Exercices de Python

par Raymond Moché
ancien élève de l'École normale d'instituteurs de Douai et de l'École normale supérieure de Saint Cloud
ancien professeur de l'Université des Sciences et Technologies de Lille
et des universités de Coïmbre et de Galatasaray

http ://gradus-ad-mathematicam.fr

Table des matières

Chapitre 1

Liste des Énoncés

Chapitre 2 : Fonctions, boucles, instructions conditionnelles

Énoncé n° 2.1 [*] : Pour commencer : des calculs.

Quelle est la valeur de π ? de e ? de $\pi \cdot e$? Calculer $\dfrac{\sin(3) \cdot e^2}{1 + \tan\left(\frac{\pi}{8}\right)}$.

Mots-clefs : constantes pré-définies e, π, calcul numérique, module `math`, fonctions `sin` et `tan`.

Énoncé n° 2.2 [*] : Pour commencer : graphes simples.

1 - Tracer le segment de droite d'extrémités (1,3) et (15,-2), dans un repère orthonormé.
2 - Tracer le graphe de la fonction `sinus` quand la variable varie de 1 à 15 radians.

Mots-clefs : Instructions simples de tracer des graphes, commandes `linspace(a,b,n)` et `array([a,…,x])` du module `numpy`, fonctions vectorisées.

Énoncé n° 2.3 [*] : Représentations graphiques avec Python.

Pour tracer un graphique, Python en calcule des points consécutifs et les relie par un segment. Habituellement, on obtient ainsi une bonne approximation de la courbe à tracer quand il y a suffisamment de points car alors, nos yeux ne font plus la différence.
On se propose d'étudier des approximations du graphique de la fonction `sin()` lorsque la variable `x` varie sur l'intervalle `[0,`π`]`.
1 - Tracer l'approximation du graphique obtenue en calculant ses points d'abscisse 0, $\pi/2$ et π.
2 - Pour `n` valant successivement 3, 6, 10, 20, tracer l'approximation du graphique obtenue en calculant les points dont les abscisses sont données par la commande `linspace(0,pi,n+1)` (cette représentation graphique est une ligne polygonale à `n` côtés) .

Mots-clefs : Représentation graphique d'une fonction, repère orthonormé, commande `axis('equal')`, d'autres commandes graphiques.

Énoncé n° 2.4 [*] : Programmer des fonctions : valeur absolue, volume du cône.

1 - Programmer une fonction qui retourne la valeur absolue de tout nombre.
2 - Programmer une fonction qui retourne le volume de tout cône droit à base circulaire, de hauteur `h` cm et de rayon `r` cm.

Mots-clefs : programmer une fonction, instruction conditionnelle `if … else …`, `return`, importer une constante ou une fonction d'un module de Python.

Énoncé n° 2.5 [*] : Trinôme du second degré.

Résoudre l'équation du second degré (E) : $ax^2 + bx + c = 0$ ($a \neq 0$)

Mots-clefs : Fonction `sqrt`, à importer du module `math`, sortie d'une fonction par `return` ou `print`, instruction conditionnelle `if: … elif: … else: …`.

Énoncé n° 2.6 [**] : Minimum de 2, 3, 4 nombres (instructions conditionnelles).

1.a - Programmer une fonction qui retourne le minimum de 2 nombres.
1.b - Programmer une fonction qui retourne le minimum et le maximum de 2 nombres.
2 - Programmer une fonction qui retourne le minimum de 3 nombres.
3 - Programmer une fonction qui retourne le minimum de 4 nombres.

Mots-clefs : Programmation de fonctions, instruction conditionnelle `if … else …`, `return`, négation d'une proposition logique, importation de constantes et de fonctions pré-définies.

Énoncé n° 2.7 [***] : Alignement de 3 points.

On se donne, à l'aide de leurs coordonnées, 3 points distincts A, B et C d'un plan rapporté à un repère orthonormé. On appelle d_1, d_2 et d_3 les distances de B à C, de A à C et de A à B. On peut supposer que $d_1 \leqslant d_2 \leqslant d_3$.
1 - Démontrer que les points sont alignés si et seulement si $d_3 = d_1 + d_2$.
2 - Sans utiliser la fonction racine carrée, programmer un algorithme qui retourne, à partir de la liste des coordonnées de A, B et C, "Les points sont alignés" ou "Les points ne sont pas alignés", suivant le cas.

Mots-clefs : Calculer avec seulement des additions et des multiplications, transformer un problème en un problème équivalent. Commentaires : tester l'égalité de 2 nombres.

Énoncé n° 2.8 [*] : Affectation de données

Soit f une fonction définie sur $\mathbb{R}$ par $f(x) = x^2 - 5x + 3$ et k un nombre réel. On souhaite déterminer tous les entiers n compris entre deux entiers donnés a et b ($a < b$) tels que $f(n) < k$.

1 - -10 est-il solution de $f(n) < 5$? Même question pour 0.

2 - Écrire un script Python appelant un nombre et affichant "oui" s'il est solution de $f(n) < 5$, "non" sinon.

3 - Modifier le script pour appeler k et afficher successivement tous les entiers solutions de $f(n) < k$ sur l'intervalle $[-10, 10]$.

4 - Modifier le script pour demander aussi a et b.

Mots-clefs : Affecter des données à un script (commandes **input** et **eval**), définir une fonction, instruction conditionnelle `if …then …else …`, boucle `pour`, conditions de type booléen, variables booléennes, masques.

Énoncé n° 2.9 [*] : Algorithme d'Euclide

> **1 -** Programmer une fonction qui, étant donnés deux entiers positifs `a` et `b`, retourne leur PGCD calculé selon l'algorithme d'Euclide.
> **2 -** En déduire le calcul de leur PPCM.

Mots-clefs : reste et quotient de la division euclidienne (`%` et `//`), algorithme d'Euclide, boucle `tant que`, importation de fonctions, module `math`.

Chapitre 3 : Listes

Énoncé n° 3.1 [*] : Simuler `n` lancers d'un dé équilibré, sous forme de liste

> En utilisant la fonction `randint` du module `random`, simuler, sous forme de liste, `n` lancers d'un dé équilibré.

Mots-clefs : liste, module `random`, commandes `randint` et `append`, boucle `pour`.

Énoncé n° 3.2 [*] : Maximum d'une liste de nombres

> **1 -** Calculer le maximum `M` d'une liste `L` de nombres, de longueur `l` comme suit :
> (a) On pose `M=L[0]`.
> (b) Ensuite, si `l` $\geqslant 2$, `i` prenant successivement les valeurs 1, ..., `l-1`, on pose `M=L[i]` si `L[i] > M`.
> On présentera la solution sous la forme d'une fonction.
> **2 -** Application : calculer le maximum d'une liste de 10, puis de 10000 nombres réels engendrée par la fonction `reels()`, cf.(3).
> **3 -** Calculer le minimum `m` de `L` en utilisant le calcul du maximum programmé ci-dessus.

Mots-clefs : boucle `pour`, instruction conditionnelle `if`, appeler une fonction nouvellement programmée, méthode `list.append()` des listes, primitives `min()` et `max()`.

Énoncé n° 3.3 [*] : Ranger une liste de nombres

> Une liste de nombres `L` est donnée.
> **1 -** À l'aide de la primitive `min()`, ranger cette liste dans l'ordre croissant (le premier terme `m` de la liste à calculer sera le minimum de `L`, le suivant sera le minimum de la liste `L` dont on aura retiré `m`, *etc*).
> **2 -** Ranger `L` dans l'ordre décroissant.

Mots-clefs : primitives `min()` et `max()`, boucle `pour`, méthodes `list.sort()` et `list.reverse()`, fonction `len()` (pour les listes), primitive `sorted()`.

Chapitre 4 : Matrices (m,)

Énoncé n° 4.1 [**] : Le lièvre et la tortue, jeu équitable ?

On lance un dé équilibré. Si le 6 sort, le lièvre gagne. Sinon la tortue avance d'une case et on rejoue. La tortue gagne si elle parvient à avancer de `n` cases ($n \geqslant 1$ donné).

1 - Modéliser et simuler ce jeu à l'aide de matrices-lignes.

2 - Calculer la probabilité pour que le lièvre gagne, que la tortue gagne.

3 - Existe-t-il une valeur de `n` pour laquelle le jeu est équitable ?

Mots-clefs : matrice-ligne, modélisation, mathématiques, probabilités, simulation.

Énoncé n° 4.2 [**] : Maximum et minimum d'une matrice-ligne

1 - Calculer le maximum d'une matrice-ligne de nombres `M` en procédant comme suit :

(a) - on choisit arbitrairement l'un de ces nombres que l'on note `ma` et on efface les nombres ⩽ `ma`.

(b) - s'il n'en reste plus, `ma` est le maximum recherché.

(c) - sinon, on recommence (a) et (b).

2 - Calculer le minimum `mi` de `M`.

Mots-clefs : matrices, masque, maximum et minimum, fonctions primitives `max` et `min`, boucle `tant que`, définition d'une fonction, appel d'une nouvelle fonction.

Chapitre 5 : Matrices ou matrices (n,m)

Énoncé n° 5.1 [*] : Calculer la trace d'une matrice.

Soit `M` une matrice à `n` lignes et `m` colonnes.
1 - On suppose d'abord que `M` est carrée (autrement dit `n=m`). Calculer la somme `t` de ses éléments diagonaux [a] et la somme `t2` des éléments de sa deuxième diagonale [b].
2 - Même question quand on ne suppose plus que `n=m` [c].

a. c'est à dire ceux dont l'indice de ligne est égal à l'indice de colonne
b. c'est à dire ceux dont l'indice `i` de ligne et l'indice `j` de colonne vérifient $i + j = n + 1$.
c. `t` s'appelle la trace de `M`.

Mots-clefs : module `numpy`, extraire un élément, une sous-matrice d'une matrice, fonction `shape()`, boucle `pour`, changer l'ordre des lignes ou des colonnes d'une matrice, fonction `rand()` du module `random` de `numpy`, fonctions `diag()`, `sum()` et `trace()` de `numpy`.

Énoncé n° 5.2 [**] : Ce carré est-il un carré magique ?

Convenons qu'une matrice carrée `M` d'ordre `n` est un carré magique d'ordre `n` si ses éléments sont les entiers de 1 à n^2 disposés de sorte que leurs sommes sur chaque ligne, sur chaque colonne et sur les deux diagonales soient égales. La valeur commune de ces sommes est appelée constante magique de `M`.
1 - S'il existe un carré magique d'ordre `n`, combien vaut sa constante magique ?
2 - `LuoShu = array([[4,9,2],[3,5,7],[8,1,6]])` est-il un carré magique ? Si oui, quelle est sa constante magique ?
3 - Même question pour `Cazalas = array([[1,8,53,52,45,44,25,32], [64,57,12,13,20,21,40,33], [2,7,54,51,46,43,26,31], [63,58,11,14,19,22,39,34], [3,6,55,50,47,42,27,30], [62,59,10,15,18,23,38,35], [4,5,56,49,48,41,28,29], [61,60,9,16,17,24,37,36]])`.
4 - Même question pour `Franklin = array([[52,61,4,13,20,29,36,45], [14,3,62,51,46,35,30,19], [53,60,5,12,21,28,37,44], [11,6,59,54,43,38,27,22], [55,58,7,10,23,26,39,42], [9,8,57,56,41,40,25,24], [50,63,2,15,18,31,34,47], [16,1,64,49,48,33,32,17]])`.

Mots-clefs : `V.size`, `M.sum(axis=1)`, `M.sum(axis=0)`, `trace(M)`, `M[arange(n-1,-1,-1), arange(0,n)]` (extraction d'une matrice 1-dimensionnelle de `M`), `hstack((,))`, `min()`, `max()`.

Énoncé n° 5.3 [***] : Transposer une matrice

Programmer une fonction qui, étant donné une matrice numérique `A`, retourne la matrice `B` dont la première colonne est la première ligne de `A`, la seconde la deuxième ligne de `A`, etc. `B` s'appelle la transposée de `A`.

Mots-clefs : Boucle `pour` imbriquée dans une autre, commande `range`, module `numpy`, fonctions `shape()` (dimension d'une matrice), `zeros()`, `vstack()` et `hstack()` (empilements de matrices), `transpose()`, méthode (de l'objet matrice) `.T`.

Chapitre 2

Fonctions, boucles, instructions conditionnelles

Liste des exercices :

Énoncé n° 2.1 [*] : Pour commencer : des calculs.

Énoncé n° 2.2 [*] : Pour commencer : graphes simples.

Énoncé n° 2.3 [*] : Représentations graphiques avec Python.

Énoncé n° 2.4 [*] : Programmer des fonctions : valeur absolue, volume du cône.

Énoncé n° 2.5 [*] : Trinôme du second degré.

Énoncé n° 2.6 [**] : Minimum de 2, 3, 4 nombres (instructions conditionnelles).

Énoncé n° 2.7 [***] : Alignement de trois points

Énoncé n° 2.8 [*] : Affectation de données

Énoncé n° 2.9 [*] : Algorithme d'Euclide

Cet ouvrage est écrit à l'aide de l'environnement `Spyder` de Python. La plupart des graphes y sont tracés à l'aide du module `matplotlib`, l'exception étant l'usage de la tortue (`turtle`)[1]. Au début d'un script contenant un tracé, il faut charger les modules cités ci-dessous. Dans notre environnement, c'est inutile. Aussi, pour qu'un éventuel lecteur ne les oublie pas, nous les avons laissés en commentaire :

```
# from numpy import *
# from matplotlib.pyplot import *
# from pylab import *
```

Avec un réglage différent de `Spyder` ou avec un autre environnement de programmation comme **idle**, il faudra peut-être activer ces commandes.

Représenter une fonction graphiquement est compliqué car les tracés peuvent être l'objet de nombreuses options. Les quelques exercices sur les graphes qui suivent sont des modèles de référence pour débuter, qui se compliquent au fur et à mesure. Il existe une excellente documentation sur `matplotlib` en français, avec beaucoup d'exemples, à savoir [16]. Ce texte en est en grande partie inspiré.

1. pour les tracés qui s'y prêtent

2.1 [*] Pour commencer : des calculs.

Quelle est la valeur de π ? de e ? de $\pi \cdot e$? Calculer $\dfrac{\sin(3) \cdot e^2}{1+\tan\left(\frac{\pi}{8}\right)}$.

Mots-clefs : constantes pré-définies e, π, calcul numérique, module `math`, fonctions `sin` et `tan`.

Dans Python, il y a des constantes pré-défines, comme e et π, des fonctions standard (utilisables immédiatement)[2] et une foule d'autres fonctions qui sont regroupées par thème dans des bibliothèques de fonctions ou `modules`. Par exemple, `sin`, `tan` sont des fonctions du module `math`. Si dans un même script, on veut se servir de la constante e et de la fonction `sin`, il faudra les importer, en important tout le module `math` comme ceci :

```
from math import *
```

ou mieux[3], en important seulement e et `sin`, comme cela :

```
from math import e, sin
```

Dans l'exemple qui suit, on a oublié d'importer π, ce qui provoque un message d'erreur :

```
from math import e,sin
e
Out[2]: 2.718281828459045
sin(pi/6)
Traceback (most recent call last):
  File "<ipython-input-3-dbaab9b2f48b>", line 1, in <module>
    sin(pi/6)
NameError: name 'pi' is not defined
```

Nous ne soucions pas du format des nombres dans cette prise de contact[4]. Les règles de calcul sont les règles usuelles. Le problème posé est trivial. Comme il y a peu à écrire, on peut travailler directement dans la console[5]. Ouvrons-en une nouvelle :

```
from math import e, pi, sin, tan
pi
Out[2]: 3.141592653589793
e
Out[3]: 2.718281828459045
pi*e
Out[4]: 8.539734222673566
sin(3)*e**2/(1+tan(pi/8))
Out[5]: 0.7373311103636621
```

2. En anglais, les "built-in functions", voir [10].
3. pour économiser du temps, de l'espace-mémoire, *etc.*
4. Nous utilisons donc les réglages de Python par défaut.
5. Normalement, on écrit dans l'interpréteur, qui est un traitement de texte adapté à Python. On écrit plus vite, les retraits - très importants en programmation Python - se font automatiquement, on peut corriger facilement, *etc.*

2.2 [*] Pour commencer : graphes simples.

