Informatique
TP5 : Initiation aux outils de calcul scientifique
CPP 1A

Romain Casati, Wafa Johal, Frederic Devernay, Matthieu Moy

Avril - juin 2014

1 Rappels et compléments sur NumPy

NumPy est une bibliothèque Python qui permet à la fois de manipuler des tableaux multidimensionels mais aussi de faire des calculs sur ces objets. En cours, vous avez vu comment

- créer un tableau avec NumPy :

```
>>> import numpy
>>> v = numpy.array([1, 2, 3])
>>> v
array([1, 2, 3])
```

- faire des opérations algébriques sur les tableaux :

```
>>> w = 3 * v - numpy.array([3, 6, 9])
>>> w
array([0, 0, 0])
```

- multiplier des matrices et des vecteurs mais nous ne nous en servirons pas dans ce TP.

NumPy peut faire beaucoup plus que ça. Comme le module `math` de Python, NumPy dispose de la plupart des fonctions mathématiques que vous connaissez. Cependant, les fonctions NumPy ont l'avantage de s'appliquer sur les tableaux (composantes par composantes).

Exercice 1 (NumPy et les fonctions) *Que donne le code suivant ?*

```
import numpy
v = numpy.array([1, 2, 3, 4]) * numpy.pi / 4  # numpy.pi = π
w = numpy.sin(v)
print(w)
```

NumPy dispose aussi de fonctions permettant de générer des tableaux.

Exercice 2 (Génération de tableaux) *Que font les fonctions de NumPy suivantes :* `zeros([num])`, `ones([num])`, `linspace(debut, fin, num)`, `random.random([num])` *?*

2 Python + Matplotlib : mieux qu'une calculatrice graphique

Matplotlib est une bibliothèque Python qui permet de faire toutes sortes de tracés. Nous allons essentiellement l'utiliser pour tracer des fonctions. Les instructions de base pour se servir de Matplotlib sont illustrées ici :

```
import matplotlib.pyplot as plt  # plt devient l'abrege de matplotlib.pyplot

x = [0, 1, 2]  # liste des abscisses
y = [1, -1, 0]  # liste des ordonnees

plt.plot(x, y)  #  trace y en fonction de x
plt.show()  # affiche la fenetre du trace
```

Exercice 3 (Premiers tracés) *En utilisant les fonctions* `linspace` *et* `cos` *de NumPy, tracer la fonction* $y = \cos(x)$ *sur* $[0, 10\pi]$. *Quelle est l'influence du nombre de points passé à la fonction* `linspace` *? Grâce à un second appel à la fonction* `plt.plot` *avant l'appel à* `plt.show`, *superposer le graphe de la fonction* $y = \exp(-x/10)\cos(x)$. *Toujours avant l'appel à* `plt.show`, *ajouter un titre avec* `plt.title('Le titre de votre choix')` *et des noms aux axes avec* `plt.xlabel('x')` *et* `plt.ylabel('y=f(x)')`.

Matplotlib n'est pas seulement capable de tracer des fonctions de variable réelle mais aussi des *courbes paramétrées*. Vous étudierez ces objets en deuxième année mais pour faire simple, vous pouvez penser à la trajectoire d'un point au cours du temps. Décrire cette trajectoire, c'est se donner à chaque instant t l'abscisse et l'ordonnée du point étudié. Plus formellement, on se donne une fonction d'un intervalle $I \subset \mathbb{R}$ de la forme $t \mapsto (x(t), y(t))$.

Exercice 4 (Courbes paramétriques) *En modifiant le code ci-dessous, tracer l'une des courbes suivantes :*

