```
# - PRACTICE TEST CATALINA CISNEROS -

#0. import numpy and LT.box
import numpy as np
import LT.box as B

#1. Create array with given values
Es = np.array([0.6617, 1.1732, 1.3325])

#2. reate an empty array size 3
peaks = np.zeros(3)


#3. Read calibration file and save it in hcal
hcal=B.get_spectrum('calibration.Spe')

#4. show in a graph, with the automatic gaussian fit
hcal.plot()

#4. Select appropiate ranges
ranges = [(120, 144), (215, 243), (249, 272)]
i = 0
#6. Loop to repeat steps 4 and 5
for (xmin, xmax) in ranges:
    hcal.fit(xmin,xmax)
    hcal.plot()
    #5. obtain the mean value and store it in the array peaks
    peaks[i] = hcal.mean.value
    i += 1

# 7. Graph E vs peaks and label x and y axis

B.pl.figure()

B.plot_exp(Es, peaks)
B.pl.xlabel('Energy [MeV]')
B.pl.ylabel('peaks [Counts]')
B.pl.title('Energy vs Peaks')

# 8. Perform a linear fit on the graph
fitcal = B.linefit(peaks, Es)
fitcal.plot()


# 9. Verify calibration
B.pl.figure()
hcalnew = B.get_spectrum('calibration.Spe', calibration=fitcal)
hcalnew.plot()
# 10. set xaxis label to energies
B.pl.xlabel('Energies (MeV)')
```

```
# 11. Create four member array
degrees = np.array([30, 40, 60, 80])
# 12. Convert to radians and store in thetas
thetas = np.deg2rad(degrees)

# 13. create an empty array for efs
efs = np.zeros(4)

# 14. create an empty array for efsErr
efsErr = np.zeros(4)

# 15. Apply calibration in 30_deg_target
htarget=B.get_spectrum('30_deg_target.Spe', calibration = fitcal)

# 16. Apply calibration in 30_deg_empty
hempty=B.get_spectrum('30_deg_empty.Spe', calibration = fitcal)

# 17. Graph both htarget and hempty
B.pl.figure()
htarget.plot()
hempty.plot()
B.pl.title('htarget and hempty')

# 18. hcompton substract target - empty
B.pl.figure()
hcompton = htarget - hempty
hcompton.plot()
B.pl.xlabel('Energies (MeV)')
B.pl.ylabel('counts')
B.pl.title('htarget-hempty')

# 19. Select appropiate ranges on htarget
hcompton.fit(0.45, 0.70)
hcompton.plot()
# 20. store peak pos mean in efs
efs[0] = np.array(hcompton.mean.value)
# 21. store err in efsErr
efsErr[0] = hcompton.mean.err

#22

deg = [40, 60, 80]
ranges = [(0.45, 0.70), (0.35, 0.60), (0.25, 0.50)]

i = [1, 2, 3]
for j, d in enumerate(deg, start=1):
    htarget=B.get_spectrum(f"{d}_deg_target.Spe", calibration = fitcal)
    hempty=B.get_spectrum(f"{d}_deg_empty.Spe", calibration = fitcal)
    B.pl.figure()
    htarget.plot()
```

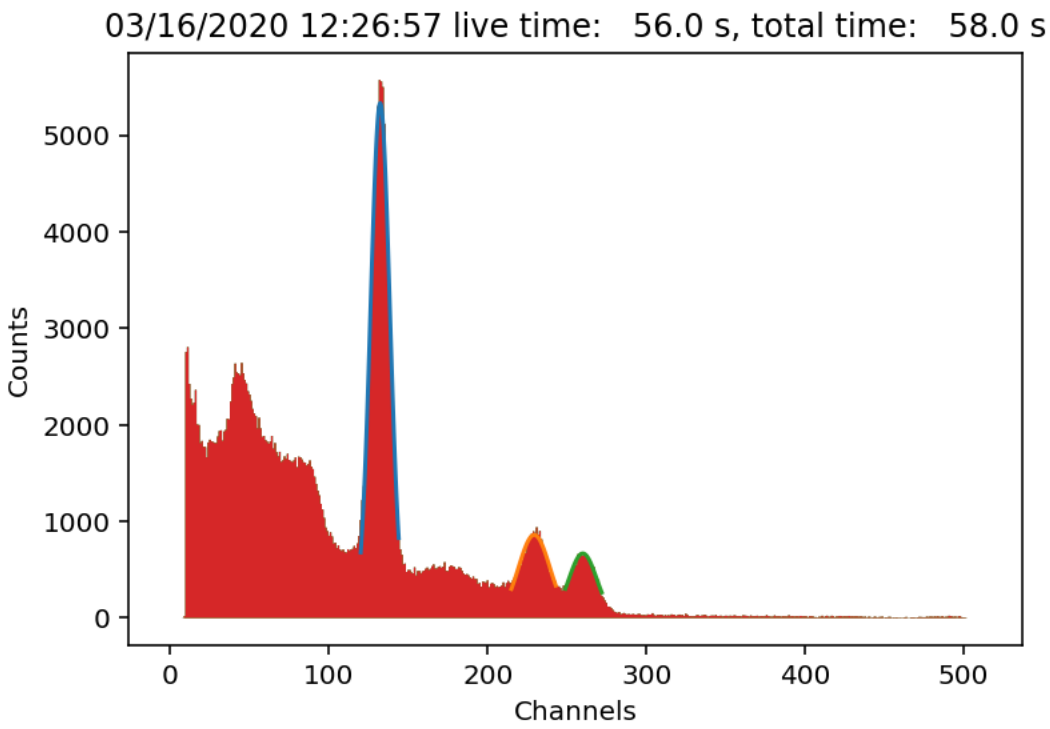
```
    hempty.plot()
    B.pl.figure()
    hcompton = htarget - hempty
    hcompton.plot()
    B.pl.xlabel('Energies (MeV)')
    B.pl.ylabel('counts')
    B.pl.title(f'{d} htarget-hempty')
    # 19. Select appropiate ranges on htarget
    xmin, xmax = ranges[j - 1]
    hcompton.fit(xmin, xmax)
    hcompton.plot()
    B.pl.title(f'{d} htarget-hempty')
    # 20. store peak pos mean in efs
    efs[j] = np.array(hcompton.mean.value)
    # 21. store err in efsErr
    efsErr[j] = hcompton.mean.err

# 23. compute e0/efs
e0= Es[0]
eratio = e0/efs
print(eratio)

# 24. graph eratio vs cos thetas
B.pl.figure()
cos_t = 1.0 - np.cos(thetas)
B.plot_exp(cos_t, eratio)
B.pl.ylabel('e-ratio')
B.pl.xlabel('cos(thetas)')
B.pl.title('1 - cos thetas vs e-ratio')

# 25. linefit to above
fit2 = B.linefit(cos_t, eratio)

# 26. slope from fit2
# 27. slope unc from fit2

# 28. offset unc from fit2
# 29. offset unc from fit2

eslope = fit2.slope
dslope = fit2.sigma_s
yoff   = fit2.offset
dyoff  = fit2.sigma_o

# 30. value of electron mass in MeV
me = e0 / eslope
dme = e0 * dslope / (eslope**2)


# 31. Print electron mass and uncertainty
print(f"Electron mass m_e = {me:.3f} ± {dme:.3f} MeV")
```

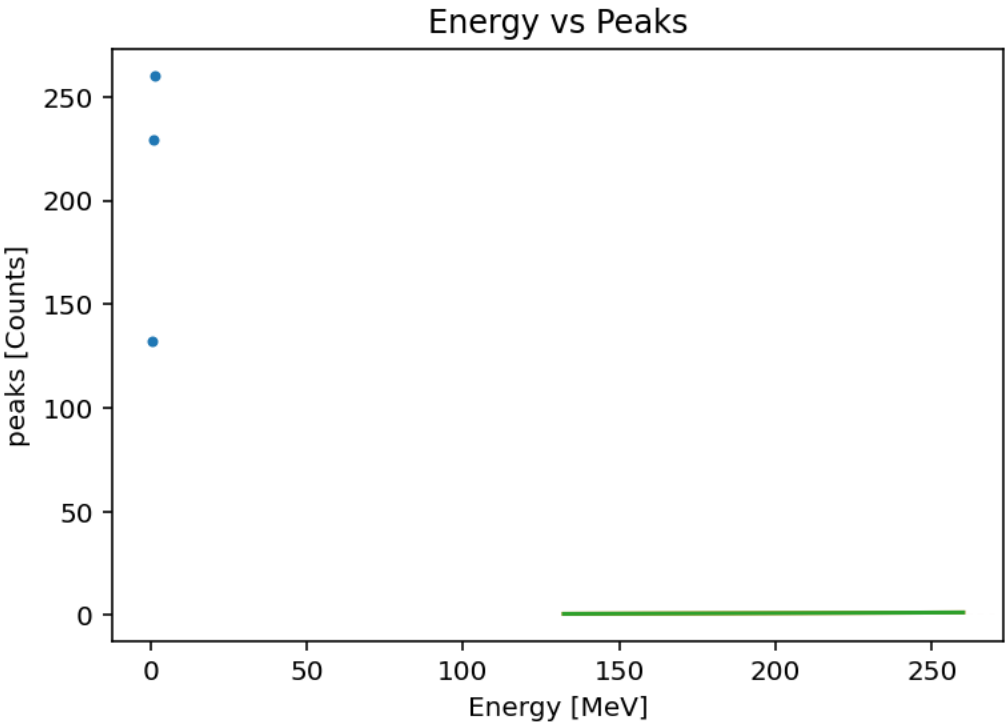
```
# 32. Display electron mass result on the graph
B.pl.text(0.05, 0.90, f"m_e = {me:.3f} ± {dme:.3f} MeV", transform=B.pl.gca().tra

# 33. Print and show offset result on the graph
print(f"Offset = {yoff:.3f} ± {dyoff:.3f} (expected ≈ 1.000)")
B.pl.text(0.05, 0.84, f"Offset = {yoff:.3f} ± {dyoff:.3f} (expected ≈ 1.000)", tr
```

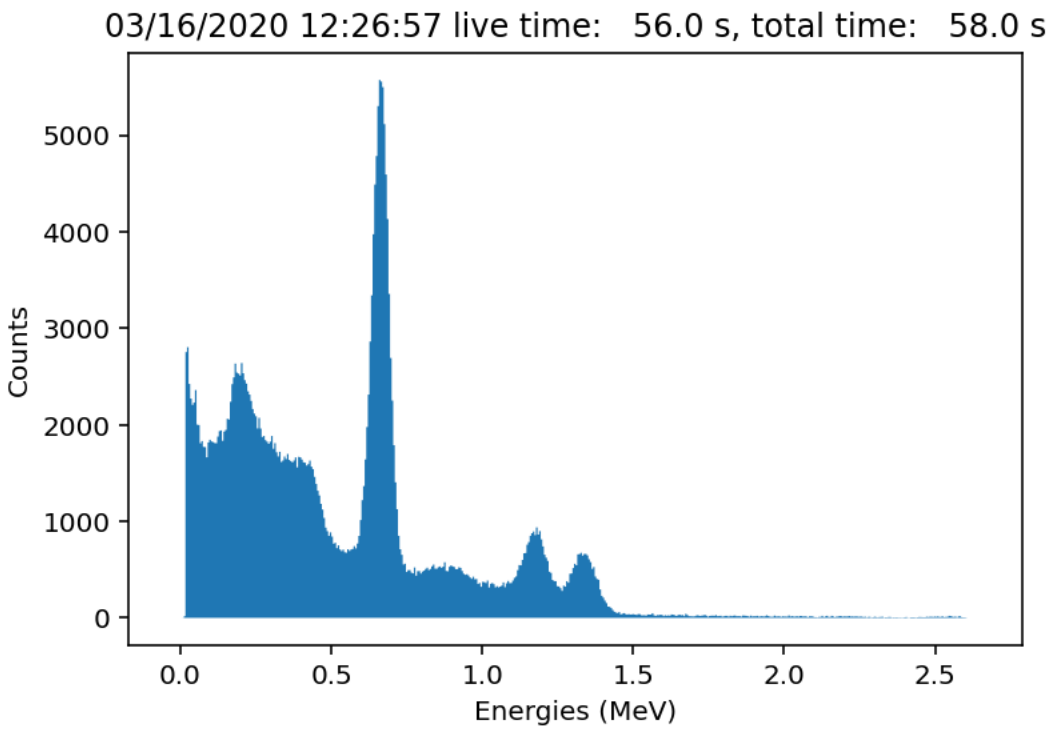
```
In [11]: %runfile '/Users/catacisneros/Library/CloudStorage/OneDrive-
Personal/FIU/Fall 2025/Intermediate Physics Lab/PracticeExam/
ComptonData/PracticeTest.py' --wdir
gen_fit kwargs =  {}
gen_fit.fit kwargs =  {}
Calculate numerical parameter derivatives with diff_step = 0.001
------------------------------------------------------------------------
Fit results:
A  =  5333.32728517083  +/-  25.389168601612944
mean  =  132.30331690555627  +/-  0.0243416432499209
sigma  =  6.057416767208668  +/-  0.02383462790792127
Chi square =  399.96110138205614
Chi sq./DoF =  18.180050062820733
------------------------------------------------------------------------
gen_fit kwargs =  {}
gen_fit.fit kwargs =  {}
Calculate numerical parameter derivatives with diff_step = 0.001
------------------------------------------------------------------------
Fit results:
A  =  859.0316525227809  +/-  8.86637634542006
mean  =  229.38235809757737  +/-  0.10055856639020294
sigma  =  9.909652171374615  +/-  0.13407710243227075
Chi square =  80.58811139456209
Chi sq./DoF =  3.099542745944696
------------------------------------------------------------------------
gen_fit kwargs =  {}
gen_fit.fit kwargs =  {}
Calculate numerical parameter derivatives with diff_step = 0.001
------------------------------------------------------------------------
Fit results:
A  =  667.670817991596  +/-  8.42832368734646
mean  =  260.04025655778435  +/-  0.11512457957445578
sigma  =  8.75266779749045  +/-  0.15909852400621516
Chi square =  33.30902815059364
Chi sq./DoF =  1.5861441976473163
------------------------------------------------------------------------
chisq/dof =  1.7620257306465904e-06
offset  =  -0.033389880571203484  +/-  0.0030160427356682963
slope =  0.005255643163395137  +/-  1.4075567730981259e-05
gen_fit kwargs =  {}
gen_fit.fit kwargs =  {}
Calculate numerical parameter derivatives with diff_step = 0.001
------------------------------------------------------------------------
Fit results:
A  =  299.29515244285056  +/-  15.130519385468645
mean  =  0.5624309794485278  +/-  0.0024500693675140587
sigma  =  0.04505935794740525  +/-  0.002396153927910696
Chi square =  50.55701710970918
Chi sq./DoF =  1.1234892691046485
------------------------------------------------------------------------
```

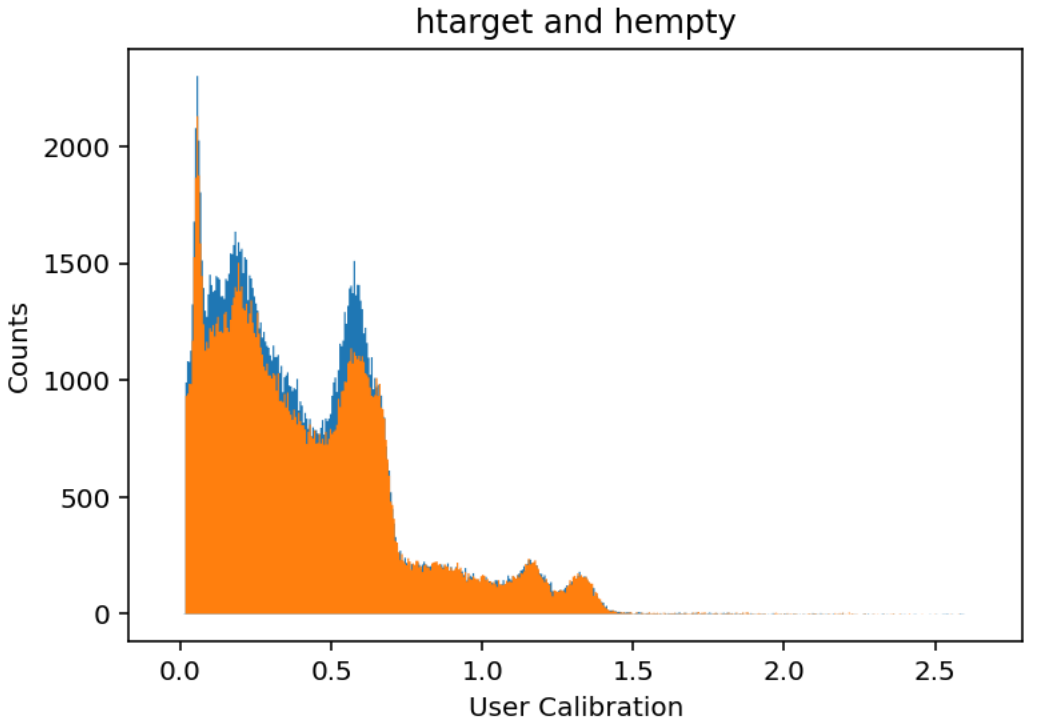
```
gen_fit kwargs =  {}
gen_fit.fit kwargs =  {}
Calculate numerical parameter derivatives with diff_step = 0.001
------------------------------------------------------------------------
Fit results:
A  =  284.0951302625165  +/-  13.763400075715516
mean  =  0.4994954549879141  +/-  0.0026596980253331608
sigma  =  0.04237669535029218  +/-  0.0030329164648254473
Chi square =  36.4735286448006
Chi sq./DoF =  0.8105228587733467
------------------------------------------------------------------------
gen_fit kwargs =  {}
gen_fit.fit kwargs =  {}
Calculate numerical parameter derivatives with diff_step = 0.001
------------------------------------------------------------------------
Fit results:
A  =  454.8645571273567  +/-  18.756180313983897
mean  =  0.39783937609426423  +/-  0.0019431665847604912
sigma  =  0.03763768922879935  +/-  0.002181867232766341
Chi square =  41.818070331876854
Chi sq./DoF =  0.9504106893608376
------------------------------------------------------------------------
gen_fit kwargs =  {}
gen_fit.fit kwargs =  {}
Calculate numerical parameter derivatives with diff_step = 0.001
------------------------------------------------------------------------
Fit results:
A  =  469.80349799927035  +/-  19.94615588915875
mean  =  0.31509308012976606  +/-  0.0015455279226512329
sigma  =  0.03204563339852531  +/-  0.0015710390299358942
Chi square =  38.113227884351915
Chi sq./DoF =  0.8469606196522648
------------------------------------------------------------------------
[1.17649992 1.32473678 1.66323406 2.10001438]
chisq/dof =  7.44006212293729e-05
offset  =  1.0054672297259393  +/-  0.008060131378959029
slope =  1.3236381242930848  +/-  0.01607578048807398
Electron mass m_e = 0.500 ± 0.006 MeV
Offset = 1.005 ± 0.008 (expected ≈ 1.000)
```

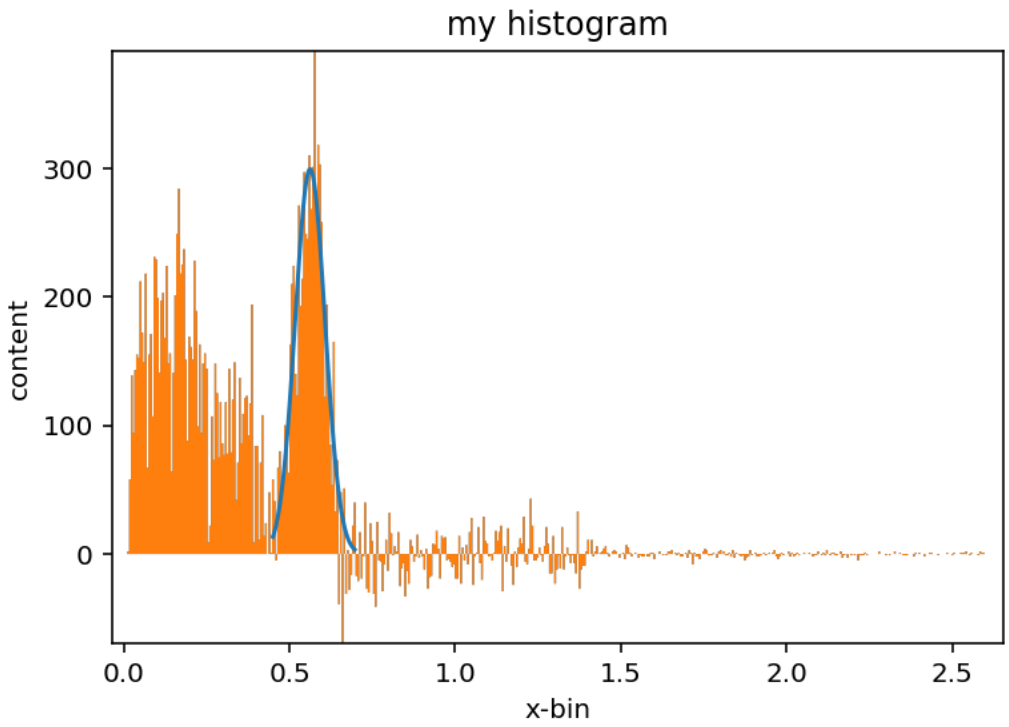
In [12]:

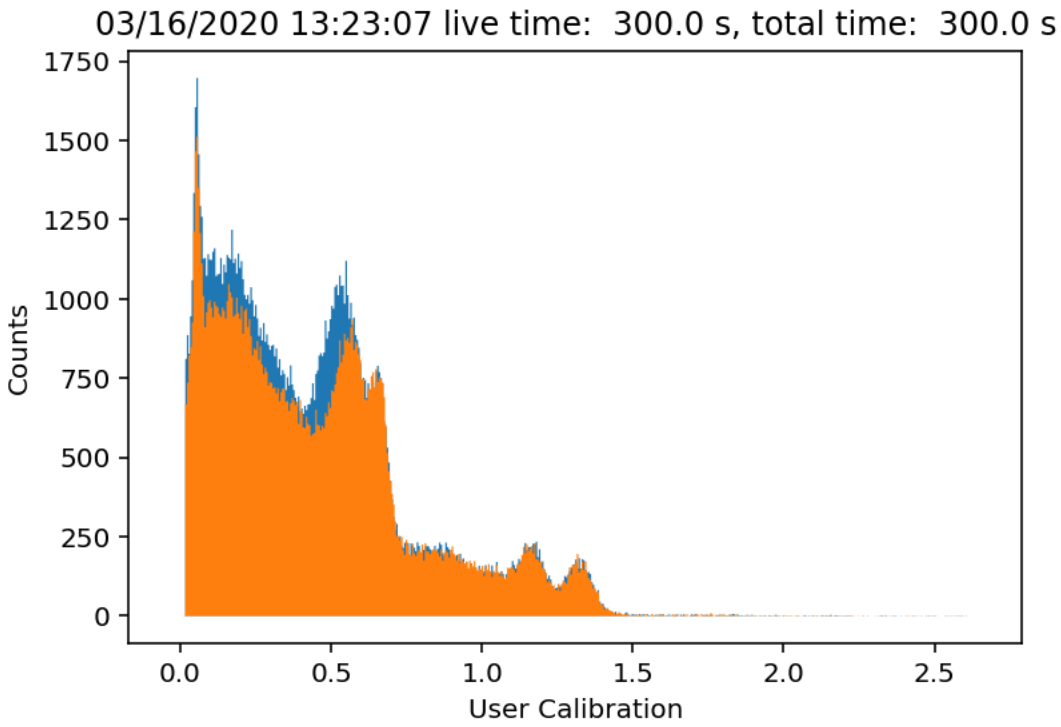

03/16/2020 12:26:57 live time: 56.0 s, total time: 58.0 s
Counts
Channels
0
1000
2000
3000
4000
5000
0
100
200
300
400
500

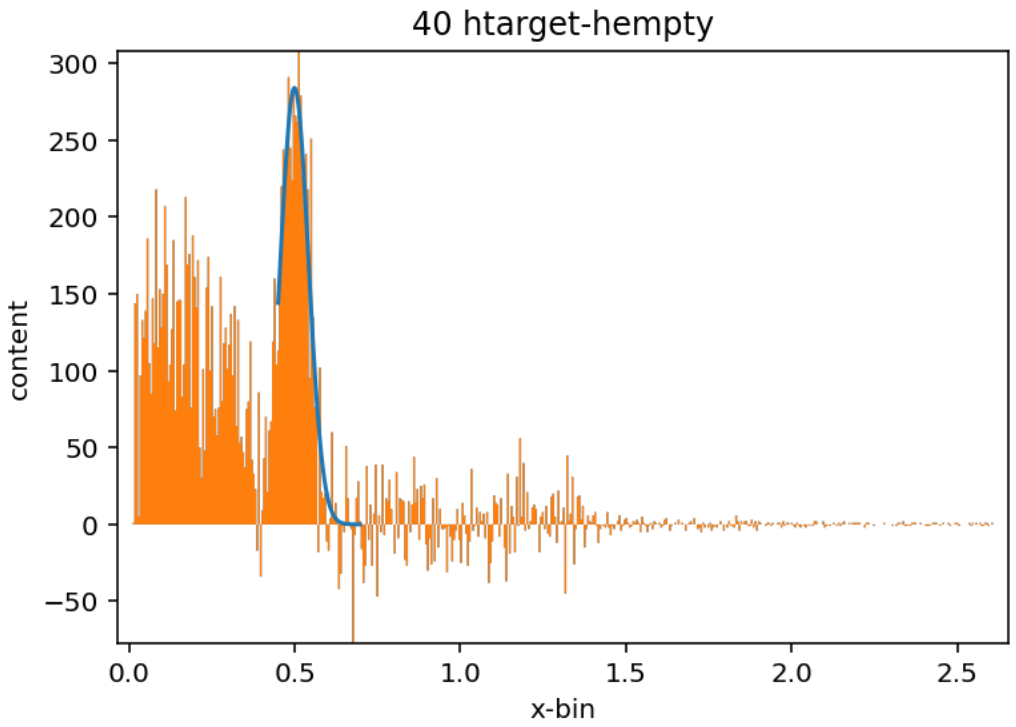

Energy vs Peaks
peaks [Counts]
Energy [MeV]
0
50
100
150
200
250
0
50
100
150
200
250

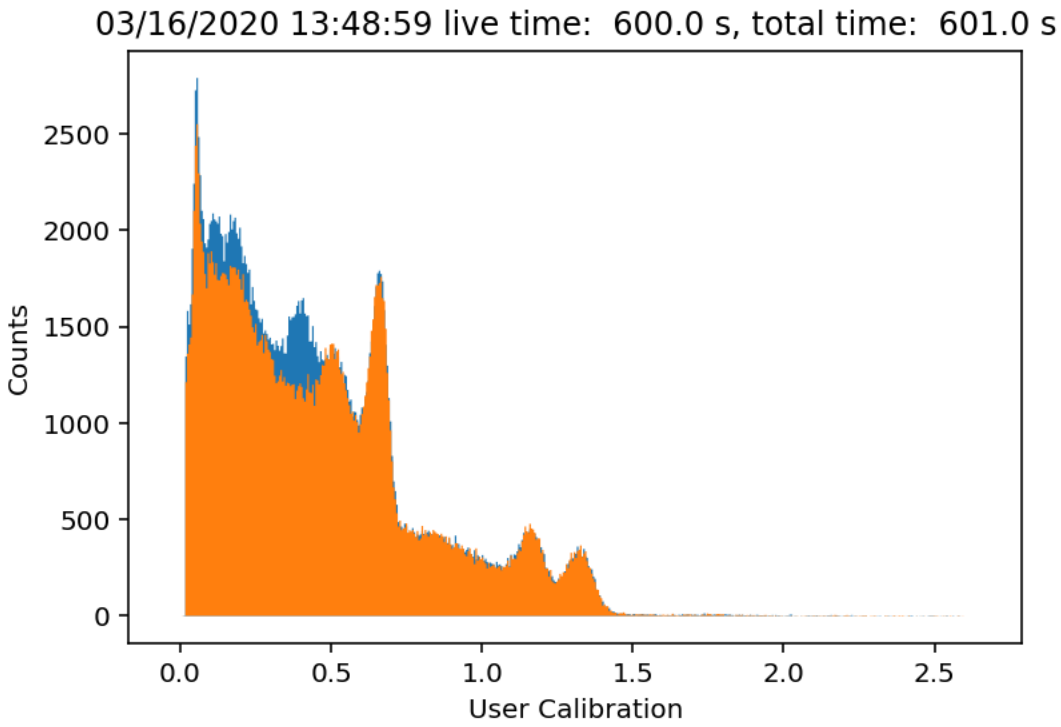

03/16/2020 12:26:57 live time: 56.0 s, total time: 58.0 s
Counts
Energies (MeV)
0
1000
2000
3000
4000
5000
0.0
0.5
1.0
1.5
2.0
2.5

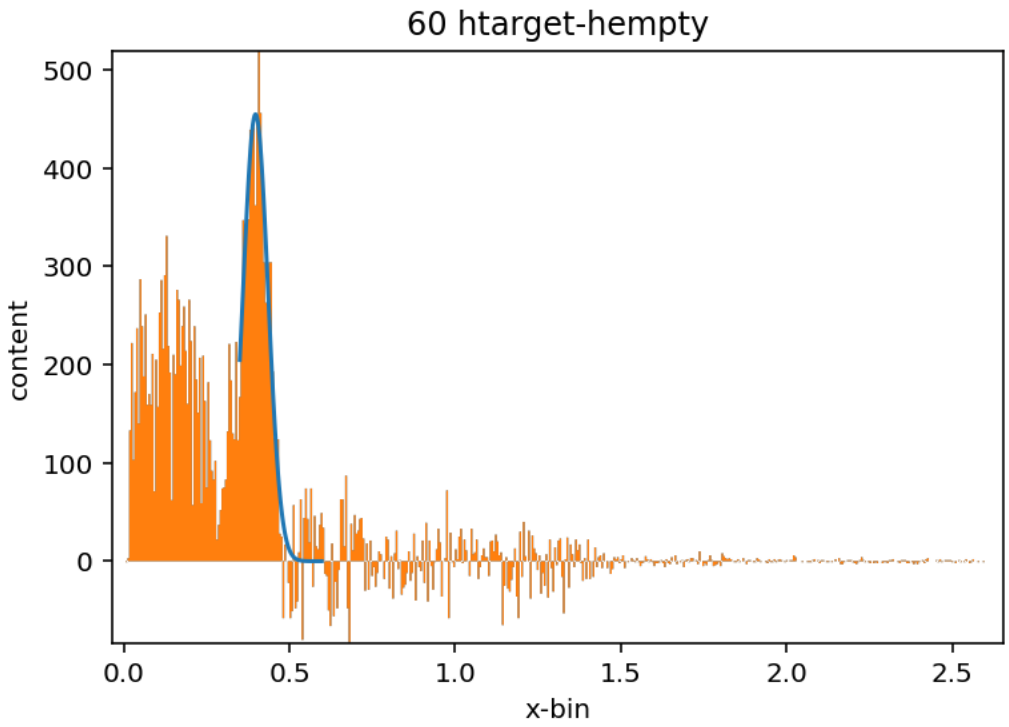

htarget and hempty
Counts
User Calibration
0
500
1000
1500
2000
0.0
0.5
1.0
1.5
2.0
2.5

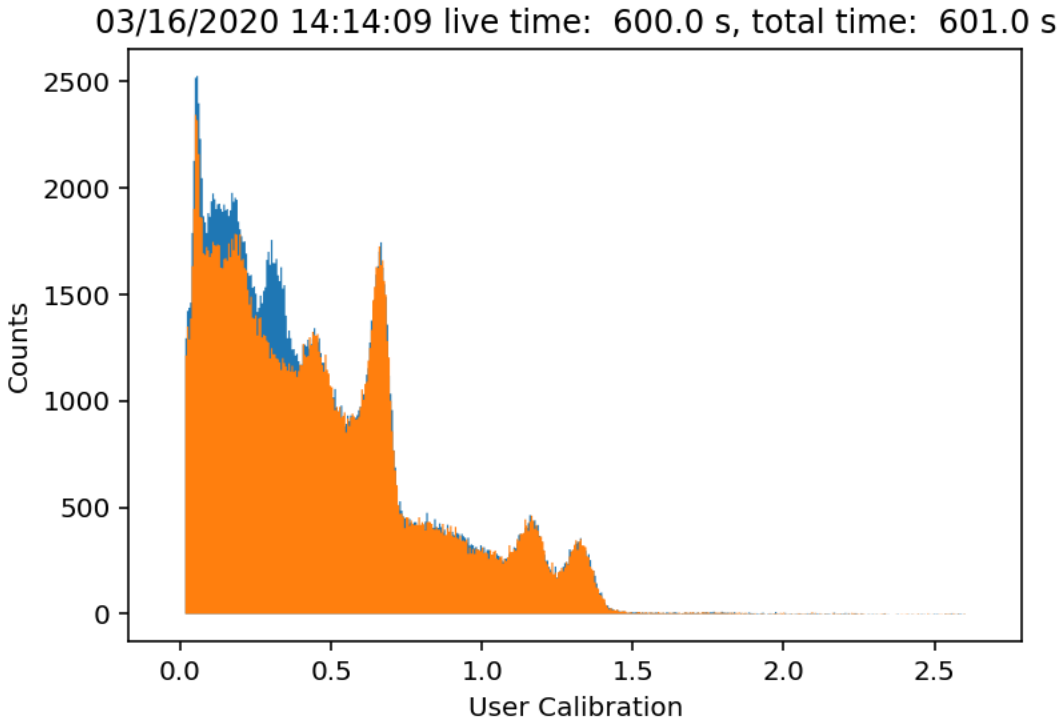

my histogram
content
x-bin
0
100
200
300
0.0
0.5
1.0
1.5
2.0
2.5

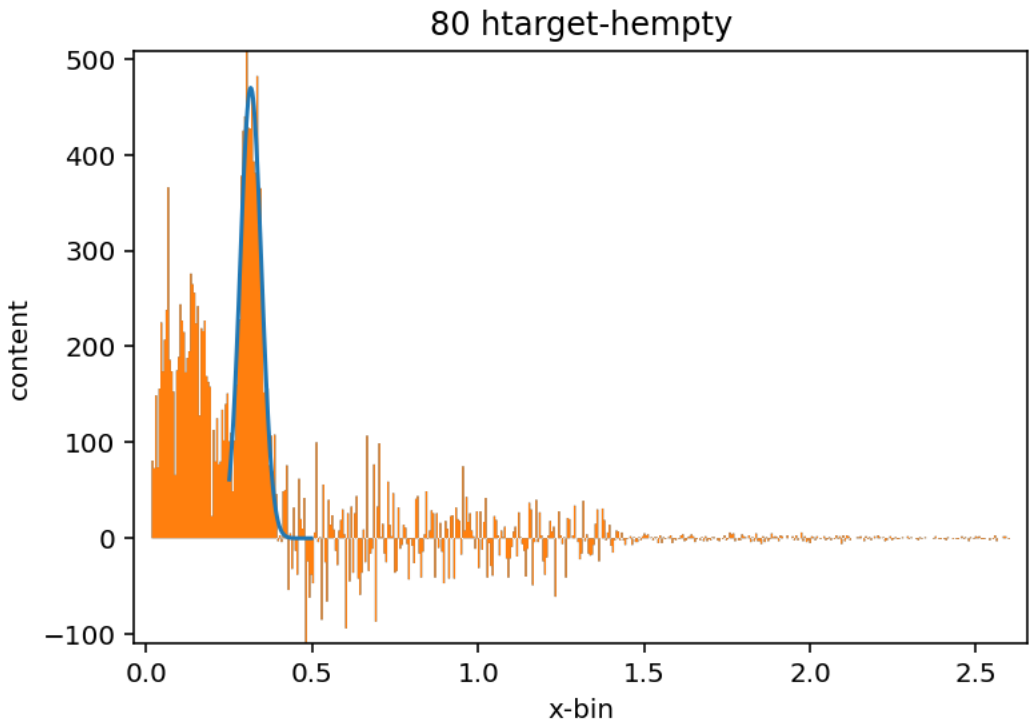

03/16/2020 13:23:07 live time: 300.0 s, total time: 300.0 s
Counts
User Calibration
0
250
500
750
1000
1250
1500
1750
0.0
0.5
1.0
1.5
2.0
2.5

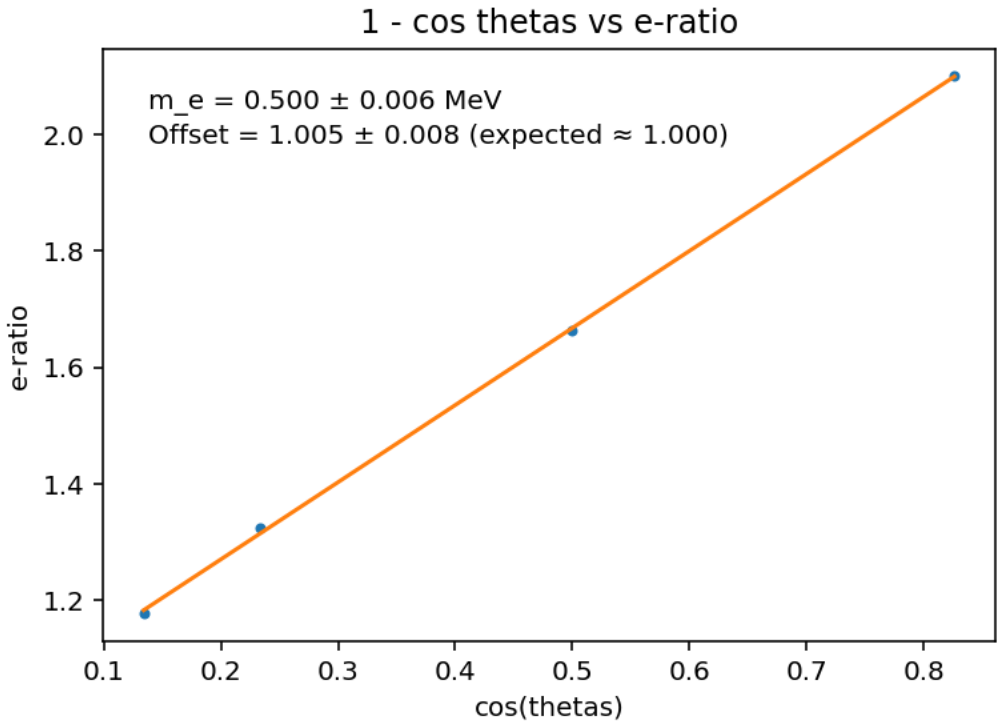

40 htarget-hempty
content
x-bin
300
250
200
150
100
50
0
−50
0.0
0.5
1.0
1.5
2.0
2.5


03/16/2020 13:48:59 live time: 600.0 s, total time: 601.0 s
Counts
User Calibration
2500
2000
1500
1000
500
0
0.0
0.5
1.0
1.5
2.0
2.5


60 htarget-hempty
content
x-bin
0
100
200
300
400
500
0.0
0.5
1.0
1.5
2.0
2.5


03/16/2020 14:14:09 live time: 600.0 s, total time: 601.0 s
Counts
User Calibration
0
500
1000
1500
2000
2500
0.0
0.5
1.0
1.5
2.0
2.5


80 htarget-hempty
content
x-bin
500
400
300
200
100
0
−100
0.0
0.5
1.0
1.5
2.0
2.5


1 - cos thetas vs e-ratio
m_e = 0.500 ± 0.006 MeV
Offset = 1.005 ± 0.008 (expected ≈ 1.000)
e-ratio
cos(thetas)
2.0
1.8
1.6
1.4
1.2
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8