Introduction to Data Analysis and Machine Learning in Physics: 2. Data modeling and fitting

Day 1: 11. April 2022

Martino Borsato, Jörg Marks, Klaus Reygers

Data modeling and fitting - introduction

Data analysis is a process of understanding and modeling measured data. The goal is to find patterns and to obtain inferences allowing to observe underlying patterns.

- There are 2 approaches to statistical data modeling
  - Hypothesis testing: is our data compatible with a certain model?
  - Determination of model parameter: use the data to determine the parameters of a (theoretical) model
- For the determination of model parameter
  - Analysis of data distributions $\rightarrow$ mean, variance, median, FWHM, ...
    allows for an approximate determination of model parameter
  - Data fitting with the least square method $\rightarrow$ an iterative process which minimizes the deviation of a model decribed by parameters from data. This determines the optimal values and uncertainties of the parameters.
  - Maximum likelihood fitting $\rightarrow$ find a set of model parameters which most likely describe the data by maximizing the probability distributions.

The parameter determination by minimization is an integral part of machine learning approaches, here a system learns patterns and predicts related ones. This is the focus in the upcoming days.

Data modeling and fitting - introduction

Data analysis is a process of understanding and modeling measured data. The goal is to find patterns and to obtain inferences allowing to observe underlying patterns.

- There are 2 approaches to statistical data modeling
  - Hypothesis testing: is our data compatible with a certain model?
  - Determination of model parameter: use the data to determine the parameters of a (theoretical) model
- For the determination of model parameter
  - Analysis of data distributions $\rightarrow$ mean, variance, median, FWHM, ...
    allows for an approximate determination of model parameter
  - Data fitting with the least square method $\rightarrow$ an iterative process which minimizes the deviation of a model decribed by parameters from data. This determines the optimal values and uncertainties of the parameters.
  - Maximum likelihood fitting $\rightarrow$ find a set of model parameters which most likely describe the data by maximizing the probability distributions.

The parameter determination by minimization is an integral part of machine learning approaches, here a system learns patterns and predicts related ones. This is the focus in the upcoming days.

Least Square (LS) Method (1)

The method determines the optimal parameters of functions to gaussian distributed measurements.

Lets consider a sample of n measurements y_i and a parametrized description of the measurement $\eta_i = f(x_i|\theta)$ with a parameter set $\theta = \theta_1, \theta_2, ...\theta_k$, dependent values x_i and measurement errors σ_i.

The parameter set should be determined such that

$$S = \sum_{i=1}^{n} \frac{(y_i - \eta_i)^2}{\sigma_i^2} = \sum_{i=1}^{n} \frac{(y_i - f(x_i|\theta))^2}{\sigma_i^2} \longrightarrow \textit{minimal}$$

In case of correlated measurements the covariance matrix of the y_i has to be taken into account. This is accomplished by defining a weight matrix from the covariance matrix of the input data. A decorrelation of the input data should be considered.

S follows a χ^2-distribution with $(n - k)$ degrees of freedom.

Least Square (LS) Method (2)

- Example LS-method

  Often the fit function $f(x, \theta)$ is linear in $\theta = \theta_1, \theta_2, ...\theta_k$

  $f(x|\theta) = \theta_1 f_1(x) + ... + \theta_k f_k(x)$

  If the model is a straight line and our parameters are θ_1 and θ_2 ($f_1(x) = 1$, $f_2(x) = x$) we have $f(x|\theta) = \theta_1 + \theta_2 x$

  The LS equation is

  $$S = \sum_{i=1}^{n} \frac{(y_i - \eta_i)^2}{\sigma_i^2} = \sum_{i=1}^{n} \frac{(y_i - \theta_1 - x_i\theta_2)^2}{\sigma_i^2} \quad \text{and with}$$

  $$\frac{\partial S}{\partial \theta_1} = \sum_{i=1}^{n} \frac{-2(y_i - \theta_1 - x_i\theta_2)}{\sigma_i^2} = 0 \quad \text{and} \quad \frac{\partial S}{\partial \theta_2} = \sum_{i=1}^{n} \frac{-2x_i(y_i - \theta_1 - x_i\theta_2)}{\sigma_i^2} = 0$$

  the parameters θ_1 and θ_2 can be determined.

