# PyPWA Documentation

*Release Development*

**PyPWA Team**

**Oct 11, 2022**

# CONTENTS

Partial Wave Analysis done right.

PyPWA is a Python partial wave analysis package that utilizes Numpy, iminuit, and PyTables to provide a high speed analysis framework that strives to help you get your work done without getting in your way.

It's a package that can be used either as a standalone program inside your terminal, or as Python script or Jupyter Notebook, whatever your preference may be.

You can take a look at our code directly here

CHAPTER

ONE

# ABOUT

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 likelihood 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 Cython or Fortran) has been used when appropriate. 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 friendly environment 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.

## 1.1 What can PyPWA do?

- Likelihood fitting with ChiSquared and Log Likelihood
- Simulation using the Monte-Carlo Rejection Sampling method
- Multi-variable binning for 4 vector particle data (in GAMP Format)
- Convert and mask data between similar data types
- Load data into an HDF5 dataset

## 1.2 Further Reading

- iMinuit
- Nestle
- PyTables (HDF5)

## 1.3 Team Members

**Current PyPWA Team members**

- Dr. Carlos Salgado Norfolk State University
- Dr. Will Phelps Christopher Newport University
- Mark Jones VPCC and Old Dominion University
- Dr. Peter Hurck University of Glasgow

**Previous PyPWA Team members**

- Brandon DeMello Old Dominion University
- Stephanie Bramlett William and Mary
- Josh Pond Virginia Peninsula Community College (VPCC)
- LaRay Hare Norfolk State University
- Christopher Banks Norfolk State University
- Michael Harris Jr Norfolk State University

**High School Interns**

- Ryan Wright Hampton Governor's School for Science and Technology
  - Ran Amplitude benchmarks on the XeonPhi

## 1.4 Citations

- Roger Barlow. Extended maximum likelihood. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 297(3):496–506, 1990.
- Ph R BEVINGTON. Data reduction and error analysis for the physical sciences. Technical Report, McGraw-Hill, 1969.
- Suh Urk Chung. Spin formalisms. Technical Report, CERN, 1971.
- WT Eadie, D Drijard, FE James, M Roos, and B Sadoulet. Statistical methods in experimental physics, 2nd reprint. 1982.

- M Jacob and Gr C Wick. On the general theory of collisions for particles with spin. *Annals of Physics*, 7(4):404–428, 1959.
- F James. Minuit reference manual, cern program library long writeup d506. *James and M. Winkler, MINUIT User's Guide, CERN*, 1994.
- Fred James, Matthias Winkler, and others. Minuit user's guide. *MIGRAD CERN*, 2004.
- David JC MacKay and David JC Mac Kay. *Information theory, inference and learning algorithms*. Cambridge university press, 2003.
- J Orear. Notes on statistics for physicists (1958). *UCRL-8417*, 1982.
- Carlos W Salgado and Dennis P Weygand. On the partial-wave analysis of mesonic resonances decaying to multiparticle final states produced by polarized photons. *Physics Reports*, 537(1):1–58, 2014.
- K Schilling, P Seyboth, and G Wolf. On the analysis of vector-meson production by polarized photons. *Nuclear Physics B*, 15(2):397–412, 1970.
- John Skilling. Nested sampling. In *AIP Conference Proceedings*, volume 735, 395–405. AIP, 2004.
- Charles Zemach. Use of angular-momentum tensors. *Physical Review*, 140(1B):B97, 1965.

# CHAPTER TWO

# INSTALLATION

PyPWA can be installed with pip or conda with Python 3.7 or newer

## 2.1 Conda

Thanks to tools provided by Anaconda, you can easily install PyPWA and all it's dependencies with a simple one line command. Check out Anaconda's user guide if you're new to using Anaconda.

```
conda install -c markjonestx pypwa
```

If you want tools from PWA2000 (GAMP, HGAMP, VAMP, PPGEN) we've included them as well

**Note:** PWA2000 is currently only available on Linux installs of Anaconda.

```
conda install -c markjonesyx pwa2000
```

## 2.2 Pip

**Warning:** Pip can interfere with your system python. Make sure to never run pip as root, and only perform local installs.

**Fetch the latest version of PyPWA and install locally**

**Note:** If you are using pip somewhere behind a firewall, you may need to pin pip's servers using `pip install --trusted-host pypi.org --trusted-host pythonhosted.org`

```
git clone --depth=1 https://github.com/JeffersonLab/PyPWA.git
cd PyPWA
pip install --local .
```

CHAPTER

THREE

# CHANGELOG

All changes important to the user will be documented in this file.

The format is based on Keep a Changelog and this project adheres to Semantic Versioning

## 3.1 Unreleased

### 3.1.1 Added

### 3.1.2 Changed

### 3.1.3 Removed

### 3.1.4 Fixed

## 3.2 4.0.0 - 2022-10-11

### 3.2.1 Added

- Anaconda environments. There are two anaconda environments included inside the source folder at the moment. `anaconda-environment.yml` and `dev-environment.yml`. These should provide a nice starting point for anyone wanting to work on or with PyPWA. Pull requests are welcomed if you think a package should be added to the base environment.
- Added PyTorch for GPU and Apple Metal support. Can be specified during install using `pip install pypwa[torch]`. Amplitude support is specified by setting the `USE_TORCH` flag to True.
- Added support for Python's Multithreading. You should only use this when computation is happening on separate nodes and/or your optimizer choice does not support passing it's values across an OS Pipe.
- Added support for Minuit's parameter array argument. Now amplitudes can be written to accept a single array containing all the array values.
- Debugging support for amplitudes is now explicit. You can set the `DEBUG` flag to True on your amplitude before simulation or fitting, and it'll run in the main process so traceback and errors will not be suppressed.

- Amplitudes can now know where they live. Amplitudes have a `THREAD` flag that is numbered from 0 to N-threads that will specify which thread the amplitude is running in. This is useful if you want to pair your processes/threads with external devices like GPUs or OpenMPI nodes.

### 3.2.2 Changed

- Data module will no longer bury the Cache object. The cache object will now reside in the same directory as the parsed data.
- Moves Emcee to an optional dependency so that PyPWA can function in a base anaconda environment. If Emcee is installed, or if emcee is specified during installation using `pip install pypwa[emcee]`, the emcee functionality will be usable.
- iMinuit has changed their ABI entirely, so the iminuit function has been changed to adapt to the new ABI.
- Updated all dependencies around ReadTheDocs to avoid GitHub flagging the dependencies for exploits.

### 3.2.3 Removed

- Project manager. There were several bugs throughout the module, and as far as we are aware, no users using the module. If you're affected by this change, please open an issue in the issue tracker to let us know.
- Removed the **command line** Binning utility. The Jupyter-based and internal binning utilities remain unaffected. If this affects you, please open an issue.
- Removed appdirs as a dependency.
- Removed CuPy support, replaced by PyTorch.
- Removed PyYaml Configuration support.

### 3.2.4 Fixed

- The bin by range function was not sampling data correctly. The intended behaviour was for each bin to be sampled by N samples, and then those samples to be shuffled to add randomization. However, because the shuffling was improperly implemented, what would occur instead is a single random event would be dropped from the sample, and then returned. This no longer occurs, and the returned bins will now be the correct length, and will be correctly shuffled.

## 3.3 3.4.0 - 2021-7-23

### 3.3.1 Added

- Peter's emcee wrapper, available at PyPWA.mcmc

### 3.3.2 Changed

- System tests are now located in tests/system_tests
- PyMask will now return exit values on call

### 3.3.3 Removed

- PySimulate has been removed since it was limited in use, and it's functionality has been consumed by the PyPWA scripting libs.

## 3.4 3.3.0 - 2021-6-20

### 3.4.1 Added

- 2D Gauss introductory tutorial to the documentation
- CuPy support for Likelihoods and Simulation. This means we now officially support NVIDIA GPU acceleration, however for now it is limited to a single GPU. If there is enough demand for this to be expanded on, support for multiple GPUs will be added.

### 3.4.2 Changed

- Particle now requires a charge to be supplied during the creation of the object. GAMP has also been modified to support the Charge being passed through to the Particle
- Depreciated internal options that were passed to Minuit have been replaced with the modern alternatives.

## 3.5 Fixed

- Likelihoods were spawning multiple processes even when USE_MP was set to false. This has been corrected, and will avoid spawning extra processes as it was intended.

## 3.6 3.2.3 - 2021-6-11

### 3.6.1 Added

- Particle Pools can now compared against other Particle Pools to see if they are storing the same content.

### 3.6.2 Fixed

- Regression from 3.2.0 where Gamp would not write out data to disk. This time by wrapping the data in a float, which should catch instances where the value stored is a pure scalar, verses instances where the data is an array with a len == 1

## 3.7 3.2.2 - 2021-6-11

### 3.7.1 Fixed

- Particles can now be masked again, the mask is no longer silently deleted when passed to the object.
- Numpy's warning about numpy.float being deprecation should be resolved.
- Any warnings about the LaTeX in the Likelihood's Docstrings being deprecated should be resolved as well.

## 3.8 3.2.1 - 2021-6-10

### 3.8.1 Fixed

- Gamp no longer combines particles with the same ID
- Fixed issue where display_raw would fail in Jupyter with Particles

## 3.9 3.2.0 - 2021-6-1

### 3.9.1 Added

- Vectors now support iPython and Jupyter Pretty printing

### 3.9.2 Changed

- Vector sanitization function has improved handling of non-array inputs

### 3.9.3 Fixed

- FourVectors variable order is now in the correct order
- Vectors now work with inputs that aren't arrays
- Patched issue with GAMP failing to write to file

## 3.10 3.1.0 - 2020-10-2

### 3.10.1 Added

- Helper functions `pwa.pandas_to_numpy` to convert Pandas data types to Numpy Structured Arrays, and `pwa.to_contiguous` to convert DataFrames and Structured Arrays columns to contiguous arrays for quicker processing and C/Fortran Support
- New experimental file format ParticleGZ, a direct-to-memory file format using pickle, csv, and Tar/GZ to compress data into a single archive for easy use.
- Reference documentation to the Read The Docs for the various modules in PyPWA.
- Initial examples section added.

### 3.10.2 Changed

- Users now have to option to request structured Numpy arrays or Pandas DataFrames from `pwa.read` and `pwa.get_reader`
- `pwa.cache` now defaults to intermediate caching, and has to be disabled for use with caching files
- Vectors str and repr field now output the mean of their theta, phi, as well as particle id and mass if they are available.
- Vectors now wrap individual numpy arrays instead of a single structured array or DataFrame. This was done to improve performance of the vector as well as to make it C contiguous.

### 3.10.3 Fixed

- `pwa.write` would fail to write CSV Numpy Arrays
- `pwa.write` would occasionally fail to detect DataFrames
- Vectors would occasionally replace their fields with just their x values.

## 3.11 3.0.0 - 2020-6-4

### 3.11.1 Added

- `ProjectDatabase` has been added handle large data manipulation on disk instead of in memory.
- Reader/Writer now share path of the file being operated on.
- Binning now works in both fixed count and ranges, and can be done entirely in memory.
- Initial Jupyter and IPython support.
- Adding lego plotting.
- Likelihoods are now standalone objects that can be combined with any optimizer.
- Resonance support now builtin using `DataFrames` as a backbone. Resonances are now saved as a two sheet excel file, and can be modified using the supplied wave and resonance objects.
- Adds support for `Numexpr` to accelerate computation.
- Simulation can be done as two separate parts through `PyPWA.simulate`
- Github Templates to help users and developers contribute to PyPWA

### 3.11.2 Changed

- Separate release tag from version info
- Package info is now stored in `PyPWA.info`
- `pydata` has officially been updated to PyPWA 3.0.
- Structured Arrays have been replaces for Pandas `DataFrames` in some cases. Vectors still based on `numpy` arrays to maintain performance.
- Reactions have been merged into `ParticlePool`.
- Vectors have been simplified to be easier to test while still being powerful to use.
- `ProcessInterfaces` now must be closed after use. This includes all Likelihood objects.
- `pwa.data` has been refactored to be easier to be completely usable by itself.

### 3.11.3 Removed

- `SlotTable` has been removed in favor of `Project`. Both use `PyTables` for the backend.
- Unsupported Python versions removed from package's classifiers.

### 3.11.4 Fixed

- GAMP no longer claims that it can read PF files.
- Cache will correctly report invalid when it's contents differ from the source file.
- `monte_carlo_simulation` and likelihoods now correctly handle exceptions that occur in the child processes.
- Pipes are correctly closed now.
- Extended Log-likelihood is now correctly calculated
- Sv Writer will now write data.
- Kv Reader will now read data.

## 3.12 3.0.0a1 - 2019-6-17

### 3.12.1 Added

- Added `numpy` reader and writer.
- Adds a helper script to clean the project directory of caches.
- Adds initial documentation for PyPWA.
- Added support for 3 Vectors, 4 Vectors, and Particles
- Added `ParticlePool` to aid in working with multiple Particles
- Added a binning utility that supports multiple binning variables and dimensions
- Added `PyTables` support, so that large datasets can be easily managed

### 3.12.2 Changed

- All program names have been lowercased
- Configuration package has been compressed into a single module
- `PySimulate` now is a library that has no UI, and has a UI portion that exclusively works with interfacing
- `Fuzzywuzzy` is now optional
- Process package is now a single module. Interface no longer uses `IS_DUPLEX`

- Bulk of program functionality moved to libs, progs being just for UI
- Builtin Plugins moved to `libs`, old plugin's plugins have still reside in plugins, but under a package with the appropriate name. I.E. data plugins are in plugins/data.
- All file related libs have been moved to libs/file
- Combined optimizers with fit library
- GAMP was updated to use Particles and ParticlePool
- Files with extra newline should parse correctly now
- CSV and TSV files will be lf instead of crlf on linux systems now

### 3.12.3 Removed

- Nestle Minimization. There is currently no clear way to have Minuit and Nestle to operate with each other nicely. Implementation for multiple optimizers will remain, as well as new associated issues created.
- Removed support for all version of Python before 3.7

## 3.13 2.2.1 - 2017-10-16

### 3.13.1 Fixed

- Setup would pull in unstable Yaml parser

## 3.14 2.2.0 - 2017-7-26

### 3.14.1 Added

- Process Plugin support for List Data
- Adds Exception handling to Processes
- PyMask support for multiple masking files.
- PyFit will now filter out events if the Bin value is 0
- The user can `AND`, `OR`, or `XOR` masks together with PyMask

### 3.14.2 Changed

- Removed previous_event from Process Interface
- Duplex Pipes are used over Simplex Pipes for Duplex Processes
- Changes `get_file_length` to using a binary buffered search.
- Moved `PyPWA.core.shared` to `PyPWA.libs`
- Split interface's plugins and internals to their own separate file based on the interfaces purpose.
- PyFit no longer assumes bins are named `'BinN'` you must specify Bin names in `'internal data'`.
- Multiplier effect for the Minimizers has been moved to the individual likelihoods.
- PyMask defaults to `AND` operations instead of `OR` now.

### 3.14.3 Fixed

- PyFit will now shutdown correctly when killed with Ctrl-C or other interrupt.
- The ChiSquared will no longer be multiplied by -1 when being minimized.
- Data Parser's Cache would crash on very large files.

## 3.15 2.1.0 - 2017-6-30

### 3.15.1 Added

- Argument Parser for simple programs where a configuration file would be unneeded overhead for the user.
- Numpy Data support for single arrays and pass fail files.
- Data Plugin now has two array types, Single Array and Columned Array
- Memory and Iterator objects now imported into PyPWA
- Iterators report length now
- Masking utility `PyMask` to mask and translate data

### 3.15.2 Changed

- Plugin Loader now returns initialized objects
- Renamed `shell` to `progs`
- Moved all shell related items into a package called shell inside progs
- Renamed `CHANGELOG.mg` to `CHANGELOG.md`
- Renamed 'blank shell module' to 'blank program module'
- Removed support for boolean and float arrays from EVIL Parser
- Renamed internal GAMP type to Tree type
- Split flat data into Columned data and standard arrays

### 3.15.3 Fixed

- ChiSquare and Empty likelihoods are now actually usable
- `setup.py` would fail on `setuptools` versions < 20

## 3.16 2.0.0 - 2017-6-5

### 3.16.1 Added

- Plugin Subsystem
- Configurator Subsystem
- Data Plugin
- SV Plugin
- EVIL Plugin
- GAMP Plugin
- Data Caching
- Processing Plugin
- iMinuit plugin
- Nestle likelihood
- PyFit plugin
- Log Likelihood Plugin
- Chi-Squared Likelihood
- PySim plugin

- Packaging

CHAPTER

# FOUR

# INTRO TO PYPWA WITH THE 2D GAUSS

The goal with this little tutorial is to walk through how those PyPWA and its collective features.

**Note:** Multiproccessing is done automatically when it's selected. However, if you have some direct C/C++ code dependency in your Function on called in your class's `__init__`, you will encounter issues. This is why each object has a `setup` function- To initialize Fortran and C++ dependencies there.

```
[1]: import numpy as np  # Vectorization and arrays
     import pandas as pd  # A powerful data science toolkit
     import numexpr as ne  # A threaded accelerator for numpy

     import PyPWA as pwa
     from IPython.display import display
```

```
/home/mark/.anaconda3_install/envs/PyPWA/lib/python3.10/site-packages/tqdm/auto.py:
↪22: TqdmWarning: IProgress not found. Please update jupyter and ipywidgets. See␣
↪https://ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
```

There are 3 different supported ways to define your kernel/amplitude in PyPWA. - Using multiprocessing: Write your kernel using Numpy and include any externally compiled code the setup method. This is the default kernel, and will result in your kernel being deployed in parallel across all threads on the host system. - Using Numexpr to use hardware threads and low level vectorization to further accelerate Numpy. There is some benefit to running Numexpr on less cores than traditional Numpy, but largely you can treat Numexpr the same as the above. - Using Torch to compute the Kernel. This will allow you to take advantage of Metal on Apple PCs, or CUDA GPUs on Linux machines. However, to utilize CUDA, you must disable Multiprocessing. At this time, CUDA does not support the main process being forked.

```
[2]: class Gauss2dAmplitude(pwa.NestedFunction):
         """
         This is a simple 2D Gauss, but built to use PyPWA's builtin
         multiprocessing module. For you, you don't need to worry about thread or
         process management, how to pass data between threads, or any of the other
         hassles that come with multithreading.

```

(continues on next page)

(continued from previous page)

```
    Instead, you just define your class while extending the NestedFunction,
    and when you pass it to the fitter or the simulator, it'll clone your
    class, split your data, and deploy to every processing thread your
    machine has.
    """

    def __init__(self):
        """
        You can override the init function if you need to set parameters
        before the amplitude is passed to the likelihood or simulation
        functions. You can see an example of this with the JPAC amplitude
        included in the other tutorials. However, you must remember
        to always call the `super` function if you do this.
        """
        super(Gauss2dAmplitude, self).__init__()

    def setup(self, array):
        """
        This function is where your data is passes too. Here you can also
        load any C or Fortran external libraries that typically would not
        support being packaged in Python's object pickles.
        """
        self.__x = array["x"]
        self.__y = array["y"]

    def calculate(self, params):
        """
        This function receives the parameters from the minimizer, and
        returns the values from there. Only the amplitude values should
        be calculated here. The likelihood will be calculated elsewhere.
        """
        scaling = 1 / (params["A2"] * params["A4"])
        left = ((self.__x - params["A1"])**2)/(params["A2"]**2)
        right = ((self.__y - params["A3"])**2)/(params["A4"]**2)
        return scaling * np.exp(-(left + right))
```

```
[3]: class NeGauss2dAmplitude(pwa.NestedFunction):
    """
    This is the same Gauss as above, but instead of using raw numpy, it
    uses numexpr, a hyper vectorized, multithreading, numerical package that
    should accelerate the calculation of your data.

    USE_MP defaults to True, but you should consider setting it to false.
    Numexpr will do some partial multithreading on its own for its
    calculations, however any part of your algorithm that is defined outside
```

(continues on next page)

(continued from previous page)

```
    Numexpr will not benefit from Numexpr. Due to this, there is an optimum
    number of threads for amplitudes with Numexpr that range from 2 threads
    to around 80% of the system threads. A good starting point is around
    50% of the CPU threads available.
    """

    USE_MP = False

    def setup(self, array):
        self.__data = array

    def calculate(self, params):
        return ne.evaluate(
            "(1/(a2*a4)) * exp(-((((x-a1)**2)/(a2**2))+(((y-a3)**2)/(a4**2))))",
            local_dict={
                "a1": params["A1"], "a2": params["A2"],
                "a3": params["A3"], "a4": params["A4"],
                "x": self.__data["x"], "y": self.__data["y"]
            }
        )
```

[4]:
```
import torch as tc

class TorchGauss2dAmplitude(pwa.NestedFunction):
    """
    Finally, this is the Torch version of the Gauss2D.

    To utilize Torch, the USE_TORCH flag must be set to True, or the
    likelihood will assume that the results will be in standard Numpy
    arrays, and not Torch Tensors.

    Torch affords us some features that Numpy does not. Specifically,
    support for both Apple's Metal Acceleration, and CUDA Acceleration.
    For Apple's Metal, there is no work required further than defining
    your amplitude in Torch due to the shared memory on Apple systems.
    To utilize CUDA, however, you must move the data to the GPU before
    the GPU can accelerate the operations. Because of the nature of
    CUDA, CUDA and Multiprocessing are not compatible, so you must
    disable multiprocessing when using CUDA.

    **WARNING** You **MUST** set USE_MP to False if you are using
    CUDA as a Torch Device!
    """

    USE_TORCH = True
```

(continues on next page)

(continued from previous page)

```
    USE_MP = False

    # device is not a flag for Amplitude, but we use it track the current
    # device that the amplitude should run on.
    device = ...  # type: tc.device

    def setup(self, array):
        # We want to always set the current device. It also helps to be able
        # to toggle GPU on and off for the entire amplitude using the USE_MP,
        # flag since the flag can be set after initialization.
        if self.USE_MP:
            self.device = tc.device("cpu")
        else:
            self.device = tc.device("cuda:0")

        # Since the data is in Pandas, we need to map it to Numpy first
        narray = pwa.pandas_to_numpy(array)

        self.__x = tc.from_numpy(narray["x"]).to(self.device)
        self.__y = tc.from_numpy(narray["y"]).to(self.device)

    def calculate(self, params):
        scaling = 1 / (params["A2"] * params["A4"])
        left = ((self.__x - params["A1"])**2)/(params["A2"]**2)
        right = ((self.__y - params["A3"])**2)/(params["A4"]**2)
        return scaling * tc.exp(-(left + right))
```

## 4.1 Using caching for intermediate steps

PyPWA's caching module supports caching intermediate steps. The advantage of using the caching module is that saving and loading values is fast; much faster than almost any other solution, and supports almost anything you can store in a variable.

PyPWA.cache has two functions in it, `read` and `write`. You can save almost anything in the cache: lists, DataFrames, dictionary's, etc. There is a chance that it won't save the value if the data isn't serializable into a pickle, and it may not be compatible between different versions of python, so I don't recommend using this for data that you can't reproduce. However, if you need to do some feature engineering, or data sanitizing, before you can use the data in whatever way you need and want to keep that data around to speed up future executions, this module will make your life a touch easier.

Below, I created a large flat DataFrame, and then binned that DataFrame into 10 bins, each with 1,000,000 events in them. Then, I saved those results into a cache object that will appear in the current working directory with the name "flat_data.intermediate".

- `pwa.read` returns two values, the first is a boolean that is True only if it was able to read the pickle, and the second is the parsed data, which will be None if it was unable to parse anything from the file, or the file doesn't exist.

- `pwa.write` has no returns, but does write the data out in Pickle format to the provided filename + the ".intermediate" extension.

[5]:
```
valid_cache, binned_flat = pwa.cache.read("flat_data")
if not valid_cache:
    flat_data = pd.DataFrame()
    flat_data["x"] = np.random.rand(10_000_000) * 20
    flat_data["y"] = np.random.rand(10_000_000) * 20
    flat_data["binning"] = np.random.rand(10_000_000) * 20
    binned_flat = pwa.bin_with_fixed_widths(flat_data, "binning", 1_000_000)
    pwa.cache.write("flat_data", binned_flat)
```

## 4.2 Simulation with bins

Simulation can be run as a whole system, you simply provide the function and data, and it'll return the masked values, or you can run the two steps independently, with the first step returning the intensities, and the second returning the masks. When your working with a single dataset, running it as a single step make sense, however if you bin your data, then running it as two steps is better so that all bins are masked against the same max value of the intensity.

- `pwa.simulate.process_user_function` takes all the same arguments as `pwa.monte_carlo_simulation` so it can be a drop in replacement. The difference is that this function will return the final values for the user's function and the max value.
- `pwa.simulate.make_rejection_list` takes the final values and either a single max value, or a list or array of max values, and it'll use the largest max value. This function will return the same value as `pwa.monte_carlo_simulation`

Below, I iterate over the bins and produce the final values and max value for each bin and store them in their own lists.

[6]:
```
simulation_params = {
    "A1": 10, "A2": 3,
    "A3": 10, "A4": 3
}

final_values = []
max_values = []
for fixed_bin in binned_flat:
    final, m = pwa.simulate.process_user_function(
        TorchGauss2dAmplitude(), fixed_bin, simulation_params
    )
    final_values.append(final)
    max_values.append(m)
```

(continues on next page)

(continued from previous page)

```
pwa.cache.write("final_values", max_values)
```

After the final values have been produced, I use `pwa.simulate.make_rejection_list` to reject events from each bin, and then store the new carved results in a fresh list.

[7]:
```
rejected_bins = []
masked_final_values = []
for final_value, bin_data in zip(final_values, binned_flat):
    rejection = pwa.simulate.make_rejection_list(final_value, max_values)
    rejected_bins.append(bin_data[rejection])
    masked_final_values.append(final_value[rejection])

pwa.cache.write("fitting_bins", rejected_bins, True)
pwa.cache.write("kept_final_values", masked_final_values, True)
```

[8]:
```
for index, simulated_bin in enumerate(rejected_bins):
    print(
        f"Bin {index+1}'s length is {len(simulated_bin)}, "
        f"{(len(simulated_bin) / 1_000_000) * 100:.2f}% events were kept"
    )
```

```
Bin 1's length is 70589, 7.06% events were kept
Bin 2's length is 70544, 7.05% events were kept
Bin 3's length is 70621, 7.06% events were kept
Bin 4's length is 70608, 7.06% events were kept
Bin 5's length is 70532, 7.05% events were kept
Bin 6's length is 70369, 7.04% events were kept
Bin 7's length is 71019, 7.10% events were kept
Bin 8's length is 70542, 7.05% events were kept
Bin 9's length is 70533, 7.05% events were kept
Bin 10's length is 70633, 7.06% events were kept
```

## 4.3 How Caching is used by the Read and Write functions

If you want your data to be parsable by standard libraries, but still want to leverage the speed of caching, you can use both, by default even! When the Cache Module is used by the Data Module, it utilizes an additional feature that is tucked away when used by itself: File Hashing. The Cache module can be told when it's caching a specific file, so before the cache is created, it will parse the source file to determine it's SHA512 Sum, and then store that inside the cache. When the file is loaded next, the saved SHA512 Sum is compared to the file's current sum, and if they match the cache is returned, otherwise the file is parsed again, and the cache is recreated.

After the below cell runs, you'll see two new files created: `first_bin.csv` and `first_bin.cache`. These two files will contain the same data, but if the CSV file is changed, even if just by a single character, the file will be parsed again on the next call of `pwa.read`

[9]:
```python
try:
    first_bin = pwa.read("first_bin.csv")
except Exception:
    first_bin = binned_flat[0]
    pwa.write("first_bin.csv", binned_flat[0])
```

## 4.4 Fitting

While you can use PyPWA's likelihoods with any minimizer, PyPWA supports Iminuit 2.X out of the box. The first thing that is done is we set up the parameters to fit against, as well as the individual names of each parameter.

Traditionally, iminuit works by reading the values from the provided function to guess what the parameters are and what to pass to the function, however since we wrap the minimized function to take advantage of GPU acceleration and multiprocessing, you must also tell iMinuit what the values are directly.

[10]:
```python
fitting_settings = {
    "A1": 1, "A2": 1,
    "A3": 1, "A4": 1,
}
```

Then below, we can simply fit those values.

