ExerciseTrendSineFit: Introduction

Least Squares Fit (LSF) method gives **solution for free parameters** $\bar{\beta}$. This method solves $\bar{\beta}$ values that

- **Minimize** χ^2 when errors σ_i are **known**
- **Minimize** R when errors σ_i are **unknown**

Download program from **LineModel.py** from homepage. The model is

$$g(t) = \beta_0 + \beta_1 t$$

LineModel.py first creates simulated data with this model, and then performs LSF to these data. The subroutines are

Data(n) produces simulated data

- $n =$ **n** random time points $t_i =$**T** drawn from an even distribution between 0 and 10
- Two random free parameter values $\bar{\beta} = [\beta_1, \beta_2] =$ **SimBETA** drawn from an even distibution between -1 and 1
- $n =$**n** random errors $\sigma_i =$**EY** drawn from normal distribution $N(m_y = 0, s_y = 0.1)$, where m_y is the mean and s_y is the standard deviation.
- Simulated data $y_i =$**Y**$= g_i + \sigma_i =$**G+EY**

IndexSortOrder(y) gives indeces **k** that are used to re-arrange **y** values into ascending order, which in this case are the time points $t_i =$**T**.

Write1(T,Y,EY) stores simulated data to file **LineModel.dat**

Model(T,BETA) computes model values $g_i =$**G**

Funct(BETA,T,Y,EY) gives variable **(Y-G)/EY** minimized in LSF $\equiv$ **optimize.leastsq**

LSF(T,Y,EY) performs LSF.

Plot1(T,Y,EY,TT,GG) plots results into **LineFit.eps**

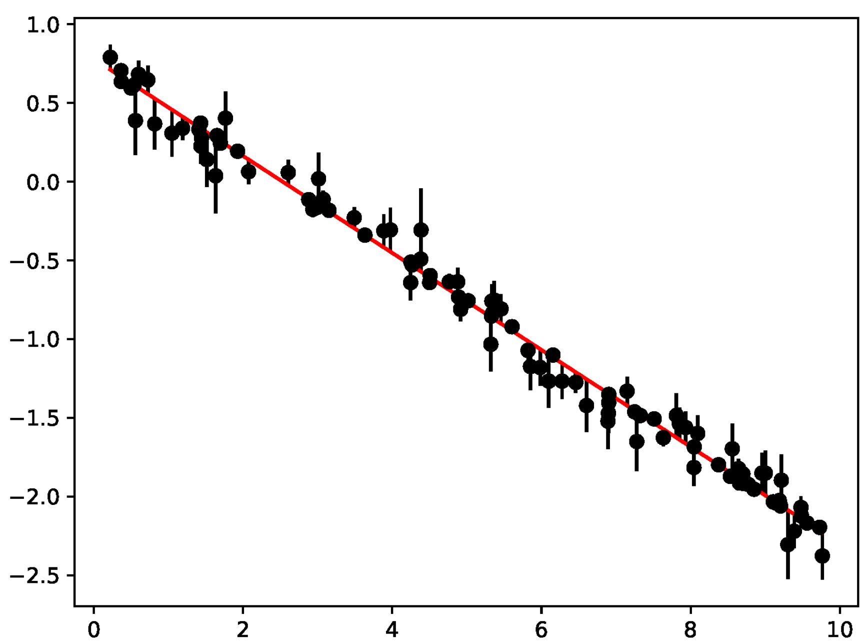
```
# ===========================================================================
# /home/jetsu/opetus/both/programs/LineModel.py
# ===========================================================================
# - Keep computations and plots apart.
# ===========================================================================
import os
import numpy as np
import pylab as pl
from scipy import optimize
os.system('clear')
# ===========================================================================
def Data(n):
    T=np.random.uniform(0,10,n)
    k=IndexSortOrder(T)
    T=T[k]
    BETAsim=-1.+np.random.uniform(0.,2.,2)
    EY=np.random.normal(0,0.1,n)
    G=Model(T,BETAsim)
    Y=G+EY
    Write1(T,Y,EY)
    return T,Y,EY,BETAsim
# ===========================================================================
def IndexSortOrder(y):
    k=sorted(range(len(y)), key=lambda i:y[i])
    return k
# ===========================================================================
def Write1(T,Y,EY):
    file1=open('LineModel.dat',"w")
    for i in range(np.size(T)):
        file1.write("%10.5f %10.5f %10.5f\n" %(T[i],Y[i],EY[i]))
    file1.close()
    return
# ===========================================================================
def Model(T,BETA):
    G=BETA[0]+BETA[1]*T
```

```
    return G
# ============================================================
def Funct(BETA,T,Y,EY):
    G=Model(T,BETA)
    return (Y-G)/EY
# ============================================================
def LSF(T,Y,EY):
    BETAtrial=np.ones(2)
    ny=optimize.leastsq(Funct,BETAtrial,args=(T,Y,EY))
    BETA=ny[0]
    return BETA
# ============================================================
def Plot1(T,Y,EY,TT,GG):
    pl.axes([0.1,0.1,0.8,0.8])
    pl.errorbar(T,Y,EY,fmt='ok',ms=6)
    pl.plot(TT,GG,'r')
    pl.savefig('LineFit.eps')
# ============================================================
#                        Main program
# - Computations ============================================
n=100
T,Y,EY,BETAsim=Data(n)
BETAfinal=LSF(T,Y,EY)
print('Simulated data ...... BETAsim =',BETAsim)
print('Least Squares Fit ... BETAfinal=',BETAfinal)
DT=np.max(T)-np.min(T)
TT=np.min(T)+(np.arange(101)/100.)*DT
GG=Model(TT,BETAfinal)
# - Plots ====================================================
Plot1(T,Y,EY,TT,GG)
```

The **scipy** subroutine **optimize.leastsq** minimizes "the sum of squares of a set of equations". We use **(Y-G)/EY**, which minimizes χ^2.

