Préparation à l'agrégation interne
TP Python

B. Boutin - benjamin.boutin@univ-rennes1.fr

17 février 2020

1 Introduction à Python

- Installer au préalable une distribution et un environnement de développement pour Python, par exemple Pyzo.
- Possibilité d'utiliser un environnement python en ligne (avec notebook Jupyter) : https://jupyter.org/try

1.1 Rudiments

Variables et types

- entiers `int`
- flottants `float`
- booléens `bool`
- chaînes de caractères `string`
- tuples `tuple` (non-modifiables)
- listes `list`

Fonctions

- Usage compact:

```
g = lambda x: x*2
```

- Usage standard:

```
def nom_de_fonction(arg1,arg2)
    """Description de la fonction"""
    INSTRUCTIONS
    return out1,out2,out3
```

- Usage avancé : packing d'un nombre arbitraire de paramètres dans un tuple

```
def product(*args):
    """Produit d'un nombre arbitraire d'arguments"""
    y = 1.
    for x in args[:]:
        y = y*x
    return y

product(1,2,3,4)
    24.0
```

Structures de contrôles : `for`, `if`, `while`

- boucle `for`

```
for element in liste:
    INSTRUCTIONS
```

- test `if` (`elif`) `else`

```
if CONDITION1:
    INSTRUCTIONS1
elif CONDITION2:
    INSTRUCTIONS2
elif CONDITION3:
    INSTRUCTIONS3
else:
    INSTRUCTIONS4
```

- boucle `while`

```
while CONDITION:
    INSTRUCTIONS
```

Exercice : définition de fonction

Définir une fonction qui à $x \in \mathbb{R}$ associe $x^2 + 1$ si $x \geq 0$ et $\sqrt{-x}$ sinon.

Exercice : calcul des n premiers termes de la suite de Fibonacci

Programmer une fonction `fibonacci` prenant en argument l'entier n et renvoyant la liste des termes $u_0, \ldots, u_n$ de la suite de Fibonacci:

$$u_0 = 1,\ u_1 = 1, \forall n \in \mathbb{N},\ u_{n+2} = u_{n+1} + u_n.$$

Exercice : calcul du pgcd de plusieurs nombres

Programmer la fonction suivante, qui calcule le pgcd de deux entiers a et b.

```
def euclide(a, b):
    """Calcul de pgcd de a et b via l'algorithme d'Euclide"""
    if a < b:
        b,a = a,b
    while b != 0:
        a,b = b,a%b
    return a

euclide(5**3*7**2,9*7*5)
```

Étant donnés trois entiers a, b, c, on peut obtenir leur pgcd en utilisant la propriété suivante:

$$\text{pgcd}(a, b, c) = \text{pgcd}(\text{pgcd}(a, b), c).$$

De cette manière, on peut calculer le pgcd d'un nombre arbitraire d'entiers par exemple en utilisant le principe de packing présenté précédemment.

Exercice : recherche d'un zéro

Par une méthode de dichotomie, déterminer un zéro de la fonction $x \mapsto \cos x - x$ à 10^{-9} près.

Programmer la méthode de Newton pour la recherche de ce même zéro.

1.2 Utilisation des bibliothèques `numpy` et `matplotlib`

Documentation pour les bilbiothèques utilisées:

- https://docs.scipy.org/doc/numpy/reference
- https://docs.scipy.org/doc/scipy/reference
- https://matplotlib.org
- https://docs.sympy.org

Recherche de zéro (suite)

Après en avoir parcouru la documentation, reprendre l'exercice précédent en utilisant les fonctions `scipy.optimize.bisect` et `scipy.optimize.newton`.

Exercice

Tracer la courbe cardioïde paramétrée suivante

$$\begin{cases} x(t) = -\cos(t) + \cos(t)^2 \\ y(t) = -\sin(t) + \cos(t)\sin(t) \end{cases}$$

et sa tangente au point de paramètre $t = \pi/2$, puis l'ensemble des tangentes aux points de paramètres $s = 2k\pi/10$ pour $k = 0, \ldots, 9$.

2 Intégration numérique

On souhaite calculer une approximation numérique de l'intégrale suivante:

$$I = \int_0^3 \sin(10x + e^x)dx.$$

Pour ce faire, on emploie la méthode des trapèzes décrite ci-après. Étant donné un entier naturel non-nul n, on doit évaluer la quantité:

$$I_n = \frac{3}{n}\sum_{i=0}^{n-1} \frac{f(x_i) + f(x_{i+1})}{2},$$

où $x_i = 3i/n$ pour $0 \leq i \leq n$.

Compléter le code suivant pour mener ce calcul à bien :

```
def f(x):
    y = ...
    return y

def Trapeze(f,n):
    S = 0
    h = 3/n
    x,y = 0,h
    for i in range(0,n):
        S = S + ...
        x,y = y,y+h
    return S

print(Trapeze(f,100))
```

Étude de convergence

Estimer l'erreur commise dans l'intégration de la fonction précédente par la méthode des trapèzes pour différentes valeurs de l'entier n. En guise de valeur de référence pour l'intégrale I, on utilisera l'approximation obtenue par la commande intégrée quad de la bibliothèque `scipy.integrate`.

```
from scipy import integrate
I = integrate.quad(f,0,3)[0]
print(I)
```

Illustrer graphiquement (par un tracé en échelle logarithmique) la majoration suivante:

$$\exists C > 0,\ \forall n \in \mathbb{N}^\star,\ |I_n - I| \leq \frac{C}{n^2}.$$

On pourra utiliser le code suivant qui met en œuvre le calcul de l'erreur $|I_n - I|$ pour différentes valeurs de n et représente l'évolution de cette erreur en échelle logarithmique.

```
Err = []
ListN = np.arange(1,200)
for n in ListN:
    In = Trapeze(f,n)
    Err.append(np.abs(In-I))

plt.plot(np.log(ListN),np.log(Err))
plt.plot(np.log(ListN),-2*np.log(ListN))
plt.title('Courbe de convergence')
plt.xlabel('log n')
plt.ylabel('log erreur')
plt.show()
```

Méthode de Simpson

Reprendre le code précédent pour utiliser maintenant la méthode de quadrature de Simpson :

$$J_n = \frac{3}{n} \sum_{i=0}^{n-1} \frac{1}{6} \left(f(x_i) + f(\tfrac{x_i + x_{i+1}}{2}) + f(x_{i+1}) \right).$$

De nouveau, estimer numériquement l'ordre de convergence effectif, c'est-à-dire la plus grande constante $\alpha > 0$ telle que

$$\exists C > 0,\ \forall n \in \mathbb{N}^\star,\ |J_n - I| \leq \frac{C}{n^\alpha}.$$

3 Équations différentielles

Afin de résoudre des équations différentielles ordinaires, on peut utiliser la fonction `odeint` de la bibliothèque Scipy.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import odeint
```

Voici un exemple simple pour la résolution de l'équation logistique

$$y'(t) = r y(t) \left(1 - \frac{y(t)}{K} \right).$$

Dans les simulations, on pourra par exemple utiliser les paramètres $r = 1.5$ et $K = 6$ ainsi que la donnée initiale $y_0 = 1$.

```
def f(y,t):
    return 1.5*y*(1.-y/6.)

y0 = 1.0
t = np.linspace(0,5,201)
sol = odeint(f,y0,t)

plt.plot(t,sol)
plt.axis('equal')
plt.show()
```

Modèle proie-prédateur

Adapter l'exemple précédent de sorte à résoudre le modèle proie-prédateur de Lotka-Volterra:

$$\begin{cases} x'(t) = +x(t)(a - by(t)) \\ y'(t) = -y(t)(c - dx(t)) \end{cases}$$

On choisira pour les simulations les paramètres $a = 2$, $b = 1$, $c = 1$ et $d = 0.3$ et pour ce qui concerne la donnée initiale : $x(0) = 1$, et $y(0) = 1$. La solution sera évaluée sur l'intervalle de temps $[0, 30]$. À l'aide de la fonction "quiver" de "matplotlib" décrite dans la documentation, tracer le portrait de phase du système différentiel, ainsi que la trajectoire solution vue comme la courbe paramétrée $t \mapsto (x(t), y(t))$.

Attracteur de Lorenz

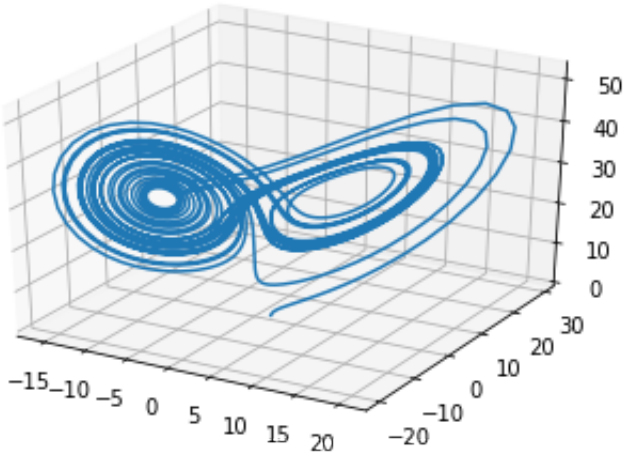
Résoudre numériquement quelques trajectoires pour le système dynamique de l'attracteur de Lorenz en dimension 3 donné par

$$\begin{cases} x'(t) = \sigma(y(t) - x(t)) \\ y'(t) = \rho x(t) - y(t) - x(t)z(t) \\ z'(t) = x(t)y(t) - bz(t). \end{cases}$$

On retiendra les paramètres $\sigma = 10$, $\rho = 30$ et $b = 2$. Les trajectoires pourront être calculées sur l'intervalle de temps $t \in [0, 10]$ pour les données initiales $(20, 0, 0)$, $(10, 0, 0)$ et $(0.1, 0, 0)$.

Pour le tracé des trajectoires en dimension 3, la commande suivante permet un tracé de courbe paramétrée en dimension 3 d'espace. Il faut préalablement avoir stocké le vecteur `sol` dans un `ndarray` à 3 colonnes.

