University of CINCINNATI ARTS AND SCIENCES

diana hep

iris hep
Institute for Research & Innovation in Software for High Energy Physics

Particles and decays in the Scikit-HEP project

Eduardo Rodrigues, Henry Schreiner
University of Cincinnati , Princeton University

Belle II Analysis Software Meeting, 21st May 2019

---

particle

PDG particle data and identification codes

Decay Language

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

The grand picture – the Scikit-HEP project

The idea, in just one sentence

The Scikit-HEP project (http://scikit-hep.org/) is a community-driven and community-oriented project with the aim of providing Particle Physics at large with a Python package containing core and common tools.

- ❑ 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
- ❑ Effort to improve discoverability of relevant tools

Collaboration

Reproducibility

Interoperability

Sustainability

PDG particle data and identification codes

Particle package – motivation

- The Particle Data Group (PDG) provides a downloadable table of particle masses, widths, charges and Monte Carlo particle ID numbers (PDG IDs)
  - Most recent file here

- It also provided an experimental file with extended information (spin, quark content, P and C parities, etc.) until 2008 *only*, see here (not widely known!)

- But *anyone* wanting to use these data, the only readily available, has to parse the file programmatically
- Why not make a Python package to deal with all these data, for everyone?

- The C++ HepPID and HepPDT libraries provide functions for processing particle ID codes in the standard particle (aka PDG) numbering scheme
- Different event generators have their separate set of particle IDs: Pythia, EvtGen, etc.
- Again, why not make a package providing all functionality/conversions, Python-ically, for everyone?

Particle package – overview

- Pythonic interface to
  PDG particle data table
  and
  particle identification codes
- With extra goodies

- 2 separate submodules

- Comprehensive documentation (docstrings)

- Continuous Integration (CI): extensive tests for excellent test coverage
  - In packages such as these, tests should target both the code itself but also the physics it deals with!
- We use Azure DevOps
  - Trivially test on Linux, macOS and Windows

Particle package – PDG IDs module overview

- Process and query PDG IDs, and more – no look-up table needed
  - Current version of package reflects the latest version of the HepPID & HepPDT utility functions defined in the C++ HepPID and HepPDT versions 3.04.01
  - It contains more functionality than that available in the C++ code … and minor fixes too
- Definition of a `PDGID` class, PDG ID literals, and set of standalone HepPID functions to query PDG IDs (`is_meson`, `has_bottom`, `j_spin`, `charge`, etc.)
  - All PDGID class functions are available standalone
- PDG ID queries also available on the command line
- `PDGID` class
  - Wrapper class for PDG IDs
  - Behaves like an int, with extra goodies
  - Large spectrum of properties and methods, i.e. the functions defined in the HepPID and HepPDT C++ libraries, with a Pythonic interface, and yet more
  - To print them all:

```
In [8]: print(PDGID(2212).info())
        A          1
        C          None
        J          0.5
```

```
In [1]: from particle import PDGID

        pid = PDGID(211)
        pid
Out[1]: <PDGID: 211>

In [2]: pid.is_meson
Out[2]: True

In [3]: pid = PDGID(99999999)
        pid
Out[3]: <PDGID: 99999999 (is_valid==False)>

In [4]: from particle.pdgid import is_meson

        is_meson(211)
Out[4]: True
```

Particle package – PDG identification code literals

- **Literals**: handy way to manipulate things with human-readable names

- **PDGID literals**
  - Provide (PDGID class) aliases for the most common particles, with easily recognisable names

```
In [5]: from particle.pdgid import literals as lid

        lid.pi_plus
Out[5]: <PDGID: 211>

In [6]: from particle.pdgid.literals import Lambda_b_0

        Lambda_b_0
Out[6]: <PDGID: 5122>

In [7]: Lambda_b_0.has_bottom
Out[7]: True
```

- **All is consistent. Ex.:**

```
In [8]: Particle.from_pdgid(-10311).pdgid == literals.K_0st_1430_0_bar
Out[8]: True
```

