IX.2 Logiciels, fonctions Python

IX.2.1 Obtenir a et b et leurs incertitudes

Regressi, Excell, Libre Office ou Open Office, Google Sheet... Plus technique, Reg Lin MC.
Il existe différents outils pour effectuer des régressions linéaires en traitant de façon rigoureuse les incertitudes. Mentionnons :

- En Python, la fonction `np.polyfit`. Gère les incertitudes sur y.

  Sans prendre en compte les incertitudes :

```
import numpy as np
a,b = np.polyfit(x,y,1)
```

  En prenant en compte les incertitudes sur y :

```
import numpy as np
a,b = np.polyfit(x,y,1,w=1/uy)
```

  En prenant en compte les incertitudes sur y et en récupérant $u(a)$ et $u(b)$:[9]

```
import numpy as np

popt, pcov = np.polyfit(x,y,1,w=1/uy,cov='unscaled') # unscaled pour retourner u(a)
                                                     # et u(b) sans correction
upopt = np.sqrt(np.abs(np.diagonal(pcov)))    # les incertitudes-types sur les parametres

# Ce qui donne :
a,b = popt
ua, ub = upopt      # incertitude type sur a, et sur b
cov_ab = pcov[0,1] # covariance entre a et b
```

- En Python, la fonction `spo.leastsq`. Gère les incertitudes sur x et y. Utilise l'algorithme de (LM) Levenberg-Marquardt pour minimiser χ^2.

```
import numpy as np
import scipy.optimize as spo

def f(x,p):
    a,b = p
    return a*x+b

# Fonction d'ecart ponderee par les erreurs (ne prend pas en compte cov(x,y))
def residual(p, y, x):
    return (y - f(x,p)) / np.sqrt(uy**2 + (a*ux)**2)

# Estimation initiale des parametres, A changer si resultat aberrant.
p0 = np.polyfit(x,y,1)

# Minimisation de la quantite residual.
# implemente l'algorithme de Levenberg-Marquardt
result = spo.leastsq(residual, p0, args=(y, x), full_output=True)

# On obtient :
popt = result[0]       # les parametres d'ajustement optimaux
pcov = result[1]       # matrice de variance-covariance estimee des parametres
upopt = np.sqrt(np.abs(np.diagonal(pcov)))
a, b  = popt           # pente, ordonnee a l'origine
ua, ub = upopt      # incertitude type sur a, sur b
cov_ab = pcov[0,1] # covariance entre a et b
```

- En Python, une méthode qui implémente la MLE (maximum likelihood estimation). Gère les incertitudes sur x et y. (cf paragraphe justification)

```
def Somme_A_Minimiser(ab,x,ux,y,uy):
    # ab[0] = pente a ; ab[1] = ordonnée à l'origine b
    n = x.size
    # xx sont les estimations des valeurs vraies des xi, un calcul de minimisation
    # montre qu'ils sont donnés par :
    xx = ((uy*uy)*x + (ux*ux)*ab[0]*(y-ab[1])) / (uy*uy + (ab[0]*ux)**2)
    # La somme à minimiser est alors :
```

9. Cf https://numpy.org/doc/stable/reference/generated/numpy.polyfit.html pour les options de polyfit ; en particulier pour l'argument cov : "If given and not False, return not just the estimate but also its covariance matrix. By default, the covariance are scaled by chi2/dof, where dof = M - (deg + 1), i.e., the weights are presumed to be unreliable except in a relative sense and everything is scaled such that the reduced chi2 is unity. **This scaling is omitted if cov='unscaled', as is relevant for the case that the weights are w = 1/sigma, with sigma known to be a reliable estimate of the uncertainty**.".
De plus, des tests montrent que "unscaled" permet bien d'obtenir les mêmes $u(a)$ et $u(b)$ qu'avec les autres méthodes (LM, ODR...).

```
        som = np.sum( ((xx-x)/ux)**2 + ((ab[0]*xx+ab[1]-y)/uy)**2 )
        return(som)

    ab0 = np.polyfit(x,y,1)  # valeurs initiales
    # On minimise par rapport à a et b, avec x,ux,y,uy fixes (ce sont les données)
    a,b = spo.minimize(Somme_A_Minimiser, ab0, args=(x,ux,y,uy)).x
    # Le .x est pour récupérer le résultat (format de spo.minimize)
```

– En Python, la méthode ODR (orthogonal). Gère les incertitudes sur x et y.

```
    # Utilise ODR de scipy. Minimise l'écart quadratique orthogonalement à la droite.
    # Utilise un Levenberg-Marquardt modifié.

    def f(ab, x):
        return ab[0]*x + ab[1]

    linear = odr.Model(f)
    mydata = odr.RealData(x,y,sx=ux,sy=uy)
    myodr = odr.ODR(mydata, linear, beta0=np.polyfit(x,y,1))
    out = myodr.run()

    # On obtient :
    # les parametres d'ajustement optimaux
    popt = out.beta
    # la matrice de variance-covariance estimee des parametres
    pcov = out.cov_beta
    # les incertitudes-types sur ces parametres
    upopt = np.sqrt(np.abs(np.diagonal(pcov)))

    # Ce qui donne :
    a,b = popt  # pente, ordonnne a l'origine
    ua, ub = upopt # incertitude type sur a, et sur b
    cov_ab = pcov[0,1] # covariance entre a et b
```

– Le programme Reg Lin MC [10], qui est adapté des consignes du Gum. Très complet, permet également des simulations Monte Carlo, ainsi que la visualisation des corrélations entre a et b. Mais un peu difficile d'approche. Utilise l'algorithme de York.

– Le logiciel Régressi [11]. Rigoureux et intuitif. Très utilisé en séance de TP. Penser à cocher "prendre en compte les incertitudes" pour que celles-ci soient considérées et que les calculs de Δa et Δb correspondent à ce qui est expliqué ici. Utilise l'algorithme de York.

#	logiciel ou fonction	retourne $u(a)$ et $u(b)$	fonctionne aussi si $u(y_i) \neq 0$	fonctionne aussi si $u(x_i) \neq 0$
1	Regressi	oui	oui	oui
2	Latis Pro	oui	oui (toutes identiques)	oui (toutes identiques)
3	Reg Lin MC	oui	oui	oui
4	Python : polyfit	oui	oui	non
5	Python : LM	oui	oui	oui
6	Python : MLE	oui	oui	oui
7	Python : ODR	oui	oui	oui

Les méthodes 5, 6 et 7 donnent des résultats équivalents (à quelques subtilités théoriques près, sans incidences pratiques). Ils sont également équivalent à ce que retourne 1 et 3 (York semble strictement équivalent à MLE). Polyfit donne la même chose dans les cas restrictifs où il s'applique (pas d'incertitudes sur x).
Exemple :
grpahe !

IX.2.2 Simulations MC : un moyen d'obtenir $u(a)$ et $u(b)$

Ne permet que de trouver $u(a)$ et $u(b)$, étant donné qu'on a déjà obtenu a et b par une des méthodes précédentes. cf aussi doc eleves pour MC.
-> tracé des a et b obtenus sur un même graphe donne leur corrélation.
Remarques :

– Le script ci-dessus diffère légèrement de ce que recommande la littérature. Il faudrait normalement générer les valeurs aléatoire autour des points du modèle, donc autour de (x_i, ax_i+b), et non pas autour des points de mesure (X_i, Y_i) comme fait ici. La raison est que les points du modèle sont les meilleurs possibles.

  Cependant, la différence est minime. NUM Rec signale que pour toute utilisation pratique, la différence est imperceptible. York semble signaler que dans le cas de la régression linéaire, il n'y a pas de différence.

10. http://jeanmarie.biansan.free.fr/reg_lin_mc.html
11. http://jean-michel.millet.pagesperso-orange.fr/regressi.html

– Les $u(a)$, $u(b)$ obtenus par MC sont les mêmes que ceux obtenus par les méthodes qui propagent l'incertitude (formules du §IX.9) (donc que celle retournées par Regressi, un tableaur, polyfit ou tout autre logiciel ou fonction (méthode de York, ODR ou MLE...)), seulement sous certaines conditions : que les erreurs sont distribuées selon des lois normales (ou binormales si X_i,Y_i corrélés) [12].

  Les tirages MC permettent donc davantage de cas, puisque la distribution de probabilité des erreurs est choisie librement.

– La fonction appelée pour réaliser chaque régression doit normalement gérer correctement les incertitudes sur les Y_i le cas échéant, et celle sur les X_i le cas échéant. Ceci signifie que s'il y a des incertitudes sur les X_i, utiliser polyfit ne convient pas.

  Toutefois, à notre niveau la différence restera faible tant que les $u(X_i)$ sont moins important que les $u(Y_i)$ (au sens où $au(X_i) < u(Y_i)$. C'est une bonne raison pour choisir pour y la grandeur qui a l'incertitude la plus grande.

12. Et si, dans le cas où $\chi^2(a,b)$ est non linéaire/quadratique en a et b ? si le DL ne l'emmène pas trop loin ?