Bayesian Star Identifier
Setup and User Guide
`github.com/otoolec2/BSI`

Conor O'Toole

July 2, 2017

Abstract

Bayesian Star Identifier (BSI), software employing Bayesian statistical algorithms for analysing Hubble Space Telescope* (HST) images of star fields is outlined, along with instructions to download and install the software to carry out said analysis. BSI makes use of the existing nested sampling software DNest4 [1], along with custom Python scripts to carry out all pre-processing necessary for compatibility with DNest4 and basic post-processing to give indications of number of sources predicted in each sample, their positions, fluxes, etc. For the point spread function (PSF), a Moffat profile is fitted to the appropriate PSF generated using the online utility `tinytim` [2]. Included in the code repository is a script to generate a random image for testing purposes, along with a sample script for conducting further analysis on a particular source of interest. A short appendix containing simple instructions for running BSI is included at the end of this document.

Contents

1 Introduction

Bayesian statistics provides a powerful set of tools for data analysis. The general practice involves the use of measured data and a theoretical model to update the predicted probability that a certain point in parameter space generated the data studied. This can be summed up in the formula

$$p(\theta|D,M) = \frac{p(\theta|M)p(D|\theta,M)}{p(D|M)}$$

where θ is the family of parameters which determine the observed data, D is the data in question and M is the model which generates it. $p(\theta|M)$ is called the *prior probability*, while $p(\theta|D,M)$ is the *posterior probability.* A standard method to obtain these posterior probabilities is to sample points in parameter space, use the model M to generate corresponding "data" which that point in parameter space corresponds to, and compare it to the actual data D being analysed. Naturally, this method is strongest for complex models or noisy data samples, where multiple points in parameter space may generate reasonable approximations to the observed data. For a more in depth discussion of Bayesian statistics, as well as the particular sampling algorithm used in this project, see the documentation provided with DNest4 [3].

The data considered in this work is HST images of star fields. Of particular interest are images of crowded regions within galaxies obtained prior to the detection of an extragalactic transient, with the aim of identifying the progenitor. Given that the precise number of source in such images is typically uncertain, with instrument noise disguising dim sources, Bayesian statistics is well-suited to analysing them. In fact, similar analyses have been conducted using DNest 4, or an earlier version [4, 5].In the model used here, Moffat profiles are used to describe the PSFs of the sources. The Moffat 25 case is assumed (ie. the parameter β below is assumed to be 2.5), and the profile is fitted to `tinytim` generated PSFs, which accurately describe the PSFs for a number of cameras, filters, and pixel array positions for the HST. This profile is described by the formula

$$PSF(x,y,\alpha) = \left(a + \frac{b(x^2+y^2)}{c\alpha^2}\right)^{-d\beta}$$

where a, b, c and d are the fitting parameters. A typical fitted profile is shown in Figure 1 (1-dimensional slice). Thus, a noise-free image containing N sources can be described at pixel (x, y) by

$$I(x,y) = \sum_{i=1}^{N} f_i \; PSF(x - x_{ci}, y - y_{ci}, \alpha_i)$$

where f_i is the flux of source i, and (x_{ci}, y_{ci}) is the position of the center of source i, and $I(x, y)$ is the intensity at pixel (x, y). The entire image can then be obtained by calculating this sum at each pixel. It should be noted that we allow α to vary for different sources, as images may contain extended sources such as binaries, which may not be described accurately by the α parameter of a point source, or may skew the predicted values of alpha.

