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

# This Python-program was developed using "Enthought Canopy v. 1.4.1
# environment, on a MacBook Pro running OS X 10.10
#
# written by T. Ihn, D-PHYS ETH Zurich, 23 Oct 2014

#
# Commands from libraries
#
from csv import reader

from numpy import arange
from numpy import array
from numpy import sqrt
from numpy import zeros
from numpy import meshgrid
from numpy import where
from numpy import flipud
from numpy import interp

from matplotlib import rcParams

from matplotlib.pyplot import figure
from matplotlib.pyplot import axes
from matplotlib.pyplot import plot
from matplotlib.pyplot import loglog
from matplotlib.pyplot import contourf
from matplotlib.pyplot import contour
from matplotlib.pyplot import colorbar
from matplotlib.pyplot import axis
from matplotlib.pyplot import xticks
from matplotlib.pyplot import yticks
from matplotlib.pyplot import tick_params
from matplotlib.pyplot import xlabel
from matplotlib.pyplot import ylabel
from matplotlib.pyplot import title
from matplotlib.pyplot import show

from scipy.optimize import fmin
from scipy.optimize import brentq
from scipy.optimize import curve_fit
```

```
#
# read data from a csv-file
#
readerout = reader(open("Data_z_t.csv","rb"),delimiter=',');
x = list(readerout);
data = array(x).astype('float')
print "Data array:"
print data

#
# convert data to SI-units
#
data[:,0]=data[:,0]/100; # convert from cm to m
data[:,1]=data[:,1]/1000; # convert from ms to s
print ""
print "Converted data array:"
print data

#
# Analysis using model I following the lecture
#
print "\nModel I:"
# define the mean square error function
def MSE1(g,data):
    diff = data[:,1]-sqrt(2*data[:,0]/g)
    res=diff.dot(diff)/len(data)
    return res

# plot the mean square error function
g = arange(9.759,9.780,0.001)
Q = zeros(shape=(len(g)))
for n in range(0,len(g)):
    Q[n] = MSE1(g[n],data)

figure(
    num=2,
    figsize=(7,5.08),
    dpi=80,
    facecolor='white')
rcParams['axes.linewidth']=2
axes([0.15,0.18,0.82,0.70],
    axisbg='lightgray')
plot(g,1e6*Q,'b-')
```

```
axis([9.759,9.779,2.85e-1,3.3e-1])
xticks([9.760,9.765,9.770,9.775],
    fontsize=24,
    fontname='Times New Roman')
yticks([0.29,0.30,0.31,0.32,0.33],
    fontsize=24,
    fontname='Times New Roman')
tick_params(width=2,length=10)
xlabel(r'$g\,\mathrm{(m/s}^2\mathrm{)}}$',
    fontsize=24)
ylabel(r'$Q\,\mathrm{(ms}^2\mathrm{)}}$',
    fontsize=24)
title('Mean square error for Model I',fontsize=24)
show()

# numerical minimization of the square error function
gg,Qmin,_,_,_ = fmin(MSE1,9.81,args=(data,),full_output=True)
N=len(data)
sigma2_est = N*Qmin/(N-1)

# standard error of the parameters

s_sigma2 = sigma2_est*sqrt(2./(N-3))

# for obtaining the standard error in g we define a function
# for the normalized mean square error minus one
def MSE1norm(s,gg,Qmin,data):
    return (len(data)-1)*(MSE1(gg+s,data)/Qmin-1)-1

# the zero of this function is the desired standard error
sg = brentq(MSE1norm,0,1,args=(gg,Qmin,data))

# output the results
print("g = (%1.4f +/- %1.4f) m/s2"% (gg[0],sg))
print("sigma2 = (%1.3f +/- %1.3f) ms2\n"% (1e6*sigma2_est,1e6*s_sigm

#
# plot data and fit
#
figure(
    num=1,
    figsize=(7,5.08),
    dpi=80,
```

```
    facecolor='white')
rcParams['axes.linewidth']=2
axes([0.15,0.18,0.82,0.70],
    axisbg='lightgray')
loglog(data[:,0],data[:,1],'bo')
loglog(data[:,0],sqrt(2*data[:,0]/gg),'r-')
axis([0.05,1.3,0.1,1])
xticks([0.1,0.2,0.5,0.7,1,1.2],
    fontsize=24,
    fontname='Times New Roman')
yticks([0.1,0.2,0.5,1],
    fontsize=24,
    fontname='Times New Roman')
tick_params(width=2,length=10)
xlabel(r'$z\,\mathrm{(m)}$',
    fontsize=24)
ylabel(r'$t\,\mathrm{(s)}$',
    fontsize=24)
title('Data and fit for Model I',fontsize=24)
show()

