# Contents

1 Table of Contents
  
  1.1 About ................................................. 3
  1.2 Installing ............................................ 3
  1.3 Input data file ....................................... 5
  1.4 Running .............................................. 5
  1.5 Center determination ................................. 7
  1.6 Radial density profile ............................... 9
  1.7 Cluster radius estimation ............................ 11
  1.8 Integrated magnitudes ............................... 12
  1.9 Cluster probability .................................. 12
  1.10 Decontamination algorithm ......................... 13
  1.11 Best fit model ...................................... 15
  1.12 Performance ....................................... 22

2 Indices and tables ........................................... 25
This is the manual of operation for the Automated Stellar Cluster Analysis (ASteCA) package. If you encounter any problems please contact me or open a new issue.

**Warning:** This documentation is in the process of being written and is **not yet finished**. Updates will be made as versions are released. Last updated: Mar 16, 2018.

**Important:** A PDF version of this manual can be downloaded here.
1.1 About

ASteCA is an open source code developed entirely in Python, designed to fully automatize the usual tests applied to stellar clusters (SC) in order to determine their characteristics: center, radius, stars’ membership probabilities and associated intrinsic/extrinsic parameters: metallicity, age, extinction, distance, binarity, and total mass.

The code is designed modularly thus allowing the user to select which tests to run and which to skip, all managed through a simple input data file.

ASteCA is intended to run without the need for user intervention but both a semi-automatic and a manual mode are made available in case user input is necessary or required.

1.2 Installing

The code has been tested with the December 2016 release of Python:

- Python - 2.7.13

The packages listed below are required to run ASteCA.

- astropy
- sciPy
- matplotlib
- numpy (installed along with matplotlib)
- (optional) R, ks, and rpy2

**Important**: If you want to use the function that obtains the cluster probability of being a true cluster (see Sect. 9), the ks statistical package is needed. This package needs R and rpy2. These are not mandatory and ASteCA will still run without them, except of course the above mentioned function.
1.2.1 Working environment

We use the conda package and environment manager to install all the necessary dependencies to run ASteCA in an isolated Python environment. To install it:

1. Go to https://conda.io/miniconda.html and download the appropriate version for your system. I recommend using the Python 3.6 version and will assume in what follows that you are running a 64 bit Linux system.

2. Install with

   `$ bash Miniconda3-latest-Linux-x86_64.sh`

   Select yes when asked: *Do you wish the installer to prepend the Miniconda3 install location to PATH in your ~/path?*

3. Close and re-open your terminal window for the changes to take effect. Move inside the directory where you extracted the ASteCA package.

4. Create a virtual environment with

   `$ conda create --name asteca27 python=2.7.14 matplotlib=2.2.0 numpy=1.14.1 scipy=1.0.0 astropy=2.0.4`

5. Activate the environment

   `$ source activate asteca27`

   (for Windows users the command is `$ activate asteca27`)

   **Important:** You can tell that the environment is activated because its name is now shown in the terminal before the $ symbol as:

   `(asteca27) $`

   You need to activate this environment each time **before** attempting to run ASteCA, otherwise no installed packages will be detected.

6. Finally (optionally), you can install the dependencies needed for the ks package with:

   `$ conda install -c bioconda r-ks`
   `$ conda install rpy2`

   Currently the r-ks package is not available for Windows systems.

   If you are a Fedora user, you’ll probably need to install a few extra libraries for the rpy2 package to work. In the case of Fedora 26, I had to install the following libraries:

   `$ dnf install mesa-libGLU-9.0.0-11.fc26.x86_64`
   `$ dnf install ncurses-compat-libs-6.0-8.20170212.fc26.x86_64`

   but these can change for different Fedora versions. No extra libraries where needed in any of the Ubuntu-based systems I tested.

1.2.2 Download

The latest packaged release (zip or tarball) can be downloaded from:
After downloading, extract the compressed file wherever you want the code to exist. Alternatively the entire project can be cloned via git with (Linux command):

```
$ git clone https://github.com/asteca/ASteCA.git
```

which will create a sub-folder named /ASteCA.

