Stream Detector Software Manual

March 31, 2018

1 Description of the problem

A set of points (x, y) in a square window contains a uniform background upon which an overdensity is superimposed that has a linear shape; we refer to this overdensity as a *stream* and the points themselves as *stars*. We wish to estimate (1) the probability that the set of stars (x, y) contains a stream, and (2) the geometrical parameters of the stream as well as its relative amplitude.

2 Significance in astronomy

The Milky Way galaxy is surrounded by a large number (dozens known) of dwarf galaxies. Due to dynamical friction (Chandra), such galaxies occasionally fall into the Milky Way; this causes them to be transformed from a nearly spherical shape to an elongated shape, eventually developing very long tails of stars in a phenomenon called tidal disruption. The tails of the disrupted dwarf galaxies contain stars streaming from that galaxy's core out toward the Milky Way's halo, and they can be observed as lines running across the night sky. These are called *tidal streams*. Their observation, however, is not a trivial matter. Since dwarf galaxies contain only very few stars to begin with (compared to the Milky Way), the number of stars in tidal streams is very small compared to the background. Thus, tidal streams are too faint certainly to observe with the naked eye, but telescopic data often requires careful analysis to reveal their presence. A census of tidal streams is important for the understanding of the history of the formation of the Milk Way galaxy, and by extension the assembly of galaxies in general.

Data features other that the stars' positions on the sky can help to distinguish between stars belonging to tidal streams and background stars. For example, in a small section of the sky, stars belonging to one stream should be roughly equidistant from the observer on the Earth, compared to the background stars which are scattered in distance. Measuring the distance is not trivial, it could be done for individual stars through the parallax method (which required several observational epochs) or in a statistical way. Stars from a single stream should also be moving in space as a group, and if kinematic data can be obtained, this could help distinguish them from the background.

Here however we focus on the case where only the position on the sky (i.e. the right ascension and declination) is known for each star. An observation is divided into square pieces in an arbitrary way, each such square (henceforth, a window) represents a possible small segment of a stream. The coordinates are normalized such that they range from -0.5 to $+0.5$ in both axes.

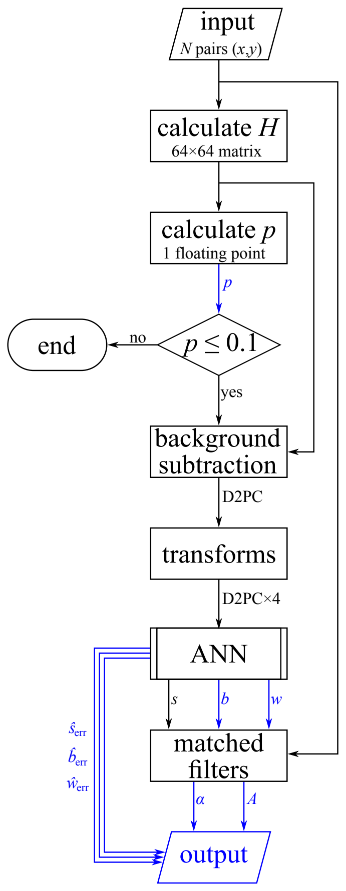

Figure 1: Flowchart of the parameter estimation of a single sample. Blue lines and text represent variables that are part of the output.

3 Algorithm

3.1 Preparatory steps

The algorithm requires large quantities of mock data to be generated that could be discarded after the preparatory steps are completed. There is no need to perform these steps as we already performed them and the data needed are already found in the data folder of the software. These steps should be repeated as needed if the program is modified or repurposed.

The mock data generation has three distinct phases: (1) generating "blanks" (i.e. samples containing background only) for the purpose of estimating the mean of counts in each element of the pair count matrix and the standard deviations, (2) generating blanks for the purpose of estimating the distribution of the noise, and (3) generating samples with streams for the purpose of training an artificial neural network (ANN). The idea behind the first two phases is explained in Section XXX, and the idea behind the third phase is explained in Section XXX.

