M1 Physique Appliquée — Université Paris-Sud XI
Année 2015-2016

Programmation et données numériques

TD12 et 13 : Incertitudes et Régression linéaire

Pour la note de contrôle continu du second semestre, il faut envoyer un rapport sur les TD12 et 13 au format pdf ou html par email à `guillaume.roux@u-psud.fr` *et* `aurelien.grabsch@u-psud.fr` *pour la semaine suivant le TD13 (pour le jeudi 28 janvier pour les groupes 2 et 3 et le jeudi 4 Février pour le groupe 1). On y inclura les lignes de codes utilisées ainsi que les graphes produits. Chaque jour de retard entraînera une diminution de 4 points de la note.*

Utilisation de l'environnement `IPython` et de `Spyder`

Plutôt que d'utiliser le `notebook` de `IPython` sous la distribution `Anaconda` qui nous a posé quelques soucis, nous allons utiliser un environnement de travail plus proche de l'utilisation initialement prévue de Python.

- ☞ `Windows` : une distribution `Python-xy` a été installée. La démarrer et choisir de lancer `Spyder`.
- ☞ `Linux` : une solution est de lancer une console `IPython` par la ligne de commande

  ```
  >>>ipython qtconsole
  ```

  Puis d'utiliser systématiquement (les fenêtres graphiques ne semblent pas fonctionner avec la configuration actuelle des ordinateurs) :

  ```
  >>>matplotlib inline
  ```

  pour que les graphes s'affichent dans la console. Vous pouvez ensuite éditer vos fichiers `.py` avec un éditeur comme `gedit` ou autre.

Introduction à IPython

Le principe est le suivant : il y a une console, ou ligne de commande dans laquelle on peut taper des instructions en Python avec `In` et `Out` pour les entrée et sorties comme pour le notebook. Pour écrire des modules ainsi que des scripts, il faut en parallèle éditer des fichiers sources `.py` contenant un code sur plusieurs ligne. `Spyder` offre une interface ou l'éditeur se trouve à gauche et la ligne de commande à droite. Il faut que vos fichiers `.py` soient dans le dossier courant. Vous trouverez ci-dessous quelques commandes utiles, que l'on peut aussi afficher avec la commande

```
quickref
```

Fichiers manipuler et naviguer dans l'arborescence des dossiers et des fichiers :

```
pwd # affiche le dossier courant
cd  # remonte à la racine du nom d'utilisateur
cd MonDossier/ # se rend dans le dossier "MonDossier/"
               # (penser à utiliser la touche TAB pour la complétion automatique)
cd ..          # remonte d'un cran dans l'arborescence
mkdir MonDossier/ # crée le dossier "MonDossier/"
rmdir MonDossier/ # crée le dossier "MonDossier/"
ls    # affiche le contenu du dossier
%copy fichier1 fichier2  # copie de fichiers
more fichier  # affiche le contenu du fichier dans la console
edit fichier  # lance l'éditeur de fichier
```

Exécuter des scripts On peut faire `f5` qui exécute la feuille active de l'éditeur. On notera que le bouton en haut à droite de la console permet également de relancer le noyau Python si besoin.

```
run script.py    # exécute script.py
run script       # exécute aussi script.py
rerun  # réexécute la dernière commande
reset  # toutes les variables définies sont effacées
precision 5 # règle la précision de l'affichage à 5 décimales
timeit [math.sin(x) for x in range(5000)] # évalue le temps d'exécution d'une commande
timeit %run script.py # exécute plusieurs fois le script
                      # et évalue le temps d'exécution
matplotlib inline  # permet d'afficher les graphes dans l'output
```

Rechargement automatique des librairies Lorsque vous allez créer et modifier un module/librairie et que vous l'avez déjà chargé une fois, `IPython` ne le rechargera pas automatiquement même s'il a été modifié. Pour rendre ce rechargement automatique, ce qui sera très utile pour ce TD, vous pouvez taper dans la console IPython les lignes suivantes :

```
>>> load_ext autoreload
>>> autoreload 2
```

Naviguer dans l'historique des commandes

- ☞ avec les 4 flèches.
- ☞ `Ctrl-a` et `Ctrl-e` pour atteindre les début et fin de ligne.
- ☞ `Ctrl-k` pour couper la fin de la ligne.
- ☞ `Ctrl-y` pour coller ce qu'il y a dans le presse-papier.
- ☞ utiliser `Tab` pour la complétion des noms.

