Informatique en CPGE (2015-2016)
Corrigé TP 10 : résolution numérique d'équations différentielles ; méthode d'Euler

Exercice 1

Objectif : programmer la méthode d'Euler pour résoudre l'équation différentielle $y' = y$ sur l'intervalle $[0; 4]$ avec $y(0) = 1$.

1.

```
def euler(a,b,y0,h,f):
    x=a
    y=y0
    liste_x=[a]
    liste_y=[y0]
    while x+h<=b:
        y+=h*f(x,y)
        liste_y.append(y)
        x+=h
        liste_x.append(x)
    return liste_x,liste_y

def f(x,y):
    return y
```

2. On complète le programme avec `print(euler(0,4,1,1,f))`. Le résultat doit être : `([0, 1, 2, 3, 4], [1, 2, 4, 8, 16])`.

3.

```
# calcul de l'erreur
e=0
h=0.1
x,y=euler(0,4,1,h,f)
for i in range(len(x)):
    if abs(exp(x[i])-y[i])>e:
        e=abs(exp(x[i])-y[i])
print("Pour h = ",h,"l'erreur est ",e)
# ou bien
x,y=euler(0,4,1,h,f)
print(x[-1],y[-1],exp(4)-y[-1]) # l'erreur max est sur le dernier terme
```

Pour la dernière valeur des deux listes, il y a un problème d'arrondi, donc le x final n'est pas toujours égal à 4. Pour remédier à ce problème, on peut modifier le test "while" dans la fonction **euler** en écrivant : `while round(x+h,6)<=b:`.

4. Afin que l'erreur maximale soit de l'ordre de 10^{-2}, on prend $h = 0.0001$.

5. Représentations graphiques.

```
import matplotlib.pyplot as plt

# écriture solution exacte
x=[i*0.1 for i in range(41)]
y=[exp(u) for u in x]
plt.plot(x,y)

# écriture sol approchée h=1
x,y=euler(0,4,1,1,f)
plt.plot(x,y)
# écriture sol approchée h=0.5
x,y=euler(0,4,1,0.5,f)
plt.plot(x,y)
# écriture sol approchée h=0.2
x,y=euler(0,4,1,0.2,f)
plt.plot(x,y)
# écriture sol approchée h=0.1
x,y=euler(0,4,1,0.1,f)
plt.plot(x,y)

plt.show()
```

Exercice 2

On reprend le programme 1 avec les modifications nécessaires sur l'écriture de la fonction f, l'écriture de la solution exacte et l'appel de la fonction **euler**.

```
from math import exp,cos,sin
import matplotlib.pyplot as plt

def f(x,y):
    return cos(2*x)-y

# solution exacte
def solex(x):
    return 3.8*exp(-x)+0.2*cos(2*x)+0.4*sin(2*x)
xex=[i*0.01 for i in range(1201)]
yex=[solex(u) for u in xex]
plt.plot(xex,yex,'r')

# solution approchée h=0.1
x,y=euler(0,12,4,0.1,f)
plt.plot(x,y)

plt.show()
```

Exercice 3

Objectif : résoudre l'équation différentielle $y' = -y$ avec $y(0) = 1$ sur l'intervalle $[0; 30]$.

1. On reprend le programme de l'exercice 1. On modifie la fonction $f(x, y) = -y$ et les appels de la fonction **euler**.

```
import matplotlib.pyplot as plt

def euler(a,y0,b,h,f):
    x=a
    y=y0
    liste_x=[a]
    liste_y=[y0]
    while x+h<=b:
        y+=h*f(x,y)
        liste_y.append(y)
        x+=h
        liste_x.append(x)
    return liste_x,liste_y

def f(x,y):
    return -y

# solution approchée h=3
x,y=euler(0,1,30,3,f)
plt.plot(x,y)

# solution approchée h=2.5
x,y=euler(0,1,30,2.5,f)
plt.plot(x,y)

plt.show()
```

2. Stabilité pour $h \leq 2$.

```
# solution approchée h=1.5
x,y=euler(0,1,30,1.5,f)
plt.plot(x,y)

# solution approchée h=2
x,y=euler(0,1,30,2,f)
plt.plot(x,y)

plt.show()
```

3. L'erreur de discrétisation e est inférieure à 10^{-1} pour $h = 0.4$. (On utilise le code de l'exercice 1 en modifiant la valeur exacte.

```
# calcul de l'erreur
e=0
h=0.4
x,y=euler(0,1,30,h,f)
for i in range(len(x)):
    if abs(exp(-x[i])-y[i])>e:
        e=abs(exp(-x[i])-y[i])
print("Pour h = ",h,"l'erreur est ",e)
```

Exercice 4

Objectif : résoudre l'équation différentielle du second ordre $y'' + y = 0$ pour $x \in [0; 10]$ avec les conditions initiales $y(0) = 0$ et $y'(0) = 1$.

