PyPWA

A Partial-Wave/Amplitude Analysis Software Framework

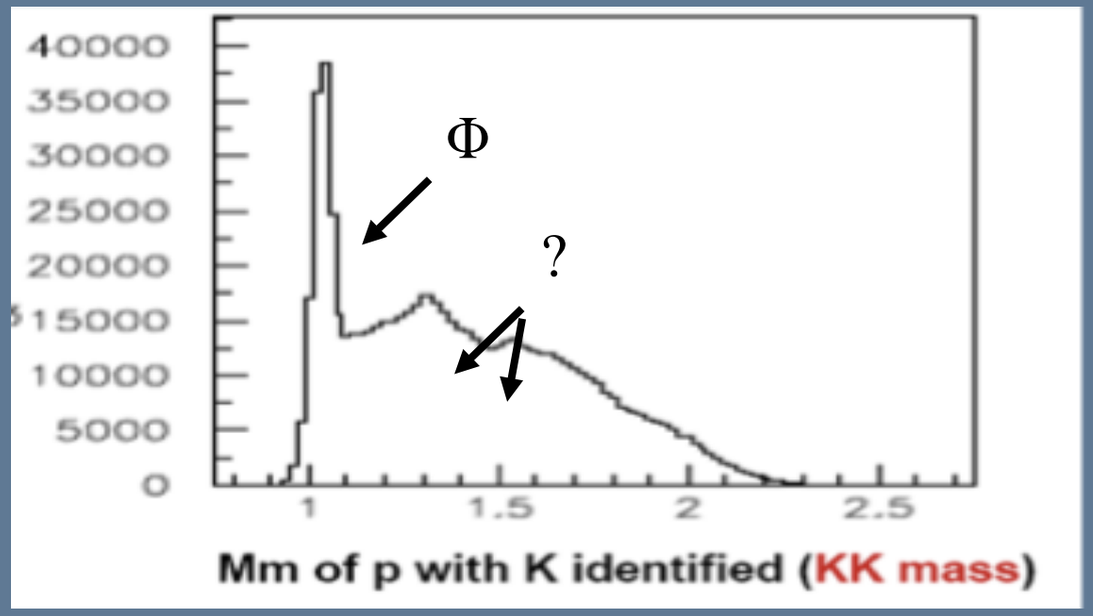
Carlos W. Salgado[1,2]

other team members

S. Bramlett[1], B. DeMello[1], M. Jones[1]
W. Phelps[3] and J. Pond[1]

Norfolk State University[1]
The Thomas Jefferson National Accelerator Facility[2]
Florida International University[3]

Jefferson Lab



Preliminary data from CLAS6-g12

Future of Spectroscopy Analysis will be on the study of resonances that are hidden?

★ overlapping
★ wide
★ many-particles final states
★ having small cross-sections
★ with large non-resonant backgrounds
★ ...

Furthermore

- Wave ambiguities
- Leakages
- Baryon contamination (JLab)

- LARGE DATA STATISTICS
- LARGE AMOUNT OF SIMULATION (MC)

In this environment we will need to find
• the poles on the S-Matrix (complex amplitudes) and
• detailed study of interference between states (wave motion)
to determine new short-lived states.

PyPWA

Our philosophy

Provide the user with a software framework to analyze resonances from multi-particle final states in photo-production.

- Types of analysis
  - OPTIMIZATION (Parameter Estimation - Fitting)
  - SIMULATION (Monte Carlo)
- Basic TOOLS/MODULAR to be use in the analysis
- Flexible (EASY to CHANGE)
- Very WELL DOCUMENTED
- Interact with multiple programing languages
- Interact with other packages
- Easy integration to ANY amplitude model written in ANY language
- Integrated use of the JLab Scientific Computing Resources
- Parallelization & Vectorization
- Own graphical package and interface with PyROOT (CERN)

PyPWA

Implementation: PYTHON basic numpy and scipy libraries

- GUI driven use of JLab resources (i.e,Farms).
- Hybrid programming, where languages are used as they are adequate for the specific task and then interfaced. For example, Python being a high-level programming language makes a better scripting language to “glue” several programming modules, and Fortran and C are more basic languages with much faster number-crunching looping.
- Vectorization works by exploiting the combined add-multiply unit of the Intel Xeon Phi and/or GPUs
- Include full documentation at code level (and also tutorials examples...)
- Many options for optimization (i.e. minimization algorithms) and plotting tools
- Many options for data formats (in and out) - auto-defined txt files /or 4-vectors...

Jefferson Lab

software structure

The PyPWA framework and toolkit is divided in

GENERAL-SHELL

- Fitting and Simulation
- Can use any model
- Interface is through a user defined Python script taken from a template.
- Integrated batch farm interface
- ISOBAR plotting can be used the ISOBAR file structure is mimicked by the user.
- Simulation produces “masks”.

ISOBAR

- Fitting and Simulation
- Exclusively uses the isobar amplitude model and photo-production (linear pol)
- Easy install and mass binning
- Takes advantage of the GAMP[1] event format and the GAMP amplitude generator utilizing “keyfiles” physics descriptions
- Interface is with GUIs
- Interacts directly and exclusively with the Jlab batch farm
- Integrated plotting through Python

[1] Cummings and Weygand (2000) -used by E852-COMPASS-CLAS

Jefferson Lab

software structure

General-shell

- Parametric fitting of data using any physics model
- Simulating data from phase space Monte Carlo using Rejection Sampling MC
- Python is a high level language which eases the writing of intensities.
- Access to all Python and PYROOT libraries (and own)
- Integration with lower level languages is easy(F2PY, CYTHON)
- Optional use of “Q factor” for signal quality
- GS has a convenient interface with Minuit or other o[tomization.

Isobar (PWA)

- Meson Spectroscopy and Partial Wave Analysis using the Isobar model
- Simulating data from phase space Monte Carlo using Rejection Sampling
- Analysis of data using mass plotting tools
- Integrated Isobar model
- Ease and Speed of use for Jlab users
- Integration directly to the batch farm
- Optional use of “Q factor” for signal quality

General Shell

The General shell side of PyPWA is focused on **openness and generality**.

The General Shell uses code inputs from the user, but can fit any model to the data by a user's choice of:

- Un-binned standard Likelihood method.
- Un-binned Extended Likelihood method.
- Binned Likelihood method.
- Least-squares

for example

$$-ln\mathscr{L} = -\sum_{i=1}^{N} Q_i ln\left[I(\vec{x}_i, \vec{a})\right] + \frac{1}{N_g}\sum_{i=1}^{N_a} I(\vec{x}_i, \vec{a})$$

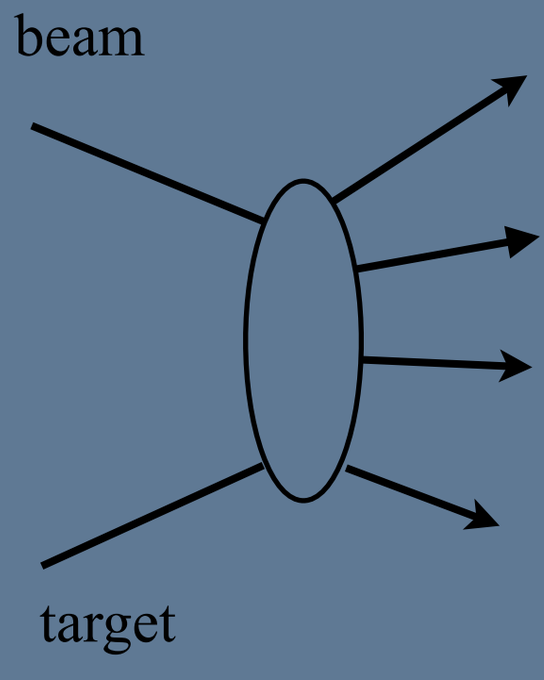
Minimization Default: **Minuit**

many others are easily available from scipy.optimize

Jefferson Lab

PWA - Isobar (Partial Wave Analysis) Formalism

Salgado&Weygand: **Phys.Rep, vol 537/1, pages 1-58 (2014)**: arXi v:131arXiv:1310.7498.



