```
"""
odr_fit_to_data.py
    A simple program to fit x,y data with uncertainties in both x & y

    Uses Orthogonal Distance Regression

    The examples provided are fits of Gaussian, Lorentzian, or linear
         functions to various data files.

    To fit your own data, you need to change:
    (1) func(p,x) to return the function you are trying to fit,
            p is the parameter vector,
            x are the independent variable(s)
            Caution: scipy.odr and scipy.optimize.curve_fit require
                 x & p in opposite orders.
    (2) the name of the data file read in by numpy.loadtxt,
    (3) the initial guesses (p_guess) for the fit parameters. The fit
             will not converge if your guess is poor.

    For a more accurate estimate of the Cumulative Distribution
        Function (CDF) by Monte Carlo simulation based on the fitted
        values, set
            Run_Monte_Carlo_CDF_Estimator  = True
            Number_of_MC_iterations        = 1000
        But note that this can take some time (e.g. > 1s per
            iteration) if the x uncertainties are large.

    For information on scipy.odr, see
    http://docs.scipy.org/doc/scipy/reference/odr.html

    Copyright (c) 2011-2017 University of Toronto
    Modifications:
        7 January 2018 by David Bailey
            Minor bug fixes, e.g. Cauchy example incorrect
        6 August 2017 by David Bailey
            Fixed some Python 2 vs 3 incompatibility, e.g. removed zip
        14 April 2017 by David Bailey
            Fixed Monte Carlo section errors introduced in last two
            modifications. Some reformating and extra comments.
        7 February 2016 by David Bailey
            Made notation and uncertainties compatible with
            curve_fit_to_data, in particular prevented uncertainty
            estimates being reduced if fit is "too good'
        7 January 2016 by David Bailey
            Made printing python 3 compatible
        11 January 2013 by David Bailey
    Original Version   :   11 September 2011 by Michael Luzi
    Contact: David Bailey <dbailey@physics.utoronto.ca>
                (www.physics.utoronto.ca/~dbailey)
    License: Released under the MIT License; the full terms are
                appended to the end of this file, and are also found
                at www.opensource.org/licenses/mit-license.php
"""
# Use python 3 consistent printing and unicode
from __future__ import print_function
from __future__ import unicode_literals
```

```
import inspect                # docs.python.org/library/inspect.html
import matplotlib
from matplotlib import pyplot # matplotlib.org/api/pyplot_api.html
import numpy                  # numpy.scipy.org/
import scipy                  # scipy.org/
import scipy.odr, scipy.special, scipy.stats

# Decide if Monte Carlo CDF values are desired
Run_Monte_Carlo_CDF_Estimator = True
# The default is set to just 11, so all the examples will run quickly
#    in a few seconds. For serious work, this should be much larger.
Number_of_MC_iterations       = 300

# You can ignore x uncertainties if desired
x_uncertainties_ignored = False

## define some possible functions to be fitted
##  You may, of course, define your own.
def Cauchy(p,x) :
    # Cauchy / Lorentzian / Breit-Wigner peak with contant background:
    B           = p[0]    #   Constant Background
    x1_peak     = p[1]    #   Height above/below background
    x1          = p[2]    #   Central value
    width_x1    = p[3]    #   Half-Width at Half Maximum
    return ( B + x1_peak*(1/(1+((x-x1)/width_x1)**2)) )
def Gauss(p,x) :
    # A gaussian or Normal peak with constant background:
    B           = p[0]    #   Constant Background
    x1_peak     = p[1]    #   Height above/below background
    x1          = p[2]    #   Central value
    width_x1    = p[3]    #   Standard Deviation
    return ( B + x1_peak*numpy.exp(-(x-x1)**2/(2*width_x1**2)) )
def Semicircle(p,x) :