  In case of linear fit functions solutions can be found by matrix inversion

Least Square (LS) Method (3)

- Use of a nonlinear fit function $f(x, \theta)$ like $\quad f(x|\theta) = \theta_1 \cdot e^{-\theta_2 x}$

  results in the LS equation

  $$S = \sum_{i=1}^{n} \frac{(y_i - \eta_i)^2}{\sigma_i^2} = \sum_{i=1}^{n} \frac{(y_i - \theta_1 \cdot e^{-\theta_2 x_i})^2}{\sigma_i^2}$$

  which we have to minimize

  $$\frac{\partial S}{\partial \theta_1} = \sum_{i=1}^{n} \frac{2e^{-2\theta_2 x_i}(\theta_1 - y_i e^{\theta_2 x_i})}{\sigma_i^2} = 0 \qquad \text{and} \qquad \frac{\partial S}{\partial \theta_2} = \sum_{i=1}^{n} \frac{2\theta_1 x_i e^{-2\theta_2 x_i}(y_i e^{\theta_2 x_i} - \theta_1)}{\sigma_i^2} = 0$$

  In a nonlinear system, the LS Ansatz leads to derivatives which are functions of the independent variable and the parameters $\rightarrow$ no closed solutions

  In general, we have gradient equations which don't have closed solutions. There are a couple of methods including approximations which allow together with numerical methods to find a global minimum, Gauss–Newton algorithm, Levenberg–Marquardt algorithm, gradient descend methods and also direct search methods.

Minuit - a programm package for minimization (1)

In general data fitting and also solving machine learning algorithms lead to a minimization problem of functions. In the 1975-1980 F. James (CERN) developed a FORTRAN-based package, MINUIT, which is a framework to handle multiparameter minimization and compute the best-fit parameter values and uncertainties, including correlations between the parameters.

The user provides a minimization function $F(X, P)$ with the parameter space $P = (p_1, ...p_k)$ and variable space X (also multi-dimensional). There is an interface via functions which influences the minimization process. MINUIT provides error calculations including correlations for the parameter space by evaluating the shape of the function in some neighbourhood of the minimum.

The package has now a new object-oriented implementation as Minuit2 library , written in C++.

During the minimization $F(X, P)$ is evaluated for various X. For the choice of $P = (p_1, ...p_k)$ different methods are used

Minuit - a programm package for minimization (2)

SEEK: Search for the minimum with Monte Carlo methods, mostly used at the start of the minimization with unknown starting values. It is not a converging algorithm.

SIMPLX: Uses the simplex method of Nelder and Mead. Function values are compared in the parameter space. Via step size control the minimum is approached. Parameter errors are only approximate, no covariance matrix is calculated.

MIGRAD: Uses an algorithm of R. Fletcher, which takes the function and the gradient to approach the minimum with a variable metric method. An error matrix and correlation coefficients are available

HESSE: Calculates the hessian matrix of second derivatives and determines the covariance matrix.

MINOS: Calculates (asymmetric) errors using likelihood profiles. The algorithm for finding the positive and negative MINOS errors for parameter n consists of varying n each time minimizing $F(X, P)$ with respect to all the others.

Minuit - a programm package for minimization (3)

Fit process with the minuit package

- The individual steps decribed above can be called several times and in different order during the minimization process.
- Each of the parameters p_i of $P = (p_1, ...p_k)$ can be set constant and released during the minimization steps.
- Problems are expected in models with strong correlation between parameters $\rightarrow$ change model to uncorrelated definitions
- Local minima, edges/steps or undefined ranges in $F(X, P)$ are problematic $\rightarrow$ simplify your model

Minuit2 - The iminuit package

iminuit is a Jupyter-friendly Python interface for the Minuit2 C++ library.

- The class `iminuit.Minuit` instanciates the minuit object. The minimizer function is given as argument. Basic steering of the fit like setting start parameters, error definition and print level is also done here.

```
from iminuit import Minuit
def fcn(x, y, z):                       # definition of the minimizer function
    return (x - 2) ** 2 + (y - x) ** 2 + (z - 4) ** 2
fcn.errordef = Minuit.LEAST_SQUARES  # for Minuit to compute errors correctly
m = Minuit(fcn, x=0, y=0, z=0)       # instanciate minuit, set start values
```

- Several methods determine the interaction with the fitting process, calls to `migrad`, `hesse` or printing of parameters and errors

```
......
m.migrad()                      # run optimiser
print(m.values , m.errors)      # print results
m.hesse()                       # run covariance estimator
```