[11]:
```python
import multiprocessing as mp
# Even though we're using Numexpr, I do want to take advantage of both
# multiprocessing and Numexpr's low level optimizations. So by selecting
# a small number of processes with Numexpr, you still get an overall
# speedup over either Numexpr or regular multiprocessing
NeGauss2dAmplitude.USE_MP = True

cpu_final_values= []
for simulated_bin in rejected_bins:
    with pwa.LogLikelihood(
            NeGauss2dAmplitude(), simulated_bin,
            num_of_processes=int(mp.cpu_count() / 2)
    ) as likelihood:
        optimizer = pwa.minuit(fitting_settings, likelihood)

        for param in ["A1", "A3"]:
            optimizer.limits[param] = (.1, None)

        for param in ["A2", "A4"]:
            optimizer.limits[param] = (1, None)

        cpu_final_values.append(optimizer.migrad())
```

[12]:
```python
gpu_final_values = []
for simulated_bin in rejected_bins:
    with pwa.LogLikelihood(TorchGauss2dAmplitude(), simulated_bin) as likelihood:
        optimizer = pwa.minuit(fitting_settings, likelihood)

        for param in ["A1", "A3"]:
            optimizer.limits[param] = (.1, None)

        for param in ["A2", "A4"]:
            optimizer.limits[param] = (1, None)

        gpu_final_values.append(optimizer.migrad())
```

## 4.5 A note about `with`

If you are new to Python, the with statement might be new to you. `with` allows you to create objects that should be closed. Traditionally, you will see `with` used with files, but we use this with Likelihoods. In the file case, when you leave the `with` block it will flush the buffers for you and close the file's handle. In the case of Likelihoods, when you leave the `with` block it will shut down any associated threads, processes, and pipes that are associated with the created Likelihood.

Below is an example of how the Likelihood works without using the `with` statement.

[13]:
```python
for simulated_bin in rejected_bins:
    likeihood = pwa.LogLikelihood(TorchGauss2dAmplitude(), simulated_bin)
    optimizer = pwa.minuit(fitting_settings, likelihood)

    for param in ["A1", "A3"]:
        optimizer.limits[param] = (.1, None)

    for param in ["A2", "A4"]:
        optimizer.limits[param] = (1, None)

    # You must remember to close the Likelihood when not using the 'with' block!
    likeihood.close()
```

## 4.6 Issues with PyPWA.cache

There are some values that can not be saved in a PyPWA's cache. Typically, it's an object from a package that takes advantage of Cython or Fortran to accelerate its execution: either because the values are stored as pointers to arrays, or uses C types too deep for Python's interpreter to analyze. A good example of this case is results from Iminuit.

As you can see below, a Runtime Warning is thrown from PyPWA's caching module about how the data can't be saved. However, the real error is that the tuple has values that can not be converted to a pure Python object for pickling.

```
[14]: try:
          pwa.cache.write("fitting_results", cpu_final_values, True)
      except RuntimeWarning as error:
          print("Caught a cache error")
          print(f"{type(error)}: {error}")
```

```
Caught a cache error
<class 'RuntimeWarning'>: Your data can not be saved in cache!
```

## 4.7 Viewing the results

Finally, we can see what the results of the fitting. The result objects from iminuit are actually Jupyter aware, so if you view a result from iminuit in Jupyter, the values will be responsive.

If you want to know what methods and parameters are available the result object returned by iminuit, you should read through their (documentation)[https://iminuit.readthedocs.io/]

```
[15]: print(f"CPU bin 1")
      cpu_final_values[0]
```

```
CPU bin 1
```

```
[15]: ────────────────────────────────────────────────────────────────────────
      │                              Migrad                                  │
      ├───────────────────────────────────┬──────────────────────────────────┤
      │ FCN = 2.259e+05                   │             Nfcn = 231           │
      │ EDM = 5.29e-07 (Goal: 0.0001)     │                                  │
      ├───────────────────────────────────┼──────────────────────────────────┤
      │          Valid Minimum            │      No Parameters at limit      │
      ├───────────────────────────────────┼──────────────────────────────────┤
      │ Below EDM threshold (goal x 10)   │         Below call limit         │
      ├───────────────┬───────────────────┼───────────┬─────────────┬────────┤
      │  Covariance   │     Hesse ok      │ Accurate  │  Pos. def.  │ Not forced │
      └───────────────┴───────────────────┴───────────┴─────────────┴────────┘
      ─────────────────────────────────────────────────────────────────────────────────
      │   │ Name │    Value   │ Hesse Err │ Minos Err- │ Minos Err+ │ Limit-  │ Limit+  │␣
      ↪Fixed │
      ├───┼──────┼────────────┼───────────┼────────────┼────────────┼─────────┼─────────┼
      │ 0 │ A1   │   10.005   │   0.008   │            │            │   0.1   │         │ ␣
      ↪     │
      │ 1 │ A2   │   3.003    │   0.008   │            │            │    1    │         │ ␣
      ↪     │
      │ 2 │ A3   │   9.995    │   0.008   │            │            │   0.1   │         │ ␣
      ↪     │
      │ 3 │ A4   │   3.006    │   0.008   │            │            │    1    │         │ ␣
      ↪     │
```

(continues on next page)

(continued from previous page)

```
└──────────────────────────────────────────────────────────────────────────────────

──────────────────────────────────────────────
│     │         A1         A2         A3         A4 │
├─────┼─────────────────────────────────────────────
│  A1 │   6.39e-05   5.59e-08   8.65e-14   8.61e-14 │
│  A2 │   5.59e-08   6.39e-05   6.89e-14  -3.45e-13 │
│  A3 │   8.65e-14   6.89e-14    6.4e-05   5.57e-08 │
│  A4 │   8.61e-14  -3.45e-13   5.57e-08    6.4e-05 │
└─────┴─────────────────────────────────────────────
```

```
[16]: print(f"GPU bin 1")
      display(gpu_final_values[0])
```

```
GPU bin 1
```

```
┌──────────────────────────────────────────────────────────────────────────┐
│                                Migrad                                    │
├──────────────────────────────────┬───────────────────────────────────────┤
│ FCN = 2.259e+05                  │              Nfcn = 231               │
│ EDM = 5.29e-07 (Goal: 0.0001)    │                                       │
├──────────────────────────────────┼───────────────────────────────────────┤
│          Valid Minimum           │        No Parameters at limit         │
├──────────────────────────────────┼───────────────────────────────────────┤
│ Below EDM threshold (goal x 10)  │           Below call limit            │
├───────────────┬──────────────────┼───────────┬─────────────┬─────────────┤
│  Covariance   │     Hesse ok     │ Accurate  │  Pos. def.  │ Not forced  │
└───────────────┴──────────────────┴───────────┴─────────────┴─────────────┘
┌───┬──────┬───────────┬───────────┬────────────┬────────────┬─────────┬─────────┬␣
↪Fixed ┐
│   │ Name │   Value   │ Hesse Err │ Minos Err- │ Minos Err+ │ Limit-  │ Limit+  │␣
↪      │
├───┼──────┼───────────┼───────────┼────────────┼────────────┼─────────┼─────────┼
│ 0 │ A1   │  10.005   │   0.008   │            │            │   0.1   │         │␣
↪      │
│ 1 │ A2   │   3.003   │   0.008   │            │            │    1    │         │␣
↪      │
│ 2 │ A3   │   9.995   │   0.008   │            │            │   0.1   │         │␣
↪      │
│ 3 │ A4   │   3.006   │   0.008   │            │            │    1    │         │␣
↪      │
└───┴──────┴───────────┴───────────┴────────────┴────────────┴─────────┴─────────┴

┌─────┬─────────────────────────────────────────┐
│     │        A1        A2        A3        A4 │
├─────┼─────────────────────────────────────────┤
│  A1 │  6.39e-05  5.59e-08         0        -0 │
│  A2 │  5.59e-08  6.39e-05        -0         0 │
│  A3 │         0        -0   6.4e-05  5.57e-08 │
```

(continues on next page)

(continued from previous page)

```
│ A4 │       -0        0 5.57e-08  6.4e-05 │
└─────────────────────────────────────────┘
```

CHAPTER

FIVE

# SIMULATION TUTORIAL

```
[1]: import PyPWA as pwa
     import numpy as npy
     import pandas
     import matplotlib.pyplot as plt
     import warnings
     warnings.filterwarnings('ignore')
```

## 5.1 *Define (import) amplitude (function) to simulate*

The function will be use by the rejection method to "carve" a new distribution into the input simulated data read in next lines.

```
[2]: #
     import AmplitudeJPACsim
     amp = AmplitudeJPACsim.NewAmplitude()
```

## 5.2 *Read input (flat) simulated data (in condense format)*

```
[3]: data = pwa.read("etapiHEL2_flat.txt")
```

Read data full (from gamp files)

```
[4]: datag = pwa.read("../TUTORIAL_FILES/raw_events.gamp")
```

The format of the input data will depend on the Amplitudes: In this example the standard HEL angles, polarization angle (alpha) and other information neccesary are given for PWA (see below)

```
[5]: data
```

```
[5]:             EventN     theta       phi     alpha  pol        tM       mass
     0              0.0  1.902160  3.800470 -1.074900  0.4 -0.026354  0.850234
```

(continues on next page)

(continued from previous page)

```
1              1.0  0.916137  1.660230 -1.438370  0.4 -0.671785  2.619530
2              2.0  1.811330  5.778960 -2.341440  0.4 -0.077582  1.268280
3              3.0  2.000940  5.250000  2.709120  0.4 -0.726730  1.253960
4              4.0  1.871130  2.697150  0.239302  0.4 -0.156830  0.927206
...              ...       ...       ...       ...  ...   ...       ...        ...
9855141  9855141.0  1.354170  5.931830  0.784408  0.4 -0.054668  1.083350
9855142  9855142.0  2.335000  3.347330 -0.748025  0.4 -0.844509  1.483240
9855143  9855143.0  1.328470  1.948730 -2.102880  0.4 -0.138508  0.940464
9855144  9855144.0  2.184290  2.491810 -1.821200  0.4 -0.314037  0.723398
9855145  9855145.0  1.570570  0.590933 -1.335300  0.4 -0.240994  1.187410

[9855146 rows x 7 columns]
```

## 5.3 *Produce simulation mask*

A boolean file of (False and True)

[6]:
```
rejection = pwa.monte_carlo_simulation(amp, data, dict(), 16)
```

Check on the waves and resonances that will be produced > This is a matrix with the weigths of each wave on each resonance

[7]:
```
amp.setup(data)
table =[]
tabler=[]

for r in range(amp.resonance.resonance_index):
    tabler.append(amp.resonance.W0[r])
    tabler.append(amp.resonance.Cr[r])
    for w in range(amp.resonance.wave_index):

        tabler.append(amp.resonance.Wave[r][w])
    table.append(tabler)
    tabler=[]

from tabulate import tabulate
headers=["Resonance","Res-weight"]
for w in range(amp.resonance.wave_index):
    headers.append(amp.resonance.wave_data[w])

print(tabulate(table,headers))
```

```
  Resonance     Res-weight    (1, 0, 0)    (1, 1, 0)    (1, 1, -1)    (1, 1, 1)    (1,
↪ 2, 0)    (1, 2, -1)    (1, 2, 1)    (1, 2, -2)    (1, 2, 2)
-----------  ------------  -----------  -----------  ------------  -----------  -----
↪------    ------------  -----------  ------------  -----------
```

(continues on next page)

(continued from previous page)

```
      0.98            0.65            1          0               0          0          ␣
↪  0                 0          0               0          0
      1.306           0.35            0          0               0          0          ␣
↪  0.33              0          0.33            0          0.33
      1.584           0.08            0          0.5             0          0.5        ␣
↪  0                 0          0               0          0
      1.722           0.1             0          0               0          0          ␣
↪  0.33              0          0.33            0          0.33
```

*Check on how many events will be kept through the masking*

[8]:
```
print(f"Removed {len(data) - npy.sum(rejection)} events from flat data.")
print(f"Kept {npy.sum(rejection)} events.")
print(f"{(npy.sum(rejection) / len(data)) * 100}% of events remain.")
```

```
Removed 9454013 events from flat data.
Kept 401133 events.
4.070289775514234% of events remain.
```

## 5.4 Apply mask to input data

new_data will contain the simulated data in the same format that data/datag

[9]:
```
new_data = data[rejection]
```

[10]:
```
new_data
```

[10]:
```
            EventN     theta       phi     alpha  pol        tM      mass
137          137.0  2.206940  1.953870 -0.247381  0.4 -0.120428  0.995635
171          171.0  1.775060  1.264290  2.600410  0.4 -0.345176  1.167540
200          200.0  0.128817  2.396510 -0.063606  0.4 -0.229984  2.269060
205          205.0  0.696665  1.981680  3.079090  0.4 -0.184068  1.885680
237          237.0  1.051710  1.753500 -2.496390  0.4 -1.500240  1.334010
...            ...       ...       ...       ...  ...   ...          ...       ...
9855028  9855028.0  0.858080  2.779910 -0.051326  0.4 -0.086823  0.981714
9855041  9855041.0  1.620860  0.822745  1.592530  0.4 -0.451578  1.311310
9855064  9855064.0  0.734074  5.887190 -1.768290  0.4 -0.099859  1.098970
9855075  9855075.0  1.502800  1.512360  0.860545  0.4 -0.162814  1.710660
9855113  9855113.0  1.324710  4.971240 -1.140430  0.4 -0.051079  1.154320

[401133 rows x 7 columns]
```

*Mask full data format*

[11]:
```
new_datag = datag[rejection]
```

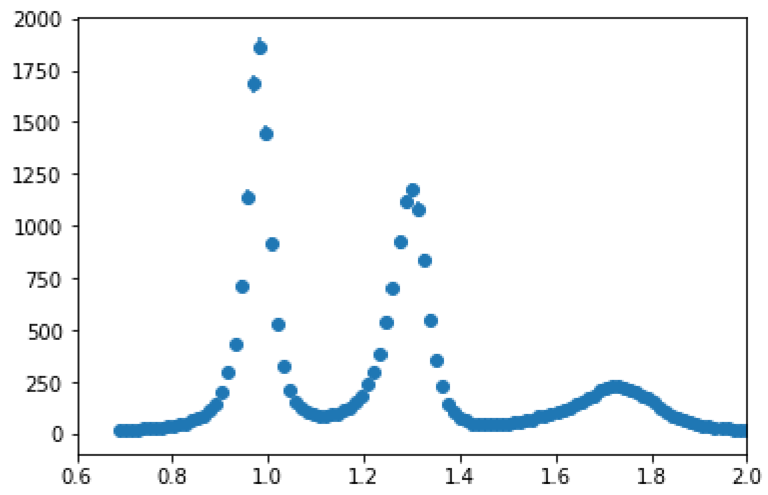
*Write new_data into a Pandas Dateframe*

```
[12]: amp.setup(new_data)
      new_data = pandas.DataFrame(new_data)
```

## 5.5 Plot simulated data intensity versus mass

```
[13]: results = amp.calculate()
      mni = npy.empty(len(new_data), dtype=[("mass", float), ("intensity", float)])
      mni["mass"] = new_data["mass"]
      mni["intensity"] = results.real
      mni = pandas.DataFrame(mni)
      counts, bin_edges = npy.histogram(mni["mass"], 200, weights=mni["intensity"])
      centers = (bin_edges[:-1] + bin_edges[1:]) / 2

      # Add yerr to argment list when we have errors
      yerr = npy.empty(100)
      yerr = npy.sqrt(counts)
      plt.errorbar(centers,counts, yerr, fmt="o")
      #plt.yscale("log")
      plt.xlim(.6, 2.)
```

```
[13]: (0.6, 2.0)
```



## 5.6 Calculate Phase difference between two waves

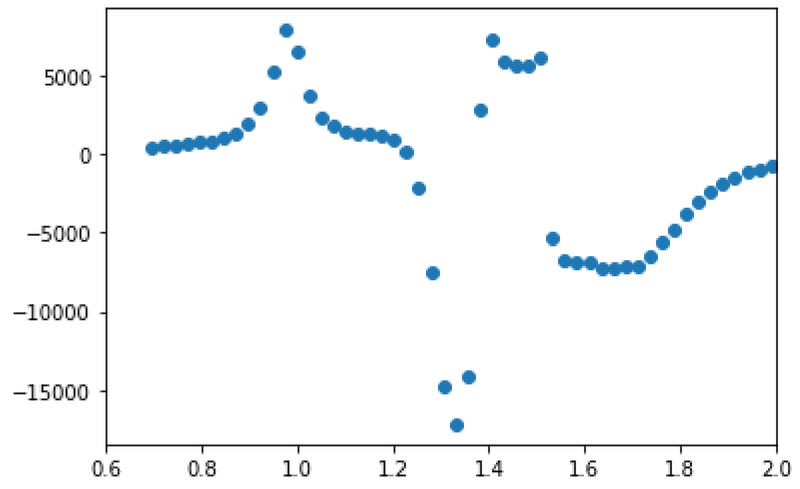
In this example first and 3erd waves in amplitude list

```
[14]: #if amp.Vs[1]15].all() != 0 and amp.Vs[0][0].all() != 0:
phasediff = npy.arctan(npy.imag(amp.Vs[2][3]*amp.Vs[1][6].conjugate())/npy.real(amp.
↪Vs[2][3]*amp.Vs[1][6].conjugate()))
#phasediff = npy.arctan(npy.imag(amp.Vs[2][1]*amp.Vs[1][2].conjugate())/npy.real(amp.
↪Vs[2][1]*amp.Vs[1][2].conjugate()))
```

*Plot PhaseMotion*

```
[15]: mnip = npy.empty(len(new_data), dtype=[("mass", float), ("phase", float)])
mnip["mass"] = new_data["mass"]
mnip["phase"] = phasediff
mnip = pandas.DataFrame(mnip)
counts, bin_edges = npy.histogram(mnip["mass"], 100, weights=mnip["phase"])
centers = (bin_edges[:-1] + bin_edges[1:]) / 2

# Add yerr to argment list when we have errors
yerr = npy.empty(100)
yerr = npy.sqrt(counts)
plt.errorbar(centers,counts, yerr, fmt="o")
plt.xlim(0.6, 2.)
```

```
[15]: (0.6, 2.0)
```



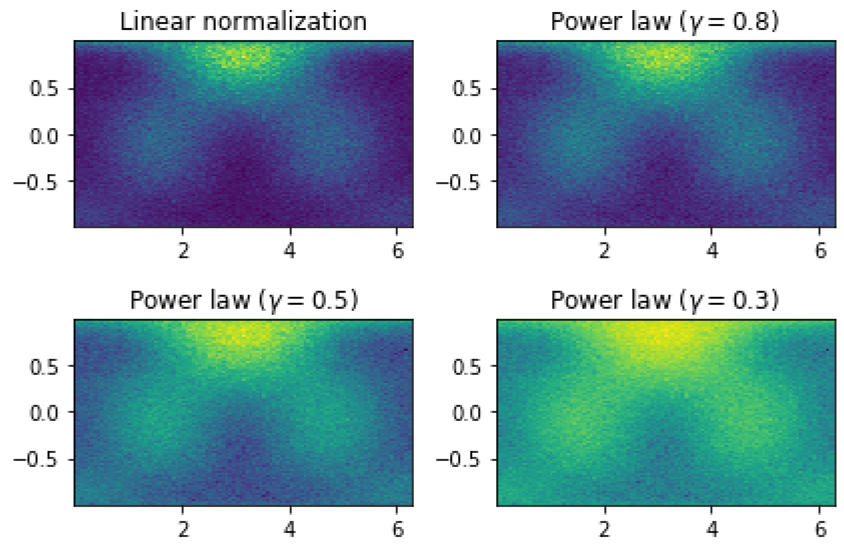
*Plot phi_HEL vs cosHEL) of simulated data (with 4 different contracts(gamma))*

```
[16]: import matplotlib.colors as mcolors
from numpy.random import multivariate_normal

gammas = [0.8, 0.5, 0.3]
```

(continues on next page)

(continued from previous page)

```
fig, axes = plt.subplots(nrows=2, ncols=2)

axes[0, 0].set_title('Linear normalization')
axes[0, 0].hist2d(new_data["phi"], npy.cos(new_data["theta"]), bins=100)
#axes[0, 0].hist2d(cut_list["phi"], npy.cos(cut_list["theta"]), bins=100)

for ax, gamma in zip(axes.flat[1:], gammas):
    ax.set_title(r'Power law $(\gamma=%1.1f)$' % gamma)
    ax.hist2d(new_data["phi"], npy.cos(new_data["theta"]),
        bins=100, norm=mcolors.PowerNorm(gamma))

fig.tight_layout()

plt.show()
```



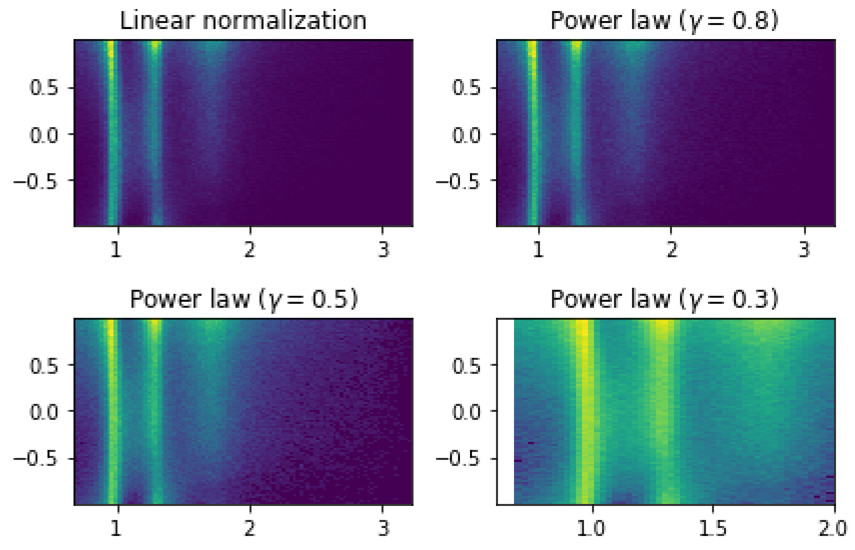
*Plot cos(theta_HEL) vs mass for simulated data (with 4 different contrasts)*

```
[17]: gammas = [0.8, 0.5, 0.3]

fig, axes = plt.subplots(nrows=2, ncols=2)

axes[0, 0].set_title('Linear normalization')
axes[0, 0].hist2d(new_data["mass"], npy.cos(new_data["theta"]),bins=100)

for ax, gamma in zip(axes.flat[1:], gammas):
    ax.set_title(r'Power law $(\gamma=%1.1f)$' % gamma)
    ax.hist2d(new_data["mass"], npy.cos(new_data["theta"]),
              bins=100, norm=mcolors.PowerNorm(gamma))
```

(continues on next page)

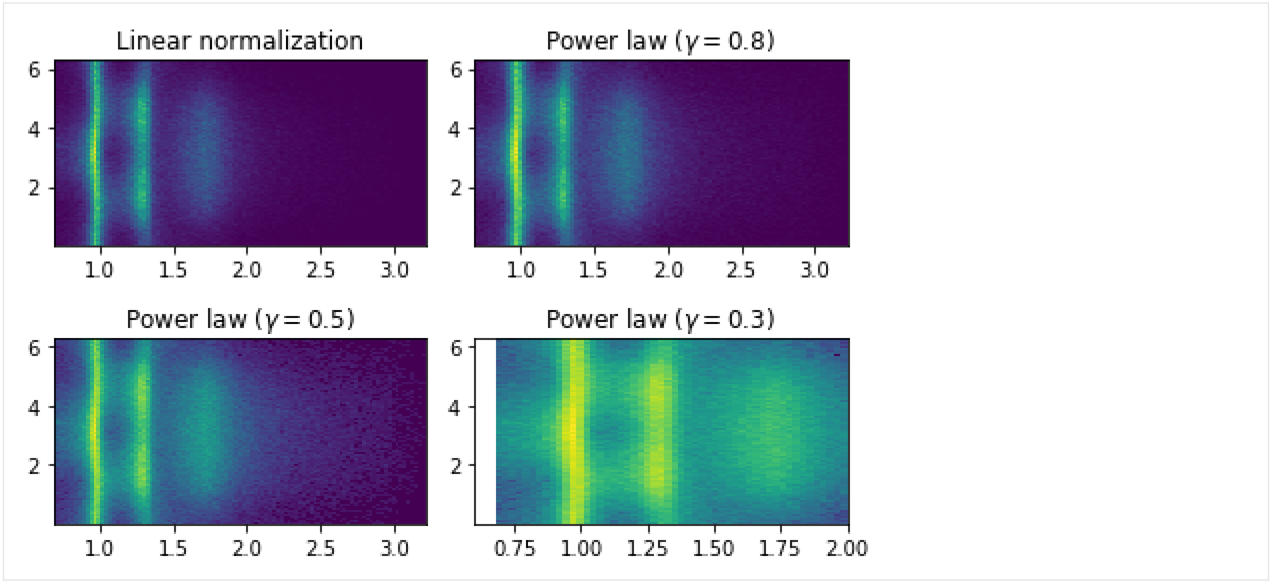
(continued from previous page)

```
fig.tight_layout()
plt.xlim(.6, 2.)
plt.show()
```



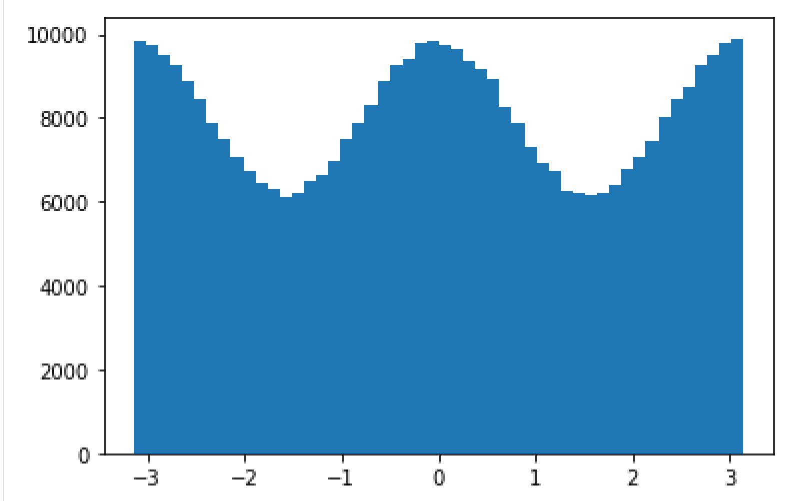
*Plot phiHEL vs mass for simulated data (with 4 different contrasts)*

```
[18]: gammas = [0.8, 0.5, 0.3]

fig, axes = plt.subplots(nrows=2, ncols=2)

axes[0, 0].set_title('Linear normalization')
axes[0, 0].hist2d(new_data["mass"], new_data["phi"],bins=100)

for ax, gamma in zip(axes.flat[1:], gammas):
    ax.set_title(r'Power law $(\gamma=%1.1f)$' % gamma)
    ax.hist2d(new_data["mass"], new_data["phi"],
              bins=100, norm=mcolors.PowerNorm(gamma))


fig.tight_layout()
plt.xlim(.6, 2.)
plt.show()
```



*Histogram of alpha/Phi* > alpha/Phi is the polarization angle

```
[19]: plt.hist(new_data["alpha"],50)
plt.show()
```



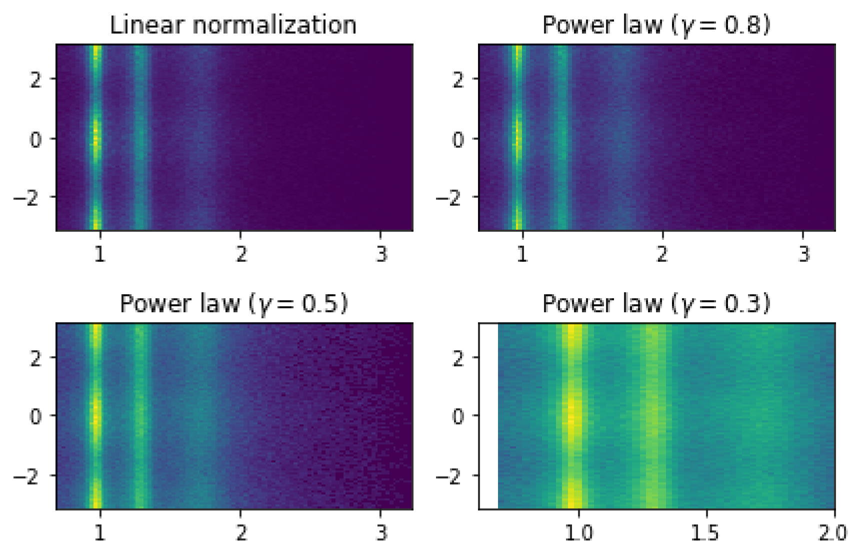
*Plot mass versus alpha/Phi*

```
[20]: gammas = [0.8, 0.5, 0.3]

fig, axes = plt.subplots(nrows=2, ncols=2)

axes[0, 0].set_title('Linear normalization')
axes[0, 0].hist2d(new_data["mass"], new_data["alpha"],bins=100)

for ax, gamma in zip(axes.flat[1:], gammas):
    ax.set_title(r'Power law $(\gamma=%1.1f)$' % gamma)
```

