C:\Program Files\MATLAB\R2012a\bin\UpdatingPMTxPMTMethod\Interpolation Investigation

I ***think***these are the conclusions:

All of the “with bound” interpolations (except nearest) are shit, because I was forcing S1 values to become an unreasonable number outside of the bounds

I think UniformDistribution\_NearestWBounds.fig is showing that, in simulated data, weighting by the means and doing the KrypCal method are the same. However, if the variance does not equal the mean (ie, things aren’t poisson noise and have some Gaussian noise in it) then weighting by the variance instead of by the mean is an improvement. **FIGURE OUT WHICH CODE MADE THIS PLOT!**

RealData\_KrypVWeights\_NoInterp is showing that without any interpolation bugs present in the data, the KrypCal method and the weighting by means method are identical (no surprise), and weighting by variance is also similar (suggesting most of the PMT noise is in fact poissonian, so this method isn’t particularly useful for LUX). **AGAIN, FIND THE CODE THAT MADE THIS – WAS PMTxPMT done without interpolation? If so it’s surprising it does as well as KrypCal with interpolation.**

UniformDistribution\_NoInterpolation\_Gauss05.fig is showing adding Gaussian noise to the simulation does make PMTxPMT method better, while UniformDistribution\_NoInterpolation.fig is showing PMTxPMT is the same as KrypCal with poisson noise only.

C:\Users\Richard\Desktop\Work\PMTxPMT\4-17-Meeting:

Suggests that PMTxPMT was working better in real data than KrypCal was, and an email with Attila and Alistair are along the same lines. However, I think I was doing something “wrong” with how zeros and negative signals were dealt with in the analysis that produced this result.

The “Normalized” PMTxPMT methods normalized the individual PMT S1 signals to the summed S1 at the center.. I think this was because I felt the variance was not a fair measure of how much weight should be used, since the larger means had large sigmas as well? It seems like later SimulatedS1 code removed this

There is an obvious problem in the energy estimate – it is doing a 3D spline interpolation over a map of “normalized” S1 means which have some zero values :

energy\_estimate(:,counter)=summed\_center\*normalized\_s1./interp3(s1xbins,s1ybins,s1zbins,Mean\_S1\_3D,Event\_X\_Coord,Event\_Y\_Coord,Event\_Z\_Coord,'spline');

**SimulatedS1.m** - Looks out of date. no poisson or Gaussian noise in it

**SimulatedS2\_v2.m** – looks like the version that was used most often. Removes Normalization from the code… why did I do that? For that matter why did it have it in the first place? Probably want to use this, but fix interpolation issues

**SimulatedS1\_v3.m** – Very similar to v2, but uses some linear interpolations with bounds of zero instead of using splines. This was obviously an attempt to fix issues with 3D interpolations, but not the right way to go about it at all.

Open Questions:

1. Should I or shouldn’t I use the normalization prior to calculating PMTxPMT map? Need to understand the weighted mean method I was using to answer this I think..
2. What was each variant of version 2 simulation code doing?
3. Where is the code where I ran it on actual LUX data and made RealData\_KrypVWeights\_NoInterp? **Answer: C:\Program Files\MATLAB\R2012a\bin\UpdatingPMTxPMTMethod\OldCode – Will need some time to figure out which one I actually want to try to update… there’s A LOT of versions. I should probably just write one from “scratch” based on the simulation code**

Commented out line 292 s1(s1<=0)=0 to see if I need that in the code or not

(LUXkrypCal\_S1\_PMTxPMT\_v1 code works inside the fiducial volume)

THIS SHIFTS THE MEAN S1\_CORRECTED VALUE MUCH LOWER

Removing normalization removes changes the variance weights. Higher signal leads to more variance without normalization, which leads to smaller weights. That’s bad

CURRENT WORK:

Changed all of my inpainting code to the inpaint function – see if the result still looks good. WORKS WELL

If so, look into changing to a cubic interp on the last interpolation. Some will end up being NaN, for those, set s1 equal to the sum of the raw S1 in the last loop that makes s1\_corrected

Try the mean^2/variance weights with LUX data—it gives the same result as KrypCal with simulations and makes more sense than normalization the S1 signal for each PMT ahead of time