Chris Huang

Spring 2018 Research

Disk Star Fitting

Abstract:

Data released by the SDSS have been fit to all parts of the Milky Way. The data we wish to analyze lies in stars in the Milky Way Disk. The goal of this project is to create a program which when given stars as a function of distance we can find a best fit for stars as a function of distance when we account for uncertainty and completeness. The stars we give it will be 2.5 by 2.5 degrees “lines” that point to the galactic anti-center.

Algorithm Overview:

1. Begin with a starting set of stars as a function of distance.
2. Convolve these stars with a Gaussian
3. Multiply these stars with their corresponding completeness coefficient
4. Run a Gradient Descent until a best fit emerges, comparing the starting set of stars with observed data.

Starting Set Data:

This starts with stars as a function of distance. The stars we use are turnoff stars and we use these stars because they are relatively abundant in the sky, and because we know the absolute magnitude of these stars.

This data is represented as number of stars in a vector of doubles. Each entry into the data represents one half of a magnitude space. Note that in the program we have hardcoded with the 0th index representing the 16-16.5 magnitude space, and the 1st index representing the 16.5-17.5 magnitude space, and etc. The 0th index would then hold the number of stars in the 16-16.5 magnitude space and the 1st index would hold the number of stars in the 16.5-17 magnitude space and etc. Note that the starting set of data can start at any value as it will converge to try and match the observed set of data.

Convolution:

This part takes a histogram of star counts as a function of distance, represented by the data set we have above, and convolves it with a Gaussian distribution centered at 4 with a standard deviation of 0.6. We do this to account with the uncertainty in the magnitude of turnoff stars. The Gaussian we use is:

As this is a discrete convolution, the Gaussian is stored in an array of 5 elements. Each element in the array corresponds to an area from a range in the Gaussian. The elements in the array correspond to the following:

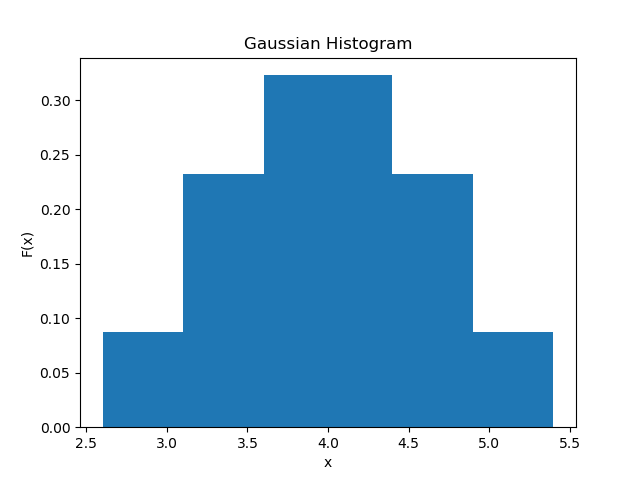


Fig 1) The Gaussian represented as a histogram. The bin volumes are shown above. Note that the first bar represents t[0], and the second represents t[1], and etc.

This convolution is slightly different from a normal discrete convolution however. When we are convolving the star counts for values on or near the start/end of the star counts, normally the parts of the Gaussian would stick off the edge of the star counts array and those parts that “stick off” would evaluate to 0. However, in our scenario if we encounter Gaussians that “stick off” the edge of the histogram, we will instead remove those bars from the Gaussian and for that step only, convolve with a Gaussian that is without its edge bars. To preserve the normality, we renormalize the Gaussian histogram by taking a bar and changing its value so its new value is its old value divided by the sum of the remaining Gaussian bars.

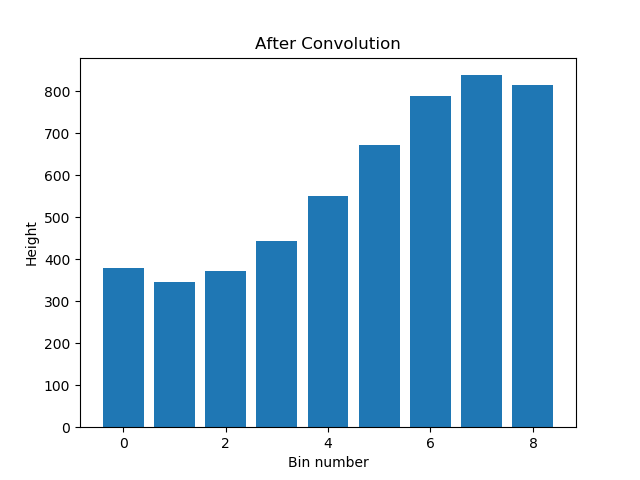
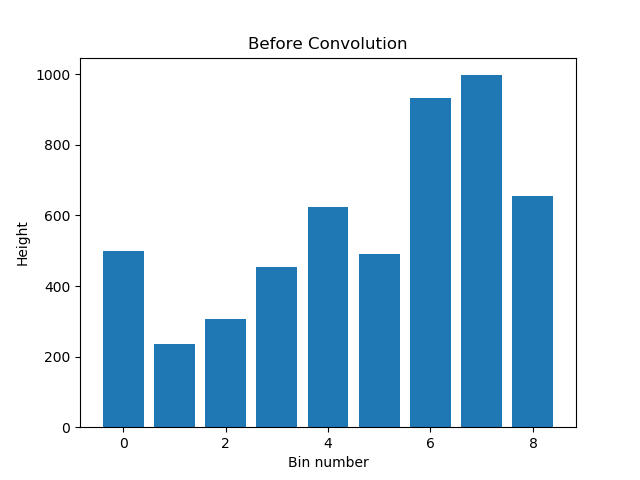


Fig 2) An example discrete convolution preformed on a random histogram using the Gaussian described earlier. Note how the convolution is “smoother” and has the same total volume.

This function has order notation O(n \* m) where n and m are the number of elements of two histogram being convolved.

Completeness Coefficient:

Once we have the convolved histogram, we need to account for detection efficiency. This is done by multiplying each bin by its respective completeness coefficient. The equation for a bin’s completeness coefficient is as followswhere g represents the magnitude.

For magnitude we use the halfway magnitude for a bin. So if a bin represents stars for magnitudes 16 to 16.5, the magnitude we would use would be 16.25.

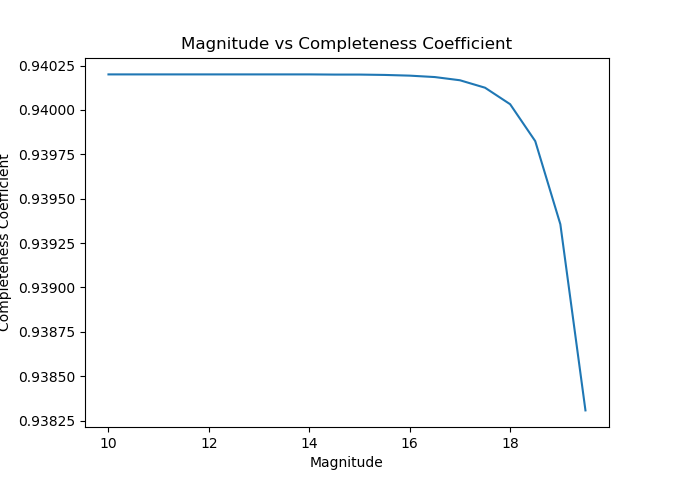


Fig 3) A function of Magnitude vs. Completeness Coefficient. This is the graph of the Completeness Coefficient equation.

Optimization:

We now take our transformed histogram and then compare it with observed data using a standard chi-squared test of fit. The optimizer is configured to run a gradient decent with the chi-squared as a measure of fitness.

To test whether the optimizer works, we use a starting made up stars as a function of distance, then convolve and multiply the completeness coefficient to get an “observed” data count. In practice this observed data would be pulled from the SDSS. We then run the optimizer and if the end result matches what we put in initially as our made up star count we have confidence that the optimizer as a whole works. The optimizer also has a min-improvement flag option. This may be needed to be changed lower or higher depending on the runtime of the optimizer. Ideally the optimizer should only take a few minutes to finish. Currently the whole process will return the fitted histogram in a vector called storage in the program.

