University Bonn Winter Term 2025/2026

Statistical Methods of Data Analysis

Lecture: PD Dr. P. Bechtle, Dr. M. Prim

Exercises: C. Breuning

Final Project

Deadline: 04.03.2026 23h55

This final project will make up 50 % of your grade. It consists of a programming part and a written documentation/report. Details and formalities are outlined below.
If you work on the course's JupyterHub and there are packages missing we can install them for you (and all others) on the JupyterHub instance.

General remarks

- Deadline for the submission: 04.03.2026 23h55. Submission is via an eCampus submission, similar to the exercises (but the submission will be located on the main site of the eCampus course). While you may collaborate in teams of two on certain parts of the project (details provided below), each individual must submit their report **and** code themselves.
- You can work on the programming part of the project in teams up to 2 persons. Please indicate in your code file and on your report who your partner is. You can submit the same code file as your partner.
- Everybody has to write the report in their own words. Sharing text among team members for this document is prohibited, as the report will determine your grade.
- The length of the report is 4-6 pages using the provided LaTeX-template. This includes all figures, references, ... We are aware that this limit is very tight. Don't worry if your explanations are short, concise and on the point.
- Your report should include an introduction, a main part and a conclusion. Also, indicate in the title the task you worked on. You find more details on the content of the report in the task descriptions below. Don't forget citations where necessary.
- Passing and Grading: To receive a passing grade in this project, your code must run (specifications see below), and your report needs to reach a certain number of points. The grade is then determined by the number of points you reach.
  - Your code must run for $\mathcal{L}_G$ (definitions etc. see below). The benchmark here is that it needs to run on the course's JupyterHub. It is acceptable if certain edge cases do not work; however, the code should run and give sensible results for a simple example (i.e. using $\mathcal{L}_G$ in a straightforward case). More specific information related to the tasks can be found outlined below.

  - When your code runs, as described, you will get a passing grade if you reach half of the points available for the report.

- Usage of ChatGPT/AI (will just be called AI from now on):
  - As a general rule of thumb: treat AI tools similar to other references. If you copy something from an AI, you have to add a citation where you got it from.
  - For the programming part: you have to specify exactly which code was copied from an AI. It is not sufficient to say something like "Some of this code was copied from this AI". You have to specify the exact code parts that were copied from an AI, so it is clear which code was written by you and which was copied from an AI. Also include which AI was used.
  - It is not allowed to let an AI write (parts of) your report. This is similar to just copying a journal article or webpage, putting it in quotes and handing this in. This would not be your work and would not give you a passing grade. Thus, letting AI write (parts of) your report would lead to the same consequence.
  - For citations in your report, treat AI similar to Wikipedia, ...: this is not sufficient/serious enough as a sole source for most citations. If you cite something in your report use journal articles, books, databases, ... where possible.

Final project description and tasks

In this project you will work on a fitting framework. The goal is not primarily to do an actual fit to some data but implement one part (algorithm) of the fitting framework yourself. To do so, we provide you with a fitting framework (which also includes a lot of comments to explain some parts to you) that uses the `iMinuit` functions and you will replace one of the functions with your own implementation. To make it more conventient for you (i.e. so you don't have to wait a long time for the fits) we provide you with the complete likelihood-function together with the "model predictions" of the parameters and their correlations and uncertainties. This makes the fit itself somewhat trivial, but this also means it will work fast and you can cross check your results at all times. Also, the goal is not to perform a fit to data but implement one part of the fitting framework yourself.

Getting this likelihood from the data, including the systematic uncertainties, can be very tedious and depending on the parameters complicated. Thus, for tasks 1 to 5, assume this has been already done and you have the given likelihood evaluated for the measured data.

