ResponseMatrix

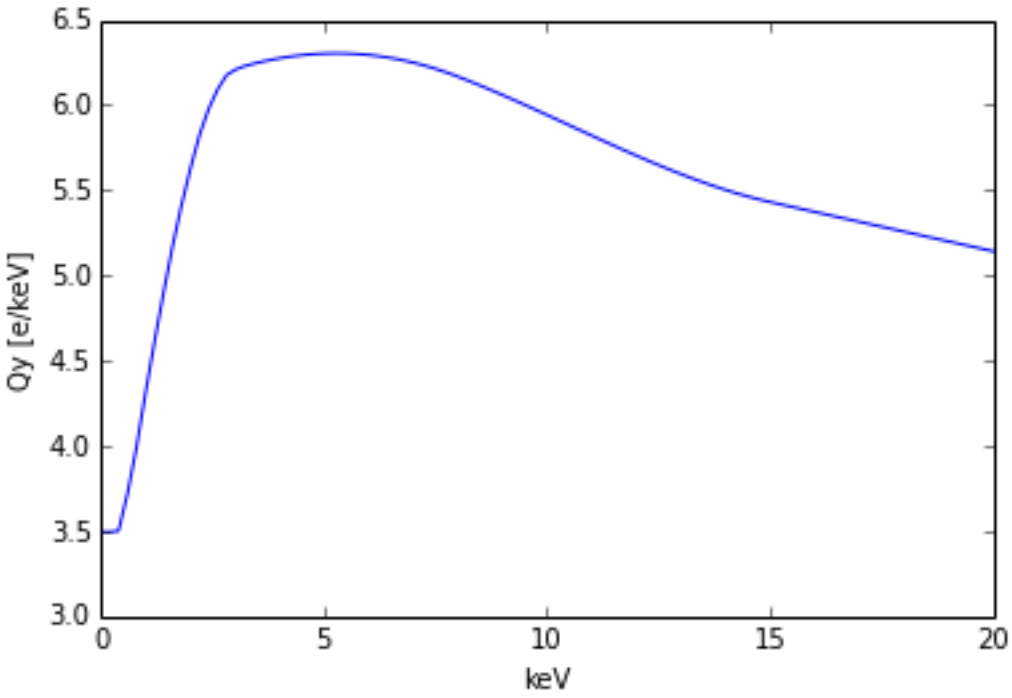
Chris Tunnell

April 2, 2014

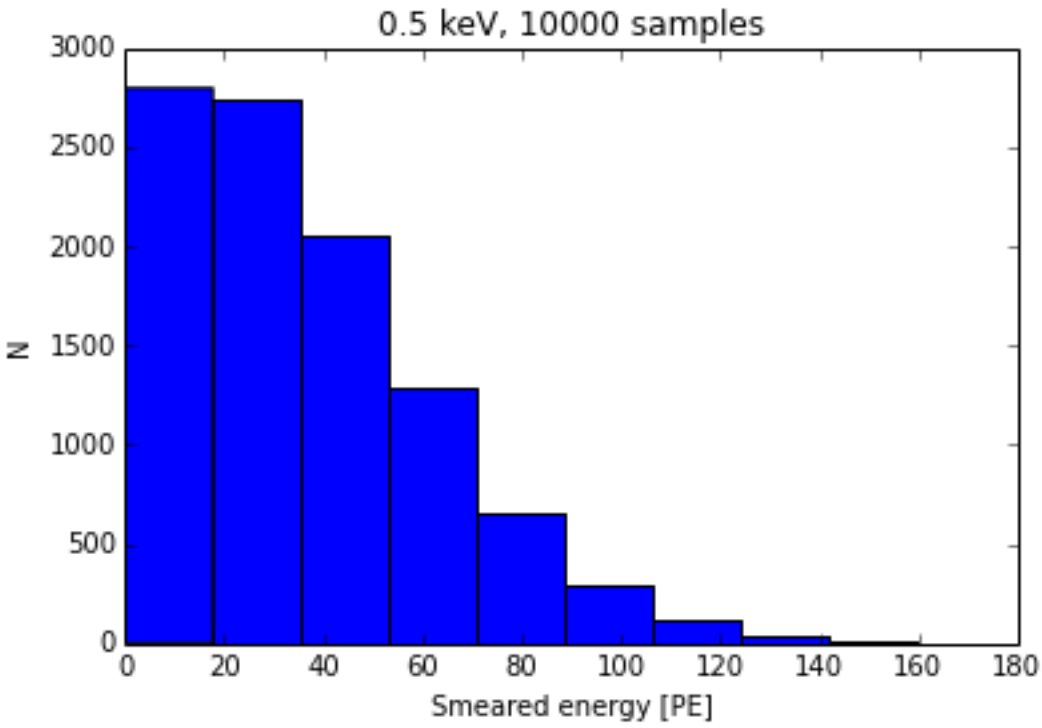
In [135]:
```
import numpy as np
import ROOT
import matplotlib.pyplot as plt
from matplotlib.ticker import NullFormatter
from matplotlib import cm
import pickle
```

