TD1

Racines d'équations

On cherche dans ce TD à résoudre numériquement les équations de la forme $f(x) = 0$. On se concentrera sur un cas concret : l'évaluation numérique de $\sqrt{2}$, c'est à dire la racine positive de l'équation $f(x) = 0$, où $f : \left\{ \begin{array}{ccc} \mathbb{R} & \to & \mathbb{R} \\ x & \to & x^2 - 2 \end{array} \right.$.

1 Prise en main de spyder

Il existe de nombreux logiciels graphiques permettant d'utiliser python. Parmi eux spyder est utilisé dans les concours de recrutement de l'éducation nationale (CAPES, agrégation).

Pour le lancer sur les ordinateurs de la salle A202, on double clique sur l'icône « Konsole » puis on tape `/opt/anaconda3/bin/spyder` (suivi de la touche Entrée).

1.1 Configuration de spyder

Au premier démarrage il pourrait vous demander de choisir l'interpréteur Python : dans ce cas cliquez sur « use this environment », et la console Python démarrera alors. C'est une fenêtre dont la dernière ligne affiche simplement `>>>`

Vous pouvez choisir dans le menu « tools » quelles fenêtres apparaissent dans la partie droite, en plus de la console. Pour ces TDs, choisissez uniquement les fenêtres « interactive help » et « workspace ».

1.2 Utilisation de la console python

L'écran de spyder est composé de plusieurs fenêtres. Le plus importante de ces fenêtres et la Console Python. C'est une fenêtre qui, au démarrage, se termine par la ligne

```
>>>
```

Remarque : selon la configuration de l'ordinateur et la version de Python, il pourrait afficher `In[1]` *au lieu de* `>>>`.

Dans cette "console Python", vous pouvez entrer et exécuter des commandes. Vous terminerez chaque commande par la touche *Entrée*. Entrez successivement les commandes `1+1`, puis `print("Bonjour !")`, `x=5`, et enfin `x**2`. Lorsque les commandes s'exécutent, l'affichage devient

```
>>> 1+1
2

>>> print("Bonjour !")
Bonjour !

>>> x=5

>>> x**2
25
```

Chaque ligne commençant par `>>>` affiche une commande que vous avez entrée. Certaines commandes ont renvoyé un résultat (par exemple `1+1` a renvoyé `2`) qui a été affiché dans la ligne suivante. D'autres commandes ont effectué une action : `print("Bonjour")` a demandé d'afficher le mot "Bonjour", tandis que `x=5`, a demandé d'enregistrer le nombre 5 comme étant la valeur de la variable `x`. La fenêtre « workspace » (une des fenêtres de spyder) permet de constater, après avoir exécuté cette commande, que la variable `x` est de type `int` (c'est un entier) et que sa valeur est `5`. Enfin, la notation `**` désigne en python les puissances. Ainsi, la commande `x**2` demande de calculer x^2, sachant qu'on a déjà spécifié que x vaut 5. Ce calcul renvoi évidemment le résultat `25`.

1.3 Utilisation de l'éditeur

Téléchargez le fichier `http://jaramillo.perso.math.cnrs.fr/Courses/MaSC4A/TD1.py` et ouvrez le (avec spyder), il apparaîtra dans la fenêtre de gauche. Le fichier est constitué de commandes en langage Python, entrecoupées de commentaires, et séparées en blocs appelés cellules. Pour créer de tels blocs, il suffit d'introduire des lignes commençant par `#%%`. Les touches Ctrl+Entrée permettent d'exécuter un tel bloc d'un seul coup. L'exécution des commandes se fait encore dans la console Python, mais les résultats de commandes ne s'affichent pas (sauf si l'on utilise la fonction `print` pour obliger à l'afficher).

1.4 Types de données

Les variables que l'on crée (par exemple la variable `x` définie ci avant) ont, comme tout objet manipulé par python, un « type », qui indique quelles opérations on peut faire.

Par exemple `"Bonjour"` est une chaîne de caractère (de type `str`) alors que `3` est un entier (de type `int`). L'addition et la multiplication n'ont pas du tout le même sens pour ces deux types d'objets, comme on peut le constater en exécutant

```
print("Bonjour "+3*"!")
print(2+3*9)
```

Listes De même constatez ce qu'il se passe en exécutant la commande `[17]+[2]`. Les valeurs entourées de crochets (comme `[17]`, ou `[17,2]`) désignent des « listes » : par exemple `[1]` désigne "la liste qui n'a qu'un seul élément, dont l'élément est égal à `1`".

Pour ces listes, le symbole `+` désigne de fait de coller bout à bout des listes.

