Computing Methods in Particle Physics

Introductory Lecture

Lectures Overview

- **Week 1**: Introduction
  - Computing tools and background necessary for working efficiently
  - Setting up a working environment for this course
  - Python, NumPy, and Matplotlib
- **Week 2**: Bayesian Inference with MCMC
- **Week 3**: Classification Problems
- **Week 4**: Neural Networks and Convolutional NNs

Goals

Machine Learning is the common theme here! But this is not a machine learning course. We are focusing on the practical aspects of using ML methods, particularly in ways that are widely applicable in particle physics.

There is not enough time to cover each of these topics in detail.
Each could easily fit into a course on its own!

Goals:

- Introduce a broad range of concepts from each topic
- Discuss a specific example or an approach
- Demo a specific, simple example that illustrates the concept and introduces software libraries

Materials

General references:

- “Effective Computation in Physics”, A. Scopatz, K. D. Huff
  - A very good overview of **modern** tools used in an analysis.
- Frequentism and Bayesianism: A Python-driven Primer
- Bayesian Analysis with Python - O. Martin
- Scikit learn user’s guide
- TMVA user’s guide

Assignments

No assignments.

But I will suggest extensions to the demonstrations and other problems that could be attempted for those that want to learn more.

I will be available to help! Email me at: giacomo.artoni@physics.ox.ac.uk

Credits

The material for these lectures was initially put together by Matthew Bass, this is nothing but an updated version of those. Most of what you will find here was in his initial lectures too and full credit should go to him!

Computing Environment & Best Practices

Background

I understand that there are varying levels of background for everyone coming into these lectures.

It's worth your attention to get up to speed on:

Programming languages
Linting
Operating systems
Working in a shell environment & shell scripting
Plotting tools
Multi-threading & parallel computing
Data management
Version control
Build systems
Debugging

A brief discussion of most of these topics follows. I highly recommend **"Effective Computation in Physics"** (link) to get up to speed on the details of these topics!

Programming Languages

There are many **languages** in use in particle physics. The dominant languages are C++, Fortran (legacy), and Python.

You may not have a choice what language you use for interacting with the simulations and raw data of your experiment. The focus of this course is not on a specific experiments requirements, but to give a flavour of what is available.

Will primarily be using Python, with some C++ examples offered.

Programming Languages - Analysis

For an analysis, focus should be on **ease of development**, **repeatability**, and **documentation**.

- Physics analyses are complicated enough, the more time you can focus on the physics, and not the language, the more effective you will be!

For this reason, many people choose **Python** for analysis. If a specific piece of code requires optimization for speed/memory, Python can interface with external libraries quite easily.

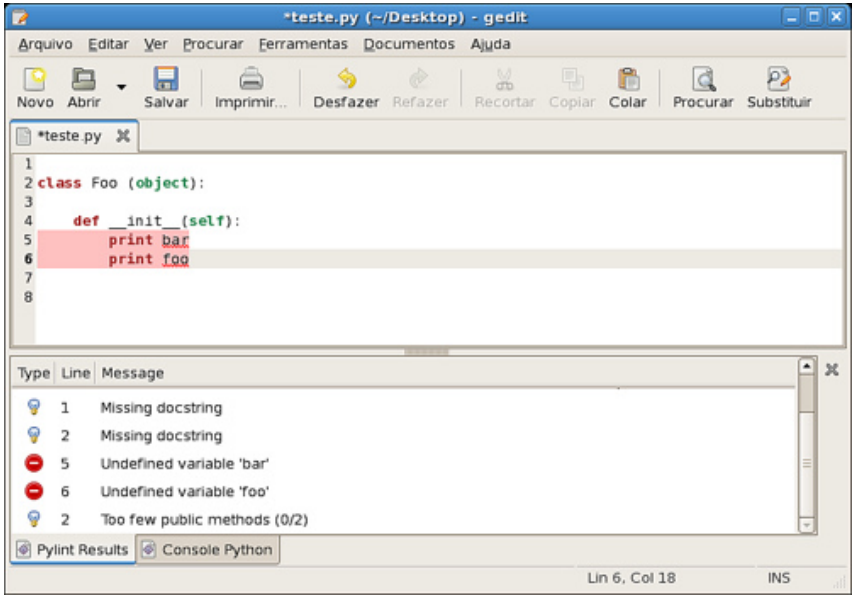
What is Linting?

A **linter** is a tool that scans your code to look for style, logic, and syntax problems:

- Conformance to style guidelines (spacing, variable/function names, etc)
- Simple coding mistakes and language misuse
- Unused variables
- Duplicated code

They generally can run as part of your development environment (text editor, emacs, vim, sublime, or IDE).

Will help you develop code that is **conducive to collaboration**!

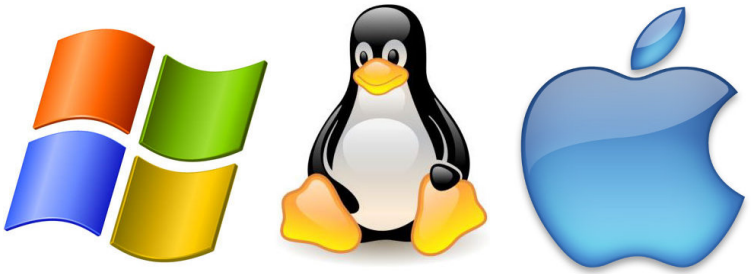


Operating Systems

In principle, should not matter whether you run Linux, Mac, or Windows.

For an analysis, ***any*** of these could work.

Be aware that it's possible to virtually run an operating system! This can be useful if you don't want to pollute your own OS, but you want to run some software stack from your experiment. Read up on virtual machines or check out a VM host. ◆



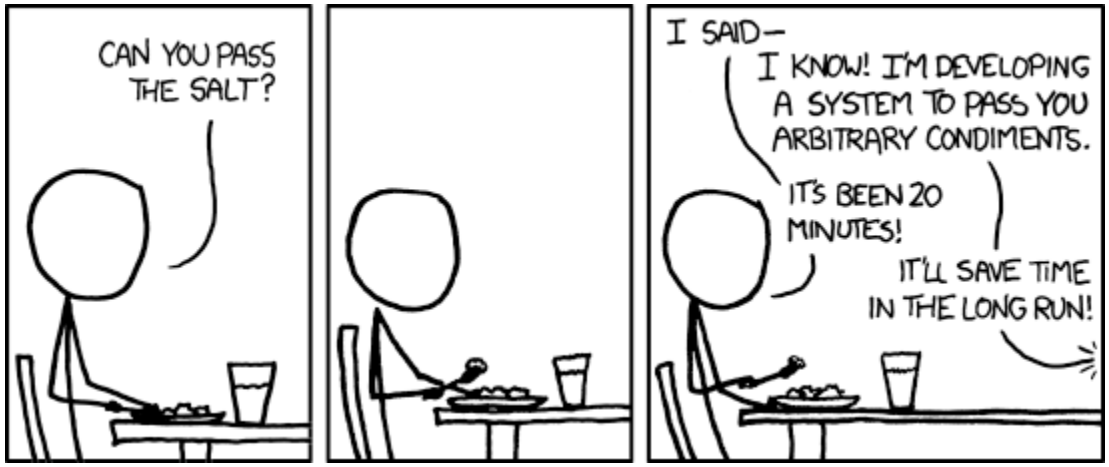
Shell Scripting

Useful to know a bit about shell scripting to build data processing schemes, automate repetitive tasks, and interact with Unix & experiment-specific tools

Most common: **Bash** (often the default shell)

You can often achieve simple tasks much more quickly than coding it in Python. Where to draw the **line** between them is up to you!

```
#!/bin/bash
for file in $*
do
 # do something on $file
 [ -f "$file" ] && cat "$file"
done
```



Plotting Tools

ROOT - A modular scientific software framework

This is the most commonly used package in particle physics, though not exclusive.

More than just for plotting of course: data management, GUIs, event displays, fitting, statistical analysis, etc.

Not heavily used outside of particle- or astro- physics.

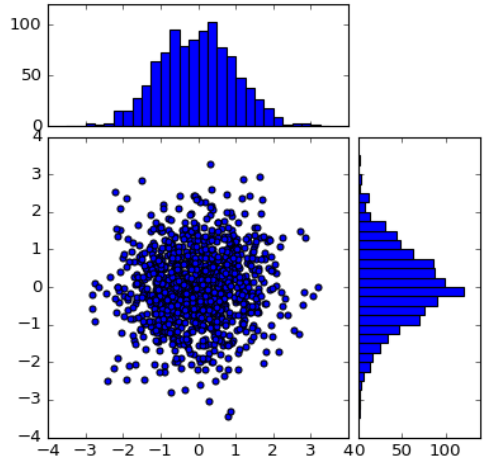
Plotting Tools

Most particle physics experiments will use ROOT by default. Keep in mind it's not the only plotting tool and is often not the best.

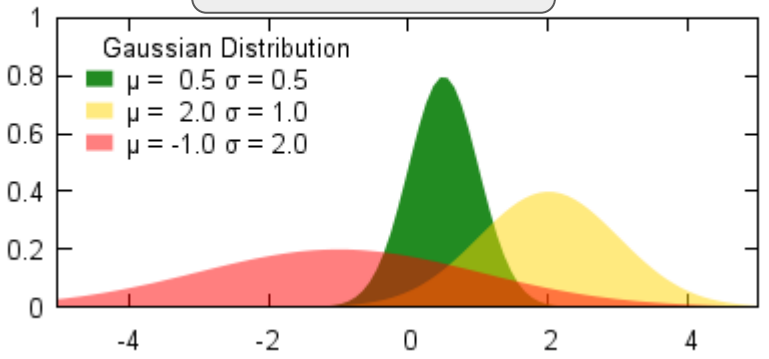
It is also good to have experience with tools that are used in industry (or at least be aware of what is out there).

My general approach: Use the galleries (Matplotlib, Gnuplot, ROOT) to find examples of the type of plot needed.

Matplotlib



Gnuplot

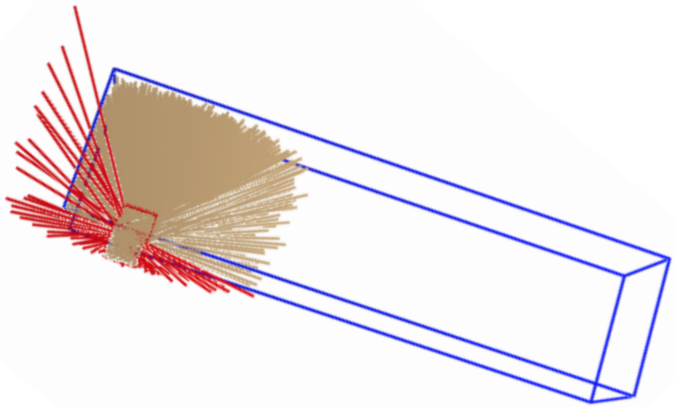


3D Visualization

Often it is most informative to view data or simulation in 3D, especially when working with spatial information. No time to cover, but can be extremely useful!

ROOT's TPad

pyqtgraph



Data Management

How do we store the data from an experiment, simulations, or output from an analysis?

Experiments all have their own data management strategies.
Many center around ROOT.

You will likely produce some simulation or analysis data objects

- **Metadata** is important for provenance (json, xml). Provenance refers to being able to track the data, algorithms, code versions, and any other information necessary for reproducibility.

Data Management - Tools

ROOT

- NTuples and TTrees can store data (floats, ints, etc.) as well as custom C++ objects
- Accessible via C++, Python, CINT, Macros
- Can store analysis objects (histograms, canvases, plots, etc.)

Databases

- Store data in tables with relationships between fields enforced
- Query very special query language, SQL
- Useful for storing relational information

HDF5, **Pandas**, Pickle… Ok, there are a lot of options. Technically they could all work for any problem. Choose based on what your colleagues are using or what you are most comfortable with.

Version Control

Version control allows for file changes to be tracked. Any text based file can be versioned.

To make your analysis **robust** and **reproducible**, it is important to be able to **rewind** changes made to your code and recall logic that went into a past analysis.

Even if you are not working in a group, it's a good **habit** to develop.

Features in common to many version control systems:

- Track changes to files across a directory structure
- Make history accessible in some way; typically with commit messages
- Assist in merging changes made by collaborators
- Implement tags for producing named version of code (e.g. "JINST_Analysis_v1")

Version Control: git & github

One of the more popular version control system repository formats is **git**.

Git is the repository; can be hosted anywhere.

Github is a popular place to host. ◆

Build Systems

A **build system** manages the dependencies and environment setup necessary to build a complex software package. It also typically handles software version requirements, cross-platform issues, and compiler issues.

CMake
CMT (Configuration Management Tool)
MRB (Multi-Repository Build)

Your experiment will likely use one of these to handle building simulation and data analysis packages. You likely will not need anything so complicated for an analysis.

Makefiles can help with some of these issues on a small scale.

Summary

These items should be considered and planned for at the beginning of *any* project.

A general approach to starting a project:

1. Setup a github (or other) repository with skeleton code
2. Plan for data sources (also a good time to consider data security, assurance, and provenance)
3. Plan for documentation (e.g. docstrings in python, or doxygen, or follow your experiment's guidelines)
4. Start coding, documenting and committing frequently

Setting up a Python Environment

Anaconda

We will use the Anaconda platform to install Python and the dependencies we need for examples in this course.

Anaconda takes care of package, dependency, and environment management and includes support for a lot of data science packages. E.g. python, ipython, numpy, scipy, ipython-notebook, matplotlib, pandas, …

1. Download the appropriate version of Anaconda (Python 3.7) for your system
2. Install using the appropriate instructions for your OS
3. Install the packages we will need via: `conda install --yes numpy scipy ipython ipython-notebook matplotlib pandas pytables nose setuptools sphinx mpi4py seaborn pymc;`
   `conda install -c conda-forge pymc3`

Testing Environment

Jupyter is an easy way to interactively work with Python, sort of like a Matlab interface with immediate feedback. Very useful for developing and tweaking plots!