- *La Lemniscate de Bernoulli* $\begin{cases} x(t) &= \frac{\sin t}{1+\cos^2 t} \\ y(t) &= \frac{\sin t \cos t}{1+\cos^2 t} \end{cases}$ *sur* $[0, 2\pi]$.
- *La spirale d'Archimède* $\begin{cases} x(t) &= t\cos(t) \\ y(t) &= t\sin(t) \end{cases}$ *sur* $[0, 10\pi]$.
- *La courbe du cœur* $\begin{cases} x(t) &= 16\sin^3 t \\ y(t) &= 13\cos t - 5\cos(2t) - 2\cos(3t) - \cos(4t) \end{cases}$ *sur* $[0, 2\pi]$.
- *Les cyclo-harmoniques* $\begin{cases} x(t) &= \left(1 + \cos\left(\frac{p}{q}t\right)\right)\cos(t) \\ y(t) &= \left(1 + \cos\left(\frac{p}{q}t\right)\right)\sin(t) \end{cases}$ *pour* p *et* q *entiers sur* $[0, 2q\pi]$.

```
import matplotlib.pyplot as plt
import numpy as np  # np devient l'abrege de numpy

t = np.linspace(..., ..., ...)

x = ...  # x(t)
y = ...  # y(t)

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

3 Résolution d'équations différentielles

On considère l'équation différentielle ordinaire (EDO) du premier ordre suivante

$$y'(x) = f(y(x), x), \quad x \in [a, b].$$

Si l'on se donne en plus une condition initiale de la forme $y(a) = y_0$, on obtient un *problème de Cauchy*. Sous certaines hypothèses de régularité sur la fonction f [1], on sait qu'il existe une unique solution y sur $[a, b]$. Même si dans certains cas (*e.g.* f linéaire à coefficients constants) on connaît explicitement cette solution, il est la plupart du temps impossible de l'écrire. On utilise alors des méthodes permettant d'approcher cette solution en calculant des approximations de proches en proches.

L'intervalle $[a, b]$ est découpé en N intervalles de taille $h = \frac{b-a}{N}$ et la solution y est approchée aux points $x_n = a + nh$, $n = 0, ..., N$. On note $y_n \approx y(x_n)$ l'approximation de y au point x_n.

3.1 Préliminaire : passage d'une fonction en paramètre

Dans la suite, nous allons avoir besoin d'écrire des fonctions qui dépendent d'autres fonctions. L'exercice suivant montre que ceci n'est pas gênant en Python.

Exercice 5 *Que fait le code suivant ? Expliquer.*

```
def f(x):
    return x ** 2
```

1. f globalement lipschitzienne par rapport à y.

```python
def g(x):
    return 2 * x

def somme_ponderee(f1, f2, a, b, x):
    return a * f1(x) + b * f2(x)

print(somme_ponderee(f, g, 2, 3, 10))
```

3.2 Méthode d'Euler

Dans la suite, on se propose d'étudier la *méthode d'Euler* qui consiste à approcher la solution en utilisant la formule de la tangente

$$y(x+h) \approx y(x) + hy'(x) \approx y(x) + hf(y(x), x).$$

Exercice 6 (Récurrence de la méthode d'Euler) *En utilisant cette dernière expression, donner l'équation de récurrence qui permet de calculer y_{n+1} en fonction de y_n, x_n, h et f.*

Exercice 7 (Implémentation) *Dans un fichier nommé* `edo.py`*, écrire une fonction* `euler(f, a, b, y0, N)` *qui retourne la liste des y_n, $n = 0, ..., N$ approchant la solution de notre EDO par la méthode d'Euler.*

Exercice 8 (Tests) *Quelle est la solution attendue pour $f(y, x) = \lambda y$ et $y(0) = 1$? Utiliser ce résultat pour tester l'implémentation de l'exercice précédent : tracer la solution exacte et la solution approchée avec Matplotlib. Quelle est l'influence du paramètre N ?*

3.3 Comparaison avec SciPy

Le module SciPy de Python est consacré au calcul scientifique avancé. En particulier, en utilisant la fonction `scipy.integrate.odeint`, il est possible d'approcher la solution d'une EDO grâce à des méthodes plus performantes (mais bien plus complexes aussi) que la méthode d'Euler. Il s'utilise comme ceci :

```python
import scipy.integrate as scint
import numpy as np
import matplotlib.pyplot as plt


def f(y, x):
    return np.cos(10 * y) * np.cos(x)  # f(y(x), x) = cos(10y(x)) cos(x)

x = np.linspace(0, 10, 100)
y = scint.odeint(f, 0, x)  # f, y0, points ou la solution est approchee