$$\mathscr{L} = Prob(N) \prod_{i=1}^{N} p(\vec{x}_i, \vec{a})$$

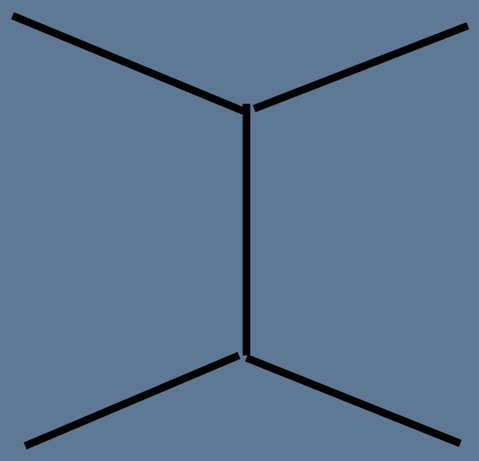
Extended likelihood

$$Prob(N) = \frac{\mathscr{N}^N}{N!} e^{-\mathscr{N}}$$

then:

$$-ln\mathscr{L} \propto \sum_{i=1}^{N} ln\left[\mathbb{P}(\vec{x}_i, \vec{a})\right] - \int_{\Omega} \mathbb{P}(\vec{x}, \vec{a}) d^n\vec{x}$$

Need a Model to fit:



$$\frac{d\sigma}{dtdM^2d\tau} \propto \sum_{ext.\ spins} |\mathscr{M}|^2 \equiv I(\tau)$$

$$I(\tau) = \sum_{ext.\ spins} \langle f|\widehat{T}\widehat{\rho}_i\widehat{T}^{\dagger}|f\rangle$$

Partial Waves

$$P = (-1)^{L+1}$$
$$C = (-1)^{L+S}$$
$$G = (-1)^{L+S+I}$$

$$I(\tau) = \sum_{k\epsilon} \sum_{b,b'} {}^{\epsilon} A_b(\tau)^{\epsilon} \; {}^{\epsilon} V_b^k \; \widehat{\rho_\gamma}^{\epsilon} V_{b'}^{k*\epsilon} A_{b'}^*(\tau)$$

The Spin Density Matrix of the incoming Photon is :

$$\rho_{\varepsilon\varepsilon'}(P,\alpha) = {}^1\!/_2 \begin{pmatrix} 1 + P\cos 2\alpha & iP\sin 2\alpha \\ -iP\sin 2\alpha & 1 - P\cos 2\alpha \end{pmatrix}$$

Unbinned Maximum Likelihood fit

$$-ln\mathscr{L} \propto \sum_{i=1}^{N} ln\left[\sum_{k\epsilon} \widehat{\rho_\gamma} \sum_{b,b'} {}^{\epsilon} V_b^k {}^{\epsilon} V_{b'}^{k*} \; {}^{\epsilon} A_b(\tau_i) \; {}^{\epsilon} A_{b'}^*(\tau_i)\right] - \eta_x \sum_{k\epsilon} \sum_{b,b'} {}^{\epsilon} V_b^k \; {}^{\epsilon} V_{b'}^{k*} \; {}^{\epsilon} \Psi_{b,b'}^x$$

MASS INDEPENDENT FIT (in bins of M and t)

Jefferson Lab

Problems of destroying "factorization"

Data events

MC events

$$\mathscr{F}(\vec{p}) = -ln\mathscr{L} = -\sum_{i=1}^{N} ln\left[\sum_{ext.\ spins}(\mathscr{M}\mathscr{M}^*)\right] + \eta_x \frac{1}{N_a}\sum_{i}^{N_a}\sum_{ext.\ spins}(\mathscr{M}\mathscr{M}^*). \quad (9)$$

$$I(\tau) = \sum_{k\epsilon}\sum_{b,b'} {}^{\epsilon}A_b(\tau)^{\epsilon}\ {}^{\epsilon}V_b^k\ \widehat{\rho_\gamma}^{\epsilon}V_{b'}^{k*\epsilon}A_{b'}^*(\tau)$$

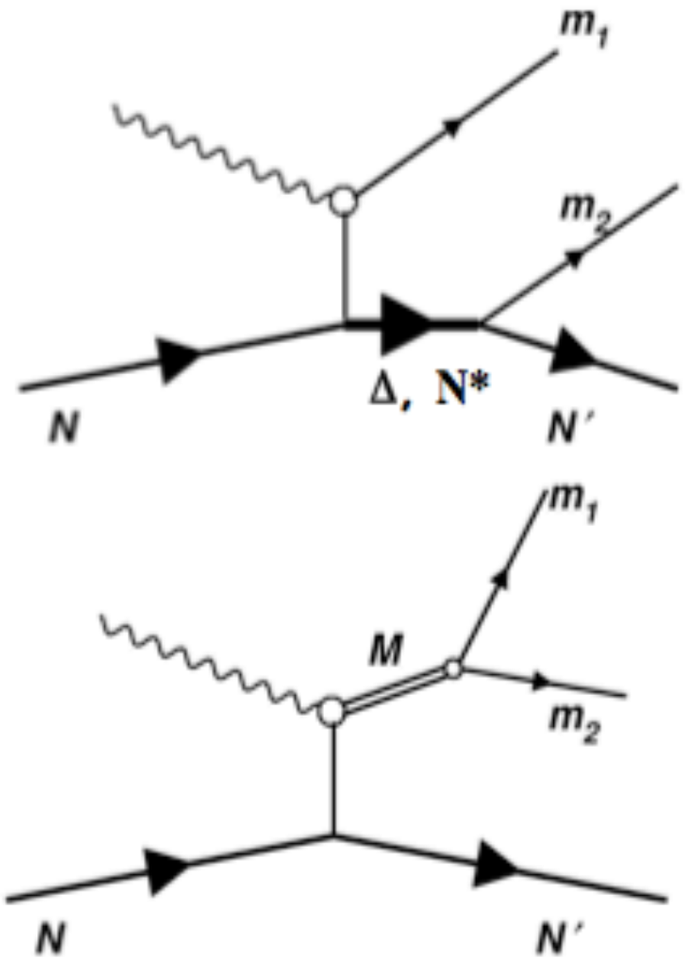
$$-ln\mathscr{L} \propto \sum_{i=1}^{N} ln\left[\sum_{k\epsilon}\widehat{\rho_\gamma}\sum_{b,b'} {}^{\epsilon}V_b^{k\epsilon}V_{b'}^{k*}\ {}^{\epsilon}A_b(\tau_i)\ {}^{\epsilon}A_{b'}^*(\tau_i)\right] - \eta_x\sum_{k\epsilon}\sum_{b,b'} {}^{\epsilon}V_b^k\ {}^{\epsilon}V_{b'}^{k*}\ {}^{\epsilon}\Psi_{b,b'}^x$$