# Inspect the residuals of the fit

res1 = data[:,1] - sqrt(2*data[:,0]/gg)

figure(
    num=3,
    figsize=(7,5.08),
    dpi=80,
    facecolor='white')
rcParams['axes.linewidth']=2
axes([0.15,0.18,0.82,0.70],
    axisbg='lightgray')
plot(data[:,0],1e3*res1,'bo')
plot([0,1.4],[0,0],'k-')
axis([0,1.4,-2,2])
xticks(arange(0,1.6,0.2),
    fontsize=24,
    fontname='Times New Roman')
yticks(arange(-2,3,1),
    fontsize=24,
    fontname='Times New Roman')
tick_params(width=2,length=10)
```

```
xlabel(r'$z\,\mathrm{(m)}$',
    fontsize=24)
ylabel(r'$\Delta t\,\mathrm{(ms)}$',
    fontsize=24)
title('Residuals for Model I',fontsize=24)
show()

#
# Analysis using Model II following the lecture
#
print "Model II:"
# define the mean square error function
def MSE2(parms,data):
    diff = data[:,1]-sqrt(2*(data[:,0]-parms[1])/parms[0])
    res=diff.dot(diff)/len(data)
    return res

# Minimization of the mean square error
parms,Qmin,_,_,_ = fmin(MSE2,[9.81,0.001],args=(data,),full_output=T
gg = parms[0]
zz0 = parms[1]
sigma2_est = N*Qmin/(N-2)

# plot the mean square error function
g = arange(9.796,9.804,0.0001)
z0 = arange(-0.0022,-0.0018,0.00001)
Q = zeros(shape=(len(z0),len(g)))
X,Y = meshgrid(g,z0)
for n in range(0,len(g)):
    for m in range(0,len(z0)):
        Q[m,n] = MSE2([g[n],z0[m]],data)

figure(
    num=4,
    figsize=(7,5.08),
    dpi=80,
    facecolor='white')
rcParams['axes.linewidth']=2
axes([0.15,0.18,0.82,0.70],
    axisbg='lightgray')
CS = contourf(X,1e3*Y,1e6*Q,10)
# We draw two contour lines into the plot, the smaller of which is s
CS2 = contour(CS,levels=[1e6*Qmin*(1+1./(N-2)),1e6*Qmin*(1+2./(N-2))
```

```
plot(gg,1e3*zz0,'wo')
title('Mean square error for Model II',fontsize=24)
xlabel(r'$g\,\mathrm{(m/s}^2\mathrm{)}$',
    fontsize=24)
ylabel(r'$z_0\,\mathrm{(mm)}$',
    fontsize=24)
cbar = colorbar(CS)
cbar.ax.set_ylabel(r'$Q(g,z_0) \mathrm{(ms}^2\mathrm{)}$',fontsize=2
show()

# We determine the standard error of the parameters from the contour
cc = CS2.collections[0].get_paths()[0]
CC = cc.vertices
sg = gg-min(CC[:,0])
sz0 = zz0-1e-3*min(CC[:,1])
s_sigma2 = sigma2_est*sqrt(2./(N-4))
# for determining rho, we extract half of the ellipse ...
maxi = where(CC[:,0]==max(CC[:,0]))[0]
mini = where(CC[:,1]==max(CC[:,1]))[0]
range = arange(mini,maxi+1)
CCh = flipud(CC[range,:])
# ... and determine the intercept with z0 = zz0:
intercept = interp(1e3*zz0,CCh[:,1],CCh[:,0])
# now we can calculate rho
rho = -sqrt(1-((intercept-gg)/sg)**2)

# Output the results
print("g = (%1.4f +/- %1.4f) m/s2"% (gg,sg))
print("z0 = (%1.2f +/- %1.2f) mm"% (1e3*zz0,1e3*sz0))
print("sigma2 = (%1.3f +/- %1.3f) ms2"% (1e6*sigma2_est,1e6*s_sigma2
print("rho = %1.2f"% (rho))

