

Exercises: download and install

- The exercises are done using the ROOT6 framework
- If you plan to work on the exercises, make sure to have ROOT6 installed
- The exercises require some files to be downloaded:
- Download the zip file
- Create a new directory
- Unzip the files in the new directory

http://www.desy.de/~sschmitt/GRK2044/tutorialUnfolding_V2.zip

Exercises: list of files

- There is one root file with histograms
- There is a library with functions which do not have to be modified (but can be interesting to look at)
- There are macros to get the exercises started
- Each macro will produce some plots
- The macros have to be modified and expanded during the exercises

tutorialIntroduction.pdf : brief documentation

tutorial_inputHisto.root ; histograms
tutorialLibrary.h tutorialLibrary.C : library

tutorialPlotInput.C : show the input data

tutorialOwnUnfoldingExample.C : exercise 1-6

tutorialOwnIteration.C : exercise 7,8

tutorialTikhonovExample.C : exercise 9,10

tutorialScanLCurve.C : exercise 11

tutorialScanSURE.C : exercise 12-14

tutorialFit.C : exercise 15

Exercises: the tutorialLibrary

Definitions are in tutorialLibrary.h

Classes:

TutorialInput : loads all required input histogram into memory for unfolding

TutorialResult: holds the result of an unfolding algorithm

TutorialUnfoldingAlgorithm : base class to run an unfolding algorithm

TutorialUnfoldTikhonov : Tikhonov unfolding

TutorialIterativeUnfolding: generic iterative unfolding (can select step function)

Auxillary classes:

TutorialUnfoldEMstep : step function for EM iterative unfolding

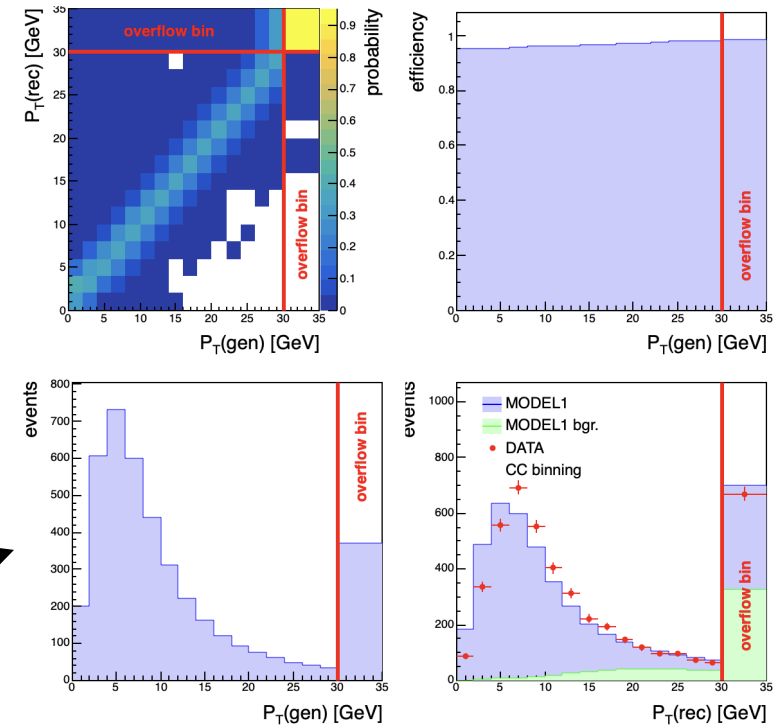
Namespaces:

TutorialPlotter : default plotting functions

Exercises: the input histograms

- The histograms include various types of distributions
- The class TutorialInput loads the required distributions such that they can be accessed by the unfolding algorithms and for plots
- TutorialPlotter::showInputPlots() can be used to visualize a set of input distributions

Example macro: in root type these commands:
.L tutorialLibrary.C+
.L tutorialPlotInput.C
tutorialPlotInput(1)



Exercises: choosing input variants

- When constructing TutorialInput, select the model, the input data for the unfolding, and the binning

```
class TutorialInput : public TNamed {
public:
    enum MODEL {
        MODEL1=1,
        MODEL2=2
    };
    enum INPUT {
        INPUT_DATA=0,
        INPUT_MODEL1 = MODEL::MODEL1,
        INPUT_MODEL2 = MODEL::MODEL2,
    };
    enum BINNING {
        COARSE,
        RECO_FINE,
        BOTH_FINE
    };
    TutorialInput(MODEL model, INPUT input, BINNING binning,
        char const *name="tutorial_inputHisto.root");
```

There are two models

There are data (without truth information)

The models also can be used as input

There are three bin sizes to choose from::

COARSE (16 bins truth, 16 bins reco)

RECO_FINE (16 bins truth, 31 bins reco)

