PyPWA Tutorial – General Shell Fitting

Step-by-step guide to general fitting using PyPWA
(using the scientific JLab batch farm)
(June 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.

Please take a look at the Installation session of the website (for environment requirements ...).

Create a PyPWA directory in your home directory.

Get all PyPWA software installed there.

NOTE: *It is advised that you read the general documentation.*

These are detailed step-by-step instructions for PWA (fitting):

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 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).
Create a two files, one with the generated and another with the accepted events, formatted as the data: for example, **raw_events.txt** with the generated events and **acc_events.txt** with all the events obtained after the full simulation.

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

[1] Log in to a CUE machine. (Note: You may need to contact your hall's scientific computing liaison 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: for example data_events.txt, raw_events.txt, acc_events.txt and QFactor.txt(if you have one) into that directory.
Copy PyPWA/generalShell/ from your home directory into that directory.

You can do any binning of the data/MC as needed this point. Then you will run the following fitting process on each of the bins.

Go to the generalShell/fitting

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

FnTemplate.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 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 (and into **Vvalues.npy**).

parFn

To define the list of parameters,a_1...a_m to be used by Minuit. (Minuit does not accept complex numbers, i.e. each complex number counts as two real parameters).

migFn

To do the minimization. By default it uses migrad from Minuit. If the user chooses to use Minuit here should enter initial values and other instructions to Minuit.
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): for example data_events.txt, Qf.txt and acc_events.txt.

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

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

-- **Run** :

[6] *run* **generalFitting.py**

The fit runs in your desktop and you will see each value of the likelihood
as being calculated on the scree, at the end the Minuit printout of the results.
The results are stored in numpy format at V**values.npy**

EXAMPLE: File Fn.py

```
import numpy,sys, os
from iminuit import Minuit
sys.path.append(os.path.join("/","w","work","clas","clasg12","salgado","omegaTheor
y","Code_w3pi"))
import math
import AMP

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

    W = wConst*(0.5*(1-params['A3'])+0.5*(3*params['A3']-1)*math.cos(theta)**2-
math.sqrt(2.0)*params['A4']*math.sin(2*theta)*math.cos(kVars['phiAD'])-
params['A5']*(math.sin(theta))**
2*math.cos(2*kVars['phiAD']))
    F=AMP.amp(kVars['sD'],kVars['tD'],kVars['uD'],params['A6'])
    #print "F",F
    Fsquare=F*numpy.conjugate(F)
    #if tDist <= 0.0:
    #   print "t",tDist
    if W <= 0.0:
        print "W",W
    #if Fsquare <= 0.0:
    #   print "FS",Fsquare
    #print "W= %s" % W
    P = kVars['P']
    if P <= 0.0:
        P = .00001
    return tDist*W*P*Fsquare

def parFn(A1,A3,A4,A5,A6): # Do not change the name of this function
    """
        This function will be largely the same for all instances. This only needs
to take the same arguments as the initial value dictionary defined below and
instantiate the params dictionary below and call the generalFit.calcLnLike
function below it with params as it's argument. This is so minuit can fit those
parameters with your pre defined initial values.
    """
    params = {'A1':A1,'A3':A3,'A4':A4,'A5':A5,'A6':A6}
    return gF.calcLnLike(params) # Do not change anything on this line

def migFn(): # Do not change this line
    AMP.dummy()
    kwdarg = gF.initial # or this one
    m = Minuit(parFn,**kwdarg) # or this one
    m.set_strategy(0) # set strategy as an int. 0: Fast 1: More accurate 2:
Slowest/most accurate
    m.set_up(0.5) # Leave this line alone
    m.migrad(ncall=1000) # set ncall >=1000

    Vvalues = m.values # Do not change
    numpy.save(os.path.join(".","Vvalues.npy"),Vvalues) # The first argument of
numpy.save is the file path/name you
                                                        # want to save the fitted
values to
def nTrueFn():
    from calcNTrue import calcNTrue
    genDir = "./kvArgsGen.txt"  #<------- This should be the only thing you have
to set for the
    params = numpy.load("Vvalues.npy")     #calculation of nTrue, just set it to
the generated MC file.
    nT = calcNTrue(genDir)
    nTrue = nT.calcNTrue(params)
    print nTrue

from generalFitting import generalFit # Do not change this line
dataDir="/v/volatile/clas/clasg12/salgado/Andres_omega/energybins/35data_events.tx
t"
accDir="/v/volatile/clas/clasg12/salgado/Andres_omega/energybins/35acc_events.txt"
QDir="/v/volatile/clas/clasg12/salgado/Andres_omega/energybins/35QFactor.txt"
#initial={'A1':2.7E+08,'fix_A1':True,'error_A1':0.1,'A2':3.0,'fix_A2':True,'error_
A2':0.01,'A3':0.3,'fix_A3':True,'error_A3':0.01,'A4':-
0.02,'error_A4':0.01,'fix_A4':True,'A5':-0.06,
'fix_A5':True,'error_A5':0.01,'A6':0.1,'fix_A6':False,'error_A6':0.01,'errordef':.
5}
initial={'A1':1.,'fix_A1':False,'error_A1':0.1,'limit_A1':(0,10.E+12),
'A3':0.3152,'fix_A3':False,'error_A3':0.01,'limit_A3':(-0.5,0.5),
'A4':-0.01651,'error_A4':0.01,'fix_A4':False,'limit_A4':(-0.1,0.1),
'A5':-0.02184,'fix_A5':False,'error_A5':0.01,'limit_A5':(-0.1,0.1),
'A6':0.,'fix_A6':True,'error_A6':0.1,'limit_A6':(-20.,20.),
'errordef':.5}
gF = generalFit(dataDir=dataDir,accDir=accDir,QDir=QDir,initial=initial)
```

```
"""
    In the above line do not change the names of any of the arguments but set the
values to the directories Of your data file, accepted MC file, list of Q
Factors(If you do not account for Q then leave it as an empty string("") and Q
will be set to 1.0 for all events and be inconsequential), and a dictionary of all
initial values for fitted parameters, as well as any other iMinuit fitting
aruguments. See iMinuit docs for more information.
"""
```