```
from __future__ import division
import numpy as np
from scipy.optimize import fmin
from scipy.special import erfc, expi
from scipy.integrate import quad

def uniform_sphere(size=None, dtype=np.double):
    """
    Generate random points isotropically distributed across the unit sphere.

    Args:
        - size: int, *optional*
            Number of points to generate. If no size is specified, a single
            point is returned.

    Source: Weisstein, Eric W. "Sphere Point Picking." Mathworld.
    """

    theta, u = np.random.uniform(0.0, 2*np.pi, size), \
        np.random.uniform(-1.0, 1.0, size)

    c = np.sqrt(1-u**2)

    if size is None:
        return np.array([c*np.cos(theta), c*np.sin(theta), u])

    points = np.empty((size, 3), dtype)

    points[:,0] = c*np.cos(theta)
    points[:,1] = c*np.sin(theta)
    points[:,2] = u

    return points

def intersect_sphere(pos, dir, r=1):
    d1 = -(dir*pos).sum(axis=1) + np.sqrt((dir*pos).sum(axis=1)**2 - (pos**2).sum(axis=1) +
    d2 = -(dir*pos).sum(axis=1) - np.sqrt((dir*pos).sum(axis=1)**2 - (pos**2).sum(axis=1) +
    d = np.max(np.vstack((d1,d2)), axis=0)
    return pos + d[:,np.newaxis]*dir

def norm(x):
    "Returns the norm of the vector `x`."
    return np.sqrt((x*x).sum(-1))
```

```
class ScintillationProfile(object):
    def __init__(self, r=12000.0, tau=4.0, c=299.8):
        self.r = r
        self.tau = tau
        self.c = c

    #@profile
    def pdf(self, t, l):
        r, c, tau = self.r, self.c, self.tau

        t = np.atleast_1d(t)

        tcut = (l+r)/c
        tmin = (r-l)/c

        x = t.copy()
        x[t > tcut] = tcut

        y = c*x*tau*np.exp(tmin/tau)*(r+l-c*tau)
        y += np.exp(x/tau)*(l**2-r**2+c**2*x*tau)*tau
        y += x*(l**2-r**2)*(expi(tmin/tau) - expi(x/tau))
        y /= 4*c*l*x*tau
        y *= np.exp(-t/tau)/tau
        y[t < tmin] = 0
        return y if y.size > 1 else y.item()

    def cdf2(self, t, l):
        r, c, tau = self.r, self.c, self.tau

        t = np.atleast_1d(t)

        tcut = (l+r)/c
        tmin = (r-l)/c

        if t < tmin:
            return np.zeros_like(t)
        if t < tcut:
            x = t
            y = l**2*(l-r+2*c*x)
            y -= l*(r**2 + 2*c*r*x - c**2*x**2)
            y += r*(r**2 + c**2*x*(-x + 2*tau - 2*np.exp((tmin-x)/tau)*tau))
            y /= 4*c*l*(l-r)*x
            return y

        x = t
        y = (2*l**2 - 2*l*r + c*r*tau - c*np.exp(-2*l/(c*tau))*r*tau)/(2*l**2 - 2*l*r)
```

```
            y2 = np.exp((tmin-tcut-x)/tau)*(np.exp(tcut/tau) - np.exp(x/tau))*r
            y2 += tau*(tcut + tmin*np.exp((tcut-tmin)/tau))
            y2 /= 2*l*c*tmin*tcut
            return y + y2

        def cdf(self, t, l):
            def _cdf(t):
                x = np.linspace((self.r-l)/self.c,t,100)
                return np.trapz(self.pdf(x,l),x)
                #y, err = quad(self.pdf,(self.r-l)/self.c,t,args=(l,),epsabs=1e-2)
                return y

            if np.iterable(t):
                return np.array([_cdf(x) for x in t])
            else:
                return _cdf(t)

        def gen_times(self, l=0, n=100):
            pos = np.atleast_2d((l,0,0))
            dir = uniform_sphere(n)
            hit = intersect_sphere(pos,dir,r=self.r)
            d = norm(hit-pos)
            return np.random.exponential(self.tau,n) + d/self.c

        #@profile
        def fit(self, t, n=10, retry=True):
            def nll(pars):
                l, = pars
                return -np.log(self.pdf(t,l=l)).sum()

            l0 = np.linspace(0,self.r,n)
            x0 = [-np.log(self.pdf(t,x)).sum() for x in l0]
            x0 = l0[np.nanargmin(x0)]
            xopt, fopt, iter, funcalls, warnflag = fmin(nll,[x0],disp=False,full_output=True)
            if warnflag:
                if n < 1e3 and retry:
                    print 'retrying with n=%i' % (n*10)
                    return self.fit(t,n*10)
                print 'warning'
            return xopt

        def ks_test(self, t, l):
            ti = np.linspace(0,(self.r+l)/self.c + 2*self.tau, 100)
            return np.max(np.abs((t < ti[:,np.newaxis]).sum(1)/len(t) - self.cdf(ti,l)))

    def get_times(pos, n, r, tau=1, c=1):
```

```
    pos = np.atleast_2d(pos)
    dir = uniform_sphere(n)
    hit = intersect_sphere(pos,dir,r=r)
    d = norm(hit-pos)
    return np.random.exponential(tau,n) + d/c

def foo(pars, ti, pmt_pos, tau, sigma, norm_pmt_pos):
    x = pars
    si = pmt_pos-x
    d = norm(si)
    cos_phi = (si*norm_pmt_pos).sum(axis=1)
    t = ti - d
    #if (t < 0).any():
    #    print 't < 0'
    #else:
    #    print 't > 0'
    MU = 0.0
    nll = t.sum()/tau - np.log(erfc((MU + sigma**2/tau - t)/(np.sqrt(2)*sigma))).sum()
    # solid angle
    nll += -np.log(cos_phi).sum() + 2*np.log(d).sum()
    return nll

def foo_gauss(pars, ti, pmt_pos, tau, sigma, norm_pmt_pos):
    x = pars
    si = pmt_pos-x
    d = norm(si)
    cos_phi = (si*norm_pmt_pos).sum(axis=1)
    t = ti - d
    nll = (t**2).sum()/(2*tau**2)
    # solid angle prop. to cos_phi/d**2
    nll += -np.log(cos_phi).sum() + 2*np.log(d).sum()
    return nll

def fit(sigma, tau, n, J=100, gauss=False, event_position=(0,0,0)):
    pos = np.atleast_2d(event_position)

    results = np.empty((J,3), dtype=float)

    for i in range(J):
        dir = uniform_sphere(n)
        pmt_pos = intersect_sphere(pos,dir,r=100)
        norm_pmt_pos = pmt_pos/norm(pmt_pos)[:,np.newaxis]
        d = norm(pmt_pos-pos)

        while True:
            # scintillation emission time
```

```
            if gauss:
                ti = np.random.normal(0,tau,n)
            else:
                ti = np.random.exponential(tau,n)
            # time to cross the detector
            ti += d
            # PMT transit time spread
            ti += np.random.normal(0,sigma,n)

            # initial guess
            x0 = np.random.random(3)/100

            if gauss:
                xopt, fopt, iter, funcalls, warnflag = \
                    fmin(foo_gauss,x0, args=(ti,pmt_pos,tau,sigma,norm_pmt_pos), disp=False,
            else:
                xopt, fopt, iter, funcalls, warnflag = \
                    fmin(foo,x0, args=(ti,pmt_pos,tau,sigma,norm_pmt_pos), disp=False, full_

            if fopt < 1e9 and warnflag == 0:
                break;
            else:
                print warnflag
                print 'fitter failed'

        results[i] = xopt
    #print

    return results#np.mean(np.std(results,axis=0))

#@profile
