CFIT: Template fits including systematic correlations

Kirill Skovpen *

Institut Pluridisciplinaire Hubert Curien, CNRS/IN2P3, Strasbourg, France

May 22, 2016

Abstract

CFIT is a tool to perform a template fit with including systematic uncertainties in a correlation matrix. Systematic correlations between templates are taken into account. The tool also allows one to estimate statistical and systematic uncertainties on the fit parameters and to perform the measurement of the scale factors.

*Electronic address: kirill.skovpen@cern.ch

Contents

1 Introduction

Analysis of systematic uncertainties in high energy physics experiments has always been a place for discussions on realism and robustness of a chosen approach. A wide variety of systematic sources ranging from an assessment based on the first principles to a completely artificial estimation implies a complexity of statistical description of such uncertainties. Statistical treatment of asymmetric statistical and systematic uncertainties in physics analyses is always mentioned as one of the most obscure and challenging tasks [3, 2].

This paper contains the documentation of the tool which was developed to perform the Monte Carlo template fit to data by including systematic uncertainties in the fit via a correlation matrix. Such methods are widely used in various physics analyses. This paper does not introduce any novelty in statistical methods used whilst it attempts to have a specific implementation for a chosen approach, as well as to adapt several ideas outlined in other papers referenced here by testing it further in a dedicated tool.

2 Install

Here are a few steps to build the shared library for CFIT:

```
git clone https://github.com/kskovpen/CFIT
cd CFIT/
make
```

3 Fit procedure

Minimisation is performed in MINUIT [1] using the following χ^2 function:

$$\chi^2 = \frac{(x_i^d - \sum\limits_{k=1}^{N_t} P_k x_i^k) V_{ij}^{-1} (x_j^d - \sum\limits_{k=1}^{N_t} P_k x_j^k)}{\sqrt{(\sigma_i^d)^2 + (\sum\limits_{k=1}^{N_t} \sigma_i^k)^2}\sqrt{(\sigma_j^d)^2 + (\sum\limits_{k=1}^{N_t} \sigma_j^k)^2}}, \quad (1)$$

where x_i^d is the number of data events in i-bin, x_i^k is the number of events for k-template in i-bin, σ_i^d is the statistical uncertainty in data in i-bin, σ_i^k — total uncertainty for k-template in i-bin, N_t is the number of templates. V_{ij} is the correlation matrix element which is constructed as follows:

$$V_{ij} = \frac{\sum_{s=1}^{N_{sys}} C_s \sigma_i^s \sigma_j^s + \delta_{ij}\sigma_i^{MCstat}\sigma_j^{MCstat} + \delta_{ij}\sigma_i^d\sigma_j^d}{\sqrt{(\sigma_i^d)^2 + (\sigma_i^{MC})^2}\sqrt{(\sigma_j^d)^2 + (\sigma_j^{MC})^2}}, \tag{2}$$

where N_{sys} is the number of systematic variations, C_s is the bin-by-bin correlation factor for s-systematics, σ_i^{MCstat} is the total statistical uncertainty in i-bin for the sum of templates, and σ_i^k is the total systematic uncertainty for k-systematic variation in i-bin:

$$\sigma_i^s = \sqrt{\sum_{k=1}^{N_t} (\sigma_i^k)^2}, \tag{3}$$

and σ_i^{MC} is the total uncertainty in i-bin:

$$\sigma_i^{MC} = \sqrt{\sum_{s=1}^{N_{sys}} (\sigma_i^s)^2 + (\sigma_i^{MCstat})^2}. \tag{4}$$

The bin-by-bin correlation factor for s-systematics is defined as:

$$C = \frac{2}{N_b(N_b-1)} \sum_{i=1}^{N_b-1} \sum_{j=i+1}^{N_b} \frac{|\sigma_i^+\sigma_j^+ + \sigma_i^-\sigma_j^-|}{\sqrt{(\sigma_i^+)^2 + (\sigma_i^-)^2}\sqrt{(\sigma_j^+)^2 + (\sigma_j^-)^2}}, \tag{5}$$

where N_b is the number of bins in the template, $(\sigma^+)_i^s$ and $(\sigma^-)_i^s$ are shape systematic uncertainties which correspond to the positive and negative one sigma shifts for s-systematics. For one-sided systematic uncertainty $C = 1$ by definition.