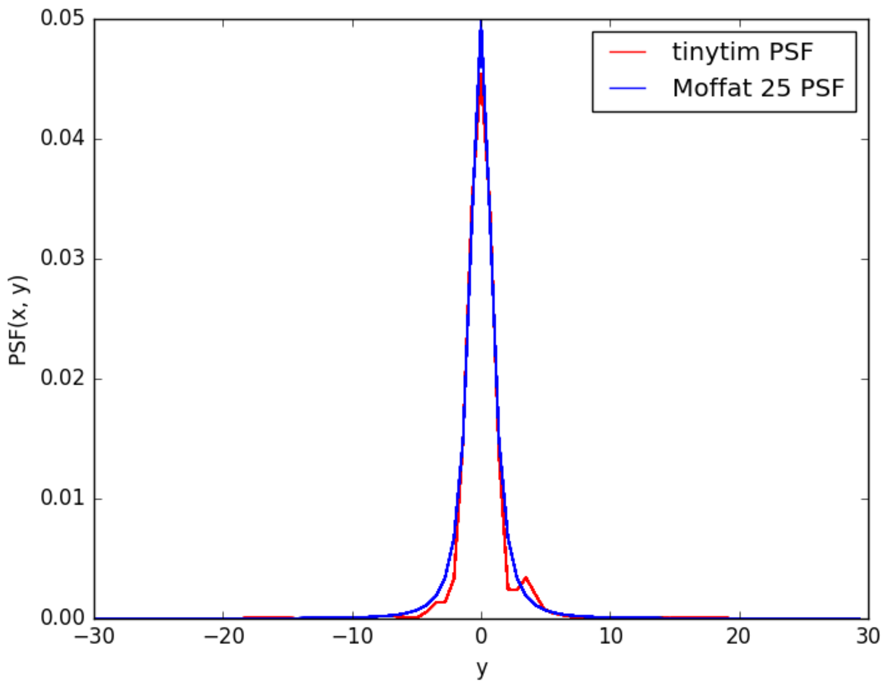

Figure 1: The Moffat 25 profile fit is quite good in the primary peak, though not with the smaller side peaks. These peaks are important, and may represent real phenomena such as Airy rings. However, the amount of flux contained in such peaks will be so much smaller than in the primary peak that for our purposes such a fit will suffice.

Within DNest4, for each sample with a given number of sources, each with randomly generated parameters x_c, y_c, f, α, the code will generate what the image depicting these sources, plus the mean of the noise profile, would look like. This is subtracted from the actual image and the square of this result divided by the variance of the noise profile. It is this array of residuals which is used to determine the *likelihood* of this sample ($P(D|\theta, M)$ in the formula above). This instructs DNest4 on whether to add or remove sources when generating the next sample (this is done using the class `RJObject` if the user wishes to consult the DNest 4 code and literature for more information). The 4 parameters being varied are all initially assumed to obey uniform distributions with limits determined both by the user and the input data.

In the following sections, the steps necessary to setup BSI, as well as a user guide and explanation of a number of the code's features will be outlined. The technicalities of DNest4 will not be explored in greater depth unless necessary, as it is intended that users need not edit the C++ code unless they wish to use dramatically different models or data formats to those used here.

2 Setup

Before downloading BSI to analyse HST images, one must first download and install DNest4. This can be obtained from [1], along with installation instructions. Note, some issues were encountered during this process, as this project was completed using a Windows machine. In particular, the Python bindings to DNest4 could not be properly installed. Hence, this software simply makes use of the C++ code for the statistical analyses. If the capabilities of the software discussed here is sufficient, then users should have no need to delve into the DNest4 source code. The exception to this is obviously if one wishes to use a different PSF, or allow a different family of parameters to be varied. Such cases are not discussed here, but DNest4 comes with a number of examples and templates which should prove useful to anyone wishing to familiarize themselves with the C++ code to a sufficient degree to begin creating new models, in particular the galaxy field example, which this code is heavily based on. It is recommended that the DNest4 installation directory be added to the PATH variable. However, if that is unsuccessful (as it was during this project), the Makefile provided with the BSI code contains a variable to which can be assigned the necessary path. Editing this variable will almost certainly be required, as for convenience during this project, the DNest4 repository was contained within the BSI parent directory. Thus set the variable `DNEST4_PATH` in the Makefile to the directory above `/DNest4` (or delete it if it is no longer required).

Following successful DNest 4 installation the necessary C++ scripts, pre- and post-processing scripts can be downloaded from `github.com/otoolec2/BSI`. The directory structure is shown in Figure 2. No installation is required for this code to work, though it does require the following Python modules to run: `pylab`, `numpy`, `os`, `sys`, `matplotlib.pyplot`, `scipy.optimize`, `datetime`, `astropy.io.fits`, `copy`. Most of these should be common for those in the field of astronomy. The Python module `dnest4` is also required, but should be present once DNest4's installation is complete. In order to test successful setup, run the `RandImage.py` script provided with the code to obtain a simple test image, then follow the instructions in Section 3 or Appendix A to run DNest4 on this sample image. Note that the `/Images`, `/Output` and `/Output/Frames` folders all contain simple examples of typical files that these folders would hold. These can all be deleted for the sake of tidiness.

```
/BSI
       /Photometry
            /Data
       /Images
       /PSFs
       /Output
            /Frames
```

Figure 2: The directory structure of the software. `/Photometry` contains all of the necessary code and information for DNest4, while all python scripts for pre- and post-processing are contained in the parent directory `/BSI`.

3 Running Software

Running the software on an image can be described in terms of three stage: pre-processing, DNest4 analysis and post-processing.