    # The upper half of a circle:
    R           = p[0]    #   Radius of circle
    x0          = p[1]    #   x coordinate of centre of circle
    y0          = p[2]    #   y coordinate of centre of circle
    from numpy import array, sqrt
    y=[]
    for i in range(len(x)) :
        y_squared = R**2-(x[i]-x0)**2
        if y_squared < 0 :
            y.append(y0)
        else :
            y.append(sqrt(y_squared)+y0)
    return array(y)
def Linear(p,x) :
    # A linear function with:
    #   Constant Background         : p[0]
    #   Slope                       : p[1]
    return p[0]+p[1]*x
def Quadratic(p,x) :
    # A quadratic function with:
    #   Constant Background         : p[0]
    #   Slope                       : p[1]
    #   Curvature                   : p[2]
    return p[0]+p[1]*x+p[2]*x**2
```

```
def Cubic(p,x) :
    # A cubic function with:
    #   Constant Background          : p[0]
    #   Slope                        : p[1]
    #   Curvature                    : p[2]
    #   3rd Order coefficient        : p[3]
    return p[0]+p[1]*x+p[2]*x**2+p[3]*x**3
def Lorentzian(p,x):
    y_mu = p[0]
    mu = p[1]
    sigma = p[2]
    return y_mu/(1+ (x-mu)**2/sigma**2)
def Log_Normal(p,x):
    y_mu = p[0]
    mu = p[1]
    sigma = p[2]
    denominator = 2*sigma**2
    numerator = -(numpy.log(x) - numpy.log(mu))**2
    return y_mu*numpy.exp(numerator/denominator)
def Sinc(p,x):
    y_mu = p[0]
    mu = p[1]
    sigma = p[2]
    numerator = numpy.sin((x-mu)/sigma)
    denominator = (x-mu)/sigma
    return y_mu*numpy.abs(numerator/denominator)

## Choose function and data to fit:
func = Sinc
if func.__name__ == "Gauss" :
    # Initial guesses for fit parameters
    p_guess   = (10.,400.,1173.9,0.4,)
    # Gaussian data with large x uncertainties
    #    This data shows how ODR can fit data that would give great
    #    trouble to regular chi-squared fits (that ignore x errors).
    data_file = "gauss_xy_errors_ugly.txt"
    # Labels for plot axes
    x_label   = "energy [KeV]"
    y_label   = "Counts"
elif func.__name__ == "Semicircle" :
    p_guess   = (1,1,1)
    # This data is a semicircle, and shows ODR doing the best it can
    #     with data that is a poor match to the fitting function
    data_file = "semicircle.txt"
    x_label   = "x"
    y_label   = "y"
elif func.__name__ == "Cauchy" :
    p_guess   = (0,1,2, 1)
    # This data is actually generated from a semicircle distribution,
    #   but the fit to a Cauchy is not bad, showing that the same data
    #   can sometimes be well fit by different distributions.
    data_file = "semicircle.txt"
    x_label   = "x"
    y_label   = "y"
elif func.__name__ == "Quadratic" :
    p_guess   = (0,1,2)
    # This data is parabola
```

```
    data_file = "parabola_xy_errors.txt"
    x_label   = "x"
    y_label   = "y"
elif func.__name__ == "Cubic" :
    p_guess   = (0,1,0,0)
    # This data is a cubic
    data_file = "cubic_data.txt"
    x_label   = "x"
    y_label   = "y"
elif func.__name__ == "Lorentzian":
    p_guess = (30,30,10)
    data_file = "ellipse_data.txt"
    x_label = "x"
    y_label = "y"
elif func.__name__ == "Log_Normal":
    p_guess = (100,30,20)
    data_file = "ellipse_data.txt"
    x_label = "x"
    y_label = "y"
elif func.__name__ == "Sinc":
    p_guess = (100,24,4)
    data_file = "ellipse_data.txt"
    x_label = "x"
    y_label = "y"
else :
    # default is linear
    func=Linear
    p_guess = (2,2)
    # Linear data with large x uncertainties
    data_file = "linear_xy_errors.txt"
    x_label   = "x"
    y_label   = "y"

## Load data to fit
#     Any lines in the data file starting with '#' are ignored
#     "data" are values, "sigma" are uncertainties
x_data, x_sigma, y_data, y_sigma = numpy.loadtxt( data_file,
                                                  comments='#',
                                                  unpack = True)
if x_uncertainties_ignored:
    # x uncertainties cannot be set exactly to zero without crashing the
    #    fit, but a tiny value seems to do the job.
    x_sigma = len(x_data)*[1e-99]

# Load data for ODR fit
data = scipy.odr.RealData(x=x_data, y=y_data, sx=x_sigma, sy=y_sigma)
# Load model for ODR fit
model = scipy.odr.Model(func)

## Now fit model to data
#   job=10 selects central finite differences instead of forward
#       differences when doing numerical differentiation of function
#   maxit is maximum number of iterations
#   The scipy.odr documentation is a bit murky, so to be clear note
#        that calling ODR automatically sets full_output=1 in the
#        low level odr function.
fitted = scipy.odr.ODR(data, model, beta0=p_guess, maxit=5000,job=10)
```

```
output = fitted.run()
p   = output.beta      # 'beta' is an array of the parameter estimates
cov = output.cov_beta  # parameter covariance matrix
#"Quasi-chi-squared" is defined to be the
#           [total weighted sum of squares]/dof
#   i.e. same as numpy.sum((residual/sigma_odr)**2)/dof or
#       numpy.sum(((output.xplus-x)/x_sigma)**2
#                                 +((y_data-output.y)/y_sigma)**2)/dof
#   Converges to conventional chi-square for zero x uncertainties.
quasi_chisq = output.res_var
uncertainty = output.sd_beta # parameter standard uncertainties
#  scipy.odr scales the parameter uncertainties by quasi_chisq.
#    If the fit is poor, i.e. quasi_chisq is large, the uncertainties
#    are scaled up, which makes sense. If the fit is too good,
#    i.e. quasi_chisq << 1, it suggests that the uncertainties have
#    been overestimated, but it seems risky to scale down the
#    uncertainties. i.e. The uncertainties shouldn't be too sensistive
#    to random fluctuations of the data, and if the uncertainties are
#    overestimated, this should be understood and corrected
if quasi_chisq < 1.0 :
    uncertainty = uncertainty/numpy.sqrt(quasi_chisq)
if False: # debugging print statements
    print("sd_beta",output.sd_beta)
    print("cov_beta",output.cov_beta)
    print("delta",output.delta)
    print("epsilon",output.epsilon)
    print("res_var",output.res_var)
    print("rel_error",output.rel_error)

######################## PRINT THE RESULTS ############################
print("*************************************************************")
print("               ORTHOGONAL DISTANCE REGRESSION")
# print out version information for debugging
import sys
print("Python",sys.version)
print("Scipy:",scipy.__version__,", Numpy:",numpy.__version__,
             ", Matplotlib:",matplotlib.__version__, sep="")
print("*************************************************************\n")
print("ODR algorithm stop reason: " + output.stopreason[0])
print("\nFit {0} Data points from file: {1}"
                                  .format(len(x_data),data_file))
print("To Model :")
print(" ",inspect.getsource(func))
if x_uncertainties_ignored:
    print("** WARNING: x uncertainties set to zero in fit. **\n")

print("Estimated parameters and uncertainties")
for i in range(len(p)) :
    print(("   p[{0}] = {1:10.5g} +/- {2:10.5g}"+
           "          (Starting guess: {3:10.5g})").\
            format(i,p[i],uncertainty[i],p_guess[i]))

# number of degrees of freedom for fit
dof = len(x_data) - len(p_guess)

# Note: odr, unlike curve_fit, returns the standard covariance matrix.
```

```
print("\nStandard Covariance Matrix : \n", cov, "\n")

print("\nCorrelation Matrix :")
for i,row in enumerate(cov):
    for j in range(len(p)) :
        print("{0:< 8.3g}".
              format(cov[i,j]/numpy.sqrt(cov[i,i]*cov[j,j])), end="")
            # Newbie Notes: "{0:< 8.3g}" left justifies output with
            #   space in front of positive numbers, with 3 sig figs;
            #   end="" suppresses new line.
    print()