Particle package – particle module overview

- Simple and natural API to deal with the PDG particle data table, with powerful search and look-up utilities

- Definition of a `Particle` class and particle name literals
  - Typical queries should be, and are, 1-liners

```
In [7]: from particle import Particle, SpinType

        Particle.findall(lambda p: p.pdgid.is_meson and p.pdgid.has_charm and p.spin_type==SpinType.PseudoScalar)

Out[7]: [<Particle: name="D+", pdgid=411, mass=1869.65 ± 0.05 MeV>,
         <Particle: name="D-", pdgid=-411, mass=1869.65 ± 0.05 MeV>,
         <Particle: name="D0", pdgid=421, mass=1864.83 ± 0.05 MeV>,
         <Particle: name="D~0", pdgid=-421, mass=1864.83 ± 0.05 MeV>,
         <Particle: name="D(s)+", pdgid=431, mass=1968.34 ± 0.07 MeV>,
         <Particle: name="D(s)-", pdgid=-431, mass=1968.34 ± 0.07 MeV>,
         <Particle: name="eta(c)(1S)", pdgid=441, mass=2983.9 ± 0.5 MeV>,
         <Particle: name="B(c)+", pdgid=541, mass=6274.9 ± 0.8 MeV>,
         <Particle: name="B(c)-", pdgid=-541, mass=6274.9 ± 0.8 MeV>,
         <Particle: name="eta(c)(2S)", pdgid=100441, mass=3637.6 ± 1.2 MeV>]
```

- Advanced usage: ability to specify or build a particle data table, conversion tools

- Particle / PDG ID searches available on the command line

Particle package – data files

- All data files stored under `particle/data/`

- PDG particle data files
  - Original PDG data files, which are in a fixed-width format
  - Code uses “digested forms” of the PDG files, stored as CSV, for optimised querying
  - Latest-ish PDG data used by default (2018 at present, 2019 update available since a few days!)
  - Advanced usage: user can load older PDG table, load a “user table” with new particles, append to default table

ID	Mass	MassUpper	MassLower	Width	WidthUpper	WidthLower	I	G	P	C	Anti	Charge	Rank	Status	Name	Quarks
...																
441	2983.9	0.5	0.5	32	0.8	0.8	0	1	-1	1	0	0	0	0	eta(c)(1S)	cC
443	3096.9	0.006	0.006	0.0929	0.0028	0.0028	0	-1	-1	-1	0	0	0	0	J/psi(1S)	cC
445	3556.17	0.07	0.07	1.97	0.09	0.09	0	1	1	1	0	0	0	0	chi(c2)(1P)	cC
...																

- Other data files
  - CSV file for mapping of PDG IDs to particle LaTeX names

Particle package – particle look-up

- `Particle` class
  - Standard look-up via `from_pdgid(…)`
  - Various other `from_X(…)` methods exist
- Large spectrum of properties and methods:
  - Get particle properties
  - Deal with underlying particle table
  - Powerful search engine …

```
In [1]: from particle import Particle

        Particle.from_pdgid(-14122)
```

Out[1]: $\bar{\Lambda}_c(2593)^-$

```
In [2]: Particle.from_string('pi-')
```

Out[2]: π^-

```
In [3]: from IPython.core.display import display, HTML, Latex

        print(Particle.from_pdgid(-10311).__repr__())
        display(HTML('\t HTML name: {0}'.format(Particle.from_pdgid(-10311).html_name)))
        display(Latex('\t LaTex name: ${0}$'.format(Particle.from_pdgid(-10311).latex_name)))
```

```
<Particle: name="K(0)*(1430)~0", pdgid=-10311, mass=1430 ± 50 MeV>
```

HTML name: $\overline{K}_0^*(1430)^0$

LaTex name: $\bar{K}_0^*(1430)^0$

```
In [11]: from particle.particle import literals as lp

         print(lp.K_0st_1430_0_bar.pdgid)
         print(Particle.from_pdgid(-10311).programmatic_name)
```

```
<PDGID: -10311>
K_0st_1430_0_bar
```

