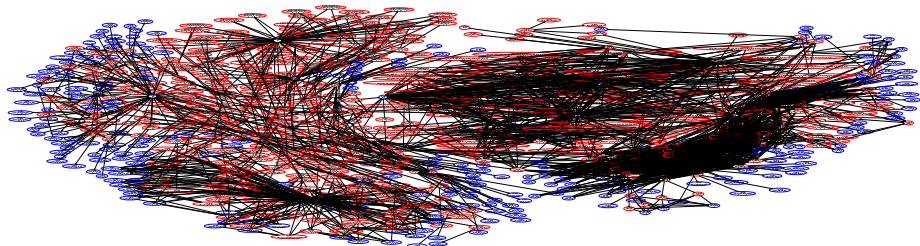


Getting started with B2DXFitters


Manuel Schiller

CERN

July 9th-10th, 2015



PDF structure of the $1 \text{ fb}^{-1} B_s^0 \rightarrow D_s^\mp K^\pm$ cFit

 setting things up

- before we can get started, we have to set up the environment
 - B2DXFitters is light in its requirements:
 - ROOT (recent 5.34 or 6.02 release or newer) and gmake are enough ([standalone builds on your laptop!](#))
 - you can (of course) also use an LHCb environment (Urania/Erasmus/DaVinvi/...)
 - will briefly go into both...
- for writing your own code, have to load the right libraries



setup in LHCb environment

setup in LHCb environment

setup in LHCb environment (1/2)

this follows the usual pattern when you compile LHCb software:

■ set up environment for building

```
> SetupProject --build-env Urania v3r0
Build-time environment for Urania v3r0 ready.
Created user project in /home/mala/cmtuser/Urania_v3r0
Current directory is '/home/mala/cmtuser/Urania_v3r0'.
Using CMTPROJECTPATH = '/home/mala/cmtuser:/local/lhcb:/local/lcg/releases:/local/lcg/app/releases:/local/lcg/external'
```

■ getpack UraniaSys and B2DXFitters

```
> getpack UraniaSys v3r0
.....
A UraniaSys/CMakeLists.txt
Checked out revision 191429.
Checked out package UraniaSys v3r0 (1/1)
Processed packages:
    UraniaSys      v3r0
> getpack PhysFit/B2DXFitters head
.....
A PhysFit/B2DXFitters/CMakeLists.txt
Checked out revision 191429.
Checked out package PhysFit/B2DXFitters head (1/1)
Processed packages:
    PhysFit/B2DXFitters  head
```

setup in LHCb environment (2/2)

this follows the usual pattern when you compile LHCb software:

■ add B2DXFitters to cmt/requirements of Urania

```
> echo "use_B2DXFitters_v+_PhysFit" >> UraniaSys/cmt/requirements
```

■ configure packages

```
> for i in UraniaSys PhysFit/B2DXFitters; do (cd $i/cmt; cmt config); done
```

■ source setup.sh to set up environment

```
> source ./UraniaSys/cmt/setup.sh
```

■ compile everything

```
> cd $URANIASYSROOT/cmt; cmt br make -j8
```

■ start hacking¹...

```
> cd $B2DXFITTEERSROOT
> source $URANIASYSROOT/cmt/setup.sh
> vim ...
```

¹diversity is important, so emacs or similar tools are permissible ;)

standalone setup



standalone setup

- useful when you don't want/need all that LHCb software crap
- requirements
 - *nix system with recent C++ compiler (C++11)
 - recent ROOT (6.02 or newer, or a recent 5.34 version) with RooFit and Minuit compiled in
 - for ROOT 5.34:
 - gccxml, ROOT compiled with reflex support
 - GNU Make
- make sure you can start ROOT from your command line

- check out and compile:

```
> svn co svn+ssh://svn.cern.ch/repos/lhcb/Urania/trunk/PhysFit/B2DXFitters B2DXFitters
...
> cd B2DXFitters/standalone
> make -j4
...
> source setup.sh
```

- start hacking...

loading libraries



- if you write your own C++ macros, this is what you have to do:
 - only ROOT 5:

```
#include "Cintex/Cintex.h"
gSystem->Load("libReflex");
gSystem->Load("libCintex");
ROOT::Cintex::Cintex::Enable();
```

- then in all ROOT versions:

```
gSystem->Load("libRooFit");
```

- then, depending on
 - standalone environment

```
gSystem->Load("/full/path/to/B2DXFitters/standalone/libB2DXFitters");
```

- LHCb software environment

```
gSystem->Load("/full/path/to/B2DXFitters/x86_64-slc6-gcc48-opt/libB2DXFittersDict");
gSystem->Load("/full/path/to/B2DXFitters/x86_64-slc6-gcc48-opt/libB2DXFittersLib");
```

- for standalone scripts (`chmod 755`) copy the prolog in `scripts/prolog.py` to the start of your script
 - handles all the ROOT 5/6, LHCb/standalone environment issues
 - will find and load the correct libraries
 - set up a few useful tweaks (`jemalloc/tcmalloc`, `ulimits`)
- in all python code, include this at the start:

```
import B2DXFitters
import ROOT
from ROOT import RooFit
```