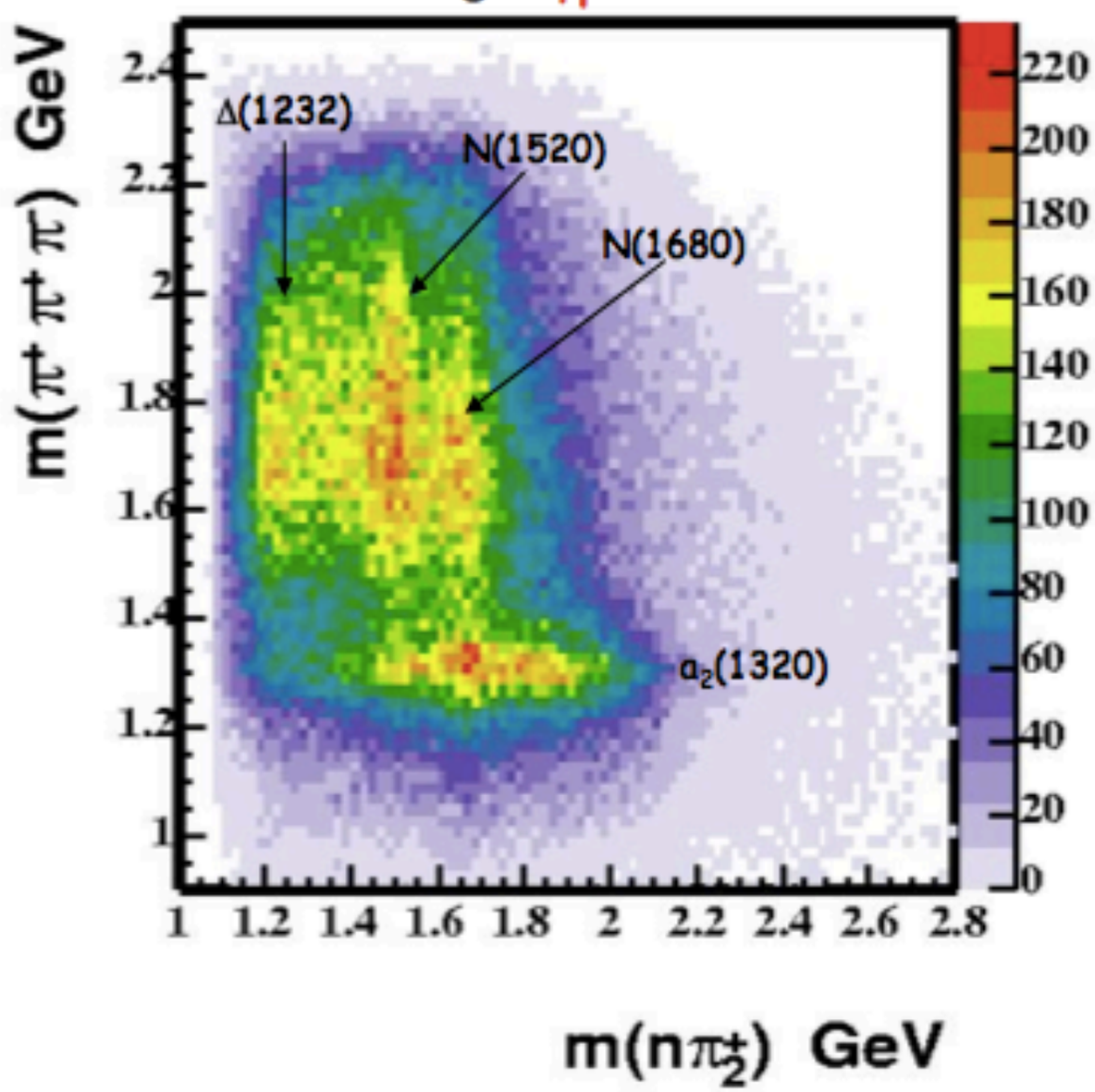
Normalization Integrals

Meson SpectroscopyStrategies at JLab

- **Use 8.4–9 GeV linearly polarized photons**
  - Identify (naturality) production mechanisms
  - Open phase space to separate meson/baryon production products
  - Sensitivity to masses up to ~ 2.8 GeV/c^2
- **Use hermetic detector with large acceptance**
  - Decay modes expected to have multiple particles
  - Hermetic coverage for charged and neutral particles
  - Medium resolution: momentum (~ 1-4%), energy (2-20%)
  - High data acquisition rate to enable amplitude analysis
- **Perform amplitude analysis**
  - Identify wide and rare (small cross sections) resonances
  - Use all available S-Matrix physics constraints on fittings
  - Identify the J^{PC} of resonances -phase motions -interference patterns
  - Check consistency of results in different decay modes