def main(N, n_max=1200):
    import matplotlib.pyplot as plt

    s = ScintillationProfile()

    ks_real = []
    for i in range(N):
        print i+1
        l = np.random.random()*s.r
        n = np.random.randint(10,n_max)
        t = get_times((l,0,0),n,s.r,s.tau,s.c)
        fit_l = s.fit(t)
        ks_real.append((n,s.ks_test(t,fit_l)))

    ks_fake = []
```

```
for i in range(N):
    print i+1
    n = np.random.randint(10,n_max)
    t = np.random.random(n)*100
    fit_l = s.fit(t, retry=False)
    ks_fake.append((n,s.ks_test(t,fit_l)))

ks_bomb = []
for i in range(N):
    print i+1
    n = np.random.randint(10,n_max)
    l = 10000.0
    pos = np.atleast_2d((l,0,0))
    dir = uniform_sphere(n)
    hit = intersect_sphere(pos,dir,r=s.r)
    d = norm(hit-pos)
    t = d/s.c
    #fit_l = s.fit(t)
    ks_bomb.append((n,s.ks_test(t,6000.0)))

ks_gauss = []
for i in range(N):
    print i+1
    n = np.random.randint(10,n_max)
    l = np.random.random()*s.r
    pos = np.atleast_2d((l,0,0))
    dir = uniform_sphere(n)
    hit = intersect_sphere(pos,dir,r=s.r)
    d = norm(hit-pos)
    t = np.random.randn(n)*s.tau + d/s.c
    fit_l = s.fit(t)
    ks_gauss.append((n,s.ks_test(t,fit_l)))

n, real = zip(*ks_real)
n, fake = zip(*ks_fake)
n, bomb = zip(*ks_bomb)
n, gauss = zip(*ks_gauss)

plt.ion()
plt.scatter(n, real, s=2, color='b', label='Real')
plt.scatter(n, fake, s=2, color='g', label='Random')
plt.scatter(n, bomb, s=2, color='r', label='Photon Bomb near Edge')
plt.scatter(n, gauss, s=2, color='m', label='Gaussian')
plt.legend(numpoints=1)
plt.xlabel('Hits')
plt.ylabel('KS Statistic')
```

```
    plt.xlim((0,n_max))
    plt.ylim((0,1))
    import code
    code.interact(local=locals())

if __name__ == '__main__':
    #main(10000)

    s = ScintillationProfile(c=299.8/1.5)

    ra = [0,1000,6000]
    na = np.linspace(100,1200,10).astype(int)
    resolution = {}
    for l in ra:
        print l
        resolution[l] = []
        for n in na:
            print n
            temp_std = []
            for i in range(1000):
                t = s.gen_times(l=l,n=n)
                fit_l = s.fit(t)
                temp_std.append(fit_l)
            resolution[l].append(np.std(temp_std))

    import matplotlib.pyplot as plt
    plt.ion()
    for key, value in resolution.iteritems():
        plt.errorbar(na,value,value/np.sqrt(1000),fmt='-o',lw=2,label='R=%.1f' % key)
    plt.legend(numpoints=1)
    plt.xlabel('Hits')
    plt.ylabel('Resolution (mm)')
    plt.show()



    #fit_results_gauss = []
    #fit_results = []
    #for sigma in np.linspace(1e-2,1,1):
    #    for tau in np.linspace(1,10,1):
    #        for n in np.linspace(10,1000,100):
    #            std_gauss = np.mean(np.std(fit(sigma,tau,n,gauss=True),axis=0))
    #            fit_results_gauss.append((sigma,tau,n,std_gauss))
    #            std = np.mean(np.std(fit(sigma,tau,n,gauss=False),axis=0))
    #            fit_results.append((sigma,tau,n,std))
    #            print len(fit_results)
```

```
##for n in np.logspace(1,3,100):
##    std = np.mean(np.std(fit(0.01,2,n),axis=0))
##    fit_results.append(std)
##    print len(fit_results)

#np.savetxt('results10_gauss_N.txt',fit_results_gauss)
#np.savetxt('results10_fixed_N.txt',fit_results)
```