Topics in Particle Physics and Astroparticles II

Guilherme Soares

Laboratory of Instrumentation and Experimental Particle Physics

7 March 2022

Index

Schedule for the Practical Classes

Week 1 7,10/3	**Week 2 14,17/3**	**Week 3 21,24/3**	**Week 4 28,31/3**	**Week 5 4,7/4**	**Week 6 11/4**	**Week 7 18,21/4**	**Study 25-29/4**	**Exam 2-6/5**
Root, Git, Colab	Fit LHC data	SHiP	HNL data	ML	ML			
		Proj1 Fit		Quizz		Proj 2 ML	Presentation	

GitHub, Google Colab and the Projects

Everything that will be done for the practical classes will be posted on a github page that is accessible to everyone **https ://github.com/Switch-bot/TFPA**.

First Project :

- Analyzing ROOT files
- Fitting data stored in TLorentzVectors
- Can be done in Python or C++

GitHub, Google Colab and the Projects

Second Project

- The Search for Hidden Particles Experiment
- Heavy Neutral Leptons and Dark Photons
- Distinguishing between Signal and Background
- Applying Machine Learning Algorithms
- **No need for ROOT**
- Google Colab does not require local installation

NO CODE WILL BE EVALUATED

There should be no need for static versions of deliverables since projects are planned to be relatively simple and delivering a link to your google colab or analogous notebook should be enough

ROOT Files

ROOT File Structure

- Not linear
- Heterogenous
- Main **TTree** object
- TTree has TBranches
- Branches have Leaves, which can range from concrete *floats* to abstract *TObjects*

ROOT Files

How do we know what is inside ?

Basic Functions

- TFile.ℓs()

Shows the main structure of the ROOT file

- Most TObjects have TObject : :**Print()** function

TTree : :Print() shows all the objects inside a TTree, including their types.

- Sometimes, **looping over objects** is the best option to explore what is inside
- In interactive mode *TBrowser T*

T is the object, so it can be any letter or word (like saying int a = 2)
This command opens a file explorer where you can see the superficial structure of the file.
We can save more than just raw data inside ROOT files.

TLorentzVectors

One of the main TObject types that we have are **TLorentzVectors**.

It is a 4-vector for position (X, Y, Z, T) or energy (Px, Py, Pz, E), but there is no actual distinction when defining it (the only thing that matters is that we know what it is).

It can perform operations between 4-vectors automatically, and can give quantities of interest like mass easily (TLorentzVector : :M()).

It cannot be utilized as an automatic variable for plotting within ROOT (There is no TTree : :Draw(TLorentzVector) option, so we need to loop over all events and fill histograms manually.)

Tutorial 1

There are 2 templates on how to read simple ROOT files available on github.com/Switch-bot/TFPA.

The simpler "Template_Basics_of_ROOT.ipynb" reads the *zjet.root* file, which is a file utilized in the LIP Summer Internships tutorials.

Check the structure of the file by utilizing the aforementioned ℓ**s()**, **Print()** and **GetEntries()** functions.

Plot the spectrum of several variables and try to identify what they are, by utilizing **TFile.TTree.Draw("TBranch")**.

Then take a closer look at the branches that are TVectors. How many values per entry do they have, and why ?

Tutorial 2 - TLorentz Vectors and Data Fitting

Available on the usual github page, the template "Template_RooFit.ipynb" reads *Skim4.root*, which is a file containing dimuon data.

Start by checking that there are 4 branches now, with all of them but "*event*" being TLorentzVectors.

Look at a few values for the branch "*event*". Why are the values like that ?

Plot the mass spectrum of dimuon events by utilizing the TLorentzVector : :M() function. Can you recognise any of the peaks ? Compare it to the mass spectrum for the *muonp_4* and *muonn_4* branches.

Check the TLorentzVector class and make use of its functions to create a new TLorentzVector that is the sum of muonP and muonN. Plot this new vector's mass and compare it to the dimuon spectrum.

As a last exercise, you can see that the binning on TH*F histograms doesn't need to be uniform. By giving the binning an array input, you can define the size of each individual bin equal to the value of each entry of the array.