

# Traitement Numérique des Données

M1 – INF 2163

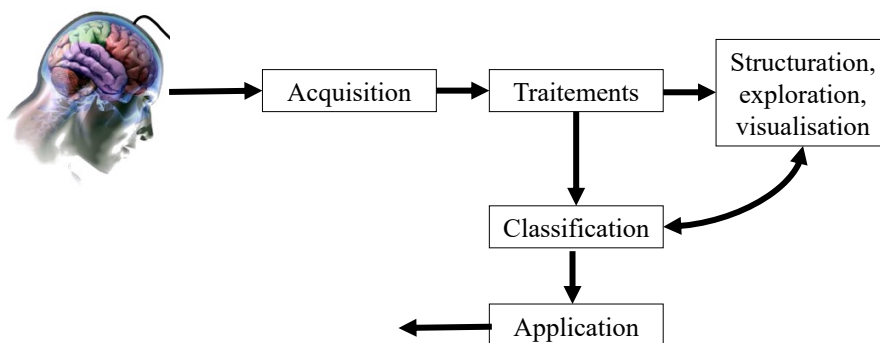
AIDN: Applications Interactives et  
Données Numériques

Sylvie Gibet

1

1

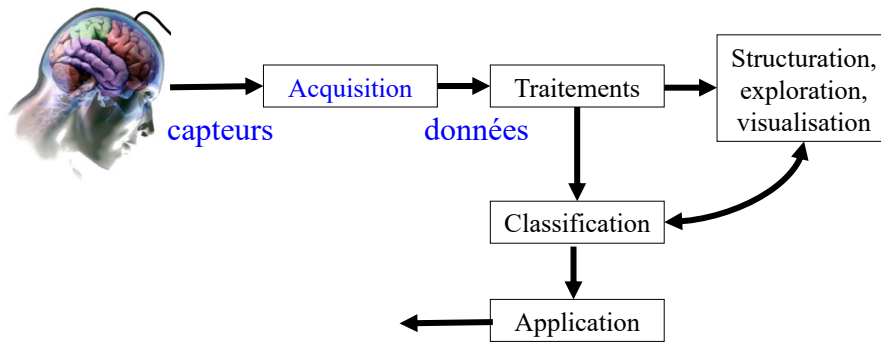
## Traitement numérique des données



2

2

# Traitement numérique des données



3

3

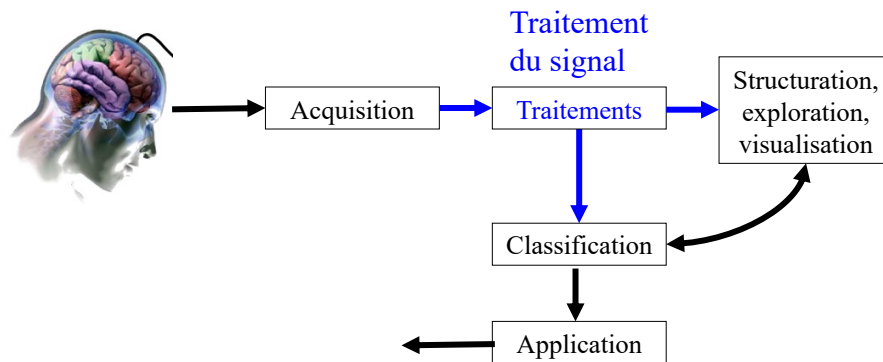
# Traitement numérique des données

- Données numériques : comment sont-elles capturées, numérisées, enregistrées
  - Des capteurs vers les données : conversion données analogiques (continues) -> données numériques
  - Théorie de l'échantillonnage
    - Fréquence d'échantillonnage (Shannon/Nyquist)
    - Représentation, analyse
    - Ré-échantillonnage (multi-sources)

4

4

# Traitement numérique des données



5

5

# Traitement numérique des données

## □ Traitement du signal :

Représenter, analyser, traiter, extraire de l'information à partir de données

### ■ Changement d'espace de représentation

Représentation de Fourier (séries de Fourier, Transformée de Fourier) : son, image, etc.

