

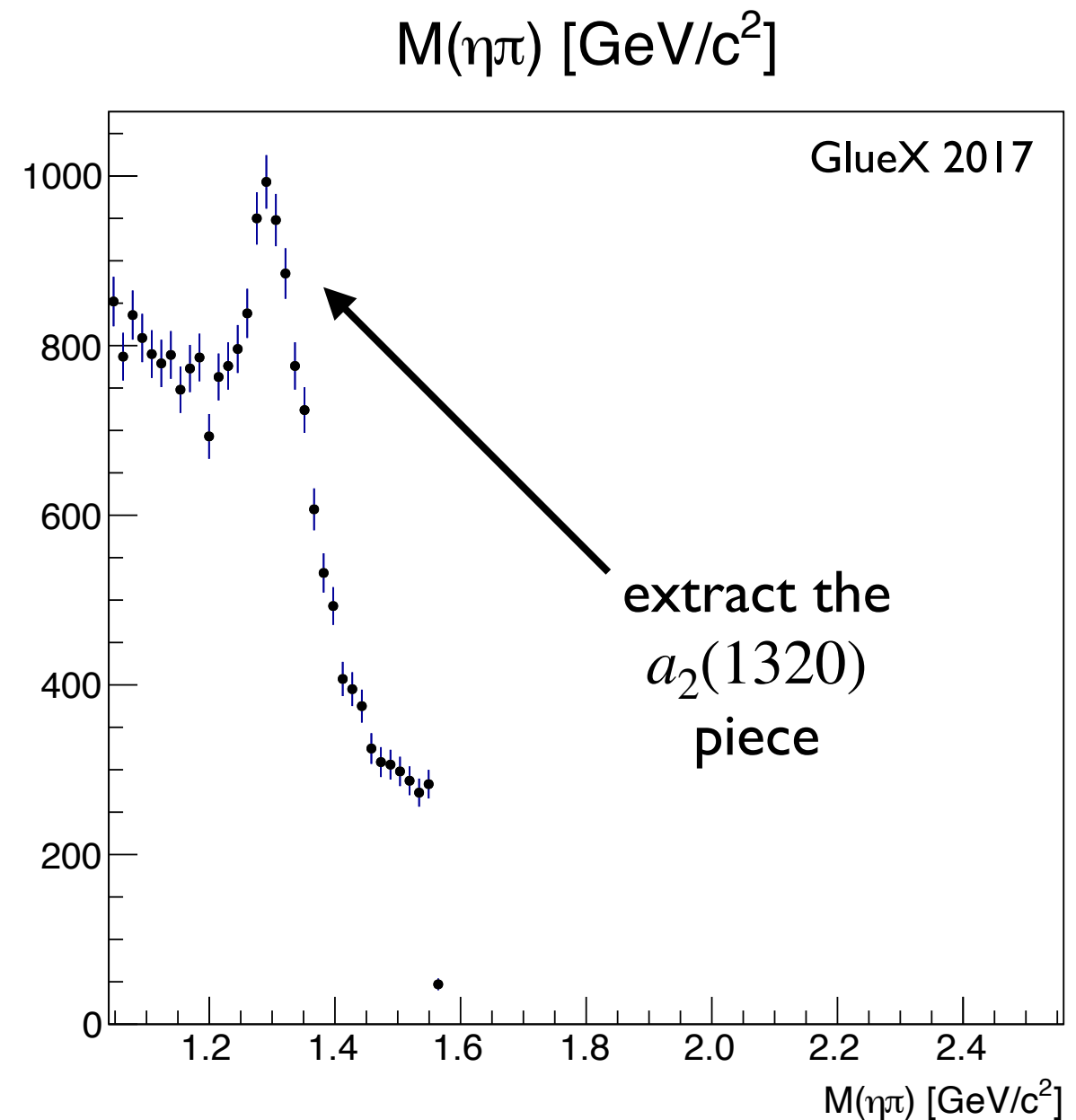
Using AmpTools to Extract the $a_2(1320)$ Yield from GlueX 2017 Data

GlueX Software Tutorial
May 23, 2022

Matthew Shepherd
Indiana University

Example Fit Goal: Extract a_2 from $\eta\pi^0$

- Use data sample from previous exercise
- The kinematics of $\eta\pi^0$ system (angular distributions) are given by the Z_ℓ^m functions discussed in the morning session
- Dynamical assumptions about how the amplitude depends on $M(\eta\pi^0)$
 - D-waves: a complex-valued Breit-Wigner with parameters consistent with the $a_2(1320)$
 - certain m-projections/reflectivities excluded based on tensor meson dominance (TMD) model
 - S-waves: fixed complex number in four coarse bins of $M(\eta\pi^0)$ (a piecewise complex function)
- Goal: extract the total efficiency corrected yield of $a_2(1320)$ (the coherent sum of all D-waves)



Required Code: AmpTools

- AmpTools: library that is distributed independently of GlueX code base
 - available through GitHub: github.com/mashephe/AmpTools
 - installed as external package in GlueX environment
 - (include version.xml lines here)
- The AmpTools library (not an executable) that provides a general interface for doing fits
 - fully functional example of how to use AmpTools is also provided in [Tutorials/Dalitz](#)
 - [copyDalitz.py](#) is provided to generate your own project based on the Dalitz tutorial (not needed for a typical GlueX user)
 - GlueX has a set of libraries and executables that rely on the core AmpTools package
 - [halld_sim/src/libraries/AMPTOOLS_...](#)
 - [halld_sim/src/programs/AmplitudeAnalysis](#)
 - Documentation concerning the theory of operation is available in the Git repository
 - AmpTools "knows" nothing about GlueX data format, physics, etc.
- Issue tracking system on GitHub is used -- report AmpTools problems there if they pertain to the core package and not the GlueX implementation

Required Code: Amplitudes

- Provide AmpTools with a method (code via a library) to convert the four-vectors of an event to a complex number
 - inherits from the Amplitude class in AmpTools which defines the interface for the object -- a template is provided to handle some necessary functions
 - see [Tutorials/Dalitz/DalitzLib/DalitzAmp/BreitWigner.h](#)
 - accepts arguments as an arbitrarily long list of strings (which will be specified in the config file)
- A collection of GlueX related amplitudes is here:
 - [halld_sim/src/libraries/AMPTOOLS_AMPS](#)
- Several optional features:
 - embed floating fit parameters (e.g., BreitWigner mass) into the calculation of the amplitude
 - perform a data reduction step to reduce four-vectors to "user variables," e.g., angles or Lorentz invariants, that are used to compute the amplitude
 - GPU acceleration of amplitude calculation -- requires additional code
 - *use of features increases complexity but can optimize performance -- not a one-size-fits-all solution: ask for advice if you considering additional development to make fits run faster*
- The executable you write knows about the existence of the amplitudes through the static registration methods of the AmpToolsInterface
 - [AmpToolsInterface::registerAmplitude\(BreitWigner\(\)\);](#)
 - register before creating an instance of AmpToolsInterface to do your fit, generate MC, ...