3.1.1 Background generation

The program `GenerateBackground.py` generates realizations of a uniform distribution in the observing window for a particular number of stars N. For each sample, the pair count matrix H_{bg} is generated, but neither the (x,y) data nor H are actually saved for each realization. Instead, an ASCII file is created that has a name of the form `bg-N-n.dat` (where `N` is the number of stars, and `n` is a serial number for the file). It contains a header with the number of realizations followed by a table of size $64^2 \times 2$ with the first column representing H_{bg}, the second representing S_{bg}, and each row representing an element in the flattened matrices. The file is updated continuously, so no harm is done if the program stops. The size of each file is 184 kilobyte (binary). An example of the usage is

```
python GenerateBackground.py 1000 50000
```

This generates 50000 realizations with $N = 1000$. It is recommended to generate a very large number of samples so that H_{bg} and S_{bg} are sufficiently smooth, we used 4×10^5 samples. As for the value of N, at least two values should be used, one representing the minimum and the other the maximum number of stars acceptable for the program. It is better to use logarithmically sampled values between the minimum and the maximum; we used $N = 500, 1000, \ldots, 8000$.

The program first tries to write to the file `bg-N-0.dat` in the data folder, if it already exists then the serial number is increased until a new file can be created, so it is easy to parallelize the process by calling multiple instances of the program. After enough samples have been accumulated for the desired values of N, the files generated by multiple instances should be consolidated by invoking the command

```
python CombineBackground.py
```

This causes all `bg-N-n.dat` files in the data folder to be consolidated to files with the names `bg-N.dat`. The original files (with `-n`) should be removed manually. The files `bg-N.dat` may not be discarded at any stage.

3.1.2 Noise distribution

The H_{bg} and S_{bg} matrices calculated above will now be used on a new set of blank samples to generate the noise parameter δ as explained in Section XX by invoking a command of the form

```
python NoiseStatistics1.py 1000 50000
```

where like before, the first parameter is N and the second is the number of samples to be generated. The number N should match each of the values used in the previous step; the number of samples should still be quite large, we used 10^5. The files generated are named `noisesamples-N-n.dat`, each contains a simple ASCII table with the first column a serial number of the sample and the second number is the δ value. As new rows are appended at the end of the file, the program could be stopped at any time; the size of these files could reach a few megabytes depending on the number of samples. As before, files from the different instances of the program are consolidated by the command

```
python CombineNoise.py
```

The next sub-step is to calculate the cumulative distribution of δ for each value of N, this is done by invoking

```
python NoiseStatistics2.py
```

This program reads all the `noisesamples-N.dat` files in the data folder, calculates the cumulative distribution and records data that can later be used to interpolate and extrapolate it; these data are written to the file `noise-statistics.dat`. After this program is executed, all files with the names `noisesamples-*.dat` may be discarded.

3.1.3 Sample generation

In the previous two steps, "blank" samples containing only background stars were generated and discarded after the relevant information was extracted. Now we generate samples containing streams as explained in Section XX, the pair count matrix and the stream parameters are saved for each sample, which means that the files can grow quite large; the (x,y) coordinates are not saved. The data is stored in the Hierarchical Data Format version 5 (HDF5) with files by the name `streams-n.h5` (where `n` is again a serial number for parallel execution). Sample generation is performed by the command of the form

```
python SampleGenerator.py 4096
```

where the only commandline argument is the number of samples to be generated by this instance; the minimum and maximum values of N_{bg} are read from the configuration file `streams.conf`, and the maximum number of $N = N_{\text{bg}} + N_{\text{fg}}$ is deduced from the previous steps. The HDF5 files could easily be corrupted if the program writing the file exits without closing the file (i.e. crashes). We attempt to reduce the risk of that by minimizing the amount of time the file is open: we buffer 32 samples at the time, and only update the file when the buffer is full, closing it when the update is complete. It is still not recommended to kill the program once it is running. Notice that the files are much larger in this case; we generated 65536 samples and the total size of the data was 2.2 gigabyte. Unlike before, there is no need to consolidate all the files generated by different instances of the program.

3.1.4 Neural network training

The last step before the preparation is complete is to train the artificial neural network (ANN) with the dataset generated in the previous step. This is done with the command

```
python ModelTraining.py
```

There are no arguments, the program finds the stream data and trains the ANN, saving the network architecture as well as weights and biases to the file `model.h5`, also in the HDF5 format. The python file contains hardcoded network architecture as well as other hyperparameters. After the training is complete, the files `stream-*.h5` may be deleted.

