TD/TP n°2 :

Prise en main des packages numpy et matplotlib
Minimisation d'une fonction d'une variable

I. Objectifs :

- Mettre en oeuvre et comparer deux algorithmes de minimisation d'une fonction d'une variable
- Prendre en main l'environnement de développement Spyder
- Utiliser les librairies numpy et matplotlib
- Rédiger un compte-rendu sous forme de notebook

II. Environnement de travail :

a. Python avec Anaconda/Spyder :

Anaconda est une plateforme logiciel libre comprenant l'environnement Python complet et de très nombreuses librairies, dont NumPy et MatPlotLib. Elle tourne sur tous les OS courants, dont Windows.

Anaconda est fourni avec l'environnement de travail Spyder (Scientific PYthon Development EnviRonment). Cet environnement comporte différentes fenêtres de travail qui permettent de travailler de manière efficace. Les trois plus importantes sont l'éditeur de texte avec vérification en ligne de la syntaxe, la console IPython et l'explorateur de variables (Figure 1). Le fonctionnement de cet environnement est très similaire à matlab, avec la possibilité d'écrire des programmes ou d'exécuter des commandes en ligne. Vous trouverez toutes les informations nécessaires à l'utilisation de cet environnement sur internet.

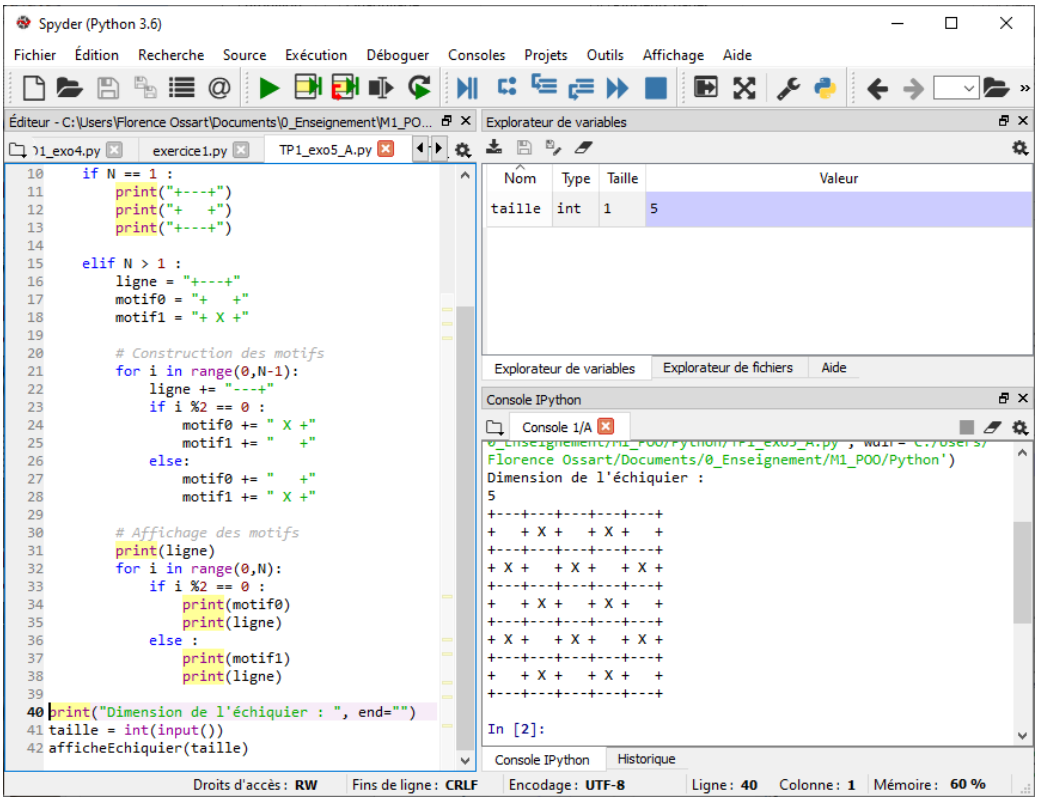


Figure 1 : Environnement de travail Spyder

b. NumPy :