1 - Tracer le segment de droite d'extrémités (1,3) et (15,-2), dans un repère orthonormé.
2 - Tracer le graphe de la fonction `sinus` quand la variable varie de 1 à 15 radians.

Mots-clefs : Instructions simples de tracer des graphes, commandes `linspace(a,b,n)` et `array([a,…,x])` du module `numpy`, fonctions vectorisées.

Solution

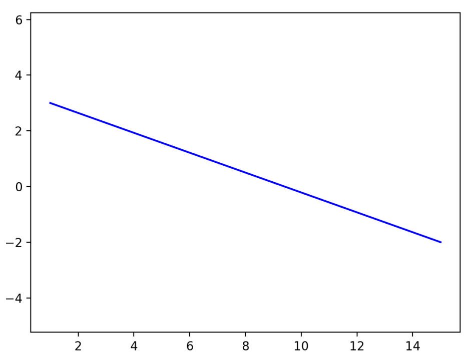
Cet exercice[6] peut paraître débile. En fait, il est important, tout graphe se réduisant à une succession de segments de droite. En effet, pour tracer un graphe, `Python` calcule les coordonnées de points du graphe et relie les points consécutifs par un segment (voir 2.3). Si ces points sont assez nombreux, les imperfections de nos yeux font que l'on voit une courbe lisse et non une ligne polygonale. De plus, il y a mieux : `Python`[7] ne trace pas vraiment des segments de droite mais des ensembles de pixels plus ou moins bien disposés le long du segment à tracer, voir [18]. Heureusement, grâce à nos mauvais yeux, tout s'arrange.

1 - Voici un script simple solution du problème :

```
# from numpy import *
# from matplotlib.pyplot import *
# from pylab import *
figure()# Création d'une figure.
x = array([1,15])# x de type 'vecteur' (= matrice à une dimension).
y = array([3,-2])# y de type 'vecteur' (= matrice à une dimension).
plot(x,y,color='blue')# Le tracé sera bleu.
axis('equal')# Le repère sera orthonormé.
savefig("at34gdroite")# Sauvegarde de la figure (fichier at34gdroite.png).
show()# Sinon, la figure n'apparaît pas.
```

Commentaires :

1 - Un vecteur est une matrice ('`array`') pour `Python`, qui n'a qu'une dimension (le nombre de ses éléments)[8]. `x` ne comprend ici que les abscisses de l'origine et de l'extrémité du segment à tracer, de même `y` avec les ordonnées.
2 - Les commandes `color='blue'`, `axis('equal')` et `savefig("at34gdroite")` ne sont pas indispensables. Voici la figure obtenue grâce à la commande `savefig("at34gdroite"` :



6. qui a pour but de fournir des modèles de tracés de graphes
7. ainsi que tous les autres logiciels de calcul
8. Une matrice à une dimension est très perturbant pour un mathématicien et est source d'erreurs de programmation.

Vers le graphe d'une fonction 'quelconque' :

Dans le script ci-dessous, après avoir remarqué que la droite qui porte le segment à tracer a pour équation

$$y = -5/14 * x + 47/14$$

nous traçons le graphe de la fonction

$$f(x) = -5/14 * x + 47/14$$

entre ses points d'abscisses 1 et 15 :

```
# from numpy import *
# from matplotlib.pyplot import *
# from pylab import *
figure()# Création d'une figure.
def f(x):
    return -5/14*x+47/14
x = linspace(1,15,2)# Commande du module numpy qui produit array([1,15]).
y = f(x)# Commande du module numpy qui produit array([3,-2]).
plot(x,y,color='blue')# Le tracé sera bleu.
axis('equal')# Le repère sera orthonormé.
savefig("at34gdroite")# Sauvegarde de la figure (fichier at34gdroite.png).
show()# Sinon, la figure n'apparaît pas.
```

Ce script est un modèle à retenir. On obtient la même figure que précédemment au moyen d'instructions qui sont certes plus compliquées mais qui conviennent à la représentation graphique d'une fonction 'quelconque'.

Précisions supplémentaires sur le script ci-dessus :

• On constate que f est définie par les instructions

```
def f(x):
    return -5/14*x+47/14
```

• La commande `linspace(a,b,n)` du module `numpy` produit `n` nombres dont `a` et `b`, qui partagent l'intervalle `[a,b]` en `n-1` intervalles de même longueur. Par exemple `linspace(1,15,2)` produit `array([1,15])`.
• Si `x=linspace(a,b,n)`, `f(x)` produit les valeurs de `f` aux `n` points de `x`. On dit que `f` est une fonction vectorisée.

Vers une figure de meilleure qualité

La figure obtenue ci-dessus est très sommaire. Son plus grave défaut est que les axes de coordonnées n'apparaissent pas. On peut repérer les abscisses sur la graduation du bas, les ordonnées sur la graduation de gauche. Pour corriger cela, on peut effacer les bords du cadre à droite et en haut et faire glisser verticalement l'axe gradué du bas et horizontalement l'axe gradué de gauche pour les faire passer par le point (0,0). On obtient ainsi les axes de coordonnées. C'est l'objet du groupe de commandes suivant :

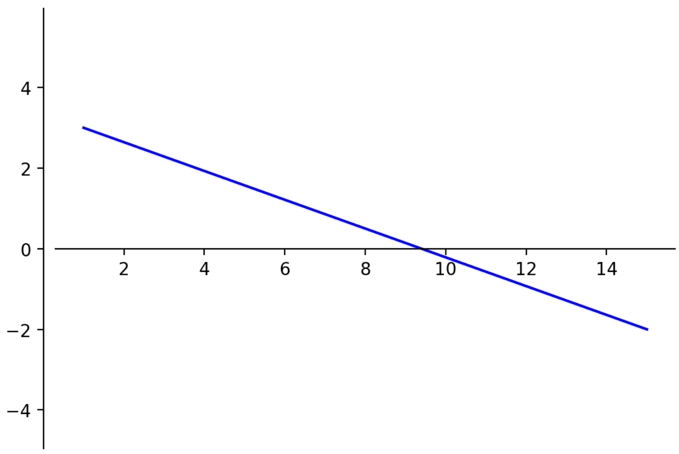
```
# Positionnement des axes de coordonnées
ax = gca()
ax.spines['right'].set_color('none')# Fait disparaître le
# bord droit du cadre.
ax.spines['top'].set_color('none')# Fait disparaître le
# bord supérieur du cadre.
ax.xaxis.set_ticks_position('bottom')# Les traits de graduation
# sur l'axe des abscisses seront en-dessous.
ax.spines['bottom'].set_position(('data',0))# Bouge verticalement le
# bord inférieur du cadre jusqu'à l'ordonnée 0.
ax.yaxis.set_ticks_position('left')# Les traits de graduation
# sur l'axe des ordonnées seront à gauche.
```

```
ax.spines['left'].set_position(('data',0))# Bouge horizontalement le bord
# gauche du cadre jusqu'à l'abscisse 0.
```

Testons donc le script obtenu après ces divers perfectionnements :

```
# from numpy import *
# from matplotlib.pyplot import *
# from pylab import *
figure()
def f(x):
    return -5/14*x+47/14
x = linspace(1,15,2)
y = f(x)
plot(x,y,color='blue')
axis('equal')
ax = gca()
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
ax.xaxis.set_ticks_position('bottom')
ax.spines['bottom'].set_position(('data',0))
ax.yaxis.set_ticks_position('left')
ax.spines['left'].set_position(('data',0))
savefig("at34gdroite2")
show()
```

En exécutant ce sript, on obtient la figure :



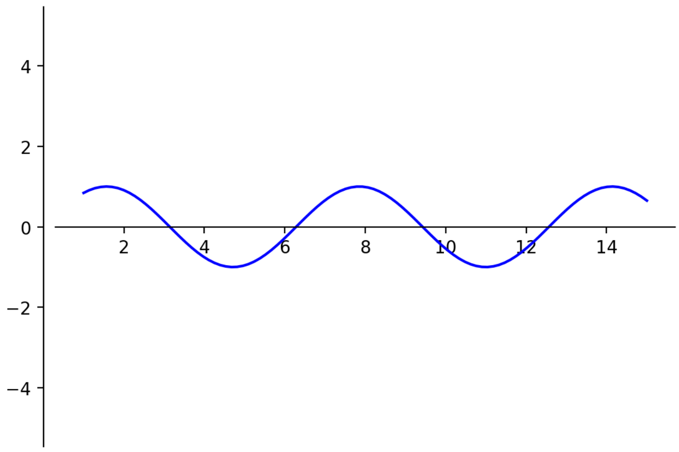
qui est nettement plus satisfaisante que la précédente.

2 - Il suffit de remplacer, dans le script précédent, la fonction `f` par `sin`[9]. Voici successivement le script utilisé et le graphe obtenu :

```
# from numpy import *
# from matplotlib.pyplot import *
# from pylab import *
from math import sin
figure()
```

9. `sinus` est une fonction du module `math`.

```
x = linspace(1,15,100)# On calcule 100 valeurs du sinus équiréparties
# entre 1 et 15.
y = sin(x)
plot(x,y,color='blue')
axis('equal')
ax = gca()
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
ax.xaxis.set_ticks_position('bottom')
ax.spines['bottom'].set_position(('data',0))
ax.yaxis.set_ticks_position('left')
ax.spines['left'].set_position(('data',0))
savefig("at34gsin")
show()
```



2.3 [*] Représentations graphiques avec Python

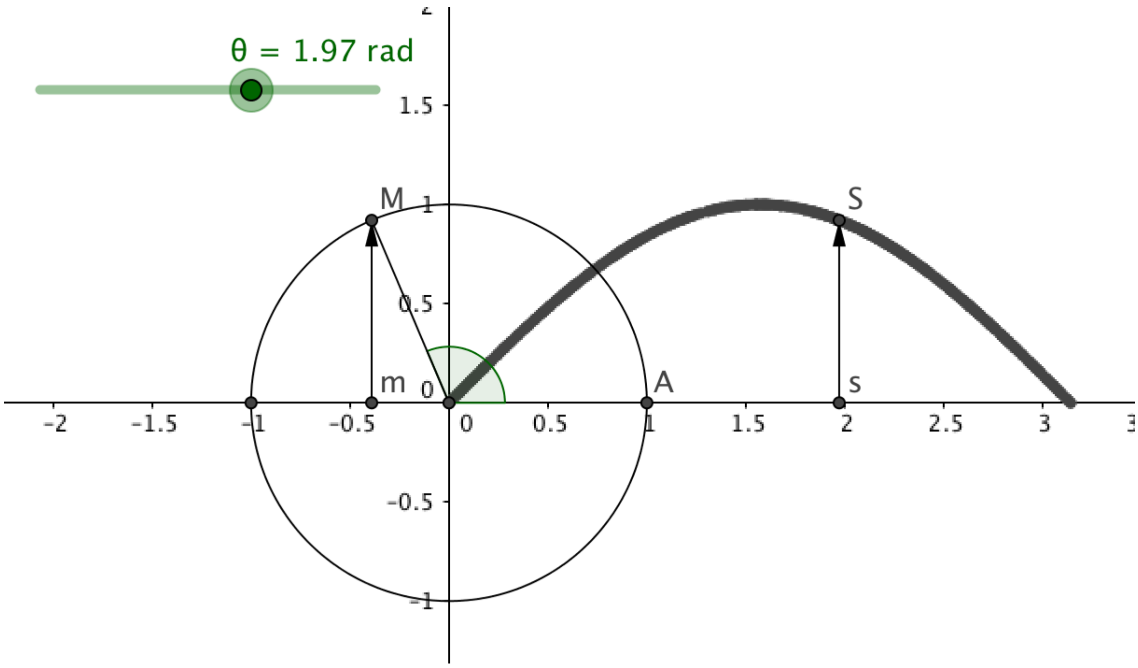
Pour tracer un graphique, Python en calcule des points consécutifs et les relie par un segment. Habituellement, on obtient ainsi une bonne approximation de la courbe à tracer quand il y a suffisamment de points car alors, nos yeux ne font plus la différence.
On se propose d'étudier des approximations du graphique de la fonction `sin()` lorsque la variable `x` varie sur l'intervalle `[0,`π`]`.
1 - Tracer l'approximation du graphique obtenue en calculant ses points d'abscisse 0, $\pi/2$ et π.
2 - Pour `n` valant successivement 3, 6, 10, 20, tracer l'approximation du graphique obtenue en calculant les points dont les abscisses sont données par la commande `linspace(0,pi,n+1)` (cette représentation graphique est une ligne polygonale à `n` côtés) .

Mots-clefs : Représentation graphique d'une fonction, repère orthonormé, commande `axis('equal')`, d'autres commandes graphiques.

Bien sûr, il est souvent impossible de représenter graphiquement une fonction parce que dans le cas courant, une représentation graphique a une infinité de points [10]. C'est pourquoi les logiciels de calcul se contentent de représenter ces graphiques par des lignes polygonales [11]. Cet exercice peut être considéré comme une suite de l'exercice 2.2.

Avant de commencer

On peut représenter simplement une sinusoïde à l'aide d'un logiciel de géométrie dynamique [12]. Le point M est repéré sur le cercle trigonométrique par l'angle de vecteurs $\theta = (\overrightarrow{OA}, \overrightarrow{OM})$. Les coordonnées de M sont $(\cos\theta, \sin\theta)$. Le point S de coordonnées $(\theta, \sin\theta)$ engendre la sinusoïde quand θ parcourt le segment $[0, \pi]$.



Solution

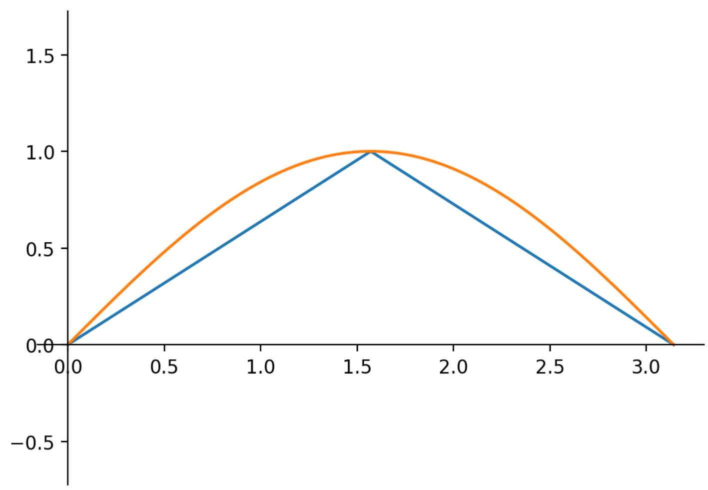
1 - On peut utiliser le script suivant :

```
from math import sin
from numpy import *
```

10. Un ordinateur ne peut exécuter une infinité de commandes.
11. dont les côtés sont eux-mêmes des successions de pixels, ce qui n'apparaîtra pas dans cet exercice
12. L'image ci-dessous a été créée à l'aide de GeoGebra.

```
from matplotlib.pyplot import *

figure()
x = array([0,pi/2,pi])# ou x = linspace(0,pi,3).
y = sin(x)
plot(x, y)
xx=linspace(0,pi,100)
yy=sin(xx)
plot(xx,yy)

axis('equal')
ax = gca()
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
ax.spines['bottom'].set_position(('data',0))
ax.spines['left'].set_position(('data',0))
# ax.xaxis.set_ticks_position('bottom')
# ax.yaxis.set_ticks_position('left')

savefig('at1s1f1')
show()
```

Ce script [13] répond à la question et produit en plus une représentation graphique acceptable [14] de `sin` entre 0 et π. C'est l'approximation la plus simple possible. On obtient :



La ligne polygonale bleue (deux côtés) ne fait pas vraiment penser à la sinusoïde. Ce n'est pas une bonne approximation de la sinusoïde, mais ce n'est pas franchement mauvais.