3.1 Pre-Processing

At this stage, all of the necessary steps needed to convert the data into a format compatible with DNest4 are completed. The user must obtain the following before executing the script `Pre-Processing.py`:

- FITS file of $m \times n$ pixel image (this code will allow for rectangular images)

- FITS file of relevant PSF obtained from `tinytim`.
- The limits on x and y within the image (these are also the limits of the uniform distributions of source positions).
- The mean and standard deviation of the noise profile associated with one's image, which can be obtained via standard analysis.
- The value assigned to "bad" pixels (eg. -200).
- Some estimation of the *maximum* number of sources likely to be in the image

The need for most of these items should be apparent. The PSF is two dimensional, so a two dimensional fit must be completed, thus the FITS file output of `tinytim` is ideal. Note, these are re-usable, so analysing a second image at a later date which would be described by the same PSF does not require the user to obtain a new file from `tinytim`. For the x and y limits, one is free to choose any numbers. However, this may make it difficult to compare DNest4's predictions with initial guesses. For instance if one is examining an image which runs from $x \in [634, 644], y \in [620, 630]$, with a source believed to be at $(639, 625)$, if one chooses limits of ± 1 for x and y, then one will have to manually perform the necessary translation later. The "bad" pixel value is necessary as it is usually a number which would certainly not occur naturally. Thus, bad pixels could actually be identified as sources by DNest4 if not accounted for. Such pixels are dealt with by simply setting them to the mean of the noise profile, so they should not register as sources. If the user is analysing an image in which they believe a source of interest is partially occluded by a bad pixel, they may have to devise a more sophisticated means of handling said pixel.

The estimate of the maximum number of sources is a result of the structure of DNest4. The algorithm requires some sort of upper limit, in particular for memory management. It is advised to overestimate, as this prevents certain possibilities from being excluded, when in fact they may be some of the most likely. However, overestimating by too much can cause the code to run significantly slower, so it is advised that the user not simply pick, say, 10,000 for a 10×10 pixel image. Note that if one does underestimate, it will become clear in the post-processing, as the histogram will skew severely towards the maximum chosen, at which point one could simply run the analysis a second time with a larger maximum.

Once all of these necessities have been obtained, the FITS file of the subject image should be placed in the `/BSI/Images` folder, and the Fits file of the PSF in the `/BSI/PSFs` folder. The user can then execute the script `Pre-Processing.py`. The user is free to alter the code directly if desired. However, in the interest of simplicity, all of the numerical data entered can be input by the user following specific prompts in the console. If the user does not wish to use such console prompts, simply comment out all code between lines 72 and 252 (multiple while loops are used to attempt to catch any user errors) and add declarations of the relevant variables:

- `xmin`, `xmax`, `ymin`, `ymax`: limits of x and y which image runs over.
- `NMAX`: Maximum number of sources likely to be in the image.
- `amin`, `amax`: Estimated minimum and maximum of α parameter(s), used as limits of uniform distribution.
- `mu`, `sigma`: Mean and standard deviation of noise profile.
- `bad_data`: Value assigned to "bad" pixels.

Note, the user must enter the relevant image and psf filenames, along with directory paths (if different from the default) on lines 61-69, as there is likely to be too much variation on a user-by-user basis. In addition, the user may wish to run the analysis on multiple images (eg. images of the same FOV, but with different filters), and so it would be awkward to enter the paths on each run.

Another point worth mentioning is that the user at no point enters the limits of the distribution of the flux of sources in the console. These limits are obtained from the image. The code simply sets the maximum as the maximum value within the image and the minimum as the minimum. If the minimum is below zero, it is reset to zero. While this may seem to ignore the contribution of noise, all limits of distributions entered here are actually just first guesses, and DNest4 can generate new distributions with maximums up to five times larger if necessary, so one could obtain predicted values of the flux larger than the maximum defined in pre-processing.

3.2 DNest4 Analysis

Once pre-processing has been completed, the actual DNest4 analysis can be run. On the very first run, the user will need to build the C++ code. Simply enter the directory `Photometry` in the console and enter the command `make`. As noted in Section 2, it will be necessary to alter the `DNEST4_PATH` variable within the makefile, or even remove it (it was added due to some issues with adding DNest4 to the PATH variable on a Windows system). This is likely to be the only cause of issues, as if DNest4 has installed successfully, all the necessary libraries and modules should be present. Note, this step should not be necessary for future analyses unless one alters the C++ code.

With the C++ code successfully built, the last step before execution is to examine the OPTIONS file in the `Photometry` directory. The values in this file can alter the speed of the code (number of particles) or the maximum number of data points it will save and how often it will save them. The defaults should be adequate for most purposes, though it may be useful, for instance, to change the maximum number of saves from 10000 to, say, 2000 for a shorter duration test run. Note the code will not need to be rebuilt following alteration of this file. For more information on this file, see the DNest4 github repository and documentation [1, 3].

