PyPWA

A Partial-Wave/Amplitude Analysis Software Framework

Carlos W. Salgado

other team members

B. DeMello, M. Jones
W. Phelps and J. Pond

Norfolk State University
and
The Thomas Jefferson National Accelerator Facility

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

- Ambiguities
- Leakages
- Contaminations

- Large Data Statistics
- Large Amount Of Simulation (MC)
- Mathematically complex amplitudes (models)
- Possible more complex methods of analysis

In this environment, we need to identify the poles on the S-Matrix and study the interference between states

PyPWA

Our philosophy

Liberate the user from software/hardware worries about amplitude analysis calculations. Provide the user with an "underneath" software/hardware framework (that is also accessible if the user needs to adjust). <> AUTOMATION

- Types of analysis
  - Parameter Estimation - Fitting
  - Model Selection - Bayesian
  - SIMULATION (Monte Carlo)
- Basic TOOLS/MODULAR to be use in the analysis
- Well Documented (Tutorials-in-code documentation-Sphinx)
- Interact with multiple programing languages
- Interact with other amplitude analysis packages
- Integrated use of the JLab Scientific Computing Resources
- Parallelization & Vectorization
- Own graphical package and interface with PyROOT (CERN)

Scientific Computing Resources at Jefferson Lab

Summary of resources at JLab:

- High Performance Computing (HPC) for LQCD, ~8,440 cores, ~380 GPUs, and 48 Xeon Phi cards
- **Batch Computing for Experimental Physics (the "farm"), ~3,800 core**s
- Multiple Disk Systems (online storage), ~1.4 Petabytes
- The Tape Library for offline storage, 10 Petabytes
- Interactive nodes, a wide area gateway node, and several system administration support nodes

All farm nodes are connected to both an Ethernet fabric and an Infiniband fabric, where the IB fabric is used for high speed access to the file servers.

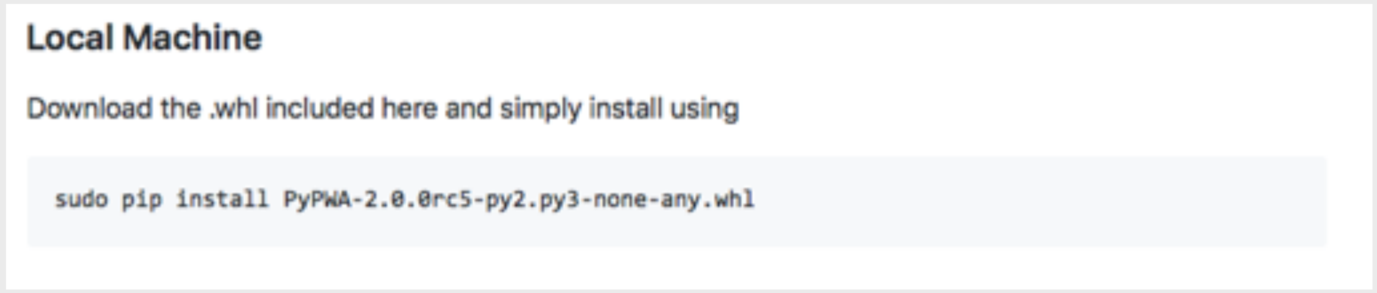
Xeon Phi (Knights Landing) + OmniPath Cluster (LQCD)

- **16p** (2016 Phi, formally known as **SciPhi-XVI**) -- 264 nodes, 64 cores, 16 GB high bandwidth memory, 192 GB main memory, Omni-Path fabric (100 Gb/s), 1TB disk

Each Knights Landing (KNL) node has 64 cores, hyper-threaded 4 ways (256 virtual cores) running at 1.3 GHz. The on-package high bandwidth memory has a bandwidth above 450 GB/s, and the main memory has a bandwidth of about 90 GB/s (available concurrently).

Implementation: PYTHON using basic numpy and scipy libraries

-Installation can be accomplished using a simple pip install command for the whole package (command as a screenshot below)


Local Machine

Download the .whl included here and simply install using

```
sudo pip install PyPWA-2.0.0rc5-py2.py3-none-any.whl
```

- Scripts/GUI driven use of JLab resources (Farms).

- Vectorization works by exploiting the combined add-multiply unit of the Intel Xeon Phi
- Include full low-level self-testing code (see below)
- 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...

- **Features and Design**
  - Pythonic OOP practices
  - Python 2 & 3 compatibility
    - Python 2.7 & Python 3.3 - 3.6
  - Use of plugins to provide ultimate flexibility for each fit.
    - Each task is a plugin: multiprocessing, data loading, each minimizer, and even the main program logic are all loaded as plugins during the startup tasks..
  - Code design inspired by Robert Martin's book: "Clean Code"
    - Explicit method and variable naming, clean tests (low-level self-testing - see below)
  - Simplified parallelization using simple extendable interfaces that can be utilized without any user involvement.
- **Tools**
  - Program is shared through a python wheel.
    - Installed using pip
    - Dependencies are automatically installed with the program.
    - Optionally can be exported to a .deb or .rpm for Linux systems.
  - Travis Build and Testing system (see below)
    - With each push up to the Github, Travis will download the changes, build the package, and run all the py.tests we have written for the package. Upon failure we are notified.
    - Package is tested against Python 2.7, and 3.3-latest
  - Documentation
    - Documentation written in restructured text, rendered using sphinx, and uploaded to Readthedocs.io
    - Little inline documentation, instead relying on clear and clean design with explicit naming to explain the function
    - of the object or method.
    - Documentation instead being written at the top of important files with a lot of functionality.

- **User interaction**.
  - Plugins, Main, and General settings are all configured in a Yaml File.
    - Simple way to generate the yml file automatically, with as few or as many options to tweak as the user would like.
  - All interactions with the program are through the command line.
  - Full logging support, with multiple tears of logging, from just warnings to full debug support.

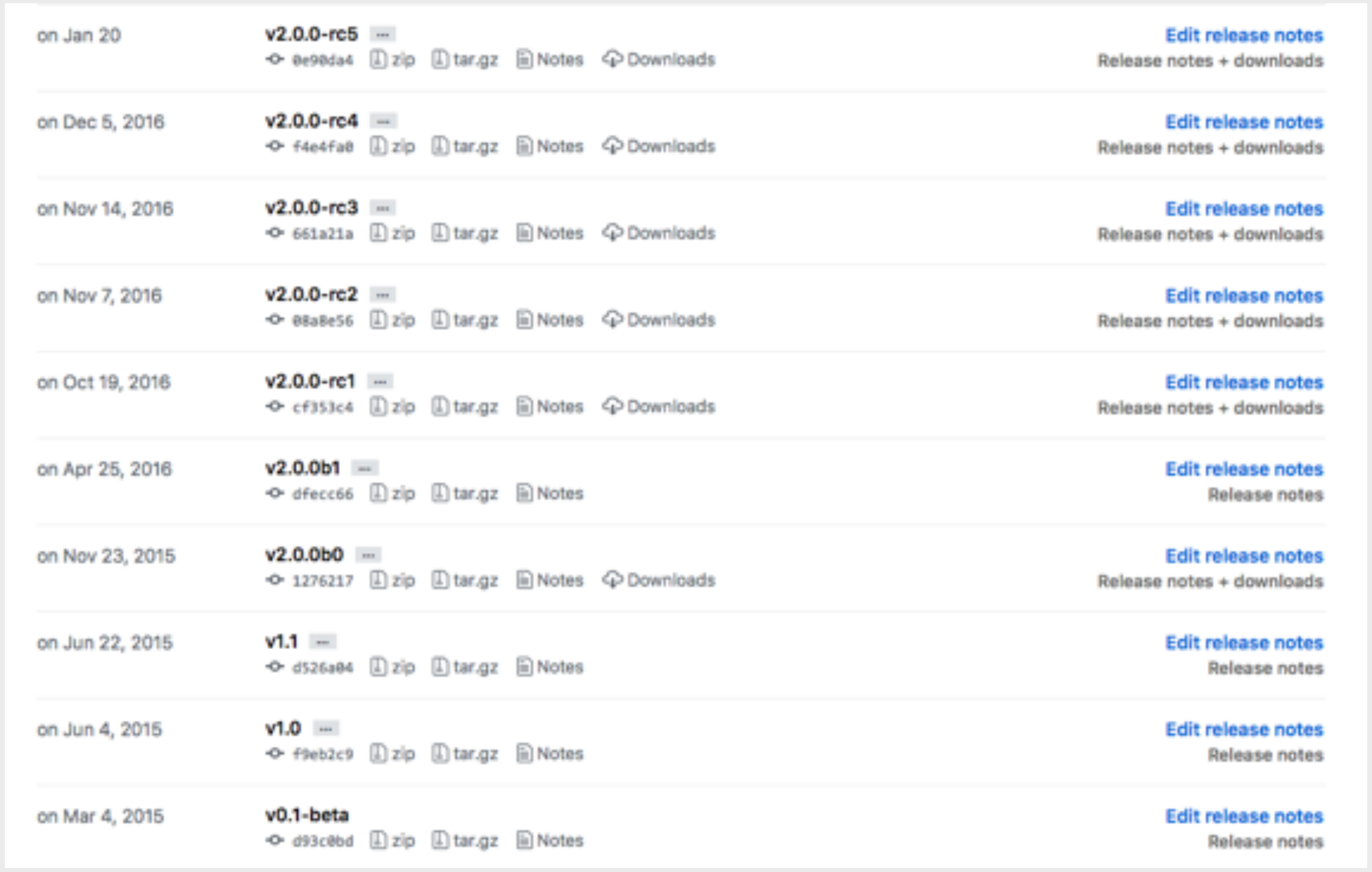
In Progress.