Problem definitions

Let $\vec{\theta} = (\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6)^T \in \mathbb{R}^6$ be the set of parameters to be fitted, $\vec{\mu} \in \mathbb{R}^6$ the model predictions and $V \in \mathbb{R}^6 \times \mathbb{R}^6$ the covariance matrix of the predictions. For now, consider a six-dimensional gaussian as likelihood:

$$\mathcal{L}_G(\vec{\theta}) = \frac{1}{(2\pi)^3\sqrt{\det(V)}} \exp\left(-\frac{1}{2}\left((\vec{\theta}-\vec{\mu})^T V^{-1}(\vec{\theta}-\vec{\mu})\right)\right).$$

A physics example for this setup could be a fit to data, where we have two parameters of interest (θ_1, θ_2) and 4 systematic uncertainties, which can be parametrised through additional nuisance parameters. In the limit of the cental limit theorem, i.e. with enough data, we can write them as gaussians, which is why we consider a 6D gaussian. This is a very simplified likelihood, but sufficient for this project. However, one of the tasks below will consider the actual likelihood you could get for a binned maximum likelihood fit.

Introduction [10 Points]

Include a *brief* introduction to the project and your task in the report, as every (lab) report, article,... should have. Then work on the following tasks and include the answers to the non-programming tasks in your report.

- Implement $-2\log\mathcal{L}_G$ so you can use it in the fitting framework. The function should have 6 input parameters $(\theta_1,\ldots,\theta_6)$ and return one number, the value of the evaluated functions. *Hint: Calculating and simplifying $-2\log\mathcal{L}_G$ first and implementing this, can be benficial for some later tasks. Also, when doing this, you can drop terms, that are irrelevant for the fit (explain if you do).*
- Which form will $-2\log\mathcal{L}_G$ have?
- Getting familiar with the fitting framework (use the given fitting framework for the following tasks):
  - Initialize a fit object with your implemented negative log-likelihood
  - Do the minimztion and give the optimal parameters with their Hesse errors
  - Use the provided functions of the Fitting Framwork to plot the error ellipsis (1σ) and the profile-likelihood at the same interval. Do this for a couple different parameter pairs (you do not need to do this for all possible combinations). Compare to the given model predictions and covariance/correlation matrix. Include **at most** one of the plots you created in this task in your report.

Implementing a part of the framework yourself [40 Points]

Independent of the task you will choose for this part, **everybody needs to provide a documentation of the written code in the report.** Provide a detailed explanation of your implementation/your code. If you made specific choices in the implementation for performance or for other reasons, include them here and explain them. If you had to make choices for constants, precisions, termination criteria, step sizes, ... explain your choices. The tasks might also contain additional tasks which you have to include into your report (like visualizations or comparisons). Details are explained below. The plots you create for the visualizations should be included in the report and are considered part of the documentation, i.e. will be part of the grade.

The goal of this part is to replace some of the `iMinuit`/fitting framework functionality with your own implementation. **Choose only one** of the following tasks (1-6).

Tasks 1-5: Write your code, such that it works with different negative log-likelihoods. You can test and run everything in this part with $-2\log\mathcal{L}_G$, but you will test another negative log-likelihood later.

1. **Gradient Descent**: In this task you replace the minimization procedure of `iMinuit` to find the optimal parameters.
   Passing criteria for the code: You should be able to find the correct mininmum of $-2\log\mathcal{L}_G$ (you can compare with the given $\vec{\mu}$ to check your results). It does not need to work for all initial guesses, but should work for initial guesses which are not already the optimal parameters or in the immediate vicinity of the optimal parameters.
   - 10 Points Explain the idea of the Gradient Descent method and how it works.

- 10 Points (for documentation of the implementation) In the `minimize` function of the fitting framework, remove the call of the `migrad()` function (which is a more sophisticated optimization algorithm) and implement a Gradient Descent algorithm to find and store the optimal parameters. For code readability, etc. it is of course allowed to add additional functions. You don't have to squeeze everything inside the `minimize` function. **Note:** You should implement this algorithm yourself. It is not allowed to use pre-built functions in some `Python` library that does the algorithm for you. There should be a working version of a Gradient Descent algorithm written by you in your submitted code that finds the best parameters for the fit. Except for that, you are, of course, allowed to use the usual `numpy`, ... functionalities.
- 10 Points Implement a visualization of your algorithm: you should be able to choose two parameters and then show for each step, how the algorithm proceeds in the plane of these two parameters from the initial guess to the found, best parameters. Show this for different initial guesses for the two parameters of interest and try to explain the taken path, including your knowledge on the form of the negative log-likelihood.
- 10 Points Compare your algorithm to the implementation you replaced. Compare the results (best parameters) and the performance of your algorithm with the `migrad()` call that was used before.

