MovingMean

October 21, 2015

Attempting to find the best fit to a moving mean gaussian for electron cloud studies

In [31]:
```
%matplotlib inline
from matplotlib import pyplot
import numpy
import scipy.special
import scipy.stats
```

First attempt: Cumulative mean, adding $\frac{\mu}{n}$, starting in 1d
Let's also define some beam-beam interactions for later

In [32]:
```
def fortran_sign(a,b):
    return numpy.sign(a) * abs(b)

def w(z):
    '''
    complex error function from
    http://arxiv.org/pdf/1411.1024.pdf
    '''
    return numpy.exp(-pow(z,2)) * (1 - scipy.special.erf(-1.0j * z))

def bbi_ks(x_norm, y_norm, r):
    '''
    Returns kx, ky
    '''
    emax = 30

    if (r >= 0.998 and r <= 1.002): # round beam approximation
        amp = (pow(x_norm,2)+pow(y_norm,2))/2.
        scale = 4*numpy.pi
        if (amp>emax):
            kx = -x_norm*scale/amp
            ky = -y_norm*scale/amp
        elif (amp<1e-4):
            kx = -x_norm*scale
            ky = -y_norm*scale
        else:
            kx = -(x_norm*scale/amp)*(1-numpy.exp(-amp))
            ky = -(y_norm*scale/amp)*(1-numpy.exp(-amp))
        return kx,ky

    if (r>1.0):
        xx = y_norm
        yy = x_norm
```

```
            rr = 1/r
        else:
            xx = x_norm
            yy = y_norm
            rr = r
        denom = 1./numpy.sqrt(2. * (1 - pow(rr,2)))
        scale = 4. * numpy.sqrt(pow(numpy.pi,3)) * denom * (1+rr)
        u = abs(xx) * denom
        v = abs(yy) * denom

        w1 = w(u + (1j*(rr*v)))
        wr1 = numpy.real(w1)
        wi1 = numpy.imag(w1)
        w2 = w((rr*u) + (1j*v))
        wr2 = numpy.real(w2)
        wi2 = numpy.imag(w2)

        arg = (1 - pow(rr,2)) * (pow(u,2) + pow(v,2))

        if (arg > emax):
            fr = wr1
            fi = wi1
        else:
            expon = numpy.exp(-arg)
            fr = wr1 - expon*wr2
            fi = wi1 - expon*wi2

        if (r>1.0):
            kx = -fortran_sign(fr,yy)*scale
            ky = -fortran_sign(fi,xx)*scale
        else:
            kx = -fortran_sign(fi,xx)*scale
            ky = -fortran_sign(fr,yy)*scale
        #print r, (kx, ky), (fr, fi), (xx, yy), scale
        return kx,ky

    def bbi_kick_mult(kx,ky,n,sig_x,sig_y,r_e=2.8e-15,gamma=1.):
        bbi_const = n*r_e / (2*numpy.pi*gamma*(sig_x + sig_y))
        kick_x = bbi_const * kx
        kick_y = bbi_const * ky
        return kick_x, kick_y

    def bbi_kick(x, y, sig_x, sig_y, n):
        r = float(sig_x)/sig_y
        x_norm = float(x)/sig_x
        y_norm = float(y)/sig_y
        kx, ky = bbi_ks(x_norm, y_norm, r)
        kick_x, kick_y = bbi_kick_mult(kx,ky,n,sig_x, sig_y)
        return kick_x, kick_y
```

In [33]:
```
class MovingCloud:
    '''
    A cloud which follows the mean
    '''
```

```
    def __init__(self,sig_x=1.,sig_y=1.,dist_decay_fun=lambda d: 1,muo=0.,no=10000,beam_weight=
        '''
        x variance: sig_x = 1.
        y variance: sig_y = 1.
        weight/distance relationship: dist_decay_fun = lambda d: 1 (no relation)
        starting mu: muo = 0.
        starting weight: no = 1000
        '''
        self.dd = dist_decay_fun
        self.m = muo
        self.n = no
        self.no = no
        self.sx = sig_x
        self.sy = sig_y
        self.bw = beam_weight

    def cloudFun(self,mu,weight):
        '''
        Recenters cloud,
        assume mu on x axis (y=0)
        weight is number of passing particles
        '''
        # Recenter cloud
        old_m = self.m
        if self.bw:
            kicks = self.get_kick(mu,0,self.no*weight)
        else:
            kicks = self.get_kick(mu,0,self.no)
        d = abs(mu - self.m)
        self.m = float(self.n*self.m + self.dd(d)*weight*mu) / (self.n + weight)
        self.n = self.n + weight
        return old_m, kicks

    def get_kick(self, x, y, n=1):
        rel_x = float(x) - self.m
        rel_y = float(y)
        kicks = bbi_kick(rel_x,rel_y,self.sx,self.sy, n)
        return kicks
```

Now that we have a cloudlike object, lets test it out on an example bunch

Bunch is a gaussian along the z axis, most particles in the middle, fewer on the edges, let's make the mean of the bunch move linearly from $-\Delta$ to $+\Delta$

