

Feuille de TP Python 06

Suites récurrentes et algorithmes numériques

1 Introduction

L'objet de ce TP est de travailler le thème des suites récurrentes, de nombres, de vecteurs, de matrices ou d'objets plus généraux...

Il permet, d'une part de passer en revue des thèmes classiques du calcul en y présentant quelques unes des méthodes les plus élémentaires et d'autre part d'aborder des thèmes de probabilité ou d'autres domaines.

1.1 Mathématiquement

Résumons mathématiquement la situation abstraite. On dispose d'un ensemble X d'objets (des nombres, des matrices, etc..) et d'une méthode de transformation de ces objets, *a.k.a* ¹ une application $\phi : X \rightarrow X$.

En prenant $x_0 \in X$, le principe de récurrence affirme l'existence d'une unique suite d'objets de X , $(x_n)_{n \in \mathbb{N}}$ vérifiant

$$\forall n \in \mathbb{N}, x_{n+1} = \phi(x_n)$$

1.2 Informatiquement

C'est ce principe général que nous allons implémenter informatiquement. Les mathématiques nous permettent d'affirmer l'existence immédiate d'une infinité d'objets, ce que l'informatique ne permet pas. Il va donc falloir nous limiter à ne calculer qu'un nombre fini d'objets x_n .

Il faut décider pour chaque problème à traiter si on ne conserve que le dernier objet calculé ou toute la suite partielle calculée.

Une telle suite partielle sera placée dans une `list` au sens Python. Le tableau 4.4 en annexe récapitule quelques commandes, fonctions et méthodes pour gérer les listes.

Des codes Python typiques pour cela sont donc

```
#La fonction f effectuant la récurrence
# et l'objet initial X0 préexistant
#n donné, on calcule Xn
Xn=X0
for i in range(n): #n tours de boucle
    Xn=f(Xn)
#A ce point Xn est le terme d'indice n
```

```
#La fonction f effectuant la récurrence
# et l'objet initial X0 préexistant
#n donné, on calcule X=[X0,.., Xn]
X=[X0] #La liste qui contient le résultat
for i in range(n): #n tours de boucle
    Xi=f(X[-1]) #X[-1] dernier élément de X
    X.append(Xi)
#A ce point X est une liste indexée de 0 à n
```

2 Exemples élémentaires

2.1 L'algorithme de Babylone

L'algorithme de Babylone est la méthode employée pour calculer « à la main » des racines carrées. Soit $A = a^2$ un nombre dont on recherche la racine carrée. L'algorithme de Babylone est la récurrence

$$x_0 \text{ donné, } \neq 0, \forall n \in \mathbb{N}, x_{n+1} = \frac{1}{2} \left(x_n + \frac{A}{x_n} \right).$$

On démontre (on le fera en TD) que si A un nombre réel strictement positif, si $x_0 \in \mathbb{R}$, $x_0 \neq 0$ alors, lorsque $n \rightarrow +\infty$,

$$x_n \rightarrow \begin{cases} +|a| & \text{si } x_0 > 0 \\ -|a| & \text{si } x_0 < 0 \end{cases}$$

1. "also known as"

1. Programmer cet algorithme dans une fonction `RacineCarree(A, x0=1, n=10)` pour calculer une racine carrée d'un nombre pour un nombre $A = A$ donné à l'avance en calculant x_n , le terme d'indice $n = n$ de la suite décrite précédemment partant de $x_0 = x_0$. Tester sur un certain nombre d'exemples et comparer les résultats avec ceux de la fonction `Numpy/Python np.sqrt`.
- 5 Ecrire une fonction `RacineCarreeSuite(A, x0=1, n=10)` retournant en liste les termes $x_0, \dots, x_n, n = n$, de la suite décrite précédemment partant de $x_0 = x_0$ pour un nombre $A = A$ donné en paramètre. Partant de $A = -1$ et $x_0 = 1.111111$, tracer le graphe $n \mapsto x_n$ de cette suite (les indices n en abscisses, les termes de la suite x_n en ordonnée). Y-a-t'il une convergence à conjecturer?
- 10 (Ouvert) Que se passe-t-il si l'on part de A nombre complexe non réel, par exemple $A = 1 + 3.i$? Rappel : en Python cette affectation se fait par `A=1+3j`