As the χ^2 function is defined only for symmetric uncertainties, systematic variations in the fitted templates are symmetrized by using the maximum systematic deviation from the nominal template as the assigned bin systematic uncertainty:

$$\sigma_i^s = S \cdot \max\Big[|\sigma^+|_i^s, |\sigma^-|_i^s\Big], \tag{6}$$

where S is the sign of the up-one-sigma variation. An alternative approach of approximation of asymmetric systematic uncertainties by dimidated Gaussian function is also implemented in this tool [2].

Systematic uncertainties on the templates are recalculated after the first minimization cycle to provide a more realistic uncertainty estimation in the χ^2 calculation procedure.

To reduce the effect of correlations in the tails of the distributions of the templates where such effects might be not well evaluated, bins yielding up to 2% of the total statistics in the tail are summed up in one bin.

4 Estimation of the fit uncertainty

In order to evaluate uncertainties on the fit parameters, the nominal templates are replaced by statistic (`SetStatVariation`) or systematic (`SetSysVariation`) variations in the shapes. The fit is performed with all uncertainties included in the fit. The use of correlation matrix computed with the nominal templates in the measurement of systematic and statistical uncertainties for the fit parameters has shown the best stability in terms of a convergence of the fit.

5 Scale factor measurement

CFIT allows one to perform the measurement of selection criteria efficiencies and the scale factors from the template fit with the assessment of associated systematic and statistical uncertainties. For this type of measurement a user has to specify histograms for all events (`AddTemplate`,`SetData`), events which passed the «tag» requirement (`AddTemplateTag`,`SetDataTag`), and events with has failed this requirement (`AddTemplateUntag`,`SetDataUntag`). Systematic and statistical correlations between the template fits for different selections are taken into account. The number of bins in templates has to be the same for a proper correlation treatment between the different types of selection. The full example is given in Sec. 7.2.

6 User method description

This section lists the functions which could be used in CFIT.

`cfit(std::string name = "")` — CFIT constructor, `name` corresponds to the user defined name of the fit variable.

`void Run(std::string option = "")` — The main function to run the fit procedure. If option "tag" is specified, the fit is performed with the post-tag templates.

`void SetVerbose(int v)` — Set verbose level. Two options are possible: no printout (0) and debug printout (1).

`void SetMorphing(OPTMORPHMODE optMorphMode=OPTMORPH_GEOMETRIC,float frac = 1.)` — Set the morphing factor (f) for systematics correlations, `OPTMORPH_CUTOFF`: $f = 0$ for bins with $b_i/b_{max} < frac$, `OPTMORPH_GEOMETRIC`: $f = \sqrt{b_i b_j}/b_{max}$, `OPTMORPH_ARITHMETIC`: $f = (b_i + b_j)/(2b_{max})$, `OPTMORPH_FUNC`: $f = b_i b_j / b_{max}^2$, where b_i is the value for i-bin, b_{max} is the maximum bin value in the histogram.

`int GetMorphing()` — Get the morphing option for systematics correlations.

`void SetOptimization(OPTMODE optMode=OPT_NONE)` — Set optimization option for systematics treatment. `OPT_NONE` — no optimization is used, `OPT_MORPH` — perform the morphing of systematic correlations according to `OPTMORPHMODE`, `OPT_SGN` — ignore correlations for bins where systematics flip sign or up and down variations are of the same sign, `OPT_SIGMA` — reset relative bin systematic uncertainties to $\sigma = \sqrt{\sum_i \sigma_i^2 b_i)/\sum_i b_i}$ for bins with $\sigma_i/\sigma < 0.1$, where b_i and σ_i are the value and the relative uncertainty for i-bin. Combination of several optimization options is also possible: `OPT_MORPH_SGN`, `OPT_MORPH_SIGMA`, `OPT_SGN_SIGMA`, `PT_MORPH_SGN_SIGMA`.

`int GetOptimization()` — Get optimization option for systematics treatment.

`void SetCovarianceMode(COVMODE covMode=COV_MAX)` — Method selection to compute the variance. `COV_MAX`: symmetrize bin systematic uncertainty by choosing the maximum deviation, `COV_BARLOW`: asymmetric systematic uncertainty approximation by dimidated Gaussian [2].

`int GetCovarianceMode()` — Get the method option for covariance computation.

`void ProducePlots(bool produce = 1)` — Produce plots with the results of the fit (`result.eps`), correlation matrix (`result.eps`), shape comparison for the templates (`shape.eps`), shapes for fractional systematic uncertainties before (`sys.eps`) and after (`sysOptimised.eps`) optimization.

`int GetNDOF()` — Get the number of degrees of freedom (N_{dof}).

`double GetChisq()` — Return the minimized χ^2/N_{dof} value from the fit.

`int GetNPar()` — Get the number of fitted templates.