### 1.2.3 Running

With the environment properly created (and activated) and the code downloaded, you can run ASteCA with:

```
(asteca27) $ python asteca.py
```

### 1.3 Input data file

The data file that contains the IDs identifying each star, the star’s coordinates and their full photometric data must follow certain rules.

1. Data must be arranged into columns using points as decimal separators.
2. All lines which are not part of the data columns must be commented out using a # symbol at the beginning.
3. ID strings must be unique, ie: no two stars should have the same ID.
4. All photometric values (either magnitudes or colors) outside the range $[-50., 50.]$ will be considered as bad photometry and any star showing such a value for any of its magnitudes/colors will be discarded.
5. All photometric columns must have an associated error column.
6. At least one magnitude column must be defined.

### 1.4 Running

To run the code simply position a cluster’s photometric data file in the input/ folder, modify the params_input.dat file to both inform the code of the distribution of columns in your photometric file and set the values for each function within the code. Photometric files can also be stored in input/ inside a sub-folder.

Every file inside the input/ folder (inside or outside a sub-folder) will be processed by ASteCA, with the exception of the membership probabilities files that end with a _memb.dat extension (see Sect. 10).

Once the file(s) are in place and the params_input.dat file correctly modified, the code can be executed with the command:

```
(asteca27) $ python asteca.py
```

This should work both in Linux based systems and OS X (Mac), but I’ve only tested it with Linux (since that’s what I use).

The CLUSTER.DAT file located in the input/ folder contains a synthetic open cluster generated via the MASS-CLEAN package with the following parameter values:
and serves as an example cluster to be analyzed with \texttt{ASteCA}.

### 1.4.1 Theoretical isochrones

\texttt{ASteCA} needs at least one set of theoretical isochrones stored in the \texttt{isochrones/} folder, to be able to apply the \textit{best-fit} function that estimates the clusters’ parameters. Currently, the only isochrone files supported are those obtained via the \texttt{CMD} service (Girardi et al. 2002), but any set in theory be used (with some changes made to the code). Please contact me if you wish to use a different set of theoretical isochrones.

The isochrones can be downloaded manually or the package \texttt{ezPadova-2} can be used to automatically fetch them from the site. I describe both possibilities below.

**Important:** The current version of the code only recognizes a handful of photometric systems from those available at CMD, but an upcoming version (v0.2.0) will extend \texttt{ASteCA} to support all systems present in the CMD service.

The systems supported are:

- 2MASS JHK\_s
- Washington CMT1T2
- SDSS ugriz
- HST/ACS WFC

**Manual download**

They isochrone files must follow a naming convention and be stored in a sub-folder inside the \texttt{isochrones/} folder, also named according to a convention that makes them readable and identifiable to the code.

The steps to manually download and store the files are:

1. The CMD isochrones files must be downloaded using the \textit{Sequence of isochrones of constant metallicity}, \texttt{Z} option. That is, isochrones of the same metallicity must all be stored in the same file.

2. Each file must have the name of the metallicity that characterizes it. For example, if you download a sequence of isochrones with metallicity \( z = 0.019 \), then the file should be called \texttt{0.019.dat}, \texttt{0.019.dat}, or \texttt{0.019.dat}, or \texttt{0_019.dat}, or \texttt{0_019000.dat}, etc.

This means that the file name can contain either a point or an underscore separating the decimal portion of the metallicity value. The number of zeros at the end of the value in the name does not matter. This is necessary because the code takes the metallicity value directly from the file name (which I plan to simplify sometime).

3. The theoretical isochrones files must be stored in a sub-folder of \texttt{isochrones/}, with the naming convention: \texttt{parsecXX_YYY} (if \texttt{PARSEC} isochrones are used). In this name, \texttt{XX} is 10, 11 or 12 depending on the version of \texttt{PARSEC} used (1.0, 1.1 or 1.2S, respectively) and ‘\texttt{YYYY}’ is \texttt{ubvi}, \texttt{wash} or \texttt{2MASS}, depending on the system chosen to generate the isochrones.
For example, if the PARSEC v1.2S tracks and the \textit{UBVRIJK} (cf. Maiz-Apellaniz 2006 + Bessell 1990) system is selected, the name of the sub-folder where the isochrones must be stored would be: \texttt{parsec12_ubvi}.