### ■ Réduction de dimension (analyse de données)

Linéaire : analyse en Composantes Principales (PCA)

Non linéaire

### ■ Filtrage

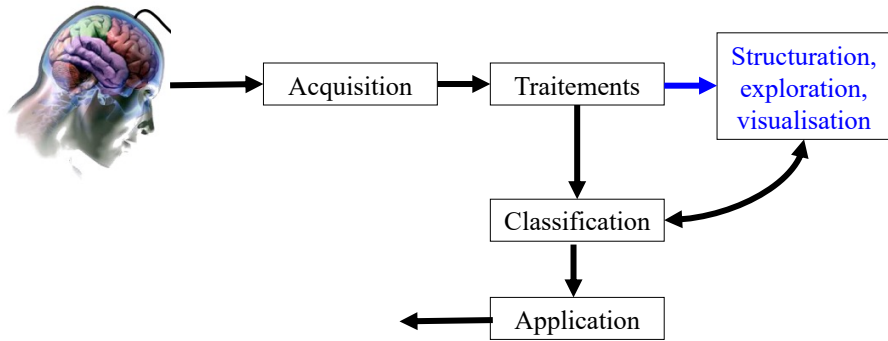
Filtres linéaires, non linéaires

Filtres spatiaux, temporels

6

6

## Traitement numérique des données



7

7

## Traitement numérique des données

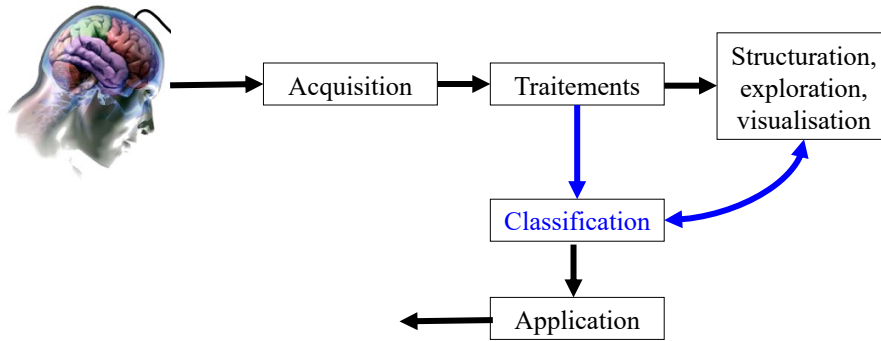
### ▣ Structuration, exploration, visualisation

- Accéder aux données
- Extraire, fusionner des données
- Faire des tests statistiques sur les données
- Visualiser les données

8

8

## Traitement numérique des données



9

9

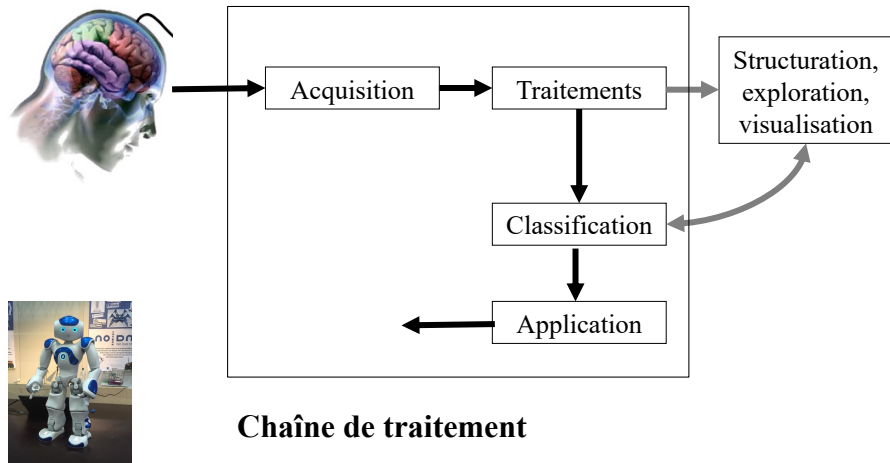
## Traitement numérique des données

- **Reconnaissance de formes, apprentissage automatique** : prédire, faire émerger des patterns (formes), classier, faire des recommandations
  - Apprendre à partir des données
    - Prédiction (régression linéaire, ...)
    - Classification supervisée
    - Classification non supervisée : clustering
    - Apprentissage par renforcement

10

10

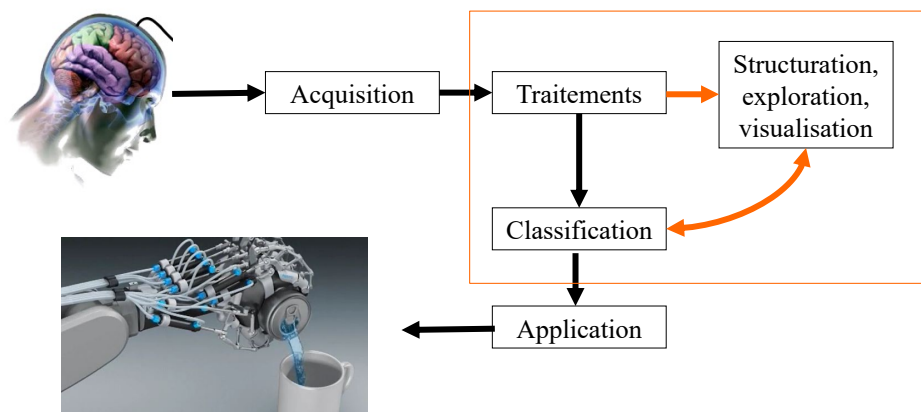
# Traitement numérique des données



11

11

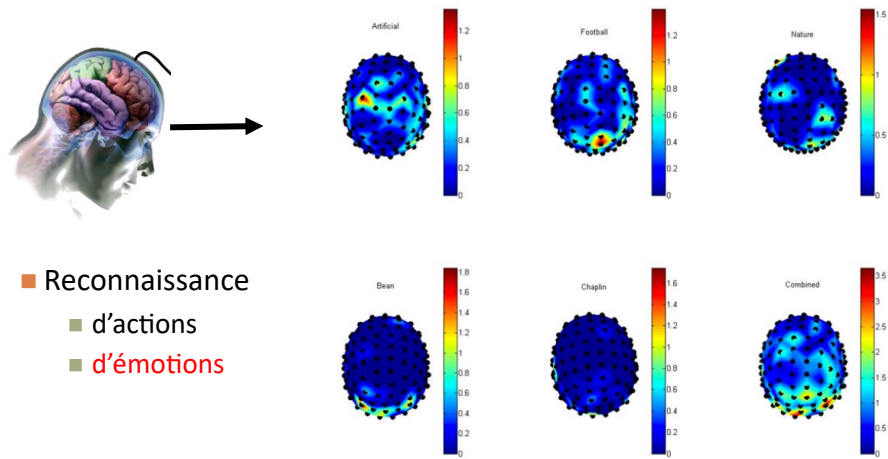
# Traitement numérique des données



12

12

## Exemple d'application : « mind-reading challenge »



13

## Exemple d'application : détection d'objets dans des images



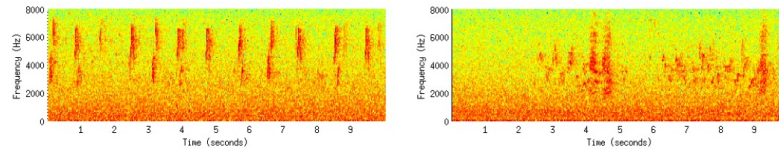
### ■ Étapes de résolution

- Sélectionner les données permettant de discriminer la/les classes données, les régions
- Capturer, enregistrer, débruiter, corriger les données
- Analyser les données (changement d'espace de représentation)
- Apprendre à partir des données
- Modéliser la frontière de discrimination à partir des données

14

14

## Exemple d'application : classification de sons



- Reconnaître des sons d'oiseaux enregistrés (IEEE MLSP 2013: Birds competition)
  - Clips audio de 19 espèces, les sons se recouvrent (645 10s clips)
  - Représentation spectrale (transformée de Fourier)
  - Classification :
    - Moitié des exemples : servent à l'apprentissage
    - Challenge : prédire l'autre moitié

15

15

## Objectifs du cours INF 2163 – TND Traitement numérique des données

- **Thématique 1 : Données numériques**
  - Théorie de l'échantillonnage : numériser des données
  - Structuration, exploration, visualisation (pandas)
- **Thématique 2 : Traitement du signal**
  - Série, transformée de Fourier discrète
  - Applications au son, à l'image, aux données capturées
- **Thématique 3 : Machine learning**
  - Apprentissage supervisé
  - Apprentissage non supervisé

