B2DXFitter: application to $B_s \rightarrow D_s^* K$

Speaker: Lorenzo Sestini

$B_s \rightarrow D_s^* K$ team: A. Bertolin, A. Dziurda, S. Gallorini, A. Lupato, M. Rotondo, L. Sestini

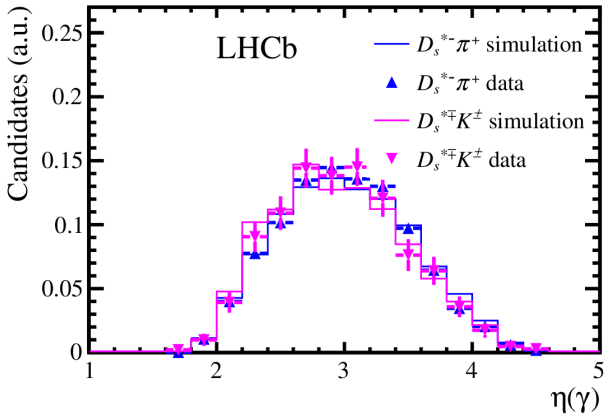
LHCb B2OC time-dependent workshop, Padova 10-07-2015

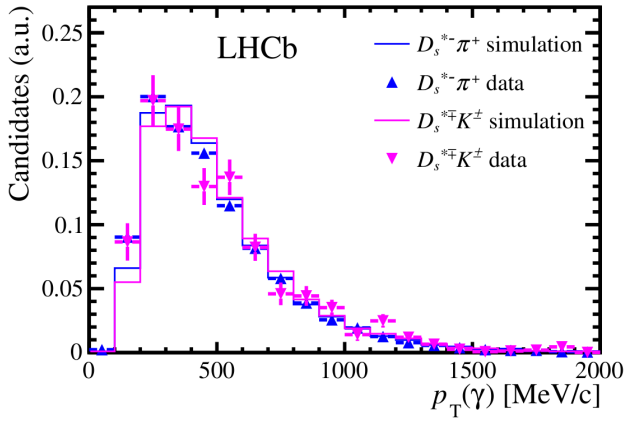
"First observation and measurement of the branching fraction for the decay $B_s \rightarrow D_s^* K$."

- LHCb-PAPER-2015-008 out on JHEP last week!
- We observed for the first time the $B_s \rightarrow D_s^* K$ decay and we measured the BR ratio:

$$\mathcal{B}(B_s^0 \rightarrow D_s^{*\mp} K^{\pm}) / \mathcal{B}(B_s^0 \rightarrow D_s^{*-} \pi^+) = (0.068 \pm 0.005^{+0.004}_{-0.003})$$

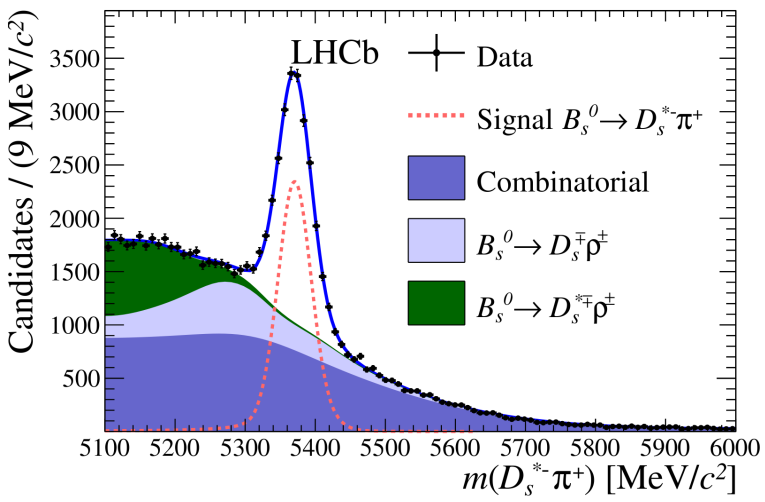
- One of the distinctive features of the present analysis is the reconstruction of the decay $D_s^* \rightarrow D_s \gamma$ in an hadron collider.

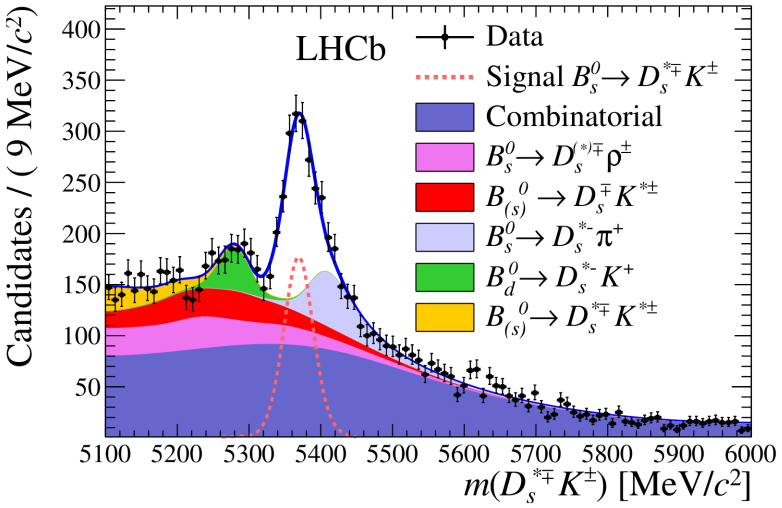




The $B_s \to D_s^* K$ and $B_s \to D_s^* \pi$ mass fits

- 1-dimensional mass fits are performed with the B2DXFitter package:

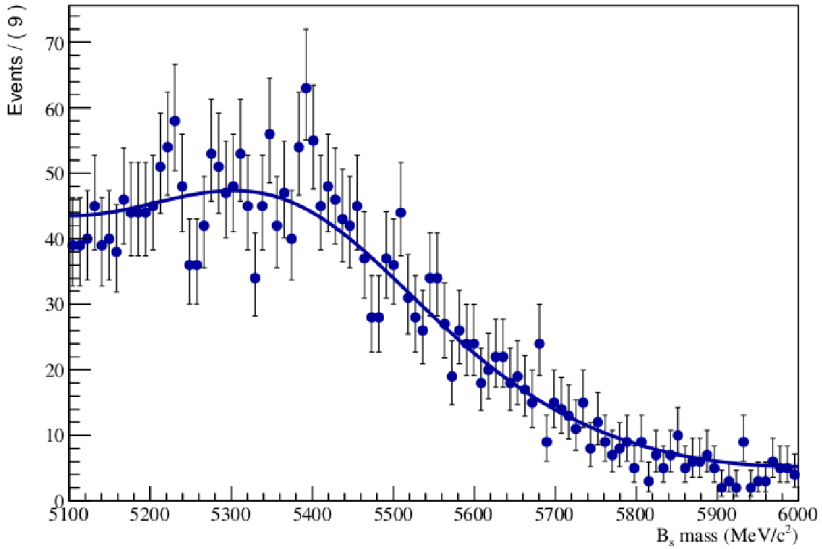




- Thanks to Agnieszka our fit is available in the official package release.
- Fit stategy:
  - The signal shape is a Crystal Ball obtained from MC → **fitSignal.py**
  - The background templates are *RooKeysPdfs* obtained from MC (with some exceptions...) → **prepareWorkspace.py**
  - In the fit: backgrounds and signal yields, some signal CB parameters are left free → **runMDFitter.py**

$B_s \rightarrow D_s^* K$ mass fit specific features: Combinatorial shape

- Describing the combinatorial background in this channel has been quite challenging.
- In our approach: the combinatorial shape has been described by a *RooKeysPdf,* obtained from data events in the $\Delta_M = M(D_s^*) - M(D_s)$ sideband [185,205] MeV.
- Basically in this region we have true D_s + random γ events.
- This sideband is far from the Δ_M signal region [124,164] MeV, and we took account of possible signal leaks in the systematics.



