PyPWA Tutorial – General Shell Fitting/Simulation - Version 2

Step-by-step guide to general fitting/simulation using PyPWA

(November 2015)

M. Jones and C. Salgado
Norfolk State University
and
The Thomas Jefferson National Accelerator Facility

All software used by GeneralShell PyPWA can be downloaded from https://github.com/JeffersonLab/PyPWA/releases.

Please take a look at the documentation at https://pypwa.jlab.org/

The usual problem to be solved by this software is the following: A model for the scattering amplitude (and therefore "intensity" = amplitude x amplitude*) that dependeds on some parameters (**params[]**) and same kinematical variables (**kVars[]**) is to be fitted to the data, therefore finding the parameters.

After the fitted parameters are obtained, we "simulate data" using the fitted parameters in the intensity formula and check that "simulated data" reproduces the kinematic properties observed in the "real data".

NOTE: *It is advised that you read the general documentation in pypwa.jlab.org for more details.*

Below we detail step-by-step instructions for (fitting/simulations):

Previous Requirements (You'll need your own software for these steps):

1) Analyze your data to select the signal and crate a file with all your signal events properties, such that if the events are characterized by the variables {x1...xn}, each line of a text file will have the structure:
x1=0.33,x2=0.0456,...,xn=0.78

This file can be named **data_events.txt.**

We also allow for the use of a Q factor (i.e. the probability for each event to be a signal, ie. Q=signal/(signal+background), to be included in the fit.
Just create a file named **QFactor.txt** with the Q values of all events, one per line, written in the same order that the events are entered.

2) Run a full Monte Carlo simulation (generate+geant(detector + reconstruction + analysis) using a flat phase-space generator (you can or can't include your t distribution there or some other "weight" here).
Create two (three) files, one with the generated and another with the accepted events, formatted as the data: for example, **gen_events.txt** with the generated events and **acc_events.txt** with all the events obtained after the full simulation. You may also need (if you want to simulate data) a pass/fail file (**events.pf**) with 0 (non-accepted) and 1 (accepted) events,a line for each event (txt).

You are ready to start using the PyPWA general fitting/simulation software.

Fitting and simulating on your desktop (these will be also work in the farm, scripts for the batch farm are being developed):

FITTING

[1] Create a directory to do your work.

Download tar (version 2) and run (from https://github.com/JeffersonLab/PyPWA/):

```
 pip install  PyPWA-2.0.0b0.tar.gz
```

(this will setup your software)

running in the ifarm (without previliges to write in system /bin) you need

/u/apps/anaconda/bin/pip install PyPWA-2.0.0-Beta0.0.2.tar.gz --user --install-option="--install-scripts={**path to home folder**}/bin"

[2] run **GeneralFitting -wc**

```
it generates two files Example.py  where you define the function to be fitted.
                       Example.yml configuration file for the fits
```

[3] Edit those two files and rename them according to your analysis.
(note that if you run the simulation right after, the new configuration will overwrite your Example files so you must rename them now)

```
for example renam Example.py → test.py
                  Example.yml → test.yml
```

For example (see full example code below):

def intFn

To define the function I(x_1...x_n,a_1,...a_m) (called here intensity) that you will be fitted to the data. Each data event is defined by n variables x_n, and the function depends on m parameters a_m that will be the output of the fit .

[4] run **GeneralFitting test.yml**

This run your fits and get the Minuit results

Results are in files: **Vvalues.npy** and **minuitCov3.npy**

SIMULATION

[1] run **GeneralSimulator -wc**

```
      produces also two file Example.py
                             Example.yml
```

The Example.py may be the same file you use before (i.e. test.py) if you just are trying to simulate with the same function you have fitted.

