PyPWA Tutorial – PyShell Fitting/Simulation V2.2.0

(EXAMPLE BELOW BOTTOM)

Step-by-step guide to general fitting/simulation using PyFit, PySimulate and PyMask

(August 2017)

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

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

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 depends on some parameters (**params[]**) and same kinematical variables (**kVars[]**) is to be fitted to the data, therefore finding the parameters that best describe the data.

We can also simulate data given a model intensity through the Rejection MC method. One can also use the simulation to check on fits, 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):

Requirements - You'll need your own software for steps (a) and (b) – but some examples can be found in "utilities" inside the PyPWA github site):

(a) Analyze your data to select the signal and create a file with all your signal events properties needed to calculate your amplitude, 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 (I.e s,t,.u Mandelstam...) or
x1,x2,...,xn
0.33,0.0456,...,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.

(b) Run a full Monte Carlo simulation (generate+geant(detector + reconstruction + analysis) using a flat phase-space generator (you can optionally include your t distribution or some other “weighted” space).
Create two 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 “detected” data) a pass/fail file (**events_acc.pf**) with 0 for non-accepted and 1 for accepted events. The PyMask (see below) will ten select events according to this “mask” file (a line for each event in txt).

SOFTWARE INSTALLATION

If you do not have the software automatically installed by your environment (I.e in HallD or HallB) follow the installation instructions on the github page.
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 the function file defining your “model” (i.e. amplitudes and intensity). IN PYTHON.
See example below: *funInt.py*

[2] run **PyFit -wc**
it generates a configuration file where you define the function to be fitted.
See example below. See example below: *PHIconf*

[3] Edit those two files and rename them according to your analysis.
(note that if you run the simulation you can use the same function file)

def intFn
To define the function intFn(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 **PyFit PHIconf**

This run your fits and get the Minuit results

Results of the fit are in file: output.npy : parameters and errors from the fit as a covariance matrix.

Selections in configuration:
[The fit can be done using Minuit or Nestle]
[The fit can be by likelihood or chisquared]
[If you include an accepted MC it will do an extended likelihood]

[The data can be binned or unbinned]
[Quality factor can be embedded in the input or given as an independent file]

SIMULATION

[1] run **PySimulate -wc**

produces also a file **: I.e PHIsim**

The **funInt.py** may be the same file you use before, if you just are trying to simulate with the same function you have fitted.

[2] run **PySimulate PHIsim**

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

Results of the fit are in file: **output_mask.pf** as a mask to be applied to original (flat) data or any other file ("gamp" original 4-momenta) in next step.

[3] run **PyMask**
for example: PyMask -i data_events.txt -m output_mask.pf -o simdata.txt
see PyMask -h (i.e. you can use more than one mask if you have an acceptance mask).

You can also use more than one mask (if you have the acceptance for MC) the accepted generated will be:

PyMask -i gen_events.txt -m output_rejection.pf -m events_acc.pf -o acc_simdata.txt

The output events will follow the format as the output extension.
The supported format at the moment are:
.csv, .tsv, .txt, .gamp and .npy.

NOTE: You can also us PyMask to convert between file types without masking by just providing and input and output. For example:

PyMask -i gen_events.txt -o gen_events.csv

==========

====>> EXAMPLES

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

```
import numpy,sys, os
import math
import AMP

def intFn(kVars,params): #You can change both the variable names and function na
me

    Const= params['A1'] #*numpy.exp(-(params['A6']*kVars['tM']))
    wConst = (3.0/(4.0*numpy.pi))
    theta = numpy.arccos(kVars["ctAD"])

# helicity -Schilling formula
    WUN = wConst*(0.5*(1-params['A2'])+0.5*(3*params['A2']-1)*numpy.cos(theta)**
2-numpy.sqrt(2.0)*params['A3']*numpy.sin(2*theta)*numpy.cos(kVars['phiAD'])-para
ms['A4']*(numpy.sin(theta))**2*numpy.cos(2*kVars['phiAD']))

# Igor, amplitude
    fortran = numpy.zeros(shape=len(kVars['sD']), dtype=numpy.complex128)
    for x in range(len(fortran)):
        fortran[x]=AMP.amp(kVars['sD'][x],kVars['tD'][x],kVars['uD'][x],params['
A5'])

    Fsquare=fortran*numpy.conjugate(fortran)

# kinematic factor
    PP = kVars['P']
    PP[PP <= 0.0] = .00001


#    values = tDist*Fsquare
    values = Const*WUN*PP*Fsquare
    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.

#initialize Igor's staff
    AMP.dummy()
    pass
```

---

file: The configuration file (i.e. PHIconf)

```
=====
Builtin Multiprocessing:
  number of processes: 8  # Number of processes to use for calculation.
Minuit:
  strategy: 1 # The strategy of Minuit. 0 for fast, 1 default, 2 for accurate
  parameters: # The parameters used inside your settings and your function
```

```
    - A1
    - A2
    - A3
    - A4
    - A5
  settings: # The settings for iMinuit's fit. See iMinuit documentation
    A1: 2.
    fix_A1: false
    limit_A1: [0.,10000.]
    A2: 0.1
    fix_A2: false
    limit_A2: [-0.5,1.5]
    A3: 0.1
    fix_A3: false
    limit_A3: [-0.5,1.5]
    A4: 0.1
    fix_A4: false
    limit_A4: [-0.5,1.5]
    A5: 0.9
    fix_A5: false
    limit_A5: [-1.,10.]
Builtin Parser:
  enable cache: true # Enable caching of all read data.
General Fitting:
  qfactor location: QFactor.txt # The path of the qfactors file.
  function's location: funInt.py # The path of your functions file
  save name: output # The name out the output files.
  setup name: the_setup # The name of your setup function.
  processing name: intFn # The name of your processing function.
  data location: data_events.txt # The path of your data file.
  generated length: 10000 # The number of generated events
  accepted monte carlo location: acc_events.txt# The path to your accepted monte
carlo file
  likelihood type: log-likelihood # Likelihood to use: Chi-Squared, Likelihood, or
Empty
#  internal data: # Internal name mapping.
#    quality factor: Qfactors
#    binned data: BinN
```

Then you run the fitting with:

run > PyFit Phiconf

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 output.npy (numpy format) and output.txt

==========

Example of 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 PHIsim

```
----------------------------------------------------------------------
Builtin Multiprocessing:
  number of processes: 4  # Number of processes to use for calculation.
Builtin Parser:
  enable cache: true # Enable caching of all read data.
Simulator:
  function's location: funInt.py # The path to the intensity function.
  parameters: # The parameters to simulate against.
    A1: 1
    A2: .3365
    A3: -0.0131
    A4: .0058
    A5: 1.02
  data location: gen_events.txt # The path to your data.
  setup name :  the_setup   #The name of the setup function, called only once
before fitting
  processing name: intFn # The name of the intensity function.
  save name: output # The name to use for saving data.
```

---

run **> PySimulate PHIsim**

this produces a file of weights : output_rejection.pf with 0 (fail) and 1 (pass)

then we run the PyMask:

> PyMask -i gen_events.txt -m output_rejection.pf -o gen_simdata.txt

or the accepted generated will be:

> PyMask -i gen_events.txt -m output_rejection.pf -m events_acc.pf -o acc_simdata.txt