#
# plot data and fit
#
figure(
    num=5,
    figsize=(7,5.08),
    dpi=80,
    facecolor='white')
rcParams['axes.linewidth']=2
axes([0.15,0.18,0.82,0.70],
    axisbg='lightgray')
```

```
loglog(data[:,0],data[:,1],'bo')
loglog(data[:,0],sqrt(2*(data[:,0]-zz0)/gg),'r-')
axis([0.05,1.3,0.1,1])
xticks([0.1,0.2,0.5,0.7,1,1.2],
    fontsize=24,
    fontname='Times New Roman')
yticks([0.1,0.2,0.5,1],
    fontsize=24,
    fontname='Times New Roman')
tick_params(width=2,length=10)
xlabel(r'$z\,\mathrm{(m)}$',
    fontsize=24)
ylabel(r'$t\,\mathrm{(s)}$',
    fontsize=24)
title('Data and fit for Model II',fontsize=24)
show()

# Inspect the residuals of the fit

res2 = data[:,1] - sqrt(2*(data[:,0]-zz0)/gg)

figure(
    num=6,
    figsize=(7,5.08),
    dpi=80,
    facecolor='white')
rcParams['axes.linewidth']=2
axes([0.15,0.18,0.82,0.70],
    axisbg='lightgray')
plot(data[:,0],1e3*res2,'bo')
plot([0,1.4],[0,0],'k-')
axis([0,1.4,-2,2])
xticks(arange(0,1.6,0.2),
    fontsize=24,
    fontname='Times New Roman')
yticks(arange(-2,3,1),
    fontsize=24,
    fontname='Times New Roman')
tick_params(width=2,length=10)
xlabel(r'$z\,\mathrm{(m)}$',
    fontsize=24)
ylabel(r'$\Delta t\,\mathrm{(ms)}$',
    fontsize=24)
```

```
title('Residuals for Model II',fontsize=24)
show()

#
# Model I using a shortcut offered by scipy.optimize
#
print('\nModel I with curve_fit:')
# Define fitting function
def fit1(zdata,g):
    return sqrt(2*zdata/g)

# fit the data with scipy.optimize.curve_fit
res = curve_fit(fit1,data[:,0],data[:,1],9.81,full_output=True)
gg = res[0][0]
sg = sqrt(res[1][0,0]) # the function returns the standard error of
print res[3]

# sigma2 estimate
residuals = data[:,1]-fit1(data[:,0],gg)
Qmin = residuals.dot(residuals)/N
sigma2_est = N*Qmin/(N-1)
s_sigma2 = sigma2_est*sqrt(2./(N-3))

print('g = (%1.4f +/- %1.4f) m/s2'% (gg,sg))
print('sigma2 = (%1.3f +/- %1.3f) ms2'% (sigma2_est*1e6,s_sigma2*1e6

#
# Model II using the shortcut offered by scipy.optimize
#
print('\nModel II with curve_fit:')
# Define fitting function
def fit2(zdata,g,z0):
    return sqrt(2*(zdata-z0)/g)

# fit the data with scipy.optimize.curve_fit
res = curve_fit(fit2,data[:,0],data[:,1],[9.81,0.001],full_output=Tr
gg = res[0][0]
zz0 = res[0][1]
# determination of the standard errors and correlation coefficient f
sg = sqrt(res[1][0,0]) # the function returns the standard error of
sz0 = sqrt(res[1][1,1])
rho = res[1][0,1]/sg/sz0
print res[3]
```

```
# sigma2 estimate
residuals = data[:,1]-fit2(data[:,0],gg,zz0)
Qmin = residuals.dot(residuals)/N
sigma2_est = N*Qmin/(N-1)
s_sigma2 = sigma2_est*sqrt(2./(N-4))

print('g = (%1.4f +/- %1.4f) m/s2'% (gg,sg))
print('z0 = (%1.2f +/- %1.2f) mm2'% (zz0*1e3,sz0*1e3))
print('rho = %1.2f'% (rho))
print('sigma2 = (%1.3f +/- %1.3f) ms2'% (sigma2_est*1e6,s_sigma2*1e6
```