Jared Richards

Lamar University

September 18, 2020

Key take-away: This work is supported in, in part, by NSF grant PHY-2012826.

Student Project 1:
PDF for Dark Scalar Boson

1. Determine the PDF for fixed masses.

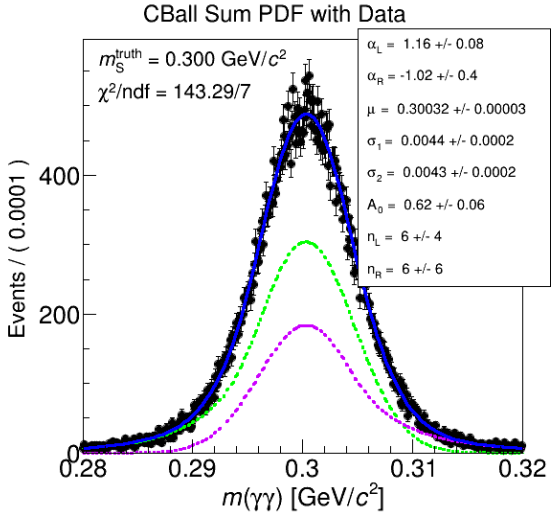
2. Parametrize the PDF w.r.t mass.

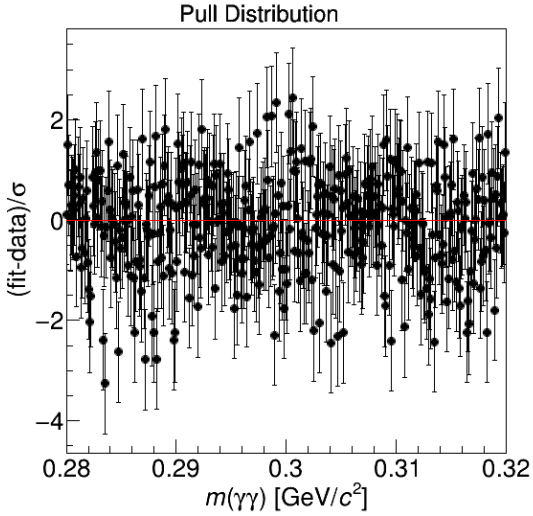
3. Conduct toy Monte Carlo simulation study to check model for introduction of biases.

Determine the PDF for fixed masses.

Mass range: $10\ \mathrm{MeV/c^2} - 410\ \mathrm{MeV/c^2}$

1) Import the histogram for a mass
2) Build a model, using 2 Crystal Ball functions
   - $\mathrm{cball1}(x;\ \mu,\ \sigma_1,\ n_1,\ \alpha_1)$
   - $\mathrm{cball2}(x;\ \mu,\ \sigma_2,\ n_2,\ \alpha_2)$
3) Fit model to Monte Carlo simulation of the signal invariant mass
4) The decision to use Crystal Ball functions is not fixed. We will try other functions.





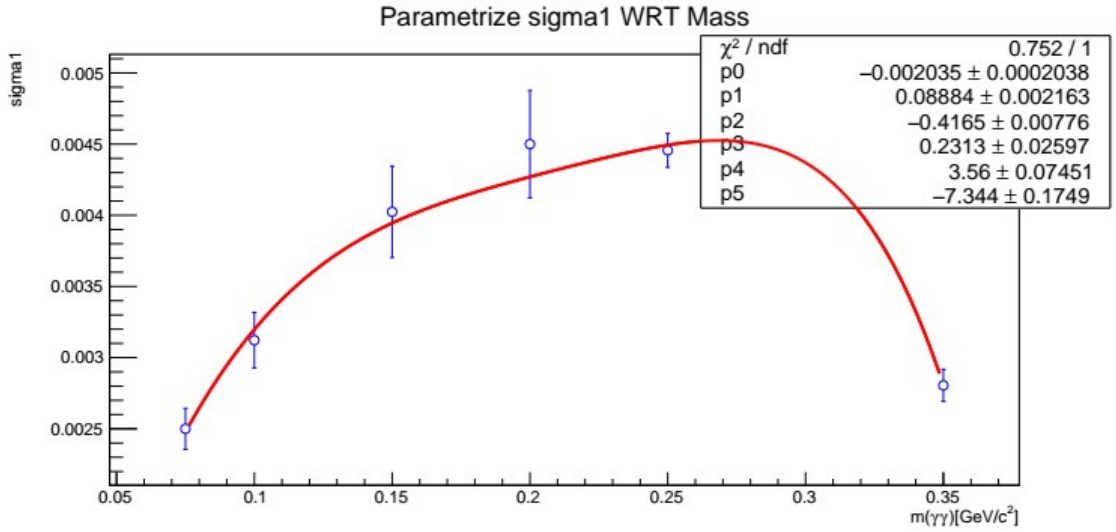
The pull distribution is good.

