

Particle decay reconstruction

P. Ronchese

Dipartimento di Fisica e Astronomia “G.Galilei”

Università di Padova

“Object oriented programming and C++” course

Simulated events

Data to read and analyze have been produced by simulating the decay of two strange particles

- $K_S^0 \rightarrow \pi^+ \pi^-$
two opposite charge particles in the final state,
with the same mass
- $\Lambda^0 \rightarrow p \pi^-$
two opposite charge particles in the final state,
with different masses
- background events
up to 10 particles, with random charges

Particles properties

- Unstable particles
 - K_S^0 : $M_K = 0.497611 \text{ GeV}/c^2$, $\tau_K = 89.54 \text{ ps}$
 - Λ^0 : $M_\Lambda = 1.115683 \text{ GeV}/c^2$, $\tau_\Lambda = 263.2 \text{ ps}$
- Stable particles
 - π^\pm : $M_\pi = 0.1395706 \text{ GeV}/c^2$
 - p : $M_p = 0.938272 \text{ GeV}/c^2$

Stable particles (pions, protons) cannot be distinguished, unstable particles must be reconstructed by making hypotheses on the masses of the decay products

Invariant mass

The mass of a decaying particle can be determined by the masses and momenta of its decay products plus the light speed in vacuum c :

$$M = \frac{\sqrt{E_{\text{tot}}^2 - p_{\text{tot}}^2 c^2}}{c^2} ; \quad E_{\text{tot}} = \sum_i E_i ; \quad p_{\text{tot}} = |\sum_i \vec{p}_i|$$

The single particle energies can be computed in an analogous way:

$$E_i = \sqrt{p_i^2 c^2 + M_i^2 c^4}$$

Usually masses and momenta are measured with units GeV/c^2 and GeV/c respectively, so that in the previous formulas $c = 1$ can be assumed:

$$M = \sqrt{E_{\text{tot}}^2 - p_{\text{tot}}^2} ; \quad E_i = \sqrt{p_i^2 + M_i^2}$$

Decay proper time

The decay time, in the laboratory and in the particle rest frame can be obtained by the flight distance d and the velocity v :

$$t_{\text{LAB}} = \frac{d}{v} \quad ; \quad t = \frac{d\sqrt{1 - v^2/c^2}}{v}$$

Combining with the relations giving momentum

$$p = \frac{Mv}{\sqrt{1 - v^2/c^2}}$$

the proper decay time is given by

$$t = \frac{dM}{p}$$

If masses and momenta are measured with units GeV/c^2 and GeV/c respectively an additional factor c must be included:

$$t = \frac{dM}{pc}$$

Data format

The file `particle_events.txt` contains the data,
written in text form

Each event contain:

- an event identifier (i.e. an `int`)
- the coordinates x, y, z of the decay point (3 `floats`)
- the number of particles (an `int`)
- for each particle 4 numbers:
 - the electric charge (an `int`)
 - the momentum components p_x, p_y, p_z (3 `floats`)

Particle data dump - version 1

Read the text file and produce a dump onto the screen
(use the “Energies data dump - version 1” example as reference)

- Create 4 arrays:
 - an array of `int` to contain particles charges,
 - an array of `float` to contain particles p_x ,
 - an array of `float` to contain particles p_y ,
 - an array of `float` to contain particles p_z .
- Create functions to:
 - read an event from file,
 - dump an event onto the screen,
- Create a `main` function to loop over the file and dump the events.

Particle data dump - version 2

Read the text file and produce a dump onto the screen
(use the “Energies data dump - version 2” example as reference)

- Create 2 `struct`s:
 - `Event` : to contain event data,
 - `Particle` : to contain particle data.
- Create functions to:
 - read an event from file,
 - dump an event onto the screen,
 - free the memory used by the event
(including the memory used by all the particles).
- Create a `main` function to loop over the file and dump the events.

Particle mass mean - version 1

Read the text file and compute invariant mass mean and r.m.s.
(use the “Energies data mean - version 1” example as reference)

- Create a function to compute the invariant mass of the decaying particle:
 - consider the two hypotheses $K_S^0 \rightarrow \pi^+\pi^-$ and $\Lambda^0 \rightarrow p\pi^-$
 - choose the one being more consistent with the known M_K and M_Λ
 - select events with mass in the range $0.490 \div 0.505 \text{ GeV}/c^2$
- Count the number of selected events
- Compute mass mean and r.m.s.

