MaCh3 user categories and documentation

Charlotte Knight

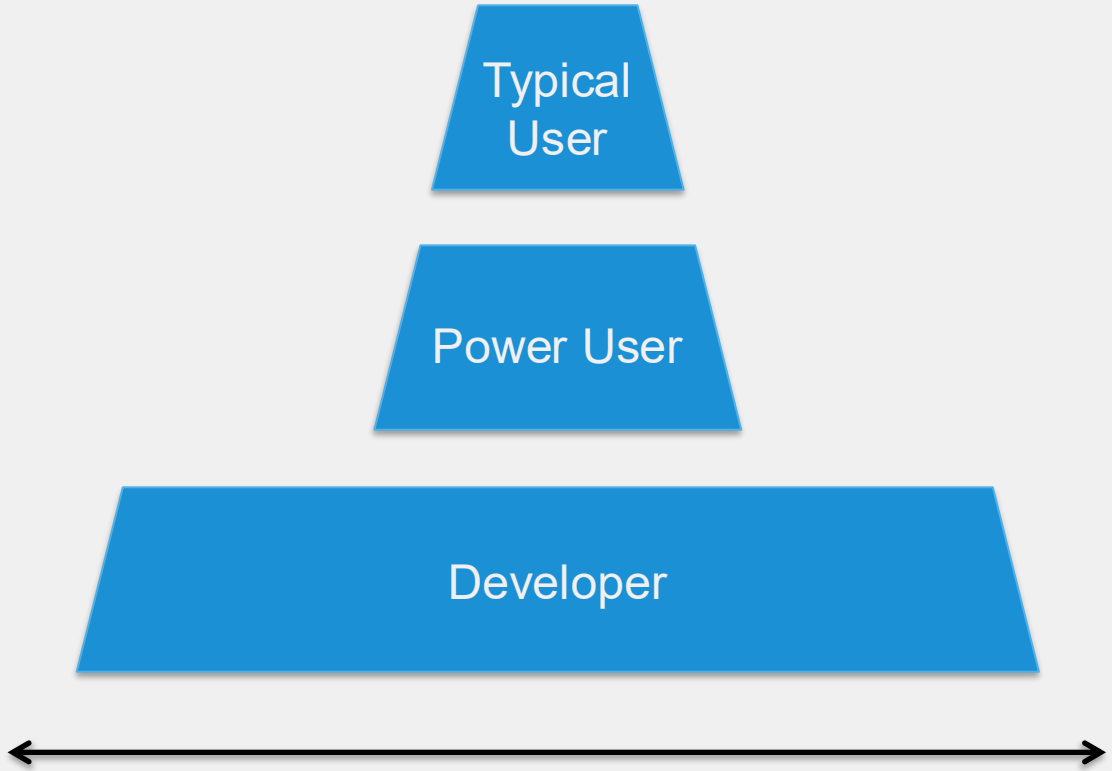
MaCh3 Workshop

05/09/2026

Types of people interacting with a framework

- A ”typical” user
  - Interacts with the framework as intended, e.g. calling high-level API or using a CLI
  - Behaviour is suited to their use case only through configuration
- A “power” user
  - Spends most of their time as a typical user but occasionally wants to extend the behaviour of the framework in a (more) supported/abstracted way
  - Examples: writing a custom optimiser in pytorch, writing a new fitting algorithm in MaCh3
- Developer who is also a user
  - Has intimate knowledge of the code + subject matter, e.g. neutrino physics
- A pure developer
  - Also intimate knowledge of the code + maybe adjacent subject, e.g. computing

Pyramid representation of roles (idealized case)



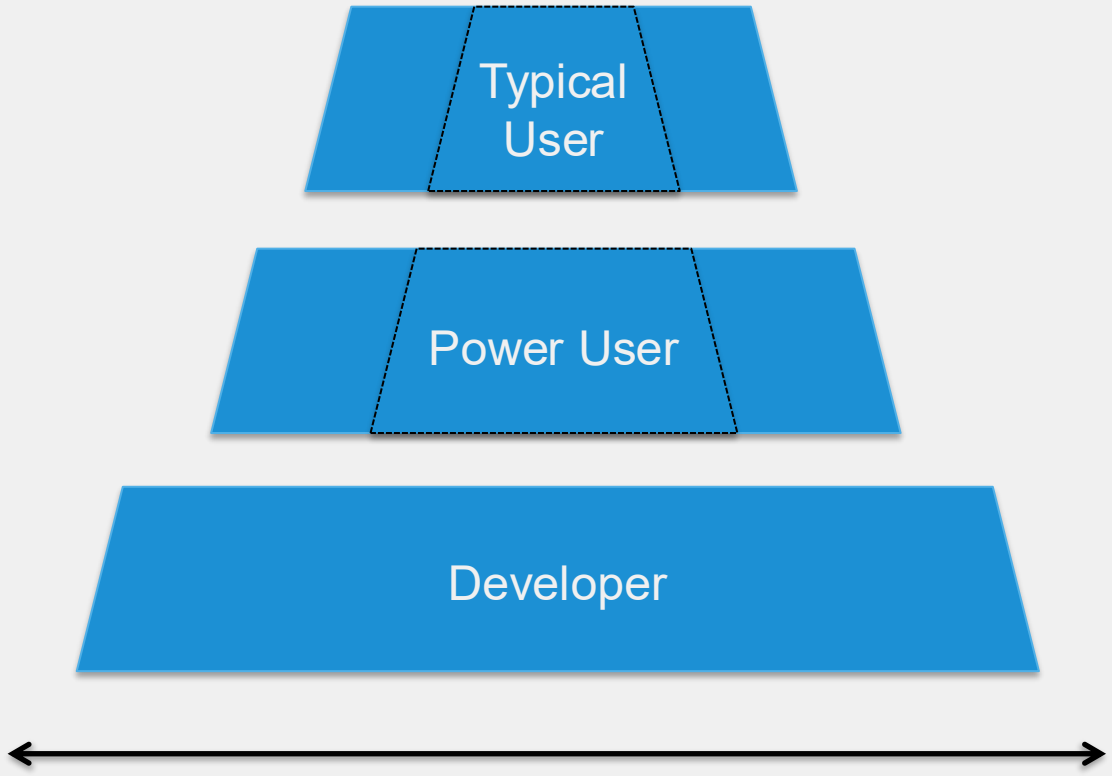
A typical user should only need to be familiar with the high-level API and concepts

A power user needs a little bit extra e.g. needs to know what class to inherit from and what methods to override

A developer knows it all, starting with high-level API -> nitty-gritty details like how we to use OpenMP for multiprocessing

*not to scale
**it is more complex than this

Pyramid representation of roles (current MaCh3 case)



*not to scale
**it is more complex than this

One line summary: I think we are somewhere between complete overlap and the idealized case

The majority of users need to write code, and often end up developing MaCh3-core

In my experience, it is trivial for users to encounter seg faults that require reading core code to understand

Is this just the nature of our problem? Does a world exist where a neutrino analysis is completely configurable and does not require coding?

Probably somewhat true… but I think there is scope for improvement.

How to enable better factorization?

Code-based improvements

- Configuration
- Abstraction where necessary (power users)
- Exception handling

Documentation-based improvements

…?



Alex will talk on user interface & configuration later

Documentation

Some of my opinions:

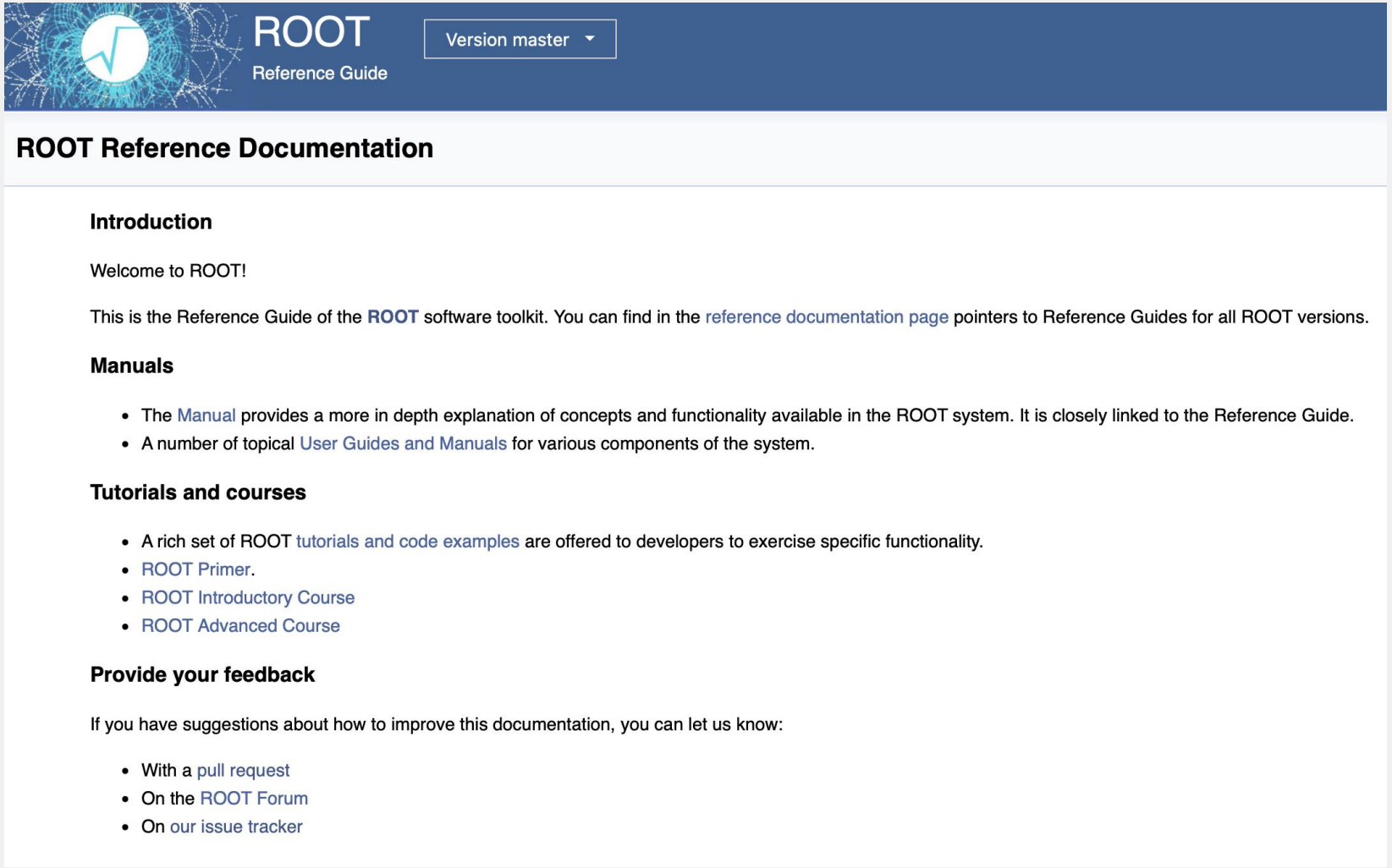
- Documentation should be written to minimize user↔expert interaction
- Users should be able to build analyses by themselves without expert input
- Ideally, experts should only be reviewing (at regular enough intervals including possibly at the beginning)
- In reality, this will never happen, but it is a good goal to drive documentation development
- When we receive questions, we should review whether the docs could be improved

How might we want to structure the docs with all this mind?

Let's explore some examples…

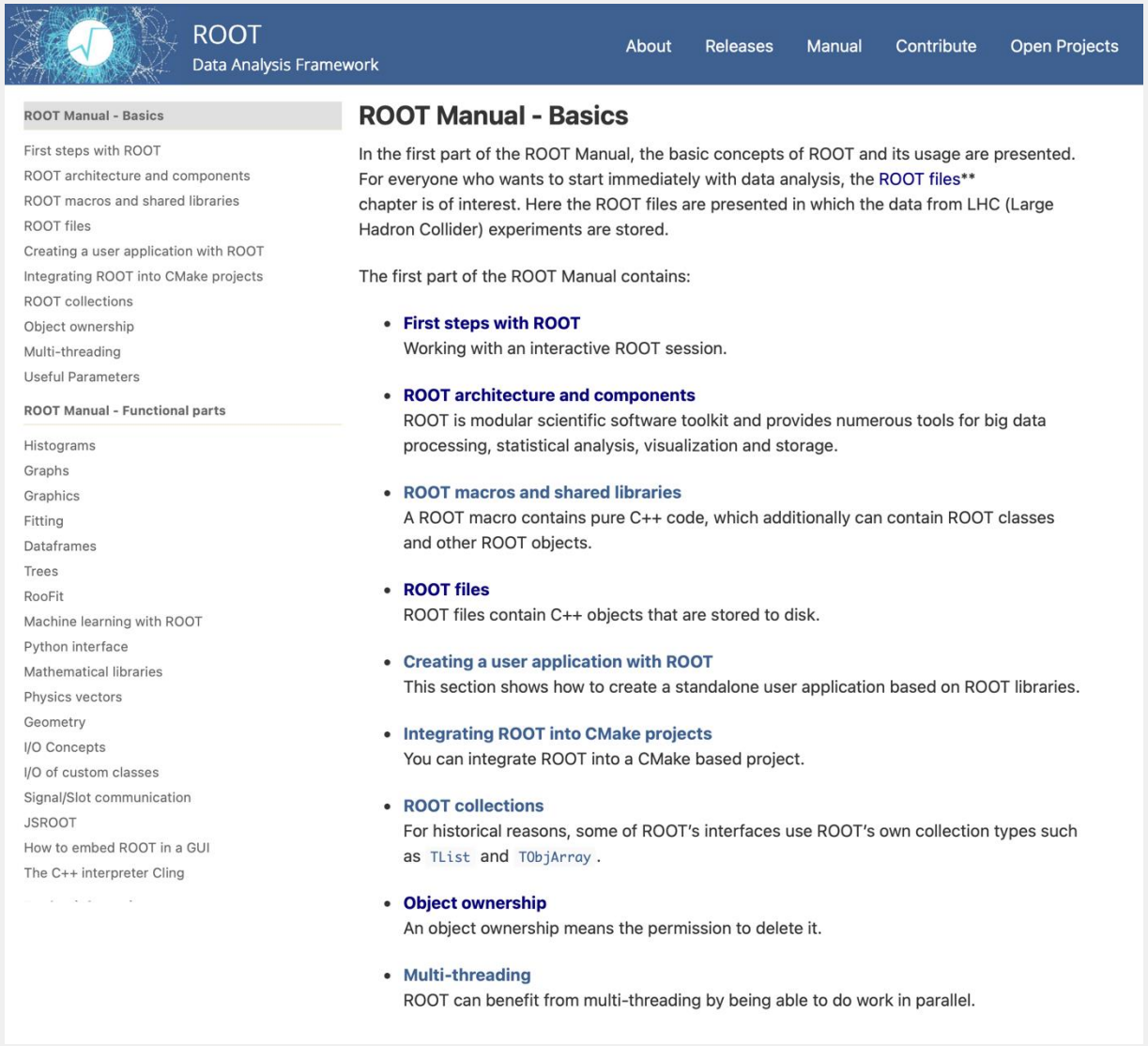
Examples: ROOT – landing page

https://root.cern/doc/master/



Examples: ROOT – the “manual”

https://root.cern/manual/basics/



ROOT
Data Analysis Framework

About Releases Manual Contribute Open Projects

ROOT Manual - Basics

First steps with ROOT
ROOT architecture and components
ROOT macros and shared libraries
ROOT files
Creating a user application with ROOT
Integrating ROOT into CMake projects
ROOT collections
Object ownership
Multi-threading
Useful Parameters