$$\text{Weighted Width, x} = \sqrt{0.62*4.4^2+(1-0.62)*4.3^2} = 4.36\left(\frac{MeV}{c^2}\right)$$

Key take-away: RooFit is used with all of its built-in options

Parametrize the PDF w.r.t. mass.

1) Import the parameter values and errors from .csv files
2) Fit each parameter to a polynomial
3) Notes:
   - We have parametrized the masses from $75\ \mathrm{MeV/c^2}$ to $350\ \mathrm{MeV/c^2}$ only.
   - The lesser and greater masses have different shapes, and have not yet been fitted.



Key take-away: Parametrization is working well between $75\ \mathrm{MeV/c^2}$ and $350\ \mathrm{MeV/c^2}$.

Conduct toy Monte Carlo simulation study to check model for introduction of biases.

1) Still developing ROOT macro

Key take-away: We have not yet conducted a toy Monte Carlo simulation

Macros in c++

1) **arg_fit.C**
   - Takes mass as an argument
   - Calculates PDF for that mass
2) **parametrize_sigma1.C**
   - Takes no arguments
   - Fits σ_1 to a polynomial
3) **parametrize_{other parameters}.C**

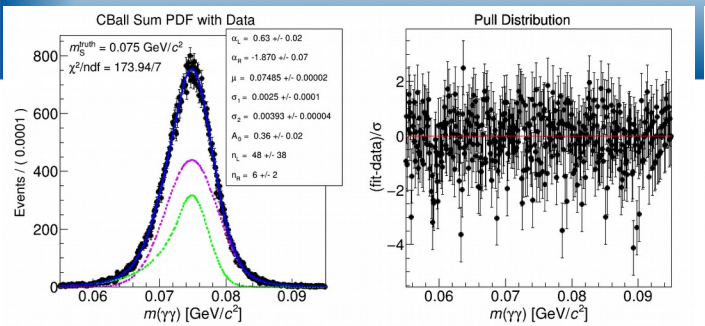
Key take-away: The code will be documented.