Physics background from associated baryon resonance production

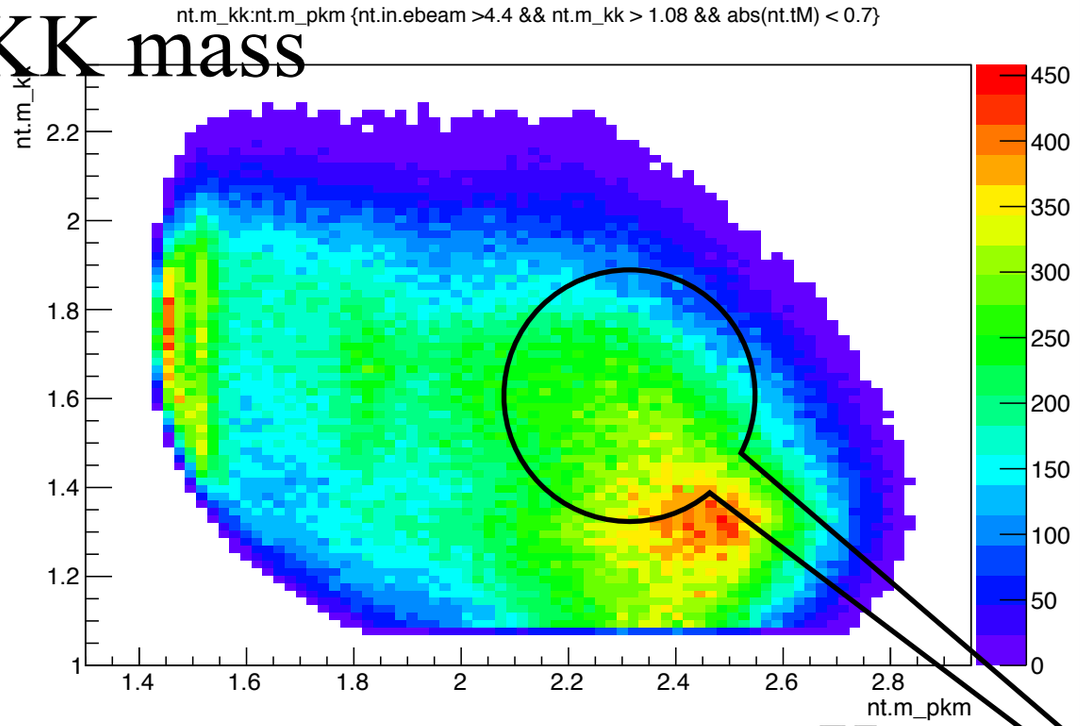




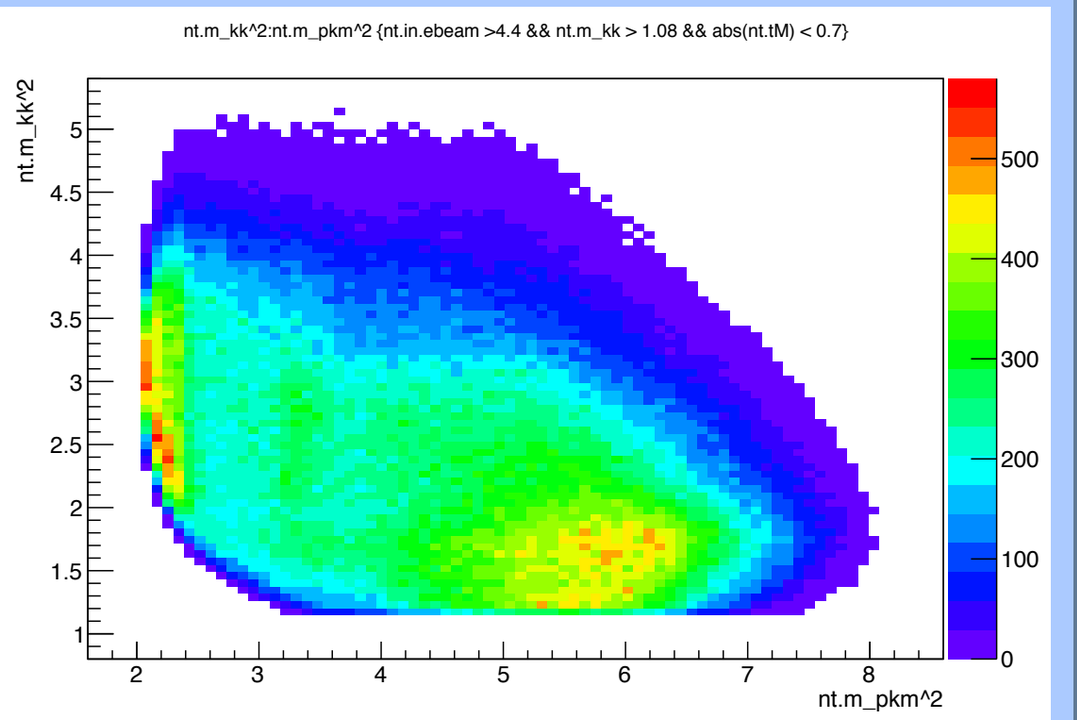
Strong kinematical cuts and a complex PWA are required

Going to higher energies increase phase space: 5.75 to 9 GeV?

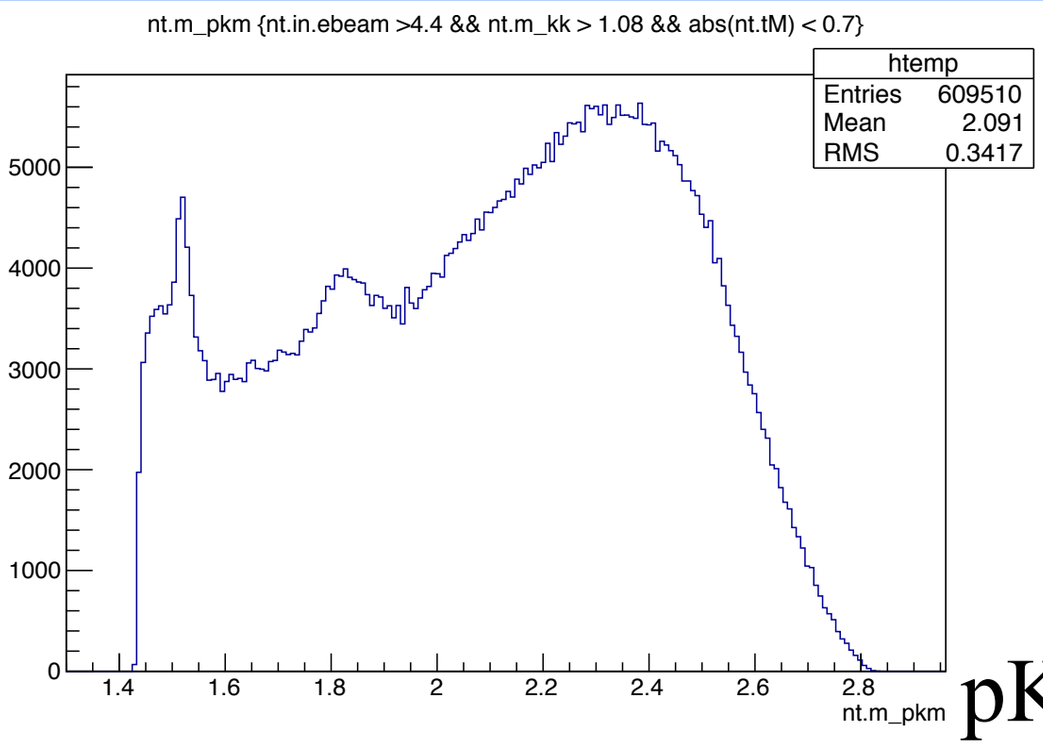
KK mass



pK mass



pK mass^2



pK mass

Interesting region for strangeonia

Examples

JPAC-CLAS-GlueX

Current uses of this software

1) Parametric fit to models

- omega decay (Dalitz) I.Danilkin, et al.
- pKK analysis Meng Shi et al.

2) PWA

CLAS g12 pKK,3pi,...

3) Include “Deck-type” effects into PWA (extended)

CLAS pKK,3pi,.. C. Fernandez et al.

some specifics for general-shell

Beginning the Process

Binning and file structures and anything else to do with the user's specific fit, or simulation is up to them. The only thing they need to start is two files for fitting, or simulation, and an extra variable parser utility for fitting.

For both fitting and simulation there is one file that the user interacts with and one (they can leave alone) used to run the fit (simulation).

Simulation and fitting take text files of variables in a specific format:

X1=0.25,X2=1.67,X3=90.5 ...

simulation produce two “masks” to be applied to each event
- production mask
- acceptance mask

User's Main Point of Contact

The main points of contact for the user within the General Shell are the **Fn.py** and **FnSim.py** files. They are in the download as **FnTemplate.py** and **FnSimTemplate.py.**

They include documentation and examples to help the user write their intensity function, but a basic knowledge of Python is required.
They are both a series of functions that each do a specific job for the calculations involved with fitting and simulation. This includes the intensity function, and the initial values and limits for fitting parameters.
These files will have to be changed for every different fit, or simulation.
Results and Plotting.

general-shell

Jefferson Lab

Example

```
def intFn(kVars,params):

    tDist = params['A1']*numpy.exp(params['A2']*(kVars['tM']))

    wConst = (3.0/(4.0*math.pi))
    W = wConst*(0.5*(1-params['A3'])+0.5*(3*params['A3']-1)*math.cos(kVars['theta'])**2
        -math.sqrt(2.0)*params['A4']*math.sin(2*kVars['theta'])*math.cos(kVars['phi'])
        -params['A5']*(math.sin(kVars['theta']))**2*math.cos(2*kVars['phi']))
    F=AMP.amp(kVars['s'],kVars['t'],kVars['u'],params['A6'])
    Fsquare=F*numpy.conjugate(F)

    return tDist*W*kVars['P']*Fsquare
```

This is an example of the sort of function you can fit with PyPWA General. This is the **intFn()** function inside **Fn.py** and it's arguments are the two keyed dictionaries, **kVars** and **params**. **Kvars** are the variables parsed from the text file, while **params** are the parameters fitted by Minuit.

this is the function

3 Fitted Function

$$I(sD, tD, uD, \theta, \phi, \mathbf{A1}, \mathbf{A2}, \mathbf{A3}, \mathbf{A4}, \mathbf{A5}) = \mathbf{A1} * W(\theta, \phi, \mathbf{A2}, \mathbf{A3}, \mathbf{A4}) * P(sD, tD, uD) * \left[F(sD, tD, uD, \mathbf{A5})\right]^2 \quad (15)$$

whre W is the Schilling et al. spin dendity matrix (no-polarization):

$$W(\theta, \phi, \mathbf{A2}, \mathbf{A3}, \mathbf{A4}) = \quad (16)$$

$$\frac{3}{4\pi}\left[0.5 * (1 - \mathbf{A2}) + 0.5 * (3 * \mathbf{A2} - 1) cos^2(\theta) - \sqrt{2} * \mathbf{A3} * sin2(\theta) cos(\phi) - \mathbf{A4} * sin^2(\theta) cos2(\phi)\right] \quad (17)$$

and θ, ϕ are Adair's angles. P is a kinematic factor given by:

$$P(sD, tD, uD) = sD * tD * uD - m_\pi^2 \left[M^2 - m_\pi^2\right]^2 \quad (18)$$

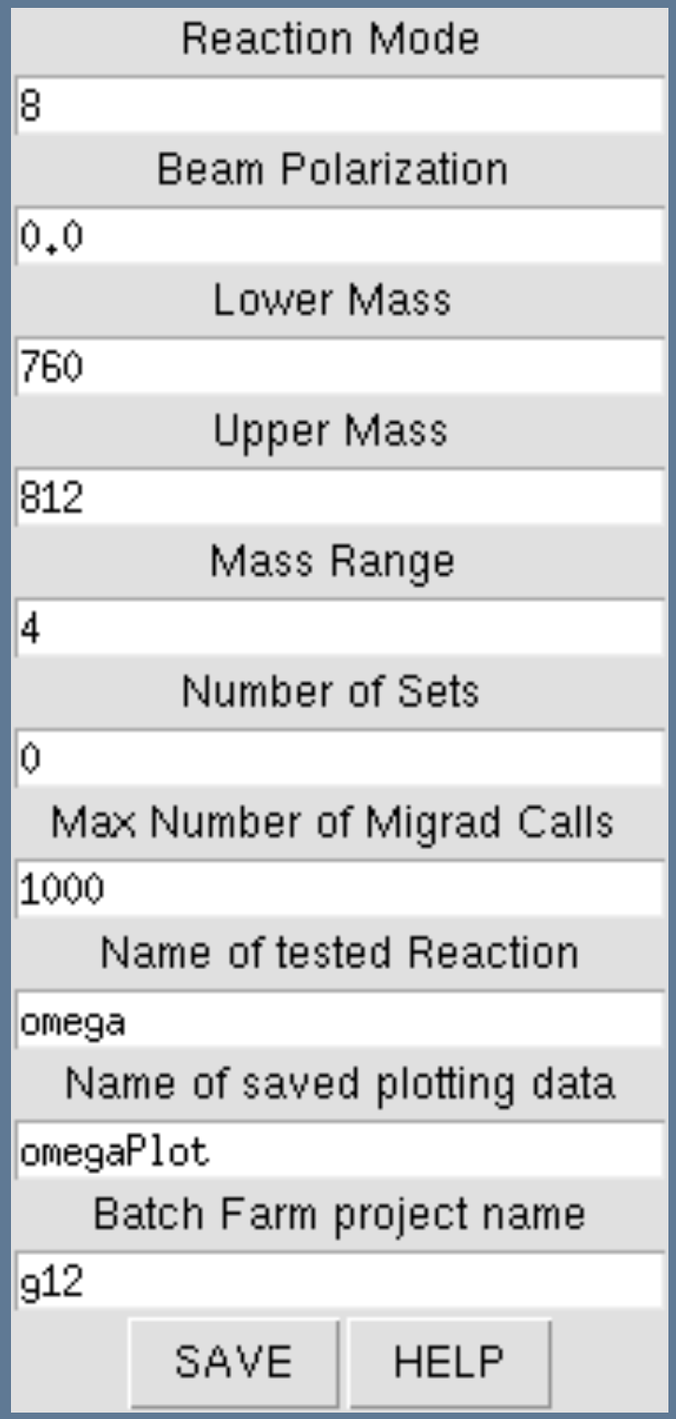
wherre sD, tD, uD are the Mandelstam variables of the decay such that:
$sD = (p_X - p_{\pi^+}), sD = (p_X - p_{\pi^-})$ and $sD = (p_X - p_{\pi^0})$.
and $p_X = p_{\pi^+} + p_{\pi^-} + p_{\pi^0}$, M is the mass of the three pion system and m_π the mass of the pion (plus).
F (sD,tD,uD,**A5**) is Igor Danilkin et al. amplitude given for a call to his fortran code.

ISOBAR - PWA

The Isobar framework is focused on ease of use and speed. So from the install process until plotting almost everything is automated.

Install is handled by a single program which opens the control GUI, creates the needed directory structure, moves files to their correct location, and does the mass binning, which can take awhile if the user has many events.

The control GUI at right is the first point of contact the user has with PyPWA and the information filled into it will be used throughout the fitting and simulating process.

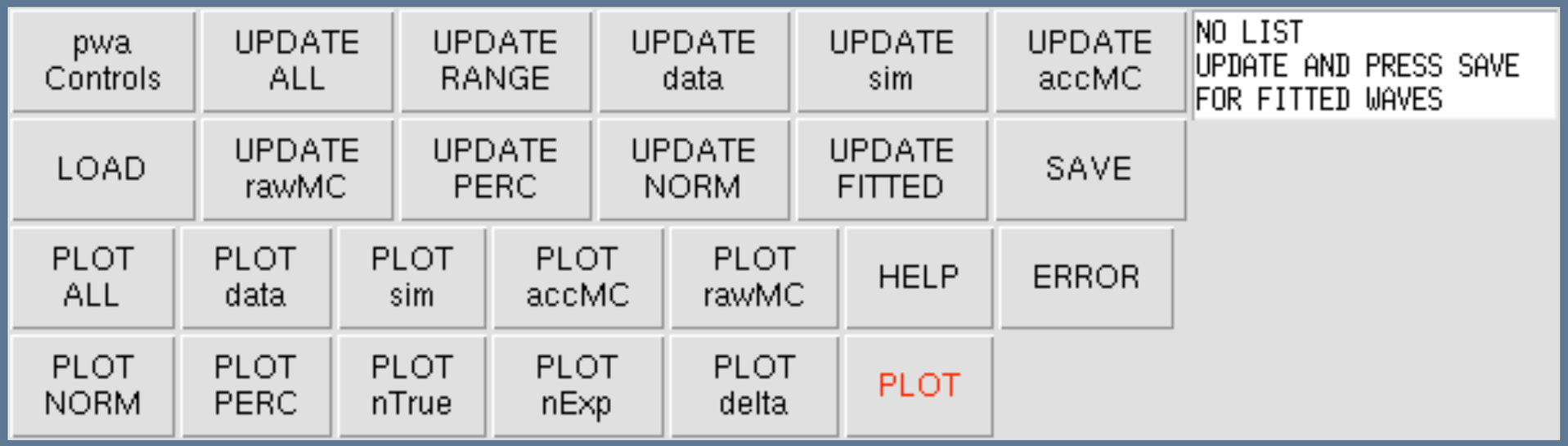


ISOBAR cont.

The Isobar framework's main point of contact for the user is the PWA_GUI at right. The left column is what appears when the program is run and the right is what appears after the FITTING button is pressed.

Each button on the right represents a different step in the fitting process and runs a different program. Each of these buttons will run the program which creates and submits many jsub files directly to Auger.

