Laura++ Tutorial: Getting Started

v3r3

Initial setup

- For full up to date instructions on downloading and building the code please see the documentation page:

  http://laura.hepforge.org/doc/
- The build instructions are also in the file doc/README

- A key point is to have the ROOT environment set up
- You must either have the ROOTSYS environment variable set or the root-config script must be in your PATH

Downloading the code

- Firstly check what is the latest tag by browsing:

  http://laura.hepforge.org/svn/tags/

- To download the tag v3r3 (for example) you can do:

```
svn co http://laura.hepforge.org/svn/tags/v3r3 Laura++
```

- Alternatively you can download and extract the corresponding tar file, which can be found on the Laura++ download page:

  http://www.hepforge.org/downloads/laura

Building the library

- The library is currently built with a handmade Makefile, which should work fine for Linux anad MacOSX
- Please email the developers in case of problems
- You should just have to do:

```
make
```

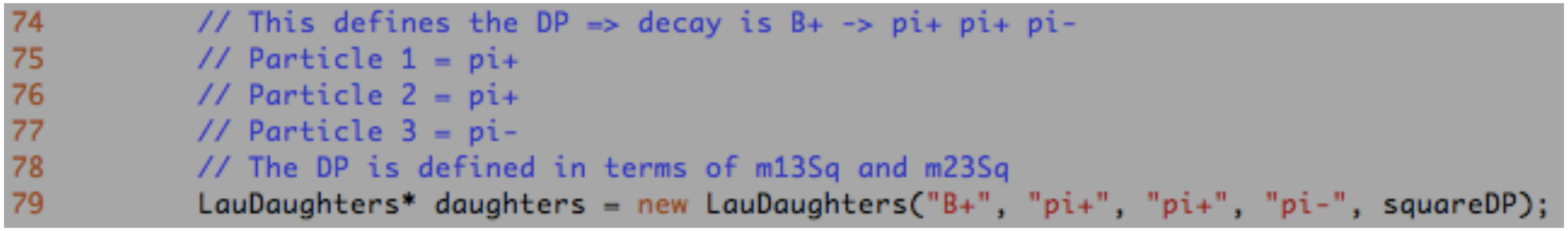
- The shared library appears in the lib directory

A first example

- Let’s use the examples/GenFit3pi.cc code as our first example

Defining the DP

- To define the DP we’re working in we do:

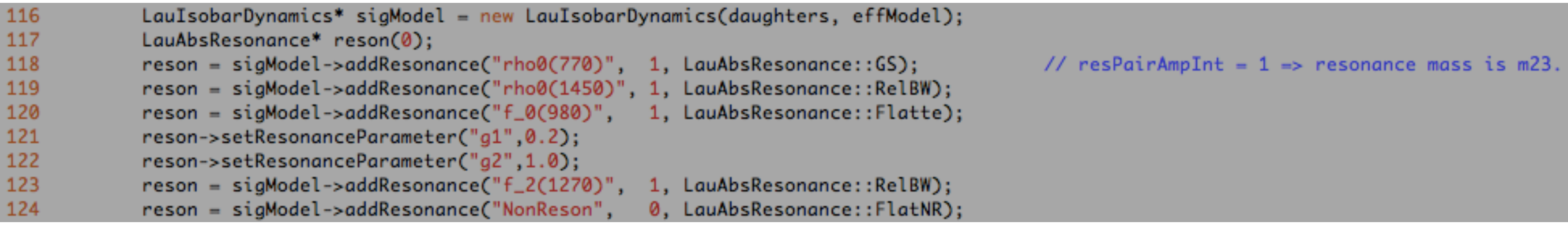


```
        // This defines the DP => decay is B+ -> pi+ pi+ pi-
        // Particle 1 = pi+
        // Particle 2 = pi+
        // Particle 3 = pi-
        // The DP is defined in terms of m13Sq and m23Sq
        LauDaughters* daughters = new LauDaughters("B+", "pi+", "pi+", "pi-", squareDP);
```

- This defines the parent particle and the 3 daughters
- (Other possible parents and daughters are defined in LauDaughters::createParticleLists)

Creating the signal model

- To define the signal DP model we do:

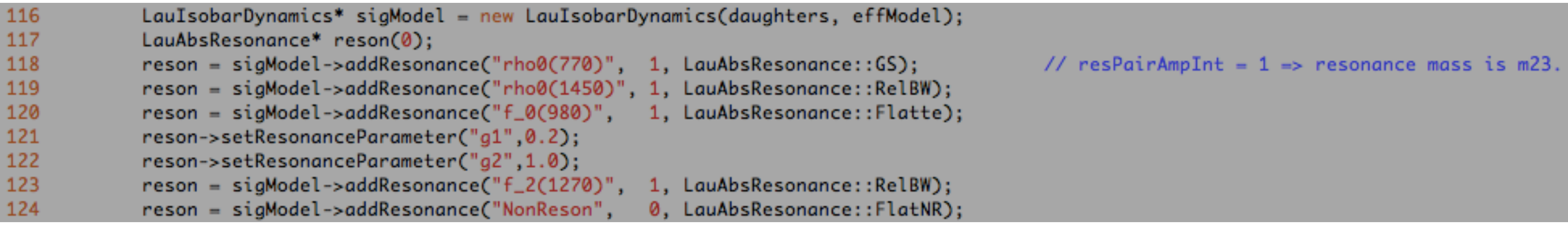


```
LauIsobarDynamics* sigModel = new LauIsobarDynamics(daughters, effModel);
LauAbsResonance* reson(0);
reson = sigModel->addResonance("rho0(770)",  1, LauAbsResonance::GS);          // resPairAmpInt = 1 => resonance mass is m23.
reson = sigModel->addResonance("rho0(1450)", 1, LauAbsResonance::RelBW);
reson = sigModel->addResonance("f_0(980)",   1, LauAbsResonance::Flatte);
reson->setResonanceParameter("g1",0.2);
reson->setResonanceParameter("g2",1.0);
reson = sigModel->addResonance("f_2(1270)",  1, LauAbsResonance::RelBW);
reson = sigModel->addResonance("NonReson",   0, LauAbsResonance::FlatNR);
```

- Here we add 4 resonances and a uniform nonresonant contribution
- The names of the resonances define their masses, width and spins
  - These can optionally be adjusted using LauAbsResonance::changeResonance
  - The resonance name definitions are given in LauResonanceMaker::createResonanceVector

Creating the signal model

- To define the signal DP model we do:



```
LauIsobarDynamics* sigModel = new LauIsobarDynamics(daughters, effModel);
LauAbsResonance* reson(0);
reson = sigModel->addResonance("rho0(770)",  1, LauAbsResonance::GS);         // resPairAmpInt = 1 => resonance mass is m23.
reson = sigModel->addResonance("rho0(1450)", 1, LauAbsResonance::RelBW);
reson = sigModel->addResonance("f_0(980)",   1, LauAbsResonance::Flatte);
reson->setResonanceParameter("g1",0.2);
reson->setResonanceParameter("g2",1.0);
reson = sigModel->addResonance("f_2(1270)",  1, LauAbsResonance::RelBW);
reson = sigModel->addResonance("NonReson",   0, LauAbsResonance::FlatNR);
```

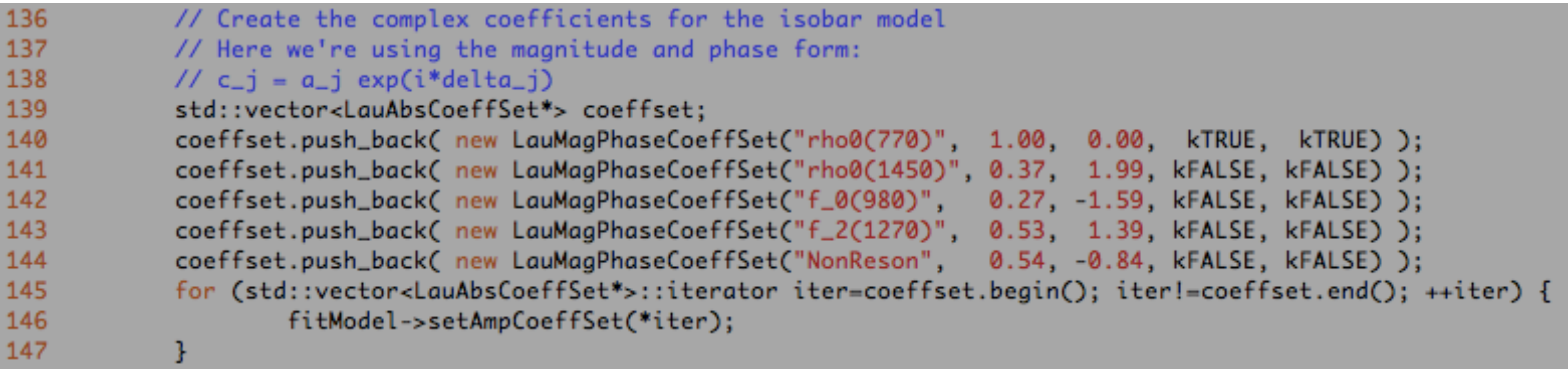
- The second argument defines which of the 3 daughters is the “bachelor” in the decay:

  parent -> resonance + bachelor

- The third argument defines the lineshape model to be used for the resonance

Assigning the c_j coefficients

- We create the fit model from the signal DP model and then assign the complex coefficients as follows:

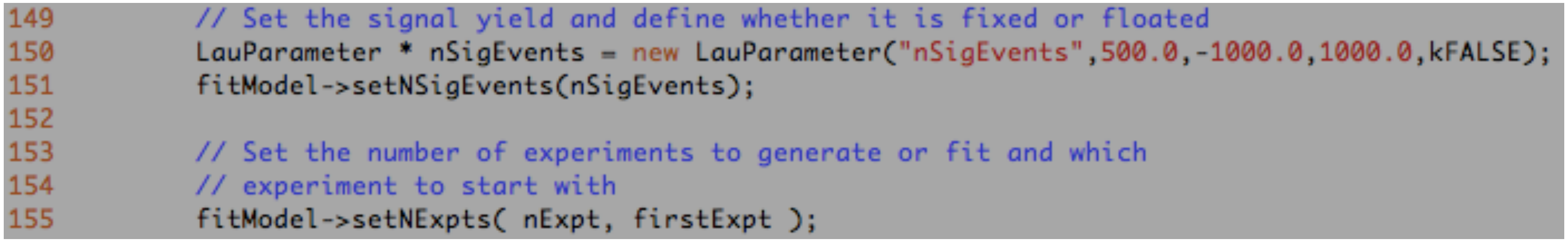


```
        // Create the complex coefficients for the isobar model
        // Here we're using the magnitude and phase form:
        // c_j = a_j exp(i*delta_j)
        std::vector<LauAbsCoeffSet*> coeffset;
        coeffset.push_back( new LauMagPhaseCoeffSet("rho0(770)",  1.00,  0.00,  kTRUE,  kTRUE) );
        coeffset.push_back( new LauMagPhaseCoeffSet("rho0(1450)", 0.37,  1.99, kFALSE, kFALSE) );
        coeffset.push_back( new LauMagPhaseCoeffSet("f_0(980)",   0.27, -1.59, kFALSE, kFALSE) );
        coeffset.push_back( new LauMagPhaseCoeffSet("f_2(1270)",  0.53,  1.39, kFALSE, kFALSE) );
        coeffset.push_back( new LauMagPhaseCoeffSet("NonReson",   0.54, -0.84, kFALSE, kFALSE) );
        for (std::vector<LauAbsCoeffSet*>::iterator iter=coeffset.begin(); iter!=coeffset.end(); ++iter) {
                fitModel->setAmpCoeffSet(*iter);
        }
```

- LauMagPhaseCoeffSet means that the two real values given are the magnitude and phase of the resonance
- Another possibility is to use LauRealImagCoeffSet
- One contribution, rho0(770), is the reference – the magnitudes and phases of the others are measured relative to that one

Setting the number of events

- To set the number of signal events and the number of toy experiments to perform, we do:



```
// Set the signal yield and define whether it is fixed or floated
LauParameter * nSigEvents = new LauParameter("nSigEvents",500.0,-1000.0,1000.0,kFALSE);
fitModel->setNSigEvents(nSigEvents);

// Set the number of experiments to generate or fit and which
// experiment to start with
fitModel->setNExpts( nExpt, firstExpt );
```

Building the executable

- To build the executable you simply need to issue the following command from within the examples directory:

```
make GenFit3pi
```

Running the toy generation

- To run the generation step do:

```
./GenFit3pi gen 100
```

- The “100” means to generate 100 experiments (try running simply “./GenFit3pi” to see all the options)
- Take a look at the gen-3pi.root file
- You have the generated DP coordinates plus some MC truth information (i.e. whether the event is a signal, background etc.) and info on which of the 100 experiments the events belong to
- Try drawing the DP and it’s projections:

```
genResults->Draw(“m23Sq:m13Sq”);
genResults->Draw(“m13”);
genResults->Draw(“cosHel13”,”m13>0.67 && m13<0.87”);
```

Running the fit

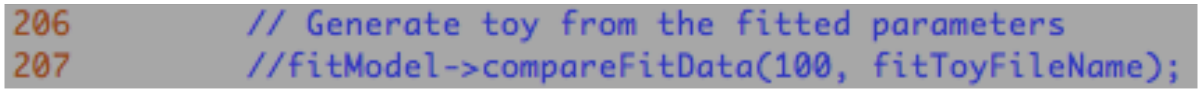
- To run the fitting step do:

```
./GenFit3pi fit 0 100
```

- (Will come back to the reason for the “0” arg)
- Take a look at the results file
- You have the negative log likelihood value, the fitted values and errors of all floating parameters, their correlations, etc.