16

16



# Organisation du cours

---

- **Partie théorique en aide à la partie pratique**
  - ▣ Cours hybride : présentiel, Moodle
  - ▣ Évaluation régulière des acquis de connaissance
  - ▣ Séance de questions / réponses
  - ▣ Présentation des séances de TP
- **Travaux pratiques**
  - ▣ **Autonomie !**
  - ▣ **Préparation nécessaire** : éléments donnés dans le cours
    - Préparer AVANT la séance de TP les exercices liés au cours
    - Questions de réflexion : aller plus loin en utilisant des données réelles, en développant quelques algorithmes

17

17

# Evaluation

---

- **Théorique**
  - ▣ Tests en ligne ou en présentiel
- **Pratique : programmes commentés en Python**
  - ▣ Compréhension ou réflexion sur une question

18

18

# Programmation en Python (scientifique)

- <http://www.python.org>
- Ipython (interactive shell)
- **Python 3 for scientific programming**  
<http://www.courspython.com/>
- **Tutoriel NumPy:**  
<https://www.tutorialspoint.com/numpy/index.htm>
- **Doc Numpy:** <http://www.numpy.org/>
- **Doc scipy:** <https://docs.scipy.org/doc/>
- **Doc matplotlib:**  
<https://matplotlib.org/tutorials/introductory/pyplot.html>

19

19

## NumPy: [www.numpy.org](http://www.numpy.org)

Scipy.org

### NumPy

NumPy is the fundamental package for scientific computing with Python. It contains among other things:

- a powerful N-dimensional array object
- sophisticated (broadcasting) functions
- tools for integrating C/C++ and Fortran code
- useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined. This allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

NumPy is licensed under the [BSD license](#), enabling reuse with few restrictions.

### Getting Started

- [Getting NumPy](#)
- [Installing the SciPy Stack](#)
- [NumPy and SciPy documentation page](#)
- [NumPy Tutorial](#)
- [NumPy for MATLAB® Users](#)
- [NumPy functions by category](#)
- [NumPy Mailing List](#)

For more information on the SciPy Stack (for which NumPy provides the fundamental array data structure), see [scipy.org](http://scipy.org).

About NumPy

[License](#)

[Old array packages](#)

20

20


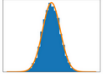
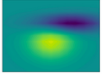
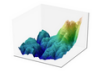
www.matplotlib.org

matplotlib

[home](#) | [examples](#) | [tutorials](#) | [pyplot](#) | [docs](#) »

### Introduction

Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and [IPython](#) shell, the [Jupyter](#) notebook, web application servers, and four graphical user interface toolkits.

Matplotlib tries to make easy things easy and hard things possible. You can generate plots, histograms, power spectra, bar charts, errorcharts, scatterplots, etc., with just a few lines of code. For examples, see the [sample plots](#) and [thumbnail gallery](#).

For simple plotting the `pyplot` module provides a MATLAB-like interface, particularly when combined with [IPython](#). For the power user, you have full control of line styles, font properties, axes properties, etc, via an object oriented interface or via a set of functions familiar to MATLAB users.

### Installation

Visit the [Matplotlib installation instructions](#).


### Documentation

This is the documentation for Matplotlib version 2.1.0.


21

21


www.scipy.org




SciPy.org




Sponsored by  
ENTHOUGHT




Install




Getting Started



Documentation




Report Bugs




Blogs

SciPy (pronounced "Sigh Pie") is a Python-based ecosystem of open-source software for mathematics, science, and engineering. In particular, these are some of the core packages:




NumPy

Base N-dimensional array package



SciPy library

Fundamental library for scientific computing




Matplotlib

Comprehensive 2D Plotting


IP[y]:  
IPython

IPython  
Enhanced  
Interactive Console



Sympy

Symbolic mathematics



pandas

Data structures & analysis

More information...

22

22

11

## Objectifs du cours INF 2163 – TND

### Traitement numérique des données

- **Thématique 1 : Données numériques**
  - ▣ **Programmation : manipulation des matrices avec numpy**
  - ▣ Théorie de l'échantillonnage : numériser les données
  - ▣ Structuration, exploration, visualisation (pandas)
- **Thématique 2 : Traitement du signal**
  - ▣ Série, transformée de Fourier discrète
  - ▣ applications au son et à l'image
- **Thématique 3 : Machine learning**
  - ▣ Apprentissage supervisé
  - ▣ Apprentissage non supervisé

23

23

## Manipulation des matrices –

### création, accès, extraction, calculs

- **Numpy : package pour Python spécialisé dans la manipulation des tableaux (array)**
    - ▣ numpy : souvent inclus dans des packages dédiés au calcul scientifique sous Python (scipy)
    - ▣ Vecteurs et Matrices (2d,Nd)
    - ▣ Les *array numpy* ne gèrent que des objets **de même type**
    - ▣ Grand nombre de fonctions pour un accès rapide aux données (recherche, extraction), pour les manipulations (tri), pour les calculs (statistique) -> voir TP1
    - ▣ Plus performants que les collections usuelles de Python : à la fois en temps de calcul et en gestion de la volumétrie
- <http://docs.scipy.org/doc/numpy/reference/index.html>

24

24

## CRÉATION D'UNE MATRICE

Création à partir de données  
Génération d'une séquence  
Chargement à partir d'un fichier

25

25

## Création d'une matrice

Importer le package numpy

`import numpy as np`

np : alias utilisé pour  
accéder aux méthodes de  
numpy

Création manuelle à partir  
de valeurs :

$$\begin{pmatrix} 1.2 & 2.5 \\ 3.2 & 1.8 \\ 1.1 & 4.3 \end{pmatrix}$$

```
a = np.array([[1.2,2.5],[3.2,1.8], [1.1,4.3]])
```

#type de la structure

```
print(type(a)) #<class 'numpy.ndarray'>
```

#type des données

```
print(a.dtype) #float64
```

#dimension

```
print(a.ndim) #2
```

#nombre de lignes et colonnes

```
print(a.shape) #(3,2) : tuple
```

#nombre de valeurs

```
print(a.size) #6
```

26

26

## Typage des données

Affichage d'une matrice  
dans la console

```
print(a) → [[ 1.2, 2.5 ]
            [ 3.2, 1.8 ]
            [ 1.1, 4.3 ]]
```

Typage des valeurs :  
implicite ou explicite

#création et typage implicite

```
a = np.array([[1,2],[3,4]])
```

```
print(a.dtype) #int32
```

#création et typage explicite : **préférable !**

```
a = np.array([[1,2],[3,4]], dtype = float)
```

```
print(a.dtype) #float64
```

27

27

## Création d'une matrice à partir d'une séquence de valeurs

#Les méthodes `arange()` et `reshape()`

```
a = np.arange(0,10).reshape(2,5) → [[ 0 1 2 3 4]
                                     [ 5 6 7 8 9 ]]
```

```
print(a)
```

#conversion d'un vecteur en matrice

```
a = np.array([2.1,3.4,6.7,8.1,3.5,7.2])
```

```
print(a.shape) # (6,)
```

#redim. en 2 lignes x 3 colonnes

```
b = a.reshape(2,3)
```

```
print(b.shape) # (2, 3)
```

```
print(b)
```

#matrices de valeurs identiques

```
a = np.zeros(shape=(2,4))
```

```
print(a)
```

#plus généralement

```
a = np.full(shape=(2,4), fill_value = 1.)
```

```
print(a)
```

```
[[ 0 1 2 3 4]
```

```
[ 5 6 7 8 9 ]]
```

`arange()` : génère une séquence de valeurs de 0 à 9

`reshape()` : réorganise en une matrice 2 lignes, 5 colonnes

**Attention, les dimensions doivent être compatibles !**

```
[[ 2.1 3.4 6.7]
```

```
[ 8.1 3.5 7.2 ]]
```

Ex : initialisation

```
[[ 0. 0. 0. 0.]
```

```
[ 0. 0. 0. 0.]
```

```
[[ 1. 1. 1. 1.]
```

```
[ 1. 1. 1. 1.]
```

28

28

## Chargement à partir d'un fichier

Les données peuvent être stockées dans un fichier texte (`loadtxt` pour charger, `savetxt` pour sauver)

	#age	poids	taille	
1	45	62	168	
2	34	68	187	
3	53	85	159	

Conversion d'une liste (type standard Python) en type array de « numpy »

