Tutoriel : Calcul numérique en Python avec Numpy et SciPy

Camille Coti

14 octobre 2016

Numpy et SciPy sont des modules Python assez gros concernant le calcul scientifique. Numpy fournit un type array et un type matrix permettant de représenter les matrices et quelques opérations mathématiques pouvant être effectuées sur ces matrices. SciPy fournit des fonctions de calcul scientifique plus avancées qui ne sont pas dans Numpy.

Outre les fonctions de calcul fournies, SciPy contient également un sous-module lib qui contient des wrappers pour les librairies de calcul externes LAPACK (`scipy.lib.lapack`) et BLAS (`scipy.lib.blas`).

1 Tableaux, matrices

On peut créer un array ou une matrix en appelant le constructeur idoine et en lui passant le contenu et, en option, le type de données à construire. On peut également changer leur forme (avec la fonction `reshape` ou `flatten`).

```
1  #!/usr/bin/python
2
3  import numpy
4
5  def main():
6      # on construit une matrice de flottants
7      A = numpy.matrix( '[1, 2, 3 ;4, 5, 6]', float )
8      print A
9      # on construit un tableau d'entiers
10     B = numpy.array( [1, 3, 5 , 2, 4, 6], int )
11     print B
12     # on transforme ce tableau en matrice 2x3
13     B = B.reshape( 2, 3 )
14     print B
15     # on construit deux matrices 2x2
16     top = numpy.array( [1, 2, 3, 4] )
17     bottom = numpy.array( [5, 6, 7, 8] )
18     top = top.reshape( 2, 2 )
19     bottom = bottom.reshape( 2, 2 )
20     # on concatene ces deux matrices : on les met l'une sur l'autre
21     ts = numpy.concatenate( (top, bottom ))
22     print ts
23
24 if __name__ == "__main__":
25     main()
```

La plupart des fonctions d'algèbre linéaire se trouvent dans le sous-module `scipy.linalg`[1].

1. Documentation : `http://docs.scipy.org/doc/scipy/reference/linalg.html`

2 Valeurs aléatoires

Numpy fournit un générateur de nombres aléatoires, qui peut générer une matrice remplie de nombres tirés aléatoirement entre 0 et 1.

```
1  #!/usr/bin/python
2
3  import numpy
4
5  def main():
6      taille = 10
7      A = numpy.matrix( numpy.random.rand( taille, taille ) )
8      print A
9
10 if __name__ == "__main__":
11 main()
```

3 Multiplication matrice-matrice

La fonction `dot` effectue un produit de matrices. On lui passe les deux matrices et elle retourne le résultat. Si les matrices ne sont pas de dimensions compatibles pour une multiplication, le programme plante avec l'erreur "ValueError : matrices are not aligned". C'est une exception, donc on peut l'attraper et la traiter.

```
1  #!/usr/bin/python
2
3  import numpy
4
5  def main():
6      taille = 10
7      A = numpy.matrix( numpy.random.rand( taille, taille ) )
8      B = numpy.matrix( numpy.random.rand( taille, taille ) )
9      C = numpy.dot( A, B )
10     print C
11
12 if __name__ == "__main__":
13     main()
```

4 Los tres amigos

Parmi les fonctions fournies par SciPy, on peut noter la présence de *Los Tres Amigos* : LU, QR et Cholesky.

4.1 LU

La factorisation LU décompose une matrice comme le produit de deux matrices triangulaire supérieure (U = upper) et triangulaire inférieure (L = lower).

La décomposition LU est effectuée dans SciPy par la fonction `lu_factor`, qui retourne une matrice contenant la matrice L (sans les 1 de la diagonale) et la matrice U. Si la diagonale de la matrice L est composée de 1, la factorisation LU est unique.

La fonction `lu_factor` est un wrapper autour des routines `*GETRF` de LAPACK.

Une fois la matrice mise sous forme LU, la fonction `lu_solve` permet de résoudre le système d'équations AX=B.

```
#!/usr/bin/python

import numpy
import scipy.linalg

def main():
    taille = 10
    rhs = 1
    A = numpy.matrix( numpy.random.rand( taille, taille ) )
    B = numpy.matrix( numpy.random.rand( taille, rhs ) )
    # Factorisation LU de la matrice A
    C = scipy.linalg.lu_factor( A )
    # Resolution du systeme AX = B en utilisant la forme factorisee dans C
    X = scipy.linalg.lu_solve( C, B )
    print C
    print X

    res = scipy.linalg.norm( numpy.dot( A, X ) - B ) / scipy.linalg.norm( A )
    print "Residus : " + res

if __name__ == "__main__":
    main()
```

4.2 QR

```
#!/usr/bin/python

import numpy
import scipy.linalg

def main():
    taille = 10
    rhs = 1
    A = numpy.matrix( numpy.random.rand( taille, taille ) )
    [Q, R] = scipy.linalg.qr( A )
    print Q
    print R

if __name__ == "__main__":
    main()
```

4.3 Cholesky

```
#!/usr/bin/python

import numpy
import scipy.linalg

def main():
    taille = 10
    A = numpy.matrix( numpy.random.rand( taille, taille ) )
    C = scipy.linalg.cho_factor( A )
    print C
```

```
11      C = scipy.linalg.cho_solve( A )
12      print C
13
14  if __name__ == "__main__":
15      main()
```