# mcDSAM

## A monte carlo implementation for determining the lifetime from a doppler shifted gamma ray (DSAM experiment)

The idea of this code is to take a TRIM input of tracking data for nuclei in a backing material to determine the specific angle/motion of an ion before decay. At this point, a lifetime is selected via a Monte Carlo random sampling. At this given decay time, the position and velocity vector the decaying nucleus is calculated. Then, the gamma is generated and the shifted energies are determined/plotted for given angles. Finally, the shifts vs cos(angle) are plotted to determine the attenuation factor, F(tau). A special thanks to Jerry Hinnefeld, who originally came up with this idea and who's help was integral to this work and getting this working.

Written by Bryce Frentz

November 2019

In [1]:

```
import math
import random
import numpy as np
%matplotlib inline
import matplotlib.pyplot as plt
plt.style.use('seaborn-white')
import scipy.optimize as sopt
from scipy.interpolate import interp1d
from scipy.interpolate import UnivariateSpline
import codecs
import matplotlib.gridspec as gridspec
```

## 1. Open the TRIM output file and extract the collision/trajectory information

In [2]:

```
trajectoryData = np.zeros(((10000, 200, 16)))  # space for 10000 ion trajectories, 200 path segments (col
lisions) per ion, and 16 data items per segment
nSegments = np.zeros(10000)                    # number of segments for each ion track (for diagnostic pur
poses), segments are between collisions
currentIon = 0                                 # for tracking which ion is used in calculation and data im
port
massAMU = 15.0030656                           # TRIM uses the atomic weight, but we need the mass of the
 isotope in amu

# TRIM collision data output
f = codecs.open('./collision/COLLISON_O_PureTa_10000.txt', 'r', encoding='iso-8859-1')

#iSeg = 1
lineCounter = 0
ion = 1
for line in f:
    #line.decode("iso-8859-1").encode("utf-8")
    lineCounter += 1

    # Pull the ion energy from header lines, and calculate initial speed of the recoil
    if line[6:16] == 'Ion Energy':
        energy = float(line[21:29])
        print('\nIon energy = ', energy, 'keV')
        v0 = ((2 * energy / massAMU / 931500)**(0.5)) * 3000   # speed in A/fs, the 3000 is for convertin
g c
        print('Ion velocity = ', v0, 'A/fs')


    # Collect useful data elements from TRIM output for each ion segment
    if (line[0:1] == '³') and (line[1:2] != '='):
        ion = int(line[1:6])               # this pulls the ion number, which starts from 1
        if ion == currentIon:              # another segment for the current ion trajectory
            iSeg = iSeg + 1                # increment the segment counter
        else:
            if ion != 1:                   # if this is ion 0001, it's not the end of a prior ion traject
ory
                nSegments[ion-2] = iSeg    # nSegments[] index starts at 0, so ion-2 at this point to put
the value in the right elements of nSegments[]
            currentIon = ion               # reset this variable
            iSeg = 1                       # start with iSeg=1; later we'll add an iSeg=0 for the piece f
rom t=0 to the first collision, which is why we subtract 2 earlier

        trajectoryData[ion-1,iSeg,0] = float(line[7:16])   # E1 - energy at beginning of segment
        trajectoryData[ion-1,iSeg,1] = float(line[17:27])  # x1 - depth at beginning of segment
        trajectoryData[ion-1,iSeg,2] = float(line[28:38])  # y1 - horizontal lateral position at beginnin
g of segment
        trajectoryData[ion-1,iSeg,3] = float(line[39:49])  # z1 - vertical lateral position at beginning
 of segment
        trajectoryData[ion-1,iSeg,4] = float(line[50:57])  # Se1 - electronic stopping power, ev/A

# end of file behavior
nSegments[ion-1] = iSeg   # at end-of-file, save the no. of segments for the last ion

lastIon = ion
# print information for diagnostic purposes
print('\n')
print('Last ion = ',lastIon)
print('Minimum Segments = ',min(nSegments[0:]))
print('Maximum Segments = ',max(nSegments[0:]))
print('\n')

# close the TRIM file
f.close()
```

```
Ion energy =  106.2 keV
Ion velocity =  11.695453786604201 A/fs


Last ion =  10000
Minimum Segments =  1.0
Maximum Segments =  114.0
```

## 2. Calculate and store other data for each segment of the ion tracks

This only needs to be calculated once per different target input.

In [3]:

```
# Calculating things like distance travelled per segment, directional cosines, energy, velocity, electron
ic stopping powers, and time at the segments

# ion counter
i = 0

while i < lastIon:

    # Calculate initial segment 0
    trajectoryData[i,0,0] = energy        # specified initial energy
    trajectoryData[i,0,4] = trajectoryData[i,1,4]  # set Se equal to value at beginning of second segment

    # Distance travelled
    # Starts at (0,0,0)
    dist = (trajectoryData[i,1,1]**2 + trajectoryData[i,1,2]**2 + trajectoryData[i,1,3]**2)**(0.5)

    # directional cosines
    # initial value is 0 in all three directions
    xcos = trajectoryData[i,1,1] / dist
    ycos = trajectoryData[i,1,2] / dist
    zcos = trajectoryData[i,1,3] / dist

    E2 = energy - trajectoryData[i,1,4] * (dist/1000.0)  # Energy after loss from stopping (E2 = E - Se*d
ist) (divide by 1000 to convert from ev/A to keV/A)
    v1 = v0
    v2 = (2*E2/massAMU/931500)**(0.5)*3000    # calculated from new energy (in A/fs)
    timeSegment = dist / ((v1+v2)/2)    # in fs -- (v1+v2)/2 is avg v for this segment

    trajectoryData[i,0,5] = E2    # store these in the tdat (trajectory data) array
    trajectoryData[i,0,6] = dist  # ...for ion 0001 (index 0) and segment 0
    trajectoryData[i,0,7] = xcos
    trajectoryData[i,0,8] = ycos
    trajectoryData[i,0,9] = zcos
    trajectoryData[i,0,10] = v1
    trajectoryData[i,0,11] = v2
    trajectoryData[i,0,12] = timeSegment

    # Now calculate and store the same data items for other segments beginning with each atomic collision
    iSeg = 1
    while iSeg < nSegments[i]:

        # distance traveled
        # sqrt((x2-x1)^2 + (y2-y1)^2 + (z2-z1)^2)
        dist = ((trajectoryData[i,iSeg+1,1] - trajectoryData[i,iSeg,1])**2 + (trajectoryData[i,iSeg+1,2]
- trajectoryData[i,iSeg,2])**2 + (trajectoryData[i,iSeg+1,3] - trajectoryData[i,iSeg,3])**2)**(0.5)

        # directional cosines
        xcos = trajectoryData[i,iSeg,1] / dist
        ycos = trajectoryData[i,iSeg,2] / dist
        zcos = trajectoryData[i,iSeg,3] / dist

        # energy and velocity at segment start
        E2 = trajectoryData[i,iSeg,0] - (trajectoryData[i,iSeg,4] * (dist/1000.0)) # E = E0 - Se*dist (in
keV/A)
        v1 = (2*trajectoryData[i,iSeg,0]/massAMU/931500)**(0.5)*3000    # in A/fs
        v2 = (2*E2/massAMU/931500)**(0.5)*3000    # in A/fs
        timeSegment = dist / ((v1+v2)/2)    # in fs

        trajectoryData[i,iSeg,5] = E2
        trajectoryData[i,iSeg,6] = dist
        trajectoryData[i,iSeg,7] = xcos
        trajectoryData[i,iSeg,8] = ycos
        trajectoryData[i,iSeg,9] = zcos
        trajectoryData[i,iSeg,10] = v1
        trajectoryData[i,iSeg,11] = v2
        trajectoryData[i,iSeg,12] = timeSegment + trajectoryData[i,iSeg-1,12]   # cumulative time at end
 of this segment

        iSeg += 1
```

```
    i += 1


#DEBUG
# print out some random ion's values to see if they're reasonable...
#ion = random.randint(1, lastIon)
#segment = 0
#while segment < nSegments[ion]:
#    print('Ion ', ion+1, ', Segment ', segment, ', time at end = ', trajectoryData[ion,segment,12])
#    segment += 1
```

### 3. Generate Doppler Shifted Gamma rays

Change this block for adjusting the input lifetime or gamma energy.

The python module `random` contains several random number generators in addition to `random.random()`. In particular, the routine `random.expovariate()` generates random numbers distributed according to a negative exponential function, rather than uniformly between 0 and 1. This is a quicker and easier way to generate our desired sampling distribution instead of defining the normalized integral.

