2020_numpy

April 14, 2021

1 Tech - calcul matriciel avec numpy

numpy est la librairie incontournable pour faire des calculs en Python. Ces fonctionnalités sont disponibles dans tous les langages et utilisent les optimisations processeurs. Il est hautement improbable d'écrire un code aussi rapide sans l'utiliser.
numpy implémente ce qu'on appelle les opérations matricielles basiques ou plus communément appelées BLAS. Quelque soit le langage, l'implémentation est réalisée en langage bas niveau (C, fortran, assembleur) et a été peaufinée depuis 50 ans au gré des améliorations matérielles.

[1]:
```
from jyquickhelper import add_notebook_menu
add_notebook_menu()
```

[1]:
```
<IPython.core.display.HTML object>
```

[2]:
```
%matplotlib inline
```

1.1 Enoncé

La librairie numpy propose principalement deux types : array et matrix. Pour faire simple, prenez toujours le premier. Ca évite les erreurs. Les array sont des tableaux à plusieurs dimensions.

1.1.1 La maîtrise du slice

Le slice est l'opérateur : (décrit sur la page indexing). Il permet de récupérer une ligne, une colonne, un intervalle de valeurs.

[3]:
```
import numpy

mat = numpy.array([[0, 5, 6, -3],
                   [6, 7, -4, 8],
                   [-5, 8, -4, 9]])
mat
```

[3]:
```
array([[ 0,  5,  6, -3],
       [ 6,  7, -4,  8],
       [-5,  8, -4,  9]])
```

[4]:
```
mat[:2], mat[:, :2], mat[0, 3], mat[0:2, 0:2]
```

[4]:
```
(array([[ 0,  5,  6, -3],
        [ 6,  7, -4,  8]]), array([[ 0,  5],
        [ 6,  7],
```

```
[-5,  8]]), -3, array([[0, 5],
[6, 7]]))
```

1.1.2 La maîtrise du nan

nan est une convention pour désigner une valeur manquante. Elle réagit de façon un peut particulière. Elle n'est égale à aucune autre y compris elle-même.

[5]:
```
numpy.nan == numpy.nan
```

[5]:
```
False
```

[6]:
```
numpy.nan == 4
```

[6]:
```
False
```

Il faut donc utiliser une fonction spéciale isnan.

[7]:
```
numpy.isnan(numpy.nan)
```

[7]:
```
True
```

1.1.3 La maîtrise des types

Un tableau est défini par ses dimensions et le type unique des éléments qu'il contient.

[8]:
```
matint = numpy.array([0, 1, 2])
matint.shape, matint.dtype
```

[8]:
```
((3,), dtype('int32'))
```

C'est le même type pour toute la matrice. Il existe plusieurs type d'entiers et des réels pour des questions de performances.

[9]:
```
%timeit matint * matint
```

```
388 ns ± 9.38 ns per loop (mean ± std. dev. of 7 runs, 1000000 loops each)
```

[10]:
```
matintf = matint.astype(numpy.float64)
matintf.shape, matintf.dtype
```

[10]:
```
((3,), dtype('float64'))
```

[11]:
```
%timeit matintf * matintf
```

```
389 ns ± 2.33 ns per loop (mean ± std. dev. of 7 runs, 1000000 loops each)
```

[12]:
```
%timeit matintf * matint
```

```
691 ns ± 7.37 ns per loop (mean ± std. dev. of 7 runs, 1000000 loops each)
```

Un changement de type et le calcul est plus long.

1.1.4 La maîtrise du broadcasting

Le broadcasting signifie que certaines opérations ont un sens même si les dimensions des tableaux ne sont pas tout à fait égales.

[13]:
```
mat
```

[13]:
```
array([[ 0,  5,  6, -3],
       [ 6,  7, -4,  8],
       [-5,  8, -4,  9]])
```

[14]:
```
mat + 1000
```

[14]:
```
array([[1000, 1005, 1006,  997],
       [1006, 1007,  996, 1008],
       [ 995, 1008,  996, 1009]])
```

[15]:
```
mat + numpy.array([0, 10, 100, 1000])
```

[15]:
```
array([[   0,   15,  106,  997],
       [   6,   17,   96, 1008],
       [  -5,   18,   96, 1009]])
```

[16]:
```
mat + numpy.array([[0, 10, 100]]).T
```

[16]:
```
array([[  0,   5,   6,  -3],
       [ 16,  17,   6,  18],
       [ 95, 108,  96, 109]])
```

1.1.5 La maîtrise des index

[17]:
```
mat = numpy.array([[0, 5, 6, -3],
                   [6, 7, -4, 8],
                   [-5, 8, -4, 9]])
mat
```

[17]:
```
array([[ 0,  5,  6, -3],
       [ 6,  7, -4,  8],
       [-5,  8, -4,  9]])
```

[18]:
```
mat == 5
```

[18]:
```
array([[False,  True, False, False],
       [False, False, False, False],
       [False, False, False, False]])
```

[19]:
```
mat == numpy.array([[0, -4, 9]]).T
```

[19]:
```
array([[ True, False, False, False],
       [False, False,  True, False],
       [False, False, False,  True]])
```

[20]:
```
(mat == numpy.array([[0, -4, 9]]).T).astype(numpy.int64)
```

[20]:
```
array([[1, 0, 0, 0],
       [0, 0, 1, 0],
       [0, 0, 0, 1]], dtype=int64)
```

[21]:
```
mat * (mat == numpy.array([[0, -4, 9]]).T).astype(numpy.int64)
```

[21]:
```
array([[ 0,  0,  0,  0],
       [ 0,  0, -4,  0],
       [ 0,  0,  0,  9]], dtype=int64)
```

1.1.6 La maîtrise des fonctions

On peut regrouper les opérations que numpy propose en différents thèmes. Mais avant il

- L'**initialisation** : array, empty, zeros, ones, full, identity, rand, randn, randint
- Les **opérations basiques** : `+`, `-`, `*`, `/`, `@`, dot
- Les **transformations** : transpose, hstack, vstack, reshape, squeeze, expend_dims
- Les **opérations de réduction** : minimum, maximum, argmin, argmax, sum, mean, prod, var, std
- Tout le reste comme la génération de matrices aléatoires, le calcul des valeurs, vecteurs propres, des fonctions commme take, …

1.1.7 Q1 : calculer la valeur du χ_2 d'un tableau de contingence

La formule est là. Et il faut le faire sans boucle. Vous pouvez comparer avec la fonction chisquare de la librairie scipy qui est une extension de numpy.

$$\chi_2 = N \sum_{i,j} p_{i.} p_{.j} \left(\frac{\frac{O_{ij}}{N} - p_{i.} p_{.j}}{p_{i.} p_{.j}} \right)^2$$

[22]:
```

```

1.1.8 Q2 : calculer une distribution un peu particulière

La fonction histogram permet de calculer la distribution empirique de variables. Pour cette question, on tire un vecteur aléatoire de taille 10 avec la fonction rand, on les trie par ordre croissant, on recommence plein de fois, on calcule la distribution du plus grand nombre, du second plus grand nombre, …, du plus petit nombre.

[23]:
```

```

1.1.9 Q3 : on veut créer une matrice identité un million par un million

Vous pouvez essayer sans réfléchir ou lire cette page d'abord : csr_matrix.

[24]:
```

```

1.1.10 Q4 : vous devez créer l'application StopCovid

Il existe une machine qui reçoit la position de 3 millions de téléphones portable. On veut identifier les cas contacts (rapidement).

[25]:
```

```

1.2 Réponses

1.2.1 Q1 : calculer la valeur du χ_2 d'un tableau de contingence

La formule est là. Et il faut le faire sans boucle. Vous pouvez comparer avec la fonction chisquare de la librairie scipy qui est une extension de numpy.

$$\chi_2 = N \sum_{i,j} p_{i.} p_{.j} \left(\frac{\frac{O_{ij}}{N} - p_{i.} p_{.j}}{p_{i.} p_{.j}} \right)^2$$

[26]:
```
import numpy
O = numpy.array([[15., 20., 13.], [4., 9., 5.]])
O
```

[26]:
```
array([[15., 20., 13.],
       [ 4.,  9.,  5.]])
```

[27]:
```
def chi_square(O):
    N = numpy.sum(O)
    pis = numpy.sum(O, axis=1, keepdims=True) / N
    pjs = numpy.sum(O, axis=0, keepdims=True) / N
    pispjs = pis @ pjs
    chi = pispjs * ((O / N - pispjs) / pispjs) ** 2
    return numpy.sum(chi) * N

chi_square(O)
```

[27]:
```
0.5798254016266716
```

1.2.2 Q2 : calculer une distribution un peu particulière

La fonction histogram permet de calculer la distribution empirique de variables. Pour cette question, on tire un vecteur aléatoire de taille 10 avec la fonction rand, on les trie par ordre croissant, on recommence plein de fois, on calcule la distribution du plus grand nombre, du second plus grand nombre, ..., du plus petit nombre.

[28]:
```
rnd = numpy.random.rand(10)
rnd
```

[28]:
```
array([0.15306357, 0.39400462, 0.92104714, 0.32028784, 0.26925031,
       0.55211557, 0.64766516, 0.20248356, 0.33447531, 0.90465112])
```

[29]:
```
numpy.sort(rnd)
```

[29]:
```
array([0.15306357, 0.20248356, 0.26925031, 0.32028784, 0.33447531,
       0.39400462, 0.55211557, 0.64766516, 0.90465112, 0.92104714])
```

[30]:
```
def tirage(n):
    rnd = numpy.random.rand(n)
    trie = numpy.sort(rnd)
    return trie[-1]

tirage(10)
```

[30]:
```
0.94924486727147
```

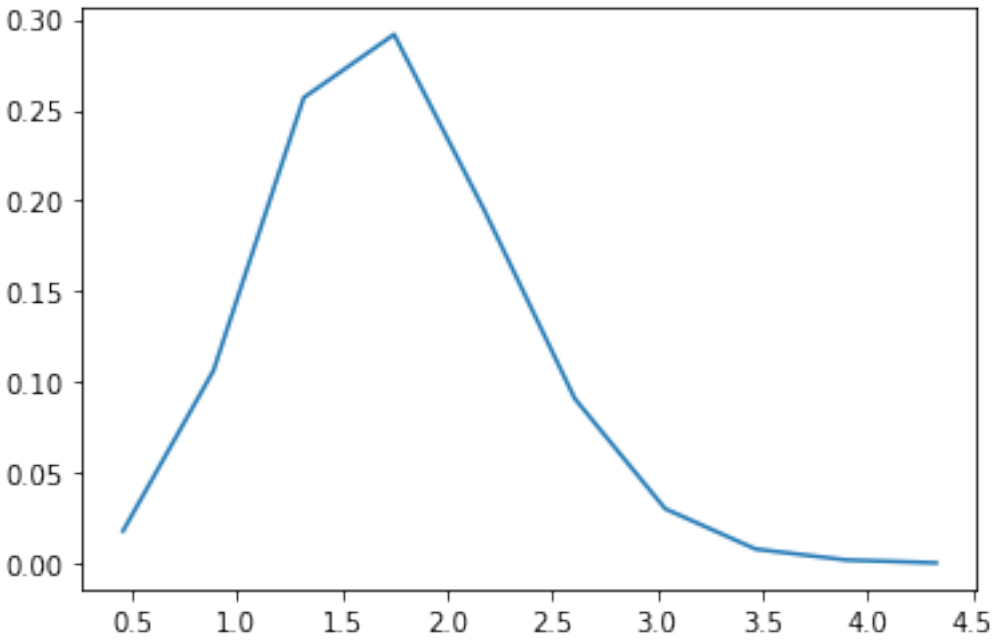
[31]:
```
def plusieurs_tirages(N, n):
    rnd = numpy.random.rand(N, n)
    return numpy.max(rnd, axis=1)

plusieurs_tirages(5, 10)
```

[31]:
```
array([2.75916927, 2.56061074, 1.79327122, 1.83069528, 1.83679582])
```

[32]:
```
t = plusieurs_tirages(5000, 10)
hist = numpy.histogram(t)
hist
```

[32]:
```
(array([  91,  532, 1284, 1458,  973,  455,  152,   41,   11,    3],
       dtype=int64),
 array([0.02832727, 0.45830362, 0.88827996, 1.31825631, 1.74823266,
        2.17820901, 2.60818535, 3.0381617 , 3.46813805, 3.8981144 ,
        4.32809074]))
```

[33]:
```
import matplotlib.pyplot as plt
plt.plot(hist[1][1:], hist[0] / hist[0].sum());
```



1.2.3 Q3 : on veut créer une matrice identité un million par un million

Vous pouvez essayer sans réfléchir ou lire cette page d'abord : csr_matrix.
$(10^6)^2 = 10^{12}$>10 Go, bref ça ne tient pas en mémoire sauf si on a une grosse machine. Les matrices creuses (ou sparses en anglais), sont adéquates pour représenter des matrices dont la grande majorité des coefficients sont nuls car ceux-ci ne sont pas stockés. Concrètement, la matrice enregistre uniquement les coordonnées des coefficients et les valeurs non nuls.

[34]:
```
import numpy
from scipy.sparse import csr_matrix
ide = csr_matrix((1000000, 1000000), dtype=numpy.float64)
ide.setdiag(1.)
```

```
c:\python372_x64\lib\site-packages\scipy\sparse\_index.py:126:
SparseEfficiencyWarning: Changing the sparsity structure of a csr_matrix is
expensive. lil_matrix is more efficient.
  self._set_arrayXarray(i, j, x)
```

1.2.4 Q4 : vous devez créer l'application StopCovid

Il existe une machine qui reçoit la position de 3 millions de téléphones portable. On veut identifier les cas contacts (rapidement).
Si on devait calculer toutes les paires de distance, cela prendrait un temps fou. Il faut ruser. Le plus simple est de construire une grille sur le territoire français puis d'associer à chaque téléphone portable la grille dans laquelle il se trouve. Dans une cellule de la grille, le nombre de paires est beaucoup plus réduit. Ce n'est pas la seule astuce qu'il faudra utiliser. Mais c'est un bon début.

[35]:
```
```