ROOT Manual - Functional parts

Histograms
Graphs
Graphics
Fitting
Dataframes
Trees
RooFit
Machine learning with ROOT
Python interface
Mathematical libraries
Physics vectors
Geometry
I/O Concepts
I/O of custom classes
Signal/Slot communication
JSROOT
How to embed ROOT in a GUI
The C++ interpreter Cling

ROOT Manual - Basics

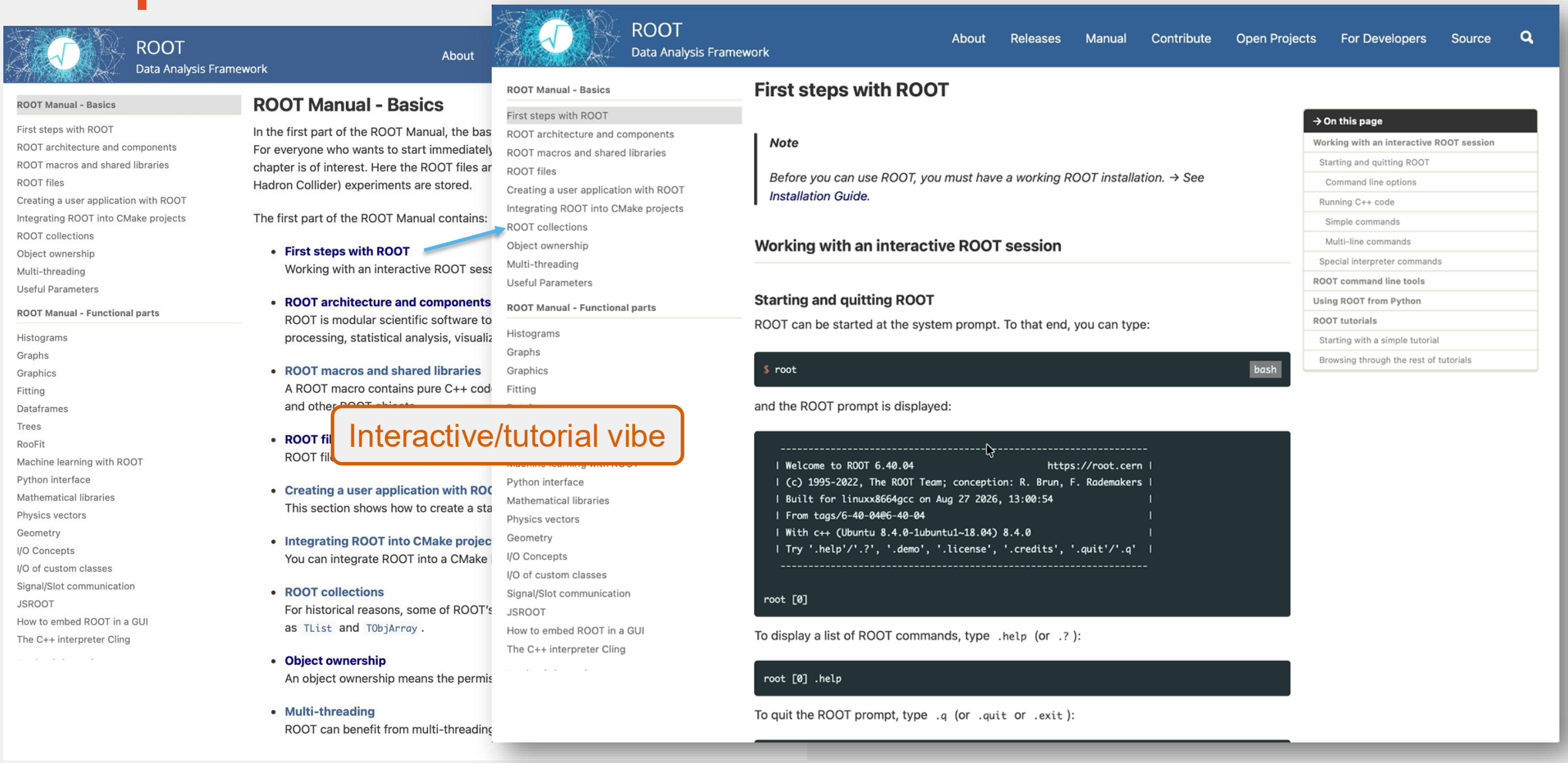
In the first part of the ROOT Manual, the basic concepts of ROOT and its usage are presented. For everyone who wants to start immediately with data analysis, the ROOT files** chapter is of interest. Here the ROOT files are presented in which the data from LHC (Large Hadron Collider) experiments are stored.

The first part of the ROOT Manual contains:

- **First steps with ROOT**
  Working with an interactive ROOT session.
- **ROOT architecture and components**
  ROOT is modular scientific software toolkit and provides numerous tools for big data processing, statistical analysis, visualization and storage.
- **ROOT macros and shared libraries**
  A ROOT macro contains pure C++ code, which additionally can contain ROOT classes and other ROOT objects.
- **ROOT files**
  ROOT files contain C++ objects that are stored to disk.
- **Creating a user application with ROOT**
  This section shows how to create a standalone user application based on ROOT libraries.
- **Integrating ROOT into CMake projects**
  You can integrate ROOT into a CMake based project.
- **ROOT collections**
  For historical reasons, some of ROOT’s interfaces use ROOT’s own collection types such as `TList` and `TObjArray`.
- **Object ownership**
  An object ownership means the permission to delete it.
- **Multi-threading**
  ROOT can benefit from multi-threading by being able to do work in parallel.

https://root.cern/manual/first_steps_with_root/

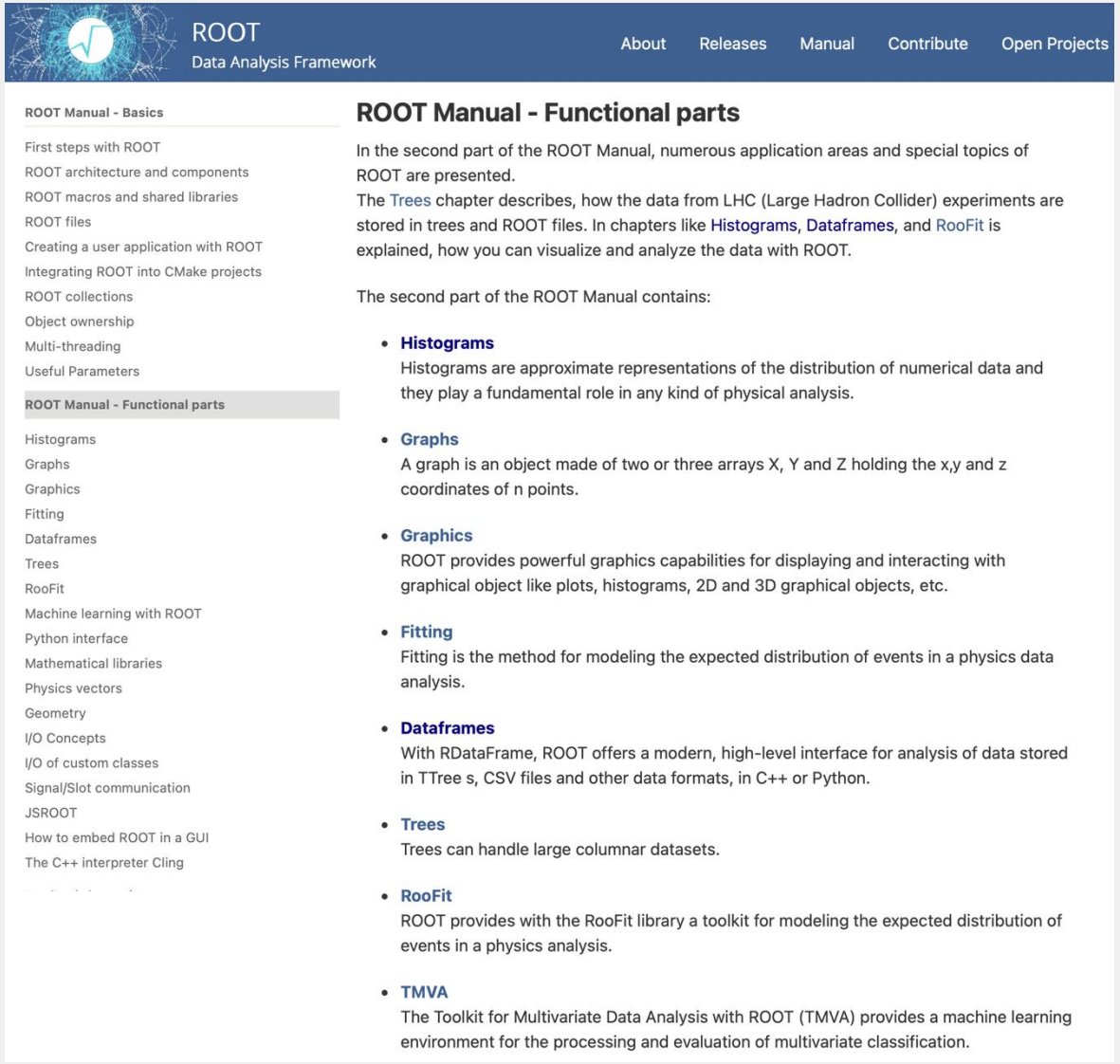
Examples: ROOT – the "manual"



IMPERIAL

Examples: ROOT – the “manual”

https://root.cern/manual/functional_parts/



ROOT
Data Analysis Framework

About Releases Manual Contribute Open Projects

ROOT Manual - Basics

First steps with ROOT
ROOT architecture and components
ROOT macros and shared libraries
ROOT files
Creating a user application with ROOT
Integrating ROOT into CMake projects
ROOT collections
Object ownership
Multi-threading
Useful Parameters

ROOT Manual - Functional parts

Histograms
Graphs
Graphics
Fitting
Dataframes
Trees
RooFit
Machine learning with ROOT
Python interface
Mathematical libraries
Physics vectors
Geometry
I/O Concepts
I/O of custom classes
Signal/Slot communication
JSROOT
How to embed ROOT in a GUI
The C++ interpreter Cling