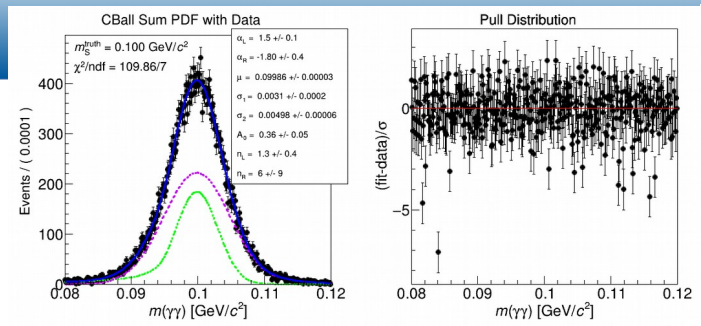
Macros in Python

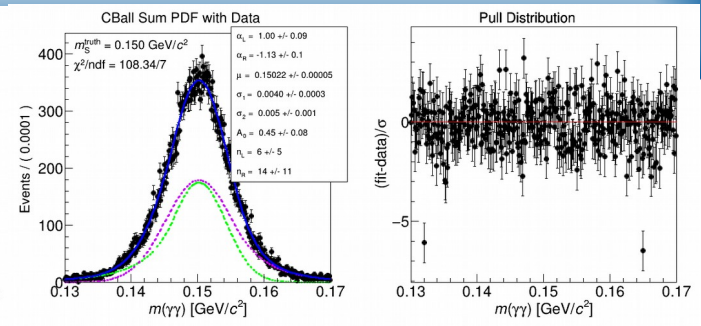
1) **fit_module.py**
   - Takes mass value and a Dictionary of parameter values as arguments
   - Calculates PDF for that mass
2) **parametrize_module.py**
   - Takes parameter as an argument
   - Fits parameter to a polynomial
3) **parameter_control_module.py**
   - Takes Dictionary of parameter values as an argument
   - Calculates new parameter values
   - Returns Dictionary of new parameter values
4) **toy_MC_study_module.py**
   - In development

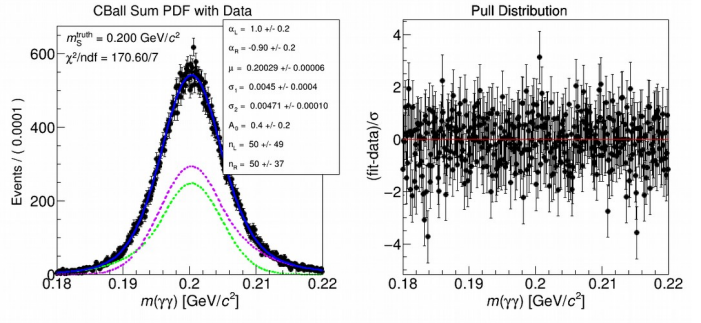
Key take-away: The code will be documented.

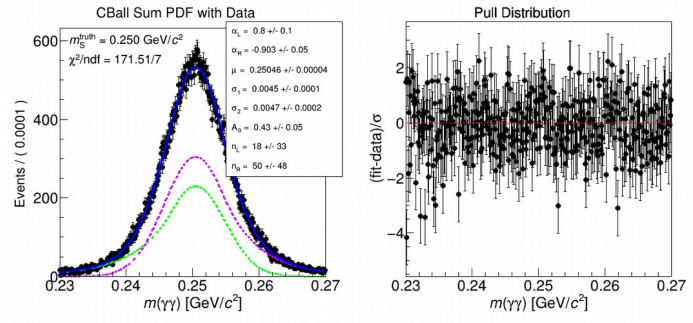
DiPhoton invariant mass after 1 iteration

