LHCb Starterkit 2024 - Practice Session

February 16, 2024

1 Part 1 - Producing your nTuple

1.1 Goals of the session

Starting from a DST files with simulated events that passed HLT2 selections, we want to tuple the information using DaVinci.

- Decay of interest : $D^0 \to K^-\pi^+$
- Simulation sample used:
  `/MC/Upgrade/Beam7000GeV-Upgrade-MagDown-Nu7.6-25ns-Pythia8/Sim10aU1/27163003/XDIGI`
- You can find the dst files (2 files with different statistics) and the `tck.json` files in this directory:
  `/afs/cern.ch/work/f/femiguel/public/Starterkit24/`
- **Please copy the files to your working area.**

1.2 Setting the configuration

Working from lxplus.

1. Create a file python tupling.py, that will contain the algorithm used to config DaVinci.
2. Create a function that will be called when executing DaVinci (typically alg config()), taking options as argument.
3. This function return make config(options, algs). (This function needs to be imported from DaVinci).
4. To make things easier here are the list of imports we will be using.
```
from PyConf.components import force_location
from DaVinci.algorithms import add_filter
from DaVinci import make_config
from FunTuple import FunTuple_Particles as Funtuple
from FunTuple import FunctorCollection
import Functors as F
```

```
from FunTuple.functorcollections import (
MCHierarchy,
Kinematics,
SelectionInfo,
MCVertexInfo,
MCKinematics,
EventInfo,
)
from Functors.math import log
from DaVinci.truth_matching import (
configured_MCTruthAndBkgCatAlg,
)
from DaVinci.reco_objects import make_pvs_v2
from DaVinci.algorithms import add_filter, get_odin, get_decreports
```

5. This is the HLT2 line: `Hlt2Charm_D0ToKmPip_XSec_Line`

6. Hint: `add_filter` in this DaVinci version is the equivalent of the `create_line_filter` we saw in the DaVinci Run 3 lesson this year. This is due to these DST files being created with a previous DV version.

1.3 Setting the options

Create the `yaml` file and fill with the following (ignore the unintended indentation):

```
   input_files:  ...  # the dst file
annsvc_config:  PATH/hlt2.tck.json
input_type:  ...
input_raw_format:  ...
data_type:  Upgrade
simulation:  true
dddb_tag:  ...
conddb_tag:  ...
ntuple_file:  ...
process:  ...
```

1.4 Execution

To run the tupling:

```
lb-run DaVinci/v62r0p1 lbexec CONFIG OPTION
```

2 Part 2 - Analysing your data

2.1 Goals of the session

We will now put to use some of the knowledge learned in this workshop. You (should) have the data you're interested in, in some ROOT ntuples. What can you do with it?

We will go through some basic steps that you would take to study a Monte Carlo sample of your signal channel of interest. This includes applying some rudimentary cuts to it, and fitting the resulting mass distribution.

2.2 Setup of the Session

You should have worked this morning on producing a ntuple for the decay $D^0 \to K^-\pi^+$, with a simple DaVinci script. As running DaVinci can take a long time to get a reasonabe amount of signal events in your ntuple, for this second part of the Session you are provided with a ROOT file that contains the TTree with the kinematic, PID, and MC-truth variables, but with much higher statistics. This file can be copied on `lxplus` from

```
/afs/cern.ch/work/f/femiguel/public/SK_NewDV/DV_MC_files.root
```

We can now do some fits to the data in our ROOT file. To do that, you have many options at your disposal. You can use `RooFit` or `minuit` like we saw in the statistics lesson. You can also use `zfit`, in which case you will need to set up an environment. You can do so by cloning the git repository from the 2022 edition: here, you should be able to set up the evironment by just running on your command line:

```
./lbConda_Starterkit_Create.sh starterkitEnv source
starterkitEnv/run bash
```

to open a bash shell in the correct environment.

You can write a python script or a jupyter notebook if you want to add comments and explanation for the steps that you'll take throughout the session. We will not mark you, so this is just for your own future reference!

Use the lessons and materials provided during the week as you go along! This Session is not meant to be test of your knowledge, but just to make you more familiar with the tools we've seen. Helpers will be around to assist with issues you may encounter.

