Correcting ${}_s\mathcal{P}lots$ in the presence of fixed yields

E. Ben Haim, L. Henry

March 16, 2016

Contents

1 Introduction **1**

2 ${}_s\mathcal{P}lots$ with fixed yields **1**

3 `RooStats` implementation of the ${}_s\mathcal{P}lots$ method **3**

4 Proposed method and test **4**

5 Conclusion **9**

1 Introduction

Subtracting background from the signal in physics analyses can be performed in several ways. The ${}_s\mathcal{P}lots$ procedure presented in Ref. [1] is an efficient and widespread method to do so; it is implemented in the `RooStats` package [2].

In this note, we focus on the effect of fixed yields on the results, which is documented in Ref. [1], Annexes B.1 and B.2. In Sec. 2, we shortly remind the ${}_s\mathcal{P}lots$ subtraction procedure and its modifications in the presence of fixed yields. We then review in Sec. 3 the implementation of ${}_s\mathcal{P}lots$ in `RooStats` and the issues it may introduce. Finally in Sec. 4, we propose an alternative constructor to the `RooStats::SPlot` class that would solve these issues, along with some tests.

2 ${}_s\mathcal{P}lots$ with fixed yields

We consider a model with N_S event species; the yield of a species k is noted N_k and its normalised PDF f_k. The ${}_s\mathcal{P}lots$ procedure allows to use the information from a fit performed on a discriminating variable X to extract the distributions of the control variable Y for the different species.

A main ingredient of the ${}_s\mathcal{P}lots$ calculation is the covariance matrix V of the fit, which can be taken from the output of a fit routine (e.g. `TMinuit` [3]). Alternatively, its inverse can be directly computed using:

$$V_{ij}^{-1} = \sum_{e=1}^{N} \frac{f_i(e) f_j(e)}{(\sum_{k=1}^{N_S} N_k f_k(e))^2}, \tag{1}$$

where the sum is running over N events, and $f_i(e)$ designates the value of the PDF f_i for the event e. We can then use the covariance matrix to compute, for each event species n, the per-event sWeight $sP_n(e)$, using:

$$sP_n(e) = \frac{\sum_{j=1}^{N_S} V_{nj} f_j(X_e)}{\sum_{k=1}^{N_S} N_k f_k(X_e)}. \tag{2}$$

The distribution of the event species n on the control variable Y is then estimated by the $sM_n(Y)$ distribution, defined by:

$$N_n . sM_n(Y) . \delta Y = \sum_{e \subset [Y-\delta Y, Y+\delta Y]} sP_n(e). \tag{3}$$

We now introduce another event species in the model, with a fixed yield N_0 and a normalised PDF f_0. The covariance matrix changes, and its inverse becomes:

$$V_{ij}^{-1} = \sum_{e=1}^{N} \frac{f_i(e) f_j(e)}{(\sum_{k=1}^{N_S} N_k f_k(e) + N_0 f_0(e))^2}, \tag{4}$$

whereas the per-event sWeight becomes:

$$sP_n(e) = \frac{\sum_{j=1}^{N_S} V_{nj} f_j(X_e)}{\sum_{k=1}^{N_S} N_k f_k(X_e) + N_0 f_0(X_e)} = \frac{\sum_{j=1}^{N_S} V_{nj} f_j(X_e)}{\sum_{k=1}^{N_S} N_k f_k(X_e)} \frac{\sum_{k=1}^{N_S} N_k f_k(X_e)}{\sum_{k=1}^{N_S} N_k f_k(X_e) + N_0 f_0(X_e)}. \tag{5}$$

This expression differs from Eq. 2 only by an event-by-event factor that depends on the yields N_k and the PDFs f_k.

In the case where the distribution of the control variable for the species with fixed yield, $M_0(Y)$, is known, the distribution of the control variable Y for the species n is:

$$N_n.sM_n(Y).\delta Y = \sum_{e \subset [Y-\delta Y, Y+\delta Y]} sP_n(e) + c_n.M_0(Y), \tag{6}$$

where

$$c_n = N_n - \sum_{j=1}^{N_S} V_{nj} \tag{7}$$

is a coefficient depending uniquely on the considered event species. It quantifies the impact of the species with fixed yields on $sM_n(Y)$, and vanishes only if $N_0 = 0$.

One of the issues of extracting sWeights using a tool that treats the varied and fixed yields on an equal footing is related to the calculation of the covariance matrix. To illustrate this, we consider a model including three species: a signal S, a combinatorial background C, and another background B with a fixed yield. Using Eq. 4, we calculate the inverse of the covariance matrix, and obtain:

$$V^{-1} = \begin{pmatrix} (V^{-1})_{SS} & (V^{-1})_{SC} & (V^{-1})_{SB} \\ (V^{-1})_{CS} & (V^{-1})_{CC} & (V^{-1})_{CB} \\ (V^{-1})_{BS} & (V^{-1})_{BC} & (V^{-1})_{BB} \end{pmatrix}, \tag{8}$$

which incorrectly includes terms related to the species with fixed yields. As the terms $(V^{-1})_{SB}$ and $(V^{-1})_{CB}$ do not vanish *a priori*, the inverse of this matrix has no clear link with the correct covariance matrix. Also, we notice that in the case where $f_B = f_C$ or $f_B = f_S$, the matrix is no longer invertible (it has two identical columns and lines), whereas sWeights should still be calculable. This shows that there is something fundamentally flawed with this approach.

3 `RooStats` implementation of the ${}_s\mathcal{P}lots$ method

The `RooStats::SPlot` method, used to calculate sWeights, computes the inverse of the covariance matrix using Eq. 1 with a list of yields that the user provides as an argument. The covariance matrix itsef is then obtained from its inverse. As shown in Sec. 2, in the case where there are some fixed yields in the arguments, this results in an incorrect covariance matrix.

However, building the `RooStats::SPlot` object using only the varied yields is also incorrect, as it would result in using Eq. 1 and Eq. 2 rather than Eq. 4 and Eq. 5. Correcting the sWeights event-by-event using Eq. 5 is not possible either, as the covariance matrix is not correctly calculated.

4 Proposed method and test

It is clear from Sec. 2 that it is necessary to differenciate the fixed yields from the others in the `RooStats::SPlot` object. To address this requirement, we propose an alternative constructor to the `RooStats::SPlot` object, shown in Fig. 1 along with the original constructor[1]. We stress that this alternative constructor does not perform the correction shown in Eq. 6. However, it calculates and stores the c_n coefficients in a new attribute of the class, as shown in Fig. 2 and Fig. 3.

We test this alternative constructor and the c_n extraction tool using a toy model containing 3 event species: a signal S, a combinatorial background C, and a peaking background B with a fixed yield. The PDFs of these species on the discriminating variable X and on the control variable Y are taken as:

- Signal (S): Gaussian ($\mu = 0$, $\sigma = 0.1$) for X; Gaussian ($\mu = 0$, $\sigma = 0.05$) for Y.
- Combinatorial background (C): Constant for X; Gaussian ($\mu =$ -0.5, $\sigma = 0.05$) for Y.
- Fixed background. (B): Gaussian ($\mu = 0$, $\sigma = 0.05$) for X; Gaussian ($\mu = 0.5$, $\sigma = 0.05$) for Y.

Both variables X and Y are defined in the interval [-1,1]. This model is chosen in order to ensure a sizable nuisance of the species with fixed yield on the discriminating variable, whereas its impact on the control variable is easy to spot. We generate a sample of 16000 events, including 5000 signal events, 1000 fixed background events, and 10000 combinatorial background events. We then assume a wrong hypothesis on the yield of the fixed background ($N_B = 1200$), in order to simulate the general case where the value of the fixed yield is not precisely known. We show the result of the one-dimensional fit on the variable X using this model, along with the projection of this fit on the variable Y, in Fig. 4.

We consider two approaches, A and B: the former is the calculation of the sWeights with the original `RooStats::SPlot` method, providing only the list of varying yields to the `RooStats::SPlot` constructor; the latter is the proposed approach, where we use the alternative `RooStats::SPlot` constructor providing the list of all yields and the list of fixed yields.

[1] Providing an empty RooArgSet as the fixedYields argument of the alternative constructor gives the same results as calling the original constructor with the same arguments

Original constructor:

```
SPlot::SPlot(const char* name, const char* title,
             RooDataSet& data, RooAbsPdf* pdf,
             const RooArgList &yieldsList, const RooArgSet &projDeps,
             bool includeWeights, bool cloneData, const char* newName):
  TNamed(name, title)
```

Implementation of the alternative constructor:

```
SPlot::SPlot(const char* name, const char* title,
             RooDataSet& data, RooAbsPdf* pdf,
             const RooArgList &allYieldsList,const RooArgList &fixedYields,
             const RooArgSet &projDeps,
             bool includeWeights, bool cloneData, const char* newName):
  TNamed(name, title)
{
  /* Skipped: Original body of the SPlot constructor. */
  // Add check that yieldsList contains only varying yields
  // Check that fixed yields are in the allYields arguments
  iter = fixedYields.createIterator();
  while((arg=(RooAbsArg*)iter->Next()))
    if (!(allYieldsList.contains(*arg)))
      {
        // Throw exception and error message.
      }
  // Call new method to build sWeights, with fixed yields
  this->AddSWeight(pdf,allYieldsList,fixedYields,projDeps,includeWeights);
}
```

Figure 1: Snippets of code showing the original (top) and the alternative (bottom) constructors for the `RooStats::SPlot` class.

```
void SPlot::AddSWeight( RooAbsPdf* pdf, const RooArgList &allYieldsList,
                        const RooArgList &fixedYields,
                        const RooArgSet &projDeps, bool includeWeights)
{
  /* Skipped: store the constant parameters (other than yields) values */
  // Store indexes of the allYieldsList corresponding to variables yields
  TIterator *it = allYieldsList.createIterator();
  RooAbsArg* arg;
  unsigned int iArg(0);
  std::vector<unsigned int> varIndexes;
  while ((arg = (RooAbsArg*) it->Next()) != NULL){
      if (!(fixedYields.find(arg->GetName())))
        varIndexes.push_back(iArg);
      iArg++;}
  // We now have two indexes over which we iterate
  Int_t nAllSpec = allYieldsList.getSize();
  Int_t nVarSpec = allYieldsList.getSize() - fixedYields.getSize();

  /* Skipped:
    -store the initial yield parameters
    -calculate the value of the component pdf for each event and
     species. */
  // Inverse of the covariance matrix
  TMatrixD covInv(nVarSpec, nVarSpec);
  /* Skipped: Initialisation to 0 of each covInv(i,j) */
  for (Int_t ievt = 0; ievt < numevents; ++ievt){
      fSData->get(ievt) ;
      // Sum for the denominator
      Double_t dsum(0);
      for(Int_t k = 0; k < nAllSpec; ++k)
        dsum += pdfvalues[ievt][k] * yieldvalues[k] ;

      for(Int_t n=0; n<nVarSpec; ++n)
        for(Int_t j=0; j<nVarSpec; ++j)
            if(includeWeights == kTRUE)
              covInv(n,j) += fSData->weight()
                *pdfvalues[ievt][varIndexes[n]]
                *pdfvalues[ievt][varIndexes[j]]/(dsum*dsum) ;
            else
              covInv(n,j) +=
                pdfvalues[ievt][varIndexes[n]]
                *pdfvalues[ievt][varIndexes[j]]/(dsum*dsum) ;}
  // Invert to get the covariance matrix
  TMatrixD covMatrix(TMatrixD::kInverted,covInv);
```

Figure 2: Proposed addition to the `RooStats::SPlot` class to compute sWeights in the presence of fixed yields (first part).

```
  // Create and label the variables used to store the SWeights
  for(Int_t k=0; k<nVarSpec; ++k){
    /* Skipped: usual variables definitions, including sWeights
     variables */

    // c_n coefficients
    wname = std::string(yieldvars[varIndexes[k]]->GetName())+"_c";
    var = new RooRealVar(wname.c_str(),wname.c_str(),0);
    double cVal = yieldvalues[varIndexes[k]];
    for (Int_t n = 0 ; n<nVarSpec; ++n)
      cVal -= covMatrix[k][n];
    var->setVal(cVal);
    fSWeightCoefs.add(*var); // new attribute of the class.
  }
  // Create and fill a RooDataSet with the SWeights
  RooDataSet* sWeightData = new RooDataSet("dataset", "", sweightset);
  for(Int_t ievt = 0; ievt < numevents; ++ievt){
      fSData->get(ievt) ;
      // sum for denominator
      Double_t dsum(0);
      for(Int_t k = 0; k < nAllSpec; ++k)
        dsum +=  pdfvalues[ievt][k] * yieldvalues[k] ;
      // covariance weighted pdf for each species
      for(Int_t n=0; n<nVarSpec; ++n){
        Double_t nsum(0) ;
        for(Int_t j=0; j<nVarSpec; ++j)
          nsum += covMatrix(n,j) * pdfvalues[ievt][varIndexes[j]] ;
        if(includeWeights == kTRUE)
          sweightvec[n]->setVal(fSData->weight() * nsum/dsum) ;
        else
          sweightvec[n]->setVal( nsum/dsum) ;
        /* Skipped: fill the dataset with sweightvec */
      }}
  // Add the SWeights to the original dataset
  fSData->merge(sWeightData);
  /* Skipped: reinitialise all parameters and yields */
  return;
}
```

Figure 3: Proposed addition to the `RooStats::SPlot` class to compute sWeights in the presence of fixed yields (second part).

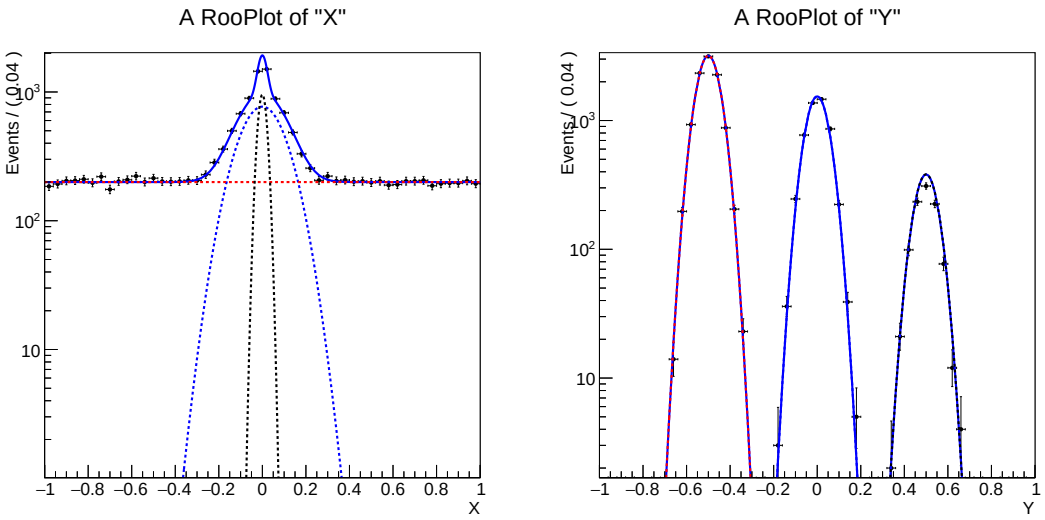

Figure 4: Result of the fit on X, projected on the X dimension (left) and the Y dimension (right). The signal is displayed in blue, the combinatorial background in red, and the peaking background in black. As expected, the projection of the result on the Y dimension shows that we overestimated (on purpose) the number of background events.

For each of these two methods, we show in Fig. 5 the samples with signal and background weights applied, both before and after the correction of Eq. 6. The results are satisfactory for approach B after the c_n correction, for both the signal and the combinatorial background distributions. On the contrary, approach A provides an acceptable description of the signal shape, but shows large discrepancies for the combinatorial background.

For each event species, we also show in Table. 1 the sum of sWeights and of its associated c_n, compared to the fitted yield. According to Ref. [1], the sum has to be compatiable with the yield, which is clearly not the case for approach A.

Species	$\sum_e sP_A(e)+c_A$	$\sum_e sP_B(e)+c_B$	Fitted yield
Signal	-7336.15	4975.27	4975.35
Comb.	16361.5	10007.5	10007.5
Bkg	-	-	1200.

Table 1: Summary of yields and sum of sWeights for the two approaches. As shown in Ref. [1], in both approaches the sum of all yields extracted from the fit is not equal to the total number of events.

