Logiciels Scientifiques

Licence Mathématiques

Licence MIASHS

Mineur Mathématiques

$2^{\text{ème}}$ année

Cours 10

Andrzej Stos

Organisation et remarques pratiques

- Épreuve de CC1 / TP noté : feuilles TP 1 - TP 5 (TP 4 ?)
  $\hookrightarrow$ dans un créneau de TP, voir avec son prof de TP.
- Épreuve de CC2 (individuelle) à la fin de semestre, sur l'ensemble des TP
- Un rattrapage éventuel : en janvier, sur l'ensemble des TP
- Sont autorisées : tous les documents distribués, vos programmes TP, vos notes. Sont interdites : calculettes, portables, accès ENT, internet.
- **L'ENT est interdit**
  $\hookrightarrow$ téléchargez tous vos fichiers sur le disque **M**.
  En particulier, téléchargez le cours sur le disque **M** aussi (ou préparez une version papier).
- Les clés USB ne fonctionnent pas dans toutes les salles de TP !
  $\hookrightarrow$ téléchargez vos fichiers sur le disque **M**, ne tardez pas jusqu'à dernière minute.

Organisation et remarques pratiques

- Pendant les épreuves le travail se fait sur le disque **L**.

  $\hookrightarrow$ Seulement les fichiers sur le disque **L** seront prises en compte à la fin de l'épreuve !

  $\hookrightarrow$ Avant le début, créer sur ce disque 3 fichiers :
  `exercice1.py`, `exercice2.py`, `exercice3.py`
  (1 fichier par exercice, pas besoins de votre nom, pas de sous-dossiers, juste 3 fichiers).

Calcul formel avec `sympy` (TP 5)

- Importation classique : `import sympy as sm`
- Importation explicite pour des fonctions / commandes fréquentes :
  `from sympy import sin, pi`
  $\hookrightarrow$ simplifie l'écriture, e.g. `sin(pi)` au lieu `sm.sin(sm.pi)`)
- `from sympy import pprint`
  $\hookrightarrow$ améliore l'affichage de formules mathématiques
- `sympy` redéfinit les fonctions usuelles !
  $\hookrightarrow$ `sm.sin` est **incomaptible** avec les tableaux `numpy` et avec les graphiques `matplolib` !

  `sympy` fournit sa propre commande `plot`.
- NB. Pour ne pas confondre les fonctionalités de `numpy` et `sympy`, éviter l'importation totale `from sympy import *`.

Calcul formel avec sympy (TP 5)

Variable *formelle* (ou *symbolique*) : sans valeur numérique
Déclaration : `sm.var('x y z')`

Manipulation : `p = x + y + 2*x` $\hookrightarrow$ donne `3*x + y`
`p.subs(x, 2)` $\hookrightarrow$ donne `6 + y`
`p.subs(x, 2).subs(y, 1)` $\hookrightarrow$ donne `7`.

Calcul numérique (`float`) est souvent approché :

```
import numpy as np
a = np.sqrt(10)
print(a**2 - 10)      ↪ 1.7763568394002505e-15
print(np.sin(np.pi / 4))      ↪ 0.7071067811865474
```

Calcul formel est **exact** :

```
import sympy as sm
print(sm.sin(sm.pi / 4))      ↪ sqrt(2) / 2
a = sm.sqrt(10)
print(a**2 - 10)      ↪ 0
```

Quelques astuces sympy

- `sympy` ne reconnait pas $x^{0.5}$ ou $x^{1/2}$. Utilisez `sqrt` !
- `solve` résout un système d'équations (avec plusieurs inconnues) et retourne un *dictionnaire*. On peut utiliser ce dictionnaire pour vérifier la solution à l'aide de `.subs`. Exemple :

  ```
  S = solve([eq1, eq2, eq3], [x, y, z])
  eq1.subs(S)
  ```

- Parfois `solve(eq, x)` donne une solution rationnelle avec un développement décimal infini (e.g. `x = 2/3`).

  Alors vérification `eq.subs(x, 2/3)` ne donne pas exactement 0.