`double GetPar(int i)` — Get the fit parameter for the i-template.

`double GetParErr(int i)` — Get the error of the fit parameter for the i-template.

`void SetInputFile(std::string fin)` — Set the name of the input file with histograms.

`void SetData(std::string name)` — Set the name of the data histogram.

`void SetDataTag(std::string name)` — Set the name of the data histogram for the post-tag fit.

`void SetDataUntag(std::string name)` — Set the name of the data histogram for the un-tag selection.

`void AddTemplate(std::string name,std::string nominalName,int colour)` — Add the title and the histogram name for the nominal template.

`void AddTemplateTag(std::string name,std::string tagName,int colour)` — Add the title and the histogram name for the nominal template for the post-tag fit.

`void AddTemplateUntag(std::string name,std::string untagName,int colour)` — Add the title and the histogram name for the nominal template for the un-tag selection.

`void GlueTemplates(std::vector<std::string> nameV,std::string nameMerged = "merged",int colour = 46)` — Vary templates with the names specified in `nameV` simultaneously in the fit. These templates are effectively merged into one template.

`void GlueTemplatesTag(std::vector<std::string> nameV,std::string nameMerged = "merged",int colour = 46)` — The same as `GlueTemplates` but for the post-tag templates. The number of merged templates could be different from what is used in the pre-tag case.

`void SetMatrixOption(std::string option)` — Set the write/read option for correlation matrix. `"WRITE"`: write the result matrix to file, `"READ"`:

read the matrix from file.

`void SetMatrixName(std::string name = "matrix")` — Set the output name for the ROOT file with correlation matrix.

`void SetLegendHeader(std::string name = "")` — Set header for the legend of histograms.

`void AddSys(std::string name,std::string upName,std::string downName)` — Add templates which correspond to systematic variations. `name`: systematics name defined by user; `upName`: histogram name ending for the one sigma up variation, the full histogram name is nominalName+upName; `downName`: histogram name ending for the one sigma down variation, the full histogram name is nominalName+downName.

`void SetSysVariation(std::string name)` — Replace nominal template histogram by a systematic variation, `name` has to be either `upName` or `downName`.

`std::string GetSysVariation()` — Get systematic variation name.

`void SetStatVariation(int rnd)` — Vary nominal template histogram within the statistical uncertainty, `rnd` is a seed which has to be greater or equal than 666.

`int GetStatVariation()` — Get statistical variation name.

`float GetNData()` — Get total sum of events in the input data histogram.

`float GetNTemplate(std::string templateName)` — Get total sum of weighted events for a given template.

`float GetErrTemplate(std::string templateName)` — Get total error for the sum of weighted events for a given template.

7 Examples

Several practical examples are located in `examples/` directory and are briefly described here.

7.1 Single fit

The example below shows how to perform a fit to data with three templates. All templates have two systematic variations, namely, `SYS1` and `SYS2`. The example is available in `examples/test.C` which reads histograms from the file test.root located in the same directory. The code could be run with the command `root run.C`. Plots with results from the fit are shown in Fig. 1-4.

```
#include "../cfit.h"

void getResults(CFIT::cfit *cf,float *par,float *err);

void test()
{
   gROOT->SetBatch();

   gSystem->Load("../libCFIT.so");

   float par[100];
   float err[100];

   CFIT::cfit *cf = new CFIT::cfit("Fit variable");
   cf->SetOptimization(OPT_MORPH_SGN_SIGMA);
   cf->SetMorphing(OPTMORPH_GEOMETRIC);
   cf->ProducePlots(1);

   cf->SetInputFile("test.root");

   cf->AddSys("SYS1","_sys1_down","_sys1_up");
   cf->AddSys("SYS2","_sys2_down","_sys2_up");

   cf->SetMatrixOption("WRITE");

   cf->SetData("h_data");

   cf->AddTemplate("template1","h_mc1",2);
   cf->AddTemplate("template2","h_mc2",3);
   cf->AddTemplate("template3","h_mc3",4);
```

```
   cf->Run();

   getResults(cf,par,err);
   float chi2 = cf->GetChisq();

   std::cout << "chi2=" << chi2 << std::endl;
   for(int i=0;i<3;i++)
     {
       std::cout << par[i] << " +- " << err[i] << std::endl;
     }

   delete cf;

   gApplication->Terminate();
}

void getResults(CFIT::cfit *cf,float *par,float *err)
{
   float chi2 = cf->GetChisq();
   int nPar = cf->GetNPar();
   for(int i=0;i<nPar;i++)
     {
        par[i] = cf->GetPar(i);
        err[i] = cf->GetParErr(i);
     }
}
```

The fit parameters along with its uncertainties are printed in the output:

```
chi2=0.881485
26.7739 +- 1.58475
48.8739 +- 1.73018
41.3997 +- 1.3972
```

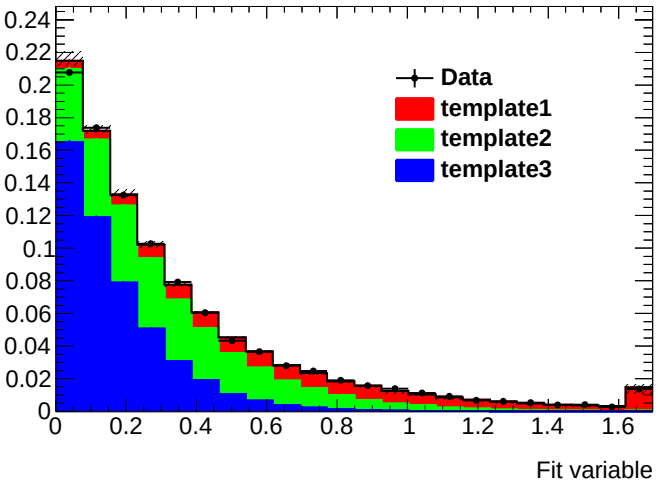

Figure 1: Result of the fit.

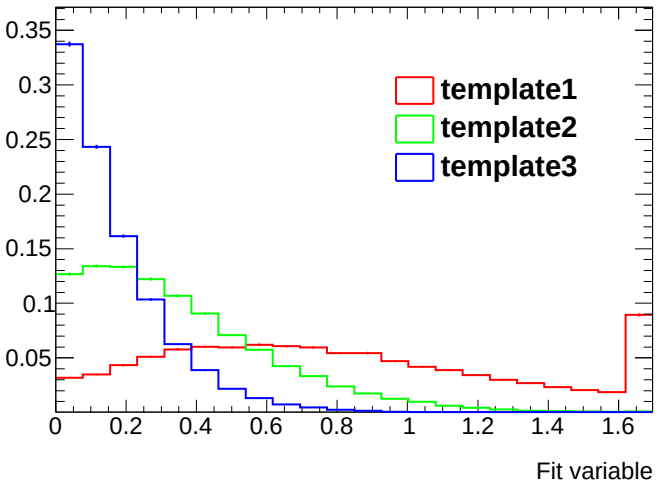

Figure 2: Shape comparison of the templates.

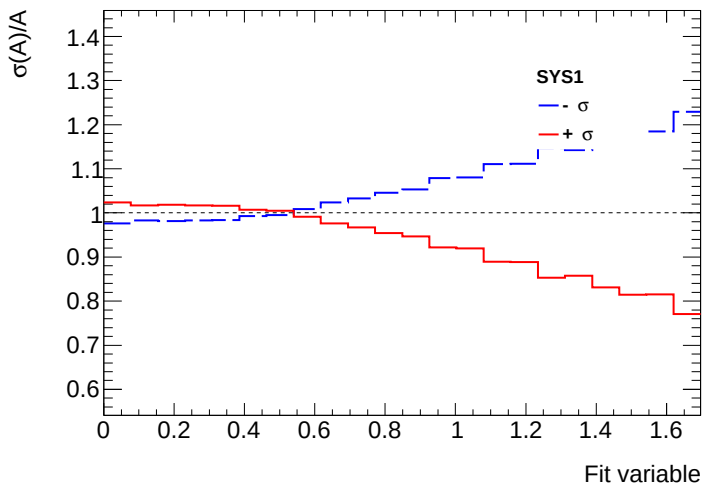

Figure 3: Fractional systematic uncertainties after optimization procedure.

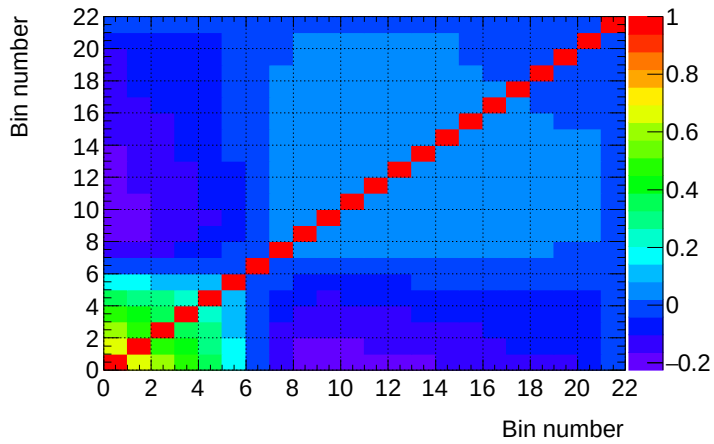

Figure 4: Correlation matrix.

7.2 Scale factor measurement

This example illustrates how to perform the scale factor measurement. The example is available in `examples/testSF.C` and could be run with the command `root runSF.C`.

```
#include "../cfit.h"

void getResults(CFIT::cfit *cf,float *par,float *err);

void testSF()
{
   gROOT->SetBatch();

   gSystem->Load("../libCFIT.so");

   float par[100];
   float err[100];
   float par_tag[100];
   float err_tag[100];

   CFIT::cfit *cf = new CFIT::cfit("Fit variable");
   cf->SetOptimization(OPT_MORPH_SGN_SIGMA);
   cf->SetMorphing(OPTMORPH_GEOMETRIC);
   cf->ProducePlots(1);

   cf->SetInputFile("test.root");

   cf->AddSys("SYS1","_sys1_down","_sys1_up");
   cf->AddSys("SYS2","_sys2_down","_sys2_up");

   cf->SetMatrixOption("WRITE");

   cf->SetData("h_data");
   cf->SetDataTag("h_data_tag");
   cf->SetDataUntag("h_data_untag");

   cf->AddTemplate("template1","h_mc1",2);
   cf->AddTemplate("template2","h_mc2",3);
```

```
cf->AddTemplate("template3","h_mc3",4);

cf->AddTemplateTag("template1","h_mc1_tag",2);
cf->AddTemplateTag("template2","h_mc2_tag",3);
cf->AddTemplateTag("template3","h_mc3_tag",4);

cf->AddTemplateUntag("template1","h_mc1_untag",2);
cf->AddTemplateUntag("template2","h_mc2_untag",3);
cf->AddTemplateUntag("template3","h_mc3_untag",4);

cf->Run();
getResults(cf,par,err);
float chi2 = cf->GetChisq();
float ndata = cf->GetNData();
float nmc1 = cf->GetNTemplate("template1");
float nmc = cf->GetNTemplate("template1")+
cf->GetNTemplate("template2")+
cf->GetNTemplate("template3");

cf->Run("tag");
getResults(cf,par_tag,err_tag);
float chi2_tag = cf->GetChisq();
float ndata_tag = cf->GetNData();
float nmc1_tag = cf->GetNTemplate("template1");
float nmc_tag = cf->GetNTemplate("template1")+
cf->GetNTemplate("template2")+
cf->GetNTemplate("template3");

float fr = nmc1/nmc;
float fr_tag = nmc1_tag/nmc_tag;

float effMC = nmc1_tag/nmc1;
float effDATA = nmc_tag/nmc*par_tag[0]/par[0]*fr_tag/fr;

std::cout << "effMC = " << effMC << std::endl;
std::cout << "effDATA = " << effDATA << std::endl;
std::cout << "sf = " << effDATA/effMC << std::endl;
```

```
   // perform statistical variation
   cf->SetMatrixOption("READ");
   cf->SetStatVariation(667);
   cf->ProducePlots(0);
   cf->Run();
   getResults(cf,par,err);
   cf->SetStatVariation(667);
   cf->Run("tag");
   getResults(cf,par_tag,err_tag);

   // do the calculation of SF here again ....

   // perform systematic variation
   cf->SetMatrixOption("READ");
   cf->SetSysVariation("_sys1_up");
   cf->Run();
   getResults(cf,par,err);
   cf->SetSysVariation("_sys1_up");
   cf->Run("tag");
   getResults(cf,par_tag,err_tag);

   // do the calculation of SF here again ....

   delete cf;

   gApplication->Terminate();
}

void getResults(CFIT::cfit *cf,float *par,float *err)
{
   float chi2 = cf->GetChisq();
   int nPar = cf->GetNPar();
   for(int i=0;i<nPar;i++)
     {
        par[i] = cf->GetPar(i);
        err[i] = cf->GetParErr(i);
     }
}
```

References

[1] F. James and M. Roos, Minuit: A System for Function Minimization and Analysis of the Parameter Errors and Correlations, Comput. Phys. Commun. 10 (1975) 343.

[2] Roger Barlow, Asymmetric Systematic Errors, `arXiv:physics/0306138`, 2003.

[3] Roger Barlow, Asymmetric Statistical Errors, `arXiv:physics/0406120`, 2004.