Joseph Bak

Computational Geometry Project

Central trajectory

A central trajectory can be defined in different ways depending on the applications and the goals. In this project, I have computed central trajectories with three different methods using synthetic data.

I used Python 3 to conduct this project. I have firstly generated one trajectory. Equally distanced points were generated (I made the overall distance of x direction to be longer than the overall distance of y direction for the demonstration purpose.) and normal random variables with 0 mean and small standard deviations are added to each point for perturbation. Connecting these points, one trajectory is generated. By perturbing each point of this trajectory using normal variables, a set of trajectories is generated.

The first way to compute a central trajectory is to find the average point at each time and then connect them to make the central trajectory. This method is intuitive and very easily computed. I used a built-in average function to find the average point.

The second way is to find a point that minimizes the maximum distance to all the points at each time and then connect them to make the central trajectory. At each time, this is equivalent to find a center of minimum radius circle that includes all points. This method cares to include outliers since one outlier can change the center dramatically. Thus, for data in which outliers have more

importance, this method is useful. I implemented a randomized incremental algorithm to find the center at each time which takes expected *O(n)* time, where *n* is the number of trajectories.

The third way is to use a subset of trajectories. At each time, I have computed the sum of distances from each point to all other points and sorted the points in non-decreasing order of the sum of distances. Using the user specified percentage of the data points in the sorted list of points, I found a center of minimum radius circle that includes these points and then connect them to make the central trajectory. Since this method uses minimum radius enclosing circles, it also takes outliers seriously. However, because we use a subset of points that are more clustered at each time, this can compute a more meaningful central trajectory. For clustered data, this method is useful. Since the same algorithm is used as in the second method, this also take expected *O(n)* time, where *n* is the number of trajectories.

I have computed mean squared error using 110 trajectories in which normal variables with 0 mean and 1.5 standard deviation are added to each point of the firstly generated trajectory in both *x* and *y* directions. The first trajectory is generated by 100 data points with *x* ranging from 0 to 50 and *y* ranging from 0 to 5 and normal variables with 0 mean and 0.1 standard deviation are added to *x* points and normal variables with 0 mean and 1.5 standard deviation are added to *y* points. The mean squared error of average trajectory is 0.1400, the mean squared error of 100% minimum radius enclosing circle trajectory is 0.4832, and the mean squared error of 70% minimum radius enclosing circle trajectory is 0.1550. Since minimum radius enclosing circle methods take the importance of outliers, they tend to give higher mean squared errors as expected.

In conclusion, there are many different ways to compute a central trajectory depending on the applications and the goals. In this project, I have implemented three notions of central trajectory, an average trajectory, a minimum radius enclosing circle trajectory, and a subset minimum radius enclosing circle trajectory and their demonstrations using Python 3. Their possible advantages are explained and their mean squared errors are computed for a comparison purpose.