Required Code: DataReader

- Provide AmpTools with a class that is able to turn a file on disk into a set of four-vectors
 - similar to Amplitude class: inherits from DataReader, uses a template for some key functions, and accepts arguments as a list of strings
 - see example in Tutorials/Dalitz
- Not usually analysis specific, but more specific to the file format
 - GlueX collection of data readers is here: halld_sim/src/libraries/AMPTOOLS_DATAIO
- Common GlueX formats for AmpTools input:
 - standard ROOT tree from tree_to_amptools : ROOTDataReader
 - FSRoot format tree: FSRootDataReader
- Complex functionality can be added:
 - perform filtering or cuts during read into AmpTools
 - bootstrap: random sample with oversampling to evaluate uncertainties
- Not all components of a fit need to use the same data reader
- Like amplitudes, readers need to be registered prior to use:
 - AmpToolsInterface::registerDataReader(DalitzDataReader());

Configuration File: General Remarks

- See sample: [Tutorials/Dalitz/run/dalitz3.cfg](#)
- all lines begin with a keyword that informs the parser how to process the rest of the line
 - no continuation character: put it all on one line
 - ordering of the lines is not important
- useful keywords for organizing files:
 - `include <file>`
 - `define <word> (defn1) (defn2) (defn3) ...`
 - `<word>` must be isolated (spaces on each side) to be replaced with one or more words
 - `loop <word> <value1> (value2) (value3) ...`
 - any line containing `<word>` will be repeated replacing `<word>` with `<value1>`, `(value2)`, ...
 - multiple loops can be in a single line but they must be of the same length N -- then the line is repeated N times stepping through all loops in sync simultaneously
- some special syntax:
 - `#` as the first character denotes a comment
 - `::` is treated like a space
 - `[parname]` -- use square brackets when the name of a parameter (instead of a numerical value) should be passed as an argument to an amplitude

Reactions, sums, and amplitudes

- Within a reaction, the intensity must be defined as a sum of coherent sums of amplitudes, where each amplitude can be a product of factors

$$\mathcal{I}(\mathbf{x}) = \sum_{\sigma} \left| \sum_{\alpha} s_{\sigma,\alpha} V_{\sigma,\alpha} A_{\sigma,\alpha}(\mathbf{x}) \right|^2$$

$$A_{\sigma,\alpha}(\mathbf{x}) = \prod_{\gamma=1}^{n_{\sigma,\alpha}} a_{\sigma,\alpha,\gamma}(\mathbf{x})$$

- Amplitudes can be scaled by a real number $s_{\sigma,\alpha}$ (default 1) and have a complex production coefficient $V_{\sigma,\alpha}$
- This matches the general form for $\eta\pi$ production:

$$I(\Omega, \Phi) = 2\kappa \sum_k \left\{ (1 - P_\gamma) \left| \sum_{\ell,m} [\ell]_{m;k}^{(-)} \text{Re}[Z_\ell^m(\Omega, \Phi)] \right|^2 + (1 - P_\gamma) \left| \sum_{\ell,m} [\ell]_{m;k}^{(+)} \text{Im}[Z_\ell^m(\Omega, \Phi)] \right|^2 + \right. \\ \left. (1 + P_\gamma) \left| \sum_{\ell,m} [\ell]_{m;k}^{(+)} \text{Re}[Z_\ell^m(\Omega, \Phi)] \right|^2 + (1 + P_\gamma) \left| \sum_{\ell,m} [\ell]_{m;k}^{(-)} \text{Im}[Z_\ell^m(\Omega, \Phi)] \right|^2 \right\}.$$

- We absorb $\sqrt{1 \pm P_\gamma}$ into the definition of Z_ℓ^m and write $[\ell]_{m;k}^{(\pm)}$ as either a BreitWigner (for $\ell = 2$) or piecewise-defined function (for $\ell = 0$): [BreitWigner.cc](#), [Zlm.cc](#), and [Piecewise.cc](#)
- Note that one $[\ell]_{m;k}^{(\pm)}$ appears in two sums: the production coefficients for each must be constrained to be the same and both terms must be included when computing anything physical from the result
- Multiple reactions are like doing multiple fits simultaneously: contributions to $\ln(L)$ add, parameters can be constrained across reactions

Example: Configuring Inputs

```
#####  
# GLOBAL VARIABLES  
#####
```

```
fit etapi0_SD_TMD_onePol
```

```
define polVal_00 0.3519  
define polAngle_00 0.0
```

```
define atwo 1.312 0.113
```

```
include starting_params.cfg
```

```
#####  
# SETUP INPUT, REACTIONS, SUMS  
#####
```

```
reaction EtaPi0_00 Beam Proton Eta Pi0
```

```
data EtaPi0_00 FSRootDataReader fsroot/tree_pi0eta__B4_M17_M7_DATA_sp17_pol0_SIGNAL_SKIM_A2.root ntFSGlueX_101_1 3
```

```
bkgnd EtaPi0_00 FSRootDataReader fsroot/tree_pi0eta__B4_M17_M7_DATA_sp17_pol0_SIDEBANDS_SKIM_A2.root ntFSGlueX_101_1 3  
fsroot/tree_pi0eta__B4_M17_M7_DATA_sp17_pol0_SIDEBANDS_SKIM_A2.root.weight ntFSGlueX_101_1_weight weight
```

```
accmc EtaPi0_00 FSRootDataReader fsroot/tree_pi0eta__B4_M17_M7_MC_sp17_pol0_SIGNAL_SKIM_A2.root ntFSGlueX_101_1 3  
fsroot/tree_pi0eta__B4_M17_M7_MC_sp17_pol0_SIGNAL_SKIM_A2.root.weight ntFSGlueX_101_1_weight weight
```

```
genmc EtaPi0_00 FSRootDataReader fsroot/tree_pi0eta__B4_M17_M7_MCGEN_sp17_pol0_GENERAL_SKIM_A2.root ntFSGlueX_101_1 3 MC
```