3.2 Processing a sample

A flowchart of the code is shown in Fig. XXX. The input is a set of N points (x,y). The first task is to calculate the pair count matrix H, which is has 64×64 elements. The next task is to calculate the p-value as described in Section XXX. If the p-value is larger than 0.1 (i.e. there is more than 10% chance that the sample contains only background), then the code returns that the sample could not be processed. If $p\leq0.1$, the code proceeds to calculating the D2PC ξ by subtracting the background from H and dividing by it. Then, three additional transforms of ξ are generated by rearranging its elements as explained in Section XXX, which leaves us with four samples that could be fed into the (already trained) ANN. The ANN returns four values for each of the three geometrical parameters s, b, and w, which are averaged and their standard deviation is recorded as a proxy for the error. Then the geometrical parameters are fed into the match filtering system, which uses the (x,y) to determine α as well as estimates the relative amplitude A.

3.2.1 Testing

4 Stream parameters

The window is a simple 2-dimensional square; we denote the coordinates as x and y, each of them ranges from -0.5 to $+0.5$. The stream is the intersection of the square with a rectangle, which could be described by three geometrical parameters: α, b, and w. The angle α is formed between the long axis of the stream (i.e. the rectangle) with the x-axis; b is the distance of the central line of the stream from the origin; w is the width of the stream (i.e. the short axis of the rectangle). Two additional parameters are the number of background stars N_{bg} (in the whole window), and the relative amplitude A, which is the excess of stars in the stream area divided by the number of stars expected from a uniform distribution in that area (this could also be expressed by N_{fg}, the number of stream stars).

The allowed range of each of the parameter is a subtle point. For the training and testing of the software, we took N_{bg} to range between 500 and 4000 and A to range between 0 and 1. This often results in samples that do not contain any distinguishable stream at all, but these are rejected anyway as explained in Section [[XX]]. The logic in choosing very low values of N to begin with is that we want to train the software to detect streams which are very difficult to spot with the human eye; if A it significantly bigger than unity, it could be relatively easily identified by other means.

The geometrical parameters are constrained in a way that makes sure there is one-to-one correspondence between a sample and the geometrical parameters. Ambiguities can occur if the outer edge (outer with respect to the origin) of the rectangle (stream) is outside the window. This could be prevented by rejecting any sample generated where all four vertices of the window are on the same side of that edge. Another limit we imposed was that the surface area of the stream should be less than half of the total; this prevents situations where nearly the whole window is covered by stream stars. This is not a critical limitation, the ANN could learn to deal with those situations as well given that the samples are distinguishable enough from a uniform distribution.

return np.abs(np.sum(np.sign(-np.sin(alpha)*X + np.cos(alpha)*Y - (b+w/2)))) != 4.0

[[maybe put a b/w graph to show the allowed values?]]

5 The directional 2-point correlation

The concept of the correlation function is very useful in cosmology. The (regular) 2-point correlation function is a function of some distance r, defined as the probability that two random galaxies in the universe are be separated by that distance r. It is often defined with respect to a uniform distribution, so to represent overdensity or clumping in different length scales, in the following way. For a particular survey of galaxies, a mock catalog of galaxies is produced given the survey's known observational biases. The number count of galaxies in the mock catalog separated by a distance between $r-\frac{1}{2}\Delta r$ and $r+\frac{1}{2}\Delta r$ (for $\Delta r\ll r$) is counted and denoted $H_{\text{bg}}(r)$. The the same counting is performed for the real galaxies in the survey which yields some other function $H(r)$. Finally the 2-point correlation function may be defined as

$$\xi=\frac{H-H_{\text{bg}}}{H_{\text{bg}}}. \tag{1}$$

For the purpose of cosmological surveys, galaxies are point particles, with the relevant distances between them far larger than their physical sizes. The same applies for surveys of stars in the Milky Way galaxy, but the 2-point correlation function of individual stars in the Galaxy does not carry a similar physical significance as that of galaxies in the universe. Here we present an extension of the idea called directional 2-point correlation (D2PC), which take into account clumping in both distance and direction. The logic is that when a human looks for a tidal stream in a 2-dimensional coordinate data, they will for clumping along certain angles of the observational window. Thus, it is reasonable that when the data is presented in this form, it will be easier for an NN algorithm to interprert... find stream or whatevs. Then also for NN it is best to

We call H the pair count matrix

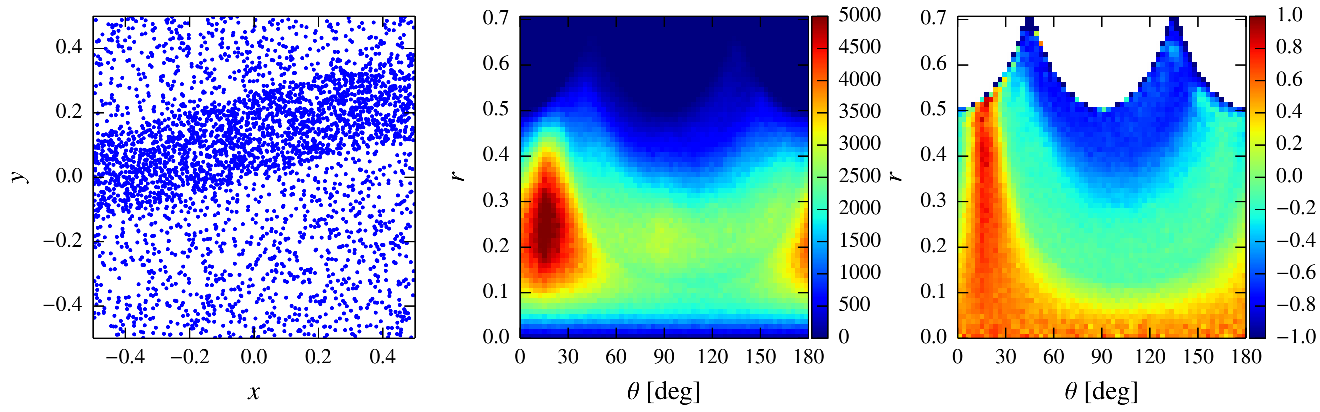

Figure 2: An example of a stream. The left panel shows the observational window with the points representing stars, the middle panel shows the pair count matrix H and the right column shows the directional 2-point correlation function ξ; both matrices have 64×64 elements. This sample contains 2000 background particles, the stream has geometrical parameters of $\alpha = 15^{\circ}$, $b = 0.125$, and $w = 0.25$. The relative amplitude $A = 3$ is much larger than the typical sample that was used for the training and testing of the artificial neural network (there, A ranged between 0 and 1) but given here so that the stream and the features of ξ are easily visible by eye. The number of foreground stars is 1541 and the total $N = 3541$, which gives $N(N-1)/2 \approx 6.3 \times 10^6$ pairs. The white regions in ξ indicated combinations of θ and r which cannot exist in a square window.

6 Background and noise analysis

[[[maybe clarify that the backgound is not the noise, it is part of the signal. the noise is the statistical fluctuation in the position]]]While the ultimate goal is estimating the geometrical parameters of a stream, an important step is understanding whether a stream is present at all, or whether its amplitude is large enough to be mathematically distinguishable from the background. This is important because the artificial neural network will always give a prediction as an output (in our case, for the geometrical parameters) even if the signal is too weak to deduce anything reliably. Thus, we want an independent measure on the strength of the signal. Qualitatively, we would like to estimate the p-value of each sample, that is, the probability that it contains only background. Our approach is to measure the deviation of the pair count matrix H of a particular sample with N stars from an idealized background using the following quantity:

$$\delta = \mathrm{rms}\left[\frac{H - H_{\mathrm{bg}}(N)}{S_{\mathrm{bg}}(N)}\right] \tag{2}$$

where $H_{\mathrm{bg}}(N)$ is the expectation value of the pair count matrix for N stars; $S_{\mathrm{bg}}(N)$ is a matrix where each element is the standard deviation of the corresponding element in H_{bg}. The subtraction and division are element-wise operation, and rms is the root mean square of all the matrix elements. The quantity δ should be close to unity if the sample contains only uniformly distributed stars; a strong deviation from unity indicates the presence of non-uniformity in the sample (which is not necessarily a stream, but it could be). The exact distribution of δ around unity is not trivial and has to be calculated numerically. Once it is known, the δ-value calculated for a particular sample can be converted to a p-value as described below.

Another difficulty in this method is to calculate the standard deviation matrix S_{bg} in the first place. The distribution of count values in every (r, θ) bin is not trivial and its first two moments (i.e. H_{bg} and S_{bg}) also have to be estimated from a mock catalog of samples containing nothing but background (uniformly distributes stars). While the dependence of H_{bg} on N is trivial, the dependence of S_{bg} on N is not. We generate background mock catalogs for five geometrically spaced values of N between 500 and 8000, and found that each element in S_{bg} can be fit very well by a power-law of N, with the indices between **????** and **????**. The software has two modes of operation: it can calculate $S_{\mathrm{bg}}(N)$ for an arbitrary value of N from the power-law fits on the background mock catalog, or to interpolate it between the N values in the catalog. We found that the latter to be slightly more reliable.

Finally, to convert the δ-value of a specific sample to a p-value we had to generate another set of samples containing only background (for the same discrete values of N_{bg}). For this new set, only δ is recorded for each sample until a relatively smooth distribution is built. The function $p(\delta)$ is the complementary cumulative distribution of δ. We construct it numerically by interpolation from $p(\delta) = 0.5$ (the median) and $p(\delta) \approx 1.24 \times 10^{-2}$ (the value corresponding to 2.5 standard deviations if it were a normal distribution).

For the right tail of the distribution, we use an extrapolation. The fit assumes that for large values of δ the distribution is a log-normal one, and the fit parameters are obtained between $1.24 \times 10^{-2} < p(\delta) < 4.55 \times 10^{-2}$ (between 2 and 2.5 "sigma"). Here again, a non-trivial dependence on N occurs and we use interpolation.

We note that the quality of this estimation is of no critical importance; once the p-value of a sample is lower than some threshold, it can be assumed that a stream exists; we are not very interested in the marginal cases. Other methods could be used to estimate the p-value, e.g. similarly to the methods described here but using the pair count in one dimension only, or directly from the raw (x, y) data. Machine learning techniques could also be used to classify samples based on weather they are likely to contain a detectable stream or be pure background.

7 The angle

The angle α ranges between $-180°$ and $+180°$, with each value giving a unique stream geometry. A degeneracy is introduced when calculating the D2PC because the vector displacement between two point in any pair forms an angle with the x-axis of either θ or $180° - \theta$, depending on which of the two points is chosen to be the head of the vector [this should be explained in Sec XX]. Thus, the information encoded in the D2PC is of the angle $\alpha' \equiv \alpha \mod 180°$. A difficulty arises when attempting to estimate the angle α' directly in the neural network by minimizing the loss $|\alpha'_{\text{pred}} - \alpha'_{\text{true}}|$ [[use the word reward]]. Take for example a stream with $\alpha' = 1°$; its D2PC is in some sense closer to a stream with $\alpha' = 179°$ than it is to a stream with $\alpha' = 10°$. If the NN predicts the former angle, the loss would be $178°$ but if it predicts the latter, the loss would be only $9°$. This creates a situation where at the edges of the α' range the network is very confused and constantly gives bad predictions. A solution could be to estimate a function of α' that wraps around at the edges, such as its sine. This creates another degeneracy because $s \equiv \sin(\alpha') = \sin(180° - \alpha')$. To summarize, the four possibilities are

$$\alpha = \begin{cases} \arcsin s \\ 180° - \arcsin s \\ -\arcsin s \\ \arcsin s - 180° \end{cases} \tag{3}$$

For example, an estimate of $s_{\text{pred}} = 0.5$ would corresponds to $\alpha_{\text{pred}} = \pm 30°$ and $\pm 150°$. It is impossible to remove the first ambiguity discussed (between $\pm 30°$ and $\mp 150°$) using the D2PC data alone, but only by going back to the raw (x, y) data. The second ambiguity (between $\pm 30°$ and $\pm 150°$) could be remedied by adding another output neuron to the network that independently attempts to predict $\cos(\alpha')$ or just its sign. This is however an unnecessary complication of the NN, and we opted for a solution that removes both degeneracies at the same time as described in Section XX.

8 Transformation of the D2PC

we have α the standard gives us H, the $s = 0.26$

$\alpha + 90°$ is shfting, we call it H_{rot} we have $s = 0.97$ in other words $s_{\text{rot}} = \sqrt{1 - s^2}$

$180 - \alpha$ this gives us $\tilde{H}$ which is the mirror of H we have $s = 0.26$ again, so $\tilde{s} = s$

lastly we have $-90 - \alpha$ which gives us $\tilde{H}_{\text{rot}}$ we have $s = 0.97$ so $\tilde{s}_{rot} = s_{rot}$

if we had a circular window, we could do an "infinite" number of transformations

since the ANN has different activation neurons for those 4 transforms, and the training set contain only one picture, they are processed as 4 completely independent pictured. thus we get a better result by averaging them and also some estimate of the error

9 Artificial neural network

To recover the geometrical parameters, we designed an artificial neural network (ANN). The input is the D2PC ξ which is a 64×64 matrix, and the output is the three geometrical parameters s, b, and w. We first apply a mask to the input to remove the matrix elements where the pair count matrix for a square window cannot be larger than zero, and then the input is flattened. This leaves us with 3324 input neurons and 3 output neurons. Since the three output parameters are bound, they were normalized to the range $[0, 1]$ and we used a sigmoid activation in the output layer.