Compilation of the code results in an executable `main.exe`. Running this executable will start the DNest4 analysis, and it will end either upon reaching the save limit or user interruption. For a 10 by 10 pixel array, a save interval of once every 10000 samples, a save limit of 10000, and a maximum source limit (`NMAX` from pre-processing) of 20, this stage takes approximately 5 hours on a Windows 10 machine with an AMD Quad-Core Processor A4-5000. The code uses very little RAM (On the order of 10's of Mb according to Windows Task Manager), the long duration is primarily due to the large number of samples and the long save interval. More complex models and larger maximum source limits will obviously extend runtime.

Note, DNest4 does possess command line options, the most notable being the ability to specify the number of threads to be used. This was not explored in this project, but if the user's machine possesses multi-threading capabilities, this could significantly reduce runtime. See the C++ file `CommandLineOptions.cpp` in the `code` directory in DNest4's installation location for more details.

3.3 Post-Processing

Following DNest 4's analysis, one can begin processing and analysing the results. Given the large number of samples returned in a typical analysis, DNest4 comes with a number of standard post-processing scripts. These have been altered as necessary and combined into a single script, `Post-Processing.py`.

The first step involves several prompts for user input, giving the user a number of options for how the results will be processed. The first is an option to view the statistical output of DNest4's default post-processing function, a simple (y/n) sufficing (note, this post-processing will be carried out regardless, as it is a crucial step). This function identifies those samples with significant posterior weight (see the DNest4 documentation for more detail) as being the most important to analyse. This reduces the amount of data being analysed typically by a factor between 5 and 10 (the less uncertain the predictions, the greater this effective sample size and vice versa). This function has been adapted, in particular to allow the posterior probability of each sample to be calculated at a later stage. This particular function also generates a number of plots showing the statistical output of DNest4 (see [3] for details). While the user should examine these the first time post-processing is carried out following a given analysis, simply to ensure the output resembles that shown in [3], these may not be of particular interest on further runs.

The user will also be asked to enter the number of sources they believe to be in the image (not the maximum number that may be in the image, but the number the user is confident are in the image, which should be smaller). If the user enters 3, say, they will then be asked to input the x and y coordinates of each of the three sources. This is to allow the user to compare the DNest4 predicted results with their initial guess. To skip this step, simply enter 0.

The remaining options relate to the calculation of the flux-weighted integral under the PSF of each source and the generation of pixellated images showing what the parameters returned in each sample would produce. Both of these are very long procedures, so the user may not wish to carry them out on a first run of `Post-Processing.py`, when only an initial idea of the results may be required. One could of course re-run DNest4 with a smaller save limit if desired, which would dramatically reduce the time taken. More crowded images are also likely to take much longer for the integral calculations in particular.

Note that as with `Pre-Processing.py`, the user can easily disable these user input prompts by commenting out lines 288-383 and either removing the `if-else` statements depending on the options chosen, or declaring the necessary flags to the desired options. The latter is simpler and would require less alteration of the code. The following variables would need to be declared in this case:

- `view_stats`: Either 'y' or 'n', depending on whether or not the user wishes to view the statistical output of DNest4.
- `Ns`: The number of sources the user believed to be in the image prior to running Dnest4.
- `est_x`, `est_y`: If `Ns` is greater than zero, these lists contain the x and y coordinates of the sources.
- `integral_flag`: Either 'y' or 'n', depending on whether or not the user wishes to calculate the integral under the PSF of each source.
- `picture_flag`: Either 'y' or 'n', depending on whether or not the user wishes to generate images corresponding to each sample.

Note that, as with `Pre-Processing.py`, the user must enter filenames and paths within the code (lines 282-285), given the likely amount of variation between users. These variables are less likely to change between runs than those for the pre-processing script, but care should still be taken.

Some basic processing is carried out at this point, with the necessary data read in, posterior probabilities calculated and all samples sorted to produce more readable data files than those produced by DNest4. In calculating the posterior probabilities, a number of steps must be taken, to obtain all of the required numerical values required. Given that we are using continuous uniform distributions for the prior distributions, in order to obtain a probability we need to "discretize" the interval. To do this we ask, "what is the probability that this parameter is within ± 0.0001 of a specific value?" for each parameter. Thus, the overall prior probability for a point in parameter space is the multiplication of 4 uniform distributions and 4 powers of 0.0002 (as the interval we are looking in is $0.0001 + 0.0001$ wide). These uniform distributions will not be the same for every sample. As mentioned in Section 3.1, DNest4 possesses the capability to generate wider distributions for the parameters if necessary. Since the predicted values of the parameters in each sample are drawn from these distributions, it is the limits of these distributions we must use in calculating the prior probability.