2.2 Une suite intervenant en dynamique des populations

Soit $p \in]0, 1[$, $d \in \mathbb{N}^*$. On considère la suite récurrente donnée par

$$p_0 \in]0, 1[, \forall n \in \mathbb{N}, p_{n+1} = (1 - p + p \cdot p_n)^d$$

Cette suite récurrente est liée à la probabilité de disparition d'une population dont chaque femelle f engendre, indépendamment des autres, à l'étape n , $X_{n,f}$ rejetons femelles suivant une loi $\mathcal{B}(d, p)$ avant de mourir.

- 15 Ecrire une fonction Python `ProbaDisparition(n, p0=0, p=0.5, d=2)` calculant/retournant la liste des termes p_0, \dots, p_n calculés suivant cette récurrence.
- Ecrire une fonction Python `Grphe(p=0.5, d=2)` traçant le graphe de la fonction itérée ainsi la droite d'équation $y = x$ en repère orthonormé.
- Effectuer quelques expériences en graphant la suite par dessus le graphe précédent. Conjecturer la convergence de cette suite et la position de la limite par rapport au point fixe 1.
- 20 Indication: Distinguer les cas $p \cdot d > 1$ (auquel cas la probabilité de disparition tend vers une limite < 1) $p \cdot d = 1$ (auquel cas la limite est 1).
- (Facultatif, à faire quand tout le reste est fini).
 - 25 (a) Ecrire une fonction effectuant la simulation du modèle probabiliste proposé sur N générations en partant d'une population ayant une femelle de départ et retournant `True` si la population a disparu avant la génération N , `False` sinon.
 - (b) En effectuant un grand nombre de tirages, évaluer la probabilité de disparition et comparer avec la limite de la suite (p_n) précédemment obtenue.

2.3 Travail demandé

Traiter l'un des deux exemples proposés (Questions 1 (fichier `babylone.py`) ou 2. (fichier `proba-extinction.py`)).

3 Recherche de solutions d'équations

Utiliser le script `solutionfxy.py`.

Décompressez l'archive `.zip` associée à ce TD, ouvrez le fichier `python/solutionfxy.py` dans Spyder ou Pyzo/IEP. Sous Pyzo/IEP, faites de `python/` votre répertoire de travail en exécutant le script avec `SHIFT-CTRL-E`

On s'intéresse au problème suivant : Etant donné une fonction f réelle de variable réelle, définie par exemple par une

35 formule, un nombre $y \in \mathbb{R}$, on cherche à trouver **une** ² solution de l'équation $f(x) = y$ d'inconnue $x \in \mathbb{R}$.

On propose deux méthodes : la méthode par dichotomie et la méthode `NEWTON`. Cette dernière est plus complexe mais beaucoup plus rapide.

On va écrire des méthodes génériques, *i.e.* la fonction f fait partie des arguments de la méthode lors de son appel. On va par ailleurs placer ces méthodes dans le module `solutionfxy.py`.

4.0 3.1 Dichotomie

La méthode de dichotomie pour résoudre l'équation $f(x) = y$ d'inconnue x se décrit de la manière suivante.

On construit deux suites $(a_n)_{n \in \mathbb{N}}$ et $(b_n)_{n \in \mathbb{N}}$ par récurrence.

1. a_0 et b_0 sont donnés au départ et vérifient $(f(a_0) - y)(f(b_0) - y) \leq 0$,
2. connaissant a_n et b_n , on calcule leur moyenne c et
 - 4.5 (a) si $(f(c) - y)(f(b_n) - y) \leq 0$, on pose $a_{n+1} = c, b_{n+1} = b_n$,
 - (b) si $(f(a_n) - y)(f(c) - y) \leq 0$, on pose $a_{n+1} = a_n, b_{n+1} = c$,

2. toutes, c'est beaucoup plus compliqué

Théorème 1. Si

1. Les nombres $a, b \in \mathbb{R}$, $a < b$ sont donnés ;
2. La fonction $f : [a, b] \rightarrow \mathbb{R}$ est continue sur l'intervalle $[a, b]$;
3. Le nombre $y \in \mathbb{R}$ est entre $f(a)$ et $f(b)$,

