Programming for Astronomy and Astrophysics

Assignment 3: Stellar Density in a Globular Cluster

Deadline: Thursday 19th October at 23:59 – submitted via Canvas as a Jupyter notebook.

Goals of the assignment

By doing this assignment you will show that you can:

- Access data in FITS files and display FITS images.
- Using online documentation as a guide, successfully apply different astropy and numpy sub-packages to carry out an analysis of astronomical imaging data.
- Write model functions, and fit simple models to data using curve-fit to obtain the model parameters.
- Plot the results you obtain, using clear labels on your plots.
- Set out your analysis in a clear and logical way in a Jupyter notebook.

Introduction

This final assignment is based on an image of a dense star cluster (globular cluster NGC 104, a.k.a. 47 Tuc) obtained by the UVIS detector of the Wide Field Camera 3 (WFC3) on the Hubble Space Telescope. The file containing the image data, `ic2r02050_drz.fits` is in FITS format. The image data you need is in the first ('SCI') extension. One of the main goals of the assignment is to carry out an analysis of the distribution of stars in the cluster, using a variety of sub-packages from astropy, numpy and scipy. A number of these sub-packages and functions are not discussed in the online Lesson material, but you should have learned enough about Python to be able to use them based on their online documentation. This is one of the most important steps towards being able to work independently with Python to do research.

What you need to do

You should use a Jupyter Notebook to complete this assignment – you can write any functions you create in the first cell and use them to read in the data and make plots in subsequent cells. Be sure to include comments (`#`) and docstrings (``` ``` ```) in your code to explain it.

1. Use the FITS functionality from astropy to open the FITS file. Access the image data in the FITS file and create a plot of it using matplotlib. You will have to adjust the range of pixel values that are shown by matplotlib in order to show the stars clearly. Look at the documentation of `pyplot.imshow`, there are relevant keyword arguments to adjust this range. The image you just created has no sky coordinates along the axes, instead it has pixel coordinates. astropy contains a module `astropy.wcs` which can deal with the World Coordinate System, which allows you to create plots with proper sky-coordinates. Read through the documentation for the `astropy.wcs` module. It shows you how you can add the sky coordinates to a plot of the image data. You should look through the FITS header and confirm that it contains a WCS section. It is that section that describes how the image pixels are

projected on the sky. Create a plot of the globular cluster NGC 104 that shows the stars in the cluster and has axes labeled with sky coordinates.

2. Next, you will use the `astropy.photutils` package to detect the stars in the image. Unless noted otherwise you should perform the following analysis using pixel coordinates. Install the `photutils` package if you don't already have it installed. The `photutils` package expects data with the background subtracted and without `nan` values. Use numpy to mask nan-values and determine the median of the image. Subtract the median from the image and after that replace the `nan`-values with zero.
3. Now, use `DAOStarFinder` from `photutils` to detect the star positions in the image. You can use `fwhm=3` and you should experiment with the value of `threshold` (which should be small compared to the flux per pixel in a typical star). Plot the detected stars over the image of the NGC 104 globular cluster – make sure that the star positions do line up with actual stars on the image and there are not many false detections of stars.
4. You will now perform some simple analysis (histograms and function fits) to find the approximate cluster center. In this part of the exercise we will again perform our analysis in pixel coordinates. Use the `numpy.histogram` function to create two histograms. One for the x-positions of the detected stars, and one for the y-positions. (We ignore the distortion in the image, and assume the cluster to be circular). Use on the order of 100 bins for each of these histograms. Create an array with bin centres (i.e. centre point of each bin) from the bin edges returned by `numpy.histogram`. Create a function for a one-dimensional Gaussian and add a term to be able to model the constant background level (due to fake detections and stars in front of the globular cluster). For both the X and Y star position histograms perform a fit. Fit the Gaussian model you created, to the bin values and bin centers. You can provide an initial set of parameters for the curve fit function using the p0 keyword argument (you can estimate values for the starting parameters from a plot of the histograms). Provide plots of your histograms with the fitted functions drawn over them.
5. Next, write a function which calculates the stellar density profile of the cluster: this is the number of stars per square arcsec vs. radius in arcsec from the centre of the cluster. To do so, you can first create a histogram of the pixel radii (i.e. radius from the measured cluster center) at which the stars were detected. The UVIS detector pixel size corresponds to 0.04 arcsec per pixel on the actual sky image. You should limit the maximum radius considered in order to not include the edges of the image, which distort the measurement (we recommend limiting the radius to < 2000 pixels from the cluster centre). Next divide the histogram values for each radial bin by the area of that bin (which corresponds to an annulus with inner and outer radii set by the bin edges). You can also estimate error bars by taking the square-root of the number of stars in each radial bin and then dividing by the area. Plot your calculated values of stellar density (with error bars) vs. radius using a log-log scale.
6. Finally, fit the stellar density profile with a 'King model' profile, commonly used for globular clusters:

$$n(r) = n_0(1 + (r/a)^2)^{-\gamma/2}$$

where r is the radius, a is a radial scale-length, γ is a power-law index and n_0 is the normalization at zero radius. You can use your error bars as the values for the sigma parameter in `curve_fit`. Plot your best-fitting model together with the data and errors (also using a log-log scale). Then use your results to calculate and state the 'core radius' (r_c) of the cluster in arcsec:

$$r_c = a\left(2^{2/\gamma} - 1\right)^{1/2}$$

Hints and tips

- Look carefully through the documentation for the functions that you will use – there are many examples given but not all of them are relevant to what you want to do.
- It helps to be able to explore the images without rescaling or replotting them each time – put the `%matplotlib notebook` command in the first notebook cell so that your plots are interactive and you can zoom in on them and explore them, e.g. to see that detected star positions match with the stars appearing on the image.
- The Gaussian fits to x- and y- positions will be better if you skip the first and last few bins in the histograms. These bins are lower artificially because they are not covered uniformly by the image. Because of this they throw off the fitting algorithm.