Particle mass mean - version 2

Read the text file and compute invariant mass mean and r.m.s.
for K_S^0 and Λ^0
(use the “Energies data mean - version 2” example as reference)

- Modify the version 1 to use `classes`:
 - create a `class` to contain event data,
 - create a `class` to compute statistics.
 - select events with total energy in the following ranges:
 - between 0.495 and 0.500,
 - between 1.115 and 1.116.
- Modify the `main` function to use the new `classes`.

Particle mass mean - version 3

Read the text file and compute invariant mass mean and r.m.s.
for K_S^0 and Λ^0
(use the “Energies data mean - version 3” example as reference)

- Modify the version 2 to use STL:
 - use a `std::string` to handle input file name,
 - use a `std::vector` to store particle pointers.

Particle mass mean - version 4

Read the text file and compute invariant mass mean and r.m.s.
for K_S^0 and Λ^0
(use the “Energies data mean - version 4” example as reference)

- Modify the version 3 to use `classes` in place of global functions:
 - create a `class` to read or simulate events,
 - create a `class` to dump events,
 - create a `class` to compute statistics.
- Create the `classes` as derivations of interfaces to get events and process them.
- Create a `class` to simulate events.
- Modify the `main` function to use the new `classes`.

Particle mass histogram - version 1

Histogram invariant mass for K_S^0 and Λ^0
(use the “Energies data plot - version 1” example as reference)

- Modify the mean-version 4 to include graphic histograms and allow multiple histograms handling:
 - inside the general analysis steering `class` pair each object computing mean and rms mass with a `TH1F` object,
 - create, set and save histograms in the same `class`, too.
- All other `classes` and the main function stay unchanged.

Particle mass histogram - version 2

Histogram invariant mass for K_S^0 and Λ^0
(use the “Energies data plot - version 2” example as reference)

- Take `AnalysisInfo` and `SourceFactory` from the Bragg analysis example.
- Modify the main function to use the new `classes` .

Particle mass histogram - version 3

Histogram invariant mass for K_S^0 and Λ^0
(use the “Energies data plot - version 3” example as reference)

- Modify the version 2 to use a “factory” to create analyzer objects:
 - take the new versions of `AnalysisSteering` and `AnalysisFactory` from the Bragg analysis example,
 - modify `EventDump` and `ParticleMass` to be handled by `AnalysisFactory`.
- Modify the main function to use the new `class`.

Particle mass histogram - version 4

Histogram invariant mass and decay time for K_S^0 and Λ^0
(use the “Energies data plot - version 4” example as reference)

- Modify the version 3 to use a “dispatcher” to loop over events:
 - take the new versions of `AnalysisSteering` and `EventSource` from the bragg analysis example,
 - modify `EventDump` and `ParticleMass` to be `ActiveObservers`,
 - reconstruct decays in a class `ParticleReco` declared as `LazyObserver` and `Singleton`,
- create `ParticleLifetime`, `ProperTime` and `LifetimeFit` classes for decay time histograms,
- Modify the main function to use the new `EventSource`.

Particle mass histogram - version 5

Histogram invariant mass and decay time for K_S^0 and Λ^0
(use the “Energies data plot - version 5” example as reference).
Add mean lifetime fit.

- Modify the version 4 to use a “dispatcher” to handle begin/end of analysis: take the new versions of `main`, `AnalysisInfo` and `AnalysisSteering` from the Bragg curve analysis example.
- Replace in `main` the analyzers loop calls to `beginJob` and `endJob` with notifications of `AnalysisInfo::begin` or `AnalysisInfo::end`.
- In `LifetimeFit` add a maximum likelihood fit to estimate the mean lifetime and the corresponding error

Particle mass histogram - version 6

Histogram invariant mass and decay time for K_S^0 and Λ^0
(use the “Energies data plot - version 6” example as reference).

- Modify the version 5 to organize the code in packages:
 - create 4 packages:
 - `AnalysisFramework`,
 - `AnalysisPlugins`,
 - `AnalysisObjects`,
 - `AnalysisUtilities`,
 - move all source files, including `main.cc`, into those packages, avoiding circular dependencies,
 - compile each package into a library but `AnalysisPlugins`, where each analyzer is to be compiled to a distinct library.
- Produce the executable by using a dummy source code.