Returning to this factor of 0.0002^4, it does not depend on any of the parameters, data or model, it is entirely user defined. Note that without this factor, the formula from Section 1 would simply return the posterior probability *distribution.* In order to obtain posterior probabilities from this, we would have to "discretize" the interval. Thus, multiplying the prior probability distribution by 0.0002^4 and using the formula as usual results in the posterior probability that the correct parameters are within ± 0.0001 of the predicted value. Finally, the posterior probabilities are normalized to sum to 1.

Following calculation of the posteriors, if the user chose so, the code will numerically calculate the flux weighted integral under the PSF of each source in each sample (integrated over the range of the whole image, plus 20%, to account for sources bleeding in at the edges, using the Moffat 25 profile from pre-processing). This gives the total flux from a single source. As noted above, the reason this is optional is that it takes many hours to compute. Some ways of speeding it up may be to change the values `dx` and `dy` to larger values (default is 0.1, which is already quite large), or to only calculate the integral for a handful of the most likely cases. This is left up to the user.

Next, the code writes two data files to the `Output` file (recommended that a new output file be created for each new analysis): `Output_Sample_Data.txt` and `Output_Sample_MetaData.txt`. The first of these two files simply contains all of the data on each sample, with multiple lines for each sample containing more than one source. The format of each line is shown in Figure 3. Note, that if the user runs a first analysis which calculates the integral under the PSF, but chooses to exclude it in further analyses for the sake of saving time, save a copy of the first output file as otherwise it will be overwritten.

```
Sample_ID Number_of_Sources X Y Flux Alpha Flux_Integral Posterior
```

Figure 3: Format of each line in `Output_Sample_Data.txt`. For samples with N sources, they will be printed over N lines, with the sample ID, number of sources and posterior remaining the same, and only source specific parameters changing. Note, that if the user chooses not to calculate the integral under each source's PSF, this entry will simply be printed as zero.

The metadata file produced contains information such as the effective sample size and the percentage of samples containing a given number of sources. A number of plots are then produced including a histogram of the number of sources predicted in each sample, as well as the positions, flux values and α values for all samples with a given number of sources. Some examples are shown in Figures 4-7.

Finally, if so chosen, the code will generate noise-free images of each sample, as well as the standardized residuals (sample image + mean of the noise subtracted from the actual data, divided by the standard deviation of the noise). Like the integral calculation, this step takes a significant length of time to complete. However, short of simply restricting it to a specific subset of samples, there appear to be few means of speeding this step up. Note, these are not FITS file images, simply .PNGs.

At this point, the core BSI software has been executed. In Section 4, some suggestions of further steps to be taken in analysing images are outlined.

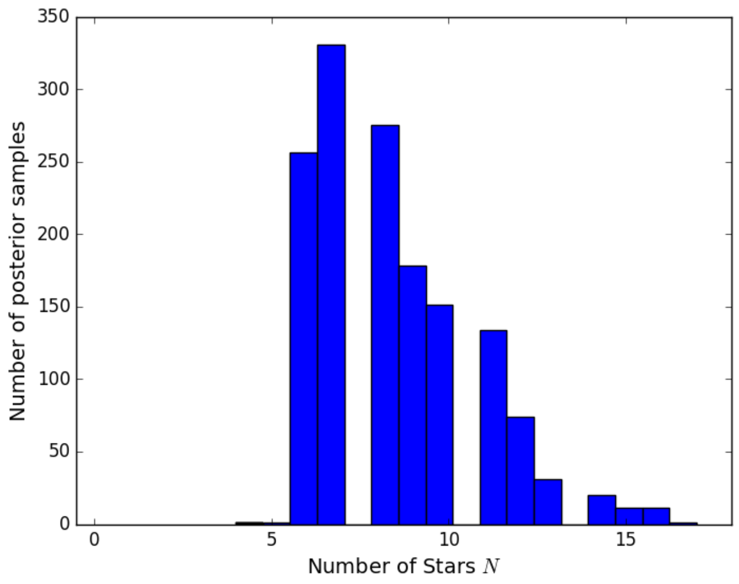

Figure 4: Histogram returned by analysis of samples produced by DNest4. Note that even though the histogram may show a given number of sources as the most predicted value, the posterior probability for a different number of sources may actually be higher.

