Positional Update and Matching Algorithm (PUMA)

Jack Line
University of Melbourne

1 Introduction

Creating a reliable catalogue with which to calibrate low frequency radio data is a necessary and complex task. Ideally, this catalogue would span multiple frequencies, providing a reliable spectral index for each source. It would also be free from any ionospheric positional offsets; it could then be used to correct for ionospheric refraction in future observations. As ionospheric offsets are expected to scale with $\sim \lambda^2$, higher frequency observations should have higher positional accuracies. It follows then that higher frequencies can be leveraged to obtain accurate positions. The difficultly comes in matching sources found from surveys observed with varying instruments and frequencies. Not only does each telescope have its own resolution and sensitivity, but the morphology of each source may change with frequency. Furthermore, each catalogue employs its own source finding algorithm, so comparison of any size of object reported is difficult. With this software, an approach is developed that utilises information of both source position and spectral information. Positions can be matched through probabilistic cross-identification, as described in (Budavári & Szalay 2008). Spectral information can be used as a second identification criteria, assuming a spectral model. The Positional Update and Matching Algorithm (PUMA) was developed to meet the needs set out here.. PUMA is an openly available code[1], which is free to use. The flow of the matching algorithm is shown in Figure 1. The following sections expand upon the methodology of each step, along with example scripts detailing how to run the software. §2 outlines the theory of Bayesian probabilistic cross-identification. §3 gives an overview of the initial positional match and calculation of positional matching probabilities. §4 explains the matching criteria implemented, giving the algorithms employed. §5 details the contents of the final matched catalogue.

2 Bayesian positional cross-matching

In a Bayesian analysis, a hypothesis H can be related to its complementary hypothesis K through the Bayes Factor

$$B(H, K|D) = \frac{P(H|D)/P(H)}{P(K|D)/P(K)}, \qquad (1)$$

[1]PUMA is stored in a repository here: https://github.com/JLBLine/PUMA

where D is some measurement set. For this application, H is the situation where the sources from multiple catalogues describe the same source, and K is where they do not. When matching n catalogues, it can be shown (see (Budavári & Szalay 2008) for further details) that the Bayes Factor is given by:

$$B = 2^{n-1} \frac{\prod w_i}{\sum w_i} \exp\left(- \frac{\sum_{i<j} w_i w_j \psi_{ij}^2}{2 \sum w_i} \right), \tag{2}$$

where ψ_{ij}^2 is the angular separation between sources in the i^{th} and j^{th} catalogues, and w_i is the weighting for the i^{th} catalogue. This is given by $w = 1/\sigma^2$, where σ^2 is the astrometric precision. This is calculated as

$$\sigma^2 = \sigma_{\mathrm{RA}}^2 + \sigma_{\mathrm{Dec}}^2. \tag{3}$$

These errors are usually quoted directly in each source catalogue. The Bayes Factor can be related to the posterior probability (in the limit of vanishing priors) through

$$P(H|D) = \frac{BP(H)}{1 + BP(H)}. \tag{4}$$

For multiple catalogues, the prior may be calculated through

$$P(H) = \frac{\nu_\star}{\prod \nu_i}, \tag{5}$$

where the scaled full sky number of sources in each catalogue ν is give by $\nu_i = 4\pi N_i/\Omega_i$, with N_i the number of sources in the catalogue and Ω_i the catalogue survey area. $\nu_\star$ is the scaled full sky number of sources in the final matched catalogue.

3 Initial Match and Posterior Probability Calculation

3.1 Initial Positional Match

Each catalogue is initially saved as a VOTable[2]. Any catalogues that are available from the VizieR database service[3] can be downloaded as such. For other catalogues, they must be converted manually. This is relatively simple using either the software TOPCAT[4] or with a python script.

Each catalogue is cross matched with a designated base catalogue. The final matched catalogue will have a similar source density to the base catalogue, so the user should select the base catalogue that suits their needs. The initial source match is carried out using the script `cross_match.py`. This script performs two functions. Firstly, relevant information from each catalogue is formatted in a standard way. The user must specify the column names and units of the catalogue, which are then converted and saved in to a 'simple' VOTable. The information taken from each catalogue is: Source Name; RA J2000(deg); Error on RA(deg); Dec J2000(deg); Error on Dec(deg); Flux(Jy);

[2]VOTable documentation - http://www.ivoa.net/documents/latest/VOT.html
[3]VizieR database service - http://vizier.u-strasbg.fr/viz-bin/VizieR
[4]TOPCAT documentation http://www.star.bris.ac.uk/~mbt/topcat/