We tested multiple designs for the network, starting from very simple ones (starting from a design with no hidden layers at all) and adding complexity gradually. Our final design is a multilayer perceptron network with two hidden layers: the first with 1662 neurons and the second with 162 neurons. In total there are 5 796 045 trainable parameters in this design. The activation function for all hidden neurons was a rectified linear unit (relu), testing various other options including a leaky relu did not improve the results. Making the network any deeper also did not improve the result. Alternative designs that included 2-dimensional convolutionary layers performed at most comparably to our chosen design, suggesting that an accuracy limit has been reached. The network was trained with 65536 samples for 150 epochs. This was not terribly computationally expensive, and took approximately an hour on a machine with a single modern GPU. The loss function was defined as the mean of

$$\tfrac{1}{3}|s_{\text{pred}} - s_{\text{true}}| + \tfrac{\sqrt{2}}{3}|b_{\text{pred}} - b_{\text{true}}| + \tfrac{\sqrt{2}}{3}|w_{\text{pred}} - w_{\text{true}}|. \tag{4}$$

The $\sqrt{2}$ factor comes from the allowed ranges of b and w (which is $1/\sqrt{2}$ for both); it makes the network treat the three parameters roughly equally.

The testing was performed with a different test of 65536 samples. To increase the accuracy of the network, and be able to estimate the error on each output, we put all four possible transformations (see Section XX) of each sample through the

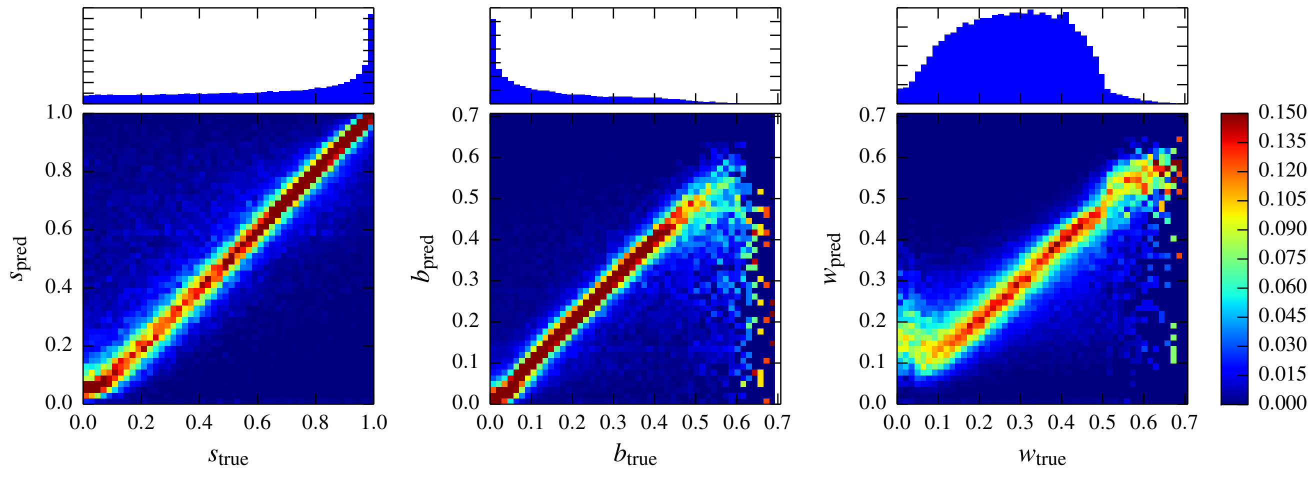

Figure 3: The predicted (recovered) values versus the true values of the three geometrical parameters which are the output of the artificial neural network shown as density maps. Since the parameters are not uniformly distributed (their histograms are shown in the top panels), to normalize the results, we divided each pixel in the maps by the number of (true) values in its column. Each predicted value is the average of four permutations of an input sample.

ANN. For each of the three output parameters, the average of the four predictions was recorded as the predicted value for the sample, and their standard deviation was recorded as the error. The accuracy of the network is shown in Figure XXX.