In [4]:

```
nGammas = 50000                       # number of gammas to generate for each angle
eGamma0 = 6793.1                      # unshifted energy of transition gamma ray (in keV)
eGamma = np.zeros((7, nGammas))       # array to store Doppler-shifted gamma energies for each angle
decaySegment = np.zeros((7, nGammas)) # recording the segment in which the nucleus decays
genGammas = np.zeros(7)               # gammas generated for each angle

tau = 3.0                               # average lifetime of state, in fs
detectorResolution = 2.0                # 1-sigma detector resolution, in keV

expAngles = [0, 45, 60, 75, 90, 111, 135]    # angles used in measurement


# Loop over angles
for angle in range(len(expAngles)):
    detectorAngle = expAngles[angle]*(math.pi/180.0)      # detector angle, converted to radians

    # Gamma counters
    nGenerated = 0      # counter for number of gammas successfully generated

    # Generate gammas
    while nGenerated < nGammas:

        time = random.expovariate(1/tau)    # Monte Carlo generator for decreasing exponential
        ion = random.randint(0,9999)         # randomly select one of the 100 ion trajectories; CHANGE TH
IS IF THE TRIM FILE GENERATES A DIFFERENT NUMBER OF IONS
        #print('DEBUG: time = ',time)       # for debugging - time of decay
        #print('DEBUG: ion = ',ion)         # for debugging - ion

        iSeg = 0
        while iSeg < nSegments[ion]:

            # find the segment where the decay occurs
            if time < trajectoryData[ion,iSeg,12]:

                # Interpolate the ion speed between the values at the beginning and end of this segment
                # Assuming a constant slow-down
                decayVelocity = trajectoryData[ion,iSeg,10] + (time - trajectoryData[ion,iSeg-1,12]) * (t
rajectoryData[ion,iSeg,11] - trajectoryData[ion,iSeg,10])
                beta = decayVelocity/3000.0     # c = 3000 A/fs

                # Find the angle between ion direction and detector direction
                cosDetector = trajectoryData[ion,iSeg,7]*math.cos(detectorAngle) + trajectoryData[ion,iSe
g,8]*math.sin(detectorAngle)

                # Determines shifted gamma energy
                # Doppler formula +/- detector resolution
                eGamma[angle][nGenerated] = eGamma0*(1 + beta*cosDetector) + random.gauss(0,detectorResol
ution)

                # record in which segment the decay occurs
                decaySegment[angle][nGenerated] = iSeg

                # counters
                nGenerated += 1
                iSeg = nSegments[ion]

            else:

                #next segment
                iSeg += 1

                # If decay takes longer than all segments
                # generate another time and try again
                #if iSeg == nSegments[ion]:
                    #print('Time ',time,' fs not found in segments 1 - ', nSegments[ion],' for ion ', ion
+1,', i = ', i)
                    #print('Trying this one again.')

    genGammas[angle] = nGenerated
```

```
print('\n')
print('Gammas generated for each angle = ', genGammas)   # should be equal to nGammas
print('\n')
print('Angle\tMean Energy\t\tMedian Energy\t\tEnergy Range')
for angle in range(len(expAngles)):
    eGammaMean = np.mean(eGamma[angle][:nGenerated-1])
    eGammaMedian = np.median(eGamma[angle][:nGenerated-1])
    eGammaMin = np.min(eGamma[angle][:nGenerated-1])
    eGammaMax = np.max(eGamma[angle][:nGenerated-1])
    print(expAngles[angle], '\t', round(eGammaMean,3), '\t\t', round(eGammaMedian,3), '\t\t(', round(eGam
maMin,3), ',', round(eGammaMax,3), ')')
print('\n')

print('Median Energy:')
for angle in range(len(expAngles)):
    eGammaMedian = np.median(eGamma[angle][:nGenerated-1])
    print(round(eGammaMedian,3))
```

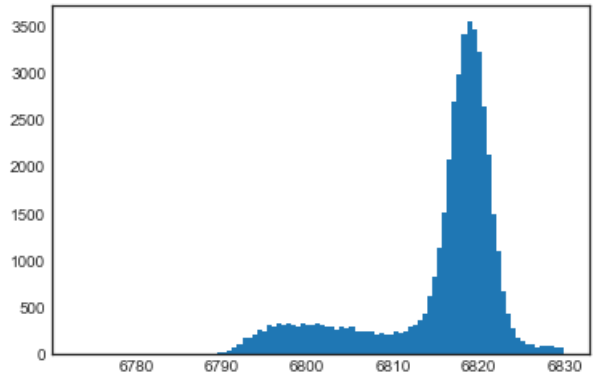
```
Gammas generated for each angle =  [50000. 50000. 50000. 50000. 50000. 50000. 50000.]


Angle    Mean Energy             Median Energy           Energy Range
0         6818.828                6818.421               ( 6786.693 , 7115.734 )
45        6811.302                6810.924               ( 6764.997 , 7016.225 )
60        6805.877                6805.558               ( 6713.684 , 6980.874 )
75        6799.691                6799.363               ( 6713.038 , 6918.852 )
90        6793.085                6793.087               ( 6697.605 , 6905.8 )
111       6783.934                6784.274               ( 6654.808 , 6862.785 )
135       6774.915                6775.309               ( 6578.624 , 6817.738 )


Median Energy:
6818.421
6810.924
6805.558
6799.363
6793.087
6784.274
6775.309
```

## 4. Plot the histograms

Here I plot various histograms for the calculation. First is a specific angle to check the validity of the method. Second is a plot of the segment in which the gamma decay occurs. All plots after that are to show all of the spectra for the given angle.

In [5]:

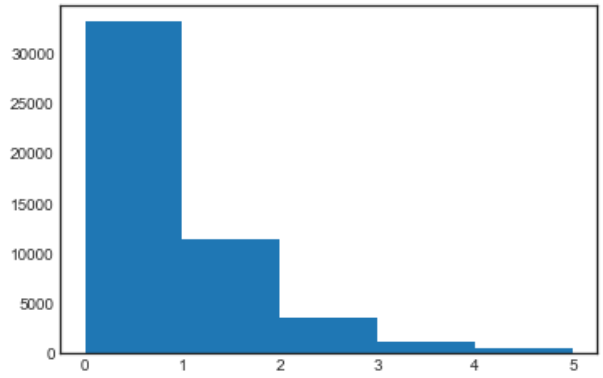
```
plt.hist(eGamma[0][:nGenerated],bins=100,range=(6773,6830))
plt.show()
```



In these spectra, there is a clear tail and bump around 6794 keV. This is due to gammas that are completely stopped before decaying.

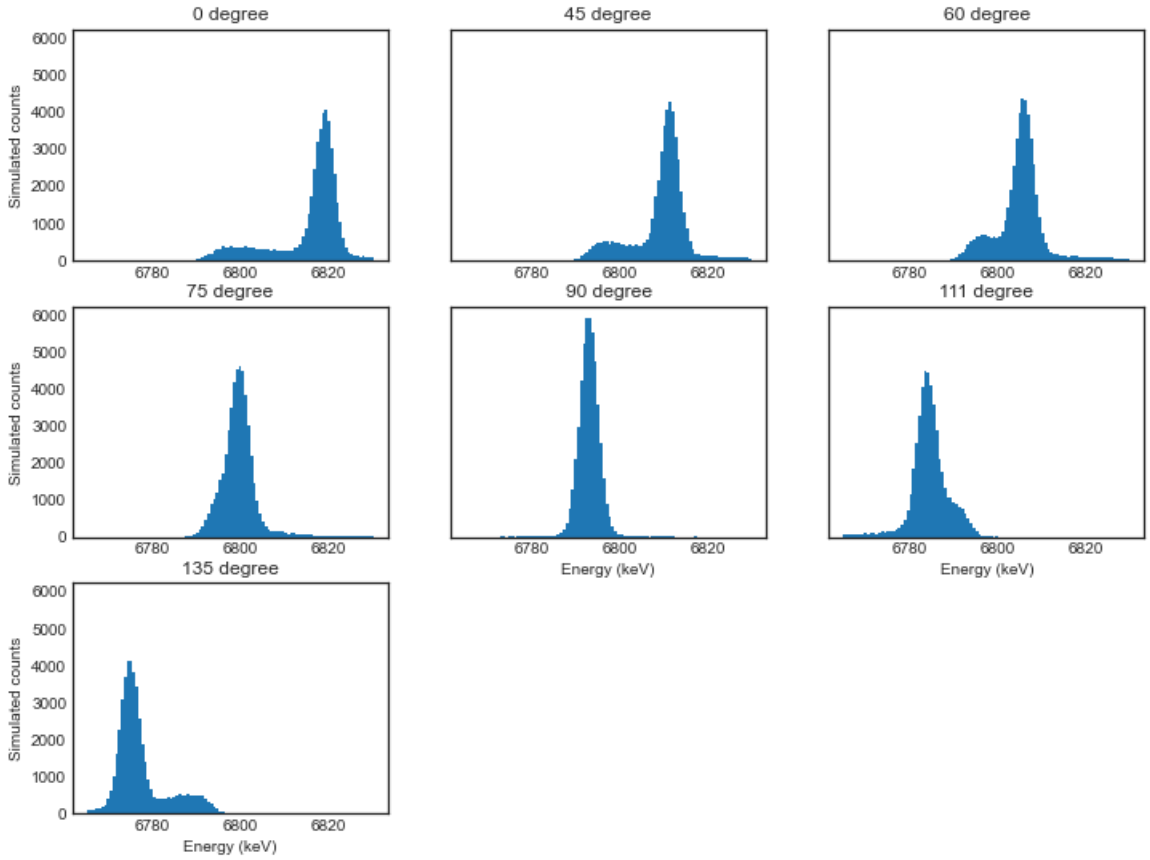
In [6]:

```
plt.hist(decaySegment[0][:nGenerated],bins=5,range=(0,5))
plt.show()
```



