CONCOURS CENTRALE•SUPÉLEC

Oral

Python

MP, PC, PSI, TSI

Analyse numérique

La plupart des fonctions présentées dans cette section nécessitent l'import du module `numpy` et de sous-modules du module `scipy`. Les instructions nécessaires aux exemples suivants sont listés ci-dessous.

```
import numpy as np
import scipy.optimize as resol
import scipy.integrate as integr
import matplotlib.pyplot as plt
```

Nombres complexes

Python calcule avec les nombres complexes. Le nombre imaginaire pur i se note `1j`. Les attributs `real` et `imag` permettent d'obtenir la partie réelle et la partie imaginaire. La fonction `abs` calcule le module d'un complexe.

```
>>> a = 2 + 3j
>>> b = 5 - 3j
>>> a*b
(19+9j)
>>> a.real
2.0
>>> a.imag
3.0
>>> abs(a)
3.6055512754639896
```

Fonctions mathématiques

La constante π s'obtient grâce à la commande `pi`.

Le module `numpy` connaît les fonctions mathématiques usuelles. La fonction partie entière s'obtient par la commande `floor`. Attention la fonction logarithme népérien a pour nom de commande `log`.

```
>>> np.exp(1)
2.7182818284590451
>>> np.cos(np.pi)
-1.0
>>> np.log(np.exp(1))
1.0
>>> np.floor(3.4)
3
>>> np.floor(-3.7)
-4
```

Résolution approchée d'équations

Pour résoudre une équation du type $f(x) = 0$ où f est une fonction d'une variable réelle, on peut utiliser la fonction `fsolve` du module `scipy.optimize`. Il faut préciser la valeur initiale x_0 de l'algorithme employé par la fonction `fsolve`. Le résultat peut dépendre de cette condition initiale.

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

>>> resol.fsolve(f, -2.)
array([-1.41421356])

>>> resol.fsolve(f, 2.)
array([ 1.41421356])
```

Dans le cas d'une fonction f à valeurs vectorielles, on utilise la fonction `root`. Par exemple, pour résoudre le système non linéaire

$$\begin{cases} x^2 - y^2 = 1 \\ x + 2y - 3 = 0 \end{cases}$$

```
def f(v):
    return v[0]**2 - v[1]**2 - 1, v[0] + 2*v[1] - 3

>>> sol = resol.root(f, [0,0])
>>> sol.success
True
>>> sol.x
array([ 1.30940108,  0.84529946])

>>> sol=resol.root(f, [-5,5])
>>> sol.success
True
>>> sol.x
array([-3.30940108,  3.15470054])
```

Calcul approché d'intégrales

La fonction `quad` du module `scipy.integrate` permet de calculer des valeurs approchées d'intégrales. Elle renvoie une valeur approchée de l'intégrale ainsi qu'un majorant de l'erreur commise. Cette fonction peut aussi s'employer avec des bornes d'intégration égales à $+\infty$ ou $-\infty$.

```
def f(x):
    return np.exp(-x)

>>> integr.quad(f, 0, 1)
(0.6321205588285578, 7.017947987503856e-15)

>>> integr.quad(f, 0, np.inf)
(1.0000000000000002, 5.842607038578007e-11)
```

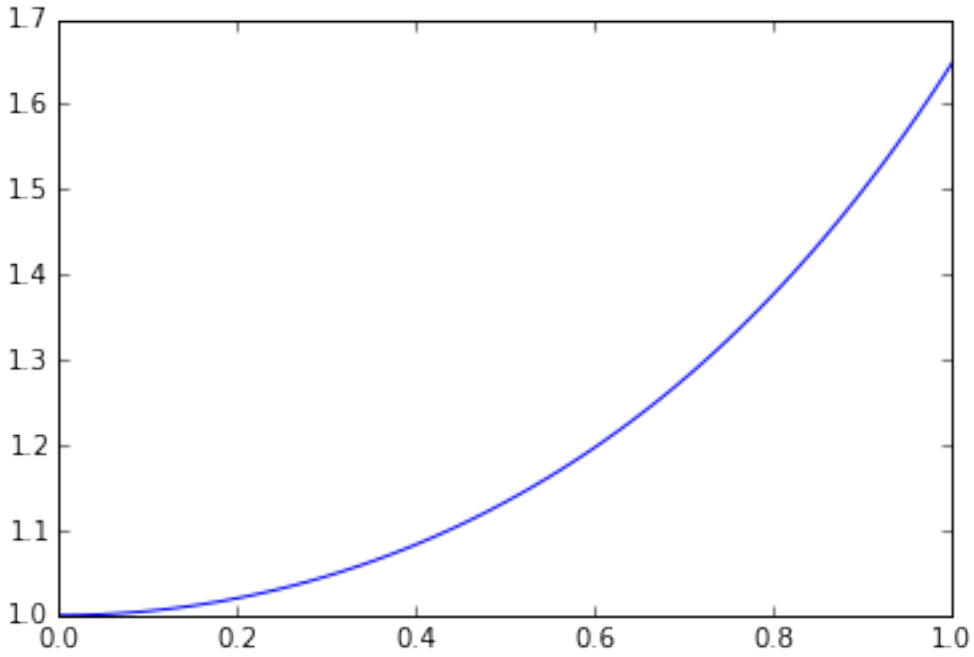
Cette fonction peut être employée pour la définition d'intégrales à paramètres. Ainsi si on veut obtenir des valeurs approchées de $\Gamma(x) = \int_0^{+\infty} e^{-t}t^{x-1}\mathrm{d}t$ pour x réel strictement positif on pourra procéder ainsi :

```
def g(x):
    def f(t):
        return np.exp(-t)*t**(x-1)
    return integr.quad(f,0,np.inf)[0]

>>> g(2)
0.9999999999999998
```

Résolution approchées d'équations différentielles

Pour résoudre une équation différentielle $x' = f(x,t)$, on peut utiliser la fonction `odeint` du module `scipy.integrate`. Cette fonction nécessite une liste de valeurs de t, commençant en t_0, et une condition initiale x_0. La fonction renvoie des valeurs approchées (aux points contenus dans la liste des valeurs de t) de la solution x de l'équation différentielle qui vérifie $x(t_0) = x_0$. Pour trouver des valeurs approchées sur $[0,1]$ de la solution $x'(t) = tx(t)$ qui vérifie $x(0) = 1$, on peut employer le code suivant.

```
def f(x, t):
    return t*x

>>> T = np.arange(0, 1.01, 0.01)
>>> X = integr.odeint(f, 1, T)
>>> X[0]
array([ 1.])
>>> X[-1]
array([ 1.64872143])
>>> plt.plot(T,X)
>>> plt.show()
```

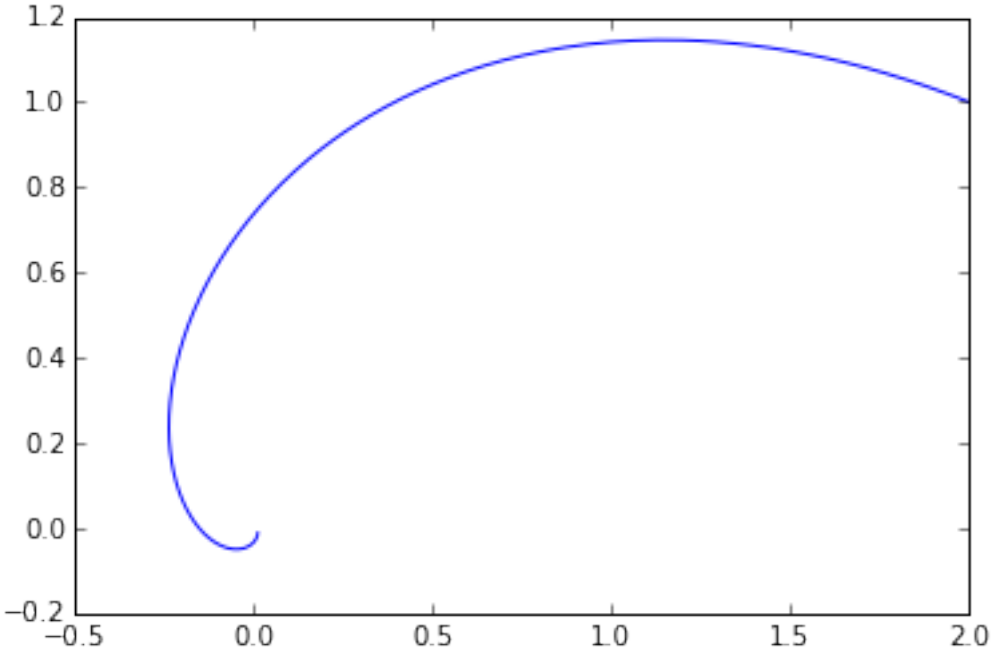


Si on veut résoudre, sur $[0,1]$, le système différentiel

$$\begin{cases} x'(t) = -x(t) - y(t) \\ y'(t) = x(t) - y(t) \end{cases}$$

avec la condition initiale $x(0) = 2, y(0) = 1$ le code devient le suivant.

```
def f(x, t):
    return np.array([-x[0]-x[1], x[0]-x[1]])

>>> T = np.arange(0, 5.01, 0.01)
>>> X = integr.odeint(f, np.array([2.,1.]), T)
>>> X[0]
array([ 2.,  1.])
>>> plt.plot(X[:,0], X[:,1])
>>> plt.show()
```



Pour résoudre une équation différentielle scalaire d'ordre 2 de solution x, on demandera la résolution du système différentiel d'ordre 1 satisfait par $X(t) = \begin{pmatrix} x(t) \\ x'(t) \end{pmatrix}$.

Ainsi, si on considère la fonction x qui vérifie l'équation différentielle $x''(t) + 2x'(t) + 3x(t) = \sin(t)$ avec les conditions initiales $x(0) = 0$, $x'(0) = 1$ et , le vecteur X vérifiera $X'(t) = \begin{pmatrix} x'(t) \\ -2x'(t) - 3x(t) - \sin(t) \end{pmatrix}$. Pour obtenir la représentation graphique de x sur l'intervalle $[0, 3\pi]$, on pourra utiliser le code suivant :

```
def f(x,t) :
    return np.array([x[1], -2*x[1] - 3*x[0] + np.sin(t)])

T = np.arange(0, 3*np.pi + 0.01, 0.01)
X = integr.odeint(f, np.array([0,1]), T)
plt.plot(T, X[:,0])
plt.show()
```