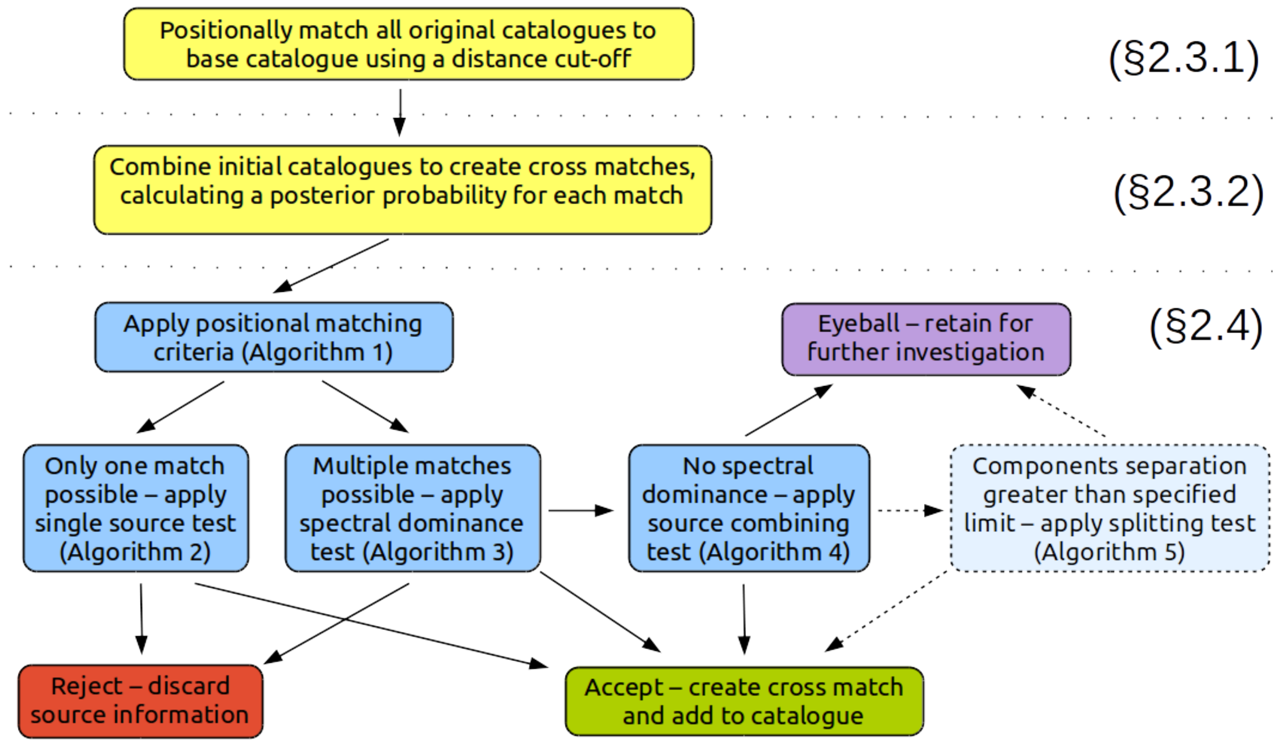
Positionally match all original catalogues to base catalogue using a distance cut-off
(§2.3.1)
Combine initial catalogues to create cross matches, calculating a posterior probability for each match
(§2.3.2)
Apply positional matching criteria (Algorithm 1)
Eyeball – retain for further investigation
(§2.4)
Only one match possible – apply single source test (Algorithm 2)
Multiple matches possible – apply spectral dominance test (Algorithm 3)
No spectral dominance – apply source combining test (Algorithm 4)
Components separation greater than specified limit – apply splitting test (Algorithm 5)
Reject – discard source information
Accept – create cross match and add to catalogue

Figure 1: The steps and outcomes of the matching algorithm are shown. Yellow boxes represent step with no criteria applied, cyan represent criteria being applied and all other colours represent final points. Each cyan box refers to a specific Algorithm, as detailed in Algorithms 1-5. The section labels on the right refer to §3.1, 3.2, and 4, which detail each step. Each section is performed by a separate programme.

Error on Flux(Jy); Major Axis(deg); Minor Axis(deg); Position Angle(deg); Flags; Field IDs. The programme will simply save a column with no information for a catalogue that does not contain one of the aforementioned columns. The latter columns have been included for possible future work. The Flags column is used to store any interesting information, such as morphology of a source. The Field ID stores the original image field that a source was found in during the survey that produced the catalogue. These standard tables are then used to perform a positional cross-match within a given cut-off distance using the STILTS[5] package. STILTS is the command line version of the cross matcher used in TOPCAT. `cross_match.py` uses the information given by the user to create a matched catalogue, where every possible match of the specified catalogue to a source in the base catalogue is saved. Each catalogue is matched to the base catalogue individually, so any number of combinations of catalogues can be used by `calculate_bayes.py` (see Figure 4). `calculate_bayes.py` also needs the source density of the catalogues to calculate Eq. 5. For each catalogue, the user specifies an area on the sky, bound by lines of RA and Dec. `cross_match.py` then counts the number of sources within this lune. This data is stored in the meta-data of both the individual 'simple' tables, and the final matched table in a standard way. This allows `calculate_bayes.py` to automatically read the data. The area of RA and Dec to measure the source density within is left to the user's discretion; an example is shown in Figure 2. An example of a cross match and details of `cross_match.py` are given in Figure 3.