5 alors

1. les suites $(a_n)_{n \in \mathbb{N}}$ et $(b_n)_{n \in \mathbb{N}}$ construites par la méthode de dichotomie convergent vers une limite commune que l'on note c . On a

$$\forall n \in \mathbb{N}, a_n \leq c \leq b_n \text{ et } |b_n - a_n| = \frac{|b - a|}{2^n}$$

2. Le nombre c est une solution de l'équation $f(x) = y$ d'inconnue $x \in [a, b]$.

3.2 Méthode de NEWTON

La méthode de NEWTON de résolution d'équation du type $f(x) = 0$ se décrit de la manière suivante.

On suppose que

- 10 1. il existe une solution de cette équation, notée x_∞ ,
2. La fonction f est de classe \mathcal{C}^1 sur un voisinage de x_∞ ,
3. f' ne s'annule pas sur un voisinage de x_∞ .

On prend x_0 « suffisamment proche » de x_∞ et on construit par récurrence la suite $(x_n)_{n \in \mathbb{N}}$ par

$$\forall n \in \mathbb{N}, x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Géométriquement, cela revient à prendre pour x_{n+1} l'abscisse du point d'intersection entre l'axe des abscisses et la tangente au graphe de f en x_n . On a plus précisément le théorème de convergence (HP)

15 **Théorème 2.** Dans les conditions exposées précédemment, il existe $1 > \varepsilon > 0$, $K > 0$ (dépendants de f), tels que si

1. $K \cdot |x_0 - x_\infty| < \varepsilon$

alors

1. la suite $(x_n)_{n \in \mathbb{N}}$ est bien définie, elle converge vers x_∞ ,
2. avec l'estimation

$$\forall n \in \mathbb{N}, K \cdot |x_{n+1} - x_\infty| \leq (K \cdot |x_n - x_\infty|)^2 \leq (K \cdot |x_0 - x_\infty|)^{2^{n+1}}$$

3.3 Travail demandé

20 1. Dichotomie

- (a) Compléter le fichier `solutionfxy.py` donné en annexe en écrivant le corps de la fonction `dichotomie` suivant les spécifications indiquées. Exécuter le fichier pour faire les tests prévus.
- (b) On considère, pour chaque $n \in \mathbb{N}$, $n \geq 4$, l'équation $x^n - \frac{n}{2}x^2 + 1 = 0$ d'inconnue $x \in [0, 1]$. On admet (c'est un exercice assez facile) que cette équation admet une unique solution α_n qui vérifie de plus $\sqrt{2} \cdot n^{-\frac{1}{2}} \leq \alpha_n \leq 2 \cdot n^{-\frac{1}{2}}$.

25 Ecrire, dans le fichier `td06.py`, un programme résolvant cette équation pour $4 \leq n \leq N = 100$ en utilisant la fonction `dichotomie` définie dans le module `solutionfxy.py`. On veut que le programme trace le graphe de la suite des nombres $(\alpha_n)_{4 \leq n \leq N}$. Exécutez ! Faire en sorte que les résultats des tests du module `solutionfxy.py` n'apparaissent plus lors de l'exécution de `td06.py` et reexécuter.

2. NEWTON.

- 30 (a) Corriger l'algorithme théorique pour qu'il résolve une équation du type $f(x) = y$ et l'implémenter dans `newton1d`. Décommenter les lignes de tests pour `newton1d` et exécuter. Les résultats sont-ils cohérents avec ceux obtenus par dichotomie ?
- (b) Compléter le fichier `solutionfxy.py` donné en annexe en écrivant le corps de la fonction `newton1d_liste` suivant les spécifications indiquées. Exécuter le fichier pour faire les tests prévus en décommentant les lignes de tests prévus.
- 35 (c) Reprendre le fichier `td06.py` et compléter le programme pour qu'il résolve aussi l'exercice en utilisant la fonction `newton1d` définie dans le module `solutionfxy.py`

4 Suites récurrentes et EDO

4.1 Problème de CAUCHY et schéma d'EULER

Une équation différentielle ordinaire se présente sous la forme

$$\frac{dX}{dt} = F(t, X) \quad (*)$$

et résoudre un problème de CAUCHY associé à l'équation (*) c'est

1. se donner un « instant initial » t_0 , une valeur initiale X_0 ,
2. déterminer un intervalle I de \mathbb{R} non trivial contenant t_0
3. déterminer la(les) fonction(s) $X : t \in I \mapsto X(t)$, de classe $\mathcal{C}^1(I)$, telle(s) que
 - (a) $X(t_0) = X_0$
 - (b)

$$\forall t \in I, X'(t) = F(t, X(t))$$

Une telle EDO est l'analogie continu d'une suite récurrente, ce qui peut se comprendre du point de vue de la physique.

1. On pose un réel positif, Δt , un « petit accroissement » de la variable t , le pas du schéma d'EULER. On peut par exemple prendre $\Delta t = \frac{1}{N}$ où N est un entier naturel.
2. On obtient alors une suite de points $(t_k)_{k \in \mathbb{Z}}$ donnée par $t_k = t_0 + k \cdot (\Delta t)$.
3. l'EDO (*) et le problème de CAUCHY associé à (*) et une valeur initiale X_0 ont pour analogie la détermination d'un segment d'entiers relatifs K contenant 0, d'une suite $(X_k)_{k \in K}$ telle que

$$\forall k \in K, X_{k+1} = X_k + (\Delta t)F(t_k, X_k) \text{ (si } k+1 \in K)$$

Appliquer le schéma d'EULER pour résoudre l'EDO (*) et le problème de CAUCHY associé, c'est calculer une suite vérifiant cette récurrence et espérer que X_k est une « bonne approximation » de $X(t_k)$.

Une étape de l'algorithme d'EULER, prolongeant les listes X et t déjà construites se traduit donc par

```
X0=X[-1]
t0=t[-1]
X1=X0+dt*F(t0,X0)
t1=t0+dt
X.append(X1)
t.append(t1)
```

4.2 EDO du second ordre

Une EDO linéaire, à coefficients constants, du second ordre est de la forme (et nous prenons maintenant t pour variable temporelle et x la quantité, **réelle ou complexe**, fonction de t)

$$\frac{d^2x}{dt^2} + a_1 \frac{dx}{dt} + a_0 \cdot x = f \quad (**)$$

où f est fonction de t . Le problème de CAUCHY pour une telle équation s'énonce de façon similaire à celle du problème de CAUCHY associé à (*) en imposant **deux conditions initiales** à l'instant t_0 : $x(t_0) = x_0$ et $x'(t_0) = y_0$ pour certains $x_0, y_0 \in \mathbb{R}$.

Une telle EDO se transforme en une EDO du premier ordre du type (*) pourvu que l'on considère X prenant ses valeurs dans \mathbb{R}^2 . Si on introduit $y = \frac{dx}{dt}$, (**) se transforme en un système d'EDO

$$\begin{cases} \frac{dx}{dt} = y \\ \frac{dy}{dt} = -a_1 \cdot y - a_0 \cdot x + f \end{cases}$$

Si on pose

$$X = \begin{pmatrix} x \\ y \end{pmatrix}$$

Le système s'écrit matriciellement

$$\frac{dX}{dt} = A \cdot X + \begin{pmatrix} 0 \\ f \end{pmatrix} \text{ où } A = \begin{pmatrix} 0 & 1 \\ -a_0 & -a_1 \end{pmatrix}$$

La condition initiale du problème de CAUCHY associé à cette équation matricielle du type (*) est $X(t_0) = X_0 = \begin{pmatrix} x_0 \\ y_0 \end{pmatrix}$ et le schéma d'EULER peut s'appliquer.

4.3 Travail demandé

Utiliser le script `euler.py`.