On reprend les éléments du programme de l'exercice 1 avec quelques modifications.

1. Modification de la fonction f.

```
def f(x,y): # y est un couple
    return (y[1],-y[0])
```

2. Modification de la fonction **euler**.

```
def euler(a,b,y0,h,f):
    x=a
    y=y0
    liste_x=[a]
    liste_y=[y0]
    while x+h<=b:
        y=(y[0]+h*(f(x,y)[0]),y[1]+h*(f(x,y)[1])) # la difficulté
        liste_y.append(y)
        x+=h
        liste_x.append(x)
    return liste_x,liste_y
```

3. L'appel de la fonction.

```
# solution approchée h=0.01
x,y=euler(0,10,(0,1),0.01,f)
```

4.

```
from math import sin  # pour la sol exacte

import matplotlib.pyplot as plt

u=[y[i][0] for i in range(len(y))]

plt.plot(x,u)

# solution exacte y(x)=sin(x)
x=[i*0.01 for i in range(1001)]
y=[sin(u) for u in x]
plt.plot(x,y)

plt.show()
```

Exercice 5

La définition de la fonction **euler** s'écrit comme dans l'exercice 1. On modifie la définition de la fonction f qui renvoie un objet de type **array** et l'appel de la fonction **euler** :

```
# y''+y=0 soit (y,y')'=(y',-y)=F(y,y')

from math import sin

from numpy import array

def f(x,y):
    return array((y[1],-y[0]))  # utilisation d'un array
def euler(a,b,y0,h,f):
    x=a
    y=y0
    liste_x=[a]
    liste_y=[y0]
    while x+h<=b:
        y=y+h*f(x,y)   # plus aucun problème de calcul
        liste_y.append(y)
        x+=h
        liste_x.append(x)
    return (liste_x,liste_y)

x,y=euler(0,10,array((0,1)),0.01,f)

u=[y[i][0] for i in range(len(y))]

plt.plot(x,u)

plt.show()
```

Exercice 6

Objectif : l'équation $\theta'' = -k_1 \sin\theta - k_2\theta'$.

```
# y''= -k1 sin(y) - k2 y' pendule amorti (si k2=0, pendule simple)
# soit (y,y')'=(y',-k1 sin(y)- k2 y)=F(y,y')

from math import sin,pi

from numpy import array

def f(x,y):
    return array((y[1],-5*sin(y[0])-0.5*y[1]))  # les valeurs de k1 et k2

def euler(a,b,y0,h,f):
    x=a
    y=y0
    liste_x=[a]
    liste_y=[y0]
    while x+h<=b:
        y=y+h*f(x,y)
        liste_y.append(y)
        x+=h
```

```
        liste_x.append(x)
    return (liste_x,liste_y)

# solution approchée h=0.01
x,y=euler(0,20,array((pi/6,0)),0.01,f)
for i in range(len(x)):
    fic1.write(str(x[i])+'\t'+str(y[i][0])+'\n')
    fic2.write(str(y[i][0])+'\t'+str(y[i][1])+'\n')

u=[y[i][0] for i in range(len(y))]
v=[y[i][1] for i in range(len(y))]

plt.plot(x,u)
plt.plot(u,v)

plt.show()
```

Exercice 7
Pour une équation du type $x'(t) = f(x(t), t)$, on utilise la fonction **odeint** de scipy.integrate.

```
from math import sin, pi
import numpy as np
import matplotlib.pyplot as plt
import scipy.integrate as integ

def f(u,t):
    return [u[1],-5*sin(u[0])-0.5*u[1]]

t=np.linspace(0,20,num=400)
sol=integ.odeint(f,[pi/6,0],t)

plt.subplot(2,1,1)
plt.grid()
plt.plot(t,sol[:,0]) # angle fonction de t
plt.subplot(2,1,2)
plt.grid()
plt.plot(sol[:,0],sol[:,1]) #diagramme de phase
plt.show()
```