**Automatic download**

To avoid having to download each isochrone file by hand, I’ve written the ezPADOVA-2 code\(^1\) which can downloaded from:

https://github.com/Gabriel-p/ezpadova

This code takes care of downloading the isochrones for a given range of metallicities, storing them in files named following the above mentioned naming convention, and place them inside a folder also with the correct name.

This way, once this code finishes you can just cut the generated folder and paste it inside the \texttt{iscohrones/} folder in ASteCA.

1.5 Center determination

Determining the central coordinates of a SC with accuracy is of utmost importance, since this parameter directly affects the generation of the radial density profile, and thus also the estimation of the SC’s radius.

\texttt{ASTeCA} determines the center value in two steps: a coarse approximation is done first and a Gaussian kernel density estimation (KDE) is then applied to obtain precise final coordinates.

The process is independent of the system of coordinates used and can be equally applied to positional data stored in pixels or degrees.

1.5.1 Approximate center values

The positional data is first binned in a 2D histogram. The bin width is either automatically calculated as 1\% of the minimum range in either coordinate (x, y), or set manually via the \texttt{params_input.dat} file in the \texttt{Structure functions parameters} block, by setting the \texttt{bin_width} parameter (CH ID line).

\textbf{Important:} Unless there is a specific reason to use the \texttt{manual} mode to set the bin width, it is recommended to use the 1\% range \texttt{auto} value. This figure has proven to be a very good heuristic value in a large number of situations.

This 2D histogram is processed with a Gaussian filter to smooth it out, using five different bandwidths of increasing values (see \texttt{get_center.py} function for more details)

The maximum density values of each Gaussian filtered version of the 2D histogram, are presented in the final output image as seen in Fig. 1.1.

Each one of these values represents an approximation to the final center coordinates obtained in the next step (see section below).

If the \texttt{semi_input.dat} file is used to pass approximate center coordinates, this analysis is skipped and only the passed value is presented in the plot.

\(^1\) Fork of original ezpa\texttt{dova} code by Morgan Fouesneau (https://github.com/mfouesneau/ezpadova).
Fig. 1.1: Approximate center coordinate values. The standard deviation values in the bottom right corner are the bandwidth values used in the Gaussian filter. The box in the top left corner contains the averaged coordinate values and their standard deviations (shown in the plot as continuous and dashed lines respectively).
1.5.2 Precise center value

Once the approximate center coordinates have been obtained, ASteCA proceeds to refine this result. To this end, the code employs the approximate center coordinates obtained in the previous step using the smallest bandwidth value (St dev: 2.0 in Fig. 1.1), or $center_{bw-min}$.

These coordinates serve as initial values for the cluster’s center. The final precise values are determined as the maximum spatial density of a two-dimensional Gaussian kernel, generated on a reduced square area of the positional diagram of the cluster; see Fig. 1.2.

![Fig. 1.2: In yellow, area where the center coordinates are searched via the KDE. Blue lines are contour curves, the center found is marked with a cross.](image)

The square area where the final center coordinates are searched is centered on the $center_{bw-min}$ values, with a side length that spans 50% of the minimum x,y range. The bandwidth of the fitted kernel is automatically defined via Scott’s rule (Scott 1992).

In Fig. 1.3 we can see the final center coordinates (equivalently: the maximum value of the positional KDE) plotted over the Gaussian filtered 2D histogram obtained using the minimum bandwidth, as defined in Sect. 5.1.

After this analysis is completed, the final center coordinates are compared with the approximate values obtained in the previous section. Two flags are raised warning the user about large deviations that could require careful revision, and stored in the output file asteca_output.dat.

The flags must meet the following conditions in order to be raised:

- $f1$: The median value for the approximate center coordinates in either x or y, is more than 10% away from the final center coordinates.
- $f2$: The standard deviation for the approximate center coordinates in either x or y, is larger than 10% of the final center coordinates.