ROOT Manual - Functional parts

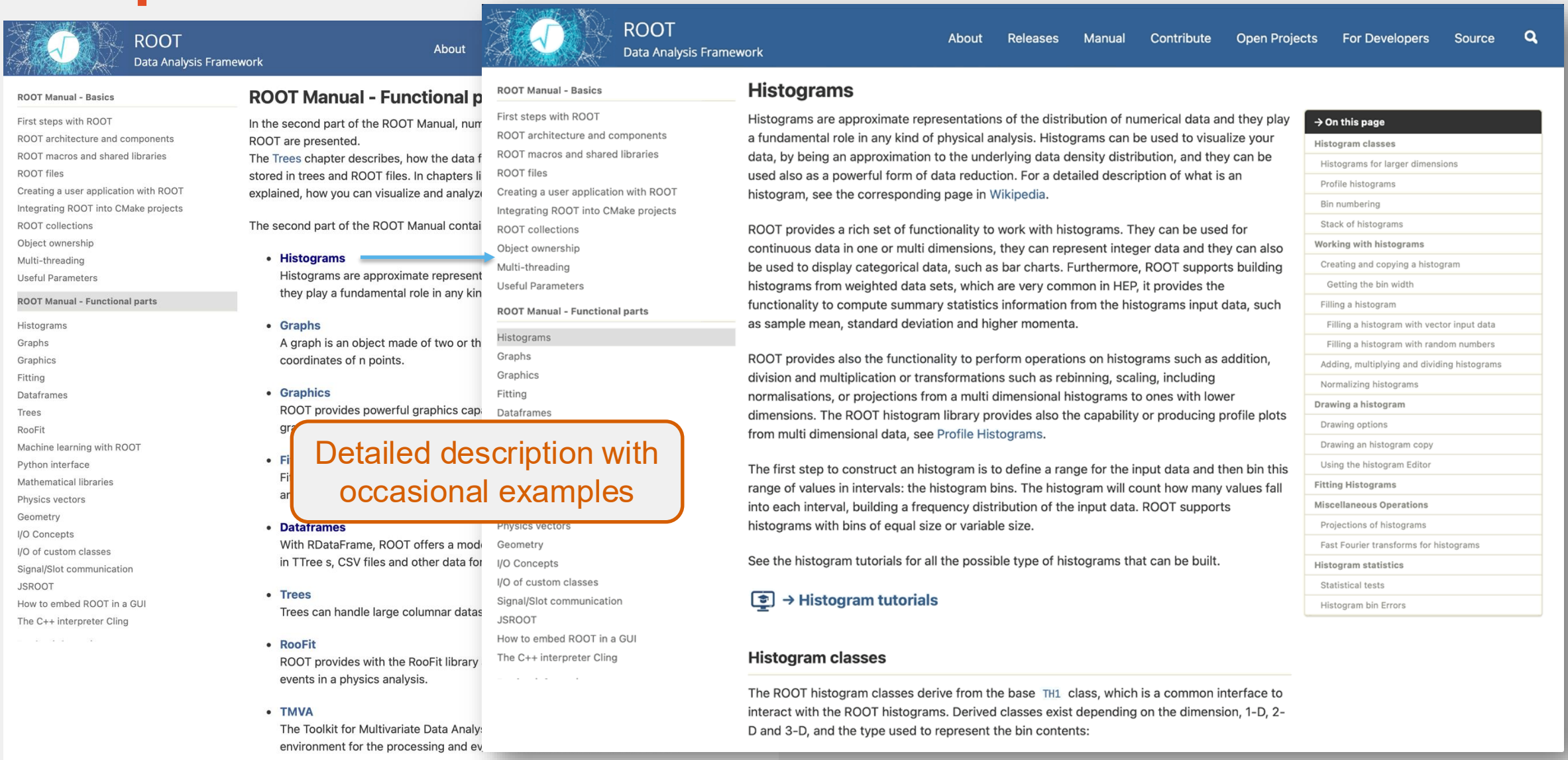
In the second part of the ROOT Manual, numerous application areas and special topics of ROOT are presented.
The Trees chapter describes, how the data from LHC (Large Hadron Collider) experiments are stored in trees and ROOT files. In chapters like Histograms, Dataframes, and RooFit is explained, how you can visualize and analyze the data with ROOT.

The second part of the ROOT Manual contains:

- **Histograms**
  Histograms are approximate representations of the distribution of numerical data and they play a fundamental role in any kind of physical analysis.
- **Graphs**
  A graph is an object made of two or three arrays X, Y and Z holding the x,y and z coordinates of n points.
- **Graphics**
  ROOT provides powerful graphics capabilities for displaying and interacting with graphical object like plots, histograms, 2D and 3D graphical objects, etc.
- **Fitting**
  Fitting is the method for modeling the expected distribution of events in a physics data analysis.
- **Dataframes**
  With RDataFrame, ROOT offers a modern, high-level interface for analysis of data stored in TTree s, CSV files and other data formats, in C++ or Python.
- **Trees**
  Trees can handle large columnar datasets.
- **RooFit**
  ROOT provides with the RooFit library a toolkit for modeling the expected distribution of events in a physics analysis.
- **TMVA**
  The Toolkit for Multivariate Data Analysis with ROOT (TMVA) provides a machine learning environment for the processing and evaluation of multivariate classification.

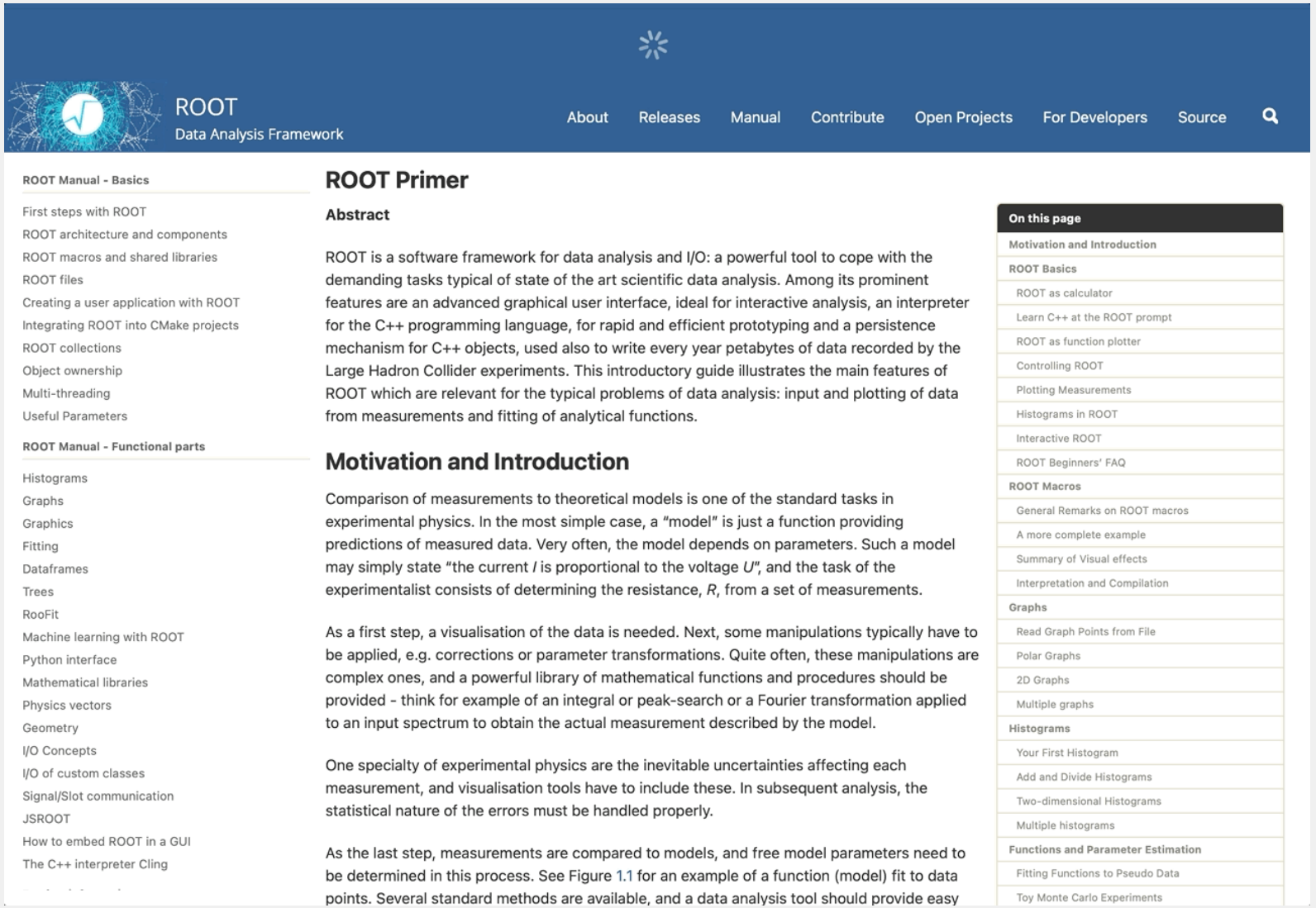
Examples: ROOT – the "manual"

https://root.cern/manual/histograms/



Examples: ROOT – “primer”

https://root.cern/primer/



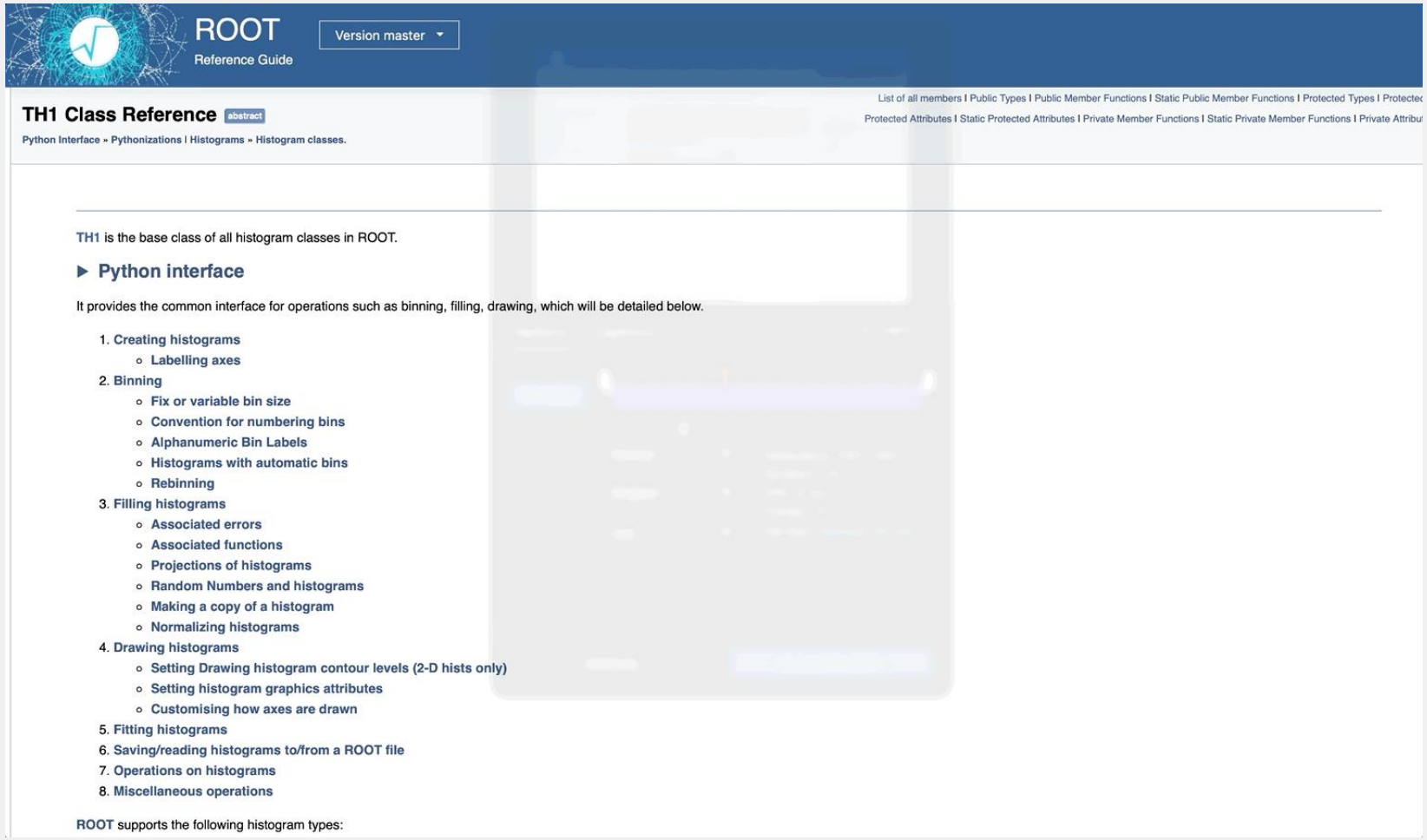
One very long description with plenty of code to describe most of what a user needs

Somewhat redundant with the “manual” but includes new material like “motivation & introduction"

IMPERIAL

Examples: ROOT – API reference

https://root.cern.ch/doc/master/classTH1.html



ROOT
Reference Guide
Version master

TH1 Class Reference abstract

List of all members | Public Types | Public Member Functions | Static Public Member Functions | Protected Types | Protected
Protected Attributes | Static Protected Attributes | Private Member Functions | Static Private Member Functions | Private Attribu

Python Interface » Pythonizations | Histograms » Histogram classes.

TH1 is the base class of all histogram classes in ROOT.

▶ **Python interface**

It provides the common interface for operations such as binning, filling, drawing, which will be detailed below.