Minuit2 - iminuit example

- The function `fcn` describes the model with parameters to be determined by data. `fcn` is minimal when the model parameters agree best with data. `fcn` has positional arguments, one for each fit parameter. `iminuit` example fit: 02_fit_exp_fit_iMinuit.ipynb

```
......
x =  np.array([....],dtype='d') # measurements x
y =  np.array([....],dtype='d') # measurements y
dy = np.array([....],dtype='d') # error in y
def xp(a, b , c):
    return a * np.exp(b*x) + c
# least-squares function = sum of data residuals squared
def fcn(a,b,c):
   return np.sum((y - xp(a,b,c)) ** 2 / dy ** 2)
# limit the range of b and fix parameter c
m = Minuit(fcn,a=1,b=-0.7,c=1,limit_b=(-1,0.1),fix_c=True)
m.migrad()                          # run minimizer
m.fixed["c"] = False                # release  parameter c
m.migrad()                          # rerun minimizer
```

Minuit2 - iminuit (3)

- Results and control information of the fit can be printed and accessed in the the prorgamm.

```
......
m = Minuit(fcn,....,print_level=1) # set flag in the initializer
m.migrad()                         # run minimizer
a_fit = m.values['a']              # get parameter value a
a_fit_error =  m.errors['a']       # get parameter error of a
print (m.values,m.errors)          # print results
```

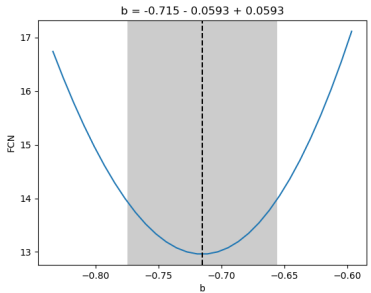
- After processing Hesse, covariance and correlation information of the fit is available

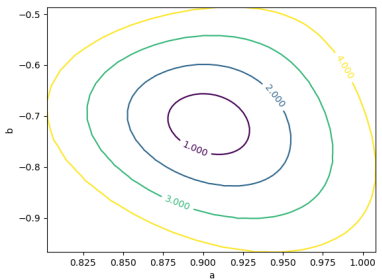
```
......
m.hesse()                           # run covariance estimator
m.matrix()                          # get covariance matrix
m.matrix(correlation=True)          # get full correlation matrix
cov = m.np_matrix()                 # save matrix to numpy
cor = m.np_matrix(correlation=True)
print(cor[0, 1])      # print correlation between parameter 1 and 2
```

Minuit2 - iminuit (4)

- Minos provides asymmetric uncertainty intervals and parameter contours by scanning one parameter and minimizing the function with respect to all other parameters for each scan point. Results are displayed with `matplotlib`.

```
......
m.minos()
print (m.get_merrors()['a'])
m.draw_profile('b')
m.draw_mncontour('a', 'b', nsigma=4)
```





Exercise 3

Plot the following data with matplotlib as in the iminuit example:

```
x:  0.2,0.4,0.6,0.8,1.,1.2,1.4,1.6,1.8,2.,2.2,2.4,2.6,2.8,3.,3.2,
    3.4,3.6, 3.8,4.
y:  0.04,0.021,0.035,0.03,0.029,0.019,0.024,0.018,0.019,0.022,0.02,
    0.025,0.018,0.024,0.019,0.021,0.03,0.019,0.03,0.024
dy: 1.792,1.695,1.541,1.514,1.427,1.399,1.388,1.270,1.262,1.228,1.189,
    1.182,1.121,1.129,1.124,1.089,1.092,1.084,1.058,1.057
```

- ☐ Exchange in the example iminuit fit `02_fit_exp_fit_iMinuit.ipynb` the exponential function by a 3rd order polynomial and perform the fit
- ☐ Compare the correlation of the parameters of the exponential and the polynomial fit
- ☐ What defines the fit quality, give an estimate

Solution: 02_fit_ex_3_sol.ipynb

Exercise 4

Plot the following data with matplotlib:

```
x:   1, 2, 3, 4, 5, 6, 7, 8, 9, 10
dx:  0.1,0.1,0.5,0.1,0.5,0.1,0.5,0.1,0.5,0.1
y:   1.1,2.3,2.7,3.2,3.1,2.4,1.7,1.5,1.5,1.7
dy:  0.15,0.22,0.29,0.39,0.31,0.21,0.13,0.15,0.19,0.13
```

- ☐ Perform a fit with iminuit. Which model do you use?
- ☐ Plot the resulting fit function in the graph with the data
- ☐ Print the covariance matrix. Can we improve the errors.
- ☐ Can you draw a contour plot of 2 of the fit parameters.

Solution: 02_fit_ex_4_sol.ipynb

PyROOT

PyROOT is the python binding for the C++ data analysis toolkit ROOT developed with and for the LHC community. You can access the full ROOT functionality from Python while benefiting from the performance of the ROOT C++ libraries. The PyROOT bindings are automatic and dynamic and are able to interoperate with widely-used Python data-science libraries as `NumPy`, `pandas`, SciPy `scikit-learn` and `tensorflow`.