#charger à partir d'un fichier, typage explicite  
#séparateur de colonne = tabulation « \t »

```
a=np.loadtxt("matrice.txt",delimiter="\t",dtype=float)
print(a)
```

```
[ [45. 62. 168.]
  [34. 68. 187.]
  [53. 85. 159. ] ]
```

#liste de valeurs

```
lst = [1.2,3.1,4.5,6.3]
print(type(lst)) # <class 'list'>
```

#conversion à partir d'une liste :

```
a = np.asarray(lst,dtype=float).reshape(2,2)
print(a)
```

```
[ [1.2 3.1]
  [4.5 6.3] ]
```

29

29

## Redimensionnement

```
[ [1.2 2.5]
  [3.2 1.8]
  [1.1 4.3] ]
```

Accoler le vecteur b en tant que nouvelle ligne (axis = 0) de la matrice a

Accoler le vecteur d en tant que nouvelle colonne (axis = 1) de la matrice a

Insertion de b en tant que nouvelle ligne (axis = 0) à la position n1

Suppression de la ligne (axis = 0) via son indice (n1)

Redimensionnement d'une matrice

#matrice de valeurs

```
a = np.array([1.2,2.5],[3.2,1.8],[1.1,4.3])
#ajouter une ligne
b = np.array([[4.1,2.6]])
c = np.append(a,b,axis=0)
print(c)
```

#ajouter une colonne

```
d = np.array([[7.8],[6.1],[5.4]])
print(np.append(a,d,axis=1))
```

#insertion

```
print(np.insert(a,1,b,axis=0))
```

#suppression

```
print(np.delete(a,1,axis=0))
```

#modifier la dimension

```
Parcourir les données ligne par ligne
h = np.resize(a,new_shape=(2,3))
print(h)
```

```
[ [1.2 2.5]
  [3.2 1.8]
  [1.1 4.3] ]
```

```
[ [1.2 2.5 7.8]
  [3.2 1.8 6.1]
  [1.1 4.3 5.4] ]
```

```
[ [1.2 2.5]
  [4.1 2.6]
  [3.2 1.8]
  [1.1 4.3] ]
```

```
[ [1.2 2.5]
  [1.1 4.3] ]
```

```
[ [1.2 2.5 3.2]
  [1.8 1.1 4.3] ]
```

30

30

## EXTRACTION DES VALEURS

31

31

## Accès indicé

```
v = np.array([[1.2,2.5],[3.2,1.8],[1.1,4.3]])
#affichage de la structure dans son ensemble
print(v)
#accès indicé - première valeur
print(v[0,0]) # 1.2
#dernière valeur – noter l'utilisation de shape (qui est un tuple)
print(v[v.shape[0]-1,v.shape[1]-1]) # 4.3
#autre solution pour affichage de toutes les valeurs, noter le rôle des :
print(v[:,:])
#plage d'indices contigus : lignes 0 à 1 (2 non inclus), toutes les colonnes
print(v[0:2,:])
#extrêmes, début to 2 (non-inclus)
print(v[:2,:])
#extrêmes, lignes 1 à dernière
print(v[1,:])
#indice négatif – dernière ligne et toutes les colonnes
print(v[-1,:])
#indices négatifs – lignes -2, -3 (soit indices lignes 1, 0) et toutes les colonnes
print(v[-2,:])
```

```
[[1.2 2.5]
 [3.2 1.8]
 [1.1 4.3]]

[[1.2 2.5]
 [3.2 1.8]]

[[1.2 2.5]
 [3.2 1.8]]

[[3.2 1.8]
 [1.1 4.3]]

[1.1 4.3]

[[3.2 1.8]
 [1.1 4.3]]
```

32

32



## Accès par condition

```
#indigage par vecteur de booléens
#si b trop court, tout le reste est considéré False
b = np.array([True,False,True],dtype=bool) print(v[b,:])

#exemple illustratif : extraire la ligne dont la somme est la plus petite
#calculer la somme des colonnes pour chaque ligne
s = np.sum(v,axis=1)
print(s) # [ 3.7  5.  5.4 ]

#repérer les lignes dont la somme est égale au minimum
#il est possible qu'il y en ait plusieurs
b = (s == np.min(s))
print(b) # [ True False False]

#application du filtre booléen
print(v[b,:])
```