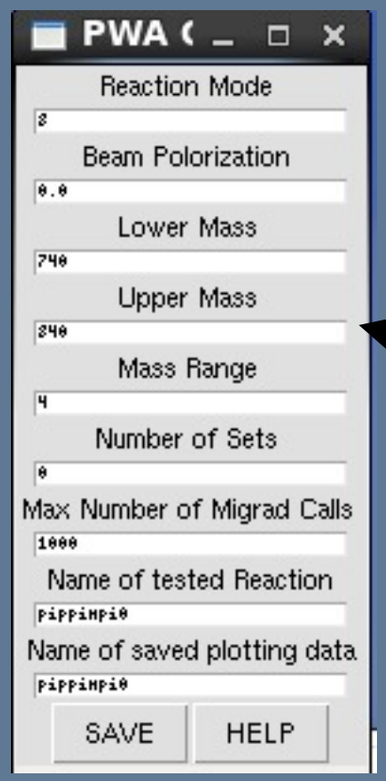
This GUI also has access to the control, the plotter, and the Waves utility.



Plotting

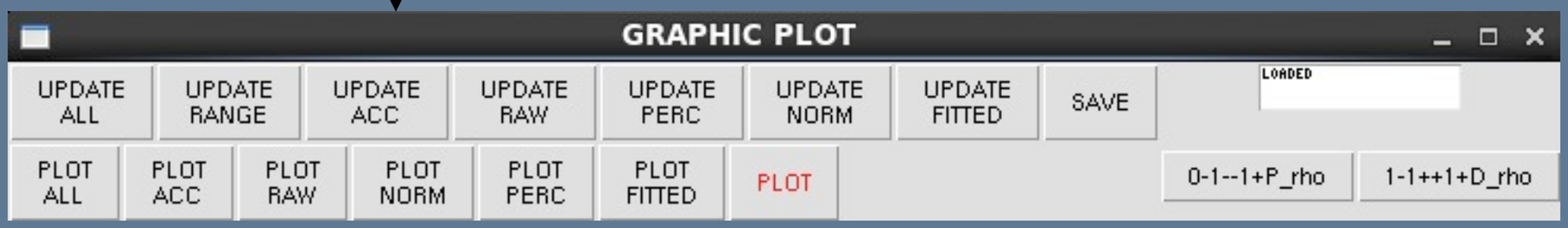
Plotting in PyPWA Isobar is handled by the above GUI which uses the MatPlotLib Python library for all plotting.
This program also consolidates all data for plotting into single file named in the control. This file can be loaded in the future and multiple files can be saved and loaded at different times.

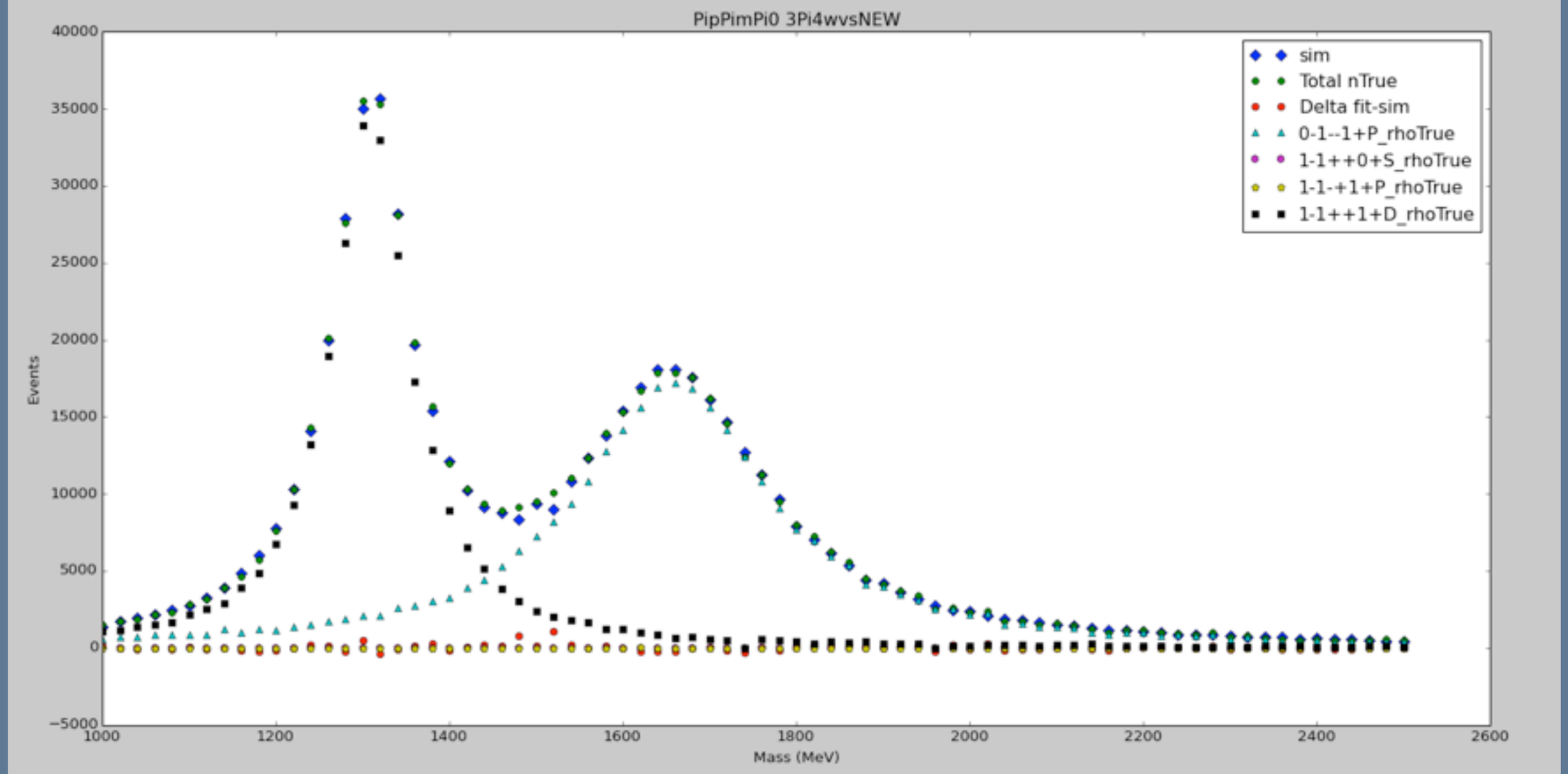
GUI driven


PWA
Reaction Mode
2
Beam Polorization
0.0
Lower Mass
740
Upper Mass
840
Mass Range
4
Number of Sets
0
Max Number of Migrad Calls
1000
Name of tested Reaction
PiPPiMPi0
Name of saved plotting data
PiPPiMPi0
SAVE
HELP


PWA CONTROLS
GRAPHIC PLOT
FITTING
SIMULATION
WAVES
exit


GUI
PWA CONTROLS
Run Gamp
data
GRAPHIC PLOT
Gen Alpha
accMC
FITTING
normint
rawMC
SIMULATION
Fitter
back
WAVES
nTrue
exit
back


GRAPHIC PLOT
UPDATE ALL
UPDATE RANGE
UPDATE ACC
UPDATE RAW
UPDATE PERC
UPDATE NORM
UPDATE FITTED
SAVE
LOADED
PLOT ALL
PLOT ACC
PLOT RAW
PLOT NORM
PLOT PERC
PLOT FITTED
PLOT
0-1--1+P_rho
1-1++1+D_rho

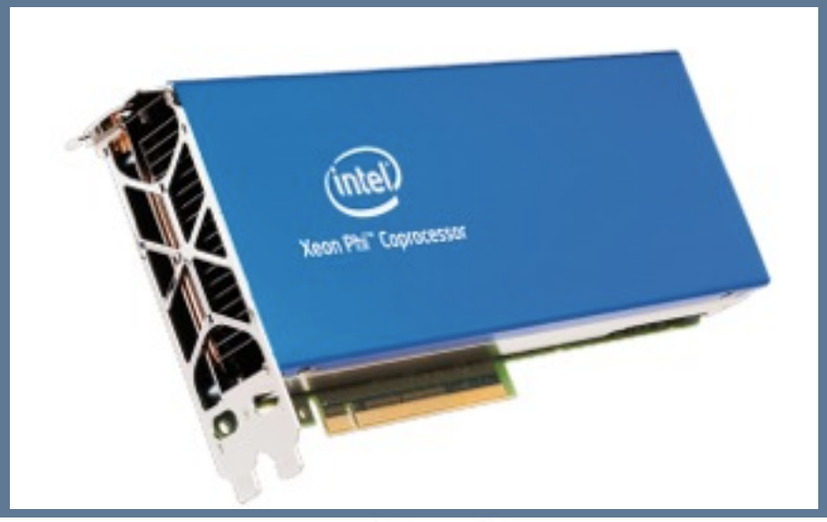

PipPimPi0 3Pi4wvsNEW
sim
Total nTrue
Delta fit-sim
0-1--1+P_rhoTrue
1-1++0+S_rhoTrue
1-1-+1+P_rhoTrue
1-1++1+D_rhoTrue
Events
Mass (MeV)
40000
35000
30000
25000
20000
15000
10000
5000
0
−5000
1000
1200
1400
1600
1800
2000
2200
2400
2600

current work...

- Adding more utilities to make General Shell even easier to use.
- Farm and plotting integration for General-Shell
- **Increased parallelization with the use of threading (farm - Xeon-Phi)**
- **Hardware acceleration with Xeon-Phi (Intel) and GPU's (Nvidia)**
- **Integrating more optimization and Monte Carlo methods**

Jefferson Lab

Intel-Xeon-Phi cards using for example **OpenMP** . Xeon Phi's contain about 61 of x86 cores that are functionally identical to those of standard laptops and desktops. There are just many more of them running at a lower clock speed to fit into a reasonable thermal design envelope (currently a PCI Express card). The maximum output is at 1TFlop and they have comparable performance with GPGPUs. Writing code for the Xeon Phi is, initially, less complicated than writing code for GPUs since it will behave as any normal CPU



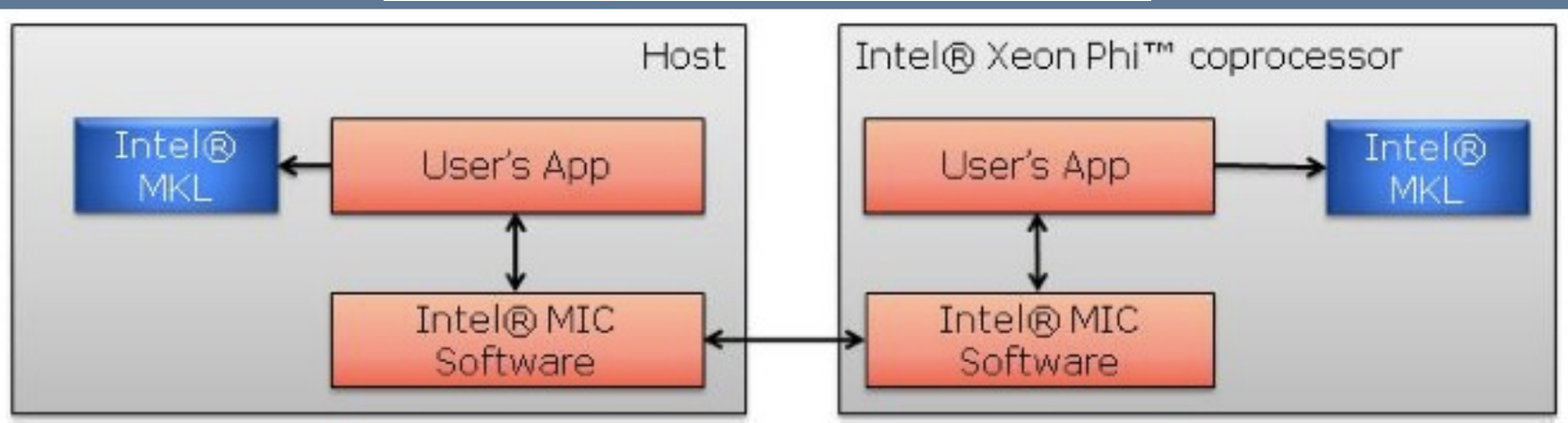


Figure 3.1: Using Intel® MKL Native Acceleration with Offload

Jefferson Lab
EXPLORING THE NATURE OF MATTER

LINKS

- PYPWA Home
- General Description
- Installation
- Tutorials
- Documentation - Sphinx
- Software Download
- Wiki
- Presentations
- References
- Contact Us

A Partial-Wave/Amplitude Analysis Software Framework

The PyPWA Project
Thomas Jefferson National Accelerator Facility
Newport News, VA

Home

The PyPWA Project aims to develop a software framework that can be used to perform parametric model fitting to data. In particular, Partial Wave and Amplitude Analysis (PWA) of multiparticle final states. PyPWA is designed for photoproduction experiments using linearly polarized photon beams. The software makes use of the resources at the JLab Scientific Computer Center (Linux farm). PyPWA extract model parameters from data by performing extended likehood fits. Two versions of the software are develop: one where general amplitudes (or any parametric model) can be used in the fit and simulation of data, and a second where the framework starts with a specific realization of the Isobar model, including extensions to Deck-type and baryon vertices corrections. Tutorials (Step-by-step instructions) leading to a full fit of data and the use of simulation software are included. Most of the code is in Python, but hybrid code (in Cyhon or Fortran) has been used when appropiate. Scripting to make use of vectorization and parallel coprocessors (Xeon-Phi and/or GPUs) are expected in the near future. The goal of this software framework is to create a user friedly enviroment for the spectroscopic analysis of linear polarized photoproduction experiments. The PyPWA Project software expects to be in a continue flow (of improvements!), therefore, please check on the more recent software download version.

Release Version 1.1 (June 22, 2015)

Version 1.1 includes several improvements, including the ability to reload the text files parsed in the General Shell, as well as a more general gampTranslator which allows for non-uniform white space in gamp files.

Bug fixes: Directory variable mistake in generalFitting is fixed.

12000 Jefferson Avenue, Newport News, VA 23606
Phone: (757) 269-7100 Fax: (757) 269-7363

contact Carlos Salgado

https://pypwa.jlab.org

Jlab web-page - Tutorials and links
wiki - github JeffersonLab/PyPWA
Sphinx generated : docs

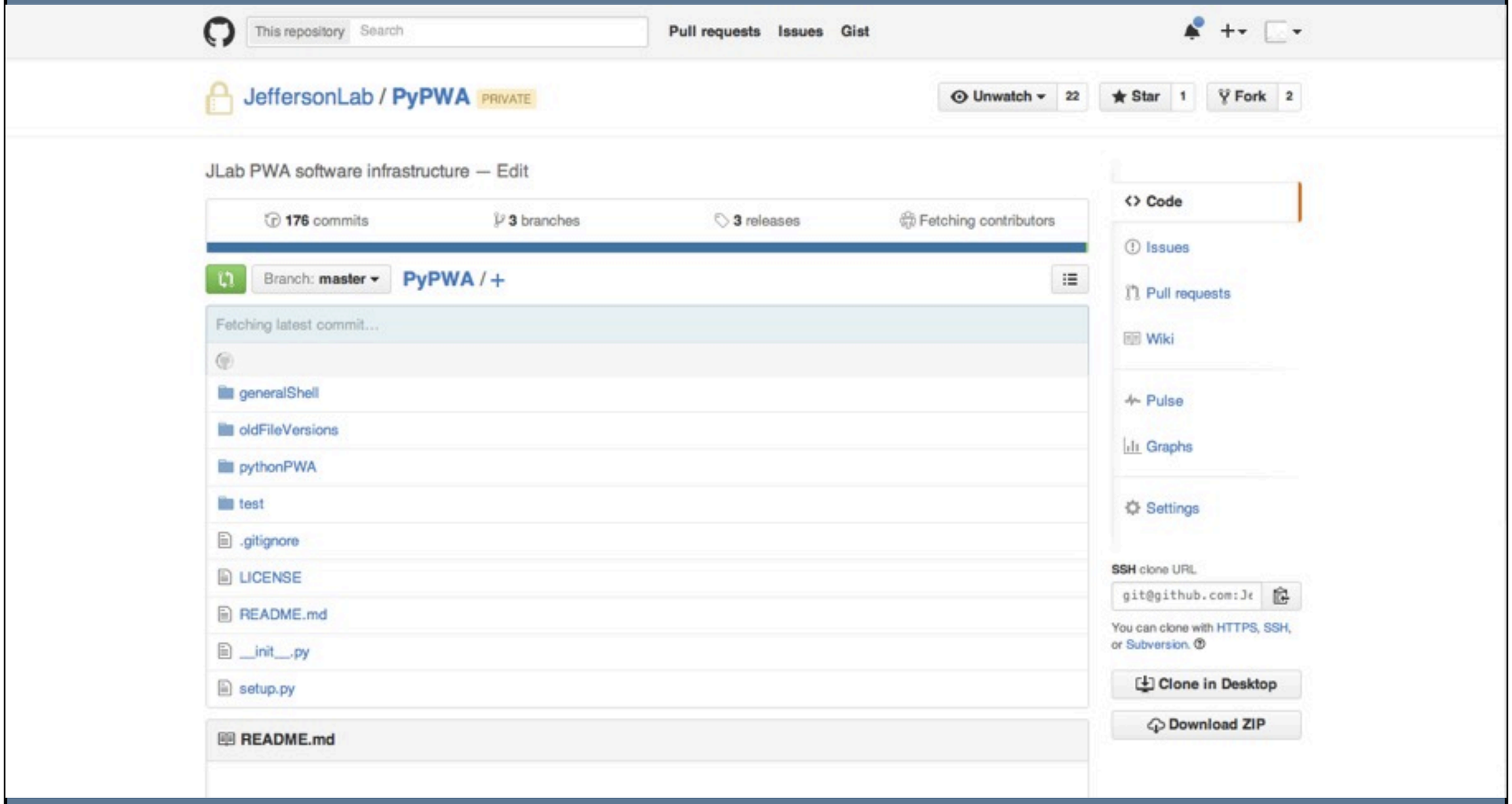


Table Of Contents

Welcome to pythonPWA's documentation!
Source Listing
Indices and tables

Next topic

dataTypes

Quick search

Go

Enter search terms or a module, class or function name.

Welcome to pythonPWA's documentation!

Source Listing

- General Shell
  - FnTemplate
  - generalFitting
  - FnSimTemplate
  - generalSim
  - kvParser
- dataTypes
  - gampEvent
  - gampParticle
  - resonance
  - wave
- fileHandlers
  - bampReader
  - gampReader
  - getWavesGen
  - gampTranslator
- utilities
  - brietWigner
  - FourVec
  - LorentzTransform
  - phaseMotion
  - ThreeVec
  - randM
  - rotation
- model
  - complexV
  - getPhi
  - intensity
  - magV
  - normInt
  - nTrue
  - prodAmp
  - spinDensity
- batchFarmSerivices
  - GUI_alpha_main
  - GUI_gamp_main
  - GUI_subPyNormInt_main
  - PWA_GUI

Jefferson Lab

PyPWA 1 documentation »

Table Of Contents

generalShell
- FnTemplate
- generalFitting
- FnSimTemplate
- generalSim
- kvParser

Quick search

Go

Enter search terms or a module, class or function name.

generalShell

This module contains the general fitting and simulation programs.

FnTemplate

This file is a template the user can utilize to w

This template includes examples of all critical

generalFitting

class generalShell.fitting.generalFitting.**ge**

This class is where the work of generalShe

__init__(*dataDir=None,accDir=None,Q*

generalFitting class default constructo

Kwargs:

dataDir (string): Filepath of data kv te

accDir (string): Filepath of accepted M

QDir (string): Filepath of Q factor text

genLen (int): Integer value for the num

initial (dictionary): Dictionary of initial

calcLnLike(*params*)

Calculates the log of the Likelihood fu

Args:

params (dictionary): Dictionary of fitte

FnSimTemplate

This file is a template the user can utilize to w

This template includes examples of all critical

dataTypes — pythonPWA 1.0.5 documentation - Mozilla Firefox

File Edit View History Bookmarks Tools Help

dataTypes — pythonPW...

file:///home/salgado/bdemello/pythonPWA/pythonPWA/docs/finalDocumentation/dataTypes.html#modu

Google

pythonPWA 1.0.5 documentation »

previous | next | modules | index

Table Of Contents

dataTypes
- gampEvent
- gampParticle
- resonance
- wave

Previous topic

fileHandlers

Next topic

utilities

This Page

Show Source

Quick search

Go

Enter search terms or a module, class or function name.

dataTypes

This module contains the various data types used within the PWA project.

gampEvent

class pythonPWA.dataTypes.gampEvent.**gampEvent**(*particles=, []accepted=None, raw=None*)

This class represents a single gamp event. That is to say that this class contains a set of particles and a flag to specify if this event is accepted into the filtered data set.

gampParticle

class pythonPWA.dataTypes.gampParticle.**gampParticle**(*particleID=None, particleCharge=None, particleXMomentum=None, particleYMomentum=None, particleZMomentum=None, particleE=None*)

This class represents a particle described in a single line of a .gamp file.

toString()

Returns a string of the particle data members delimited by newlines.

resonance

class pythonPWA.dataTypes.resonance.**resonance**(*cR=1.0, wR=, []w0=1.0, r0=0.5, phase=0.0*)

This class represents a resonance.

toString()

Returns a string of the resonance data memebers delimited by newlines.

wave

class pythonPWA.dataTypes.wave.**wave**(*epsilon=0, complexamplitudes=, []w0=1000.0, r0=100.0, beta=0, k=0*)

This class represents a PWA wave.

toString()

returns a string of all the wave properties delimited by newlines.

pythonPWA 1.0.5 documentation »

previous | next | modules | index

Summary

- **PyPWA, both General and Isobar, provides a flexible software framework for Amplitude/Partial-Wave analysis.**
- **Python is a high level language which eases the writing of scripts to write amplitudes or to interface with intensities.**
- **Access to all Python libraries (scipy, numpy,...)**
- **Integration directly to the JLab SciComp (batch farm)**
- **Integration with lower level languages is easy**
- **Includes a complete package of PWA (Isobar) in the Isobar model interfaced by GUIs**
- **Easy interface of PWA with extensions to the Isobar model.**
- **Download PyPWA at pypwa.jlab.org.**

... is a work in progress.