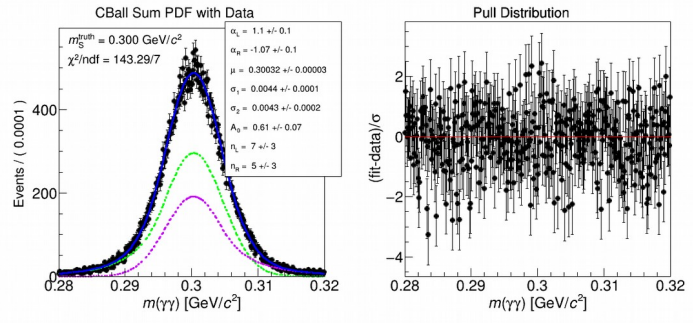


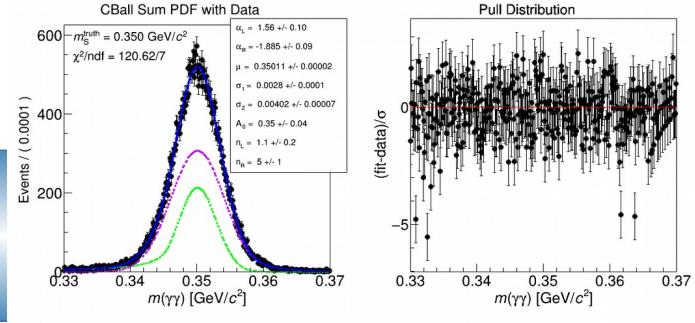










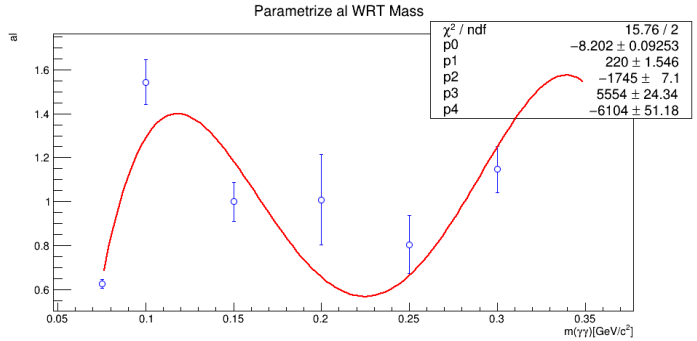


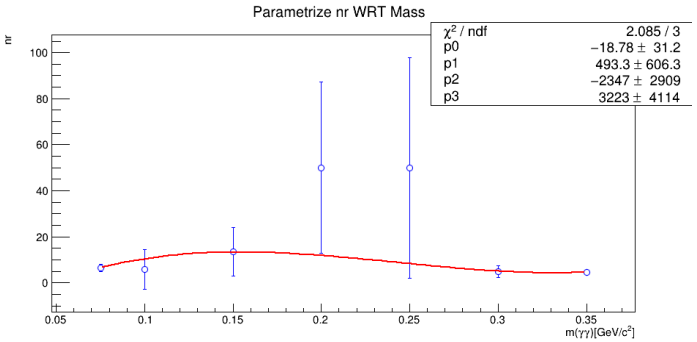
Key take-away: All of the pull distributions are good.

Smoothing Function Parameter WRT Mass

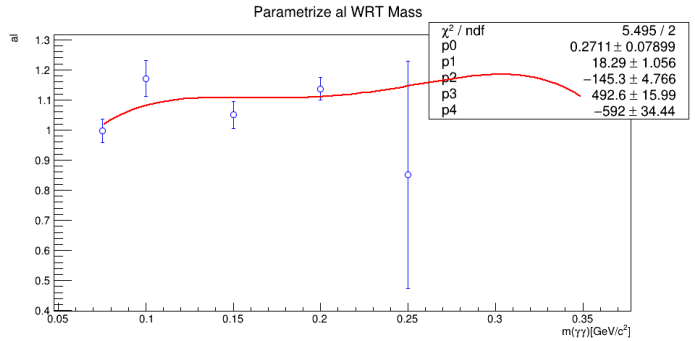
1) All function parameters were smoothed via an iterative method.
2) Two parameters are shown:
   - α_L
   - n_R

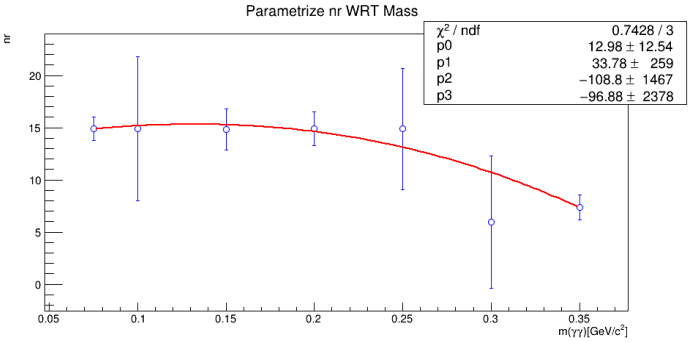
1 iteration





6 iterations





Key take-away: The iterative Parameter smoothing is working.

