ExampleBootstrap

Download **python** program **ExampleBootstrap.py** from home-page. Give the following commands

python ExampleBootstrap.py
evince OriginalData.eps &
evince ArtificialData.eps &

Rerun

python ExampleBootstrap.py

Note that figure **OriginalData.eps** does not change, but a different figure **ArtificialData.eps** appears five times in about ten seconds.

The model in **ExampleBootstrap.py** is

$$g(t) = \beta_1 + \beta_2 \cos(2\pi f t) + \beta_3 \sin(2\pi f t) + \beta_4 \cos(4\pi f t) + \beta_5 \sin(4\pi f t),$$

where the frequency $f = 1/P$ and the period is $P = 2.4$. The five free parameter $\bar{\beta}$ values are drawn from an even distribution between -1 and 1.

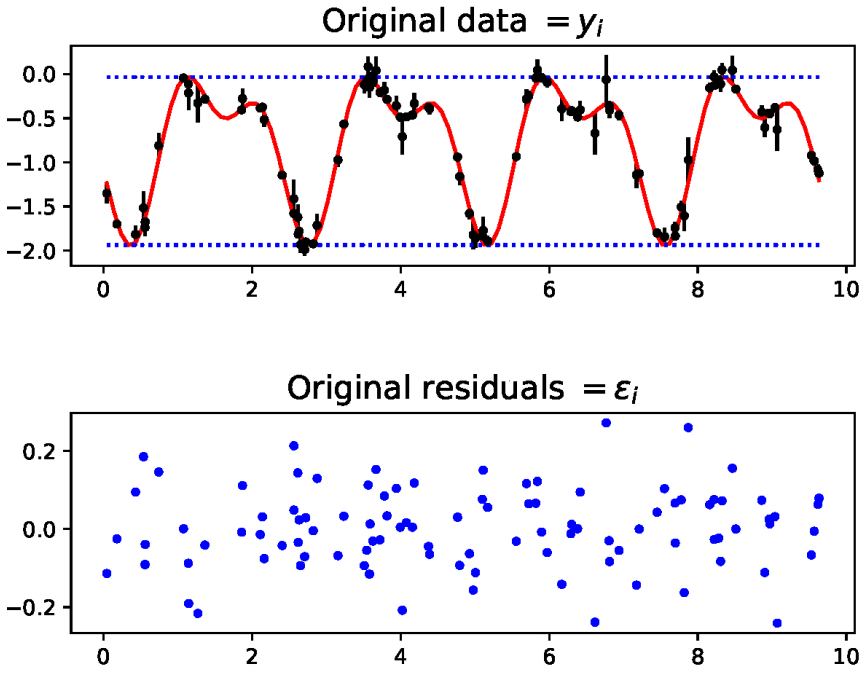
Problem: **Analytical solution** for the peak to peak amplitude A of this $g(t)$ curve is very difficult. **Analytical solution** of the error σ_A is even more difficult. **Bootstrap** gives a simple **numerical solution** for $A \pm \sigma_A$.

Analytical solution of $A \pm \sigma_A$

We do not derive the exact analytical solution for the peak to peak amplitude A and its error σ_A, but we do list some necessary stages below