(continues on next page)

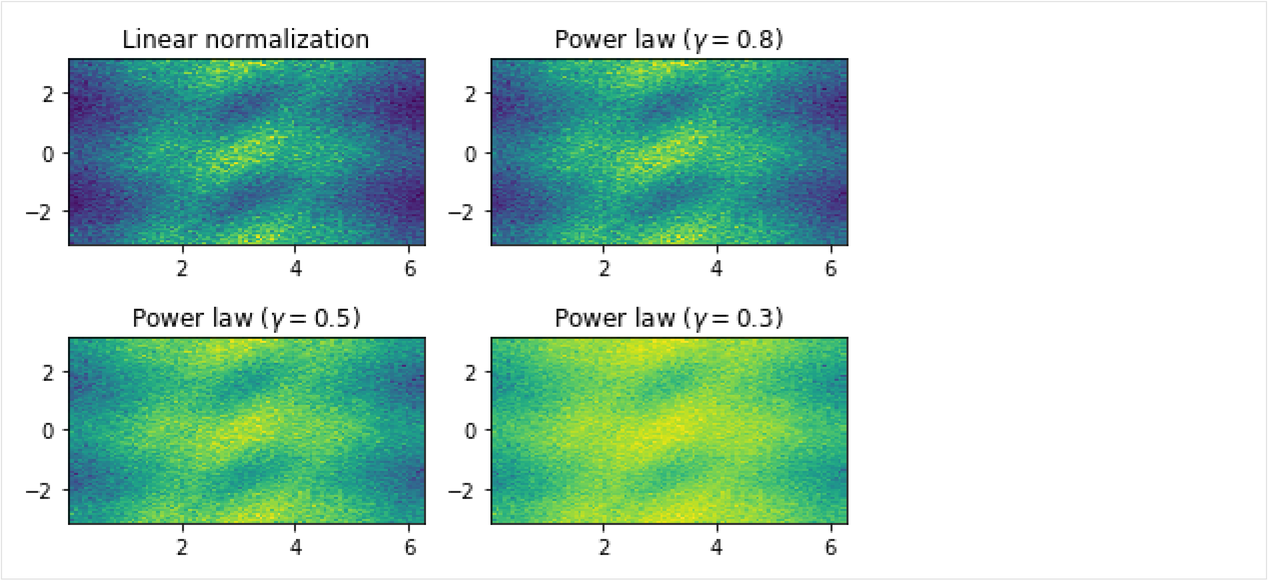
(continued from previous page)

```
    ax.hist2d(new_data["mass"], new_data["alpha"],
              bins=100, norm=mcolors.PowerNorm(gamma))


fig.tight_layout()
plt.xlim(.6, 2.)
plt.show()
```



*Plot phi_HEL versus alpha/Phi*

```
[21]: gammas = [0.8, 0.5, 0.3]

fig, axes = plt.subplots(nrows=2, ncols=2)

axes[0, 0].set_title('Linear normalization')
axes[0, 0].hist2d(new_data["phi"],new_data["alpha"],bins=100)

for ax, gamma in zip(axes.flat[1:], gammas):
    ax.set_title(r'Power law $(\gamma=%1.1f) $' % gamma)
    ax.hist2d(new_data["phi"], new_data["alpha"],
              bins=100, norm=mcolors.PowerNorm(gamma))


fig.tight_layout()

plt.show()
```



## 5.7 *Write simulated data to disk*

```
[23]: new_data.to_csv("simdata_JPAC.csv", index=False)
```

Write gamp data after maked

```
[24]: pwa.write("raw_simulated_JPAC.gamp",new_datag)
```

## 5.8 *Calculate Moments (for JPAC or Std) and Asymmetries*

```
[25]: H000,H010,H011,H020,H021,H022,H100,H110,H111,H120,H121,H122,sigma4,sigmay = amp.
      ↪calculate_moments_JPAC()
      #H00,H11,H10,H20,H21,H22 = amp.calculate_moments_STD()
```

*Plot (all) Moments versus mass*

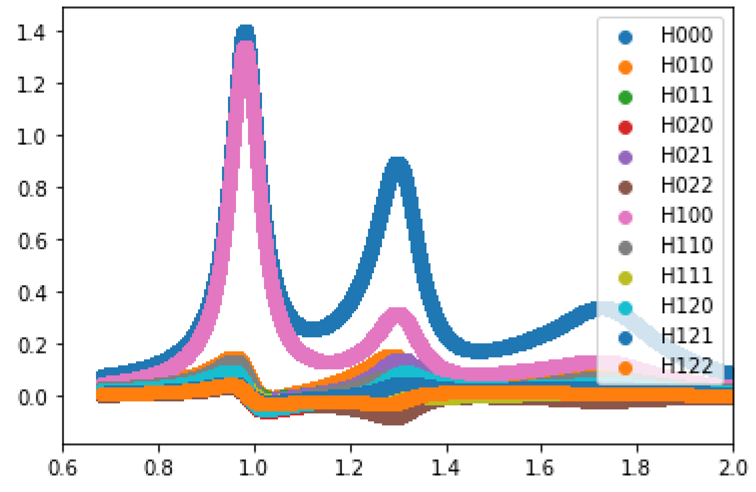
```
[26]: plt.scatter(new_data["mass"],H000,LABEL="H000")
      plt.legend(loc='upper right')
      plt.scatter(new_data["mass"],H010,LABEL="H010")
      plt.legend(loc='upper right')
      plt.scatter(new_data["mass"],H011,LABEL="H011")
      plt.legend(loc='upper right')
      plt.scatter(new_data["mass"],H020,LABEL="H020")
      plt.legend(loc='upper right')
      plt.scatter(new_data["mass"],H021,LABEL="H021")
      plt.legend(loc='upper right')
```

(continues on next page)

(continued from previous page)

```
plt.scatter(new_data["mass"],H022,LABEL="H022")
plt.legend(loc='upper right')
plt.scatter(new_data["mass"],H100,LABEL="H100")
plt.legend(loc='upper right')
plt.scatter(new_data["mass"],H110,LABEL="H110")
plt.legend(loc='upper right')
plt.scatter(new_data["mass"],H111,LABEL="H111")
plt.legend(loc='upper right')
plt.scatter(new_data["mass"],H120,LABEL="H120")
plt.legend(loc='upper right')
plt.scatter(new_data["mass"],H121,LABEL="H121")
plt.legend(loc='upper right')
plt.scatter(new_data["mass"],H122,LABEL="H122")
plt.legend(loc='upper right')
plt.xlim(0.6, 2.)
```

```
[26]: (0.6, 2.0)
```



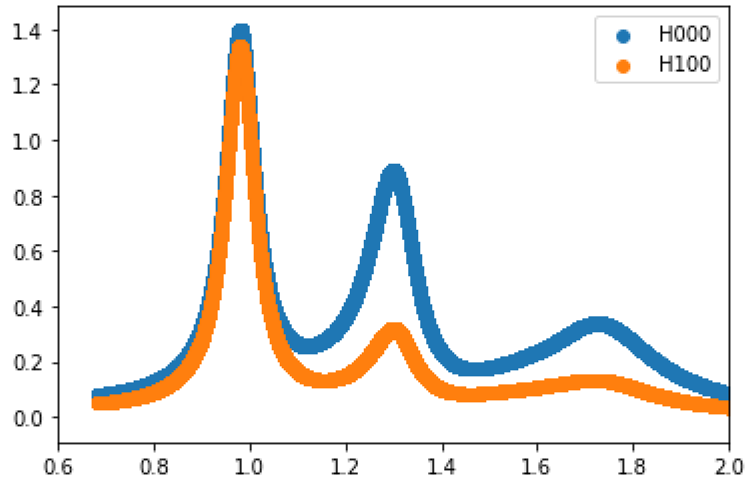
PLot each moment vs mass

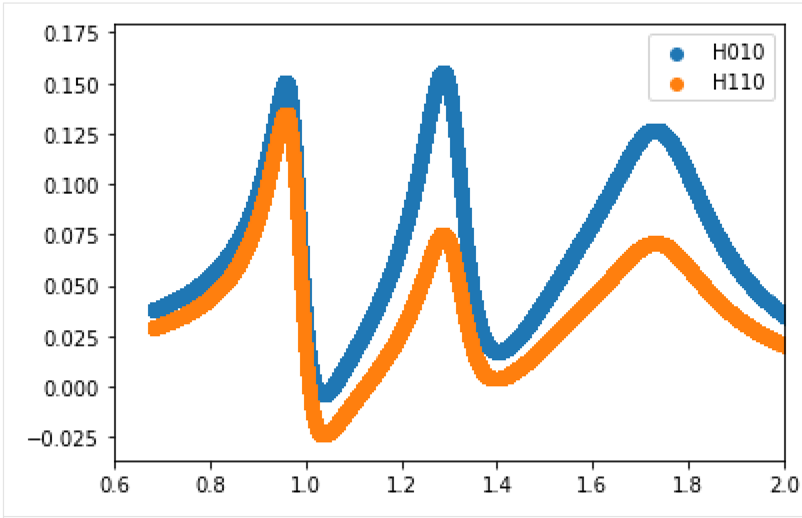
```
[27]: plt.xlim(0.6, 2.)
plt.scatter(new_data["mass"],H000,LABEL="H000")
plt.scatter(new_data["mass"],H100,LABEL="H100")
plt.legend(loc='upper right')
plt.show()
plt.xlim(0.6, 2.)
plt.scatter(new_data["mass"],H010,LABEL="H010")
plt.scatter(new_data["mass"],H110,LABEL="H110")
plt.legend(loc='upper right')
plt.show()
plt.xlim(0.6, 2.)
plt.scatter(new_data["mass"],H011,LABEL="H011")
```

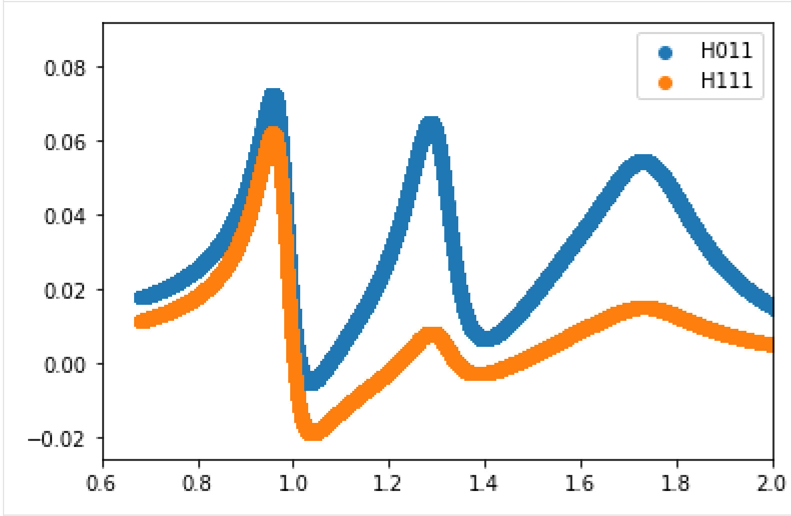
(continues on next page)

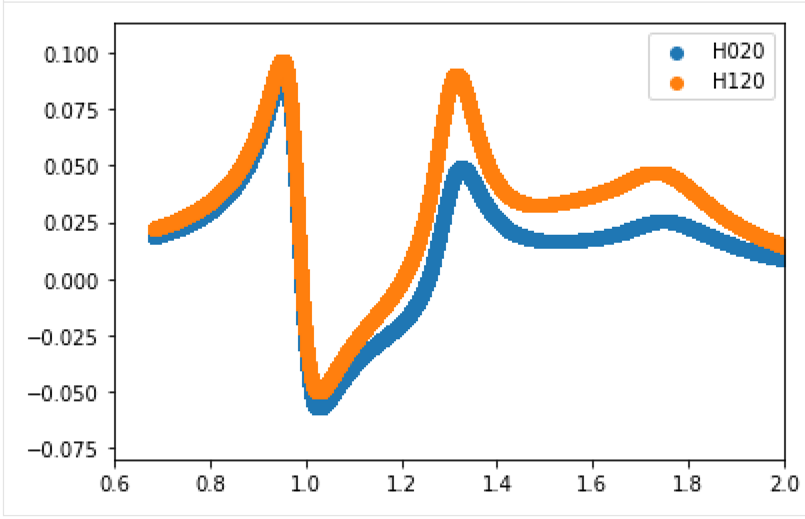
(continued from previous page)

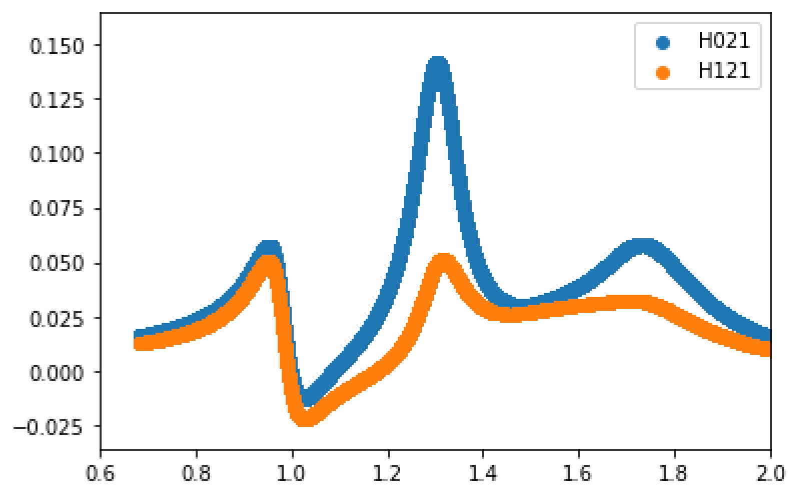
```
plt.scatter(new_data["mass"],H111,LABEL="H111")
plt.legend(loc='upper right')
plt.show()
plt.xlim(0.6, 2.)
plt.scatter(new_data["mass"],H020,LABEL="H020")
plt.scatter(new_data["mass"],H120,LABEL="H120")
plt.legend(loc='upper right')
plt.show()
plt.xlim(0.6, 2.)
plt.scatter(new_data["mass"],H021,LABEL="H021")
plt.scatter(new_data["mass"],H121,LABEL="H121")
plt.legend(loc='upper right')
plt.show()
plt.xlim(0.6, 2.)
plt.scatter(new_data["mass"],H022,LABEL="H022")
plt.scatter(new_data["mass"],H122,LABEL="H122")
plt.legend(loc='upper right')
plt.show()
```

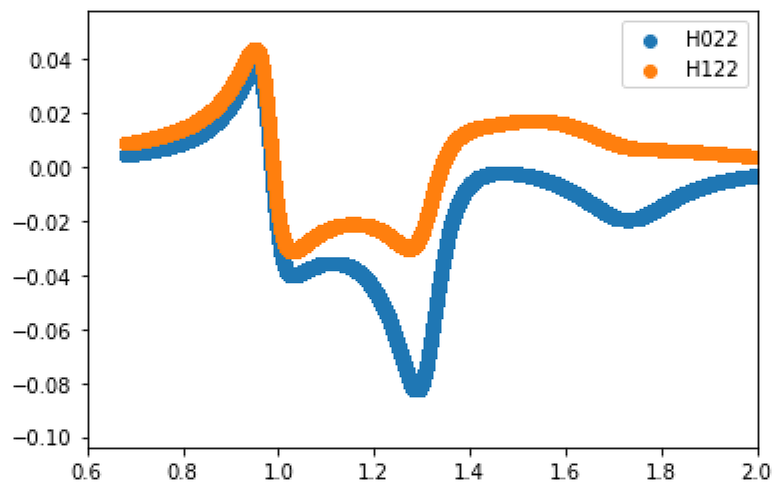



H010
H110
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
−0.025
0.6
0.8
1.0
1.2
1.4
1.6
1.8
2.0


H011
H111
0.08
0.06
0.04
0.02
0.00
−0.02
0.6
0.8
1.0
1.2
1.4
1.6
1.8
2.0

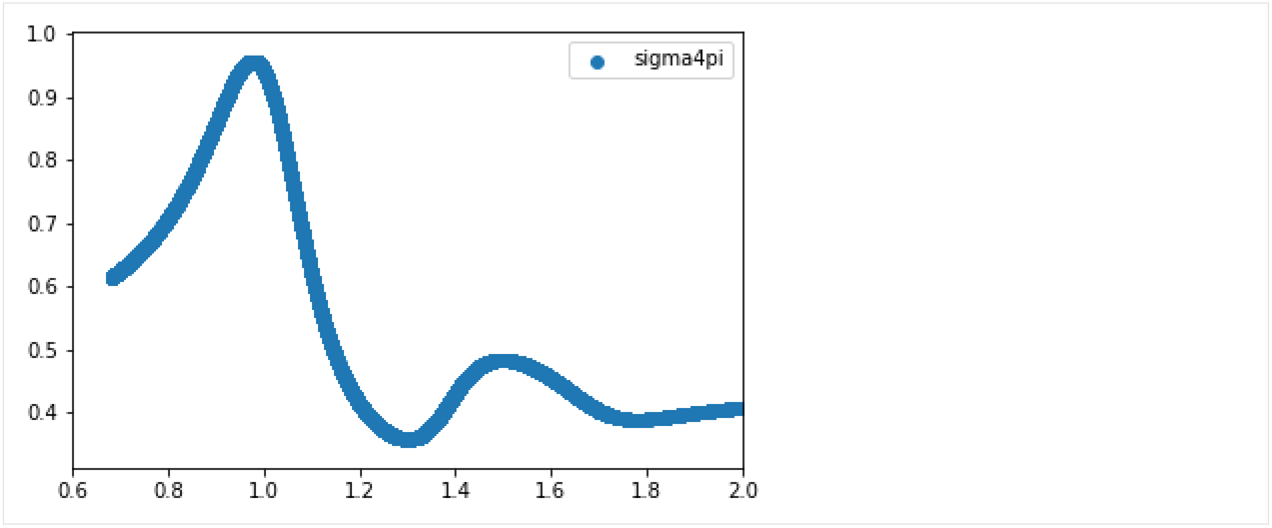

H020
H120
0.100
0.075
0.050
0.025
0.000
−0.025
−0.050
−0.075
0.6
0.8
1.0
1.2
1.4
1.6
1.8
2.0





Plot asymmetry Sigma_4PI

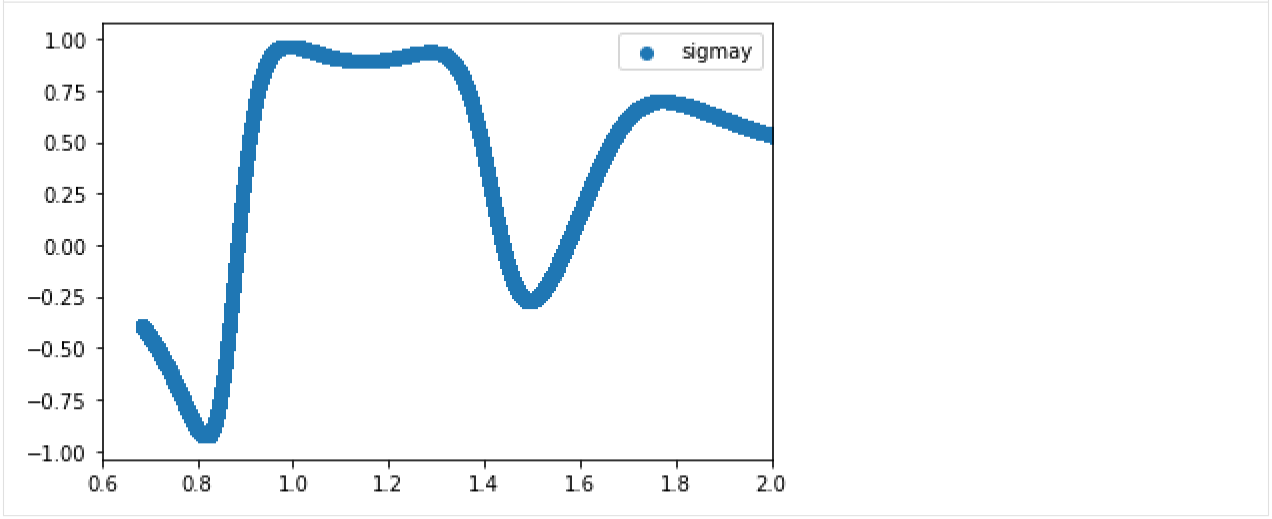
```
[28]: plt.xlim(0.6, 2.)
      plt.scatter(new_data["mass"],sigma4,LABEL="sigma4pi")
      plt.legend(loc='upper right')
      plt.show()
```



Plot asymmetry Sigma_y

```
[29]: plt.xlim(0.6, 2.)
      plt.scatter(new_data["mass"],sigmay,LABEL="sigmay")
      plt.legend(loc='upper right')
```

```
[29]: <matplotlib.legend.Legend at 0x7f0c54a93390>
```



```
[ ]:
```

CHAPTER

SIX

# FITTING TUTORIAL

[1]:
```
import PyPWA as pwa
import pandas
import numpy as npy
from IPython.display import display
import warnings
warnings.filterwarnings('ignore')
```

Define Waves for Fit (and input initial values of minuit and fitted parameters).* >In this example (as expected by the defined amplitude) >each wave is defined by (epsilon.l.m) and each parameter has a real and imaginary part. >i.e a epsilon=-1, l=1 (P wave), m=1 will produce Vs(r.-1.1.1) and Vs(i.-1.1.1) names. >(In this example the imaginary part of the P-wave is kept fixed at 0 value >in the fit) ============================================================

[2]:
```
Vs = {"errordef": 1}
initial = []
for param_type in ["r", "i"]:
    initial.append(f"{param_type}.1.0.0")
    initial.append(f"{param_type}.1.1.0")
    initial.append(f"{param_type}.1.1.1")
    initial.append(f"{param_type}.1.2.0")
    initial.append(f"{param_type}.1.2.1")
    initial.append(f"{param_type}.1.2.2")
#    initial.append(f"{param_type}.1.3.1")
#    initial.append(f"{param_type}.1.4.1")

    Vs[f"{param_type}.1.0.0"] = 10
    Vs[f"limit_{param_type}.1.0.0"] = [-500, 1500.]
    Vs[f"error_{param_type}.1.0.0"] = .1
#    Vs[f"fix_i.-1.0.0"] = True

    Vs[f"{param_type}.1.1.0"] = 10
    Vs[f"limit_{param_type}.1.1.0"] = [-500, 1500.]
    Vs[f"error_{param_type}.1.1.0"] = .1
```

(continues on next page)

(continued from previous page)

```
    Vs[f"{param_type}.1.1.1"] = 10
    Vs[f"limit_{param_type}.1.1.1"] = [-500, 1500.]
    Vs[f"error_{param_type}.1.1.1"] = .1
    Vs[f"fix_i.1.1.1"] = True

    Vs[f"{param_type}.1.2.0"] = 10
    Vs[f"limit_{param_type}.1.2.0"] = [-500, 1500.]
    Vs[f"error_{param_type}.1.2.0"] = .1

    Vs[f"{param_type}.1.2.1"] = 10
    Vs[f"limit_{param_type}.1.2.1"] = [-500, 1500.]
    Vs[f"error_{param_type}.1.2.1"] = .1

    Vs[f"{param_type}.1.2.2"] = 10
    Vs[f"limit_{param_type}.1.2.2"] = [-500, 1500.]
    Vs[f"error_{param_type}.1.2.2"] = .1

#    Vs[f"{param_type}.1.3.1"] = 10
#    Vs[f"limit_{param_type}.1.3.1"] = [-500, 1500.]
#    Vs[f"error_{param_type}.1.3.1"] = .1

#    Vs[f"{param_type}.1.4.1"] = 10
#    Vs[f"limit_{param_type}.1.4.1"] = [-500, 1500.]
#    Vs[f"error_{param_type}.1.4.1"] = .1

Vs[f"i.1.1.1"] = 0.1
```

## 6.1 Read data and montecarlos (accepted and generated) samples

```
[3]: datasample =  pandas.read_csv("simdata_JPAC-np.csv")
#accmcsample = pandas.read_csv("simdata5.csv")
#rawmcsample = pandas.read_csv("simdata5.csv")
#datasample = pwa.read("etapi_data_data.txt")
#accmcsample = pwa.read("etapi_acc.txt")
accmcsample = pwa.read("../TUTORIAL_FILES/etapi_acc.txt")
rawmcsample = pwa.read("../TUTORIAL_FILES/etapi_raw.txt")
```

## 6.2 Binning of the data/monte-carlo and define amplitude (function) to fit* > Here the user difine number of bins, variable to be binned and range

[4]:
```
#import AmplitudeOLDfit
#amplitude = AmplitudeOLDfit.FitAmplitude(initial)
import AmplitudeJPACfit
amplitude = AmplitudeJPACfit.FitAmplitude(initial)
#Define number of bins
nbins = 20
binsda = pwa.bin_by_range(datasample, "mass", nbins, .6, 2.0)
binsma = pwa.bin_by_range(accmcsample, accmcsample["mass"], nbins, .6, 2.0)
binsmr = pwa.bin_by_range(rawmcsample, rawmcsample["mass"], nbins, .6, 2.0)
```

*Check that bins have enough number of events for fit*

[5]:
```
for bin in binsda:
    print(len(bin))
```

```
6577
9364
16674
45953
127024
48129
21406
23014
41661
58948
20078
5179
3404
5434
11026
18505
19504
13486
8425
5481
```

## 6.3 Fitting with Minuit and Extended LogLikelihood

Look at other possibilities through pypwa (use the ?pwa command
or see https://pypwa.jlab.org or https://github.com/JeffersonLab/PyPWA)

[6]:
```
from IPython.display import display
intensities = []
for the_bin in binsda:
#    amp = AmplitudeJPACfit.FitAmplitude(initial)
    amplitude.setup(the_bin)
#    intensities.append(amplitude.calculate(Vs))
    display(pandas.DataFrame(amplitude.calculate(Vs)))
```

```
                0
0      524.818443
1      112.544543
2       75.976211
3       64.049006
4      143.808291
...           ...
6572   105.370331
6573    78.846342
6574   120.701116
6575  1477.089631
6576   123.955803

[6577 rows x 1 columns]
```

```
                0
0       86.996618
1      447.912162
2      161.396588
3      759.107477
4      471.011055
...           ...
9359   161.296093
9360    68.394902
9361   120.358836
9362   146.710380
9363  1196.737830

[9364 rows x 1 columns]
```

```
                0
0      102.392089
1      595.649766
2       65.344248
3      280.705037
```

(continues on next page)

(continued from previous page)

```
4      602.855631
...           ...
16669   67.959747
16670  124.297777
16671  301.015157
16672  456.641653
16673   28.929693

[16674 rows x 1 columns]
```

```
                0
0        4.379040
1       29.951455
2        7.905595
3        7.029131
4      710.321421
...           ...
45948   10.878720
45949  945.802674
45950  206.911768
45951  179.314430
45952    9.352145

[45953 rows x 1 columns]
```

```
                  0
0         45.201774
1         43.788671
2         68.660084
3          9.205558
4        114.399723
...             ...
127019   126.680693
127020  1049.419573
127021    51.796865
127022   157.035089
127023    38.986098

[127024 rows x 1 columns]
```

```
                0
0      274.367106
1       35.553065
2      522.821937
3       92.515519
4       33.446217
...           ...
```

(continues on next page)