2 - Le script suivant définit une fonction appelée `sinpolygone(n)` qui retourne comme ci-dessus deux tracés sur la même fenêtre graphique : la représentation graphique de `sin` ainsi que la ligne polygonale

`(0,0) --> (pi/n,sin(pi/n)) --> (2pi/n,sin(2pi/n)) --> ... --> (pi,sin(pi)).`

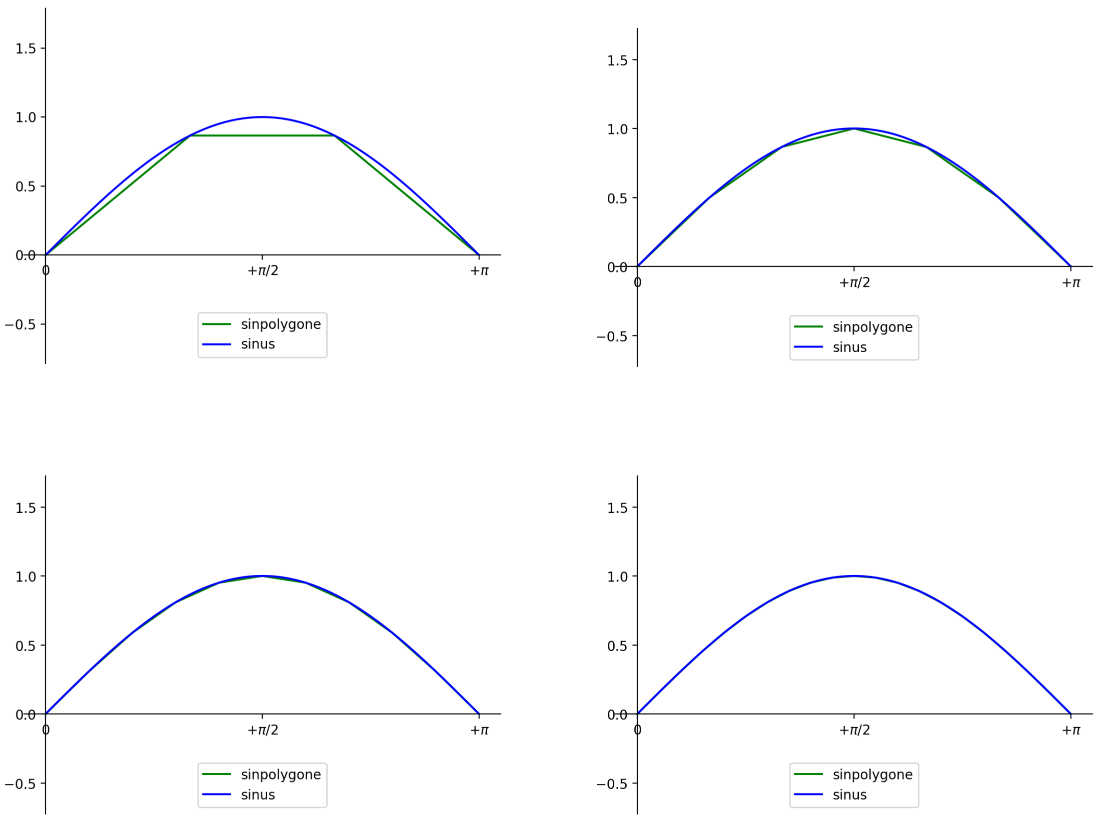
qui est censée en être une approximation.

```
from math import sin
from numpy import *
```

13. Pour tout ce qui concerne le module `matplotlib`, voir [16]. Python dispose d'autres modules graphiques.
14. On a déjà expliqué pourquoi il n'est pas possible de représenter graphiquement `sin`.

```
from matplotlib.pyplot import *

def sinpolygone(n) :# n est le nombre de côtés de la ligne polygonale.
    figure()
    x=linspace(0,pi,n+1)
    y=sin(x)
    xx=linspace(0,pi,100)
    yy=sin(xx)
    plot(x,y,color='green', label="sinpolygone"), plot(xx,yy,color='blue',
    label="sinus")
    axis('equal')
    xticks([0, pi/2, pi],[ r'$0$', r'$+\pi/2$', r'$+\pi$'])
    ax = gca()
    ax.spines['right'].set_color('none')
    ax.spines['top'].set_color('none')
    ax.spines['bottom'].set_position(('data',0))
    ax.spines['left'].set_position(('data',0))
    legend(loc='lower center')
    savefig('at1s4f1')
    show()
```

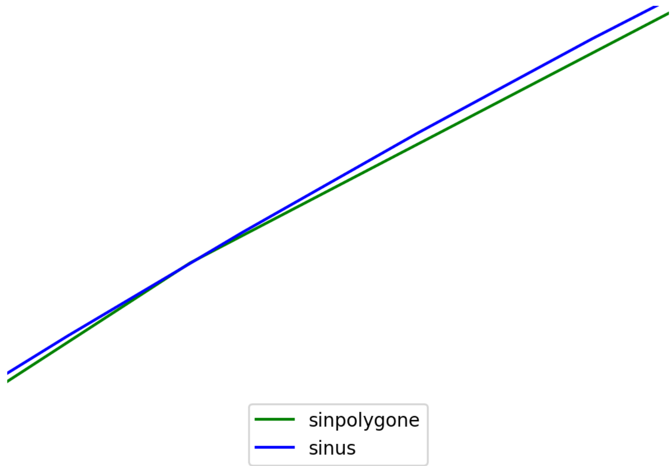
Nous avons regroupé ci-dessous en un tableau à 2 lignes et 2 colonnes l'approximation de la sinusoïde obtenue pour n = 3 (3 côtés), puis n = 6, 10, 20 (de gauche à droite et de haut en bas), en tapant `sinpolygone(3)`, *etc*, dans la console.[15]



On constate que pour `n=20` (et même pour des valeurs plus petites de `n`), on ne distingue plus la sinusoïde

15. Les tracés ont été ordonnés à l'aide de `Latex`.

de la ligne polygonale. C'est évidemment une impression fausse : voici un agrandissement d'une partie de ce dernier tracé (capture d'écran) :

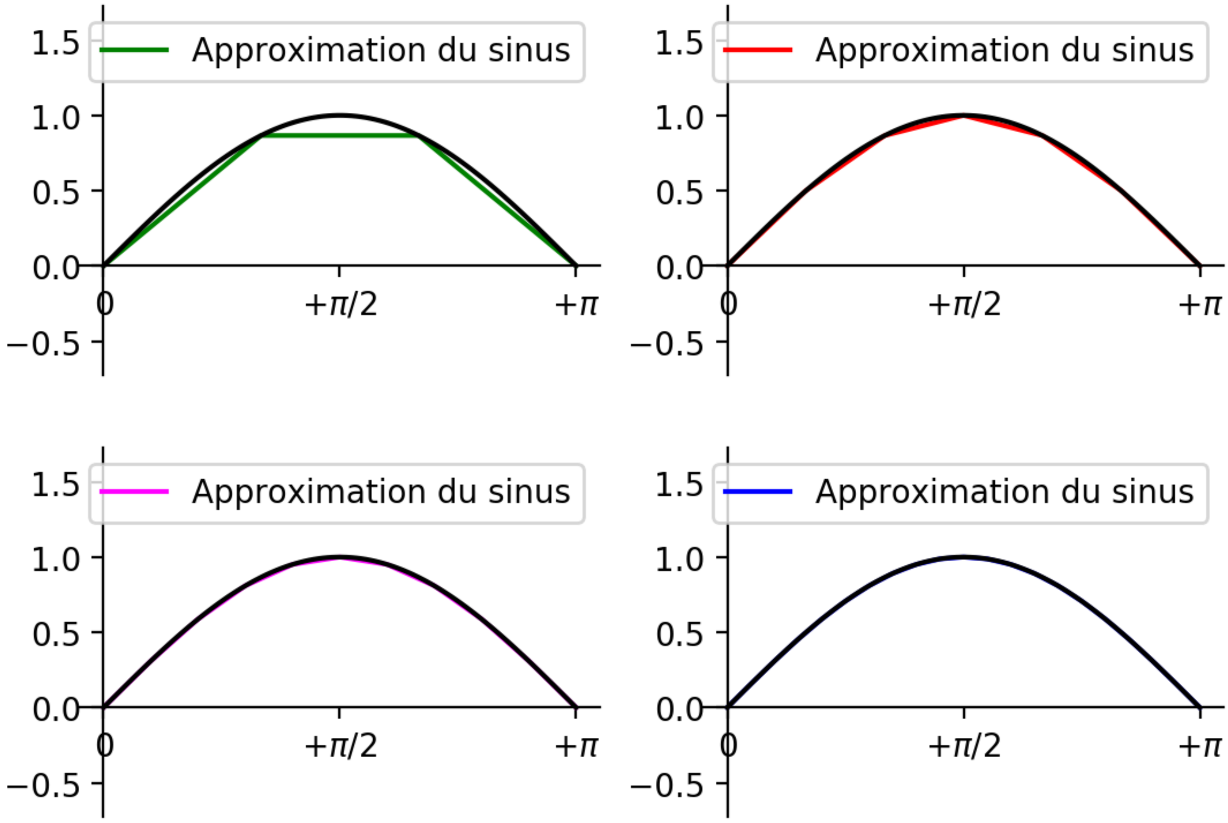


Compléments :

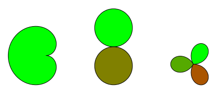
Nous aurions pu faire le tableau des 4 graphes à l'aide de la commande `subplot` de `matplotlib`, par exemple avec le script suivant :

```
from math import *
from numpy import *
from matplotlib.pyplot import *

figure()
A=array([3,6,10,20])
l=['green', 'red','magenta', 'blue']
for i in range(1,5) :
    xx=linspace(0,pi,100)
    yy=sin(xx)
    subplot(2,2,i)
    x=linspace(0,pi,A[i-1]+1)
    y=sin(x)
    axis('equal')
    plot(x,y,color=l[i-1], label="Approximation du sinus")
    plot(xx,yy,color='black')
    xticks([0, pi/2, pi],[ r'$0$', r'$+\pi/2$', r'$+\pi$'])
    ax = gca()
    # ax.xaxis.set_ticks_position('bottom')
    # ax.yaxis.set_ticks_position('left')
    ax.spines['right'].set_color('none')
    ax.spines['top'].set_color('none')
    ax.spines['bottom'].set_position(('data',0))
    ax.spines['left'].set_position(('data',0))
    legend(loc='upper right')

  show()
savefig('at1s2f1')
```

Voici le tableau de tracés obtenu en exécutant le script précédent :



Approximation du sinus
1.5
1.0
0.5
0.0
−0.5
0
+π/2
+π
Approximation du sinus
1.5
1.0
0.5
0.0
−0.5
0
+π/2
+π
Approximation du sinus
1.5
1.0
0.5
0.0
−0.5
0
+π/2
+π
Approximation du sinus
1.5
1.0
0.5
0.0
−0.5
0
+π/2
+π



2.4 [*] Programmer des fonctions : valeur absolue, volume du cône

---

1 - Programmer une fonction qui retourne la valeur absolue de tout nombre.
2 - Programmer une fonction qui retourne le volume de tout cône droit à base circulaire, de hauteur `h` cm et de rayon `r` cm.

Mots-clefs : programmer une fonction, instruction conditionnelle `if … else …`, `return`, importer une constante ou une fonction d'un module de Python.

Solution

1 - Il est difficile de trouver un exercice plus simple ! Appelons `vabsolue` cette fonction. Nous la définissons par le script :

```
# La fonction vabsolue retourne la valeur absolue de tout nombre x.
def vabsolue(x) :
    if x >= 0 :
        return x
    else  :
        return -x
```

Exemple : calculons la valeur absolue de -2.17 (nombre réel) et de -125 (nombre entier), après avoir chargé `vabsolue` dans la console [16] :

```
runfile('Chemin␣de␣la␣fonction␣vabsolue')
# Le chemin détaillé s'inscrit automatiquement quand on exécute le script.
vabsolue(-2.17)
Out[2]:  2.17
vabsolue(-125)
Out[3]:  125
```

On obtient un nombre réel dans le premier cas, un entier dans le second. Bien sûr, pour calculer une valeur absolue, on utilise habituellement la fonction pré-définie `abs(x)` (voir [10]).

2 - La formule qui donne le volume v du cône, ici en cm^3, est $v = \dfrac{\pi r^2 h}{3}\cdot$ Programmer une fonction de deux variables ne pose pas de problème :

```
# La fonction volcone retourne le volume d'un cône droit
# à base circulaire de rayon r et de hauteur h.
# Charger pi du module math.
def  volcone(r,h) :
    return  (pi*r**2*h)/3
```

Avant d'exécuter `volcone`, il faut charger la valeur de la constante π du module `math`.
Exemple :

```
from math import pi # Importation de pi.
runfile('Chemin␣de␣la␣fonction␣volcone')
volcone(1.88,3.17)
Out[3]:  11.732851629089138
pi*1.88*1.88*3.17/3 # Vérification (inutile) par calcul direct.
```

16. Nous supprimons systématiquement les lignes blanches pour gagner de la place.

```
Out[6]:  11.732851629089138
```

On aurait pu importer le module `math` tout entier par `from math import *`.

2.5 [*] Trinôme du second degré

Résoudre l'équation du second degré (E) : $ax^2 + bx + c = 0$ ($a \neq 0$)

Mots-clefs : Fonction `sqrt`, à importer du module `math`, sortie d'une fonction par `return` ou `print`, instruction conditionnelle `if: …elif:…else:….`

Solution

La programmation de la solution doit envisager les 3 cas : $b^2 - 4ac > 0$, $b^2 - 4ac = 0$ et $b^2 - 4ac < 0$, d'où l'utilisation de l'instruction conditionnelle `if: …elif:…else:….` Les scripts ci-dessous permettront aux professeurs de proposer des trinômes moins banals que les trinômes habituels à coefficients entiers dont la résolution est plus ou moins évidente. Les deux scripts se distinguent par leurs sorties : la fonction `trinome` (instruction `return`) retourne une `liste`, éventuellement vide[17], que l'on peut utiliser dans des calculs ultérieurs ; la seconde `trinomebis` (instruction `print`) donne les solutions de (E), s'il y en a, au moyen d'une phrase en français ; mais il ne semble pas qu'on puisse récupérer ces solutions pour des calculs ultérieurs. La sortie de `trinomebis` est de type `nonetype`.

```
# La fonction trinome ci-dessous retourne la liste des solutions de
# l'équation (E) : ax^2+bx+c=0 (où a est un réel non nul).
# L'utilisation de la fonction racine carrée impose qu'elle soit
# importée du module math.
from math import sqrt
def trinome(a,b,c):
    delta=b**2-4*a*c
    if delta > 0:
        x1 = (-b+sqrt(delta))/(2*a)
        x2 = (-b-sqrt(delta))/(2*a)
        return [x1,x2]
    elif delta == 0:
        x12 = -b/(2*a)
        return [x12]
    else :
        return []
```

Voici quelques exemples d'application. On a ajouté le type de la solution et le calcul de la somme des carrés des racines. Les commentaires n'ont pas été retournés par Python. Ils ont été ajoutés lors de la rédaction.

```
runfile('Chemin de la fonction trinome')
L=trinome(1,2*(2+sqrt(3)),7+4*sqrt(3))# Exemple n°1.
L
Out[3]: [-3.732050807568877]# La solution unique de (E).
type(L)
Out[4]: list
L[0]**2# Calcul de la somme des carrés des solutions.
Out[5]: 13.928203230275509# La somme des carrés des solutions de (E).
L = trinome(-10,2*(2+sqrt(3)),7+4*sqrt(3))# Exemple n°2.
L
Out[7]: [-0.8645761419679951, 1.6109863034817706]# Les 2 solutions de (E).
type(L)
Out[8]: list
```

17. Il s'agit, suivant le cas, de 0 (liste vide), 1 ou 2 nombres séparés par une virgule et placés entre crochets

```
L[0]**2+L[1]**2# Calcul de la somme des carrés des solutions.
Out[9]: 3.3427687752661224# La somme des carrés des solutions de (E).
L = trinome(10,2*(2+sqrt(3)),7+4*sqrt(3))# Exemple n°3.
L
Out[11]: []# (E) n'a pas de solution.
type(L)
Out[12]: list
```

Remplaçons `return` par `print` :

```
# La fonction trinome ci-dessous retourne les solutions de
# l'équation (E) : ax^2+bx+c=0 (où a est un réel non nul).
# L'utilisation de la fonction racine carrée impose qu'elle soit
# importée du module math.
from math import sqrt
def trinomebis(a,b,c):
    delta=b**2-4*a*c
    if delta > 0:
        x1 = (-b+sqrt(delta))/(2*a)
        x2 = (-b-sqrt(delta))/(2*a)
        print('(E)␣a␣deux␣solutions␣',x1,'␣et␣',x2,'.')
    elif delta == 0:
        x12 = -b/(2*a)
        print('(E)␣a␣une␣solution␣:␣',x12,'.')
    else:
        print("(E)␣n'a␣␣pas␣de␣solution.")
```

Voici un exemple de chaque cas :

```
runfile('Chemin␣de␣la␣fonction␣trinomebis')
L = trinomebis(1,0,-2)
(E) a deux solutions  1.4142135623730951  et  -1.4142135623730951 .
type(L)
Out[3]: NoneType
L = trinomebis(1,2*(2+sqrt(3)),7+4*sqrt(3))
(E) a une solution :  -3.732050807568877 .
type(L)
Out[5]: NoneType
L = trinomebis(1,2,2)
(E) n'a␣␣pas␣de␣solution.
type(L)
Out[7]:␣NoneType
```

2.6 [**] Minimum de 2, 3, 4 nombres (instructions conditionnelles)

1.a - Programmer une fonction qui retourne le minimum de 2 nombres.
1.b - Programmer une fonction qui retourne le minimum et le maximum de 2 nombres.
2 - Programmer une fonction qui retourne le minimum de 3 nombres.
3 - Programmer une fonction qui retourne le minimum de 4 nombres.