Iterative Smoothing Code Snippet

loop_script.py

```python
import fit_module
import parameter_control_module
import parametrize_module
# import toy_MC_study_module
import os
import glob


# Initialize the parameter values [mean, min, max, min_Factor, max_Factor]
# Example: When min_maxFactor of al = 0.4, alMin = alMean - 0.4*alMean
#          When max_maxFactor of al = 0.4, alMax = alMean + 0.4*alMean
Dict = {'al': [1.0, -3.0, 3.0, 0.2, 0.25],
        'ar': [-1.3, -3.0, 3.0, 0.25, 0.2],
        'sigma1': [0.0037, 0.0025, 0.0045, 1, 0.15],
        'sigma2': [0.0043, 0.0035, 0.005, 1, 0.15],
        'A0': [0.5, 0.1, 1.0, 0.75, 1],
        'nl': [10.0, 0.0, 50.0, 0.5, 0.25],
        'nr': [10.0, 0.0, 50.0, 0.5, 0.25]}

# Set the number of iterations: for n iterations set to range(0, n)
for iter in range(0, 6):
    # --------------------!!!!!WARNNG!!!!!----------------------
    # The next 2 lines will delete all .csv files in current directory
    for filename in glob.glob("*.csv"):
        os.remove(filename)
    for mass_index in range(3, 10):
        fit_module.fitFunc(mass_index, Dict)

    Dict = parameter_control_module.parameterControlFunc(Dict)

parametrize_module.main()

```

parameter_control_module.py

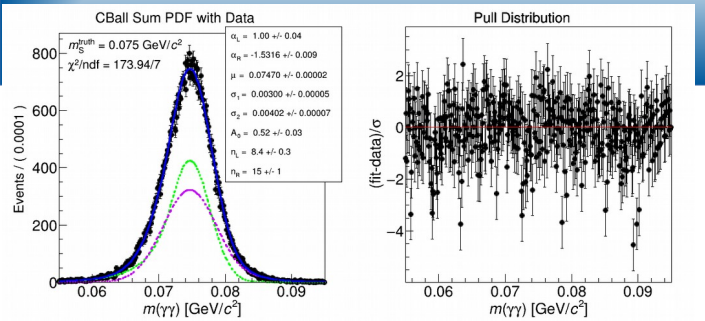
```python
import csv
from statistics import mean


def parameterControlFunc(inDictionary):
    """Reads .csv files produced by fit_module.fitFunc(), calculates
    new mean, min, and max values for parameters, stores new values
    in a dictionary, and returns the dictionary."""

    # Initialize a list for parameter indexing/zipping parameter values
    alList = []
    arList = []
    sigma1List = []
    sigma2List = []
    meanList = []
    A0List = []
    nlList = []
    nrList = []
    paramMatrix = [alList, arList, sigma1List, sigma2List,
                   meanList, A0List, nlList, nrList]

    # Read the parameter .csv files and write to the lists
    paramNameList = ["al", "ar", "sigma1", "sigma2", "mean", "A0", "nl", "nr"]
    for paramName, paramList in zip(paramNameList, paramMatrix):
        with open('{}.csv'.format(paramName)) as paramFile:
            readCSV = csv.reader(paramFile, delimiter=',')
            for row in readCSV:
                paramList.append(float(row[1]))

```

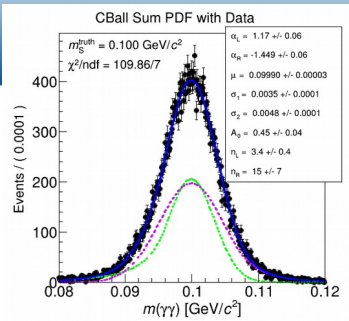
parameter_control_module.py

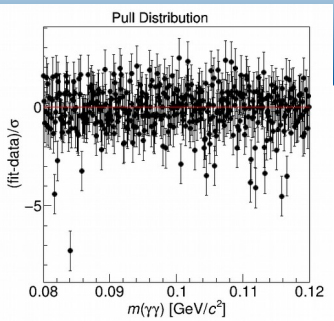
```python
    # Calculate the mean values of the newly created parameter lists
    # ---------------------------------------------------------------
    alMean = mean(alList)
    arMean = mean(arList)
    # arMean = -0.1
    sigma1Mean = mean(sigma1List)
    sigma2Mean = mean(sigma2List)
    A0Mean = mean(A0List)
    nlMean = mean(nlList)
    nrMean = mean(nrList)

    # Calculate minimum values for a range used in fit_module.fitFunc()
    # ---------------------------------------------------------------
    # Minimum as a factor of the mean
    alMin = alMean - abs(alMean) * inDictionary['al'][3]
    arMin = arMean - abs(arMean) * inDictionary['ar'][3]
    # arMin = -3.0
    sigma1Min = sigma1Mean - abs(sigma1Mean) * inDictionary['sigma1'][3]
    sigma2Min = sigma2Mean - abs(sigma2Mean) * inDictionary['sigma2'][3]
    A0Min = A0Mean - abs(A0Mean) * inDictionary['A0'][3]
    nlMin = nlMean - abs(nlMean) * inDictionary['nl'][3]
    nrMin = nrMean - abs(nrMean) * inDictionary['nr'][3]

    # Calculate maximum values for a range used in fit_module.fitFunc()
    # ---------------------------------------------------------------
    # Maximum as a factor of the mean
    alMax = alMean + abs(alMean) * inDictionary['al'][4]
    arMax = arMean + abs(arMean) * inDictionary['ar'][4]
    # arMin = -3.0
    sigma1Max = sigma1Mean + abs(sigma1Mean) * inDictionary['sigma1'][4]
    sigma2Max = sigma2Mean + abs(sigma2Mean) * inDictionary['sigma2'][4]
    A0Max = A0Mean + abs(A0Mean) * inDictionary['A0'][4]
    nlMax = nlMean + abs(nlMean) * inDictionary['nl'][4]
    nrMax = nrMean + abs(nrMean) * inDictionary['nr'][4]

```

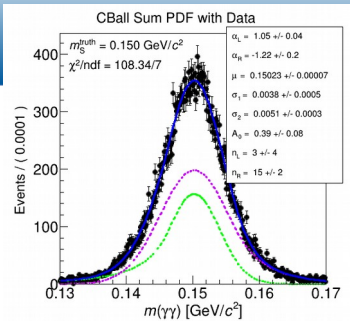
Key take-away: The parameter_control_module calculates new parameter control values for each loop iteration.

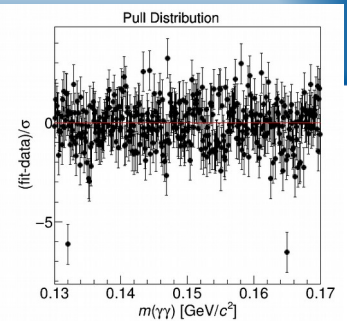
DiPhoton invariant mass after 6 iterations