10 Matched filtering

The ANN described above can only estimate the parameter s, which as noted in Section XX, could relate to the angle α in four different ways. The two remaining tasks are to remove this 4-fold degeneracy, and to estimate the relative amplitude A. This is done by constructing four simple stream-shaped filters corresponding to the four possible values of α, and applying them to the raw (x,y) data. The process is then to count the stars in each filter N_{stream}, and the option with the larger star count wins. Then

$$\begin{aligned} N_{\text{bg}} &= \frac{N_{\text{tot}} - N_{\text{stream}}}{1-a} \\ N_{\text{fg}} &= N_{\text{stream}} - aN_{\text{bg}} \\ A &= \frac{N_{\text{fg}}}{aN_{\text{bg}}} \end{aligned}$$

where a is the relative surface area of the stream in the window (calculated from the predicted geometrical parameters).

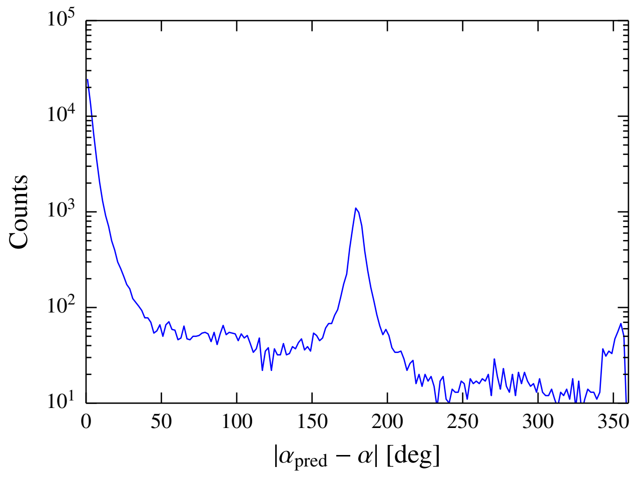

Figure 4: note that its logarithmic so actually first peak is ~23 time higher than the second and ~400 higher than the third

11 Prediction results

The results are shown in Fig. 3. The testing data set contained 65536 samples, and as noted in section XX, each predicted value represents an average of the four transforms. The average loss for this training set was 0.062 and the parameter

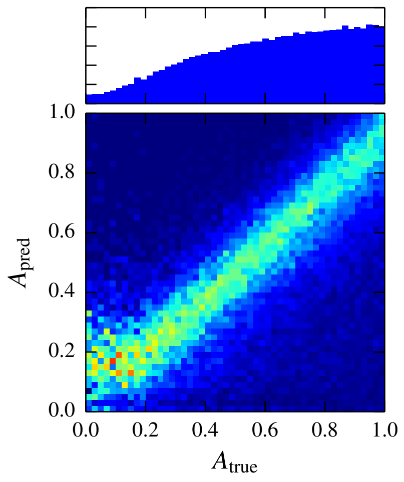

Figure 5: Same as figure XXX but for A, which is not estimated by the artificial neural network directly, but from its results in combination with the raw (x,y) data. colorbar is the same as that figure.

estimations have very little to no bias. The success of estimating the angle α is shown in Fig. XXX through the distribution of $|\alpha_{\rm pred} - \alpha|$. The strong peak around zero indicated that the procedure describes in Section XXX is largely successful in removing the degeneracy and recovering the correct value of α. The second peak, which is ~ 22 times smaller, indicates that the recovering of s was successful, but the wrong value of α has been chosen by the matched filters (e.g. $\alpha = 30^\circ$ was recovered as $\alpha_{\rm pred} = -150^\circ$). The last peak, which is ~ 400 times smaller than the first, indicates confusion between angles very close to 0° and 180°, with again s recovered successfully. The last parameter evaluated in our algorithm is the relative amplitude A, which is not directly measured by the ANN but by the matched filtering system. It is presented in Fig. XXX. The spread in this parameter is higher than the three geometrical parameters, with the average of $|A_{\rm pred} - A|$ being 0.125.

