Juno 大作业实验报告

刘亚龙　郝传晖

August 9, 2022

Contents

1 综述

1.1 摘要:Ghost Hunter

本程序是一个江门中微子实验Juno的模拟程序，用了Python对顶点模拟、光子生成、光学过程、PMT响应进行了建模与算法实现，程序模拟中液闪球大小以及PMT位置等各项数据和指标均来自Juno的真实数据。在模拟结束之后本程序可以绘制一份分页的PDF，反映本程序的模拟结果.使得我们能够用计算机模拟Juno实验中发生的物理过程。

1.2 程序算法特色

本程序基于信息论的思想，从逼近最小有用信息量的角度出发进行了算法创新，同时实现了整个程序内部没有任何联立直线曲面方程的运算。通过应用Numpy高效的矢量运算功能、Scipy的KDTree算法、以及球面差分模拟算法，在保持高精度的前提下的让本程序的整体运行时间缩短到了极致。从最开预估的常规算法5-6h左右的运行时间经过不断的算法创新和优化，目前在一台普通的轻薄商务笔记本上能够在5min之内跑完光学过程并得到较高精度的结果，8min之内可以跑出精度非常高的结果。对于CPU性能较好的游戏本采用多核并行加速上述时间甚至可以缩短到2-3min。这也是第一个在保持高精度的同时将Juno整体运行时间缩短到个位数分钟的程序。

1.3 程序结构与说明

本程序的仓库内包含了若干个文件，具体结构如图所示，下面就这些文件的组织结构和功能进行说明：

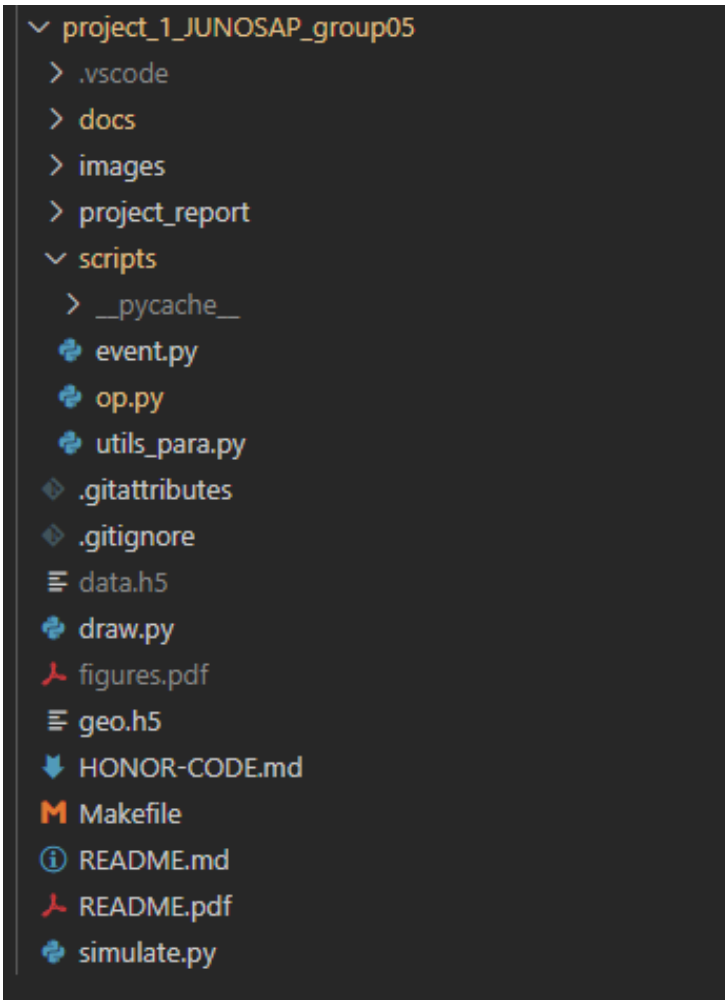

图1：项目结构

•scripts

该文件夹中收录了event.py 、op.py、utils_para.py 三个Python脚本

event.py内包含了生成球体内体密度均匀的顶点、按照非齐次泊松过程生成光子的程序，用于生成结构化数组ParticleTruth。

op.py 是一个模拟光学过程的程序，通过追踪光子位置和轨迹，读入PMT位置最终计算出来光子击中哪一个PMT，生成一个结构化数组PETruth。

uils_para.py 是一个工具包，记录了几个常用函数以及本试验使用到的各种常数。

• docs

该文件夹收录了一些除标准准要求之外做的图像和一些文档资料

• draw.py

用于绘制分页pdf的作图程序

• simulate.py

模拟的主程序，在主程序之中调用了scripts文件夹中的三个脚本程序，共同完成模拟

• MakeFile

MakeFiles数据生产流水线，执行make *** 命令就可以启动模拟程序完成数据的生成模拟，执行make clean就可以清除生成的data.h5等文件

• project_report

该文件夹是实验报告的文件夹，内部储存了该实验报告

1.4 程序使用方法

在shell环境中进入该仓库目录，运行

$$make \quad data.h5$$

可以生成data.h5数据集

$$make \quad figures.pdf$$

可以生成pdf图像

$$make \quad clean$$

可以删除上述生成的所有文件。

注意，在生成上述文件的时候需要保障系统有足够的储存空间，因为生成的data.h5大小很大,以及需要Python 的版本大于 3.9.2 ，matplotlib版本大于 3.5，在绘图的时候需要提前清空释放一部分被占用的系统内存防止爆内存，否则程序可能会出现奇怪的问题。

2 产生顶点和光子

2.1 程序设计思路

首先在本实验之中我们需要在一个液闪球体之中均匀生成4000个顶点,然后对于每一个顶点按照给定的$\lambda(t)$函数给出非齐次泊松过程的采样过程生成光子，每一个顶点产生的光子的期望值为10000

2.1.1 顶点的生成

为了达成上述的效果本程序使用了球坐标来产生体密度均匀的顶点分布，具体的实现方法是使用Numpy的随机数的功能，在r方向使用np.random.power()来生成概率密度和r^2正比，在[0,1)区间内的4000个采样分布, 将power()之中的第一个参数取为3即可产生和平方成正比的概率密度函数

而对于θ方向使用了和$sin\theta$正比的概率密度，但是Numpy之中并没有直接实现的函数功能，因此只能采用先生成[0,1)范围内的均匀分布，然后使用线性映射将其定义域调整为反三角函数的定义域，经过反三角函数的处理，就能够得到一个需要的分布

对于ϕ方向则使用均匀的概率密度，最终结果返回一个拥有4000 个元素的Numpy数组，再将这些球坐标中的顶点转化到直角坐标之中，这样就能够得到在球体内部均匀分布的4000个顶点。

为了初步检验我们生成的顶点的正确性，程序中专门设计了一个画图功能，在每次生成顶点之后会在docs文件夹之中画出一个顶点的分布图

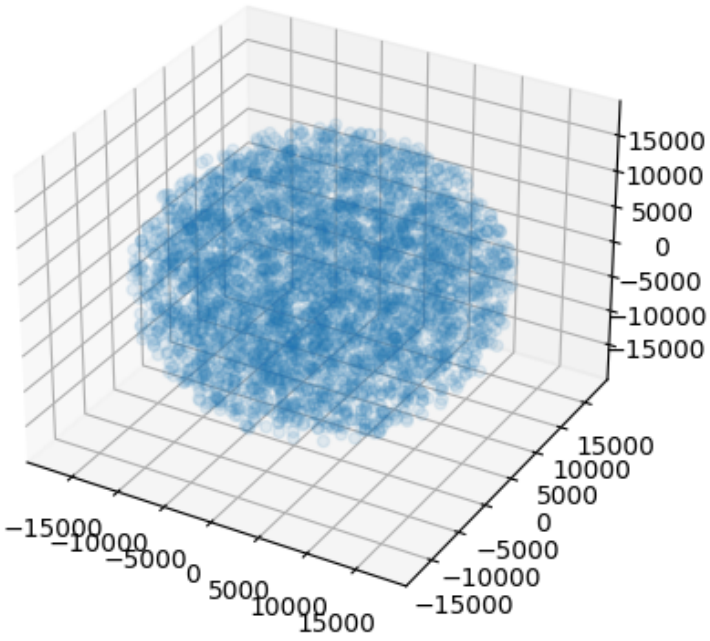

图2：顶点的分布图

2.1.2 光子的生成

在成功生成了定点之后我们考虑光子的生成，对于不同顶点而言，光子的生成过程是独立而随机的，因此我们编写一个可以对一个顶点生成需要的光子数目的函数就可以完成这一功能。

对于给定的分布，我们需要先进行预先的处理，为了保证这个函数在$[0,\infty)$上积分的值为10000，我们需要提前计算出来对应的归一化系数

```
In[3]:=
Integrate[Exp[-x/10] * (1 - Exp[-x/5]), {x, 0, Infinity}]
积分      指数形式          指数形式          无穷大

Out[3]= 20/3
```

图3：归一化系数的计算

我们可以计算出归一化系数为

$$NF = \frac{1000}{\sqrt{3}}$$

同时我们做出非齐次泊松分布的$\lambda(t)$图像如下：

In[11]:=

```
Plot[1500 * Exp[-x / 10] * (1 - Exp[-x / 5]), {x, 0, 80}, PlotRange → All]
```

绘图 指数形式 指数形式 绘制范围 全部

Out[11]=

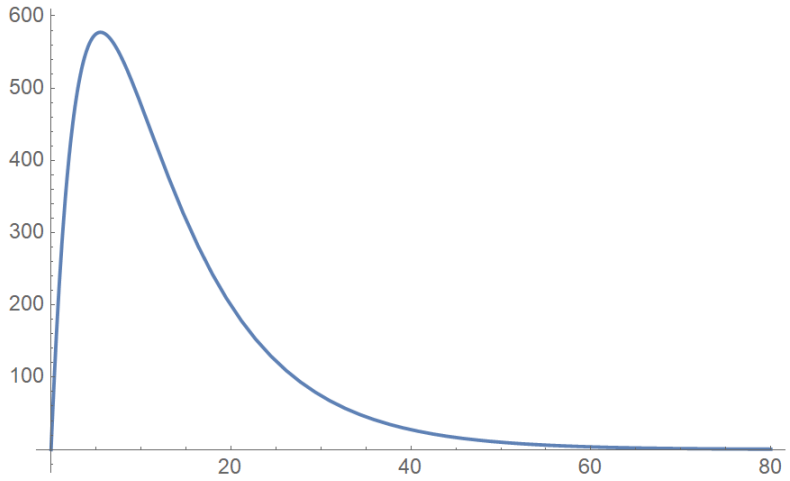

图4：$\lambda(t)$图像

下一步是进行泊松采样。对于这一过程，一开始的想法是采用线性分段的方法，选取一些等间隔的参考点，计算出来每一个小区间内部的产生光子的数目但是因为精度问题导致没有办法精确的确定出来产生光子的时间，因此经过查阅概率论相关的资料，本程序采用了蒙特卡洛方法来完成这一采样过程：

从前面的图像之中可以看出，在超过60ns的时刻之后，产生的光子相比前面的时刻可以忽略不计，因此我们只考虑模拟在60ns以内生成的光子

首先为了确定蒙特卡洛方法的撒点总数，我们令n 为以λ_{max} 为参量的泊松过程在60ns内事件发生次数，使用Numpy的random. poisson()功能生成单次取样的发生事件总数，然后我们将上述过程重复10000次，记N 为事件发生的平均次数（取整），作为蒙特卡洛的模拟撒点次数这一方法的概率论方向的理论依据参见概率论相关书籍的取舍法章节，然后我们生成两组总数为N的随机数，第一组均匀映射到[0 60ns]的区间上，记为phtorigin，另外一组记为index 则作为一个指标来判断是否保留这一个点，我们将第一组的随机采样点数组输入前面计算出来的函数，计算出每一个点对应的t时刻对应的λ的值然后将这一个Numpy数组整体除以λ_{max}，形成一组新的指标。

下一步是从原来生成的N个点中选取符合要求的点，这里我们采用取舍法，具体的方法如下：我们使用Numpy数组内置的where() 功能进行一个指标的筛选，具体代码如下图所示：

```
index = rng.random((1, N))
a = dis_f(ph_t_origin)/utils_para.lambda_max
ratio = np.where(a >index)[1]   #使用了numpy的where检索功能
t_array = ph_t_origin[:,ratio]
```

图5：where()功能代码的说明

在这里where()内的条件是

$$\frac{\lambda(t)}{\lambda_{max}} > index$$

这一表达式可以满足在t 时刻产生的所有事件按照$p = \frac{\lambda(t)}{\lambda_{max}}$的概率保留的要求，同时又不需要额外的计算量。在 Numpy的where()功能中，这一函数的返回值是一个数组[, s]，其中s即为满足条件的、我们应当保留的点在原来总数为N的蒙特卡洛数组phtorigin的序号。

考虑到Numpy的抽象索引功能，以及我们之前生成的phtorigin 是一个$1 \times N$ 的数组，我们可以直接将s作为第二个维度的索引输入，返回的值就是我们经过取舍法采样之后符合要求的光子生成时间组成的数组，经过多次的试验，该程序产生的光子的数目期望值为10000，

唯一的问题是相比于线性插值的方法该方法得到的最终光子生成序列是未经过排序的，但是考虑到最后我们会追踪每一条光线，并计算其时间，这一缺点似乎并无影响。

到此为之我们在script目录下编写了一个event.py,内部定义了生成顶点和生成光子的函数，返回的均为一个结构化数组，其具体的结构参见函数的注释。

3 光学过程

在event.py脚本之中，我们调用在前一个过程之中生成的event.py这一脚本，调用内部产生顶点和产生光子的函数，来完成PETruth的生成。具体的思路如下：

3.1 程序实现思路

首先我们通过一个循环不断的调用在scripts之中的函数，按照已经编写好的高斯采样方法进行光子的生成，这样可以生成一个总的长度约为4000万的光子序列，接下来来我们就能够调用op.py之中的函数进行下一步的模拟追踪。我们接下来依次对过程之中需要的函数进行分析

这里需要强调如果直接用numpy数组计算的的话会导致内存爆炸，因此我们采用分循环的做法，每一次循环只处理少部分顶点生成的光子，最终将结果拼接起来生成最后的petruth

接下来简述光子的模拟过程，首先我们知道了顶点位置以及生成光子的速度矢量，我们下一步要判断这一个光子打在球面上的位置，常规的解法是我们求解出来这条线的方程、以及球面的方程，将两个方程联立就可以得到线在球面上的焦点，这种方法存在这很大的问题首先是计算机求解曲面和线联立的方程时效率很低，其次是对于一条线段而言和球面的交点有2个，计算机为了求解出来两个相交点需要至少双倍的计算量来逼近，再这之后还要求解光子位移的距离，对于总的计算量而言，我们需要求解4000万次联立直线和球面方程这样庞大的计算量。因此我们引入了一个函数line来处理这个问题

3.2 line()函数

line()函数完全采用了矢量的算法求解上面的过程，不需要任何的求解方程的步骤就可以得出光子在液闪界面上的打击点，具体的实现思路如下：

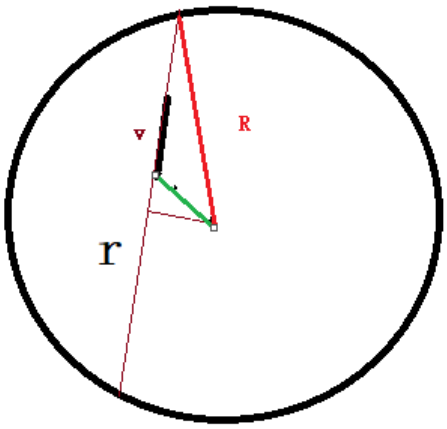

图6：line实现逻辑

如图所示，黑色矢量和绿色矢量分别是光子生成的速度方向矢量和顶点的位置矢量，我们将这两个矢量点乘，考虑到生成的速度矢量是归一的，我们相当于求解了顶点到垂足的距离，记为m(这里可正可负，正表示两个矢量夹角为锐角，反之为钝角)，其方向就是系数乘上速度矢量（已经归一化），因此我们可以用同样的方法求解出来球心到垂足的高h，这样下一步通过代数运算就能够求出来垂足到达线段和球面交点之间的距离s，使用s-m就可以得到

光子的总路程，然后我们将总路程乘以光子的单位速度矢量，加上顶点的矢量，就得到了球面交点的矢量，

$$\vec{R} = \vec{r} + \vec{v} \times (s - m)$$

将这一路程除以介质的光速就可以算出光子在球内的传播时间。这样我们不需要求解任何方程，全程使用Numpy的数组以及矢量运算功能就能够高效的快速的批量处理线段和球面相交问题。

3.3 折射、反射、全反射

在求解了交点之后我们处理反射、全反射、以及折射的问题首先对于全反射而言，只要发生一次全反射，后续光子就一直无法打出液闪球体，因此我们利用一个点乘求解cos值的函数就可以处理这个问题，我们根据条件索引找出发生全反射的光子对于cos值小于全反射临界值的光子我们直接剔除。

剔除全反射的光子之后，剩下的光子会在界面上发生反射和折射，因此我们需要使用菲涅耳定律来处理这件事，我们根据菲涅耳定律算出光子在界面上发生折射和反射的比例对于发生反射的光子，我们认为在发生一次反射之后便不会发生第二次反射，可以认为发生一次反射之后的光子全部折射出去。

对于折射的光子，我们依然采用矢量的方法来计算经过折射之后的方向矢量,将法向矢量和入射矢量按照一定的系数叠加就可以得到折射后的矢量,我们可以构建出折射定律的矢量表达形式

```
def refraction(point, in_vector, cos_in, cos_out):
    ''''''
    normal_vector = point / R_i
    out_vector = n * in_vector + (cos_out - n*cos_in) * normal_vector
    return out_vector
```

图7：折射代码

这样我们通过Numpy数组的矢量点乘和矢量加法就能够叠加出来经过折射之后的速度矢量方向

3.4 打击的PMT编号

下一步需要处理经过折射之后的光子究竟打击到了哪一个PMT上，这一过程是Juno实验之中最为关键的一点，如果采取了不恰当的方法，就会在这一过程之中消耗大量的时间。

3.4.1 引论：判断击中PMT究竟需要多少的信息？

为了精确的求解模拟的光线会打中哪一个PMT，考虑到PMT可以认为是一个半球，如果按照传统的解法，我们需要判断一条光线和17612个PMT的相对关系，这样还不够，因为每一个光线可能和多个PMT的球面都有交点，也可能都没有交点，我们为了完全确定光线打击哪一个PMT，原则上需要联立17612个球面方程和约400000000条光线，在从中选取出来相交点之中的第一个解。这样的计算量虽然可以完全精确的描述光线和PMT的相互作用，其计算量和信息量也是无比庞大的

