

INFO-H-100 - Programmation

TP 6 Calcul Itératif

Lorsque l'exercice demande d'écrire une fonction, écrivez la fonction demandée et testez-la avec plusieurs valeurs pertinentes.

Cette séance portant sur les boucles `while`, veuillez à ne pas utiliser de boucle `for`.

Ex. 1. Nous supposons que `range` n'existe pas. Ecrire une fonction `my_range(a, b)` qui produit une liste ordonnée contenant tous les entiers dans $[a, b[$.

Ex. 2. Ecrire une fonction `my_range_step(a, b, step)` qui produit une liste ordonnée contenant tous les entiers dans $[a, b[$ séparés de `step`. On considère que $a < b$ et que `step` est strictement positif.

Ex. 3. Ecrire une fonction `linspace` qui renvoie une liste des n réels équidistants entre *begin* et *end* **inclus**.

```
>>> linspace(2,6,21) #begin = 2, end = 6, n = 21
[2.0, 2.2, 2.4, 2.6, ..., 5.4, 5.6, 5.8, 6.0]
```

Ex. 4. Ecrire une fonction qui calcule la valeur approchée de π sur base de la série suivante (due à Euler). Les calculs s'arrêtent lorsque la valeur du dernier terme ajouté est inférieure à 10^{-6} ou lorsque le nombre de termes considérés atteint 10000.

$$\frac{\pi^2}{6} = 1 + \frac{1}{2^2} + \frac{1}{3^2} + \frac{1}{4^2} + \dots$$

Ex. 5. Ecrire une fonction qui calcule la valeur approchée de π sur base de la série suivante. Les calculs s'arrêtent lorsque la valeur du dernier terme ajouté est inférieure à un ϵ donné (c'est à dire en variable globale) ou lorsque le nombre de termes considérés atteint une borne également donnée.

$$\frac{\pi}{8} = \frac{1}{1 \times 3} + \frac{1}{5 \times 7} + \frac{1}{9 \times 11} + \dots$$

Ex. 6. Ecrire une fonction qui calcule la valeur approchée de π sur base de la série suivante (due à Leibniz). Les calculs s'arrêtent lorsque la valeur du dernier terme ajouté est inférieure à un ϵ donné (c'est à dire en variable globale) ou lorsque le nombre de termes considérés atteint une borne également donnée.

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \dots$$

Ex. 7. Ecrire une fonction qui calcule la valeur approchée de π sur base de la série suivante. Les calculs s'arrêtent lorsque la valeur du dernier terme ajouté est inférieure à un ϵ donné, ou lorsque le nombre de termes considérés atteint une borne également donnée.

$$\frac{\pi}{4} = 4 \cdot \arctan\left(\frac{1}{5}\right) - \arctan\left(\frac{1}{239}\right)$$

avec la série :

$$\arctan(x) = \frac{x}{1+x^2} \left(1 + \frac{2}{3} \cdot \frac{x^2}{1+x^2} + \frac{2}{3} \cdot \frac{4}{5} \cdot \left(\frac{x^2}{1+x^2}\right)^2 + \dots \right) = \frac{x}{1+x^2} \sum_{i=0}^{\infty} c_i \left(\frac{x^2}{1+x^2}\right)^i$$

où $c_0 = 1$

et pour $i > 0$, $c_i = \frac{2 \cdot i}{2 \cdot i + 1} c_{i-1}$

Ex. 8. Deux suites (x_i) et (y_i) sont définies par

$$\begin{aligned} y_0 &= 2 \\ x_0 &= 1 \\ x_{i+1} &= \frac{2}{y_{i+1}} \\ y_{i+1} &= \frac{y_i + x_i}{2} \end{aligned}$$

Nous admettons que les suites convergent toutes les deux vers $\sqrt{2}$, avec de plus $x_i < \sqrt{2} < y_i$. Ecrire une fonction `rac2` qui calcule deux approximations supérieure et inférieure de $\sqrt{2}$, de différence inférieure à ε (prévoyez aussi d'arrêter les calculs si le nombre d'étapes dépasse une borne fixée).

Ex. 9. Soit r un nombre réel positif. La suite de nombres réels x_n est spécifiée par :

$$\begin{aligned} x_1 &= 1, \\ x_{n+1} &= \frac{1}{2} \left(x_n + \frac{r}{x_n} \right), \end{aligned}$$

Cette suite, obtenue par une méthode générale, due à Newton, converge très rapidement vers \sqrt{r} .

Ecrire une fonction qui calcule la valeur approchée de \sqrt{r} . L'exécution s'arrêtera lorsque la différence entre deux approximations successives sera inférieure à un ε donné.

Ex. 10. Soit r un nombre réel positif. Deux suites de nombres réels x_n et y_n sont spécifiées par :

$$\begin{aligned} x_1 &= 1, \\ y_1 &= 1, \end{aligned}$$

et pour $n \geq 1$:

$$\begin{cases} x_{n+1} = x_n + r \cdot y_n \\ y_{n+1} = x_n + y_n \end{cases}$$

Il n'est pas difficile de prouver $y_n \neq 0$ pour $n \geq 1$, et aussi que le quotient $\frac{x_n}{y_n}$ converge vers \sqrt{r} .

Ecrire une fonction qui calcule la valeur approchée $\frac{x_n}{y_n}$ de \sqrt{r} , où n est la plus petite valeur pour laquelle la condition suivante est vraie :

$$(x_n)^2 < (r + \varepsilon)(y_n)^2 \text{ et } (x_n)^2 > (r - \varepsilon)(y_n)^2.$$

(ε étant comme d'habitude une valeur donnée). L'exécution sera interrompue si le nombre d'étapes devient trop grand.

Ex. 11. La méthode d'Euler est une procédure numérique pour résoudre par approximation des équations différentielles du premier ordre avec condition initiale.¹

Soit une équation différentielle ordinaire de la forme

$$u'(t) = f(u(t)), \quad u(0) = u_0$$

où u est l'inconnue, f est une fonction donnée et u_0 la condition initiale également donnée. Supposons que nous voulons calculer une solution approchée de l'équation différentielle pour les $n \geq 1$

1. Une équation différentielle est une équation formulée en terme de relations entre des fonctions, typiquement une fonction et ses dérivées. Le but est de déterminer la fonction. Par exemple, pour l'équation différentielle $u''(t) = -u(t)$ où l'inconnue est la fonction u , les solutions possibles sont \sin et \cos (avec des coefficients dépendant des conditions initiales). En calcul numérique, la solution d'une équation différentielle ne sera pas une expression analytique de la fonction recherchée, mais sa valeur en un certain nombre de points.

instants $t_1 \dots t_n$ de 0 à T où $T > 0$ est donné. Soit $\Delta t = T/n$ et soit u_k l'approximation de la solution exacte $u(t_k)$, la méthode d'Euler propose de calculer chaque u_k en fonction de u_{k-1} à l'aide de la formule $u_k = u_{k-1} + f(u_{k-1})\Delta t$.

Ecrire une fonction `euler` qui prend comme paramètres la fonction f , les entiers T et n et le réel u_0 et qui renvoie une paire formée de la liste des instants t_k et de la liste des valeurs u_k aux instants correspondants. Utilisez la fonction `linspace` de l'exercice 3.

```
>>> def f(u): return (2*u) - 1
>>> euler(f, 6, 12, 2.0)
([0.0, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0, 5.5, 6.0],
 [2.0, 3.5, 6.5, 12.5, 24.5, 48.5, 96.5, 192.5, 384.5, 768.5, 1536.5, 3072.5, 6144.5])
```

Par exemple, le réel $u[2]$ est égal à $u[1] + (2*u[1] - 1) * t/n$ et donc à $3.5 + (7 - 1) * 0.5$ ce qui donne 6.5.

INFO-H-100 - Programmation

TP 6 Calcul Itératif

Corrections

Solution de l'exercice 1:

```
def my_range(a,b):
    liste = []
    while (a < b):
        liste.append(a)
        a += 1

    return liste
```

Solution de l'exercice 2:

```
def my_range_step(a,b,step):
    liste = []
    while (a < b):
        liste.append(a)
        a += step

    return liste
```

Solution de l'exercice 3:

```
def linspace(begin, end, n):
    step = (end - begin) / (n - 1)
    ls = []
    x = 0
    while x < n:
        ls.append(begin + x * step)
        x += 1
    return ls
```

Solution de l'exercice 4:

```
EPS = 1E-6
NB_MAX = 10000

import math

def piEuler():
    nbTermes = 1
    terme = 1
    somme = 1
    while terme >= EPS and nbTermes < NB_MAX:
        nbTermes += 1
        terme = 1.0 / nbTermes ** 2
        somme += terme
    return math.sqrt(6*somme)
```

Solution de l'exercice 5:

```
def pi11():
    i = 1
    terme = 1.0/3
    somme = terme
    while terme >= EPS and i < NB_MAX:
        i += 1
        terme = 1.0/(4*i-3)/(4*i-1)
        somme += terme
    return 8*somme
```

Solution de l'exercice 6:

```
def piLeibnitz():
    nbTermes = 1
    signe = 1
    terme = 1
    somme = 1
    while terme >= EPS and nbTermes < NB_MAX:
        nbTermes += 1
        signe = -signe
        terme = 1.0 / (2 * nbTermes - 1)
        somme += signe*terme
    return 4*somme
```

Solution de l'exercice 7:

```
def arctan(x):
    facteur0 = x**2 / (1 + x**2)
    i = 1
    facteur1 = 1
    facteur2 = 1
    terme = 1
    somme = 1
    while terme >= EPS and i < NB_MAX:
        facteur1 *= 2.0*i / (2*i + 1)
        i += 1
        facteur2 *= facteur0
        terme = facteur1*facteur2
        somme += terme
    return facteur0/x*somme

def piArctan():
    return 4*(4*arctan(1.0/5) - arctan(1.0/239))
```

Solution de l'exercice 8:

```
def rac2():
    (x, y) = (1, 2)
    i = 0
    while y - x >= EPS and i < NB_MAX:
        y = (x + y)/2.0
        x = 2.0/y
        i += 1
    return (x, y)
```

Solution de l'exercice 9:

```
def sqrtNewton(r):
    old = 0
    x = 1.0
    while abs(x - old) >= EPS:
        old = x
        x = (x + r/x)/2.0
    return x
```

Solution de l'exercice 10:

```
def proche(x, y, r):
    x2 = x * x
    y2 = y * y
    return x2 < ((r + EPS)*y2) and x2 > ((r - EPS)*y2)

def racine(r):
    i = 1
    (x, y) = (1, 1)
    while not proche(x, y, r) and i < NB_MAX:
        (x, y) = (x + r*y, x + y)
        i += 1
    return float(x)/y
```

Solution de l'exercice 11:

```
def linspace(begin, end, n):
    step = (end - begin) / (n - 1)
```

```
ls = []
x = 0
while x < n:
    ls.append(begin + x * step)
    x += 1
return ls

def euler(f, T, n, u0):
    dt = T/float(n)
    t = linspace(0,T,n+1)
    u = []
    u.append(u0)
    for k in range(n):
        u.append(u[k] + dt*f(u[k]))
    return t, u

def f(u):
    return (2*u) - 1
```