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

Exercice 9 (Comparaison avec SciPy) *Utiliser SciPy ainsi que la méthode d'Euler pour approcher la solution du problème de Cauchy associée à $f(y, x) = \cos(10y)\cos(x)$ et $y(0) = 0$. On prendra soin d'utiliser de même nombre de pas N dans les deux méthodes. Comparer visuellement les deux solutions à l'aide de Matplotlib.*

3.4 Le pendule simple (en bonus)

Une EDO du second ordre que vous connaissez bien est celle du pendule simple :

$$\theta'' + \omega_0^2 \sin\theta = 0. \tag{1}$$

Toute EDO d'ordre supérieure à 1 peut être ramenée à une EDO d'ordre 1. Pour le pendule simple, il suffit de poser $y = (y_1, y_2) = (\theta, \theta')$ et de remarquer que $y' = (y_2, -\omega_0^2 \sin y_1)$.

Exercice 10 *En utilisant cette dernière remarque, approcher la solution du pendule simple avec SciPy et la méthode d'Euler pour les conditions initiales $y_0 = (\frac{\pi}{2}, 0)$. Que peut-on dire de la conservation de l'énergie mécanique*[2] *$E_m = \frac{1}{2}\theta'^2 - \omega_0^2 \cos\theta$?*

2. Cette quantité n'est pas tout à fait l'énergie mécanique, elle y est seulement proportionnelle.

4 Correction des exercices

Correction exercice 1 *On s'attend à trouver* $\left[\frac{\sqrt{2}}{2}, 1, \frac{\sqrt{2}}{2}, 0\right]$

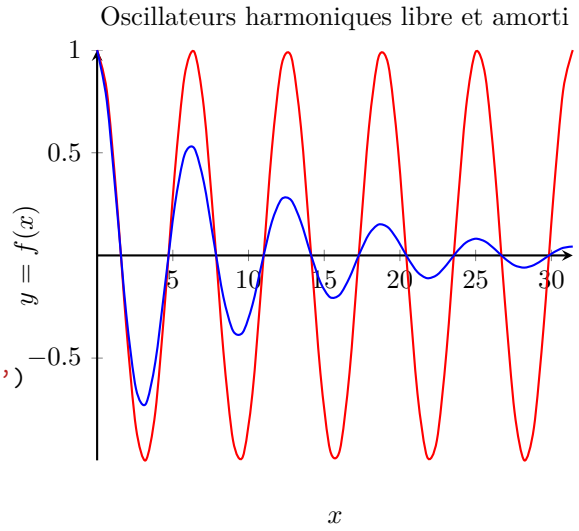
```
[  7.07106781e-01   1.00000000e+00   7.07106781e-01   1.22464680e-16]
```

Correction exercice 2

- `zeros([num])` *retourne un tableau de taille* `num` *ne contenant que des 0.*
- `ones([num])` *retourne un tableau de taille* `num` *ne contenant que des 1.*
- `linspace(a, b, n)` *retourne un échantillonage uniforme de l'intervalle* $[a, b]$ *de taille* n. *Les éléments sont les* $a + k\frac{b-a}{n-1}$ *pour* $k = 0, ..., n-1$.
- `random.random([num])` *retourne un vecteur de taille* `n` *dont les éléments sont tirés aléatoirement entre 0 et 1.*

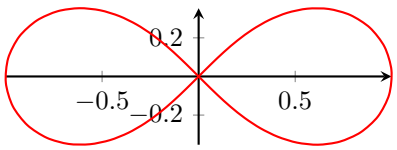
Correction exercice 3 *Si le nombre de points passé à la fonction* `linspace` *n'est pas suffisant, le graphe ressemble à une ligne brisée et n'est pas lisse.*

```
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(0, 10 * np.pi, 100)
y1 = np.cos(x)
y2 = np.exp(-x/10) * np.cos(x)

plt.plot(x, y1)
plt.plot(x, y2)

plt.title('Oscillateurs harmoniques libre et amorti')
plt.xlabel('x')
plt.ylabel('y = f(x)')
plt.show()
```