```
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure()
ax = pl.axes(projectio='3d')
ax.plot3D(sol[:,0],sol[:,1],sol[:,2])
plt.show()
```



4 Interpolation de Lagrange

Theorème. *Soient f une fonction à valeurs réelles définie sur un intervalle $I = [a, b]$ et $n + 1$ points deux à deux distincts $a_0, a_1, \ldots, a_n$ de l'intervalle I. Il existe un unique polynôme P_n de degré au plus n vérifiant*

$$\forall i \in \{0, \ldots, n\}, \;\; p_n(a_i) = f(a_i).$$

Ce polynôme d'interpolation de Lagrange s'obtient dans la base des polynômes de Lagrange associée aux points $a_0, \ldots, a_n$

$$P_n(x) = \sum_{i=0}^{n} f(a_i) L_i(x), \quad L_i(x) = \prod_{j \neq i} \frac{x - a_j}{a_i - a_j}.$$

Pour le choix des points a_i, on peut retenir entre autres possibilités:

- les points équidistants de l'intervalle I: $a_i = a + i(b-a)/(n+1)$, $i \in \{0, \ldots, n\}$;
- les points de Chebyshev : $a_i = \frac{a+b}{2} + \frac{b-a}{2} \cos\left(\frac{(2i+1)\pi}{2n+2}\right)$, $i \in \{0, \ldots, n\}$.

Exercice :

Définir une fonction `Lagrange(a,i,x)` qui prend en argument le tuple (ou la liste) des points `a` et renvoie la valeur $L_i(x)$.
Définir une fonction `Interpolation(f,a,x)` qui renvoie la valeur $P_n(x)$.

Exercice : le phénomène de Runge

Soient I l'intervalle $I = [-1, 1]$ et f la fonction définie sur I par

$$f(x) = \frac{1}{1 + 25x^2}.$$

À l'aide des fonctions précédemment définies, construire et visualiser le polynôme d'interpolation de Lagrange pour $n = 6$ obtenu dans le cas des points équidistants ainsi que pour les points de Chebyshev.

5 Séries de Fourier

Ayant noté $[x]$ la partie entière d'un réel x, on considère la fonction 1-périodique

$$f(x) = e^{x - [x]}.$$

Après quelques longs calculs, on obtient les sommes partielles de la série de Fourier de f, données par

$$S_N(x) = (e-1) + \sum_{n=1}^{N} \frac{2(e-1)}{1 + 4\pi^2 n^2} \cos(2\pi n x) - \sum_{n=1}^{N} \frac{4\pi n (e-1)}{1 + 4\pi^2 n^2} \sin(2\pi n x).$$

Sur une même figure, comparer les graphes de f, S_3, S_5 et S_9.

Pour la série de Fourier de $g(x) = |\sin(\pi x)|$ sur l'intervalle $[0, 1]$:

$$R_N(x) = \frac{2}{\pi} - \sum_{n=1}^{N} \frac{4}{\pi(4n^2 - 1)} \cos(2\pi n x),$$

examiner numériquement la vitesse de convergence de l'erreur $\sup_{[0,1]} |g - R_N|$ en fonction de N.

Remarque : quelques possibilités de calcul symbolique sont proposées par la bibliothèque `sympy`:

```
from sympy import fourier_series, pi
from sympy.abc import x
s = fourier_series(x**2,(x,-pi,pi))
s.truncate(n=4)
```