data: signal region events usually with unity weight, may contain backgrounds

bkgnd: (often) weighted sample that is statistically consistent with the background contribution to the signal region (GlueX: weighted RF sidebands + others)

accmc: accepted signal MC, consistent with the signal portion of the data sample (GlueX: remove beam accidentals)

genmc: generated MC, used for denominator in efficiency calculations be mindful of M, t regions, branching fractions, etc.; (GlueX: this should include the tagger efficiency)

reaction: number of particles matches number of four-vectors provided by reader; AmpTools doesn't care about particle names unless they are the same, then the amplitude is symmetrized

n.b.: other GlueX software, e.g., gen_amp, may use these names



Example: Setting Up Amplitudes

```
sum EtaPi0_00 ReZ_1-P
sum EtaPi0_00 ImZ_1+P
sum EtaPi0_00 ReZ_1+P
sum EtaPi0_00 ImZ_1-P
```

$$I(\Omega, \Phi) = 2\kappa \sum_k \left\{ (1 - P_\gamma) \left| \sum_{\ell, m} [\ell]_{m; k}^{(-)} \text{Re}[Z_\ell^m(\Omega, \Phi)] \right|^2 + (1 - P_\gamma) \left| \sum_{\ell, m} [\ell]_{m; k}^{(+)} \text{Im}[Z_\ell^m(\Omega, \Phi)] \right|^2 + \right. \\ \left. (1 + P_\gamma) \left| \sum_{\ell, m} [\ell]_{m; k}^{(+)} \text{Re}[Z_\ell^m(\Omega, \Phi)] \right|^2 + (1 + P_\gamma) \left| \sum_{\ell, m} [\ell]_{m; k}^{(-)} \text{Im}[Z_\ell^m(\Omega, \Phi)] \right|^2 \right\}.$$

```
#####
# DEFINE AMPLITUDES
#####
```

S-wave amplitudes

```
amplitude EtaPi0_00::ReZ_1-P::S0- Zlm 0 0 +1 -1 polAngle_00 polVal_00
amplitude EtaPi0_00::ImZ_1+P::S0- Zlm 0 0 -1 +1 polAngle_00 polVal_00
amplitude EtaPi0_00::ReZ_1-P::S0- Piecewise 1.04 1.56 4 23 Neg ReIm [pcwsBin_0ReNeg] [pcwsBin_0ImNeg]
[pcwsBin_1ReNeg] [pcwsBin_1ImNeg] [pcwsBin_2ReNeg] [pcwsBin_2ImNeg] [pcwsBin_3ReNeg] [pcwsBin_3ImNeg]
amplitude EtaPi0_00::ImZ_1+P::S0- Piecewise 1.04 1.56 4 23 Neg ReIm [pcwsBin_0ReNeg] [pcwsBin_0ImNeg]
[pcwsBin_1ReNeg] [pcwsBin_1ImNeg] [pcwsBin_2ReNeg] [pcwsBin_2ImNeg] [pcwsBin_3ReNeg] [pcwsBin_3ImNeg]
```

```
amplitude EtaPi0_00::ImZ_1-P::S0+ Zlm 0 0 -1 -1 polAngle_00 polVal_00
amplitude EtaPi0_00::ReZ_1+P::S0+ Zlm 0 0 +1 +1 polAngle_00 polVal_00
amplitude EtaPi0_00::ImZ_1-P::S0+ Piecewise 1.04 1.56 4 23 Pos ReIm [pcwsBin_0RePos] [pcwsBin_0ImPos]
[pcwsBin_1RePos] [pcwsBin_1ImPos] [pcwsBin_2RePos] [pcwsBin_2ImPos] [pcwsBin_3RePos] [pcwsBin_3ImPos]
amplitude EtaPi0_00::ReZ_1+P::S0+ Piecewise 1.04 1.56 4 23 Pos ReIm [pcwsBin_0RePos] [pcwsBin_0ImPos]
[pcwsBin_1RePos] [pcwsBin_1ImPos] [pcwsBin_2RePos] [pcwsBin_2ImPos] [pcwsBin_3RePos] [pcwsBin_3ImPos]
```

D-wave amplitudes

```
amplitude EtaPi0_00::ImZ_1-P::a2_D0+ Zlm 2 0 -1 -1 polAngle_00 polVal_00
amplitude EtaPi0_00::ReZ_1+P::a2_D0+ Zlm 2 0 +1 +1 polAngle_00 polVal_00
amplitude EtaPi0_00::ImZ_1-P::a2_D0+ BreitWigner atwo 2 2 3
amplitude EtaPi0_00::ReZ_1+P::a2_D0+ BreitWigner atwo 2 2 3
```

amplitude factors with the same reaction::sum::amplitude are multiplied together



Example: Constraints and Initialization

- We need to constrain the production coefficients ($V_{\sigma,\alpha}$) in front of the specific $[\ell]_m^{(\pm)}$ amplitude that appears in two coherent sums:

```
constrain EtaPi0_00::ImZ_1-P::S0+ EtaPi0_00::ReZ_1+P::S0+
constrain EtaPi0_00::ReZ_1-P::S0- EtaPi0_00::ImZ_1+P::S0-
constrain EtaPi0_00::ImZ_1-P::a2_D0+ EtaPi0_00::ReZ_1+P::a2_D0+
constrain EtaPi0_00::ReZ_1-P::a2_D0- EtaPi0_00::ImZ_1+P::a2_D0-
```

- Initialize (optional) production coefficients with initialized commands -- only need to initialize one of the constrained amplitudes

```
initialize EtaPi0_00::ReZ_1-P::S0- cartesian 1 0 fixed
initialize EtaPi0_00::ReZ_1+P::S0+ cartesian 1 0 fixed
initialize EtaPi0_00::ReZ_1-P::a2_D0- cartesian 27.2 26.9
initialize EtaPi0_00::ReZ_1+P::a2_D0+ cartesian -12.2 19.9
```

- in our example the freedom in the S-wave is in the parameters that make up the piecewise function -- we keep the lead coefficient fixed to one
- Generally required: one of the production coefficients in each coherent sum should be initialized with the flag "real" to avoid overall phase ambiguities
- Parameters are declared and initialized in one line:

```
parameter pcwsBin_0ImNeg -307.092917227354
parameter pcwsBin_0ImPos 423.787950826515
parameter pcwsBin_0ReNeg -29.1570576067053
parameter pcwsBin_0RePos 343.007639956881
parameter pcwsBin_1ReNeg -25.8844360633494
parameter pcwsBin_1RePos 340.647573417975
```

```
parameter pcwsBin_1ImPos 0.0 fixed
parameter pcwsBin_1ImNeg 0.0 fixed
```

technical note: these two things together coupled with the fact we are using a Breit-Wigner function in the D wave remove the phase ambiguity