Mots-clefs : Programmation de fonctions, instruction conditionnelle `if …else …`, `return`, négation d'une proposition logique, importation de constantes et de fonctions pré-définies.

Solution

Quand on veut calculer le minimum d'une liste de nombres, on utilise bien évidemment la fonction pré-définie `min` de Python, voir [10]. Ici, notre but est de calculer le minimum de 2, 3 ou 4 nombres sans utiliser cette fonction.

1.a - Cette question est élémentaire :

```
# La fonction min2nom retourne le minimum de 2 nombres a et b.
def min2nom(a,b) :
    if a <= b:
        return a
    else :
        return b
```

Application :

```
runfile('Chemin de la fonction min2nom')
min2nom(-7.17,-7.28)
Out[5] :  -7.28
```

2.b - Ce n'est guère plus compliqué :

```
# minmax2nom retourne le minimum et le maximum de 2 nombres a et b.
def minmax2nom(a,b) :
    if a <= b:
        return [a,b]
    else :
        return [b,a]
```

Remarques :
a - On a choisi ici de faire apparaître le résultat sous forme d'une `liste` de 2 nombres.
b - Si `L` est une liste de nombres, `max(L) = -min(-L)`. Savoir calculer un minimum suffit.
Application :

```
runfile('Chemin de la fonction minmax2nom')
minmax2nom(1.7,-3.84)
Out[7] :  [-3.84,  1.7]
```

Si on remplace [a,b] et [b,a] après la commande `return` du script par a,b et b,a, le résultat apparaîtra sous forme de tuple. Dans l'exemple ci-dessus, on obtiendra (-3.84, 1.7) [18].

18. On peut sans dommage escamoter les notions de `liste` et de `tuple`.

3 - Ça se complique un peu car nous utilisons ci-dessous des instructions conditionnelles emboîtées. C'est aussi plus intéressant. Par exemple, on utilise

$$(a \geqslant b) \text{ ou } (a \geqslant c) \quad \Longleftrightarrow \quad \neg\,(a < b \text{ et } a < c)$$

(la négation de "l'un ou l'autre", c'est "ni l'un, ni l'autre").

```
# La fonction min3nom retourne le minimum de 3 nombres a, b et c.
def min3nom(a,b,c):
    if (a > b) or (a > c):
        if b <= c:
            return b
        else:
            return c
    else:
            return a
```

On fait très attention, bien sûr, aux indentations.
Application :

```
runfile('Chemin de la fonction min3nom')
from math import pi
min3nom(-2.1,3.2,-pi)
Out[10]: -3.141592653589793
```

Remarques :
On a dû importer la valeur de `pi` du module `math`.

4 - Le problème paraît plus difficile car il y aura plus d'instructions conditionnelles emboîtées ainsi que des difficultés logiques. Voici une solution commentée :

```
# min4nombis retourne le minimum de 4 nombres a, b, c et d.
def min4nombis(a,b,c,d):
    if (a >= b) or (a >= c) or (a >= d):
        # Le minimum est égal à b ou c ou d.
        if (b >= c) or (b >= d):
            # Le minimum est égal à c ou d.
            if c >= d :
                return d
            else:
                return c
        else :
            return b
    else:
        return a
```

Cela repose sur l'équivalence logique :

$$(a \geqslant b) \text{ ou } (a \geqslant c) \text{ ou } (a \geqslant d) \quad \Longleftrightarrow \quad \neg\,(a < b \text{ et } a < c \text{ et } a < d)$$

Application : dans l'exemple ci-dessous, nous devons importer la fonction `cos` et la constante `pi` du module `math`.

```
runfile('Chemin de la fonction min4nombis')
from math import pi, cos# Importation de la constante pi et de la fonction cos.
min4nombis(3.14,-2.71,1.4142,-cos(pi/6))
Out[3]: -2.71
```

C'est plus simple si l'on se permet d'utiliser la fonction `min3nom`[19], qu'il faudra alors importer[20] :

```
# min4nom retourne le minimum de 4 nombres a, b, c et d.
# Elle utilise la fonction min3nom.
def min4nom(a,b,c,d):
    if a >= b:
        return min3nom(b,c,d)
    else:
        return min3nom(a,c,d)
```

Application : pour reprendre l'exemple ci-dessus, importons donc, outre la fonction `min3nom`, `cos` et `pi` du module `math`.

```
runfile('Chemin de la fonction min4nom')
from min3nom import *
from math import cos,pi
min4nom(3.14,-2.71,1.4142,-cos(pi/6))
Out[4]: -2.71
```

19. De même, on aurait simplifié le script définissant `min3nom` en utilisant la fonction `min2nom`.
20. si on ne l'a pas définie ou utilisée au cours de la même session

2.7 [***] Alignement de 3 points

On se donne, à l'aide de leurs coordonnées, 3 points distincts A, B et C d'un plan rapporté à un repère orthonormé. On appelle d_1, d_2 et d_3 les distances de B à C, de A à C et de A à B. On peut supposer que $d_1 \leqslant d_2 \leqslant d_3$.
1 - Démontrer que les points sont alignés si et seulement si $d_3 = d_1 + d_2$.
2 - Sans utiliser la fonction racine carrée, programmer un algorithme qui retourne, à partir de la liste des coordonnées de A, B et C, "Les points sont alignés" ou "Les points ne sont pas alignés", suivant le cas.

Mots-clefs : saisie de données, affectation de variables, additions et multiplications de nombres, problèmes équivalents. Dans les commentaires sur le test de l'égalité de 2 nombres : équation d'une droite.

On peut supposer que $d_1 \leqslant d_2 \leqslant d_3$ parce que l'on peut échanger, si nécessaire, les lettres A, B et C. L'échange éventuel de ces lettres est effectué ci-dessous par la commande `sorted([d1ca,d2ca,d3ca])`.

Solution

Dans cet exercice, les calculs sont faciles. Les difficultés sont ailleurs.

1 - On peut s'appuyer sur un peu de géométrie : 2 cercles dont les centres sont distants de d et de rayons r_1 et r_2 plus petits que d sont tangents si et seulement si $d = r_1 + r_2$.

2 - On calcule facilement le carré d'une distance à l'aide d'additions et de multiplications tandis que le calcul d'une distance n'est pas élémentaire, voir les commentaires, plus loin.

On sait bien que $a = b$ n'est pas équivalent à $a^2 = b^2$. Pourtant, au moyen de 2 élévations au carré, dans le cas particulier de l'exercice présent, $d_1 + d_2 = d_3$ équivaut à $4d_1^2 d_2^2 = (d_3^2 - d_1^2 - d_2^2)^2$. Démontrer cette équivalence est en soi un exercice intéressant. On en déduit une solution de notre problème sous forme de script :

```
ent=eval(input('Liste des coordonnées de A, B et C :'))
d3ca = (ent[2]-ent[0])**2+(ent[3]-ent[1])**2
# d3ca est le carré de la distance de A à B.
d2ca = (ent[0]-ent[4])**2+(ent[1]-ent[5])**2
# d2ca est le carré de la distance de C à A.
d1ca = (ent[4]-ent[2])**2+(ent[5]-ent[3])**2
# d1ca est le carré de la distance de B à C.
L=sorted([d1ca,d2ca,d3ca])
if 4*L[0]*L[1]==(L[2]-L[0]-L[1])**2 :
    print('les points A, B et C sont alignés')
else :
    print('les points A, B et C ne sont pas alignés')
```

Par exemple, exécutons ce script lorsque la liste des entrées est égale à `[-1,3,2,0,4,-2]`, puis à `[-4.74, 34.46, 0.17, 6.58, 5, -5.32]`. On obtient :

```
Liste des coordonnées de A, B et C :[-1,3,2,0,4,-2]
les points A, B et C sont alignés
```

```
Liste des coordonnées de A, B et C :[-4.74, 34.46, 0.17, 6.58, 5, -5.32]
les points A, B et C ne sont pas alignés
```

Le premier essai est évident car B et C se trouvent manifestement sur la droite de pente `-1` qui passe par `A`. Pour le deuxième cas, il pourrait être utile de faire un graphe.

Fin de la solution

Commentaires

1 - Calculer une distance impose d'utiliser la fonction racine carrée qui appartient au module `math`. Il faudra l'importer. Voici ce qui arrive quand on oublie l'importation :

```
sqrt(2)
Traceback (most recent call last):
  File "<ipython-input-9-40e415486bd6>", line 1, in <module>
    sqrt(2)
NameError: name 'sqrt' is not defined
```

La fonction `sqrt` est inconnue ! Quand on importe le module en question, ça marche :

```
from math import *# ou from math import sqrt
sqrt(2)
Out[2]: 1.4142135623730951
```

2 - Faisons un autre essai du script ci-dessus :

```
Liste des coordonnées de A, B et C :[2.4, 0.197, -1.16, -0.2658, 7.7, 0.886]
les points A, B et C ne sont pas alignés
```

Ce résultat est faux. Il est en effet facile de vérifier à la main que les points A, B et C se trouvent sur la droite d'équation `y=0.13*x-0.115`. Vérifions avec Python :

```
0.13*2.4-0.115==0.197
Out[1]: True
0.13*(-1.16)-0.115==-0.2658
Out[4]: True
0.13*7.7-0.115==0.886
Out[2]: False
0.13*7.7-0.115
Out[3]: 0.8860000000000001
```

On constate que pour Python, `C` n'est pas sur la droite `(AB)`. Cela provient du fait que Python calcule mal la quantité `0.13*7.7-0.115`. Il ne trouve pas 0.886 mais 0.8860000000000001, ce qui est faux. Ce problème est imparable[21]. Habituellement, on se contente d'une quasi-égalité : on considère que deux nombres x et y sont égaux si $|x-y| \leqslant \varepsilon = 2.220446049250313 * 10^{-16}$.

21. Voir [1], chapitre 4, 1 : « Les nombres en notation scientifique ».

2.8 [**] Affectation de données

Soit f une fonction définie sur $\mathbb{R}$ par $f(x) = x^2 - 5x + 3$ et k un nombre réel. On souhaite déterminer tous les entiers n compris entre deux entiers donnés a et b ($a < b$) tels que $f(n) < k$.

1 - -10 est-il solution de $f(n) < 5$? Même question pour 0.

2 - Écrire un script Python appelant un nombre et affichant "oui" s'il est solution de $f(n) < 5$, "non" sinon.

3 - Modifier le script pour appeler k et afficher successivement tous les entiers solutions de $f(n) < k$ sur l'intervalle $[-10, 10]$.

4 - Modifier le script pour demander aussi a et b.

Mots-clefs : Affectation de données (commandes **input** et **eval**), instruction conditionnelle `if` … `then` … `else` …, boucle `pour`, instruction `print`, itérateur `range`. On admet que `for n in range(a,b+1) :` où `a` et `b` sont des entiers tels que `a < b` donne successivement à `n` les valeurs `a`, `a+1`, …, `b`.

Solution

1 - Directement dans la console :

```
(-10)**2-5*(-10)+3  <  5
Out[1]:  False
```

Ceci permet d'introduire les conditions de type booléen et les variables booléennes [22]. Ici, la variable booléenne prend la valeur `False`, ce qui signifie que l'inégalité `f(-10) < 5` est fausse.

2 - Le script ci-dessous convient :

```
n=eval(input('la valeur de n est :'))
if n**2 - 5*n + 3 < 5:
    print('oui')
else:
    print('non')
```

Appliquons ce script [23] pour `n = -4` après l'avoir chargé dans la console de Python :

```
la valeur de n est : -4
non
```

Remarque : La commande `n=input('la valeur de n est :')` introduirait `n` comme une chaîne de caractères et non un nombre entier. Il faut l'«évaluer» à l'aide de la commande `eval`.

3 - Cette question est une répétition fastidieuse de la précédente. Les répétitions seront assurées par une boucle `pour`.

```
k=eval(input('la valeur de k est :'))
for n in range(-10,11):
    if n**2 - 5*n + 3 < k:
        print(n)
```

Faisons l'essai pour `k = 17`. Ce qu'on obtient n'est pas très joli :

22. Voir [1], chapitre 1, 7 La structure conditionnelle.
23. qui a été écrit dans l'interpréteur

```
la valeur de k est :17
-1
0
1
2
3
4
5
6
```

4 - On a maintenant un problème à 3 paramètres : k (réel), a, b (entiers tels que $a < b$).

```
k=eval(input('la valeur de k est :'))
a=eval(input('la valeur de a est :'))# par hypothèse, a est un entier.
b=eval(input('la valeur de b est :'))# par hypothèse, b est un entier.
for n in range(a,b+1):# On cherche les solutions dans [a,b].
    if n**2 - 5*n + 3 < k:
        print(n)
```

L'essai ci-dessous est fait pour `k = -2.2`, `a = -7` et `b = 22`.

```
la valeur de k est :-2.2
la valeur de a est :-7
la valeur de b est :22
2
3
```

Fin de la solution

Remarque et compléments :

Ce qui suit ne convient pas à des débutants en Python.

1 - On pourrait aussi traiter cet exercice à l'aide du signe du trinôme $x^2 - 5x + 3 - k$.

2 - On aurait pu rédiger cette solution à l'aide d'une fonction-Python, voir la définition dans le script ci-dessous aux lignes 6 et 7.

3 - On peut illustrer l'exercice précédent à l'aide d'une représentation graphique :

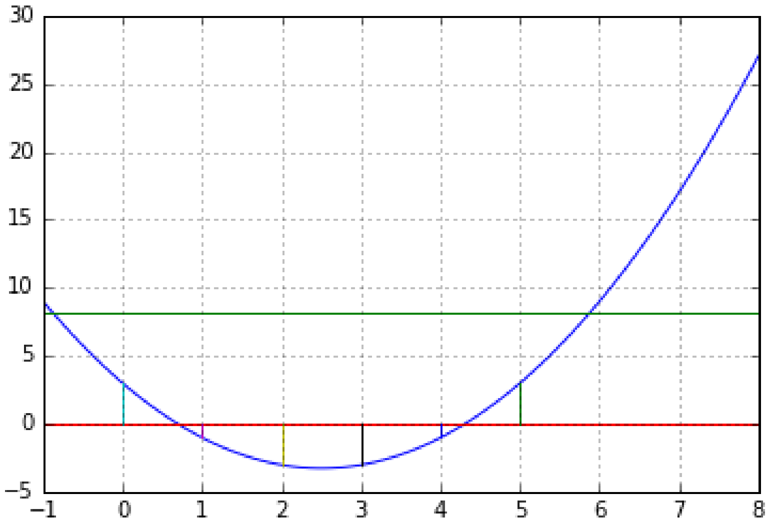
```
from numpy import *
from matplotlib.pyplot import*
k=eval(input('La valeur de k est :'))#k est un nombre réel.
a=eval(input('La valeur de a est :'))# a est un entier.
b=eval(input('La valeur de b est :'))# b est un entier > a.
def f(x):
    return x**2 - 5*x + 3
abs=linspace(a,b,100)# [a,b] est partagé par 99 points équidistants
# en 100 intervalles égaux.
ord=f(abs)# f est vectorisée.
plot(abs,ord)# Graphe de f entre a et b.
plot([a,b],[k,k])# Graphe de y=k entre a et b.
plot([a,b],[0,0])# Graphe de y=0 entre a et b.
c=arange(a,b+1)# c est la matrice [a, a+1, ..., b].
d=f(c)# f est vectorisée.
masque=(d<k)
y=d[masque]
```

```
x=c[masque]
for i in x:
    plot([i,i],[0,f(i)])# Graphe de x=i entre 0 et f(i).

grid(True)
savefig('sec2fig5.png')# Production d'une image au format png.
show()# On affiche tous les graphes simultanément.
```

Application :

```
la valeur de k est :8.21

la valeur de a est :-1

la valeur de b est :8
```

Voici le graphe obtenu (image png produite par Python) :



2.9 [*] Algorithme d'Euclide.

1 - Programmer une fonction qui, étant donnés deux entiers positifs `a` et `b`, retourne leur PGCD [a] calculé selon l'algorithme d'Euclide.
2 - En déduire le calcul de leur PPCM [b].

a. plus grand diviseur commun
b. plus petit multiple commun

Mots-clefs : reste et quotient de la division euclidienne (`%` et `//`), algorithme d'Euclide, boucle `tant que`, importation de fonctions, module `math`.

Solution [24]

1 - La justification de l'algorithme d'Euclide est bien connue : si `r` désigne le reste de la division euclidienne de `a` par `b` [25], l'ensemble des diviseurs communs de `a` et `b` est le même que l'ensemble des diviseurs communs de `b` et `r`. Comme `r < b`, on tombe sur un problème plus simple si, au lieu de chercher l'ensemble des diviseurs communs de `a, b`, on cherche l'ensemble des diviseurs communs de `b, r`. Il est même possible que `r = 0`. Sinon, on recommence. On est sûr d'arriver à 0. Or 0 étant divisible par tout entier positif, l'ensemble des diviseurs communs de `n)`, entier positif quelconque et 0 est l'ensemble des diviseurs de `n`, le plus grand étant évidemment `n` lui-même. L'algorithme ci-dessous se trouve ainsi justifié.

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

Applications [26] :

```
from euclide import *
euclide(177,353)
Out[2] : 1
euclide(123456789,987654321)
Out[3] : 9
```

Les calculs sont pratiquement instantanés : l'algorithme d'Euclide est très efficace. Bien sûr, Python fournit une fonction primitive qui fait la même chose : la fonction `gcd()` [27] :

```
from math import gcd
math.gcd(177,353)
Out[5] : 1
math.gcd(123456789,987654321)
Out[6] : 9
```

2 - Le calcul du PPCM de deux entiers positifs se déduit du calcul de leur PGCD parce que le produit du PPCM par le PGCD est égal au produit des deux nombres. En continuant le calcul ci-dessus, on obtient :

24. Le calcul du PGCD de `p` entiers positifs est repris à l'exercice ??.
25. Plus généralement, dans l'algorithme d'Euclide, `a` peut être un entier de signe quelconque, `b` un entier de signe quelconque non nul.
26. On remarquera sur le listing qui suit que la fonction `euclide()` a été chargée dans la console en l'appelant exactement comme un module (créé par nous). Autre possibilité : si on vient de la rédiger dans l'interpréteur, on peut aussi la charger en l'exécutant.
27. Le code source de la fonction `gcd()` du module `math`, voir [11] est exactement celui de la fonction `euclide()`.

```
print(177*353//1)
62481
print(123456789*987654321//9)
13548070123626141
```

Les PPCM recherchés sont respectivement égaux à 62481 et 13548070123626141.

Chapitre 3

Listes

Liste des exercices :

Énoncé nº 3.1 [*] : Simuler `n` lancers d'un dé équilibré, sous forme de liste

Énoncé nº 3.2 [*] : Maximum d'une liste de nombres

Énoncé nº 3.3 [*] : Ranger une liste de nombres

Ce chapitre contient des listes et des boucles `pour`.

Une liste (voir [1], pp. 20-27) est une séquence d'éléments rangés dans un certain ordre. Ces éléments peuvent être de types différents. Nous ne considèrerons dans ce chapitre que des listes de nombres. On dispose, pour manipuler des listes, d'un certain nombre d'instruments appelés `méthodes` (voir [1], p. 27 et les exercices qui suivent). On pourrait imaginer des calculs sur des listes, par exemple additionner ou multiplier deux listes de nombres de même longueur. Ce serait sans grand intérêt : les listes ne sont pas faites pour ça.

Quand on veut tester un algorithme qui calcule le maximum d'une liste de nombres, on a besoin de listes de nombres (pour les tests). Si on les fabrique à la main, elles seront courtes, forcément. L'intérêt d'un logiciel de calcul n'apparaîtra pas [1]. Si l'on veut disposer d'une liste de 1 000 000 termes afin de tester un algorithme, on engendre habituellement un million de nombres aléatoires. On n'a pas besoin de savoir comment ces listes sont fabriquées. Dans ce chapitre, nous utiliserons systématiquement les suites engendrées par les deux fonctions `reels()` et `entiers()` suivantes :

```
from random import gauss
from copy import deepcopy
def reels(n):
    L=[]# Liste vide.
    for i in range(n):
        L.append(gauss(7,10))# gauss est une fonction du module random.
    Lo=deepcopy(L)# deepcopy  est une fonction du module copy.
    return[L,Lo]
# Lo est une copie indépendante de L. Si on modifie l'une de ces listes,
# l'autre ne change pas.
# reels() retourne une liste composée de 2 éléments qui sont des listes.
```

et

```
from random import randint
from copy import deepcopy
```

1. Pourquoi calculer le maximum d'une liste de 10 nombres, que l'on voit au premier coup d'œil ?

```
def entiers(n):
    L=[]# Liste vide.
    for i in range(n):
        L.append(randint(-50,100))# randint est une fonction du module random.
    Lo = deepcopy(L)# deepcopy est une fonction du module copy.
    return[L,Lo]
# Lo est une copie indépendante de L. Si on modifie l'une de ces listes,
# l'autre ne change pas.
# entiers() retourne une liste composée de 2 éléments qui sont des listes.
```

qui retournent un 2-uplets (L, L_0) formé d'une liste `L` de `n` nombres réels ainsi qu'une copie `Lo` de `L` qui est souvent utile parce que si `L` est modifiée au cours de l'exécution d'un script, `Lo` aura conservé l'état initial de `L`.

Exemple : Engendrons une suite `A` de 10 entiers et une copie indépendante `B`.

```
import entiers as en
[A,B]=en.entiers(10)
A
Out[3]: [-2, -23, -2, -5, 69, 32, 32, 21, -21, -47]
B
Out[4]: [-2, -23, -2, -5, 69, 32, 32, 21, -21, -47]
```

ou

```
from entiers import *
[A,B] = entiers(10)
A
Out[3]: [40, -43, 98, -3, 56, 75, -2, 19, -43, 57]
B
Out[4]: [40, -43, 98, -3, 56, 75, -2, 19, -43, 57]
```

3.1 [*] Simuler `n` lancers d'un dé équilibré, sous forme de liste

---

> En utilisant la fonction `randint()` du module `random`, simuler, sous forme de liste, `n` lancers d'un dé équilibré.

Mots-clefs : liste, module `random`, fonction `randint()`, méthode `append()` de l'objet liste, boucle `pour`.

Solution

`randint(a,b)` où `a` et `b`, $a \leqslant b$ sont des entiers, retourne un nombre entier tiré au hasard dans l'intervalle `[a,b]`, extrémités comprises. Par exemple, `randint(1,6)` retourne un nombre tiré au hasard dans l'ensemble $\{1, \ldots, 6\}$ et par conséquent `randint(1,6)` simule le lancer d'un dé. Cela fournit le premier terme de la liste [2] recherchée. À l'aide d'une boucle `pour`, on ajoutera à cette liste les tirages suivants comme dans le script ci-dessous, à l'aide de la commande `append()` [3], [4]. La fonction `lancersDe` est une solution du problème posé.

```python
from random import randint
def lancersDe(n) :
    L=[]# Liste vide.
    for i in range(n) :# Pour engendrer une liste de n termes.
        L.append(randint(1,6))# randint est une fonction du module random.
    return(L)
# lancerDe(n) retourne une simulation d'une suite de n lancers d'un dé.
```

Exemple : simuler 15 lancers successifs d'un dé.

```python
runfile('Chemin de la fonction lancersDe')
lancersDe(15)
Out[2] : [4, 4, 6, 4, 2, 6, 6, 3, 1, 5, 4, 4, 2, 1, 5]
```

---

2. Sur la notion de `liste` dans Python, voir [1], chapitre 1, section 9.
3. `append()` est une méthode de l'objet liste. Les méthodes usuelles des listes sont décrites dans [1], déjà cité.
4. Par exemple, directement dans la console :

L = [1,2,3,4]
L.append(5)
L
Out[11] : [1, 2, 3, 4, 5]

3.2 [*] Maximum d'une liste de nombres

---

1 - Calculer le maximum `M` d'une liste `L` de nombres, de longueur `l` comme suit :

(a) On pose `M=L[0]`.
(b) Ensuite, si `l` $\geqslant$ 2, `i` prenant successivement les valeurs 1, ..., `l-1`, on pose `M=L[i]` si `L[i] > M`.
On présentera la solution sous la forme d'une fonction.

2 - Application : calculer le maximum d'une liste de 10, puis de 10 000 nombres réels engendrée par la fonction `reels()`, cf.(3).

3 - Calculer le minimum `m` de `L` en utilisant le calcul du maximum programmé ci-dessus.

Mots-clefs : boucle `pour`, instruction conditionnelle `if`, appeler une fonction nouvellement programmée, méthode `list.append()` des listes, commande `len()` [5], primitives `min()` et `max()`.

Solution

1 - Nous proposons la fonction `atlas()` suivante, conforme à l'énoncé :

```
def atlas(L) :# L est une liste de nombres (au moins 1).
    for i in range(len(L)) :# len(L) : longueur de L (notée l dans l'énoncé).
        if L[i]>L[0] :
            L[0] = L[i]
    return L[0]
```

Remarquons que lors d'une exécution d'`atlas()`, seul le premier terme de `L` est changé et ceci seulement dans le cas où il n'en est pas le maximum. La fonction `atlas()` retourne un nombre de type `float` [6], qui est le type des nombres produits par la fonction `reels()`.

2 - On aura besoin de charger dans la console les fonctions `reels()` et `atlas()`, ce qui se fait via les commandes suivantes :

```
from atlas import *
from reels import *
```

Appliquons-les pour calculer le maximum d'une liste de 10 réels fournie par la fonction `reels()` :

```
from reels import *
from atlas import *
[L,L0] = reels(10)
L
Out[4] :
[10.531352495090177,
 12.825675783103737,
 10.077230201543921,
 6.782970458445565,
 1.9166906476453551,
 15.849889158355168,
 -0.7293027684673463,
 11.407060614760322,
 -3.7720041187274074,
```

5. `len(L)` retourne la longueur de la liste `L`.
6. nombre réel à virgule flottante

```
 6.436637330311071]
L0
Out[5] :
[10.531352495090177,
 12.825675783103737,
 10.077230201543921,
 6.782970458445565,
 1.9166906476453551,
 15.849889158355168,
 -0.7293027684673463,
 11.407060614760322,
 -3.7720041187274074,
 6.436637330311071]
atlas(L)
Out[6] :  15.849889158355168
L
Out[7] :
[15.849889158355168,
 12.825675783103737,
 10.077230201543921,
 6.782970458445565,
 1.9166906476453551,
 15.849889158355168,
 -0.7293027684673463,
 11.407060614760322,
 -3.7720041187274074,
 6.436637330311071]
```

On constate bien qu'au cours de l'exécution de la commande `atlas(L)`, seule la première valeur de la liste `L` a changé. Calculons de même le maximum de 10 000 nombres :

```
[L,L0] = reels(10000)
atlas(L)
Out[9] :  47.553319964137636
```

Les calculs sont à peu près instantanés. Il est facile de vérifier ces résultats[7] qui sont donnés directement par la primitive `max()`[8] :

```
max(L0)
Out[9] :  47.553319964137636
```

3.1 - Normalement, on calcule le minimum de `L` à l'aide de la primitive `min()` et de l'instruction `min(L)`.
3.2 - Une autre solution serait de programmer une fonction analogue de `atlas()` qui donnerait le minimum au lieu de donner le maximum (il suffit de remplacer `>` par `<` dans le script définissant `atlas()`).
3.3 - On peut déduire le calcul du minimum d'une liste de nombres du calcul du maximum. En effet, le minimum de `n` nombres est l'opposé du maximum des opposés de ces nombres. C'est compliqué parce que `L` étant une liste de nombres, `- L` n'a pas de sens pour Python. Il faut changer les signes des éléments de `L` un par un à l'aide d'une boucle `pour`. Dans le script ci-dessous, la liste obtenue est notée `Lm` :

```
def hera(L) :
    Lm = []
```

7. qui n'ont pas besoin d'être vérifiés
8. Elle est disponible sans qu'il soit besoin de charger un module.

```
    for i in range(len(L)) :
        Lm.append(-L[i])
    return Lm
```

Remarquons que dans le script de la fonction `hera`, on peut remplacer la commande `Lm.append(-L[i])` par la commande `Lm = Lm + [-L[i]]` [9] ou par la commande `Lm += [-L[i]]`.

Application : calculons le minimum d'une liste `L` produite par la commande `[L,L0] = reels(100000)` [10] :

```
from hera import *
[L,L0] = reels(100 000)
Lm = hera(L)
-atlas(Lm)
Out[13] : -35.7044541020379
min(L)
Out[14] : -35.7044541020379
```

Cette solution est. évidemment mauvaise.

9. Concaténation de listes.
10. Les fonctions `atlas()` et `reels()` ont déjà été importées.

3.3 [*] Ranger une liste de nombres

Une liste de nombres L est donnée.
1 - À l'aide de la primitive `min()`, ranger cette liste dans l'ordre croissant (le premier terme m de la liste à calculer sera le minimum de L, le suivant sera le minimum de la liste L dont on aura retiré m, *etc*).
2 - Ranger L dans l'ordre décroissant.

Mots-clefs : primitives [11] `min()` et `max()`, boucle **pour**, méthodes `list.sort()` et `list.reverse()`, commandes `L[: p]` et `L[p :]` [12], fonction `len()` (pour les listes), primitive `sorted()`.

Solution [13]

Pour ranger L dans l'ordre croissant puis dans l'ordre décroissant, on utilise normalement les méthodes `list.sort()` [14], [15] et `list.reverse()` [16].

1 - L'énoncé suggère le procédé suivant :

- le premier terme de la future liste dans l'ordre croissant sera le minimum m de L,
- le second sera le minimum de la liste obtenue en enlevant m de L (une seule fois si m y figure plusieurs fois)
- et ainsi de suite.

Cela se traduit par la fonction `aphrodite()` suivante qui retourne toute liste de nombres donnée après l'avoir rangée dans l'ordre croissant :

```
def aphrodite(L):
    Lcroissante = []# [] : liste vide
    for i in range(len(L)):# i vaut 0 puis 1 ... et enfin len(L)-1
        m = min(L)
        Lcroissante.append(m)# Ajoute m à la fin de la liste Lcroissante.
        L.remove(m)# Retire la première occurence de m dans L.
    return(Lcroissante)
```

L perd un élément à chaque passage de la boucle. À la fin, L est vide. La fonction `aphrodite()` a été écrite dans l'éditeur de texte de l'environnement de programmation utilisé (`Spyder`). Pour ranger dans l'ordre croissant la liste `L= [-2,4.17,8.79,-0.57,0.19,-4.23]`, exécutons `aphrodite()`, qui est ainsi chargée dans la console, ce qui se traduit par :

```
runfile('Chemin de la fonction aphrodite')
```

puis tapons :

```
L = [-2,4.17,8.79,-0.57,0.19,-4.23]
aphrodite(L)
Out[3]: [-4.23, -2, -0.57, 0.19, 4.17, 8.79]
L
Out[4]: []
```

11. cf. [1], section 4, p. 37 : une primitive désigne une fonction de base fournie par Python.
12. listes des p premiers termes et des p derniers termes de la liste L
13. Ceci est un exercice d'apprentissage. Il existe de nombreuses méthodes de tri.
14. remplace la liste par la même, ordonnée dans l'ordre croissant
15. ou la primitive `sorted()`
16. remplace la liste par la même, ordonnée dans l'ordre décroissant

On a vérifié que la valeur finale de `L` est `[]` et on a trouvé `Lcroissante=[-4.23, -2, -0.57, 0.19, 4.17, 8.79]`.

On a déjà dit qu'en temps normal, on aurait utilisé la méthode `list.sort()` ou la primitive `sorted()` [17]. La valeur originale de `L` ayant été récupérée par un copier-coller, cela donne :

```
L = [-2,4.17,8.79,-0.57,0.19,-4.23]
L.sort()
L
Out[9]: [-4.23, -2, -0.57, 0.19, 4.17, 8.79]
L = [-2,4.17,8.79,-0.57,0.19,-4.23]
sorted(L)
Out[11]: [-4.23, -2, -0.57, 0.19, 4.17, 8.79]
```

2 - Il suffit de modifier le script qui définit la fonction `aphrodite()` en remplaçant `min` par `max`.

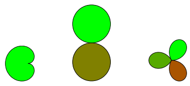
Remarque importante :

La liste `L` ci-dessus étant très courte et sans intérêt, remplaçons-la par une liste de 10 000 réels engendrée par la commande `reels(10000)` et ordonnons-la à l'aide de la fonction `aphrodite()` puis de la fonction `sorted()` :

```
from reels import *
[L,Lo] = reels(10000)# Lo est une copie indépendante de L.
from aphrodite import *
Lcroissante = aphrodite(L)# Suite L rangée dans l'ordre croissant
# à l'aide de la fonction aphrodite.
Lcroissante[:4]# 4 plus petits termes de L.
Out[6]:
[-29.41317007426497,
 -28.747176136743192,
 -26.928746199250646,
 -25.74374951009743]
Lcroissante[9996:]# 4 plus grands termes de L.
Out[7]:
[40.68458999805904,
 41.64978402811625,
 41.821776781909804,
 43.184709110328164]

Lcroissante2 = sorted(Lo)# Suite L rangée dans l'ordre croissant
# à l'aide de la fonction sorted().
Lcroissante2[:4]# Vérification : 4 plus petits termes de L.
Out[9]:
[-29.41317007426497,
 -28.747176136743192,
 -26.928746199250646,
 -25.74374951009743]
Lcroissante2[9996:]# Vérification : 4 plus grands termes de L.
Out[10]:
[40.68458999805904,
 41.64978402811625,
 41.821776781909804,
 43.184709110328164]
```

17. Voir [1], chapitre 1, 9. Les listes, p.27.

La primitive `sorted()` est beaucoup plus rapide que la fonction `aphrodite()` à tel point que si on veut ranger une liste d'un million de nombres, par exemple, on sera très tenté d'interrompre le premier calcul tellement c'est long alors que le second dure à peu près une seconde.



Chapitre 4

Matrices (m,)

Liste des exercices :

Énoncé nº 4.1 [**] : Le lièvre et la tortue, jeu équitable ?

Énoncé nº 4.2 [**] : Maximum et minimum d'une matrice-ligne

Les matrices relèvent du module `numpy` de Python qu'il faut donc charger dans la console lorsqu'on en utilise. Malheureusement, `numpy` distingue deux sortes de matrices, ce que l'on ne fait pas en mathématiques [1] et qui perturbera, au début, les mathématiciens :
• les matrices `(m, )` ou matrices à une dimension [2] ou vecteurs [3], par exemple

```
import numpy as np
A = np.array([1,2,3])
A
Out[3]: array([1, 2, 3])
type(A)
Out[4]: numpy.ndarray
A.dtype
Out[5]: dtype('int64')# Les éléments de A sont des entiers codés en 64 bits.
A.shape# ou shape(A)
Out[6]: (3, )# A est bien une matrice (m, ).
```

• les matrices multidimensionnelles (on se limitera aux matrices 2-dimensionnelles) [4]. Ainsi dans la console, à la suite de l'exemple précédent :

```
B = np.array([[1.1,2.2,3.3],[4.4,5,6]])
B
Out[8]:
array([[ 1.1,  2.2,  3.3],
       [ 4.4,  5. ,  6. ]])
type(B)
Out[9]: numpy.ndarray
B.dtype
```

1. En mathématiques, au niveau élémentaire, les matrices sont de simples tableaux rectangulaires de nombres qui ont donc «deux dimensions» : le nombre des lignes `n` et le nombre des colonnes `m` résumées en `(n,m)`.
2. sic
3. terme ambigü
4. Ce sont les vraies matrices, les matrices des mathématiciens. Elles servent principalement à représenter des applications linéaires, ce qui explique la curieuse définition du produit matriciel, qui correspond à la composition des applications.

```
Out[10]: dtype('float64')
B.shape
Out[11]: (2, 3)
```

On peut convertir une matrice `(m, )` en une matrice `(n,m)` comme suit :

```
C = A[np.newaxis]
C
Out[13]: array([[1, 2, 3]])# À comparer avec la sortie [3].
type(C)
Out[14]: numpy.ndarray
C.shape
Out[15]: (1, 3)# À comparer avec la sortie [6].
```

Ce chapitre comprend seulement des exercices sur les matrices (m,).

4.1 [**] Le lièvre et la tortue, jeu équitable ?

On lance un dé équilibré. Si le 6 sort, le lièvre gagne. Sinon la tortue avance d'une case et on rejoue. La tortue gagne si elle parvient à avancer de `n` cases ($n \geqslant 1$ donné).

1 - Modéliser et simuler ce jeu à l'aide de listes Python.

2 - Calculer la probabilité pour que le lièvre gagne, que la tortue gagne.

3 - Existe-t-il une valeur de `n` pour laquelle le jeu est équitable ?

Mots-clefs : liste, modélisation, mathématiques, probabilités, simulation.

Cet exercice très connu a peu d'intérêt calculatoire. C'est avant tout un exercice de modélisation[5], très simple s'il est bien modélisé. Les bonnes questions pour commencer sont : y a-t-il toujours un vainqueur ? combien faut-il de lancers ?

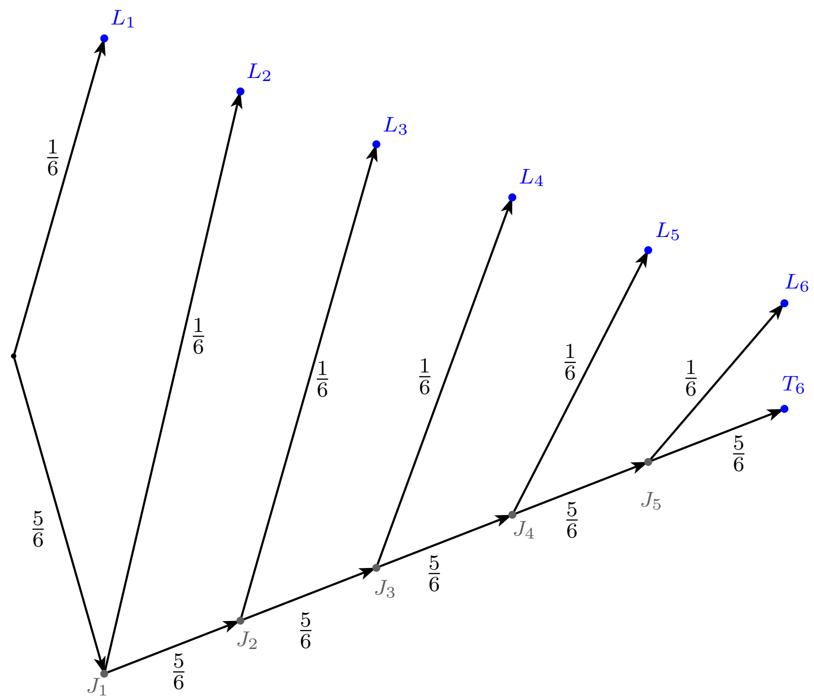
Il est évident que le jeu sera fini au `n`$^{\text{ième}}$ lancer parce qu'un 6 sera sorti, auquel cas le lièvre aura gagné ; sinon, la tortue gagnera au `n`$^{\text{ième}}$ lancer car elle aura avancé de `n` cases.

1 - Solution n°1 : Il n'est pas nécessaire dans tous les cas de lancer `n` fois le dé. Par exemple, si 6 sort au premier lancer et si `n>1`, les lancers suivants ne servent à rien car le lièvre a gagné au premier coup. Le nombre de lancers nécessaires pour que l'une des deux bestioles gagne est aléatoire. Cela conduit à un graphe[6] avec des branches de longueurs différentes. Par exemple[7], si `n = 6`,

• J_i, $i = 1, \ldots, 5$ désigne l'événement : « Après le i$^{\text{ème}}$ lancer, le jeu continue »,

• L_i, $i = 1, \ldots, 6$ l'événement « Le lièvre gagne au i$^{\text{ème}}$ lancer »,

• T_6 l'événément « La tortue gagne (nécessairement au 6$^{\text{ème}}$ lancer) ».



Cette modélisation sous forme de graphe est assez jolie et pratique. Dans ce cadre, nous simulerons le jeu par la fonction `lafontaine()` définie ci-dessous. On observera qu'elle traduit exactement l'énoncé : si un 6

5. On ne peut chercher de solution que dans le cadre d'une théorie mathématique ! Sinon, comment pourrait-on justifier nos calculs ?

6. qu'il est inutile de tracer

7. pour le plaisir

apparaît, elle retourne « Le lièvre a gagné », les calculs s'arrêtent. Sinon, les calculs continuent et s'ils vont jusqu'au bout, elle retourne « La tortue a gagné »[8].

```
from random import *
def lafontaine(n):
    for i in range(n):#Il y aura au plus n lancers du dé.
        if randint(1,6) == 6:
            return 'Le lièvre a gagné'
    return 'La tortue a gagné'
```

Exemple :

```
lafontaine(6)
Out[10]: 'Le lièvre a gagné'
lafontaine(3)
Out[11]: 'La tortue a gagné'
```

1 - Solution n°2 : Dans cette solution et dans la suivante, on imagine que l'on effectue les `n` lancers, même si un `6` sort avant le dernier lancer[9], [10]. On peut représenter ces `n` lancers à l'aide de `n` variables aléatoires indépendantes X_1, ..., X_n ne prenant chacune que les valeurs 1, 2, 3, 4, 5, 6 avec la probabilité $\frac{1}{6}$.
Simuler le jeu, c'est donc d'abord simuler `n` lancers successifs d'un dé, ce que l'on peut faire avec la fonction `lancersDe` de l'exercice 3.1, qui retournera la liste des chiffres sortis ; c'est ensuite interpréter cette liste : l'événement «L'une de ces variables a pris la valeur 6» modélise la victoire du lièvre, son complémentaire la victoire de la tortue. On obtient ainsi la fonction notée `esopeliste()`, très simple :

```
from lancersDe import *
def esopeliste(n):
    if max(lancersDe(6)) == 6:
        return 'Le lièvre a gagné'
    else:
        return 'La tortue a gagné'
```

Exemple : Importons la fonction `esopeliste()` dans la console puis exécutons-la 5 fois, dans le cas `n=6` :

```
from esopeliste import *
esopeliste(6)
Out[2]: 'Le lièvre a gagné'
esopeliste(6)
Out[3]: 'La tortue a gagné'
esopeliste(6)
Out[4]: 'La tortue a gagné'
esopeliste(6)
Out[5]: 'Le lièvre a gagné'
esopeliste(6)
Out[6]: 'Le lièvre a gagné'
```

1 - Solution n°3 : C'est la même que la solution n°2 sauf que l'on simule les `n` lancers du dé à l'aide d'une matrice-ligne retournée par la fonction `randint()`[11] du module `numpy.random`. au lieu d'une liste fournie

8. Dans ce script, on voit aussi comment utiliser la commande `return` pour arrêter une boucle.
9. C'est une modélisation différente de la précédente.
10. Si on veut représenter ce jeu par un graphe, chaque branche listant les résultats d'une suite de `n` lancers du dé, ce graphe aura 6^n branches de longueur `n`. Par exemple, si `n=6`, il aura $6^6 = 46\,656$ branches, ce qui est beaucoup trop. Heureusement, tracer un tel graphe est inutile.
11. commande `random.randint(1,7,n)` ou simplement `randint(1,7,n)`

par la fonction `entiersDe()`. C'est donc encore plus simple. La fonction simulatrice correspondante a été appelée `esopeligne()`.

```
from numpy.random import *
def esopeligne(n):
    if max(randint(1,7,n)) == 6:
        return 'Le lièvre a gagné'
    else:
        return 'La tortue a gagné'
```

Par exemple, simulons 4 fois le jeu du lièvre et de la tortue dans le cas `n=6`, directement dans la console :

```
from esopeligne import *
esopeligne(6)
Out[2]: 'Le lièvre a gagné'
esopeligne(6)
Out[3]: 'Le lièvre a gagné'
esopeligne(6)
Out[4]: 'La tortue a gagné'
esopeligne(6)
Out[5]: 'Le lièvre a gagné'
```

2 - Solution pour la première modélisation : On peut imaginer le graphe pour `n` quelconque. Il y a une seule branche à `n` rameaux, chacun de probabilité $\frac{5}{6}$, qui mène à la victoire de la tortue. La probabilité pour que la tortue gagne est donc $\left(\frac{5}{6}\right)^n$. Comme l'événement "Le lièvre gagne" et l'événement "La tortue gagne" sont complémentaires, le lièvre gagne donc avec la probabilité $1-\left(\frac{5}{6}\right)^n$.

2 - Solution pour la deuxième modélisation : La probabilité pour que la tortue gagne est $\left(\frac{5}{6}\right)^n$ puisque la probabilité pour que chacune des variables aléatoires ne prennent pas la valeur 6 est $\frac{5}{6}$ et que ces variables aléatoires sont indépendantes. On obtient donc les mêmes résultats dans les deux modélisations. On n'avait d'avance aucun doute à ce sujet [12].

3 - La question posée signifie : "Y a-t-il une valeur de `n` telle que $\left(\frac{5}{6}\right)^n = \frac{1}{2}$ ou $5^n = 2^{n-1} \times 3^n$. Si une telle valeur de `n` existait, on en déduirait que 3 divise 5, ce qui est faux [13]. On peut aussi résoudre l'équation $\left(\frac{5}{6}\right)^x = \frac{1}{2}$, $x \in [1, +\infty[$ dont l'unique solution, qui n'est pas un nombre entier, est $x = \frac{\log 2}{\log 6 - \log 5}$, que l'on peut calculer avec Python :

```
from math import *
x=(log(2))/(log(6)-log(5))
x
Out[3]: 3.801784016923929
```

12. Dans des problèmes compliqués, on peut imaginer que des modélisations différentes conduisent à des résultats différents, auquel cas se pose le problème de la validation des modèles. Dans les cas simples, ce problème ne se pose jamais.

13. ou qu'un même entier $\geqslant 1$ peut avoir deux décompositions comme produit de nombres premiers, ce qui est faux aussi.

4.2 [**] Maximum et minimum d'une matrice-ligne

1 - Calculer le maximum d'une matrice-ligne de nombres `M` en procédant comme suit :
(a) - on choisit arbitrairement l'un de ces nombres, noté `ma` [a] et on efface les nombres $\leqslant$ `ma`.
(b) - s'il n'en reste plus, `ma` est le maximum recherché.
(c) - sinon, on recommence (a) et (b).
2 - Calculer le minimum `mi` de `M`.

a. Ci-dessous, `ma` est le premier terme de `M`.

Mots-clefs : matrices-lignes, commande `len()`, masque, maximum et minimum, fonctions primitives `max()` et `min()`, boucle `tant que`, importation de fonctions, module `time`, fonction `clock()`, module `copy`, fonction `deepcopy()`.

Solution

S'agissant de matrices, on aura besoin d'importer le module `numpy` ; on utilisera un masque de matrice [14], [15], ce qui fait l'intérêt de cet exercice au demeurant très court.

1 - On peut utiliser le script commenté suivant qui définit une fonction notée `maxime()` qui, comme la primitive `max()` [16], retourne le maximum d'une matrice-ligne de nombres donnée.

```
# La fonction maxime utilise le masque M > ma.
from numpy import *

def maxime(M) : # M est une matrice-ligne de nombres.
    while len(M)>=1 :# Tant que M n'est pas vide.
        ma = M[0]# Le choix de ma dans M est arbitraire.
        M = M[M>ma]# On ne conserve que les éléments de M > ma.
        # La longueur de M diminue à chaque passage de la boucle.
    return(ma)
```

La commande `M = M[M>ma]` ne retient de `M` que les nombres `> ma`, autrement dit efface les autres.

Exemple : Importons la fonction `maxime()` dans la console [17], [18], puis cherchons le maximum de la matrice-ligne `M` retourné par la commande `M = randn(1000000)` [19] :

```
from maxime import *
from numpy.random import *
M = randn(1000000)
maxime(M)
Out[7] : 4.8898710392903109
```

Le calcul a été pratiquement instantané.

2 - Pour calculer le minimum d'une matrice `Mat`, on voudrait utiliser l'égalité

`min(Mat) = - max(-Mat)`, autrement dit, `min(Mat) = - maxime(-Mat)`.

14. cf. [7], p. 17-18.
15. On verra que les masques de matrices sont très utiles.
16. `max()` fait mieux. En effet, `max()`s retourne le maximum d'une matrice-ligne ou d'une liste de nombres alors que `maxime()` ne s'applique pas aux listes.
17. On la charge comme si c'était un module.
18. ce qui provoque aussi le chargement de `numpy`
19. `randn` est une fonction du module `numpy.random` qu'il faut aussi charger dans la console. Il est inutile de savoir que `M` est un échantillon de la loi normale centrée réduite de taille 1 000 000.

vraie pour n'importe quelle matrice `Mat`. Malheureusement, la valeur initiale de `M` a été perdue dans le calcul de `maxime(M)`. On aurait dû conserver cette valeur initiale la ré-utiliser dans le calcul du minimum. Cela se fait avec la fonction `deepcopy()` du module `copy` [20].

Exemple : Engendrons une matrice-ligne de 1 000 000 de nombres flottants comme ci-dessus, puis nous calculons son maximum et son minimum ; enfin, faisons apparaître ces valeurs à l'aide de `print()`.

```
from maxime import *
from copy import deepcopy
from numpy.random import randn

M = randn(1000000)
Mbis = deepcopy(M)
ma = maxime(M)
mi = -maxime(Mbis)
print(ma, mi)
4.59811940548 -4.59811940548
```

Fin de l'exercice

Compléments : comparaison des vitesses de `maxime()` et de `max()`.

Pour comparer les vitesses de `maxime()` et de la primitive `max()`, considérons le script suivant, qui donne les durées du calcul du maximum de la matrice-ligne `M` ci-dessous par `maxime()`, puis par `max()`. Pour fixer les idées, nous lançons une nouvelle session de calcul en tapant dans l'interpréteur :

```
from time import clock # Pour compter les durées de calcul.
from copy import deepcopy # Pour fabriquer une copie indépendante de M.
from maxime import * # Importation de la fonction maxime.
# Cette importation comprend le module numpy.

M=random.randn(1000000)
Mbis=deepcopy(M)
d=clock()
print('Le maximum de M par maxime est',maxime(M))
f=clock()
print('Le temps de calcul par maxime est',f-d)
d=clock()
print('Le maximum de Mbis par max est',max(Mbis))# Autre calcul du maximum.
f=clock()
print('Le temps de calcul par max est',f-d)
```

puis nous exécutons ce script :

```
runfile('Chemin du script')
Le maximum de M par maxime est 4.69931038606
Le temps de calcul par maxime est 0.025789999999999758
Le maximum de Mbis par max est 4.69931038606
Le temps de calcul par max est 0.07035400000000003
```

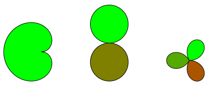
La comparaison des durées de calcul est surprenante car la fonction `maxime()` paraît plus rapide que la fonction primitive de Python `max()`.

20. Voir [1], p. 23-25

Faisons un deuxième essai. `M` est maintenant un échantillon de taille 80000 de la loi uniforme sur l'intervalle d'entiers [-100000, 250000] [21], [22]. On obtient :

```
runfile('Chemin du script modifié')
Le maximum de M par maxime est 249999
La durée du calcul par maxime est 0.0018669999999999841
Le maximum de M par max est 249999
La durée du calcul par max est 0.008494999999999992
```

Cela semble confirmer la remarque précédente sur la vitesse de `maxime()` et de `max()`.



21. ce que l'on peut ignorer ; sur la fonction `randint` du module `numpy.random`, voir [1], p. 163.
22. Il suffit, dans le script précédent, de remplacer la commande `M=random.randn(1000000)` par la commande `M=randint(-100000,250001,size = 80 000)`.

Chapitre 5

Matrices (matrices (n,m))

Liste des exercices :

Énoncé nº 5.1 [*] : Calculer la trace d'une matrice.

Énoncé nº 5.2 [**] : Ce carré est-il un carré magique ?

Énoncé nº 5.3 [***] : Transposer une matrice

Les matrices ou matrices n-dimensionnelles (`ndarray` pour Python) sont les matrices des mathématiciens. Le module `numpy` leur est consacré, voir [12].

5.1 [*] Calculer la trace d'une matrice

Soit M une matrice à n lignes et m colonnes.
1 - On suppose d'abord que M est carrée (autrement dit n=m). Calculer la somme t de ses éléments diagonaux [a] et la somme t2 des éléments de sa deuxième diagonale [b].
2 - Même question quand on ne suppose plus que n=m [c].

a. c'est à dire ceux dont l'indice de ligne est égal à l'indice de colonne
b. c'est à dire ceux dont l'indice i de ligne et l'indice j de colonne vérifient $i + j = n + 1$.
c. t s'appelle la trace de M.

Mots-clefs : module numpy, extraire un élément, une sous-matrice d'une matrice, fonction shape(), boucle pour, changer l'ordre des lignes ou des colonnes d'une matrice, fonction rand() du module random de numpy, fonctions diag(), sum() et trace() de numpy.

Solution

1 - Cette question, très simple, ne comporte qu'une boucle pour. On initialise la somme t par t = A[0,0], puis on ajoute les termes diagonaux suivants, s'il y en a. Cela donne la fonction traCe() que nous définissons comme suit :

```
# La fonction traCe() retourne la trace de toute matrice carrée A.
from numpy import *
def traCe(A) :
n = shape(A)[0]# Nombre de lignes de A.
    # shape(A) est la dimension de A. C'est un t-uplet.
    t = A[0,0]# Nombre en haut, à gauche de A.
    for i in range(1,n) :# i prend successivement les valeurs 1, ..., n-1
        t = t + A[i,i]
    return t
```

Exemples :

```
from traCe import *
A = array([[1,2],[4.7,-8]])# A est une matrice carrée d'ordre 2.
traCe(A)
Out[2]: -7.0
A = array([[-0.5]])# A est une matrice à une ligne et une colonne.
traCe(A)
Out[3]: -0.5
```

Pour le calcul de t2, on peut programmer une fonction analogue à la fonction traCe(), par exemple :

```
from numpy import *
def traCce(A) :# A est une matrice carrée.
    n = shape(A)[0]# Nombre de lignes de A.
    t2 = A[0,n-1]# Nombre en haut, à droite de A.
    for i in range(1,n,1) :# i prend successivement les valeurs 2, ..., n-1
        t2 = t2 + A[i,n-1-i]
    return t2
```

<u>Autre solution :</u> On peut ramener le calcul de t2 au calcul d'une trace. En effet, on peut changer facilement l'ordre des colonnes [1] d'une matrice. Par exemple, étant donné une matrice A, la commande

1. ou des lignes

`A[:,range(shape(A)[1]-1,-1,-1)]` retourne la matrice dont la première colonne est la dernière colonne de `A`, etc[2]. Dans cette manipulation, les diagonales se sont échangées. Par conséquent le calcul de `t2` ci-dessus se résume à :

```
t2 = traCe(A[:,range(shape(A)[1]-1,-1,-1)])
```

Exemple :[3]

```
from traCe import *
A = random.rand(4,4)# A est une matrice à 4 lignes et 4 colonnes de nombres
# tirés selon la loi uniforme dans [0,1[ , voir le module random de numpy.
A
Out[3]:
array([[ 0.18018729,  0.45399873,  0.53690069,  0.79493604],
       [ 0.77820911,  0.37673151,  0.81483884,  0.94317088],
       [ 0.06192849,  0.84753751,  0.20882323,  0.27883573],
       [ 0.5368506 ,  0.2511917 ,  0.33050611,  0.7245143 ]])
t2 = traCe(A[:,range(shape(A)[1]-1,-1,-1)])
t2
Out[5]: 2.9941629870258559
0.79493604+0.81483884+0.84753751+0.5368506
Out[6]: 2.9941629899999995
```

La différence constatée provient de la troncation effectuée lors de l'affichage de `A`.

2 - Si l'on ne suppose pas que `A` est une matrice carrée et si l'on pose `p = min(n,m)`, il est clair que `t` est la somme des éléments diagonaux de la matrice extraite `A[0:p,0:p]`. Idem pour l'autre diagonale. On est donc ramené au problème précédent.

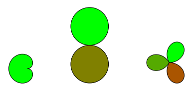
Exemple : Le script qui suit permet de calculer la trace `t` d'une matrice rectangulaire `A`[4] ainsi que la somme `t2` des éléments de sa deuxième diagonale de trois manières différentes, en utilisant, outre la fonction `traCe()` définie ci-dessus, les fonctions `trace()`[5], `diag()`[6] et `sum()`[7] du module `numpy`.

```
from traCe import *# ce qui importe le module numpy

A = random.rand(4,6)
print(A)
p = min(shape(A))
B = A[0:p,0:p]
C = B[:,range(shape(B)[1]-1,-1,-1)]
t,tbis,tbbis = traCe(B), sum(diag(B)), trace(B)# Trois calculs de la
# trace de A.
t2,t2bis,t2bbis = traCe(C), sum(diag(C)), trace(C)# Trois calculs de la
# somme des éléments de la deuxième diagonale.
print(t,tbis,tbbis,t2,t2bis,t2bbis)
```

Exemple :
Chargeons le script précédent dans la console :

2. de même, la commande `A[range(shape(A)[1]-1,-1,-1),:]` retourne la matrice dont la première ligne est la dernière ligne de `A`, etc
3. Il est très commode d'utiliser des matrices aléatoires quand on a besoin de tester des scripts.
4. éventuellement carrée
5. `trace(A)` est la trace de la matrice `A`.
6. `diag(A)` est la matrice-ligne des éléments diagonaux de la matrice `A`.
7. `sum(A)` est la somme des éléments de la matrice `A`.

```
runfile('Chemin du script précédent')
Reloaded modules: traCe
[[ 0.31233025  0.11421513  0.66913549  0.67042439  0.0210983   0.02319104]
 [ 0.03356263  0.51434873  0.08582213  0.00259724  0.31993838  0.85780133]
 [ 0.54226341  0.66872535  0.04980886  0.02022969  0.85437218  0.32762207]
 [ 0.20211522  0.72709937  0.6666164   0.84249907  0.77062698  0.01884415]]
1.71898689944 1.71898689944 1.71898689944 1.62708708326 1.62708708326
1.62708708326
```



5.2 [**] Ce carré est-il un carré magique ?

Convenons qu'une matrice carrée `M` d'ordre `n` est un carré magique d'ordre `n` si ses éléments sont les entiers de 1 à n^2 disposés de sorte que leurs sommes sur chaque ligne, sur chaque colonne et sur les deux diagonales soient égales. La valeur commune de ces sommes est appelée constante magique de `M`.
1 - S'il existe un carré magique d'ordre `n`, combien vaut sa constante magique ?
2 - `LuoShu = array([[4,9,2],[3,5,7],[8,1,6]])` est-il un carré magique ? Si oui, quelle est sa constante magique ?
3 - Même question pour `Cazalas = array([[1,8,53,52,45,44,25,32], [64,57,12,13,20,21,40,33], [2,7,54,51,46,43,26,31], [63,58,11,14,19,22,39,34], [3,6,55,50,47,42,27,30], [62,59,10,15,18,23,38,35], [4,5,56,49,48,41,28,29], [61,60,9,16,17,24,37,36]])`.
4 - Même question pour `Franklin = array([[52,61,4,13,20,29,36,45], [14,3,62,51,46,35,30,19], [53,60,5,12,21,28,37,44], [11,6,59,54,43,38,27,22], [55,58,7,10,23,26,39,42], [9,8,57,56,41,40,25,24], [50,63,2,15,18,31,34,47], [16,1,64,49,48,33,32,17]])`.

Mots-clefs : `V.size` [8], `M.sum(axis=1)`, `M.sum(axis=0)`, `trace(M)`, `M[arange(n-1,-1,-1), arange(0,n)]` (extraction d'une matrice 1-dimensionnelle de `M`), `hstack((,))`, `min()`, `max()`.

Voir l'intéressant article [17] de Wikipedia sur les carrés magiques.

Solution

1 - La somme des éléments de ce carré est $1+2+\cdots+n^2 = \frac{n^2(n^2+1)}{2}\cdot$ Ce nombre est aussi n fois la constante magique [9]. La constante magique est donc $\dfrac{n(n^2+1)}{2}\cdot$

2, 3 et 4 - Ces questions sont faciles puisque, une matrice d'ordre `n` dont les éléments sont les entiers de 1 à n^2 étant donnée, il suffit de calculer les sommes de ses éléments sur chaque ligne, chaque colonne et sur les deux diagonales et de les comparer. On pourra utiliser les commandes suivantes :
• `M.sum(axis=1)` retourne la matrice à une dimension des sommes des lignes de `M`,
• `M.sum(axis=0)` retourne la matrice à une dimension des sommes des colonnes de `M`,
• `trace(M)` retourne la somme des éléments de la diagonale principale de `M` [10] ,
• `M[arange(n-1,-1,-1), arange(0,n)]` est la matrice 1-dimensionnelle extraite de la matrice `M` dont les éléments ainsi définis : leurs indices de ligne forment la matrice à une dimension `arange(n-1,-1,-1)`, leurs indices de colonne forment la matrice à une dimension `arange(0,n)` [11].
• On peut aussi calculer la somme des éléments de la deuxième diagonale de `M` avec la commande : `trace(A[:,range(shape(A)[1]-1,-1,-1)])` [12].