[5]STILTS documentation - http://www.star.bris.ac.uk/~mbt/stilts/

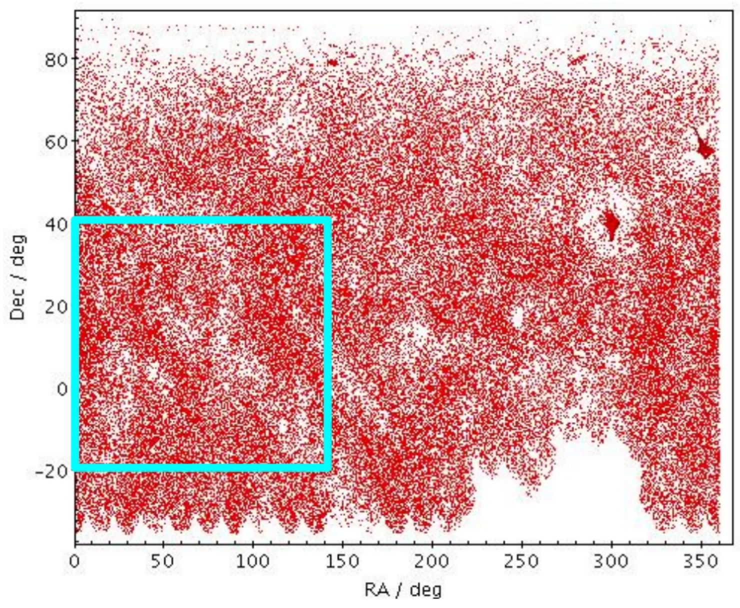

Figure 2: TOPCAT has been used to plot the positions of all sources in the VLSSr (Lane et al. 2012) catalogue. To calculate the source density of the catalogue, `cross_match.py` takes given RA and Dec bounds, and counts the number of sources within that area. In this example, the limits are represented by the cyan lines. It is left to the user to pick an area that will give a representative source density of the entire catalogue. For example, if too small an area, or a particularly over-dense area such as that at RA=300^0, Dec=40^0 is selected, an unrepresentative source density will be calculated.

3.2 Bayesian Match Calculation

The theory outlined in §2 is implemented in `calculate_bayes.py`. This programme uses any number of the 'matched' VOTables created by `cross_match.py`, combines them, and then calculates a posterior probability for every possible combination of sources. It creates an output text file that contains each group of matched sources, each possible combination of the matched sources, and a posterior probability for each combination. A group of possible matched sources simply means sources from all of the matched catalogues that are within a certain distance (which was specified by the user) of a source in the base catalogue. `calculate_bayes.py` makes these groups simply by grouping all of the matches from the 'matched' input VOTables that have the same source name from the base catalogue. Once the groups of sources have been found, each combination that includes one source from each catalogue is created.

To calculate the posterior probability (Eq. 4) for each combination, B (Eq. 2) and $P(H)$ (Eq. 5) must be calculated. For the weights in Eq. 2, the quoted positional errors from each catalogue are used as shown in Eq. 3. The rest of the calculation of B is straight forward. To calculate $P(H)$, the number of sources scaled to a full sky coverage for each catalogue (ν_i), as well as that of the final matched catalogue ($\nu_\star$) must be known. These values have already been calculated by `cross_match.py`, and are read in automatically. Once a combination of potential matches has been formed, `calculate_bayes.py` identifies the present catalogues, and uses the applicable ν_i to calculate $P(H)$. $\nu_\star$ is assumed to be the source density of the base catalogue. An example bash script to run `calculate_bayes.py` is shown in Figure 4.

```
#!/bin/sh
python cross_match.py --make_table --separation=180 \
 --table1=mwacs_all_b3_140206.vot \
 --details1=Name,RAJ2000,e_RAJ2000,DEJ2000,e_DEJ2000,180,\
 S180,e_S180,PA,MajAxis,MinAxis,Fit,- \
 --units1=deg,arcmin,deg,arcmin,Jy,Jy,deg,arcmin,arcmin \
 --ra_lims1=0,100 --dec_lims1=-55,-20 \
 --prefix1=mwacs \
 --table2=culgoora.vot \
 --details2=Cul,_RAJ2000,RA_err,_DEJ2000,Dec_err,\
 80,S_80_,f80_err,PA,MajAxis,MinAxis,Refs,-,160,S_160_, \
 f160_err PA,major,minor,-,ID \
 --units2=deg,deg,deg,deg,Jy,Jy,deg,arcmin,arcmin \
 --ra_lims1=0,360 --dec_lims1=-40,-30 \
 --prefix2=culg
```

Figure 3: An example bash script to run `cross_match.py` is shown. In this example the catalogue `mwacs_all_b3_140206.vot` is used as a base, and `culgoora.vot` is matched to it. Running this script will create three outputs: `simple_mwacs.vot`, `simple_culg.vot` and `matched_mwacs_culg.vot`. For this script, any source in `culgoora.vot` within 3arcmins of a source in `mwacs_all_b3_140206.vot` is considered a match. This is specified by the `--separation=180` command (angular separation in units of arcsec). For each catalogue, the names of the columns and the frequency of the observations in MHz are entered through `--details`. Each column is separated by a comma. The order of columns goes as: `--details = Name, RA, RA_error, Dec, Dec_error, Frequency, Flux, Flux error, PA, Major_Axis, Minor_Axis, Flag, Field ID, Frequency2, Flux2, Flux_error2 ...` Any column that the catalogue does not contain is input as '`-`'. Some catalogues (such as culg in the example) contain observations at more than one frequency. These extra frequencies and fluxes are added at the end of the string. Any number of fluxes can be added. The units of each column are specified through the `--units` option. The order in which the units are reported are as follows: `--units = RA, RA_error, Dec, Dec_error, Flux, Flux error, PA, Major_Axis, Minor_Axis`. It is assumed that all fluxes are reported in the same units if more than one flux is present. The supported units are `deg,arcsec,arcmin,min,sec` for angle and `Jy,mJy` for flux. During this process, the source density of each catalogue is estimated, for later use in the programme `calculate_bayes.py`. The area over which to count sources is specified by `--ra_lims` and `--dec_lims` for each catalogue. For each limit, the lower bound is entered first, then the upper bound following a comma, in units of degrees. If the RA bounds cross over the RA=0 line, the numerically larger RA value is entered first. Both catalogues are given a prefix string using `--prefix`. This is used to name columns in the 'simple' tables, as well as the output tables. These prefixes are also used by `calculate_bayes.py`. Finally, the `--make_table` option instructs `cross_match.py` to create the 'simple' tables. If the option isn't passed, the programme will simply search for the appropriate 'simple' tables.

```
#!/bin/sh
python calculate_bayes.py --primary_cat=mwacs --primary_freq=180 \
 --matched_cats=culg,nvss --matched_freqs=80~160,1400 \
 --out_name=bayes_mw-c-n.txt
```

Figure 4: An example bash script to run `calculate_bayes.py` is shown. In this example, posterior probabilities are calculated for all possible combinations when matching Culgoora (Slee 1995) (prefix `culg`) and NVSS (Condon et al. 1998) (prefix `nvss`) to the base catalogue MWACS (Hurley-Walker et al. 2014) (prefix `mwacs`). This script requires `cross_match.py` to be run twice, to have created the two files `matched_mwacs_culg.vot` and `matched_mwacs_nvss.vot`. These files should be in the directory that `calculate_bayes.py` is run in. The user specifies the prefixes used during the creation of the 'matched' catalogues with the commands `--primary_cat` for the base catalogue and `--matched_cats` for the matched catalogues. The same is done for the frequencies of the catalogues through `--primary_freq` and `--matched_freq`. Each name and frequency is separated by a comma. If a catalogue has more than one frequency (such as `culg`), all frequencies are separated by a '∼'. In this example, `culg` has measurements at 80 and 160MHz, so 80∼160 is input. Groups of matched sources, along with all possible combinations of those sources and associated posterior probabilities, are output to a text file. This is named with the command `--out_name`.

4 Matching Criteria

The information generated by `calculate_bayes.py` is used by `make_table.py` to create a final matched catalogue, by applying the steps shown in the lower section of Figure 1. In general, it uses two criteria to test for a match: positional and spectral. These are described in more detail in § 4.1-4.4. To apply a spectral criteria, a model must be assumed. The choice is to either assume a simple power law (find refs), or adopt more complicated models. These bring computational expense, not only through the fitting, but through checking that the fits are good. Further to this, often there are only two or three data points; this makes it difficult to confidently choose one model over the other. Efforts have therefore been made to accept a match purely by positional information wherever possible. For completeness however, all sources entered in to the catalogue created by `make_table.py` are given a 'goodness of fit' label, which describes how well they fit a power law. Details of the output of `make_table.py` are given in § 5.

4.1 Positional Criteria

As described in § 2, $P(H|D)$ indicates how likely it is that the information from each catalogue is describing the same source, based purely on the position, positional accuracy and source density of each catalogue. This does, however, neglect the resolution of each instrument. It may be the case that a catalogue with lower resolution is indeed describing the same source, but averaging over many sources, thus measuring a combined flux and position. Comparing these two sources will give a very low probability that they are exactly the same source. This is accounted for in Algorithms 1 and 2. Algorithm 1 deals with groups that have multiple sources matched from the same catalogue, whereas Algorithm 2 deals with just one source from each catalogue. Both will accept combinations that have a probability above a certain threshold, or that are within the resolution of the base catalogue. Two thresholds are used; an upper threshold, P_u, and a lower one, P_l. These are used to distinguish likely and unlikely combinations, and can be set by the user. Algorithm 1 creates a list of all positionally possible combinations, which are then used in conjunction with Algorithms 2, 3 and 4. Algorithm 2 either accepts a single combination positionally or through a spectral test.