2. **BFGS**: In this task you replace the minimization procedure of `iMinuit` to find the optimal parameters.
   Passing criteria for the code: You should be able to find the correct mininmum of $-2\log\mathcal{L}_G$ (you can compare with the given μ to check your results). It does not need to work for all initial guesses, but should work for initial guesses which are not already the optimal parameters or in the immediate vicinity of the optimal parameters.

   - 10 Points Explain the idea of the BFGS method and how it works.
   - 10 Points (for documentation of the implementation) In the `minimize` function of the fitting framework, remove the call of the `migrad()` function and implement a BFGS algorithm. For code readability, etc. it is of course allowed to add additional functions. You don't have to squeeze everything inside the `minimize` function. **Note:** You should implement this algorithm yourself. It is not allowed to use pre-built functions in some `Python` library that does the algorithm for you. There should be a working version of a BFGS algorithm written by you in your submitted code that finds the best parameters for the fit. Except for that, you are, of course, allowed to use the usual `numpy`, ... functionalities.
   - 10 Points Implement a visualization of your algorithm: you should be able to choose two parameters and then show for each step, how the algorithm proceeds in the plane of these two parameters from the initial guess to the found, best parameters. Show this for different initial guesses for the two parameters of interest and try to explain the taken path, including your knowledge on the form of the negative log-likelihood.
   - 10 Points Compare your algorithm to the implementation you replaced. Compare the results (best parameters) and the performance of your algorithm with the `migrad()` call that was used before. If your algorithm performs significantly worse, what could you improve?

3. **Hesse matrix**: In this task you calculate the Hesse errors yourself.
   Passing criteria for the code: You should be able to find the correct Hesse errors and covariance matrix for $-2\log\mathcal{L}_G$ (you can compare with the given V to check your results).

- 10 Points Choose a **numerical** algorithm to calculate the Hesse matrix and calculate the Hesse errors from the Hesse matrix. Explain the idea of the algorithm and how it works. You can assume you found the optimal parameters (and do so by running the implemented `minimize()` function in the fitting framework with the `iMinuit` tools).
- 10 Points (for documentation of the implementation) In the `calc_hesse` function of the fitting framework, remove the call of the `hesse()` function and implement an algorithm to numerically calculate the Hesse matrix and Hesse errors. For code readability, etc. it is of course allowed to add additional functions. You don't have to squeeze everything inside the `calc_hesse` function. **Note:** You should implement this algorithm, i.e. the numerical calculation of the Hesse matrix, yourself. It is not allowed to use pre-built functions in some `Python` library that does the numerical calculations for you. There should be a working implementation of the numerical calculations and the finding of the errors in your submitted code. Except for that, you are, of course, allowed to use the usual `numpy`, ... functionalities.
- 10 Points Think about a way to visualize your chosen numerical algorithm and include the visualization in your report. If you are unsure whether the visualization you chose is sufficient/good, ask your tutor before the end of the lecture period.
- 10 Points Use your calculated matrix to find the errors of the optimal parameters. Compare your algorithm to the implementation you replaced. Compare the results (Hesse errors) and the performance of your algorithm with the `hesse()` call that was used before. If your algorithm performs significantly worse, what could you improve?

4. **Likelihood Profile**: In this task you calculate profiles/contours for the likelihood.
Passing criteria for the code: You should be able to find the profile-likelihood for all combinations of θ_i and θ_j (you do not need to include all in the code but all combinations should work) for $-2\log\mathcal{L}_G$. At least the 39 % interval profile needs to work and deliver a mostly smooth contour plot (you can compare with the `iMinuit` function `draw_mncontour`). It does not need to be optimized but should run in a reasonable time.