Let's start with even distribution

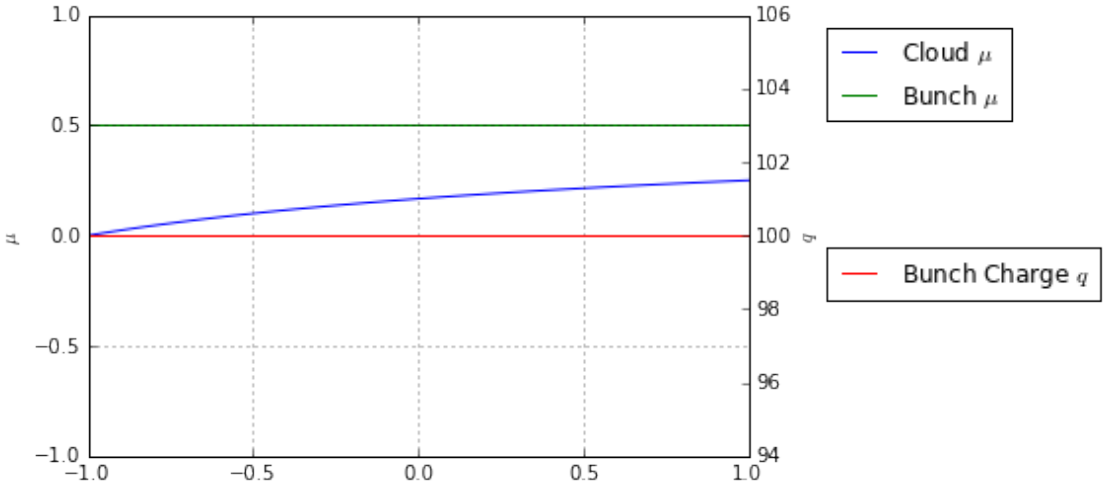
```
In [34]: def linear(startpos,endpos,num_buckets):
             '''
             A basic linear mean function from (x0,y0) to (xn,yn)
             '''
             startx, startm = startpos
             endx, endm = endpos
             dx = float(endx-startx)/num_buckets
             dmu = float(endm-startm)/(endx - startx)
             def fun(x):
                 return startm + ((x-startx)*dmu)
             return fun
```

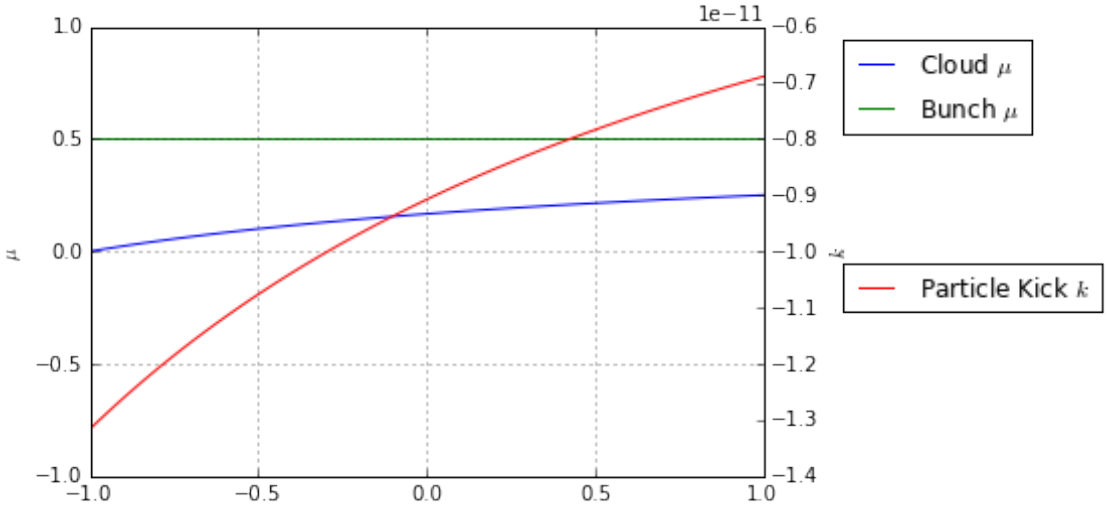
```
def makeBunch(sx,buckets,qs=1000,filllam=None,mulam=None):
    '''
    Fills up to n particles
    returns [particles] and [means]
    '''
    dx = (2*sx)/buckets
    if mulam is None:
        mulam = lambda x: 0
    if filllam is None:
        filllam = lambda x: int(qs/buckets)
    xs = numpy.linspace(-sx,sx,1+int(buckets))
    qs = numpy.array( [filllam(x) for x in xs] )
    mus = numpy.array( [mulam(x) for x in xs] )
    return qs, mus

def plotTransit(charge_buckets, mu_buckets, cloud, ylim=None):
    mu_kicks = [cloud.cloudFun(mu_buckets[i],charge_buckets[i]) for i in range(len(qs))]
    cloudmus = numpy.array([v[0] for v in mu_kicks])
    x_kicks = numpy.array([v[1][0] for v in mu_kicks])

    fig, ax1 = pyplot.subplots()
    ax1.plot(xs,cloudmus,label="Cloud $\mu$")
    ax1.plot(xs,mus, label="Bunch $\mu$")
    ax1.set_ylabel("$\mu$")
    ax1.grid()
    if ylim is not None:
        ax1.set_ylim(ylim)
    pyplot.legend(bbox_to_anchor=(1.1, 1),loc=2)
    ax2 = ax1.twinx()
    ax2.plot(xs,qs,label="Bunch Charge $q$",color="r")
    ax2.set_ylabel("$q$")
    pyplot.legend(bbox_to_anchor=(1.1, 0.5),loc=2)
    pyplot.show()

    fig, ax1 = pyplot.subplots()
    ax1.plot(xs,cloudmus,label="Cloud $\mu$")
    ax1.plot(xs,mus, label="Bunch $\mu$")
    ax1.set_ylabel("$\mu$")
    ax1.grid()
    if ylim is not None:
        ax1.set_ylim(ylim)
    pyplot.legend(bbox_to_anchor=(1.1, 1),loc=2)
    ax2 = ax1.twinx()
    ax2.plot(xs,x_kicks,label="Particle Kick $k$",color="r")
    ax2.set_ylabel("$k$")
    pyplot.legend(bbox_to_anchor=(1.1, 0.5),loc=2)
    pyplot.show()
```