Ouvrez le fichier `python/euler.py` dans Spyder ou Pyzo/IEP. Sous Pyzo/IEP, faites de `python/` votre répertoire de travail en exécutant le script avec SHIFT-CTRL-E

- 5 1. Ecrire le corps de la fonction `euler` suivant les spécifications demandées dans sa documentation. Il s'agit d'une fonction générique permettant de construire un schéma d'EULER à partir de n'importe quelle fonction F .
2. Décommenter les tests à partir de `print("EULER")`. Exécuter. Que font ces tests ?
3. Compléter ces tests pour qu'ils résolvent l'équation

$$\frac{d^2x}{dt^2} + a_1 \frac{dx}{dt} + a_0 \cdot x = 0$$

pour des valeurs de a_0 , a_1 et des conditions initiales reflétant la diversité des situations. (Solutions exponentielles, périodiques, oscillantes amorties...)

- 10 4. Dans la fonction `ED01in2` définie dans la batterie de tests
 - (a) Faire en sorte que, graphiquement, on puisse comparer les solutions numériques aux solutions abstraites.
 - (b) Inventer et mettre en oeuvre un indicateur permettant une comparaison numérique entre les solutions approchées et les solutions réelles et placer une impression du résultat en fin de fonction.

4.4 Listes

<code>L=[]</code> ou <code>L=list()</code>	fabrique une liste vide à remplir
<code>L=[1.0, 'aa']</code>	fabrique une liste avec un préremplissage
<code>len(L)</code>	donne le nombre d'éléments de la liste, ils sont indicés de 0 à <code>len(L)-1</code>
<code>L[i]</code>	donne la valeur de l'élément de L d'indice i si $0 \leq i \leq \text{len}(L) - 1$
<code>M=L[i:j]</code>	extraite une sous-liste M dont les éléments ceux de L sont indicés de i à $j-1$
<code>L[i:j]=M</code>	remplace la sous-liste des éléments de L indicés de i à $j-1$ par la liste M
<code>M=L</code>	Les deux étiquettes M et L sont sur la même liste en mémoire
<code>M=L[:]</code> ou <code>M = L.copy()</code>	créé une copie de L et lui affecte l'étiquette M
<code>L[-1]</code>	donne le dernier élément de la liste
<code>L.append(obj)</code>	ajoute l'objet <code>obj</code> en fin de liste
<code>L.extend(M)</code>	fusionne les listes L et M en plaçant M à la fin de L
<code>L.insert(index,obj)</code>	insère l'objet <code>obj</code> avant l'indice i , décale le reste de L
<code>L[i:i] = M</code>	insère la liste M à l'indice i de la liste L, décale le reste de L
<code>L.pop()</code>	efface le dernier élément de la liste L et le retourne
<code>L.pop(i)</code>	efface l'élément i de la liste L et le retourne
<code>L.reverse()</code>	inverse <i>en place</i> l'ordre des éléments de la liste
<code>L.sort()</code>	trie <i>en place</i> les éléments de la liste, pourvu que l'on puisse comparer les éléments
<code>M=sorted(L)</code>	M devient une copie triée de la liste L
<code>V=np.array(L)</code> ou <code>V=np.asarray(L)</code>	V devient un vecteur numpy composé avec les éléments de la liste de nombres L. Si L est une liste de listes de nombres, compose une matrice numpy.
<code>for obj in L :</code> Exemple : <pre>for k in L: print(k**2)</pre>	boucle sur les éléments de L, ne modifie pas L. Modifier L dans la boucle peut avoir des conséquences inattendues. Si on prend <code>L=[1, 2, 4]</code> affiche successivement 1, 4, 16.
<code>M = [expression(obj) for obj in L]</code> Exemple : <code>M=[k+1 for k in L]</code>	fabrique une liste en bouclant sur les éléments de L et en leur appliquant <code>expression()</code> , ne modifie pas L. Si on prend <code>L=[1,2,4]</code> fabrique la nouvelle liste <code>[2,3,5]</code> et l'affecte à M.
<code>M = [expr(obj) for obj in L if cond(obj)]</code> Exemple : <code>M=[k**2 for k in L if k>1]</code>	fabrique une liste en bouclant sur les éléments de L, ne gardant que ceux vérifiant <code>cond()</code> et en leur appliquant <code>expr()</code> , ne modifie pas L. Si on prend <code>L=[1, 2, 4]</code> fabrique la nouvelle liste <code>[4, 16]</code> et l'affecte à M.