Exam Project - Fitter with the Gradient Descent method

Scientific Programming II - High Performance Computing

Irene Celestino

July 15, 2025

1 Introduction

The project goal is a program capable of performing a χ^2 or likelihood fit on a set of binned or unbinned data, using as minimisation strategy the Gradient descent. The user can decide the type of fit (χ^2 or Poisssonian likelihood, binned or unbinned) to perform and the fit model among a set of available functions. In addition, there is the possibility to test the fit performance using a random number generator that can generate data according to a given probability density function.

The program should be able to:

- Give the possibility to the user to choose the fit strategy: the fit model can be chosen from a set of predefined functions; the function to be minimised, called from now on `FCN`, can be chosen to be a χ^2 or a $-2\log\mathscr{L}$. A more detailed definition of the available `FCN`s is reported in Section 2.1;
- Take as input a dataset provided by the user or to randomly generate the data according to a given probability density function, which depends on the chosen model;
- Minimize the given `FCN` using the iterative gradient descent method, with the gradient computed numerically at each iteration. Return the minimum `FCN` and the best estimate for the parameters;
- Estimate the parameter covariance matrix: at first the Hessian matrix is estimated numerically as the second derivative of the `FCN` around the best fit parameters; then, the Hessian matrix is inverted using the following functions of the `LAPACK` package: `dgetrf` for the LU decomposition and `dgetri` for the inversion of the LU-decomposed matrix;
- Export the fit results and the original data in a text file, which can be read by python, to plot the fit and the pulls.

2 Description of the Methodology

2.1 χ^2 and likelihood fit

Likelihood-based fitting aims to find the parameters $\vec{p}$ that maximize the likelihood $\mathscr{L}(\vec{p}|X)$ of observing the data X. In practice, one minimizes the quantity

$$\lambda(\vec{p}|X) \equiv -2\log\frac{\mathscr{L}(\vec{p}|X)}{\max_{\vec{q}}\mathscr{L}(\vec{q}|X)}\,,$$

which has very useful properties from the statistical point of view. The fitting strategy depends on the choice of $\mathscr{L}(\vec{p}|X)$, which depends on the features of data. In particular, if the input dataset is *unbinned*, i.e. it is given as a set of measured (x_i, y_i, σ_{y_i}), a standard assumption is that the likelihood is Gaussian. In detail, given a set of predictions $f(x_i;\vec{p})$ for y_i, which depends on a set of parameters $\vec{p}$, the Gaussian Likelihood can be written as:

$$\mathscr{L}_{\text{Gauss}}(\vec{p}|X) = \prod_{i=1}^{N}\frac{1}{\sqrt{2\pi}\sigma_{y_i}}e^{-\left(\frac{y_i-f(x_i,\vec{p})}{\sigma_{y_i}}\right)^2}\,.$$

The maximum likelihood is obtained for the set of parameters such that $f(x_i,\vec{p}_{\text{best}}) = y_i$:

$$\max_{\vec{q}}\mathscr{L}_{\text{Gauss}}(\vec{q}|X) = \prod_{i=1}^{N}\frac{1}{\sqrt{2\pi}\sigma_{y_i}}$$

Therefore, $\lambda(\vec{p}|X)$ can be estimated as

$$\lambda_{\text{unbinned, Gauss}}(\vec{p}|X) = -2\log\frac{\mathscr{L}_{\text{Gauss}}(\vec{p}|X)}{\max_{\vec{q}}\mathscr{L}_{\text{Gauss}}(\vec{q}|X)} = \sum_{i=1}^{N}\left(\frac{y_i - f(x_i,\vec{p})}{\sigma_{y_i}}\right)^2\,. \tag{1}$$

This is the standard χ^2 function, which measures the squared deviation of the data from the model, normalized by uncertainties.

Another possible case is that the input dataset is *binned*, i.e. it is given in the form of a histogram, where x_i represents the bin center in the i-th bin, y_i represents the number of counts in the i-th bin and $\sigma_{y_i} = \sqrt{y_i}$. In this context, defining $f(x_i;\vec{p})$ as the predicted count in the i-th bin, a Poisson likelihood is more appropriate:

$$\mathscr{L}_{\text{Poiss}}(\vec{p}|X) = \prod_{i=1}^{N}\frac{f(x_i,\vec{p})^{y_i}e^{-f(x_i,\vec{p})}}{y_i!}\,.$$

Also in this case, the maximum likelihood is obtained for the set of parameters such that $f(x_i,\vec{p}) = y_i$, so that $\lambda(\vec{p}|X)$ is given by

$$\lambda_{\text{binned, Poiss}}(\vec{p}|X) = -2\log\frac{\mathscr{L}_{\text{Poiss}}(\vec{p}|X)}{\max_{\vec{q}}\mathscr{L}_{\text{Poiss}}(\vec{q}|X)} = 2\sum_{i=1}^{N}\left(f(x_i,\vec{p}) - y_i + y_i\log\frac{y_i}{f(x_i,\vec{p})}\right)\,. \tag{2}$$