(continued from previous page)

```
48124   25.919379
48125   75.859785
48126   48.402383
48127  293.509456
48128  421.146119

[48129 rows x 1 columns]
```

```
                0
0       45.795303
1       15.878769
2       21.845367
3       14.488416
4      344.731653
...           ...
21401   59.818207
21402   13.964827
21403  128.051507
21404  176.612802
21405  245.816445

[21406 rows x 1 columns]
```

```
                 0
0       118.713985
1        67.395399
2        27.073464
3       112.184943
4        67.211745
...            ...
23009    20.237335
23010  1213.036052
23011    53.378732
23012    99.544672
23013    97.962964

[23014 rows x 1 columns]
```

```
                 0
0       148.878988
1        88.325662
2        53.639603
3        58.828367
4      1037.404460
...            ...
41656   507.615053
41657   317.901180
```

(continues on next page)

(continued from previous page)

```
41658    30.504177
41659    62.302882
41660   728.571526

[41661 rows x 1 columns]
```

```
                0
0      110.002848
1      100.119518
2       35.670939
3      104.887662
4       73.708474
...           ...
58943  135.970979
58944   99.755129
58945  155.988587
58946   80.796297
58947  116.174652

[58948 rows x 1 columns]
```

```
                0
0       18.156982
1      770.564549
2      120.300152
3       62.457731
4       65.750478
...           ...
20073   84.231031
20074  914.184940
20075   60.042079
20076  412.368861
20077   19.055842

[20078 rows x 1 columns]
```

```
                0
0      398.611511
1       55.839262
2       77.243593
3     1304.790980
4       72.391068
...           ...
5174    36.506975
5175  1115.010985
5176   143.689987
5177    48.646845
```

(continues on next page)

(continued from previous page)

```
5178   464.702500

[5179 rows x 1 columns]
```

```
                0
0       60.668913
1       34.574231
2      135.356656
3     1201.104334
4       60.986209
...           ...
3399  1428.391920
3400  1399.261054
3401    66.499178
3402    98.375089
3403    73.720043

[3404 rows x 1 columns]
```

```
               0
0     103.818553
1     113.029484
2     105.535405
3     956.658340
4      37.009373
...          ...
5429  109.659727
5430  245.627433
5431   89.075206
5432   46.218475
5433   68.453003

[5434 rows x 1 columns]
```

```
                 0
0       117.917716
1        60.368189
2       106.026393
3        18.873867
4       867.596778
...            ...
11021    49.616098
11022  1108.300986
11023   121.330039
11024   625.404605
11025   103.814676

[11026 rows x 1 columns]
```

```
                0
0      1110.502242
1       345.911006
2        39.504982
3        99.835906
4        75.525220
...            ...
18500   529.065292
18501   119.301695
18502   106.702676
18503   276.974839
18504   538.951930

[18505 rows x 1 columns]
```

```
               0
0      68.286312
1      39.953749
2      60.416889
3      50.038970
4      64.739444
...          ...
19499  77.684112
19500 129.915471
19501 106.508003
19502  77.068989
19503  35.953535

[19504 rows x 1 columns]
```

```
               0
0     339.181092
1     122.302602
2     634.641964
3     151.480484
4      95.661702
...          ...
13481 720.673161
13482 235.111937
13483 729.677239
13484  48.733459
13485  92.768180

[13486 rows x 1 columns]
```

```
               0
0     171.258809
1     540.359474
```

(continues on next page)

(continued from previous page)

```
2       73.843787
3       70.820574
4       55.000461
...           ...
8420   177.519620
8421   435.200540
8422  1199.026642
8423  1107.307499
8424   118.770827

[8425 rows x 1 columns]
```

```
               0
0      83.857029
1     114.315472
2     208.307961
3     188.421035
4      17.385281
...          ...
5476  697.596041
5477   94.189151
5478  114.175654
5479   71.607560
5480   60.128946

[5481 rows x 1 columns]
```

[7]:
```
results = []
for index, dbin in enumerate(binsda):
    with pwa.LogLikelihood(
        amplitude, dbin, binsma[index], generated_length=len(binsmr[index])) as␣
↪Likelihood:
        results.append(
            pwa.minuit(initial, Vs, Likelihood, 1, 2, 2000)
        )
```

*Looking at the results of fitting*

[8]:
```
failed = 0
for index, result in enumerate(results):
    display(f"Bin #{index}:")
    display(result.get_fmin())
    try:
        display(result.get_param_states())
    except:
        failed += 1
```

(continues on next page)

(continued from previous page)

```
        pass

display(f"{(failed / len(results)) * 100}% ({failed})Failed. ")
```

```
'Bin #0:'
```

```
------------------------------------------------------------------
| FCN = -5.432E+04              |      Ncalls=767 (767 total)      |
| EDM = 0.000149 (Goal: 1E-05)  |            up = 1.0              |
------------------------------------------------------------------
|  Valid Min.   | Valid Param.  | Above EDM | Reached call limit |
------------------------------------------------------------------
|     True      |     True      |   False   |       False        |
------------------------------------------------------------------
| Hesse failed  |   Has cov.    | Accurate  | Pos. def. | Forced |
------------------------------------------------------------------
|     False     |     True      |   True    |   True    | False  |
------------------------------------------------------------------
```

```
------------------------------------------------------------------------------------
↪--------
|   | Name    |   Value   | Hesse Err | Minos Err- | Minos Err+ | Limit-   | Limit+ ␣
↪| Fixed |
------------------------------------------------------------------------------------
↪--------
| 0 | r.1.0.0 |    133    |    18     |            |            |  -500    |  1500  ␣
↪|       |
| 1 | r.1.1.0 |     6     |     9     |            |            |  -500    |  1500  ␣
↪|       |
| 2 | r.1.1.1 |   -1.4    |    2.2    |            |            |  -500    |  1500  ␣
↪|       |
| 3 | r.1.2.0 |    55     |    14     |            |            |  -500    |  1500  ␣
↪|       |
| 4 | r.1.2.1 |    67     |    11     |            |            |  -500    |  1500  ␣
↪|       |
| 5 | r.1.2.2 |    66     |     9     |            |            |  -500    |  1500  ␣
↪|       |
| 6 | i.1.0.0 |    79     |    30     |            |            |  -500    |  1500  ␣
↪|       |
| 7 | i.1.1.0 |    -7     |     12    |            |            |  -500    |  1500  ␣
↪|       |
| 8 | i.1.1.1 |   0.10    |   0.10    |            |            |  -500    |  1500  ␣
↪|  yes  |
| 9 | i.1.2.0 |    57     |    17     |            |            |  -500    |  1500  ␣
↪|       |
| 10| i.1.2.1 |    36     |    19     |            |            |  -500    |  1500  ␣
↪|       |
```

(continues on next page)

(continued from previous page)

```
| 11| i.1.2.2 |    30     |    20     |            |            |  -500   |  1500   ␣
↪|       |
------------------------------------------------------------------------------------
↪--------
```

```
'Bin #1:'
```

```
------------------------------------------------------------------
| FCN = -8.009E+04              |    Ncalls=1359 (1359 total)    |
| EDM = 0.00172 (Goal: 1E-05)   |            up = 1.0            |
------------------------------------------------------------------
|  Valid Min.   | Valid Param.  | Above EDM | Reached call limit |
------------------------------------------------------------------
|     True      |     True      |   False   |       False        |
------------------------------------------------------------------
| Hesse failed  |   Has cov.    | Accurate  | Pos. def. | Forced |
------------------------------------------------------------------
|     False     |     True      |   False   |   False   |  True  |
------------------------------------------------------------------
```

```
------------------------------------------------------------------------------------
↪--------
|   | Name    |   Value   | Hesse Err | Minos Err- | Minos Err+ | Limit-  | Limit+  ␣
↪| Fixed |
------------------------------------------------------------------------------------
↪--------
| 0 | r.1.0.0 |    -50    |     60    |            |            |  -500   |  1500   ␣
↪|       |
| 1 | r.1.1.0 |    -39    |     22    |            |            |  -500   |  1500   ␣
↪|       |
| 2 | r.1.1.1 |    2.5    |    6.5    |            |            |  -500   |  1500   ␣
↪|       |
| 3 | r.1.2.0 |    -4     |     33    |            |            |  -500   |  1500   ␣
↪|       |
| 4 | r.1.2.1 |    -33    |     27    |            |            |  -500   |  1500   ␣
↪|       |
| 5 | r.1.2.2 |    -23    |     28    |            |            |  -500   |  1500   ␣
↪|       |
| 6 | i.1.0.0 |    186    |    18     |            |            |  -500   |  1500   ␣
↪|       |
| 7 | i.1.1.0 |    -7     |     17    |            |            |  -500   |  1500   ␣
↪|       |
| 8 | i.1.1.1 |   0.10    |   0.10    |            |            |  -500   |  1500   ␣
↪|  yes  |
| 9 | i.1.2.0 |    82     |     5     |            |            |  -500   |  1500   ␣
↪|       |
| 10| i.1.2.1 |    76     |    11     |            |            |  -500   |  1500   ␣
↪|       |
```

(continues on next page)

(continued from previous page)

```
| 11| i.1.2.2 |    81     |     8     |            |            |  -500   |  1500  ␣
↪|       |
------------------------------------------------------------------------------------
↪--------
```

```
'Bin #2:'
```

```
------------------------------------------------------------------
| FCN = -1.504E+05              |    Ncalls=2607 (2607 total)    |
| EDM = 0.00186 (Goal: 1E-05)   |            up = 1.0            |
------------------------------------------------------------------
|  Valid Min.   | Valid Param.  | Above EDM | Reached call limit |
------------------------------------------------------------------
|     False     |     True      |   False   |        True        |
------------------------------------------------------------------
| Hesse failed  |   Has cov.    | Accurate  | Pos. def. | Forced |
------------------------------------------------------------------
|     False     |     True      |   False   |   True    | False  |
------------------------------------------------------------------
```

```
------------------------------------------------------------------------------------
↪--------
|   | Name    |   Value   | Hesse Err | Minos Err- | Minos Err+ | Limit-  | Limit+ ␣
↪| Fixed |
------------------------------------------------------------------------------------
↪--------
| 0 | r.1.0.0 |    100    |    160    |            |            |  -500   |  1500  ␣
↪|       |
| 1 | r.1.1.0 |    -6     |     10    |            |            |  -500   |  1500  ␣
↪|       |
| 2 | r.1.1.1 |    -4     |     7     |            |            |  -500   |  1500  ␣
↪|       |
| 3 | r.1.2.0 |    50     |    50     |            |            |  -500   |  1500  ␣
↪|       |
| 4 | r.1.2.1 |    70     |    40     |            |            |  -500   |  1500  ␣
↪|       |
| 5 | r.1.2.2 |    60     |    50     |            |            |  -500   |  1500  ␣
↪|       |
| 6 | i.1.0.0 |    260    |    60     |            |            |  -500   |  1500  ␣
↪|       |
| 7 | i.1.1.0 |    2.7    |    2.9    |            |            |  -500   |  1500  ␣
↪|       |
| 8 | i.1.1.1 |   0.10    |   0.10    |            |            |  -500   |  1500  ␣
↪|  yes  |
| 9 | i.1.2.0 |    78     |    28     |            |            |  -500   |  1500  ␣
↪|       |
| 10| i.1.2.1 |    60     |    40     |            |            |  -500   |  1500  ␣
↪|       |
```

(continues on next page)

(continued from previous page)

```
| 11| i.1.2.2 |    70     |    40     |            |            |  -500   |  1500   ␣
↪|         |
------------------------------------------------------------------------------------
↪--------
```

```
'Bin #3:'
```

```
------------------------------------------------------------------
| FCN = -4.538E+05              |    Ncalls=1108 (1108 total)    |
| EDM = 0.00788 (Goal: 1E-05)   |            up = 1.0            |
------------------------------------------------------------------
|  Valid Min.   | Valid Param.  | Above EDM | Reached call limit |
------------------------------------------------------------------
|     False     |     True      |   True    |       False        |
------------------------------------------------------------------
| Hesse failed  |   Has cov.    | Accurate  | Pos. def. | Forced |
------------------------------------------------------------------
|     False     |     True      |   True    |   True    | False  |
------------------------------------------------------------------
```

```
------------------------------------------------------------------------------------
↪--------
|   | Name    |   Value   | Hesse Err | Minos Err- | Minos Err+ | Limit-  | Limit+  ␣
↪| Fixed |
------------------------------------------------------------------------------------
↪--------
| 0 | r.1.0.0 |    395    |    28     |            |            |  -500   |  1500   ␣
↪|         |
| 1 | r.1.1.0 |    -8     |     9     |            |            |  -500   |  1500   ␣
↪|         |
| 2 | r.1.1.1 |    -5     |     4     |            |            |  -500   |  1500   ␣
↪|         |
| 3 | r.1.2.0 |    45     |    11     |            |            |  -500   |  1500   ␣
↪|         |
| 4 | r.1.2.1 |     4     |    14     |            |            |  -500   |  1500   ␣
↪|         |
| 5 | r.1.2.2 |    27     |    12     |            |            |  -500   |  1500   ␣
↪|         |
| 6 | i.1.0.0 |    310    |    40     |            |            |  -500   |  1500   ␣
↪|         |
| 7 | i.1.1.0 |     8     |    11     |            |            |  -500   |  1500   ␣
↪|         |
| 8 | i.1.1.1 |   0.10    |   0.10    |            |            |  -500   |  1500   ␣
↪|  yes  |
| 9 | i.1.2.0 |    82     |    11     |            |            |  -500   |  1500   ␣
↪|         |
| 10| i.1.2.1 |    125    |    10     |            |            |  -500   |  1500   ␣
↪|         |
```

(continues on next page)

(continued from previous page)

```
| 11| i.1.2.2 |    110     |    11      |            |            |  -500    |  1500   ␣
↪|        |
-------------------------------------------------------------------------------------
↪--------
```

```
'Bin #4:'
```

```
------------------------------------------------------------------
| FCN = -1.371E+06              |    Ncalls=2001 (2001 total)    |
| EDM = 0.00707 (Goal: 1E-05)   |            up = 1.0            |
------------------------------------------------------------------
|  Valid Min.   | Valid Param.  | Above EDM | Reached call limit |
------------------------------------------------------------------
|     False     |     True      |   False   |        True        |
------------------------------------------------------------------
| Hesse failed  |   Has cov.    | Accurate  | Pos. def. | Forced |
------------------------------------------------------------------
|     False     |     True      |   False   |   True    | False  |
------------------------------------------------------------------
```

```
-------------------------------------------------------------------------------------
↪--------
|   | Name    |   Value    | Hesse Err | Minos Err- | Minos Err+ | Limit-   | Limit+  ␣
↪| Fixed |
-------------------------------------------------------------------------------------
↪--------
| 0 | r.1.0.0 |    580     |    60     |            |            |  -500    |  1500   ␣
↪|        |
| 1 | r.1.1.0 |    10      |     7     |            |            |  -500    |  1500   ␣
↪|        |
| 2 | r.1.1.1 |     5      |     5     |            |            |  -500    |  1500   ␣
↪|        |
| 3 | r.1.2.0 |    -77     |    13     |            |            |  -500    |  1500   ␣
↪|        |
| 4 | r.1.2.1 |    -82     |    11     |            |            |  -500    |  1500   ␣
↪|        |
| 5 | r.1.2.2 |    -76     |    10     |            |            |  -500    |  1500   ␣
↪|        |
| 6 | i.1.0.0 |    640     |    50     |            |            |  -500    |  1500   ␣
↪|        |
| 7 | i.1.1.0 |    -9      |     4     |            |            |  -500    |  1500   ␣
↪|        |
| 8 | i.1.1.1 |   0.10     |   0.10    |            |            |  -500    |  1500   ␣
↪|  yes   |
| 9 | i.1.2.0 |    96      |     6     |            |            |  -500    |  1500   ␣
↪|        |
| 10| i.1.2.1 |    95      |     9     |            |            |  -500    |  1500   ␣
↪|        |
```

(continues on next page)

(continued from previous page)

```
| 11| i.1.2.2 |    100    |    11     |            |            |  -500   |  1500   ␣
↪|         |
------------------------------------------------------------------------------------
↪--------
```

```
'Bin #5:'
```

```
------------------------------------------------------------------
| FCN = -4.759E+05              |    Ncalls=1560 (1560 total)    |
| EDM = 0.000158 (Goal: 1E-05)  |            up = 1.0            |
------------------------------------------------------------------
|  Valid Min.   | Valid Param.  | Above EDM | Reached call limit |
------------------------------------------------------------------
|     True      |     True      |   False   |       False        |
------------------------------------------------------------------
| Hesse failed  |   Has cov.    | Accurate  | Pos. def. | Forced |
------------------------------------------------------------------
|     False     |     True      |   True    |   True    | False  |
------------------------------------------------------------------
```

```
------------------------------------------------------------------------------------
↪--------
|   | Name    |   Value   | Hesse Err | Minos Err- | Minos Err+ | Limit-  | Limit+  ␣
↪| Fixed |
------------------------------------------------------------------------------------
↪--------
| 0 | r.1.0.0 |    -3     |     16    |            |            |  -500   |  1500   ␣
↪|         |
| 1 | r.1.1.0 |    165    |    11     |            |            |  -500   |  1500   ␣
↪|         |
| 2 | r.1.1.1 |    125    |    13     |            |            |  -500   |  1500   ␣
↪|         |
| 3 | r.1.2.0 |   -2.9    |    5.5    |            |            |  -500   |  1500   ␣
↪|         |
| 4 | r.1.2.1 |   -1.8    |    7.3    |            |            |  -500   |  1500   ␣
↪|         |
| 5 | r.1.2.2 |    2.1    |   11.2    |            |            |  -500   |  1500   ␣
↪|         |
| 6 | i.1.0.0 |    475    |     7     |            |            |  -500   |  1500   ␣
↪|         |
| 7 | i.1.1.0 |     6     |     5     |            |            |  -500   |  1500   ␣
↪|         |
| 8 | i.1.1.1 |   0.10    |   0.10    |            |            |  -500   |  1500   ␣
↪|   yes   |
| 9 | i.1.2.0 |   -106    |     7     |            |            |  -500   |  1500   ␣
↪|         |
| 10| i.1.2.1 |   -118    |     7     |            |            |  -500   |  1500   ␣
↪|         |
```

(continues on next page)

(continued from previous page)

```
| 11| i.1.2.2 |   -95.4   |    2.8    |            |            |  -500   |  1500   ␣
↪|       |
------------------------------------------------------------------------------------
↪--------
```

```
'Bin #6:'
```

```
------------------------------------------------------------------
| FCN = -1.975E+05              |    Ncalls=1431 (1431 total)    |
| EDM = 6.85E-06 (Goal: 1E-05)  |            up = 1.0            |
------------------------------------------------------------------
|  Valid Min.   | Valid Param.  | Above EDM | Reached call limit |
------------------------------------------------------------------
|     True      |     True      |   False   |       False        |
------------------------------------------------------------------
| Hesse failed  |   Has cov.    | Accurate  | Pos. def. | Forced |
------------------------------------------------------------------
|     False     |     True      |   True    |   True    | False  |
------------------------------------------------------------------
```

```
------------------------------------------------------------------------------------
↪--------
|   | Name    |   Value   | Hesse Err | Minos Err- | Minos Err+ | Limit-  | Limit+  ␣
↪| Fixed |
------------------------------------------------------------------------------------
↪--------
| 0 | r.1.0.0 |   -1.1    |    14.4   |            |            |  -500   |  1500   ␣
↪|       |
| 1 | r.1.1.0 |    94     |     4     |            |            |  -500   |  1500   ␣
↪|       |
| 2 | r.1.1.1 |    74     |     7     |            |            |  -500   |  1500   ␣
↪|       |
| 3 | r.1.2.0 |     6     |     8     |            |            |  -500   |  1500   ␣
↪|       |
| 4 | r.1.2.1 |   -0.4    |    7.8    |            |            |  -500   |  1500   ␣
↪|       |
| 5 | r.1.2.2 |     6     |    13     |            |            |  -500   |  1500   ␣
↪|       |
| 6 | i.1.0.0 |   237.5   |    3.0    |            |            |  -500   |  1500   ␣
↪|       |
| 7 | i.1.1.0 |     3     |     4     |            |            |  -500   |  1500   ␣
↪|       |
| 8 | i.1.1.1 |   0.10    |   0.10    |            |            |  -500   |  1500   ␣
↪|  yes  |
| 9 | i.1.2.0 |   -150    |     3     |            |            |  -500   |  1500   ␣
↪|       |
| 10| i.1.2.1 |   -146    |     4     |            |            |  -500   |  1500   ␣
↪|       |
```

(continues on next page)

(continued from previous page)

```
| 11| i.1.2.2 |  -139.8   |    2.8    |            |            |  -500   |  1500   ␣
↪|         |
------------------------------------------------------------------------------------
↪--------
```

```
'Bin #7:'
```

```
------------------------------------------------------------------
| FCN = -2.144E+05              |    Ncalls=1645 (1645 total)    |
| EDM = 2.65E-05 (Goal: 1E-05)  |            up = 1.0            |
------------------------------------------------------------------
|  Valid Min.   | Valid Param.  | Above EDM | Reached call limit |
------------------------------------------------------------------
|     True      |     True      |   False   |       False        |
------------------------------------------------------------------
| Hesse failed  |   Has cov.    | Accurate  | Pos. def. | Forced |
------------------------------------------------------------------
|     False     |     True      |   True    |   True    | False  |
------------------------------------------------------------------
```

```
------------------------------------------------------------------------------------
↪--------
|   | Name    |   Value   | Hesse Err | Minos Err- | Minos Err+ | Limit-  | Limit+ ␣
↪| Fixed |
------------------------------------------------------------------------------------
↪--------
| 0 | r.1.0.0 |    -80    |     19    |            |            |  -500   |  1500   ␣
↪|       |
| 1 | r.1.1.0 |    -5     |     6     |            |            |  -500   |  1500   ␣
↪|       |
| 2 | r.1.1.1 |    -12    |     7     |            |            |  -500   |  1500   ␣
↪|       |
| 3 | r.1.2.0 |    0.7    |   27.3    |            |            |  -500   |  1500   ␣
↪|       |
| 4 | r.1.2.1 |     4     |    31     |            |            |  -500   |  1500   ␣
↪|       |
| 5 | r.1.2.2 |    -8     |     25    |            |            |  -500   |  1500   ␣
↪|       |
| 6 | i.1.0.0 |   -139    |     11    |            |            |  -500   |  1500   ␣
↪|       |
| 7 | i.1.1.0 |   -3.5    |    2.6    |            |            |  -500   |  1500   ␣
↪|       |
| 8 | i.1.1.1 |   0.10    |   0.10    |            |            |  -500   |  1500   ␣
↪|  yes  |
| 9 | i.1.2.0 |   203.4   |    2.6    |            |            |  -500   |  1500   ␣
↪|       |
| 10| i.1.2.1 |   198.9   |    3.0    |            |            |  -500   |  1500   ␣
↪|       |
```

(continues on next page)

(continued from previous page)

```
| 11| i.1.2.2 |    195    |     4     |            |            |  -500   |  1500   ␣
↪|         |
------------------------------------------------------------------------------------
↪--------
```

```
'Bin #8:'
```

```
------------------------------------------------------------------
| FCN = -4.129E+05              |    Ncalls=1510 (1510 total)    |
| EDM = 1.59E-06 (Goal: 1E-05)  |            up = 1.0            |
------------------------------------------------------------------
|  Valid Min.   | Valid Param.  | Above EDM | Reached call limit |
------------------------------------------------------------------
|     True      |     True      |   False   |       False        |
------------------------------------------------------------------
| Hesse failed  |   Has cov.    | Accurate  | Pos. def. | Forced |
------------------------------------------------------------------
|     False     |     True      |   False   |   False   |  True  |
------------------------------------------------------------------
```

```
------------------------------------------------------------------------------------
↪--------
|   | Name    |   Value   | Hesse Err | Minos Err- | Minos Err+ | Limit-  | Limit+  ␣
↪| Fixed |
------------------------------------------------------------------------------------
↪--------
| 0 | r.1.0.0 |   -3.1    |    16.3   |            |            |  -500   |  1500   ␣
↪|         |
| 1 | r.1.1.0 |    2.1    |    5.3    |            |            |  -500   |  1500   ␣
↪|         |
| 2 | r.1.1.1 |   0.06    |   3.26    |            |            |  -500   |  1500   ␣
↪|         |
| 3 | r.1.2.0 |    156    |    29     |            |            |  -500   |  1500   ␣
↪|         |
| 4 | r.1.2.1 |    243    |    20     |            |            |  -500   |  1500   ␣
↪|         |
| 5 | r.1.2.2 |    167    |    27     |            |            |  -500   |  1500   ␣
↪|         |
| 6 | i.1.0.0 |    105    |     8     |            |            |  -500   |  1500   ␣
↪|         |
| 7 | i.1.1.0 |    1.0    |    4.9    |            |            |  -500   |  1500   ␣
↪|         |
| 8 | i.1.1.1 |   0.10    |   0.10    |            |            |  -500   |  1500   ␣
↪|  yes   |
| 9 | i.1.2.0 |   -248    |    19     |            |            |  -500   |  1500   ␣
↪|         |
| 10| i.1.2.1 |   -169    |    29     |            |            |  -500   |  1500   ␣
↪|         |
```

(continues on next page)

(continued from previous page)

```
| 11| i.1.2.2 |   -223    |     20    |            |            |  -500   |  1500  ␣
↪|         |
------------------------------------------------------------------------------------
↪--------
```

```
'Bin #9:'
```

```
------------------------------------------------------------------
| FCN = -6.071E+05              |    Ncalls=1829 (1829 total)    |
| EDM = 0.00232 (Goal: 1E-05)   |            up = 1.0            |
------------------------------------------------------------------
|  Valid Min.   | Valid Param.  | Above EDM | Reached call limit |
------------------------------------------------------------------
|     False     |     True      |   True    |       False        |
------------------------------------------------------------------
| Hesse failed  |   Has cov.    | Accurate  | Pos. def. | Forced |
------------------------------------------------------------------
|     False     |     True      |   False   |   False   |  True  |
------------------------------------------------------------------
```

```
------------------------------------------------------------------------------------
↪--------
|   | Name    |   Value   | Hesse Err | Minos Err- | Minos Err+ | Limit-  | Limit+ ␣
↪| Fixed |
------------------------------------------------------------------------------------
↪--------
| 0 | r.1.0.0 |    50     |    21     |            |            |  -500   |  1500  ␣
↪|       |
| 1 | r.1.1.0 |    -9     |     9     |            |            |  -500   |  1500  ␣
↪|       |
| 2 | r.1.1.1 |   -0.8    |   10.2    |            |            |  -500   |  1500  ␣
↪|       |
| 3 | r.1.2.0 |    -80    |    80     |            |            |  -500   |  1500  ␣
↪|       |
| 4 | r.1.2.1 |    25     |    89     |            |            |  -500   |  1500  ␣
↪|       |
| 5 | r.1.2.2 |    -30    |    80     |            |            |  -500   |  1500  ␣
↪|       |
| 6 | i.1.0.0 |   0.16    |   10.69   |            |            |  -500   |  1500  ␣
↪|       |
| 7 | i.1.1.0 |     6     |     3     |            |            |  -500   |  1500  ␣
↪|       |
| 8 | i.1.1.1 |   0.10    |   0.10    |            |            |  -500   |  1500  ␣
↪|  yes  |
| 9 | i.1.2.0 |   -344    |    18     |            |            |  -500   |  1500  ␣
↪|       |
| 10| i.1.2.1 |   -357    |     7     |            |            |  -500   |  1500  ␣
↪|       |
```

(continues on next page)

(continued from previous page)

```
| 11| i.1.2.2 |   -340    |     7     |            |            |  -500   |  1500   ␣
↪|        |
------------------------------------------------------------------------------------
↪--------
```

```
'Bin #10:'
```

```
------------------------------------------------------------------
| FCN = -1.854E+05              |    Ncalls=1098 (1098 total)    |
| EDM = 0.00301 (Goal: 1E-05)   |            up = 1.0            |
------------------------------------------------------------------
|  Valid Min.   | Valid Param.  | Above EDM | Reached call limit |
------------------------------------------------------------------
|     False     |     True      |   True    |       False        |
------------------------------------------------------------------
| Hesse failed  |   Has cov.    | Accurate  | Pos. def. | Forced |
------------------------------------------------------------------
|     False     |     True      |   False   |   False   |  True  |
------------------------------------------------------------------
```

```
------------------------------------------------------------------------------------
↪--------
|   | Name    |   Value   | Hesse Err | Minos Err- | Minos Err+ | Limit-  | Limit+  ␣
↪| Fixed |
------------------------------------------------------------------------------------
↪--------
| 0 | r.1.0.0 |    36     |    11     |            |            |  -500   |  1500   ␣
↪|        |
| 1 | r.1.1.0 |    -8     |     16    |            |            |  -500   |  1500   ␣
↪|        |
| 2 | r.1.1.1 |    1.6    |    3.4    |            |            |  -500   |  1500   ␣
↪|        |
| 3 | r.1.2.0 |    202    |    12     |            |            |  -500   |  1500   ␣
↪|        |
| 4 | r.1.2.1 |    203    |     6     |            |            |  -500   |  1500   ␣
↪|        |
| 5 | r.1.2.2 |    195    |     4     |            |            |  -500   |  1500   ␣
↪|        |
| 6 | i.1.0.0 |    31     |    15     |            |            |  -500   |  1500   ␣
↪|        |
| 7 | i.1.1.0 |    -48    |     7     |            |            |  -500   |  1500   ␣
↪|        |
| 8 | i.1.1.1 |   0.10    |   0.10    |            |            |  -500   |  1500   ␣
↪|  yes   |
| 9 | i.1.2.0 |    -30    |     70    |            |            |  -500   |  1500   ␣
↪|        |
| 10| i.1.2.1 |    17     |    66     |            |            |  -500   |  1500   ␣
↪|        |
```

(continues on next page)

(continued from previous page)

```
| 11| i.1.2.2 |     5     |    69     |            |            |  -500   |  1500   ␣
↪|       |
------------------------------------------------------------------------------------
↪--------
```

```
'Bin #11:'
```

```
------------------------------------------------------------------
| FCN = -4.068E+04              |    Ncalls=1721 (1721 total)    |
| EDM = 0.000158 (Goal: 1E-05)  |            up = 1.0            |
------------------------------------------------------------------
|  Valid Min.   | Valid Param.  | Above EDM | Reached call limit |
------------------------------------------------------------------
|     True      |     True      |   False   |       False        |
------------------------------------------------------------------
| Hesse failed  |   Has cov.    | Accurate  | Pos. def. | Forced |
------------------------------------------------------------------
|     False     |     True      |   True    |   True    | False  |
------------------------------------------------------------------
```

```
------------------------------------------------------------------------------------
↪--------
|   | Name    |   Value   | Hesse Err | Minos Err- | Minos Err+ | Limit-  | Limit+  ␣
↪| Fixed |
------------------------------------------------------------------------------------
↪--------
| 0 | r.1.0.0 |    16     |    59     |            |            |  -500   |  1500   ␣
↪|       |
| 1 | r.1.1.0 |     6     |     9     |            |            |  -500   |  1500   ␣
↪|       |
| 2 | r.1.1.1 |     6     |     8     |            |            |  -500   |  1500   ␣
↪|       |
| 3 | r.1.2.0 |    -70    |    60     |            |            |  -500   |  1500   ␣
↪|       |
| 4 | r.1.2.1 |    -70    |    60     |            |            |  -500   |  1500   ␣
↪|       |
| 5 | r.1.2.2 |    -80    |    60     |            |            |  -500   |  1500   ␣
↪|       |
| 6 | i.1.0.0 |    66     |    19     |            |            |  -500   |  1500   ␣
↪|       |
| 7 | i.1.1.0 |    2.8    |    4.8    |            |            |  -500   |  1500   ␣
↪|       |
| 8 | i.1.1.1 |   0.10    |   0.10    |            |            |  -500   |  1500   ␣
↪|  yes  |
| 9 | i.1.2.0 |    70     |    70     |            |            |  -500   |  1500   ␣
↪|       |
| 10| i.1.2.1 |    70     |    60     |            |            |  -500   |  1500   ␣
↪|       |
```

(continues on next page)

(continued from previous page)

```
| 11| i.1.2.2 |    70     |    70     |            |            |  -500    |  1500  ␣
↪|        |
------------------------------------------------------------------------------------
↪--------
```

```
'Bin #12:'
```

```
------------------------------------------------------------------
| FCN = -2.508E+04              |    Ncalls=1372 (1372 total)    |
| EDM = 7.06E-05 (Goal: 1E-05)  |            up = 1.0            |
------------------------------------------------------------------
|  Valid Min.   | Valid Param.  | Above EDM | Reached call limit |
------------------------------------------------------------------
|     True      |     True      |   False   |       False        |
------------------------------------------------------------------
| Hesse failed  |   Has cov.    | Accurate  | Pos. def. | Forced |
------------------------------------------------------------------
|     False     |     True      |   True    |   True    | False  |
------------------------------------------------------------------
```

```
------------------------------------------------------------------------------------
↪--------
|   | Name    |   Value   | Hesse Err | Minos Err- | Minos Err+ | Limit-   | Limit+ ␣
↪| Fixed |
------------------------------------------------------------------------------------
↪--------
| 0 | r.1.0.0 |    -36    |     15    |            |            |  -500    |  1500  ␣
↪|        |
| 1 | r.1.1.0 |    -9     |     5     |            |            |  -500    |  1500  ␣
↪|        |
| 2 | r.1.1.1 |    -5     |     5     |            |            |  -500    |  1500  ␣
↪|        |
| 3 | r.1.2.0 |    60     |    23     |            |            |  -500    |  1500  ␣
↪|        |
| 4 | r.1.2.1 |    57     |    23     |            |            |  -500    |  1500  ␣
↪|        |
| 5 | r.1.2.2 |    51     |    22     |            |            |  -500    |  1500  ␣
↪|        |
| 6 | i.1.0.0 |    36     |    16     |            |            |  -500    |  1500  ␣
↪|        |
| 7 | i.1.1.0 |    1.2    |    5.0    |            |            |  -500    |  1500  ␣
↪|        |
| 8 | i.1.1.1 |   0.10    |   0.10    |            |            |  -500    |  1500  ␣
↪|  yes   |
| 9 | i.1.2.0 |    56     |    24     |            |            |  -500    |  1500  ␣
↪|        |
| 10| i.1.2.1 |    58     |    22     |            |            |  -500    |  1500  ␣
↪|        |
```

(continues on next page)

(continued from previous page)

```
| 11| i.1.2.2 |    54     |    21     |            |            |  -500   |  1500   ␣
↪|       |
------------------------------------------------------------------------------------
↪--------
```

```
'Bin #13:'
```

```
------------------------------------------------------------------
| FCN = -4.279E+04              |    Ncalls=1079 (1079 total)    |
| EDM = 0.000103 (Goal: 1E-05)  |            up = 1.0            |
------------------------------------------------------------------
|  Valid Min.   | Valid Param.  | Above EDM | Reached call limit |
------------------------------------------------------------------
|     True      |     True      |   False   |       False        |
------------------------------------------------------------------
| Hesse failed  |   Has cov.    | Accurate  | Pos. def. | Forced |
------------------------------------------------------------------
|     False     |     True      |   True    |   True    | False  |
------------------------------------------------------------------
```

```
------------------------------------------------------------------------------------
↪--------
|   | Name    |   Value   | Hesse Err | Minos Err- | Minos Err+ | Limit-  | Limit+  ␣
↪| Fixed |
------------------------------------------------------------------------------------
↪--------
| 0 | r.1.0.0 |    15     |    10     |            |            |  -500   |  1500   ␣
↪|       |
| 1 | r.1.1.0 |    43     |     7     |            |            |  -500   |  1500   ␣
↪|       |
| 2 | r.1.1.1 |    31     |     9     |            |            |  -500   |  1500   ␣
↪|       |
| 3 | r.1.2.0 |   -11     |    15     |            |            |  -500   |  1500   ␣
↪|       |
| 4 | r.1.2.1 |   -12     |    12     |            |            |  -500   |  1500   ␣
↪|       |
| 5 | r.1.2.2 |   -18     |    16     |            |            |  -500   |  1500   ␣
↪|       |
| 6 | i.1.0.0 |   -20     |     4     |            |            |  -500   |  1500   ␣
↪|       |
| 7 | i.1.1.0 |     4     |     6     |            |            |  -500   |  1500   ␣
↪|       |
| 8 | i.1.1.1 |   0.10    |   0.10    |            |            |  -500   |  1500   ␣
↪|  yes  |
| 9 | i.1.2.0 |   102.4   |    3.1    |            |            |  -500   |  1500   ␣
↪|       |
| 10| i.1.2.1 |    102    |     3     |            |            |  -500   |  1500   ␣
↪|       |
```

(continues on next page)

(continued from previous page)

```
| 11| i.1.2.2 |    97     |     4     |            |            |  -500   |  1500   ␣
↪|        |
-------------------------------------------------------------------------------------
↪--------
```

```
'Bin #14:'
```

```
------------------------------------------------------------------
| FCN = -9.5E+04                |    Ncalls=1294 (1294 total)    |
| EDM = 0.000535 (Goal: 1E-05)  |            up = 1.0            |
------------------------------------------------------------------
|  Valid Min.   | Valid Param.  | Above EDM | Reached call limit |
------------------------------------------------------------------
|     True      |     True      |   False   |       False        |
------------------------------------------------------------------
| Hesse failed  |   Has cov.    | Accurate  | Pos. def. | Forced |
------------------------------------------------------------------
|     False     |     True      |   True    |   True    | False  |
------------------------------------------------------------------
```

```
-------------------------------------------------------------------------------------
↪--------
|   | Name    |   Value   | Hesse Err | Minos Err- | Minos Err+ | Limit-   | Limit+  ␣
↪| Fixed |
-------------------------------------------------------------------------------------
↪--------
| 0 | r.1.0.0 |    -11    |     14    |            |            |  -500   |  1500   ␣
↪|        |
| 1 | r.1.1.0 |    2.4    |    6.6    |            |            |  -500   |  1500   ␣
↪|        |
| 2 | r.1.1.1 |    2.8    |    4.6    |            |            |  -500   |  1500   ␣
↪|        |
| 3 | r.1.2.0 |    115    |    23     |            |            |  -500   |  1500   ␣
↪|        |
| 4 | r.1.2.1 |    90     |    26     |            |            |  -500   |  1500   ␣
↪|        |
| 5 | r.1.2.2 |    97     |    28     |            |            |  -500   |  1500   ␣
↪|        |
| 6 | i.1.0.0 |    -4     |     14    |            |            |  -500   |  1500   ␣
↪|        |
| 7 | i.1.1.0 |   -2.9    |    6.1    |            |            |  -500   |  1500   ␣
↪|        |
| 8 | i.1.1.1 |   0.10    |   0.10    |            |            |  -500   |  1500   ␣
↪|  yes   |
| 9 | i.1.2.0 |    103    |    25     |            |            |  -500   |  1500   ␣
↪|        |
| 10| i.1.2.1 |    123    |    19     |            |            |  -500   |  1500   ␣
↪|        |
```

(continues on next page)

(continued from previous page)

```
| 11| i.1.2.2 |    112    |    25     |            |            |  -500   |  1500   ␣
↪|         |
------------------------------------------------------------------------------------
↪--------
```

```
'Bin #15:'
```

```
------------------------------------------------------------------
| FCN = -1.692E+05              |    Ncalls=1111 (1111 total)    |
| EDM = 0.00244 (Goal: 1E-05)   |            up = 1.0            |
------------------------------------------------------------------
|  Valid Min.   | Valid Param.  | Above EDM | Reached call limit |
------------------------------------------------------------------
|     False     |     True      |   True    |       False        |
------------------------------------------------------------------
| Hesse failed  |   Has cov.    | Accurate  | Pos. def. | Forced |
------------------------------------------------------------------
|     False     |     True      |   False   |   False   |  True  |
------------------------------------------------------------------
```

```
------------------------------------------------------------------------------------
↪--------
|   | Name    |   Value   | Hesse Err | Minos Err- | Minos Err+ | Limit-  | Limit+  ␣
↪| Fixed |
------------------------------------------------------------------------------------
↪--------
| 0 | r.1.0.0 |     8     |    19     |            |            |  -500   |  1500   ␣
↪|         |
| 1 | r.1.1.0 |   -2.5    |    9.3    |            |            |  -500   |  1500   ␣
↪|         |
| 2 | r.1.1.1 |     3     |     4     |            |            |  -500   |  1500   ␣
↪|         |
| 3 | r.1.2.0 |    193    |     9     |            |            |  -500   |  1500   ␣
↪|         |
| 4 | r.1.2.1 |    198    |     3     |            |            |  -500   |  1500   ␣
↪|         |
| 5 | r.1.2.2 |    190    |    15     |            |            |  -500   |  1500   ␣
↪|         |
| 6 | i.1.0.0 |    40     |    40     |            |            |  -500   |  1500   ␣
↪|         |
| 7 | i.1.1.0 |    -28    |     13    |            |            |  -500   |  1500   ␣
↪|         |
| 8 | i.1.1.1 |   0.10    |   0.10    |            |            |  -500   |  1500   ␣
↪|   yes   |
| 9 | i.1.2.0 |    -22    |     75    |            |            |  -500   |  1500   ␣
↪|         |
| 10| i.1.2.1 |   -2.7    |    59.9   |            |            |  -500   |  1500   ␣
↪|         |
```

(continues on next page)

(continued from previous page)

```
| 11| i.1.2.2 |    -25    |     87    |            |            |  -500   |  1500   ␣
↪|         |
------------------------------------------------------------------------------------
↪--------
```

```
'Bin #16:'
```

```
------------------------------------------------------------------
| FCN = -1.798E+05              |    Ncalls=1159 (1159 total)    |
| EDM = 0.00754 (Goal: 1E-05)   |            up = 1.0            |
------------------------------------------------------------------
|  Valid Min.   | Valid Param.  | Above EDM | Reached call limit |
------------------------------------------------------------------
|     False     |     True      |   True    |       False        |
------------------------------------------------------------------
| Hesse failed  |   Has cov.    | Accurate  | Pos. def. | Forced |
------------------------------------------------------------------
|     False     |     True      |   False   |   False   |  True  |
------------------------------------------------------------------
```

```
------------------------------------------------------------------------------------
↪--------
|   | Name    |   Value   | Hesse Err | Minos Err- | Minos Err+ | Limit-  | Limit+  ␣
↪| Fixed |
------------------------------------------------------------------------------------
↪--------
| 0 | r.1.0.0 |     9     |    13     |            |            |  -500   |  1500   ␣
↪|       |
| 1 | r.1.1.0 |   -2.1    |    5.4    |            |            |  -500   |  1500   ␣
↪|       |
| 2 | r.1.1.1 |    2.3    |    3.1    |            |            |  -500   |  1500   ␣
↪|       |
| 3 | r.1.2.0 |    201    |     7     |            |            |  -500   |  1500   ␣
↪|       |
| 4 | r.1.2.1 |    196    |    17     |            |            |  -500   |  1500   ␣
↪|       |
| 5 | r.1.2.2 |    197    |     7     |            |            |  -500   |  1500   ␣
↪|       |
| 6 | i.1.0.0 |    40     |    23     |            |            |  -500   |  1500   ␣
↪|       |
| 7 | i.1.1.0 |    -14    |     13    |            |            |  -500   |  1500   ␣
↪|       |
| 8 | i.1.1.1 |   0.10    |   0.10    |            |            |  -500   |  1500   ␣
↪|  yes  |
| 9 | i.1.2.0 |    17     |    77     |            |            |  -500   |  1500   ␣
↪|       |
| 10| i.1.2.1 |    50     |    70     |            |            |  -500   |  1500   ␣
↪|       |
```

(continues on next page)

(continued from previous page)

```
| 11| i.1.2.2 |    19     |    81     |            |            |  -500   |  1500   ␣
↪|       |
------------------------------------------------------------------------------------
↪--------
```

```
'Bin #17:'
```

```
------------------------------------------------------------------
| FCN = -1.193E+05              |    Ncalls=1486 (1486 total)    |
| EDM = 0.000675 (Goal: 1E-05)  |            up = 1.0            |
------------------------------------------------------------------
|  Valid Min.   | Valid Param.  | Above EDM | Reached call limit |
------------------------------------------------------------------
|     True      |     True      |   False   |       False        |
------------------------------------------------------------------
| Hesse failed  |   Has cov.    | Accurate  | Pos. def. | Forced |
------------------------------------------------------------------
|     False     |     True      |   False   |   False   |  True  |
------------------------------------------------------------------
```

```
------------------------------------------------------------------------------------
↪--------
|   | Name    |   Value   | Hesse Err | Minos Err- | Minos Err+ | Limit-  | Limit+  ␣
↪| Fixed |
------------------------------------------------------------------------------------
↪--------
| 0 | r.1.0.0 |    21     |    13     |            |            |  -500   |  1500   ␣
↪|       |
| 1 | r.1.1.0 |    0.5    |    7.1    |            |            |  -500   |  1500   ␣
↪|       |
| 2 | r.1.1.1 |  -0.013   |   3.887   |            |            |  -500   |  1500   ␣
↪|       |
| 3 | r.1.2.0 |    165    |     7     |            |            |  -500   |  1500   ␣
↪|       |
| 4 | r.1.2.1 |    164    |    16     |            |            |  -500   |  1500   ␣
↪|       |
| 5 | r.1.2.2 |    166    |     6     |            |            |  -500   |  1500   ␣
↪|       |
| 6 | i.1.0.0 |    28     |    25     |            |            |  -500   |  1500   ␣
↪|       |
| 7 | i.1.1.0 |    15     |    17     |            |            |  -500   |  1500   ␣
↪|       |
| 8 | i.1.1.1 |   0.10    |   0.10    |            |            |  -500   |  1500   ␣
↪|  yes  |
| 9 | i.1.2.0 |    11     |    96     |            |            |  -500   |  1500   ␣
↪|       |
| 10| i.1.2.1 |    40     |    80     |            |            |  -500   |  1500   ␣
↪|       |
```

(continues on next page)

(continued from previous page)

```
| 11| i.1.2.2 |    13     |    97     |            |            |  -500   |  1500   ␣
↪|        |
------------------------------------------------------------------------------------
↪--------
```

```
'Bin #18:'
```

```
------------------------------------------------------------------
| FCN = -7.069E+04              |    Ncalls=1547 (1547 total)    |
| EDM = 0.000113 (Goal: 1E-05)  |            up = 1.0            |
------------------------------------------------------------------
|  Valid Min.   | Valid Param.  | Above EDM | Reached call limit |
------------------------------------------------------------------
|     True      |     True      |   False   |       False        |
------------------------------------------------------------------
| Hesse failed  |   Has cov.    | Accurate  | Pos. def. | Forced |
------------------------------------------------------------------
|     False     |     True      |   True    |   True    | False  |
------------------------------------------------------------------
```

```
------------------------------------------------------------------------------------
↪--------
|   | Name    |   Value   | Hesse Err | Minos Err- | Minos Err+ | Limit-  | Limit+  ␣
↪| Fixed |
------------------------------------------------------------------------------------
↪--------
| 0 | r.1.0.0 |    12     |    49     |            |            |  -500   |  1500   ␣
↪|        |
| 1 | r.1.1.0 |   -2.7    |    9.7    |            |            |  -500   |  1500   ␣
↪|        |
| 2 | r.1.1.1 |    0.5    |    4.2    |            |            |  -500   |  1500   ␣
↪|        |
| 3 | r.1.2.0 |    110    |    120    |            |            |  -500   |  1500   ␣
↪|        |
| 4 | r.1.2.1 |    90     |    150    |            |            |  -500   |  1500   ␣
↪|        |
| 5 | r.1.2.2 |    90     |    140    |            |            |  -500   |  1500   ␣
↪|        |
| 6 | i.1.0.0 |    -20    |    41     |            |            |  -500   |  1500   ␣
↪|        |
| 7 | i.1.1.0 |   -1.8    |   13.4    |            |            |  -500   |  1500   ␣
↪|        |
| 8 | i.1.1.1 |   0.10    |   0.10    |            |            |  -500   |  1500   ␣
↪|  yes   |
| 9 | i.1.2.0 |    -80    |    170    |            |            |  -500   |  1500   ␣
↪|        |
| 10| i.1.2.1 |    -90    |    150    |            |            |  -500   |  1500   ␣
↪|        |
```

(continues on next page)

(continued from previous page)

```
| 11| i.1.2.2 |    -90    |    140    |            |            |  -500   |  1500   ␣
↪|       |
------------------------------------------------------------------------------------
↪--------
```

```
'Bin #19:'
```

```
------------------------------------------------------------------
| FCN = -4.365E+04              |    Ncalls=1778 (1778 total)    |
| EDM = 0.000126 (Goal: 1E-05)  |            up = 1.0            |
------------------------------------------------------------------
|  Valid Min.   | Valid Param.  | Above EDM | Reached call limit |
------------------------------------------------------------------
|     True      |     True      |   False   |       False        |
------------------------------------------------------------------
| Hesse failed  |   Has cov.    | Accurate  | Pos. def. | Forced |
------------------------------------------------------------------
|     False     |     True      |   True    |   True    | False  |
------------------------------------------------------------------
```

```
------------------------------------------------------------------------------------
↪--------
|   | Name    |   Value   | Hesse Err | Minos Err- | Minos Err+ | Limit-  | Limit+  ␣
↪| Fixed |
------------------------------------------------------------------------------------
↪--------
| 0 | r.1.0.0 |     4     |    23     |            |            |  -500   |  1500   ␣
↪|       |
| 1 | r.1.1.0 |     5     |    14     |            |            |  -500   |  1500   ␣
↪|       |
| 2 | r.1.1.1 |    -10    |     13    |            |            |  -500   |  1500   ␣
↪|       |
| 3 | r.1.2.0 |    -24    |     29    |            |            |  -500   |  1500   ␣
↪|       |
| 4 | r.1.2.1 |    -5     |     31    |            |            |  -500   |  1500   ␣
↪|       |
| 5 | r.1.2.2 |    -22    |     28    |            |            |  -500   |  1500   ␣
↪|       |
| 6 | i.1.0.0 |    26     |     5     |            |            |  -500   |  1500   ␣
↪|       |
| 7 | i.1.1.0 |     3     |     3     |            |            |  -500   |  1500   ␣
↪|       |
| 8 | i.1.1.1 |   0.10    |   0.10    |            |            |  -500   |  1500   ␣
↪|  yes  |
| 9 | i.1.2.0 |    103    |     7     |            |            |  -500   |  1500   ␣
↪|       |
| 10| i.1.2.1 |   106.7   |    2.7    |            |            |  -500   |  1500   ␣
↪|       |
```

(continues on next page)

(continued from previous page)

```
| 11| i.1.2.2 |    103    |     6     |             |             |  -500   |  1500  ␣
↪|       |
------------------------------------------------------------------------------------
↪--------
```

```
'0.0% (0)Failed. '
```

*Checking the waves used (and filling waves variable for later use)*

[9]:
```
waves = amplitude.make_elm(result.values)
#waves = AmplitudeJPACfit.FitAmplitude.make_elm(result.values)
waves
```

[9]:
```
   e  l  m
0  1  0  0
1  1  1  0
2  1  1  1
3  1  2  0
4  1  2  1
5  1  2  2
```

*Filling strings with wave names (for plotting)*

[10]:
```
wave = npy.empty(len(waves),dtype="int")
string = npy.empty(len(waves),dtype="U5")
for index, w in waves.iterrows():
    #print(w["l"])
    string[index]= "{}{}{}".format(w["e"],w["l"],w["m"])
    wave[index]=w.name
```

*Getting the bin mass values and number of events in datasample for those bins*

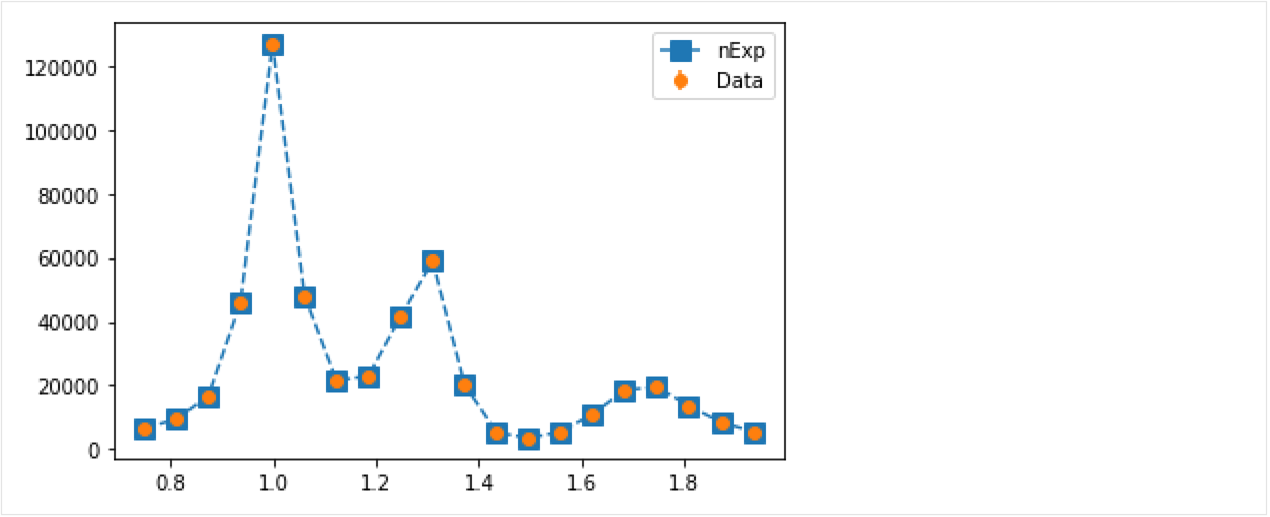
[11]:
```
bmass=[]
mcounts=[]
for index, bin in enumerate(binsda):
    if len(bin)==0:
        bmass.append(0.)
        mcounts.append(0.)
    else:
        bmass.append(npy.average(bin["mass"]))
        mcounts.append(len(bin))
```

## 6.4 Calculating the expected number of events in a a mass bin

[12]:
```
total_nExp = npy.empty(len(binsma))
for index, the_bin, result in zip(range(len(binsma)), binsma, results):
#    amp = testAmplitudeFit.FitAmplitude(initial)
#    amp = AmplitudeJPACfit.FitAmplitude(initial)
    amp=amplitude
    amp.setup(the_bin)
    total_nExp[index] = npy.average(amp.calculate(result.values))
```

*Plot expected number of events vs mass and data vs mass (both should agree if fitting worked)*

[13]:
```
import matplotlib.pyplot as plt
mni = npy.empty(len(total_nExp), dtype=[("mass", float), ("int", float)])
mda = npy.empty(len(mcounts), dtype=[("mass", float), ("intd", float)])
mni["mass"] = bmass
mni["int"] = total_nExp
mni = pandas.DataFrame(mni)
counts, bin_edges = npy.histogram(mni["mass"], nbins, weights=mni["int"])
mda["mass"] = bmass
mda["intd"] = mcounts
mda = pandas.DataFrame(mda)
dcounts, bin_edges = npy.histogram(mda["mass"], nbins, weights=mda["intd"])
#dcounts, bin_edges = npy.histogram(bmass, nbins, weights=total_nExp)
centers = (bin_edges[:-1] + bin_edges[1:]) / 2

# Add yerr to argment list when we have errors
yerr = npy.empty(nbins)
yerr = npy.sqrt(counts)
myerr = npy.empty(nbins)
myerr = npy.sqrt(dcounts)
#plt.errorbar(centers,counts, yerr, fmt="s",linestyle="dashed",markersize='10',label=
↪"nExp")
plt.errorbar(centers,total_nExp, yerr, fmt="s",linestyle="dashed",markersize='10',
↪label="nExp")
plt.errorbar(centers,mcounts, myerr, fmt="o",label="Data")
#plt.xlim(.6, 3.2)
#plt.ylim(0.,65000)
plt.legend(loc='upper right')
plt.show()
```



Calculate initial intensities (in case we need to check them)

```
[14]: intensities = []
for the_bin in binsda:
#    amp = AmplitudeJPACfit.FitAmplitude(initial)
    amp.setup(the_bin)
    intensities.append(amp.calculate(Vs))
intesities=pandas.DataFrame(intensities)
```

## 6.5 Calculate the expected number of events for each wave (vs mass)

```
[15]: wave_nExp = npy.empty([len(waves),len(binsma)],"f16")
for i in range(len(waves)):
    for index, the_bin, result in zip(range(len(binsma)), binsma, results):
#        amp = testAmplitudeFit.FitAmplitude(initial)
        amp.setup(the_bin)
        wave_nExp[i][index] = npy.average(amp.calculate_wave(result.values,wave[i]))
```