Cela donne la fonction `gorgibus` suivante :

```
# La fonction gorgibus retourne suivant le cas :
# 'M est un carré magique dont la constante magique est tant'
# ou
```

8. longueur de la matrice à une dimension `V`
9. Il y a n lignes, la somme de chacune d'elles étant la constante magique.
10. `trace` est une fonction du module `numpy`.
11. Il s'agit donc de la matrice 1-dimensionnelle des éléments de la deuxième diagonale de `M` lus de la droite vers la gauche et de haut en bas.
12. Voir l'exercice 5.1.

```
# 'M n'est pas un carré magique'.
from numpy import *
def gorgibus(M):
    n = M[:,0].size# Nombre de lignes et de colonnes de M.
    slignes = M.sum(axis=1)# Matrice 1-dimensionnelle
    # des sommes sur les lignes.
    scolonnes = M.sum(axis=0)# Matrice 1-dimensionnelle
    # des sommes sur les colonnes.
    sdiagonales = array([trace(M),sum(M[arange(n-1,-1,-1), arange(0,n)])])
    # Matrice 1-dimensionnelle à 2 éléments : la trace de M
    # et la somme des éléments de la 2 éme diagonale.
    sommes = hstack((slignes,scolonnes,sdiagonales))
    # sommes est une matrice 1-dimensionnelle de longueur 2*n+2.
    mi = min(sommes)
    ma = max(sommes)
    if mi == ma:
        return 'Le carré testé est un carré magique de constante
        magique '+str(mi)+'.'
    else:
        return "Le carré testé n'est pas un carré magique."
```

L'égalité de toutes les sommes calculées a été testée en comparant leur maximum et leur minimum.

Exemple :

```
from gorgibus import *
A = array([[1,2,3],[4,5,6],[7,8,9]])
A
Out[3]:
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
gorgibus(A)
Out[4]: "Le carré testé n'est pas un carré magique."
```

Application aux matrices `LuoShu`, `Cazalas` et `Franklin` :

```
LuoShu = array([[4,9,2],[3,5,7],[8,1,6]])
gorgibus(LuoShu)
Out[3]: 'Le carré testé est un carré magique de constante magique 15.'

Cazalas = array([[1,8,53,52,45,44,25,32],
[64,57,12,13,20,21,40,33],
[2,7,54,51,46,43,26,31],
[63,58,11,14,19,22,39,34],
[3,6,55,50,47,42,27,30],
[62,59,10,15,18,23,38,35],
[4,5,56,49,48,41,28,29],
[61,60,9,16,17,24,37,36]])
gorgibus(Cazalas)
Out[5]: 'Le carré testé est un carré magique de constante magique 260.'

Franklin = array([[52,61,4,13,20,29,36,45],
[14,3,62,51,46,35,30,19],
```

```
[53,60,5,12,21,28,37,44],
[11,6,59,54,43,38,27,22],
[55,58,7,10,23,26,39,42],
[9,8,57,56,41,40,25,24],
[50,63,2,15,18,31,34,47],
[16,1,64,49,48,33,32,17]])
gorgibus(Franklin)
Out[7]: "Le␣carré␣testé␣n'est␣pas␣un␣carré␣magique."

Franklin.sum(axis=1)
Out[8]: array([260, 260, 260, 260, 260, 260, 260, 260])
Franklin.sum(axis=0)
Out[9]: array([260, 260, 260, 260, 260, 260, 260, 260])
n = Franklin[:,0].size
n
Out[11]: 8
array([trace(Franklin),sum(Franklin[arange(n-1,-1,-1), arange(0,n)])])
Out[12]: array([228, 292])
```

On constate que les deux premiers carrés proposés sont magiques tandis que le troisième ne l'est pas, contrairement à ce qui est dit dans [17]. Ce dernier est seulement semi-magique, les sommes de ses lignes et de ses colonnes étant égales à 260 tandis que les sommes sur les diagonales sont différentes.

5.3 [***] Transposer une matrice

Programmer une fonction qui, étant donné une matrice numérique `A`, retourne la matrice `B` dont la première colonne est la première ligne de `A`, la seconde la deuxième ligne de `A`, etc. `B` s'appelle la transposée de `A`.

Mots-clefs : Boucle `pour` imbriquée dans une autre, commande `range`, module `numpy`, fonctions `shape()` (dimension d'une matrice), `zeros()`, `vstack()` et `hstack()` (empilements de matrices), `transpose()`, méthode (de l'objet matrice) `.T`.

Solution n°1 - Il y a évidemment des commandes de Python qui règlent la transposition des matrices, voir les exemples ci-dessous [13], [14] :

```
from numpy import *
A = array([[1.5,2,3,4,5],[6,7,8,9,10],[11,12,13,14,15]])
#A est une matrice (3,5) de réels.
print(A)
B = transpose(A)
print(B)#B est une matrice (5,3).
Bb = A.T
print(Bb)# On constate que B = Bb.
A1 = array([[1,2,3,4,5]])# A1 est une matrice (1,5) d'entiers.
print(A1)
B1 = transpose(A1)
print(B1)# B1 est une matrice (5,1).
B1b = A1.T
print(B1b)# On constate que B1 = B1b.
A2 = array([[1],[6],[11]])# A2 est une matrice (3,1) d'entiers.
print(A2)
B2 = transpose(A2)
print(B2)# B2 est une matrice (1,3).
B2b = A2.T
print(B2b)# On constate que B2 = B2b.
```

Exécutons ce script dans la console :

```
runfile('Chemin du script précédent')
[[  1.5   2.    3.    4.    5. ]
 [  6.    7.    8.    9.   10. ]
 [ 11.   12.   13.   14.   15. ]]
[[  1.5   6.   11. ]
 [  2.    7.   12. ]
 [  3.    8.   13. ]
 [  4.    9.   14. ]
 [  5.   10.   15. ]]
[[  1.5   6.   11. ]
 [  2.    7.   12. ]
 [  3.    8.   13. ]
 [  4.    9.   14. ]
```

13. `transpose()` est une fonction du module `numpy` qui retourne la transposée de toute matrice.
14. `.T` est une méthode de l'objet matrice qui effectue une transposition.

```
[    5.     10.     15.  ]]
[[1  2  3  4  5]]
[[1]
 [2]
 [3]
 [4]
 [5]]
[[1]
 [2]
 [3]
 [4]
 [5]]
[[  1]
 [  6]
 [11]]
[[  1   6  11]]
[[  1   6  11]]
```

Solution n°2 - Programmons maintenant nous-mêmes une fonction transposant toute matrice `A`. Pour cela, il suffit de remarquer que l'élément qui est sur la i$^{\text{ème}}$ ligne et la j$^{\text{ème}}$ colonne de la matrice transposée `B` est `A(j,i)` [15]. Si `l` et `c` désignent respectivement le nombre de lignes et le nombre de colonnes de `A` [16], [17], `i` courra de 1 à `c` et `j` courra de 1 à `l`. Il y aura donc une boucle `pour` imbriquée dans une boucle `pour` [18], [19] dans l'algorithme [20] définissant la fonction appelée `transposze()` ci-dessous :

```
from numpy import *
def transposze(A,a):# a est le type des éléments de A (int ou float)
    B = zeros((shape(A)[1],shape(A)[0]),dtype=a)
    for i in range(shape(A)[1]) :
        for j in range(shape(A)[0]) :
            B[i,j] = A[j,i]
    return B
```

Voici quelques exemples de calculs avec `transposze()` :

```
from transposze import *
A = random.rand(3,7)
A
Out[3]:
array([[ 0.92220832,  0.94755735,  0.15934383,  0.30450855,  0.24393324,
         0.25750654,  0.56433994],
       [ 0.08792988,  0.83437309,  0.07245987,  0.26802698,  0.81544128,
         0.67399154,  0.97167771],
       [ 0.77857619,  0.03994942,  0.43719886,  0.91426575,  0.45471858,
         0.6136873 ,  0.88858259]])
B = transposze(A,float)
B
```

15. `B(i,j)=A(j,i)`
16. donnés par la commande `shape(A)`, qui produit le 2-uplet `(l,c)`
17. Rappel : module `numpy` chargé, `A[0,0]` est un nombre, `array([0,0])` une matrice à une dimension réduite à un élément, `array([[0,0]])` une matrice réduite à un élément.
18. `shape(A)[0]` est le nombre de lignes de A.
19. `shape(A)[1]` est le nombre de colonnes de A.
20. Ce script présente une faiblesse. Normalement, Python doit donner le type des éléments de A que nous sommes ici obligés de saisir à la main.

```
Out[5] :
array([[ 0.92220832,  0.08792988,  0.77857619],
       [ 0.94755735,  0.83437309,  0.03994942],
       [ 0.15934383,  0.07245987,  0.43719886],
       [ 0.30450855,  0.26802698,  0.91426575],
       [ 0.24393324,  0.81544128,  0.45471858],
       [ 0.25750654,  0.67399154,  0.6136873 ],
       [ 0.56433994,  0.97167771,  0.88858259]])
A = random.randint(1,1001,(6,4))
A
Out[7] :
array([[462, 385, 989, 495],
       [158, 442, 881, 949],
       [933, 657, 206, 671],
       [981, 151, 557, 514],
       [176, 381, 516, 736],
       [481, 566,  49, 893]])
transposze(A,int)
Out[8] :
array([[462, 158, 933, 981, 176, 481],
       [385, 442, 657, 151, 381, 566],
       [989, 881, 206, 557, 516,  49],
       [495, 949, 671, 514, 736, 893]])
```

Solution n°3 - Pour aller plus vite, essayons de manipuler directement des lignes ou des colonnes. La première ligne de la matrice transposée `B` de `A` est `A[:,0]` [21]. Ensuite, il suffit d'empiler verticalement les autres lignes de `B` obtenues de la même façon à l'aide de la fonction `vstack()` [22]. Cela donne la fonction de transposition appelée `meli` ci-dessous :

```
from numpy import *
def meli(A):
    B = A[:,0]# Matrice (shape(A)[0], ).
    for i in range(1,shape(A)[1]):
        B = vstack((B,A[:,i]))# B est une matrice (i+1, shape(A)[0]).
    return B
```

La fonction `meli()` a un défaut : si `A` est une matrice (n,1) [23], elle retourne une matrice (n,) [24].

Exemple :

```
import numpy as np
A = np.random.rand(3,1)
A
Out[9] :
array([[ 0.75579927],
       [ 0.51379142],
       [ 0.15625903]])
from meli import *
meli(A)
Out[11]: array([ 0.75579927,  0.51379142,  0.15625903])
```

21. `A[:,0]` est une matrice (m,).
22. `vstack()` empile les matrices et les matrices (m,).
23. n lignes, 1 colonne
24. alors que l'on voudrait une matrice (1,n).

```
shape(meli(A))
Out[12] :  (3,)
```

On peut corriger ce défaut :

```
# La fonction melo() retourne la transposée de toute matrice A
# sous la forme d'une matrice.
import numpy as np
def melo(A) :
    n,m = np.shape(A)# shape(A) est un 2-uplet.
    B = A[:,0]# Matrice (m, ).
    if m == 1 :
        B = B[:,np.newaxis]# Création d'une matrice (n,1)
        # à partir d'une matrice (n, ).
    else :
        for i in range(1,m) :
            B = np.vstack((B,A[:,i]))# B est une matrice (i+1,n).
            #A la fin de la boucle, c'est une matrice (m,n).
    return B
```

Exemple :

```
import numpy as np
from melo import *
A = np.random.rand(2,3)
A
Out[3] :
array([[ 0.46926205,  0.79861174,  0.8205224 ],
       [ 0.59468157,  0.66871929,  0.83150586]])
melo(A)
Out[4] :
array([[ 0.46926205,  0.59468157],
       [ 0.79861174,  0.66871929],
       [ 0.8205224 ,  0.83150586]])
A = np.random.rand(1,3)
A
Out[6] : array([[ 0.69332173,  0.34597962,  0.50277346]])
melo(A)
Out[7] :
array([[ 0.69332173],
       [ 0.34597962],
       [ 0.50277346]])
A = np.random.rand(3,1)
A
Out[9] :
array([[ 0.75579927],
       [ 0.51379142],
       [ 0.15625903]])
```

Bibliographie

[1] Alexandre Casamayou-Boucau, Pascal Chauvin, Guillaume Connan *Programmation en Python pour les mathématiques*, 2ème édition, Dunod Éditeur (2016).

Ce manuel est en fait indispensable pour un professeur de mathématiques qui débute en Python.

[2] Wikibooks *Mathématiques avec Python et Ruby.*
http://fr.wikibooks.org/wiki/Math%C3%A9matiques_avec_Python_et_Ruby OK

[3] Gérard Swinnen *Apprendre à programmer avec Python 3*, Eyrolles Éditeur, 3ème édition (2016).

Python est un langage de programmation général. Ce manuel ne vise pas particulièrement les professeurs de mathématiques.

Programmes de mathématiques au lycée

[4] *Seconde, programme de mathématiques*
http://cache.media.education.gouv.fr/file/30/52/3/programme_mathematiques_seconde_65523.pdf

[5] *Seconde, mathématiques, Ressources pour la classe de Seconde*
http://cache.media.eduscol.education.fr/file/Programmes/17/8/Doc_ress_algo_v25_109178.pdf

Documentation sur des modules de Python

Cours de Télécom ParisTech

Le cours de Télécom ParisTech, de Alexandre Gramfort & Slim Essid, est disponible aussi en version pdf. Il est intéressant :
http://www.prepas.org/2013/Info/Liesse/Telecom/

En détail :

[6] *Introduction à Python*
http://www.prepas.org/2013/Info/Liesse/Telecom/1-Intro-Python.html
http://www.prepas.org/2013/Info/Liesse/Telecom/1-Intro-Python.pdf

[7] *Numpy et matplotlib*
http://www.prepas.org/2013/Info/Liesse/Telecom/2-Numpy.html
http://www.prepas.org/2013/Info/Liesse/Telecom/2-Numpy.pdf

[8] *Librairie d'algorithmes pour le calcul scientifique en Python*
http://www.prepas.org/2013/Info/Liesse/Telecom/3-Scipy.html
http://www.prepas.org/2013/Info/Liesse/Telecom/3-Scipy.pdf

Autres sources de documentation sur les modules

[9] Python : The Standard Python Library
https://docs.python.org/py3k/library/index.html

Liste complète des modules courants de Python. Celle liste permet d'accéder à la documentation sur les fonctions pré-définies de Python via ses modules.

[10] *Built-in functions*
https://docs.python.org/3/library/functions.html
Documentation officielle en anglais sur les fonctions toujours disponibles de «Python».

[11] *math - Mathematical functions*
https://docs.python.org/3/library/math.html
Documentation officielle en anglais sur le module «math».

[12] *Numpy reference*
https://docs.scipy.org/doc/numpy/reference/index.html
Documentation officielle en anglais décrivant les fonctions, modules et objets de «Numpy».

[13] *random - Generate pseudo-random numbers*
https://docs.python.org/3/library/random.html
Documentation officielle en anglais sur le module «random».

[14] *Random sampling*
https://docs.scipy.org/doc/numpy/reference/routines.random.html#permutations
Documentation officielle sur la génération de nombres aléatoires, des tirages aléatoires, sur les différents types courants de lois de probabilité, du module «numpy.random».

[15] *Matplotlib User's Guide*
http://matplotlib.org/users/
Documentation en anglais sur le module «Mathplotlib». On préfèrera [16].

[16] *Tutoriel Matplotlib* sur le site «Developpez.com» :
http://python.developpez.com/tutoriels/graphique-2d/matplotlib/
Très bonne documentation en français sur le module «Mathplotlb».

Articles divers

[17] Wikipedia *Carré magique (mathématiques)*
https://fr.wikipedia.org/wiki/Carre_magique_(mathematiques)#Ordre_12

[18] Jean-Marc Duquesnoy, Pierre Lapôtre, Raymond Moché *Algorithme de Bresenham 1 et 2* :
http://gradus-ad-mathematicam.fr/Premiere_Algorithmique2.htm