1. Creating histograms
   - Labelling axes
2. Binning
   - Fix or variable bin size
   - Convention for numbering bins
   - Alphanumeric Bin Labels
   - Histograms with automatic bins
   - Rebinning
3. Filling histograms
   - Associated errors
   - Associated functions
   - Projections of histograms
   - Random Numbers and histograms
   - Making a copy of a histogram
   - Normalizing histograms
4. Drawing histograms
   - Setting Drawing histogram contour levels (2-D hists only)
   - Setting histogram graphics attributes
   - Customising how axes are drawn
5. Fitting histograms
6. Saving/reading histograms to/from a ROOT file
7. Operations on histograms
8. Miscellaneous operations

ROOT supports the following histogram types:

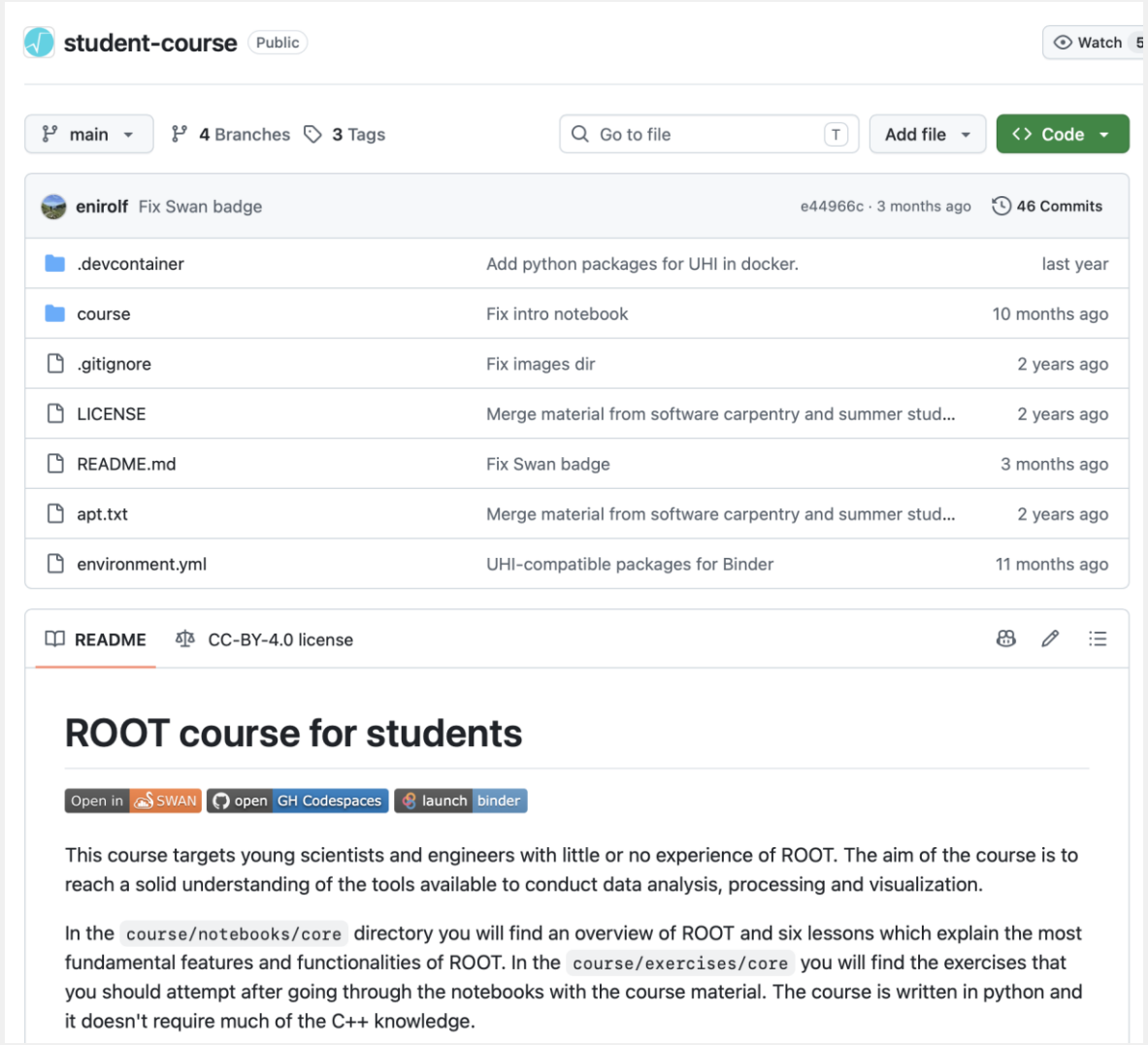
Combination of description & examples + pure API documentation

Description must be somewhat redundant with “manual” and “primer”

IMPERIAL

Examples: ROOT – introductory course

https://github.com/root-project/student-course/tree/25.07



student-course Public

Watch 5

main · 4 Branches · 3 Tags · Go to file T · Add file · Code

enirolf Fix Swan badge — e44966c · 3 months ago — 46 Commits

.devcontainer	Add python packages for UHI in docker.	last year
course	Fix intro notebook	10 months ago
.gitignore	Fix images dir	2 years ago
LICENSE	Merge material from software carpentry and summer stud...	2 years ago
README.md	Fix Swan badge	3 months ago
apt.txt	Merge material from software carpentry and summer stud...	2 years ago
environment.yml	UHI-compatible packages for Binder	11 months ago

README · CC-BY-4.0 license

ROOT course for students

Open in SWAN · open GH Codespaces · launch binder

This course targets young scientists and engineers with little or no experience of ROOT. The aim of the course is to reach a solid understanding of the tools available to conduct data analysis, processing and visualization.

In the `course/notebooks/core` directory you will find an overview of ROOT and six lessons which explain the most fundamental features and functionalities of ROOT. In the `course/exercises/core` you will find the exercises that you should attempt after going through the notebooks with the course material. The course is written in python and it doesn't require much of the C++ knowledge.

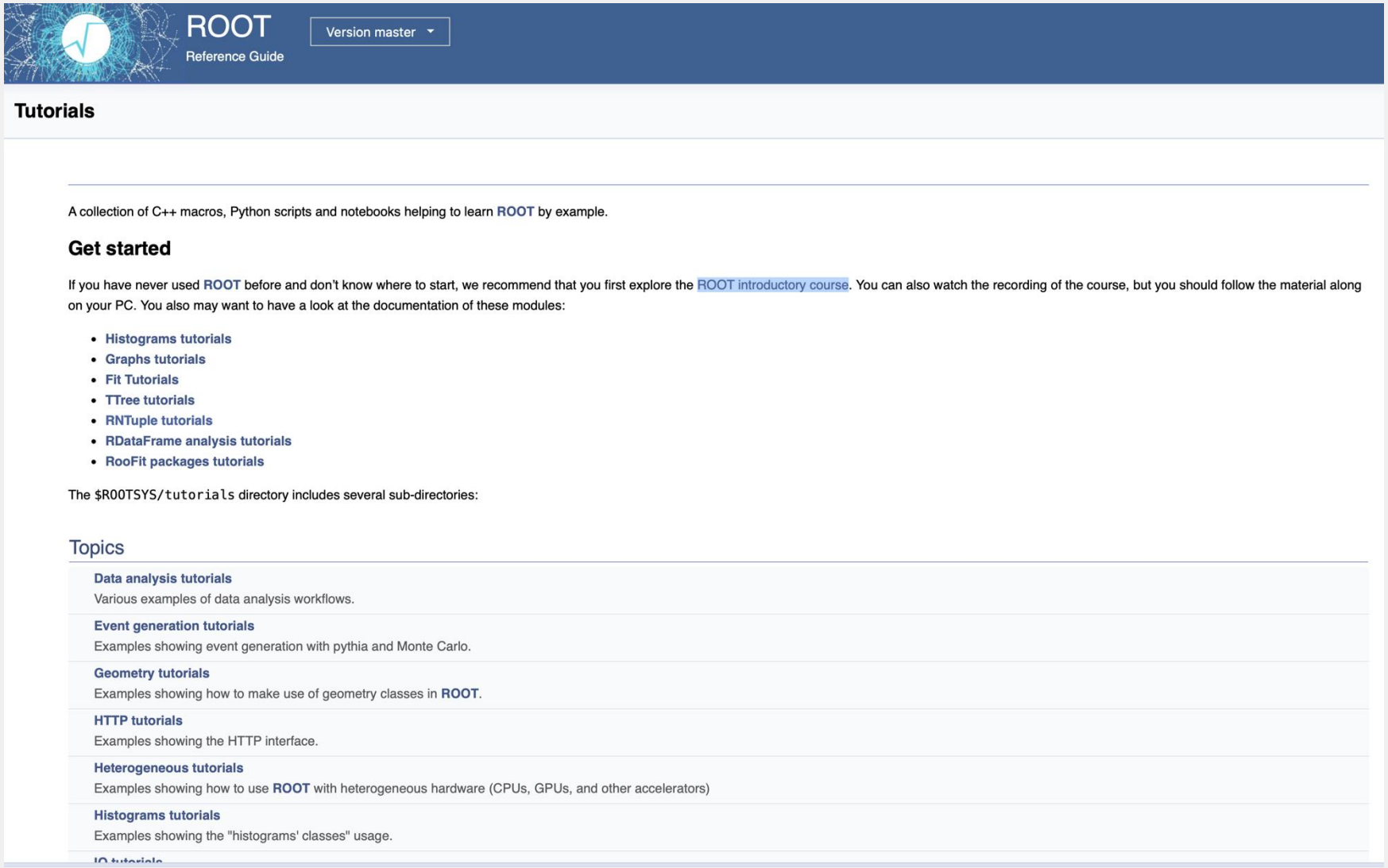
Notebook (pyroot) based introduction

Result of an in-person course

Avaliable on binder -> very easy to get started and very interactive

Examples: ROOT – tutorials

https://root.cern/doc/master/group__Tutorials.html



ROOT
Reference Guide
Version master

Tutorials

A collection of C++ macros, Python scripts and notebooks helping to learn ROOT by example.

Get started

If you have never used ROOT before and don't know where to start, we recommend that you first explore the ROOT introductory course. You can also watch the recording of the course, but you should follow the material along on your PC. You also may want to have a look at the documentation of these modules:

- Histograms tutorials
- Graphs tutorials
- Fit Tutorials
- TTree tutorials
- RNTuple tutorials
- RDataFrame analysis tutorials
- RooFit packages tutorials

The `$ROOTSYS/tutorials` directory includes several sub-directories:

Topics

Data analysis tutorials
Various examples of data analysis workflows.

Event generation tutorials
Examples showing event generation with pythia and Monte Carlo.

Geometry tutorials
Examples showing how to make use of geometry classes in ROOT.

HTTP tutorials
Examples showing the HTTP interface.

Heterogeneous tutorials
Examples showing how to use ROOT with heterogeneous hardware (CPUs, GPUs, and other accelerators)

Histograms tutorials
Examples showing the "histograms' classes" usage.

IO tutorials

IMPERIAL

Examples: ROOT – tutorials

https://root.cern/doc/master/group__Tutorials.html

ROOT Reference Guide — Version master

Tutorials

A collection of C++ macros, Python scripts and notebooks helping to learn RO

Get started

If you have never used ROOT before and don't know where to start, we recomm
on your PC. You also may want to have a look at the documentation of these modules:

- Histograms tutorials
- Graphs tutorials
- Fit Tutorials
- TTree tutorials
- RNTuple tutorials
- RDataFrame analysis tutorials
- RooFit packages tutorials

The `$ROOTSYS/tutorials` directory includes several sub-directories:

Topics

Data analysis tutorials
Various examples of data analysis workflows.

Event generation tutorials
Examples showing event generation with pythia and Monte Carlo.

Geometry tutorials
Examples showing how to make use of geometry classes in **ROOT**.

HTTP tutorials
Examples showing the HTTP interface.

Heterogeneous tutorials
Examples showing how to use **ROOT** with heterogeneous hardware (CPUs, GPUs, and other accelerators)

Histograms tutorials
Examples showing the "histograms' classes" usage.

IO tutorials

My personal favourite way of learning

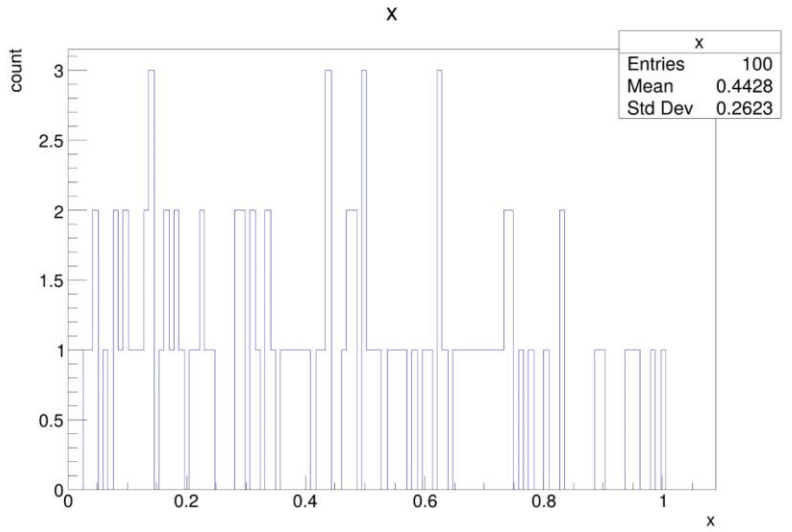
Lots of real-life examples & gives impression of best-practices

Detailed Description

View Notebook | Open in SWAN

Simple RDataFrame example in C++.

This tutorial shows a minimal example of RDataFrame. It starts without input data, generates a new column x with random numbers, and finally draws a histogram for x.




```
void df000_simple()
{
   // Create a data frame with 100 rows
   ROOT::RDataFrame rdf(100);

   // Define a new column `x` that contains random numbers
   auto rdf_x = rdf.Define("x", []{ return gRandom->Rndm(); });

   // Create a histogram from `x`
   auto h = rdf_x.Histo1D("x");

   // At the end of this function, the histogram pointed to by `h` will be deleted.
   // Draw a copy of the histogram object instead:
   h->DrawClone();
}
```

Date
September 2021

Examples: ROOT – my takeaways

- Wow! ROOT has a lot of different forms of documentation
- Must be very difficult to maintain
- But gives users the option to choose whichever form suits their learning style best
  - Chosen options can also be problem specific
  - If I need to remember what a particular method is called -> API reference
  - If I want to know how RDataFrames are supposed to be used -> RDataFrame tutorials

- MaCh3 is not the same as ROOT
  - ROOT is primarily a library
  - MaCh3 is both a library and a package (in the sense that it provides executables)
  - I suspect we may wish to have a combination of API reference, user guide (manual-like), and tutorials

Examples: CMS Combine

- Combine is CMS's primary statistical inference tool – over 90% of analyses use it
- Features include:
  - General* model and likelihood building
  - Maximum likelihood fits + Bayesian posterior calculation (rarely used)
  - Plenty of statistical inference techniques
- Majority of users never write a piece of code to use Combine
  - Selections & impact of systematics derived before hand
  - Inputs like histograms or analytical functions handed to Combine and it does the rest
  - Many users start without input from Combine experts & get approval by an expert at relevant stage in analysis review
    - Help is always available if needed

*arguably not as general as MaCh3, but general enough for majority of LHC analyses

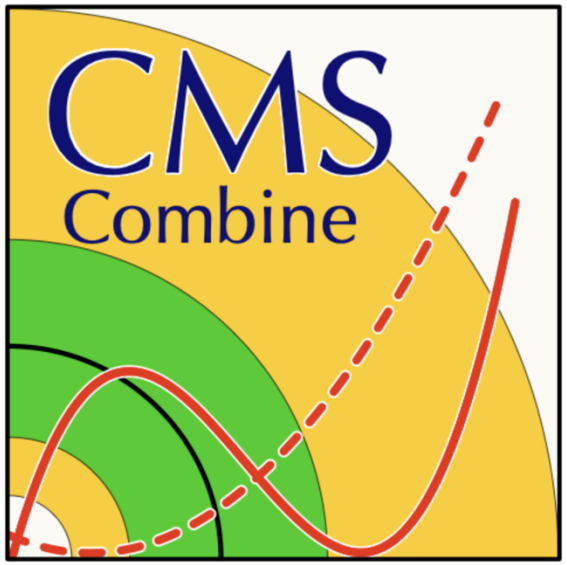
Examples: CMS Combine – landing page

Link

Combine v11.1.0 | Search | GitHub v11.1.0 ☆88 ⑂436

Home | What Combine Does | Setting Up the Analysis | Running Combine | Tutorials | In-Browser Tools | Links & FAQ

Introduction



These pages document the RooStats / RooFit - based software tool used for statistical analysis within the CMS experiment - COMBINE. Note that while this tool was originally developed in the Higgs Physics Analysis Group (PAG), its usage is now widespread within CMS.