  On peut effectuer vérification exacte avec, par exemple, `eq.subs(x, Rational(2, 3))`.

Analyse des données : lire des fichiers (TP6)

On va travailler seulement des fichiers texte (à lignes)
On va traiter une ligne à la fois :

```
with open("M:/LogSci/test.txt", encoding="utf-8") as f:
    for ligne in f:
        print(ligne)
        # mémoriser des éléments nécessaires, faire une analyse...
```

Attention : la variable `ligne` est une chaîne de caractères qui se termine par un caractère spécial NEWLINE.

Pour la supprimer : `ligne = ligne[:-1]`

Analyse de données : fichiers structurés CSV (tableur)

Fichier `cac40.csv`
l'identifiant de la valeur, la date, le cours d'ouverture, le plus haut, le plus bas, le cours de clôture et le volume de titres échangés.

Exemple :

FR0003500008 ;02/01/06 ;4731,92 ;4757,54 ;4727,09 ;4754,92 ;1277809
FR0003500008 ;03/01/06 ;4761,95 ;4803,23 ;4755,72 ;4776,98 ;3929618
FR0003500008 ;04/01/06 ;4820,90 ;4838,52 ;4799,19 ;4838,52 ;4546846

Lecture, extraction de cours d'ouverture :

```
co = [ ]
with open("M:/LogSci/cac40.csv", encoding="utf-8") as f:
    for ligne in f:
        l = ligne.replace(",", ".")
        donnees = l.split(";")
        co.append(float(donnees[2]))
```

- Parfois on utilise d'autres séparateurs (virgule, espace).
- Une seule commande open pour multiple données suffit !

Analyse de données : fichiers structurés CSV (tableur)

Remarques pratiques

- L'explorateur de fichiers Windows cache l'extension `"csv"`. Ne le rajoutez pas à la main (`"cac40.csv.csv"`)
- Attention ! On **évitera d'ouvrir les fichiers avec Excel ou Open Office** (risque d'endommager des données) !

  $\hookrightarrow$ On *pourrait éventuellement* faire une copie du fichier.
- Solution en Python pour voir un aperçu d'un fichier

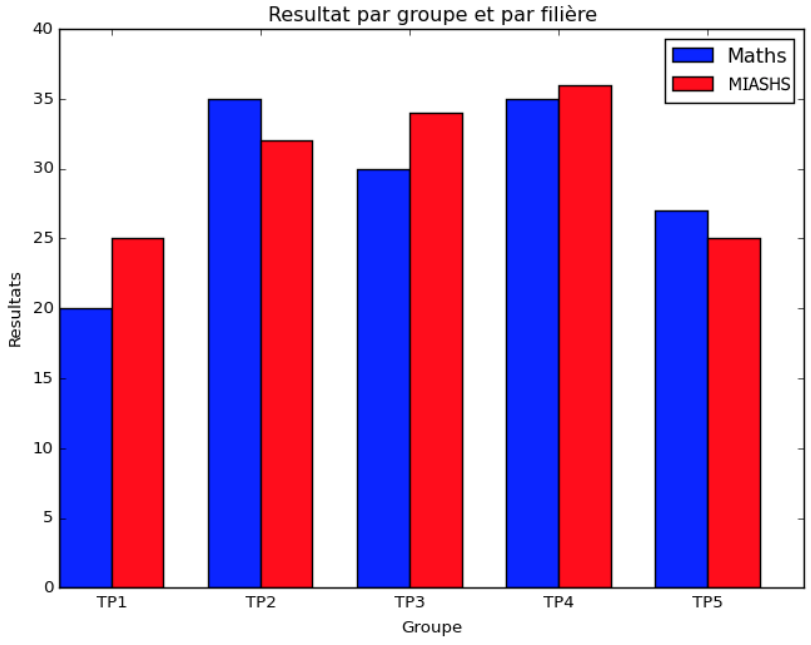
```
with open("cac40.csv", encoding = "utf-8") as f:
    for _ in range(5):
        print(f.readline())
```

- **Attention !** Il ne faut pas confondre `f.readline()` avec `f.readlines()`. Pour ne pas saturer la mémoire, on **s'interdit** d'utiliser `f.readlines()` en TP !

