The Scikit-HEP Project

Eduardo Rodrigues
University of Cincinnati

PyHEP 2019 Workshop
16-18 October 2019
Abingdon, U.K.
How’s the Python scientific ecosystem like, outside HEP?

Jake VanderPlas, *The Unexpected Effectiveness of Python in Science*, PyCon 2017
Scikit-HEP project – the grand picture

- Create an ecosystem for particle physics data analysis in Python
- Initiative to improve the interoperability between HEP tools and the scientific ecosystem in Python
  - Expand the typical tool kit set for particle physicists
  - Set common APIs and definitions to ease “cross-talk”
- Initiative to build a community of developers and users
  - Community-driven and community-oriented project
- Effort to improve discoverability of (domain-specific) relevant tools

Collaboration  Reproducibility  Interoperability  Sustainability
Scikit-HEP project – new website, thanks Henry!

The Scikit-HEP project is a community-driven and community-oriented project with the aim of providing Particle Physics at large with an ecosystem for data analysis in Python. The project started in Autumn 2016 and is under active development.

It is not just about providing core and common tools for the community. It is also about improving the interoperability between HEP tools and the scientific ecosystem in Python, and about improving on discoverability of utility packages and projects.

For what concerns the project grand structure, it should be seen as a toolset rather than a toolkit. The project defines a set of five pillars, which are seen to embrace all major topics involved in a physicist’s work. These are:

- **Datasets**: data in various sources, such as ROOT, Numpy/Pandas, databases, wrapped in a common interface.
- **Aggregations**: e.g. histograms that summarize or project a dataset.
- **Modeling**: data models and fitting utilities.
- **Simulation**: wrappers for Monte Carlo engines and other generators of simulated data.
- **Visualization**: interface to graphics engines, from ROOT and Matplotlib to even beyond.

**Toolset packages**
Scikit-HEP project – overview of (most of the) packages

There are other packages: test data, tutorials, org stats, etc.

And other packages, which tend to be now superseded, hence deprecated ...
Data manipulation and interoperability

**uproot**
Minimalist ROOT I/O in pure Python and Numpy

**uproot-methods**
Pythonic mix-ins for non-I/O ROOT classes

**formulate**
Easy conversions between different styles of expressions

See dedicated talk
(Does it still need an intro ;-)?)

- Trivially and Python-ically read ROOT files
- Need only Numpy, no ROOT, using this pure I/O library!

- Design and dependencies:

Write ROOT files: newest development, see dedicated talk
Event processing

Manipulate arrays of complex data structures as easily as Numpy
Event processing – awkward-array package

❑ Provide a way to analyse variable-length and tree-like data in Python, by extending Numpy’s idioms from flat arrays to arrays of data structures

❑ Pure Python+Numpy library for manipulating complex data structures even if they
  - Contain variable-length lists (jagged/ragged)
  - Are deeply nested (record structure)
  - Have different data types in the same list (heterogeneous)
  - Are not contiguous in memory
  - Etc.

❑ This is all very relevant and important for HEP applications!

```bash
pip install awkward  # maybe with sudo or --user, or in virtualenv
pip install awkward-numba  # optional: integration with and optimization by Numba
```

❑ Package being re-implemented in C++, with a simpler interface and less limitations
  - Major endeavour

❑ Work-in-progress, see https://github.com/scikit-hep/awkward-1.0 and dedicated talk ...
boost-histogram

Python bindings for the
C++14 Boost::Histogram library

See status and prospects in
dedicated session!
Provides (pybind11) **Python bindings for the C++14 Boost.Histogram library** (multi-dimensional templated header-only)
- Python(-ic) API mimics the C++ library as much as possible, aside changes for Python performance and idioms

Development via productive exchange of features/ideas between boost-histogram and Boost.Histogram

Binary wheels for all major platforms, supports for all Python versions

Alpha release, on the verge of becoming Beta

A histogram is seen as **collection of Axis objects and a storage**
- Several types available, e.g. circular axis

Example usage:

```python
import boost_histogram as bh

# Compose axis however you like
hist = bh.histogram(bh.axis.regular(2, 0, 1),
                    bh.axis.regular(4, 0.0, 1.0))

# Filling can be done with arrays, one per dimension
hist.fill([.3, .5, .2],
          [.1, .4, .9])

# Numpy array view into histogram counts, no overflow bins
counts = hist.view()
```

---

Regular axis

Optional underflow

Optional overflow

Storage (Static Dynamic)

Accumulator int, double, unlimited, ...

Regular axis with log transform

 bh.axis.circular(0,0,2*np.pi)
A fair amount of interest in the (HEP) community to develop a histogramming sub-ecosystem that meets our requirements

Involves packages for core functionality such as filling, plotting, serialisation, and interoperability

Interaction with popular fitting packages is also paramount

---

Diedrichs Rodrigues

PyHEP 2019, Abingdon, U.K., 16th October 2019
Fitting

iminuit
Python interface to the MINUIT2 C++ package

probfit
Cost function builder. For fitting distributions.