# Calculate initial residuals and adjusted error 'sigma_odr'
#                 for each data point
delta   = output.delta   # estimated x-component of the residuals
epsilon = output.eps     # estimated y-component of the residuals
# (dx_star,dy_star) are the projections of x_sigma and y_sigma onto
#    the residual line, i.e.  he differences in x & y between the data
#    point and the point where the orthogonal residual line intersects
#    the ellipse' created by x_sigma & y_sigma.
dx_star = ( x_sigma*numpy.sqrt( ((y_sigma*delta)**2) /
                ( (y_sigma*delta)**2 + (x_sigma*epsilon)**2 ) ) )
dy_star = ( y_sigma*numpy.sqrt( ((x_sigma*epsilon)**2) /
                ( (y_sigma*delta)**2 + (x_sigma*epsilon)**2 ) ) )
sigma_odr = numpy.sqrt(dx_star**2 + dy_star**2)
# residual is positive if the point lies above the fitted curve,
#             negative if below
residual = ( numpy.sign(y_data-func(p,x_data))
             * numpy.sqrt(delta**2 + epsilon**2) )

print(("\nQuasi Chi-Squared/dof   = {0:10.5f}," +
                            " Chi-Squared CDF = {1:10.5f}%").
       format(quasi_chisq,
              100.*float(scipy.special.chdtrc(dof,dof*quasi_chisq))))
print("   WARNING:Above CDF is not valid for large x uncertainties!")

print("\nTo use Monte Carlo simulation to more accurately estimate")
print('       CDF for large x uncertainties, re-run program with ')
print('     "Run_Monte_Carlo_CDF_Estimator = True" and')
print('     "Number_of_MC_iterations >= 1000."', end="")
print(' This may take some time\n')

print("Run_Monte_Carlo_CDF_Estimator = {0}"\
                 .format(Run_Monte_Carlo_CDF_Estimator))

if Run_Monte_Carlo_CDF_Estimator :
    print("\n**** Running Monte Carlo CDF Estimator ****")
    print("Number_of_MC_iterations = {0}"
                .format(Number_of_MC_iterations), end="")
    # Initialize Monte Carlo output distributions
    x_dist           = Number_of_MC_iterations*[None]
    y_dist           = Number_of_MC_iterations*[None]
    p_dist           = Number_of_MC_iterations*[None]
    quasi_chisq_dist = Number_of_MC_iterations*[None]
    # Initialize timing measurement
    import time
    start_time = time.clock()
```

```
    for i in range(Number_of_MC_iterations) :
        # Starting with the x and x uncertainty (x_sigma) values from
        #    the data, calculate Monte Carlo values assuming a normal
        #    gaussian distibution.
        x_dist[i] = numpy.random.normal(x_data,x_sigma)
        # Calculate y using Monte Carlo x, then smear by y uncertainty
        y_dist[i] = numpy.random.normal(func(p,x_data),y_sigma)
        # Fit the Monte Carlo x,y pseudo-data to the original model
        data_dist = scipy.odr.RealData( x=x_dist[i], y=y_dist[i],
                                        sx=x_sigma,  sy=y_sigma)
        model_dist = scipy.odr.Model(func)
        fit_dist = scipy.odr.ODR(data_dist, model_dist, p, maxit=5000,
                                        job=10)
        output_dist = fit_dist.run()
        p_dist[i] = output_dist.beta
        quasi_chisq_dist[i] = output_dist.res_var
    end_time = time.clock()

    print(" simulations in {0} seconds.".\
          format(end_time-start_time))
    # Sort the simulated quasi-chi-squared values
    quasi_chisq_sort = numpy.sort(quasi_chisq_dist)
    # Find the index of the sorted simulated quasi-chi-squared value
    # nearest to the data quasi-chi-squared value, to estimate the cdf
    print("\nFraction of quasi-chisquared values larger" +
                " than observed value:")
    print("    Monte Carlo CDF = {0:6.1f}%".format(
            100.*(1.0-numpy.abs(quasi_chisq_sort-quasi_chisq).argmin()
                                    /float(Number_of_MC_iterations))))
    print("    Minimum, Mean, and Maximum Quasi Chi-Squared values:",
                                end="")
    print("{0:6.2g} {1:6.2g} {2:6.2g}".\
          format(numpy.min(quasi_chisq_dist),
                         numpy.average(quasi_chisq_dist),
                         numpy.max(quasi_chisq_dist)))
    print("\nAverage and Standard Deviation of MC Fit parameters")
    ave_p = numpy.average(numpy.array(p_dist),axis=0)
    std_p = numpy.std(    numpy.array(p_dist),axis=0)
    min_p = numpy.min(    numpy.array(p_dist),axis=0)
    max_p = numpy.max(    numpy.array(p_dist),axis=0)
    for i in range(len(p)) :
        print (("   p[{0}] = {1:12.6g} +/- {2:12.6g} ;"
                + " (Min = {3:12.6f}, Max = {4:12.6f}").format(
                      i, ave_p[i], std_p[i], min_p[i], max_p[i] ))
    print("\nCheck for any Fit Biases in MC Fit parameters"
            +"(Significance and ratio)")
    # Check if fit on Monte Carlo data tends to give an average output
    #    value for the parameters different from the input parameters.
    for i in range(len(p)) :
        print("   p[{0}] = {1:< 6.2f}   ({2:<12.7g}+/-{3:<12.7g})"
              .format(i, (ave_p[i]/p[i]-1)/
                      (std_p[i]/p[i]/numpy.sqrt(Number_of_MC_iterations-1)),
                      ave_p[i]/p[i],
                      std_p[i]/p[i]/numpy.sqrt(Number_of_MC_iterations-1)))

## Plot
```

```
# create figure with light gray background
fig = pyplot.figure(facecolor="0.98")

# 3 rows, 1 column, subplot 1
#   3 rows are declared, but there are only 2 plots; this leaves room
#       for text in the empty 3rd row
fit = fig.add_subplot(211)
# remove tick labels from upper plot (for clean look)
fit.set_xticklabels( () )
pyplot.ylabel(y_label)

pyplot.title("Orthogonal Distance Regression Fit to Data")
# Plot data as red circles, and fitted function as (default) line.
#   For a smooth look,generate many x values for plotting the model
stepsize = (max(x_data)-min(x_data))/1000.
margin = 50*stepsize
x_model = numpy.arange( min(x_data)-margin,max(x_data)+margin,
                                stepsize)
fit.plot(x_data,y_data,'ro', x_model, func(p,x_model),markersize=4)
# Add error bars on data as red crosses.
fit.errorbar(x_data, y_data, xerr=x_sigma, yerr=y_sigma, fmt='r+')
fit.set_yscale('linear')
#   draw starting guess as dashed green line ('r-')
fit.plot(x_model, func(p_guess,x_model), 'g-', label="Start",
                        linestyle="--")

# output.xplus = x + delta
a = numpy.array([output.xplus,x_data])
# output.y = f(p, xfit), or y + epsilon
b = numpy.array([output.y,y_data])
fit.plot( numpy.array([a[0][0],a[1][0]]),
          numpy.array([b[0][0],b[1][0]]),
          'k-', label = 'Residuals')
for i in range(1,len(y_data)):
    fit.plot( numpy.array([a[0][i],a[1][i]]),
              numpy.array([b[0][i],b[1][i]]),
              'k-')
fit.legend(loc='upper left')
fit.grid()

# separate plot to show residuals
residuals = fig.add_subplot(212) # 3 rows, 1 column, subplot 2
residuals.errorbar(x=x_data,y=residual,yerr=sigma_odr,
                            fmt="r+", label = "Residuals")
# make sure residual plot has same x axis as fit plot
residuals.set_xlim(fit.get_xlim())
# Draw a horizontal line at zero on residuals plot
pyplot.axhline(y=0, color='b')
# Label axes
pyplot.xlabel(x_label)
# These data look better in 'plain', not scientific, notation, and if
#   the tick labels are not offset by a constant (as done by default).
pyplot.ticklabel_format(style='plain', useOffset=False, axis='x')
pyplot.ylabel(y_label)

residuals.grid()
pyplot.savefig(func.__name__+"_Fit_Outlier_Adjusted.png")
```

```
pyplot.show()

##### END of odr_fit_to_data.py

"""
Full text of MIT License:

Permission is hereby granted, free of charge, to any person obtaining
a copy of this software and associated documentation files (the
"Software"), to deal in the Software without restriction, including
without limitation the rights to use, copy, modify, merge, publish,
distribute, sublicense, and/or sell copies of the Software, and to
permit persons to whom the Software is furnished to do so, subject to
the following conditions:

The above copyright notice and this permission notice shall be
included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF
ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED
TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT
SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR
ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE
OR OTHER DEALINGS IN THE SOFTWARE.
"""
```