1.6 Radial density profile
Fig. 1.3: Cluster’s center found via the KDE analysis. The Bin value in the top right corner is the bin width of the 2D histogram mentioned in the previous section, used to obtain the approximate center coordinates.

**Todo:** Not finished.

The radial density profile (RDP) is obtained...
1.7 Cluster radius estimation

**Important:** This section is outdated.

**Todo:** Not finished.

We begin by counting the number of RDP points that fall below a given maximum tolerance interval above $d_{field}$. When the number of these points reaches a fixed value, $n_{fixed}$, the point closest to $d_{field}$ is stored as the most probable radius for this iteration. This process is repeated increasing the tolerance around $d_{field}$ which results in another probable radius being stored, not necessarily the same found in the previous iteration. The final estimation for $r_{cl}$ and its associated error comes from averaging the set of radius values stored this way and taking its standard deviation, respectively.
1.8 Integrated magnitudes

**Todo:** Not finished.

To obtain the integrated magnitude for the cluster region cleaned from field stars contamination, the following equations are used.

First, assume:

\[ I_{cl+fl} = I_{cl} + I_{fl} \]  (1.1)

then:

\[ V_{cl}^* - V_{cl+fl}^* = -2.5 \log(I_{cl}/I_{cl+fl}) \]

using Eq. (1.1) we have:

\[ V_{cl}^* - V_{cl+fl}^* = -2.5 \log(1 - I_{fl}/I_{cl+fl}) \]  (1.2)

and given:

\[ V_{fl}^* - V_{cl+fl}^* = -2.5 \log(I_{fl}/I_{cl+fl}) \Rightarrow I_{fl}/I_{cl+fl} = 10^{(V_{fl}^*-V_{cl+fl}^*)/-2.5} \]  (1.3)

we can combine now Eqs. (1.2) and (1.3) to obtain:

\[ V_{cl}^* = -2.5 \log(1 - 10^{(V_{fl}^*-V_{cl+fl}^*)/-2.5}) + V_{cl+fl}^* \]

1.9 Cluster probability

**Todo:** Not finished.

**ASteCA** evaluates the probability that a spatial overdensity is a true stellar cluster, rather than a random aggregate of field stars.

This is done via the `kde.test` function provided by the `ks` package, written for the **R** statistical software. The function applies an N-dimensional kernel density estimator (KDE) based algorithm, to assess the similarity between two N-dimensional arrangements of elements. In our case, the arrangements are CMDs, TCDs, or any arbitrary N-dimensional photometric diagram (PD), and the elements are obviously observed stars. The result of this comparison is quantified by a p-value.

The p-value of a hypothesis test is the probability, assuming the null hypothesis is true, of observing a result at least as extreme as the value of the test statistic.


A strict mathematical derivation of the method can be found in Duong et al. (2012).

The null hypothesis, \( H_0 \), is that both PDs were drawn from the same underlying distribution, where a lower p-value indicates a lower probability of \( H_0 \) being true. The function is used to compare the cluster region’s PD with the PD of every defined field region, and the PD of each field region with that of the remaining field regions. This results in two sets of p-values: one for the cluster vs field region analysis, and another for the field vs field region analysis.
The entire process is repeated a fixed number of times (100 by default). For each run a random shift is applied to the position of stars in the PDs before the calculus is made, to account for photometric errors.

The two sets of p-values are smoothed by a one-dimensional KDE resulting in the curves shown in Fig. 1.4. The blue and red curves represent the cluster vs field region and field vs field region PD analysis. For a real star cluster we expect the blue curve to show lower p-values than the red curve, meaning that the cluster region PD has a different arrangement of stars when compared with the PDs of surrounding field regions.

![Fig. 1.4: (Left) Function applied on a synthetic cluster, the curves are clearly separated with the blue one ($KDE_{cl}$, cluster vs field region PD analysis) showing much lower values than the red curve ($KDE_{fl}$, field vs field region PD analysis). The final probability value obtained is close to 1 (or 100%). (Right) same analysis performed on a field region where no cluster is present. The curves are now very similar resulting in a very low probability of the region containing a true star cluster.](image)

Both curves represent probability density functions, which means their total area is unity. Their domains are restricted between [0,1] with a small drift beyond these limits due to the 1D KDE processing. The total area that these two curves overlap (shown in gray in the figure) is thus a good estimate of their similarity. This means that the overlap area is proportional to the probability that the analyzed region holds a true cluster.