Notation **ny** is adopted in referring to a variable that is not used later in the program. Note that the first component of this variable, **ny[0]**, contains the final free parameter values **BETA**. Any trial values **BETAtrial** will give the same result, because the model is **linear**.

One example of **LineFit.eps** is shown below. Note that **LineModel.py** always produces a different figure, because it always analyses a different sample of random data.



ExerciseTrendSineFit: Problem

In an earlier **ExerciseTrendSine**, we used the trend plus signal model

$$g(t) = \beta_1 + \beta_2(t - t_1)/(t_n - t_1) + \beta_3 \sin\left[2\pi(t - \beta_4)/\beta_5\right]. \tag{1}$$

to simulate artificial data. The free parameter values were fixed to

$\beta_1 = -5 =$ trend level

$\beta_2 = 10 =$ trend slope

$\beta_3 = 1 =$ signal amplitude

$\beta_4 = 2$ signal epoch

$\beta_5 = 3$ signal period

One such simulated data sample is stored into homepage file **TrendSine.dat**.

The above $g(t)$ model of Eq. 1 is **non-linear**, because the partial derivatives $\partial g/\partial\beta_4$ and $\partial g/\partial\beta_5$ contain free parameters. However, this model can be rewritten to the form

$$g(t) = \beta_1 + \beta_2(t - t_1)/(t_n - t_1) + \beta_3 \cos\left[(2\pi t)/\beta_5\right] + \beta_4 \sin\left[(2\pi t)/\beta_5\right]. \tag{2}$$

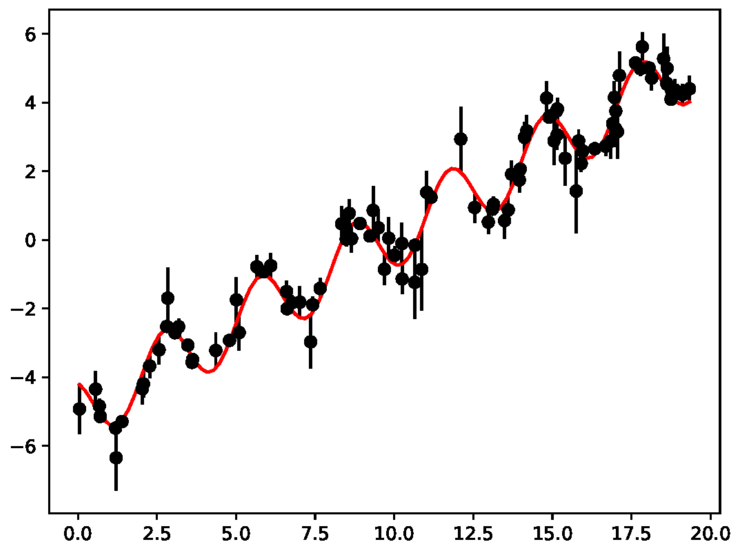
This model is still **non-linear**, because the partial derivative $\partial g/\partial\beta_5$ contains free parameters. Let us assume that we would know that the correct period value is $\beta_5 = P = 3$. In this case, the model

$$g(t) = \beta_1 + \beta_2(t - t_1)/(t_n - t_1) + \beta_3 \cos\left[(2\pi t)/3\right] + \beta_4 \sin\left[(2\pi t)/3\right] \tag{3}$$

is **linear**, and the **Least Squares Fit (LSF)** solutions for the free parameters $\bar{\beta} = [\beta_1, \beta_3, \beta_3, \beta_4]$ are **unambiguous**.

Download file **TrendSine.dat** from course home-page. The data are
column 1 = t_i = **T** = observing times
column 2 = y_i = **Y** = observations
column 3 = σ_i = **EY** = errors

Edit your **python** program **ExerciseTrendSineFit.py**, which performs a **Least Squares Fit** using the above model $g(t)$ of Eq. 3 for these data. Show your results in figure **TrendSineFit.eps**. Your **TrendSineFit.eps** figure should resemble the one shown below. Send your files **Exercise-TrendSineFit.py** and **SineFit.eps** to the assistant.



Tip(s): Here is one subroutine for reading the datafile **file='TrendSine.dat'.**

```
def Readfile(file2):
    T=np.loadtxt(file2,skiprows=0,usecols=(0,))
    Y=np.loadtxt(file2,skiprows=0,usecols=(1,))
    EY=np.loadtxt(file2,skiprows=0,usecols=(2,))
    return T,Y,EY
```