In [59]:
```
Qyfile = ROOT.TFile('LightChargeYield_ER_NR.root')
def get_Qy(edep):
    g = Qyfile.Get('Qy_LM_run58_median')
    return g.Eval(edep)
```

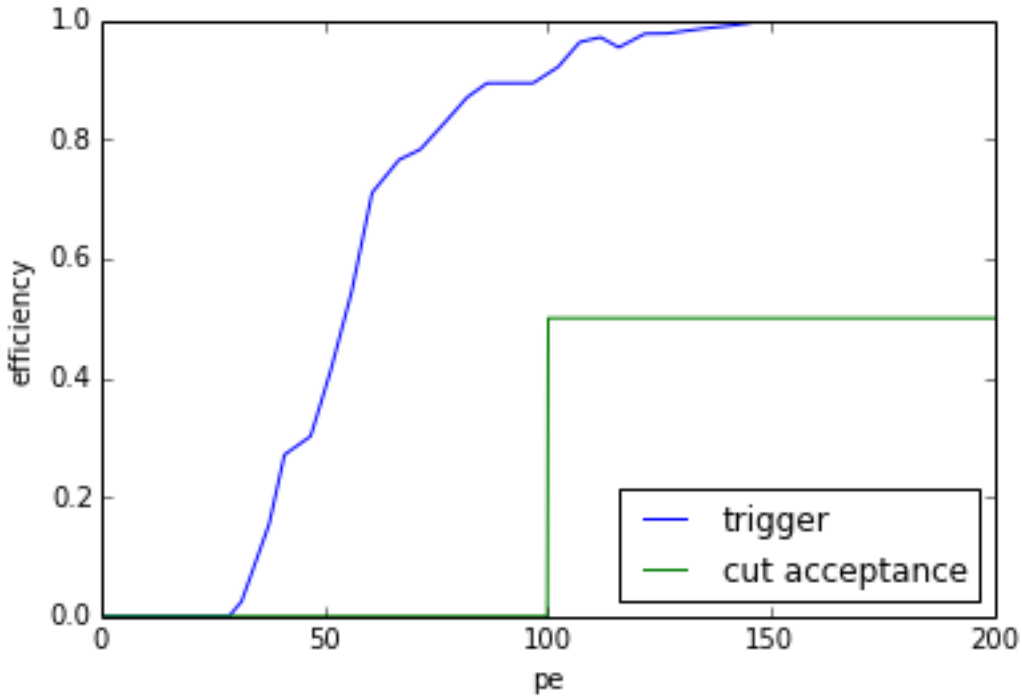
In [60]:
```
x = np.linspace(0, 20, 100)
y = [get_Qy(i) for i in x]
plt.plot(x, y)
plt.xlabel('keV')
plt.ylabel('Qy [e/keV]')
plt.savefig('limit_qy.png')
```



In [138]:
```
def S2(edep):
    """Find S2

    var: edep [kev]

    all electrons make to gas?
    should be truncated gauss
    """
    Ne = np.random.poisson(edep * get_Qy(edep))
    Ng = Ne
    #poisson

    if Ng == 0:
        return 0

    S2obs = np.random.normal(loc=19.67 * Ng,
                             scale=6.98*np.sqrt(Ng))
    if S2obs < 0:
        return 0

    return S2obs

def plot_S2(energy):
    N = 10000
    samples = [S2(energy) for i in range(N)]
    plt.hist(samples)
    plt.xlabel('Smeared energy [PE]')
    plt.ylabel('N')
    plt.title('%0.1f keV, %d samples' % (energy, N))
    plt.savefig('limit_pe.png')
    plt.show()

plot_S2(0.5)
```



In [62]:
```
def trigger_efficiency(pe):
    x = [1.524, 28.646, 31.442, 37.592, 41.044, 46.914, 51.039, 55.96, 60.645, 66.581,
    y = [0.0,  0.0, 0.025, 0.154, 0.271, 0.302, 0.402, 0.54, 0.71, 0.765, 0.783, 0.829

    return np.interp(pe, x, y)

def cut_acceptance(pe):
    return np.where(pe > pe.size*[100], 0.5, 0.0)


x = np.linspace(0, 200, 1000)
plt.plot(x, trigger_efficiency(x), label='trigger')
plt.plot(x, cut_acceptance(x), label='cut acceptance')


plt.ylim(0,1)
plt.xlabel('pe')
plt.ylabel('efficiency')
plt.legend(loc=4)
plt.savefig('limit_acceptance.png')
```



In [122]:
```
true_energy_bins = np.linspace(0, 10, 100) # keV
measured_energy_bins = np.linspace(0, 1000, 100) # pe

Z, binsx, binsy = np.histogram2d([], [],
                    bins=(true_energy_bins,
                        measured_energy_bins))
N = 0

def c(x):
    """ get center"""
    return 0.5*(x[1:]+x[:-1])
```

In [64]:
```
def add_points(size = 10):
    trial_energy = true_energy_bins[0] + np.random.random(size) * (true_energy_bins[-1

    true_spectra, recon_spectra = smeared_spectra(trial_energy)

    z, binsx, binsy = np.histogram2d(true_spectra, recon_spectra,
                        bins=(true_energy_bins,
                              measured_energy_bins))

    global N
    global Z
    N += size
    Z += z
    return z
```

In [102]:
```
def smeared_spectra(true_energy):
    """Smear energy

     true_energy - keV
    """
    size = true_energy.size
    smeared_energy = np.array([S2(x) for x in true_energy])
```

```
    energies = np.vstack((true_energy, smeared_energy.copy()))

    mask = np.random.random(size) < trigger_efficiency(smeared_energy)
    mask = mask & (smeared_energy != 0)
    energies = energies.compress(mask, axis=1)

    # Cut acceptance
    mask =  np.random.random(energies.shape[1]) < cut_acceptance(energies[1])

    energies = energies.compress(mask, axis=1)

    return energies
```

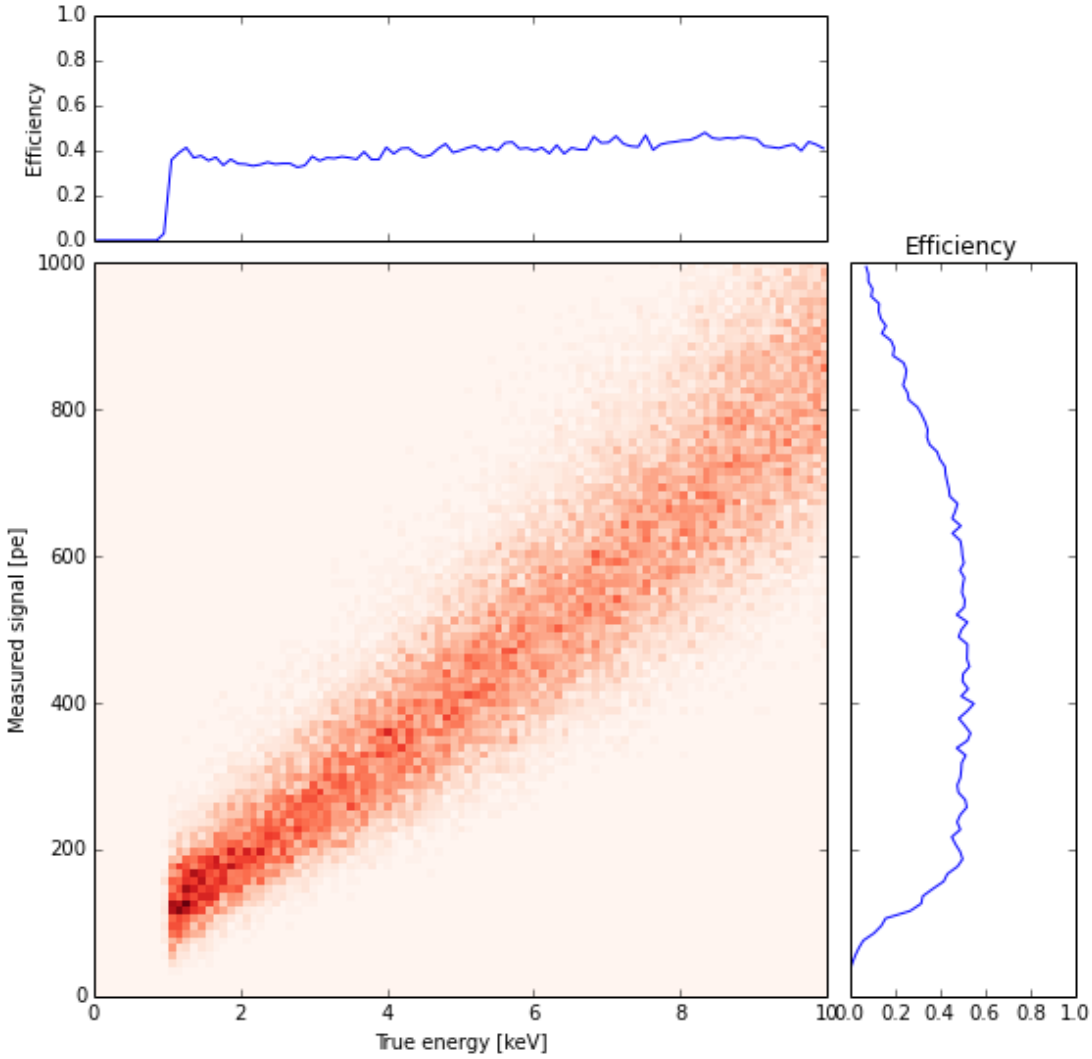
In [103]:
```
smeared_spectra(np.array([10]))
```