   - 10 Points Come up with an algorithm, that gets two of the parameters and some confidence level (e.g. 68 %, 39 % or 1σ) as input and calculates the likelihood-profile at this interval for the given parameters. The contours should behave like the contours you get from `mncontour` or `draw_mncontour` in the `iMinuit` package. Explain your algorithm, i.e. what are the conditions for finding the correct contour, how do you find/calculate the points of the contour, ... Also think about how to optmize your algorithm.
   - 10 Points (for documentation of the implementation) Implement your algorithm. Replace the code inside the `likelihood_profile` function in the fitting framework. You implementation should have the same input parameters as this function and have the same behavior as the `mncontour()` call in this function. For finding the optimal parameters you can use the already implemented `minimize` function in the fitting framwork. For doing additional minimizations in your algorithm you are allowed to use pre-built functions like the `migrad()` function in `iMinuit` or something similar.
   - 10 Points Implement the plotting of your contour (what `draw_likelihood_profile` does in the fitting framework). Use the points you get from the code in your implementation in `likelihood_profile` to draw the contour. `likelihood_profile` should give you enough points, so you get a mostly smooth contour similar to `draw_mncontour` in `iMinuit`.
   - 10 Points Compare your resulting contour to the contour drawn and calcaulted by the `iMinuit` function `draw_mncontour`/`mncontour` (e.g. for the two parameters of interest).

Also compare to `draw_error_ellipsis` in the fitting framework, which uses `draw_contour` from `iMinuit`. Compare the performance of your algorithm with the performance of the `iMinuit` functions. If your algorithm performs significantly worse, what could you improve?

5. **Markow Chain Monte Carlo**: Instead of using a minimzation algorithm and different algorithms to determine the unertainties, you could also do a Markow Chain Monte Carlo (MCMC) simulation to sample the likelihood and extract the optimal parameters and their uncertainties directly from the simulated sample.
   Passing criteria for the code: The code should be able to successfully sample $-2\log\mathcal{L}_G$ and be able to extract the optimal parameters and a profile (or mean, mode, quantiles and densities for the bayesian interpretation) for two given parameters. You should be able to specify the sample size, the number of draws for the burn-in phase of the MCMC and whether to discard or keep the draws from the burn-in phase. You do not need to implement auto-tuning (it is also not necessary for a "perfect grade" but if you are motivated definitely a nice to have).

   - 10 Points The goal is to implement the Metropolis-Hastings algorithm. Explain the idea of MCMC and this algorithm and how it works.
   - 10 Points (for documentation of the implementation) Implement this algorithm to sample the negative log-likelihood (replace the code in the `mcmc()` function in the fitting framework). **Note:** You should implement this algorithm yourself. It is not allowed to use pre-built functions in some `Python` library that does the MCMC for you. There should be an implementation of some working MCMC algorithm in your submitted code. Except for that, you are, of course, allowed to use the usual `numpy`, ... functionalities.
   - Now, you can choose, whether you want to do a frequentist or bayesian interpretation (only do one of them):
     Frequentist:
     - 10 Points (for documentation of the implementation) Use your generated sample to extract the minimum of the negative log-likelihood and the related optimal parameters for the likelihood. Also extract the likelihood-profile (similar to what `mncontour` does in `iMinuit`) for two parameters (e.g. the 1σ contour for the two parameters of interest). For this task, you do not need to minimize/resample/... anything. You can do everything with your generated sample. For finding the minimum you can just calculate the likelihood for your sampled points and extract the minimal calculated value and the associated parameters. For the profile you do a 2D projection and find points that fit the criterion for the profile contour (+ some interval around that, since you sampled a finite number of discrete points. This leaves you with bins with multiple points). Then you find the minimum including the other parameters in these "bins".
     - 10 Points Plot your calculated profile and compare your result (optimal parameters and profile) to the fitting framework (you can compare your profile plot with the plot you get from `draw_profile_likelihood` in the fitting framework, which uses `draw_mncontour` from `iMinuit`).

     Bayesian:
     - 10 Points Create density plots of the sampled parameters and comment on their shape, etc.
     - 10 Points (Documentation of implementation, mentioned explanations/description and comparison) Calculate the mean, mode and the quantiles of the parameters. There is a `pymc`

example in the provided code, which you can use as comparison, but you should implement these functionalities yourself for your algorithm. If you choose the bayesian interpretation, provide an explicit, exact description of the values you extracted and their meaning. Compare them to the results of the fitting framework (this will be especially important later for the more complicated likelihood).

Hint for MCMC: Depending on the number of sample points and number of chains this can require quite some computing resources, so you might want to start small and then adapt all parameters until your sample size is sufficiently large.

6. **Getting the likelihood:** The tasks 1-5 considered actual fitting procedures. One important task in doing data analyses with maximum likelihood fits is getting the likelihood. In this task, you will use some toy simulations and data to extract the likelihood and perform a fit with this likelihood. Since this task needs some more explanations, we moved the Description for task 6 to the end of this document, to keep these instructions clear.

Testing a more complicated Likelihood [10 Points]

This task is only for people who chose one of the tasks 1 to 5. People working on task 6 do not need to do this task (and have some other additional tasks for these points in their task description). The likelihood $\mathcal{L}_G$ is not very complicated. Now, we want to test your implementation with a more complicated likelihood function, a Rosenbrock function. There are still six parameters for the fit but now the negative log-likelihood is given as

$$-2\log\mathcal{L}_R(\vec{\theta}) = \sum_{i=1}^{5}(100\cdot(\theta_{i+1}-\theta_i^2)^2+(1-\theta_i)^2)\,.$$

The optimal parameters and covariance matarix are not μ and V anymore (this was just for the gaussian likelihood). Run your implemented algorithm from the previous task with this new negative log-likelihood. Answer the following questions (only for your chosen task). If your code does not work for the more complicated likelihood, discuss the reasons this may have and answer the questions where possible.

1. **Gradient Descent**:
   - If your minimization converged, visualize the steps of your minimization again (for different initial guesses). This function has more than one minimum, did you find the global minimum?
   - If your minimization does not converge: Stop the minimization after a reasonable amount of steps and show those. What could be problems, why your minimization doesn't converge?
   - Also run the fit with the given fitting framework with the `iMinuit` implementations and compare the results and the performance with your algorithm.
2. **BFGS**:
   - If your minimization converged, visualize the steps of your minimization again (for different initial guesses). This function has more than one minimum, so did you find the global minimum?
   - If your minimization does not converge: Stop the minimization after a reasonable amount of steps and show those. What could be problems, why your minimization doesn't converge?

- Also run the fit with the given fitting framework with the `iMinuit` implementations and compare the results and the performance with your algorithm.

3. **Hesse matrix**:

   - Again, visualize your algorithm and comment on potential differences to the gaussian likelihood.
   - Also run the fit with the given fitting framework with the `iMinuit` implementations and compare the results and the performance with your algorithm.

4. **Likelihood Profile**:

   - Also run the fit with the given fitting framework with the `iMinuit` implementations and compare the results and the performance with your algorithm.
   - Compare your likelihood-profile to the error ellipsis you get with `draw_error_ellipsis` which uses the `iMinuit` function `draw_contour`. What do you notice and what is the reason for this? Also try higher contours than 1σ, e.g. 1.5σ (you can try event higher contours but without some help `iMinuit` might also fail at some point).

5. **Markow Chain Monte Carlo**:

   - Also run the fit with the given fitting framework with the `iMinuit` implementations and compare the results (optimal parameters and contour of the two parameters of interest) and the performance with your algorithm. Again, if you did the bayesian interpretaion, explicitly describe what the parameters and results mean, that you extracted.

Conclusion and format of the report [5+5 Points]

(For all tasks again) Summarize what you did in this project, how it worked and what might need improvement. There will be 5 points for the overall format of your report. It should be written in a clear and well-structured manner, plots should be clear and readble and contain all necessary information,...

Description for task 6

Passing criteria for the code: You can draw (and plot) the varying histograms for four systematic uncertainties and the signal and background Monte Carlo sample. You also implemented the negative log-likelihood using these histograms and performed a fit with the provided data and the derived negative log-likelihood and visualized the fitted parameters in at least one of the mentioned ways in task 6.4.

In this task you will extract the negative log-likelihood from some models and systematic uncertainties. If you want to see a study using this technique, you can have a look at [1]. However, for this task you do not need to understand this paper (or particle physics).

With the files for this final project you also find a data file containing "measured" data points (of some random variable m). We will consider the following model that should describe the measured data: We have a background and a signal process, which are random variables drawn from

- a gaussian with $\mu = 91.2, \sigma = 15$ for the background
- and a gaussian with $\mu = 125, \sigma = 17$ for the signal.

Both distributions are truncated at $m_{\text{min}} = 60$ and $m_{\text{max}} = 160$. Thus, your Monte Carlo samples with the parameters above should only contain Monte Carlo data points between m_{min} and m_{max}. Also, the data only contains data points in this range. The data has been measured in 20 equidistant bins between m_{min} and m_{max}. Use this binning for all histograms, ... in your task.

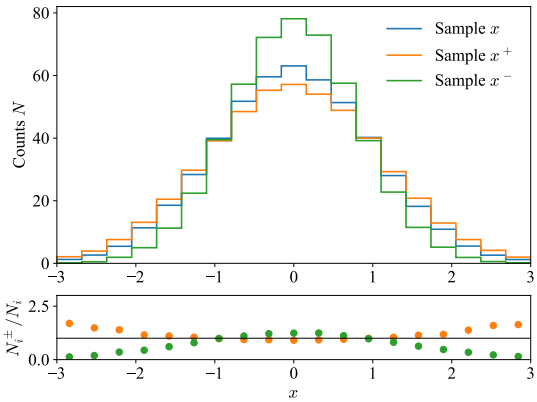
Task 6.1: Creating Monte Carlo Samples 10 Points (Histograms and Documentation)

Create a Monte Carlo signal and a Monte Carlo background sample for the distributions described above. You can use all built-in `numpy` tools etc. Create a histogram of your background sample, your signal sample and the combined sample. Draw high statistics samples, to avoid dealing with statistical uncertainties on these Monte Carlo samples (i.e. these uncertainties can be ignored later) and then introduce weights to scale your histogram so that you get 500 entries in your background and 30 entries in your signal histogram. We expect that these numbers describe our data.

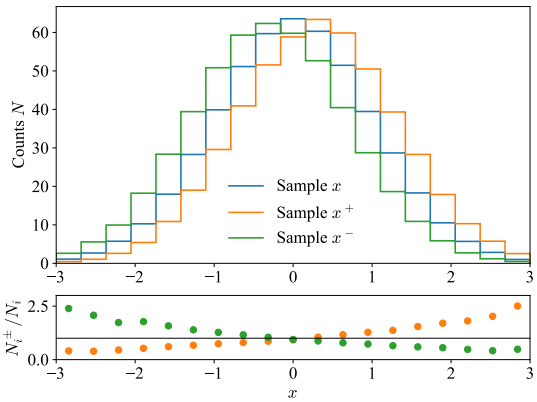
Task 6.2: Histograms for systematic uncertainties 10 Points (Histograms and Documentation)

Now, we want to introduce four systematic uncertainties. To do this, you create variations of the background and the signal sample and histograms. An example for such a systematic uncertainty could be the resolution of a detector, which would have an effect on the width of the distributions. To introduce this as a systematic uncertainty you create a histogram (from a varied sample) that is wider and one which is more narrow. More general, for each systematic uncertainty you create a histogram which is varied in one direction of this effect (e.g. more narrow) and one, which is varied in the opposite direction of this effect (in the example: wider)[1]. These histograms will later be used in the likelihood to include this systematic uncertainty in the fit. You find examples of such varied histograms in Fig. 1 for a unit gaussian. Create four such histograms for different systematic variations, with variations in both directions, for the signal and the background sample, respectively. Also include the ratio part (lower plot in each histogram) in your histograms when you visualize them and include the histograms in your report.

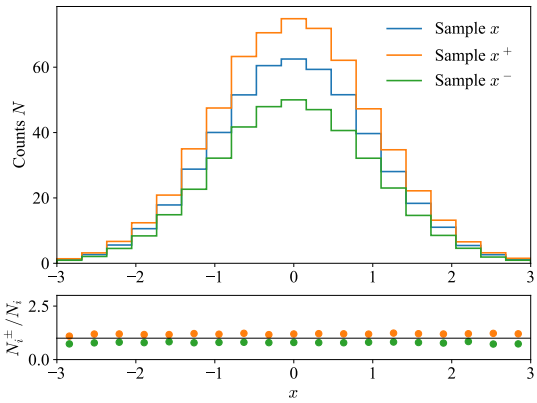
[1] In a real world study, the size of these effects would be measured in separate analyses. In our example just make a pick for your variation (e.g. 5 %) and treat it as if this had been determined in a separate measurement.



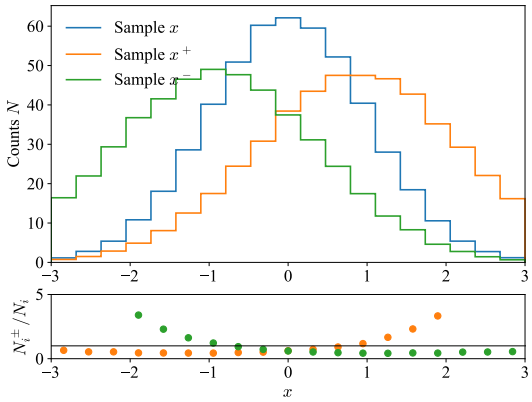
(a) More narrow/wider (e.g. detector resolution)



(b) Shift left/right (e.g. momentum scale)



(c) Scaled up/down (e.g. detector efficiency)



(d) Asymetric scaling/shift (e.g. differences in detection areas (trigger effects))

Figure 1: Example variation histograms for systematic uncertainties for a unit gaussian.

Task 6.3: The likelihood function and fit 20 Points (Likelihood, Fit and Documentation)

Now you have everything to put together the likelihood. The likelihood consists of multiple parts. First, you create a product of poisson distributions $P(d_i; V_i(\vec{\theta}))$ for each bin i. Here, d_i is the number of data points in bin i and $\vec{\theta}$ are the parameters to be fitted. $V_i(\vec{\theta})$ can be constructed the following way:

$$V_i(\vec{\theta}) = \theta_1 \cdot b_i^n + \theta_2 \cdot s_i^n + \sum_{j=3}^{6} |\theta_j| \left((b_i^{j\pm} - b_i^n) + (s_i^{j\pm} - s_i^n) \right)$$

where b_i are the entries in the generated background samples and s_i in the signal samples in bin i. Here, n denotes the nominal histograms, i.e. from your original Monte Carlo sample. The b^j and s^j are the histograms from the variations with

$$b_i^{j\pm} = \begin{cases} b^{j+} & \theta_j > 0 \\ b^{j-} & \theta_j \leq 0 \end{cases}$$

and analogously for s^j. Here, $+$ denotes the variation in one direction (e.g. the wider histogram) and $-$ in the other direction (more narrow in this example).

After constructing this part of the likelihood, multiply gaussians $G(\theta_j; \mu = 0, \sigma = 1)$ for each parameter of the systematic uncertainties, to constrain the parameters of the systematic uncertainties.

Give and implement the complete likelihood $\mathcal{L}$ (or $-2 \log \mathcal{L}$) as constructed above. Then, using `iMinuit` or a similar tool (you can use everything in the provided fitting framework or something else), perform a maximum likelihood fit to find the optimal parameters.

Task 6.4: Optimal parameters and uncertainties 10 Points (Documentation and Visualization)

Do the following visualizations for the optimal parameters and their uncertainties. *For passing the coding part you need to implement at least one of them, for all the points in this task you need to provide both.*

- Plot the correct, stacked signal + background Monte Carlo histogram and the data points. Include error bars on the data points (counting experiment) and a systematic error band on the Monte Carlo histogram. It should look something like one of the panels in Fig. 9 in [1] (you don't have to understand the histogram there, it is just to show you how it could look like with the different histograms, data points and uncertainties). The systematic error bands can be found by error propagation of $V_i(\vec{\theta})$ but without the terms for θ_1 and θ_2. Use the uncertainties and the covariance matrix found by the fit for $\theta_3, \ldots, \theta_6$ for the error propagation.
- Create a plot similar to Fig. 5 in [1] but instead of the measured values that are shown there, you show the fitted parameters and their uncertainties. So, on the y axis you have the 6 different parameters and show the fitted values and uncertainties the fit finds in x direction.

Also, comment on your fit result. Are the fitted values including their uncertainties what you expect?

References

[1] ATLAS Collaboration, "Differential cross-section measurements of higgs boson production in the $H \to \tau^+\tau^-$ decay channel in pp collisions at $\sqrt{s} = 13\,\text{TeV}$ with the atlas detector," *Journal of High Energy Physics*, vol. 2025, no. 3, Mar. 2025, ISSN: 1029-8479. arXiv: 2407.16320.