An overlap area of 1 means that the curves are exactly equal, pointing to a very low probability of the overdensity being a true cluster; the opposite is true for lower overlap values. We thus obtain the probability that the overdensity is a real cluster as:

$$P_{cl}^{KDE} = 1 - overlap$$

These values are shown in Fig. 1.4 in the upper left corner.

### 1.10 Decontamination algorithm

**Todo:** Not finished.
A correct determination of the most probable true cluster members, allows to recover a much clearer picture of a cluster’s photometric structure. It is also essential to ensure a more accurate and all around better estimation of the cluster’s fundamental parameters.

Since it is not possible to unequivocally discriminate between true members and field region stars from photometry alone (even when proper motions or radial velocities are available), the decontamination algorithms (DA, also membership probability assignment algorithms) in the literature usually give each star in the cluster region a probability of being a true member.

The process by which these membership probabilities (MP) are assigned by the code, is explained in detail in the article where ASteCA was introduced. The algorithm is based basically on a Bayesian scheme, and the repeated estimation of a ‘clean’ cluster region by means of a random removal of stars.

The first four binning schemes that can be used by the DA are taken from the astroML package; the rest are well known rules:

- blocks: Bayesian Blocks.
- knuth: Knuth’s rule.
- scott: Scott’s rule.
- freedman: Freedman-Diaconis rule.
- sturges: Sturge’s rule.
- sqrt:
- bb:

These same binning methods are made available to the Dolphin likelihood equation in the Best fit block, see Sect. 11.2.

Once the DA process has finished, every star inside the cluster region will have a MP associated to it. These MPs are stored, along with the data on each star in the cluster region, in a file named XXXX_memb.dat (where XXXX is the name of the analyzed cluster).

1.10.1 Reduced membership

After MPs have been assigned to all the stars within the cluster region, what follows is the selection criteria, referred here as the “Reduced membership” algorithm. This process determines which stars, among all those located within the cluster region, will be used by the best fit process (see Sect. 11).

Several options are available in ASteCA, associated to this function. These options are handled by the Reduced membership block in the params_input.dat file.

- local: combines the assigned MPs with a cell by cell removal of stars in the photometric diagram. The stars discarded are those with lower MPs and the number is equal to the excess of field stars present in each cell.
- n_memb: use the N stars with the highest MPs, where N is the approximate number of members found via the cluster region vs field region star density comparison.
- mp_05: selects only those stars with MP>=0.5, i.e.: stars with higher probability of being a cluster member than a field star.
- top_h: selects the top half of all stars in the cluster region with the highest MPs.
- man: the minimum probability value has to be set manually via the prob_min parameter (0, 1).
- mag: switches the prob_min value to a mag_min value so the user can select a minimum magnitude to be used in the best fit function, instead of a minimum probability (the “minimum magnitude” value is the value of the dimmest star wished to be taken into account in the fitting process).
• *skip:* use all stars in the cluster region.

### 1.11 Best fit model

**Todo:** Not finished.

The functions involved in the best fit process aim at obtaining the observed cluster’s parameters (metallicity, age, extinction, distance, binarity, mass), along with the uncertainties in each value.

The process basically consists in comparing the observed cluster’s photometric diagram with the diagrams of many synthetic clusters, and selecting the one that shows the “best match”.

#### 1.11.1 Synthetic clusters

**Todo:** Not finished.

The process by which synthetic star clusters (SSC) are generated by *ASTeCA* involves several steps, described in the sections that follow.

The result of these steps is a SSC with given metallicity, age, extinction, distance, mass and binarity values, affected by a maximum magnitude cut, completeness star removal and photometric errors mimicking those of the input observed cluster.