- Particle literals
  - Easily recognizable names for manipulations, e.g. in plots

Particle package – powerful particle search

- `Particle.find(…)` – search a single match (exception raised if multiple particles match the search specifications)
- `Particle.findall(…)` – search a list of candidates

```
In [16]: print(Particle.find('J/psi').describe())

         Name: J/psi(1S)      ID: 443          Latex: $J/\psi(1S)$
         Mass  = 3096.900 ± 0.006 MeV
         Lifetime = 7.09e-12 ± 2.2e-13 ns
         Q (charge)         = 0       J (total angular) = 1.0      P (space parity) = -
         C (charge parity) = -        I (isospin)       = 0        G (G-parity)     = -
             SpinType: SpinType.Vector
             Quarks: cC
             Antiparticle name: J/psi(1S) (antiparticle status: Same)
```

- Powerful search methods that can query any particle property!
- One-line queries

Particle package – future directions & developments

- **Addition of particle IDs and names relevant to other MC programs**
  - **(Yes, not consistent across programs!)**
  - **Useful IDs such as those used in PYTHIA, Geant and EvtGen**
- **Bring in other communities where Particle is / can be relevant**
  - **Ongoing discussions with astroparticle physics community**
  - **Particle IDs used in EPOS, CORSIKA, DpmJet, QGSJet, Sybill, UrQMD, …**
- **Ongoing discussions with PDG group**
  - **Provide the right tool**
  - **Can we provide more?**
  - **Stay tuned …**

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

DecayLanguage package – motivation and overview

AMPGEN

Library and set of applications for fitting and generating multi-body particle decays using the isobar model

Motivation

- Ability to describe decay-tree-like structures
- Provide a translation of decay amplitude models from AmpGen to GooFit
  - Idea is to generalise this to other decay descriptions

GooFit
CUDA/OpenMP
Fitting Framework
for C++ & Python

- Any experiment uses event generators which, among many things, need to describe particle decay chains
- Programs such as EvtGen rely on so-called .dec decay files
- Many experiments need decay data files
- Why not make a Python package to deal with decay files, for everyone?

Overview

- Tools to translate decay amplitude models from AmpGen to GooFit, and manipulate them
- Tools to parse decay files and programmatically manipulate them, query, display information
  - Descriptions and parsing built atop the Lark parser

DecayLanguage package – conversion of decay models / representations

- **Decay chains**
  - **A universal modelling of decay chains would profit many use cases,**
    e.g. description of components for amplitude analyses
- **Present code understands AmpGen syntax and can generate code for the GooFit fitter**
- **Note:**
  makes use of the `Particle` package

```
[2]: lines, parameters, constants, states = AmplitudeChain.read_ampgen(text='''
     EventType D0 K- pi+ pi+ pi-

     D0[D]{K*(892)bar0{K-,pi+},rho(770)0{pi+,pi-}}                    0 1 0.1 0 1 0.1

     K(1460)bar-_mass  0 1460 1
     K(1460)bar-_width 0  250 1

     a(1)(1260)+::Spline::Min 0.18412
     a(1)(1260)+::Spline::Max 1.86869
     a(1)(1260)+::Spline::N 34
     ''')

[3]: lines[0].all_particles

[3]: {<Particle: name="rho(770)0", pdgid=113, mass=775.3 ± 0.2 MeV>,
      <Particle: name="pi+", pdgid=211, mass=139.57061 ± 0.00024 MeV>,
      <Particle: name="pi-", pdgid=-211, mass=139.57061 ± 0.00024 MeV>,
      <Particle: name="K*(892)~0", pdgid=-313, mass=895.55 ± 0.20 MeV>,
      <Particle: name="K-", pdgid=-321, mass=493.677 ± 0.016 MeV>,
      <Particle: name="D0", pdgid=421, mass=1864.83 ± 0.05 MeV>}
```

DecayLanguage package – decay files

"Master file" DECAY.DEC

- Gigantic file defining decay modes for all relevant particles, including decay model specifications
- LHCb example:
  ~ 450 particle decays, thousands of decay modes, over 11k lines in total

```
Define dm 0.507e12
...

Alias      B0sig              B0
Alias      anti-B0sig         anti-B0
ChargeConj B0sig              anti-B0sig
...

Decay pi0
0.988228297   gamma   gamma                    PHSP;
0.011738247   e+      e-      gamma            PI0_DALITZ;
0.000033392   e+      e+      e-      e-       PHSP;
0.000000065   e+      e-                       PHSP;
Enddecay
...

CDecay tau+
...
```

User .dec files

- Needed to produce specific MC samples
- Typically contain a single decay chain
  (except if defining inclusive samples)

```
# Decay file for [B_c+ -> (B_s0 -> K+ K-) pi+]cc

Alias      B_c+sig        B_c+
Alias      B_c-sig        B_c-
ChargeConj B_c+sig        B_c-sig
Alias      MyB_s0         B_s0
Alias      Myanti-B_s0    anti-B_s0
ChargeConj MyB_s0         Myanti-B_s0

Decay B_c+sig
  1.000     MyB_s0     pi+     PHOTOS PHSP;
Enddecay
CDecay B_c-sig
Decay MyB_s0
    1.000     K+     K-     SSD_CP 20.e12 0.1 1.0 0.04 9.6 -0.8 8.4 -0.6;
Enddecay
CDecay Myanti-B_s0
```

DecayLanguage package – Lark parser grammar for decay files

- Decay file parser grammar: `decfile.lark` !
- This file is enough to parse and understand decay files

```
start : _NEWLINE? (line _NEWLINE)+ ("End" _NEWLINE)?
?line : define | pythia_def | alias | chargeconj | commands | decay | cdecay | setlspw

pythia_def : "PythiaBothParam" LABEL ":" LABEL "=" (LABEL | SIGNED_NUMBER)
setlspw : "SetLineshapePW" label label label value
cdecay : "CDecay" label
define : "Define" label value

alias : "Alias" label label
chargeconj : "ChargeConj" label label

?commands : global_photos
global_photos : boolean_photos
boolean_photos : "yesPhotos" -> yes
               | "noPhotos"  -> no

decay : "Decay" particle _NEWLINE decayline+ "Enddecay"
decayline : value particle* photos? model _NEWLINE // There is always a ; here
value : SIGNED_NUMBER
photos : "PHOTOS"

label : LABEL
particle : LABEL // Add full particle parsing here

model : MODEL_NAME model_options?
model_options : (value | LABEL)+

%import common.WS_INLINE
%import common.SIGNED_NUMBER

// New lines filter our comments too, and multiple new lines
_NEWLINE: ( /\r?\n[\t ]*/ | COMMENT )+

MODEL_NAME.2 : "BaryonPCR"|"BTO3PI_CP"|"BTOSLLALI"|"BTOSLLBALL"|"BTOXSGAMMA"|"BTOXSLL"| ...
LABEL : /[a-zA-Z0-9\/\-+*_()']+/
COMMENT : /[;#][^\n]*/

// We should ignore comments
%ignore COMMENT

// Disregard spaces in text
%ignore WS_INLINE
```

DecayLanguage package – decay file parsing and display

- **Parsing should be simple**
  - **Expert users can configure parser choice and settings, etc.**
- **After parsing, many queries are possible …**

