PyPWA Manual – General Shell Simulation

Step-by-step guide to simulation using PyPWA
(using the scientific JLab batch farm)
(April 2015)

J. Pond and C. Salgado
Norfolk State University
and
The Thomas Jefferson National Accelerator Facility

All software used by PyPWA can be downloaded from https://pypwa.jlab.org/. Installing PyPWA software needs full internet access i.e go to jlabs1 as ifarm1102 doesn't have internet access.

For your new installation: Create a PyPWA directory in your home directory.

Untar your software.

NOTE: *It is advised that you read the general documentation, in the wiki PyPWA, or the website before proceeding. For an overview of the general formalism consult: C. W. Salgado and D. P. Weygand, Physics Report 537 (2014) 1-58 and references within.*

These are detailed step-by-step instructions for SIMULATION using the GeneralFitting program:

You can simulate sets of events in "gamp" format (raw/generated) and accepted) according to a given production amplitude. The angular distributions of the simulated events can be then compared with your data to check your fits.

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

1) Run a full monte carlo simulation (generate + Geant(detector simulation) + Reconstruction(and analysis)) using a flat phase-space generator (you can simulate using your reaction's t distribution or other values that will not be fitted). Create a gamp formatted file: **raw_events.gamp** with all the generated events. Create a pass/fail file **events.pf** (with 0 (fail) and 1 (pass) in each line and the same order than the gamp file)**.** This file will map the accepted events after the full simulation (for weighting of the events without resubmitting new simulation).

[1] Log in to a CUE machine (e.g. ifarm1102). (Note: You may need to contact your hall's scientific computing liason to get access to the Jlab scientific computing farm!)

[2] Create a directory named as you like (for example after your reaction: e.g. Pippimpi0 – called here "MAIN")

[3] Move your files: raw_events.gamp, events.pf into that directory.
Copy also, from your home directory, PyPWA/generalShell/simulation into that directory.

Go to the generalShell/simulation

[4] The user interfaces the software **only** through one file, get your prefer editor to edit:

FnSimTemplate.py

here you find the following python functions (you need minimum knoledge of python to do this, follow the example provided below):

intFn

To define the function I(x_1...x_n,a_1,...a_m) (called here intensity) that you will be use to weight events. Each data event is defined by n variables x_n, and the function depends on m parameters a_m that will be given by the user (i.e. it could be the result of previous fit of the model to data).

kvFn

To define the list of variables,x_1...x_n to be used by intFn.

simFn

To do the simulation.
Also the number of generated events in the MC, and the pointer to different data, MC directories.

Here you also specify the input files (path directory) and the output files directories.

[5] Make your changes and save it as **FnSim.py**

The fitting is done minimizing the negative logarithmic value of an extended likelihood.

-- Run :

[6] *run* **generalSim.py i**

The simulation runs to calculate a list of intensities (iList.npy)in your desktop that will be in the directory you run the program.

[7] *find the maximun of iList (intensities)* *for example open a (i)python session:*

```
ifarm1102> ipython
Python 2.7.7 |Anaconda 2.0.1 (64-bit)| (default, Jun  2 2014, 12:34:02)
Type "copyright", "credits" or "license" for more information.

In [1]: import numpy as np

In [2]: i=np.load("iList.npy")

In [3]: print i.max()
25.8494593022

In [4]: np.save("iMax.npy",[i.max()])

In [5]:
Do you really want to exit ([y]/n)?
ifarm1102>
```

[8] *run* **generalSim.py s**

This will run the simulation with these four outputs:

```
accOUT.gamp  → weighted, accepted events
pfOUT.gamp → un-weighted accepted events
rawOUT.gamp → weighted raw events
nTrue.npy → this is the “true” value (weighted total number of events )
```

Example: FnSim.py

```
#! /u/apps/anaconda/bin/python2.7
import numpy,sys, os
import math

sys.path.append("../pythonPWA/pythonPWA")
from utilities.FourVec import FourVector

def intFn(kVars,params): # Do not change the name of this function
    """
        This is where you define your intensity function. Do not change the name
of the function. The names of the arguments are up to you, but they both need to
be dictionaries, with the first one being the kinematic variables, either from a
gamp event or a list. And the second being the fitted parameters. All fitted
parameters need to be floating point numbers. If a parameter of your function is a
complex number make the real part one fitted variable and the imaginary part
another. Your function should return a float.
    """

    tDist = params['A1']/kVars['sM'] #*numpy.exp(params['A2']*(kVars['tM']))
    wConst = (3.0/(4.0*math.pi))

      #FOR THIS EXAMPOLE ONLY TWO ANGLES ARE USED INN THE SPIN DENSITY
    theta = 0.6
    phi= 1.6

    W = wConst*(0.5*(1-params['A2'])+0.5*(3*params['A2']-1)*math.cos(theta)**2-
math.sqrt(2.0)*params['A3']*math.sin(2*theta)*math.cos(phi)-
params['A4']*(math.sin(theta))**2*math.cos(
2*phi))

    if W <= 0.0:
        print "W",W
    #if Fsquare <= 0.0:

    P = kVars['P']
    if P <= 0.0:
        P = .00001
    return tDist*W*P

def kvFn(event): # Do not change anything on this line
    """
        This is where you define the function that accepts a pythonPWA gamp event
object and returns a keyed dictionary of the kinematic variables of the event for
your intensity. This is an example of best practices for calculating the
Mandulstrum variables from a gamma P -> Pi+ Pi- Pi0 event.
    """

    for particles in event.particles:
        if particles.particleID == 14.0: #recoil proton
            p = FourVector(float(particles.particleE),
                           float(particles.particleXMomentum),
                           float(particles.particleYMomentum),
float(particles.particleZMomentum))
        if particles.particleID == 1.0: #photon gamma
            bm = FourVector(float(particles.particleE),
                           float(particles.particleXMomentum),
                           float(particles.particleYMomentum),
                           float(particles.particleZMomentum))
        if particles.particleID == 9.0: #p-
            pim = FourVector(float(particles.particleE),
                           float(particles.particleXMomentum),
                           float(particles.particleYMomentum),
                           float(particles.particleZMomentum))
        if particles.particleID == 8.0: #p+
            pip = FourVector(float(particles.particleE),
                           float(particles.particleXMomentum),
                           float(particles.particleYMomentum),
                           float(particles.particleZMomentum))
    ptarget = FourVector(.938, 0.,0.,0.)
    initp = bm + ptarget
    finalp = pip + pim + p
    pi0 = initp - finalp
    P1 = FourVector(bm.E,0.0,0.0,bm.E)
    P2 = ptarget
    P3 = pip + pim + pi0
    P4 = p
    sD= (P1+P2).dot(P1+P2)
    tD= (P1-P3).dot(P1-P3)
    uD= (P1-P4).dot(P1-P4)

    m_pion=0.139570
    m_omega=P3.dot(P3)
    stu = sD*tD*uD
    P=(stu-(m_pion**2)*(m_omega**2-m_pion**2)**2)/4.;

    return {'sD':sD,'tD':tD,'uD':uD,'P':P} #example

def simFn():
    """
        This is the function that will do the actual simulating. Fill out the
filepaths below.
    """
    #AMP.dummy()

nTrueDir="/v/volatile/clas/clasg12/salgado/Andres_omega/Fits/simulation/nTrue.npy"
#Location to save the nTrue File

inputGampDir="/v/volatile/clas/clasg12/salgado/omegaSIM/simulation/760_MeV/flat/ev
ents.gamp" #Loaction of the Generated MC Gamp File

inputPfDir="/v/volatile/clas/clasg12/salgado/omegaSIM/simulation/760_MeV/flat/even
ts.pf"#Location of the Generated acceptance PF file

outputPFGampDir="/v/volatile/clas/clasg12/salgado/Andres_omega/Fits/simulation/pfO
ut.gamp"#Location where the accepted, unweighted Gamp file will be saved

outputRawGampDir="/v/volatile/clas/clasg12/salgado/Andres_omega/Fits/simulation/ra
wOut.gamp"#Location where the unaccepted, weighted Gamp file will be saved

outputAccGampDir="/v/volatile/clas/clasg12/salgado/Andres_omega/Fits/simulation/ac
cOut.gamp"#Location where the accepted, weighted Gamp file will be saved

    iList =
numpy.load("/v/volatile/clas/clasg12/salgado/Andres_omega/Fits/simulation/iList.np
y")#Location of the list of intensities.

    iMax =
numpy.load("/v/volatile/clas/clasg12/salgado/Andres_omega/Fits/simulation/iMax.npy
")#Location of the Maximum intensity of the WHOLE fit


gS.simulate(nTrueDir,inputGampDir,outputRawGampDir,outputAccGampDir,inputPfDir,out
putPFGampDir,iList,iMax)

from generalSim import generalSim
inputGampDir="/v/volatile/clas/clasg12/salgado/omegaSIM/simulation/760_MeV/flat/ev
ents.gamp"
gS = generalSim(gampDir = inputGampDir)
if sys.argv[1] == "i":#example The first argument of numpy.save() is the filepath
of the iList to be saved.
    numpy.save("iList.npy",gS.calcIList({'A1':7.,'A2':3.0,'A3':-0.021,'A4':-
0.007,'A5':1.}))#example The first argument of numpy.save() is the filepath of the
iList to be saved.
elif sys.argv[1] == "s":
    simFn()
```