Nombres à virgule de type *float* Par défaut, les nombres à virgule en python sont de type `float`, comme on peut le constater en exécutant par exemple :

```
a=2+10**-5
print("a est de type ",type(a)," et vaut ",a)
```

Toutefois ces nombres gardent en mémoire peu de chiffres après la virgule, conduisant parfois à des erreurs, comme ci-dessous :

```
2+10**-20-2
```

L'ordinateur obtient `0.0` au lieu de 10^{-20}. La raison est qu'il calcule d'abord `2+10**-20`, c'est à dire 2,00000000000000000001. N'étant pas configuré pour conserver assez de chiffres après la virgule, il arrondi cela à 2,0. Puis il soustrait 2 et obtient donc que $2+10^{-20}-2 \simeq 0$ au lieu de trouver que cela vaut 10^{-20}.

Nombres décimaux Pour avoir une précision suffisante, il faut demander à Python de garder beaucoup de chiffres après la virgule dans ses calculs. Pour cela on peut par exemple utiliser le module "`decimal`", qui contient des fonctions permettant de manipuler des nombres ayant de nombreux chiffres après la virgules. Exécutez le code suivant et constatez la différence :

```
from decimal import Decimal
print(Decimal(2)+Decimal(10)**-20)
print(Decimal(2)+Decimal(10)**-20-2)
```

Le nombre `Decimal(10)` se comporte comme le nombre 10, mais Python sait que quand il doit faire des calculs avec ce nombre, il doit garder beaucoup de chiffres après la virgule.

La ligne `from decimal import Decimal` signifie que parmi les nombreuses fonctions définies dans le module `decimal` on souhaite uniquement *importer* (c'est à dire donner accès à) la fonction `Decimal`.

Configuration du module *decimal* Pour spécifier combien de chiffres conserver après la virgule en utilisant le module `decimal`, on peut utiliser les fonctions `getcontext` et `setcontext` comme ci-dessous :

```
from decimal import getcontext, setcontext, Context
setcontext(Context(prec=120))
```

En tapant `getcontext()` dans la console, vous pouvez afficher la configuration du module `decimal`, sous la forme d'une liste de paramètres. Le premier est `prec=...` qui indique le nombre de chiffres conservés après la virgule.

Par exemple, la commande `setcontext(Context(prec=120))` impose la valeur 120 pour ce paramètre, donc elle fixe à 120 le nombre de chiffres après la virgule utilisés en manipulant les nombres crées avec la commande `Decimal`.

2 Dichotomie

2.1 Principe : tâtonnement

Le principe de la dichotomie consiste à formaliser et automatiser la notion intuitive de « tâtonnement », dont les premières étapes sont, pour cet exemple :

- On note tout d'abord que $f(1) = -1 < 0$ et que $f(2) = 2 > 0$.
  On conclue que $\sqrt{2} \in [1\,;\,2]$
- On cherche ensuite à déterminer dans quelle moitié de l'intervalle se trouve la racine : est-elle plus grande ou plus petite que 1,5 ? On calcule donc $f(1{,}5) = 0{,}25$.
  C'est positif donc la racine est plus petite que 1,5.
  On conclue que $\sqrt{2} \in [1\,;\,1{,}5]$
- Pour savoir si la racine est plus grande ou plus petite que 1,25, on calcule $f(1{,}25) = -0{,}4375$.
  On conclue que $\sqrt{2} \in [1{,}25\,;\,1{,}5]$
- On continue le procédé jusqu'à avoir obtenu un intervalle suffisamment petit.

2.2 Formalisation par récurrence

De manière plus mathématique, on définit deux suites $(x_n)_{n\in\mathbb{N}}$ et $(y_n)_{n\in\mathbb{N}}$ par :

- $x_0 = 1$, $y_0 = 2$
- Pour chaque $n \in \mathbb{N}$,
  si $f(\frac{x_n+y_n}{2}) > 0$ alors $\begin{cases} x_{n+1} = x_n \\ y_{n+1} = \frac{x_n+y_n}{2} \end{cases}$ et sinon $\begin{cases} x_{n+1} = \frac{x_n+y_n}{2} \\ y_{n+1} = y_n \end{cases}$

On montrera en exercice qu'avec ces définitions, on a à chaque étape $\sqrt{2} \in [x_n\,;\,y_n]$, et que les deux suites $(x_n)_{n\in\mathbb{N}}$ et $(y_n)_{n\in\mathbb{N}}$ convergent toutes les deux vers $\sqrt{2}$.

2.3 Implémentation

Le code ci-dessous calcule les premières valeurs $(x_0,x_1,\ldots,x_{10})$ et $(y_0,y_2,\ldots,y_{10})$, et affiche x_{10} et y_{10} :

```
def f(x):
    return x**2-2
x=[1.0];y=[2.0]
for n in range(0,10):
    if f((x[n]+y[n])/2)>0:
        x=x+[x[n]]
        y=y+[(x[n]+y[n])/2]
    else:
        x=x+[(x[n]+y[n])/2]
        y=y+[y[n]]
print("x[10]=",x[10])
print("y[10]=",y[10])
```

Remarques

- Au cours du calcul, la variable `x` est une liste. Elle contient les valeurs déjà calculées de la suite. `x[0]` désigne le premier élément de la liste (donc x_0), `x[1]` le deuxième (donc x_1), etc. Comme vous pouvez le constater dans l'explorateur de variables, `x` et `y` contiennent chacun 11 valeurs à la fin du calcul.
- La commande

  ```
  def f(x):
      return x**2-2
  ```

  définit la fonction `f` : $x \mapsto x^2 - 2$. Vous pouvez constater dans la console qu'en tapant `f(1)`, on calcule un nombre qui est bien égal à $f(1) = -1$.
- La ligne `x=[1.0];y=[2.0]` initialise les listes `x` et `y`, et spécifie ainsi que `x[0]` vaut `1.0` et que `y[0]` vaut `2.0`. Par la même occasion, on a indiqué à python que l'on ne va pas manipuler des entiers, mais plutôt des nombres à virgule (de type `float`).
- `range(0,10)` désigne la liste `[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]`, qui commence à 0 et s'arrête juste avant 10.
- `for n in range(0,10):` spécifie que les lignes suivantes seront répétées pour chaque valeur `n` prise dans la liste `range(0,10)`.
- L'indentation (l'espacement au début de chaque ligne) permet à python de savoir quelle instruction est à l'intérieur ou à l'extérieur d'une boucle ou d'une condition "`if`". Ainsi, l'ensemble du bloc

  ```
  if f((x[n]+y[n])/2)>0:
      x=x+[x[n]]
      y=y+[(x[n]+y[n])/2]
  else:
      x=x+[(x[n]+y[n])/2]
      y=y+[y[n]]
  ```

  est exécuté à chaque itération de la boucle `for n in range(0,10)`. En revanche les deux dernières lignes ne sont exécutées qu'après avoir fini de répéter cette boucle.
- La commande `x=x+[...]` (resp `y=y+[...]`) ajoute une valeur à la liste `x` (resp `y`). Dans ce code, cela ajoute la valeur x_{n+1} (resp y_{n+1}) calculée à partir de la formule de récurrence qui définit les suites $(x_n)_{n\in\mathbb{N}}$ et $(y_n)_{n\in\mathbb{N}}$.

Remarque : il existe aussi une autre opération sur les listes dont on pourrait avoir besoin : l'extraction d'une sous-liste :

Par exemple `x[0:5:2]` désigne la liste `[x[0],x[2],x[4]]` (on commence par l'indice 0, on s'arrête juste avant 5 et on va de deux en deux). Pour d'autres exemples, vous pouvez exécuter les commandes `"anticonstitutionnellement"[3:-5:3]` et `[1,5,3,9][-2:]`.

2.4 Représentation graphique

On peut représenter graphiquement les valeurs de (x_n) et (y_n) que l'on vient de calculer, en exécutant le code suivant :

```
from matplotlib import pyplot
pyplot.plot(range(11),x,"g+")
pyplot.plot(range(11),y,"r.")
pyplot.show()
```

La première ligne importe `pyplot` à partir du module `matplotlib`. Cela définit notamment la fonction `pyplot.plot`, qui permet de positionner des points sur un graphique. Ainsi `pyplot.plot(range(11),x,"g+")` positionne des points symbolisés par des croix vertes de la forme "+", et dont les abscisse sont les éléments de `range(11)` (c'est à dire les nombres 0, 1, ..., 10) et les ordonnées sont les éléments de `x`. De même la ligne suivante positionne des points dont les ordonnées sont les éléments de `y`. Enfin la commande `pyplot.show()` affiche le résultat.

2.5 Exercices

1. Prouvez que, pour tout $n \in \mathbb{N}$, on a $y_n - x_n = 1/2^n$. Prouvez ensuite que (x_n) est croissante et (y_n) décroissante, et en déduire que les deux suites x_n et y_n convergent et ont la même limite.

   Montrer que $\forall n \in \mathbb{N}, f(x_n) \leq 0 < f(y_n)$ et en déduire que la limite des suites x_n et y_n est bien $\sqrt{2}$.
2. En utilisant cette méthode de dichotomie, calculez les 10 premières décimales de $\sqrt{2}$.