Extending to Multiple Polarization States

- We treat each polarization state as an independent "reaction"
 - data are statistically independent; similar to a coupled-channel or multi-experiment fit, which are ideas supported in AmpTools
 - minimize a global sum of $-2 \ln(L)$
 - constrain amplitudes across reactions up to a floating scale parameter of $\mathcal{O}(1)$ that absorbs differences in integrated luminosity for each data set
 - accepted and generated MC can (likely) be reused for each reaction

- Extend our fit configuration using the AmpTools looping structure:

```
loop LOOPREAC EtaPi0_00 EtaPi0_45 EtaPi0_90 EtaPi0_135
loop LOOPPOLANG polAngle_00 polAngle_45 polAngle_90 polAngle_135
loop LOOPPOLVAL polVal_00 polVal_45 polVal_90 polVal_135
```

which effectively creates four copies of many lines of the config file:

```
reaction LOOPREAC Beam Proton Eta Pi0
amplitude LOOPREAC::ReZ_1-P::S0- Zlm 0 0 +1 -1 LOOPPOLANG LOOPPOLVAL
```

- See full example in: [etapi0_SD_TMD_allPol.cfg](#)

Running the Fit and Visualizing the Result

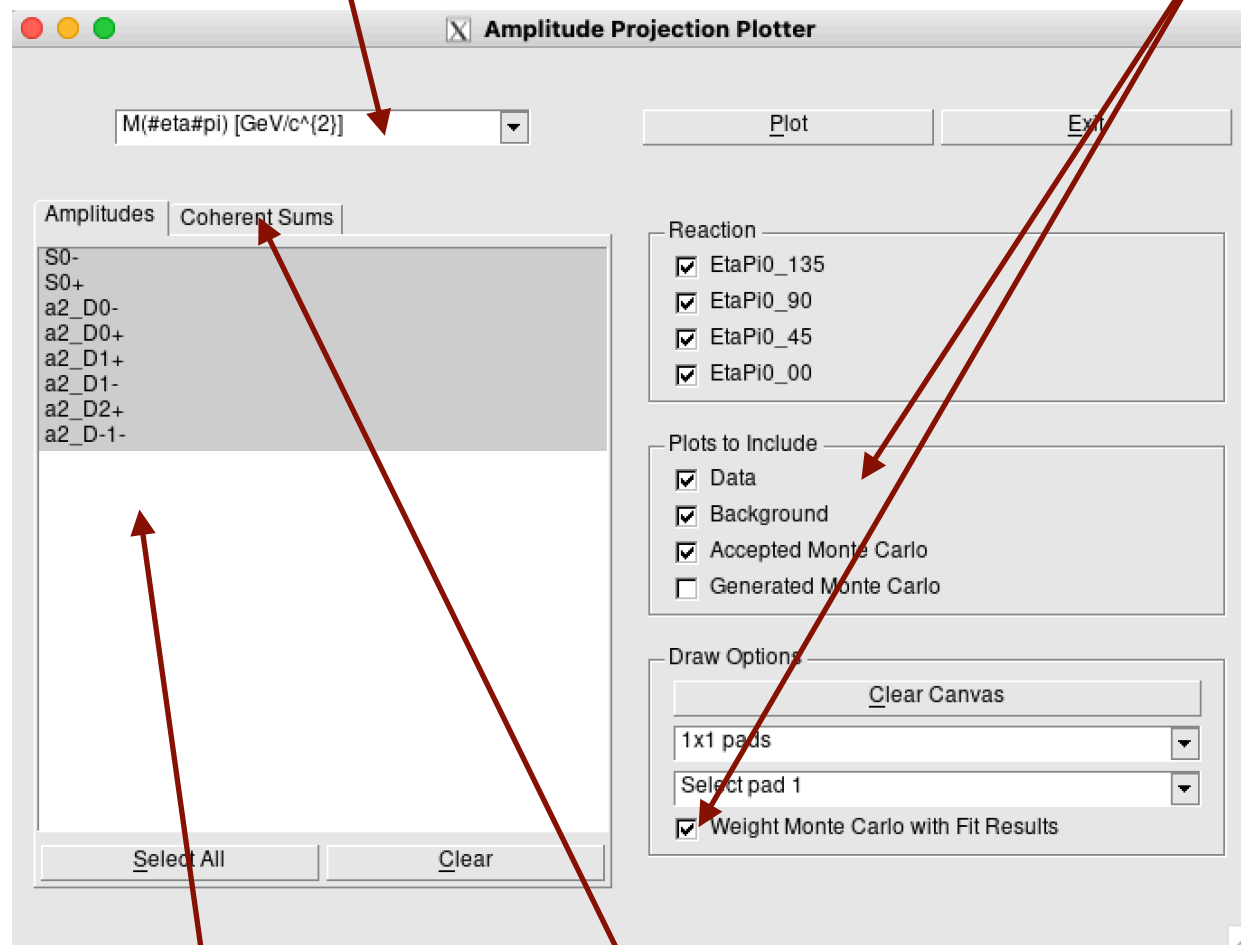
- To run a single core fit interactively invoke fit (source code)
 - run `fit -h` to see options
 - typical: `fit -c etapi0_SD_TMD_allPol.cfg` (wait for it.... wait for it... ...still waiting...)
 - output: `etapi0_SD_TMD_allPol.fit`
- Visualization requires the creation of PlotGenerator class to fill histograms
 - see example [DalitzPlotGenerator](#) in the AmpTools Dalitz Tutorial
 - collection of GlueX plot generators: [halld_sim/libraries/AMPTOOLS_DATAIO](#)
 - for this tutorial see: [etaPiPlotGenerator](#)
- And you need an executable that uses this class to generate the plots you want
 - see AmpTools Dalitz Tutorial example: [plotResults](#)
 - good for batch processing of many plots -- non-interactive
 - plots of GlueX data that follow were made with: [plot_etapi0](#) (in `halld_sim`)
 - uses AmpPlotter library distributed with AmpTools to provide a GUI front-end to the underlying PlotGenerator
 - good for interactively exploring the amplitudes in a single fit
 - starts ROOT-based X11 session: best to run locally.... which, for now, requires a local build of `halld_sim` 😞

