INFO-H-100 - Programmation
TP 16 — Calcul numérique et `matplotlib`.

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

Ex. 1. `pylab` est un module python qui regroupe des librairies de calcul scientifique dont `matplotlib`, une librairie permettant d'afficher des graphiques (courbes, histogrammes, ...).

Ecrivez le code suivant dans un fichier et exécutez le avec Python 2.6.

```
import pylab

x = range(-1000,1000)
y = map(lambda u: u**2, x)

pylab.plot(x,y)
pylab.show()
```

Si vous voulez utilisez `pylab` chez vous, vous pouvez par exemple installer la distribution EPD disponible à l'adresse `http://www.enthought.com/products/epd_free.php`.

Ex. 2. A l'aide d'un `map`, d'un `lambda` et d'un `range`, écrire une fonction 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. 3. A l'aide de `matplotlib`, de la fonction précédente et d'un `map`, écrire une fonction qui dessine une fonction $f(x)$ donnée en paramètres pour tous les 10000 réels x entre 0 et 10.

```
>>> import math
>>> dessine(math.sin)
```

Ex. 4. Dans la documentation, trouvez comment choisir la couleur de chaque courbe ainsi que d'associer du texte aux axes. Tester ces fonctionnalités. Conseil : lisez les exemples sur

- `http://matplotlib.sourceforge.net/users/pyplot_tutorial.html`
- `http://matplotlib.sourceforge.net/api/pyplot_api.html#matplotlib.pyplot.plot`

Ex. 5. Ecrire une fonction qui dessine sur le même graphique les fonctions $f(x)$ passées en paramètres via une liste de fonctions (plusieurs `plot` avec un `show` en toute fin).

```
>>> import math
>>> dessineListe([math.sin, lambda x:math.cos(0.5*x)])
```

Dessiner les fonctions $sin(x)$, $2sin(x)$, $sin(2x)$ et $sin(x^2)$ pour tous les 10000 réels x entre 0 et 10.

Ex. 6. La méthode d'Euler est une procédure numérique pour résoudre par approximation des équations différentielles du premier ordre avec condition initiale.

Soit une équation différentielle ordinaire de la forme

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

où $f(u)$ 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 $n >= 1$ 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 la liste des u_k et la liste des temps t.

```
>>> euler(lambda u:2*u-1, 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.

Ex. 7. A l'aide de la fonction `euler`, dessiner des approximations de l'équation différentielle

$$u' = u^q, \quad u_0 = 1, \quad t \in [0, T]$$

avec différents paramètres : $q = 1$, $q = 2$ et $q = 3$, $\Delta t = 0.01$ et $\Delta t = 0.1$. $T = 1$ si $q = 1$ et $T = 1/(q-1) - 0.1$ sinon.

Ex. 8. A l'aide de la fonction `euler`, dessiner des approximations de l'équation différentielle

$$\frac{dy}{dx} = \frac{1}{2(y-1)}, \quad y(0) = 1 + \sqrt{\epsilon}, \quad x \in [0, 4],$$

où $\epsilon > 0$ est un très petit réel avec différents paramètres : $\Delta x = 0.1$, $\Delta x = 0.25$ et $\Delta x = 1$.

Ex. 9. Le code suivant permet d'obtenir le nombre de secondes nécessaires pour exécuter 100 fois l'appel `expo(3,1000)`.

```
>>> def expo(x,n):
        res = 1
        for i in range(n):
                res = res*x
        return res

>>> import timeit
>>> timer1 = timeit.Timer("expo(3,1000)", "from __main__ import expo")
>>> timer1.timeit(100)
0.02171802520751953
```

Voici deux façons de calculer les nombres de Fibonacci. La première, récursive, est particulièrement inefficace.

```
def fiboRec(n):
    """n >= 0"""
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fiboRec(n-1) + fiboRec(n-2)
```

```
def fiboIter(n):
    f = [0, 1]
    for i in range(2,n+1):
        f.append(f[i-1] + f[i-2])
    return f[n]
```

1. Ecrire une fonction qui renvoie la liste des temps de 10 exécutions de $fiboRecursif(n)$ pour un n variant de 1 à 15.
2. Ecrire une fonction qui renvoie la liste des temps de 10 exécutions de $fiboIteratif(n)$ pour un n variant de 1 à 15.
3. Dessiner ces deux courbes à l'aide `pylab.plot`.

Ex. 10. Dessiner et comparer les temps d'exécutions des algorithmes de tris par insertion et fusion (*merge*) pour des listes de 10 à 1000 entiers (par pas de 50) entiers choisis aléatoirement.

INFO-H-100 - Programmation
TP 16 — Calcul numérique et `matplotlib`.
Corrections

Solution de l'exercice 2:

```
def linspace(begin, end, n):
    step = (end-begin)/float(n-1)
    return map(lambda x : begin+x*step ,range(n))
```

Solution de l'exercice 3:

```
import pylab

def linspace(begin, end, n):
    step = (end-begin)/float(n-1)
    return map(lambda x : begin+x*step ,range(n))

def dessine(fct):
    x = linspace(0,10,10000)
    y = map(fct,x)
    pylab.plot(x,y)
    pylab.show()
```

Solution de l'exercice 4:

```
import pylab
import math

def linspace(begin, end, n):
    step = (end-begin)/float(n)
    return map(lambda x : begin+x*step ,range(n+1))

def testColorsAndLegend():
    x = linspace(-10,10,1000)
    y1 = map(lambda u:math.sin(u),x)
    y2 = map(lambda u:2*math.sin(2*u),x)
    pylab.plot(x, y1, 'g', label='sin(x)')
    pylab.plot(x, y2, 'r', label='2 sin(2x)')
    pylab.xlabel("x")
    pylab.ylabel("y")
    pylab.legend()
    pylab.show()
```

Solution de l'exercice 5:

```
import pylab
def linspace(begin, end, n):
    step = (end-begin)/float(n-1)
    return map(lambda x : begin+x*step ,range(n))

def dessineListe(li):
    x = linspace(0,10,10000)
    for f in li:
        y = map(f,x)
        pylab.plot(x,y)
    pylab.show()
```

Solution de l'exercice 6:

```
import pylab
import math

def linspace(begin, end, n):
    step = (end-begin)/float(n-1)
    return map(lambda x : begin+x*step ,range(n))

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

def ex():
    t,u = euler(f,6,12,2.0)
    pylab.plot(t,u)
    pylab.show()
```

Solution de l'exercice 7:

```
import pylab

def linspace(begin, end, n):
    step = (end-begin)/float(n-1)
    return map(lambda x : begin+x*step ,range(n))

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 fQ1(u):
    return u

def fQ2(u):
    return u**2

def fQ3(u):
    return u**3

def ex():
    t11,u11 = euler(fQ1,1,10,1)
    t12,u12 = euler(fQ1,1,100,1)

    t21,u21 = euler(fQ2,(1.0)-0.1,10,1)
    t22,u22 = euler(fQ2,(1.0)-0.1,100,1)

    t31,u31 = euler(fQ3,(1.0/2)-0.1,10,1)
    t32,u32 = euler(fQ3,(1.0/2)-0.1,100,1)

    pylab.plot(t11,u11)
    pylab.plot(t12,u12)
    pylab.plot(t21,u21)
    pylab.plot(t22,u22)
    pylab.plot(t31,u31)
    pylab.plot(t32,u32)
    pylab.show()
```

Solution de l'exercice 8:

```
import pylab
import math

def linspace(begin, end, n):
    step = (end-begin)/float(n-1)
    return map(lambda x : begin+x*step ,range(n))

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(y):
    return 1.0/(2*(y-1))


def ex():
    eps = 0.00001
    t1,u1 = euler(f, 4, 40, 1+eps**0.5)
    t2,u2 = euler(f, 4, 16, 1+eps**0.5)
    t3,u3 = euler(f, 4, 4, 1+eps**0.5)

    pylab.plot(t1,u1)
    pylab.plot(t2,u2)
    pylab.plot(t3,u3)

    pylab.show()
```

Solution de l'exercice 9:

```
import pylab

def fiboRecursif(n):
    """n >= 0"""
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fiboRecursif(n-1) + fiboRecursif(n-2)

def fiboIteratif(n):
    f = [0, 1]
    for i in range(2,n+1):
        f.append(f[i-1] + f[i-2])
    return f[n]

def timesFiboRec(ns):
    import timeit
    times = []
    for n in ns:
        t = timeit.Timer('fiboRecursif('+str(n)+')','from __main__ import fiboRecursif')
        times.append(t.timeit(10))
    return times

def timesFiboIter(ns):
    import timeit
    times = []
    for n in ns:
        t = timeit.Timer('fiboIteratif('+str(n)+')','from __main__ import fiboIteratif')
        times.append(t.timeit(10))
    return times

def plotTimesFibos():
    ns = range(1,16)
    t1 = timesFiboRec(ns)
    t2 = timesFiboIter(ns)
    pylab.plot(ns,t1,'r')
    pylab.plot(ns,t2,'g')
    pylab.show()
```

Solution de l'exercice 10:

```
def merge_sort(t):
    n = len(t)
    if n > 1:
        (t1, t2) = split(t)
        t1 = merge_sort(t1)
        t2 = merge_sort(t2)
        return merge(t1, t2)
    else:
        return t

def split(t):
```

```
    """ precondition: len(t) >= 2 """
    mid = len(t) / 2
    t1 = t[:mid]
    t2 = t[mid:]
    return (t1, t2)

def merge(t1, t2):
    if len(t1) == 0:
        return t2
    elif len(t2) == 0:
        return t1
    elif t1[-1] > t2[-1]:
        last = t1.pop()
        new = merge(t1, t2)
        new.append(last)
        return new
    else:
        last = t2.pop()
        new = merge(t1, t2)
        new.append(last)
        return new

def insertion_sort(v):
    n=len(v)
    for i in range(1,n):
        sauve=v[i]
        j=i-1
        while j >= 0 and v[j] > sauve:
            v[j+1]=v[j]
            j=j-1
        v[j+1]=sauve

def testMergeSort(size):
    import random
    liste = map(lambda x : random.randint(-1000,1000), [0]*size)
    merge_sort(liste)

def testInsertionSort(size):
    import random
    liste = map(lambda x : random.randint(-1000,1000), [0]*size)
    insertion_sort(liste)

def testsSorts():
    import timeit
    import pylab
    sizes = range(10,1001,50)
    timesMerge = []
    timesInsert = []

    for n in sizes:
        timer1 = timeit.Timer('testMergeSort('+str(n)+')','from __main__ import testMergeSort')
        t1 = timer1.timeit(10)
        timesMerge.append(t1)
        timer2 = timeit.Timer('testInsertionSort('+str(n)+')','from __main__ import testInsertionSort')
        t2 = timer2.timeit(10)
        timesInsert.append(t2)

    pylab.plot(sizes,timesMerge)
    pylab.plot(sizes,timesInsert)
    pylab.show()
```