**Isochrone interpolation**

The first step is done by the `interp_isoch` function stored in the `get_isoch_params` function. This function generates 1500 points in the theoretical isochrone, as shown in Fig. 1.5.

**Isochrone shift**

**Todo:** Not finished.

**Maximum magnitude cut**

**Todo:** Not finished.

**Mass distribution**

**Todo:** Not finished.
Fig. 1.5: Interpolation of theoretical isochrone of given metallicity and age.
Fig. 1.6: Shift interpolated theoretical isochrone according to given extinction and distance modulus values.

\[ E_{(B-V)} = 0.50 \]
\[ (m-M)_0 = 10.00 \]
Fig. 1.7: Remove stars beyond the maximum magnitude value in the observed cluster.
IMF sampling

The initial mass function (IMF) is the distribution of initial masses for a population of stars. For a population of \( N \) stars with masses \( m_i \) and a total mass of \( M_T \):

\[
IMF \rightarrow \xi(m) = \frac{dn}{dm} \rightarrow dn = \xi(m)dm
\]

\[
M_T = \sum_{i=1}^{N} m_i \rightarrow M_T = C \int_{m_i}^{m_h} m(n)dn =
\]

\[
= C \int_{m_i}^{m_h} m \xi(m)dm
\]

where \( m_l \) and \( m_h \) are the mass limits for the IMF (\( m_h \) is fixed to 100\( M_\odot \) in the code) and \( C \) is a normalization constant. Setting the total mass to unity, \( M_T = 1M_\odot \), allows us to obtain the normalization constant \( C_1 \) and treat the normalized IMF as a PDF:

\[
M_T = 1M_\odot \rightarrow C_1 = \frac{1}{\int_{m_i}^{m_h} m \xi(m)dm}
\]

and thus the normalized IMF can be written as:

\[
PDF(m) = \xi(m)_{norm} = C_1 \xi(m)
\]

This is the first step, performed by the `get-IMF-PDF` function for a given selected IMF (Chabrier 2001, Kroupa et al. 1993, Kroupa 2002)

Once the PDF is generated, every time a new synthetic cluster is created the `get-mass-dist` function is called from within `synth-cluster`. This former function takes the PDF and samples a number of masses randomly from it, following the probabilities distribution given by the PDF, until the mass fixed by the `total-mass` parameter is achieved.

The `get-mass-dist` function thus returns a distribution of masses probabilistically sampled from a certain IMF, whose masses sum up to a total cluster mass.

Binarity

Todo: Not finished.

Completeness

Todo: Not finished.

Errors

Todo: Not finished.
1.11.2 Likelihood

Todo: Not finished.

The best fit synthetic cluster (SYC) is obtained via one of two likelihood equations. The tolstoy method in the Cluster parameters assignation block (BF ID line) uses Eqs. 9, 10 & 11 defined in the original article describing ASteCA.

The dolphin method uses Eq. 10 in the Dolphin (2002) article. This equation compares the Hess diagrams of the observed SC vs the SYC, as shown in Fig. 1.8.

![Observed and synthetic clusters](image)

Fig. 1.8: Observed and synthetic clusters (left) are compared via their respective Hess diagrams (center and right). The minimum \( -\ln(PLR) \) value determines the best fit SYC. Click on the image to enlarge.

When dolphin is used to obtain the best fit SYC, the membership probabilities obtained by the decontamination algorithm (see Sect. 10) are included in the equation via the weights parameter in the `numpy.histogramdd` function.

1.11.3 Genetic algorithm

Todo: Not finished.

The implementation of the GA can be divided into several blocks composed of operators, as shown in Fig. 1.9.

The first step consists in drawing a random initial population, that is a set of \( N_{pop} \) random models taken from \( B \). This set is evaluated to obtain the \( L_i \) likelihood for each \( B_i \) model via Eq. ?? and fed to the GA loop as the starting generation.

The selection block (in gray) contains the selection operator that picks models to reproduce from the generation pool with a breeding probability given by their ranked fitness (the rank-based fitness function associates higher \( L_i \) values...
Fig. 1.9: Genetic algorithm work tree.
with higher probabilities) and the encode operator which translates the parameter values that define each chosen model into a binary chromosome.

The **breeding** block (in green) pairs chromosomes randomly and produces a new generation via the crossover and mutation operators that combine each pair of parent chromosomes into two (theoretically fitter) offsprings.

The **evaluation** block (in red) is finally applied where each offspring chromosome is decoded into a \( B_i \) model, the best solutions from the previous iteration added into this new generation by the elitism operator (to ensure the fittest chromosomes are not lost) and the evaluation operator which obtains the \( L^*_i \) values associated with this generation. Every generation produces a “best solution” or fittest chromosome/model; we count the number of times this solution remains unchanged throughout generations and store it as \( N_{bs} \). If \( N_{bs} \) reaches a fixed value \( N_{ei} \), the extinction/immigration operator is applied. This operator prevents the GA from getting stuck in a local minima by discarding every chromosome but the fittest one (extinction) and introducing \( N_{pop} - 1 \) new random chromosomes into the generation pool (immigration); the number of times this operator is run consecutively is stored in \( N_{ei}^{ci} \).

An **exit switch** is put in place to terminate the GA if no improvement on the best solution found happened after \( N_{es} \) consecutive applications of the extinction/immigration operator, i.e.: when \( N_{ei}^{ci} = N_{es} \), in which case the GA is forced to halt.

The GA iterates for a maximum of \( N_{gen} \) generations unless the exit switch forces a premature termination, in either case the result is the best fit model found for \( A \), the observed SC, represented by \( L_A(z^*, a^*, d^*, e^*) \). Every variable involved in the GA process (e.g.: \( N_{gen}, N_{pop}, N_{ei}, \) etc.) can be modified via ASSteCA’s input data file.

### 1.11.4 Bootstrap

**Todo:** Not finished.

The bootstrap function is employed to assign uncertainties to each cluster parameter found via the GA (see Sect 11.3).

### 1.12 Performance

The code is optimized as far as my ability with python allows, and hopefully more improvements in this area will be made in the future.

Nonetheless, there are a number of simple things the user can do to reduce the time it takes ASSteCA to process a cluster. Be aware that each one has a cost though, mainly in that the final result will lack estimation of certain parameters when some of these options are used.

1. The KDE p-value test can take up a substantial amount of time to finish. If the cluster is a clear physical entity (i.e.: there are no doubts about its existence as a true clustering of stars) then you can safely skip this test.

2. If you already ran the Bayesian decontamination algorithm function for a given cluster, and the center, radius and number of field regions remains the same in the `params_input.dat` file, then you can use the members file already generated to avoid running this function again. To do this, in the Bayesian decontamination algorithm block select the read mode, then cut the `XXX_memb.dat` file generated (where `XXX` is the name of the cluster; this file is saved along with the output images) and paste it in the `input/` folder, next to the file that contains the photometric data of the cluster.

This way the code will not run this function again, instead it will read the membership probability values already generated.
Warning: This can only be done if the structural parameters (center and radius), the error rejection function used, and the number of field regions defined remains the same. Otherwise the values for the membership probabilities may change.

3- Lower the bootstrap number in the Best synthetic cluster fit function. This option must be used with care and avoided whenever possible. The bootstrap process assigns errors to the cluster’s fundamental parameters (metallicity, age, extinction, distance modulus, mass, binarity) found by the genetic algorithm. If you disable this feature (by setting a value of 1 or 0) two things will happen: first, parameters will have no errors estimation (which is not good); second, the processing time of the GA will be significantly reduced.

The reduced processing time is due to the bootstrap function being the one that takes up most of the running time in the code. This is so because, to obtain the errors estimates, it runs the GA again a number N of times. For example, if the GA takes 1 hour to complete and the bootstrap function was set to run N=10 times, it will take the bootstrap function approximately 10 hours to finish.

Important: One of the biggest advantages of using ASteCA is the possibility of associating statistically meaningful errors to the cluster’s parameters. A parameter that has an assigned value but no error estimate is substantially less valuable (and trustworthy) than one with a reasonable associated error.
CHAPTER 2

Indices and tables

- genindex
- modindex
- search