COMBINE provides a command-line interface to many different statistical techniques, available inside RooFit/RooStats, that are used widely inside CMS.

The package exists on GitHub under https://github.com/cms-analysis/HiggsAnalysis-CombinedLimit

For more information about Git, GitHub and its usage in CMS, see http://cms-sw.github.io/cmssw/faq.html

The code can be checked out from GitHub and compiled on top of a CMSSW release that includes a recent RooFit/RooStats, or via standalone compilation without CMSSW dependencies. See the instructions for installation of COMBINE below.

Table of contents

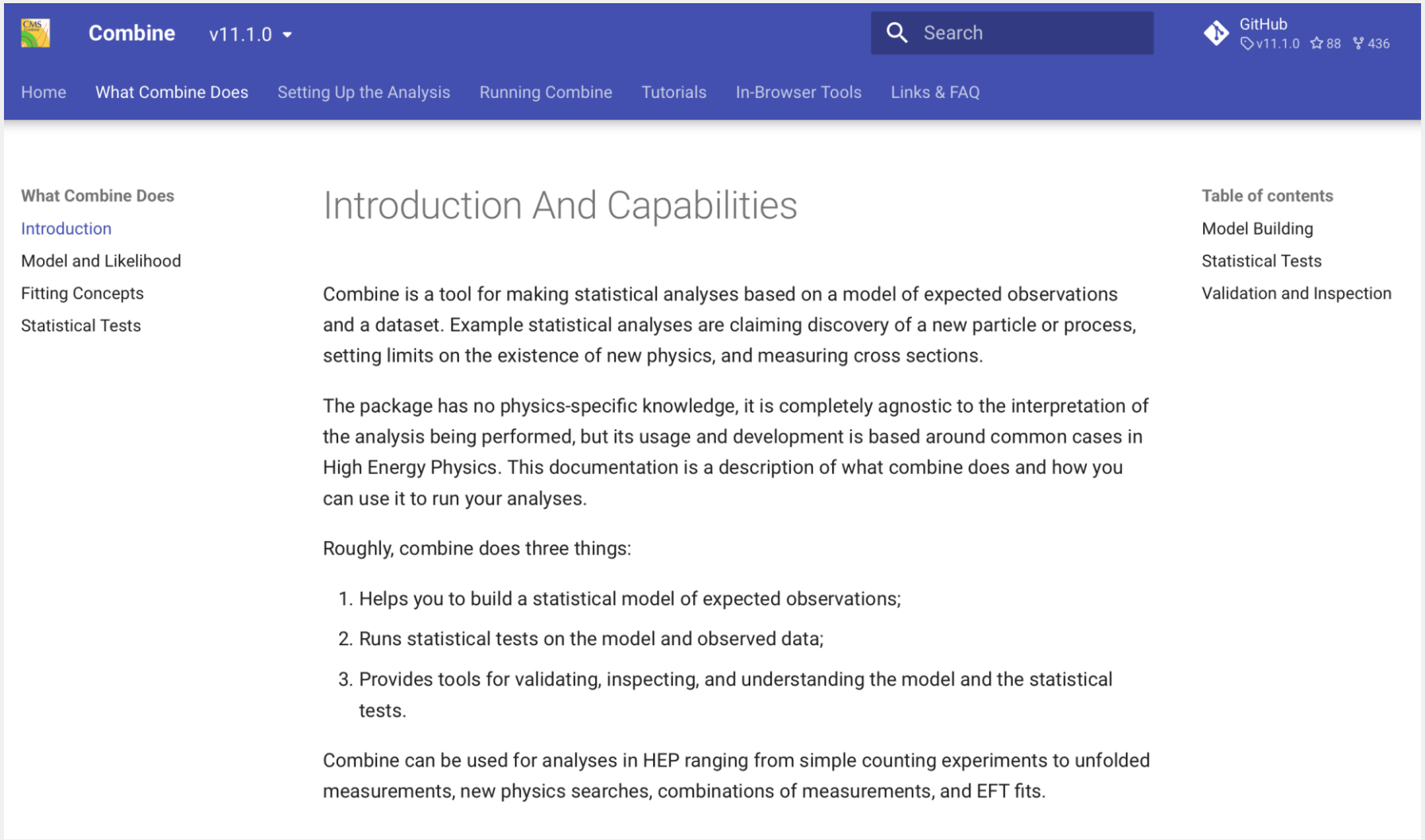
Installation instructions
Within CMSSW (recommended for CMS users)
Combine v11 - recommended version
Legacy versions
Combine v9
Combine v8: CMSSW_10_2_X release series
SLC6/CC7 release CMSSW_8_1_X
Outside of CMSSW
Standalone compilation
Standalone compilation with LCG
Standalone compilation with conda (CMake-based)
Standalone compilation with pixi (CMake-based)
Pre-compiled image with CVMFS dependency
Pre-compiled versions in Docker
Standalone compilation with CernVM
What has changed between tags?
For developers
CombineHarvester/CombineTo…

Installation instructions

Installation instructions and recommended versions can be found below. Since v9.0.0, the versioning follows the semantic versioning 2.0.0 standard. Earlier versions are not guaranteed to follow the standard.

Examples: CMS Combine – What it does

Link



Combine v11.1.0 | Search | GitHub v11.1.0 ☆88 436

Home | What Combine Does | Setting Up the Analysis | Running Combine | Tutorials | In-Browser Tools | Links & FAQ

What Combine Does
- Introduction
- Model and Likelihood
- Fitting Concepts
- Statistical Tests

Introduction And Capabilities

Combine is a tool for making statistical analyses based on a model of expected observations and a dataset. Example statistical analyses are claiming discovery of a new particle or process, setting limits on the existence of new physics, and measuring cross sections.

The package has no physics-specific knowledge, it is completely agnostic to the interpretation of the analysis being performed, but its usage and development is based around common cases in High Energy Physics. This documentation is a description of what combine does and how you can use it to run your analyses.

Roughly, combine does three things:

1. Helps you to build a statistical model of expected observations;
2. Runs statistical tests on the model and observed data;
3. Provides tools for validating, inspecting, and understanding the model and the statistical tests.

Combine can be used for analyses in HEP ranging from simple counting experiments to unfolded measurements, new physics searches, combinations of measurements, and EFT fits.

Table of contents
- Model Building
- Statistical Tests
- Validation and Inspection

Starts with concepts

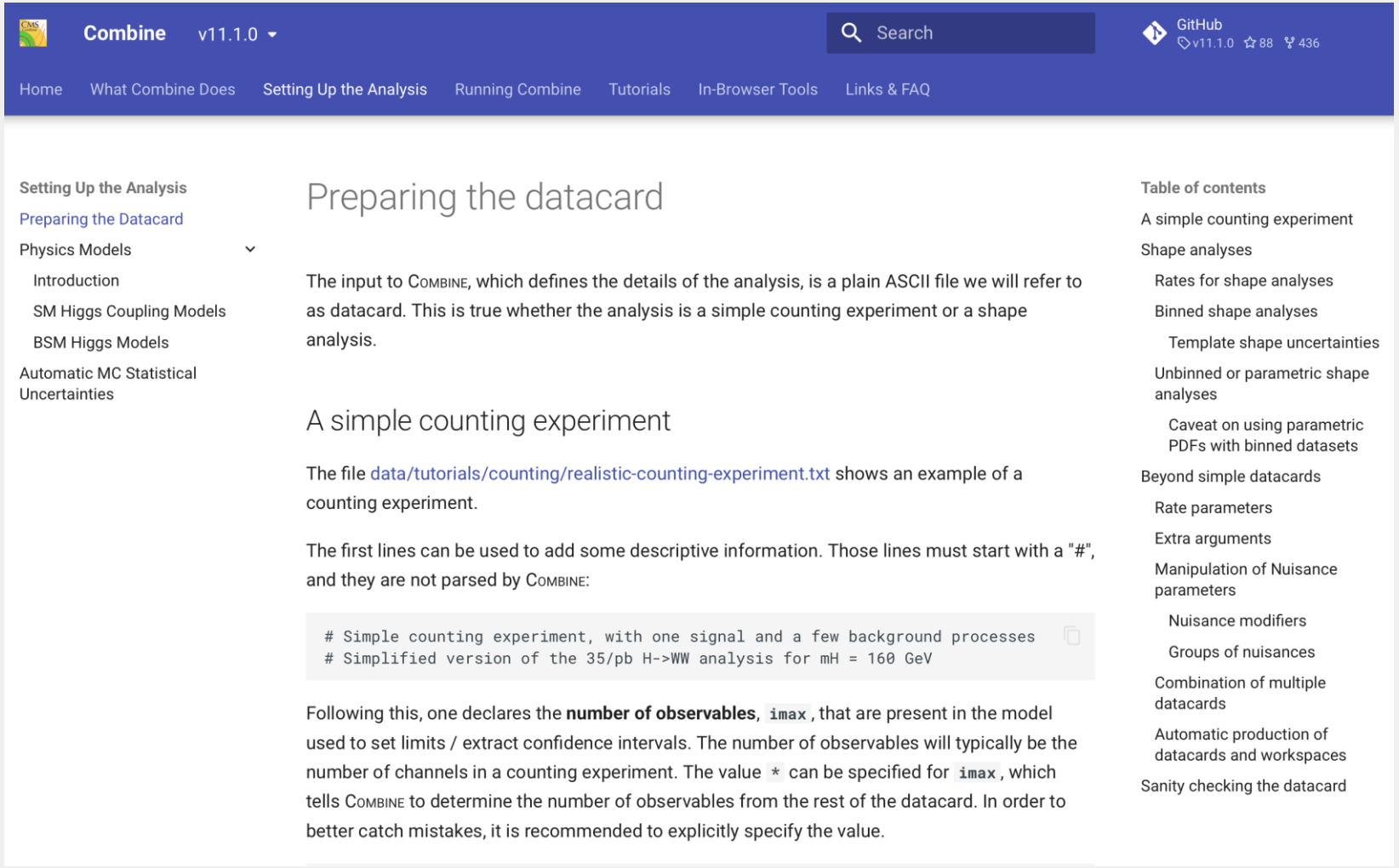
Some background in statistics

What types of likelihoods and fits it supports

All implementation agnostic

Examples: CMS Combine – Setting up the Analysis

Link



Combine v11.1.0 | Search | GitHub v11.1.0 ☆88 436

Home | What Combine Does | Setting Up the Analysis | Running Combine | Tutorials | In-Browser Tools | Links & FAQ

Setting Up the Analysis
- Preparing the Datacard
- Physics Models
  - Introduction
  - SM Higgs Coupling Models
  - BSM Higgs Models
- Automatic MC Statistical Uncertainties

Preparing the datacard

The input to COMBINE, which defines the details of the analysis, is a plain ASCII file we will refer to as datacard. This is true whether the analysis is a simple counting experiment or a shape analysis.

A simple counting experiment

The file data/tutorials/counting/realistic-counting-experiment.txt shows an example of a counting experiment.

The first lines can be used to add some descriptive information. Those lines must start with a "#", and they are not parsed by COMBINE:

```
# Simple counting experiment, with one signal and a few background processes
# Simplified version of the 35/pb H->WW analysis for mH = 160 GeV
```

Following this, one declares the **number of observables**, `imax`, that are present in the model used to set limits / extract confidence intervals. The number of observables will typically be the number of channels in a counting experiment. The value `*` can be specified for `imax`, which tells COMBINE to determine the number of observables from the rest of the datacard. In order to better catch mistakes, it is recommended to explicitly specify the value.

Table of contents
- A simple counting experiment
- Shape analyses
  - Rates for shape analyses
  - Binned shape analyses
    - Template shape uncertainties
  - Unbinned or parametric shape analyses
    - Caveat on using parametric PDFs with binned datasets
- Beyond simple datacards
  - Rate parameters
  - Extra arguments
  - Manipulation of Nuisance parameters
    - Nuisance modifiers
    - Groups of nuisances
  - Combination of multiple datacards
  - Automatic production of datacards and workspaces
- Sanity checking the datacard

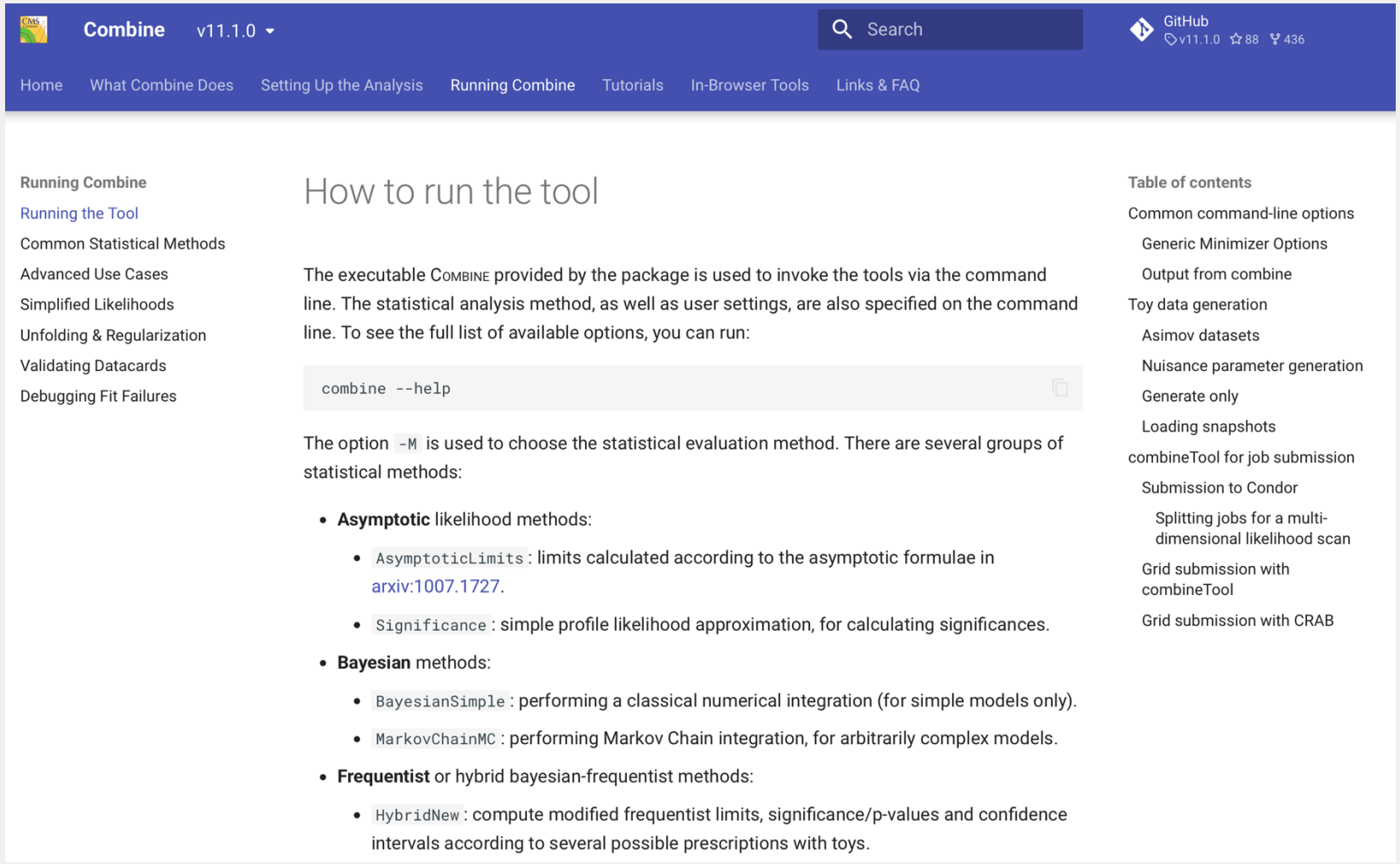
Description of how required inputs and how to configure them

Explained with examples but not interactive

Gradual increase in complexity

Examples: CMS Combine – Running Combine

Link



Combine v11.1.0 | Search | GitHub v11.1.0 ☆88 436

Home | What Combine Does | Setting Up the Analysis | Running Combine | Tutorials | In-Browser Tools | Links & FAQ

Running Combine
- Running the Tool
- Common Statistical Methods
- Advanced Use Cases
- Simplified Likelihoods
- Unfolding & Regularization
- Validating Datacards
- Debugging Fit Failures

How to run the tool

The executable COMBINE provided by the package is used to invoke the tools via the command line. The statistical analysis method, as well as user settings, are also specified on the command line. To see the full list of available options, you can run:

```
combine --help
```

The option `-M` is used to choose the statistical evaluation method. There are several groups of statistical methods:

- **Asymptotic** likelihood methods:
  - `AsymptoticLimits`: limits calculated according to the asymptotic formulae in arxiv:1007.1727.
  - `Significance`: simple profile likelihood approximation, for calculating significances.
- **Bayesian** methods:
  - `BayesianSimple`: performing a classical numerical integration (for simple models only).
  - `MarkovChainMC`: performing Markov Chain integration, for arbitrarily complex models.
- **Frequentist** or hybrid bayesian-frequentist methods:
  - `HybridNew`: compute modified frequentist limits, significance/p-values and confidence intervals according to several possible prescriptions with toys.

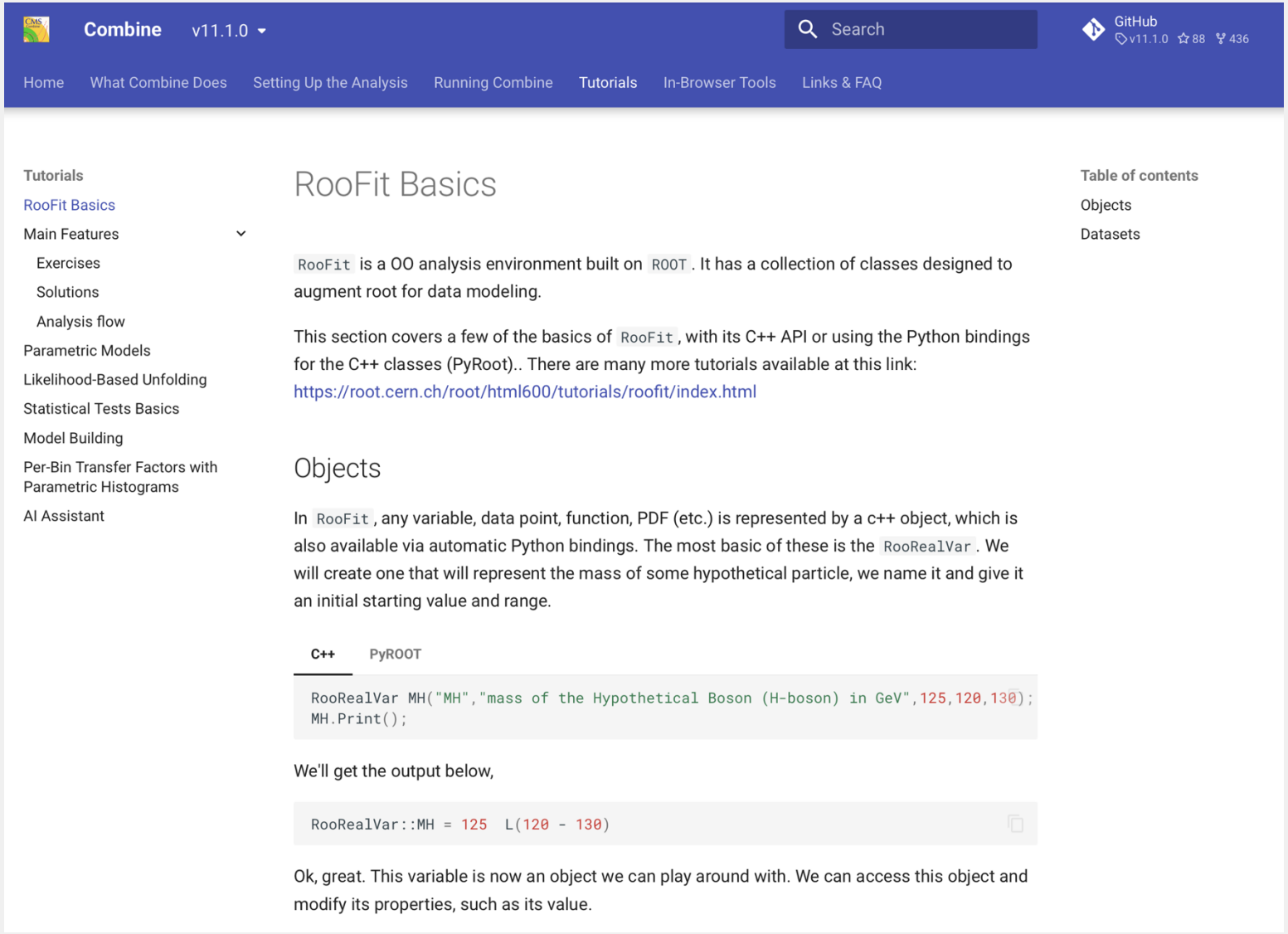
Table of contents
- Common command-line options
  - Generic Minimizer Options
  - Output from combine
- Toy data generation
  - Asimov datasets
  - Nuisance parameter generation
  - Generate only
  - Loading snapshots
- combineTool for job submission
  - Submission to Condor
    - Splitting jobs for a multi-dimensional likelihood scan
  - Grid submission with combineTool
  - Grid submission with CRAB

How to run the various fits and tests

Description of all the options

Examples: CMS Combine – Tutorials

Link



Interactive, follow-along tutorials

Covers main/commonly used features in Combine

IMPERIAL

Examples: CMS Combine – AI agent?!

Link

Combine v11.1.0 | Search | GitHub v11.1.0 ☆88 436

Home | What Combine Does | Setting Up the Analysis | Running Combine | Tutorials | In-Browser Tools | Links & FAQ

Tutorials
- RooFit Basics
- Main Features
  - Exercises
  - Solutions
  - Analysis flow
- Parametric Models
- Likelihood-Based Unfolding
- Statistical Tests Basics
- Model Building
- Per-Bin Transfer Factors with Parametric Histograms
- AI Assistant

An AI Assistant for Combine

This tutorial introduces an **AI assistant for Combine**: a chat-based agent that answers Combine questions *with citations* to the real documentation, paper, source code, and forum, and that can *run* Combine commands to reproduce, diagnose, and confirm results.

It is meant to lower the barrier to using Combine. You can ask, in plain language, "how do I set an upper limit on my signal strength?" or "why am I getting this error from `FitDiagnostics`?", while keeping the answers grounded in Combine's own sources rather than in a model's memory.

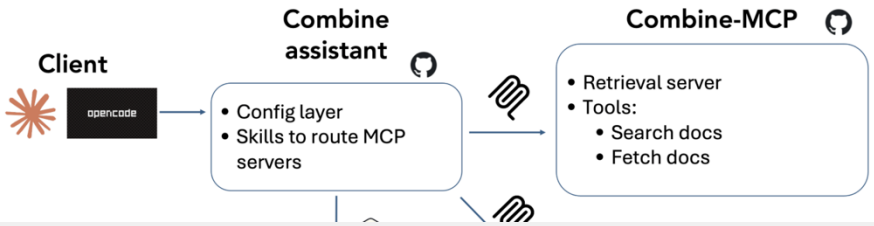
Experimental

This is a research prototype, not (yet) an official CMS tool. Always verify what it tells you against the linked sources, and review any command before you trust its output, especially for anything that affects a physics result.

Table of contents
- How it is built
- Running Combine: two paths
- Choosing a model
- Example 1: combagent on lxplus (main workflow)
- Example 2: Claude Code, locally
- Where to go next

How it is built

You do not need to understand the internals to use it, but a quick picture helps.

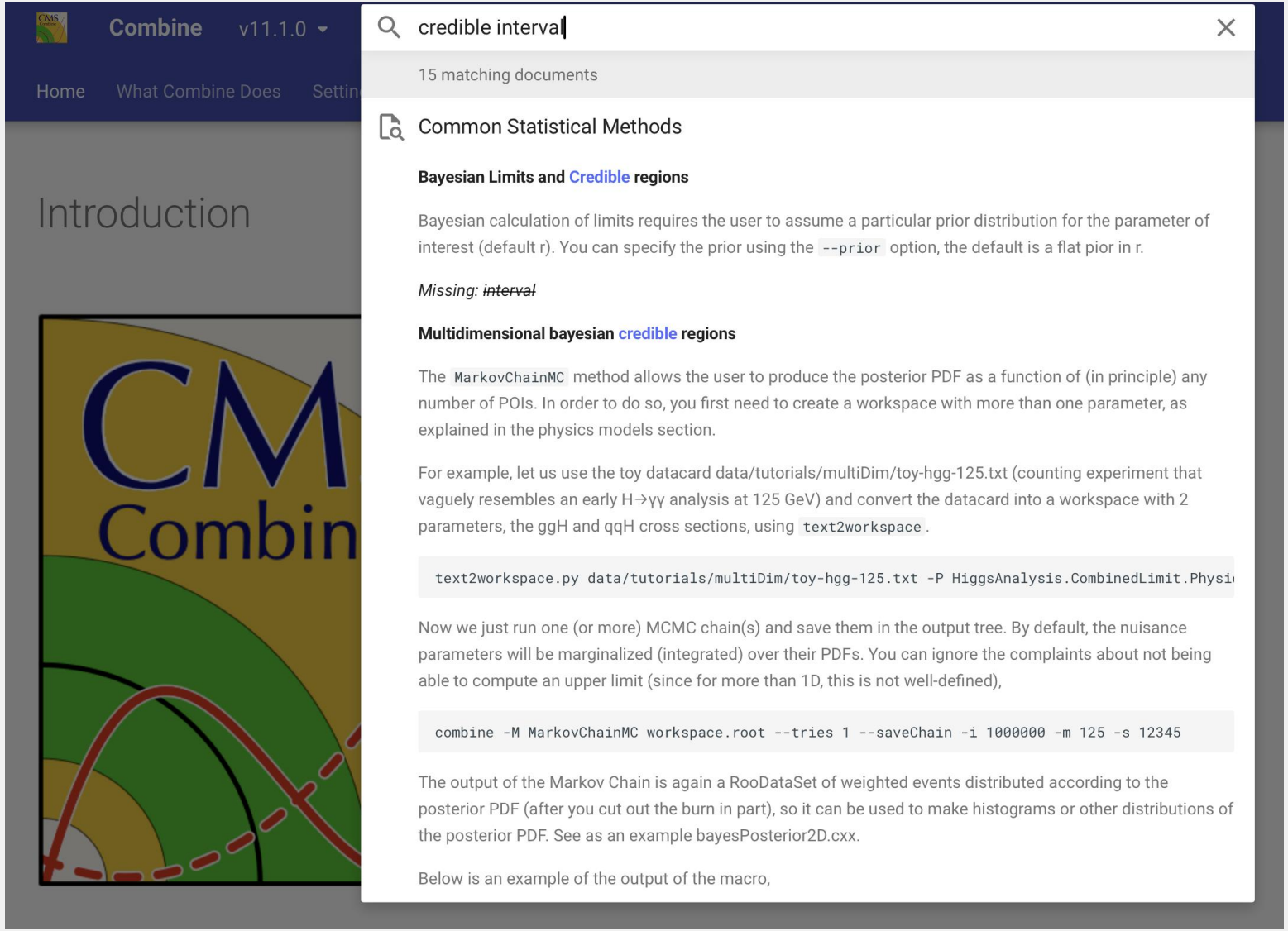


Still experimental… not sure how well it works

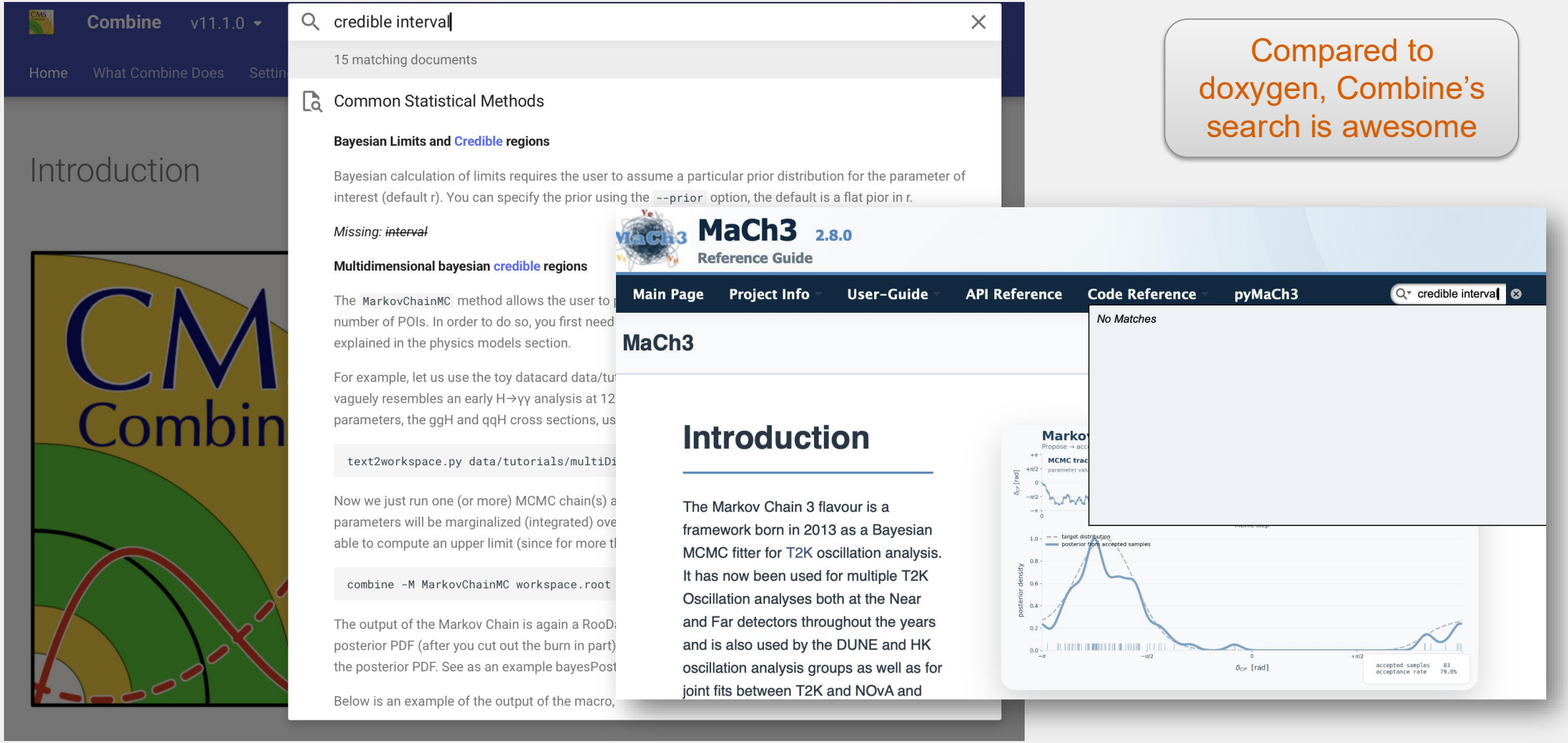
In principle, quicker replies to queries & reduces load on experts

Presumably requires strong documentation to work well

Examples: CMS Combine – search



Examples: CMS Combine – search



Compared to doxygen, Combine's search is awesome

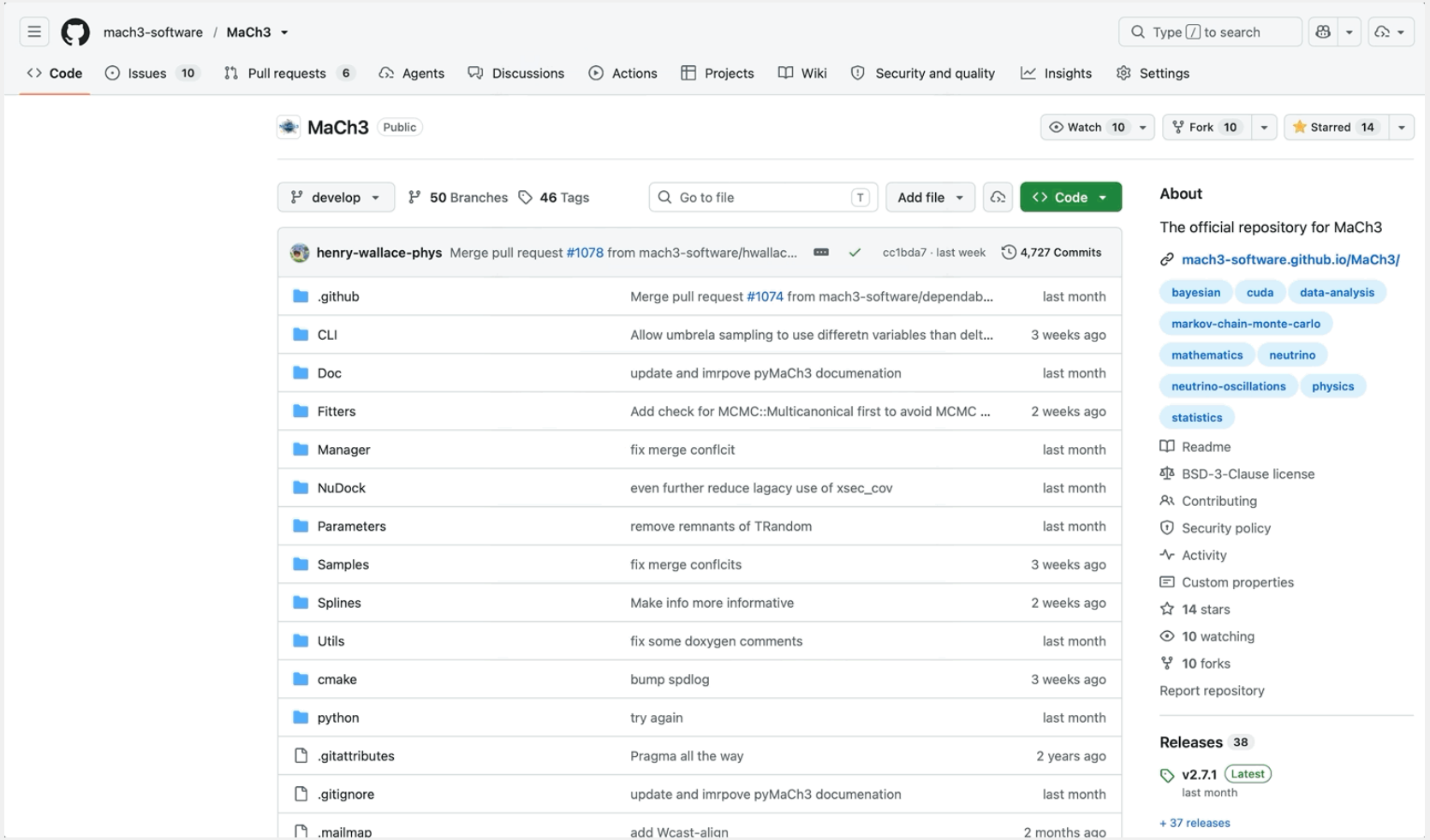
IMPERIAL

Examples: CMS Combine – my takeaways

- More streamlined than the ROOT docs
  - Less redundancy
- I like the combination of detail/description based docs & tutorials covering main topics
  - User can approach in whatever order they wish
- The separation of concepts, configuration, and running feels a natural way to write the description part (at least for Combine)
- It has no API reference 😱 which mustn't be great for developers but for users, makes no difference since no one is expected to write code for combine
  - Interact entirely via config files and CLI tools
- Searching is very effective

Examples: MaCh3 – GitHub README

https://github.com/mach3-software/MaCh3



Brief description

Citation instructions

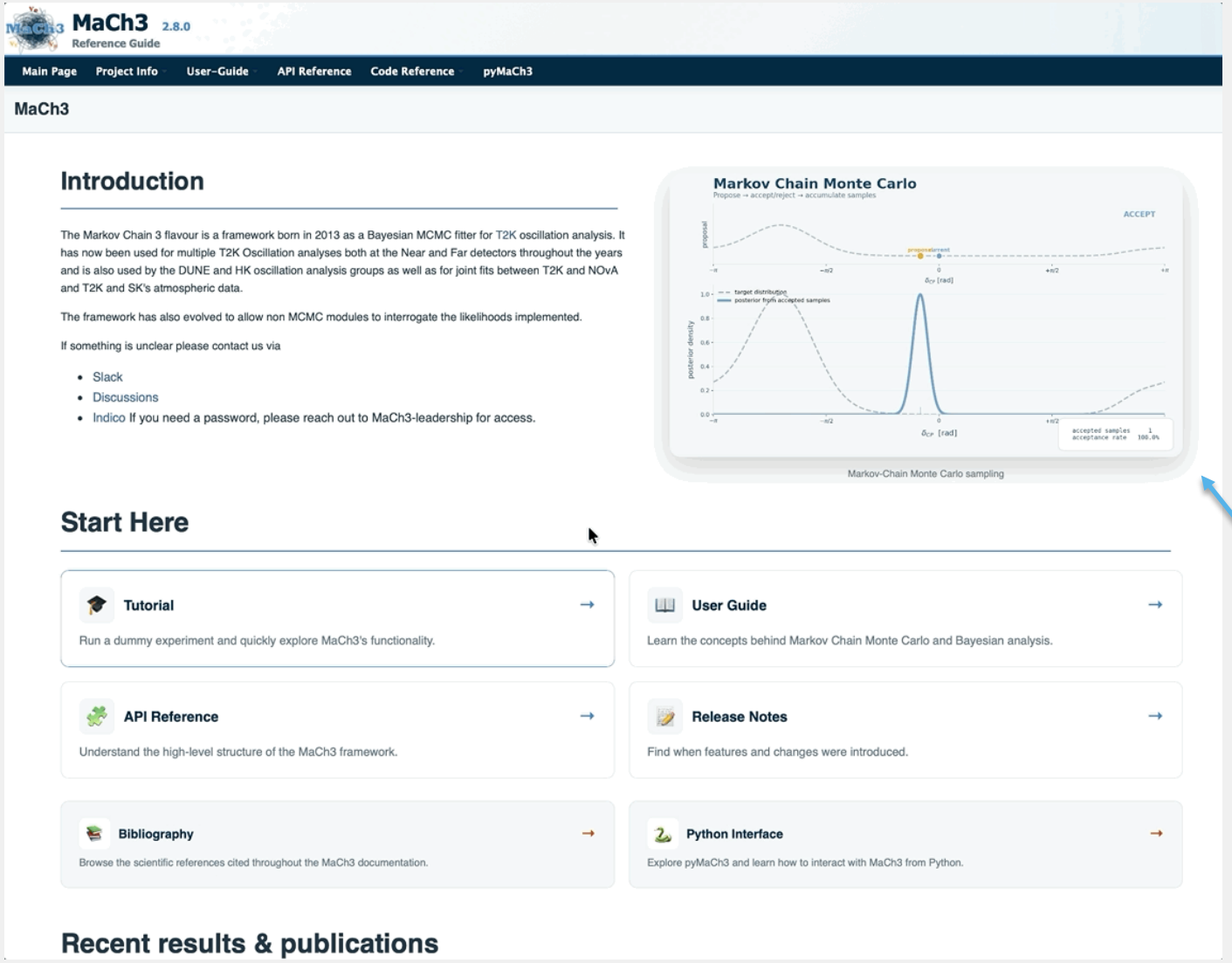
Links to help & docs

Installation instructions

Should we move this to the docs? 🤔

Examples: MaCh3 – Doxygen landing page

https://mach3-software.github.io/MaCh3/



Main categories:

- User guide
- Tutorial
- API reference

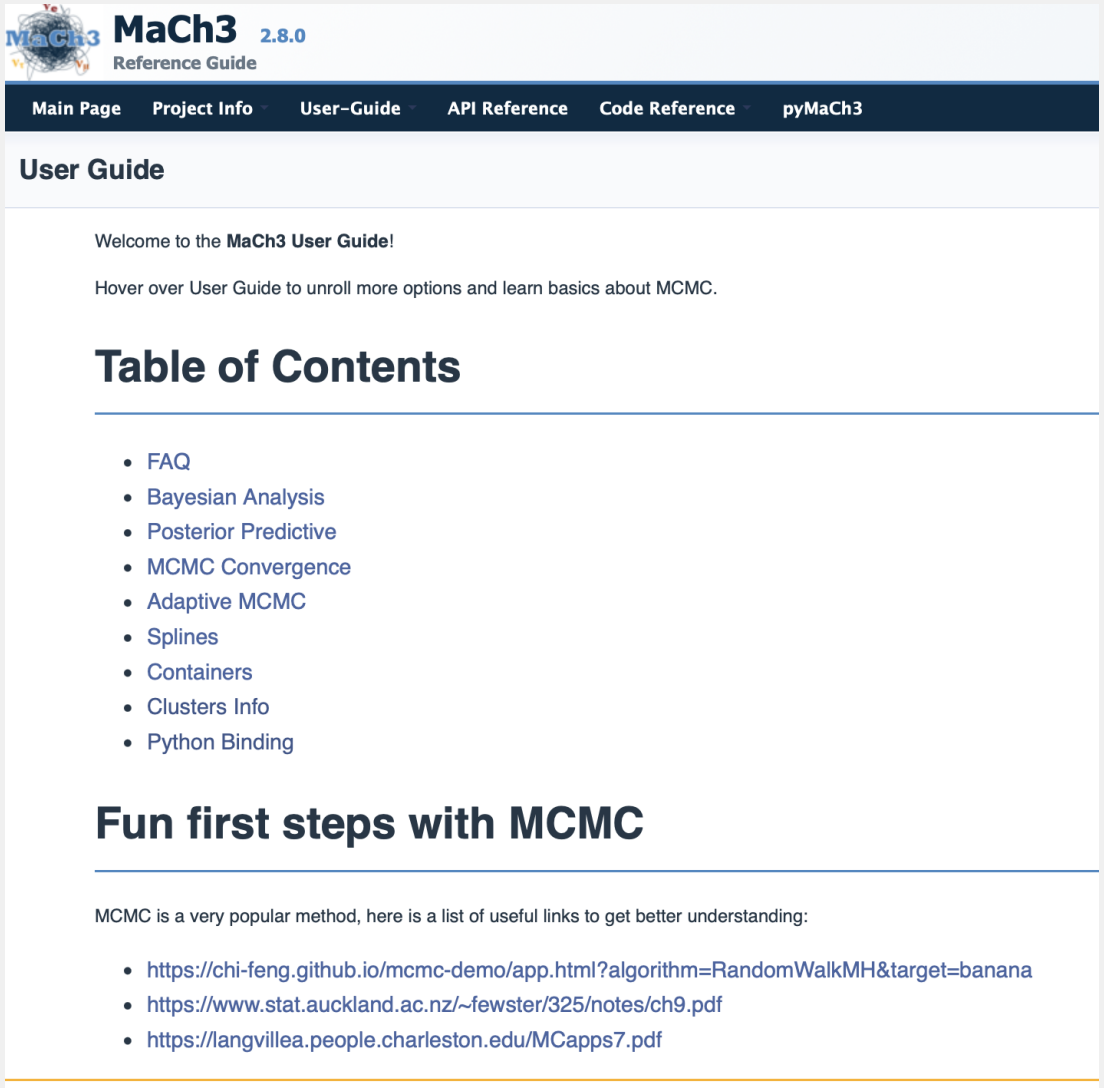
Roughly what I hand in mind already exists 👍

And very nice animation from Kamil 🤩

IMPERIAL

Examples: MaCh3 – User Guide

https://mach3-software.github.io/MaCh3/UserGuide.html



MaCh3 2.8.0
Reference Guide

Main Page | Project Info | User-Guide | API Reference | Code Reference | pyMaCh3

User Guide

Welcome to the **MaCh3 User Guide**!

Hover over User Guide to unroll more options and learn basics about MCMC.

Table of Contents

- FAQ
- Bayesian Analysis
- Posterior Predictive
- MCMC Convergence
- Adaptive MCMC
- Splines
- Containers
- Clusters Info
- Python Binding

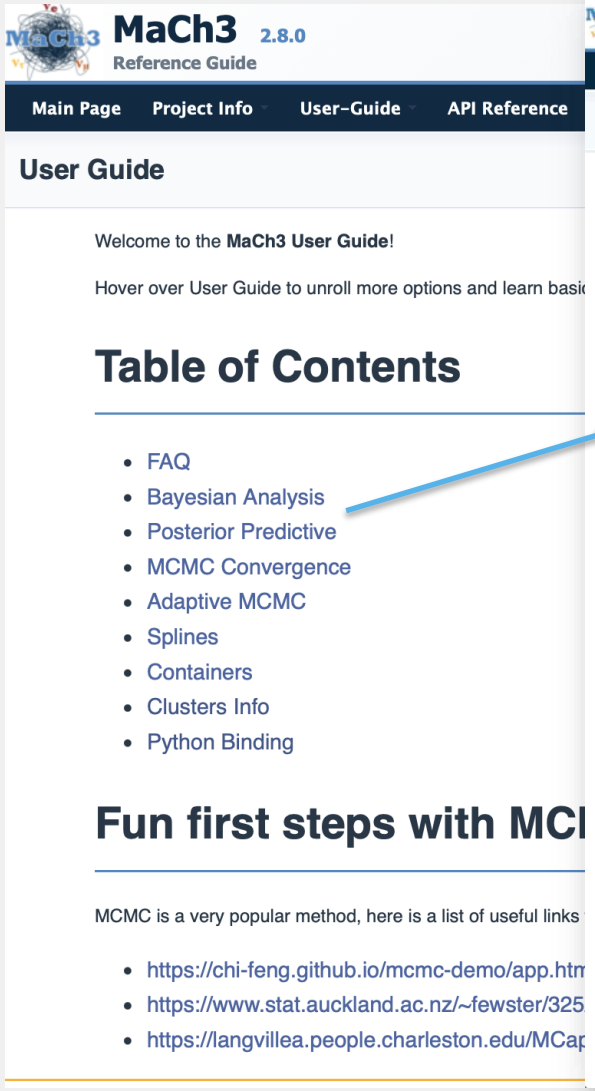
Fun first steps with MCMC

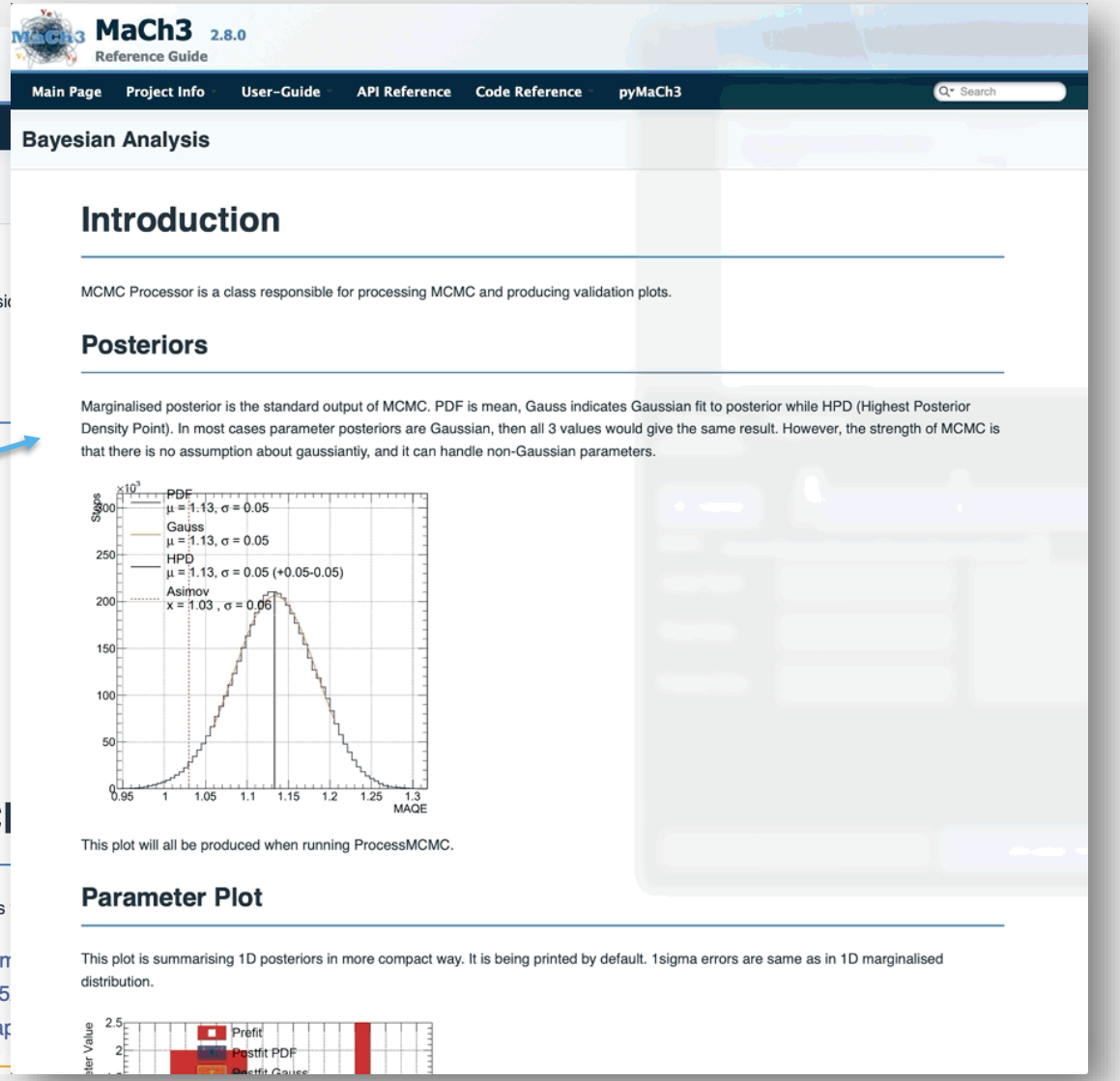
MCMC is a very popular method, here is a list of useful links to get better understanding:

- https://chi-feng.github.io/mcmc-demo/app.html?algorithm=RandomWalkMH&target=banana
- https://www.stat.auckland.ac.nz/~fewster/325/notes/ch9.pdf
- https://langvillea.people.charleston.edu/MCapps7.pdf

Examples: MaCh3 – User Guide (Bayesian Analysis)

Link





Given the title, a user would expect an introduction to MCMC

If I recall correctly, the user guide was copied from the old GitHub Wiki

Could do with a full review… I'll say more later

IMPERIAL

Examples: MaCh3 – Tutorial

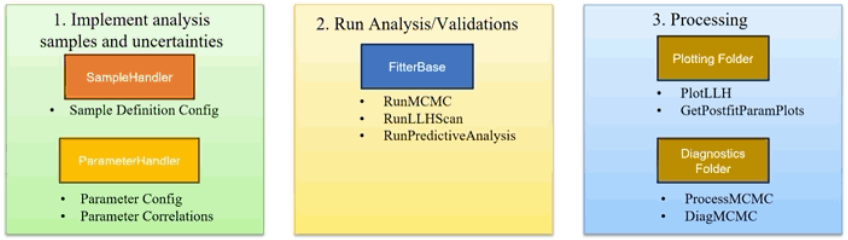
https://github.com/mach3-software/MaCh3Tutorial/tree/main/Tutorial

README Contributing BSD-3-Clause license

Welcome to MaCh3 Tutorial

After this tutorial you should know how to run MCMC, implement systematic uncertainties, new samples, and how to plot standard Bayesian diagnostics.

There are several standard parts of MaCh3 analysis like sample/systematic implementations, validations and plotting. This tutorial will help you get used to different parts of MaCh3.



MaCh3 is predominantly C++ software, although some functionality is available through python as well. See the table of contents for more.

Code Documented Container Image

Table of contents

1. How to Start?
2. How to Run MCMC
   i. Config Overrides
   ii. Processing MCMC Outputs
   iii. Plotting MCMC Posteriors
   iv. Plotting Correlation Matrix
3. Posterior Predictive Analysis

Fantastic that we have this, makes onboarding easier! ✅

Takes users through the typical steps a MaCh3 user would take ✅

Assumes an experiment-specific repo has already been made

If we expect users to extend code, we should show examples of doing that

Could be useful to split tutorial into a minimal example + extras

Extras can include more complexity, approaching real exps.

Examples: MaCh3 – my takeaways

- We have roughly the right structure, but it is a bit bare
- Descriptions of concepts is lacking
  - MCMC statistics (links I found are also not great)
  - How the code base is structured, how the classes interact with each other etc.
- Comprehensive description of configuration also needed
- Current tutorial is a great start, and would benefit with the addition of more
  - e.g. how to make and add a new spline
- Exact strategy depends on how configurable we make MaCh3 and where the user-developer line gets drawn

- Personally, I just use GitHub Copilot on the repo to ask questions these days 😬

Next steps / discussion points

- Let's form a list of MaCh3 activities in order of user→developer
  - e.g. install MaCh3, run a fit, make posterior plot, new selection, … new GPU implementation
  - Then go from top to bottom, and check how well supported it is from a user perspective
    - Does the code need to be more configurable?
    - How well does the documentation address this?

- How general do we want MaCh3 to be?
  - How configurable must we be? Is a user expected to write C++?
- What aspects of other framework's docs do we want to adopt?
  - e.g. from Combine: "Concepts, Configuration, Running" style to a user guide
  - e.g. multiple smaller tutorials

BACKUP