12 Error analysis

The error estimate for each parameter can also be plotted as a function of the real error; however it is more useful to invert the axes and use logarithmic scaling, as shown in Fig. XXX. Let us denote the actual error in s as $s_{\rm err} \equiv |s_{\rm pred} - s_{\rm true}|$ and the error estimate (the standard deviation of the predictions of the four transforms) as $\hat{s}_{\rm err}$ (and similarly for b and w). As we see in the figure, there is a relation with a large scatter between $\hat{s}_{\rm err}$ and $s_{\rm err}$ but it is not an equality relation, which means that the error estimate is biased. The best fitting straight lines in logarithmic scale correspond to the following relations, that could be used to correct for the bias in the error estimate:

$$s_{\rm err} = 0.6384\,\hat{s}_{\rm err}^{0.8664}$$
$$b_{\rm err} = 0.3241\,\hat{b}_{\rm err}^{0.7129}$$
$$w_{\rm err} = 0.0638\,\hat{w}_{\rm err}^{0.2000}$$

where the equality sign above is taken to mean that this a fitting formula, the actual values are scattered around this relation, which fixes the bias. We note however that the bias correction does not affect the order of magnitude of the error in the relevant range, and thus not very important for any particular sample. It may be helpful in the statistical sense.

It is interesting to note that there is a simple relation between the p-value of a sample (which is measured from the D2PC in an independent way of the ANN) and the average error, as shown in Fig. XXX.

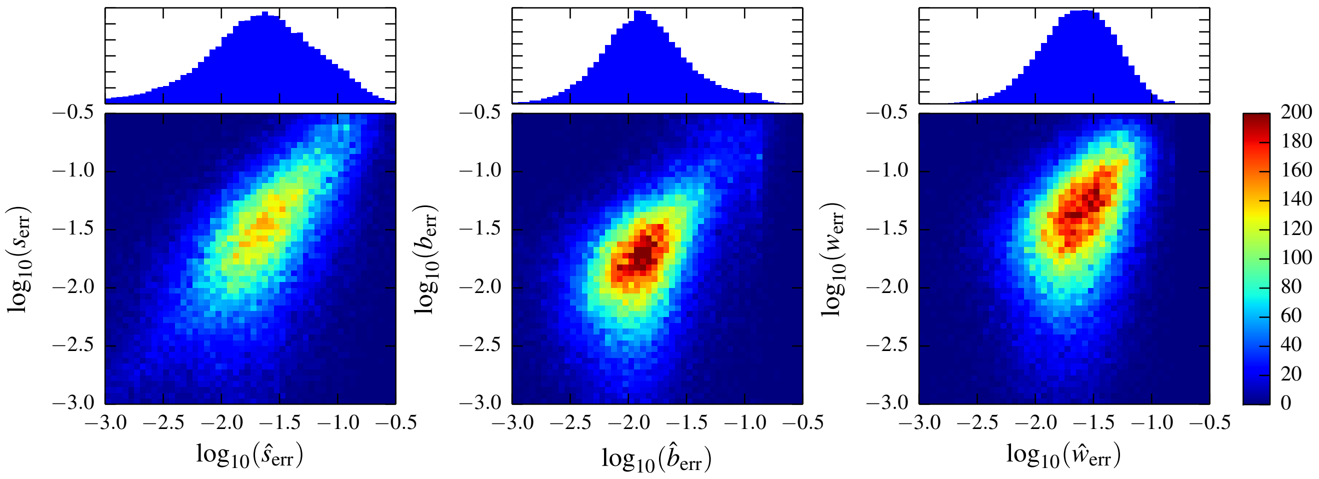

Figure 6: very good estimate, unlike fig 1, we don't divide by the whatever. log normal

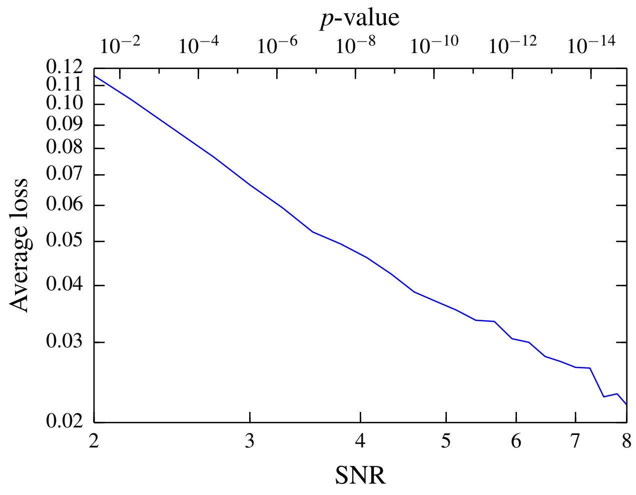

Figure 7: The average loss (defined in equation XXX) of the testing dataset in bins of signal-to-noise ratio (alternatively, p-value). As expected, the neural networks performs better the stronger the signal is, but note that while p drops by approximately 14 orders of magnitude, the average error only drops by a factor of ~ 5.