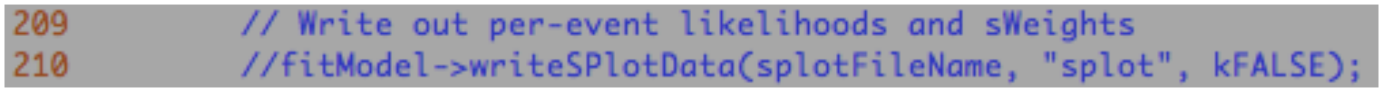
Plotting the fit results

- To plot the results of the fit it is necessary to generate toy MC from the fit results and to then plot those events
- The generation is done automatically if you uncomment the line:



```
        // Generate toy from the fitted parameters
        //fitModel->compareFitData(100, fitToyFileName);
```

- Similarly there is a line that will automate the writing out of sPlot information:

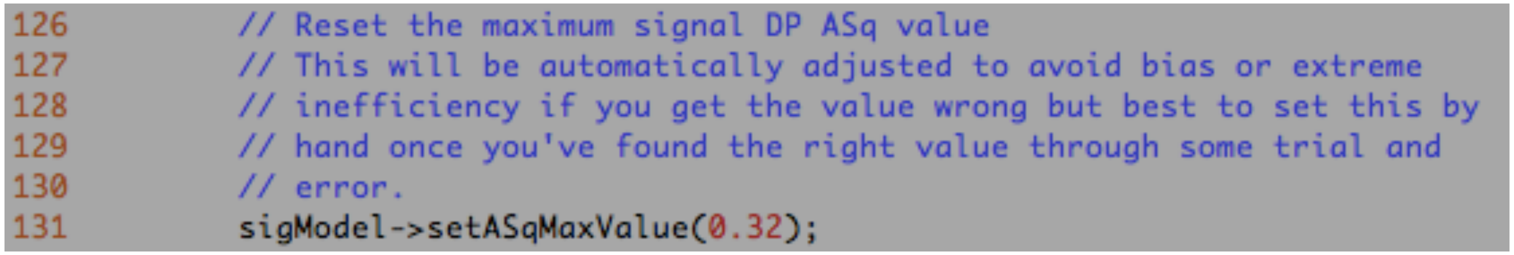


```
        // Write out per-event likelihoods and sWeights
        //fitModel->writeSPlotData(splotFileName, "splot", kFALSE);
```

- Have a play with these!

Some fiddly technical points – 1

- We skipped over this part of the code:



```
// Reset the maximum signal DP ASq value
// This will be automatically adjusted to avoid bias or extreme
// inefficiency if you get the value wrong but best to set this by
// hand once you've found the right value through some trial and
// error.
sigModel->setASqMaxValue(0.32);
```

- This sets the “ceiling” of the accept/reject for the toy MC generation
- If you get it badly wrong then Laura++ will adjust it automatically
- Best to do one very high stats run, initially setting this value to be rather small
- Then use the value that was found (plus a small safety margin) for future runs

Some fiddly technical points – 2

- When running the fit you had to give an argument “0”, which is simply used to name the output file (“fit3pi_0_expt_0-99.root”)
- If you run again changing this to “1” you’ll get another file called “fit3pi_1_expt_0-99.root”
- The reason for this is so that you can perform many fits with randomised starting values so that you can make sure to find the global minimum in what can be a very complicated parameter space
- Some code is provided in the examples directory that can extract the best fit from N (called ResultsExtractor)

Other examples

- There are several other (more complex) examples in the macros directory
- For example, GenFitKpipi.cc is a fit to both B^{+} and B^{-} samples (the likelihood is a function of the DP plus some other discriminating variables) to determine *CP* violation parameters in the presence of backgrounds, efficiency variation, etc.
- Other examples include the use of the K-matrix, using various forms of the c_j coeffs, using PDFs other than the DP, etc.