Python, numpy, pylab pour le calcul numérique dans un notebook Sage

Eric Sonnendrücker, Université de Strasbourg

29 septembre 2009

Opérations arithmétiques

Assignation : `a=1`

Addition, soustraction, multiplication : `a+b, a-b, a*b`

Division $a, b \in \mathbb{R}$: `a/b` résultat dans $\mathbb{R}$

Division $a, b \in \mathbb{Z}$: quotient `q=a/b` et reste `r=a%b` (a=qb+r)

Incrémentation : `a+=b` équivaut à `a=a+b`
(idem `a-=b, a*=b, a/=b`)

Exponentiation : `a**b` (*Rem: dans Sage également* `a^b`.)

Opérateurs de comparaison

Egalité : `a==b`

Inégalité stricte : `a<b, a>b`

Inégalité large : `a<=b, a>=b`

Différence : `a!=b`

Logique a et b : `a and b`

Logique a ou b : `a or b`

Listes

Une liste python est définie entre crochets, les éléments étant séparés par une virgule. Elles peuvent contenir différents types d'éléménts. Ex. `l=[1,2,3.5,'a',-1.2,'sum']`

Accès à l'élément $i \in [0, N-1]$: `l[i]`

Accès aux éléments i à j-1 (à partir de 0) : `l[i:j]`

Ajout d'un élément x à la fin : `l.append(x)`

Insertion d'un élément x à la position i : `l.insert(i,x)`

Inversion de la liste : `l.reverse()`

Listes d'entiers :
`range(N)` : liste des entiers de 0 à N-1
`range(M,N)` : liste des entiers de M à N-1
`range(M,N,P)` : liste des entiers de M à N-1 avec un pas P

Création de listes à partir de fonctions :
`[i**2 for i in range(1,5)]` : liste des 4 premiers carrés
`[[i**2*j**3 for i in range(5)] for j in range(4)]` : liste de liste

Fonctions

Fonctions courtes : utiliser le mot clef `lambda`

```
f=lambda x:x**2+1
```

Fonctions longues : mot clef `def`

Rem : contenu de la fonction défini par l'indentationp.

```
def f(x):
    y=x-1
    return y**2+1
```

Plusieurs variables d'entrée et de sortie par ex. avec `lambda`

```
f=lambda x,y:x**2+y**2
```

Plusieurs variables de sortie, uniquement avec `def`

```
def f(x,y):
    s=x+y
    d=x-y
    return s,d
```

Utilisation : `sum,diff = f(3,5)`

Application de f à tous les élts d'une liste l : `map(f,l)`

Boucles et instructions conditionnelles

Blocs logiques définis par l'indentationp.

Boucle de 0 à N-1 (N entier fixé) :

```
for i in range(N):
     print i**2
```

Boucle de N à M-1 avec des pas de P (N,M,P entiers fixés) :

```
for i in range(N,M,P):
     print i**2
```

Boucle sur une liste quelconque :
par ex. `ll=[1, 3, 4, 7, 5, 2]`
Rem : on pourrait aussi avoir des réels dans la liste

```
for i in ll:
     print i**2
```

Instructions conditionnelles :

```
def f(x):
   if x>=1:
       return 0.0
   elif x<0:
       return -1.0
   else:
       return 1.0
```

Modules

Python est un langage modulaire. De nombreuses fonctions supplémentaires sont définis dans des modules que l'on doit importer pour y avoir accès.

Un module toto contient toutes les expressions et fonctions python définis dans le fichier toto.py

Import d'un module toto :
`import toto` : accès à l'élément x de toto avec `toto.x`
`import toto as t` : accès à l'élément x de toto avec `t.x`
`from toto import *` : accès à l'élément x de toto avec `x`
Conflit potentiel de noms entre plusieurs modules.

Contenu du module toto : `dir(toto)` (après `import toto`)

Recharger toto après l'avoir modifié : `reload(toto)`

Nous utiliserons dans la suite : `import numpy as np` (calcul numérique), `import pylab as pl` (graphiques)

Tableaux (numpy)

Tableaux optimisés pour le calcul numérique contrairement aux listes simples.

`np.array([1,2,3])` : Transforme une liste en tableau (*Tous les éléments de la liste doivent être de même type*)

Création de tableaux standards :
`np.empty(N,dtype='int')` : tableau vide de N entiers
`np.empty(N,dtype='float')` : tableau vide de N réels
`dtype` *est optionnel dans toutes les créations de tableaux et fixé par défaut à* `float`
`np.zeros(N)` : tableau de N zéros
`np.ones((N,M))` : tableau de dimension (N,M) de uns
`np.zeros_like(a)` : tableau de zéros de même dimension et type qu'un tableau `a` (idem `np.ones_like`, `np.empty_like`).
`np.arange(M,N,P)` : tableau allant de M à N avec pas P (M et P optionnels, les valeurs par défaut sont M=0 et P=1)

`np.linspace(a,b,N)` : tableau allant de a à b avec N éléments uniformément répartis

`b=a.copy()` : copie d'un tableau a dans un tableau b
Rem : `b=a` *ne copie pas les éléments, mais crée juste une référence, i.e. si a est modifié b l'est aussi.*

Opérations sur les tableaux (numpy)

`a.shape` : obtenir les dimensions de a

`a=a.reshape(8,3)` pour pl. ex. `a=np.ones((4,6))` : modifier les dimensions, en conservant la longueur totale

