TP n°2 : équations différentielles, fonctions

PTSI Lycée Eiffel

20 octobre 2017

1 Fonctions en Python

1.1 Lecture et analyse de programmes

Pour chacun des programmes suivants, **AVANT** de les exécuter sous Python, lisez le programme, cherchez à comprendre ce qu'il fait, et prévoyez ce que renverra $f(10)$ ainsi que $f\left(\frac{1}{2}\right)$, puis vérifiez vos hypothèses en recopiant les programmes et en les exécutant.

Première fonction

```
def f(x) :
    y=3*x
    z=y*x
    return(z-y)
```

Deuxième fonction

```
def f(x) :
    if x < x*x :
        return x
    else :
        return x*x
```

Troisième fonction

```
def f(x) :
    a=0
    s=0
    while s<x :
        a=a+1
        s=s+a
    return a
```

1.2 Quelques petits exemples pour commencer à programmer vous-même

1. Programmer une fonction en Python prenant comme paramètres un nombre x et un entier n et calculant x^n (on pourra évidemment programmer une boucle à l'intérieur de la définition de la fonction).
2. Programmer une fonction en Python prenant comme paramètre un entier n et calculant le n-ème terme de la suite de Fibonacci (définie, rappelons-le, par les conditions initiales $F_0 = 0$ et $F_1 = 1$ et par la relation de récurrence $F_{n+2} = F_{n+1} + F_n$).
3. Programmer une fonction en Python prenant comme paramètre un entier n et calculant la somme des chiffres de n (en base 10, bien entendu).

1.3 Pour s'entraîner avec la récursivité

Reprendre chacun des trois exercices précédents, mais en écrivant à chaque fois une fonction récursive. Dans le cas du calcul de puissance, réfléchir à une version vraiment intelligente, ne calculant pas x^n sous la forme $x \times x^{n-1}$ mais plutôt sous la forme $x^{\frac{n}{2}} \times x^{\frac{n}{2}}$ (du moins si n est pair). On essaiera même d'estimer le nombre de multiplications nécessaires pour calculer x^n. Dans le cas de la suite de Fibonacci, on testera le programme récursif sur des valeurs de n de l'ordre de quelques dizaines, et on se demandera pourquoi c'est beaucoup plus lent que la version non récursive.

2 Résolution d'équations différentielles d'ordre 1

Le but de cette première partie est d'arriver à résoudre de façon approchée des équations différentielles du premier ordre, du type $f'(t)+a(t)f(t) = b(t)$, et à tracer à l'aide du module matplolib.pyplot une allure de la courbe représentative de la fonction f. Pour celà on va créer une fonction Python prenant comme paramètres les valeurs suivantes : la valeur ti correspondant au temps initial à partir duquel on veut résoudre l'équation, la valeur tf jusqu'à laquelle on va résoudre l'équation (autrement dit, on résout sur l'intervalle $[ti, tf]$), la valeur fi correspondant à la condition initiale $f(ti)$, ainsi qu'un entier n indiquant le nombre d'étapes de calcul qu'on va effectuer. La procédure que nous allons utiliser fonctionne en effet de la façon suivante :

- on crée une liste **abscisse** de valeurs temporelles comportant exactement $n+1$ valeurs régulièrement espacées dont la première est ti et la dernière tf. Ainsi, si on résout sur $[0,1]$ avec $n = 10$, la liste doit être $[0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1]$.
- On crée ensuite une liste **ordonnee** qui ne contient initialement que la valeur fi. On va ensuite compléter de proche en proche la liste avec des valeurs approchées de $f(x_i)$, où les réels x_i sont ceux apparaissant dans la liste abscisse crée ci-dessus. Pour cela, on utilisera l'approximation $f'(x_i) \simeq \dfrac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i}$, avec par ailleurs par définition $f'(x_i) = b(x_i) - a(x_i)f(x_i)$. Autrement dit, on calculera $f(x_{i+1}) \simeq f(x_i) + h * (b(x_i) - a(x_i)f(x_i))$, où on a noté $h = x_{i+1} - x_i$ (valeur constante tout au long de l'exécution de l'algorithme).
- On trace ensuite la courbe obtenue à partir des deux listes abscisse et ordonnee.

1. Appliquer cette méthode pour résoudre l'équation $f' = f$ sur l'intervalle $[0,5]$ avec $fi = 1$ et $n = 10$. Modifier ensuite la valeur de n pour l'augmenter, et comparer les courbes obtenues.
2. Tracer simultanément à vos courbes approchées la courbe de la vraie solution (que vous devez normalement reconnaitre).
3. Résoudre de façon approchée l'équation différentielle $f'+f = K$, où K est une constante qu'on pourra faire varier, avec la condition initiale $fi = 0$ (on choisira un intervalle de résolution adapté).

3 Un problème plus détaillé

La suite de Syracuse est définie par récurrence de la façon suivante : u_0 est un entier naturel supérieur ou égal à 1, et pour tout entier n, on pose $u_{n+1} = \dfrac{1}{2}u_n$ si u_n est pair (attention, c'est bien la valeur u_n qui doit être paire et non n), et $u_{n+1} = 3u_n + 1$ si n est impair. On conjecture que, quelle que soit la valeur de u_0, on finit par obtenir un entier n pour lequel $u_n = 1$ (ensuite, la suite devient périodique de période 3 et de valeurs successives 1, 4 et 2). Pour un entier n_0 donné (correspondant à la valeur de u_0), on définit le vocabulaire suivant :

- l'orbite de n_0 est la liste de toutes les valeurs prises par la suite avant d'atteindre 1.
- son temps de vol est le nombre d'étapes nécessaires avant d'atteindre 1.
- son altitude est la plus grande valeur prise par la suite.

- son temps de vol en altitude est le nombre d'étapes nécessaire avant de passer pour la première sous la valeur initiale.

1. Écrire la liste des premiers termes de la suite lorsque $n_0 = 13$ (on le fera à la main, pour vérifier ensuite les différentes programmes écrits).
2. Écrire une fonction Python **orbite(n)** qui renvoit l'rbite de l'entier n (on doit donc afficher à l'arrivée une liste de valeurs de la suite.
3. Écrire trois fonctions **tempsdevol(n)**, **altitude(n)**, **tempsenaltitude(n)** affichant les valeurs correspondantes.
4. Déterminer, parmi les entiers inférieurs ou égaux à un million, celui qui a la plus haute altitude (et donner l'altitude correspondante), celui qui a le plus long temps de vol, puis celui qui a le plus long temps de vol en altitude.