- The addition of a graphical and text interface.
- The ability to store live point data from Nestle of the fitting process from the Minimizer's perspective.
- Packaging into a Mac OS X Bundle, Debian .deb, and Redhat .rpm complete with all dependencies.
- Complete documentation of the design and internals of the program for future developers.

PyPWA build passing

-~80% test coverage, meaning 80% of the lines of code are covered under low-level unit testing.

simple logical consistence - numerical ascertain



on Jan 20	**v2.0.0-rc5** ··· 0e90da4 zip tar.gz Notes Downloads	Edit release notes Release notes + downloads
on Dec 5, 2016	**v2.0.0-rc4** ··· f4e4fa8 zip tar.gz Notes Downloads	Edit release notes Release notes + downloads
on Nov 14, 2016	**v2.0.0-rc3** ··· 661a21a zip tar.gz Notes Downloads	Edit release notes Release notes + downloads
on Nov 7, 2016	**v2.0.0-rc2** ··· 08a8e56 zip tar.gz Notes Downloads	Edit release notes Release notes + downloads
on Oct 19, 2016	**v2.0.0-rc1** ··· cf353c4 zip tar.gz Notes Downloads	Edit release notes Release notes + downloads
on Apr 25, 2016	**v2.0.0b1** ··· dfecc66 zip tar.gz Notes	Edit release notes Release notes
on Nov 23, 2015	**v2.0.0b0** ··· 1276217 zip tar.gz Notes Downloads	Edit release notes Release notes + downloads
on Jun 22, 2015	**v1.1** ··· d526a04 zip tar.gz Notes	Edit release notes Release notes
on Jun 4, 2015	**v1.0** ··· f9eb2c9 zip tar.gz Notes	Edit release notes Release notes
on Mar 4, 2015	**v0.1-beta** d93c0bd zip tar.gz Notes	Edit release notes Release notes

software structure

The PyPWA framework and toolkit is divided in

GENERAL-SHELL (PyFit,PySim)

- Fitting and Simulation.
- User can input any model.
- Interface is through user defined Python scripts using templates.
- Integrated batch farm interface.
- Multithreaded.
- Simulation produces “masks” to be used on user formatted MC.

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 (4-momenta) and the GAMP amplitude generator utilizing “keyfiles” for physics descriptions.
- Optional use of “Q factor” - quality
- Interface is with GUIs
- Interacts directly and exclusively with the JLab batch farm
- Integrated plotting through Python

[1] Cummings and Weygand (PWA2000)

General Shell
PyFit - PySim

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

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

Minimization (Default): **Minuit or Nestle**

many others are easily available from scipy.optimize

some specifics of the general-shell

For both fitting and simulation there is one file that the user interacts (configuration file) and one the model is provided.

The model can be provided in FORTRAN/C++/Python/Jave into a python shell provided by PyPWA

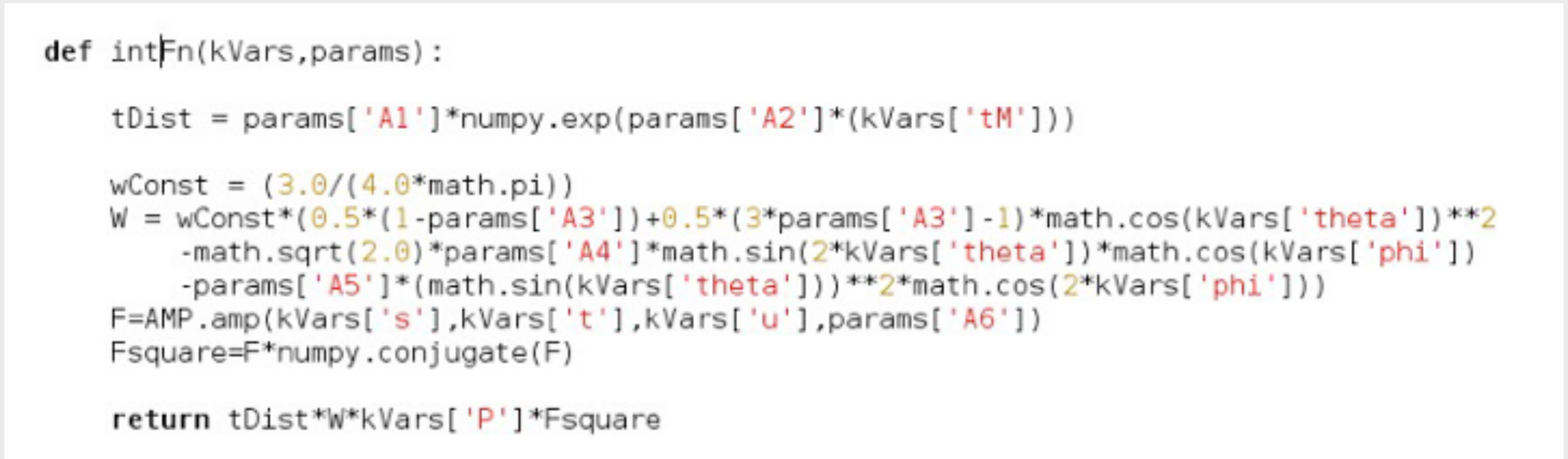
Simulation and fitting take text files of variables in a general txt format:

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

simulation starts normally with already simulated phase space and produces two “masks” to be applied to those events
- production mask
- acceptance mask

according to the model (by rejection sapling)

Example (using Nestle as minimizer)



```
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
```

```
def setup_function():
   AMP.dummy()
   pass

def prior_function(x):
   y = numpy.array([10.E+12*x[0], 25*x[1]-15]
   return y
```

This is an example of the sort of function you can fit
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 Nestle.

For a fit the you run

`>Py Fit -wc` it produces a configuration file →

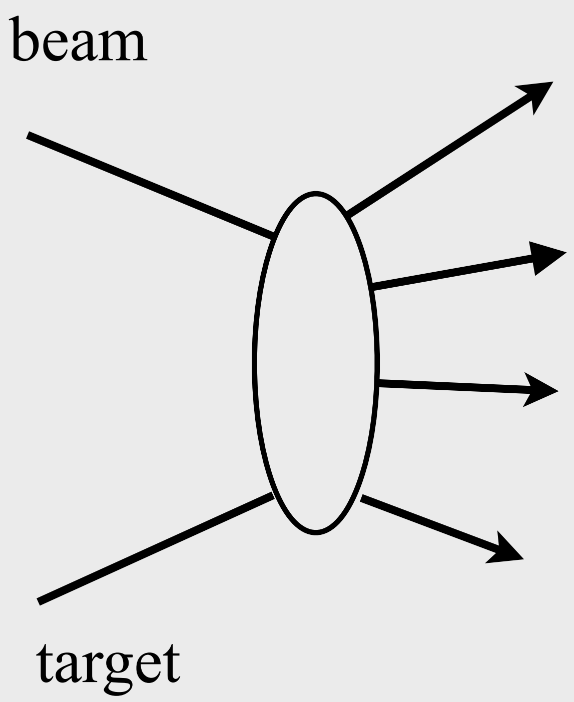
```
Builtin Parser:
  enable cache: true
Builtin Multiprocessing:
  number of processes: 8
Nestle:
  ndim: 5
  prior name: prior_function
  prior location: Fn.py
  npoints: 2500
  method: multi
General Fitting:
  function's location: Fn.py
  save name: output
  setup name: setup_function
  processing name: intFn
  data location: /volatile/data.txt
  likelihood type: loglike
```

and the run

`>PyFit conf.file`

PWA - Isobar (Partial Wave Analysis) Formalism

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



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

Extended likelihood

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

The Spin Density Matrix of the incoming Photon is calculated

Unbinned Maximum Likelihood fit

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

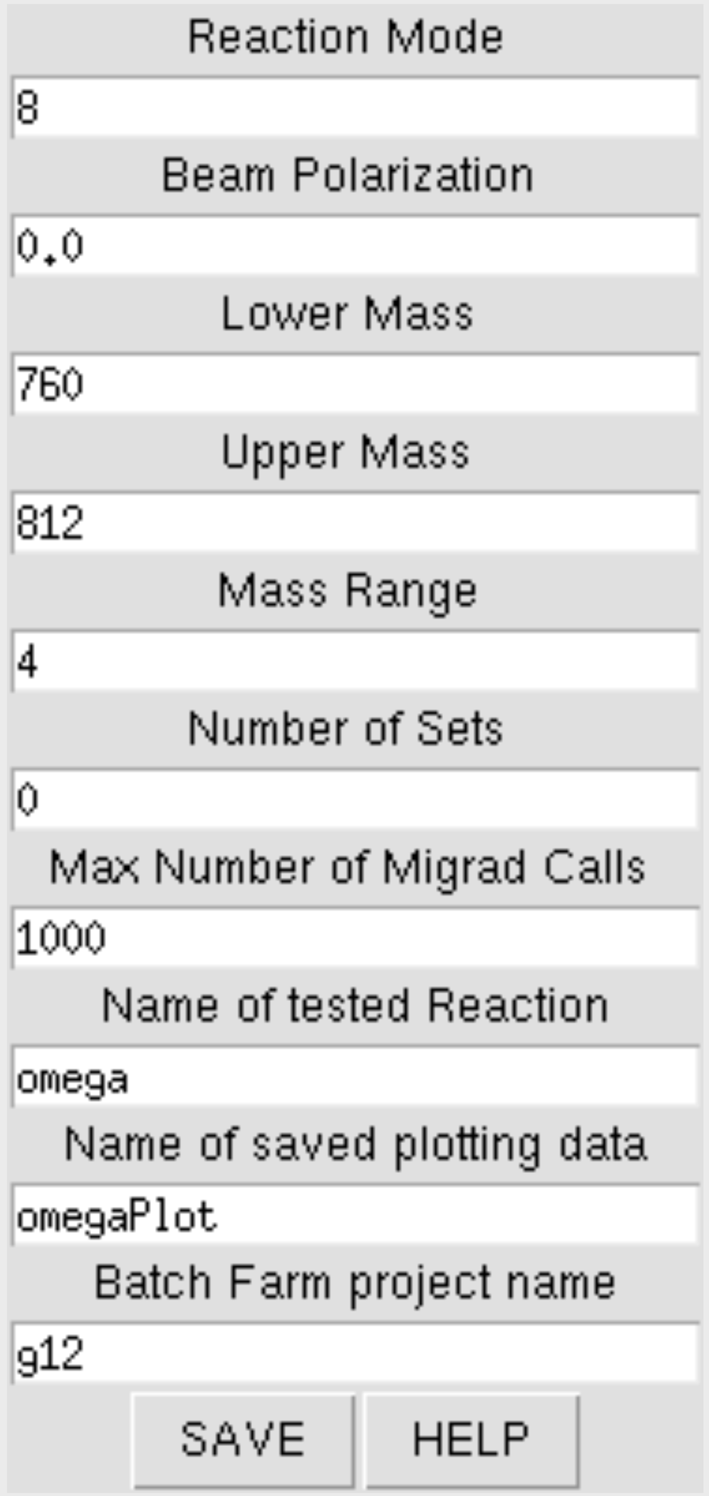
GUI (draft)

ISOBAR - PWA

The Isobar framework is focused on ease 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.



Reaction Mode
8
Beam Polarization
0.0
Lower Mass
760
Upper Mass
812
Mass Range
4
Number of Sets
0
Max Number of Migrad Calls
1000
Name of tested Reaction
omega
Name of saved plotting data
omegaPlot
Batch Farm project name
g12

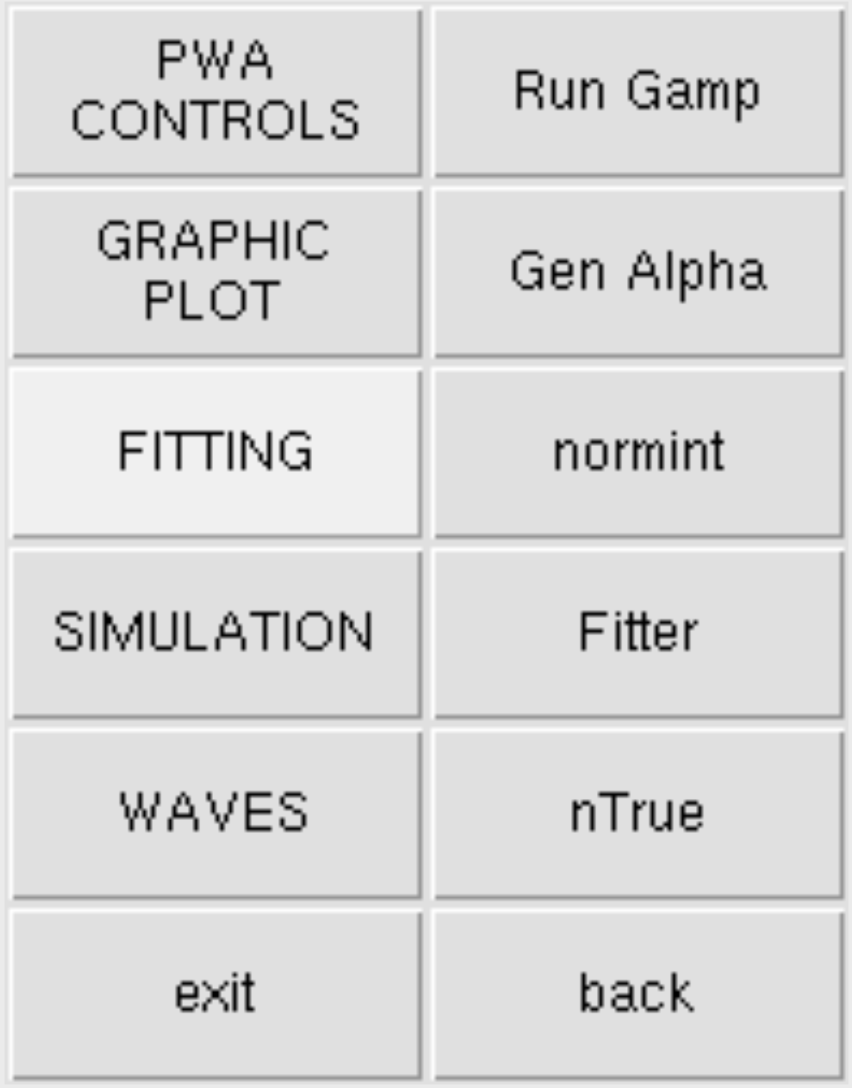
ISOBAR cont.

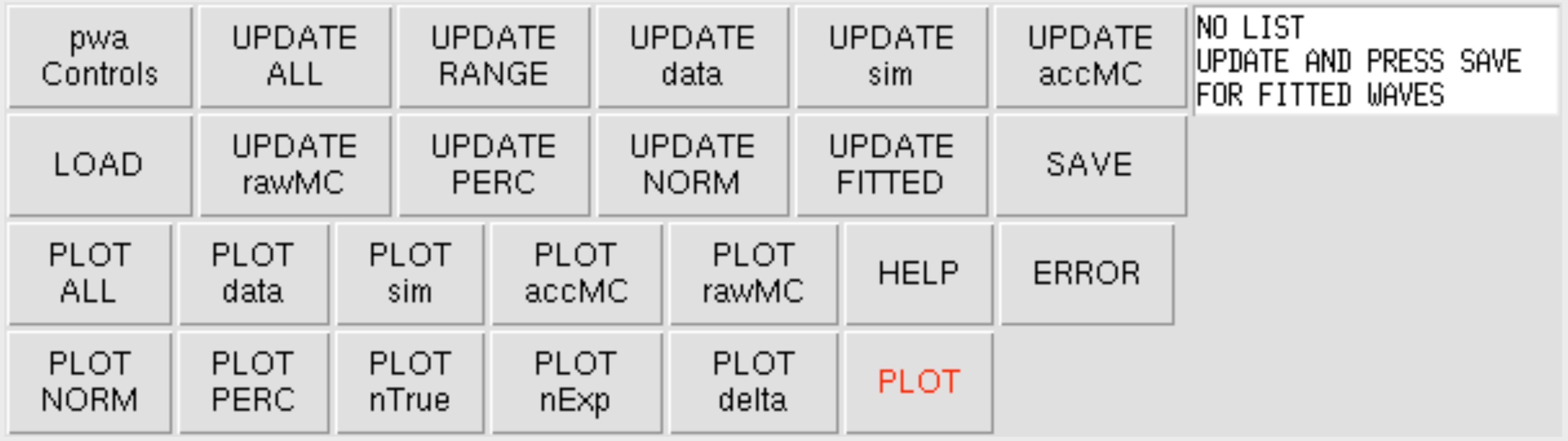
The Isobar framework's main point of contact for the user is the PWA_GUI at right. The left column is what appears when the program is run and the right is what appears after the FITTING button is pressed.

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.

GUI (draft)





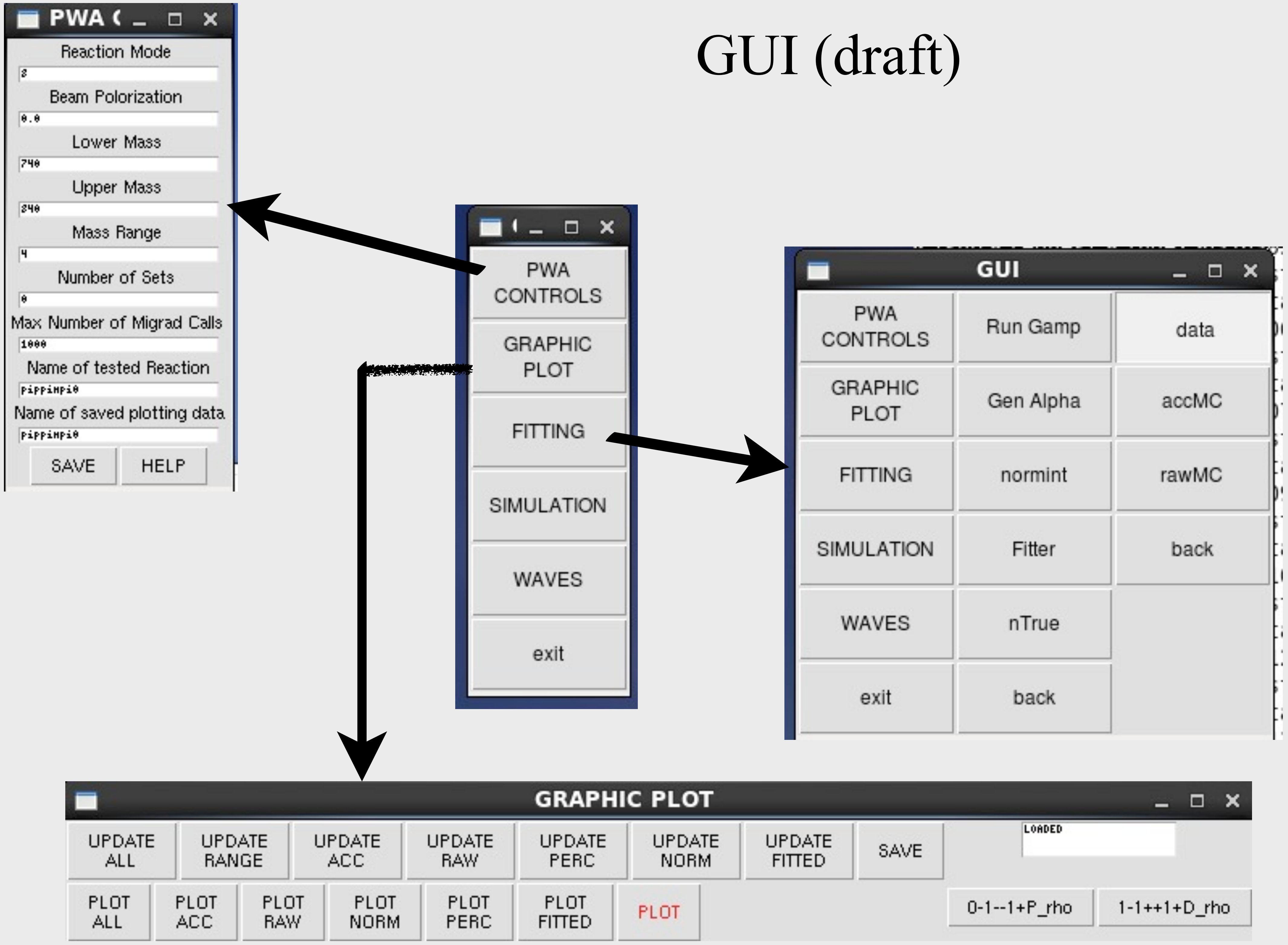
```
NO LIST
UPDATE AND PRESS SAVE
FOR FITTED WAVES
```

GUI (draft)

Plotting

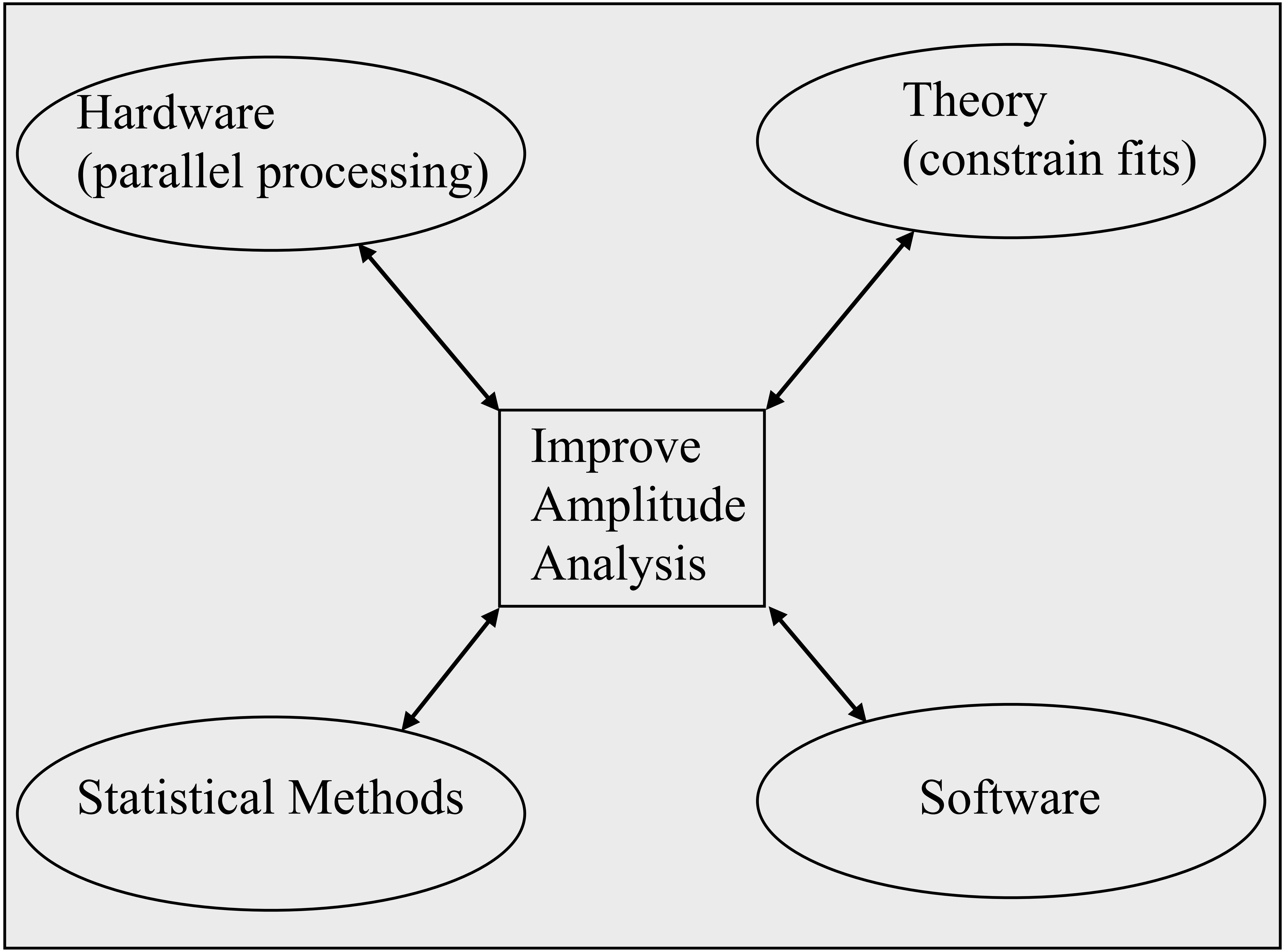
Plotting in PyPWA Isobar is handled by the above GUI which uses the MatPlotLib Python library for all plotting.
This program also consolidates all data for plotting into single file named in the control. This file can be loaded in the future and multiple files can be saved and loaded at different times.

GUI (draft)



PWA (
Reaction Mode: 8
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

GRAPHIC PLOT: LOADED


Hardware
(parallel processing)
Theory
(constrain fits)
Improve
Amplitude
Analysis
Statistical Methods
Software

MODEL SELECTION

A Bayesian Approach

(or wave-set selection)

(CLAS members Dave I, Derek G.)

$$PR = \frac{prob\left(A|D,I\right)}{prob\left(B|D,I\right)} = \frac{prob\left(A|I\right)}{prob\left(B|I\right)} \cdot \boxed{\frac{prob\left(D|A,I\right)}{prob\left(D|B,I\right)}}$$

Prior ratio

Bayes' Factor

ratio of " evidences"

$$prob\left(D|p,I\right) prob\left(p,I\right) = prob\left(p,D|I\right) = prob\left(D|I\right) prob\left(p|D,I\right)$$

Likelihood x Prior = Joint = **Evidence** X Posterior

$$Z(Evidence) = prob\left(D|I\right) = \int L(x)\pi(x)dx$$

To evaluate evidence from a flat prior is very expensive (time)

Methods to improve on prior

1. Laplace Aprox. (Gaussians)
2. Inportance Sampling
3. Anneled importance
4. Variational Bayes
5. Hamilton Aprox.
6. **Nested Sampling**

MODEL SELECTION

A Bayesian Approach

Nested Sampling

Skilling J., 2004, in Fischer R., Preuss R., Toussaint U. V., eds, American Institute of Physics Conference Series Nested Sampling. pp 395–405

BOOK: **Data Analysis;A Bayesian Tutorial**; Sivia and Skilling.

- Parameter estimation (Max. Like.)
- Model Selection (Evidence)

$$Z(Evidence) = prob\left(D|I\right) = \int L(x)\pi(x)dx$$

$$X(L) = \int_{-\infty}^{-\log L} dE\; g(E) = \int_{P(d|m,M,I)>L} dm\; P(m|M,I)$$

$$Z = \int_{-\infty}^{\infty} dE\; g(E)e^{-E} = \int_{0}^{1} dX\; L(X) \approx \sum_{i} L_i(X_{i-1} - X_i)$$

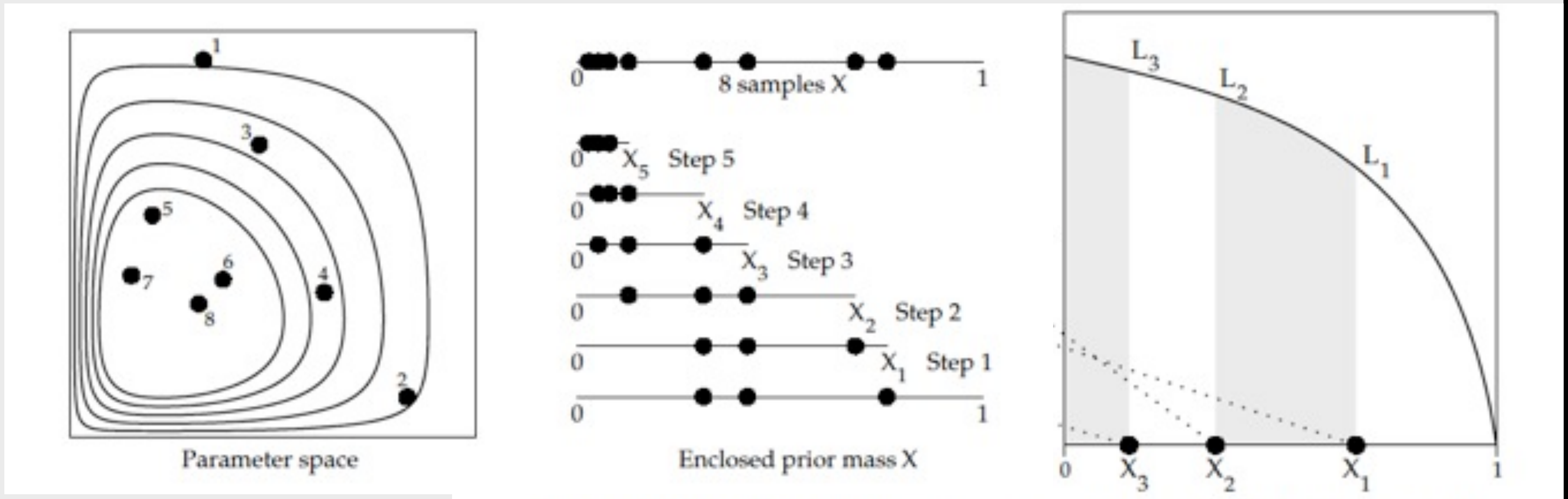


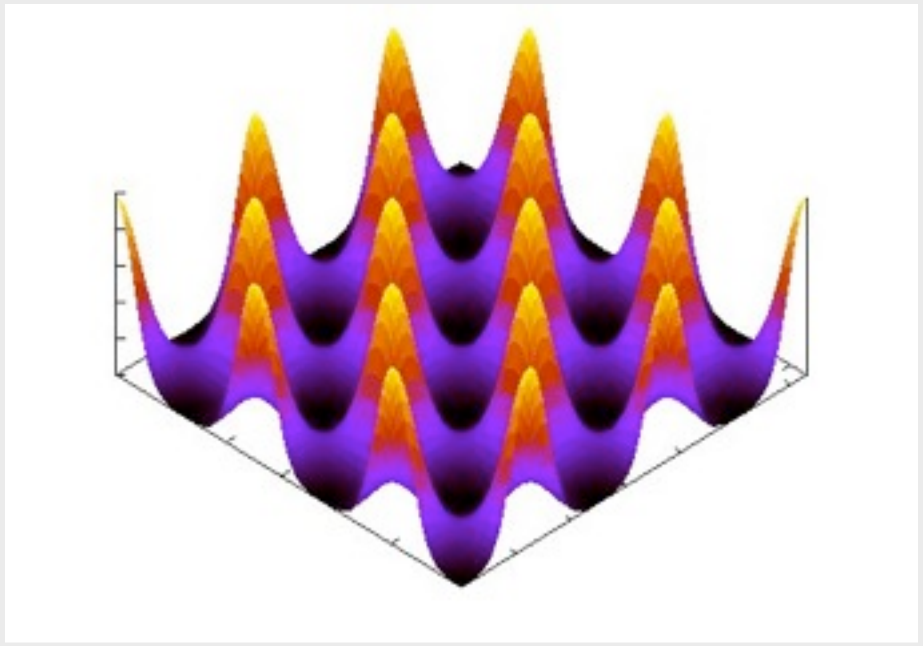
Figure 3: Nested likelihood contours are sorted to enclosed prior mass X.

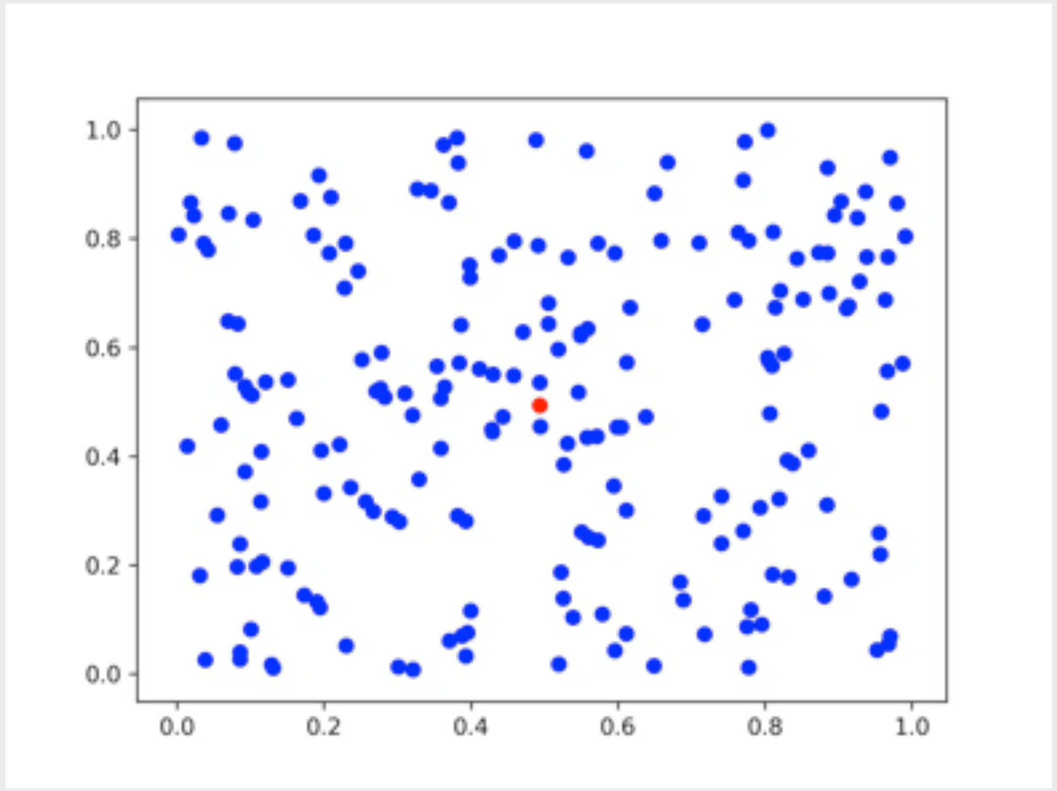
Nestle (python from Multinest - Nested Sampling)

Python code implementation by K. Barbary (Berkeley Institute for Data Analysis)

http://kbarbary.github.io/nestle/index.html

Classical example of multiple maxima: Eggbox





other types of “Model Selection”: Penalize Likelihood (i.e LASSO)

$$PR = \frac{prob\left(A|D,I\right)}{prob\left(B|D,I\right)} = \frac{prob\left(A|I\right)}{prob\left(B|I\right)} \underbrace{\boxed{\frac{prob\left(D|pA,A,I\right)}{prob\left(D|pB,B,I\right)}}}_{\boxed{\text{Likelihood ratio}}} \underbrace{\frac{\delta pA(pB_{max}-pB_{min})}{\delta pB(pA_{max}-pA_{min})}}_{\boxed{\text{Penalty}}}$$

$$\mathbb{L}_{LASSO} = \mathbb{L} \prod_{i}^{waves} e^{-\lambda\left|\mathbb{A}_i\right|}$$

Guegan et al.(2015)

$$\mathbb{L}_{Cauchy} = \mathbb{L} \prod_{i}^{waves} \frac{1}{1+\frac{\left|\mathbb{A}_i\right|^2}{\Gamma^2}}$$

B.Grube (COMPASS)

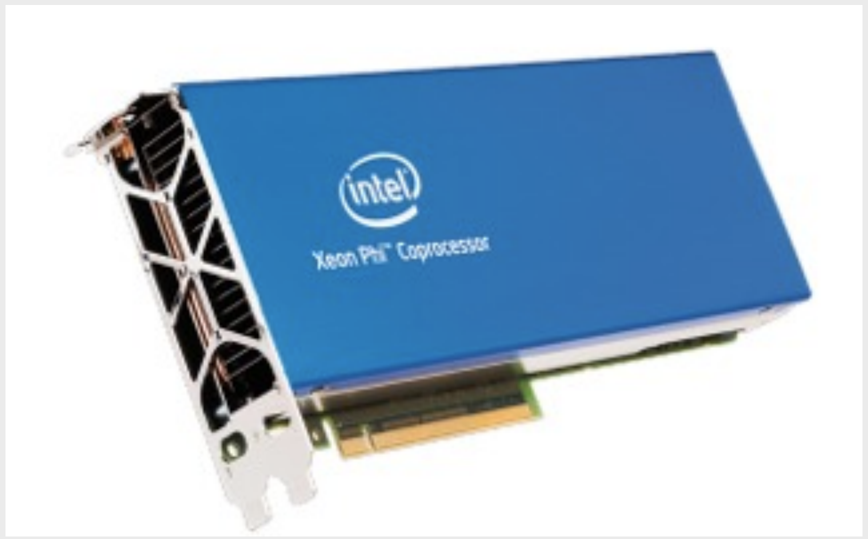
C. Adolph et al., PRD 95 (2017) 032004

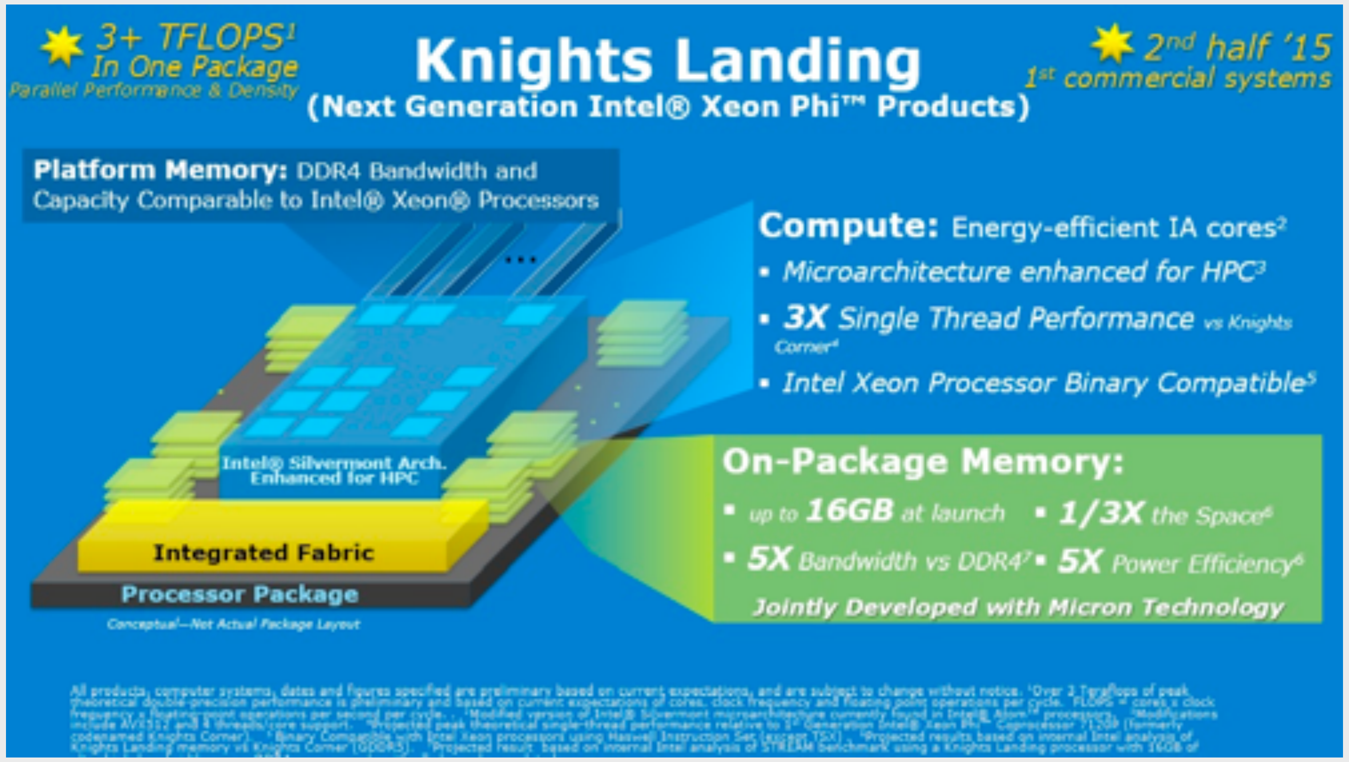
How to quantify bias?

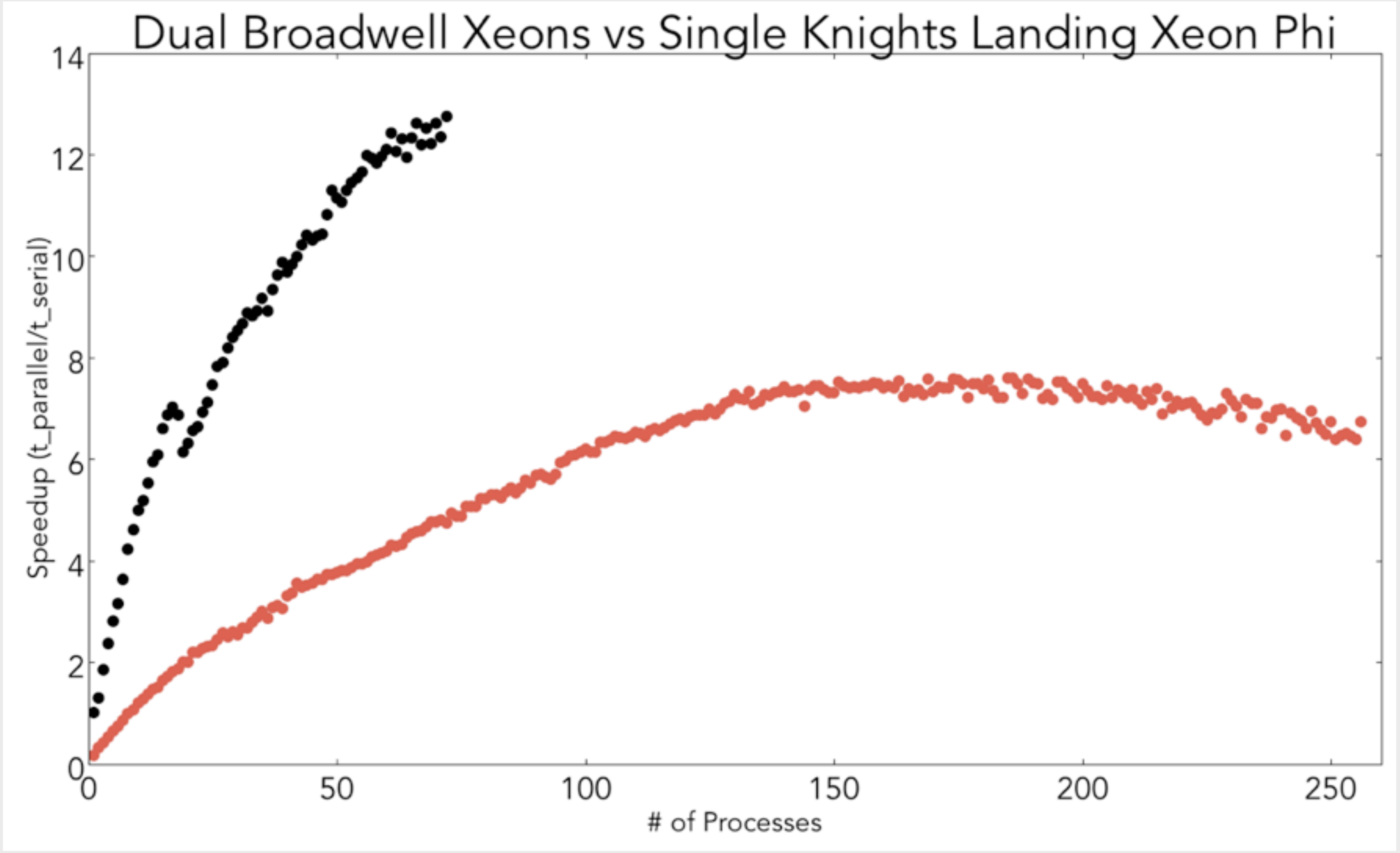
Intel-Xeon-Phi cards using for example **OpenMP** . Xeon Phi's contain about 61 of x86 cores that are functionally identical to those of standard laptops and desktops. There are just many more of them running at a lower clock speed to fit into a reasonable thermal design envelope (currently a PCI Express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

Knight's Landing:

- 64 Silvermont cores
- Socketed and PCI-Express versions available
- Back to homogeneous computing?



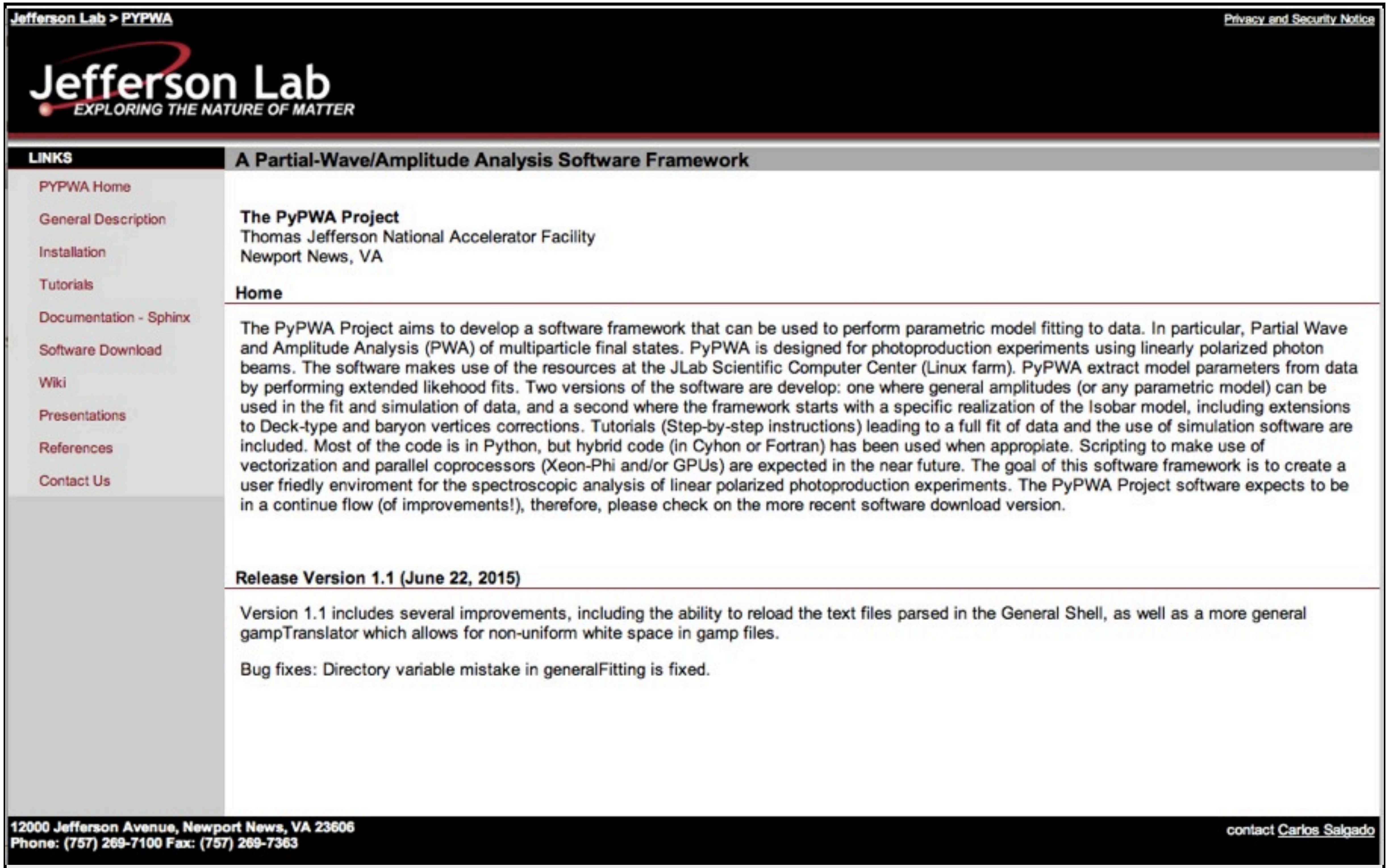

3+ TFLOPS¹ In One Package
Parallel Performance & Density
Knights Landing
(Next Generation Intel® Xeon Phi™ Products)
2nd half '15
1st commercial systems
Platform Memory: DDR4 Bandwidth and Capacity Comparable to Intel® Xeon® Processors
Compute: Energy-efficient IA cores²
Microarchitecture enhanced for HPC³
3X Single Thread Performance vs Knights Corner⁴
Intel Xeon Processor Binary Compatible⁵
Intel® Silvermont Arch. Enhanced for HPC
Integrated Fabric
Processor Package
Conceptual—Not Actual Package Layout
On-Package Memory:
up to 16GB at launch
1/3X the Space⁶
5X Bandwidth vs DDR4⁷
5X Power Efficiency⁶
Jointly Developed with Micron Technology



Main point -> -Python multiprocessing affords users the ability to utilize all of the cores on their local machine without having to fight with any specialized tools for parallelizing their code

-The Dual Xeon system should be expected to perform better, it’s more than twice the cost.

-The fitter scales well considering it is python multiprocessing and it does not compromise on portability.


Jefferson Lab > PYPWA

Privacy and Security Notice

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.

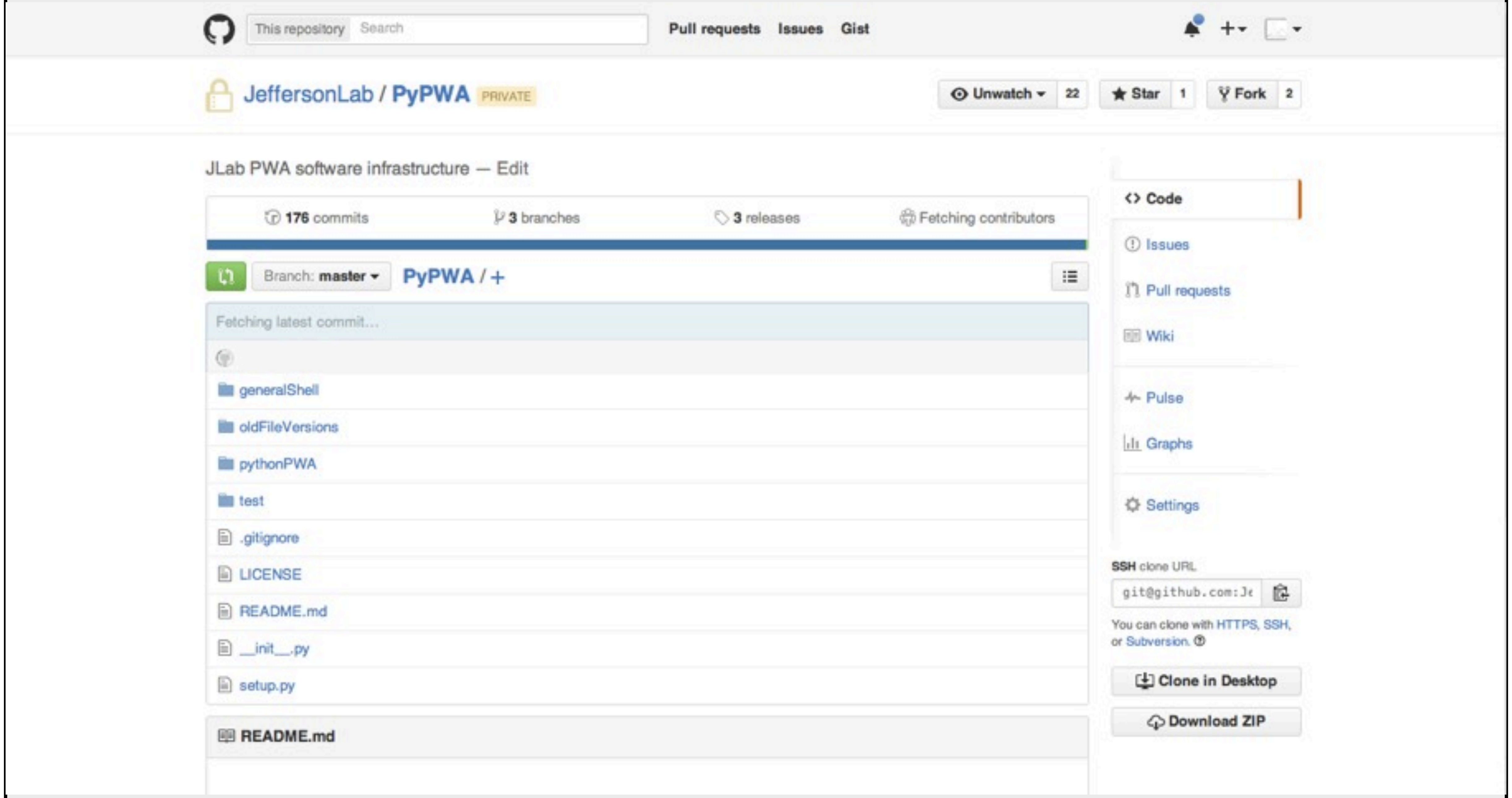
Bug fixes: Directory variable mistake in generalFitting is fixed.

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
code: github JeffersonLab/PyPWA
Sphinx generated : docs


This repository Search
Pull requests Issues Gist

JeffersonLab / PyPWA PRIVATE

Unwatch 22 Star 1 Fork 2

JLab PWA software infrastructure — Edit

176 commits | 3 branches | 3 releases | Fetching contributors

Branch: master PyPWA / +

Fetching latest commit…

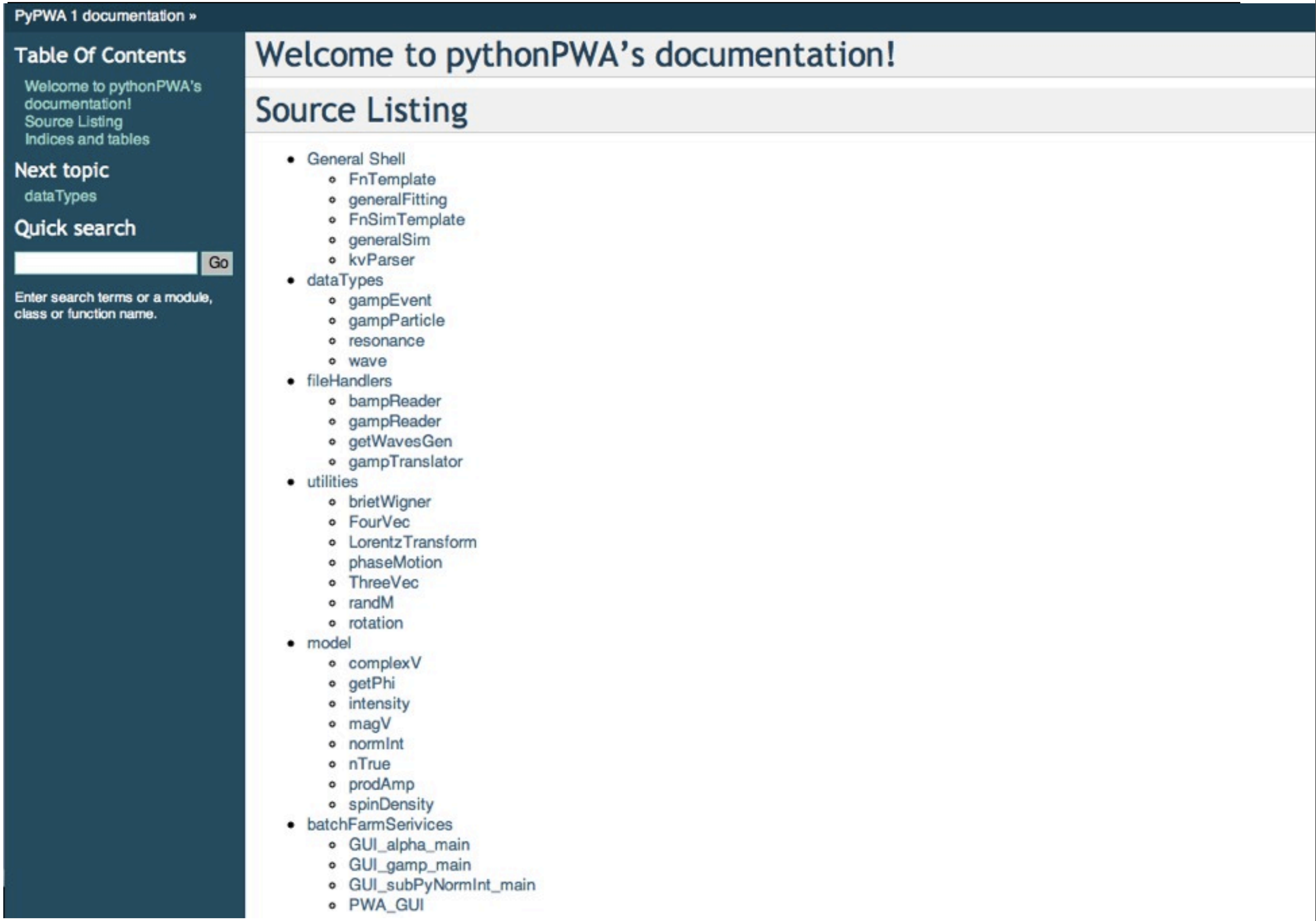
- generalShell
- oldFileVersions
- pythonPWA
- test
- .gitignore
- LICENSE
- README.md
- __init__.py
- setup.py

README.md

SSH clone URL

git@github.com:Je

You can clone with HTTPS, SSH, or Subversion.



PyPWA 1 documentation »

Table Of Contents

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

Summary

• PyPWA, both General and Isobar tools provide a “underneath” software framework for user’s Amplitude/Partial-Wave analysis.

• Integration directly to the JLab SciComp

• Integration with lower level languages is easy - Any amplitude in mostly any language (i.e FORTRAN) can be used directly.

• Python multiprocessing affords users the ability to utilize all of the cores on their local machine without having to fight with any specialized tools for parallelizing their code.

• Includes a complete package of PWA (Isobar) in the Isobar model interfaced by GUIs

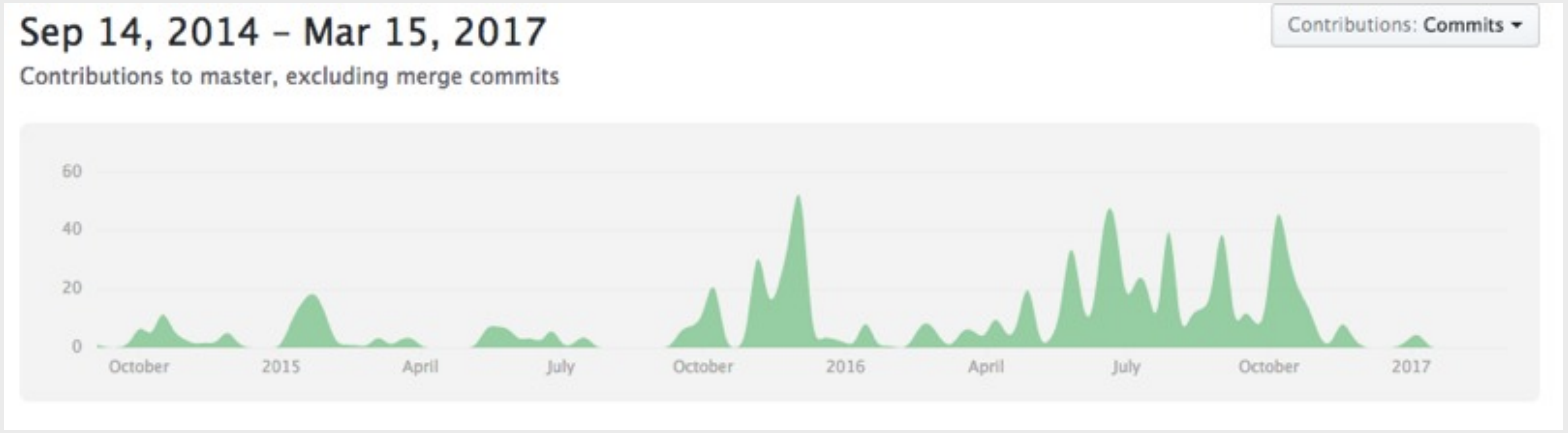
• ~80% test coverage, meaning 80% of the lines of code are covered under unit testing .

• The code is designed with flexibility in mind, allowing users/us to create plugins for new data types, minimizers, and amplitudes.

• Currently we have two different minimizers built in, a package containing several nested sampling algorithms (Nestle), and good ol’ Minuit.

• Installation can be accomplished using a simple pip install command for the whole package

• utilizes optimized fortran/C++ libraries on the backend by using numpy/scipy/whatever



• **Download PyPWA at**
https://github.com/JeffersonLab/PyPWA/

... is a work in progress.

welcome partners to use all/parts of the infrastructure
... and contribute!... just contact us at https:/pypwa.jlab.org