[2] edit those two files

[3] run **GeneralSimulator Example.yml**

Running the simulator produces a file with weights (i.e. weights.txt)

[4] run **python gamMasker.py [arguments]**

```
usage: gampMasker.py [-h] [-pf ACCEPTANCE_MASK] [-pfOut ACCEPTED_OUT]
                     [-w WEIGHTED_MASK] [-wOut WEIGHTED_OUT] [-b]
                     [-bOut BOTH_OUT] [-c]
                     file

A tool for producing gamp files from phase space using pre-calculated mask files.

positional arguments:
  file                  The full filepath/name of the gamp or text file to be
                        masked.

optional arguments:
  -h, --help            show this help message and exit
  -pf ACCEPTANCE_MASK, --acceptance_mask ACCEPTANCE_MASK
                        The full filepath/name of the pf acceptance (.txt)
                        file to use.
  -pfOut ACCEPTED_OUT, --accepted_out ACCEPTED_OUT
                        The full filepath/name of the accepted output file.
  -w WEIGHTED_MASK, --weighted_mask WEIGHTED_MASK
                        The full filepath/name of the weight mask (.npy) file
                        to use.
  -wOut WEIGHTED_OUT, --weighted_out WEIGHTED_OUT
                        The full filepath/name of the weighted output file.
  -b, --both_masks      Use both the acceptance and weighted masks.
  -bOut BOTH_OUT, --both_out BOTH_OUT
                        The full filepath/name of the accepted and weighted
                        output file.
  -c, --custom_mask     Use a custom mask.
```

==
==============>> EXAMPLE---

file : test.py (with definition of function intFn)

```
import numpy,sys, os
import math

def intFn(kVars,params): #You can change both the variable names and function name
    the_size = len(kVars.values()[0]) #You can change the variable name here, or
```

```
set the length of values by hand
    values = numpy.zeros(shape=the_size) # needed
    for x in range(the_size):            #  needed


        tDist= params['A1']
        wConst = (3.0/(4.0*math.pi))
        theta = math.acos(kVars["ctAD"][x])


        WUN = wConst*(0.5*(1-0.1786)+0.5*(3*0.17861)*math.cos(theta)**2-
math.sqrt(2.0)*0.02653*math.sin(2*theta)*math.cos(kVars['phiAD'][x])
+0.03021*(math.sin(theta))**2*math.cos(2*kVars['phiAD'][x]))

        # B. Kubis paper

        z = math.sqrt(kVars['x'][x]**2+kVars['y'][x]**2)
        #phi = math.atan2(kVars['y'][x],kVars['x'][x])



        FB = 1. + 2*params['A2']*z
        #+ 2*params['A3']*(z**(1.5))*math.sin(3*phi)
      #+2*params['A4']*z**2+2*params['A5']*(z**(5/2))*math.sin(3*phi)


        PP = kVars['P'][x]
        if PP <= 0.0:
           PP = .00001

        values[x] = tDist*WUN*PP*FB
    return values

def the_setup(): #This function can be renamed, but will not be sent any
arguments.
    #This function will be ran once before the data is Minuit begins.

    Pass
```

file test.yml

```
Likelihood Information:
    Generated Length : 1000000   #Number of Generated events
    Function's Location : test.py   #The python file that has the functions in it
    Processing Name : intFn  #The name of the processing function
    Setup Name :  the_setup   #The name of the setup function, called only once
before fitting
Data Information:
    Data Location :
/volatile/clas/clasg12/salgado/Andres_omega/energybins/totaldat_events.txt #The
location of the data
```

```
    Accepted Monte Carlo Location:
/volatile/clas/clasg12/salgado/Andres_omega/Fits/simulation/FLAT_values.txt   #The
location of the Accepted Monte Carlo
    QFactor List Location :
/volatile/clas/clasg12/salgado/Andres_omega/energybins/totalQF_events.txt #The
location of the Qfactors
Minuit's Settings:
    Minuit's Initial Settings :
{'A1':1.E+4,'fix_A1':False,'error_A1':0.1,'limit_A1':[0,10.E+12],
                               'A2':0.1,'fix_A2':False,'error_A2':0.01}
                               #'A3':0.1,'fix_A3':False,'error_A3':0.01,
                               #'A4':0.01,'fix_A4':False,'error_A4':0.01,
                               #'A5':0.01,'fix_A5':False,'error_A5':0.01}

    Minuit's Parameters: [ A1, A2]   #The name of the Parameters passed to Minuit
    Minuit's Strategy : 1
    Minuit's Set Up: 0.5
    Minuit's ncall: 1000
General Settings:
    Number of Threads: 4   #Number of threads to use. Set to one for debug
    Use QFactor: True   #Boolean, using Qfactor or not
```

Then you run the fitting with:

```
                         GeneralFitting test.yml
```

you will get something like

```
------------------------------------------------------------------------
Parsing files into memory.

Loading users function.

Starting minimization.


**************************************************
*                    MIGRAD                      *
**************************************************

-713205.759386
-713294.0393
-713117.445367
-713484.412273
-712926.766456
-713784.603913
-712623.502057
-713388.851401
-713022.327294
=====================================
-939196.612064
-1024410.19375
```

```
-1119061.50277

…...

-1233198.54486
-1233198.53324
-1233198.53376
-1233198.54475
-1233198.54485
-1233197.4007
Elapsed Time: 311 sec Call: 135/1000 (2.3037 sec/call).
Est. time to maxcall: 1992.70 sec.
FCN=-1.233199e+06 FROM MIGRAD  STATUS=VALID   135 CALLS
EDM=1.765887e-06 ERROR MATRIX ACCURATE POSDEF
NO.       NAME         VALUE         ERROR  FIXED
  0         A1  1.969182e+09  1.730527e+07
  1         A2  3.009784e-01  1.148702e-02
--------------------------------------
**********************************************************************
-------------------------------------------------------------------------------------
fval = -1233198.5452729429 | total call = 135 | ncalls = 135
edm = 1.7658866337391053e-06 (Goal: 5e-06) | up = 0.5
-------------------------------------------------------------------------------------
|          Valid |    Valid Param | Accurate Covar |         Posdef |    Made Posdef |
-------------------------------------------------------------------------------------
|           True |           True |           True |           True |          False |
-------------------------------------------------------------------------------------
|     Hesse Fail |        Has Cov |      Above EDM |                |  Reach calllim |
-------------------------------------------------------------------------------------
|          False |           True |          False |             '' |          False |
-------------------------------------------------------------------------------------

------------------------------------------------------------------------------------------
--
|      | Name  |  Value   | Para Err |   Err-   |   Err+   |  Limit-  |  Limit+  |
|
------------------------------------------------------------------------------------------
--
|    0 |    A1 =  1.969E+09 |  1.731E+07 |          |          |  0       |  1E+13   |
|
|    1 |    A2 =  0.301   |  0.01149 |          |          |          |          |
|
------------------------------------------------------------------------------------------
--

**********************************************************************
```

The values of the fit are saved in Vvalues.npy and the covariant matrix in minuitCov3.npy

==

For simulation
I want to simulate events weighted by the amplitudes fitted before.
Generate a phase-space MC in .txt (same variables as you have in your fit) or gamp (4-momenta information.

File Example.yml

```
--------------------------------------------------------------
Simulator Information:
    Function's Location : test.py   #The python file that has the functions in it
    Processing Name : intFn  #The name of the processing function
    Setup Name :  the_setup   #The name of the setup function, called only once
before fitting
    Parameters : { A1: 1.969E+09, A2: 0.301}
    Number of Threads: 4
Data Information:
    Monte Carlo Location :
/volatile/clas/clasg12/salgado/Andres_omega/Fits/simulation/gen_events_small.txt
#The location of the MC
    Save Location : weights.txt #Where you want to save the weights
```

---

run   **GeneralSimulator -Example.yml**

this produce a file of weights : **weights.txt** with 0 (fail) and 1 (pass)

then we run the Masker:

for txt files:

```
python gampMasker.py -pf events.pf -pfOut pfEV.txt -w weights.txt
-wOut wnEV.txt -b -bOut bEV.txt gen_events.txt
```

three files are generated with:

pfEV.txt : generated events weighted by acceptance only
wnEV.txt : generated events weighted by the amplitude function
bEV.txt : generated events weighted vy amplitude and acceptance (" simulated data")

You can also mask a gamp file using:

```
python gampMasker.py -pf events.pf -pfOut pfEV.gamp -w weights.txt
-wOut wnEV.gamp -b -bOut bEV.txt gen_events.gamp
```