Viewing the Output

pick a plot to generate

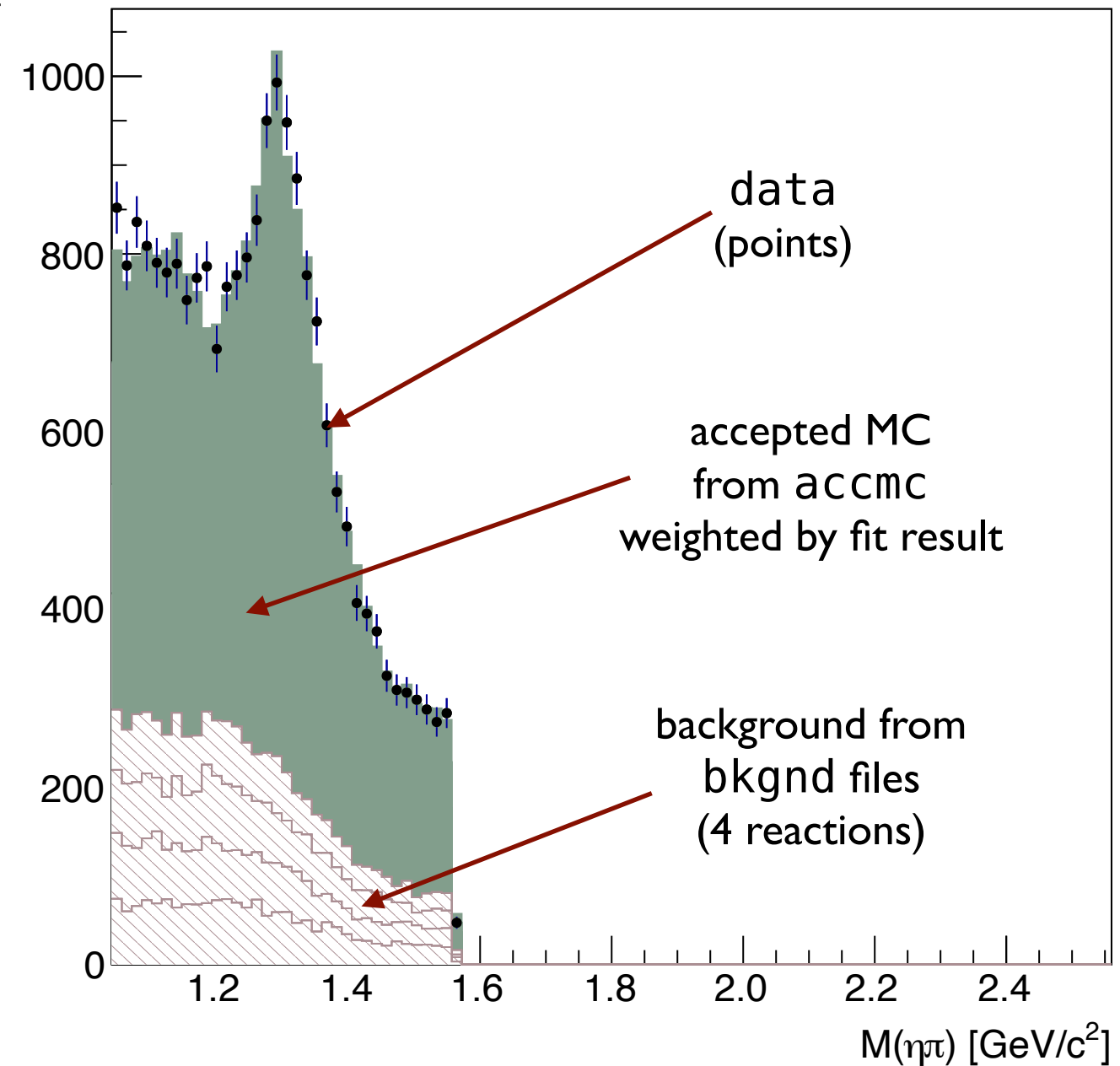
common configuration
for comparing fit to
the data

$M(\eta\pi)$ [GeV/c²]