Extraction de sous-tableaux d'un tableau a:
`a[:,0]` extrait la première colonne
`a[2:3,1]` extrait les troisièmes et quatrièmes lignes de la 2ème colonne
`a[:2,:]` extrait les deux premières lignes

`a[-2:,:]` extrait les deux dernières lignes

Opérations mathématiques : toutes les opérations arithmétiques et fonctions classiques s'appliquent terme à terme pour un tableau a :
`2*a+1; a**2; 1+np.sin(a)` (*utiliser le sin de numpy !*)

Opération globales :
maximum global : `a.max()`, maximum de chaque colonne `a.max(0)`, maximum de chaque ligne `a.max(1)`
Les fonctions suivantes s'appliquent de la même manière soit globalement, soit par direction : minimum : `a.min()`, somme : `a.sum()`, produit : `a.prod()`, moyenne : `a.mean`, somme cumulative : `a.cumsum()`

Opérations booléennes sur un tableau a :
`a>0.5` : tableau (dtype=bool) contenant True à la place des éléments où la condition est vraie, False à la place des autres.
`(a>0.5) & (a<0.7)` : True là où les deux conditions sont vérifiées, `(a>0.5) | (a<0.2)` : True là où une des deux conditions est vérifiée
`(a>0.5).all()` : True si tous les élts vérifient la condition
`(a>0.5).any()` : True si un élément vérifie la condition
`np.where(a>0.5,b,c)` : a,b,c tableaux numpy de même dimensionp. Retourne un tableau de même dimension contenant les éléments correspondants de b là où la condition sur a est vérifiée et ceux de c ailleurs.

Calcul numérique matriciel (numpy)

Les matrices sont des tableaux à deux dimensions, les vecteurs des tableaux à une dimensionp.

Création de matrices :
`np.diag(v)` : matrice avec le vecteur v sur la diagonale. Ex. `np.diag(np.arange(N))`, `np.diag(np.ones(N))`
`np.diag(v,i)` : matrice avec le vecteur v sur la ième diagonale supérieure.
`np.diag(v,-i)` : matrice avec le vecteur v sur la ième diagonale inférieure.
`np.identity(N)` : matrice identité de taille np.

Opération sur des matrices :
`A.transpose()` : tranposée de A
`A.conj().transpose()` : adjointe complexe de A
`np.dot(A,B)` : produit des matrices A et B
`np.inner(x,y)` : produit scalaire des vecteurs x et y
`np.linalg.det(A)` : déterminant de la matrice A
`np.linalg.inv(A)` : inverse de la matrice A
`np.linalg.solve(A,b)` : solution x du système linéaire Ax=b
`np.linalg.eigvals(A)` : valeurs propres de la matrice A
`np.linalg.eig(A)` : valeurs et vecteurs propres de A

Différentiation et intégration numérique (numpy)

`np.trapz(y)` : intégrale du vecteur y avec la formule des trapèzes. Par défaut avec un pas de 1. Les abscisses ou le pas peuvent être donnés.
`np.diff(y)` : différences divisées de y

Tracé de courbes (pylab)

On utilise le module pylab qui fait partie de Matplotlib `http://matplotlib.sourceforge.net/`

`import pylab as pl`

pylab trace des courbes définies par des valeurs numériques, i.e. des tableaux de points.

`x=pl.linspace(-np.pi,np.pi,100)` tableau de 100 points entre $-\pi$ et π (`linspace` *est aussi défini dans pylab !*)

`pl.plot(sin(x))` : crée le graphe du sinus en les points de x en fonction de l'indice de x

`pl.plot(x,sin(x))` : crée le graphe du sinus en les points de x en fonction de x

`pl.plot(x,sin(x),'g-',x,cos(x),'r:')` : crée le graphe de sin en trait continu vert et de cos en trait pointillé rouge.
Couleurs : `'b'` : bleu, `'g'` : vert, `'r'` : rouge, `'c'` : cyan, `'m'` : magenta, `'y'` : jaune, `'k'` : noir, `'w'` : blanc.
Styles de ligne : `'-'` : continue, `'--'` : tirets, `'-.'` : points-tirets, `':'` : pointillée, `'.'` : un point à chaque valeur de x, `'o'` : un cercle à chaque valeur de x, `'v'` : un triangle à chaque valeur de x... et d'autres !

`pl.savefig('file')` : affiche tous les graphiques créés depuis le dernier appel à pl.clf() (file est un nom de fichier utilisé par sage en interne.

`pl.clf()` : efface le buffer contenant les graphiques

Organisation des figures (pylab)

`pl.title('titre du graphe')` : ajoute un titre sur la figure

`pl.xlabel('legx')`, `pl.ylabel('legy')`: ajoute une légende sur l'axe des x, des y

`pl.text(x,y,'texte')` : 'texte' à la position x,y sur l'image.

`pl.subplot(325)` : crée des sous-figures (3 lignes et 2 colonnes) dans un graphique et rend active la 5ème. Utiliser avec le même nombre de lignes et de colonnes avant chaque sous-figure.

Graphiques 2D (pylab)

`x=y=pl.linspace(-3,3,100); X,Y=pl.meshgrid(x,y)` : Création d'un maillage 2D.

`pl.imshow(np.exp(-X**2-Y**2))` : graphe d'une fonction de X,Y en dégradé de couleurs.

`pl.contour(X,Y,np.exp(-X**2-Y**2))` : graphe d'une fonction de X,Y en lignes de niveaux.

`pl.contourf(X,Y,np.exp(-X**2-Y**2))` : graphe d'une fonction de X,Y en lignes de niveaux remplies.