Informatique
TP3 : Calcul numérique d'une intégrale
CPP 1A

Djamel Aouane, Frédéric Devernay, Matthieu Moy

mars 2015

1 Zéro de fonction

On donne le code suivant (vu en cours), qui permet de calculer le zéro d'une fonction f par dichotomie (f est définie au début du code) :

```
import math

def f(x):
    return math.sin(x) - 0.5

def zero_dichotomie(a, b, epsilon):
    while b - a > epsilon:
        pivot = (a + b) / 2
        value = f(pivot)
        if value <= 0:
            a = pivot
        else:
            b = pivot
    return a

print("Solution :", math.pi / 6)
print("Approximation à 0.00001 pres :",
      zero_dichotomie(-1, math.pi / 2, 0.00001))
print("Erreur :",
      math.pi / 6 - zero_dichotomie(-1, math.pi / 2, 0.00001))
```

Cet algorithme ne marche que sous certaines hypothèses : $a \leq b$, $f(a) \leq 0$, $f(b) > 0$, et f doit être continue sur $[a, b]$.

Exercice 1 *Vérifiez que l'exemple fonctionne bien quand les hypothèses sont vérifiées.*

Pour comprendre le comportement de `zero_dichotomie`, on pourra utiliser le débogueur (cf. TP2) ou afficher les valeurs des variables à chaque itération (ce qui est recommandé pendant l'écriture d'un programme, mais la version finale de la fonction doit seulement retourner la valeur finale et surtout ne rien imprimer).

Exercice 2 *Essayez une fonction f pour laquelle $f(a)$ et $f(b)$ sont de même signe (par exemple, $f(x) = x^2 + 1$). L'algorithme ne fonctionne pas dans ce cas. Modifiez votre fonction pour rejeter ce cas explicitement. On peut utiliser* `raise RuntimeError("raison de l'erreur")` *permet d'arrêter brutalement l'exécution du programme avec un message d'erreur.*

Dans le cas où $f(a) > 0$ et $f(b) \leq 0$, une astuce est d'inverser les valeurs de a et b avant de démarrer le calcul. L'algorithme de recherche par dichotomie marche si $b < a$ à un détail près : la condition d'arrêt de la boucle doit maintenant porter sur la valeur absolue de $b - a$, et non sur $b - a$ (qui serait toujours négatif). En Python, la valeur absolue de `x` est obtenue avec `abs(x)`.

Exercice 3 *Testez l'algorithme pour le cas où $f(a) > 0$ et $f(b) \leq 0$. Modifiez-le en échangeant a et b si besoin pour qu'il marche sur ce cas.*

Exercice 4 *Que se passe-t-il si la fonction a deux zéros ou plus entre a et b ? Si elle n'est pas monotone ? Si elle n'est pas continue (p.ex. $f(x) = 1/x$ avec $a = -2, b = 1$) ? Pour ces cas, on ne peut pas tester simplement les propriétés mathématiques de la fonction, donc on ne peut pas rejeter explicitement ces cas : c'est la responsabilité de l'utilisateur d'appeler la fonction* `zero_dichotomie` *correctement.*

2 Listes en python

Une liste en python est une suite de valeurs. Les éléments de la liste peuvent être de types quelconques (entiers, flottants, chaînes de caractères), et sont repérés par leur position dans la liste, aussi appelé index. Le premier élément d'une liste `a` a l'index 0, et sa valeur peut être obtenue par l'expression `a[0]`. Le nombre d'éléments d'une liste s'obtient avec la fonction `len()`. Le dernier élément de la liste `a` a pour index `len(a)-1`, et pour valeur `a[len(a)-1]`, mais on peut y accéder également en utilisant un index négatif : `a[-1]` est le dernier élément, `a[-2]` l'avant-dernier, etc. Les éléments d'une liste peuvent être *modifiés*.

On peut ajouter un élément `13.5` à la fin d'une liste `a` avec `a.append(13.5)`. Exemple :

```
>>> a = [ 1.5, 7.2, 10., 0.7]
>>> a
[1.5, 7.2, 10.0, 0.7]
>>> len(a)
4
>>> a[2]
10.0
>>> a[4]
Traceback (most recent call last):
  File "<pyshell#27>", line 1, in <module>
    a[4]
IndexError: list index out of range
>>> a[-1]
0.7
>>> a[3] = 13.5
>>> a
[1.5, 7.2, 10.0, 13.5]
>>> a.append(7.8)
>>> a
[1.5, 7.2, 10.0, 13.5, 7.8]
```

Affectation et passage comme paramètre : Si `a` est une liste, alors l'instruction `b=a` *ne recopie pas la liste*. `a` et `b` désignent alors la même liste (on parle d'*alias*). Une modification de la liste désignée par `b` modifie donc la liste désignée par `a`. Il en est de même lorsqu'on passe une liste comme paramètre d'une fonction, le paramètre effectif est un alias vers la liste définie par l'appelant de la fonction : si on modifie la liste à l'intérieur de la fonction, alors la liste de l'appelant est modifiée.

```
>>> a = [0, 1, 2, 3]
>>> b = a
>>> b
[0, 1, 2, 3]
>>> b[2] = 42
>>> b
[0, 1, 42, 3]
>>> a
[0, 1, 42, 3]
>>> def f(l):
    l[3] = 67
>>> f(a)
>>> a
[0, 1, 42, 67]
```

Utilisation avancée : (non nécessaire pour le TP) Des éléments d'une même liste peuvent avoir des types différents. On peut utiliser une liste pour donner l'ensemble des valeurs que doit prendre la variable de boucle dans une boucle `for` : dans la boucle `for i in [1,3.14,'abc']:`, il y aura 3 itérations, et la variable i prendra successivement les valeurs `1`, `3.14`, et `'abc'`. D'autres fonctionalités liées aux listes sont décrites dans `https://docs.python.org/3/tutorial/datastructures.html`

3 Méthode des trapèzes

Le calcul numérique d'une intégrale par la méthode des trapèzes consiste à calculer une valeur approchée de l'intégrale $\int_a^b f(x)dx$ en utilisant une interpolation linéaire de f par intervalles.

Sur un intervalle $[a, b]$, la valeur de l'intégrale peut être approchée par la surface du trapèze passant par les points $[a, 0], [a, f(a)], [b, f(b)], [b, 0]$:

$$\int_a^b f(x)dx \approx (b-a)\left[\frac{f(a)+f(b)}{2}\right]$$

On montre que cette méthode est d'ordre 1 (tout comme la méthode du point milieu vue en cours), c'est-à-dire qu'elle donne un résultat exact si f est un polynôme de degré au plus 1 (la méthode de Simpson est d'ordre 3).

Pour calculer une meilleur approximation de cette intégrale, on découpe l'intervalle $[a, b]$ en n intervalles plus petits, de taille $\frac{b-a}{n}$, et on calcule la somme des aires (signées) des trapèzes :

$$\int_a^b f(x)dx \approx \frac{b-a}{2n}\sum_{k=1}^{n}\left[f\left(a+(k-1)\frac{b-a}{n}\right)+f\left(a+k\frac{b-a}{n}\right)\right]$$

Exercice 5 *Écrivez tous les termes de la somme ci-dessus pour $n = 3$, et observez qu'il est possible de faire le calcul avec exactement $n + 1 = 4$ appels à la fonction f.*

Écrire une fonction python `integrale_trapezes(a, b, n)` *qui calcule l'intégrale de la fonction* `f(x)` *de a à b avec n intervalles, en faisant au plus $n + 1$ appels à la fonction f.*

Exercice 6 *Pour $f(x) = x \sin(x^2)$, calculez l'erreur numérique de son intégrale sur l'intervalle $[0, \sqrt{\pi}]$, pour 2 à 30 intervalles (voir la solution pour tracer le graphe de l'erreur).*

On souhaite maintenant calculer une intégrale par la méthode des trapèzes à partir d'une liste de mesures physiques : on dispose d'un appareil mesurant la vitesse d'un mobile à des instants donnés, et on souhaite calculer sa position en intégrant la vitesse par la méthode des trapèzes.

Les dates et les vitesses sont données par les listes `dates` et `vitesse`. La première date est toujours 0 (`dates[0] == 0.`), et les dates sont strictement croissantes. Les deux listes ont le même nombre d'éléments : `vitesse[2]` est la vitesse à la date `dates[2]`. On suppose que la position du mobile à la date $t = 0$ est $x = 0$.

Exercice 7 *Écrivez la fonction python* `position_mobile_index(dates, vitesses, index)` *qui calcule la position du mobile à la date* `dates[index]` *en intégrant la vitesse par la méthode des trapèzes (inutile d'optimiser les accès à la liste* `vitesses` *comme à la question précédente, l'accès à un élément d'une liste est moins coûteux que le calcul d'une fonction compliquée).*

Vérifiez que `position_mobile_index(dates, vitesses, 0)` *vaut toujours 0.*

Exercice 8 *Écrivez la fonction python* `index_date(dates, t)` *qui renvoie l'index* `i` *de la dernière date telle que* `dates[i]<=t`.

La fonction doit fonctionner même si `t > dates[-1]` *(la dernière date).*

Exercice 9 *Écrivez la fonction python* `position_mobile_date(dates, vitesses, t)` *qui calcule la position du mobile à la date* `t` *en intégrant la vitesse par la méthode des trapèzes.*

On suppose que `0 <= t < dates[-1]` *(la dernière date).*

Remarquez qu'on peut utiliser `index_date` *et* `position_mobile_index` *pour calculer presque toute l'intégrale, et qu'on doit seulement traiter séparément le cas du dernier trapèze. Pour traiter le cas du dernier trapèze, commencez par calculer la vitesse à la date t par interpolation linéaire.*

Données de test :

```
d = [0., 3., 3.5, 5., 10.]
v = [1., 0., 1.5, 2.,  0.]
position_mobile_index(d, v, 0) = 0
position_mobile_index(d, v, 1) = 1.5
position_mobile_index(d, v, 2) = 1.875
index_date(d, 0.) = 0
index_date(d, 1.) = 0
index_date(d, 3.) = 1
index_date(d, 3.5) = 2
index_date(d, 5.) = 3
index_date(d, 6.) = 3
position_mobile_date(d, v, 0.) = 0.0
position_mobile_date(d, v, 1.) = 0.8333333333333334
position_mobile_date(d, v, 3.) = 1.5
position_mobile_date(d, v, 3.5) = 1.875
position_mobile_date(d, v, 5.) = 4.5
position_mobile_date(d, v, 6.) = 6.3
```

4 Solution des exercices

Exercices 2 et 3 : Ajouter au début de la fonction `zero_dichotomie` :

```
fa = f(a)
fb = f(b)
if fa * fb > 0: # f(a) et f(b) de même signe
    raise RuntimeError("f(a) et f(b) sont de meme signe, impossible de trouver un zero")
if fa > 0:
    # f decroissante: echange a et b
    c = b
    b = a
    a = c
```

et remplacer la condition du while par

```
while abs(b - a) > epsilon:
```

Exercice 5 : En posant $\delta = \frac{b-a}{n}$, la formule peut se ré-écrire :

$$\begin{aligned}
\int_a^b f(x)dx &\approx \frac{\delta}{2}\sum_{k=1}^{n}\left[f\left(a+(k-1)\delta\right)+f\left(a+k\delta\right)\right] \\
&= \frac{\delta}{2}\left[\sum_{k=0}^{n-1} f\left(a+k\delta\right)+\sum_{k=1}^{n} f\left(a+k\delta\right)\right] \\
&= \frac{\delta}{2}\left[f(a)+\sum_{k=1}^{n-1} f\left(a+k\delta\right)+\sum_{k=1}^{n-1} f\left(a+k\delta\right)+f(b)\right] \\
&= \delta\left[\frac{f(a)}{2}+\sum_{k=1}^{n-1} f\left(a+k\delta\right)+\frac{f(b)}{2}\right]
\end{aligned}$$

Intuitivement, dans la somme, on fait apparaître successivement le bord droit d'un trapèze (divisé par deux) et le bord gauche du trapèze suivant (divisé par deux également). En groupant ces termes, on obtient un seul bord de trapèze, qui n'est plus divisé par deux. Le code Python correspondant est :

```
def integrale_trapezes(a, b, N):
    res = f(a) / 2
    delta = (b - a) / N
    for i in range(1, N):
        res = res + f(a + i * delta)
    res = res + f(b) /2
    return res * delta
```

Note : comme pour `zero_dichotomie`, on la fonction `integrale_trapezes` suppose qu'on a défini une fonction `f`. Pour exécuter la fonction `integrale_trapezes`, il faudrait également un programme principal qui appelle cette fonction, par exemple, `print(integrale_trapezes(0, 3.5, 100))`.

Exercice 6 :

```
def f(x):
    return x*math.sin(x*x)

# Solution exacte de l'intégrale calculée à la main :
SOLUTION = 1.0

def erreur(n):
    return integrale_trapezes(0., math.sqrt(math.pi), n) - SOLUTION
```

```
for n in range(2,31):
    print("erreur(",n,"):", erreur(n))

# bonus: tracé de la fonction
import pylab

n = range(2, 31)
s = list(map(erreur,n))
pylab.plot(n, s)
pylab.show()
```

Note : `list(map(erreur,n))` construit une liste en appliquant la fonction erreur à chaque élément de `n`.

Exercice 7 :

```
def position_mobile_index(dates, vitesses, index):
    res = 0
    for i in range(index):
        res = res + (dates[i+1]-dates[i]) * (vitesses[i]+vitesses[i+1]) / 2
    return res
```

Exercice 8 : On donne ici la solution avec une recherche linéaire. Les dates étant croissantes, on pourrait aussi aller plus vite avec une recherche par dichotomie.

```
def index_date(dates, t):
    i = 0
    l = len(dates)
    while i < l and dates[i] <= t:
        i = i + 1
    # ici, soit i == l, soit dates[i] > t
    return i - 1  # on doit retourner l'index precedent, t.q. dates[i] <= t
```

Exercice 9 :

```
def position_mobile_date(dates, vitesses, t):
    # precondition: t < dates[-1]
    i = index_date(dates, t)
    res = position_mobile_index(dates, vitesses, i)
    # ici, on a dates[i] <= t < dates[i+1]
    # Dernier trapèze : x va de dates[i] à t, y va de vitesses[i] à lastv
    lastv = vitesses[i]+(vitesses[i+1]-vitesses[i])*(t-dates[i])/(dates[i+1]-dates[i])
    return res + (t-dates[i]) * (vitesses[i] + lastv) / 2.
```