SNO+ Scintillation

Anthony LaTorre

January 24, 2014

Abstract

In this paper I find an analytic expression for the time distribution at the edge of a spherical detector for light emitted with an exponential time distribution within the detector. I show how this distribution can be used to fit for the radius of an event from only the hit times of all PMTs, and find that it is possible to reconstruct the radius with a resolution of approximately 2cm at a radius of 1m from 1200 photons. I also show how the distribution can be used as a cut to filter out non-scintillation events. I do not consider the effects of multiple time constants in the scintillation light, scattering, reflection, and PMT jitter. The PMT jitter can be added simply by convolving the distribution with a Gaussian, and the multiple time constants by taking an appropriate sum over the different time constants. The effects of scattering and reflection will be harder to take into account. Nevertheless, I believe it can still be useful as a preliminary cut.

1 Time PDF at Detector

The time distribution for light at the edge of a spherical detector coming from light emitted with an exponential time distribution within the detector will not be exponential due to the differences in time it takes photons to reach different parts of the detector. Assuming all the photons travel directly to the PMTs at a constant velocity c, and the decay constant of the scintillator is τ, the time distribution at the edge of the detector, $f(t)$, from an event occuring at $\mathbf{x}'$ is equal to

$$f(t) = \int_{\mathbf{x}' \in \text{sphere}} \frac{1}{\tau} e^{-\frac{t - \frac{|\mathbf{x}-\mathbf{x}'|}{c}}{\tau}} \frac{\cos\phi}{4\pi|\mathbf{x}-\mathbf{x}'|^2} \tag{1}$$

To evaluate this integral, note that the result will only depend on L, the radial distance of the event. If we let R equal the radius of the detector, we can then integrate over θ and φ to evaluate Equation 1 (See Figure 1).

$$f(t) = \frac{1}{\tau} e^{-\frac{t}{\tau}} \int_{\varphi} \int_{x=\cos\theta} e^{\frac{\sqrt{L^2 - 2LRx + R^2}}{c\tau}} \frac{R^2(R - Lx)}{4\pi \left(L^2 - 2LRx + R^2\right)^{3/2}} \tag{2}$$

The $\cos\theta$ integral requires care, because we can only integrate over those parts of the sphere where it is possible for light to have made it, i.e. we can only integrate over θ which satisfy

$$(ct)^2 \geq L^2 - 2LR\cos\theta + R^2$$

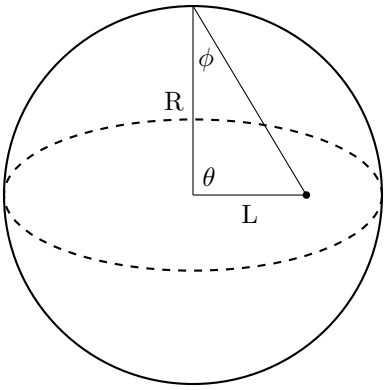

Figure 1: Sphere Integral

The result is a piecewise function

$$f(t) = \frac{1}{\tau} e^{-\frac{t}{\tau}} \begin{cases} 0 & t < t_m \\ \frac{t_c\left(t\tau e^{\frac{t_m}{\tau}} - t_m\left(t\mathrm{Ei}\left(\frac{t_m}{\tau}\right) - t\mathrm{Ei}\left(\frac{t}{\tau}\right) + \tau e^{t/\tau}\right)\right) + t\tau^2\left(e^{t/\tau} - e^{\frac{t_m}{\tau}}\right)}{2t\tau(t_c - t_m)} & t_m < t < t_c \\ \frac{t_c\left(t_m\left(\mathrm{Ei}\left(\frac{t_c}{\tau}\right) - \mathrm{Ei}\left(\frac{t_m}{\tau}\right)\right) + \tau e^{\frac{t_m}{\tau}}\right) + \tau\left(e^{\frac{t_c}{\tau}}(\tau - t_m) - \tau e^{\frac{t_m}{\tau}}\right)}{2\tau(t_c - t_m)} & t > t_c \end{cases} \tag{3}$$

where $\mathrm{Ei}(x)$ is the exponential integral function, and t_m and t_c are defined as

$$t_m = \frac{R - L}{c}$$

$$t_c = \frac{R + L}{c}$$

and represent the time it would take a photon to travel the shortest and longest distances to the sphere respectively. Note that although the expression in Equation 3 looks complicated, it is only the terms involving t that are really important. In particular, for $t > t_c$, $f(t)$ is nothing but an exponential; all the terms right of the bracket are just normalizations. The reason for this is that after light has had a chance to reach all parts of the detector, everything arriving appears as an exponential with time constant τ due to the memoryless-ness of the exponential distribution.

We can simplify Equation 3 considerably when the argument of the exponential integral is large. In this case we can expand it asymptotically around ∞. To second order in this expansion,

$$\mathrm{Ei}(x) = e^x\left(\frac{1}{x^2} + \frac{1}{x}\right)$$

Expanding $\mathrm{Ei}(x)$ to second order in $f(t)$, we get

$$f(t) = \frac{1}{\tau} e^{-\frac{t}{\tau}} \begin{cases} 0 & t < t_m \\ \frac{\tau t_m e^{t/\tau}\left(t_c t_m + t^2\right) - t^2\tau(t_c + t_m)e^{\frac{t_m}{\tau}}}{2t^2 t_m(t_c - t_m)} & t_m < t < t_c \\ \frac{\tau(t_c + t_m)\left(t_m e^{\frac{t_c}{\tau}} - t_c e^{\frac{t_m}{\tau}}\right)}{2t_c t_m(t_c - t_m)} & t > t_c \end{cases} \tag{4}$$

To test the form of Equation 3, I wrote a simple Monte Carlo. I used values close to those for SNO+ for the scintillation time and detector radius. A plot of the hit time distribution for an event at a radius of 10m is shown in Figure 2 along with Equation 3.

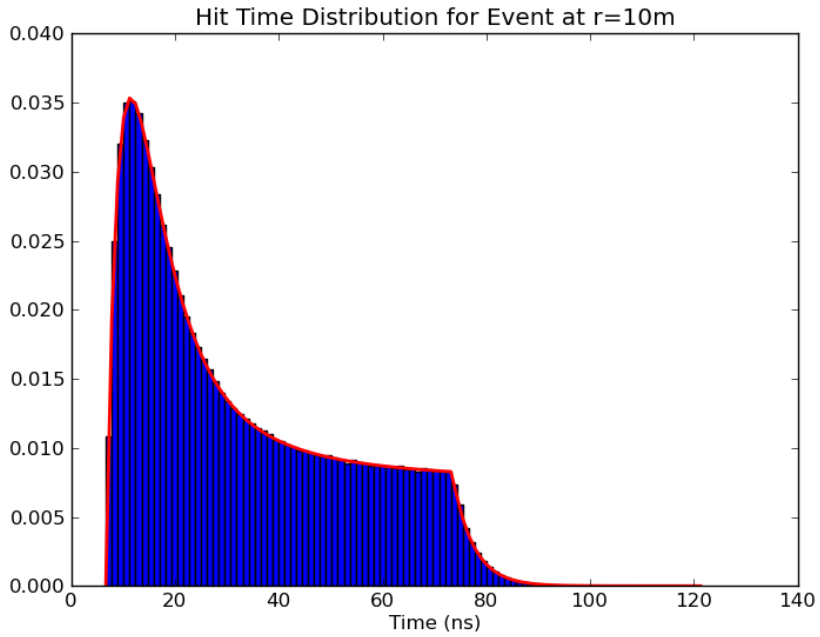

Figure 2: Time Distribution for Event at R=10m

2 Fitting

We can use Equation 3 to fit for the radius of an event from *nothing but the hit time distribution.* We construct a likelihood function from Equation 3

$$\mathscr{L}(L) = \prod_i f(t_i)$$

A sample implementation is shown in Section 4. The resolution as a function of the number of hits is shown in Figure 3 for events at a radius of 0, 1, and 6m. The first three lines use the above equation, while the three lines labeled "Full" use a full likelihood fit in three dimensions. It is interesting to note that the full likelihood resolution does not depend on the radial distance of the event. The reason for this is that the hit distribution flattens out as the radius of the event increases, and so the resolution decreases if you are only looking at the hit time distribution. When fitting in three dimensions, the likelihood effectively "sees" the underlying exponential distribution, and therefore the resolution increases.

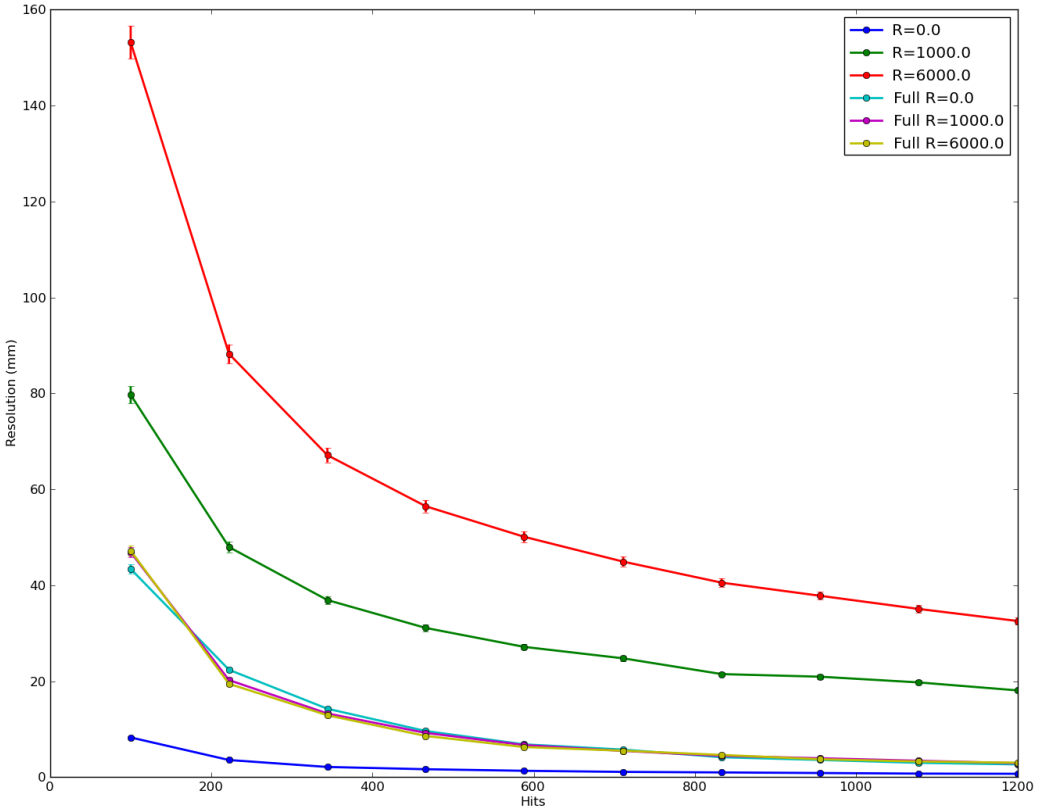

Figure 3: Resolution as a function of Hits

3 Filter

Given a set of hit times, we can calculate a KS test statistic for a hypothesized (or fit) value of L, the radial distance of the event. A scatter plot of the KS test statistic for 1,000 events as a function of the number of hits for light coming from several different sources is shown in Figure 4. The real events were generated with a true radial position drawn randomly between 0 and 6m, the radial position was then fit, and the KS test statistic evaluated at the fit radial distance. The photon bomb events were generated near the edge of the detector at 10m, and the KS statistic was computed at a radial distance of 6m, at the edge of the acrylic vessel. The idea was to mimick a flashing PMT that had accidentally fit just within the acrylic vessel. The random times were distributed uniformly over a 100ns interval, fit for a best radius, and the KS statistic was evaluated there. The gaussian data was generated just like a real scintillation event, but with normally distributed initial times.

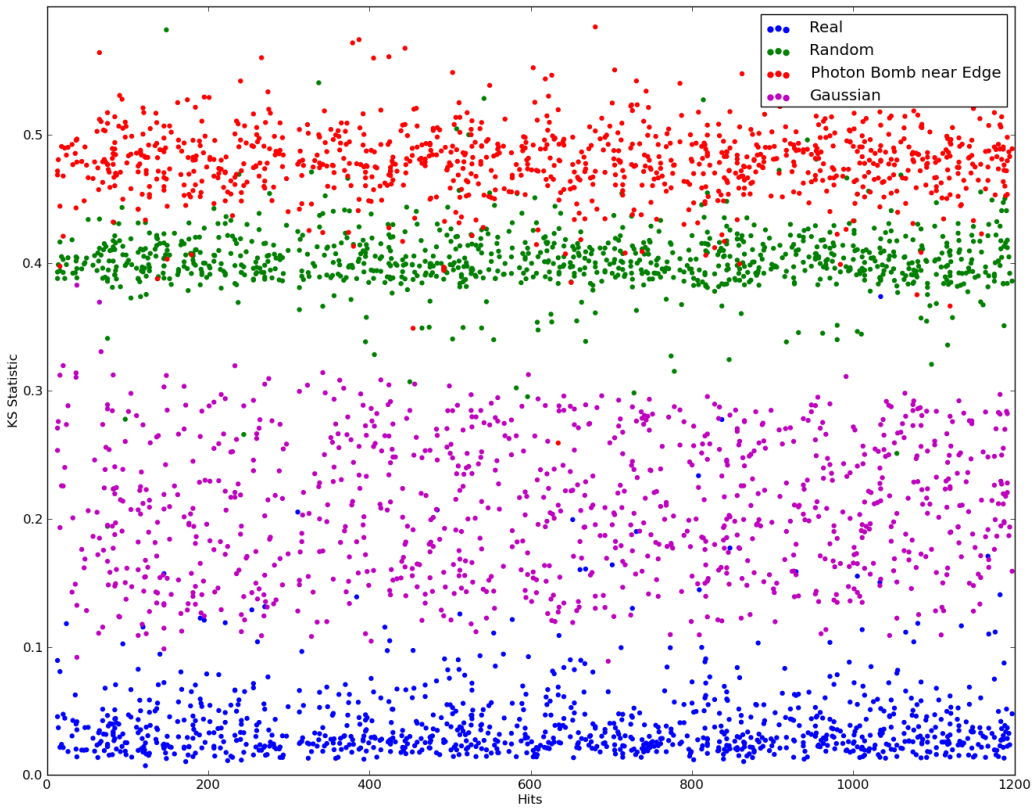

Figure 4: KS Test

4 Sample Implementation

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

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):
    """
    Returns the intersection between rays starting at `pos` and traveling
    in the direction `dir` and a sphere of radius `r`.
    """
    d1 = -(dir*pos).sum(axis=1) + np.sqrt((dir*pos).sum(axis=1)**2 - \
        (pos**2).sum(axis=1) + r**2)
    d2 = -(dir*pos).sum(axis=1) - np.sqrt((dir*pos).sum(axis=1)**2 - \
        (pos**2).sum(axis=1) + r**2)
    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

    def pdf(self, t, l):
        """
        Returns the pdf for a hit at time `t` from an event
        at radius `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 cdf(self, t, l):
        """
        Returns the cdf for a hit at time `t` from an event
        at radius `l`.
        """
        def _cdf(t):
            x = np.linspace((self.r-l)/self.c,t,100)
            return np.trapz(self.pdf(x,l),x)

        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):
        """
        Generates a set of `n` hit times for an event at radius
        `l`.
        """
        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

    def fit(self, t, n=10, retry=True):
        """
        Fit for the radius of an event from a given set of
        hit times `t`. Seed the initial position by searching
        for the most likely radius at `n` radii. If the fit
        fails and `retry` is True, try the fit again by seeding
        the fit with the best point from `n`*10 trial radii.
        """
        def nll(pars):
            l, = pars
            return -np.log(self.pdf(t,l=l)).sum()

        # seed the fit with the best radius in l0
        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 'fit failed.'
        return xopt

    def ks_test(self, t, l):
        """
        Returns the KS test statistic for the hit times `t` at
        radius `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)))
```