3. En utilisant cette méthode de dichotomie, calculez les 100 premières décimales de $\sqrt{2}$.

   Astuce : Vous aurez peut-être besoin d'utiliser le module `decimal` pour obtenir une précision suffisante.

4. Calculez les 10 premières décimales de la racine réelle positive du polynôme $X^4+3X^3+12X-2018$, en utilisant toujours cette même méthode.

3 Méthode de la sécante

Plutôt de choisir à chaque étape le milieu de l'intervalle précédent (méthode par dichotomie), la méthode de la sécante consiste à estimer la position de la racine à partir des valeurs calculées précédemment :

On définit par récurrence la suite x_n définie par $x_0 = 1$, $x_1 = 2$, et pour $n \geq 1$, $x_{n+1} = x_n - \frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})} f(x_n)$.

le code ci-dessous calcule les premières valeurs x_0, x_1, x_2, …, x_5 :

```
def f(x):return x**2-2
x=[1.0,2.0]
fx=[f(1.0),f(2.0)]
for n in range (1,5):
    x=x+[x[n]-(x[n]-x[n-1])/(fx[n]-fx[n-1])*fx[n]]
    fx=fx+[f(x[n+1])]
print(x)
```

3.1 Exercices

1. Représenter à main levée la fonction f et la relation de récurrence. Montrer que x_{n+1} est l'abscisse de l'intersection entre l'axe des abscisses d'une part, et d'autre part la droite passant par les points de coordonnées $(x_{n-1}, f(x_{n-1}))$ et $(x_n, f(x_n))$. En déduire pourquoi cette méthode s'appelle "méthode de la sécante".
2. Calculez les premières valeurs x_0, x_1, …, x_{15}.
   Astuce : vous pourriez avoir besoin de préciser à Python de faire ses calculs avec de nombreux chiffres après la virgule.
3. Montrer que si x_n tends vers $\sqrt{2}$, alors $|x_n - \sqrt{2}| \sim |\frac{x_n^2-2}{2\,x_n}|$.

   En conséquence, on pose $\delta_n = |\frac{x_n^2-2}{2\,x_n}|$, qui permet d'estimer l'erreur $|x_n - \sqrt{2}|$.
4. Représenter graphiquement la précision δ_n en fonction du nombre d'itérations.
5. Comparer la vitesse de convergence de cette méthode avec celle de la méthode par dichotomie :
   (a) Rappeler la définition du symbole $\mathcal{O}(\dots)$. Montrer que pour la méthode de dichotomie, $n = \mathcal{O}(\ln \delta_n)$. En déduire qu'il existe $a \in \mathbb{R}^+$ tel qu'à partir d'un certain rang, $\delta_n \leq e^{-a\,n}$. Quelle valeur peut prendre a ?
   (b) Montrer graphiquement que pour la méthode de la sécante, δ_n tends plus vite vers 0 et on a $n = \mathcal{O}(\ln(-\ln \delta_n))$ c'est à dire qu'il existe $a \in \mathbb{R}^+$ tel qu'à partir d'un certain rang, $\delta_n \leq e^{-e^{a\,n}}$.
   Pour éviter des complication causées par les calculs de nombres très petits, on se contentera de représenter les 10 premières valeurs de δ_n.
   Pouvez-vous déterminer approximativement quelle valeur peut prendre ce nombre a ?

4 Méthode de Newton

Plutôt que de considérer la sécante, on considère désormais la tangente à la fonction f au point x_n.

4.1 Exercice

1. Montrer que l'on doit alors considérer la relation de récurrence $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$.
2. Calculer les 10 premières valeurs prises par cette suite. Comparer graphiquement la marge d'erreur avec les deux méthodes précédentes.

Remarque : (on ne ne vous demande pas de le démontrer, mais) les formules de Taylor permettent de montrer que $\ln \delta_n = \mathcal{O}(2^n)$, de sorte que l'on a $\delta_n \leq e^{-e^{a\,n}}$ pour toute valeur a telle que $a < \ln 2$.

Remarque historique : La même relation de récurrence $x_{n+1} = x_n - \frac{x_n^2-2}{2x_n}$ a été proposée dès l'antiquité par Héron d'Alexandrie pour calculer $\sqrt{2}$ sur la base d'arguments géométriques (sans connaître la notion de dérivée).

L'argument géométrique est que le rectangle le côtés x_n et $y_n = 2/x_n$ une surface égale à 2, et pour se rapprocher d'un carré ayant cette aire, on peut considérer un côté de $x_{n+1} = \frac{x_n+y_n}{2}$ (et l'autre de $y_{n+1} = 2/x_{n+1}$). Il se trouve que pour cet exemple, cela coïncide avec la relation de récurrence obtenue par la méthode de Newton.