Algorithm 1: Positional selection criteria for multiple source matches. If there are multiple cross match combinations, one or more matched catalogues have more than one source matched to the base catalogue. These are labelled as 'repeated' catalogues. The algorithm accepts a combination if it is either likely, or if the repeated source is within the resolution of the base catalogue. The retained combinations are then investigated through Algorithms 3 and 4. P_u can be modified by the user. At all stages, statistics of the matching process are gathered to propagate through to the final matched catalogue.

Data: A group of matched sources and all possible cross match combinations with associated positional probabilities.
Result: Either: Return a list of positionally accepted combinations; reject all combinations.

```
Record the number of combinations ;
Work out which catalogues are repeated ;
Create retained_combinations list ;
Calculate the distance of the repeated sources to the base catalogue ;
for each combination $i$:
    if $P(H|D_i) > P_u$:
        accept combination; append to retained_combinations
    elif source from repeated catalogue is within resolution of base catalogue source + error:
        accept combination; append to retained_combinations
    else:
        reject combination
Record the number of retained combinations ;
if number of retained combinations == 0:
    reject group; label as rejected positionally   '''There is a possible extension here to
    look if one particular source from the other catalogues is causing the low
    matching probabilities'''
else:
    return retained_combinations
```

4.2 Spectral Dominance Test

If more than one match is reported as possible by Algorithm 1, the combinations are tested spectrally to see if one combination clearly dominates the others. The spectral energy distribtions (SEDs) of the sources are expected to follow a simple power law at radio frequencies (find eg ref). The spectral data $\boldsymbol{f}$ at frequencies $\boldsymbol{\nu}$ are tested by fitting a liner model $\boldsymbol{F} = \alpha \ln(\boldsymbol{\nu}) + \beta$ to a log of the data using weighted least squares. This is done using the python package `statsmodels`.[6] The residuals of these fits are then investigated to ascertain the deviation of the data from a linear

[6]statmodels documentation - `http://statsmodels.sourceforge.net/devel/index.html`

Algorithm 2: Positional selection criteria for a single source cross match. If there is only one combination possible, and it has a positional probability over a given threshold, it is accepted without scrutinizing the spectral data. This avoids assuming any spectral model. If the match is below P_u , all matched sources are checked to be within the resolution of the base catalogue. As there was only one possible match, a high positional probability was expected, so a spectral test is applied. If the residuals ϵ, χ^2_{red} of a fit to a power law (as detailed in §4.2) are below a certain threshold ϵ^2_u, $\chi^2_{red,u}$, the source is accepted. At all stages, statistics of the matching process are gathered to propagate through to the final matched catalogue. The traditional test of $\chi^2_{red} <= 2$ is used as a label for a 'good' or 'bad' fit.

Data: A single match combination with associated positional probability and spectral information.

Result: Either: Accept the combination and enter in to the matched catalogue; reject combination

```
Fit a power law to the spectral data ;
Calculate the residuals of fit ;
if $P(H|D) > P_u$:
    label as accepted positionally ;
    if $\chi^2_{red} <= 2$:
        accept combination i; label as good fit
    else:
        accept combination i; label as bad fit
else:
    Calculate the distance of all matched sources to the base catalogue ;
    if all sources within resolution of base catalogue source + error:
        if $\epsilon^2 > \epsilon^2_u$ and $\chi^2_{red} > \chi^2_{red,u}$:
            reject combination; label as rejected spectrally
        else:
            if $\chi^2_{red} <= 2$:
                accept combination i; label as good fit and accepted spectrally
            else:
                accept combination i; label as bad fit and accepted spectrally
    else:
        reject combination; label as rejected positionally    '''There is a possible extension
        here to look if one particular source from the other catalogues is causing
        the low matching probabilities'''
```

fit. The goal of this spectral fitting is to weed out large deviations from linearity; it is designed to allow for some curvature of the data. Given that there are a limited number of low frequency radio catalogues to match, the number of data points are low. This rules out easily using models that account for curvature, as the number of parameters to fit quickly outnumbers the number of data points. As such, the spectral test detailed here has been designed to be simple and robust, and tunable by the user to meet any desired criteria as much as possible.

To test the 'goodness' of the fit, the reduced chi-squared value χ^2_{red} is usually inspected (e.g. (Hogg et al. 2010)). This value sums the residuals of the fit in units of the variance of the data, and scales for the complexity of the fitted model as

$$\chi^2_{red} = \frac{1}{K} \sum_{i=1}^{n} \left(\frac{\ln(f_i) - F_i}{\sigma_i} \right)^2 , \tag{6}$$

where n is the number of matched catalogues, σ_i is quoted error on flux , and $K = 1/(n-p)$, with p being the number of parameters set (p=2 for a linear fit). As a general rule, a result of $\chi^2_{red} <= 2$ is considered a good fit, as the model lies within twice the observed variance of the data. However, as explored in (Andrae et al. 2010), χ^2_{red} comes with its own uncertainty due to noise in the data. This uncertainty grows as the number of data points reduces. Further to this, χ^2_{red} is reliant on the errors on each observations being drawn from a Gaussian distribution. Given that each data set contains data points subject to differing error analyses, the extent to which this is true is difficult to ascertain. Practically, it has been found that χ^2_{red} works well at classifying the fit at low flux densities, where the quoted errors are comparable to the size of the residuals. Conversley, a data set that displays similar curvature at a much higher flux density yields a very large χ^2_{red} value, as the errors are small compared to the residuals of the fit. Using χ^2_{red} with some cut-off threshold then tends to reject the brightest sources, which when unresolved are the best sources to calibrate on (e.g. (Mitchell et al. 2008)). To include the brightest sources, a second residual metric, ϵ, is defined as

$$\epsilon = \frac{1}{n} \sum_{i=1}^{n} \left(\frac{|f_i - \exp(F_i)|}{f_i} \right) . \tag{7}$$