Combinatorial shape

- In the default fit configuration: the combinatorial is modeled through a gaussian + exponential shape. Parameters are free in the mass fit.

- To perform the mass fit with the *RooKeysPdf* combinatorial a specific option has been introduced: --rookeysforcomb

```
python runBsDsstKMassFitterOnData.py --merge -m both -o kkpi --dim 1 --configName
Bs2DsstKConfigForNominalMassFit --fileName work_bsdsstk.root --wider -s WS_Mass.root --rookeysforcomb
```

- The sideband region can be defined in the **python configuration file** (Bs2DsstKConfigForNominalMassFit.py in our case):

```
configdict["CreateRooKeysPdfForCombinatorial"]["All"] =
{"Cut":"FDelta_R<1.0&&FDelta_M>205.&&FDelta_M<245&&((FDsBac_M-FDs_M)<3370||(FDsBac_M-FDs_M)>3440)", "Rho":3.5,
"Mirror":"Both"}
```

- The results comparison from the two approaches (gaussian+exponential vs RooKeysPdf) can be a way to estimate the systematic error associated to the combinatorial describtion.

Systematic uncertainties in templates estimation

- In the fitter default configuration: background templates are obtained using the one-dimensional kernel estimation ***RooKeysPdf.***
- These are **non parametric PDFs.**
- In principle we should have a large number of MC events to obtain a template with negligible statistical error (but systematic errors from Data/MC differences still remains).
- But it is a common situation that, after selection cuts, the MC sample for our background studies remains with few events!
- How can we estimate the **statistical fluctuations in templates estimation** ?
- How can we take account of these **errors in the fit final result** ?

The bootstrap technique

- The **bootstrap technique** provides a means to evaluate how much trust our density estimation.

- It is a solid and mathematically proved statistical method. You can find mathematical proofs in the following documentation:

  - *B. Efron*, **"Bootstrap methods: another look at the jackknife"**, The Annals of Statistics 7 (1979) 1.
  - *I. Narsky and F. Porter*, **"Statistical Analysis Techniques in Particle Physics"**, Wiley-VCH, 2013.

- Inside the B2OC WG, this is the first analysis where this method has been used.

- The implementation in our code is easy ... but the algorithm can be CPU-time consuming!

The bootstrap algorithm

- The **bootstrap algorithm** goes as follow:

  1) We estimate a *RooKeysPdf* (*p*) from a set of *n* Monte Carlo events $\{x_1 , x_2 \dots x_n\}$.

  2) We extract randomly (uniformly) *n* events from $\{x_1 , x_2 \dots x_n\}$, admitting repetitions!

  3) In this way we form a new set of events $\{x^*_1 , x^*_2 \dots x^*_n\}$ where the same event may appears multiple times. This is called the **bootstrapped sample**.

  4) We estimate a *RooKeysPdf* (p^*) from the set $\{x^*_1 , x^*_2 \dots x^*_n\}$. This is called the **bootstrapped PDF**.

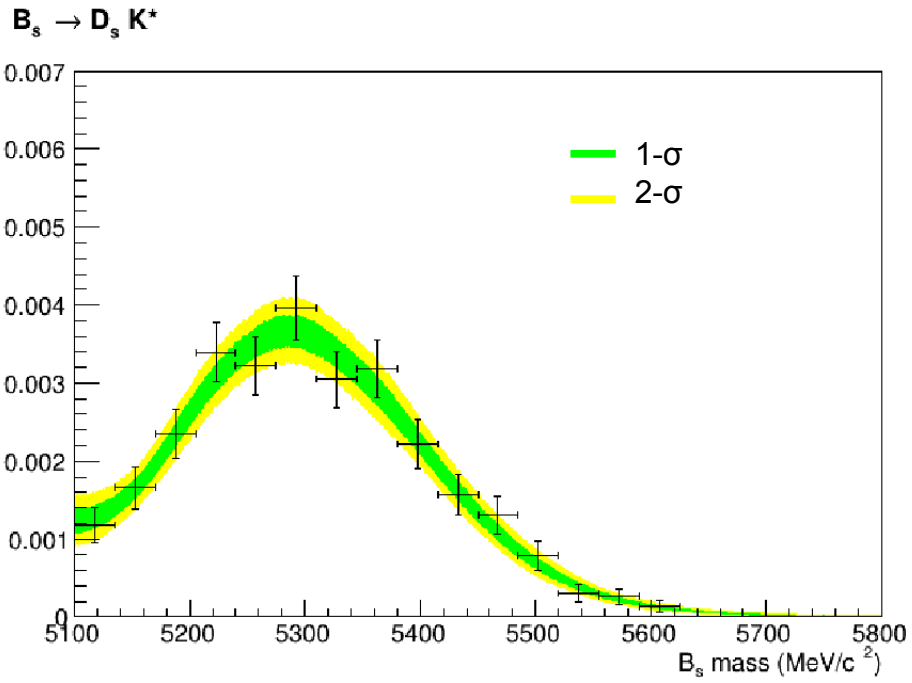
  5) We repeat the points from 2 to 4 *N* times to obtain a set of *N RooKeysPdf* $\{p^*_1 , p^*_2 \dots p^*_N\}$.

  6) **The distribution of the *p** mimics the distribution of *p* about the real unknown PDF**.

Bootstrap visualization

- In order to visualize the statistical error in the template shape we can generate a large number (*N*) of bootstrapped PDFs.

- Then we plot the 1-σ and 2-σ density bands of the superposition of the *N* PDFs.

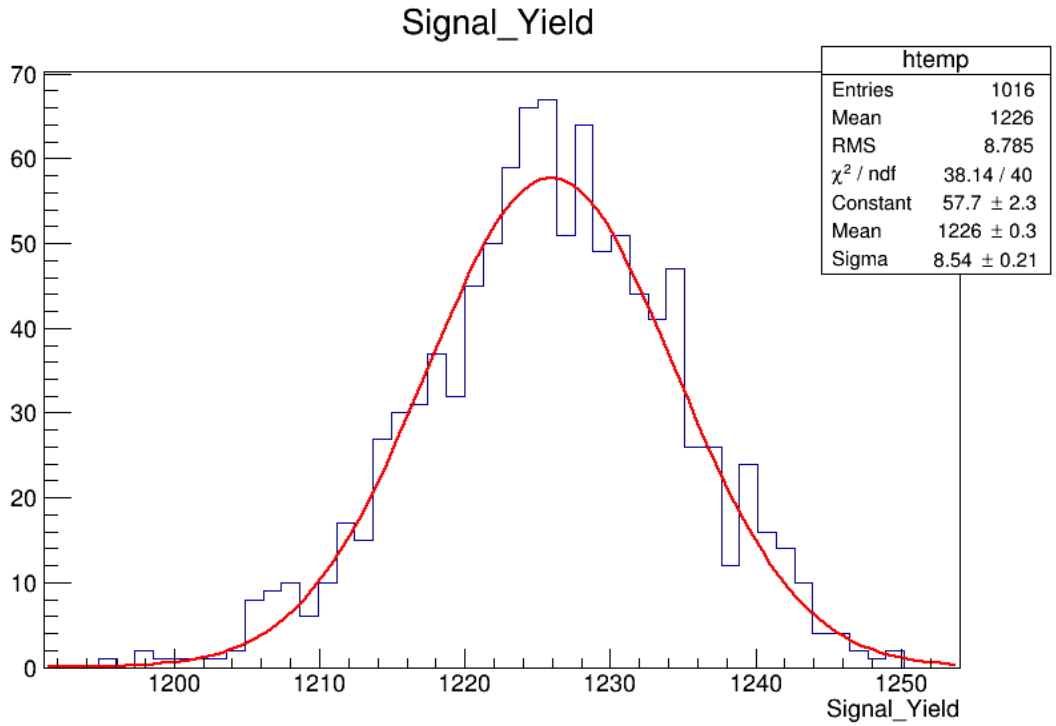


The MC sample events distribution (bar points) is overlayed to the band plot.