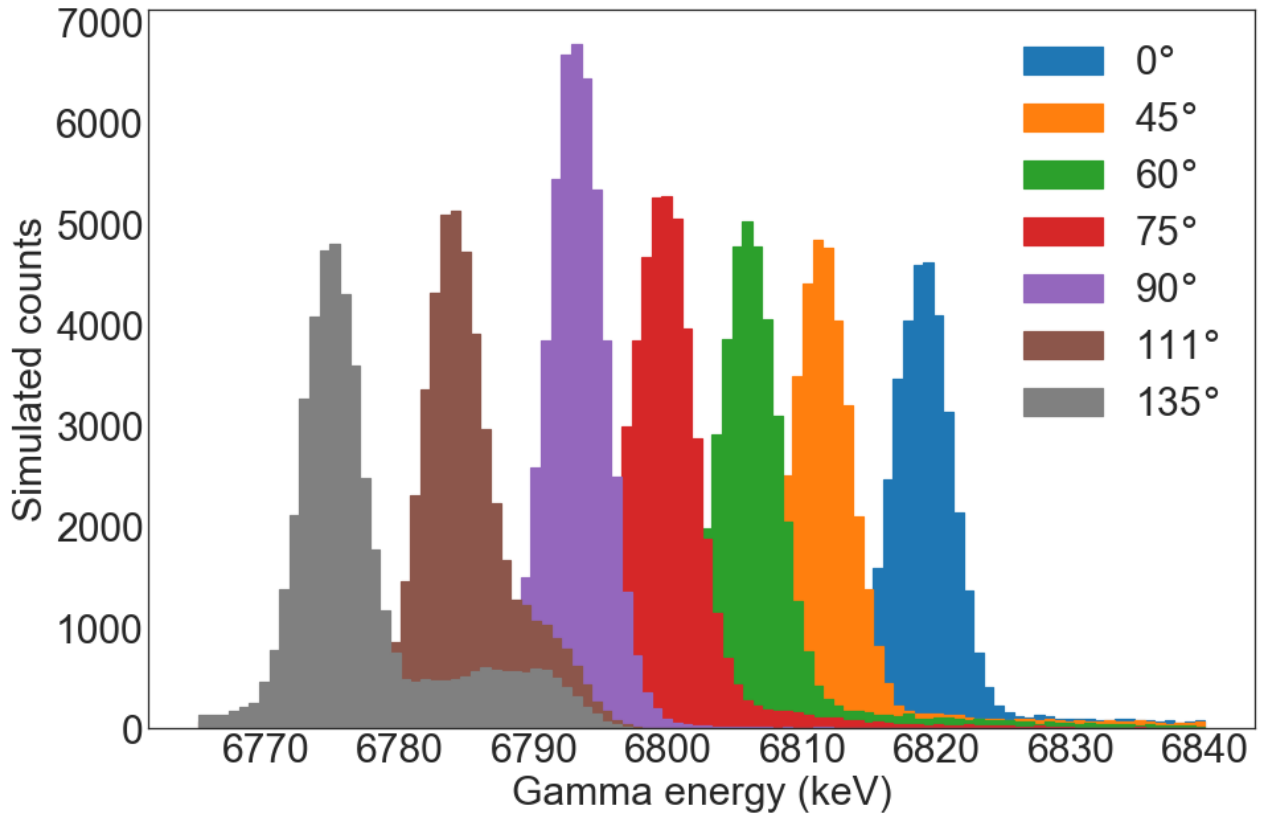
In [7]:

```
# Create nine subplots sharing y axis
fig_size = plt.rcParams["figure.figsize"]
fig_size[0] = 12
fig_size[1] = 9
plt.rcParams["figure.figsize"] = fig_size
fig, axs = plt.subplots(3, 3, sharey = True, sharex = False)

axs[0, 0].hist(eGamma[0][:nGenerated-1],bins=100,range=(6765,6830))
axs[0, 0].set_title('0 degree')
axs[0, 0].set(ylabel = 'Simulated counts')
axs[0, 1].hist(eGamma[1][:nGenerated-1],bins=100,range=(6765,6830))
axs[0, 1].set_title('45 degree')
axs[0, 2].hist(eGamma[2][:nGenerated-1],bins=100,range=(6765,6830))
axs[0, 2].set_title('60 degree')
axs[1, 0].hist(eGamma[3][:nGenerated-1],bins=100,range=(6765,6830))
axs[1, 0].set_title('75 degree')
axs[1, 0].set(ylabel = 'Simulated counts')
axs[1, 1].hist(eGamma[4][:nGenerated-1],bins=100,range=(6765,6830))
axs[1, 1].set_title('90 degree')
axs[1, 1].set(xlabel = 'Energy (keV)')
axs[1, 2].hist(eGamma[5][:nGenerated-1],bins=100,range=(6765,6830))
axs[1, 2].set_title('111 degree')
axs[1, 2].set(xlabel = 'Energy (keV)')
axs[2, 0].hist(eGamma[6][:nGenerated-1],bins=100,range=(6765,6830))
axs[2, 0].set_title('135 degree')
axs[2, 0].set(ylabel = 'Simulated counts')
axs[2, 0].set(xlabel = 'Energy (keV)')

axs[2, 1].remove()
axs[2, 2].remove()

#axs[2, 2].hist(expAngles[:7],bins=100,range=(0,180))
#axs[2, 2].set_title('Histogram of Angles')

#for ax in axs.flat:
    #ax.set(ylabel='y-label')

# Hide x labels and tick labels for top plots and y ticks for right plots.
#for ax in axs.flat:
    #ax.label_outer()

#fig.suptitle('Test stacking spectra')
```


0 degree
45 degree
60 degree
75 degree
90 degree
111 degree
135 degree
Simulated counts
Energy (keV)
0
1000
2000
3000
4000
5000
6000
6780
6800
6820

In [8]:

```
# All angles on one plot

#plt.style.use('bmh')
plt.style.use('seaborn-white')

# Create the plot and add the different datasets
fig, ax = plt.subplots(figsize=(15,10))
l0 = ax.hist(eGamma[0][:nGenerated-1], label='0$\degree$',   bins=100, range=(6765,6840), histtype='step'
, stacked=True, fill=True, linewidth=1.2)
l1 = ax.hist(eGamma[1][:nGenerated-1], label='45$\degree$',  bins=100, range=(6765,6840), histtype='step'
, stacked=True, fill=True, linewidth=1.2)
l2 = ax.hist(eGamma[2][:nGenerated-1], label='60$\degree$',  bins=100, range=(6765,6840), histtype='step'
, stacked=True, fill=True, linewidth=1.2)
l3 = ax.hist(eGamma[3][:nGenerated-1], label='75$\degree$',  bins=100, range=(6765,6840), histtype='step'
, stacked=True, fill=True, linewidth=1.2)
l4 = ax.hist(eGamma[4][:nGenerated-1], label='90$\degree$',  bins=100, range=(6765,6840), histtype='step'
, stacked=True, fill=True, linewidth=1.2)
l5 = ax.hist(eGamma[5][:nGenerated-1], label='111$\degree$', bins=100, range=(6765,6840), histtype='step'
, stacked=True, fill=True, linewidth=1.2)
l6 = ax.hist(eGamma[6][:nGenerated-1], label='135$\degree$', bins=100, range=(6765,6840), histtype='step'
, stacked=True, fill=True, linewidth=1.2, color='gray')

# Formatting
title = 'Monte Carlo Generated Gamma Spectrum for all angles, tau = '+str(tau)+' fs'
#ax.set_title(title, fontsize=30)
ax.set_ylabel('Simulated counts', fontsize=30)
ax.set_xlabel('Gamma energy (keV)', fontsize=30)
ax.tick_params(labelsize=30)
ax.legend(loc='best', shadow=True, fontsize=30)
#plt.xlim(6760, 6855)

plt.show()
```



## 5. Aggregate shift data

For this, I repeat the earlier cell where I generate the doppler shifted gammas (step 3) for each of the given lifetimes. With each, I copy the energy array to this cell to store the shifted lifetimes. Therefore, I have all of the data in one place to continue the analysis. Each code block represents a calculation for a different target.