Out [103]:
```
array([], shape=(2, 0), dtype=float64)
```

In [104]:
```
for i in range(1):
    add_points(1000)
```

In [143]:
```
def plot(x, y, Z):
    reds = cm.Reds
    reds.set_bad('white')
    reds.set_under('white')

    nullfmt   = NullFormatter()         # no labels

    # definitions for the axes
    left, width = 0.1, 0.65
    bottom, height = 0.1, 0.65
    bottom_h = left_h = left+width+0.02

    rect_scatter = [left, bottom, width, height]
    rect_histx = [left, bottom_h, width, 0.2]
    rect_histy = [left_h, bottom, 0.2, height]

    # start with a rectangular Figure
    plt.figure(1, figsize=(8,8))

    axScatter = plt.axes(rect_scatter)
    axHistx = plt.axes(rect_histx)
    axHisty = plt.axes(rect_histy)

    # no labels
    axHistx.xaxis.set_major_formatter(nullfmt)
    axHisty.yaxis.set_major_formatter(nullfmt)

    axScatter.imshow(response_matrix, interpolation='nearest', origin='low',
                extent=[true_energy_bins[0], true_energy_bins[-1],
                        measured_energy_bins[0], measured_energy_bins[-1]],
                 aspect='auto',
                 cmap=reds)

    axScatter.set_xlabel('True energy [keV]')
    axScatter.set_ylabel('Measured signal [pe]')

    # now determine nice limits by hand:
    binwidth = 0.25
    xymax = np.max( [np.max(np.fabs(x)), np.max(np.fabs(y))] )
    lim = ( int(xymax/binwidth) + 1) * binwidth

    axScatter.set_xlim((x[0], x[-1]))
```

```
    axScatter.set_ylim((y[0], y[-1]))

    axHistx.plot(c(x), Z.sum(0))
    axHisty.plot(Z.sum(1), c(y))



    axHistx.set_xlim(axScatter.get_xlim())
    axHistx.set_ylim((0, 1))

    axHisty.set_xlim((0,1))
    axHisty.set_ylim(axScatter.get_ylim())


    axHisty.set_title("Efficiency")
    axHistx.set_ylabel("Efficiency")
    for extension in ['png', 'eps', 'pdf']:
        plt.savefig('plots/%s.%s' % ('response_matrix', extension))

    plt.show()

plot(true_energy_bins,
     measured_energy_bins,
     response_matrix)
```



In [136]:
```
def save(x, y, z):
    f = open( "response_matrix.p", "wb" )
    pickle.dump( x, f)
    pickle.dump( y, f)
    pickle.dump( z, f)
    f.close()

save(true_energy_bins,
     measured_energy_bins,
     response_matrix)
```

In []: