PyPWA 3.0

by

Mark Jones

Norfolk State University – Currently a Computer Science student at ODU

July 17, 2020

Table of Contents

1. Introducing PyPWA
2. Fitting and Simulation
    - 2.1 Getting Started
    - 2.2 Fitting
    - 2.3 Simulation
3. Working with Data in Python
    - 3.1 Cache as an Intermediate Step
    - 3.2 HDF5 and ProjectData
4. Closing

Introducing PyPWA

Quick History of PyPWA

PyPWA 1.0 was a GUI controlled collection of scripts designed to run the ISOBAR model/PWA in the JLAB batch farm.

PyPWA 2.0 was a rewrite of 1.0 that was packaged using Python's build tools, installed directly into your PATH, featured an extensively plugin based design, heavy reliance on Numpy, and multiprocessing as it's brand new features.

What’s new in PyPWA 3.0

The latest version of PyPWA is an improvement on the previous PyPWA versions to increase the flexibility of the package

- Pandas, Jupyter, and HDF5 support.
- Installs directly from Anaconda Cloud, Including isobar amplitude generator: GAMP.
- Likelihood objects support all standard Python optimizers.
- Numba and Numexpr support for fast computation.
- Built-in 3 and 4 Vector objects.

What does PyPWA strive to do?

PyPWA is a collection of tools designed to allow for highly flexible fitting, simulation, and data manipulation.

Our goal is to have a set of tools that can solve any problem related to Partial Wave or Amplitude Analysis, while still being flexible enough that you as the user can pick and choose the pieces you want, without having to use any pieces you don’t.

```
import PyPWA as pwa
# You can view the
# builtin tools with:
help(pwa)
```

Getting started with PyPWA

PyPWA can be installed in both Mac OS X or Linux64 Anaconda or Miniconda installation with:

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

We currently support installing the isobar amplitude generator (GAMP) in Linux64 with:

```
$ conda install -c markjonestx pwa2000
```

For now we only support GAMP on Linux systems, though if there is enough interest we can support Mac OS X as well.

Note: From here on, the code samples will use pwa instead of PyPWA, for in-depth tutorials, visit **`https://pypwa.jlab.org`**

Fitting and Simulation

Combining other Languages with PyPWA

Arguably the most powerful feature about Python is it's ability to bind to other languages. Python can act as a "glue" between different libraries, giving access to powerful libraries in a simple syntax.

- C/C++ libraries can be binded using Cython or Boost.
- Java can be connected using Py4J or JPype.
- Fortran can be compiled native against Python using Numpy's own F2Py.

Python even supports calling shell scripts and executable through the built-in subprocess module.

Defining an Amplitude

Amplitudes that are meant to be used with PyPWA are defined as objects that extend from `pwa.NestedFunction`, and can be defined in both Python and Cython

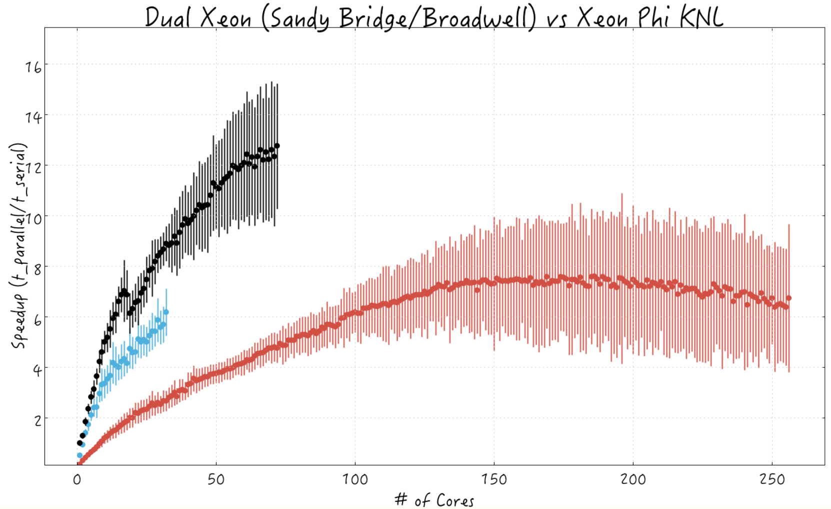
```python
class ExampleAmplitude(pwa.NestedFunction):
    USE_MP = True  # You can override built-in
                   # multiprocessing with this
    def setup(self, data):
        # Setup data and constants

    def calculate(self, params):
        # Receives parameters and returns
        # an array of the results
```

Built-in Multiprocessing

PyPWA has built-in support for accelerating Amplitudes by using all the CPU threads the hardware has. This works by leveraging Python’s Multiprocessing library, effectively creating an “Easy” Multithreading Amplitude.

- The standard system was originally designed for use with single threaded Fortran dalitz analysis that needed to scale to use all the system’s processing resources without modifying the original Fortran code.
- While it’s a reliable feature, you’re not forced to use it. You can still use Cython + OpenMP, CUDA, or whatever you desire just by setting `USE_MP=False`.
- Initial research shows that blending libraries like Numexpr and our Built-in Multiprocessing can give minor speedups over either method isolated by themselves.

Built-in Multiprocessing Cont.



Fitting

When you want to fit your amplitude, you embed your data and amplitude into a pre-existing cost object. We currently have support for both χ^2 and the Log-Likelihood.

- We have built-in support for minuit through `pwa.minuit`.
- Our cost objects support being called as a function, for use with any Python fitting library.
- The log-likelihood supports extended, binned, or standard.
- The χ^2 supports binned and unbinned.

iminuit

Short Fitting Example

```
data = pwa.read(”example.csv”)
binned = pwa.read(”binned.csv”)

with pwa.ChiSquared(amp, data, binned) as Chi:
    results = pwa.minuit(params_names, settings, amp, .5)

”””
params_names: is a list of the names of the parameters for
    your amplitude accepts.
settings: is a dictionary of settings to be passed directly
    to iminuit. Any setting you find on iminuit's documentation
    at ReadTheDocs can be passed directly here.
```

Iminuit Documantation

Simulation

We’ve included a monte carlo simulation function that returns a “pass-fail” boolean array that can be used to mask your data to produce simulated data.

```
pf = pwa.monte_carlo_simulation(amp, data, params)
simulated = data[pf]

”””
params: is the value passed to your ”calculate” method.
”””
```

The “pass-fail” array can be used to mask any array that has the same *N* elements as the “data” array.

Working with Data in Python

Data Representation in Python

All large datasets in Python are represented using two different libraries. Numpy's Arrays, or Panda's DataFrames.

- Numpy Arrays operate similarly to arrays in other languages, but come with built-in vectorization for standard operations.
- DataFrames are built on-top of Numpy arrays, but offer advanced features that make data analysis a breeze.

Working with Data

PyPWA supports loading CSV, TSV, Numpy, and GAMP. When using PyPWA to read or write your data, you're data will be returned in DataFrames or Vectors with builtin caching to accelerate future reads.

- Caching is validated using SHA512 Sums of the original file.
- Caching can be used to store intermediate values.
- You can read files using `pwa.read`, or write `pwa.write`.
- You can parse or write a single event at a time with `pwa.get_reader` or `pwa.get_writer`.
- In memory binning using `pwa.bin_with_fixed_widths` and `pwa.bin_by_range`.

Working with Data Example

```
data = pwa.read(”example.csv”)  # Caches the data for
                                # future reads

pwa.write(”example2.csv”, data)  # Will make a cache for
                                 # example2.csv on write

# You can forcefully disable the caching on reads or writes
data = pwa.read(”example.csv”, False)
```

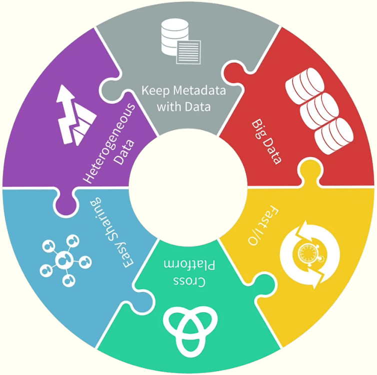
Cache as an Intermediate Step

Caching can be used without the SHA512 sums to store intermediate values that you may wish to save for future use that can be recomputed if needed.

```python
# Generating 100 random values to cache
valid, data = pwa.cache.read(”cached”, True)
if not valid:
    data = numpy.random.rand(100)
    pwa.cache.write(”cached”, data)
```

HDF5 and ProjectDataBase

HDF5 is a numerical database designed by the HDF Group designed as a portable format for heterogeneous data. It’s used to store trained models in A.I. research, and can be used with almost any numerical data.
We have support for using it with CSV or GAMP / Vector data to operate on larger-than-RAM datasets, or for multi-data file datasets for binning or portability using our `pwa.ProjectDatabase` library.



Closing

Ongoing work

Coming in future version of PyPWA 3:

- More documentation and examples.
- GPU Support.
- More built-in visualizations.
- Bootstrapping using GPUs and Batch Farm.

Research ongoing from PyPWA contributors:

- A.I. acceleration of a select amplitudes using Keras and Tensorflow.
- GPU (CUDA) acceleration for select amplitudes.

Finding Out More

- You can visit our Github at: `https://github.com/JeffersonLab/PyPWA/`
- Our base documentation at: `https://pypwa.readthedocs.io/en/master/`
- Our website at: `https://pypwa.jlab.org/`
- or you can also contact us directly on the Github Issues page, or by contacting us at salgado@jlab.org or maj@jlab.org