Progress Report: (Week of June 5)

Samuel Wong, Michael Poon

June 5, 2018

1 Samuel's Accomplished Tasks

1.1 Changed the Main Program to Using Natural Units

I created a function that converts galactocentric coordinate to natural coordinate by dividing position by 8 and velocity by 220. I structured to main program code such that in the beginning, when it asks for user's input, it is in physcial units. Then it uses galactic coordinate to search on Gaia catalog. After Mathew's code returns the sample of stars, the main program converts everything to natural units before putting it into KDE. When the function tests the uniformity at a point, it also converts the user-given galactocentric point to natural unit first.

This way, the energy and angular momentum function do not have to be changed at all.

1.2 Tested Fake Density

I made up a density function that explicitly depends only on energy and angular momentum. In particular, this fake density has no I_3. I evaluated uniformity on a few points here. The results are acceptable: all 4 dot products are on the scale of 10^{-8}, which are close enough to zero.

1.3 Tested Quasiisothermal Density Function

I used the quasiisothermal density function (qdf) provided by Gaia to perform the exact same test. I noticed that qdf returns "nan" on a lot of points, so I chose test points carefully. The results are also pretty good. We want at least one number of the 4 dot products to be non-zero. And the result is that even for the smallest 4 dot products we got, the largest component is on the scale of 10^{-2}, which is a significant enough difference from 10^{-8}.

1.4 Figured Out the Limit of the Program

Although we have a fake density function that is analytic and that we know the dot products should be exactly 0, the program still gave number on the scale of 10^{-8}. It seems too large for computer error. It turns out that the problem is due to numerical derivative. This function takes a dx for the approximation of derivative. I did some research and learned that the limit of this method is that if dx is too small, computer numerical error adds up and the result is wrong. On the other hand, if dx is too large, the approximation is off.

According to Wikipeida, the best dx to use when we are differentiating the function at the point x is $dx = 1.5 \times 10^{-8}x$. Since the x we are using are mostly on the scale between 1 and 100, I just set $dx = 10^{-8}$.

Since this is an approximation, we should not expect the accuracy of the program to be any better than 10^{-8}. So I concluded that for this program, anything on the scale of 10^{-8} or smaller can be considered zero.

Combining this understanding with the two tests above, I conclude that our code passes the feasibility test. That is, it is able to tell whether there is I_3.

1.4.1 Result of Test

For record, here are some sample results of the two tests.

qdf test

$$(x, y, z, vx, vy, vz) = (1, 0, 0.1, 0.1, 1.1, 0)$$

$$\nabla\rho \cdot w_0 = -0.017600017412602392$$

$$\nabla\rho \cdot w_1 = 2.755227164042471e - 05$$

$$\nabla\rho \cdot w_2 = -0.0824306851216613$$

$$\nabla\rho \cdot w_3 = 6.998240539971299e - 06$$

fake density test (fake density = energy**2 + angular momentum**2)

$$(x, y, z, vx, vy, vz) = (1, 0, 0.1, 0.1, 1.1, 0)$$

$$\nabla\rho \cdot w_0 = -8.465207318453238e - 08$$

$$\nabla\rho \cdot w_1 = -5.016540884788829e - 09$$

$$\nabla\rho \cdot w_2 = -5.822756993244699e - 07$$

$$\nabla\rho \cdot w_3 = -4.03879075473333865e - 09$$

qdf test

$$(x, y, z, vx, vy, vz) = (1, 1, 1, 1, 1, 1)$$

$$\nabla\rho \cdot w_0 = 0.02179621073087118$$

$$\nabla\rho \cdot w_1 = 0.025592540925335944$$

$$\nabla\rho \cdot w_2 = -0.7061519062631407$$

$$\nabla\rho \cdot w_3 = 1.7878869656866615e - 05$$

fake density test (fake density = energy**2 + angular momentum**2)

$$(x, y, z, vx, vy, vz) = (1, 1, 1, 1, 1, 1)$$

$$\nabla\rho \cdot w_0 = -5.801330038579522e - 08$$

$$\nabla\rho \cdot w_1 = 3.517717682175272e - 08$$

$$\nabla\rho \cdot w_2 = -1.0187098473807277e - 08$$

$$\nabla\rho \cdot w_3 = -6.2952830062572296e - 09$$

2 Michael's Accomplished Tasks

2.1 Created Meshgrid Function

Instead of searching point by point in $\mathbb{R}^6$ to evaluate the KDE function, I created a 6-dimensional meshgrid list of $\mathbb{R}^6$ tuples so the main program can loop over any subspace in $\mathbb{R}^6$.

2.2 Created Main Program with Grid

I incorporated the meshgrid function into the main program and ran preliminary tests with Gaia DR2 data. Most of the code ran well except we received several "nan" results for the directional derivatives. This is because when evaluating a meshgrid point at the origin of the milky way, the potential(MWPotential2014) is undefined. To fix this, I added an if statement in the main program that checks whether each point include the origin and if so, raises an exception.

I was originally going to add a code the create mesh grid tool that artificially delete all points in the grid that include the origin. But I had troubles with numpy "delete" function.

3 Mathew's Accomplished Tasks

3.1 Implemented local search

Using `gaia_tools`, I have reimplemented the search function using a locally downloaded copy of the Gaia data. There were a number of challenges with implementing this search. First, `gaia_tools` had a minor incompatibility problem with Windows that I had to fix. Also, loading the entire Gaia catalogue is very memory intensive. As a trade-off, the search function (which now uses numpy arrays and astropy coordinate transformations, rather than ADQL) is extremely fast. I will work on more efficient ways to store the data in memory.

4 Next Steps

Allow main program to select point or meshgrid search