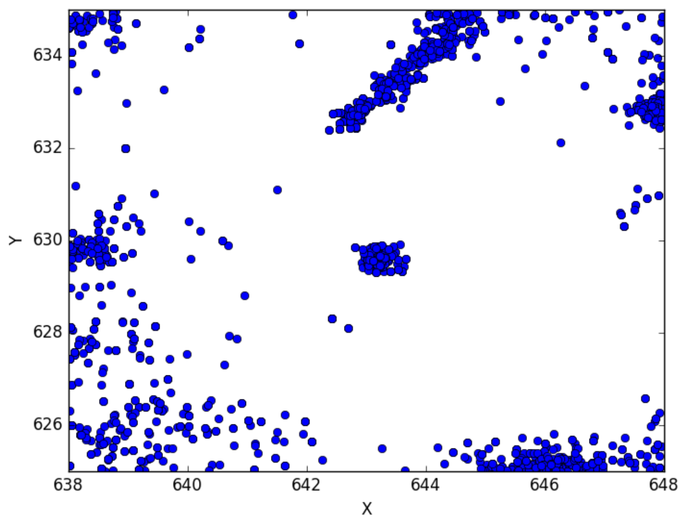

Figure 5: All predicted positions from all samples with 8 sources, returned by the same analysis as in Figure 4. The central source, which was the object of interest for this particular analysis appears to have been predicted with a reasonable degree of certainty, whereas other sources have a much greater spread in their predicted positions. Note as well the oblong cluster of points above the central source, which correspond to an extended source in the image, likely a binary pair.

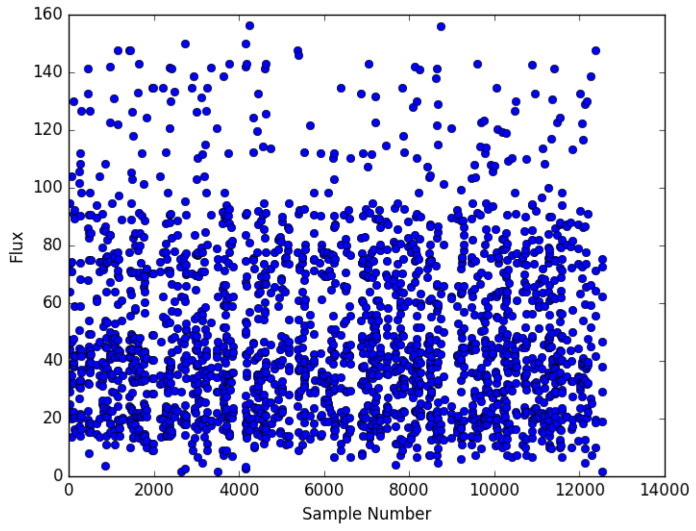

Figure 6: All fluxes corresponding to the same samples as Figure 5. Note that since each sample predicts 8 sources, for each sample number on the x-axis of this plot, there are 8 points plotted. It is very difficult to obtain more information from this plot, as too many of the predicted sources have similar fluxes. This indicates a more specific analysis is required (see Section 4). Note also that even though the maximum flux passed to DNest4 was approx. 55, DNest4 could still predict brighter sources, as discussed in Section 3.1.

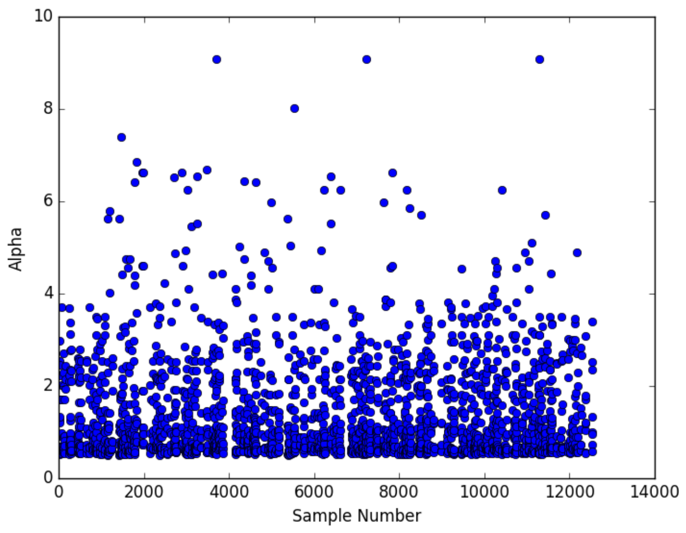

Figure 7: α parameters for all samples depicted in Figures 5 and 6. Note there is far less spread on these than in the predicted fluxes. This is likely due to the fact that if all the sources were point sources, they should possess the same α parameter. The presence of the extended source mentioned previously results in greater values of alpha being predicted however.

4 Further Analysis

As mentioned in Section 3.3, the post-processing carried out in the provided script is kept relatively simple. This is primarily intended to keep options open to users who are interested in different aspects of the analysis (eg. investigating if a source is in a specific position, sure a source is in a certain place but not sure of flux, etc.), but also due to the nature of the data output by DNest4. One natural extension of the analysis carried out would be to work out average positions of sources over multiple samples, such as when a large number of predictions are clustered in a particular area (see Figure 5). However, DNest4 saves parameters of sources in certain "slots" in its output. A source predicted in basically the same position in two samples may not be in the same slot in each sample. Thus, working out an average position becomes much more complicated. One possibility might be to take all predicted sources with positions within a certain box in the image. However, in noisier images, this box may be so big that it captures completely different sources. Thus, carrying out such an analysis is at the user's discretion. A sample script, `Analysis.py`, which carries out this step is included in the code, simply examining the average position, flux and α parameter for a specific source from a HST image (image not included). The output plots from this script are shown below. Further steps could be to only consider sources coming from samples with posterior probabilities greater than a certain value, etc.

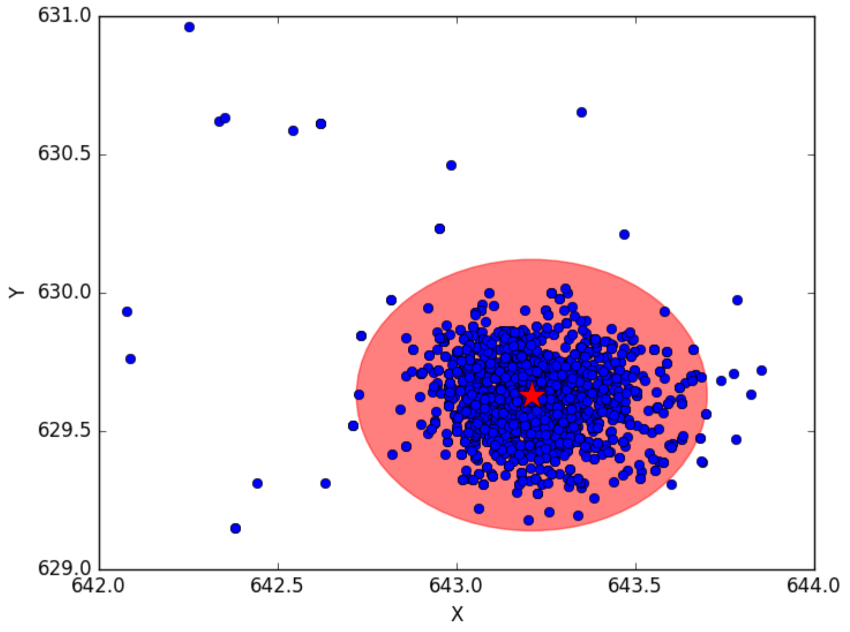

Figure 8: These are the predicted positions of all sources within a user-defined box across *all* samples. The red star indicates the average position, while the red circle is the circle of radius 3σ, where σ is the standard deviation of these positions.

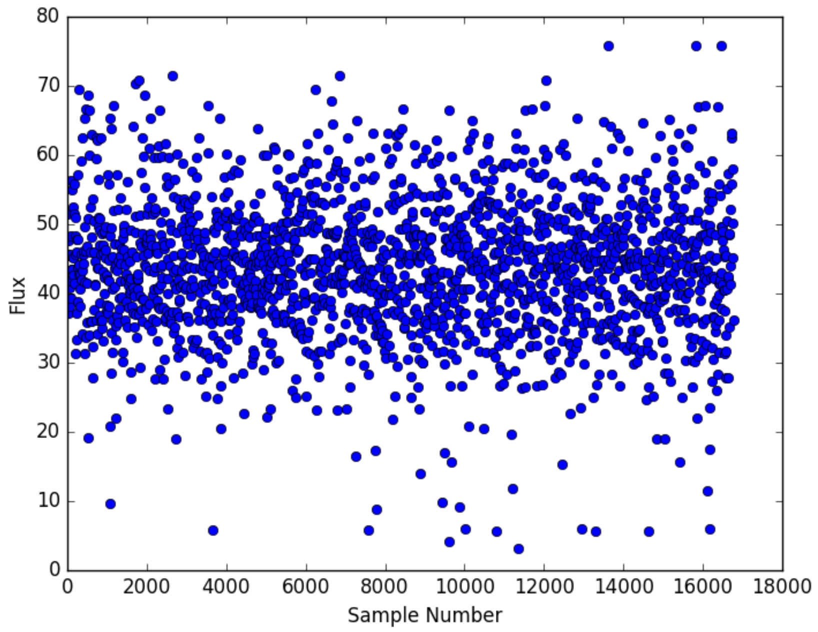

Figure 9: Fluxes of all sources within the same box as Figure 8. The average flux is calculated as 43.32778, with standard deviation 10.6849.

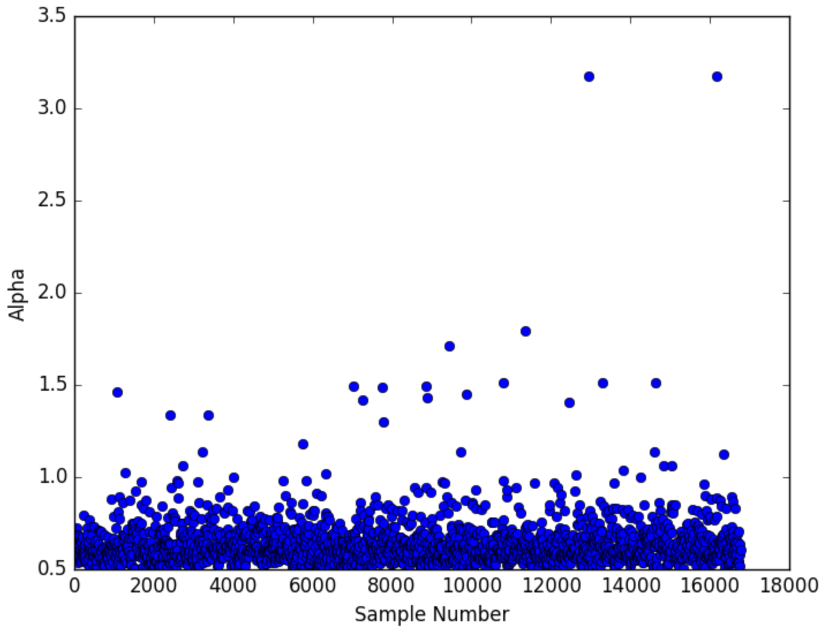

Figure 10: α parameters of all sources within the same box as Figures 8 and 9. The average value of α is 0.64497 with standard deviation 0.14461.

Appendix A: Instructions

What follows is a "recipe" for running the software on a FITS file image. It assumes the user wishes to use the code and options for DNest4 in their default state, with only minor changes such as file names, etc. See Section 3 for more details on these steps, and how users can change the code and options if desired. Those instructions in red are "first run" instructions, that do not need to be completed every time an analysis is carried out.

1. Install DNest4 (see [3] for instructions).
2. Download software from: `github.com/otoolec2/BSI`
3. Alter `DNEST4_PATH` variable if necessary in the Makefile in /BSI/Photometry.
4. Open directory /BSI.
5. Copy subject image FITS file into /Images directory.
6. Obtain relevant PSF from `tinytim`.
7. Copy PSF into /PSFs directory.
8. Alter `image_filename` and `psf_filename` on lines 62 and 63 of `Pre_Processing.py`.
9. Open console and enter command `cd .../BSI` followed by Enter (insert the path to BSI in place of "...").
10. Input command `python Pre-Processing.py` in the console, followed by Enter (alter if your python distro. is installed in a seperate directory and not part of the standard `PATH` variable.
11. Follow the prompts printed to the console (or alter code as described in Section 3.1 if desired).
12. Input command `cd Photometry` followed by Enter.
13. Input command `make` followed by Enter. (Only necessary on first run, unless user changes C++ code)
14. Execute main.exe (`./main.exe` in cygwin, but may differ for Mac or Linux users).
15. When DNest4 has completed execution, or the user has cancelled it, input `cd ../` followed by Enter.
16. Input command `python Post_Processing.py` followed by enter.
17. Follow the prompts printed to the console (or alter code as described in Section 3.3 if desired).
18. Rename `Output` directory and create new directory called `Output` containing an empty folder called `Frames` to prevent further analyses overwriting the output of a previous run.

Acknowledgements

The work described in this document was completed with support from the University College Dublin Seed Funding Scheme 2017. I would also like to thank my supervisor Dr. Morgan Fraser for all of his support throughout the project.

References

[1] github.com/eggplantbren/DNest4.
DNest4 repository.

[2] http://tinytim.stsci.edu/cgi-bin/tinytimweb.cgi. tinytim website.

[3] D. Foreman-Mackey B. Brewer. DNest4: Diffusive Nested Sampling in C++ and Python. arxiv.org/abs/1606.03757, 2017.

[4] D. W. Hogg B. Brewer, D. Foreman-Mackey. Probabilistic Catalogs for Crowded Stellar Fields. *Astronomical Journal*, 146:7, 2013.

[5] B. Brewer. Inference for Trans-dimensional Bayesian Models with Diffusive Nested Sampling. arxiv.org/abs/1411.3921, 2014.