$v = \begin{bmatrix} 1.2 & 2.5 \\ 3.2 & 1.8 \\ 1.1 & 4.3 \end{bmatrix}$

$\begin{bmatrix} 1.2 & 2.5 \\ 1.1 & 4.3 \end{bmatrix}$

$\begin{bmatrix} 1.2 & 2.5 \end{bmatrix}$

33

## Tri et recherche

```
#recherche valeur max des lignes (axis = 0) pour chaque colonne
print(np.max(v,axis=0)) # [ 3.2  4.3 ] -- décryptage : 3.2 est le max des lignes pour la
colonne 0, 4.3 est le max des lignes pour la colonne 1
#recherche valeur max des colonnes (axis = 1) pour chaque ligne
print(np.max(v,axis=1)) # [ 2.5  3.2  4.3]
#recherche indice de valeur max des lignes (axis = 0) pour chaque colonne
print(np.argmax(v,axis=0)) # [ 1  2 ]

#tri des lignes (axis = 0) pour chaque colonne
#la relation entre les valeurs d'une même ligne est perdue !!!
print(np.sort(v,axis=0))

#récupération des indices triés
print(np.argsort(v,axis=0))
```

$v = \begin{bmatrix} 1.2 & 2.5 \\ 3.2 & 1.8 \\ 1.1 & 4.3 \end{bmatrix}$

$\begin{bmatrix} 1.1 & 1.8 \\ 1.2 & 2.5 \\ 3.2 & 4.3 \end{bmatrix}$

$v = \begin{bmatrix} 2 & 1 \\ 0 & 0 \\ 1 & 2 \end{bmatrix}$

34

34

# ITÉRATIONS

35

35

## Parcours d'une matrice – boucle indicée

Avec les indices, nous pouvons accéder aux valeurs de la matrice comme bon nous semble  
(ligne par ligne ou colonne par colonne)

#boucles indicées

s = 0.0

for i in range(0,v.shape[0]):

    for j in range(0,v.shape[1]):

        print(v[i,j])

        s = s + v[i,j]

print("Somme = ",s)

v = [ [1.2 2.5 ]  
      [3.2 1.8]  
      [1.1 4.3] ]

1.2

2.5

3.2

1.8

1.1

4.3

Somme = 14.1

36

36

## Parcours d'une matrice – itérateurs

Avec les itérateurs, nous pouvons accéder aux valeurs de la matrice sans avoir à recourir aux indices (ligne par ligne, colonne par colonne)

```
v = [ [1.2 2.5]
      [3.2 1.8]
      [1.1 4.3] ]
```

#itérateur - accès ligne par ligne

```
s = 0.0
for x in np.nditer(v):
    print(x)
    s=s+x
print("Somme = ",s)
```

1.2  
2.5  
3.2  
1.8  
1.1  
4.3  
Somme = 14.1

#itérateur - accès colonne par colonne

#"F" pour " Fortran order "

```
s = 0.0
for x in np.nditer(v,order="F"):
    print(x)
    s = s +x
print("Somme = ",s)
```

1.2  
3.2  
1.1  
2.5  
1.8  
4.3  
Somme = 14.1

Les itérateurs de NumPy sont sophistiqués et puissants, voir :  
<http://docs.scipy.org/doc/numpy/reference/arrays.nditer.html>

37

## CALCUL MATRICIEL

38

38

## Fonctions matricielles

---

```
x = [[1.2  2.5 ]
      [3.2  1.8]
      [1.1  4.3]]
y = [ [2.1  0.8]
      [1.3  2.5]]

#transposition
print(np.transpose(x))

#multiplication
print(np.dot(x,y))

#déterminant
print(np.linalg.det(y))

#inversion
print(np.linalg.inv(y))
```

39

39

## Fonctions matricielles

---

```
x = [[1.2  2.5 ]
      [3.2  1.8]
      [1.1  4.3]]
y = [ [2.1  0.8]
      [1.3  2.5]]

#résolution d'équation : y.a = z (a = y-1.z)
z = np.array([1.7,1.0]) print(np.linalg.solve(y,z)) # [0.8195 -0.0261]

#vérification
print(np.dot(np.linalg.inv(y),z)) # [0.8195 -0.0261]

#matrice symétrique avec XTX
s = np.dot(np.transpose(x),x)
print(s)

#valeurs et vecteurs propres d'une matrice symétrique
print(np.linalg.eig(s))
```

40

40