Launch a jupyter notebook from the shell: `jupyter notebook`

The Jupyter backend will launch in the terminal and a browser session will open.

Side note: see here for a list of interesting ipynb examples

ROOT

We will have at least one example that uses ROOT. Two options for getting access to ROOT:

- Use the version on the PPUnix cluster (**recommended**)
- Install it locally. I can (try to) help if you run into problems here.

Python Review

The PP department has a nice Python tutorial here. I will not review the language again, but do **ask questions** in the examples that follow if something is not clear!

Python is an interpreted scripting language (as opposed to a compiled language like C++) that is:

- Object oriented
- High-level
- Supports modularity and has a rich repository of libraries

Extensive library availability makes complicated analysis tasks more manageable. We will see a few examples of this now...

NumPy Overview

See NumPy notebook

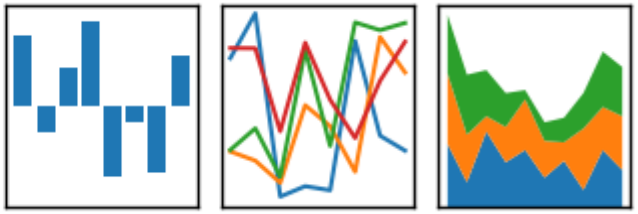
Matplotlib Overview

See MatPlotLib notebook

Pandas Overview

pandas

$$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$$



Pandas is an easy to use, numpy based data analysis library. High performance, uses numpy to optimize operations.

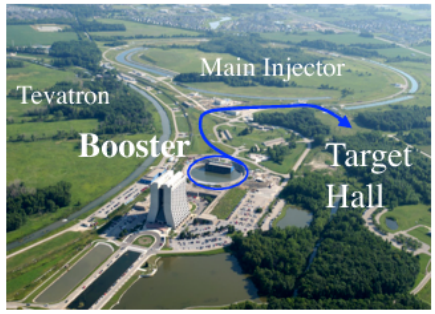
Example: Will use dataset from **MiniBooNE** monte carlo simulation of signal and background events. Note: we **do not** need to know what the **features** are to work with the data!

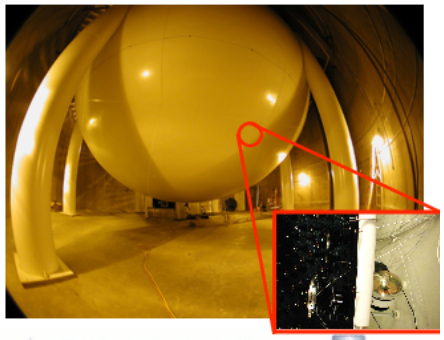
See the UCI Machine Learning Repository for more datasets.

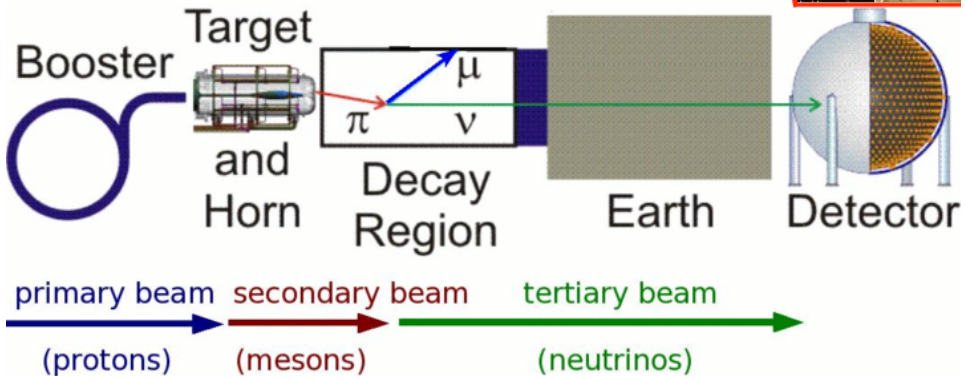
- Higgs Data Set (LHC-like MC Simulations, 28 features, 2 classes: Sig, Bkg)
- HEPMASS Data Set (28 features, 2 classes: Sig, Bkg, multiple masses)

Pandas example notebook is here.

MiniBooNE Overview







Suggested Exercises

1. Install anaconda and set up a working Jupyter/Python environment.

2. Download the demonstration notebooks from this lecture. Make sure they work in your environment and try modifying them to make sure you understand them.
   a. Work through the NumPy, Matplolib, and Pandas notebooks
   b. Modify the pandas notebook to use another dataset

3. If you have not worked with git or github before:
   a. Create a github account. Create a new repository and commit a file.