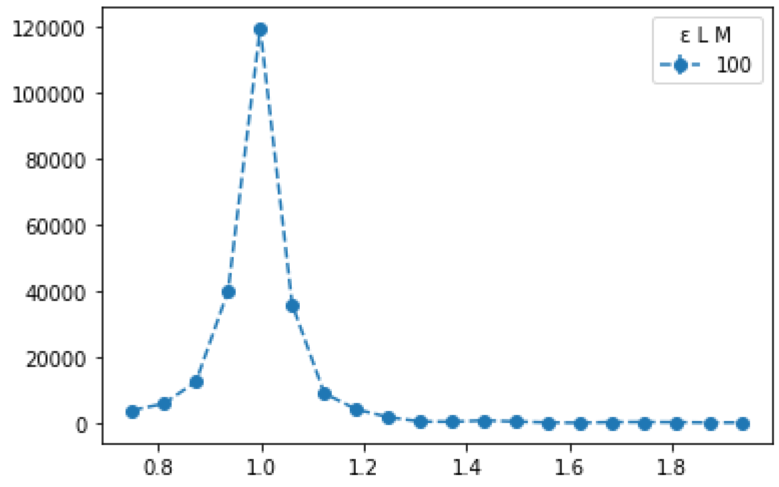
*Plot expected number of events vs mass for each wave*

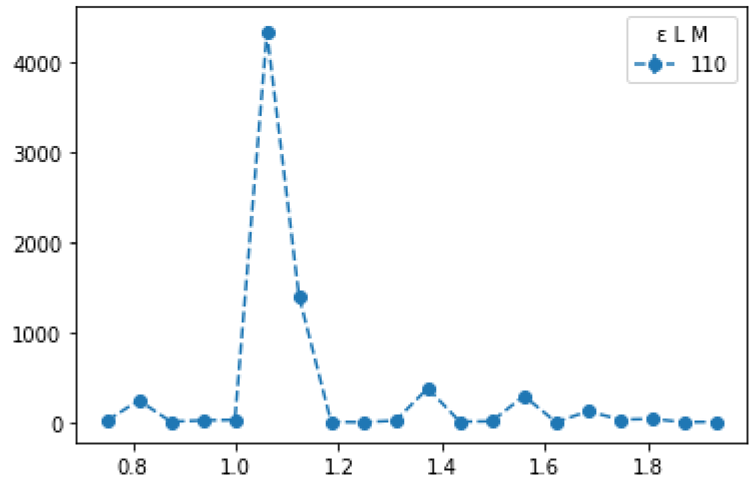
```
[16]: for i in range(len(waves)):
    mni0 = npy.empty(len(wave_nExp[i]), dtype=[("mass", float), ("int0", float)])
    mni0["mass"] = bmass
    mni0["int0"] = wave_nExp[i]
    mni0 = pandas.DataFrame(mni0)
    counts0, bin_edges = npy.histogram(mni0["mass"], nbins, weights=mni0["int0"])
    centers = (bin_edges[:-1] + bin_edges[1:]) / 2

# Add yerr to argment list when we have errors
```

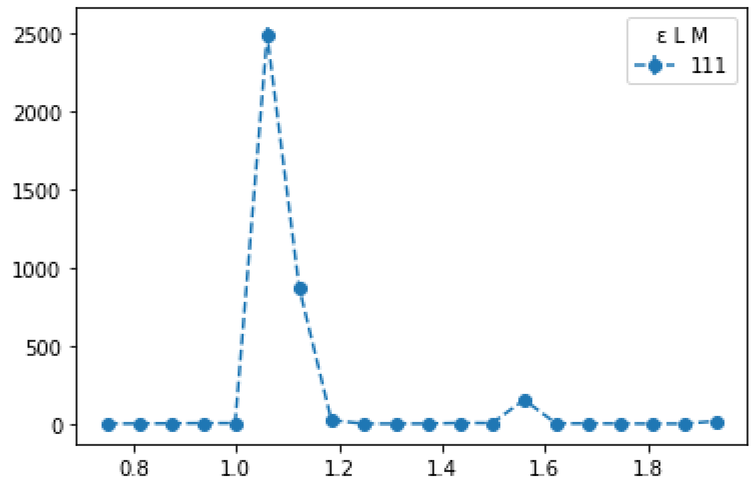
(continues on next page)

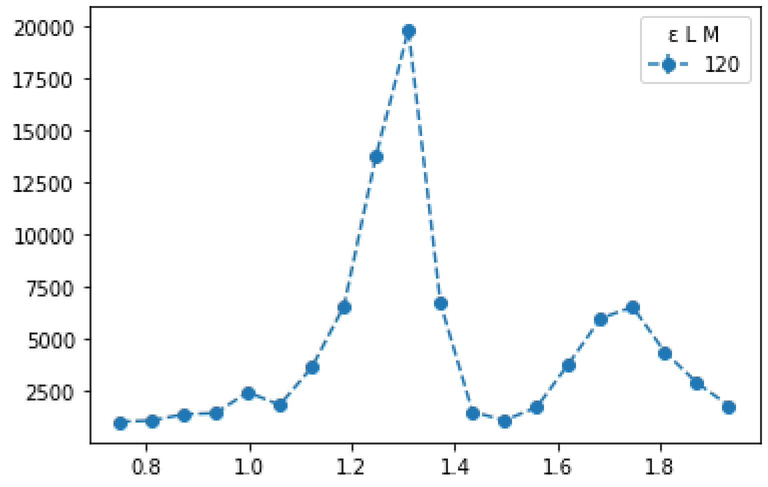
(continued from previous page)

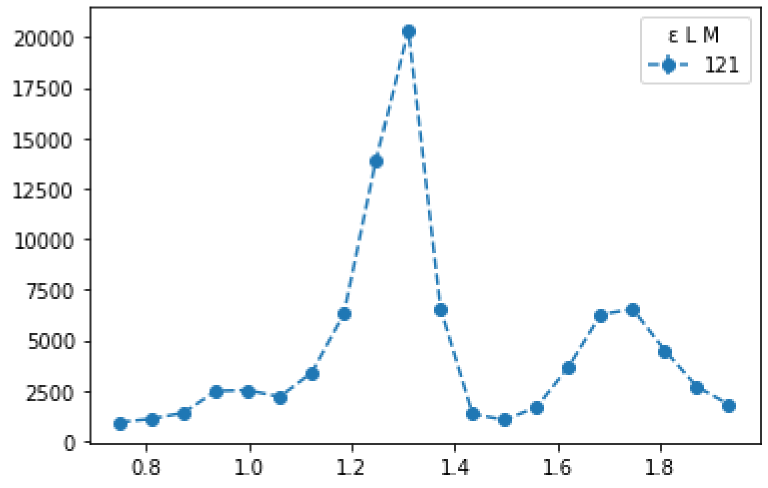
```
    yerr = npy.empty(nbins)
    yerr = npy.sqrt(counts0)
    plt.errorbar(centers,wave_nExp[i], yerr, linestyle="dashed", fmt="o",
↪label=string[i])

    plt.legend(loc='upper right', title=" L M")
    plt.show()
```

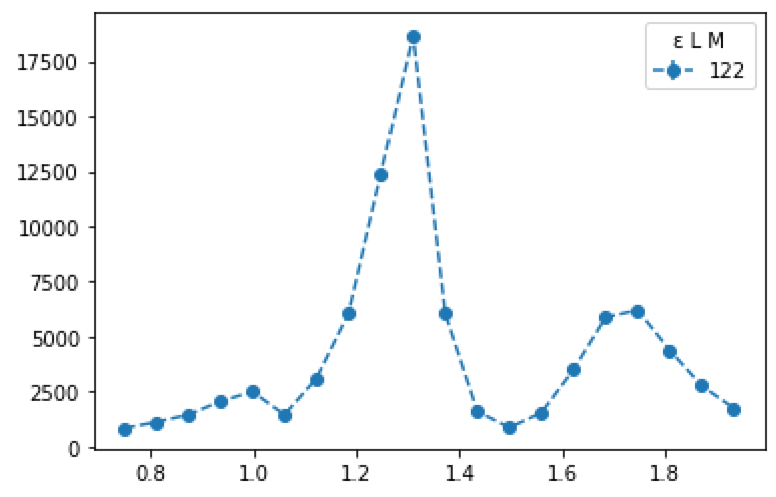





ε L M
111
2500
2000
1500
1000
500
0
0.8
1.0
1.2
1.4
1.6
1.8

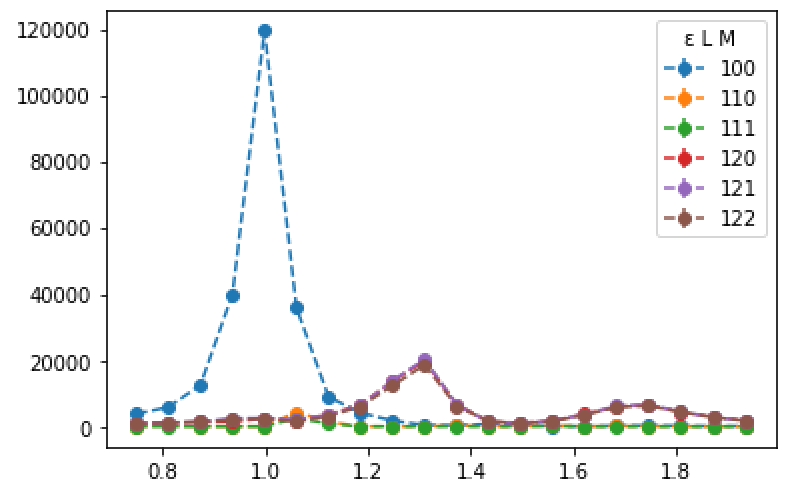

ε L M
120
20000
17500
15000
12500
10000
7500
5000
2500
0.8
1.0
1.2
1.4
1.6
1.8


ε L M
121
20000
17500
15000
12500
10000
7500
5000
2500
0
0.8
1.0
1.2
1.4
1.6
1.8



## 6.6 Expected number of events vs mass for each wave in a same plot

```
[17]: for i in range(len(waves)):
          plt.errorbar(centers,wave_nExp[i], yerr, linestyle="dashed",fmt="o",
      ↪label=string[i])
          plt.legend(loc='upper right', title=" L M")
      plt.show()
```



Calculate Moments

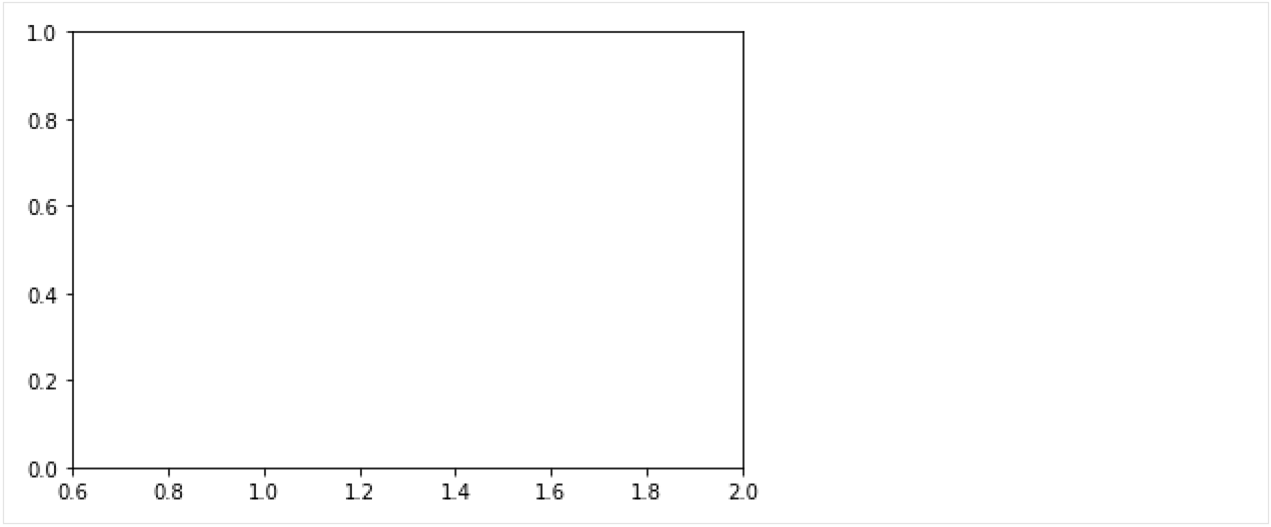
```
[18]: H000,H010,H011,H020,H021,H022,H100,H110,H111,H120,H121,H122,sigma4,sigmay = amp.
      ↪calculate_moments_JPAC(result.values)
      #H00,H11,H10,H20,H21,H22 = amp.calculate_moments_STD()
```

## 6.7 Calculate PhaseMotion between two waves

In this example between 2nd and 3rd waves in amp list

```
[21]: plt.xlim(0.6, 2.)
plt.scatter(mni["mass"],H000,LABEL="H000")
plt.scatter(mni["mass"],H100,LABEL="H100")
plt.legend(loc='upper right')
plt.show()
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-21-d1e42b4db3a2> in <module>
      1 plt.xlim(0.6, 2.)
----> 2 plt.scatter(mni["mass"],H000,LABEL="H000")
      3 plt.scatter(mni["mass"],H100,LABEL="H100")
      4 plt.legend(loc='upper right')
      5 plt.show()

~/.conda/envs/pypwa-development/lib/python3.7/site-packages/matplotlib/pyplot.py in␣
↪scatter(x, y, s, c, marker, cmap, norm, vmin, vmax, alpha, linewidths, verts,␣
↪edgecolors, plotnonfinite, data, **kwargs)
   2846         verts=verts, edgecolors=edgecolors,
   2847         plotnonfinite=plotnonfinite, **({"data": data} if data is not
-> 2848         None else {}), **kwargs)
   2849     sci(__ret)
   2850     return __ret

~/.conda/envs/pypwa-development/lib/python3.7/site-packages/matplotlib/__init__.py␣
↪in inner(ax, data, *args, **kwargs)
   1597     def inner(ax, *args, data=None, **kwargs):
   1598         if data is None:
-> 1599             return func(ax, *map(sanitize_sequence, args), **kwargs)
   1600 
   1601         bound = new_sig.bind(ax, *args, **kwargs)

~/.conda/envs/pypwa-development/lib/python3.7/site-packages/matplotlib/axes/_axes.py␣
↪in scatter(self, x, y, s, c, marker, cmap, norm, vmin, vmax, alpha, linewidths,␣
↪verts, edgecolors, plotnonfinite, **kwargs)
   4441         y = np.ma.ravel(y)
   4442         if x.size != y.size:
-> 4443             raise ValueError("x and y must be the same size")
   4444 
   4445         if s is None:

ValueError: x and y must be the same size
```

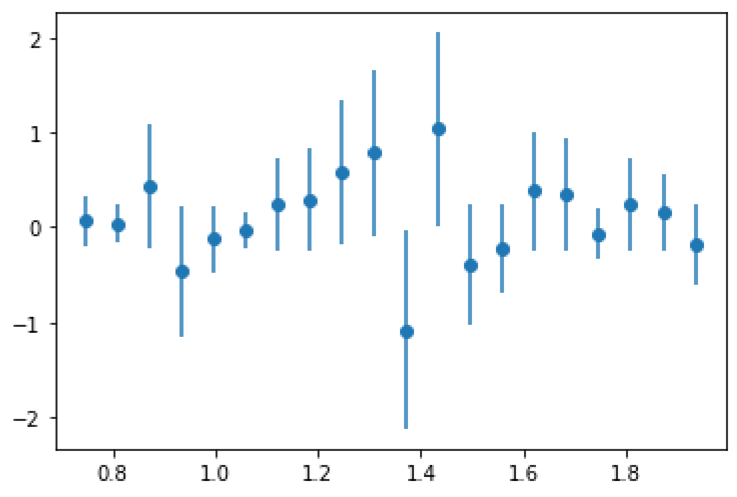


[18]:
```
phase = npy.empty(len(binsda))
for index, the_bin, result in zip(range(len(binsda)), binsda, results):
#    amp = testAmplitudeFit.FitAmplitude()
#    amp = AmplitudeJPACfit.FitAmplitude(initial)
    amp.setup(the_bin)
    phase[index] = amp.Phasediff(result.values,wave[4],wave[2])
```

*Plot PhaseMotion*

[19]:
```
mnip = npy.empty(len(bmass), dtype=[("mass", float), ("intp", float)])
mnip["mass"] = bmass
mnip["intp"] = phase
mnip = pandas.DataFrame(mnip)
countsp, bin_edges = npy.histogram(mnip["mass"], nbins, weights=mnip["intp"])
#countsp, bin_edges = npy.histogram(mnip["mass"], 25)

centers = (bin_edges[:-1] + bin_edges[1:]) / 2

# Add yerr to argment list when we have errors
yerr = npy.empty(nbins)
yerr = npy.sqrt(npy.abs(countsp))
plt.errorbar(centers,countsp, yerr, fmt="o")
#plt.xlim(.6, 2.5)
```

[19]:
```
<ErrorbarContainer object of 3 artists>
```



## 6.8 Write fitted values (of production amplitudes) to disk

```
[20]: final_values = pandas.DataFrame([res.np_values() for res in results],␣
      ↪columns=results[0].parameters)
      final_values.to_csv("final_values_JPAC.csv", index=False)
```

```
[21]: pandas.DataFrame([res.np_errors() for res in results], columns=results[0].parameters)
```

```
[21]:       r.1.0.0    r.1.1.0    r.1.1.1    r.1.2.0    r.1.2.1    r.1.2.2  \
      0    2.852249   2.454338   2.233434   2.357102   2.383402   2.626812
      1    4.143564   6.957555   2.850610   2.725406   2.577218   3.350895
      2    5.692306   4.785399   2.383390   8.044186   5.157728   3.613297
      3    0.000000   0.000000   0.000000   0.000000   0.000000   0.000000
      4   44.106133  13.751319  19.829211  10.964590  13.588421  15.587226
      5   34.436933   9.159502  11.498057   5.682668   8.133921   7.363222
      6    7.198252   8.374227   4.763529   4.189970   4.520014   3.353074
      7   11.671223   5.996570   4.279334   7.625037   5.538663  12.454837
      8    6.475163   5.382701   5.416978   8.161982   8.444590   8.678492
      9    8.721956   8.181113   8.761297  13.376127  16.434883  13.947150
      10   8.493679   6.084142   8.505738  14.310373  10.479354   6.749547
      11   3.855473   5.012773   4.741625   7.609596   3.837674   5.775665
      12   3.037836   4.442392   5.428389   3.954342   3.764583   3.667913
      13   5.321230   7.210871   6.045126   4.710401   3.813843   4.735120
      14   3.567710   3.193201   3.337286   4.090491   3.627887   4.866442
      15   6.727505   3.681380   4.365790   3.104282   5.219031   4.103600
      16   6.177552   2.676033   3.828072   3.833456   3.289779   5.726847
      17   2.933203   3.324188   3.654907   3.609211   4.131418   2.916676
      18   3.051973   3.116716   3.546187   5.042575   3.908125   5.411048
      19   2.687607   3.314947   4.027697   2.654148   4.601623   2.709061
```

(continues on next page)

(continued from previous page)

```
     i.1.0.0    i.1.1.0  i.1.1.1    i.1.2.0    i.1.2.1    i.1.2.2
0   30.355296  23.058165      0.1  29.117763  19.515329  21.461233
1   24.972032  16.099636      0.1  20.407775  15.720090  14.143814
2   35.220094  15.107411      0.1  14.095800  13.392294  16.332394
3    0.000000   0.000000      0.1   0.000000   0.000000   0.000000
4   12.307185   5.567356      0.1  10.009993   9.033432   6.326170
5   28.735040  13.816089      0.1  11.345555  17.985841  17.540164
6   17.183687  11.151904      0.1  13.529245  17.287970  16.054413
7   19.612032   6.767201      0.1  15.038266  14.425965  25.855571
8   17.661079   8.321693      0.1  17.488448  10.915935  22.474796
9   17.630661  12.990305      0.1  25.651122  14.814478  25.210855
10   8.246809   5.742596      0.1  10.345708   5.497219   9.727813
11   7.119417  12.499101      0.1   5.919945   5.846466   7.079995
12  13.051691  11.682621      0.1   9.414705   9.861358  12.532426
13  17.598563  16.636521      0.1  15.900800  14.740404  13.281171
14   8.841624   9.924517      0.1   8.551742   7.499289   9.294340
15  12.789609  16.679240      0.1  19.571438  13.145244  15.012328
16  18.829520  13.113563      0.1  20.012938  17.978662  23.420750
17  19.305438  13.300829      0.1  20.013559  11.407195  22.344017
18  18.860401  11.611010      0.1  21.188980  12.235497  23.397019
19  23.221308  25.126377      0.1  25.907302  15.642661  23.220975
```

[22]:
```
final_values
```

[22]:
```
       r.1.0.0     r.1.1.0     r.1.1.1     r.1.2.0     r.1.2.1     r.1.2.2  \
0   111.704412   60.453623   59.805884   61.078017   61.115139   62.403759
1   137.562715   64.190450   62.573028   65.067900   65.866904   70.420397
2   198.184937   64.268233   66.615670   64.595816   64.606409   69.656890
3   337.491982   72.959054   66.363838   71.371487   88.351834   80.233547
4   191.948115   72.071364  103.066780   91.422853   83.604409   72.393911
5  -229.743178   82.435298   69.243004   91.917776  101.022541  107.038100
6  -151.579375   86.849678   71.505687  129.835388  127.271735  119.766565
7   -76.655901   75.693118  103.524504  146.645818  156.565194  152.959609
8   -65.816781   86.599772  124.420103  216.389109  200.722624  222.109393
9   -26.800948   67.763534  105.828129  269.445703  216.992426  244.480875
10  -67.342887   58.718273   74.548323  111.526792   83.759368  129.187741
11  -13.410233  144.691054  144.451525   39.366624   43.680159   36.585875
12   -5.720551  162.913340  153.937952   58.902362   60.435890   57.743208
13    1.810090  165.935186  165.998990   78.621711   77.283796   79.933338
14    1.808067  165.754963  161.907458  105.701858  106.483733  107.956861
15    4.371681  144.679254  145.952516  149.573552  143.805461  152.500957
16   14.178999  130.629862  126.308624  157.756320  155.465998  155.974918
17   15.816585  114.488989  117.105987  129.755851  126.990058  125.500639
18   12.896210   97.348436   99.699072  101.426950  103.016910  102.753527
19   17.379010   80.181185   83.618056   83.784640   88.080221   85.754140
```

(continues on next page)

(continued from previous page)

```
       i.1.0.0    i.1.1.0  i.1.1.1     i.1.2.0     i.1.2.1     i.1.2.2
0     1.320825   5.792675      0.1    2.262332    4.399611   -5.139527
1    -2.994290  29.600163      0.1    6.868300    2.703924   -7.967649
2    30.953681  20.593808      0.1   45.946604   29.400564   10.318831
3   110.288478  -5.319404      0.1   -1.097682  -44.011232  -12.085719
4   585.360143   5.557063      0.1  -13.656104  -10.491047   -6.151311
5   275.792660  29.126976      0.1  -10.282243   -3.671119   -0.001849
6    79.726460  59.105635      0.1  -22.104475   31.613361   10.078527
7   -57.516859  68.733135      0.1   75.957406   47.150633   61.484341
8    13.190943  59.468474      0.1   99.732686  131.442304   84.706465
9    42.960919  39.327445      0.1  130.453626  215.905357  153.900684
10 -127.001798  -1.308107      0.1 -132.286247 -157.557282 -123.256984
11   37.882643  -3.078285      0.1   76.413219   73.946001   83.273424
12  -28.108761  11.247377      0.1  -20.433062  -25.311471  -50.581979
13  -63.243685   0.625528      0.1    4.086042  -17.541890   17.548998
14  -24.282072 -11.740948      0.1   50.413737   43.696063   41.968264
15   71.885659  14.356363      0.1   13.678425   53.303994  -21.211547
16   39.733446  -0.988772      0.1  -25.118227  -10.691878  -22.150941
17   23.952670   6.442825      0.1   20.606003   31.539561    6.359807
18   11.472671   6.887124      0.1   19.385896   17.023952   23.056549
19  -10.540275  -8.638290      0.1    5.881675  -16.019906    1.178200
```

```
[ ]:
```

CHAPTER

SEVEN

# PREDICTION TUTORIAL

Simulates "true data" corrected by acceptance, according to
the fitted values for the amplitude

[1]:
```
import PyPWA as pwa
import numpy as npy
import pandas
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
```

## 7.1 Define waves and amplitude (function) to simulate

[2]:
```
initial=[]
for param_type in ["r", "i"]:
    initial.append(f"{param_type}.1.0.0")
    initial.append(f"{param_type}.1.1.0")
    initial.append(f"{param_type}.1.1.1")
    initial.append(f"{param_type}.1.2.0")
    initial.append(f"{param_type}.1.2.1")
    initial.append(f"{param_type}.1.2.2")
#import AmplitudeOLDfit
#amp = AmplitudeOLDfit.FitAmplitude(initial)
#
import AmplitudeJPACsim
amp = AmplitudeJPACsim.NewAmplitude()
#
#import AmplitudeJPACfit
#amp = AmplitudeJPACfit.FitAmplitude(initial)
```

## 7.2 Read input (flat) simulated (generated) data

```
[3]: data = pwa.read("etapiHEL_flat.txt")
```

Read flat data in gamp format (full information)

```
[4]: datag = pwa.read("../TUTORIAL_FILES/raw_events.gamp")
```

*Define number of bins (MUST be the same as the fitted parameters)*

```
[5]: nbins=20
     bins = pwa.bin_by_range(data, "mass", nbins, .6, 2.0)
```

*Calculate the mass value and number of events in each bin*

```
[6]: bmass=[]
     mcounts=[]
     for index, bin in enumerate(bins):
         if len(bin)==0:
             bmass.append(0.)
             mcounts.append(0.)
         else:
             bmass.append(npy.average(bin["mass"]))
             mcounts.append(len(bin))
```

## 7.3 Read parameters from fit

```
[7]: par = pwa.read("final_values_JPAC.csv")
```

*Prepare for a binned simulation* > Find intensities in each bin and max intensity for each bin.

```
[8]: int_values = []
     max_values = []
     params={}
     for index, bin in enumerate(bins):
         for param_type in ["r", "i"]:
             params.update({f"{param_type}.1.0.0":par[f"{param_type}.1.0.0"][index]})
             params.update({f"{param_type}.1.1.0":par[f"{param_type}.1.1.0"][index]})
             params.update({f"{param_type}.1.1.1":par[f"{param_type}.1.1.1"][index]})
             params.update({f"{param_type}.1.2.0":par[f"{param_type}.1.2.0"][index]})
             params.update({f"{param_type}.1.2.1":par[f"{param_type}.1.2.1"][index]})
             params.update({f"{param_type}.1.2.2":par[f"{param_type}.1.2.2"][index]})


         [int,intmax] = pwa.simulate.process_user_function(amp,bin,params,16)
```

(continues on next page)

(continued from previous page)

```
    int_values.append(int)
    max_values.append(intmax)
```

## 7.4 Simulate events for each bin (produce mask and mask them)

[9]:
```
rejected_bins = []
masked_final_values = []
for int_value, bin in zip(int_values, bins):
    rejection = pwa.simulate.make_rejection_list(int_value, max_values)
    rejected_bins.append(bin[rejection])
    masked_final_values.append(int_value[rejection])
```

*Check how many events simulated by bin*

[10]:
```
for index, simulated_bin in enumerate(rejected_bins):
    print(
        f"Bin {index+1}'s length is {len(simulated_bin)}, "
        f"{(len(simulated_bin) / len(bin)) * 100:.2f}% events were kept"
    )
```

```
Bin 1's length is 4850, 1.79% events were kept
Bin 2's length is 6545, 2.42% events were kept
Bin 3's length is 10388, 3.84% events were kept
Bin 4's length is 25492, 9.42% events were kept
Bin 5's length is 65913, 24.34% events were kept
Bin 6's length is 27495, 10.15% events were kept
Bin 7's length is 15419, 5.69% events were kept
Bin 8's length is 17872, 6.60% events were kept
Bin 9's length is 31916, 11.79% events were kept
Bin 10's length is 45793, 16.91% events were kept
Bin 11's length is 20121, 7.43% events were kept
Bin 12's length is 10427, 3.85% events were kept
Bin 13's length is 10301, 3.80% events were kept
Bin 14's length is 12541, 4.63% events were kept
Bin 15's length is 15215, 5.62% events were kept
Bin 16's length is 18517, 6.84% events were kept
Bin 17's length is 17288, 6.39% events were kept
Bin 18's length is 12478, 4.61% events were kept
Bin 19's length is 8226, 3.04% events were kept
Bin 20's length is 5718, 2.11% events were kept
```