3 Exploring the data

First, you can use `uproot` to read and load in the data contained in the ntuple. Sort through the ROOT file to find the location of your TTree. Once you're able to laod it, you can start by loading in the D^0 mass branch, which is called

D0_M. Try and plot it and see what it looks like! You should see a peak over a continuous background.

Later, you can start loading in more branches. Here's a list of the ones that will be useful for today:

D0_M, D0_TRUEID, Kminus_PIDK, Kminus_PIDmu, Kminus_TRUEID, piplus_PIDK, piplus_PIDmu, piplus_TRUEID

You can have a look at them by plotting them and seeing what their distributions look like. We can note a few things about these branches:

- The ntuple contains both $D^0 \to K^-\pi^+$ and the charge-conjugate process $\overline{D}^0 \to K^+\pi^-$. The names of the branches of the children K^- and π^+ are only meant to reflect what the process is, but it doesn't mean that those branches *only* contains kaons and pions of a given charge.
- The TRUEID variables return an identification number assigned at MC level to any particle candidate in the ntuple. This is useful because simulated data that passes through the nominal LHCb reconstruction may produce candidates which do not correspond to the original signal generated by Gauss (as reconstruction is not 100% efficient). The TRUEID for D^0, K^- and π^+ are 421, -321, 211, respectively. More information on the conventions for this numbering scheme can be found here.
- The PID variables can be interpreted as a likelihood that a particle candidate is of a given species. For example, Kminus_PIDK returns the likelihood that the K^- candidate is actually a kaon; Kminus_PIDmu returns the likelihood that the K^- candidate is actually a muon; and so on. Note that the number returned is not a probability, *i.e.* a number from 0 to 1, but as a rule of thumb the chance of a particle being of a given species is higher the higher the value of the corresponding PID.

We can use these to apply cuts on our data! For example, when using simulation, a common selection is what is known as *truth-matching*. This means requiring that the TRUEID of the candidates match what we expect. Because we have both D^0 and $\overline{D}^0$ decay candidates, we have to require that the *absolute value* of the TRUEID matches the known values.

Apply the truth-matching cut on your data and look at how the shape of the distribution changes. You should see that most of the background gets removed, and only the peak remains.

Alternatively, you can play with the PID variables and apply different cuts to see how the distribution is affected.

3.1 Fitting your data

We're ready now to start doing some fits! We want to fit the mass distribution of the D^0 candidates. First, start by setting up your observable which will be the proxy for the variable D0_M.

As you have seen by plotting the data, we can start by modelling the peak as a simple gaussian. Set up the parameters for the mean and width, as well as a yield parameter. Try doing an extended unbinned fit with the gaussian to the non-truth matched data, and print out the result. Is it a good fit?

Next, you can plot the data and the fitted model together to see how well the gaussian models the data. You can also try to plot the 1D likelihood function for the mean of the gaussian.

Advanced: you can also try to plot what are known as *pulls*. You can think of pulls as a very naive measure of the goodness-of-fit. If you plot your data as a histogram, then the pull relative to one of its bins is defined as such:

$$pull = \frac{N_{bin} - y_{bin}}{\sqrt{N_{bin}}} \qquad y_{bin} = \int_{bin} pdf(m)dm$$

This is the difference between the number of entries in a given bin N_{bin} with the integral of the extended pdf over that bin, normalised by the poissonian error $\sqrt{N_{bin}}$. Generally, in a good fit, the distribution of pulls for all bin should be centered in zero and be symmetric. They should be randomly scattered and no systematic trend should be visible.

3.2 Improving the model

Since we're not modelling the background at the moment, we shouldn't expect to see a good fit when plotting it against our data. So, we can add to our model a second pdf, which represents the background. This can be modelled by an exponential function. Remember to set up the yield for this as well and sum the resulting extended pdf with the one that we made for the gaussian peak.

Try to fit this updated model to the data, and plot it. Does it look better?

Advanced: Plot the pulls for this as well.

4 Advanced: Storing the result

Now comes the part for you to reflect and be creative. We want ideally to store the information of the fit somewhere, how can we do that?

Hint: Would dictionaries be useful in this case?