- much easier than the C++ version, right?
I guess you wanted to code in python all along...

package structure



reminder: package structure

- important to understand package structure (subdirectories):

B2DXFitters	header files for C++ algorithms
cmt	used for cmt (building)
data	various data files (templates), config files
dict	ROOT dictionaries (reflection information)
doc	release.notes, other documentation
python	reusable python code
scripts	fitting (python) scripts
src	C++ sources
standalone	standalone build dir (symlinks to src/*.cxx)
tutorial	material for hands-on session (feel free to add!)
- looking for RooFit classes: [B2DXFitters](#), [src](#) ([,](#) [standalone](#))
- looking for reusable parts of fit: [python/B2DXFitters](#)
- concrete fit implementations: [scripts](#)

hands-on session time fit



hands-on session time fit

- now you know how to set things up, and where things are
- now we can start the tour...

time fit hands-on session

- five examples in tutorial²:
 - time-tut000.py: average η , perfect σ_t , no acceptance
 - time-tut001.py: average η , σ_t , no acceptance
 - time-tut002.py: average η , σ_t , spline acceptance
 - time-tut003.py: per-event η , average σ_t , spline acceptance
 - time-tut004.py: per-event η , σ_t , spline acceptance

²please svn up



time-tut000.py

- time-tut000.py: average η , perfect σ_t , no acceptance
- suggestions
 - look at it
 - notice the config file reading functions from `B2DXFitters.utils`
 - generation and fit pdfs separate
 - pdf and fit result written to ROOT files
 - run it: `./time-tut000.py SEED`
- next steps:
 - add average σ_t (use yesterday's slides as a guide)



time-tut001.py

- time-tut001.py: average η , σ_t , no acceptance
- suggestions
 - look at it
 - have a good look at setConstantIfSoConfigured
 - let's play with/discuss settings for Optimize, NumCPU...
 - maybe can discuss settings for numerical integration (on the side...)
 - run it: ./time-tut001.py SEED
- next steps:
 - add spline acceptance (use yesterday's slides as a guide)

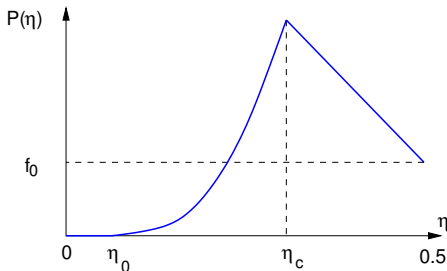


time-tut002.py

- time-tut002.py: average η , σ_t , spline acceptance
- suggestions
 - look at it
 - biggest change: PDF builder function for separate PDFs for generation and fitting
 - look into ROOT files saved
 - let's discuss the blinding mechanism
 - familiarise yourself with `scripts/printFitResult.py`
 - run it: `./time-tut002.py SEED`
- next steps:
 - add per-event η (use yesterday's slides as guide)

hint on mistag distributions

- for toys, often you want something quick and dirty, no need for full template



- MistagDistribution has roughly right shape
- can tune average mistag of distribution as parameter:
 - zero up to η_0 – try $\eta_0 = 0.07$
 - then quadratic until η_c
 - then linear to point $(0.5, f)$ – try $f = 0.25$
 - tunes η_c to get desired average, e.g. 0.35



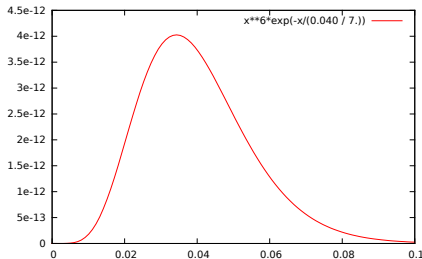
time-tut003.py

- time-tut003.py: per-event η , average σ_t , spline acceptance
- suggestions
 - look at it
 - familiarise yourself with scripts/make_histos.py (pull and residual plots)
 - run it: ./time-tut003.py SEED
- next steps:
 - add per-event σ_t (use yesterday's slides as a guide)

hint on mistag distributions

- again, for simple toys, you may not need a template
- can use

$$P(\sigma_t) \sim \sigma_t^6 e^{-\frac{\sigma_t}{\sigma_{avg}/7}}$$



- has about the right shape, can tune expectation value (σ_{avg})



time-tut004.py

- time-tut004.py: per-event η , σ_t , spline acceptance
- suggestions
 - look at it
 - run it: `./time-tut004.py SEED`
- next steps:
 - now you are ready to look at the “big beasts of the plain”, `scripts/runBs2DsKCPAsymm0bsFitter-cFit.py` ($D_s K$ cFit) and `scripts/plotBs2DsHTimeModels.py`
 - these are much uglier than the the examples (real world, you know...;)
 - if you've worked through the examples, you have all the tools you need to build your own fit (and look at the cFit to see what not to do)

- that's it
 - let me know what was useful (or not useful)
 - open to all kinds of suggestions (more/different material, presentation style, wishlist...)