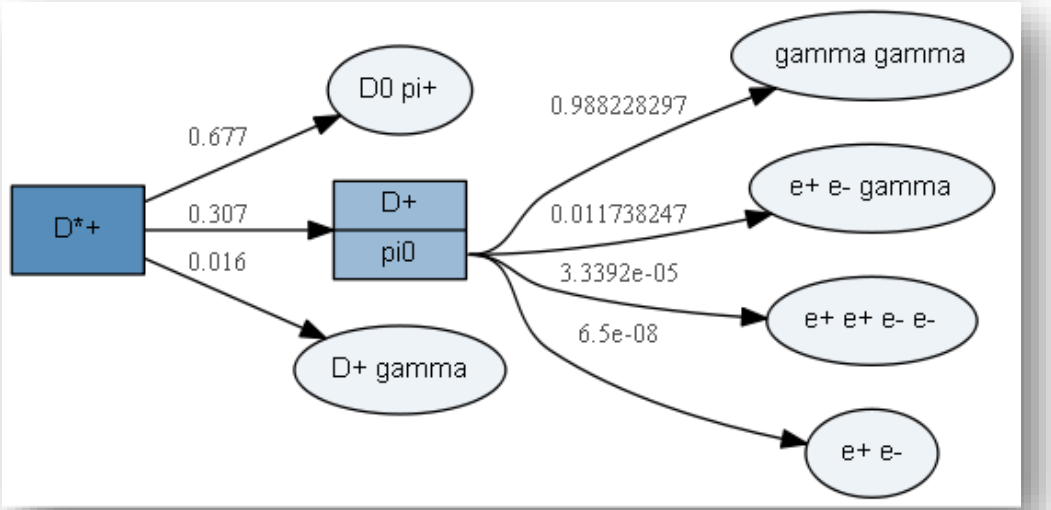
```python
from decaylanguage import DecFileParser

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

chain = p.build_decay_chain('D*+', stable_particles=['D+', 'D0'])

# Display the decay chain
```

- **One can also visualise the decay chain ☺**




```
Decay D*+
0.6770    D0  pi+                    VSS;
0.3070    D+  pi0                    VSS;
0.0160    D+  gamma                  VSP_PWAVE;
Enddecay

Decay D*-
0.6770    anti-D0  pi-                VSS;
0.3070    D-       pi0                VSS;
0.0160    D-       gamma              VSP_PWAVE;
Enddecay

Decay D0
1.0   K-      pi+                  PHSP;
Enddecay

Decay D+
1.0   K-   pi+   pi+   pi0    PHSP;
Enddecay

Decay pi0
0.988228297   gamma   gamma                  PHSP;
0.011738247   e+      e-      gamma          PI0_DALITZ;
0.000033392   e+      e+      e-      e-     PHSP;
0.000000065   e+      e-                     PHSP;
Enddecay
```

(Considered by itself, this file in in fact incomplete,
as there are no instructions on how to decay the anti-D0 and the D-.
Good enough for illustration purposes, though.)

Particle package – future directions & developments

Decay models / representations

- Implement more backend formats: GooFit in Python, etc.
- Longer term – implement decay logic inside model descriptions
  - Provide a reference for other packages

Decay files

- Streamline and enhance the .dec parser
  - Ex.: syntax such as

```
p.find_decay_chains(final_state=['K+', 'K-', 'pi+', 'pi-'], extra_particles=['pi0'])
```

    could be a neat/trivial way to query the master DECAY.DEC and
    "find all decay chains leading to either 'K+ K- pi+ pi-' or 'K+ K- pi+ pi- pi0'"
- Provide a universal description and visualisation of decay trees (a lot done on this in the last week …)
  - We already have customers interested, e.g. visualisation of decays in pyhepmc

Interested ? Want to try it ? And …

Particle

- GitHub: https://github.com/scikit-hep/particle/
- Releases: PyPI 

- Kindly recognise software work – cite us: 

DecayLanguage

- GitHub: https://github.com/scikit-hep/decaylanguage
- Releases: PyPI 

See interactive demos by Henry Schreiner

Scikit-HEP project

- GitHub: https://github.com/scikit-hep/
- Website: http://scikit-hep.org/
- Get in touch: http://scikit-hep.org/get-in-touch.html

… want to contribute ?

- We are writing down ideas / “issues” / TODOs on the repositories … have a look!
  - Example for `Particle`: