Experience with the B2DXFitters package in $B_s^0 \to D_s^{\pm} K^{\mp}$ analysis

Giulia Tellarini, Stefania Vecchi

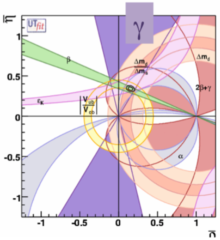
University of Ferrara & INFN

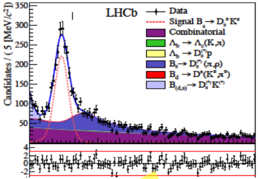
B2DXfitters workshop, 9^{th}-10^{th} July Padova

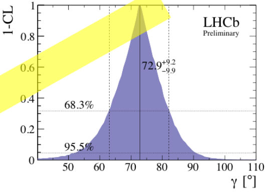
10^{th} July 2015

Measuring γ in Bs-> DsK

- **γ is the least accurate UT-angle of CKM**.
  - **LHCb:** First measurement in time-dependent CP asymmetry measurement of in Bs-> DsK decays
    - Multidimensional Fitter (low BR, lot of B-background)
    - Fast Bs oscillations
    - FlavourTagging
    - Using 2011 ($1fb^{-1}$@7TeV) data **: $\gamma=(115\ ^{+28}_{-43})°$**
    - **Pubblished on J. High Energy Phys.11 2014 060**;
  - Analysis of the full sample **($3fb^{-1}$@ 7 + 8 TeV)**
    - The core of the analysis is the same of previous analysis
    - Already started:
    - **Expected larger contribution to the γ measurement**







Introduction

Start by saying:

- $B_s^0 \to D_s^{\pm} K^{\mp}$ analysis is fully performed using the B2DXFitters package → the developers are totally involved in the analysis itself :-)
- When I started to face the B2DXFitters package I was as "the man in the street" from the point of view of analysis experience
- I use the package "a spizzichi e bocconi" → "dribs and drabs"

What I used of the B2DXFitters package

- *For the Flavour Tagging Calibration in $B_s^0 \to D_s^- \pi^+$*
  1. *MDFitter to the $B_s^0 \to D_s^- \pi^+$ data (Reco12) → "run" python script*
  2. *Time sFit to the $B_s^0 \to D_s^- \pi^+$ data (Reco12) → "run" python script*
  3. *a selection of B2DXfitter classes (Reco12 & Reco14) → my own fitters*
- *Toy study on the effects of an increased statistics for the 3fb^{-1} $B_s^0 \to D_s^{\pm} K^{\mp}$ analysis*
  1. *Toys generation → "run" python script*
  2. *MDFitters + sFit for each toy → "run" python script*
- *Studies on Correlations among observables for the 3fb^{-1} $B_s^0 \to D_s^{\pm} K^{\mp}$ analysis*

- My own difficulties and suggestions will be pointed out in the presentation

B2DXfitters used for Flavour Tagging Calibration Reco12

`Urania v2r2`

- $B_s^0 \to D_s^{\pm} K^{\mp}$ 1fb^{-1} needed the Reco12 Flavour Tagging Calibration
- at that time the SSKnnet appears on the scene
- we did the calibration of OS and SSK taggers in $B_s^0 \to D_s^- \pi^+$ using the same machinery (MDFitter + sFit/cFit) used for the $B_s^0 \to D_s^{\pm} K^{\mp}$ analysis

The package was already able to fit $B_s^0 \to D_s^- \pi^+$ data and to provide the flavour tagging calibration. The only developments were to implement also the fit in category of mistag (Manuel did a great job in this direction for the cFit) and to include the new taggers variables (SSKnnet and OSKnnet η and q_{tag}). The difficulties were:

- to know what has to be written as command line to get a given result/output
- to understand what each code really needs as input and what it provides as output (workspace, ntuples, log files)
- to manage workspaces, to extract variables distribution [...]

→ these difficulties have been overcome thanks to the great help and support of Manuel and Agnieszka

Suggestions for B2DXFitters developers

Suggestion from the "man in the street":

- to write somewhere an example of command line → this makes easier to use in the right way the options list you usually report at the end of the python scripts
- print out before running the actual configuration and also the other possible configurations
  ⇒ Agnieszka's yesterday tutorial helps a lot in this direction

B2DXfitters used for Flavour Tagging Calibration Reco12 + Reco14

Urania v2r2

In parallel we developed our own fitters to $B_s^0 \to D_s^- \pi^+$ data. It serves as a check to Reco12 calibration but also to calibrate Reco14 taggers. The fitter follows the $B_s^0 \to D_s^\pm K^\mp$ analysis sFit approach, with some mitigations:

- No MDFitter → just $m_{B_s^0}$ fit
- No per-event time error → average time error modeled by a Triple Gaussian
- Analytical model for time acceptance

The fitter uses B2DXfitter classes:

- MistagCalibration → it calibrates the mistag event by event within the time fit
- DecRateCoeff → it evaluates the "experimental" coefficient of trigonometric functions in the B decay rate
- PowLawAcceptance → it defines the acceptance function
- RooBinnedPdf → given a PDF, it creates a binned PDF
- RooEffResModel → it produces an effective resolution model

In the next slides I'll enter deeper in the use of each class

B2DXfitters package: MistagCalibration

- MistagCalibration class defines automatically within the time fit the relationship between the predicted mistag η and the measured one (Eq.1)
- it takes as input the η value for each event and the $< \eta >$
- it enters in the definition of the decay rate coefficients in the case we want to perform the event by event calibration
- running the fit it evaluates the tagging parameters p_0 and p_1

$$\omega = p_0 + p_1 \cdot (\eta - < \eta >) \tag{1}$$

```
MistagCalibration eta_c("eta_c","calibrated mistag ev by ev", eta, p0,p1,eta_mean);
```

where eta, eta_mean, p0 and p1 are RooRealVar

B2DXfitters package: DecRateCoeff

- DecRateCoeff is a powerful class that together with the RooBDecay allows to build easily the decay time PDF for B decays
- it evaluates the experimental coefficients: dilution due to imperfect tagging, production asymmetry, detection asymmetry and tagging efficiency asymmetry
- it has different constructors for guaranteeing the right flexibility

```
#ifdef EVbyEV // for event by event tagging calibration
  RooArgSet observ0(t,rec,tag,eta);
  DecRateCoeff cosh("cosh", "cosh",DecRateCoeff::CPEven, rec, tag, one, one, eta,pdf_eta_binned,effTag, eta_c,zero, zero,zero)
  DecRateCoeff cos ("cos",  "cos", DecRateCoeff::CPOdd,  rec, tag, one, one, eta,pdf_eta_binned,effTag, eta_c,zero, zero,zero)
#else         // for fit in mistag category calibration
  RooArgSet observ0(t,rec,tag);
  RooRealVar mistag("mistag","Average mistag",0.38,0.,0.6);//binnando eta uso un altro costruttore dei coeff.
  DecRateCoeff cosh("cosh", "cosh",DecRateCoeff::CPEven, rec, tag, one, one,effTag, mistag,zero, zero,zero);
  DecRateCoeff cos ("cos",  "cos", DecRateCoeff::CPOdd,  rec, tag, one, one,effTag, mistag,zero, zero,zero);
#endif
```

- in the first case η is a variable of the fit $\rightarrow$ PDF(η) is required
- in the second case the average value of the mistag is a free parameter that can be eventually split in categories $\rightarrow$ an average value of mistag is provided by the fit

B2DXfitters package: PowLawAcceptance and RooBinnedPdf

- PowLawAcceptance describes an analytical function which can be used as acceptance function

$$a(t) = \begin{cases} 0 & : \; when \; (at)^n - b < 0 \; or \; t < 0.2ps \\ (1 - \frac{1}{1+(at)^n - b}) \cdot (1 - \beta t) & : \; otherwise \end{cases} \quad (2)$$

```
//Acceptance parameters determined on Reco14 Bs2DsPi 2011+2012 data
RooRealVar  tacc_turnon("tacc_turnon","time acceptance turn on",1.1236);//,0.7,1.7 );// GIULIA:1.1236);
RooRealVar  tacc_beta("tacc_beta","time acceptance beta",0.03926);//,0.003,0.3);// GIULIA: 0.03926);
RooRealVar  tacc_expo("tacc_expo","time acceptance expo",1.7905);//,1.0,2.6);// GIULIA: 1.7905);
RooRealVar  tacc_offset("tacc_offset","time acceptance offset", 0.0373);//,0.0003,0.3);// GIULIA: 0.0373);

PowLawAcceptance time_acc("time_acc", "time_acc",tacc_turnon, t, tacc_offset, tacc_expo, tacc_beta);
```

- RooBinnedPdf allows make a binned Pdf starting from an unbinned function
- We make a binned acceptance function to get a faster time fit
- A time binning should be defined at first (RooUniformBinning)

```
RooUniformBinning time_bin(t.getMin(), t.getMax(), 100, "acceptanceBinning");
t.setBinning(time_bin,"acceptanceBinning");

RooAbsReal& timeasabsreal = t;
RooBinnedPdf bin_time_acc("bin_time_acc","bin_time_acc", timeasabsreal, "acceptanceBinning", time_acc);
bin_time_acc.setForceUnitIntegral(kTRUE);
```

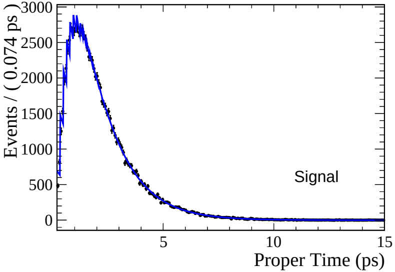
B2DXfitters package: RooEffResModel

- RooEffResModel allows to combine two functions producing an "effective resolution model"
- I combine together the time resolution model (sum of three gaussians: RooAddModel) with the time acceptance function (RooBinnedPdf)

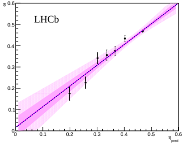
```
RooEffResModel t_resol_accbin("time_res_acc","time_res_acc",t_resol, bin_time_acc);

//----------- Time Probability Density Function --------------------//
RooBDecay pdf_sig_t_Bs_fit("pdf_sig_t_Bs_fit","Signal Pdf(t,tag,rec) of B0_s",t,tau_Bs,Dgamma_Bs,cosh, zero,cos, zero\
, Dm_Bs,t_resol_accbin,RooBDecay::SingleSided);
```

- So finally using the RooBDecay class enriched by all the ingredients seen till now we get a very appreciable description of B meson decay!



Time sFit
2011+ 2012 $B_s^0 \to D_s^- \pi^+$ data
for SSKnnet calibration



B2DXfitters used for $B_s^0 \to D_s^{\pm} K^{\mp}$ Toy Studies

Urania v2r3

For the $B_s^0 \to D_s^{\pm} K^{\mp}$ 3fb^{-1} analysis we had to generate 1000 $B_s^0 \to D_s^{\pm} K^{\mp}$ toys in which the statistics is increased by a factor 3 in order to check if any bias arises in the CP observables. The fitter is set to the nominal values used for the 1fb^{-1} analysis.

- The machinery was already done
- We only have to change the statistics for generating the events and also the fixed yields for fitters and pulls
- We produce them in batch mode (*bsub* in 2nd queue) in bunch of 25 toys

The difficulties arise from the fact that:

- the configuration options are repeated in different configuration files → no crosscheck about consistency
- it was hard to understand how the different backgrounds are grouped and named
- we met some problems to open the workspaces (we had not the right libraries), it has been solved in this way:

```
root -l
gSystem->Load("libCintex");
Cintex::Enable();
gSystem->Load("/afs/cern.ch/user/g/gtellari/cmtuser/Urania_v2r3/PhysFit/B2DXFitters/x86_64-slc6-gcc48-opt/libB2DXFittersLib");
gSystem->Load("/afs/cern.ch/user/g/gtellari/cmtuser/Urania_v2r3/PhysFit/B2DXFitters/x86_64-slc6-gcc48-opt/libB2DXFittersDict");
```

Suggestions for B2DXFitters developers

Suggestion from a "man in the street":

- To write somewhere an example of command line → this makes more easier to use in the right way the options list you usually report at the end of the script
- To produce a unique configuration file for generator, fitters ecc
- To define somewhere how the different backgrounds (with floating yields and fixed yields) are grouped and be consistent in each part of the analysis with such grouping → maybe in the twiki page/package documentation is the right place
- Pulls from log files?!?
- A reminder rather than a suggestion: to include the taggers (OS,SSK) combination in the fit script ⇒ Manuel is working on it and this is a security

B2DXfitters used for Correlation Studies

`Urania v2r3`

For the $B_s^0 \to D_s^{\pm} K^{\mp}$ 3fb^{-1} analysis a deeper investigation on the correlations among observables is desired.

1. observables scatter plots are produced for signal and backgrounds using the truth MC (2011) information
2. python prepareBsDsKMassFitterOnData3D5M.py –debug –MC –configName Bs2DsKConfigForNominalMassFitBDTGA
3. a fast simulation has been developed based on the root class TGenPhaseSpace to emulate the $B_s^0 \to D_s^{\pm} K^{\mp}$ and its background events
   - TGenPhaseSpace allows to generate in phase space n-body decays
   - We have developed an emulator for the $B_s^0 \to D_s^{\pm} K^{\mp}$ analysis using two B2DXFitter classes: QRDecomposition and DecayTreeTupleSucksFitter
   - Both are used to perform a kinematic fit to the B_s^0 decay with the D_s mass constraint

```
DecayTreeTupleSucksFitter fitter(1.96849, 0.494, 0.494, 0.1395 );
```

Once the D_s daughters candidates are reconstructed a kinematic fit is performed:

```
bool FLAG = fitter.fit(pM,pD1,pD2,pD3);
if(FLAG==false) continue;
```

Summary

- The B2DXFitters package is a very powerful ensemble of classes and scripts

Good Job !!!

- Many items are already well covered
- From my experience it's easier (not faster) and instructive to use a class instead of using a wrapped run (python) script
- Easier from the point of view of knowledge and awareness about what things do or could do