select amplitudes

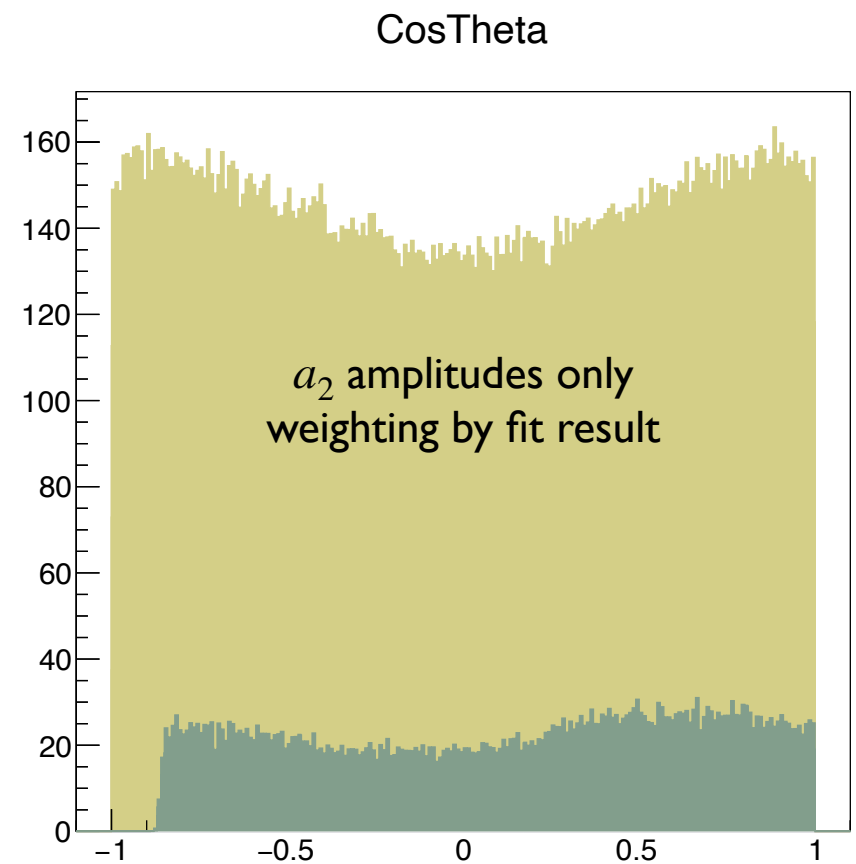
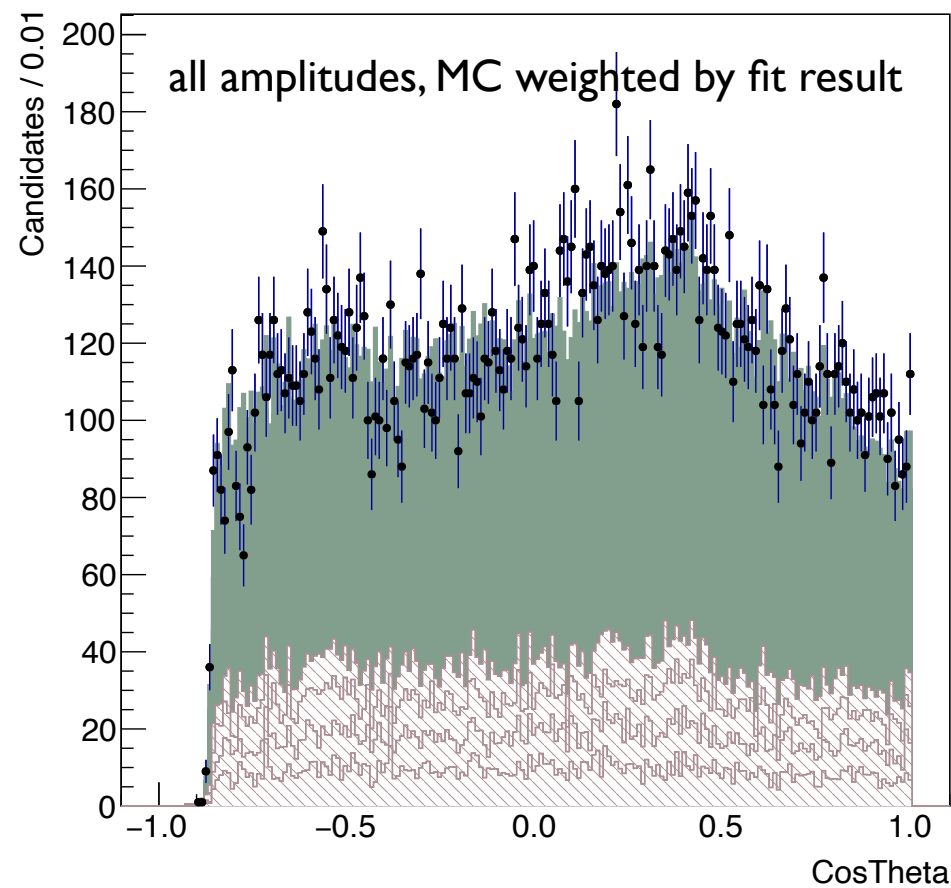
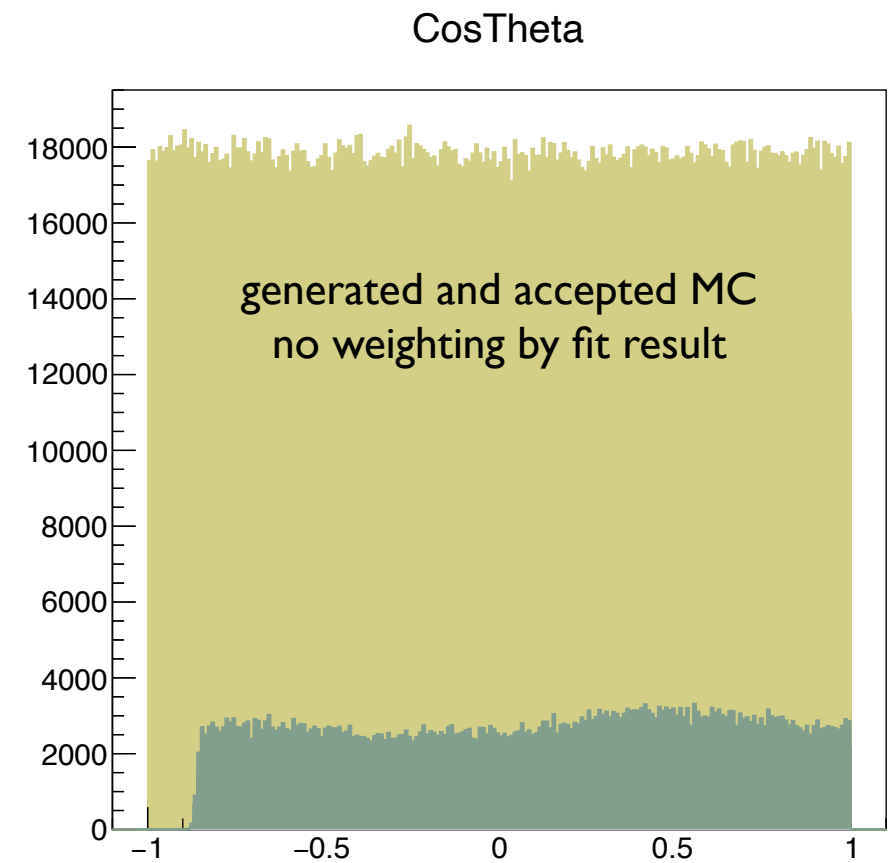
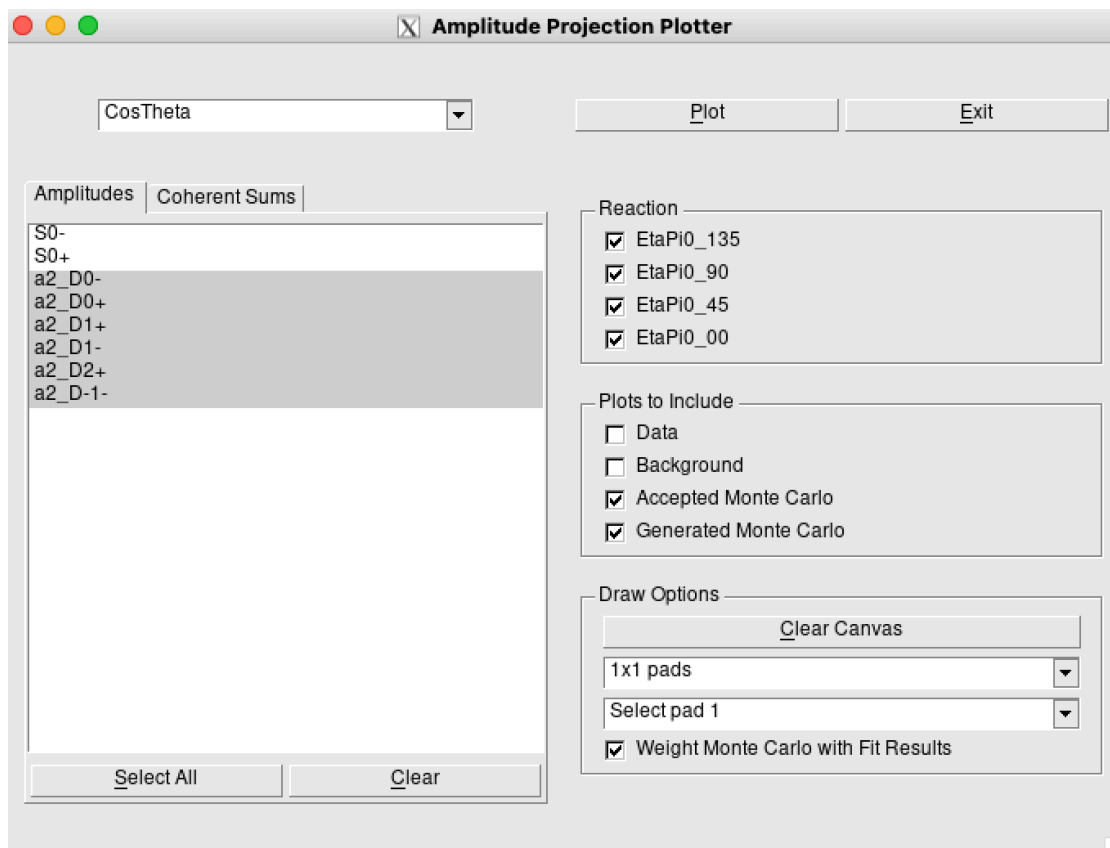
typical: leave all
coherent sums on



