Programmation et données numériques

TD14 : Interpolation et fits non-linéaires

Compléments sur la régression linéaire

Formule de propagation des erreurs

On rappelle que si deux variables aléatoires X et Y sont reliées par une loi du type $y = f(x)$, on peut estimer le lien entre les écart-types de la manière suivante

$$\sigma_y^{prop} \simeq \left|\frac{df}{dx}\right| \sigma_x \tag{1}$$

☞ On considère une relation linéaire de la forme $y = ax + b$. Que peut-on dire de la formule de propagation des erreurs dans ce cas ? Comprendre puis compléter le script ci-dessous qui génère des données bruitées $\{x_k, y_k\}$ et qui réutilise la bibliothèque `Statistique.py` du TD12/13.

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

sigma = 0.1

modele = lambda x: x + 1.0
sigma_prop = lambda x: sigma

def data_y(x,Nstat=10000):
    y = []
    for k in range(Nstat):
        xk = x + gauss(0.0,sigma)
        y.append (.....)
    return y

plt.hist(data_y(0.1),bins=51)
plt.hist(data_y(0.5),bins=51)
plt.show()

Npoints = 51
xk = np.linspace(0.1, 3.0, Npoints)
ratio = []
for x in xk:
    res = np.sqrt(variance(data_y(x)))
    ratio.append(res/sigma_prop(x))

txt = "Test de la formule de propagation des erreurs"
plt.plot(xk,ratio,'ko',linewidth=3)
plt.plot(xk,[1.0]*len(xk),'r-',linewidth=2)
plt.xlim([min(xk)-0.5,max(xk)+0.5]);
plt.title(txt)
plt.xlabel("x"); plt.ylabel("$\sigma_y/\sigma_y^{prop}$")
plt.tight_layout()
plt.show()
```

Observer le comportement du rapport σ_y/σ_y^{prop} pour des σ_x de plus en plus grands.

☞ Modifier le script ci-dessus pour étudier la formule de propagation des erreurs pour un modèle de type $y = x^q$ avec q donné, typiquement un entier $q = 2, 3, \ldots$. Observer le comportement du rapport σ_y/σ_y^{prop} pour des `sigma` de plus en plus grands ainsi que le comportement des deux distributions tracées au début. Quel argument qualitatif sur la moyenne de y et son écart-type donne lieu à un bon accord avec la formule attendue. En déduire une interprétation pour les valeurs de x pour lesquelles le rapport devient moins bon.

☞ Reproduire la discussion pour le modèle $y = e^x$. Quand la formule de propagation des erreurs est-elle raisonnable dans ce cas ?

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

$$y_k = a\, x_k + b\ . \tag{2}$$

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 :

$$y(x) = a_0 + a_1\, x + a_2\, x^2 + \ldots + a_m\, x^m \tag{3}$$

☞ 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 $S = 4k_BTR$. 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 et fits non-linéaires

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

$$m = \mathrm{th}(m/T) \tag{4}$$

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()
```