Création d'un module On va maintenant utiliser cela pour créer un module, ou librairie, que l'on appelle `Statistique` et qui doit par conséquent être écrit dans un fichier appelé `Statistique.py`

- ☞ Créer un fichier `Statistique.py`. Y inclure les définitions suivantes que l'on complétera, `d` étant une liste. On importe `numpy` et `pyplot` afin de disposer par la suite de toutes les fonctions mathématiques ainsi que des autres fonctions utiles de numpy.

```
import numpy as np
import matplotlib.pyplot as plt

def moyenne(d):
    if len(d): return  .....
    else:      return 0.0

def variance(d):
    if len(d) < 2: return 0.0
    moy = .....
    return sum([ .....  for x in d ])/(len(d)-1.0)
```

- ☞ Vous pouvez alors importer les fonctions de cette librairie ou bien la librairie en entier, ou bien la librairie renommée

```
from Statistique import moyenne, variance
from Statistique import *
import Statistique as stat
```

Faites un essai de l'utilisation de ces fonctions avec une liste de votre choix en utilisant la ligne de commande. Essayer les différents modes d'importation en utilisant si besoin la commande `reset` pour effacer les précédentes importations.

☞ De même, vous pouvez créer un script que l'on appelera `script.py` et utilisant ces fonctions. Lancez le script depuis la ligne de commande `IPython`.

☞ Écrire un script `scriptPlot.py` qui affiche un plot selon

```
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(0,10,101)
y = np.sqrt(x)
plt.plot(x,y)
plt.show()
```

Autour du coefficient de corrélation linéaire

On rappelle la définition du coefficient de corrélation linéaire r pour une série de mesure de couples (x_k, y_k) :

$$r = \frac{\sum_k (x_k - \bar{x})(y_k - \bar{y})}{\sqrt{\sum_k (x_k - \bar{x})^2 \sum_k (y_k - \bar{y})^2}} \tag{1}$$

On souhaite écrire une fonction qui effectue ce calcul, puis l'utiliser pour illustrer son comportement.

☞ Quelles sont les valeurs possibles de r si les points (x_k, y_k) sont alignés selon une droite ?

☞ Compléter la fonction ci-dessous à inclure dans le fichier `Statistique.py`

```
def CoefficientCorrelation(x,y):
    if len(x)!=len(y) or not len(x):
        print "Error"; return None
    Ks = range(len(x))
    xbar, ybar =     .....
    sigma_xy =     .....
    sigma_x =     .....
    sigma_y =     .....
    return sigma_xy/np.sqrt(sigma_x*sigma_y)
```

☞ Traçons un exemple de données construites selon les lignes dans un script `Correlations.py` :

```
from Statistique import CoefficientCorrelation, plt
from random import uniform

N = 10
x, y = [ uniform(0,1) for i in range(N) ],[ uniform(0,1) for i in range(N) ]
r = CoefficientCorrelation(x,y)
plt.plot(x,y,'o')
plt.title("$r= {:.5f}$".format(r))
plt.tight_layout()
plt.show()
```

Les données `x` et `y` sont-elles indépendantes ? Calculer leur coefficient de corrélations pour quelques nombres d'échantillons N différents.

☞ On souhaite effectuer une étude plus systématique en traçant la distribution de r pour différents N

```
N = 10
data = []
for stat in range(20000):
        .....
n, bin, patches = plt.hist(data, bins=21, range=(-1.0,1.0), normed=1)
plt.show()
```

Compléter la ou les lignes manquantes dans la boucle. Discutez ce que vous observez pour $N = 2, 3, 4$ puis augmentez, raisonnablement N jusqu'à 100.

☞ On se place dans le cas suivant :

```
theta = [ uniform(0,2*np.pi) for i in range(N) ]
x, y = np.cos(theta), np.sin(theta)
```

Les variables x et y sont-elles indépendantes ? Calculer quelques coefficients de corrélations. Effectuer de nouveau l'étude systématique de la distribution de r et discuter les résultats.

☞ Enfin, pour des exemples tirés de la vie réelle de corrélations douteuses, on pourra consulter le site `http://www.tylervigen.com/spurious-correlations`.

Une barre d'erreur statistique

On illustre ici certains points discutés en cours sur une seule barre d'erreur. On écrira un script `BarreErreur.py`.

☞ Pour commencer, créer une fonction qui va simuler une série de N mesures distribuées selon une loi uniforme

```
from Statistique import moyenne, variance, plt
from random import uniform

mu, dispersion = 3.245, 0.234
vmin, vmax = mu-dispersion/2., mu+dispersion/2.
def Serie(N):
    return      .....
print Serie(3)
```

Compléter la ligne ci-dessus. Quelle est la variance exacte, à entrer dans une variable `exactvar`, de cette loi en fonction de `dispersion` ?

☞ On veut tracer la distribution de la moyenne obtenue pour différents nombre de mesures. Compléter le code ci-dessous et commenter le graphe :

```
Nmesures = [1,2,3,5,10,24,50,120,300]
Stat = 20000
res = []
for N in Nmesures:
    res.append (..... Serie(N  ).....)

i = 0
for r in res [::2]:
    i += 1
    n, bin, patches = plt.hist(r, bins=51, range=(vmin,vmax), \
                               normed=1, color=str(i/float(len(res))) )
plt.xlim  ([.....,....])
plt.xlabel("estimation de la moyenne")
plt.ylabel(u"densité de probabilité")
plt.tight_layout()
```

Rajouter une ligne pour faire afficher une légende sur le graphe.

☞ Observons maintenant comment la barre d'erreur diminue avec le nombre de mesures N. Compléter le script ci-dessous pour obtenir le graphe montrant les moyennes des résultats pour différents N avec une barre d'erreur correspondant à un écart-type. Quel est le comportement attendu ? Modifier les deux lignes `plt.plot(....)` pour tracer ce comportement exact en fonction de N

```
y = [   .....   for r in res ]
sigma_y = [   .....   for r in res ]

ns = np.logspace(0,3,100)
plt.plot ([0.5,500],[ mu,mu],'k--')
plt.plot   (.....)
plt.plot   (.....)
plt.errorbar(Nmesures, y, yerr=sigma_y, fmt='o', color='b', capthick=2, capsize=6)
plt.xscale('log')
plt.xlim ([0.5,500]);  plt.ylim([vmin,vmax])
plt.xlabel("nombre de mesures")
plt.ylabel(u"valeur estimée avec barre d\'erreur")
```

Régression linéaire

On souhaite maintenant rajouter dans la librairie `Statistique` une fonction `RegressionLineaire` qui va appliquer les formules vues en cours et que l'on rappelle maintenant : on cherche à valider un modèle linéaire $y(x) = ax + b$ et estimer ses coefficients a et b ainsi que leurs incertitudes σ_a et σ_b : à partir d'un jeu de données $\{x_k, y_k, \sigma_k\}$ dont seules les y_k sont affectées de barres d'erreur ou non (prendre $\sigma_k = 1$ dans ce cas). Nous avons montré que les estimateurs étaient

$$a = \frac{SS_{xy} - S_x S_y}{\Delta} \quad \sigma_a = \sqrt{\frac{S}{\Delta}} \quad ; \quad b = \frac{S_y S_{xx} - S_x S_{xy}}{\Delta} \quad \sigma_b = \sqrt{\frac{S_{xx}}{\Delta}} \quad ; \quad \Delta = SS_{xx} - S_x^2$$

où

$$S = \sum_k \frac{1}{\sigma_k^2} , \quad S_x = \sum_k \frac{x_k}{\sigma_k^2} , \quad S_y = \sum_k \frac{y_k}{\sigma_k^2} , \quad S_{xx} = \sum_k \frac{x_k^2}{\sigma_k^2} , \quad S_{xy} = \sum_k \frac{x_k y_k}{\sigma_k^2} .$$

☞ Compléter la fonction ci-dessous à inclure dans le fichier `Statistique.py` et qui va exécuter la régression linéaire pour renvoyer les résultats.

```
def RegressionLineaire(x,y,sigma=[]):
    if not len(sigma): sigma = [1.0]*len(x)
    if len(x)!=len(y) or len(x)!=len(sigma) or len(x) < 3:
        print "Error"; return None
    Ks =  .....
    S   = sum([ 1.0/sigma[k]**2       for k in Ks])
    Sx  =          .....................................
    Sxx =          .....................................
    Sxy =          .....................................
    Sy  =          .....................................
    Delta =          .....................................
    a, sigma_a =    ............,        ..............
    b, sigma_b =    ............,        ..............
    return (a,sigma_a,b,sigma_b)
```

☞ Il peut de même être utile de calculer le χ^2 réduit

$$\chi^2 = \frac{1}{N-2} \sum_{k=1}^{N} \left(\frac{y_k - y(x_k)}{\sigma_k} \right)^2$$

où $y(x)$ est la fonction testée pour modéliser les données.

```
def Chi2(x,y,sigma=[],model=lambda x:x):
    if not len(sigma): sigma = [1.0]*len(x)
    if len(x)!=len(y) or len(x)!=len(sigma) or len(x) < 3:
        print "Error"; return None
    return        ...............................
```

Compléter cette fonction et l'inclure dans la bibliothèque `Statistique`.

☞ On se sert maintenant de la librairie pour simuler des régressions linéaires. Tout d'abord dans le cas d'une régression sans barre d'erreur et pour un bruit distribué selon une gaussienne de largeur σ. Compléter le script `FitLineaire.py` qui a la structure ci-dessous :

```
from Statistique import *
from random import gauss

a0, b0 = 1.0, 2.0
exacte = lambda x: a0*x + b0
sigma = 0.4
Npoints = 10
xk = np.linspace(0.0,5.0, Npoints)
yk0 = [ exacte(x) for x in xk]
yk = [ gauss (.....)  for k in range(len(xk)) ]

a,sigma_a,b,sigma_b = RegressionLineaire(xk,yk)
linmodel = lambda x: a*x + b

chi2 = Chi2(xk,yk,model = ....)
r = CoefficientCorrelation(xk,yk)
txt = "$a = {:.4f}\pm {:.4f}$ " \
      "$b = {:.4f}\pm {:.4f}$ " \
      "$\chi^2 = {:.4f}$ " \
      "$r = {:.4f}$ ".format(a,sigma_a,b,sigma_b,chi2,r)

plt.plot(xk,yk0,'k--',linewidth=3)
plt.errorbar(xk, yk, yerr=None, fmt='o', color='b', capthick=2, capsize=6)
plt.plot(xk,[linmodel(x) for x in xk],'r-',linewidth=2)
plt.xlim([-0.5,5.5])
plt.title(txt)
plt.xlabel("x"); plt.ylabel("y")
```

Discuter qualitativement l'influence de `sigma` et de `Npoints` sur r et χ^2, σ_a et σ_b.
Lorsque $N \to \infty$, vers quelle valeur doit tendre χ^2 ?
Bonus : montrer que dans notre situation $\sigma_b \simeq \sqrt{12}/(5\sqrt{N})$ *et* $\sigma_b \simeq 2/\sqrt{N}$ *pour* N *grand.*

☞ **Limite de la régression linéaire, le quadruplet d'Anscombe** : On trouvera sur `http://lptms.u-psud.fr/wiki-cours/index.php/Quadruplet_d'Anscombe` les données qui représentent le quartet ou quadruplet d'Anscombe. Dans un script `Anscombe.py`, faire afficher les données puis effectuer la régression linéaire de chacun des jeux de données en donnant a, b, χ^2 et r. Que constatez-vous ? Quelle(s) réserve(s) en tirer concernant la régression linéaire ?

☞ Introduire maintenant une barre d'erreur expérimentale σ^{exp}, indépendante du bruit, et qui traduit une incertitude de mesure par exemple. Discuter le comportement de r et de χ^2 pour des régressions linéaires avec cette barre d'erreur en prenant différentes situations comme $\sigma^{\text{exp}} > \sigma$ et $\sigma^{\text{exp}} < \sigma$. On note ci-dessous quelques modifications sur le script

```
sigmak = [ sigma_exp ]*len(xk)
a,sigma_a,b,sigma_b = RegressionLineaire(xk,yk,sigmak)
 ....
chi2_stat = Chi2(xk,yk,sigmak,model = ....)
 ....
plt.errorbar(xk, yk, yerr=sigmak, fmt='o', color='b', capthick=2, capsize=6)
 ....
```