Start with particle "pipe" traveling above cloud

In [35]:
```
buckets = 100
n = 10000
sx = 1
dx = 0.1
```

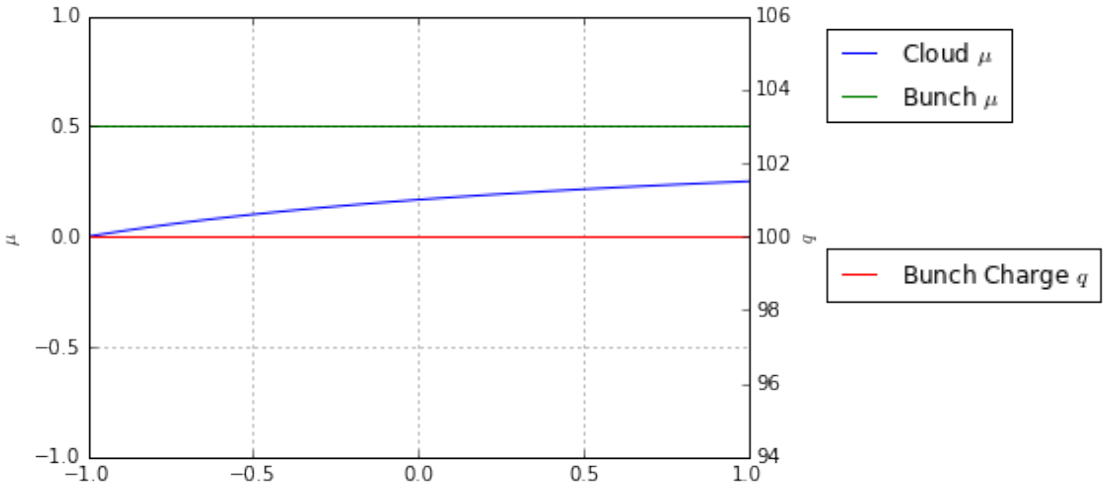
```
srtm = -1
endm = 1
xs = numpy.linspace(-sx,sx,1+buckets)
qs, mus = makeBunch(sx,buckets,qs=n,mulam=lambda x: 0.5)
mcloud = MovingCloud()

plotTransit(qs,mus,mcloud,ylim=(-1.,1.))
```

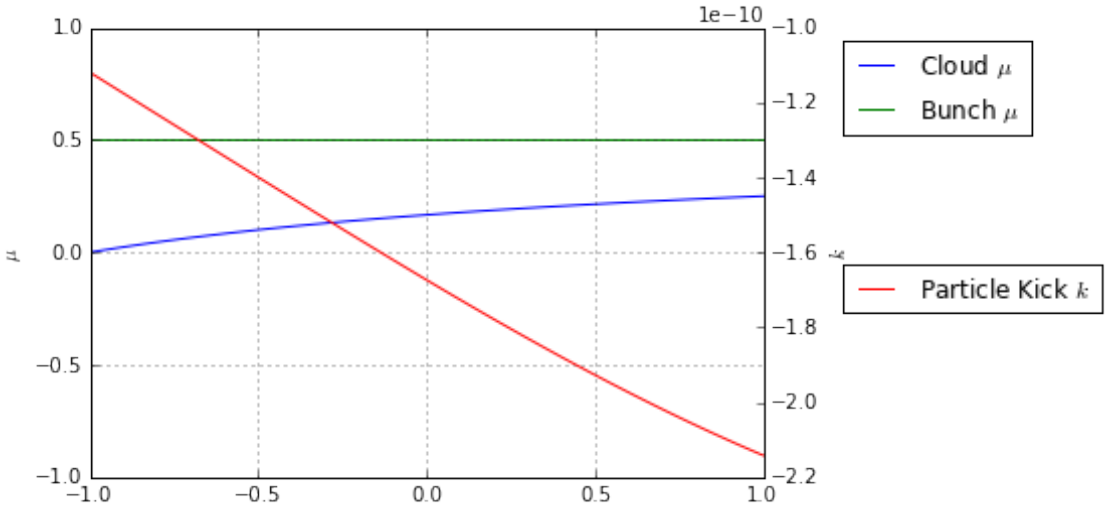




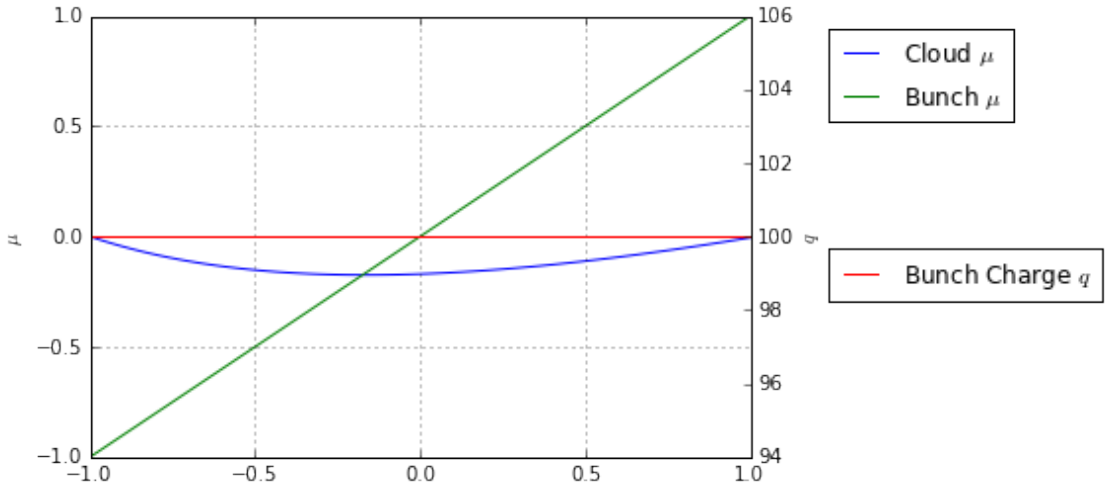
In [36]:
```
buckets = 100
n = 10000
sx = 1
dx = 0.1
srtm = -1
endm = 1
xs = numpy.linspace(-sx,sx,1+buckets)
```

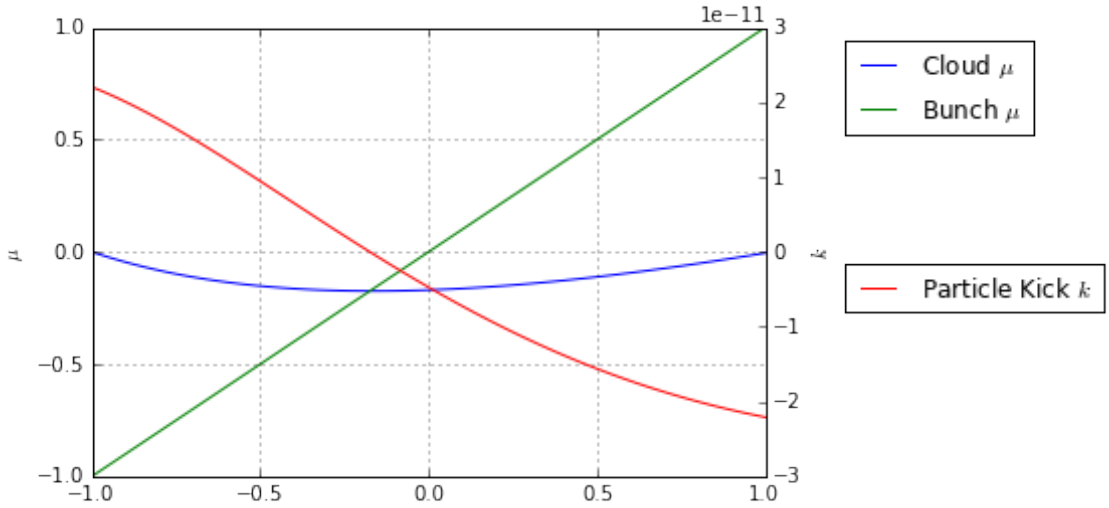
```
qs, mus = makeBunch(sx,buckets,qs=n,mulam=lambda x: 0.5)
mcloud = MovingCloud(sig_x=0.1,sig_y=0.1)

plotTransit(qs,mus,mcloud,ylim=(-1.,1.))
```





In [37]:
```
buckets = 100
n = 10000
sx = 1
dx = 0.1
srtm = -1
endm = 1
xs = numpy.linspace(-sx,sx,1+buckets)
qs, mus = makeBunch(sx,buckets,qs=n,mulam=linear((-1,-1),(1,1),buckets))
mcloud = MovingCloud()

plotTransit(qs,mus,mcloud)
```

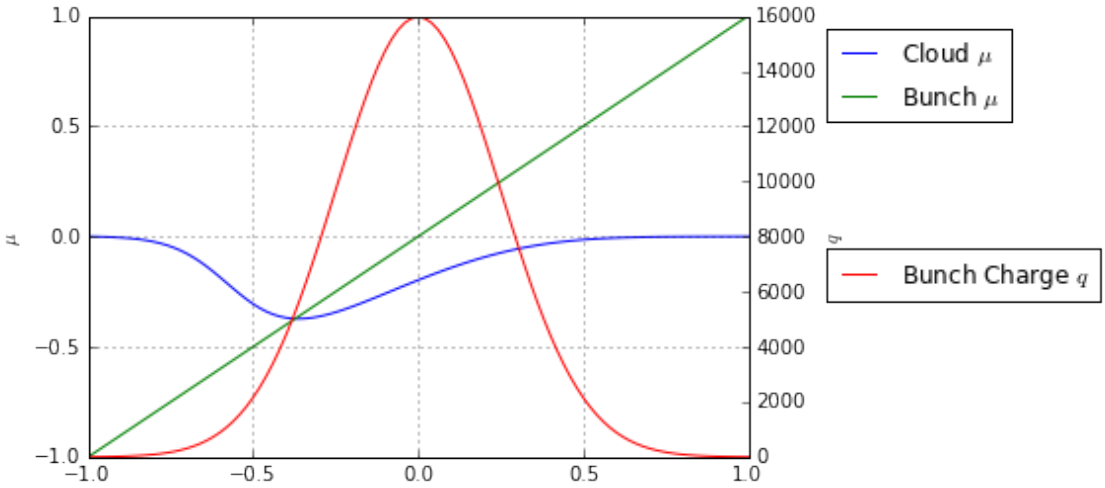


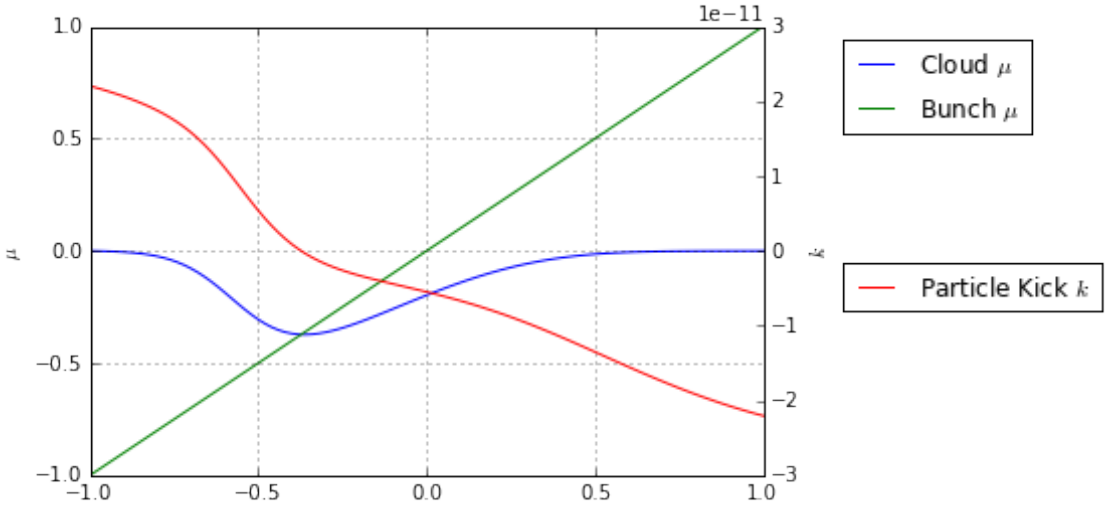


Now make the bunch gaussian with the following command

```
In [38]: def normal(x,sigma,mu=0):
             return (1./(numpy.sqrt(2*numpy.pi)*sigma))*numpy.exp(-pow(x-mu,2)/(2*pow(sigma,2)))
```

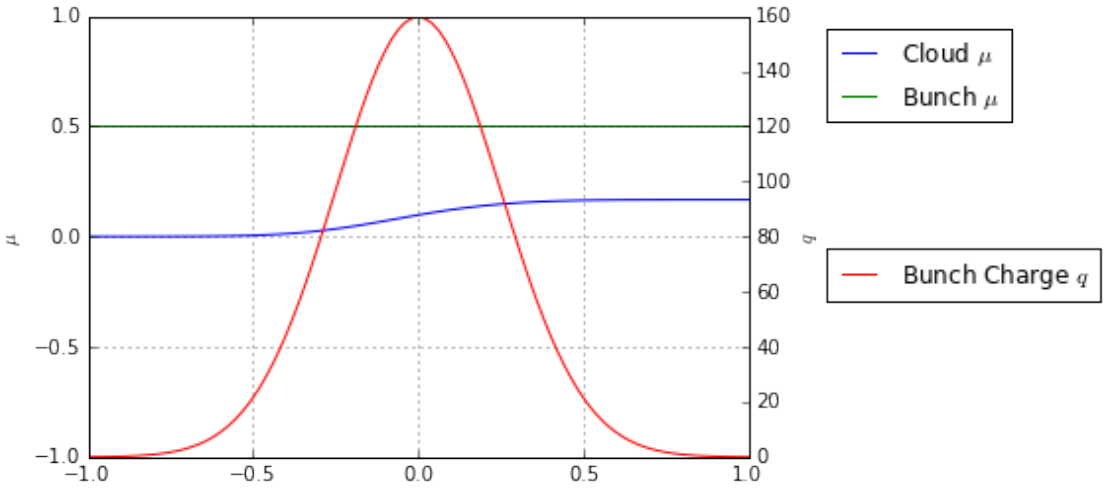
```
In [39]: buckets = 100
         n = 10000
         sx = 1
         dx = 0.1
         srtm = -1
         endm = 1
         xs = numpy.linspace(-sx,sx,1+buckets)
         qs, mus = makeBunch(sx,buckets,filllam=lambda x: n*normal(x,0.25),mulam=linear((-1,-1),(1,1),bu

         mcloud = MovingCloud()

         plotTransit(qs,mus,mcloud)
```