This residual is designed to test the deviation of a fit from the data in units of the observed value. Note that this residual is performed in the natural units of the flux density, to ensuring all flux values are positive and a correct scaling is found. By scaling ϵ by the number of data points, ϵ describes the total deviation of the data points from a linear fit as a fraction of the magnitude of the data points. This metric performs well at high flux densities, but poorly at low flux densities as the experimental error becomes a large fraction of the measured flux density. χ^2_{red} and ϵ are then complimetary metrics. Both are used with cut-off thresholds ϵ_u, $\chi^2_{red,u}$ in all spectral tests to accept a combination. These thresholds can be adjusted by the user to suit their needs. Algorithm 3 tests the combinations output Algorithm 1, by calculating χ^2_{red} and ϵ for each combination. It then compares these values, and if one combination has residuals that are 3 times smaller than all other combinations, it labels it as 'spectrally dominant'. If this combination is also far more positionally likely than all other combinations, it is accepted without further investigation of the spectral data. If there is no clear dominant combination, the group of sources is passed on to Algorithm 4.

4.3 Source Combination Test

If there is a group of combinations with no clear dominant combination, it is possible that catalogues with lower resolutions are describing a combined source that catalogues with higher resolutions are reporting as multiple sources. Algorithm 4 tests this by combining the flux of the matched sources from the repeated catalogues. It then tests the new spectral data by calculating both ϵ and χ^2_{red} of the new data. If the new combination has residuals under the residual thresholds, an updated position is found by combing the positions of the sources through a weighting scheme described by

$$w_i = \frac{f_i}{\sum_i^n f_i}, \tag{8}$$

Algorithm 3: A test for spectral dominance. If one combination has residuals that are at least 3 times smaller than all other combinations, and is positionally likely whilst all other combinations are unlikely, accept the source. Positional and spectral dominance are required at the same time, to rule out chance alignment of sources with particular fluxes. Otherwise, the combinations are passed on to Algorithm 4. At all stages, statistics of the matching process are gathered to propagate through to the final matched catalogue. The traditional test of $\chi^2_{red} <= 2$ is used as a label for a 'good' or 'bad' fit.

Data: A list of possible combinations output by Algorithm 1 (retained_combinations), with associated $P(H|D_i)$ and ϵ_i, χ^2_{red} for each combination.

Result: Either: Isolate a dominant source and accept; send for a combination check in Algorithm 4.

```
create string pos_dom = "none" ;
create string spec_dom = "none" ;
for each combination $i$:
    calculate residual ratio $\epsilon_i$ / $\epsilon_j$ where $i \neq j$ ;
    calculate residual ratio $\chi^2_{red,i}$ / $\chi^2_{red,j}$ where $i \neq j$ ;
    if all residual ratios are <= 0.33 for either $\epsilon$ or $\chi^2_{red}$:
        spec_dom = "combination i"    '''This means the residuals for this combination
        are at least 3 times smaller than all other combinations'''
    else:
        pass
    if $P(H|D_i) > P_u$ and all $P(H|D_j) < P_l$ where $i \neq j$:
        pos_dom = "combination i"    '''This means that this one combination is likely
        and all others are unlikely'''
    else:
        pass
if pos_dom == spec_dom:    '''This means combination i is positionally most likely
and spectrally dominant'''
    label combination i as spectrally accepted ;
    if $\chi^2_{red} <= 2$:
        accept combination i; label as good fit
    else:
        accept combination i; label as bad fit
else:
    pass retained_combinations on to Algorithm 4
```

where n is the number of repeated catalogue sources, and f is the flux of each source. These weights are applied to create the new RA position and error as

$$RA_{\text{new}} = \sum_i^n w_i RA_i \quad , \quad \sigma^2_{\text{new}} = \sum_i^n (w_i \sigma_i)^2; \tag{9}$$

this is similarly applied for Dec. If the combining test fails, the group source information is labelled to investigate further by eye. There is a point to note with this test. If only two catalogues are matched, there are only two data points to consider in a fit; a fit to two data points cannot have any residuals. Currently, for two data points, both χ^2_{red} and ϵ default to zero. This means if there is more than one source from one of the catalogues matched together, a spectral dominance test (Algorithm 3) will never find a difference between the residuals, and will always pass on to the combined source test (Algorithm 4). Combining the sources will still gives residuals of zero, so all of those sources will always pass the combination test, and be given an updated weighted position.

Algorithm 4: A test for source combining. If no one combination passes Algorithm 3, try combing the fluxes from the sources from the same catalogue. If the combined fluxes pass a spectral test, create a new position for the combined source, weighting the RA and Dec of each source by its flux. If splitting is implemented, pass to Algortihm 5. Otherwise accept the combined source. If the combination of fluxes does not pass, send the combinations to be investigated by eye. At all stages, statistics of the matching process are gathered to propagate through to the final matched catalogue.

Data: A list of possible combinations output by Algorithm 3 (retained_combinations), with associated spectral and positional information.

Result: Either: Accept a combination including an updated combined flux and weighted position; send to Alorithm 5 for splitting; send to be investigated by eye.

```
combine the fluxes of the source from the same catalogue ;
combine the flux errors ;
calculate $\epsilon^2$ with the new combined flux and error;
if $\epsilon^2 <= \epsilon^2_u$ or $\chi^2_{red} <= \chi^2_{red,u}$:
    if distance between repeated sources > $d_{split}$, and splitting implemented:
        send to Algorithm 5
    else:
        create a weighted RA, Dec and errors as described in Eq. 9 ;
        label as accepted by combination ;
        accept combination with updated position, flux and errors ;
        if $\chi^2_{red} <= 2$:
            label as good fit
        else:
            label as bad fit
else:
    send source information to be investigated by eye; label as retained after combining
```

4.4 Source splitting test

Combining sources as described in §4.3 confines the output catalogue to the resolution of the base catalogue. `make_table.py` also comes with an option to 'split' the combined sources. This is handled by Algorithm 5. If the sources from each repeated catalogue are separated by a distance geater than some user specified cut off, d_{split}, PUMA will attempt to split the combined source in to components. If there is more than one catalogue with repeated sources, currently PUMA requires that each catalogue has the same amount of components. This simplifies matching the components from the repeated catalogues. The repeated sources components are then matched by distance, to create new components sources. The flux density from each catalogue that had only one source matched is split in to components based on the following weighting scheme. For each catalogue m of n repeated sources, weights are created and averaged as

$$w_k = \frac{1}{m}\sum_{j}^{m} w_{j,i} \quad , \tag{10}$$

where $w_{j,i}$ is the w_i (Eq 8) weight in the j^{th} catalogue. The flux density of each single source is then split as

$$\mathbf{f_k} = f_s \mathbf{w_k} \tag{11}$$

where f_s is the flux density of the source, and $\mathbf{w_k}$ is a vector of length n of weights w_k. Applying the weights in this manner uses all the information from the repeated catalogues to create an accurate SED for each component.

Once an SED has been created for each component, they are spectrally tested as in Algorithm 3. If all components pass the spectral test, the sources are accepted. Each component is given a name based on the position of the original base source, with a letter appended to distinguish between components. If one or more components fail the spectral test, the source is flagged to investigate by eye.

Algorithm 5: A test for source splitting. If a source can be combined, but the components to be combined are separated by a distance larger than the user specified d_{split}, the combination is tested for splitting. If more than one catalogue has repeated sources, the Algorithm requires they have the same amount of sources. Each set of repeated sources are matched

Data: A list of source information output by Algorithm 4.
Result: Either: Accept split component sources; retain source information for investigation by eye

```
count number of repeated sources for each catalogue ;
if all repeated catalogues have the same number of sources:
    match the components of the repeated catalogues ;
    split the flux of the single catalogues as described in Eq. 10 ;
    spectrally test each new component as described in Algorithm 3 ;
    if all components pass spectral test:
        accept all components
    else:
        send source information to be investigated by eye; label as retained after splitting
else:
    return info to Algorithm 4 for combinational test
```

5 Final Matched Table

Once `make_table.py` has applied the algorithms described in §4, a list of accepted sources is left. These are output to a VOTable, the contents of which are described in Table 1. If the source was accepted either at the positional or spectral stage, the position reported is that of the most

```
#!/bin/sh
python make_table.py --matched_cats=mwacs,vlssr,mrc,sumss,nvss \
 --cat_freqs=180,74,408,843,1400 \
 --pref_cats=nvss,sumss,mwacs,mrc,vlssr \
 --input_bayes=bayes_mw-v-m-s-n.txt \
 --prob_thresh=0.8,0.95 \
 --epsilon_thresh=0.1 --chi_thresh=10 \
 --resolution=00:03:00 --split=00:01:30 \
 --output_name=puma_mw-v-m-s-n_split \
 --verbose
```

Figure 5: The order of catalogues matched with `calculate_bayes.py` is given by the `--matched_cats` command, with the frequency of those catalogues given in the same order by `--cat_freqs`. The order of positional preference of those catalogues is given by `--pref_cats`. The path and name of the text file output by `calculate_bayes.py` is given by `--input_bayes`. P_l and P_u are input through `--prob_thresh`, separated by a comma. ϵ_u and $\chi^2_{red,u}$ are given by `--epsilon_thresh` and `--chi_thresh` respectively. The resolution of the base catalogue is given in degrees as a string of degrees:arcmins:arcsecs, through the `--resolution` command (3arcmins in this example). The output name of the catalogue is given by `--output_name`. Two optional commands exist. Adding `--split` will instruct PUMA to attempt to split any combined sources in to components, if those components are separated by a distance greater than that specified (in degrees:arcmins:arcsecs). Adding `--verbose` will print out a comprehensive breakdown of the statistics of the matches.

preferred catalogue. The order of preference for the catalogues is given by the user (see Figure 5). If the source was made by combining sources, the new weighted position is used if the combined sources are from the preferred catalogue.

As described in §4.2, a model of

$$\boldsymbol{F} = \alpha \ln(\boldsymbol{\nu}) + \beta$$

is fit to each combination. When this fit is performed, the frequencies are entered in MHz and the flux densities in Jy. When performing the linear fit, the package `statsmodels` also outputs the standard error of the fitted parameters $(\sigma^2_\beta, \sigma^2_\alpha$. These fits and errors are reported in the final VOTable. They are used to extrapolate a flux density and error at the frequency of the base catalogue through the equations

$$f_{ext} = e^{\beta}\nu^{\alpha} \quad , \quad \sigma^2_f = \frac{1}{f^2_{ext}}\left(\sigma^2_\beta + \sigma^2_\alpha[\ln(\nu)]^2\right) \tag{12}$$

where again the frequency is in MHz.

By default, `make_table.py` also prints out a small summary of the fitting results, giving statistics on the types of matches being made. The summary of the print out for the script in Figure 5 is shown in Figure 6. All sources that were either rejected or retained to investigate are output to a text file named `*output_name*-`, ordered by the highest flux first. This allows the brightest sources to be evaluated and added to the catalogue by eye first.

```
------------------------------------------------------------------------
+++++++++++TOTAL NUMBER OF SOURCES: 13997 +++++++++++++++++++++++++++++++
------------------------------------------------------------------------
All sources accepted:  13709
    accepted by position:  9711
    accepted by spectral:  1715
    accepted by combine:  2283
All sources rejected:  26
    rejected by position:  13
    rejected by spectral:  13
    rejected by combine:  0
All sources retained to eyeball:  262
    retained by position:  0
    retained by spectral:  0
    retained by combine:  262
------------------------------------------------------------------------
```

Figure 6: The default print out for the script shown in Figure 5.

References

Andrae, R., Schulze-Hartung, T., & Melchior, P. (2010). *eprint arXiv:1012.3754*, p. 12

Budavári, T. & Szalay, A.S. (2008). *The Astrophysical Journal*, **679**(1): 301–309

Condon, J.J., Cotton, W.D., Greisen, E.W. et al. (1998). *The Astronomical Journal*, **115**(5): 1693–1716

Hogg, D.W., Bovy, J., & Lang, D. (2010). *eprint arXiv:1008.4686v1*

Hurley-Walker, N., Morgan, J., Wayth, R.B. et al. (2014). *Publications of the Astronomical Society of Australia*, **31**

Lane, W.M., Cotton, W.D., Helmboldt, J.F. et al. (2012). *Radio Science*, **47**(6)

Mitchell, D.a., Greenhill, L.J., Wayth, R.B. et al. (2008). *IEEE Journal of Selected Topics in Signal Processing*, **2**(5): 707–717

Slee, O. (1995). *Australian Journal of Physics*, **48**(1): 143–186

Table 1: Details of the content of the final matched catalogue output by `make_table.py`

Column Name	Description
`Name`	A name of the source based on the updated J2000 coordinates
`*base*_name`	The name of the source from the original base catalogue. Replace *base* with whatever prefix was chosen during creation
`updated_RA_J2000`, `updated_DEC_J2000`, `updated_RA_err`, `updated_DEC_err`	The new J2000 RA and Dec given to the source (deg), chosen from a preferred catalogue as selected by the user. In the case of a combination of sources, the position and errors are calculated through Eq 9
`original_RA_J2000`, `original_DEC_J2000`	The original base catalogue source position (deg)
`S_*freq*`, `e_S_*freq*`	The flux density and error reported by each catalogue (Jy) for a match, where *freq* is the frequency of the observation in MHz.
`SI`, `e_SI`, `Intercept`, `e_Intercept`	The spectral index (SI), intercept and associated errors to a weighted linear fit to the data as described in §5
`S_*freq*_ext`, `e_S_*freq*_ext`	The flux density (Jy) at the frequency of the base catalogue, as extrapolated by the parameters of the linear fit. These are calculated through Eq. 12
`Number_cats`	The number of catalogues matched, and hence the number of data points available to fit models to.
`Number_matches`, `Number_retained`	The number of possible combinations found by the original distance cut performed by `cross_match.py`, and the number retained after applying Algorithm 1.
`Type_matches`	The stage at which the match was accepted. Either position, spectral, combine, or split (if the user enables splitting).
`Low_resids`	Returns a 0 for a fit with $\chi^2_{red} <= 2$, otherwise returns a 1.