```python
# Importation des bibliothèques
import numpy as np
import scipy.optimize as so
import matplotlib.pyplot as plt

# Données : on rentre theta et I (y) puis cos**2theta (x)

theta = np.pi*np.array([0,10,20,30,40,50,60,70,80,90])/180

I=np.array([0.4,0.37,0.32,0.26,0.19,0.12,0.09,0.005,0.001,0])

cos2theta=(np.cos(theta))**2


#forcer la regression linéaire (on utilise ensuite curfit)
def fct(x,k):
    return k*x
#regression avec curve_fit : la diagonale de la matrice covariance (pcov) nous donne
la variance des différents paramètres
popt , pcov = so.curve_fit(fct,cos2theta, I)
pente=popt[0]
u_pente = np.sqrt(np.diag(pcov)[0])
Imod = pente*cos2theta

print(pente)
print(u_pente)

# Tracé des points et du modèle
plt.figure("Loi de Malus")
plt.plot(cos2theta,I,'bo',label="points expérimentaux")
plt.plot(cos2theta, Imod, 'r-', label="Droite de régression")
plt.xlabel("cos2theta")
plt.ylabel("I")
plt.legend(loc="upper center")
plt.grid()

# graphique 2: les résidus
plt.figure("Résidus")
plt.plot(cos2theta, I-Imod, "g", label="residus")
plt.xlabel("cos2theta")
plt.ylabel("Résidus")
plt.legend(loc="upper center")
print("les résidus sont paraboliques")
```

Code python avec incertitudes et MC

```
# -*- coding: utf-8 -*-

"""Régression linéaire."""

# Importation des bibliothèques
import numpy as np
import numpy.random as rd
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
from scipy import *

# Données

theta = np.pi*np.array([0,10,20,30,40,50,60,70,80,90])/180
utheta=np.pi*1/(2*np.sqrt(3))*ones(len(theta))/180
I=np.array([0.4,0.37,0.32,0.26,0.19,0.12,0.09,0.005,0.001,0.001])
uI=np.array([0.013,0.012,0.011,0.0090,0.007,0.005,0.004,0.002,0.001,0.001])
cos2theta=(np.cos(theta))**2
ucos2=2*np.cos(theta)*np.sin(theta)*utheta


#forcer la regression linéaire (on utilise ensuite curfit)
def fct(x,k):
    return k*x

# Nombre de simulations
N = 10000
# Détermination des pentes et ordonnées à l'origine
asim = []
for i in range(N):
    Xsim = cos2theta + ucos2*rd.uniform(-1, 1, len(cos2theta))
    Ysim = I + uI*rd.uniform(-1, 1, len(I))
    pente = curve_fit(fct,Xsim, Ysim)
    asim.append(pente[0])

ma, ua = np.mean(asim), np.std(asim, ddof=1)

print("a =", ma)
print("u(a) =", ua)

# Tracé des points et du modèle
Imod = ma*cos2theta
plt.figure(1)
plt.subplot(1, 2, 1)
plt.errorbar(cos2theta, I, xerr=ucos2, yerr=uI, fmt='bo', label="Points
expérimentaux")
plt.plot(cos2theta, Imod, 'c--', label="Modèle linéaire")
plt.xlabel(r"$cos2theta")
plt.ylabel(r"$I$")
plt.grid(), plt.legend(loc='best')

    #Trace des résidus.
z= (I-ma*cos2theta)
plt.figure(2)
plt.plot(cos2theta, z, 'bo')
plt.fill_between([min(cos2theta)-1, max(cos2theta)+1], y1=-2, y2=2, color='cyan',
alpha=0.1)
plt.xlabel(r"$cos2theta$"), plt.xlim(min(cos2theta)-1, max(cos2theta)+1)
plt.ylabel(r"$Z$"), plt.ylim(-3, 3)
plt.grid()
plt.show()

    #Trace des résidus normalisés
z_n = (I-ma*cos2theta)/uI
plt.figure(3)
plt.errorbar(cos2theta, z_n, xerr=ucos2, yerr=uI, fmt='o', label="(barres d'erreur)")
plt.fill_between([min(cos2theta)-1, max(cos2theta)+1], y1=-2, y2=2, color='cyan',
```

Code python avec incertitudes et MC

```
alpha=0.1)
plt.xlabel(r"$cos2theta$"), plt.xlim(min(cos2theta)-1, max(cos2theta)+1)
plt.ylabel(r"$Z$"), plt.ylim(min(z_n)-1, max(z_n)+1)
plt.grid()
plt.show()
```

Code python avec incertitudes et chi2

```
# -*- coding: utf-8 -*-

# Importation des bibliothèques

import scipy.optimize as spo
import scipy.stats as sstats

import numpy as np
from scipy import *
import matplotlib.pyplot as plt
# ce code permet de faire une régression linéaire sans utiliser la méthode Monté
Carlo et de calculer le chi^2

# Données : on rentre theta et I (y) puis cos**2theta (x) et leur incertitudes

## Code pour la régression linéaire permettant d'obtenir les incertitudes sur la
pente  et le chi^2
def reg_lin(x,y,ux,uy):
    """Effectue une regression lineaire y = ax+b.
      Affiche a, b, les incertitudes types et elargies, le chi2 reduit, etc.
      Produit un graphique avec barres d'erreurs.
       - x et y sont les donnees
       - ux et uy sont les incertitudes types sur x et y
       Ne permet pas de pendre en compte la covariance entre x et y.

    """

    # Fonction f decrivant la courbe a ajuster aux donnees
    def f(x,a):
        return a*x


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

    # Estimation initiale des parametres.
    # a changer si resultat aberrant.
    p0 = 0

    # Minimisation de la quantite residual.
    # On utilise l'algorithme des moindres carres non-lineaires
    # disponible dans la biliotheque scipy
    result = spo.leastsq(residual, p0, args=(y, x), full_output=True)

    # On obtient :
    # les parametres d'ajustement optimaux
    a = result[0];
    # la matrice de variance-covariance estimee des parametres
    acov = result[1];
    # les incertitudes-types sur ces parametres
    ua = np.sqrt(np.abs(np.diagonal(acov)))


    # Calcul de la valeur du "chi2 reduit" pour les parametres ajustes
    chi2 = np.sum(np.square(residual(a,y,x)))
    chi2r = chi2/(x.size-a.size)


    # Affichage des resultats
    print("---- regression lineaire ----")
    print("a = " + str("%.4f" % a) + ", u(a) = " + str("%.4f" % ua))
    print("chi2 = " + str("%.2f" % chi2))
    print("chi2 reduit = " + str("%.2f" % chi2r))
    print("----")
```

Code python avec incertitudes et chi2

```
    # Trace de x en fonction de y.
    plt.figure(1)
    plt.errorbar(x, y, xerr=ux, yerr=uy, fmt='o', label="(barres d'erreur)")
    plt.plot(x,a*x,label="y = " + str("%.2f" % a)+" x " )
    plt.xlabel(u'x')
    plt.ylabel(u'y')
    plt.grid()
    plt.legend(loc='best')
    plt.show()

    #Trace des résidus.
    z = (y-a*x)
    plt.figure(2)
    plt.plot(x, z, 'bo')
    plt.fill_between([min(x)-1, max(x)+1], y1=-2, y2=2, color='cyan', alpha=0.1)
    plt.xlabel(r"$x$"), plt.xlim(min(x)-1, max(x)+1)
    plt.ylabel(r"$Z$"), plt.ylim(-3, 3)
    plt.grid()
    plt.show()

    #Trace des résidus normalisés
    z_n = (y-a*x)/uy
    plt.figure(3)
    plt.errorbar(x, z_n, xerr=ux, yerr=uy, fmt='o', label="(barres d'erreur)")
    #plt.plot(x, z_n, 'bo')
    plt.fill_between([min(x)-1, max(x)+1], y1=-2, y2=2, color='cyan', alpha=0.1)
    plt.xlabel(r"$x$"), plt.xlim(min(x)-1, max(x)+1)
    plt.ylabel(r"$Z$"), plt.ylim(min(z_n)-1, max(z_n)+1)
    plt.grid()
    plt.show()

# ---- Application ----
theta = np.pi*np.array([0,10,20,30,40,50,60,70,80,90])/180
utheta=np.pi*1/(2*np.sqrt(3))*ones(len(theta))/180
I=np.array([0.4,0.37,0.32,0.26,0.19,0.12,0.09,0.005,0.001,0.001])
uI=np.array([0.013,0.012,0.011,0.0090,0.007,0.005,0.004,0.002,0.001,0.001])
cos2theta=(np.cos(theta))**2
ucos2=2*np.cos(theta)*np.sin(theta)*utheta

reg_lin(cos2theta,I,ucos2,uI)
```