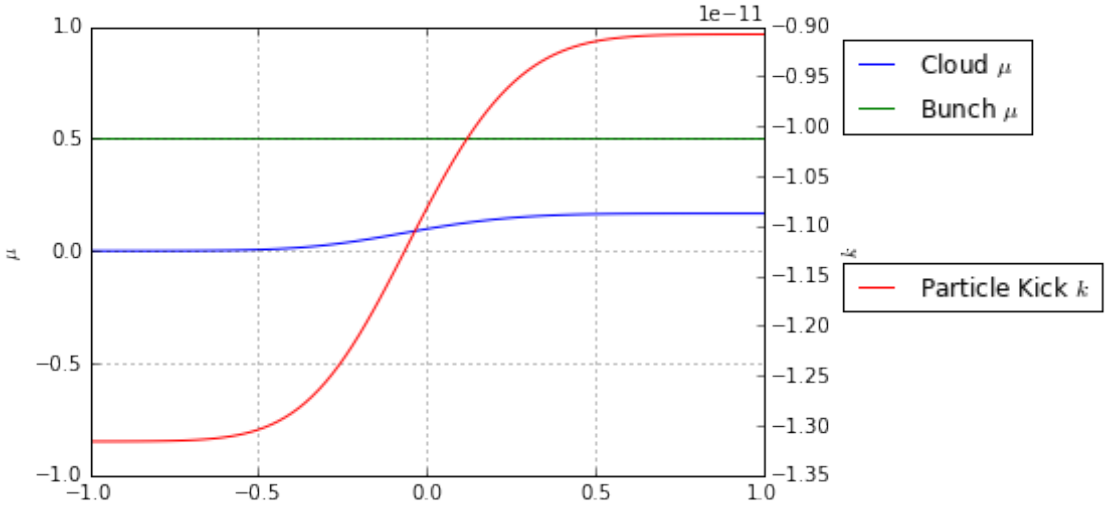




Gaussian with offset

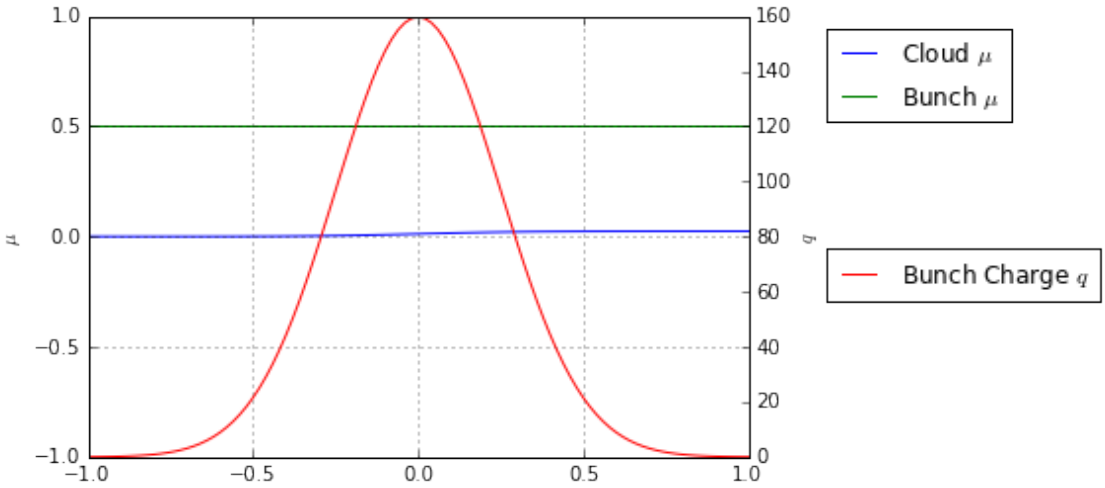
```
In [40]: buckets = 100
         n = 10000
         sx = 1
         dx = 0.1
         srtm = -1
         endm = 1
         xs = numpy.linspace(-sx,sx,1+buckets)
         fill_lam = lambda x: (float(n)/buckets)*normal(x,0.25)
         mu_lam = lambda x: 0.5
         qs, mus = makeBunch(sx,buckets,filllam=fill_lam,mulam=mu_lam)

         mcloud = MovingCloud()
         plotTransit(qs,mus,mcloud,ylim=(-1.,1.))
```

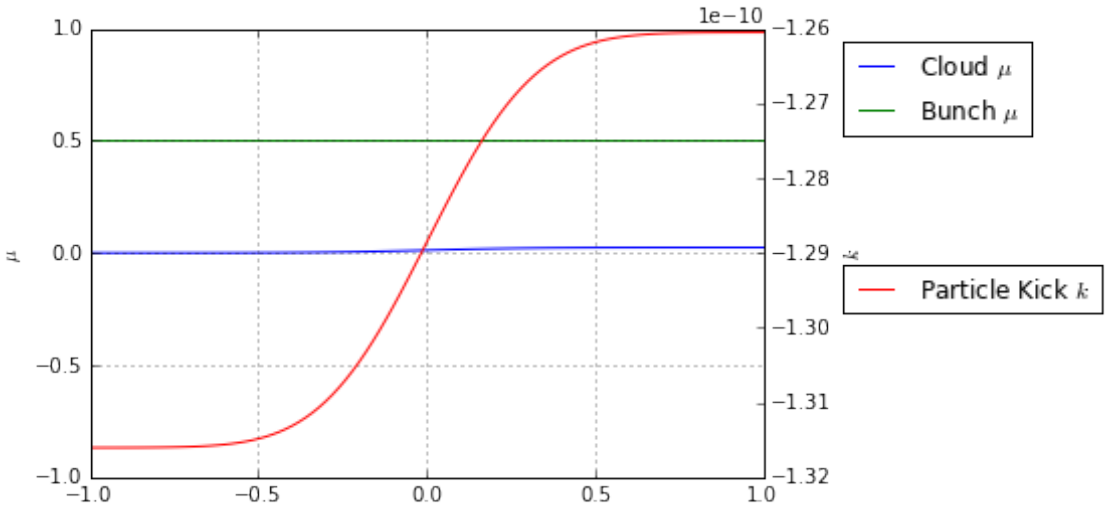




Let's try with a denser cloud

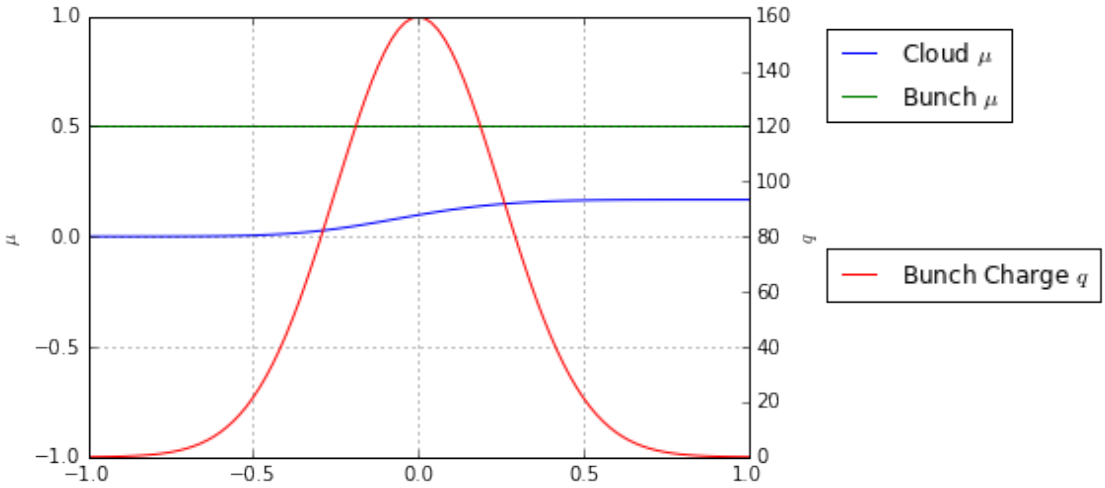
```
In [41]: buckets = 100
         n = 10000
         sx = 1
         dx = 0.1
         srtm = -1
         endm = 1
         xs = numpy.linspace(-sx,sx,1+buckets)
         fill_lam = lambda x: (float(n)/buckets)*normal(x,0.25)
         mu_lam = lambda x: 0.5
         qs, mus = makeBunch(sx,buckets,filllam=fill_lam,mulam=mu_lam)
         mcloud = MovingCloud(no=100000)
         plotTransit(qs,mus,mcloud,ylim=(-1.,1.))
```

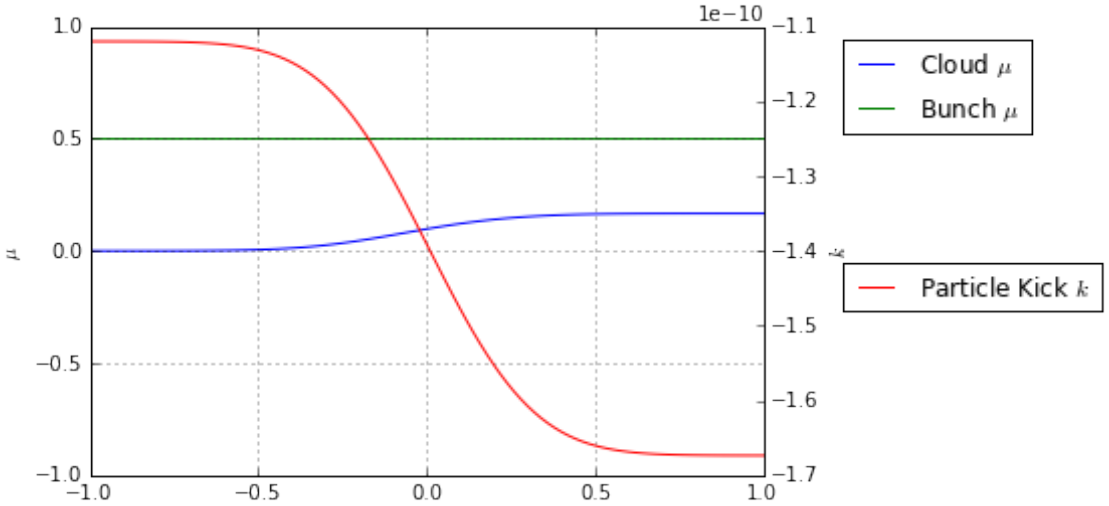




The tail is getting kicked less than the head, maybe because it's inside the gaussian cloud, let's make the cloud tighter to find out.

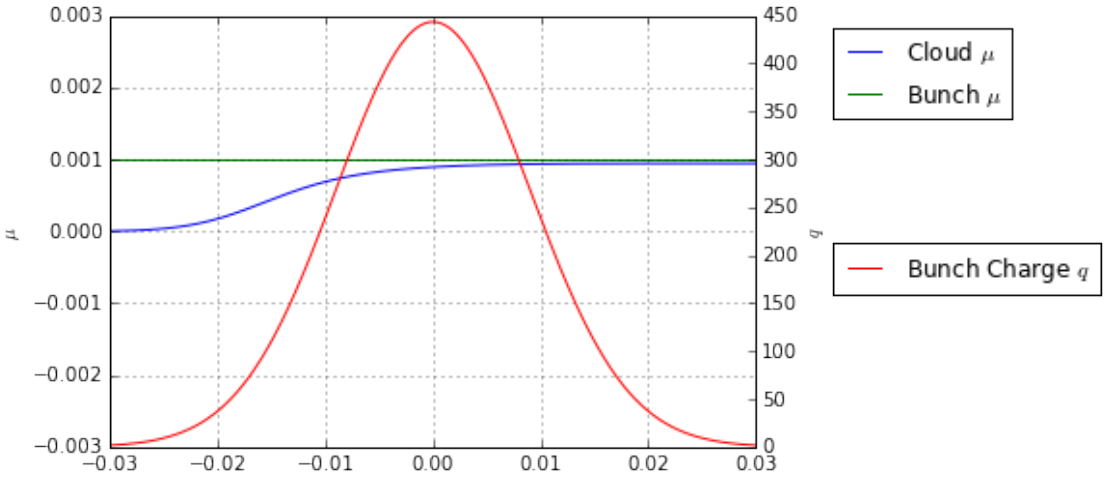
In [42]:
```
buckets = 100
n = 10000
sx = 1
dx = 0.1
srtm = -1
endm = 1
xs = numpy.linspace(-sx,sx,1+buckets)
fill_lam = lambda x: (float(n)/buckets)*normal(x,0.25)
mu_lam = lambda x: 0.5
qs, mus = makeBunch(sx,buckets,filllam=fill_lam,mulam=mu_lam)
mcloud = MovingCloud(sig_x=0.1,sig_y=0.1,no=10000)
plotTransit(qs,mus,mcloud,ylim=(-1.,1.))
```

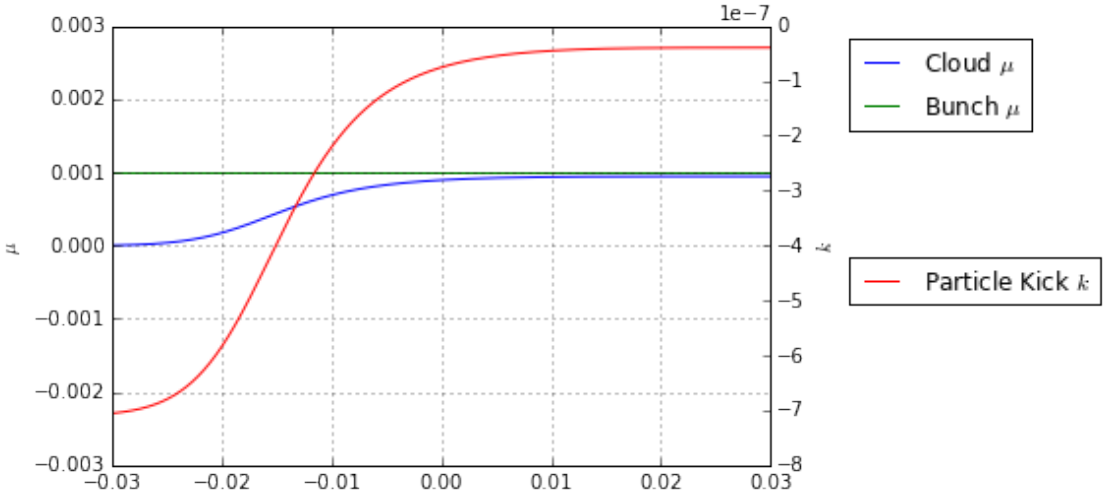




Now let's try some more reasonable length scales

```
In [43]: buckets = 1000
         n = 10000
         sx = 0.03
         dx = 0.1
         srtm = -1
         endm = 1
         xs = numpy.linspace(-sx,sx,1+buckets)
         fill_lam = lambda x: (float(n)/buckets)*normal(x,0.009)
         mu_lam = lambda x: 0.001
         qs, mus = makeBunch(sx,buckets,filllam=fill_lam,mulam=mu_lam)
         mcloud = MovingCloud(sig_x=0.0001,sig_y=0.001,no=10000)
         plotTransit(qs,mus,mcloud,ylim=(-.003,.003))
```


Cloud μ
Bunch μ
Bunch Charge q
μ
q


1e−7
Cloud μ
Bunch μ
Particle Kick k
μ
k