M1 Physique Appliquée — Université Paris-Sud XI
Année 2016-2017

Programmation et données numériques

TD : Régression linéaire, interpolation et fits non-linéaires

Régression linéaire

Implémentation des formules

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)
```

Réduction à un problème linéaire

On considère des données que l'on souhaite modéliser par une décroissance exponentielle de la forme $z_k = \alpha e^{-t_k/\tau}$.

☞ Montrer que le modèle ci-dessus peut se ramener à un modèle linéaire de la forme

$$y_k = a\, x_k + b\ . \tag{1}$$

Expliciter la relation entre y_k et z_k, entre x_k et t_k ainsi qu'entre a et τ, et b et α. En utilisant la formule de propagation des erreurs, évaluer les σ_{y_k} en fonction des σ_{z_k} ainsi que les incertitudes sur les valeurs de τ^{fit} et α^{fit} tirées des résultats de la régression linéaire.

☞ Comprendre et compléter le script ci-dessous, puis observer ce qu'il se passe lorsque σ_t augmente. On pourra prendre en compte ou non les barres d'erreur dans la régression linéaire.

```
from Statistique import RegressionLineaire, np, plt
from random import gauss

alpha, tau = 2.0, 1.0
modele = lambda t: alpha*np.exp(-t/tau)
sigma_t = 0.001

Npoints = 20
tk = np.linspace (0.0,5.0, Npoints)
zk0 = modele(tk) # [ modele(t) for t in tk]
zk = [ zk0[k] + gauss(0.0,sigma_t) for k in range(Npoints) ]
sigma_zk = [ sigma_t ]*Npoints

xk = ....
yk = ....
sigma_yk = ....

a,sigma_a,b,sigma_b = RegressionLineaire (......)

txt = "$a = {:.4f}\pm {:.4f}$ " \
      "$b = {:.4f}\pm {:.4f}$ ".format(a,sigma_a,b,sigma_b)
linmodele = lambda x: a*x+b
plt.errorbar(xk, yk, yerr=sigma_yk, fmt='o', color='b', capthick=2, capsize=6)
plt.plot(xk,[linmodele(x) for x in xk],'r-',linewidth=2)
plt.xlim([min(xk)-0.5,max(xk)+0.5])
plt.title(txt)
plt.xlabel("x"); plt.ylabel("y")
plt.show()

alpha_fit, tau_fit = ...., ....
sigma_alpha, sigma_tau = ...., ....

txt = "$\\alpha = {:.4f}\pm {:.4f}$ " \
      "$\\tau = {:.4f}\pm {:.4f}$ " \
      .format(alpha_fit,sigma_alpha,tau_fit,sigma_tau)
expmodele = lambda t: alpha_fit*np.exp(-t/tau_fit)

plt.errorbar(tk, zk, yerr=sigma_zk, fmt='o', color='b', capthick=2, capsize=6)
tk = np.linspace(min(tk),max(tk),10*Npoints)
plt.plot(tk,[expmodele(t) for t in tk],'r-',linewidth=2)
plt.xlim([min(tk)-0.5,max(tk)+0.5])
plt.title(txt)
plt.xlabel("t"); plt.ylabel("z")
plt.show()
```

Fit polynomial

On a vu en cours que le premier de la régression linéaire se généralisait à une combinaison linéaire de fonctions quelconques. Cela permet en particulier d'effectuer des fits par des polynômes, soit des modèles de la forme :

$$y(x) = a_0 + a_1\, x + a_2\, x^2 + \ldots + a_m\, x^m \tag{2}$$

- ☞ Écrire la matrice de design ou matrice de modèle pour cet exemple et remarquer sa structure simple.
- ☞ En python, le fit polynomial par les formules de régression linéaire généralisée a été implémenté dans `scipy` qui fournit une bibliothèque de manipulation des polynômes. On pourra consulter la documentation de `numpy.polynomial`. Un polynôme est déterminé par la donnée des ses coefficients a_i. Compléter et interpréter le script suivant :

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

xmin, xmax, Npoints = 0.0, 2.0*np.pi, 30
x = np.linspace(xmin,xmax,10*Npoints)
f = lambda x: np.sin(x)

sigma = 0.2
xb = np.linspace(xmin,xmax,Npoints)
yb = f(xb) + sigma*np.random.normal(size=len(xb))

from numpy.polynomial import polynomial as P
m = 2
print "Fit polynomial avec m=",m
coeff, stats = P.polyfit (..., ..., m,full=True)
fitpoly = P.Polynomial(...)

plt.scatter(xb,yb)
plt.plot(x,f(x))
plt.plot(x, fitpoly(x), 'r-')
plt.show()
```

Jouer ensuite sur les valeurs de m et de σ. Si vous augmentez m jusqu'au nombre de points, que se passe-t-il ?

Application à la mesure de la constante de Boltzmann

Vous avez mesuré en TP sur les mesures et la détection le bruit de Johnson Nyquist. Il est attendu qu'il suive la densité spectrale de bruit suive le modèle linéaire suivant $S = 2k_BTR$. Compléter le script ci-dessous pour obtenir votre estimation de la constant de Boltzmann k_B. Si vous ne retrouvez plus vos données, vous pouvez utiliser celles ci-dessous :

```
from Statistique import RegressionLineaire, plt

xk = [3300,47000,2200,5000,1000,4700,10000,100]
yk = [2.6e-17,4.0e-16,1.8e-17,4.3e-17,8.3e-18,3.8e-17,8.2e-17,8.0e-19]
sigma_yk = [3.5e-17,3.5e-17,2.5e-18,6.0e-18,7.0e-18,4.5e-19,9.5e-18,3.1e-19]

a,sigma_a,b,sigma_b = RegressionLineaire(xk,yk,sigma_yk)

txt = "$a = {:.4g}\pm {:.4g}$ and " \
      "$b = {:.4g}\pm {:.4g}$ ".format(a,sigma_a,b,sigma_b)
```

```
meilleur_kB = 1.38064852e-23
Temp = 293
print "constante de Boltzmann obtenue: k_B = {:.5g} +/- {:.2g}" \
    .format (....,....)
print u"constante de Boltzmann tabulée: k_B = {:.5g}".format(meilleur_kB)

linmodele = lambda x: a*x+b
plt.errorbar(xk, yk, yerr=sigma_yk, fmt='o', color='b', capthick=2, capsize=6)
plt.plot(xk,[linmodele(x) for x in xk],'r--',linewidth=1)
plt.title(txt)
plt.show()
```

Optimisation : zéro d'une fonction

La recherche de zéro d'une fonction permet de trouver les solutions d'une équation de la forme $f(x) = 0$. On donne pour cela en général une valeur initiale de x appelée `guess` autour de la laquelle on recherche une solution.

On prend l'exemple du modèle d'Ising en champ moyen qui décrit une transition de phase vers un état magnétique ordonné à basse température se caractérisant par une aimantation non-nulle. L'aimantation m satisfait à l'équation auto-cohérente suivante

$$m = \mathrm{th}(m/T) \tag{3}$$

où T est la température. Analytiquement, il n'exister pas de solution sous forme d'une fonction simple tabulée. Graphiquement, étudier les solutions revient à chercher les points d'intersection entre les courbes $y = m$ et $y = \mathrm{th}(m/T)$ pour T donnée.

- ☞ Justifier qu'il existe deux régimes en fonction de T : un où il y a une seule solution pour m et l'autre avec trois solutions. Quelle est la température qui sépare ces deux régimes ?
- ☞ En ayant regardé la documentation, utiliser `fsolve` de `scipy.optimize` pour résoudre numériquement le problème en interprétant et complétant le script ci-dessous :

```
import numpy as np
from scipy.optimize import fsolve
import pylab as plt

guess = 0.0
IsingEq = lambda x, T: x - np.tanh(x/T)
Ising   = lambda T: fsolve(...,   ...., args=(T,))

Ts = np.linspace(0.0,1.4,101)

guess =  ....
plt.plot(Ts,[ Ising(T) for T in Ts])
guess =  ...
plt.plot(Ts,[ Ising(T) for T in Ts])
guess =  ...
plt.plot(Ts,[ Ising(T) for T in Ts])
plt.ylim(-1.1,1.1)
plt.ylabel(u"paramètre d'ordre")
plt.xlabel(u"Température")
plt.title(u"Transition de phase")
plt.grid(True)
plt.show()
```

Fits non-linéaires

La bibliothèque `scipy.optimize` fournit la fonction `curve_fit` qui permet d'effectuer un fit non-linéaire général.

Retour sur la décroissance temporelle

On reprend l'exemple précédent d'une décroissance exponentielle et on utilise maintenant la méthode de fit non-linéaire itérative. Lire la documentation de `curve_fit` pour lui passer les bons arguments. Jouer sur `sigma` et sur `guess` pour voir le comportement du fit. Modifier ensuite l'appel de la fonction pour que le fit prenne en compte les barres d'erreur.

```
from scipy.optimize import curve_fit

guess = ....
fitfunc = lambda x, a, b: a*np.exp(-x/b)
p, pcov = curve_fit (..., ..., ..., p0 = guess)
sigma_p = np.sqrt(np.diag(pcov))

txt = "$\\alpha = {:.4g}\pm {:.4g}$ " \
      "$\\tau = {:.4g}\pm {:.4g}$ ".format(p[0],sigma_p[0],p[1],sigma_p[1])

plt.errorbar(tk, zk, yerr=sigma_zk, fmt='o', color='b', capthick=2, capsize=6)
plt.plot(tk,modele(tk),'k')
plt.plot(tk, fitfunc (tk,p [0], p [1]), 'r')
plt. title (txt)
plt.xlabel("t"); plt.ylabel("z")
plt.show()
```

Application au fit de deux raies gaussiennes

Le script ci-dessous étudie le fit d'un signal de deux raies gaussiennes de même largeur, séparées d'une distance d et d'amplitude relative contrôlée par A. Jouer avec la distance, avec A et le bruit σ pour voir la qualité du fit. Rappelons que le fit dépend du choix du guess initial.

```
import numpy as np
from random import gauss

dist, A, sigma_g = 3.0, 0.7, 0.5
modele = lambda x: A*np.exp(-.5*((x-dist/2.)/sigma_g)**2) \
                 + (1.0-A)*np.exp(-.5*((x+dist/2.)/sigma_g)**2)

sigma = 0.02
Npoints = 50
xk = np.linspace(-5.0,5.0,Npoints)
yk = [ modele(x) + gauss(0.0,sigma) for x in xk ]
sigma_k = [ sigma ]*Npoints

from scipy.optimize import curve_fit

guess = [1.0,1.0,1.0,1.0,1.0]
fitfunc = lambda x, a1,s1,a2,s2,d: a1*np.exp(-((x-d/2.)/s1)**2)\
                                 + a2*np.exp(-((x+d/2.)/s2)**2)
p, pcov = curve_fit( fitfunc ,xk,yk,p0 = guess, sigma=sigma_k)
sigma_p = np.sqrt(np.diag(pcov))

txt = "$a_1 = {:.3g}$ $s_1 = {:.3g}$ " \
```

```
    "$a_2 = {:.3g}$ $s_2 = {:.3g}$ " \
    "$d = {:.3g}$".format(p[0],p[1],p[2],p[3],p[4])

import matplotlib.pyplot as plt

plt.errorbar(xk, yk, yerr=sigma_k, fmt='o', color='b', capthick=2, capsize=6)
x = np.linspace(-5.0,5.0,10*Npoints)
plt.plot(x,modele(x),'k')
plt.plot(x, fitfunc(x,p[0], p[1], p[2], p[3], p[4]), 'r')
plt.title(txt)
plt.xlabel("$\\nu$"); plt.ylabel("$I(\\nu)$")
plt.show()
```