See dedicated talk
Fitting – iminuit package

- Provides **Python interface to the MINUIT2 C++ package** (built on Cython)
- Most commonly used for likelihood fits of models to data, and to get model parameter error estimates from likelihood profile analysis
- Used in many other HEP and non-HEP packages
- Binary wheels for all major platforms

**Brainstorm on a future version 2**

- Would be an interface overhaul to make iminuit easier to use, with a more consistent public interface
- Would use **pybind11** instead of Cython
  - Cython’s C++ support is very limited and there are a lot of bugs/missing features that we currently need to work around
- Switch is not pressing though, since (1) version 2 would not offer new exciting features and (2) breaking the interface will probably hamper adoption
  - iminuit is used as a backend by many projects and changing the interface would require work there

**Example usage:**

```python
from iminuit import Minuit

def f(x, y, z):
    return (x - 2)**2 + (y - 3)**2 + (z - 4)**2

m = Minuit(f)

m.migrad()  # run optimiser
print(m.values)  # {x: 2, y: 3, z: 4}

m.hesse()  # run covariance estimator
print(m.errors)  # {x: 1, y: 1, z: 1}
```
Particles & decays

PDG particle data and identification codes

Parse decay files, describe and convert particle decays between digital representations

See dedicated talk
Particles and decays – Particle package

- Pythonic interface to the Particle Data Group (PDG) particle data table and MC particle identification codes
- With many extra goodies
- Simple and natural APIs

- Main classes for queries and look-ups:
  - Particle
  - PDGID
  - Command-line queries also available

- Powerful and flexible searches as 1-liners, e.g.
Particles and decays – DecayLanguage package

- Tools to parse decay files (aka .dec files) and programmatically manipulate them, query, display information
- Universal representation of particle decay chains
- Tools to translate decay amplitude models from AmpGen to GooFit, and manipulate them

Parse and visualise a decay chain:

```python
from decaylanguage import DecFileParser, DecayChainViewer

dfp = DecFileParser('Dst.dec')
dfp.parse()

chain = dfp.build_decay_chains('D^+', stable_particles=['D+', 'D0'])
DecayChainViewer(chain)
```

Represent a complex decay chain:

```python
dm1 = DecayMode(0.0124, 'K_S0 pi0', model='PHSP')
dm2 = DecayMode(0.692, 'pi+ pi-')
dm3 = DecayMode(0.98823, 'gamma gamma')
dc = DecayChain('D0', [{'D0':dm1, 'K_S0':dm2, 'pi0':dm3})
```

dc.print_as_tree()

```
D0
  --> K_S0
      |  --> pi+
      |  --> pi-
  --> pi0
      |  --> gamma
      |  --> gamma
```
Statistics tools and utilities

scikit-stats

Statistics tools and utilities
Statistics tools and utilities – scikit-stats package

- A recent package
- Being actively developed in collaboration with authors of fitting frameworks, for example, to make sure the needs are covered
  - E.g., zfit (see dedicated talk)

- Plans among IRIS-HEP colleagues to improve/enhance interoperability of statistics tools (e.g. pyhf – see dedicated talk) and fitting frameworks (e.g. RooFit, GooFit, zfit)
  - Common APIs, conversions to enable inter-exchange of models
- Requires community discussion, which is starting at https://gitter.im/HSF/PyHEP-fitting
Simulation

numpythia

Interface between PYTHIA and NumPy

pyhepmc

Python wrapper for the HepMC3 C++ library
Generate events with Pythia and pipe them into NumPy arrays

```python
from numpythia import Pythia, hepmc_write, hepmc_read
from numpythia import STATUS, HAS_END_VERTEX, ABS_PDG_ID

params = {"Beams:eCM": 13000, "WeakSingleBoson:ffbar2gmZ": "on", "23:onMode": "off", "23:onIfAny": "13", "WeakZ0:gmZmode": 2}

pythia = Pythia(params=params)
selection = ((STATUS == 1) & ~HAS_END_VERTEX)

for event in pythia(events=100):
    array = event.all(selection)
    muplus = array[array["pdgid"] == 13]
```

Possible to feed those events into FastJet using pyjet

```python
from pyjet import cluster
from pyjettestdata import get_event

vectors = get_event()
sequence = cluster(vectors, R=1.0, p=-1)
jets = sequence.inclusive_jets()  # list of PseudoJets
```
Simulation – pyhepmc packages

- **HepMC3**: a new rewrite of the C++ HepMC event record for MC generators

- **pyhepmc**: Python wrapper for the HepMC3 C++ library
  - Bindings built on pybind11
  - Supports all Python versions
  - On PyPI as source distribution
  - Beta release version 0.4.3