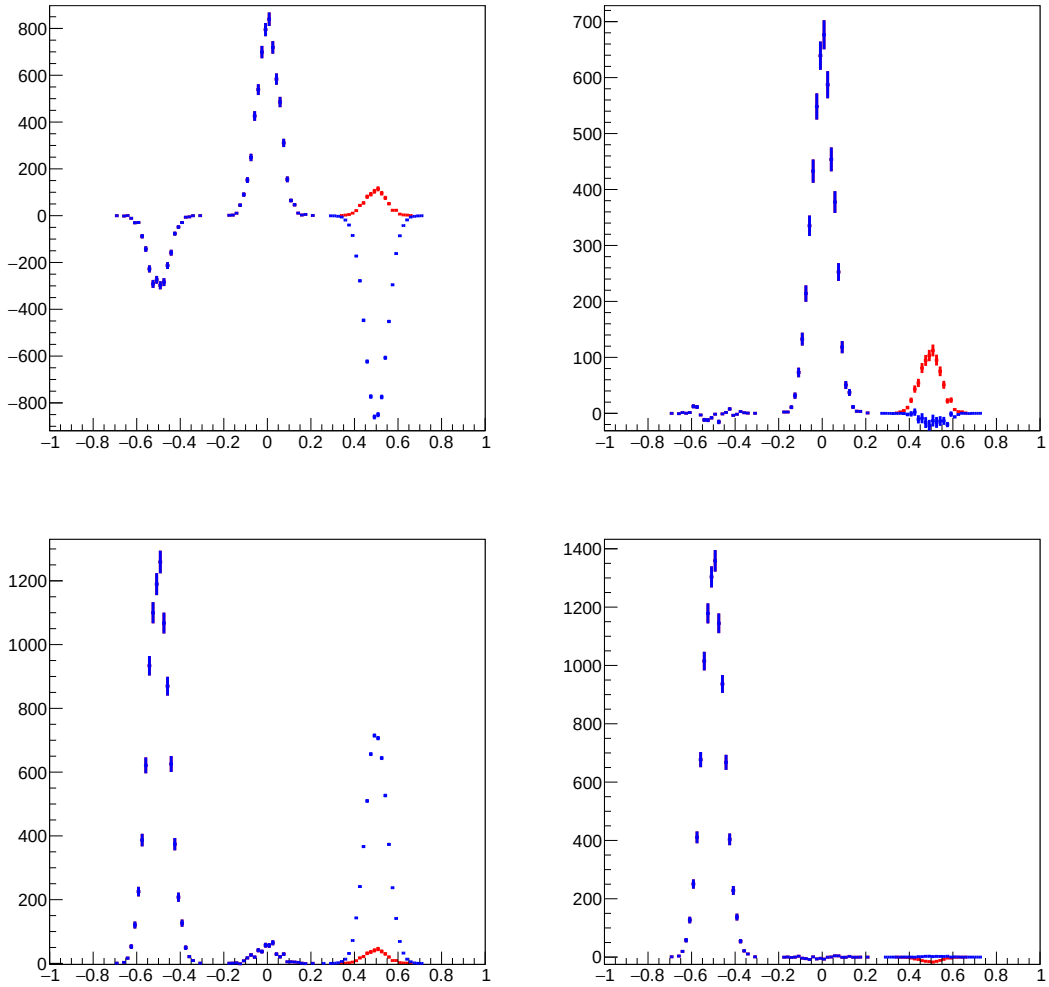

Figure 5: Distribution of Y in data samples sWeighted according to signal (combinatorial) weights are shown on top (bottom). Results of the approach A(B) are shown on the left (right). The red points correspond to the distribution of the sum of sWeights, whereas the blue points represent the final distribution (after correction using Eq. 6).

5 Conclusion

Models that include an event species with fixed yields require a specific treatment when using the ${}_s\mathcal{P}lots$ method, that the `RooStats::SPlot` implementation does not provide. In this note, we propose a straightforward modification of this class that allows to extract the correct ${}_s\mathcal{P}lots$ in the case of fixed yields. These modifications do not remove the need to correct the distributions using Eq. 6, but allow to calculate the c_n coefficients inside the `RooStats::SPlot` object in a way that is coherent with the sWeights extraction.

We tested this additional code, both in terms of compatibility with the former implementation (not shown here), and in terms of expected results. The results are satisfactory, and show a clear improvement compared to the original approach, especially in terms of normalisation properties.

References

[1] M. Pivk and F. R. Le Diberder, *sPlot: a statistical tool to unfold data distributions*, .

[2] *RooStats package webpage*, `https://root.cern.ch/root/html/ROOFIT_ROOSTATS_Index.html`.

[3] *TMinuit webpage*, `https://root.cern.ch/doc/v606/classTMinuit.html`.