*Stak data for all bins in one file (new_data)*

[11]:
```
for index, the_bin in zip(range(len(rejected_bins)), rejected_bins):
    if index ==0:
```

(continues on next page)

(continued from previous page)

```
        new_data=pandas.DataFrame(the_bin)

    new_data =  new_data.append(the_bin,ignore_index=True)

#print(new_data)
```

[12]:
```
new_data
```

[12]:
```
          EventN     theta      phi     alpha  pol        tM      mass
0          432.0  1.075030  3.11080 -2.279760  0.4 -0.036087  0.730692
1         3277.0  1.106290  1.62386  2.912540  0.4 -0.021695  0.706537
2         5034.0  2.092900  1.77176 -1.243810  0.4 -0.065768  0.730773
3         5547.0  0.201674 -1.60246 -1.418660  0.4 -0.028844  0.701134
4         6527.0  1.325940  3.00931 -1.637510  0.4 -0.023814  0.722924
...          ...       ...      ...       ...  ...       ...       ...
387360  9849089.0  1.940630 -1.85137  1.307640  0.4 -0.521166  1.968390
387361  9851359.0  1.055120 -2.61521 -2.743190  0.4 -0.403025  1.960020
387362  9853817.0  0.189324 -1.00286 -1.850720  0.4 -0.370487  1.944500
387363  9854636.0  1.043920  2.97397 -0.604751  0.4 -0.325758  1.971750
387364  9854972.0  0.607401 -1.62378 -1.638410  0.4 -0.282602  1.978580

[387365 rows x 7 columns]
```

## 7.5 Calculate the number of expected events versus mass bins

[13]:
```
total_nExp = npy.empty(len(rejected_bins))
for index, the_bin in zip(range(len(rejected_bins)), rejected_bins):

    for param_type in ["r", "i"]:
        params.update({f"{param_type}.1.0.0":par[f"{param_type}.1.0.0"][index]})
        params.update({f"{param_type}.1.1.0":par[f"{param_type}.1.1.0"][index]})
        params.update({f"{param_type}.1.1.1":par[f"{param_type}.1.1.1"][index]})
        params.update({f"{param_type}.1.2.0":par[f"{param_type}.1.2.0"][index]})
        params.update({f"{param_type}.1.2.1":par[f"{param_type}.1.2.1"][index]})
        params.update({f"{param_type}.1.2.2":par[f"{param_type}.1.2.2"][index]})
    amp.setup(the_bin)
    total_nExp[index] = npy.average(amp.calculate(params))
```

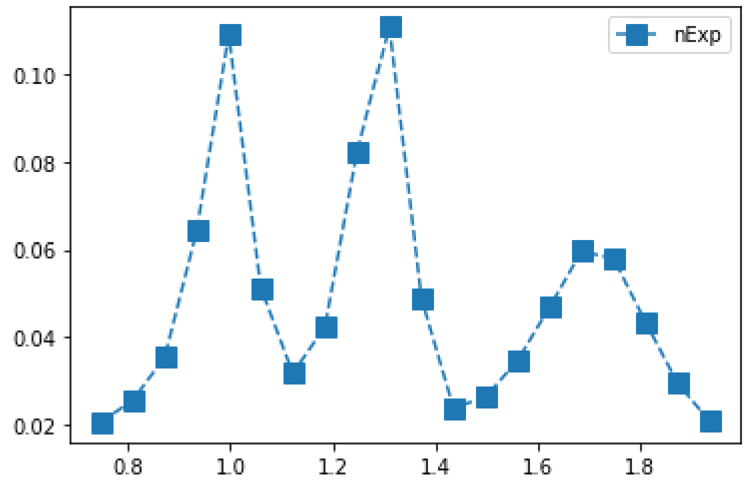
*Plot number of true (100% acc) events versus mass*

[14]:
```

mni = npy.empty(len(total_nExp), dtype=[("mass", float), ("int", float)])
mni["mass"] = bmass
mni["int"] = total_nExp
```

(continues on next page)

(continued from previous page)

```
mni = pandas.DataFrame(mni)
counts, bin_edges = npy.histogram(mni["mass"], nbins, weights=mni["int"])

centers = (bin_edges[:-1] + bin_edges[1:]) / 2

# Add yerr to argment list when we have errors
yerr = npy.empty(nbins)
yerr = npy.sqrt(counts)
yerr=0.
plt.errorbar(centers,counts, yerr, fmt="s",linestyle="dashed",markersize='10',label=
↪"nExp")

#plt.xlim(.6, 3.2)
#plt.ylim(0.,65000)
plt.legend(loc='upper right')
plt.show()
```



## 7.6 Calculate Phase difference between two waves

[15]:
```
V1=npy.ndarray(len(par))
V2=npy.ndarray(len(par))
V1 = par["r.1.1.1"]+par["i.1.1.1"]*(1j)
V2 = par["r.1.2.1"]+par["i.1.2.1"]*(1j)
if V1.all() != 0 and V2.all() != 0:
    phasediff = npy.arctan(npy.imag(V1*npy.conj(V2))/npy.real(V1*npy.conj(V2)))
```

*Plot PhaseMotion*

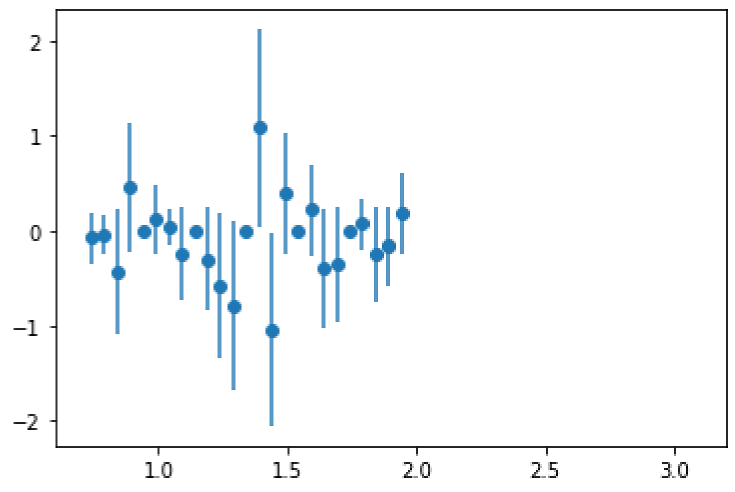
[16]:
```
mnip = npy.empty(len(bins), dtype=[("mass", float), ("phase", float)])
mnip["mass"] = bmass
```

(continues on next page)

(continued from previous page)

```
mnip["phase"] = phasediff
mnip = pandas.DataFrame(mnip)
counts, bin_edges = npy.histogram(mnip["mass"], 25, weights=mnip["phase"])
centers = (bin_edges[:-1] + bin_edges[1:]) / 2

# Add yerr to argment list when we have errors
yerr = npy.empty(100)
yerr = npy.sqrt(npy.abs(counts))
plt.errorbar(centers,counts, yerr, fmt="o")
plt.xlim(0.6, 3.2)
```

[16]:
```
(0.6, 3.2)
```



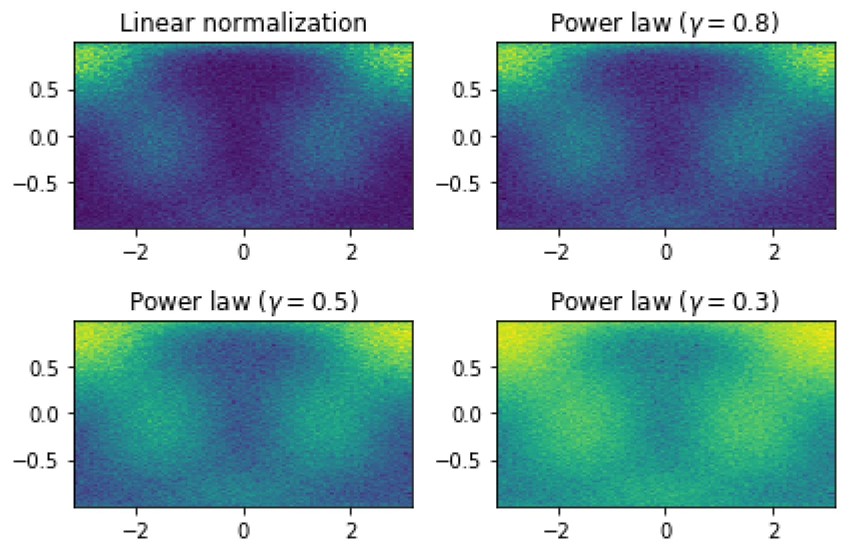
*Plot phi_HEL vs cos(theta_HEL) of predicted data (with 4 different contracts(gamma))*

[17]:
```
import matplotlib.colors as mcolors
from numpy.random import multivariate_normal

gammas = [0.8, 0.5, 0.3]

fig, axes = plt.subplots(nrows=2, ncols=2)

axes[0, 0].set_title('Linear normalization')
axes[0, 0].hist2d(new_data["phi"], npy.cos(new_data["theta"]), bins=100)
#axes[0, 0].hist2d(cut_list["phi"], npy.cos(cut_list["theta"]), bins=100)

for ax, gamma in zip(axes.flat[1:], gammas):
    ax.set_title(r'Power law $(\gamma=%1.1f)$' % gamma)
    ax.hist2d(new_data["phi"], npy.cos(new_data["theta"]),
        bins=100, norm=mcolors.PowerNorm(gamma))

fig.tight_layout()
```

(continues on next page)

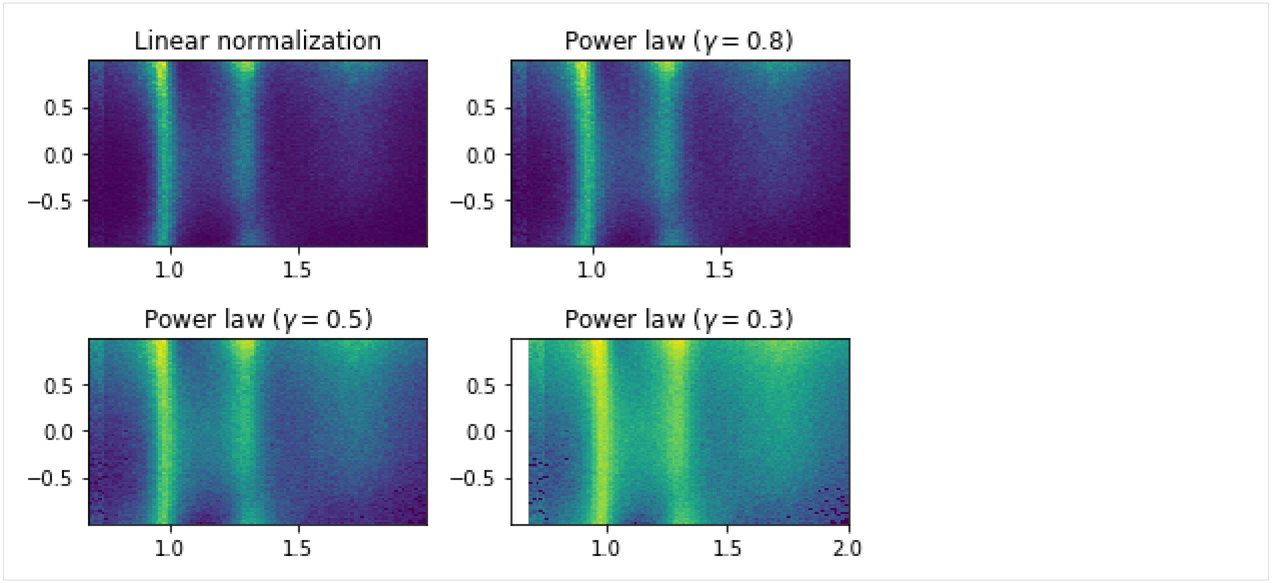
(continued from previous page)

```
plt.show()
```



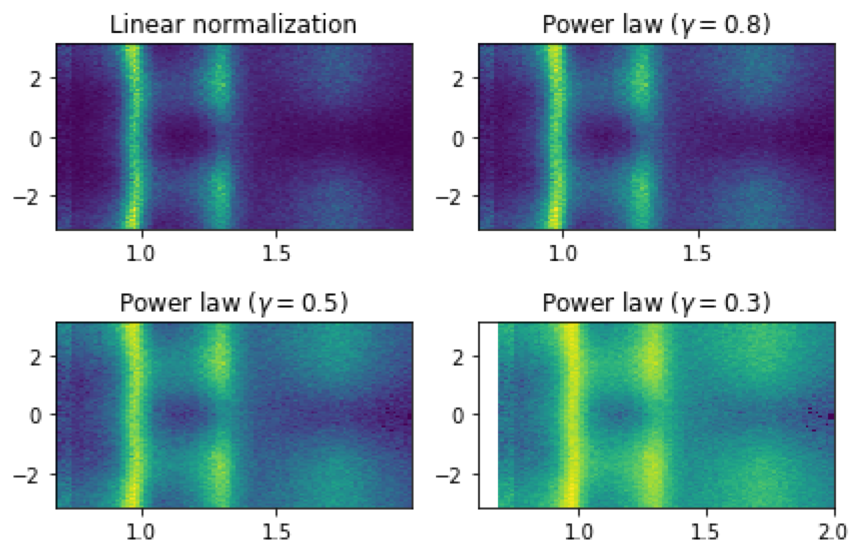
*Plot cos(theta_HEL) vs mass for simulated data (with 4 different contrasts)*

```
[18]: gammas = [0.8, 0.5, 0.3]

fig, axes = plt.subplots(nrows=2, ncols=2)

axes[0, 0].set_title('Linear normalization')
axes[0, 0].hist2d(new_data["mass"], npy.cos(new_data["theta"]),bins=100)

for ax, gamma in zip(axes.flat[1:], gammas):
    ax.set_title(r'Power law $(\gamma=%1.1f)$' % gamma)
    ax.hist2d(new_data["mass"], npy.cos(new_data["theta"]),
              bins=100, norm=mcolors.PowerNorm(gamma))


fig.tight_layout()
plt.xlim(.6, 2.)
plt.show()
```



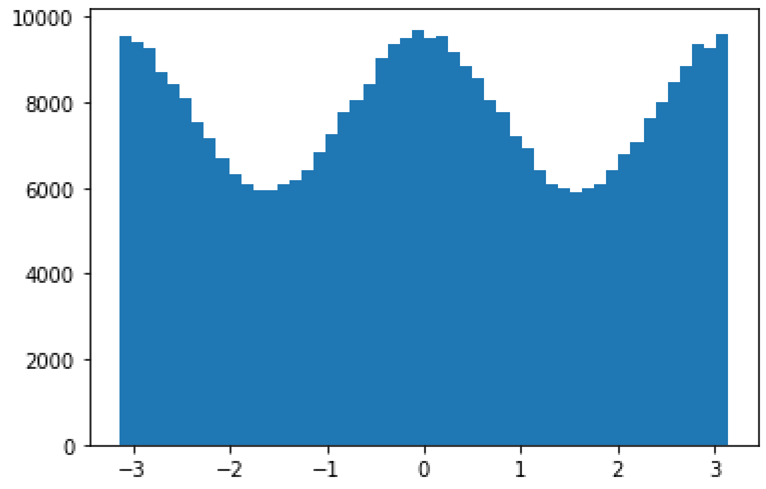
*Plot phi_HEL vs mass for simulated data (with 4 different contrasts)*

```
[19]: gammas = [0.8, 0.5, 0.3]

fig, axes = plt.subplots(nrows=2, ncols=2)

axes[0, 0].set_title('Linear normalization')
axes[0, 0].hist2d(new_data["mass"], new_data["phi"],bins=100)

for ax, gamma in zip(axes.flat[1:], gammas):
    ax.set_title(r'Power law $(\gamma=%1.1f)$' % gamma)
    ax.hist2d(new_data["mass"], new_data["phi"],
              bins=100, norm=mcolors.PowerNorm(gamma))


fig.tight_layout()
plt.xlim(.6, 2.)
plt.show()
```



```
[ ]:
```

```
[20]: plt.hist(new_data["alpha"],50)
      plt.show()
```



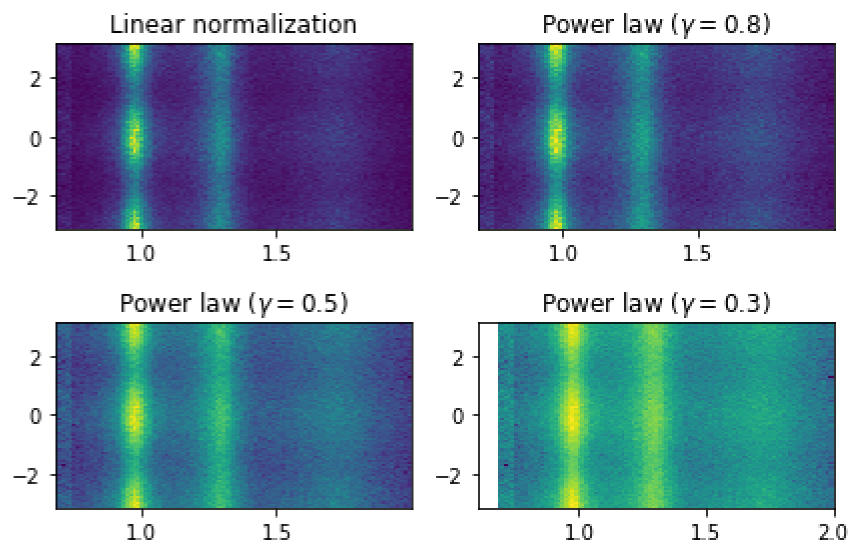
*Plot mass versus alpha/Phi*

```
[21]: gammas = [0.8, 0.5, 0.3]

      fig, axes = plt.subplots(nrows=2, ncols=2)

      axes[0, 0].set_title('Linear normalization')
      axes[0, 0].hist2d(new_data["mass"], new_data["alpha"],bins=100)

      for ax, gamma in zip(axes.flat[1:], gammas):
```

(continues on next page)

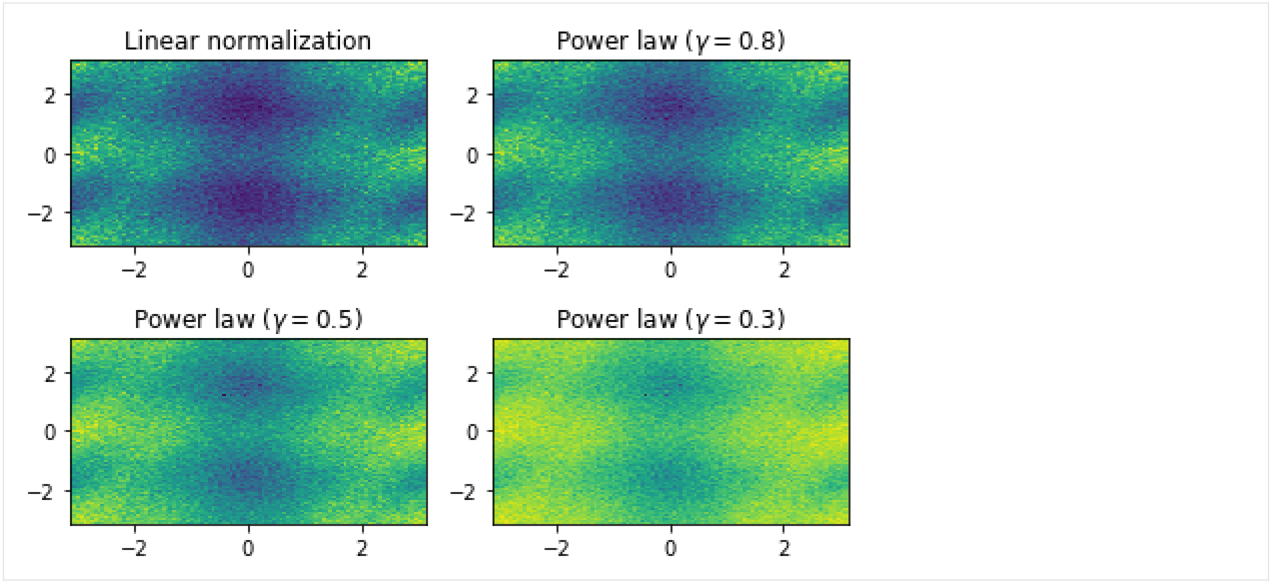
(continued from previous page)

```
    ax.set_title(r'Power law $(\gamma=%1.1f)$' % gamma)
    ax.hist2d(new_data["mass"], new_data["alpha"],
              bins=100, norm=mcolors.PowerNorm(gamma))


fig.tight_layout()
plt.xlim(.6, 2.)
plt.show()
```



*Plot mass versus alpha/Phi*

[22]:
```
gammas = [0.8, 0.5, 0.3]

fig, axes = plt.subplots(nrows=2, ncols=2)

axes[0, 0].set_title('Linear normalization')
axes[0, 0].hist2d(new_data["phi"],new_data["alpha"],bins=100)

for ax, gamma in zip(axes.flat[1:], gammas):
    ax.set_title(r'Power law $(\gamma=%1.1f) $' % gamma)
    ax.hist2d(new_data["phi"], new_data["alpha"],
              bins=100, norm=mcolors.PowerNorm(gamma))


fig.tight_layout()

plt.show()
```



## 7.7 Write predicted (true) data to disk

[23]:
```
new_data.to_csv("predictedjpac.csv", index=False)
```

Write gamp format predicted to data (true)

[21]:
```
x = (data["EventN"]).astype(int)
y = (new_data["EventN"]).astype(int)

predm = npy.isin(x,y)
pdatag = datag[predm]
pwa.write("raw_predicted.gamp",pdatag)
```

Write predicted accepted data > events.pf is a mask of accepted data that has been produced by Geant

[24]:
```
acc = pwa.read("events.pf")
accn = acc.to_numpy()

mask_acc_phy = npy.logical_and(predm,accn)
pdatag_acc = datag[mask_acc_phy]
pdata_acc = data[mask_acc_phy]

pdata_acc.to_csv("predictedJPAC_ACC.csv", index=False)
pwa.write("acc_predictedJPAC.gamp",pdatag_acc)
```

CHAPTER

EIGHT

# WORKING WITH DATA

## 8.1 Reading and Writing Data

This is the reference documentation for the functions and classes inside PyPWA that can be used for parsing and writing data to disk. There exists four different methods to do so:

- *Reading and Writing Data*
- *Basic Data Sanitization*
- *Data Iterators and Writers*
- *Working with HDF5*
- *Caching*

PyPWA also defines a vector data types and collections for working with Particles, Four Vectors, and Three Vectors, which can be found *here.*

### 8.1.1 Reading and Writing Data

Reading and writing from disk to memory. This method will load the entire dataset straight into RAM, or write a dataset straight from RAM onto disk.

`PyPWA.`**`read`**(*filename*, *use_pandas=False*, *cache=True*, *clear_cache=False*)

Reads the entire file and returns either DaataFrame, ParticlePool, or standard numpy array depending on the data found inside the file.

**Parameters**

- **`filename`** (`Path, str`) – File to read.
- **`use_pandas`** (`bool`) – Determines if a numpy data type or pandas data type is returned.
- **`cache`** (`bool, optional`) – Enables or disables caching. Defaults to the enabled. Leaving this enabled should do no harm unless there something is broken with caching. Disable this if returning the wrong data for debug purposes. If it continues to return the incorrect data when disabled then caching isn't the issue.

- **clear_cache** (`bool, optional`) – Forcefully clears the cache for the files that are parsed. Instead of loading the cache, it'll delete the cache and write a new cache object instead if cache is enabled.

**Returns**

- *DataFrame* – If the file is a kVars file, CSV, or TSV
- *npy.ndarray* – If the file is a numpy file, PF file, or single column txt file
- *ParticlePool* – If parsing a gamp file

**Raises**

**RuntimeError** – If there is no plugin that can load the data found

`PyPWA.`**write**(*filename*, *data*, *cache=True*, *clear_cache=False*)

Reads the entire file and returns either DaataFrame, ParticlePool, or standard numpy array depending on the data found inside the file.

**Parameters**

- **filename** (`Path, str`) – The filename of the file you wish to write
- **cache** (`bool, optional`) – Enables or disables caching. Defaults to the enabled. Leaving this enabled should do no harm unless there something is broken with caching.
- **clear_cache** (`bool, optional`) – Forcefully clears the cache for the files that are parsed. It'll delete the cache and write a new cache object instead when cache is enabled.

**Raises**

**RuntimeError** – If there is no plugin that can load the data found

### 8.1.2 Basic Data Sanitization

Allows quick converting of data from Pandas to Numpy, as well as preps data to be passed to non-Python function's and classes; Such as Fortran modules compiled with f2py, or C/C++ modules bound by Cython.

`PyPWA.`**pandas_to_numpy**(*df*)

Converts Pandas DataTypes to Numpy

Takes a Pandas Series or DataFrame and converts it to Numpy. Pandas does have a built in *to_records* function, however records are slower than Structured Arrays, while containing much of the same functionality.

**Parameters**

**df** (`Pandas Series or DataFrame`) – The pandas data structure that you wish to be converted to standard Numpy Structured Arrays

**Returns**

The resulting Numpy array or structured array containing the data from the original DataFrame or Series. If it was a Series with each row named (like an element from a DataFrame) it'll be a Structured Array with length=1, if it was a standard Series it'll return a single Numpy Array, and if it was a DataFrame the results will be stored in Structured array matching the types and names from the DataFrame.

**Return type**
Numpy ArrayLike

`PyPWA.`**`to_contiguous`**`(`*data*, *names*`)`

Convert DataFrame or Structured Array to List of Contiguous Arrays

This takes a data-set and a list of column names and converts those columns into Contiguous arrays. The reason to use Contiguous arrays over DataFrames or Structured arrays is that the memory is better aligned to improve speed of computation. However, this does double the memory requirements of your data-set since this copies all the events over to the new array. Use only in amplitudes where you need to maximize the speed of your amplitude.

**Parameters**

- **data** (`Structured Array, DataFrame, or Dict-like`) – This is the data frame or Structured array that you want to extract columns from
- **names** (`List of Column Names or str`) – This is either a list of columns you want from the array, or a single column you want from the array

**Returns**
If you provide only a single column, it'll only return a single array with the data from that array. However, if you have supplied multiple columns in a list or tuple, it'll return a tuple of arrays in the same order as the supplied names.

**Return type**
ArrayLike or Tuple[ArrayLike]

### 8.1.3 Data Iterators and Writers

Reading and writing a single event at a time instead of having the entire contents of the dataset memory at once. This is good choice if you are wanting to rapidly transform the data that is on disk.

**`class`** `PyPWA.`**`DataType`**`(`*value*`)`

Enumeration for type of data to be read or written using the reader and writer.

Because of how the reader and writer are designed they can not inspect the data before it starts working with the data. This enum is used to specify the type of data you're working with.

- BASIC = Standard arrays with no columns
- STRUCTURED = Columned array (CSV, TSV, DataFrames)
- TREE_VECTOR = Particle Data (GAMP)

PyPWA.**get_writer**(*filename*, *dtype*)

Returns a writer that can write to the file one event at a time

**Parameters**

- **filename** (str, Path) – The file that you want to write to
- **dtype** (DataType) – Specifies the type of that needs to be written. TREE_VECTOR is used for ParticlePools and only works with the '.gamp' extension for now. STRUCTURED_ARRAY is used for both numpy structured arrays and pandas DataFrames. BASIC is used for standard numpy arrays.

**Returns**

A writer that can read the file, defined in PyPWA.plugins.data

**Return type**

templates.WriterBase

**Raises**

**RuntimeError** – If there is no plugin that can write the data found

**See also:**

`write`

Writes a ParticlePool, DataFrame, or array to file

## Examples

The writer can be used to write a ParticlePool one event at a time

```
>>> writer = get_writer("example.gamp", DataType.TREE_VECTOR)
>>> for event in particles.iter_events():
>>>     writer.write(event)
>>> writer.close()
```

PyPWA.**get_reader**(*filename*, *use_pandas=False*)

Returns a reader that can read the file one event at a time

**Note:** The return value from the reader coule bd a pointer, if you need to keep the event without it being overwrote on the next call, you must call the copy method on the returned data to get a unique copy.