This is the tedious part, as you are essentially running the earlier cell over and over and over and over again - each time copying the energies generated earlier into the appropriate locations in their distribution down here. However, when you're done, all the data is here in this one place. For a backup I'm also copying the data to an excel file.

**After this has been entered, don't change the data in this section**

In [9]:

```
# Ta30 target


# Calculate the full shift gamma ray energies
fullShift = np.zeros(7)

for angle in range(len(expAngles)):
    detectorAngle = expAngles[angle]*(math.pi/180.0)
    shift = eGamma0 * (1 + (v0/3000)*math.cos(detectorAngle))
    fullShift[angle] = round(shift,3)

    #DEBUG
    #print(round(shift,3))


# Energies for lifetimes and angles
# lifetimes = 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 1.2, 1.5, 2.0, 5.0
# angles = 0, 45, 60, 75, 90, 111, 135


energiesTa30 = np.zeros((15, 7))
energiesTa30[0][:] = fullShift[:]     # full shift
energiesTa30[1][:] = [6819.529, 6811.788, 6806.303, 6799.951, 6793.088, 6783.635, 6774.404]    # tau = 0.
1 fs
energiesTa30[2][:] = [6819.463, 6811.720, 6806.273, 6799.923, 6793.090, 6783.685, 6774.452]    # tau = 0.
2 fs
energiesTa30[3][:] = [6819.432, 6811.707, 6806.242, 6799.853, 6793.093, 6783.690, 6774.497]    # tau = 0.
3 fs
energiesTa30[4][:] = [6819.376, 6811.649, 6806.204, 6799.843, 6793.074, 6783.730, 6774.541]    # tau = 0.
4 fs
energiesTa30[5][:] = [6819.321, 6811.619, 6806.144, 6799.803, 6793.060, 6783.753, 6774.594]    # tau = 0.
5 fs
energiesTa30[6][:] = [6819.268, 6811.560, 6806.106, 6799.771, 6793.083, 6783.782, 6774.612]    # tau = 0.
6 fs
energiesTa30[7][:] = [6819.223, 6811.530, 6806.068, 6799.746, 6793.098, 6783.814, 6774.662]    # tau = 0.
7 fs
energiesTa30[8][:] = [6819.190, 6811.473, 6806.030, 6799.705, 6793.096, 6783.855, 6774.704]    # tau = 0.
8 fs
energiesTa30[9][:] = [6819.106, 6811.438, 6806.022, 6799.693, 6793.106, 6783.910, 6774.757]    # tau = 0.
9 fs
energiesTa30[10][:] = [6819.088, 6811.400, 6805.960, 6799.680, 6793.104, 6783.904, 6774.788]   # tau = 1.
0 fs
energiesTa30[11][:] = [6818.967, 6811.313, 6805.890, 6799.625, 6793.084, 6783.977, 6774.871]   # tau = 1.
2 fs
energiesTa30[12][:] = [6818.840, 6811.176, 6805.820, 6799.572, 6793.130, 6784.080, 6774.990]   # tau = 1.
5 fs
energiesTa30[13][:] = [6818.616, 6811.053, 6805.695, 6799.454, 6793.096, 6784.170, 6775.192]   # tau = 2.
0 fs
energiesTa30[14][:] = [6818.081, 6810.639, 6805.392, 6799.359, 6793.110, 6784.395, 6775.545]   # tau = 5.
0 fs

# DEBUG
#print(energiesTa30)
```

In [10]:

```
# Ta10 target


# Calculate the full shift gamma ray energies
fullShift = np.zeros(7)

for angle in range(len(expAngles)):
    detectorAngle = expAngles[angle]*(math.pi/180.0)
    shift = eGamma0 * (1 + (v0/3000)*math.cos(detectorAngle))
    fullShift[angle] = round(shift,3)

    #DEBUG
    #print(round(shift,3))


# Energies for lifetimes and angles
# lifetimes = 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 1.2, 1.5, 2.0, 5.0
# angles = 0, 45, 60, 75, 90, 111, 135


energiesTa10 = np.zeros((15, 7))
energiesTa10[0][:] = fullShift[:]     # full shift
energiesTa10[1][:] = [6819.536, 6811.766, 6806.323, 6799.936, 6793.092, 6783.627, 6774.415]    # tau = 0.
1 fs
energiesTa10[2][:] = [6819.478, 6811.740, 6806.267, 6799.891, 6793.094, 6783.647, 6774.458]    # tau = 0.
2 fs
energiesTa10[3][:] = [6819.419, 6811.697, 6806.213, 6799.856, 6793.093, 6783.731, 6774.521]    # tau = 0.
3 fs
energiesTa10[4][:] = [6819.355, 6811.635, 6806.188, 6799.843, 6793.118, 6783.736, 6774.551]    # tau = 0.
4 fs
energiesTa10[5][:] = [6819.314, 6811.577, 6806.145, 6799.809, 6793.088, 6783.776, 6774.591]    # tau = 0.
5 fs
energiesTa10[6][:] = [6819.249, 6811.562, 6806.123, 6799.791, 6793.103, 6783.802, 6774.639]    # tau = 0.
6 fs
energiesTa10[7][:] = [6819.214, 6811.509, 6806.049, 6799.731, 6793.082, 6783.831, 6774.672]    # tau = 0.
7 fs
energiesTa10[8][:] = [6819.174, 6811.472, 6806.035, 6799.727, 6793.103, 6783.893, 6774.739]    # tau = 0.
8 fs
energiesTa10[9][:] = [6819.111, 6811.423, 6805.994, 6799.687, 6793.077, 6783.916, 6774.783]    # tau = 0.
9 fs
energiesTa10[10][:] = [6819.033, 6811.394, 6805.959, 6799.643, 6793.083, 6783.922, 6774.826]   # tau = 1.
0 fs
energiesTa10[11][:] = [6818.954, 6811.322, 6805.896, 6799.604, 6793.109, 6783.992, 6774.908]   # tau = 1.
2 fs
energiesTa10[12][:] = [6818.834, 6811.199, 6805.807, 6799.557, 6793.090, 6784.087, 6775.003]   # tau = 1.
5 fs
energiesTa10[13][:] = [6818.635, 6811.014, 6805.659, 6799.439, 6793.095, 6784.202, 6775.152]   # tau = 2.
0 fs
energiesTa10[14][:] = [6818.287, 6810.778, 6805.516, 6799.387, 6793.109, 6784.263, 6775.456]   # tau = 5.
0 fs

# DEBUG
#print(energies)
```

