Calcul scientifique avec Python

1) Présentation générale des modules à utiliser

- Le module `numpy` est utilisé pour le calcul vectoriel (algèbre linéaire)

- Le module `scipy` est utilisé pour le calcul scientifique (interpolation, optimisation, résolution approchée d'équations différentielles). Le module `scipy` regroupe un certain nombre de sous-modules qui sont autant de boîtes à outils de routines courantes de calcul numérique, notamment les modules `linalg`, `integrate`, `odeint`.

- Le module `matplotlib` sert à tracer des courbes. qui a un sous-module pyplot qui gère le tracé des figures.

On peut charger les modules directement dans l'environnement : `from math import *`

mais aussi utiliser un alias pour alléger les appels aux fonctions du module :

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

2) Calcul vectoriel (et matriciel)

```
from numpy import *
```

`numpy` ajoute notamment le type `array` (tableau) qui est proche du type `list` (liste) avec la condition supplémentaire que tous les éléments sont du même type. En pratique, il s'agit donc d'un tableau d'entiers, de flottants voire de booléens. On peut transformer les listes en tableaux, et réciproquement :

```
l = [1,2,4] ; a = array(l) ; l = list(a)
```

Sur les tableaux, les opérations s'effectuent terme à terme :

`array([1,1,1])+array([1,0,1])` renvoie `array([2,0,2])`, à ne pas confondre avec l'addition sur les listes (concaténation).

Les tableaux peuvent être multidimensionnels :

`array([[1,2],[3,4]])` crée la matrice $\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$ définie par le tableau des vecteurs lignes.

Exemple : `a = array([ [0]*2 ]*2 ) ; a[0,0] = 1 ; print(a)` # affiche $\begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}$.

`a.shape` renvoie le couple (n, p) donnant les nombres de lignes et de colonnes.

Les commandes `zeros(n)` et `zeros((n,p))` créent des vecteurs et matrices de coefficients initialisés à 0.

`a = diag([1,2])` définit la matrice diagonale de coefficients diagonaux 1 et 2.

`dot(a,b)` renvoie le produit matriciel des matrices représentées par les tableaux a et b.

Remarque : Ne pas confondre `dot(a,a)` avec `a**2` (où tous les coefficients sont élevés au carré).

Exemples de fonctions du module `linalg` :

```
a = array([[1, 2], [3, 4]]) ; b = array([1, 0])
```

```
linalg.eigvals(a) ; linalg.det(a) ; linalg.inv(a)
```

`linalg.matrix_power(a,n) ; linalg.solve(a,b)`

Remarque : On peut obtenir la liste des fonctions du module `linalg` en utilisant `help(linalg)`.

3) Calcul intégral et différentiel

```
import numpy as np
```

a) *Exemple* : Calcul de $\int_0^{\pi/2} \sin(t)\ dt$.

```
from scipy.integrate import quad
```

`quad(sin,0,pi/2) #` renvoie (0.9999999999999999, 1.1102230246251564e-14)

Remarque : `quad(f,a,b)` renvoie un couple `(s,e)`, où s est une valeur approchée de $\int_a^b f(t)\ dt$ et e une majoration de l'erreur commise. Ainsi, `(s,e)=quad(f,a,b) ; print(s)` affiche la valeur approchée de l'intégrale.

b) *Exemple* : Résolution numérique de $y''(t) = e^{-t}\ y'(t) + y(t)$, avec les conditions initiales $y(0) = 1$ et $y'(0) = 0$.

L'équation équivaut au système d'ordre 1 défini par $\begin{cases} y'(t) = z(t) \\ z'(t) = e^{-t}\ z(t) + y(t) \end{cases}$ avec $\begin{cases} y(0) = 1 \\ z(0) = 0 \end{cases}$

La fonction `odeint` du module `scipy.integrate` permet de résoudre numériquement les systèmes différentiels d'ordre 1 résolus, c'est-à-dire les systèmes de la forme $X'(t) = f(X(t), t)$ connaissant $X(t_0)$.

Dans l'exemple traité, on a $X(t) = (y(t), z(t))$ et $X(0) = (1, 0)$. On résout ici pour $t \in [0, 10]$

On commence par définir la fonction f :

```
def f(X,t) :  return (X[1], -X[0]-exp(-t)*X[1])
```

Puis on définit la subdivision régulière de $[a, b]$ contenant $n = 100$ points (donc le pas est $\frac{b-a}{n-1}$).

```
n = 100 ; sub = linspace(0,10,n)
```

Autrement dit, de façon générale, `linspace(a,b,n)` est le tableau de longueur n dont les éléments sont les $t_k = a + k\frac{b-a}{n-1}$, avec $0 \leq k < n$. On peut le remplacer par `sub = (b-a)/(n-1)*array(range(n))`.

Enfin, on utilise la fonction `odeint` :

```
from scipy.integrate import odeint
```

`Z = odeint(f,[1,0],sub)` # affecte à la variable `Z` les composantes de la solution approchée de X sur l'intervalle $[a, b]$ avec le pas donné pour t dans la résolution numérique.

Plus précisement, `Z` est une matrice (= tableau de deux dimensions), ici $n \times 2$, de sorte que pour tout $0 \leq k < n$, `Z[k]` est un tableau donnant une valeur approchée de $X(t_k)$, ici $(y(t_k), y'(t_k))$.

Par exemple, `Z[n]` vaut ici [−0.657352560.23917261].

Pour obtenir uniquement la valeur approchée de y en $t = 10$, il suffit donc de considérer `Z[n,0]`.