data
(points)

accepted MC
from accmc
weighted by fit result

background from
bkgnd files
(4 reactions)



Useful Quantities from FitResults

- The *.fit output can be used to create an object of type FitResults which is intended to provide a C++ interface to the results of a fit
- In the session2e directory see displayResults.cc for an example of how to create this object and print some useful information to the screen
- copy displayResults.cc and the Makefile to your own directory and play with it

```

===== REACTIONS =====
Number of reactions:  4
(reaction name):  (observed signal events from fit)  [contribution to -2 ln L]
EtaPi0_00:  3866.11 +/- 55.0971  [-37262.7]
EtaPi0_45:  3652.82 +/- 100.789  [-34984.4]
EtaPi0_90:  4105.35 +/- 109.639  [-38119.9]
EtaPi0_135:  3748.66 +/- 102.619  [-35453.1]
TOTAL:  15372.9 +/- 276.389  [-145820]

===== AMPLITUDES =====
(amplitude name):  (acceptance corrected yield) (fit fraction) [detection efficiency]
a2_D0-   :  2350.81 +/- 829.937  (0.0224004) [0.117551]
a2_D0+   :  3781.52 +/- 800.061  (0.0360334) [0.117551]
a2_D1-   :  5327.23 +/- 2208.54  (0.0507621) [0.150105]
a2_D1+   :  1198.49 +/- 561.491  (0.0114202) [0.150106]
a2_D-1-  :  6256.73 +/- 2144.09  (0.0596192) [0.150105]
a2_D2+   :  10255.8 +/- 1525.7   (0.0977258) [0.153705]
S0+      :  63909.5 +/- 1078.84  (0.608981) [0.144667]
S0-      :  11910.2 +/- 200.964  (0.11349)  [0.145053]

-----
ALL a2(1320)   :  29178 +/- 1415.98  (0.278031) [0.147189]

===== MINUIT DIAGNOSTICS =====
Error Matrix Status ( 3 = full/accurate ):  3
Last Command Status ( 0 = normal ):  0
Estimated Distance to Minimum:  5.49816e-06
    
```

Efficiency-corrected number
of $a_2(1320)$ events in the data sample
(assuming $\mathcal{B}(a_2 \rightarrow \eta\pi^0) = 100\%$
and $\mathcal{B}(\eta \rightarrow \gamma\gamma) = 100\%$)

MPI Acceleration

- Invoking `fit -c etapi0_SD_TMD_allPol.cfg` on an ifarm node will give you time to eat lunch before convergence:

```
MIGRAD evaluation total wall time: 1466.2 s.  
average time per function call: 199.89 ms.  
  
LIKELIHOOD AFTER MINIMIZATION: -1.4582e+05
```

- AmpTools supports MPI (Message Passing Interface) acceleration: multiple single-threaded processes (potentially across nodes of a cluster) working on a common fit while communicating through the MPI layer
 - allows aggregating RAM across cluster nodes for large data sets
 - enhances speed because each process runs in parallel
 - can be used to accelerate fits on multi-core CPU much like multi-threading a single job
- Requires registration of Amplitudes and parallelized DataReaders (via an AmpTools template) in an executable that can setup and manage the MPI layer:
 - for GlueX use: `fitMPI` (in `halld_sim`)
 - if `fitMPI` is compiled with your release of `halld_sim`, it is likely you don't need to do anything but load the MPI module in your shell

```
module use /apps/modulefiles  
module load mpi/openmpi3-x86_64
```
 - full documentation on ifarm/GlueX MPI usage for fitting is here:
https://halldweb.jlab.org/wiki/index.php/HOWTO_use_AmpTools_on_the_JLab_farm_with_MPI