1. Derive first time derivative $\dot{g}(t)$ of $g(t)$.
2. Solve the two, or the four cases, where $\dot{g}(t_i) = 0$.
3. Solve second derivative $\ddot{g}(t)$. Use it to separate minima and maxima: $t_{\mathrm{min},1}$, $t_{\mathrm{min},2}$ (if present), $t_{\mathrm{max},1}$ and $t_{\mathrm{max},2}$ (if present) where $\dot{g}(t) = 0$.
4. Susbtitute the global minimum epoch value and the global maximum epoch value into $g(t)$. This gives the values of global minimum g_{min} and maximum g_{max} of $g(t)$. The equation for desired peak to peak amplitude is $A = g_{\mathrm{max}} - g_{\mathrm{min}}$. It contains numerous trigonometric functions.
5. Solving the error σ_A of the peak to peak amplitude A requires the solution of all partial derivatives $\partial A / \partial \beta_i$. This is just the first step of the tedious analytical solution. It is unnecessary to describe this solution further. However, we do list some other complications below.
6. Bootstrap sometimes reveals that some minima and maxima do not exits in all samples of artificial data $y_i^\star$. This means that these features may not exist within the error limits of the free parameters β_i. It would be very challenging to code these exceptions into the analytical solution.
7. It is also possible that the primary minimum in original y_i data becomes a secondary minimum in artificial data sample $y_i^\star$, and vice versa. The same can happen to the primary and secondary maxima. Again, an analytical solution of these cases would be challenging.
8. Finally, we have only discussed the peak to peak amplitude $A \pm \sigma_A$ above. The analytical solutions for minima and maxima $t_{\mathrm{min},1}$, $t_{\mathrm{min},2}$, $t_{\mathrm{max},1}$ and $t_{\mathrm{max},2}$ would be equally difficult. Not to mention the estimates for their errors.

Figure 1: **OriginalData.eps**



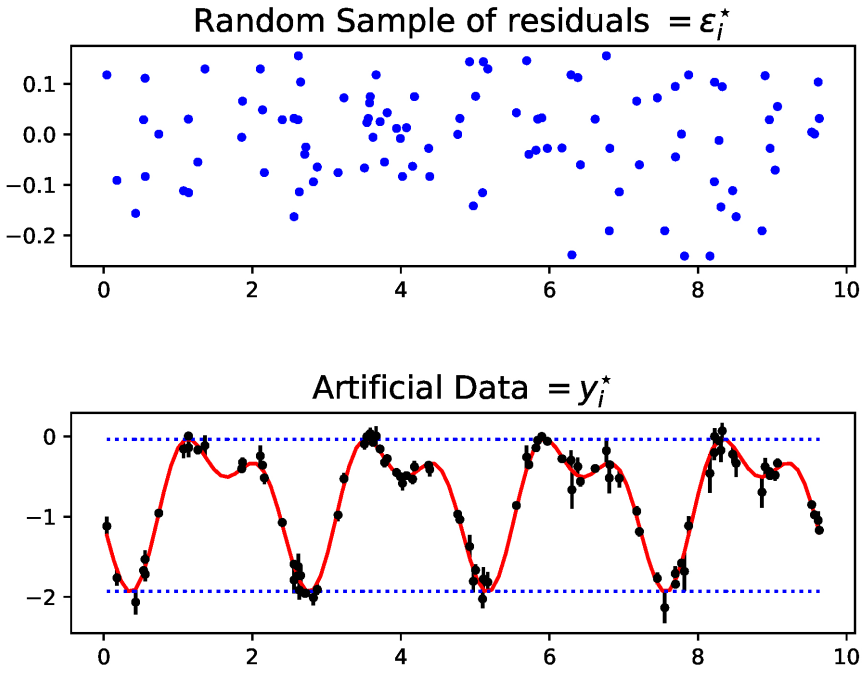
Numerical solution of $A \pm \sigma_A$

1. Model $g(t) =$ **GG** for **original data** $y_i =$ **Y** gives the value for peak to peak amplitude $A = g_{\max}(t) - g_{\min}(t) =$ **np.max(GG)-np.min(GG)**.
2. Model $g(t) =$ **GG** for each j :th **artificial data** sample $y_i^\star = g_i + \epsilon_i^\star =$ **StarY** gives one estimate for peak to peak amplitude $A_j = g_{\max}(t) - g_{\min}(t) =$ **np.max(GG)-np.min(GG)**. The error σ_A is the standard deviation of all these A_j estimates.

Subroutines

Data(n) creates $n =$ **n** observations $t_i =$ **T**, $y_i =$ **Y** and $\sigma_i =$ **EY**.
- All five free parameter values $\bar{\beta} =$ **BETAsim** are drawn from a uniform random distribution
$U(-1, 1) =$ **BETAsim=-1.+np.random.uniform(0.,2.,5)**

Figure 2: **ArtificialData.eps**



- Residuals ϵ_i = **EPSILON** are drawn from a normal distribution $N(0, 0.1)$ = **RESIDUALS=np.random.normal(0,0.1,n)**

IndexSortOrder, **Model**, **Funct** and **LSF** subroutines have already been explained, e.g. in **LineFit.py** or **ExerciseSineFit**.

