographie asymétrique les plus utilisés encore actuellement. Un chiffrement asymétrique permet de transmettre un message crypté à une personne A sans avoir eu auparavant besoin de transmettre une clef secrète à la personne B qui chiffre le message. La création par A d'une clef publique suffit à B pour chiffrer le message et pour que A puisse le déchiffrer avec sa clef privée. Il y a trois grandes étapes dans l'algorithme RSA :

**Création des clefs.** La personne A voulant recevoir un message secret choisit deux très grands nombres premiers  $p$  et  $q$  qu'elle garde secrets. Elle calcule ensuite  $n = pq$  et l'indicatrice d'Euler  $\varphi(n) = (p-1)(q-1)$  qui compte le nombre d'entiers compris entre 1 et  $n$  qui sont premiers avec  $n$ . Puis elle choisit un exposant de chiffrement  $e$  qui est premier avec  $\varphi(n)$ . La clef publique de la personne A est donnée par le couple  $(n, e)$ . Finalement, la personne A calcule l'exposant de déchiffrement  $d$  qui est l'inverse de  $e$  modulo  $\varphi(n)$ , *i.e.* tel que  $ed = 1 \pmod{\varphi(n)}$ . La clef privée de A est  $(p, q, d)$ .

**Chiffrement du message.** Pour chiffrer son message, la personne B le transforme tout d'abord en un nombre entier  $M < n$ . Le message chiffré est alors donné par :

$$C = M^e \pmod{n}.$$

**Déchiffrement du message.** Le message chiffré  $C$  est alors transmis à A. Pour le déchiffrer, A calcule :

$$M = C^d \pmod{n}$$

qui est à nouveau le message original.

### Remarque

Les nombres premiers  $p$  et  $q$  doivent être vraiment aléatoires, sinon il est possible de deviner leurs valeurs. Les nombres aléatoires générés par le module `random` le sont avec l'algorithme de Mersenne Twister. Cet algorithme n'est pas considéré comme cryptographiquement sûr dans le sens où une observation d'environ un millier de nombres générés aléatoirement par cet algorithme suffit à prédire toutes les itérations futures. Pour générer des nombres aléatoires cryptographiquement sûrs il faudrait utiliser le module `secrets`.

## 11 Cryptographie

a) Montrer mathématiquement que le message déchiffré correspond bien au message original.

**Indication :** Si  $a = b \pmod{\varphi(n)}$  et  $M$  est premier avec  $n$ , alors  $M^a = M^b \pmod{n}$ .

b) À partir de  $e$  et  $\varphi(n)$  donnés, écrire une fonction `bezout(e, phi)` permettant de déterminer  $d$  tel que  $ed = 1 \pmod{\varphi(n)}$ .

**Indication :** Utiliser l'algorithme d'Euclide généralisé qui permet de déterminer le PGCD  $g$  entre deux nombres  $a$  et  $b$  ainsi que  $x$  et  $y$  satisfaisant  $ax + by = g$ .

c) Écrire un algorithme `generate_keys(length)` qui génère des nombres premiers  $p$  et  $q$  tels que  $n$  ait `length` bits, puis détermine  $\varphi(n)$ ,  $e$  et  $d$  et enfin retourne la clef publique  $(n, e)$  et la clef privée  $(p, q, d)$ .

d) En choisissant d'encoder chaque caractère sur 8 bits, une chaîne de caractères de longueur  $\ell$  s'écrit comme une liste  $(a_0, a_1, \dots, a_{\ell})$  avec chaque  $0 \leq a_i \leq 255$ . Cette liste peut être convertie en un entier  $k$  de la façon suivante :

$$k = \sum_{i=0}^{\ell} a_i 256^i$$

Écrire une fonction `toint` et une fonction `tostr` permettant respectivement de convertir une chaîne de caractères en cet entier et inversement.

e) Écrire une fonction pour chiffrer un texte avec une clef publique et une autre permettant de le déchiffrer avec la clef privée. Pour cela il faut s'assurer que le texte soit convertible en un entier inférieur à  $n$ , sinon il faut le découper en blocs et les chiffrer séparément.

### Exercice 11.6 : !!! Casser le chiffrement RSA

Voici une clef publique :

```
(73722206893746878039310298412768333517547506486427363913406174815240823284857,
33921003048397584579835360477549223828723590186917811609938274008840181968499)
```

et un message chiffré avec cette clef publique :

```
[22973877239788873882837788687834740958145091979501565167824992597825600406974,
48503379361942356829127901273483580639426474600801539865525830360302350602689,
2224798454942012298637628855810886175704245737423608868190613249161861526055,
4500720701216145302036625979462397411127541711596515042635302843142748047486,
35445000935671280429079877747553363121645781096430386417428307485228904386825,
48627712259501563035806415688912560481020577805784144969245386699539463833735,
71868389092768589092947834441169271512227882469129366706758279960751502739157,
26019603019482382085505727901092092122241438959147654068117475447783602572984,
65039729472521706954510624828984329309712256390395416184704490683377874857302,
11805320141319662342135552217286927868466795280631816945032387836622798495117]
```

a) Décrypter le message précédent !

**Indication :** Il est probablement nécessaire de choisir un algorithme adapté, par exemple utilisant des cribles quadratiques (QS, MPQS, SIQS).