In the limit of a histogram populated by a large number of events, since the limit of a Poissonian with mean μ is a Gaussian with mean μ and sigma $\sqrt{\mu}$, it is also possible to perform a binned fit on a histogram using a Gaussian Likelihood, which yields the following expression for $\lambda(\vec{p}|X)$:

$$\lambda(\vec{p}|X)_{\text{binned, Gauss}} \approx \sum_{i=1}^{N} \left(\frac{y_i - f(x_i, \vec{p})}{\sqrt{y_i}} \right)^2 . \tag{3}$$

Therefore, given a dataset, the `FCN` that needs to be minimised with respect to the model parameters $\vec{p}$ can be:

1. $\lambda(\vec{p}|X)_{\text{unbinned, Gauss}}$ for an *unbinned* fit with Gaussian Likelihood (1)
2. $\lambda(\vec{p}|X)_{\text{binned, Poiss}}$ for a *binned* fit with Poissonian Likelihood (2)
3. $\lambda(\vec{p}|X)_{\text{binned, Gauss}}$ for a *binned* fit with Gaussian Likelihood (3).

2.2 Gradient Descent Minimisation

Once the $\lambda(\vec{p}|X)$ function is chosen according to what is described in the previous section, the fitting procedure consists of minimizing it with respect to all the parameters $\vec{p}$. The strategy chosen to perform this multivariate minimisation is the Gradient Descent method. At first, it is necessary to set an objective function `FCN` (in this case $\lambda(\vec{p}|X)$), which depends on a set of parameters $\vec{p}$, of which initial guesses $\vec{p}^{(0)}$ are given. The best fit parameters, which minimize the `FCN`, are estimated with an iterative procedure:

- The starting point is an initial set of parameters $\vec{p} = \vec{p}^{(0)}$;
- The gradient of the `FCN`, which is a vector of the same size of $\vec{p}$, is computed numerically. The i-th component is given by:

$$\nabla\texttt{FCN}_i = \frac{\texttt{FCN}(p_0^{(0)}, p_1^{(0)}, ..., \boldsymbol{p_i^{(0)}} + \boldsymbol{\delta_i}, ...p_n^{(0)}) - f(p_0^{(0)}, p_1^{(0)}, ..., \boldsymbol{p_i^{(0)}} - \boldsymbol{\delta_i}, ...p_n^{(0)})}{2\delta_i} ,$$

  with $\vec{\delta}$ a set of **steps**, one for each parameter, which can be given as input or set as default to 10^{-6};
- The parameters are then updated in the opposite direction of the gradient, according to a set of learning rates $\vec{\alpha}$:

$$\vec{p}_i^{(1)} = \vec{p}_i^{(0)} - \alpha_i \nabla\texttt{FCN}_i .$$

  The learning rates are as default taken equal to the steps.
- The iterative procedure is continued until the maximum number of iterations is reached or $|\vec{\nabla}\texttt{FCN}|$ falls below a fixed **tolerance** h, which can be given as input or set as default to 10^{-6}.

2.3 Covariance Matrix Estimate

Once the minimum of the objective function `FCN` has been reached through the gradient descent procedure, it is important to estimate the uncertainty on the fitted parameters. In likelihood-based fits, this is typically done by computing the *covariance matrix*, which provides a quantitative estimate of both the variances (on-diagonal elements) and the correlations (off-diagonal elements) between the fit parameters. The covariance matrix V is estimated as the inverse of the Hessian matrix H, evaluated with the parameters that minimize the FCN. The Hessian H is the matrix of second partial derivatives of the objective function:

$$H_{ij} = \frac{\partial^2 \texttt{FCN}}{\partial p_i \partial p_j} \,.$$

Numerically, each second derivative can be estimated using finite difference methods. If h_i and h_j are small displacements in the i-th and j-th directions, the second derivative can be approximated as:

$$H_{ij} \approx \begin{cases} \dfrac{\texttt{FCN}_{(i+)} - 2\texttt{FCN}_{\text{min}} + \texttt{FCN}_{(i-)}}{4\delta_i^2} & \text{if } i = j \\[2ex] \dfrac{\texttt{FCN}_{(i+j+)} - \texttt{FCN}_{(i+j-)} - \texttt{FCN}_{(i-j+)} + \texttt{FCN}_{(i-j-)}}{4\delta_i\delta_j} & \text{if } i \neq j \end{cases}$$

with $\vec{p}$ the best-fit parameters, $\texttt{FCN}_{\text{min}}$ the FCN computed with the best-fit parameters and

$$\texttt{FCN}_{(i\pm j\mp)} = \texttt{FCN}(p_0, p_1, ..., \boldsymbol{p_i} \pm \boldsymbol{\delta_i}, ..., \boldsymbol{p_j} \mp \boldsymbol{\delta_j}, ..., p_n) \,.$$

A well-defined minimum corresponds to a *positive-definite* Hessian, with all diagonal elements positive, indicating that the FCN has a convex shape in the neighborhood of the minimum.

Once the Hessian has been computed, its inverse is obtained, giving the covariance matrix. In this project, the Hessian matrix is inverted using the following functions of the LAPACK package: `dgetrf` for the LU decomposition and `dgetri` for the inversion of the LU-decomposed matrix. The square roots of the diagonal elements of V provide the *errors* on each fitted parameter:

$$\sigma_i = \sqrt{V_{ii}} \,,$$

while the off-diagonal elements give the *correlation* between two parameters:

$$\rho_{ij} = \frac{V_{ij}}{\sigma_i \sigma_j}$$

3 Input

The program requires an **input data card** specifying the generation and fitting options. The user must provide the path to this input card as a command-line argument when executing the program. In addition to standard execution, the program supports several special modes, including:

- Help mode (`--help`, `-h`): displays usage instructions and input format. Within the help mode, the user can request the generation of a default input card template;
- Model details mode (`--model` or `-m`): prints more detailed information about the available models and their parameters.

If the command-line argument is not one of the above, it is taken as the input data card path and, if the corresponding file does not exist, the program stops with an error message. If the number of command-line arguments is different than 1, the program prints instructions for the three available modes (Help, Model details, input card).

The input card is a text file where each line defines an option as `key value(s)`, with the first column being the option name and the rest its value(s). The divider between the key and its values can be one (or more) spaces or one (or more) characters in the following list: $\{= \; : \; , \; ; \; | \; \}$. For certain options, such as the initial parameters, it is possible to specify multiple values on the same line, provided that the number of values matches the expected number of parameters for the chosen fit model. The options can be specified in any order and only two of them are compulsory (the likelihood type and the fit model), while the other ones are optional, so they can be either omitted or reported with no values. If the compulsory options are not provided with an acceptable value, the program stops, while if the optional options are not provided, the program assigns them a default value. All the assigned values are printed as a recap to the terminal by the program after the input card has been successfully read. Any content after the first empty line in the card will be ignored. Lines beginning with "!" or "#" or "/" are identified as comments and ignored. Also inline comments are allowed and ignored. Some of the configurable options include:

- The user can choose the fit type, which is an integer: 1 for *unbinned* fit with Gaussian Likelihood (1), 2 for *binned* fit with Poissonian Likelihood (2) or 3 for *binned* fit with Gaussian Likelihood (3).
- The user can choose the fit model $f(x_i, \vec{p})$ from a list of predefined analytical options. Some examples are:
  - For *binned* fits: Gaussian, Exponential
  - For *unbinned* fits: Polynomial (Linear, Quadratic)

  The user can optionally provide the initial guesses of the parameters, the steps, and the maximum number of iterations for the minimisation procedure.
- The user can optionally provide a path to the data text file, which can have one column for binned fits, containing x_i data, which will be reported in a histogram by the program, or three columns for unbinned fits containing (x_i, y_i, σ_{y_i}) data. If the format of the text file is not what the program expects from the fit type option, it stops with an error message.
- If no data is provided by the user, the fit is performed on a set of data randomly generated by the program. This generation is done using the model chosen by the user, setting the parameters to the initial guess or, if they are not provided, using default parameters. The user can choose the number of randomly generated data in the input card.
- The user can choose the name of the text file to save the generated data in a format that can be read as input data file when running again the program. i.e. with one column for binned datasets and three columns for unbinned datasets.
- The user can choose the name of the text file to save the data and fit results, the name of another text file in which to save the full fit statistics and the name of a *verbose* log file, in which all the details of each iteration of the fitting procedure are reported.

4 Output

- The program always outputs to the terminal the fit results and the covariance matrix and can optionally print it to a dedicated file;
- The program outputs (if requested in the input card) a text column file containing the input data (or the histogram in binned fits) (x_i, y_i, σ_{y_i}), the model predictions $f(x_i, \vec{p})$ and the fit statistics ($\min\lambda$, ndof, p_i, σ_i, ρ_{ij});
- The program outputs (if requested in the input card) a text column file in which the generated dataset is saved in a format that can be read as input data file when running again the program. i.e. with one column for binned datasets and three columns for unbinned datasets;
- The program can output optionally in a provided *verbose* log file details of all the iterations of the minimisation strategy, so that the user can understand, for instance, what happened if the fit did not converge;
- The project includes a python script, employing numpy and matplotlib, to plot the data with the best-fit model overlaid and the pulls. The python script takes as input the text column file with data and predictions produced by the Fortran program. The Fortran program automatically runs the python script to produce the plot.