Le calcul numérique en Python

Romain Tavenard

Le module `numpy` fournit, en Python, un ensemble de fonctions et d'objets utiles au calcul numérique. Nous allons nous y intéresser aujourd'hui à travers trois applications illustratives.

1 VéloStar

Le fichier `full_data_bikes.csv` disponible sur CURSUS décrit l'état de deux stations de VéloStar durant quelques heures avec un intervalle de 10 minutes entre deux relevés consécutifs.

1. Il existe, dans le module `numpy`, une fonction qui permet de charger des matrices écrites dans des fichiers texte : la fonction `loadtxt` (aide en ligne). Utilisez-la pour charger le fichier `full_data_bikes.csv` disponible sur CURSUS en faisant bien attention aux points suivants :

— ouvrez le fichier pour vous rendre compte du délimiteur utilisé ;
— remarquez que la première ligne est une en-tête et ne doit donc pas être lue par `numpy` ;
— remarquez également le type des données à charger ;
— enfin, il faut faire comprendre à `numpy` que la première colonne doit être transformée avant d'être utilisée, pour cela, la fonction `f_convert` du squelette disponible sur CURSUS pourra être utilisée.

2. Affichez la matrice ainsi chargée et ses dimensions.
3. Affichez le contenu correspondant à la station de vélo Villejean (valeur 0 sur la première colonne).
4. Vérifiez que, pour une station donnée de votre choix, le nombre total d'emplacements vélo (somme des vélos disponibles et des emplacements libres) est constante. Vous pourrez pour cela utiliser la fonction `numpy.sum` (aide en ligne).
5. Affichez la médiane des deuxième et troisième colonnes pour les ensembles d'individus suivants :

— toutes les lignes du jeu de données ;
— toutes les lignes correspondant à la station Villejean ;

— toutes les lignes correspondant à la station République.

Une médiane se calcule, en Python, à l'aide de la fonction `numpy.median` (aide en ligne).

2 Résolution d'un système d'équations inhomogène

2.1 Rappels mathématiques

Dans la suite de ce document, nous considèrerons le système suivant :

$$\begin{aligned} a_{1,1}x_1 + \cdots + a_{1,n}x_n &= b_1 \\ \vdots & \\ a_{n,1}x_1 + \cdots + a_{n,n}x_n &= b_n \end{aligned}$$

que nous envisagerons sous sa forme matricielle :

$$A \cdot x = b$$

où

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,n} \\ \vdots & \vdots & & \vdots \\ a_{n,1} & a_{n,2} & \dots & a_{n,n} \end{pmatrix}$$

$$x = (x_1, \dots, x_n)$$

$$b = (b_1, \dots, b_n)$$

Rappelons que cette équation admet une solution x_{sol} unique si et seulement si A est inversible et qu'alors cette solution est

$$x = A^{-1} \cdot b$$

Rappelons également que si A est inversible, son inverse peut s'écrire :

$$A^{-1} = \frac{1}{det(A)} \cdot com(A)^t$$

2.2 Résolution d'un tel système d'équations

D'après les formules précédentes, pour résoudre ce type de système, il nous faudra inverser la matrice A et donc calculer son déterminant et la transposée de sa comatrice. Pour ce qui est de la transposition, `numpy` fournit une fonction à cet effet, nous l'utiliserons :

```
transposee_a = numpy.transpose(a)
```

6. Pour le reste, ce sera à vous de coder les fonctions `det` et `comatrice`. De plus, le calcul du déterminant d'une matrice (n, n) faisant intervenir des sous-matrices de taille $(n-1, n-1)$, vous devrez coder une fonction `enlever_ligne_colonne` qui retourne une copie de la matrice fournie en entrée dans laquelle on a supprimé la ligne i et la colonne j. Pour cela, vous utiliserez la fonction `delete` de `numpy` (aide en ligne).
7. Une fois ces fonctions codées, vous pouvez réaliser la fonction `resoudre_systeme_lineaire_inhomogene` qui prend en entrée une matrice A, un vecteur b et retourne x_{sol}. Vous pourrez tester cette fonction à l'aide des données suivantes :

   ```
   a = numpy.array([[2, 3, 3, 1],
                    [-4, -6, 3, 2],
                    [-1, 1, 1, 1],
                    [-2, -1, 1, 1]])
   b = numpy.array([15, 3, 5, 1])
   ```

8. Cette partie du TD avait pour but de vous faire manipuler `numpy`. La fonction de résolution de systèmes que vous avez implémentée existe en fait déjà dans `numpy`. Testez le résultat obtenu avec votre implémentation à celui fourni par :

   ```
   x = numpy.linalg.solve(a, b)
   ```

3 Régression linéaire

Le fichier `ozone.txt` disponible sur CURSUS décrit des mesures de niveau d'ozone associé à des variables explicatives (température, nébulosité, *etc.*).

9. Chargez ce fichier en n'utilisant que les colonnes correspondant aux variables `maxO3`, `Ne9`, `T12` et `maxO3v` et en ignorant la première ligne. Affichez les dimensions du jeu de données ainsi chargé.

Dans la suite, nous allons chercher à ajuster un modèle linéaire aux données :

$$Y = X \cdot \beta + \varepsilon$$

où Y est un vecteur colonne correspondant à la variable à expliquer, X une matrice de dimensions $n \times d$, où n est le nombre d'individus dans le jeu de données et d le nombre de variables explicatives. Le problème consiste donc à apprendre le vecteur β de taille d tel que $\|\varepsilon\|$ soit la plus faible possible. Le vecteur $\hat{\beta}$ permettant de minimiser cette quantité s'écrit :

$$\hat{\beta} = (X^T \cdot X)^{-1} \cdot X^T \cdot Y$$

10. Calculez le vecteur $\hat{\beta}$ associé au problème pour lequel Y est la colonne `maxO3` (le niveau maximal d'ozone dans la journée) et X la colonne `Ne9` (nébulosité à 9h). Dans ce cas, $X^T \cdot X$ est une simple valeur et son inverse peut être calculé sans recours à une fonction d'algèbre linéaire.
11. Calculez $\|\varepsilon\|$ en utilisant la fonction `numpy.linalg.norm` (aide en ligne).
12. Répétez la manipulation précédente en prenant cette fois pour matrice X toutes les colonnes du jeu de données sauf la première. Dans ce cas, pour inverser $X^T \cdot X$, vous devrez utiliser la fonction `numpy.linalg.inv` (aide en ligne). Remarquez qu'en augmentant le nombre de variables explicatives, $\|\varepsilon\|$ a baissé.