- ROOT/PyROOT can be installed easily within anaconda3 (ROOT version 6.22.02 or later) or is available in the CIP jupyter2 Hub
- Tools for statistical analysis, a math library with optimized algorithms, multivariate analysis, visualization and simulation of data.
- Storing data including objects and classes with compression in files is a very powerfull aspect for any data analysis project
- Within PyROOT Minuit2 can be accessed easily either with predefined functions or your own function definition
- For advanced statistical analyses and data modeling likelihood fitting with the packages **rooFit** and **rooStats** is available.

- Example reading the invariant mass measurements of a D^0 from a text file and determine μ and σ 02_fit_histFit.ipynb run with: python3 -i 02_fit_histFit.py

```
import numpy as np
import math
from ROOT import TCanvas, TFile, TH1D, TF1, TMinuit, TFitResult
data = np.genfromtxt('D0Mass.txt', dtype='d') # read data from text file
c = TCanvas('c','D0 Mass',200,10,700,500)     # instanciate output canvas
d0 = TH1D('d0','D0 Mass',200,1700.,2000.)     # instanciate histogramm
for x in data :                               # fill data into histogramm d0
     d0.Fill(x)
def pyf_tf1_params(x, p):                     # define fit function
     return p[0] * math.exp (-0.5 * ((x[0] - p[1])**2 / p[2]**2))
func = TF1("func",pyf_tf1_params,1840.,1880.,3)
# func = TF1("func",'gaus',1840.,1880.)  # use predefined function
func.SetParameters(500.,1860.,5.5)    # set start parameters
myfit = d0.Fit(func,"S")              # fit function to the histogramm data
print ("Fit results: mean=",myfit.Parameter(0)," +/- ",myfit.ParError(0))
c.Draw()                                      # draw canvas
myfile = TFile('myOutFile.root','RECREATE')   # Open a ROOT file for output
c.Write()                                     # Write canvas
d0.Write()                                    # Write histogram
myfile.Close()                                # close file
```

Fit Options

The list of fit options is given in parameter option.

option	description
"W"	Set all weights to 1; ignore error bars
"U"	Use a User specified fitting algorithm (via SetFCN)
"Q"	Quiet mode (minimum printing)
"V"	Verbose mode (default is between Q and V)
"E"	Perform better Errors estimation using Minos technique
"B"	User defined parameter settings are used for predefined functions like "gaus", "expo", "poln", "landau". Use this option when you want to fix one or more parameters for these functions.
"M"	More. Improve fit results. It uses the IMPROVE command of **TMinuit** (see **TMinuit::mnimpr**). This algorithm attempts to improve the found local minimum by searching for a better one.
"R"	Use the Range specified in the function range
"N"	Do not store the graphics function, do not draw
"0"	Do not plot the result of the fit. By default the fitted function is drawn unless the option "N" above is specified.
"+"	Add this new fitted function to the list of fitted functions (by default, any previous function is deleted)
"C"	In case of linear fitting, do not calculate the chisquare (saves time)
"F"	If fitting a polN, use the minuit fitter
"EX0"	When fitting a **TGraphErrors** or TGraphAsymErrors do not consider errors in the coordinate
"ROB"	In case of linear fitting, compute the LTS regression coefficients (robust (resistant) regression), using the default fraction of good points "ROB=0.x" - compute the LTS regression coefficients, using 0.x as a fraction of good points
"S"	The result of the fit is returned in the **TFitResultPtr** (see below Access to the Fit Result)

Exercise 5

Read text file FitTestData.txt and draw a histogramm using PyROOT.

- Determine the mean and sigma of the signal distribution. Which function do you use for fitting?
- The option S fills the result object.
- Try to improve the errors of the fit values with minos using the option E and also try the option M to scan for a new minimum, option V provides more output.
- Fit the background outside the signal region use the option R+ to add the function to your fit

  Solution: 02_fit_ex_5_sol.ipynb

iPython Examples for Fitting

The different python packages are used in example iPython notebooks to demonstrate the fitting of a third order polynomial to the same data available as numpy arrays.

- LSQ fit of a polynomial to data using Minuit2 with iminuit and matplotlib plot:
  02_fit_iminuitFit.ipynb
- Graph fitting with pyROOT with options using a python function including confidence level plot:
  02_fit_fitGraph.ipynb
- Graph fitting with numpy and confidence level plotting with matplotlib:
  02_fit_numpyFit.ipynb
- Graph fitting with a polynomial fit of scikit-learn and plotting with matplotlib:
  02_fit_scikitFit.ipynb