我们紧接着有一个问题：确定一个光子打到哪一个PMT上真的需要这么多的信息量吗？回顾我们的问题，我们生成PETruth的结构化数组事实上并不需要光子打到PMT球面上的精确位置，需要的信息仅仅是光子是否打中某一个pmt以及对应编号，从信息论的角度而言，前者的信息量要比后面的信息量大若干个数量级。因此从信息和运算总量的观点出发，我们能否找到恰好能够描述打击PMT的信息量，是我们处理这个问题的关键：我们能否避开球面和直线方程的联立过程中产生的冗余庞杂的信息，来判断命中的究竟是哪一个PMT呢？

图8：一个关于信息量的示意图

3.4.2 核心算法介绍

我们在本程序中采取了如下的算法，可以在保证高精度的条件下用最小的信息量筛选出光子击中的PMT序号：

首先我们在本程序之中使用了Scipy当中提供的KDTree算法，该算法能够高效的判断某点到其他N个点的距离的最小值，以及最小值点的标号，其复杂度为$O(log(n))$，远远小于常规算法的$O(n)$

接下来最核心的问题是我们有了KDTree算法之后我们能否将光线与球面相交的问题转化为点和点之间距离的问题。为了尽可能的从算法上逼近问题所需要的最小信息量，我们采取了如下的方法:

PMT的半径为0.5m，而PMT的球心所在的球面半径为19.5m，那么我们从半径为19m到半径为19.5m之间画出若干个球面，这样的每一个球面都会和 17612个PMT的球面相交，每一个的截面上都有17612个圆，这样我们就把一个三维空间中的曲面切成了一层层的切片，我们只要判断在某一个球面上光子的打击点是否在相交构成的17612个圆内就可以用球面差分的方法模拟出来光子究竟命中哪一个PMT的过程。

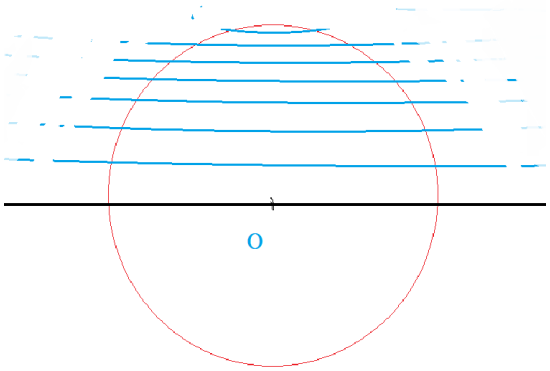

图9：切片的示意图

从19m半径的球面开始，我们使用上章节全反射、折射过程之中line()函数在液闪球面上的打击点作为起始点，折射后的速度矢量作为方向矢量输入新的line()函数，找到第一个差

分的球面上的打击点，因为给定的Geo文件使用的是球坐标来描述PMT的位置，我们可以轻松快捷的构建出来差分球面上的圆心位置，并用KDTree算法计算出打击点到这17612个圆心的最小距离以及编号，如果这一距离小于圆的半径，那么我们认为光线打击了对应编号的PMT。之后我们依次增大半径，采用更大的球面重复上述过程，直到半径到达19.5m。

原计划采用的差分数是20，这样的差分不但能够较为精确的判断究竟打击哪一个PMT，在速度上也远远优于常规的联立方程。后来在不断的测试过程中，结合PMT大小、内部液闪的半径，以及PMT的相对位置，经过计算不需要取这么多球层也能够得到精度较好的结果这部分的KDTree的球面切割数目可以根据模拟需要的精度来定义，如果要求精度非常高，就多加上几层的切片数，单次KDTree算法的所花的时间也非常少，Juno模拟而言即便多次采用KDTree算法花费的时间也远少于联立方程。经过测试即便采用50甚至更高的差分，对于一台配置较低的普通的家庭学生版笔记本计算机而言，最终的光学过程总时间也能够保证在15分钟以内完成.当然如此之高的精度对于我们的模拟过程的而言是完全没有必要的，对于这种切分方法而言，存在一定误差的仅仅是一些紧紧擦过PMT球边缘打入的光子才有的情况，在8层以上增加层数对于单个PMT上的打击数而言误差仅仅是个位上的变动，因此完全没有必要过高的追求切分的层数。

联系我们在上一部分的构建的line()算法，我们就将一个联立40000000 条线方程和17612个球面方程求解的问题，转化成了一个完全不需要联立方程，完全依靠Numpy数组矢量点乘运算以及KDTree算法判断的问题。这部分的程序即避免了所谓的联立求解方程带来的计算量，又充分利用了 Numpy数组高效处理矢量运算的特性。还使用了KDTree算法求解最近距离和最近的点。这样的算法上的创新使本程序能够用最接近于真实有用的信息量高效的判断打中PMT的编号。也是本程序能够在高精度的前提下将运行时间缩短到到极致的核心。

3.5 PETruth的生成

在经历了上一步，在判断了光子究竟打中了哪一个PMT之后，我们同时也得到了光子的总时间以及来自光子来自哪一个事件，下一步就是将分10步循环运算得到的结果拼成一个大的PETruth，并且作为一个结构化数组导出，储存到data.h5之中。这样就完成了光学过程的最后一步生成了我们需要的PEtruth。

4 绘图

本部分需要绘制分页的3页PDF,其分别为顶点密度随半径r的分布，PMT打击时间的分布直方图，以及probe的热力分布图

4.1 定点密度随着半径r的分布

在这一部分的绘图中，我们将作出一个直方折线图，来反映判断顶点是否在液闪球内均匀分布，为此我们统计出来所有的顶点分布的 r，我们选定一个区间作为直方图的bin=n，然后对这一个分布列采用np自带的histgram功能返回三个数组，分别为一个bin中的点的个数、长度为n+1的 bin为边界的位置，之后我们可以将每一个bins的体积用相邻bin边界之间的球壳做一个修正，将每一组数据都除以对应的体积，这样我们就能够将这个数据导出并做出最终的直方图，通过观察这一直方图理论上除了靠近原点处有较大的误差之外，其他都符合生成的要求，最终的密度除以顶点体密度的值在1左右波动，这也能充分佐证我们在生成顶点的时候确实满足了体密度均匀的要求

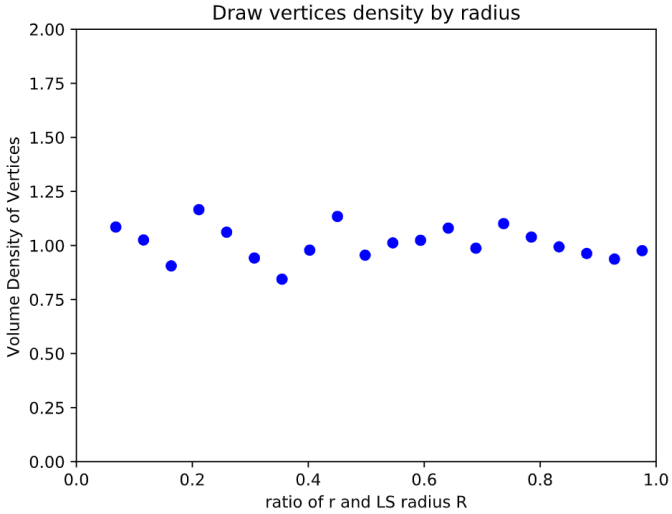

图10：打击时间图

4.2 PMT打击时间分布直方图

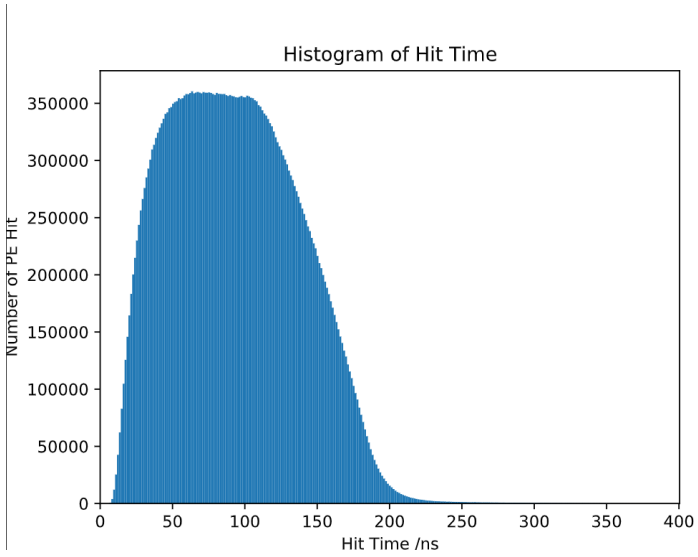

图11：打击时间图

为了绘制这一打击时间构成的分布直方图，我们将data之中的petruth的打击时间全部读取出来，然后使用直方图的绘制功能进行绘制，并添加轴标题。观察图像我们可以发现这一图像基本上和光子的生成过程的分布很相似，这也充分反映了PMT的分布几乎是各向同性的，顶点的生成也是一个的均匀的，在PMT上的波形和在顶点上产生的波形也仅仅是相差了一个由于传播导致的在时间方向的拉伸，经过我们的估计，考虑到光子在球内的传播以及经过一次反射之后的最长路径，最终的时间应该会分布在[0 250ns],图像也确实满足了相关的要求。

4.3 probe热力学分布图

绘制这一图的本质目的是要研究对于不同位置的顶点对于PMT的贡献，我们需要geo ，PMT顶点的位置，以及PETRUTH的表，具体的思路如下：首先我们对于一个pethuth而言，长度约为4000 * 10000,按照这个序列的顺序我们依次列出每一个光子打中的pmt的位置，以及光子来自顶点的位置，这样我们就得到了两个40000000 x 3数组，下一步我们调用之前编写好的函数对于这内部的矢量进行点乘的方法来计算夹角cos值，随后使用np的反三角函数来返回一个长度为40000000，属于[0 π]的序列，这就是对应的theta分布，以及我们将petruth中的顶点按照petruth上的顺序计算出来相对球心的位置 r，这两个序列就是我们需要做极坐标热力图的原始数据，但是考虑到不同位置的r的顶点体密度并不一样，因此我们还需要计算出来顶点的体密度的分布

同样的我们根据顶点位置vertice 以及pmt位置geo，可以构造一个长度为17612 * 4000的序列，在这个序列中我们保证每一个生成的顶点的位矢以及每一个PMT的位矢之间都要点乘我们按照如下的方式构建这一个序列

$$s1 = np.tile(pmt\_loc\_carti, (sim\_number, 1))$$

$$s2 = vertices.repeat(17612, axis = 0)$$

这两个序列长度均为17612*4000，但是一个是先将一号顶点位置重复17612次，然后同样的方法处理第二个顶点，另一个序列是先将17612个pmt依次列出，然后将上述过程重复4000遍，这样在按照标号点乘就可以匹配出来一个完全对称的点乘序列，然后我们对于这一个序列采取一样的处理方法，

之后我们利用二维的直方图分布，将rbin和thetabin作为设定好的值传入histgram2d，并读取其返回值并进行一个修正，$data/data_0$就是经过修正后的数据点我们在设置好标题，颜色之后就可以进行绘图，绘制出来的图像如下:

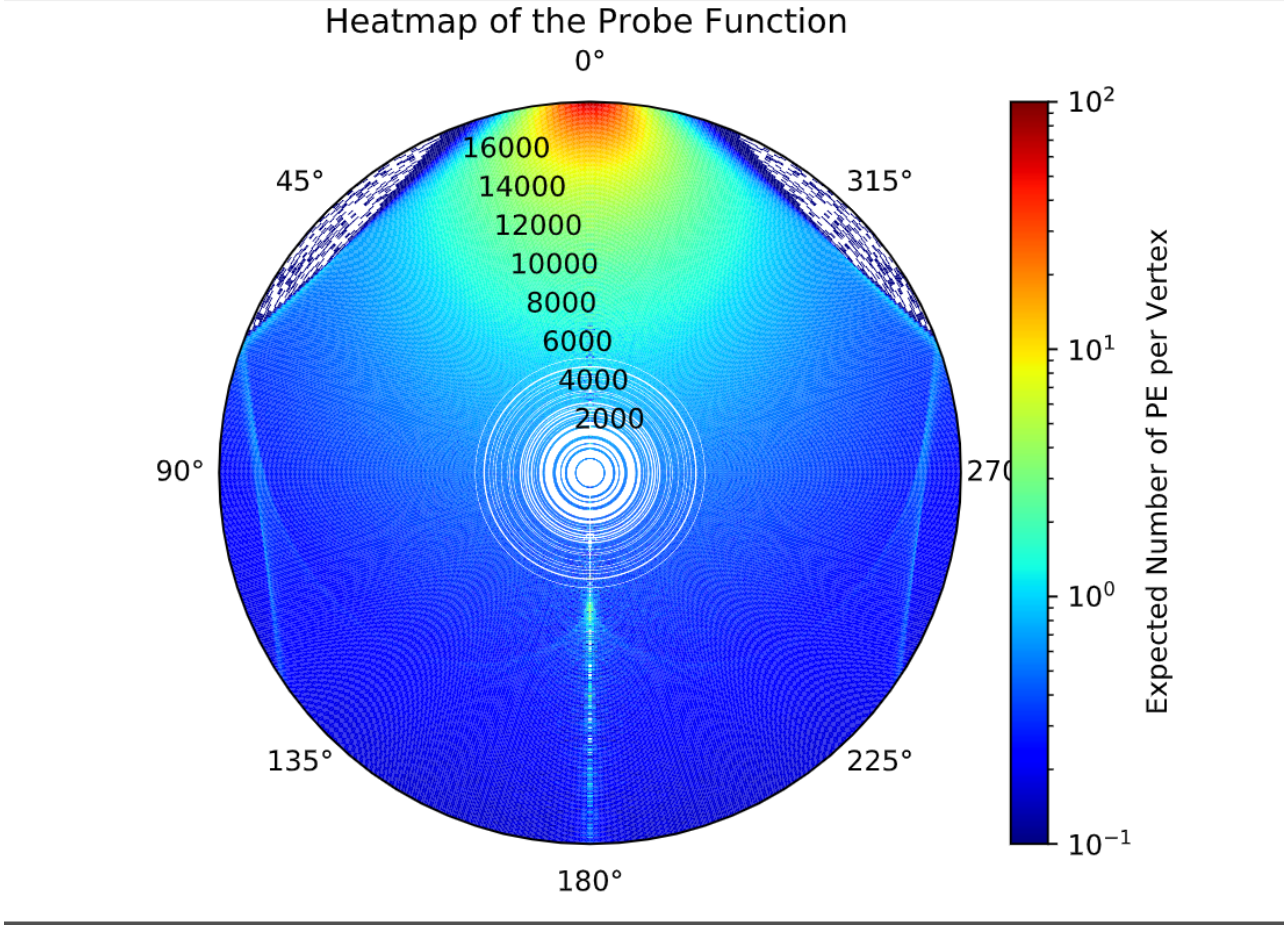

图11：Probe热力学图

在图上我们可以观察到在0°附近有一个密度较高的区域，这部分区域的颜色以红色和黄色为主，这也充分反映了对于同侧靠近PMT的顶点产生的光子有很大一部分可以被PMT 接受到。一定程度上体现了平方反比的性质。

我们观察在靠近上侧大约在45°和315°方位的图像，发现这个地方出现了空白和几个零散的点，这一区间代表了模拟过程之中的全反射性质，其中出现了一定的空白这是因为为了做出更加丝滑和细腻的Probe图像，我们将作图的r方向和θ方向的区间取的十分的密集，并且对于缺少数据点的格点区间没有设置自动填充0，这样做是有意而为之，原因是我们可以在这部分的图像上观察到一次反射:

理论上不设置自动填充就不会出现对应的色块，但是在全反射区域仍然出现了零星的像素点，这些像素点并非凭空生成，而是全反射区域产生的光子经过球内表面的一次反射之后射出的、最终打到PMT上的光子，如果没有一次反射，这些区域就不会有零星的像素点。

在图像的中下部，我们可以清晰的观察到三道竖着的亮纹以及一个类似与ω形状的亮纹，这也是由于一次反射的性质导致的。

5 总结

5.1 加分项目

5.1.1 5分：完整模拟17612个PMT的光学过程

这一部分可以说对本程序而言手到擒来。按照课程要求这一部分的程序只要在24h之内完成这一部分的计算即可，本程序的算法早已远远超过了要求

5.1.2 5分：进阶光学????????

在开始大作业的时候原计划完成这部分的进阶光学功能，但是在完成本部分的过程之中发现了一些端倪：如果要考虑PMT表面的反射，就必须精确的知道光线打在PMT上面的位置。这也意味着我们在模拟的过程中实际上产生了精细描述PMT上落点位置的数据和信息。逆向思维考虑这一问题：只判断PMT击中编号最少究竟需要多少信息量？这启发了我们从信息论的角度去看待这一问题。可以毫不夸张的说，整个程序采用的创新算法的灵感正是来源于这一部分。正是经过了思考和一定的测试，让我们真正意识到了一件事:

模拟过程产生的数据信息总量远远大于判断击中PMT编号所需要的信息量

因此一个自然而然的想法就是我们能不能不去模拟这些冗余的信息，直接逼近判断PMT打击编号的最小信息量。在模拟的过程之中尽可能不产生任何其他多余的数据，以一个全新的方式将整个程序的运行时间压缩到极致，不做任何多余的模拟。经过若干次的尝试和实验，我们终于找到了一个可以高精度几乎完美逼近最小信息量的方法。这正是前面详细介绍的球面差分法，用大球壳对原来的PMT阵列进行切片，再判断光子是否击中PMT的算法。

正如一句谚语：

上帝给人关上一扇门的同时总是会为他打开一扇窗。

我们在算法中扔掉了PMT打击点位置的精确信息，而作为丢掉这些数据的交换，我们获得了常规算法无法匹敌的程序运行速度。即使不采用Numba 的@njit处理循环，不采用任何并行加速手段，一个高配置的游戏本仍然可以在4min左右跑完整个光学过程。这也是JUNO模拟程序第一次以算法上的创新将运行时间缩短到10min、甚至5min以内。

当常规的算法用尽了各种方法加速，(如Numba的@njit，使用并行加速.....)却仍然无法突破上限的时候，本程序的算法从信息论的角度启示着我们：

对于生成PETruth而言，常规算法的绝大多数操作都是在生成和处理冗余的信息

因此我们向老师和助教申请将本程序独创的基于信息论观点的算法创新作为一个全新的加分项目，来代替原来的进阶光学过程。