use mpirun to spawn multiple fitMPI processes that work together:
 mpirun -np 33 fitMPI -c etapi0_SD_TMD_allPol.cfg

```
top - 10:00:32 up 16 days, 20:52, 111 users, load average: 39.07, 27.71, 23.38
Tasks: 2352 total, 54 running, 2268 sleeping, 27 stopped, 2 zombie
%Cpu(s): 37.8 us, 3.9 sy, 5.5 ni, 52.5 id, 0.1 wa, 0.0 hi, 0.0 si, 0.0 st
KiB Mem : 26380076+total, 641736 free, 49034316 used, 21412470+buff/cache
KiB Swap: 16777212 total, 15212576 free, 1564636 used. 21381155+avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
197943	staylor	24	4	2686300	1.7g	108360	S	702.3	0.7	841:37.07	hd_root
210045	shrestha	20	0	3432624	1.3g	15224	S	104.6	0.5	102:03.46	java
129604	shepherd	20	0	906556	200164	101132	R	100.0	0.1	0:34.55	fitMPI
129610	shepherd	20	0	906552	200152	101128	R	100.0	0.1	0:34.71	fitMPI
129615	shepherd	20	0	905512	199644	101128	R	100.0	0.1	0:34.65	fitMPI
129619	shepherd	20	0	905512	199636	101128	R	100.0	0.1	0:34.62	fitMPI
129620	shepherd	20	0	905512	199640	101128	R	100.0	0.1	0:34.66	fitMPI
129621	shepherd	20	0	905512	199668	101152	R	100.0	0.1	0:34.62	fitMPI
129625	shepherd	20	0	905508	199636	101128	R	100.0	0.1	0:34.63	fitMPI
129633	shepherd	20	0	905508	199660	101152	R	100.0	0.1	0:34.56	fitMPI
231757	tianye	20	0	2702388	2.4g	19196	R	100.0	1.0	20:30.32	analysis_R_PVDI
34907	ebarriga	20	0	577388	276372	113384	R	99.7	0.1	13:57.64	root.exe
92606	ebarriga	20	0	566036	265088	113384	R	99.7	0.1	3:52.32	root.exe
109116	ebarriga	20	0	571328	270348	113384	R	99.7	0.1	2:33.11	root.exe
129602	shepherd	20	0	2750848	760116	110540	R	99.7	0.3	0:34.65	fitMPI
129603	shepherd	20	0	905516	200140	101628	R	99.7	0.1	0:34.67	fitMPI
129605	shepherd	20	0	906560	200144	101128	R	99.7	0.1	0:34.59	fitMPI
129606	shepherd	20	0	906552	200160	101128	R	99.7	0.1	0:34.69	fitMPI
129607	shepherd	20	0	906560	200156	101124	R	99.7	0.1	0:34.59	fitMPI
129609	shepherd	20	0	906556	200176	101152	R	99.7	0.1	0:34.67	fitMPI
129611	shepherd	20	0	906556	200156	101128	R	99.7	0.1	0:34.64	fitMPI
129612	shepherd	20	0	906556	200172	101152	R	99.7	0.1	0:34.58	fitMPI
129613	shepherd	20	0	906556	200156	101128	R	99.7	0.1	0:34.64	fitMPI
129614	shepherd	20	0	906552	200160	101132	R	99.7	0.1	0:34.64	fitMPI
129617	shepherd	20	0	905512	199640	101128	R	99.7	0.1	0:34.62	fitMPI
129618	shepherd	20	0	906548	200152	101128	R	99.7	0.1	0:34.61	fitMPI
129622	shepherd	20	0	905508	199640	101128	R	99.7	0.1	0:34.63	fitMPI
129623	shepherd	20	0	905508	199660	101156	R	99.7	0.1	0:34.63	fitMPI
129624	shepherd	20	0	905508	199660	101152	R	99.7	0.1	0:34.59	fitMPI
129626	shepherd	20	0	905508	199632	101128	R	99.7	0.1	0:34.64	fitMPI
129627	shepherd	20	0	905508	199644	101128	R	99.7	0.1	0:34.58	fitMPI
129628	shepherd	20	0	905508	199660	101156	R	99.7	0.1	0:34.61	fitMPI
129629	shepherd	20	0	905508	199632	101128	R	99.7	0.1	0:34.58	fitMPI
129631	shepherd	20	0	905508	199640	101132	R	99.7	0.1	0:34.58	fitMPI
129632	shepherd	20	0	905508	199640	101132	R	99.7	0.1	0:34.56	fitMPI
129634	shepherd	20	0	906552	200180	101148	R	99.7	0.1	0:34.63	fitMPI
134969	zyy	20	0	341184	35184	13908	R	99.7	0.0	0:09.12	python
205469	jsalvg	20	0	867980	475752	118240	R	99.7	0.2	109:15.29	clas12root
206053	jsalvg	20	0	910108	518000	118224	R	99.7	0.2	108:39.93	clas12root
90143	gluex	20	0	1606796	1.0g	58408	S	99.3	0.4	46:51.29	hdgeant4
129608	shepherd	20	0	906556	200148	101128	R	99.3	0.1	0:34.65	fitMPI
129616	shepherd	20	0	906556	200152	101128	R	99.3	0.1	0:34.54	fitMPI
129630	shepherd	20	0	905508	199644	101128	R	99.3	0.1	0:34.58	fitMPI

(single process ifarm CPU)

MIGRAD evaluation total wall time: 1466.2 s.
 average time per function call: 199.89 ms.
 LIKELIHOOD AFTER MINIMIZATION: -1.4582e+05



MIGRAD evaluation total wall time: 65.21 s.
 average time per function call: 8.8204 ms.
 LIKELIHOOD AFTER MINIMIZATION: -1.4582e+05

Do **NOT** do this on the ifarm interactive node as a matter of regular practice!!!

(...but isn't it beautiful to see 33 jobs each at >99% CPU working together on one fit?)

GPU Acceleration

- AmpTools supports acceleration using GPUs (Graphical Processing Units) for numerical computation
 - GPU nodes are available on the ifarm -- these are suitable for fast interactive fitting
- Requires the user to write the amplitude calculation as a GPU "kernel" in a language similar to C called CUDA
 - technically more involved: potential for code-divergence as CPU and GPU code must both be maintained
 - Example in Dalitz Tutorial: [BreitWigner_kernel.cu](#)
- Many GlueX amplitudes are already implemented in CUDA for GPU fitting
- GPU acceleration is also useful for interactive plotting
- Speed gains are highly dependent on particular fit strategy, GPU hardware, etc. -- ask for advice on optimization
- Multiple GPUs can be used (even across nodes) by using the MPI layer where each process uses a GPU

(single process ifarm CPU)

```
MIGRAD evaluation total wall time: 1466.2 s.  
average time per function call: 199.89 ms.  
LIKELIHOOD AFTER MINIMIZATION: -1.4582e+05
```



```
MIGRAD evaluation total wall time: 26.426 s.  
average time per function call: 3.9056 ms.  
LIKELIHOOD AFTER MINIMIZATION: -1.4582e+05
```

GlueX Documentation: https://halldweb.jlab.org/wiki/index.php/HOWTO_use_AmpTools_on_the_JLab_farm_GPUs

Summary

- There is a suite of libraries and executables built off of the AmpTools framework for doing amplitude analysis on GlueX data
 - AmpTools is largely independent of physics process or experiment
 - common GlueX-specific problems to be solved in the context of AmpTools
- A variety of resources are available for issues, questions, help:
 - [AmpTools Fits Best Practices](#)
 - [AmpTools Issues on GitHub](#)
 - GlueX Software Google group
 - meetings of the Amplitude Analysis Working Group
 - use old technology and phone a friend: *mashephe@indiana.edu*