Informatique

Bibliothèque `numpy`

PSI : Lycée Rabelais

Pré-requis

Cours d'informatique de première année

Objectifs

Comprendre les fonctions usuelles utilisées lors des concours (`numpy`, `matplotlib`,...)

Notion de bibliothèque

1 Découverte de numpy

Dans les sujets de concours, un certain nombre de fonctions "usuelles" sont données en fin de sujet. Il ne faut pas les découvrir le jour J. On se propose ici de les découvrir puis de les utiliser.

1.1 Qu'est ce que le module `numpy` ?

`numpy` est une bibliothèque de calcul numérique. Elle contient un certain nombre de fonctions permettant de résoudre de manière numérique des problèmes mis en équation dans d'autres disciplines scientifiques (physique, biologie, SI...) : résolution d'équations différentielles, de systèmes d'équations...

L'objectif de ce cours est d'apprendre à utiliser les fonctions les plus classiques de cette bibliothèque afin de résoudre des problèmes simples de simulation numérique. Les résultats obtenus pourront être comparés aux solutions analytiques calculées (si elles existent) ou à des résultats expérimentaux.

1.2 Comment importer un module ?

Il y a globalement deux méthodes pour importer une bibliothèque :

- `import numpy as np` : de cette manière, l'ensemble du module numpy est importé (c'est la méthode utilisée dans la plupart des sujets). Il est renommé ici en `np` : cela évite simplement de recopier `numpy` à chaque instruction.

  Exemples : calcul de sin(17) à l'aide du module numpy : `np.sin(17)` ((avec 17 en radians) ; calcul de la racine carrée de 2 : `np.sqrt(2)` ;

  Remarque : on pourra utiliser la même méthode pour n'importer qu'une partie de la bibliothèque nécessaire. On écrira par exemple : `import numpy.random as rd`. Dans ce cas, on importe uniquement le sous-module `random` issu de la bibliothèque `numpy`. On se limite dans l'importation des bibliothèques puisque cela nécessite de la mémoire.

- `from numpy import *` : de cette manière, l'ensemble des sous-modules de la bibliothèque `numpy` est importé (c'est la méthode utilisée parfois dans les sujets de CCINP). Il faut lire "À partir de la bibliothèque numpy, importer tous les modules".

  Exemple : calcul du sinus de 17 avec le module `numpy`, on écrira directement : `sin(17)`. De même pour la racine carrée de 2 : `sqrt(2)`.

  Si on souhaite importer seulement les fonctions sinus et racine carrée depuis le module `numpy`, on pourra écrire : `from numpy import sin,sqrt`

1.3 Type de données utilisées par `numpy`

`Python`est un langage de programmation qui permet de manipuler de nombreux types : entier (`int`), flottants (`float`), chaînes de caractères (`str`), ou encore des listes, regroupement d'éléments pouvant être de types différents.

La bibliothèque `numpy` permet de manipuler des données dans des **tableaux**, nommés `array`. Les tableaux sont plus adaptés que les listes pour réaliser des calculs numériques car ils permettent des opérations usuelles que les listes ne permettent pas nativement.

On considère dans la suite du cours que la bibliothèque `numpy` a été importée en écrivant `import numpy as np`.

	`1+A` renvoie	`2*A` renvoie	`A/2` renvoie	`A-5` renvoie
`A=[[1,2,3],[7,6,5]]`				
`A=np.array([[1,2,3],[7,6,5]])`				

Les tableaux `numpy` se manipulent par ailleurs comme des listes par ailleurs : la longueur du tableau peut être obtenue par `len(A)`, `A[0]` renvoie `[1,2,3]`, `A[0][1]` renvoie 2...

Aucune connaissance spécifique des fonctions de la bibliothèque `numpy` n'est exigible dans le programme officiel. Les sujets de concours sont tenus de rappeler les structures de code nécessaires pour utiliser ces fonctions. On trouvera à la fin de ce document les annexes des concours Centrale Supélec (2022) et CCINP (2023, PSI et PC) relatives aux fonctions et opérations disponibles lors des écrits d'informatique.

La suite du cours donne quelques exemples d'utilisation de `numpy` à travers la résolution d'équations différentielles.

2 Résolution numérique d'une équation différentielle

2.1 Introduction sur un cas simple

L'évolution au cours du temps de nombreux systèmes physiques passe souvent par l'établissement d'une équation différentielle portant sur une grandeur caractéristique. Sous certaines conditions, cette équation admet une solution unique et exacte. Cependant, de nombreuses équations différentielles, notamment non linéaires, ne respectent pas ceci : il faut alors avoir recours à l'analyse numérique.

2.1.1 Description du problème

Considérons le cas d'une chute d'une masse ponctuelle dans l'air. La présence d'air est modélisée par une force de frottement proportionnelle au carré de la vitesse. Si on nomme $y(t)$ la vitesse de la masse, l'application des lois de Newton entraîne la relation : $y' + A.y^2 = B$, avec A et B des constantes réelles.

Pour les applications, nous prendrons $A = 2$ et $B = 10$, et on étudiera le mouvement de la masse sur l'intervalle $[t_0; t_f]$ (on prendra $t_0 = 0$ et $t_f = 2$) avec la condition initiale $y(0) = 0$ (vitesse initiale de la masse nulle). Nous devons donc résoudre l'équation $y' = -2.y^2 + 10$.

Cette équation différentielle n'est pas linéaire et l'outil numérique permet de donner une approximation de sa solution. La **méthode d'Euler** est une des méthodes numériques associée à la résolution des équations différentielles. C'est cette méthode qui est explicitement au programme.

⋆⋆⋆ L'idée première de la méthode est de **discrétiser le problème**. Cela signifie que l'on va subdiviser l'intervalle $[t_0; t_f]$ en n parties égales. On posera $h = \dfrac{t_f - t_0}{n}$ appelé le **pas**. On notera également $t_k = t_0 + k.h$ où $0 \leq k < n$.
Avant discrétisation, on cherchait donc à résoudre :

$$\begin{cases} y' = -2y^2 + 10 \\ y(t_0) = y_0 \end{cases} \text{ sur l'intervalle } [t_0; t_f]$$

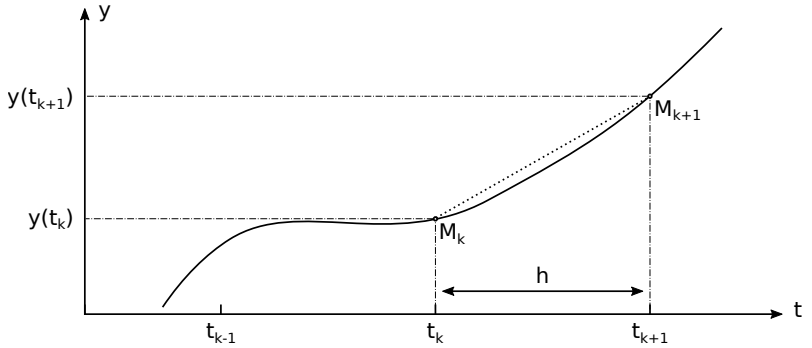
Après discrétisation, on a maintenant à résoudre :

$$\begin{cases} y'(t_k) = -2.(y(t_k))^2 + 10 \\ y(t_0) = y_0 \end{cases}$$

pour toutes les valeurs t_k de l'ensemble $\{t_0, t_1, ..., t_k, ...t_f\}$

⋆⋆⋆ On cherche ensuite à approximer le terme $y'(t_k)$. On choisit d'approximer cette valeur à la pente de la droite passant par les points M_k et M_{k+1} (voir figure ci-dessous). On a donc :

$$y'(t_k) \approx \frac{y(t_{k+1}) - y(t_k)}{t_{k+1} - t_k} \quad \text{donc} \quad y'(t_k) \approx \frac{y(t_{k+1}) - y(t_k)}{h}$$



⋆ ⋆ ⋆ La dernière étape consiste, pour chaque k, à approximer $y(t_k)$ par une valeur y_k en partant de $k = 0$. Pour obtenir y_{k+1} à partir de y_k, sur l'intervalle $[t_k, t_{k+1}]$, on fait donc l'approximation :

$$y'(t_k) \approx \frac{y_{k+1} - y_k}{h}$$

Mais compte-tenu de l'équation différentielle à résoudre, on sait que : $y'(t_k) = -2(y(t_k))^2 + 10 \approx -2(y_k)^2 + 10$. On a donc, pour $0 \leq k < n$:

$$y_{k+1} = y_k + h.(-2.(y_k)^2 + 10) \quad \text{et la condition initiale sur } y_0$$

Exercice

Calculer les trois premiers termes y_0, y_1 et y_2 en supposant $n = 1000$.

