Introduction to the `AmpTools` Package using the Dalitz Tutorial

Ryan Mitchell

Revised: January 13, 2022

Abstract

This document introduces several of the core features of the `AmpTools` package using a simplified Dalitz analysis as an example. The Dalitz tutorial shows how to use the `AmpTools` package to generate Monte Carlo distributions, write amplitudes, fit data, and plot the results.

Contents

1 Introduction

The `AmpTools` package is a versatile set of tools that allows a user to define custom amplitudes, which can then be used to generate Monte Carlo distributions, to fit data, and to make plots, among other tasks. As the tools are both physics and experiment agnostic, the user has the freedom to write applications that are highly customizable. But in return, the user must provide a number of derived classes to write data to files (Section 4: `DalitzDataIO/DalitzDataWriter`), to read data from files (Section 6: `DalitzDataIO/DalitzDataReader`), to calculate amplitudes (Section 9: `DalitzAmp/BreitWigner`), and to plot fit results (Section 15: `DalitzPlot/DalitzPlotGenerator`). The Dalitz tutorial provides a simple example, from beginning to end, of how to write these derived classes, and how to then build executables from the resulting libraries.

The Dalitz tutorial considers a fictitious decay of a 3.0 GeV/c^2 particle to three 200 MeV/c^2 particles. Two-body Breit-Wigner distributions are used to illustrate amplitude construction.

This tutorial could be used as a starting point for new analyses. It is recommended that the steps presented here are followed consecutively to gradually build up an analysis from simply reading in data to fitting data and plotting the results.

2 Preliminaries

Only one external package is required to run the Dalitz tutorial: `ROOT`. Once `ROOT` is installed (directions for installation can be found elsewhere), the executable `root-config` should be available in the path. The make system will obtain information about the `ROOT` installation using `root-config`.

The latest fixed release of the `AmpTools` code, along with related packages and tutorials, can be cloned or downloaded as a tarball from `https://github.com/mashephe/AmpTools`. Once the source code is unpacked, it can be compiled from the top level with `make`. However, for compiling subsets of the tree it may be advantageous to set the following:

```
> setenv AMPTOOLS_HOME [top directory]
> setenv AMPTOOLS $AMPTOOLS_HOME/AmpTools
> setenv AMPPLOTTER $AMPTOOLS_HOME/AmpPlotter
> setenv DALITZ $AMPTOOLS_HOME/Tutorials/Dalitz
```

The `$AMPTOOLS` directory contains the source code for the `AmpTools` framework; `$AMPPLOTTER` contains code for a GUI to view fit results; and `$DALITZ` contains the Dalitz tutorial to be discussed in this document. Check the file `$AMPTOOLS_HOME/Makefile.settings` to verify the compiler settings look reasonable. To compile the entire package do:

```
> cd $AMPTOOLS_HOME
> make
```

3 Directory Structure

The Dalitz tutorial package contains a number of subdirectories. Three of these directories include class definitions that are derived from `AmpTools` classes and will be compiled into a static library associated with this example:

- `DalitzLib/DalitzDataIO`: This directory holds custom classes to read and write data. The `DalitzDataWriter` class is derived from the `DataWriter` class of `AmpTools`, and the `DalitzDataReader` class is derived from the `DataReader` class. They are used to read and write data in a custom `ROOT` tree format.
- `DalitzLib/DalitzAmp`: This directory holds user-defined amplitudes derived from the `Amplitude` class of `AmpTools`. The `BreitWigner` class is a simple example of such an amplitude.

- `DalitzLib/DalitzPlot`: This directory holds the `DalitzPlotGenerator` class, which is derived from the `PlotGenerator` class of `AmpTools`. This class holds instructions for making custom plots of fit results.

Other directories contain examples of how to use libraries to build and run executables:

- `DalitzExe`: This directory holds example code that can be used to build executables from the libraries generated above. Each executable is described in later sections.
- `run`: This directory is used to run executables.

Note that the `DalitzExe` and `run` directories could be separated from the other directories. The directory `run` contains no code to compile. When compiling from the top level, the products of `DalitzLib` are installed in `lib` and `DalitzExe` are installed in `bin`. To build the entire example one can run the following commands.

```
> cd $DALITZ
> make
```

To build only the executables, for example the `generatePhaseSpace` executable, the `Makefile` in the `DalitzExe` directory can be used:

```
> cd $DALITZ/DalitzExe
> make generatePhaseSpace
```

Alternatively, all of the executables (only) can be built at once:

```
> cd $DALITZ/DalitzExe
> make
```

If the executables are built by invoking make from within `$DALITZ/DalitzExe` then the build products will remain in that directory.

For executables that require MPI support, the name of the `*.cc` file should include the case-insensitive string "MPI". All `*.cc` files in the `DalitzExe` directory that have "MPI" somewhere in the name will be built as an executable with MPI support provided that `MPI=1` is used as an argument to `make`.

4 Setting up a Data Writer: DalitzDataIO/DalitzDataWriter

The main purpose of a DataWriter class in AmpTools is to take a Kinematics object, which contains four-vectors, and write its contents to an output file. The user must write a class derived from the DataWriter class to customize the output, including the format of the output file, the variables to be written, and so on. In the DalitzDataWriter example, output files are in a ROOT tree format and contain the four-vectors of the three fictitious final state particles of the Dalitz tutorial.

5 Generating Phase Space: DalitzExe/generatePhaseSpace

The generatePhaseSpace application is an example of how to use the DalitzDataWriter to write data to an output file. It makes use of the TGenPhaseSpace class from ROOT. A fictitious decay of a particle with mass 3.0 GeV/c^2 to three 200 MeV/c^2 particles is used for illustration. To compile and run the example, use:

```
> cd $DALITZ/DalitzExe
> make generatePhaseSpace_exe
> cd $DALITZ/run
> $DALITZ/DalitzExe/generatePhaseSpace phasespace.gen.root 100000
```

This will make a file called phasespace.gen.root with 100000 events generated according to phase space.

6 Setting up a Data Reader: DalitzDataIO/DalitzDataReader

The DataReader class in AmpTools is designed to read data from an input file and pack it into a Kinematics object, which can then be distributed to other classes or applications. The user must write a class deriving from the DataReader class to customize the data format (which should match the format written by the DataWriter). The DalitzDataReader is an example that can read the output files from the DalitzDataWriter.

7 Making Plots: DalitzExe/plotData

The `plotData` executable is an example that uses the `DalitzDataReader` to read events from an input file and then makes histograms of kinematic variables using `ROOT`. To compile and run, use:

```
> cd $DALITZ/DalitzExe
> make plotData_exe
> cd $DALITZ/run
> $DALITZ/DalitzExe/plotData phasespace.gen.root plots.phasespace.gen.root
```

This will read the input file `phasespace.gen.root` (generated in Section 5) and output a file of histograms called `plots.phasespace.gen.root`. Histograms can be viewed in `ROOT` using, for example:

```
> cd $DALITZ/run
> root plots.phasespace.gen.root
root>  hs12s23->Draw("colz")
```

The resulting histogram is shown in Figure 1a.

8 Simulating Detector Effects: DalitzExe/toyAcceptance

The `toyAcceptance` example uses the `DalitzDataReader` to read events from an input file, accepts or rejects events based on a toy efficiency function, then writes the accepted events to an output file using the `DalitzDataWriter`. Here is an example of how to compile and run:

```
> cd $DALITZ/DalitzExe
> make toyAcceptance_exe
> cd $DALITZ/run
> $DALITZ/DalitzExe/toyAcceptance phasespace.gen.root phasespace.acc.root
```

This takes the input file `phasespace.gen.root` (generated in Section 5) and outputs a file called `phasespace.acc.root` containing events that survive the toy simulation.

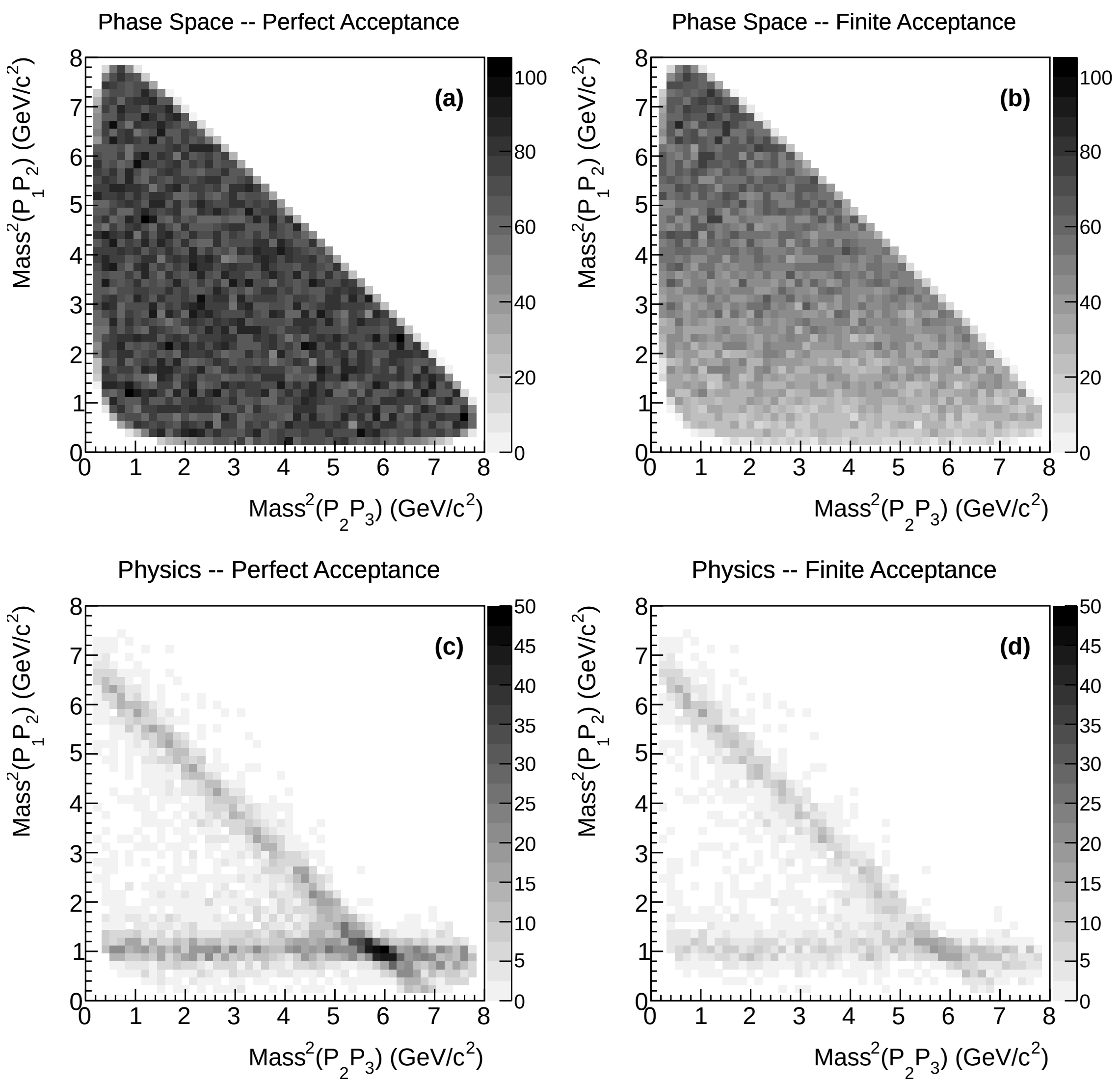

Figure 1: Dalitz distributions for a 3.0 GeV/c^2 particle decaying to three 200 MeV/c^2 particles. Plots were generated using the `plotData` executable with input data files described in the text.

To view the resulting kinematic distributions, one can use the `plotData` application described in Section 7:

```
> cd $DALITZ/run
> $DALITZ/DalitzExe/plotData phasespace.acc.root plots.phasespace.acc.root
```

The resulting Dalitz plot is shown in Figure 1b.

9 Setting up an Amplitude: `DalitzAmp/BreitWigner`

One of the most important aspects of the `AmpTools` package is the flexibility it offers in defining physics amplitudes. The user must write amplitude classes that derive from the `Amplitude` class. The `BreitWigner` class (found in the `DalitzAmp` directory) is a simple example. Pay special attention to the `calcAmplitude` method, where the amplitude is actually calculated.

10 Setting up a Configuration File: `run/dalitz1.cfg`

The `AmpTools` package uses the `AmplitudeManager` class to manage amplitudes. As there are many steps in initializing an `AmplitudeManager`, configuration files (parsed by the `ConfigFileParser` class) can be used to simplify the process. Configuration files can be used to define which amplitudes should be used, how they should be combined (coherently or incoherently, for example), what the free parameters are, their initial values, and so on. A simple example is the `dalitz1.cfg` file found in the `run` directory of the Dalitz tutorial. This example defines two Breit-Wigner amplitudes: one of mass 1.0 GeV/c^2 decaying to particles 1 and 2; and one of mass 1.5 GeV/c^2 decaying to particles 1 and 3. See the top of the `dalitz1.cfg` file for a list of commands that can be used in configuration files.

11 Parsing a Configuration File: `DalitzExe/parseConfigFile`

The `parseConfigFile` application is a simple example showing how to parse a configuration file using the `ConfigFileParser` class, and then display the configuration information to the screen. This is a useful way to make sure there are no mistakes in a configuration file. To compile and run, use:

```
> cd $DALITZ/DalitzExe
> make parseConfigFile_exe
> cd $DALITZ/run
> $DALITZ/DalitzExe/parseConfigFile dalitz1.cfg
```

This parses and displays information from the dalitz1.cfg file found in the run directory. The dalitz3.cfg file is a more complicated configuration file that includes additional free parameters:

```
> $DALITZ/DalitzExe/parseConfigFile dalitz3.cfg
```

12 Printing Amplitude Values: DalitzExe/printAmplitudes

The printAmplitudes example parses a configuration file using the ConfigFileParser, reads in events using the DalitzDataReader, sets up an AmpToolsInterface, calculates BreitWigner amplitudes and total intensities, and then prints amplitude and intensity values to the screen. This can be a useful way to make sure amplitudes are coded correctly. Compile and run in the usual way:

```
> cd $DALITZ/DalitzExe
> make printAmplitudes_exe
> cd $DALITZ/run
> $DALITZ/DalitzExe/printAmplitudes dalitz1.cfg phasespace.gen.root
```

This takes as input the dalitz1.cfg configuration file (Section 10) and the data file phasespace.gen.root (Section 5).

13 Generating Physics Distributions: DalitzExe/generatePhysics

The generatePhysics example generates events according to amplitudes specified in a configuration file. To do so, it parses a configuration file using the ConfigFileParser, sets up an AmpToolsInterface, generates phase space events using the TGenPhaseSpace class from ROOT, calculates intensities from those events and uses an accept/reject method to mimic the desired physics distribution, then writes the resulting events to a data file using a DalitzDataWriter. To compile and run, use:

```
> cd $DALITZ/DalitzExe
> make generatePhysics_exe
> cd $DALITZ/run
> $DALITZ/DalitzExe/generatePhysics dalitz1.cfg physics.gen.root 100000
```

This inputs the `dalitz1.cfg` configuration file (Section 10), outputs a data file called `physics.gen.root`, and does accept/reject starting with 100000 phase space events.

The `toyAcceptance` executable can be used on the output file to simulate the effects of a finite acceptance:

```
> $DALITZ/DalitzExe/toyAcceptance physics.gen.root physics.acc.root
```

Finally, histograms can be produced from both the generated and accepted data files using the `plotData` executable:

```
> $DALITZ/DalitzExe/plotData physics.gen.root plots.physics.gen.root
> $DALITZ/DalitzExe/plotData physics.acc.root plots.physics.acc.root
```

The results are shown in Figures 1c and 1d, respectively.

14 Fitting Amplitudes: `DalitzExe/fitAmplitudes`

Amplitudes can be fit to data following the `fitAmplitudes` example. The example shows how to set up an `AmpToolsInterface` class, perform a fit, and then output results to a text file:

```
> cd $DALITZ/DalitzExe
> make fitAmplitudes_exe
> cd $DALITZ/run
> $DALITZ/DalitzExe/fitAmplitudes dalitz1.cfg
```

This takes a configuration file `dalitz1.cfg` as input and outputs a text file with results called `dalitz1.fit`. The configuration file specifies that the fit is done to data in the file `physics.acc.root` (Section 13), and the acceptance is evaluated using the files `phasespace.gen.root` (Section 5) and `phasespace.acc.root` (Section 8).

Fits using other example configuration files can also be performed. The `dalitz2.cfg` configuration file corresponds to a case with perfect acceptance. The `dalitz3.cfg` configuration file lets the masses and widths of the resonances float.

15 Setting up a Plot Generator: DalitzPlot/DalitzPlotGenerator

To use the results of a fit to make plots, a user can write a class that derives from the `PlotGenerator` class of the `AmpTools` package. The `DalitzPlotGenerator` (found in the `DalitzPlot` directory) is an example of how to do this. Plots are generated for data, acceptance-corrected fit results, and acceptance-uncorrected fit results. To visualize the quality of the fit over certain projections of the data, one can compare the data plots with the acceptance-uncorrected fit plots. Plots of fit results using only single amplitudes are also constructed and can be used to visualize the contributions of individual amplitudes.

16 Plotting Fit Results: DalitzExe/plotResults

The `plotResults` application uses the `DalitzPlotGenerator` to make `ROOT` histograms from fit results. Compile and run it like this:

```
> cd $DALITZ/DalitzExe
> make plotResults_exe
> cd $DALITZ/run
> $DALITZ/DalitzExe/plotResults dalitz1.fit dalitz1.root
```

This takes as input the `dalitz1.fit` text file containing fit results (Section 14). The output is `dalitz1.root`, a `ROOT` file containing histograms.

To plot data with fit results overlaid, one can use, for example:

```
> root dalitz1.root
root>  hm12dat->Draw("e")
root>  hm12acc->Draw("hist,same")
root>  hm12acc1->Draw("hist,same")
root>  hm12acc2->Draw("hist,same")
```

The `hm12acc1` and `hm12acc2` histograms show the contributions of the first and second amplitudes, respectively. Similarly, for the acceptance-corrected results, one could use:

```
> root dalitz1.root
root>  hm12gen->Draw("hist")
```

```
root>  hm12gen1->Draw("hist,same")
root>  hm12gen2->Draw("hist,same")
```

Results are shown in Figure 2.

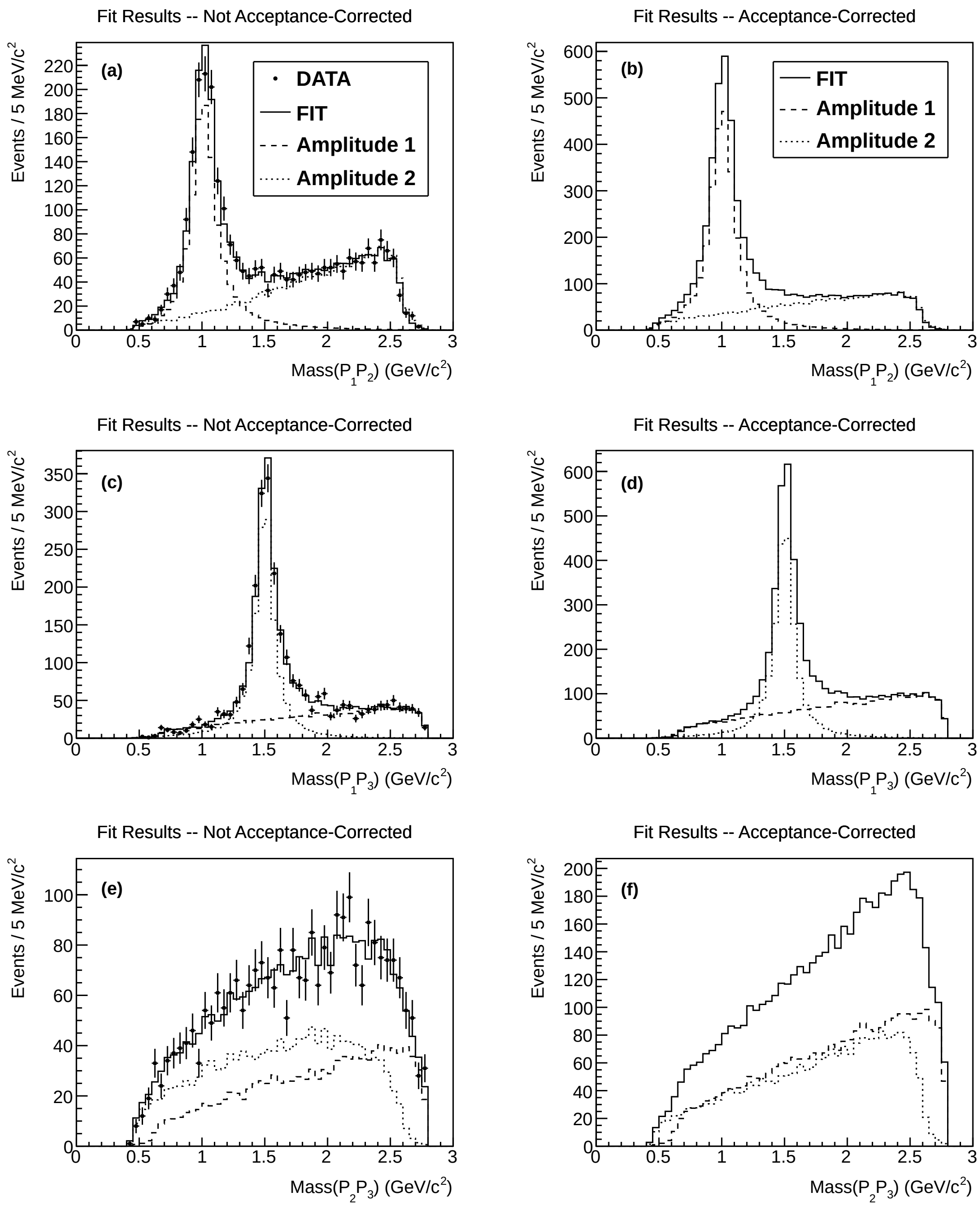

Figure 2: Fit results. The left column shows a comparison of data with fit results not corrected for acceptance effects. The right column shows the acceptance-corrected fit results. (a-b) Projection onto $M(P_1P_2)$. (c-d) Projection onto $M(P_1P_3)$. (e-f) Projection onto $M(P_2P_3)$.