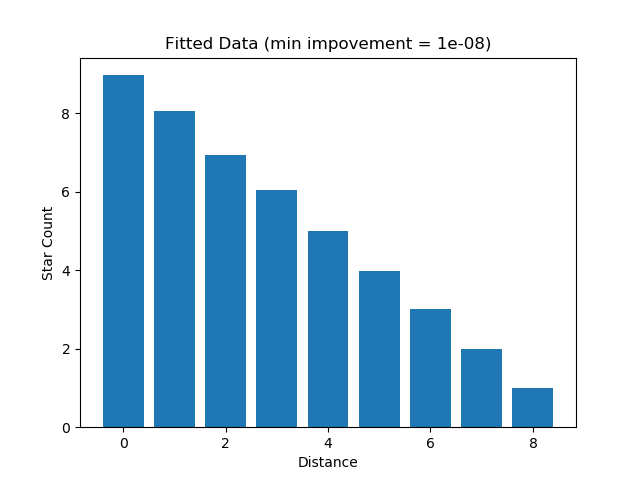
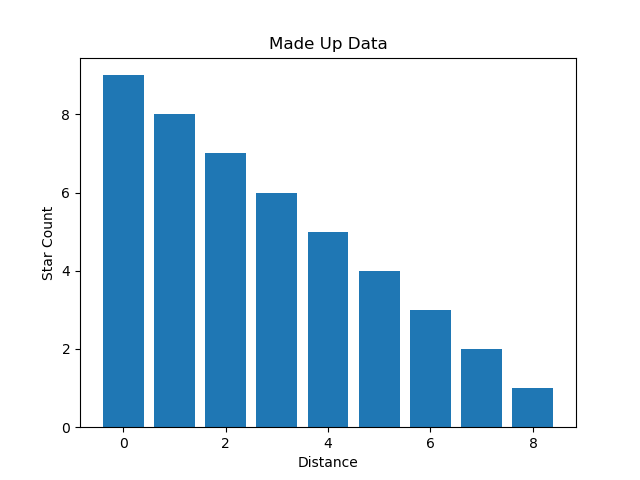


Fig 4) A set of “made up” data to test the optimization process and its resultant output. Note how the two match. The Distance units on the bottom run from 0-8 however these map to their own magnitude spaces. In this case 0 maps to Magnitudes 16-16.5, 1 maps to 16.5-17, 2 maps to 17-17.5, and etc. This is done this way to improve the formatting of the histogram.

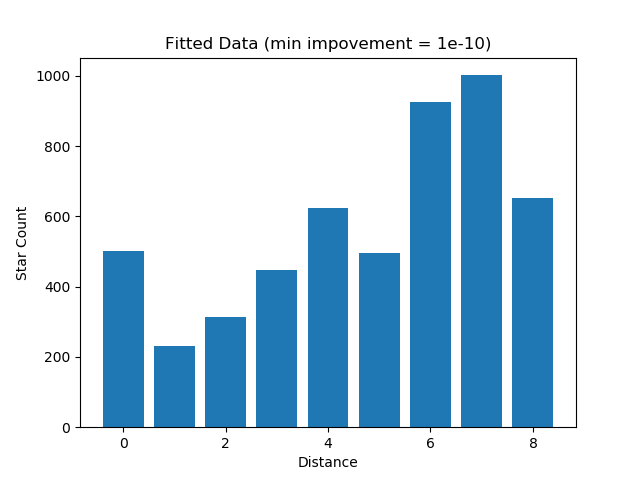
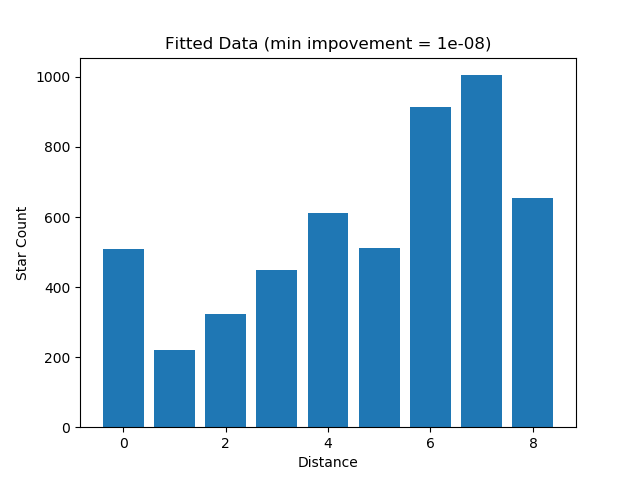
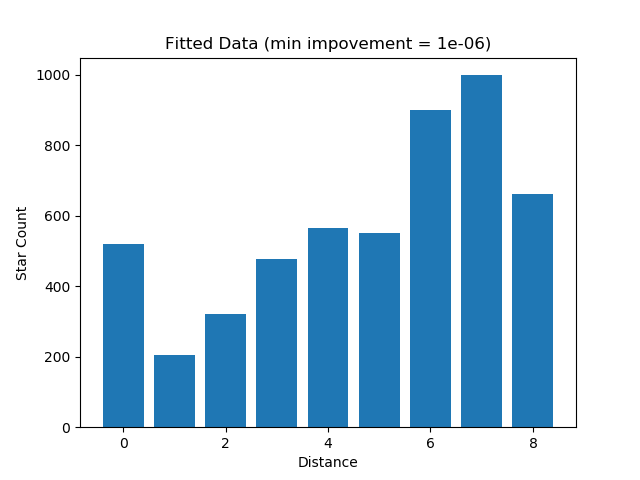
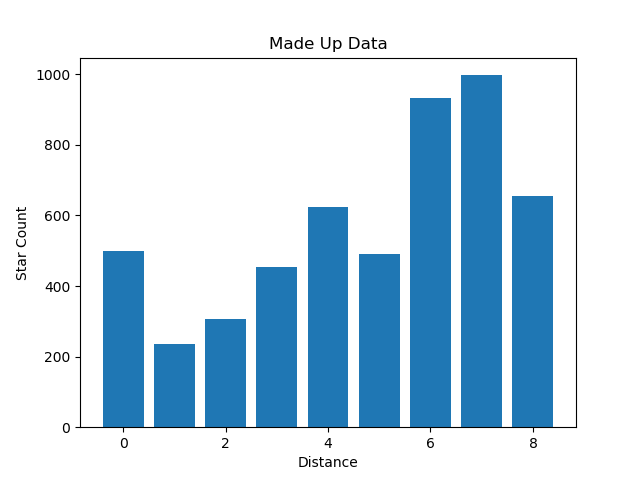


Fig 5) Some made up data and the best fits for different min improvements. Note how the smallest min improvement yielded the best fitting data. For this case, the 1e-06 ran only for a few seconds while the 1e-10 took about one minute to run. However, the 1e-10 found trends not noticed in the 1e-06. The Distance units on the bottom run from 0-8 however these map to their own magnitude spaces. In this case 0 maps to Magnitudes 16-16.5, 1 maps to 16.5-17, 2 maps to 17-17.5, and etc. This is done this way to improve the formatting of the histogram.

Extra Functionality:

There are also functions that find volumes and distances for magnitudes. These are not used but can be applied to find densities in each bin at the very end once the optimizer has finished fitting values.

To do:

1. The program is configured for testing, taking in “made up” data and spitting out the best fit histogram. The program needs to be set up to read in real data.
2. Data needs to be parsed and 2.5 by 2.5 “squares” need to identified and isolated. This data lies in a file called MSTODiskStars\_weissj3.csv and can be found on GitHub. This file is very large, and this will not be an easy task.
3. Parsed data then needs to be run and results need to be stored.