Error propagation

- We repeated the fit *N* times, each time substituting the template *p* with a bootstrapped PDF *p** obtained as described before.

- We plotted the distribution of the *N* fit results (signal yield):



- The mean of the distribution is the fit result in the original configuration.

- The width can be considered as **statistical error**.

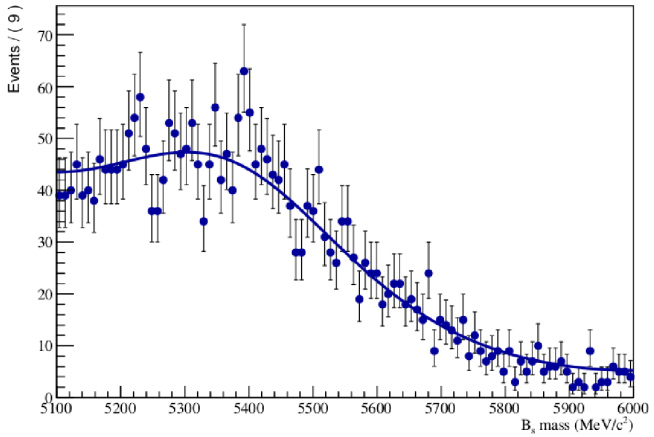
- We evaluated errors from each background separately and then we added them in quadrature to estimate the **total error.**

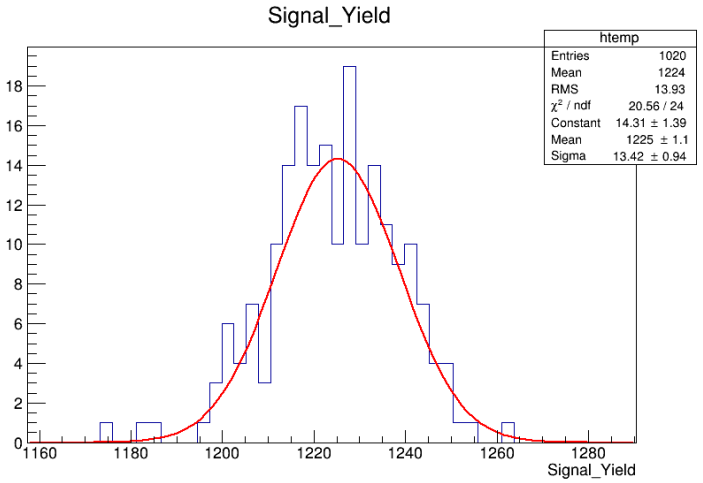
A special case: Combinatorial Background

- We obtained the combinatorial shape as a RooKeysPdf from the Δ_M sideband [185,205] MeV.

- **We cannot reduce the statistical error** (for the others background we could simply ask for more MC events).

- In this case the bootstrap technique can be **a reliable way to estimate the error on the background modeling**.





Implementation

- We **didn't modify the existing package code**.

- The method we used is **“quick and dirty”**, surely it isn't the best way to do it... Quick: in the sense of coding time, not execution time...

- The bootstrapped sample is obtained by using a **ROOT C++ macro**, “bootstrap.C”.

- A bash code runs several time bootstrap.C, the fit preparation python and the fit itself. Then the fit result is saved in a .txt file.

- At each iteration **the bootstrapped sample is deleted**, in order to avoid memory problems.

- We need to use a special configuration file in this procedure.

bootstrap.C highlights

- **Initialize TRandom3 with a saved seed:**

```
mySeedFile.open(string_seed);
UInt_t seed;
if (mySeedFile.is_open())  mySeedFile >> seed;  else  seed=65539;
TRandom3 rand(seed);
```

- **Boostrap cycle:**

```
for (int evt=0; evt<Nentries; evt++) {

        tree->GetEvent((int)rand.Uniform(Nentries)); //extract a random event
        .
        .
        b_tree-> Fill(); //Fill the bootstrapped tree
}
```

- **Save the last seed:**

```
remove(string_seed);
ofstream outputSeedFile(string_seed);
UInt_t new_seed=rand.GetSeed();
outputSeedFile << new_seed;
```

Bash code

```
for i in {1..1000}
do

   rm -f fit_result            //Delete the last fit result
   root -l -q -b 'bootstrap.C("/afs/cern.ch/work/a/abertoli/public/DsstrK/bdt_dev/Filter_Bs2DsstK_Bs2DsKst_up.root",
"/afs/cern.ch/work/a/abertoli/public/DsstrK/bdt_dev/Filter_Bs2DsstK_Bs2DsKst_dw.root" , "Filter_Bs2DsstK_Bs2DsKst_up_boot.root" ,
"Filter_Bs2DsstK_Bs2DsKst_dw_boot.root")'     //Bootstrap macro


   python prepareBsDsstKMassFitterOnData3D5M.py --configName Bs2DsstKConfigForNominalMassFit --Data --MC --Signal --Comb -s
work_bsdsstk.root   //preparation

   python runBsDsstKMassFitterOnData.py --merge -m both -o kkpi --dim 1 --configName Bs2DsstKConfigForNominalMassFit --fileName
work_bsdsstk.root --wider -s WS_Mass.root --rookeysforcomb | tee fit_results  //run the fit and save the result


   cat fit_result | grep "nSig_both_kkpi_Evts    1.2000e+03" >> FitResults_Bs2DsKst_boot.txt //take the event yield number and put it into a
.txt file


done
```

Configuration file

- In the data file *data/config_Bs2DsstK.txt* you have simply to substitute your regular MC sample name with the name of the bootstrapped one:

```
#MC FileName KKPi MD
.
.
.
{"Mode":"Bs2DsKst",
 "FileName":"/afs/cern.ch/user/l/lsestini/cmtuser/Urania_v2r4/PhysFit/B2DXFitters/scripts/Filter_Bs2DsstK_Bs2DsKst_dw_boot.root",
 "TreeName":"tuple;1"}
.
.
.
#MC FileName KKPi MU
.
.
.
{"Mode":"Bs2DsKst",
 "FileName":"/afs/cern.ch/user/l/lsestini/cmtuser/Urania_v2r4/PhysFit/B2DXFitters/scripts/Filter_Bs2DsstK_Bs2DsKst_up_boot.root",
 "TreeName":"tuple;1"}
.
.
.
```

Execution time

- At each iteration the most time is spent in the **bootstrap + RooKeysPdfs evaluation + Fit process**.

- The Fit time is the same for each bootstrapped background in study. The time for the bootstrap process and the RooKeysPdfs evaluation depends on how may MC events we have in the sample.

- So **more MC events we have more is the CPU time** ... but less is the statistical error!

- Running the entire algorithm may take several days ... If the MC statistics for one background sample is really high we can decide to consider its error negligible.

Final thoughts about the bootstrap tecnique

- **Density kernel estimation** is now common in our analysis.

- It is also common that after the selection cuts our MC samples are **statistically limited**. This fact can lead to an error in our measurement.

- With the bootstrap tecnique **we can estimate this error** on the fit result.

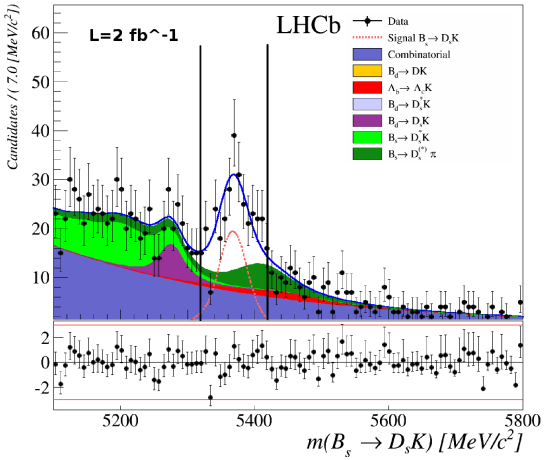
- In this way we have a parameter that can help us to decide if it is necessary to ask for more MC...

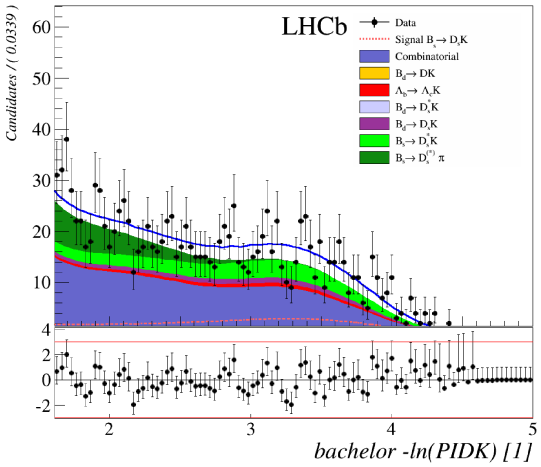
- ... but if we are estimating the PDF on a control data sample we have no way to reduce this uncertainty.

- **The implementation is simple**, even without touching the original fit code. It could be nice in the the future to see it hardcoded in the package

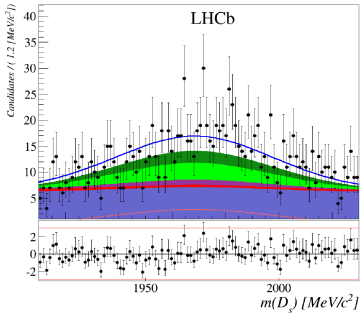
$B_s \rightarrow D_s (\rightarrow KK\pi\pi^0) K$

- Here in Padova we studied the possibilty to add the **$D_s \rightarrow KK\pi\pi^0$ decay channel to the $B_s \rightarrow D_s$ K analysis**.
- We extracted the signal yield with a **3-dimensional fit** (B_s mass, D_s mass, bachelor PIDK).





- We have to decide if only **150 events** are worth the effort...
- But we have still to try with the Stripping21!
- The $KK\pi\pi^0$ decay mode mass fit is available in the **official package**.



Yields in signal region M(B_s) [5320, 5420] MeV

Signal	149.5 ± 17.7	$\sigma_{\text{signal yield}}$=11,8 %
Comb	102.3	
$B_{(s,d)} \rightarrow D_s^{(*)}\pi$	47.6	
$B_d \rightarrow DK$	0	
$B_d \rightarrow D_s K$	4.6	
$\Lambda_b \rightarrow \Lambda_c K$	12.5	
$B_{(s,d)} \rightarrow D_s^* K$	4.3	

Next steps on $B_s \rightarrow D_s^* K$ and Conclusions

- Move on a **2- (or 3-) dimensional mass fit**. This should be easy with our current package version.
- **Time-dependent analysis**: extract the γ measurement.
- Need to work on the code to **adapt the $B_s \rightarrow D_s K$ gamma-fitter to our channel**.
- The overall experience with the B2DXFitter package has been really **positive**. The code in the current version is **flexible** (i.e. adding backgrounds can be done directly in the configuration files) and can be easily adapted to other decay channels.
- We suggest to create something like a twiki page where we can write the informations about all the different channel-specific features in the fitter (i.e. --rookeysforcomb for *$B_s \rightarrow D_s^* K$).*

Backup slides