**Parameters**

- **filename** (str, Path) – File to read
- **use_pandas** (bool) – Determines if a numpy data type or pandas data type is returned.

**Returns**
A reader that can read the file, defined in PyPWA.plugins.data

**Return type**
templates.ReaderBase

**Raises**
**RuntimeError** – If there is no plugin that can load the data found

**See also:**

read
Reads an entire file into a DataFrame, ParticlePool, or array

### Examples

The reader can be used inside a standard *for* loop

```
>>> reader = get_reader("example.gamp")
>>> for event in reader:
>>>     my_kept_event = event.copy()
>>>     regular_event = event
>>> reader.close()
```

### 8.1.4 Caching

Using pickles to quickly write and read data straight from disk as intermediate caching steps. These are special functions that allow caching values or program states quickly for resuming later. This is a good way to save essential data for a Jupyter Notebook so that if the kernel is rebooted, data isn't lost.

PyPWA.cache.**read**(*path*, *intermediate=True*, *remove_cache=False*)

Reads a cache object

This reads caches objects from the disk. With its default settings it'll read the file as if it were a cache file. If intermediate is set to False, the path will be the source file, and it'll load the cache file as long as the source file's hash hasn't changed. It can also be

**Parameters**

- **path** (Path or str) – The path of the source file, or path where you want the intermediate step to be stored.
- **intermediate** (bool) – If set to true, the cache will be treated as an intermediate step, this means it will assume there is no data file associated with the data, and will not check file hashes. By default this is True
- **remove_cache** (bool) – Setting this to true will remove the cache.

**Returns**
The first value in the tuple is whether the cache is valid or not and the second value in the returned tuple is whatever data was stored in the cache.

**Return type**
Tuple[bool, any]

`PyPWA.cache.`**`write`**(*path*, *data*, *intermediate=True*)

Writes a cache file

With its default settings, it will treat the path as a save location for the cache as an intermediate step. If intermediate is set to false, it'll write the cache file into a computed cache location and store the source file's hash in the cache for future comparison.

**Parameters**

- **`path`** (`Path or str`) – The path of the source file, or path where you want the intermediate step t0 be stored.
- **`data`** (`Any`) – Whatever data you wish to be stored in the cache. Almost anything that can be stored in a variable, can be stored on disk.
- **`intermediate`** (`bool`) – If set to true, the cache will be treated as an intermediate step, this means it will assume there is no data file associated with the data, and will not check file hashes.

## 8.2 Binning

We provide functions that make binning data in memory an easy process, however for HDF5 a future more in-depth example and documentation will be made available.

`PyPWA.`**`bin_with_fixed_widths`**(*dataframe*, *bin_series*, *fixed_size*, *lower_cut=None*, *upper_cut=None*)

Bins a dataframe by fixed using a series in memory

Bins an input array by a fixed number of events in memory. You must put all data you want binned into the DataFrame or Structured Array before use. Each resulting bin can be further binned if you desire.

If the fixed_size does not evenly divide into the length of bin_series, the first and last bin will contain overflows.

**Parameters**

- **`dataframe`** (`DataFrame or Structured Array`) – The dataframe or numpy array that you wish to break into bins
- **`bin_series`** (`Array-like`) – Data that you want to bin by, selectable by user. Must have the same length as dataframe. If a column name is provided, that column will be used from the dataframe.
- **`fixed_size`** (`int`) – The number of events you want in each bin.

- **lower_cut** (`float, optional`) – The lower cut off for the dataset, if not provided it will be set to the smallest value in the bin_series
- **upper_cut** (`float, optional`) – The upper cut off for the dataset, if not provided will be set to the largest value in the bin_series

**Returns**
A list of array-likes that have been masked off of the input bin_series.

**Return type**
List[DataFrame or Structured Array]

**Raises**
**`ValueError`** – If the length of the input array and bin array don't match

**Warning:** This function does all binning in memory, if you are working with a large dataset that doesn't fit in memory, or if you overflow while you are binning, you must use a different binning method

**See also:**

**`PyPWA.libs.file.project`**
A numerical dataset that supports binning on disk instead of in-memory. It's slower and requires more steps to use, but should work even on memory limited systems.

### Examples

Binning a DataFrame with values x, y, and z using z to bin

```
>>> data = {
>>>     "x": npy.random.rand(1000), "y": npy.random.rand(1000),
>>>     "z": (npy.random.rand(1000) * 100) - 50
>>>    }
>>> df = pd.DataFrame(data)
>>> list(df.columns)
["x", "y", "z"]
```

This will give us a usable DataFrame, now to make a series out of z and use it to make 10 bins.

```
>>> binning = df["z"]
>>> range_bins = bin_with_fixed_widths(df, binning, 250)
>>> len(range_bins)
4
```

Each bin should have exactly 250 events in size

```
>>> lengths = []
>>> for abin in range_bins:
>>>    lengths.append(len(abin))
[250, 250, 250, 250]
```

That will give you 4 bins with exaactly the same number of events per bin, plus 2 more bins if needed.

PyPWA.**bin_by_range**(*dataframe*, *bin_series*, *number_of_bins*, *lower_cut=None*, *upper_cut=None*, *sample_size=None*)

Bins a dataframe by range using a series in memory

Bins an input array by range in memory. You must put all data you want binned into the DataFrame or Structured Array before use. Each resulting bin can be further binned if you desire.

**Parameters**

- **dataframe** (DataFrame or Structured Array) – The dataframe or numpy array that you wish to break into bins
- **bin_series** (Array-like) – Data that you want to bin by, selectable by user. Must have the same length as dataframe. If a column name is provided, that column will be used from the dataframe.
- **number_of_bins** (int) – The resulting number of bins that you would like to have.
- **lower_cut** (float, optional) – The lower cut off for the dataset, if not provided it will be set to the smallest value in the bin_series
- **upper_cut** (float, optional) – The upper cut off for the dataset, if not provided will be set to the largest value in the bin_series
- **sample_size** (int, optional) – If provided each bin will have a randomly selected number of events of length sample_size.

**Returns**
A list of array-likes that have been masked off of the input bin_series.

**Return type**
List[DataFrame or Structured Array]

**Raises**
**ValueError** – If the length of the input array and bin array don't match

**Warning:** This function does all binning in memory, if you are working with a large dataset that doesn't fit in memory, or if you overflow while you are binning, you must use a different binning method

**See also:**

**PyPWA.libs.file.project**
A numerical dataset that supports binning on disk instead of in-memory. It's slower and requires more steps to use, but should work even on memory limited systems.

### Notes

The range is selected using a simple method:

$$(max - min)/num_of_bins$$

### Examples

Binning a DataFrame with values x, y, and z using z to bin

```
>>> data = {
>>>     "x": npy.random.rand(1000), "y": npy.random.rand(1000),
>>>     "z": (npy.random.rand(1000) * 100) - 50
>>>    }
>>> df = pd.DataFrame(data)
>>> list(df.columns)
["x", "y", "z"]
```

This will give us a usable DataFrame, now to make a series out of z and use it to make 10 bins.

```
>>> binning = df["z"]
>>> range_bins = bin_by_range(df, binning, 10)
>>> len(range_bins)
10
```

That will give you 10 bins with a very close number of values per bin

PyPWA.**bin_by_list**(*data*, *bin_series*, *bin_list*)
Bins a dataframe by list of bin limits using a series in memory

Bins an input array by list of bin limits in memory. You must put all data you want binned into the DataFrame or Structured Array before use. Each resulting bin can be further binned if you desire.

**Parameters**

- **data** (`DataFrame or Structured Array`) – The dataframe or numpy array that you wish to break into bins
- **bin_series** (`Array-like`) – Data that you want to bin by, selectable by user. Must have the same length as dataframe. If a column name is provided, that column will be used from the dataframe.
- **bin_list** (`list`) – The list of bin limits used to create the bins.

**Returns**
A list of array-likes that have been masked off of the input bin_series.

**Return type**
List[DataFrame or Structured Array]

**Raises**
`ValueError` – If the length of the input array and bin array don't match

**Warning:** This function does all binning in memory, if you are working with a large dataset that doesn't fit in memory, or if you overflow while you are binning, you must use a different binning method

**See also:**

**`PyPWA.libs.file.project`**
A numerical dataset that supports binning on disk instead of in-memory. It's slower and requires more steps to use, but should work even on memory limited systems.

## Examples

Binning a DataFrame with values x, y, and z using z to bin

First create the list which defines all the bin limits >>> bin_limits = [1,3,7,10]

```
>>> dataset = {
>>>     "x": npy.random.rand(1000), "y": npy.random.rand(1000),
>>>     "z": (npy.random.rand(1000) * 100) - 50
>>>    }
>>> df = pd.DataFrame(dataset)
>>> list(df.columns)
["x", "y", "z"]
```

This will give us a usable DataFrame, now to make a series out of z and use it to make the 3 defined bins bins.

```
>>> binning = df["z"]
>>> range_bins = bin_by_list(df, binning, bin_limits)
>>> len(range_bins)
3
```

That will give you 3 bins with custom bin limits

## 8.3 Builtin Vectors

PyPWA includes support for both 3 and 4 vector classes, complete with methods to aid operating with vector data. Each vector utilizes Numpy for arrays and numerical operations.

**class** PyPWA.**ParticlePool**(*particle_list*)

Stores a collection of particles together as a list.

By default the particles are represented as their angles and mass, however internally the particles are still stored as the Four Momenta.

**display_raw**()

Display's the file

**property event_count: int**

**get_event_mass**()

**get_particles_by_id**(*particle_id*)

**get_particles_by_name**(*particle_name*)

**get_s**()

**get_t**()

**get_t_prime**()

**iter_events**()

**iter_particles**()

**property particle_count: int**

**split**(*count*)

Split's the particles in N groups.

This is required to be a method on any object that needs to be passed to the processing module.

**Parameters**
**count** (int) – How many ParticlePools to return

**Returns**
A list of particle pools that can be passed to different process groups.

**Return type**
List[*ParticlePool*]

**property stored: List[Particle]**

**class** PyPWA.**Particle**(*particle_id*, *charge*, *e*, *x=None*, *y=None*, *z=None*)

Numpy backed Particle object for vector operations inside PyPWA.

By default, Particle is represented through the particles angles and mass. However, internally the particle is stored as four momenta just as it's stored in the GAMP format.

**Parameters**

- **particle_id** (`int`) – The Particle ID, used to determine the particle's name and charge.
- **charge** (`int`) – The particle's Charge as read from the GAMP file.
- **e** (`int, npy.ndarray, float, or DataFrame`) – Can be an integer to specify size, a structured array or DataFrame with x y z and e values, a single float value, or a Series or single dimensional array, If you provide a float, series, or array, you need to provide a float for the other options as well.
- **x** (`int, npy.ndarray, float, or DataFrame, optional`) –
- **y** (`int, npy.ndarray, float, or DataFrame, optional`) –
- **z** (`int, npy.ndarray, float, or DataFrame, optional`) –

**See also:**

`FourVector`
For storing a FourVector without particle ID

`ParticlePool`
For storing a collection of particles

**property charge: int**

Immutable charge for the particle produced from the ID.

**display_raw**()

Displays the contents of the Particle as Four Momenta

**get_copy**()

Returns a deep copy of the Particle.

**Returns**
Copy of the particle.

**Return type**
*Particle*

**property id: int**

Immutable provided ID at initialization.

**property name: str**

Immutable name for the particle produced from the ID.

**split**(*count*)

Splits the Particle for distributed computing.

Will return N Particles which together will have the same number of elements as the original Particle.

**Parameters**
`count` (`int`) – The amount of Particles to produce from current particle.

**Returns**
The list of Particles

**Return type**
List[*Particle*]

**class** `PyPWA.`**FourVector**(*e*, *x=None*, *y=None*, *z=None*)

DataFrame backed FourVector object for vector operations inside PyPWA.

**Parameters**

- **e** (`int, np.ndarray, float, or DataFrame`) – Can be an integer to specify size, a structured array or DataFrame with x y z and e values, a single float value, or a Series or single dimensional array, If you provide a float, series, or array, you need to provide a float for the other options as well.
- **x** (`int, np.ndarray, float, or Series, optional`) –
- **y** (`int, np.ndarray, float, or Series, optional`) –
- **z** (`int, np.ndarray, float, or Series, optional`) –

**See also:**

`ThreeVector`
For storing a standard X, Y, Z vector

`Particle`
For storing a particle, adds support for a particle ID

**class** `PyPWA.`**ThreeVector**(*x*, *y=None*, *z=None*)

DataFrame backed ThreeVector object for vector operations inside PyPWA.

**Parameters**

- **x** (`int, npy.ndarray, float, or DataFrame`) – Can be an integer to specify size, a structured array or DataFrame with x y and z values, a single float value, or a Series or single dimensional array, If you provide a float, series, or array, you need to provide a float for the other options as well.
- **y** (`int, npy.ndarray, float, or DataFrame, optional`) –
- **z** (`int, npy.ndarray, float, or DataFrame, optional`) –

**See also:**

`FourVector`
For storing a vector with it’s energy.

CHAPTER

NINE

# SIMULATION AND FITTING

PyPWA defines both the monte carlo simulation method as well as the several likelihoods. To use these, the cost function or amplitude needs to be defined in a support object.

- *Defining an Amplitude* describes how to define a function for use with the simulation and likelihoods.
- *Simulating* describes the Monte Carlo Simulation methods.
- *Likelihoods* describes the built in likelihoods. These likelihoods also automatically distribute the fitting function across several processors.
- *Fitting* describes the built in minuit wrapper, as well as how to use the Likelihood objects with other optimizers.

## 9.1 Defining an Amplitude

Amplitudes or cost functions can be defined for using either an Object Oriented approach, or a Functional programming approach. If using pure functions for the function, wrap the calculation function and optional setup function in *PyPWA.FunctionalAmplitude*, if using the OOP approach, extend the *PyPWA.NestedFunction* abstract class when defining the amplitude.

It is assumed by both the Likelihoods and Monte Carlo that the calculate functions of either methods will return a standard numpy array of final values.

**class** `PyPWA.`**`NestedFunction`**

Interface for Amplitudes

These objects are used for calculating the users' amplitude. They're expected to be initialized by the time they are sent to the kernel, and will be deep-copied for each process. The setup will be called first to initialize data and anything else that might need to be done, and then the calculate function will be called for each call to the likelihood.

Set USE_MP to false to execute on the main thread only, this is best for when using packages like numexpr that handle multi-threading themselves.

Set USE_TORCH to calculate the likelihood using PyTorch. Assumes that all data returned from the NestedFunction will be in a Tensor.

Set USE_THREADS to calculate the likelihood using threads. This is best if the likelihood is dependent on waiting for responses from hardware or network devices; or if you are working with data that can not be forked.

Set USE_GPU to calculate the likelihood using GPU. If this is set to true, then USE_MP will be set to false, and USE_THREADS and USE_TORCH will be set to True internally. This will raise a RuntimeError if the GPU is not available.

Set DEBUG to True to disable all multiprocessing and threads, this will prevent errors from being buried in tracebacks.

> **Warning:** If you enable USE_MP and USE_THREADS, then a RuntimeError will be raised, since Multiprocessing and threads are not compatible.

**See also:**

`FunctionAmplitude`
: For using the old amplitudes with PyPWA 3

**abstract calculate**(*parameters*)

Calculates the amplitude

**Parameters**
**parameters** (`Dict[str, float]`) – The parameters sent to the process by the optimizer

**Returns**
The array of results for the amplitude, these will be summed by the likelihood. A tensor is expected when USE_TORCH is true

**Return type**
npy.ndarray, Series, or Tensor

**abstract setup**(*data*)

Sets up the amplitude for use.

This is where the data that will be used for this specific process will be passed to.

**Parameters**
**data** (`DataFrame or npy.ndarray`) – The data that will be used for calculation

**class** PyPWA.**FunctionAmplitude**(*setup*, *processing*)

Wrapper for Legacy PyPWA 2.X amplitudes

The old amplitudes were two simple functions that would be passed to the kernels, a single setup function and a calculate function. Now the amplitudes are objects. This wraps the functions and presents them as the new Amplitude object

**Parameters**

- **setup** (`Callable[[], ] function with no arguments or returns`) – The old setup function that would be used
- **processing** (`Callable[[pd.DataFrame, Dict[str, float]], float]`) – The old processing function

**See also:**

`NestedFunction`
For defining new functions

**calculate**(*parameters*)

Calculates the amplitude

**Parameters**
**parameters** (`Dict[str, float]`) – The parameters sent to the process by the optimizer

**Returns**
The array of results for the amplitude, these will be summed by the likelihood. A tensor is expected when USE_TORCH is true

**Return type**
npy.ndarray, Series, or Tensor

**setup**(*data*)

Sets up the amplitude for use.

This is where the data that will be used for this specific process will be passed to.

**Parameters**
**data** (`DataFrame or npy.ndarray`) – The data that will be used for calculation

## 9.2 Simulating

There are two choices when using the Monte Carlo Simulation method defined in PyPWA: Simulation in one pass producing the rejection list, or simulation in two passes to produce the intensities and finally the rejection list. Both methods will take advantage of SMP where available.

- If doing a single pass, just use the *PyPWA.monte_carlo_simulation* function. This will take the fitting function defined from *Defining an Amplitude* along with the data, and return a single rejection list.
- If doing two passes for more control over when the intensities and rejection list, use both *PyPWA.simulate.process_user_function* to calculate the intensity and local max value, and *PyPWA.simulate.make_rejection_list* to take the global max value and local intensity to produce the local rejection list.

`PyPWA.`**`monte_carlo_simulation`**(*amplitude*, *data*, *params=None*, *processes=2*)

Produces the rejection list This takes a user defined intensity object along with it's associated data, and generates a pass/fail array to be used to mask any dataset of the same length as data.

**Parameters**

- **`amplitude`** (`Amplitude derived from AbstractAmplitude`) – A user defined amplitude or pre-made PyPWA amplitude that you wish to carve your data with.
- **`data`** (`Structured Array, DataFrame, or BaseFolder from Project`) – This is the data you want to be passed to the *setup* function of your amplitude. If you provide a Structured Array or DataFrame the entire calculation will occur in memory with the selected number of processes. If you provide a Project BaseFolder the calculation will rely entirely on the Amplitude.
- **`params`** (`Dict[str, float], optional`) – An optional dictionary of parameters that will be passed to the AbstractAmplitude's *calculate* function.
- **`processes`** (`int, optional`) – Selects the number of processes to run with, defaults to the number of processes detected through multiprocessing

**Returns**

A masking array that can be used with any DataFrame or Structured Array to cut the events to the generated shape

**Return type**

boolean npy.ndarray

**Raises**

**`ValueError`** – If the data is not understood. If you received this, check your data to ensure its a supported type

## Examples

How to cut your data with results from monte_carlo_simulation

```
>>> rejection = monte_carlo_simulation(Amplitude(), data)
>>> carved = data[rejection]
```

`PyPWA.simulate.`**`process_user_function`**(*amplitude*, *data*, *params=None*, *processes=2*)

Produces an array of values for the calculated function.

**Parameters**

- **`amplitude`** (`Amplitude derived from AbstractAmplitude`) – A user defined amplitude or pre-made PyPWA amplitude that you wish to carve your data with.
- **`data`** (`Structured Array, DataFrame, or BaseFolder from Project`) – This is the data you want to be passed to the *setup* function of your amplitude.

If you provide a Structured Array or DataFrame the entire calculation will occur in memory with the selected number of processes. If you provide a Project BaseFolder the calculation will rely entirely on the Amplitude.

- **params** (`Dict[str, float], optional`) – An optional dictionary of parameters that will be passed to the AbstractAmplitude's *calculate* function.
- **processes** (`int, optional`) – Selects the number of processes to run with, defaults to the number of processes detected through multiprocessing

**Returns**
The final values computed from the user's function and the max value computed for that dataset.

**Return type**
(float npy.ndarray, float)

**Raises**
**`ValueError`** – If the data is not understood. If you received this, check your data to ensure its a supported type

`PyPWA.simulate.`**`make_rejection_list`**`(`*intensities*, *max_value*`)`

Produces the rejection list from pre-calculated function values. Uses the values returned by process_user_function.

**Parameters**

- **intensities** (`Numpy array or Pandas Series`) – This is a single dimensional array containing the final values for the user's function.
- **max_value** (`List, Tuple, Set, nd.ndarray, or float`) – The max value for the entire dataset, or list of all the max values from each dataset. Only the largest value from the list will be used.

**Returns**
A masking array that can be used with any DataFrame or Structured Array to cut the events to the generated shape

**Return type**
boolean npy.ndarray

## 9.3 Likelihoods

PyPWA supports 3 unique likelihood types for use with either the Minuit wrapper or any optimizer that expects a function. All likelihoods have built in support for SMP when they're called, and require to be closed when no longer needed.

- *PyPWA.LogLikelihood* defines the likelihood, and works with either the standard log likelihood, the binned log likelihood, or the extended log likelihood.
- *PyPWA.ChiSquared* defines the ChiSquared method, supporting both the binned and standard ChiSquare.

- *PyPWA.EmptyLikelihood* does no post operation on the final values except sum the array and return the final sum. This allows for defining unique likelihoods that have not already been defined, fitting functions that do not require a likelihood, or using the builtin multi processing without the weight of a standard likelihood.

**class** PyPWA.**LogLikelihood**(*amplitude*, *data*, *monte_carlo=None*, *binned=None*, *quality_factor=None*, *generated_length=1*, *is_minimizer=True*, *num_of_processes=2*)

Computes the log likelihood with a given amplitude.

To use the standard log likelihood, you only need to provide data, If binned and quality factor are not provided, they will default to 1. If you wish to use the Extended Log Likelihood, you must provide monte_carlo data. The generated length will be set to the length of the monte_carlo, unless a generated length is provided.

**Parameters**

- **amplitude** (`AbstractAmplitude`) – Either an user defined amplitude, or an amplitude from PyPWA
- **data** (`DataFrame or npy.ndarray`) – Data that will be passed directly to the amplitude
- **monte_carlo** (`DataFrame or npy.ndarray, optional`) – Data that will be passed to the monte_carlo
- **binned** (`Series or npy.ndarray, optional`) – Array with bin values. This won't be used if monte_carlo is provided.
- **quality_factor** (`Series or npy.ndarray, optional`) – Array with quality factor values
- **generated_length** (`int, optional`) – The generated length of values for use with the monte_carlo, this value will default to the length of monte_carlo
- **is_minimizer** (`bool, optional`) – Specify if the final value of the likelihood should be multiplied by -1. Defaults to True.
- **num_of_processes** (`int, optional`) – How many processes to be used to calculate the amplitude. Defaults to the number of threads available on the machine. If USE_MP is set to false or this is set to zero, no extra processes will be spawned

### Notes

Standard Log-Likelihood. If not provided, $Q_f$ and binned will be set to 1:

$$L = \sum Q_f \cdot binned \cdot log(Amp(data))$$

Extended Log-Likelihood. If not provided, the Q_f will be set to 1, and generated_length will be set to len(monte_carlo)

$$L = \sum Q_f \cdot log(Amp(data)) - \frac{1}{generated\_length} \cdot \sum Amp(monte\_carlo)$$

**close**()

Closes the likelihood This needs to be called after you're done with the likelihood, UNLESS, you created the likelihood using the *with* statement

**class** PyPWA.**ChiSquared**(*amplitude, data, binned=None, event_errors=None, expected_values=None, is_minimizer=True, num_of_processes=2*)

Computes the Chi-Squared Likelihood with a given amplitude.

This likelihood supports two different types of the ChiSquared, one with binned or one with expected values.

To use the binned ChiSquared, you need to provide data and binned values, to use the expected values, you need to provide data, event_errors, and expected_values.

**Parameters**

- **amplitude** (AbstractAmplitude) – Either an user defined amplitude, or an amplitude from PyPWA
- **data** (DataFrame or npy.ndarray) – The data that will be passed directly to the amplitude
- **binned** (Series or npy.ndarray, optional) – The array of bin values, should be the same length as data
- **event_errors** (Series or npy.ndarray, optional) – The array of errors, should be the same length as data
- **expected_values** (Series or npy.ndarray, optional) – The array of expected values, should be the same length as data
- **is_minimizer** (bool, optional) – Specify if the final value of the likelihood should be multiplied by -1. Defaults to True.
- **num_of_processes** (int, optional) – How many processes to be used to calculate the amplitude. Defaults to the number of threads available on the machine. If USE_MP is set to false or this is set to zero, no extra processes will be spawned

**Raises**

**ValueError** – If binned values or expected/errors are not provided

### Notes

Binned ChiSquare:

$$\chi^2 = \frac{(Amp(data) - binned)^2}{binned}$$

Expected values:

$$\chi^2 = \frac{(Amp(data) - expected)^2}{errors}$$

**close**()

Closes the likelihood This needs to be called after you're done with the likelihood, _Unless_, you created the likelihood using the *with* statement

**class** PyPWA.**EmptyLikelihood**(*amplitude*, *data*, *num_of_processes=2*)

Provides the multiprocessing benefits of a standard likelihood without a defined likelihood.

This allows you to include a likelihood into your amplitude or to run your amplitude without a likelihood entirely.

**amplitude**

Either an user defined amplitude, or an amplitude from PyPWA

**Type**
AbstractAmplitude

**data**

The data that will be passed directly to the amplitude

**Type**
DataFrame or npy.ndarray

**num_of_processes**

How many processes to be used to calculate the amplitude. Defaults to the number of threads available on the machine. If USE_MP is set to false or this is set to zero, no extra processes will be spawned

**Type**
int, optional

**close**()

Closes the likelihood This needs to be called after you're done with the likelihood, UNLESS, you created the likelihood using the *with* statement

## 9.4 Fitting

PyPWA supplies a single wrapper around iMinuit's module. This is a convenience function to make working with Minuit's parameters easier. However, if wanting to use a different fitting function, like Scikit or Scipy, the likelihoods should work natively with them.

Most optimizers built in Python assume the data is some sort of global variable, and the function passed to them is just accepting parameters to fit against. The Likelihoods take advantage of this by wrapping the data and the defined functions a wrapper that attempts to scale the function to several processors, while providing function-like capabilities by taking advantage of Python's builtin __*call*__ magic function.

This should allow the likelihoods to work with any optimizer, as long as they're expecting a function or callable object, and as long as the parameters they pass are pickle-able.

`PyPWA.`**`minuit`**`(`*settings*, *likelihood*`)`

Optimization using iminuit

**Parameters**

- **`settings`** (`Dict[str, Any]`) – The settings to be passed to iminuit. Look into the documentation for iminuit for specifics
- **`likelihood`** (`Likelihood object from likelihoods or single function`) –

**Returns**

The minuit object after the fit has been completed.

**Return type**

iminuit.Minuit

**Note:** See Iminuit's documentation for more imformation, as it should explain the various options that can be passed to iminuit, and how to use the resulting object after a fit has been completed.

# CHAPTER TEN

# PLOTTING

As an attempt to make plotting in Python easier, we are building a plotting library that attempts to solve the more specific plotting needs when working with high energy physics. The first plotting tool we have is to reproduce ROOT's LEGO plot, but more will come in the future.

`PyPWA.`**`make_lego`**(*x_data*, *y_data*, *bins=None*, *cmap='jet'*, *ax=None*, *elev=10*, *azim=215*)

Produces a 3D Lego plot, similar to what is produced by ROOT. This is similar to a 2D Histogram, but treats x and y as x and z, and projects the occurrences into the y dimension.

**Parameters**

- **x_data** (`ndarray or Series`) – X data for the lego plot
- **y_data** (`ndarray or Series`) – Y data for the lego plot
- **bins** (`int, optional`) – Number of bins to create when making the lego plot.
- **cmap** (`str or matplotlib.colors.ListedColormap, optional`) – cmap to use when creating the lego plot. It takes either a string of the name for matplotlib, or a matplotlib cmap
- **ax** (`Axes3D, optional`) – An axes object to place the lego plot into. The axes must be an axes that supports 3d projection or it will cause the function to error.
- **elev** (`int, optional`) – Adjusts the elevation of the lego-plot
- **azim** (`int, optional`) – Adjusts the azimuth of the resulting image. It's value is a angle between 0 and 360 degrees.

**Returns**

The axes object of the plot

**Return type**

Axes3D

### Notes

If the number of bins isn't provided, it's instead calculated using one half of Sturge's Rule rounded up:

$$\lceil (1/2)(1 + 3.322 \cdot log(N_{events}) \rceil$$

# INDEX