# Progress Report: (Week of June 19)

Samuel Wong

June 19, 2018

## 1 Samuel's and Ayush's Accomplished Tasks

### 1.1 Came Up with the Concept of Pixel for Sample_V_On_Set

I have a working version of the code sample_v_on_set that takes a set of data and interpolate the sample velocity. I have hidden all the hard part of optimizing for accuracy versus time to the definition of two numbers: $R-pixel$ and $z-pixel$. These two numbers describe how fine the grid are. The smaller the pixels, the longer the runtime, but higher accuracy. We are planning to have an algorithm that takes from the user a maximum error allowed and generate two idea pixels. But currently, I am leaving them as predefined constants in the code to test the function.

### 1.2 Entertained the Idea of Interpolating on Subset

I was considering the idea of choosing a subset of the input data and interpolate those values. This way, the interpolation time is not a burden, since those interpolation would have been done anyway. But choosing a subset that is a good representation of the data is hard. Perhaps Voroni that Michael is working on might help. But I am leaving this as a future option after we finish the rectangular

### 1.3 Debugged Sample_V_On_Set

I had some problems with Sample_V_On_Set with bugs about the order at which I go through a double for loop. It turns out that as I loop through a matrix, I am thinking of x-y coordinate while numpy thinks in y-x coordinate.

### 1.4 Found a Limit in the Galpy Sample V Code

So far I had in mind that a Galpy sample V function that can take any point as input and output in about the same time. Because of this and the desire to extend the sample V, we wanted sample v on set to do the same thing and avoided the idea of setting limit of input data.

It turns out this cannot be the case. We realized that in natural unit, galpy sample v freezes at the point (15,0). This turns out to be 120 kpc away from the center of MW in radius. At (14.9,0), sample V still works. It seems (15,0) is the point where it stops working, or at least way too slow.

Learning about this limit on Galpy sample V, we realized that we can set a similar requirement on user input of Sample_V_On_Set. So the hope now is that in the given limit of the input, we can find a pixel good enough that it gives good accuracy as well as good time. Then there is no need to optimize for pixels!

### 1.5 Wrote a Test Code that Calls and Tests Sample_V_On_Set

Wrote a module that calls Sample_V_On_Set, generates a group of (R,z) data to test it, and calculate the fractional error afterward, as well as timing the program. The larger input, the more time advantage Sample_V_On_Set has.

However, the problem is that the fractional error is really large, about 500%. So I changed both the Sample_V_On_Set and the test module so that they average out the velocity multiple times. But this does not cuts down the error.

## 2 Ayushs Accomplished Tasks

### 2.1 Tested the Limit of Galpy Sample V by Timing

I tested at what point sample V stops working by timing sample V on a series of input and plotted the result in notebook.

## 3 Michael's Accomplished Tasks

### 3.1 Using KMeans as an optimal clustering algorithm

In analyzing varios clustering algorithms on Python's Machine Learning Library: Scikit-Learn, we chose KMeans as the best solution due to its fast runtime (linear in Big O), clustering style and the ability to choose the number of cluster centres.

### 3.2 Tested STD vs IQR vs Cluster Scale on KMeans

Due to the fact that KMeans clusters spherically in Euclidean space, in higher dimensions (greater than 2), dimensions $(x, y, z, v_x, v_y, v_z)$ with a greater range dominate the clustering process. This is clear for velocities (See Defining KMeans notebook). To remedy this, we experimented with dividing each velocity manually by a constant called Cluster Scale, or divided each dimension by its respective STD or IQR. This made clustering better empirically when displaying clusters in 2 3D plots of $(x, y, z)$ and $(v_x, v_y, v_z)$.

### 3.3 Minibatch KMeans

Minibatch KMeans is an optimized version of KMeans for larger sample sizes of greater dimensions. This variation runs regular KMeans in multiple smaller batches to reduce memory usage and uses previously generated cluster centres as the initialization for the following batches. Minibatch KMeans uses a gradient descent technique to find a well clustered centres after convergence is reached.

# 4 Mathew's Accomplished Tasks

## 4.1 Wrote Search Local

## 4.2 Changed Search Local Such that Spyder can Reload With Data Saved