2.1.2 Mise en œuvre de la résolution avec des listes

```
## Conditions initiales
t0 = 0
y0 = 0

## Preparation au stockage des yk et des tk dans des listes
T = ..............
Y = ..............

## Donnees de la simulation
tf = ..............
N = 50 #On peut tester avec differentes valeurs de N
h = float(tf)/N

## Resolution
for .............. in .............. :
    tsuivant = ..............
    ysuivant = ..............
    ..............
    ..............

import matplotlib.pyplot as plt

plt.plot(T,Y,'r',label='Methode d\'Euler')
plt.legend()
```

2.1.3 Mise en œuvre de la résolution avec des tableaux `numpy`

```
import numpy as np

## Donnees de la simulation
t0 = ..............  # instant initial
tf = ..............  # instant final
y0 = ..............  # vitesse initiale

N = 50 #On peut tester avec differentes valeurs de N
h = tf/N


## Creation des tableaux numpy

T = ..............
Y=..............


for ..............  in ..............  :
    ..............

plt.plot(T,Y,'b',label='Methode d\'Euler avec des tableaux numpy')
plt.legend()
```

2.1.4 Mise en œuvre de la résolution par `odeint`

Il existe des fonctions `Python` déjà implantées dans la plupart des distributions, qui sont travaillées de manière à résoudre efficacement les équations différentielles. On peut citer la fonction `odeint` disponible dans le sous-module `integrate` de la bibliothèque `scipy`, dont la documentation est donnée ci-dessous.

`from scipy.integrate import odeint`

`sol = odeint(func, y0, t)`

`Parameters`

`func :` `callable(y, t, ...)` ! → Computes the derivative of y at t.

`y0 :` `array` → Initial condition on y (can be a vector).

`t :` `array` → A sequence of time points for which to solve for y.

`Returns`

`sol :` `array` → shape (len(t), len(y0))

Résolution par `odeint`

```
1
2  ## Comparaison avec odeint, optimise pour la resolution d'ED
3  from scipy.integrate import odeint
4
5  def equation (Y_ode,t):
6      return -2*Y_ode**2+10
7
8  Y_ode=odeint(equation, [0], T)
9  plt.plot(T,Y_ode,'b',label='Avec odeint')
10 plt.legend()
```

À utiliser avec discernement !

L'utilisation de la fonction `odeint` - sauf mention contraire - est autorisée dans les disciplines contenant des capacités numériques : physique,chimie et SI.

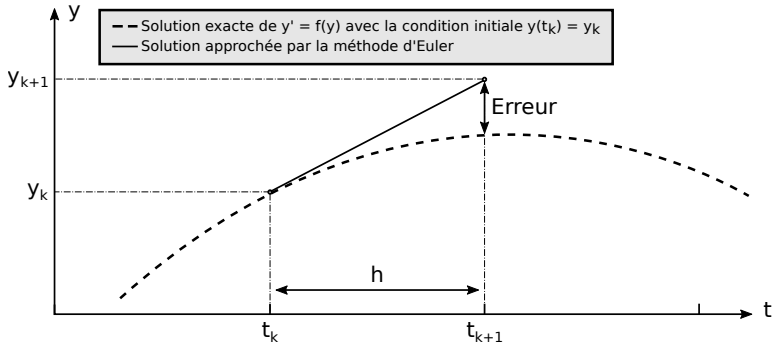
2.2 Complément : précision et complexité de la méthode d'Euler

La théorie montre que sous certaines hypothèses il existe une constante K telle que, pour tout k :

$$|y(t_k) - y_k| \leq K.|h|$$

L'erreur commise $|y(t_k) - y_k|$ est donc proportionnelle à h : la méthode d'Euler est dite **d'ordre 1**. Par exemple, pour augmenter la précision d'un facteur 10, il faut diviser h par 10.

Dans ce cours d'informatique, on s'intéressera surtout à la complexité en temps et en espace. Il est clair que la fonction précédente a une complexité linéaire en la taille n de la subdivision.



3 Généralisation de la méthode d'Euler

3.1 Équation différentielle d'ordre supérieur à 1

Dans votre cours de mathématiques, vous avez abordé la résolution sur un intervalle I des équations différentielles linéaires. Dans ce cas, il est possible de les écrire sous la forme :

$$Y'(t) = A(t)Y(t) + B(t)$$

Avec :

- B, une fonction de I dans $\mathscr{M}_{n,1}(\mathbb{R})$;
- A, une fonction de I dans $\mathscr{M}_{n,n}(\mathbb{R})$;
- et Y la fonction inconnue de I dans $\mathscr{M}_{n,1}(\mathbb{R})$.

Sous certaines conditions (continuité de A et B), cette équation, accompagnée de la condition initiale $Y(t_0) = Y_0$ où t_0 est un élément de I, admet une solution unique. C'est le théorème de Cauchy-Lipschitz. Ce type d'équation est un cas particulier d'équations de la forme $Y'(t) = F(t, Y(t))$, où F est une fonction de $\mathbb{R} \times \mathbb{R}^n$ dans $\mathbb{R}^n$ (on a assimilé $\mathbb{R}^n$ à $\mathscr{M}_{n,1}(\mathbb{R})$).

On va donc expliciter la méthode d'Euler afin de résoudre, sur un intervalle du type $[t_0; t_0+T]$ et avec la condition initiale $Y(t_0) = Y_0$, des équations différentielles de la forme : $Y'(t) = F(t, Y(t))$. Comme précédemment, on retrouve les mêmes étapes. On va donc :

- subdiviser l'intervalle en n parties égales. On posera $h = T/n$ (le pas) et pour $0 \leq k < n$, $t_k = t_0 + kh$.
- approcher, pour chaque k, $Y(t_k)$ par le vecteur Y_k en partant de $k = 0$ (où $Y(t_0) = Y_0$). Pour cela, on remarque que :

$$Y(t_{k+1}) - Y(t_k) = \int_{t_k}^{t_{k+1}} Y'(t)dt$$

Pour obtenir Y_{k+1} à partir de Y_k, on approche, sur l'intervalle $[t_k; t_{k+1}]$, $Y'(t)$ par $Y'(t_k)$, égal du fait de l'équation à $F(t_k; Y(t_k))$, lui même approché par $F(t_k; Y_k)$. Ceci nous donne la relation :

$$Y_{k+1} - Y_k = F(t_k; Y(t_k)) \times h \quad \text{ou encore} \quad Y_{k+1} = Y_k + F(t_k; Y(t_k)) \times h$$

3.2 Exemple du pendule

Rappelons l'équation régissant le mouvement du pendule (sans frottements) : $\dfrac{\mathrm{d}^2\theta}{\mathrm{d}t^2} = \dfrac{g}{l}\sin\theta$ où θ désigne l'angle du fil avec la verticale, l la longueur du fil et g la constante de gravitation. Remarquons qu'il s'agit d'un exemple d'équation différentielle non linéaire.

3.2.1 Écriture de la résolution par la méthode d'Euler avec des listes

Question 1. En posant $\omega(t) = \dfrac{\mathrm{d}\theta}{\mathrm{d}t}(t)$ et $Y(t) = (\theta(t), \omega(t))$, écrire l'équation différentielle sous la forme $Y'(t) = F(t, Y(t))$.

Question 2. Expliquer la méthode de résolution.

Question 3. Compléter la fonction `Euler_pendule(l,T,theta_0,n)`, en utilisant une fonction intermédiaire `F`, qui retourne l'approximation par la méthode d'Euler de $Y(t)$ entre $t = 0$ et $t = T$ avec la valeur de θ en $t = 0$ égale à `theta_0` et une vitesse initiale nulle. On prendra $g = 10\,\mathrm{m.s^{-2}}$.

On stockera les valeurs successives de θ_k et ω_k dans deux listes `list_theta` et `list_omega`.

```
def F(Y):
    FY0 = ..........
    FY1 = ..........
    return [..... , .....]

def Euler_pendule (l,T, theta_0 ,n):
    g =10.0
    list_theta =[ theta_0 ]# list_theta [i] reprEsente theta_ {i}
    list_omega =[0] # list_omega [i] reprEsente omega_ {i}
    h=T/n
    t=0 # reprEsente t_i
    for i in range (...) :
      theta_suivant =...
      omega_suivant =...
      t =...
      ...
      ...
    return [ list_theta , list_omega ]
```

3.2.2 Deuxième méthode avec numpy

On voit que l'on peut en s'arrangeant continuer à utiliser les listes `Python`, mais pour ce genre de problème d'analyse numérique, on préfère souvent utiliser les tableaux qui sont fournis par le module numpy.

Question 4. Écrire une deuxième version, en complétant le modèle ci-dessous, qui utilise les tableaux du module numpy. En notant $Y_k =^t ((y_1)k,(y_2)k)$, on stockera les résultats dans un tableau $TY = (Y_0,...,Y_n)$, initialisé par : `TY=np.zeros([2,n+1])`

```
def F(Y,t):
    ......
    ......

def Euler_pendule2 (l,T, theta_0 ,n):
    h=T/n
    g =10.0
    TY=np. zeros ([2 ,n +1])
    Y0 ......
    ## initialisation
    TY ......
    ## incrementation
    for i in range (1,n+1) :
        TY [:,i]= TY [:,i -1] + ........
    return Y
```

Question 5. Représenter la solution en fonction du temps (tester par exemple avec $l = 1\,\text{m}$, $T = 5\,\text{s}$, $\theta_0 = 0,5\,\text{rad}$ et $n = 5000$).

3.3 Cas des systèmes d'équations différentielles

On peut se ramener à la forme $Y'(t) = F(t, Y(t))$ dans le cas d'un système d'équations différentielles. Prenons par exemple :

Exemple

$$\begin{cases} x'(t) &= t^2 x(t) + \sin y(t) \\ y'(t) &= x(t)^3 \end{cases}$$

Le système peut alors se ramener sous la forme :

Exemple

$$\begin{pmatrix} x'(t) \\ y'(t) \end{pmatrix} = \begin{pmatrix} t^2 x(t) + \sin y(t) \\ x(t)^3 \end{pmatrix} \quad \text{ici} \quad F\left(t, \begin{pmatrix} x \\ y \end{pmatrix}\right) = \begin{pmatrix} t^2 x + \sin y \\ x^3 \end{pmatrix}$$

3.3.1 Mise en œuvre de la résolution avec des tableaux `numpy`

```
## Definition de la fonction F
def Fnumpy(t,Y):
x = Y[0]
y = Y[1]
f0 = (t**2)*x + np.sin(y)
f1 = x**3
return np.array([f0,f1])

## Fonction Euler(t,Y0,n,F) : resoud le systeme d'ED connaissant la
   fonction F et la valeur initiale du vecteur Y

def Eulernumpy(t,Y0,n,F):
## Preparation au stockage des vecteurs Yk
TY=...........
.......... # initialisation de la premiere colonne : Y0
## Donnees de la simulation
h = ..........
## Resolution
for k in range(1,n+1):
## Appel de F(t,Y)
.................................
## Renvoi des resultats
return TY
import matplotlib.pyplot as plt
n = 10000
t0 = ..........
T = ..........
Y0 = .......... # tableau numpy contenant les conditions initiales
temps=.......... # tableau numpy des temps, a generer par linspace(debut,
    fin, nombre de points)
```

```
resnumpy = Eulernumpy(...........)
xnumpy = ...........
ynumpy = ...........
plt.plot(temps,xnumpy,'b',label='x')
plt.plot(temps,ynumpy,'r',label='y')
plt.legend()
plt.grid()
```

Annexe Centrale

Dans l'énoncé du sujet d'informatique du concours Centrale Supélec 2022, on peut lire :

"Les seuls langages de programmation autorisés dans cette épreuve sont `Python` et SQL. Pour répondre à une question, il est possible de faire appel aux fonctions définies dans les questions précédentes. Dans tout le sujet on suppose que les bibliothèques `math`, `numpy` et `turtle` sont rendues accessibles grâce à l'instruction :

```
import math, numpy as np, turtle
```

Si les candidats font appel à des fonctions d'autres bibliothèques, ils doivent préciser les instructions d'importation correspondantes.

Dans ce sujet, le terme « liste » appliqué à un objet Python signifie qu'il s'agit d'une variable de type `list`. Les termes « vecteur » et « tableau » désignent des objets `numpy` de type `np.ndarray`, respectivement à une dimension ou de dimension quelconque. Enfin le terme « séquence » représente une suite itérable et indiçable, indépendamment de son type `Python`, ainsi un tuple d'entiers, une liste d'entiers et un vecteur d'entiers sont tous trois des « séquences d'entiers »."

Le sujet contient aussi une annexe présentant les opérations et fonctions disponibles pour répondre aux questions.

Opérations et fonctions disponibles

Fonctions

- `a % b` calcule le reste de la division euclidienne de `a` par `b`, son signe est toujours celui de `b`
  `5 % 3` → 2, `-5 % 3` → 1.
- `isinstance(o, t)` teste si l'objet `o` est de type `t`
  `isinstance(-13, int)` → `True`, `isinstance((1, 2, 3), tuple)` → `True`.
- `range(n:int)` renvoie la séquence des `n` premiers entiers ($0 \to n-1$)
  `list(range(5))` → `[0, 1, 2, 3, 4]`.
- `range(d:int, f:int, s:int)` construit une séquence d'entiers espacés de `s` débutant à `d` et finissant avant `f`
  `list(range(0, -10, -2))` → `[0, -2, -4, -6, -8]`.
- `enumerate(s)` itère sur la séquence `s` en renvoyant, pour chaque élément de `s`, un couple formé de son indice et de l'élément considéré
  `list(enumerate([3, 1, 4, 1, 5]))` → `[(0, 3), (1, 1), (2, 4), (3, 1), (4, 5)]`.
- `min(a, b, …)`, `max(a, b, …)` renvoie son plus petit (respectivement plus grand) argument
  `min(2, 4, 1)` → 1, `max("Un", "Deux")` → `"Un"`.
- `math.sqrt(x)` calcule la racine carrée du nombre `x`.
- `round(n)` arrondit le nombre `n` à l'entier le plus proche.
- `math.pi` valeur approchée de la constante π.

Opérations sur les listes

- `len(u)` donne le nombre d'éléments de la liste `u`
  `len([1, 2, 3])` → 3, `len([[1,2], [3,4]])` → 2.
- `u.count(e)` renvoie le nombre d'éléments de la liste `u` égaux à `e`
  `[1, 2, 3, 1, 5].count(1)` → 2.
- `u + v` construit une liste constituée de la concaténation des listes `u` et `v`
  `[1, 2] + [3, 4, 5]` → `[1, 2, 3, 4, 5]`.
- `n * u` construit une liste constituée de la liste `u` concaténée `n` fois avec elle-même
  `3 * [1, 2]` → `[1, 2, 1, 2, 1, 2]`.
- `u[i] = e` remplace l'élément d'indice `i` de la liste `u` par `e`.
- `u[i:j] = v` remplace les éléments de la liste `u` dont les indices sont compris dans l'intervalle $[\![i, j[\![$ par ceux de la séquence `v` (peut modifier la longueur de la liste `u`).
- `u.append(e)` ajoute l'élément `e` à la fin de la liste `u` (identique à `u[len(u):] = [e]`). On suppose que cette opération a une complexité temporelle en $O(1)$.

— `u.extend(v)` ajoute les éléments de la séquence `v` à la fin de la liste `u` (identique à `u[len(u):] = v`. On suppose que cette opération a une complexité temporelle en $O(\texttt{len}(\texttt{v}))$.

— `u.insert(i, e)` insère l'élément `e` à la position d'indice `i` dans la liste `u` (en décalant les éléments suivants) ; si `i >= len(u)`, `e` est ajouté en fin de liste (identique à `u[i:i] = [e]`).

— `del u[i]` supprime de la liste `u` son élément d'indice `i` (identique à `u[i:i+1] = []`).

— `del u[i:j]` supprime de la liste `u` tous ses éléments dont les indices sont compris dans l'intervalle $[\![\texttt{i}, \texttt{j}[\![$ (identique à `u[i:j] = []`).

— `u.reverse()` modifie la liste `u` en inversant l'ordre de ses éléments. On suppose que cette opération a une complexité temporelle en $O(\texttt{len}(\texttt{u}))$.

— `e in u` teste si l'élément `e` apparait dans la liste `u`.

Opérations sur les tableaux

— `np.array(s)` crée un nouveau tableau contenant les éléments de la séquence `s`. La forme de ce tableau est déduite du contenu de `s`

`np.array([[1, 2, 3], [4, 5, 6]])` $\rightarrow \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$.

— `a.ndim` nombre de dimensions du tableau `a`.

— `a.shape` tuple donnant la taille du tableau `a` pour chacune de ses dimensions.

— `a.size` nombre total d'éléments du tableau `a`.

— `np.around(a)` produit un tableau dont les éléments sont ceux du tableau `a` arrondis à l'entier le plus proche.

— `a @ b` calcule le produit matriciel des deux tableaux `a` et `b`. Les dimensions de `a` et `b` doivent être compatibles. On suppose que la multiplication d'une matrice $n \times n$ et d'un vecteur a une complexité temporelle en $O(n^2)$.

Module graphique `turtle`

Le module `turtle`, inspiré du langage Logo, simule une « tortue » robot qui se déplace sur l'écran. Cette tortue est munie d'un « crayon » qui, quand il est baissé, trace à l'écran la trajectoire de la tortue. Au début du programme, la tortue est située au centre de la fenêtre de visualisation, crayon baissé et orientée vers la droite de l'écran.

— `turtle.pendown()`, `turtle.penup()` baisse ou lève le crayon.

— `turtle.right(a)`, `turtle.left(a)` fait pivoter la tortue sur place à droite ou à gauche de `a` degrés.

— `turtle.forward(n)`, `turtle.back(n)` fait avancer ou reculer la tortue de `n` pixels (si `n` est négatif, le mouvement est inversé) ; `n` peut être un entier ou un nombre à virgule flottante.

— `turtle.circle(r, a)` fait parcourir par la tortue un arc de cercle de `a` degrés dont le centre est situé `r` pixels à gauche de la tortue. Si `r` est positif, le centre est effectivement à gauche de la tortue et l'arc de cercle est parcouru dans le sens trigonométrique ; si `r` est négatif, le centre est situé à droite de la tortue et l'arc de cercle est parcouru dans le sens horaire. Si `a` est positif, l'arc de cercle est parcouru en marche avant, la tortue tourne donc à gauche si `r` est positif et à droite si `r` est négatif. Si `a` est négatif, l'arc de cercle est parcouru en marche arrière, la tortue recule sur sa gauche si `r` est positif et sur sa droite si `r` est négatif.

— `turtle.home()` ramène la tortue au centre de la fenêtre, orientée vers la droite de l'écran.

— `turtle.reset()` efface la fenêtre et repositionne la tortue au centre, orientée vers la droite de l'écran.

Fonctions SQL

— `COUNT(*)` fonction agrégative qui retourne le nombre de lignes chaque groupe.

— `COUNT(e)` fonction agrégative qui calcule le nombre de valeurs non nulles chaque groupe de lignes.

— `MIN(e)`, `MAX(e)` fonctions agrégatives qui calculent respectivement le minimum et le maximum de `e` pour chaque groupe de lignes ; `e` est une expression de type numérique, chaine de caractère ou date, heure.

— `SUM(e)` fonction agrégative qui somme `e` pour chaque groupe de lignes ; `e` est une expression de type numérique ou durée.

— `EXTRACT(c FROM d)` extrait la composante `c` de la date `d`, `d` peut être n'importe quelle expression de type `date`, `timestamp`, `time` ou `interval` et `c` peut prendre une des valeurs (liste non exhaustive) `'century'`, `'year'`, `'quarter'` (trimestre, 1 à 4), `'month'` (mois, 1 à 12), `'day'` (jour du mois, 1 à 31), `'dow'` (jour de la semaine, 0 à 6, 0 désignant dimanche), `'doy'` (jour de l'année, 1 à 366).

Annexe CCINP

3.4 Épreuves d'informatique PSI

Dans l'épreuve d'informatique du CCINP 2022, on peut lire : "Dans tout le sujet, il sera supposé que les modules python `numpy`,`matplotlib.pyplot` sont déjà importés dans le programme. Les fonctions et méthodes autorisées sont indiquées dans l'**annexe** à la fin du **Document Réponse**. De manière générale, l'utilisation des méthodes autres que les méthodes `append` et `remove` ne sera pas acceptée. L'utilisation des fonctions **min** ou **max** sera proscrite."L'annexe en question est la suivante :

ANNEXE

Rappels des syntaxes en Python

Remarque : sous Python, l'import du module numpy permet de réaliser des opérations pratiques sur les tableaux : from numpy import *. Les indices de ces tableaux commencent à 0.

	Python
tableau à une dimension	`L=[1,2,3]` (liste) `v=array([1,2,3])` (vecteur)
accéder à un élément	`v[0]` renvoie 1 (`L[0]` également)
ajouter un élément	`L.append(5)` uniquement sur les listes
supprimer la première occurrence d'un élément d'une liste	`L.remove(element)`
calculer le produit vectoriel entre 2 vecteurs coplanaires	`cross([x1,y1],[x2,y2])` donne $x1\,y2 - x2\,y1$
évaluer la racine carrée d'un nombre flottant	`sqrt(4)`
évaluer la valeur absolue d'un nombre flottant	`abs(-2)`
tableau de 0 (2 lignes, 3 colonnes)	`zeros((2,3))`
séquence équirépartie quelconque de 0 à 10.1 (exclu) par pas de 0.1	`arange(0,10.1,0.1)`
séquence de 100 valeurs équiréparties de 0 à 10.0 (inclus)	`linspace(0,10,100)`
affiche y en fonction de x où x et y sont des listes ou vecteurs de même dimension.	`plot(x,y)`

Modification des énoncés

Selon les années, les consignes fournies et les fonctions rappelées en annexe peuvent varier (autorisation ou interdiction de certains modules, de certaines fonctions...). Il faut donc être particulièrement vigilant dans la lecture de l'énoncé !

À titre d'exemple, voici l'annexe de l'épreuve d'informatique du CCINP 2023 :

ANNEXE

Rappels des syntaxes en Python

Fonctionnalités	Python
détermination du nombre de zéros dans la liste `X`	`X.count(0)`
définir une chaîne de caractères	`mot = 'Python'`
taille d'une chaîne	`len(mot)`
extraire des caractères (avec le même fonctionnement des indices que pour les extractions de sous-listes)	`mot[2:7]`
éliminer le \n en fin d'une ligne	`ligne.strip()`
découper une chaîne de caractères selon un caractère passé en argument. On obtient une liste qui contient les caractères séparés. Dans l'exemple ci-contre, on découpe à chaque occurrence du caractère ","	`mot.split(',')`
ouverture d'un fichier en lecture et lecture des données (`data` est une liste de chaînes de caractères dont la taille est le nombre de lignes du fichier lu)	`with open('nom_fichier','r') as f:` `  data = f.readlines()`

3.5 Épreuve de modélisation PC

L'épreuve de modélisation 2023 de la filière PC présentait de manière complète les fonctions et bibliothèques pouvant être utilisées lors de l'épreuve.

I. - Bibliothèque NUMPY

Dans les exemples ci-dessous, la bibliothèque **`numpy`** a préalablement été importée à l'aide de la commande : **`import numpy as np`**.

On peut alors utiliser les fonctions de la bibliothèque, dont voici quelques exemples :

- **`np.linspace(start, stop, N point) :`**
  - description : renvoie un nombre d'échantillons espacés uniformément, calculés sur l'intervalle [start, stop]
  - argument d'entrée : début, fin et nombre d'échantillons dans l'intervalle
  - argument de sortie : un tableau

Commande	Résultat
`np.linspace(1, 4, 5)`	`[1., 1.75, 2.5, 3.25, 4.]`

- **`np.zeros(i) :`**
  - description : renvoie un tableau de taille i rempli de zéros.
  - argument d'entrée : un scalaire
  - argument de sortie : un tableau

Commande	Résultat
`np.zeros(5)`	`[0, 0, 0, 0, 0]`

- **`np.array(liste)` :**
  - description : crée une matrice (de type tableau) à partir d'une liste.
  - argument d'entrée : une liste définissant un tableau à 1 dimension (vecteur) ou 2 dimensions (matrice)
  - argument de sortie : un tableau (matrice)

Commande	Résultat
`np.array([4, 3, 5])`	`[4, 3, 5]`

- **`A[i,j]` :**
  - description : retourne l'élément (i + 1, j + 1) de la matrice A. Pour accéder à l'intégralité de la ligne i + 1 de la matrice A, on écrit A[i, :]. De même, pour obtenir toute la colonne j + 1 de la matrice A, on utilise la syntaxe A[: , j]
  - argument d'entrée : une liste contenant les coordonnées de l'élément dans le tableau A
  - argument de sortie : l'élément (i + 1, j + 1) de la matrice A

Commande	Résultat
`A = np.array([[1, 2, 1],[4, 6, 3], [1, 3, 8]])` `A[1, 2]`	`3`

- **`chaine.split(motif)` :**
  - description : divise une chaîne de caractères en une liste ordonnée de sous-chaînes, place ces sous-chaînes dans un tableau et retourne le tableau. La division est effectuée en recherchant un motif
  - argument d'entrée : motif
  - argument de sortie : un tableau

Commande	Résultat
`A = 'azert yuiop'` `A.split(' ')`	`['azert',` `'yuiop']`

II. - Bibliothèque MATPLOTLIB.PYPLOT

Cette bibliothèque permet de tracer des graphiques. Dans les exemples ci-dessous, la bibliothèque `matplotlib.pyplot` a préalablement été importée à l'aide de la commande : `import matplotlib.pyplot as plt`.

- description : fonction permettant de tracer un graphique de n points dont les abscisses sont contenues dans le vecteur x et les ordonnées dans le vecteur y. Cette fonction doit être suivie de la fonction `plt.show()` pour que le graphique soit affiché
- argument d'entrée : un vecteur d'abscisses x (tableau de n éléments) et un vecteur d'ordonnées y (tableau de n éléments). La chaîne de caractères 'SC' précise le style et la couleur de la courbe tracée. Des valeurs possibles pour ces deux critères sont :

Valeurs possibles pour S (style) :

Description	Ligne continue	Ligne traitillée	Marqueur rond	Marqueur plus
Symbole S	-	--	o	+

Valeurs possibles pour C (couleur) :

Description	bleu	rouge	vert	noir
Symbole C	b	r	g	k

- argument de sortie : un graphique

```
x= np.linspace(3,25,5)
y=sin(x)
plt.plot(x,y,'-b') # tracé d'une ligne bleue continue
plt.title('titre_graphique') # titre du graphe
plt.xlabel('x') # titre de l'axe des abscisses
plt.ylabel('y') # titre de l'axe des ordonnées
plt.show()
```