- Development done in close collaboration with HepMC3 team
  - Idea is to provide pyhepmc as the official bindings, included in the HepMC3 distribution
Visualisation

Minimal viewer of Vega / Vega-Lite plots in your web browser from local or remote Python processes
Minimal viewer of Vega and Vega-Lite graphics from Python
- The Python process generating the graphics does not need to be on the same machine as the web browser viewing them

0 dependencies - can be installed as single file, used as a Python library or as a shell command, watching a file or stdin

Example:

```python
import vegascoper
canvas = vegascoper.LocalCanvas()
canvas("https://vega.github.io/vega/examples/stacked-bar-chart.vg.json")
```

Altair can use VegaScope as a renderer:

```python
import vegascoper
canvas = vegascoper.LocalCanvas()
canvas("https://vega.github.io/vega/examples/stacked-bar-chart.vg.json")

import altair as alt
alt.renderers.enable('vegascoper')
RendererRegistry.enable('vegascoper')

from vega_datasets import data
cars = data.cars()
alt.Chart(cars).mark_point().encode(x='Horsepower',
y='Miles_per_Gallon',
color='Origin',
).interactive()

Rendered at http://localhost:56574
```
Units and constants

hepunits

Units and constants in the HEP system of units
Units and constants in the HEP system of units

- Not the same as the SI system of units

Trivial package, but handy

Typical usage:

```python
from hepunits.constants import c_light
from hepunits.units import picosecond, micrometer

tau_bs = 1.5 * picosecond  # a particle lifetime, say the Bs meson's
tau Bs = c_light * tau Bs # ctau of the particle, ~450 microns
ptau Bs = ctau Bs # result in HEP units, so mm

print(ctau Bs)  # result in micrometers

print(ctau Bs / micrometer)  # result in micrometers

0.44968868700000003
```

More “advanced”:

```python
from hepunits import c_light, GeV, meter, ps
from math import sqrt

def ToF(m, p, l):
    """Time-of-Flight - particle path length l / (c * beta)""
    one_over_beta = sqrt(1 + m*m/(p*p))
    return (l * one_over_beta / c_light)

from particle.particle.literals import pi_plus, K_plus  # particle name literals

delta = (ToF(K_plus.mass, 10*GeV, 10*meter) - ToF(pi_plus.mass, 10*GeV, 10*meter)) / ps
print("At 10 GeV, Delta-TOF(K-pi) over 10 meters = {:.5} ps".format(delta))

At 10 GeV, Delta-TOF(K-pi) over 10 meters = 37.374 ps
```
Who uses (some of) Scikit-HEP?

- Groups, other projects, HEP experiments
- Links are important, especially if they strengthen the overall ecosystem
- Community adoption going up \(\Leftrightarrow\) we're on the right path ;-
- Rewarding to collaborate / work with / interact with many communities
  - Responsibility and importance of sustainability …

Software projects

- **Coffea** - a prototype *Analysis System* incorporating Scikit-HEP packages to provide a lightweight, scalable, portable, and user-friendly interface for columnar analysis of HEP data. Some of the sub-packages of Coffea may become Scikit-HEP packages as development continues.

- **zfit** - The *zfit* project - it provides a model fitting library based on TensorFlow and optimised for simple and direct manipulation of probability density functions.

Experiment collaborations

- **Belle II** - the Belle II experiment at KEK, Japan.
- **CMS** - the Compact Muon Solenoid experiment at CERN, Switzerland.

Phenomenology projects

- **flavio** - flavour physics phenomenology in the Standard Model and beyond.
The *scikit-hep* package has historically contained a variety of things:
- Kinematics and geometry classes for HEP
- Modelling module
- Visualisation utilities
- Etc.

The project has evolved and a different route has emerged as more adequate …

**Vision for the future:** have the *scikit-hep* package become a metapackage for the Scikit-HEP project

**Benefit especially for stacks for experiments:** *scikit-hep* tags defining compatible releases of the whole toolset
- Clear what "*scikit-hep* version 1.0.0" is
- Stable stacks installable in a simple way
- Having a well-defined stack also helps in analysis preservation matters, widely discussed at present

This is (still) work-in-progress …
Interested ? Want to try it ? And contribute ?

- We are a community ⇒ everybody welcome !
  - Particularly interesting to have a good sampling from the various experiments

- A lot to be done, still …

- … and we need feedback too !

Links

- GitHub: https://github.com/scikit-hep/
- Website: http://scikit-hep.org/

Get in touch

- Gitter channel: https://gitter.im/Scikit-HEP/community
- Forum for anyone: scikit-hep-forum@googlegroups.com
- Get in touch with the team “privately”: scikit-hep-admins@googlegroups.com

Thank you