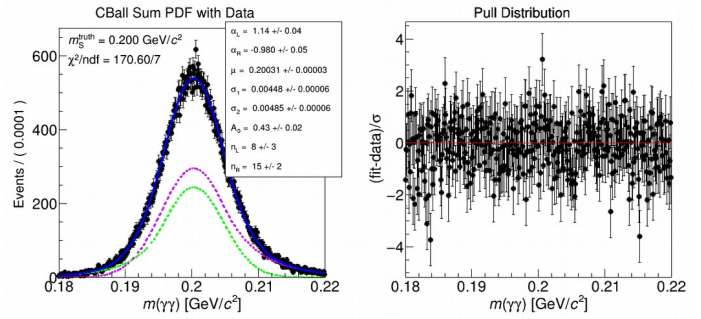


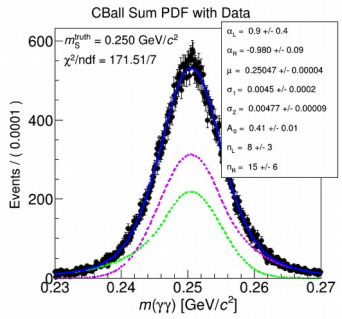


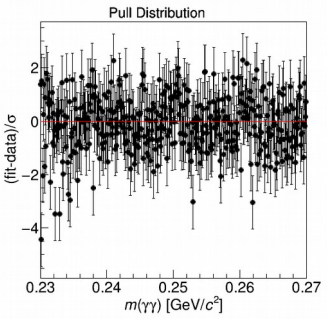


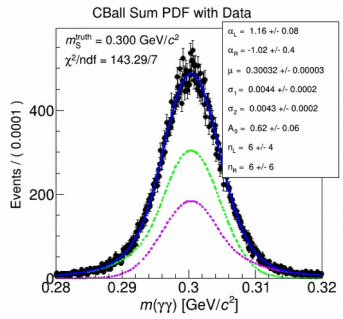


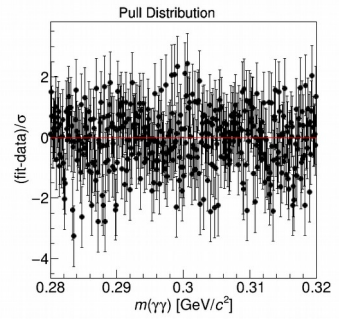


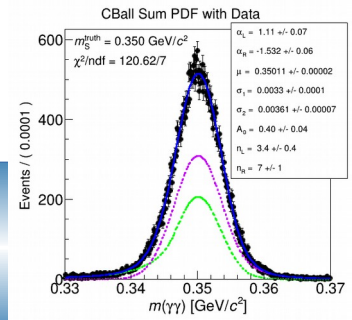


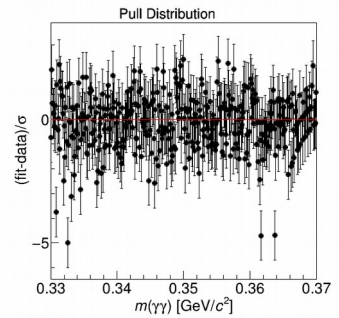












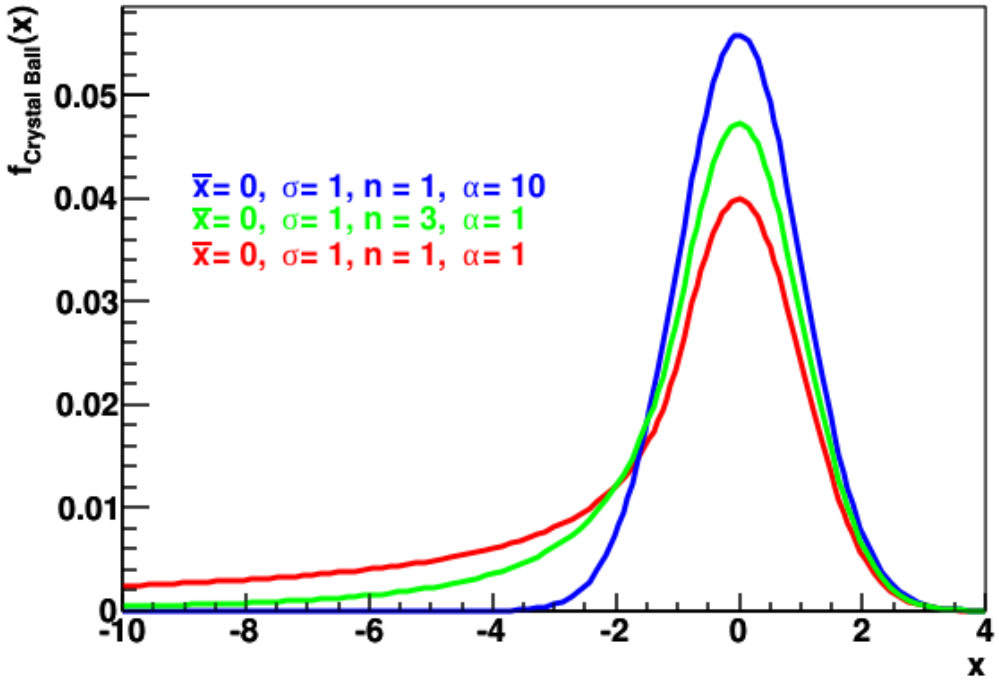
Key take-away: All of the pull distributions are still good.

Conclusion

1) To do list:
   - Complete toy MC study module.
   - Conduct toy MC study.
   - Adjust model as needed.
2) The toy MC study will show each parameter, its error, and its pull distribution, allowing us to adjust our parameter controls more precisely.
3) I am enjoying the programming and am hopeful that the results will be useful.

Key take-away: I'd like to go to graduate school.

Crystal Ball Function



Key take-away:

Dark Scalar Boson, S

$$\eta \rightarrow \pi^0 + \mathrm{S}(\rightarrow \gamma + \gamma)$$

Key take-away: