Master Thesis Journal

Möller, Julius

July 2022

Contents

1 INTEGRAL Query

The first step is to write a class capable of taking retrieving a list of all INTEGRAL pointings observing any given astronomical object or sky coordinates. Additionally it should be able to filter said list of pointings according to relevant parameters, such as the start and end times, coordinates, science window version and type, and various other parameters. The implementation of this class did not require any sophisticated algorithms, and as such will not be discussed further.

2 INTEGRAL Pointing Clustering

2.1 Clustering Conditions

The fundamental idea of this project is the following: the INTEGRAL background event rates per detector per energy bin is assumed to be constant between sufficiently close (in terms of time and orientation) pointings, and can thus be determined analytically for bunches of several pointings via maximum likelihood calculations. This finally allows us to apply Bayesian inference fits to determine source parameters. Since these fits are done on each cluster of pointings, it is pivotal to write an algorithm that clusters a list of INTEGRAL pointings under the following conditions:

- The angular distance between pointing orientations should not exceed a predetermined value within each cluster.
- The angular distance between pointing orientations should be higher than a predetermined value within each cluster. Many successive INTEGRAL pointings have nearly identical sky orientations, for which the coded mask produces nearly identical patterns; meaning that the above described method of eliminating the background parameter in the fits would not work. Hence a minimum angular distance within each cluster is necessary.
- The temporal distance between pointings should not exceed a predetermined value within each cluster.
- The number number of pointings in each cluster should be within some predetermined range of cluster sizes. Depending on the given list of pointings, clusters with fewer pointings than the preferred range may be unavoidable, but clusters with more pointings than the preferred range can definitely be avoided. A simple mechanism to encourage this is to minimize the total number of clusters, while never exceeding the maximum cluster size.
- The pointings should be as close as possible in order to justify the assumption that background rates are constant. Hence the average effective distance (composed of temporal and angular distance) between pointings should be minimized.

2.2 Clustering Algorithm

Clustering points in space together is a common problem in computer science, hence many sophisticated algorithms exist to do so. However, the above listed conditions are rather unique, and I could not find any established clustering algorithms that would satisfy them. The common sentiment for clustering algorithms is that there is no such thing as the perfect clustering algorithm; instead the performance of the algorithm depends on how well it is suited to the type of distributions found in the data. In our data, the pointings are distributed in three-dimensional space (one time and two angular coordinates). However, they are not distributed randomly; instead the sky coordinates of INTEGRAL are adjusted slowly as time progresses. Hence the data are distributed along a line through the time dimension, rather than randomly distributed in the 3D space. With that in mind, the following clustering algorithm was developed:

1. Given some list of pointings looking at an astronomical object, it is quite likely that INTEGRAL spends some time looking in the general direction, and then spends a lot of time looking at entirely different sections of the sky before returning to the astronomical object. Pointings with large temporal distances will never be clustered together, so the run-time of the algorithm can be reduced significantly by splitting the entire query of pointings into smaller, disconnected regions, so that the effective distance between any two pointings from different regions is larger than the predetermined maximum effective distance. However, in order to split the pointings into disconnected regions, the effective distance between any pair of pointings needs to be calculated. Given N pointings in our query, this requires computing an N by N matrix, which is computationally very expensive for large N. To improve this, the pointings are first split into preliminary regions, such that the temporal distance between any two temporally successive pointings in a preliminary region does not exceed the maximum effective distance.

2. Now that we have reduced our large query into several smaller preliminary regions, we can compute the distance matrix for each preliminary region. Hence we have computed the distance matrix for any pair of pointings, except that we have only actually computed the distance for any relevant pair of pointings, and setting the value for pointings in different preliminary regions to some value higher than the maximum effective distance.

3. While our preliminary regions are a good start, it is entirely possible that these are composed of several actual regions; i.e. that a preliminary region contains several regions of entirely disconnected pointings, where the effective distance between any pair of pointings from different regions is larger than the maximum effective distance. Since we have already computed the distance matrix for pointings within preliminary regions, we can use the following algorithm to split these into actual regions:

   (a) Start with some point in the preliminary region, and assign it to a new region while removing it from the preliminary region.
   (b) Iterate through the preliminary region, and add any point with a distance smaller than the effective distance to the region, and remove said point from the preliminary region.
   (c) Repeat the step 3b for every pointing added to the region in the previous step.
   (d) If there are no more pointings to add to the region, repeat from step 3a until the preliminary region is empty.
   (e) Repeat steps 3a to 3d for all preliminary clusters.

4. Now that have split the query into regions, we can cluster each region independently. This is done using the following algorithm:

   (a) First we need create an initial clustering, for which we can take advantage of the fact the the pointings are distributed along a temporal line in the 3D parameter space by iterating of the pointings in each regions according to the start date:
      i. Create a cluster from the first un-clustered pointing.
      ii. Iterate over some predetermined number of temporally successive pointings:
         A. If the pointing can be added to the cluster without breaking the conditions listed in section 2.1, add it to the cluster and repeat from step 4(a)ii. If not, continue the iteration.
         B. When the iteration is finished or the cluster has reached is maximum size, continue from 4(a)i.
   (b) While the initial clustering usually produces good results, there is no reason to assume that it is anywhere near optimal. For this reason we attempt an improvement on this clustering in the following way:
      i. Randomly choose a sub-optimal cluster, weighted by cluster size. This is any cluster with less pointings than a predetermined number.
      ii. Randomly choose a close sub-optimal cluster, weighted by distance from the first cluster and size. A second parameter is used to determine the maximum size of this cluster.
      iii. Connect these two sub-optimal clusters through a path of clusters in the following way:
         A. Find the pair of pointings with the least distance in the two clusters.
         B. Find the the pointings closest to the first of the two above pointings, that are not already part of the cluster path. Weigh these using their distance to the pointing, and by the angle between the vector connecting the first pointing to the target pointing, and the vector connecting the first pointing to the observed pointing. Randomly choose a pointing based on their weights.
         C. If the chosen pointing is in the target cluster, the path is complete. If not, repeat from step 4(b)iiiA using the cluster from the newly selected pointing as the first cluster.
      iv. We will now re-cluster all pointings part of the cluster path:
         A. Randomly choose a non-clustered pointing on the cluster path, weighted by its average distance to all other points on the cluster path. This way, the outlier points, which are most likely to be left without good clusters are clustered first. Create a cluster with this point.
         B. Randomly choose a non-clustered neighboring pointing on the cluster path, weighted by its distance to the current cluster and the distance to all other points on the cluster path. If it satisfies the conditions from section 2.1, add it to the cluster.
         C. Repeat step 4(b)ivB until no points can be added to the cluster.
         D. Then repeat from step 4(b)ivA until all points on the cluster path are clustered.
      v. Calculate a cost of the new and old clustering configuration. Fewer clusters are bettern, and the average distance of all clusters is used as a tie-breaker. Keep the new configuration if it is better, otherwise keep the old one.
   (c) Repeat from step 4b until the number of succesive rejected improvements exceeds a predetermined value.