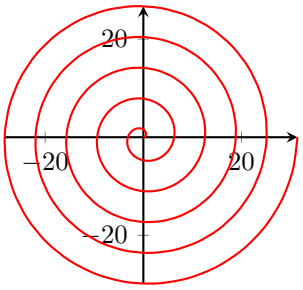


Correction exercice 4

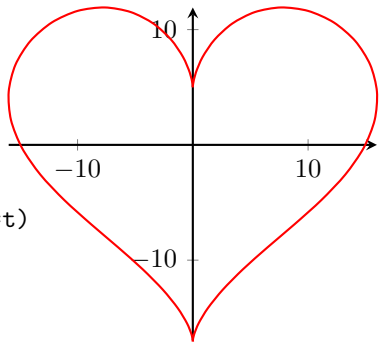
```
import matplotlib.pyplot as plt
import numpy as np

t = np.linspace(0, 2 * np.pi, 100)
# Lemniscate de Bernoulli
x = np.sin(t) / (1 + np.cos(t) ** 2)
y = np.sin(t) * np.cos(t) / (1 + np.cos(t) ** 2)

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



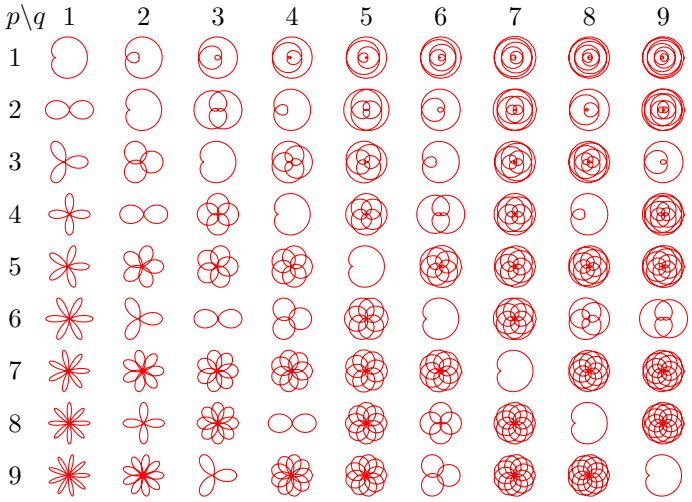

```
import matplotlib.pyplot as plt
import numpy as np

t = np.linspace(0, 10 * np.pi, 300)
# Spirale d'Archimede
x = t * np.cos(t)
y = t * np.sin(t)

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



```python
import matplotlib.pyplot as plt
import numpy as np

t = np.linspace(0, 2 * np.pi, 100)
# Courbe du coeur
x = 16 * np.sin(t) ** 3
y = 13*np.cos(t) - 5*np.cos(2*t) - 2*np.cos(3*t) - np.cos(4*t)

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




```python
import matplotlib.pyplot as plt
import numpy as np

p = 8
q = 3

t = np.linspace(0, 2 * q * np.pi, 400)
# Cyclo-harmonique
tmp = 1 + np.cos(p / q * t)
x = tmp * np.cos(t)
y = tmp * np.sin(t)

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



Correction exercice 5 *C'est la valeur 260 qui est retournée. En effet, la fonction* `somme_ponderee` *appelle* `f` *sur la valeur 10 (ce qui retourne 100) et multiplie le résultat par 2 (200). Le second terme de la somme fait appel à* `g` *sur la valeur 10 (ce qui retourne 20) et multiplie la valeur par 3 (60).*

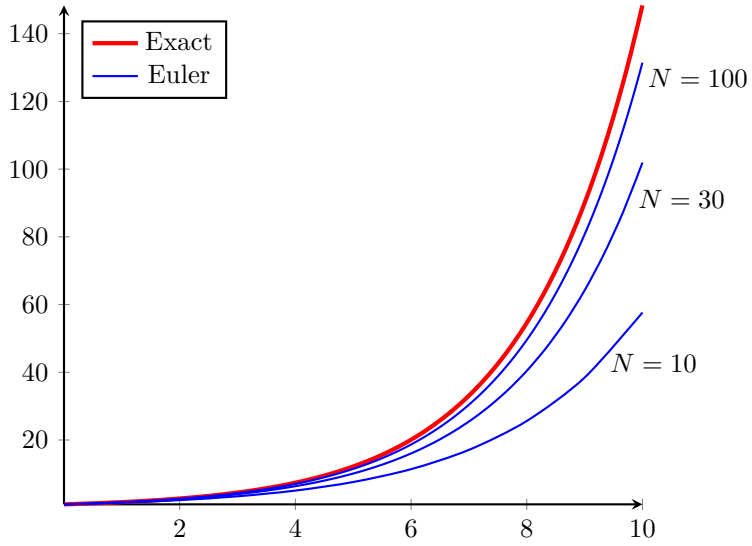
Correction exercice 6 $y_{n+1} = y_n + hf(y_n, x_n)$.

Correction exercice 7

```python
# fichier edo.py

def euler(f, a, b, y0, N):
    y = y0
    x = a
    h = (b - a) / N
    res = [y]
    for i in range(N):
        y = y + h * f(y, x)
        x = x + h
        res.append(y)
    return res
```

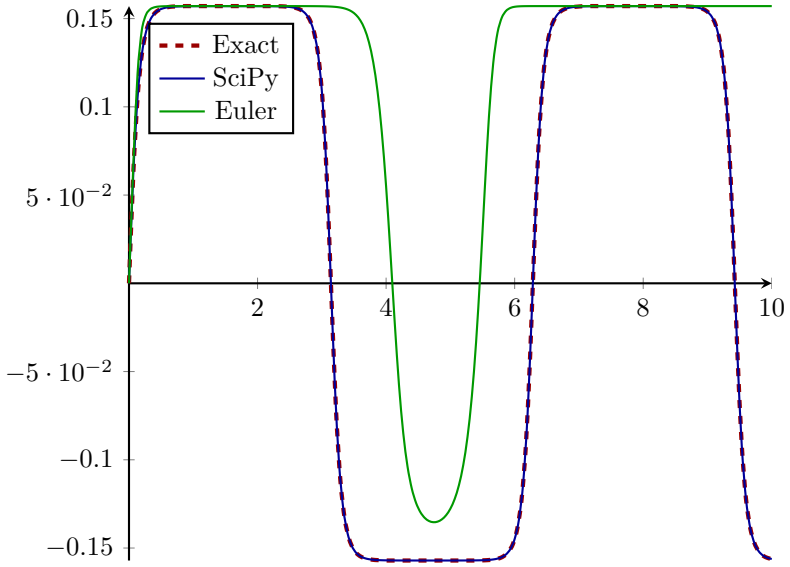
Correction exercice 8 *La solution est* $y(x) = e^{\lambda x}$.

```python
import numpy as np
import matplotlib.pyplot as plt
import edo

def f(y, x):
    return 0.5 * y  # on choisit λ = 1/2

N = 100
a = 0
b = 10
x = np.linspace(a, b, N + 1)
y = edo.euler(f, a, b, 1, N)

plt.plot(x, y)
# truc : y(x) = e^{λx} = e^{f(x,x)}
plt.plot(x, np.exp(f(x, x)))
plt.show()
```



Correction exercice 9

```python
import scipy.integrate as scint
import edo
import numpy as np
import matplotlib.pyplot as plt

def f(y, x):
    return np.cos(10 * y) * np.cos(x)

a = 0
b = 10
N = 150
y0 = 0

x = np.linspace(a, b, N + 1)
y_scipy = scint.odeint(f, y0, x)
y_euler = edo.euler(f, a, b, y0, N)

plt.plot(x, y_scipy)
plt.plot(x, y_euler)
plt.show()
```



Correction exercice 10

```
import numpy as np
import scipy.integrate as scint
import edo
import matplotlib.pyplot as plt

a = 0
b = 10
w02 = 10
y0 = np.array([np.pi / 2, 0])
N = 1000

def f(y, x):
    return np.array([y[1], -w02 * np.sin(y[0])])

def energy(theta, thetadot):
    return 0.5 * thetadot ** 2 - w02 * np.cos(theta)

x = np.linspace(a, b, N + 1)
y_scipy = scint.odeint(f, y0, x)
y_euler = np.array(edo.euler(f, a, b, y0, N))

theta_scipy = y_scipy[:, 0]
thetadot_scipy = y_scipy[:, 1]
theta_euler = y_euler[:, 0]
thetadot_euler = y_euler[:, 1]

plt.plot(x, energy(theta_scipy, thetadot_scipy))
plt.plot(x, energy(theta_euler, thetadot_euler))
plt.show()
```