Analyse de données : diagramme en bâtons

```
n_groupes = 5
moyenne_maths = (20, 35, 30, 35, 27)
moyenne_miashs = (25, 32, 34, 36, 25)

index = np.arange(n_groupes)
largeur = 0.35
plt.bar(index, moyenne_maths, largeur,
color='b', label='Maths')
plt.bar(index + largeur, moyenne_miashs,
largeur, color='r', label='MIASHS')
plt.xlabel('Groupe')
plt.ylabel('Resultats')
plt.title('Resultat par groupe et par
filière')
plt.xticks(index + largeur, ['TP1', 'TP2
'TP3', 'TP4', 'TP5'])
plt.legend()
```



Attention : **Ne pas copier-coller** à partir d'un pdf !

Numérotation avec `enumerate`

```
L = ['ala', 'ma', 'kota' ]
for i, item in enumerate(L):
    print(i, item)
```

↪

```
0 'ala'
1 'ma'
2 'kota'
```

Compléments de Python

Compréhensions : accumuler des résultats d'un calcul en boucle

Exemple 'classique'

```
L = [ ]
for i in range(100):
    L.append(mon_calcul(i))
```

Construction élégante :

`L = [mon_calcul(i) for i in range(100)]` $\hookrightarrow$ utile en TP 7 !

Cette construction peut aussi remplacer `map` :

```
def decale(x, dec):
    return x + dec
L = [5, 6, 7, 8]
K = [decale(x, i) for (i, x) in enumerate(L)]
```

Extrema

Souvent on est amené a calculer le minimum (maximum) d'une liste `L` et l'*indice* d'élément minimal (maximal)

minimum : `min(L)`

maximum : `max(L)`

avec `numpy` :

indice d'élément minimal : `np.argmin(L)`

indice d'élément maximal : `np.argmax(L)`

$\hookrightarrow$ utile par exemple pour TP 6.

Compléments de Python

Listes imbriquées : listes composées de listes.

Exemple 1 : représenter une liste d'entrées (éléments) d'une base de données ou d'une feuille de tableur.

```
L = [ ['Roger', 'Moore', 1973], ['Sean', 'Connery', 1962],
    ['Pierce', 'Brosnan', 1995], ['Daniel', 'Craig', 2006]]
```

Cela représente aussi un 'tableur' à 4 lignes et 3 colonnes.

```
print(len(L))
↪ 4
```

Exemple 2 : représenter une matrice. Soit

$$M = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

En Python :

```
M = [ [1, 2, 3], [4, 5, 6], [7, 8, 9] ]
```

Trier une liste

```
L = [4, 7, 8, 2, 3, 1, 5, 9, 0]
```

- Trier sur place (en modifiant la liste originale) :
  `L.sort()`
- Trier dans l'ordre inverse (descendant)
  `L.sort(reverse=True)`
- Trier une liste sans la modifier :
  `K = sorted(L)`
- Trier selon un critère (sur place) :
  Exemple : trier une liste de réels selon la distance à $r = 3.14$

```
def distance(x):
    return abs(x - 3.14)

L.sort(key=distance)
print(L)
↪ [3, 4, 2, 5, 1, 0, 7, 8, 9]
```

Simulations aléatoires - vocabulaire de base (TP 7)

- On effectue une expérience dont les issues sont aléatoires (exemple : jeu de pile ou face).
- La probabilité d'un événement s'estime par sa *fréquence* observée dans une simulation.
  Exemple : si la pièce de monnaie est symétrique, on s'attend à ce que dans une longue série de lancers, la fréquence observée des piles soit proche de 1/2.
- Si à chaque issue on affecte une valeur (e.g. "pile = 0, face = 1"), alors on peut s'intéresser à l'espérance mathématique (gain moyen). En simulation, ce sera juste la moyenne des valeurs observées. On l'appelle *moyenne empirique*.
- L'information complète sur notre expérience, c'est l'ensemble des probabilités pour toutes les issues possibles (la loi). Si on parle d'une simulation, on l'appelle la *loi empirique*, la répartition des résultats. On la represente à l'aide d'un *histogramme* (`plt.hist`).

Simulations aléatoires - outils (TP 7)

Source de "hasard" = générateur des nombres pseudo-aléatoires. Les deux commandes suivantes du module `numpy.random` seront à la base de toutes nos simulations.

1. `rand()` – un nombre au hasard dans [0,1], de façon *uniforme* ; `rand(n)` produit un *tableau* de `n` éléments.
   Propriété importante : si $p \in [0,1]$, alors

   $$\mathbb{P}(\text{rand}() \in [0,p]) = p.$$

   Cela permet de simuler, par exemple, une pièce *biaisée*, où un coté est plus probable que l'autre. Autrement dit :

   ```
   if rand() < p:
       return 1
   else:
       return 0
   ```

2. `randint(a, b)` produit un entier dans $[a, b-1]$ de façon équiprobable ; `randint(a, b, n)` produit un *tableau* de longueur `n`.

Simulations aléatoires - structure du programme (TP 7)

1. Écrire une *fonction* qui réalise l'expérience.

2. Lancer cette fonction un *grand nombre de fois* pour obtenir des statistiques.

Exemple : Quelle est la probabilité que dans une série de 5 lancers d'une pièce, le nombre de Piles est supérieur à 4 ?

Représentation : 1 = Pile, 0 = Face

5 lancers : `randint(0, 2, 5)`.

Simulations aléatoires - structure du programme (TP 7)

Première étape : écrire une fonction

```
def cinq_lancers():
     return randint(0, 2, 5)
```

Deuxième étape : exécuter un *grand nombre* de simulations (possiblement mais pas forcément dans une fonction)

```
nbr_expe = 10000      # plus grand c'est, mieux c'est
fav = 0
for _ in range(nbr_expe):
    if sum(cinq_lancers) >= 4:
        fav += 1
print("Proba(N_Piles >= 4) = ", fav / nbr_expe)
```

Remarque : Donnez toujours une description avant le résultat numérique.

Simulations aléatoires - histogramme (TP 6 et 7)

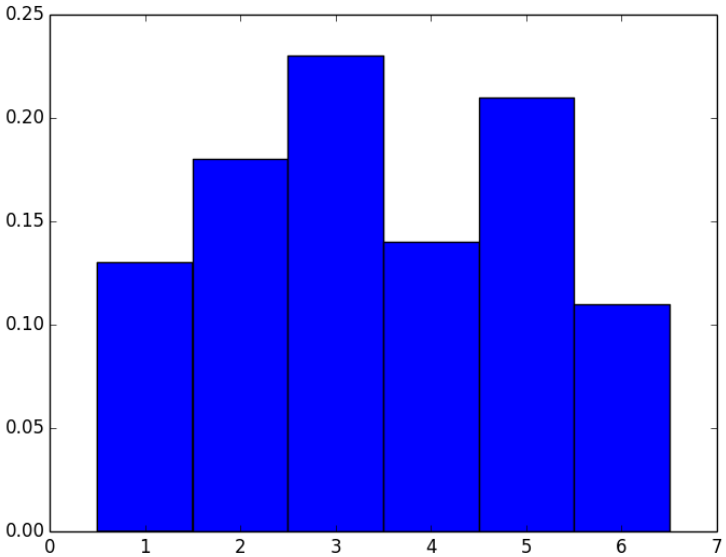
Histogramme visualise la répartition des données (ou loi empirique)

Schéma fondamental :

- construire (lire, calculer...) une liste de données
- utiliser la commande `hist` avec des bons paramètres.

Exemple :

```
from numpy.random import randint
L = randint(1, 7, 100)
plt.hist(L, bins=np.arange(0.5, 7, 1), normed=1)
```



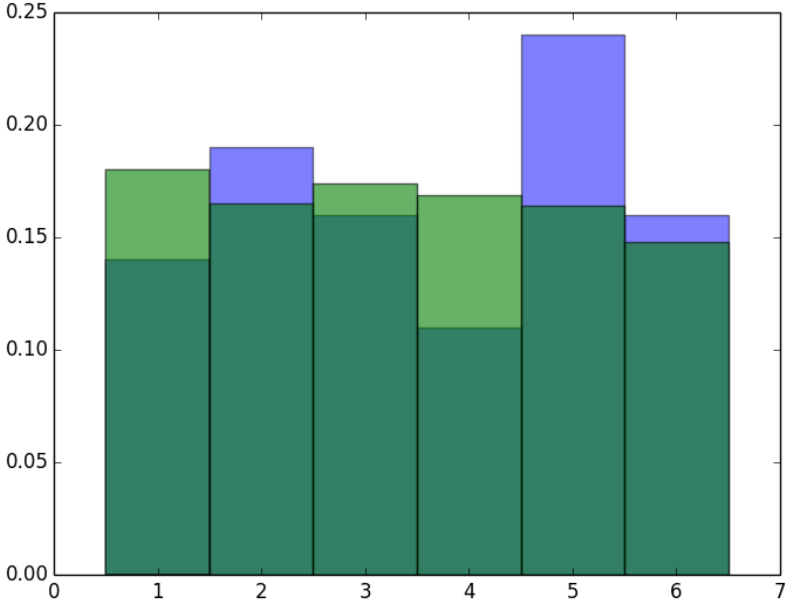
`bins` :
définition des boîtes

`normed=1` :
histogramme représente des probabilités

Simulations aléatoires - histogramme (TP 6 et 7)

Superposition de deux histogrammes : taille d'échantillon

```
L = randint(1, 7, 100)
plt.hist(L, bins=np.arange(0.5, 7, 1), normed=1,
alpha=0.5)
L = randint(1, 7, 1000)
plt.hist(L, bins=np.arange(0.5, 7, 1), normed=1,
alpha=0.6)
```



alpha = paramètre de transparence

en bleu : taille 100

en vert : taille 1000

$\hookrightarrow$ En statistiques, la taille d'échantillon est toute importante !

Simulations aléatoires - estimation (TP 7)

Nombre d'expériences et l'intervalle de confiance

Une série de simulations avec `nbr_expe = 1000` :

Proba(Piles ≥ 4) $= 0.2$
Proba(Piles ≥ 4) $= 0.1960$
Proba(Piles ≥ 4) $= 0.184$
Proba(Piles ≥ 4) $= 0.178$

Proba(Piles≥ 4) $= 0.191\ldots$

Une variabilité d'ordre 0.02. On peut deviner que la vraie valeur est environ 0.19... ou 0.185... ou 0.195...

Question : comment quantifier la précision d'estimation ?

Réponse mathématique : Intervalle de confiance !
Si f est la fréquence observée dans un échantillon de taille n, alors l'intervalle de confiance au niveau 95% est le suivant :

$$\left[f - \frac{1}{\sqrt{n}},\ f + \frac{1}{\sqrt{n}} \right]$$

La vraie valeur de p est dans cet intervalle avec la probabilité 95%.

Simulations aléatoires - estimation (TP 7)

Précision d'estimation

$$\left[f - \frac{1}{\sqrt{n}},\ f + \frac{1}{\sqrt{n}}\right]$$

Formule simplifiée (niveau Seconde) ! Heuristique :

- Si, on veut que l'erreur d'estimation soit d'ordre 0.01 (dans 95% de cas...), il faut au moins $n = 10000$ simulations.
- Pour l'erreur d'ordre 0.001, il faudra au moins 1 million expériences

NB. Commencez les simulations avec un petit nombre d'expériences (e.g. `nbr_expe = 10`), puis l'augmentez progressivement.
N'ayez pas peur de calculs qui durent quelques secondes (10s - 20s).