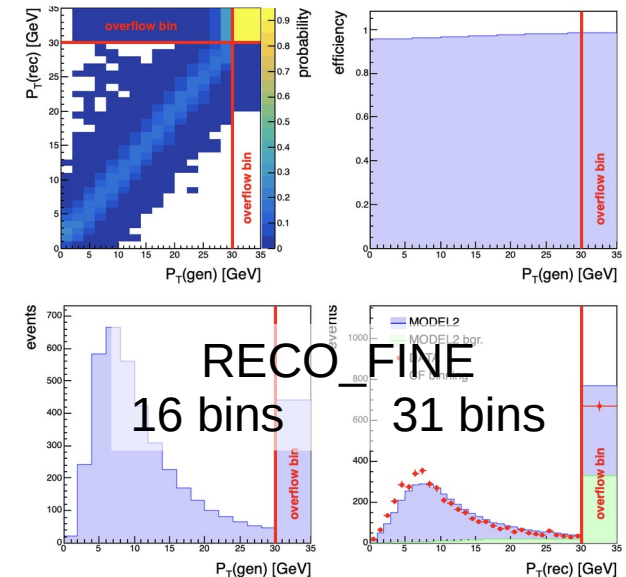
BOTH_FINE (31 bins truth, 31 bins fine)

Example: tutorialPlotInput(2)

...

case 2:

```
input=new TutorialInput(TutorialInput::MODEL2,
    TutorialInput::INPUT_DATA,
    TutorialInput::RECO_FINE);
```



The tutorialPlotter utilities

```
namespace TutorialPlotter {  
    // show input data  
    void showInputPlots(TutorialInput const &input);  
  
    // show unfolding result in truth and reco space  
    // optionally show correlation coefficients  
    void compareResultModelTruth(TutorialUnfoldingResult const *result,  
                                  bool showCorrelations);  
  
    // draw scan of regularisation parameter  
    // returns location of best scan parameter  
    int drawLCurve(vector<TutorialUnfoldingResult *> const &scan);  
    int drawSURE(vector<TutorialUnfoldingResult *> const &scan, bool useLogX);  
  
    // compare two unfolding results against each other  
    // comparison in truth space and in data space  
    void compareTwoResultsSameData(TutorialUnfoldingResult const *result1,  
                                    TutorialUnfoldingResult const *result2);  
}
```

These utilities can be used to obtain sets of plots from the classes in the library.

Exercises 1-6

- Exercise 1: run `tutorialOwnUnfoldingExample.C` (matrix inversion), discuss the result
 - Exercise 2: implement the bin-by-bin method, discuss the result
 - Exercise 3: repeat (1-3) using MODEL2 for the unfolding, what changes?
 - Exercise 4: plot the prediction error, compare bin-by-bin and matrix inv.
Hint: use `TUnfoldingResult::getYhat()`, define a new histogram or graph
- extra exercises, only if there is time left:
- Exercise 5*: implement the minimum χ^2 use binning RECO_FINE and unfold the data.
Compare to Exercise 1.
 - Exercise 6*: repeat the likelihood fit, with MODEL2 and 1000 toy samples. Plot the observed prediction error, compare to χ^2 distribution (how many degrees of freedom?)

Exercises 7-10

- Exercise 7: try `tutorialOwnIteration.C` Look at the unfolding result for various choices of the number of iterations (0..100). Plot the observed prediction error as a function of $n(\text{iter})$
- Exercise 8: modify `tutorialOwnIteration.C` use the bin-by-bin method as step function. Repeat exercise 7 (with max. iterations=20)
- Exercise 9: try `tutorialTikhonovExample.C` Look at the unfolding result for various choices of s . Plot the observed prediction error as a function of s
- Exercise 10: modify `tutorialTikhonovExample.C` to plot the vector z and the eigenvalues D . Why are the dimension of z and D different? What could be a good choice of s , such that insignificant modes z_i are suppressed?

Hint: use the methods `TutorialUnfoldTikhonov::getEigenValues()` and `TutorialUnfoldTikhonov::getZ()`

Exercise 11-15

- Exercise 11: try out the L-curve scan tutorialScanLCurve.C. Best s ?
- Exercise 12: try out the SURE minimisation tutorialScanSURE.C. Best s ?
- Exercise 13: modify tutorialScanSURE.C to apply the SURE scan to the EM iterative method. What is the best number of iterations?
- Exercise 14: modify tutorialScanSURE.C to apply the SURE scan to the bin-by-bin iterative method. What is the best number of iterations?
- Exercise 15: try/modify the macro tutorialFit.C to fit the respective “best” unfolding result with a function, taking into account the covariance matrix. Compare the results to the “data” truth: peak=6 GeV, width=1.8 GeV

Solutions

- Solutions are available

<http://www.desy.de/~sschmitt/GRK2044/tutorialSolutions.zip>

- These could be useful for exercises building on code developed in other exercises