« NumPy est une bibliothèque pour langage de programmation Python, destinée à manipuler des matrices ou tableaux multidimensionnels ainsi que des fonctions mathématiques opérant sur ces tableaux » [1]. L'objet de base utilisé dans NumPy est le tableau multidimensionnel homogène, c'est-à-dire constitué d'éléments de même type. Cet objet est manipulé grâce à la classe `ndarray`, aussi référencée par l'alias `array`. Les fonctions mathématiques classiques sont redéfinies dans NumPy de façon à s'appliquer à des tableaux, évitant ainsi l'utilisation de boucles. Par exemple, si `x` est un tableau de réels, l'instruction `np.sin(x)` renvoie un tableau de même dimension que `x` qui contient le résultat de la fonction sinus appliquée à chacun des éléments de `x`.

Voici certains des attributs de la classe ndarray les plus couramment utilisés :

- `ndarray.ndim` nombre de dimensions du tableau
- `ndarray.shape` dimension du tableau
- `ndarray.size` nombre total d'éléments du tableau

Il existe de nombreuses méthodes pour une manipulation facile des tableaux. En voici quelques exemples :

- `array` création d'un tableau à partir d'une liste de valeurs
- `zeros` création d'un tableau plein de zéros
- `arange` création d'une suite arithmétique de valeurs
- `linspace` création d'une suite valeurs réparties dans un intervalle
- `reshape` redimensionnement d'un tableau

L'exemple suivant, issu de numpy.org, montre le résultat de quelques unes de ces instructions.

```
>>> import numpy as np
>>> a = np.arange(15).reshape(3, 5)
>>> a
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14]])
>>> a.shape
(3, 5)
>>> a.ndim
2
>>> a.dtype.name
'int64'
>>> a.itemsize
8
>>> a.size
15
>>> type(a)
<type 'numpy.ndarray'>
>>> b = np.array([6, 7, 8])
>>> b
array([6, 7, 8])
>>> type(b)
<type 'numpy.ndarray'>
```

A côté des méthodes permettant la création des tableaux, il existe des méthodes de calcul sur les données des tableaux : addition, produit terme à terme, produit matriciel, et plus généralement application de fonctions mathématiques terme à terme (voir exemple ci-dessous, issu de numpy.org).

[1] Extrait de wikipédia

```
>>> a = np.array( [20,30,40,50] )
>>> b = np.arange( 4 )
>>> b
array([0, 1, 2, 3])
>>> c = a-b
>>> c
array([20, 29, 38, 47])
>>> b**2
array([0, 1, 4, 9])
>>> 10*np.sin(a)
array([ 9.12945251, -9.88031624,  7.4511316 , -2.62374854])
>>> a<35
array([ True, True, False, False])
```

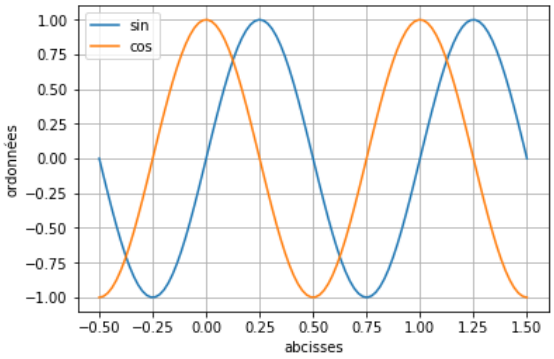
c. Matplotlib (matplotlib.org) :

La bibliothèque Matplotlib, déjà utilisée dans le TD/TP1, fournit des outils de représentation graphique de fonctions en 2D, à partir de tableaux de points. Il existe de nombreux cours et tutoriaux en ligne pour apprendre à utiliser cette bibliothèque.

Pour utiliser les fonctions de la bibliothèque Matplotlib, il faut l'importer, en utilisant un alias pour alléger les appels aux fonctions. D'une manière générale, les fonctions de la bibliothèque attendent des tables de valeurs, sous format ndarray. Matplotlib est donc utilisée conjointement avec numpy.

Les figures ci-dessous donnent un exemple d'utilisation de matplotlib et le graphe qui en résulte dans la fenêtre d'exécution.

```
1 # -*- coding: utf-8 -*-
2
3 import matplotlib.pyplot as plt
4 import numpy as np
5
6 x = np.linspace(-0.5,1.5,100)
7 y1 = np.sin(x*2*np.pi)
8 y2 = np.cos(x*2*np.pi)
9
10 plt.plot(x,y1)
11 plt.plot(x,y2)
12 plt.grid()
13 plt.ylabel("ordonnées")
14 plt.xlabel("abcisses")
15 plt.legend(["sin","cos"])
16
```



d. Exemples de sites à consulter pour se former :

- guide de démarrage rapide de NumPy : https://numpy.org/devdocs/user/quickstart.html
- NumPy for MATLAB users : https://numpy.org/devdocs/user/numpy-for-matlab-users.html
- guide de démarrage de matplotlib : https://matplotlib.org/tutorials/introductory/usage.html#sphx-glr-tutorials-introductory-usage-py
- https://courspython.com/apprendre-numpy.html#
- http://math.mad.free.fr/wordpress/wp-content/uploads/intronumpy.pdf

III. Recherche du minimum d'une fonction par la méthode de dichtomie

Le but de cette partie du TP est de mettre en œuvre la méthode de dichotomie afin de déterminer la position du minimum d'une fonction d'une variable dont on ne connaît pas les dérivées.

a. Présentation de la méthode à mettre en œuvre

Schéma général de l'algorithme de minimisation par dichotomie :

Cette méthode de recherche s'applique quand on n'a pas accès aux dérivées de la fonction à minimiser. En revanche, elle nécessite de connaître un intervalle sur lequel la fonction est « unimodale[2] » et présente un minimum unique. L'algorithme réduit progressivement la taille de cet intervalle par un processus itératif pour avoir un encadrement de la position du minimum de plus en plus précis.

- Notation : $[x_{1,0}, x_{5,0}]$ désigne l'intervalle de départ. $[x_{1,n}, x_{5,n}]$ désigne l'intervalle à la fin de l'itération n.
- On dispose d'une procédure *dichotomie* (détaillée dans la section suivante) qui permet de déterminer un sous-intervalle contenant le minimum.
  Formellement : $[x_{1,n}, x_{5,n}] = dichotomie([x_{1,n-1}, x_{5,n-1}])$ avec $x_{1,n} \geq x_{1,n-1}$ et $x_{5,n} \leq x_{5,n-1}$
- Pour obtenir un encadrement de la position du minimum avec une précision ε donnée, il faut réduire l'intervalle tant que $(x_{5,n} - x_{1,n}) > \varepsilon$.
- Algorithme :
  - Choix de l'intervalle initial $[x_{1,0}, x_{5,0}]$
  - Initialisation : $[x_{1,n-1}, x_{5,n-1}] \leftarrow [x_{1,0}, x_{5,0}]$
  - Tant que $(x_{5,n-1} - x_{1,n-1}) > \varepsilon$:
    - $[x_{1,n}, x_{5,n}] \leftarrow dichotomie([x_{1,n-1}, x_{5,n-1}])$
    - $[x_{1,n-1}, x_{5,n-1}] \leftarrow [x_{1,n}, x_{5,n}]$

Procédure *dichotomie* :

- Hypothèse : l'algorithme repose sur l'hypothèse que la fonction f est unimodale sur l'intervalle $[x_1, x_5]$ et possède un minimum unique. L'intervalle est découpé en 4 sous-intervalles de largeurs identiques, et la comparaison de la valeur de la fonction aux différentes bornes des sous-intervalles permet d'identifier dans quels sous-intervalles le minimum se trouve.
- On introduit les points intermédiaires : $x_3 = \frac{1}{2}(x_1 + x_5)$, $x_2 = \frac{1}{2}(x_1 + x_3)$ et $x_4 = \frac{1}{2}(x_3 + x_5)$.
- On calcule la valeur de la fonction en ces points, ce qui donne les valeurs f_1, f_2, f_3, f_4 et f_5.
- On compare ces valeurs entre elles afin d'identifier et d'éliminer les intervalles sur lesquels la fonction est monotone. Cinq cas de figure peuvent se présenter :
  1. Si $f_1 < f_2 < f_3 < f_4 < f_5$: alors le minimum est nécessairement dans l'intervalle $[x_1, x_2]$
  2. Si $f_1 > f_2 < f_3 < f_4 < f_5$: alors le minimum est nécessairement dans l'intervalle $[x_1, x_3]$
  3. Si $f_1 > f_2 > f_3 < f_4 < f_5$: alors le minimum est nécessairement dans l'intervalle $[x_2, x_4]$
  4. Si $f_1 > f_2 > f_3 > f_4 < f_5$: alors le minimum est nécessairement dans l'intervalle $[x_3, x_5]$
  5. Si $f_1 > f_2 > f_3 > f_4 > f_5$: alors le minimum est nécessairement dans l'intervalle $[x_4, x_5]$

  On obtient ainsi un nouvel intervalle dont la largeur a été divisée par 4 dans les cas 1 et 5, et par 2 dans les cas 2, 3 et 4

[2] La fonction f est unimodale sur $[a, b]$ si elle dérivable, et que sa sa dérivée s'annule et change de signe en un point unique de l'intervalle, noté c.

b. Travail à réaliser :

Ecrire une méthode qui met en œuvre la méthode de dichotomie présentée dans la section précédente et détermine le minimum d'une fonction donnée. Les paramètres de la méthode sont : le nom de la fonction à minimiser, les bornes de l'intervalle de départ de recherche, la précision attendue (largeur de l'intervalle final).

La méthode renvoie les variables suivantes : liste des bornes inférieures et supérieures des intervalles successifs, nombre d'itérations, indicateur d'erreur (False si l'algorithme s'est déroulé sans erreur, True dans le cas contraire).

Cette méthode sera testée avec la fonction-test $f_1(x) = (x+1)^2 + 7\sin(x)$, dont on recherchera le minimum dans l'intervalle $[-4, +4]$. La méthode sera nommée 'minimumDichotomie'. Une fois mise au point, elle devra pouvoir être appelée sans erreur dans le programme principal suivant (fichier .py fourni).

```
#%% Programme de test de la recherche de minimum par dichotomie

#%% Fonction test n°1
def f1(x) :
    return (x+1)**2 + 7*np.sin(x)

#%% Recherche du minimum de f1 sur l'intervalle [-4,4]
x_min = -4
x_max = +4

f = f1
precision = 1e-1
bornes_min, bornes_max, n_iter, ier = minimumDichotomie(f,x_min,x_max,precision)
x_min, y_min = bornes_min[0][-1], bornes_min[1][-1]
x_max, y_max = bornes_max[0][-1], bornes_max[1][-1]

# Visualisation des résultats
plt.plot(bornes_min[0],bornes_min[1],'rs', label = 'x_min')
plt.plot(bornes_max[0],bornes_max[1],'bs', label = 'x_max')
plt.legend()
plt.xlabel('Valeurs de $x$')
plt.ylabel('Valeurs de $f_1(x)$')
plt.title('Recherche du minimum de $f_1$ par dichotomie')
plt.grid()

message = 'Precision = {}'.format(precision)
message += '\nCV en {} iterations'.format(n_iter)
message += '\nBorne inférieure :'
message += '\n  x_min = {:6.4f}'.format(x_min)
message += '\n  y_min = {:6.4f}'.format(y_min)
message += '\nBorne supérieure :'
message += '\n  x_max = {:6.4f}'.format(x_max)
message += '\n  y_max = {:6.4f}'.format(y_max)
plt.text(1,-5,message)
```

Tester le programme pour différentes précisions et relever le nombre d'itérations nécessaires pour atteindre une précision donnée.

Tester ensuite le programme pour la fonction $f_2(x) = (x+1)^2 + 10\sin(x)$. Que se passe-t-il ? Quelle précaution faut-il prendre avant de lancer la recherche de minimum par dichotomie ? Dans votre réflexion, il faut garder à l'esprit que la fonction à minimiser peut-être très coûteuse à calculer et que l'on cherche à limiter le nombre d'évaluations de la fonction à minimiser.

<u>Méthode de travail préconisée :</u>

- Ecrire et mettre au point la fonction dichotomie dans l'environnement spyder
- Réaliser les tests et faire le compte rendu dans un notebook. Pour ne pas surcharger le notebook, vous pouvez importer les fonctions que vous avez développées. Il faut alors fournir le fichier notebook « .ipynb » et les fichiers des fonctions « .py »

IV. Recherche du minimum d'une fonction par la méthode de Newton

Le but de cette partie du TP est de mettre en œuvre la méthode de Newton afin de déterminer la position du minimum d'une fonction d'une variable dont on connaît pas les dérivées d'ordre 1 et 2. Cette méthode est très efficace dans le cas de fonctions convexes (dérivée seconde positive).

a. Présentation de la méthode à mettre en œuvre

La méthode de Newton est basée sur l'approximation de la fonction par son développement limité d'ordre 2 (parabole), Le minimum de l'approximation parabolique est facile à déterminer, et on construit une succession de solutions approchées.

Algorithme :

On souhaite déterminer le minimum de la fonction f, supposée convexe sur $\mathbb{R}$.

Soit x_n, la solution approchée à l'itération n.

La fonction est approximée par $\tilde{f}_n$ son développement limité d'ordre 2 en x_n:

$$\tilde{f}_n(x) = f(x_n) + f'(x_n)(x - x_n) + \frac{1}{2}f''(x_n)(x - x_n)^2$$

La nouvelle solution approchée x_{n+1} est le minimum de $\tilde{f}_n$, obtenu en annulant la dérivée de $\tilde{f}_n$.

$$\tilde{f}_n'(x) = f'(x_n) + f''(x_n)(x - x_n)$$

$$\tilde{f}_n'(x_{n+1}) = 0 \iff f'(x_n) + f''(x_n)(x_{n+1} - x_n) = 0 \text{ , d'où : } x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)}$$

On obtient ainsi une nouvelle solution approchée.

b. Travail à réaliser

Ecrire une méthode qui met en œuvre la méthode de Newton présentée dans la section précédente et détermine le minimum d'une fonction donnée. Les dérivées d'ordre 1 et 2 seront programmées à partir de leur expression analytique. Vous pouvez utiliser les outils de dérivation formelle si vous le souhaitez, mais surtout *pas de dérivée numérique par différence finie* !

Les paramètres de la méthode sont : le nom de la fonction à minimiser, le nom de la fonction dérivée d'ordre 1, le nom de la fonction dérivée d'ordre 2, le point de départ de la recherche, la précision attendue, le nombre maximal d'itérations autorisé.

La méthode renvoie au moins les informations suivantes : solution trouvée, nombre d'itérations, indicateur d'erreur (False si l'algorithme s'est déroulé sans erreur, True dans le cas contraire).

Tester la méthode avec la fonction-test $f_1(x) = (x + 1)^2 + 7\sin(x)$, et analyser son comportement pour différents points de départ de la recherche.

Tester le programme pour différentes précisions et relever le nombre d'itérations nécessaires pour atteindre une précision donnée.

Faire une analyse comparative de la méthode de dichotomie et de la méthode de Newton.