In [11]:

```
# Mo30 target


# Calculate the full shift gamma ray energies
fullShift = np.zeros(7)

for angle in range(len(expAngles)):
    detectorAngle = expAngles[angle]*(math.pi/180.0)
    shift = eGamma0 * (1 + (v0/3000)*math.cos(detectorAngle))
    fullShift[angle] = round(shift,3)

    #DEBUG
    #print(round(shift,3))


# Energies for lifetimes and angles
# lifetimes = 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 1.2, 1.5, 2.0, 5.0
# angles = 0, 45, 60, 75, 90, 111, 135


energiesMo30 = np.zeros((15, 7))
energiesMo30[0][:] = fullShift[:]     # full shift
energiesMo30[1][:] = [6819.539, 6811.773, 6806.310, 6799.934, 6793.095, 6783.648, 6774.425]    # tau = 0.
1 fs
energiesMo30[2][:] = [6819.450, 6811.722, 6806.233, 6799.900, 6793.101, 6783.675, 6774.468]    # tau = 0.
2 fs
energiesMo30[3][:] = [6819.386, 6811.695, 6806.202, 6799.849, 6793.104, 6783.725, 6774.547]    # tau = 0.
3 fs
energiesMo30[4][:] = [6819.324, 6811.628, 6806.160, 6799.817, 6793.123, 6783.750, 6774.569]    # tau = 0.
4 fs
energiesMo30[5][:] = [6819.272, 6811.579, 6806.144, 6799.787, 6793.103, 6783.795, 6774.650]    # tau = 0.
5 fs
energiesMo30[6][:] = [6819.194, 6811.516, 6806.093, 6799.748, 6793.094, 6783.821, 6774.692]    # tau = 0.
6 fs
energiesMo30[7][:] = [6819.154, 6811.432, 6806.030, 6799.730, 6793.092, 6783.876, 6774.754]    # tau = 0.
7 fs
energiesMo30[8][:] = [6819.077, 6811.425, 6805.984, 6799.675, 6793.105, 6783.930, 6774.789]    # tau = 0.
8 fs
energiesMo30[9][:] = [6819.039, 6811.354, 6805.950, 6799.650, 6793.115, 6783.944, 6774.842]    # tau = 0.
9 fs
energiesMo30[10][:] = [6818.954, 6811.330, 6805.919, 6799.623, 6793.083, 6783.986, 6774.891]   # tau = 1.
0 fs
energiesMo30[11][:] = [6818.866, 6811.200, 6805.832, 6799.585, 6793.098, 6784.051, 6774.985]   # tau = 1.
2 fs
energiesMo30[12][:] = [6818.707, 6811.083, 6805.730, 6799.488, 6793.092, 6784.120, 6775.113]   # tau = 1.
5 fs
energiesMo30[13][:] = [6818.498, 6810.908, 6805.561, 6799.371, 6793.093, 6784.245, 6775.277]   # tau = 2.
0 fs
energiesMo30[14][:] = [6818.017, 6810.573, 6805.316, 6799.334, 6793.113, 6784.413, 6775.647]   # tau = 5.
0 fs

# DEBUG
#print(energies)
```

In [12]:

```
# W30 target


# Calculate the full shift gamma ray energies
fullShift = np.zeros(7)

for angle in range(len(expAngles)):
    detectorAngle = expAngles[angle]*(math.pi/180.0)
    shift = eGamma0 * (1 + (v0/3000)*math.cos(detectorAngle))
    fullShift[angle] = round(shift,3)

    #DEBUG
    #print(round(shift,3))


# Energies for lifetimes and angles
# lifetimes = 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 1.2, 1.5, 2.0, 5.0
# angles = 0, 45, 60, 75, 90, 111, 135


energiesW30 = np.zeros((15, 7))
energiesW30[0][:] = fullShift[:]     # full shift
energiesW30[1][:] = [6819.539, 6811.779, 6806.313, 6799.914, 6793.096, 6783.630, 6774.400]    # tau = 0.1
fs
energiesW30[2][:] = [6819.466, 6811.743, 6806.263, 6799.916, 6793.096, 6783.674, 6774.482]    # tau = 0.2
fs
energiesW30[3][:] = [6819.429, 6811.688, 6806.218, 6799.858, 6793.085, 6783.711, 6774.515]    # tau = 0.3
fs
energiesW30[4][:] = [6819.356, 6811.643, 6806.179, 6799.807, 6793.097, 6783.753, 6774.542]    # tau = 0.4
fs
energiesW30[5][:] = [6819.281, 6811.579, 6806.142, 6799.805, 6793.101, 6783.788, 6774.608]    # tau = 0.5
fs
energiesW30[6][:] = [6819.220, 6811.532, 6806.068, 6799.746, 6793.085, 6783.833, 6774.670]    # tau = 0.6
fs
energiesW30[7][:] = [6819.196, 6811.469, 6806.039, 6799.722, 6793.084, 6783.858, 6774.701]    # tau = 0.7
fs
energiesW30[8][:] = [6819.140, 6811.429, 6806.000, 6799.680, 6793.111, 6783.899, 6774.749]    # tau = 0.8
fs
energiesW30[9][:] = [6819.062, 6811.401, 6805.958, 6799.661, 6793.097, 6783.933, 6774.801]    # tau = 0.9
fs
energiesW30[10][:] = [6819.005, 6811.326, 6805.923, 6799.628, 6793.082, 6783.957, 6774.859]   # tau = 1.0
fs
energiesW30[11][:] = [6818.918, 6811.250, 6805.832, 6799.566, 6793.103, 6784.030, 6774.925]   # tau = 1.2
fs
energiesW30[12][:] = [6818.758, 6811.148, 6805.753, 6799.499, 6793.105, 6784.119, 6775.042]   # tau = 1.5
fs
energiesW30[13][:] = [6818.566, 6810.995, 6805.639, 6799.425, 6793.105, 6784.206, 6775.231]   # tau = 2.0
fs
energiesW30[14][:] = [6818.300, 6810.862, 6805.591, 6799.443, 6793.134, 6784.269, 6775.382]   # tau = 5.0
fs

# DEBUG
#print(energies)
```

In [13]:

```
# Ta2N3 Target


# Calculate the full shift gamma ray energies
fullShift = np.zeros(7)

for angle in range(len(expAngles)):
    detectorAngle = expAngles[angle]*(math.pi/180.0)
    shift = eGamma0 * (1 + (v0/3000)*math.cos(detectorAngle))
    fullShift[angle] = round(shift,3)

    #DEBUG
    #print(round(shift,3))


# Energies for lifetimes and angles
# lifetimes = 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 1.2, 1.5, 2.0, 5.0
# angles = 0, 45, 60, 75, 90, 111, 135


energiesTa2N3 = np.zeros((15, 7))
energiesTa2N3[0][:] = fullShift[:]     # full shift
energiesTa2N3[1][:] = [6819.507, 6811.762, 6806.303, 6799.914, 6793.123, 6783.652, 6774.428]    # tau =
 0.1 fs
energiesTa2N3[2][:] = [6819.464, 6811.707, 6806.237, 6799.887, 6793.102, 6783.703, 6774.474]    # tau =
 0.2 fs
energiesTa2N3[3][:] = [6819.381, 6811.653, 6806.200, 6799.834, 6793.098, 6783.733, 6774.528]    # tau =
 0.3 fs
energiesTa2N3[4][:] = [6819.325, 6811.615, 6806.143, 6799.788, 6793.099, 6783.777, 6774.609]    # tau =
 0.4 fs
energiesTa2N3[5][:] = [6819.263, 6811.539, 6806.083, 6799.752, 6793.107, 6783.841, 6774.648]    # tau =
 0.5 fs
energiesTa2N3[6][:] = [6819.180, 6811.490, 6806.022, 6799.704, 6793.085, 6783.865, 6774.715]    # tau =
 0.6 fs
energiesTa2N3[7][:] = [6819.109, 6811.422, 6806.006, 6799.663, 6793.104, 6783.913, 6774.776]    # tau =
 0.7 fs
energiesTa2N3[8][:] = [6819.068, 6811.381, 6805.943, 6799.648, 6793.091, 6783.983, 6774.828]    # tau =
 0.8 fs
energiesTa2N3[9][:] = [6819.017, 6811.344, 6805.945, 6799.599, 6793.102, 6783.999, 6774.855]    # tau =
 0.9 fs
energiesTa2N3[10][:] = [6818.929, 6811.309, 6805.865, 6799.578, 6793.118, 6784.027, 6774.918]   # tau =
 1.0 fs
energiesTa2N3[11][:] = [6818.934, 6811.305, 6805.875, 6799.598, 6793.079, 6784.059, 6774.927]   # tau =
 1.2 fs
energiesTa2N3[12][:] = [6818.728, 6811.070, 6805.717, 6799.472, 6793.100, 6784.186, 6775.112]   # tau =
 1.5 fs
energiesTa2N3[13][:] = [6818.583, 6810.990, 6805.610, 6799.403, 6793.094, 6784.223, 6775.233]   # tau =
 2.0 fs
energiesTa2N3[14][:] = [6818.643, 6811.161, 6805.805, 6799.590, 6793.121, 6784.024, 6775.016]   # tau =
 5.0 fs

# DEBUG
#print(energies)
```

In [14]:

```
# Pure Ta Target


# Calculate the full shift gamma ray energies
fullShift = np.zeros(7)

for angle in range(len(expAngles)):
    detectorAngle = expAngles[angle]*(math.pi/180.0)
    shift = eGamma0 * (1 + (v0/3000)*math.cos(detectorAngle))
    fullShift[angle] = round(shift,3)

    #DEBUG
    #print(round(shift,3))


# Energies for lifetimes and angles
# lifetimes = 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 1.2, 1.5, 2.0, 5.0
# angles = 0, 45, 60, 75, 90, 111, 135


energiesPureTa = np.zeros((15, 7))
energiesPureTa[0][:] = fullShift[:]     # full shift
energiesPureTa[1][:] = [6819.524, 6811.783, 6806.310, 6799.940, 6793.092, 6783.632, 6774.420]    # tau =
 0.1 fs
energiesPureTa[2][:] = [6819.467, 6811.730, 6806.269, 6799.908, 6793.092, 6783.665, 6774.476]    # tau =
 0.2 fs
energiesPureTa[3][:] = [6819.437, 6811.692, 6806.215, 6799.868, 6793.096, 6783.734, 6774.515]    # tau =
 0.3 fs
energiesPureTa[4][:] = [6819.360, 6811.634, 6806.169, 6799.831, 6793.085, 6783.747, 6774.560]    # tau =
 0.4 fs
energiesPureTa[5][:] = [6819.292, 6811.612, 6806.151, 6799.773, 6793.124, 6783.794, 6774.613]    # tau =
 0.5 fs
energiesPureTa[6][:] = [6819.249, 6811.552, 6806.120, 6799.764, 6793.100, 6783.836, 6774.672]    # tau =
 0.6 fs
energiesPureTa[7][:] = [6819.169, 6811.493, 6806.044, 6799.740, 6793.091, 6783.850, 6774.696]    # tau =
 0.7 fs
energiesPureTa[8][:] = [6819.123, 6811.438, 6806.010, 6799.697, 6793.107, 6783.909, 6774.755]    # tau =
 0.8 fs
energiesPureTa[9][:] = [6819.075, 6811.395, 6805.969, 6799.687, 6793.094, 6783.939, 6774.810]    # tau =
 0.9 fs
energiesPureTa[10][:] = [6819.021, 6811.351, 6805.951, 6799.623, 6793.090, 6783.982, 6774.847]   # tau =
 1.0 fs
energiesPureTa[11][:] = [6818.934, 6811.305, 6805.875, 6799.598, 6793.079, 6784.059, 6774.927]   # tau =
 1.2 fs
energiesPureTa[12][:] = [6818.758, 6811.148, 6805.753, 6799.499, 6793.105, 6784.119, 6775.042]   # tau =
 1.5 fs
energiesPureTa[13][:] = [6818.613, 6811.025, 6805.678, 6799.445, 6793.090, 6784.199, 6775.174]   # tau =
 2.0 fs
energiesPureTa[14][:] = [6818.461, 6810.925, 6805.606, 6799.478, 6793.103, 6784.163, 6775.273]   # tau =
 5.0 fs

# DEBUG
#print(energies)
```

## 6. Plotting shifts to dermine the attenuation factors

In [15]:

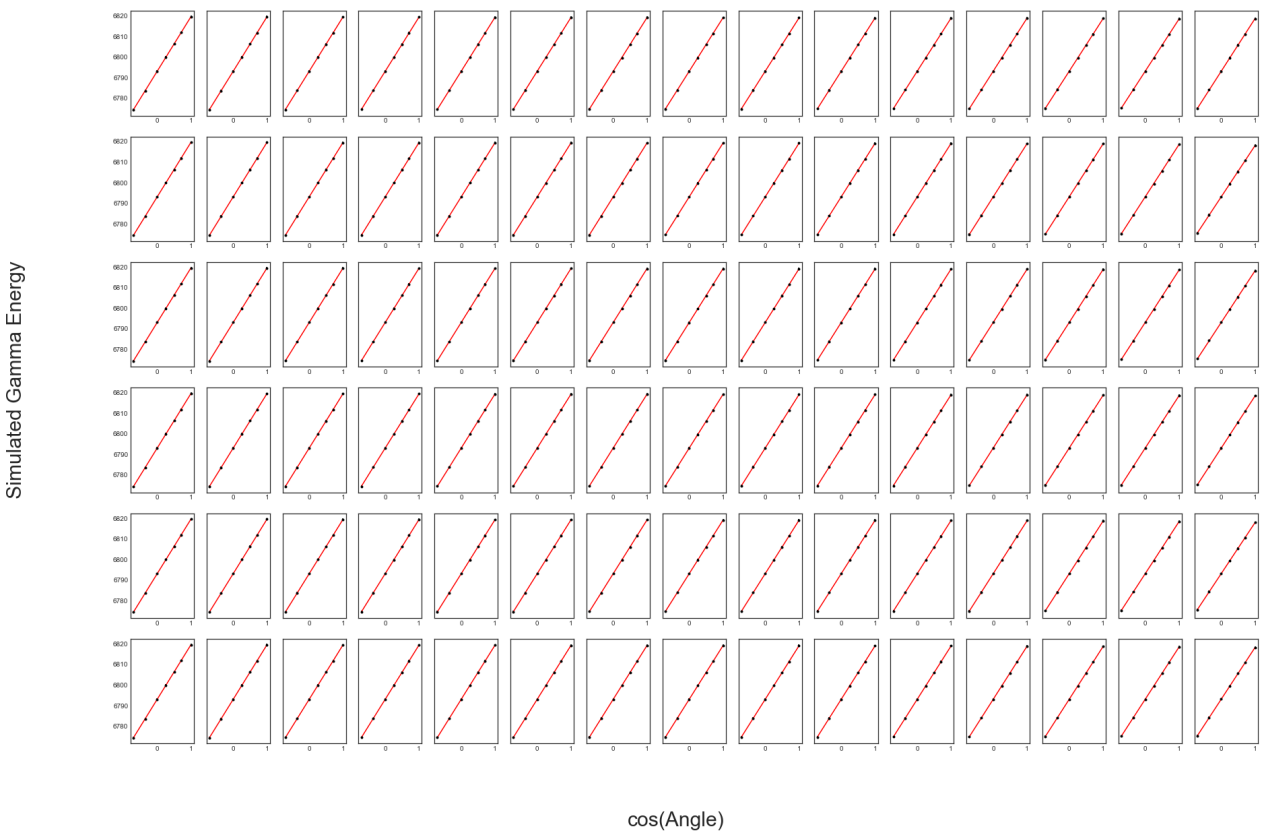
```
# Some preliminary definition of data structures

slopes = np.zeros((6, 15))      # Slopes for doppler shift plots
slopeUncs = np.zeros((6,15))    # uncertainties on doppler shift slopes
attFactors = np.zeros((6, 15))  # F(tau) for different lifetime/target combo
attUncs = np.zeros((6, 15))     # Uncertainties on F(tau)

cosAngles = np.zeros(7)         # cos(angle)
for angle in range(len(expAngles)):
    cosAngles[angle] = math.cos(expAngles[angle]*(math.pi/180.0))

# The generated gamma energies for each of the targets (entered manually in section 5)
targets = [energiesTa2N3, energiesTa30, energiesTa10, energiesPureTa, energiesMo30, energiesW30]

# Lifetimes in a list
lifetimes = [0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 1.2, 1.5, 2.0, 5.0]

# DEBUG
#for backing in targets:
#    print(backing[1][1])
```

In [16]:

```
# Pretty self explanatory
def line(x, m, b):
    return m*x + b
```

In [17]:

```
# Create and fit all doppler shift plots

# 6 target materials and 15 lifetimes tested
fig, axs = plt.subplots(6, 15, figsize=(30,20), sharex=False, sharey=True)


yerr = 0.5              # Representative error on centroid positioning from experiment


for i in range(0, 6):            # 6 targets
    for j in range(0, 15):       # 15 lifetimes

        y = targets[i][j][:]     # Get the current lifetime/target combination of energies
        #DEBUG
        #print(y)

        axs[i][j].errorbar(cosAngles, y, yerr, fmt='.k')   # Plot

        dy = [0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5]                                # Sigma for the calculati
on
        parameters, covariance = sopt.curve_fit(line, cosAngles, y, sigma=dy)   # Fit the line to each da
taset
        fitX = np.linspace(-0.7,1.0,20)
        fitY = line(fitX, parameters[0], parameters[1])
        axs[i][j].plot(fitX,fitY,'r-')                     # Plot the fit result

        paramUncs = np.sqrt(np.diag(covariance))           # Calculate the uncertainties in fit parameter
s
        slopes[i][j] = parameters[0]
        slopeUncs[i][j] = paramUncs[0]

fig.suptitle('Doppler Shift for different lifetime/target combinations', fontsize=40)
fig.text(0.5, 0.04, 'cos(Angle)', ha='center', fontsize=30)
fig.text(0.04, 0.5, 'Simulated Gamma Energy', va='center', rotation='vertical', fontsize=30)
plt.show()

#DEBUG
#print(slopes)
#print(slopeUncs)
```

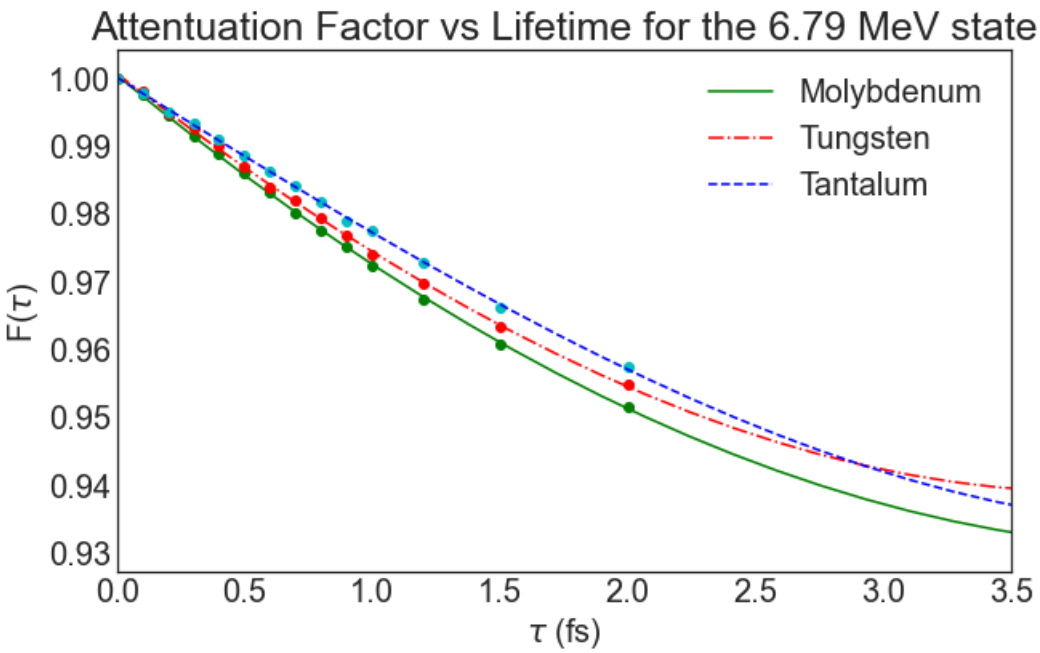



In [18]:

```
# Calculate the attenuation factors from the slopes

for i in range(0, 6):
    for j in range(0, 15):
        factor = slopes[i][j]/slopes[i][0]
        uncertainty = factor * np.sqrt((slopeUncs[i][j]/slopes[i][j])**2 + (slopeUncs[i][0]/slopes[i][0])
**2)
        attFactors[i][j] = round(factor, 4)
        attUncs[i][j] = round(uncertainty, 4)

# DEBUG
#print(attFactors)
#print(attUncs)
```

In [19]:

```
# Plot the attenuation factors
#plt.errorbar(lifetimes, attFactors[1][:], attUncs[1][:], marker='o', ls='none', ms=5, label='Ta30')#, fmt='.k')
#plt.errorbar(lifetimes, attFactors[2][:], attUncs[2][:], marker='o', ls='none', ms=5, label='Ta10')#, fmt='.k')
#plt.errorbar(lifetimes, attFactors[4][:], attUncs[4][:], marker='o', ls='none', ms=5, label='Mo30')#, fmt='.k')
#plt.errorbar(lifetimes, attFactors[5][:], attUncs[5][:], marker='o', ls='none', ms=5, label='W30')#, fmt='.k')

fig_size = plt.rcParams["figure.figsize"]
fig_size[0] = 10
fig_size[1] = 6
plt.rcParams["figure.figsize"] = fig_size

# Beautify
plt.title(r'Attentuation Factor vs Lifetime for the 6.79 MeV state', fontsize=25)
plt.xlabel(r'${\tau}$ (fs)', fontsize=20)
plt.ylabel(r'F(${\tau}$)', fontsize=20)
plt.tick_params(labelsize=20)


# Interpolation
x = np.linspace(0, 5, num=15, endpoint=True)
xnew = np.linspace(0, 5, 30)
splMo = UnivariateSpline(lifetimes, attFactors[4][:])
splMo.set_smoothing_factor(1)
splW = UnivariateSpline(lifetimes, attFactors[5][:])
splW.set_smoothing_factor(1)
splTa30 = UnivariateSpline(lifetimes, attFactors[1][:])
splTa30.set_smoothing_factor(1)
splTa10 = UnivariateSpline(lifetimes, attFactors[2][:])
splTa10.set_smoothing_factor(1)
plt.plot(lifetimes, attFactors[4][:], 'go')
plt.plot(xnew, splMo(xnew), 'g-', label='Molybdenum')
plt.plot(lifetimes, attFactors[5][:], 'ro')
plt.plot(xnew, splW(xnew), 'r-.', label='Tungsten')
plt.plot(lifetimes, attFactors[1][:], 'co')
plt.plot(xnew, splTa30(xnew), 'b--', label='Tantalum')
# plt.plot(lifetimes, attFactors[2][:], 'bo')
# plt.plot(xnew, splTa10(xnew), 'b--', label='Tantalum (low)')


plt.legend(loc='upper right', shadow=True, fontsize=20)

plt.xlim(0,3.5)
plt.show()
```




In [20]:

```
def lookup(F, err, spl):

    test = 0
    options = []                    # lifetimes in the correct range
    value = []                      # lifetime corresponding to F
    z = np.linspace(0, 5, 500)      # potential lifetimes
    for lifetime in z:
        if (spl(lifetime) < (F+err)) and (spl(lifetime) > (F-err)):
            options.append(lifetime)                                    # lifetimes in F+/-err
        if (spl(lifetime) < (F+0.0001) and (spl(lifetime) > (F-0.0001))):
            value.append(lifetime)                                      # lifetime for F
            test = 1

    if value == []:
        for lifetime in z:
            if (spl(lifetime) < (F+0.0005) and (spl(lifetime) > (F-0.0005))):
                value.append(lifetime)                                      # lifetime for F
                test = 2

    if value == []:
        for lifetime in z:
            if (spl(lifetime) < (F+0.01) and (spl(lifetime) > (F-0.0001))):
                value.append(lifetime)                                      # lifetime for F
                test = 3

    # DEBUG
    print(test)
    print(value)

    #print(options)
    tau = np.mean(value[:])
    print('Tau =    ', round(tau, 3))
    print('F(tau) = ', round(float(spl(tau)), 3))
    print('Range:    (', round(options[0], 3), ', ', round(options[len(options)-1], 3), ')\n', sep='')
```

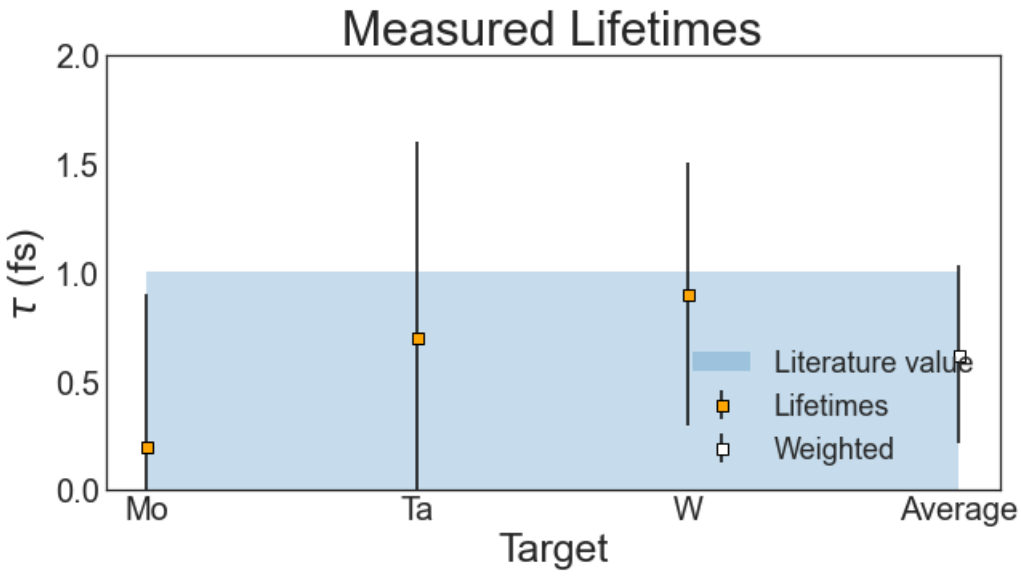
In [21]:

```
# Find values
lookup(0.995, 0.019, splMo)
lookup(0.978, 0.015, splW)
lookup(0.942, 0.04, splTa10)
lookup(0.983, 0.019, splTa30)
```

```
1
[0.18036072144288576]
Tau =     0.18
F(tau) =  0.995
Range:    (0.0, 0.862)

1
[0.8517034068136272]
Tau =     0.852
F(tau) =  0.978
Range:    (0.281, 1.523)

1
[3.3366733466933867, 3.346693386773547, 3.356713426853707, 4.719438877755511, 4.7294589178356
71, 4.739478957915831]
Tau =     4.038
F(tau) =  0.94
Range:    (0.752, 5.0)

1
[0.7414829659318637]
Tau =     0.741
F(tau) =  0.983
Range:    (0.0, 1.633)
```

In [26]:

```
targs = ['Mo', 'Ta', 'W']
taus = [0.2, 0.7, 0.9]
tauerrs = [0.7, 0.9, 0.6]

weighted = ['Average']
weight = [1/(x**2) for x in tauerrs]
weightedAverage = np.average(taus, weights = weight)
weightedError = np.sqrt(1/(1/(tauerrs[0]**2) + 1/(tauerrs[1]**2) + 1/(tauerrs[2]**2)))

#lit = ['Lit value']
#litVal = [1]
#litErr = [1.5]

from pylab import *
# Create the plot, set some options, and add the datasets
fig, ax = plt.subplots(figsize=(10,5))
rcParams['xtick.labelsize'] = 20
rcParams['ytick.labelsize'] = 20
ax.errorbar(targs, taus, yerr=tauerrs, fmt='.k', label='Lifetimes',marker='s', markersize=7, markerfaceco
lor='orange', markeredgecolor='black')   # Plot
ax.errorbar(weighted, weightedAverage, yerr=weightedError, fmt='.k', label='Weighted',marker='s', markers
ize=7, markerfacecolor='w', markeredgecolor='black')   # Plot
#ax.errorbar(lit, litVal, yerr=litErr, fmt='.k', label='Lit. value',marker='s', markersize=7, markerfacec
olor='black', markeredgecolor='black')   # Plot

# Formatting
title = 'Measured Lifetimes'
ax.set_title(title, fontsize=30)
ax.set_ylabel(r'$\tau$ (fs)', fontsize=25)
ax.set_xlabel('Target', fontsize=25)
ax.tick_params(labelsize=20)

alltargs = targs+weighted#+lit
ax.fill_between(alltargs, 1.0, 0, alpha=.25, label='Literature value')

legend(loc='lower right',fontsize=18)
plt.ylim(0,2)
plt.show()
```

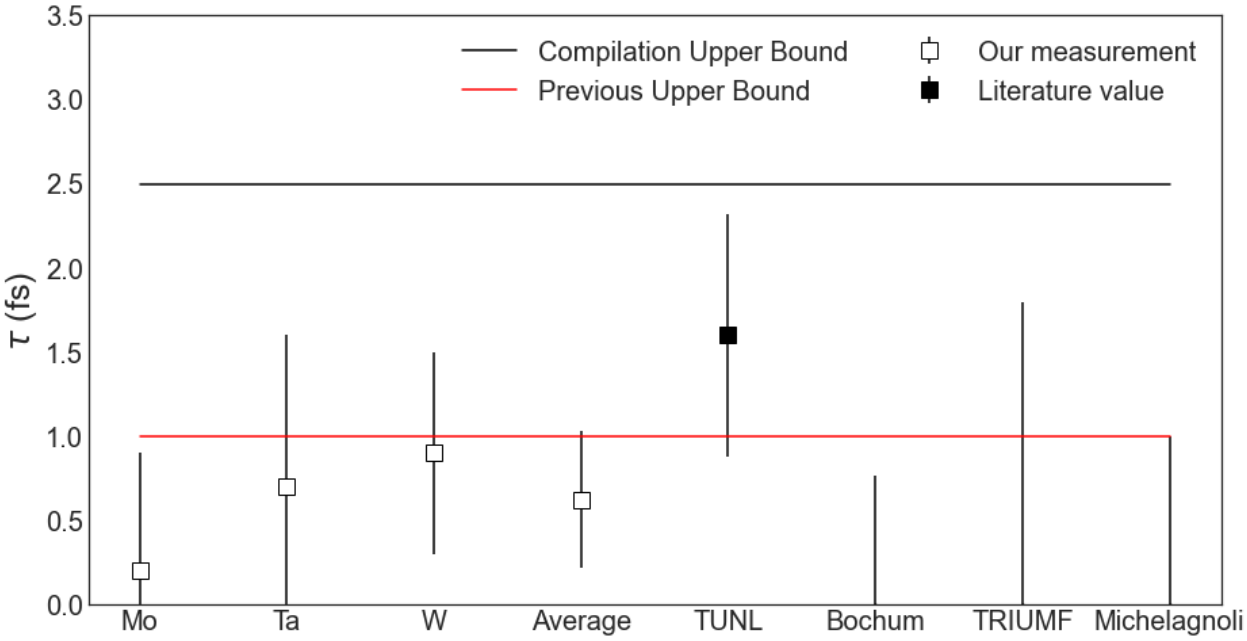


In [23]:

```
tunl = ['TUNL']
tunlVal = [1.6]
tunlErr = [0.72]

bochum = ['Bochum']
bochumVal = [0]
bochumErr = [0.77]

triumf = ['TRIUMF']
triumfVal = [0]
triumfErr = [1.8]

mich = ['Michelagnoli']
michVal = [0]
michErr = [1]
```

In [24]:

```
# Create the plot, set some options, and add the datasets
fig, ax = plt.subplots(figsize=(15,8))
rcParams['xtick.labelsize'] = 20
rcParams['ytick.labelsize'] = 20
ax.errorbar(targs, taus, yerr=tauerrs, fmt='.k', label='Our measurement',marker='s', markersize=12, marke
rfacecolor='w', markeredgecolor='black')   # Plot
ax.errorbar(weighted, weightedAverage, yerr=weightedError, fmt='.k',marker='s', markersize=12, markerface
color='w', markeredgecolor='black')   # Plot
ax.errorbar(tunl, tunlVal, yerr=tunlErr, fmt='.k',marker='s', label='Literature value', markersize=12, ma
rkerfacecolor='black', markeredgecolor='black')   # Plot
ax.errorbar(bochum, bochumVal, yerr=bochumErr, fmt='.k',marker='s', markersize=1, markerfacecolor='black'
, markeredgecolor='black')   # Plot
ax.errorbar(triumf, triumfVal, yerr=triumfErr, fmt='.k',marker='s', markersize=1, markerfacecolor='black'
, markeredgecolor='black')   # Plot
ax.errorbar(mich, michVal, yerr=michErr, fmt='.k',marker='s', markersize=1, markerfacecolor='black', mark
eredgecolor='black')   # Plot

# Formatting
title = 'Measured and Literature Lifetimes of the 6.79 MeV State in $^{15}$O'
#ax.set_title(title, fontsize=30)
ax.set_ylabel(r'$\tau$ (fs)', fontsize=25)
#ax.set_xlabel('Target', fontsize=25)
ax.tick_params(labelsize=20)

alltargs = targs+weighted+tunl+bochum+triumf+mich
ours = targs+weighted
#ax.fill_between(ours, 3.5, 0, alpha=0.25, color='g')
#ax.fill_between(alltargs, 2.5, 0, alpha=.25, label='Compilation')
plt.plot(alltargs, [2.5]*8, 'k-', label='Compilation Upper Bound')
plt.plot(alltargs, [1.0]*8, 'r-', label='Previous Upper Bound')

legend(loc='best',fontsize=20, ncol=2)
plt.ylim(0,3.5)
plt.show()
```




In [25]:

```
print(weightedAverage)
print(weightedError)
```

```
0.6232049947970862
0.4064516129032258
```

In [ ]:

```

```