decorrelation_correction

January 8, 2019

1 1A.data - Décorrélation de variables aléatoires - correction

On construit des variables corrélées gaussiennes et on cherche à construire des variables décorrélées en utilisant le calcul matriciel. (correction)

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

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

1.1 Création d'un jeu de données

1.1.1 Q1

La première étape consiste à construire des variables aléatoires normales corrélées dans une matrice $N \times 3$. On cherche à construire cette matrice au format numpy. Le programme suivant est un moyen de construire un tel ensemble à l'aide de combinaisons linéaires. Complétez les lignes contenant des .....

```
In [2]: import random
        import numpy as np

        def combinaison():
            x = random.gauss(0,1) # génère un nombre aléatoire
            y = random.gauss(0,1) # selon une loi normale
            z = random.gauss(0,1) # de moyenne null et de variance 1
            x2 = x
            y2 = 3*x + y
            z2 = -2*x + y + 0.2*z
            return [x2, y2, z2]

        li = [ combinaison () for i in range (0,100) ]
        mat = np.matrix(li)
        mat[:5]
```

```
Out[2]: matrix([[-0.63295784, -2.80012415,  0.22050058],
                [-1.21821148, -3.04992927,  3.24455663],
                [-0.32571451, -0.93074193,  0.50069519],
                [-0.13063124, -1.07137214, -0.56615199],
                [-0.36056318, -1.50832676,  0.32408593]])
```

1.1.2 Q2

```
In [3]: npm = mat
        t   = npm.transpose ()
```

```
a   = t @ npm
a  /= npm.shape[0]
a
```

```
Out[3]: matrix([[ 0.82547076,  2.51922244, -1.58633195],
                [ 2.51922244,  9.04578378, -3.50440536],
                [-1.58633195, -3.50440536,  4.40003306]])
```

a est la matrice de covariance.

1.1.3 Corrélation de matrices

1.1.4 Q3

```
In [4]: cov = a
```

```
In [5]: var = np.array([cov[i,i]**(-0.5) for i in range(cov.shape[0])])
        var.resize((3,1))
        varvar = var @ var.transpose()
        varvar
```

```
Out[5]: array([[ 1.21142995,  0.36595362,  0.52471223],
               [ 0.36595362,  0.11054874,  0.15850718],
               [ 0.52471223,  0.15850718,  0.22727102]])
```

```
In [6]: cor = np.multiply(cov, varvar)
        cor
```

```
Out[6]: matrix([[ 1.        ,  0.92191858, -0.83236777],
                [ 0.92191858,  1.        , -0.5554734 ],
                [-0.83236777, -0.5554734 ,  1.        ]])
```

1.1.5 Q4

```
In [7]: def correlation(npm):
            t   = npm.transpose ()
            a   = t @ npm
            a  /= npm.shape[0]
            var = np.array([cov[i,i]**(-0.5) for i in range(cov.shape[0])])
            var.resize((3,1))
            varvar = var @ var.transpose()
            return np.multiply(cov, varvar)

        correlation(npm)
```

```
Out[7]: matrix([[ 1.        ,  0.92191858, -0.83236777],
                [ 0.92191858,  1.        , -0.5554734 ],
                [-0.83236777, -0.5554734 ,  1.        ]])
```

1.2 Calcul de la racine carrée

1.2.1 Q6

Le module numpy propose une fonction qui retourne la matrice P et le vecteur des valeurs propres L :

```
L,P = np.linalg.eig(a)
```

Vérifier que $P'P = I$. Est-ce rigoureusement égal à la matrice identité ?

In [8]:
```
L,P = np.linalg.eig(a)
```

In [9]:
```
P.transpose() @ P
```

Out[9]:
```
matrix([[  1.00000000e+00,  -6.13577229e-17,  -2.23247265e-16],
        [ -6.13577229e-17,   1.00000000e+00,  -1.20740141e-16],
        [ -2.23247265e-16,  -1.20740141e-16,   1.00000000e+00]])
```

C'est presque l'identité aux erreurs de calcul près.

1.2.2 Q7

`np.diag(l)` construit une matrice diagonale à partir d'un vecteur.

In [10]:
```
np.diag(L)
```

Out[10]:
```
array([[  1.17360418e+01,   0.00000000e+00,   0.00000000e+00],
       [  0.00000000e+00,   1.31745703e-03,   0.00000000e+00],
       [  0.00000000e+00,   0.00000000e+00,   2.53392830e+00]])
```

1.2.3 Q8

Ecrire une fonction qui calcule la racine carrée de la matrice $\frac{1}{n}M'M$ (on rappelle que M est la matrice `npm`). Voir aussi Racine carrée d'une matrice.

In [11]:
```
def square_root_matrix(M):
    L,P = np.linalg.eig(M)
    L = L ** 0.5
    root = P @ np.diag(L) @ P.transpose()
    return root

root = square_root_matrix(cov)
root
```

Out[11]:
```
matrix([[ 0.27891067,  0.69732306, -0.51129263],
        [ 0.69732306,  2.85042611, -0.65923845],
        [-0.51129263, -0.65923845,  1.92458244]])
```

On vérifie qu'on ne s'est pas trompé.

In [12]:
```
root @ root - cov
```

Out[12]:
```
matrix([[  0.00000000e+00,  -1.33226763e-15,  -8.88178420e-16],
        [ -8.88178420e-16,  -8.88178420e-15,   4.44089210e-16],
        [ -6.66133815e-16,   1.77635684e-15,   1.77635684e-15]])
```

1.3 Décorrélation

In [13]:
```
np.linalg.inv(cov)
```

Out[13]:
```
matrix([[ 702.44132815, -141.03278518,  140.9237308 ],
        [-141.03278518,   28.4757628 ,  -28.16665145],
        [ 140.9237308 ,  -28.16665145,   28.60079705]])
```

1.3.1 Q9

Chaque ligne de la matrice M représente un vecteur de trois variables corrélées. La matrice de covariance est $V = \frac{1}{n} M'M$. Calculer la matrice de covariance de la matrice $N = MV^{-\frac{1}{2}}$ (mathématiquement).

1.3.2 Q10

Vérifier numériquement.

1.4 Simulation de variables corrélées

1.4.1 Q11

A partir du résultat précédent, proposer une méthode pour simuler un vecteur de variables corrélées selon une matrice de covariance V à partir d'un vecteur de lois normales indépendantes.

1.4.2 Q12

Proposer une fonction qui crée cet échantillon :

```
In [14]: def simultation (N, cov) :
             # simule un échantillon de variables corrélées
             # N : nombre de variables
             # cov : matrice de covariance
             # ...
             return M
```

1.4.3 Q13

Vérifier que votre échantillon a une matrice de corrélations proche de celle choisie pour simuler l'échantillon.