PlotOriginalData(T,Y,EY,G,TT,GG) plots
- **Original data** and its model $g(t)$ = GG in **OriginalData.eps** (**Figure 1: upper panel**)
- **Original data** residuals $\epsilon_i = y_i - g_i$ = **EPSILON=G-Y** in **OriginalData.eps** (**Figure 1: lower panel**)
- Note command **pl.clf()** which closes **OriginalData.eps**. This allows opening of new eps-files, like **ArtificialData.eps**.

- **Numerical solution** of peak to peak amplitude for **original data**
A = **OrigA=np.max(GG)-np.min(GG)** = distance between the two horizontal dotted lines.

BootStrap(T,Y,EY,OrigG,Rounds,Waitingtime) performs the bootstrap procedure.
- Residuals for **original data** are
$\epsilon_i = y_i - g_i$ = **OrigEpsilon=Y-OrigG**
- Number of created artificial bootstrap data samples is **Rounds**.
- Random indeces between 0 and $n-1$ are
l=np.floor(np.size(T)*np.random.uniform(0,1,np.size(T)))
l=l.astype(int)
- Random sample of errors is $\sigma_i^\star$ = **StarEY=OrigEY[l]**
- Random sample of residuals is
$\epsilon_i^\star$ = **StarEPSILON=OrigEPSILON[l]**
- Artificial data sample is
$y_i^\star = g_i + \epsilon_i^\star$ = **StarY=OrigG+StarEPSILON**
- Free parameters of the model for **artificial data** are
StarBETA=LSF(T,StarY,StarEY).
- Uniformly distributed dense grid of time points **TT** are between t_1 = **np.min(T)** and t_n = **np.max(T)**.
- Model for **artificial data**
$g(t)$ = **GG=Model(TT,StarBETA)**
- **Numerical estimates** of peak to peak amplitude for **artificial data**
A=PlotArtificialData(T,StarY,StarEY,StarEPSILON,TT,GG)
are collected to variable **BootA**.
- Bootstrap stops for **Waitingtime** seconds. This allows the viewer to see how **ArtificialData.eps** changes.

PlotArtificialData(T,StarY,StarEY,StarEPSILON,TT,GG) plots

- **Random sample of** residuals $\epsilon_i^\star$ = **StarEPSILON** in **ArtificialData.eps** (**Figure 2: upper panel**)
- **Artificial data** and its model $g(t)$ = GG **ArtificialData.eps** (**Figure 2: lower panel**)
- **Numerical solution** of peak to peak amplitude for **artificial data** A = **OrigA=np.max(GG)-np.min(GG)** = distance between the two horizontal dotted lines.

```
# ==========================================================
# /home/jetsu/opetus/both/programs/ExampleBootstrap.py
# ==========================================================
# - Keep computations and plots apart.
# ==========================================================
import os
import numpy as np
import pylab as pl
import time
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.,5)
    EPSILON=np.random.normal(0,0.1,n)
    G=Model(T,BETAsim)
    Y=G+EPSILON
    EY=0.*T+0.1
    return T,Y,EY,BETAsim
# ==========================================================
def IndexSortOrder(y):
    k=sorted(range(len(y)), key=lambda i:y[i])
    return k
# ==========================================================
def Model(T,BETA):
    P=2.4
    F=1./P
    ny=2.*np.pi*F*T
    G=BETA[0]+BETA[1]*np.cos(1.*ny)+BETA[2]*np.sin(1.*ny)
    G=      G+BETA[3]*np.cos(2.*ny)+BETA[4]*np.sin(2.*ny)
    return G
```

```
# ==========================================
def Funct(BETA,T,Y,EY):
    G=Model(T,BETA)
    return (Y-G)/EY
# ==========================================
def LSF(T,Y,EY):
    BETAtrial=np.ones(5)
    ny=optimize.leastsq(Funct,BETAtrial,args=(T,Y,EY))
    BETA=ny[0]
    return BETA
# ==========================================
def PlotOriginalData(T,Y,EY,G,TT,GG):
    pl.axes([0.1,0.6,0.8,0.3])
    pl.errorbar(T,Y,EY,fmt='ok',ms=3)
    pl.plot(TT,GG,'r')
    pl.plot(TT,0.*TT+np.max(GG),':b')
    pl.plot(TT,0.*TT+np.min(GG),':b')
    A=np.max(GG)-np.min(GG)
    print('ORIGINAL DATA, Peak to peak amplitude = ',A)
    pl.title('Original data $= y_i$',fontsize=15)
    pl.axes([0.1,0.1,0.8,0.3])
    EPSILON=Y-G
    pl.plot(T,EPSILON,'ob',ms=3)
    pl.title('Original residuals $=\epsilon_i$',fontsize=15)
    pl.savefig('OriginalData.eps')
    pl.clf()
    return A
# ==========================================
def BootStrap(T,Y,EY,OrigG,Rounds,Waitingtime):
    OrigEPSILON=Y-OrigG
    OrigEY=EY
    for i in range(0,Rounds):
        print('Bootsrap round ',i+1,' out of ',Rounds)
        l=np.floor(np.size(T)*np.random.uniform(0,1,np.size(T)))
        l=l.astype(int)
        StarEY=OrigEY[l]
        StarEPSILON=OrigEPSILON[l]
        StarY=OrigG+StarEPSILON
        StarBETA=LSF(T,StarY,StarEY)
        DT=np.max(T)-np.min(T)
        TT=np.min(T)+(np.arange(101)/100.)*DT
        GG=Model(TT,StarBETA)
        A=PlotArtificialData(T,StarY,StarEY,StarEPSILON,TT,GG)
        print('Next random sample waiting time is ',Waitingtime,'seconds ')
        print('.....')
        if (i == 0):
```

```
            BootA=A
        else:
            BootA=np.append(BootA,A)
        time.sleep(Waitingtime)
    return BootA
# ==========================================
def PlotArtificialData(T,StarY,StarEY,StarEPSILON,TT,GG):
    pl.axes([0.1,0.6,0.8,0.3])
    pl.plot(T,StarEPSILON,'ob',ms=3)
    pl.title('Random Sample of residuals $=\epsilon^{\star}_i$',fontsize=15)
    pl.axes([0.1,0.1,0.8,0.3])
    pl.errorbar(T,StarY,StarEY,fmt='ok',ms=3)
    pl.plot(TT,GG,'r')
    pl.plot(TT,0.*TT+np.max(GG),':b')
    pl.plot(TT,0.*TT+np.min(GG),':b')
    A=np.max(GG)-np.min(GG)
    print('ARTIFIAL DATA, Peak to peak amplitude = ',A)
    pl.title('Artificial Data $= y_i^{\star}$',fontsize=15)
    pl.savefig('ArtificialData.eps')
    pl.clf()
    return A
# ==========================================
#                          Main program
#  - Computations ==========================
n=100
T,Y,EY,BETAsim=Data(n)
BETAfinal=LSF(T,Y,EY)
G=Model(T,BETAfinal)
DT=np.max(T)-np.min(T)
TT=np.min(T)+(np.arange(101)/100.)*DT
GG=Model(TT,BETAfinal)
OrigA=PlotOriginalData(T,Y,EY,G,TT,GG)
Rounds=5
Waitingtime=10
BootA=BootStrap(T,Y,EY,G,Rounds,Waitingtime)
print('==========================================')
print('ORIGINAL DATA peak to peak amplitude = ',OrigA)
print('Bootstrap gave following ',Rounds,' peak to peak amplitude estimates')
print(BootA)
print('Peak to peak amplitude error = ',np.std(BootA))
```