

**CirBinDis**  
Circumbinary disk analyser  
version 0.1

Paul Magnus Sørensen-Clark  
Jerome Bouvier

July 25, 2015

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Installing</b>	<b>3</b>
2.1	Software requirements . . . . .	3
2.1.1	Installing <b>AstroPy</b> . . . . .	3
2.2	Downloading and updating . . . . .	4
2.3	Make an alias . . . . .	4
<b>3</b>	<b>Preparing your data</b>	<b>4</b>
<b>4</b>	<b>Algorithm</b>	<b>5</b>
4.1	Loading data . . . . .	5
4.2	Cropping . . . . .	5
4.3	Rotating . . . . .	5
4.4	Sylinder . . . . .	6
4.5	Binning . . . . .	6
4.5.1	Algorithm . . . . .	6
4.5.2	Reasoning . . . . .	6
4.6	Mean density of bins, weighted and integrated . . . . .	7
4.6.1	Mass integral . . . . .	7
4.6.2	The $z$ -limits . . . . .	7
4.7	Integrating intensity . . . . .	9
4.8	Full algorithm summary . . . . .	9
<b>5</b>	<b>Configuring and running CirBinDis</b>	<b>10</b>
5.1	Input parameters . . . . .	10
5.2	Executing the code . . . . .	13
5.3	Output . . . . .	13
5.4	The plotting environment . . . . .	13
<b>6</b>	<b>Troubleshooting</b>	<b>14</b>
6.1	Contact the author . . . . .	14
<b>7</b>	<b>Acknowledgments</b>	<b>14</b>
<b>8</b>	<b>Source code summary</b>	<b>14</b>
8.1	input.xml . . . . .	14
8.2	cirbindis.py . . . . .	14
8.3	DensityMap.py . . . . .	14
8.4	Star.py . . . . .	15
8.5	Functions.py . . . . .	15
8.6	plot.py . . . . .	15
8.7	make_testdata.py . . . . .	15

Contact:

Paul Magnus Sørensen-Clark: paulmag91@gmail.com

Jerome Bouvier: [jerome.bouvier@obs.ujf-grenoble.fr](mailto:jerome.bouvier@obs.ujf-grenoble.fr)

<https://github.com/PaulMag/cirbindis>

# 1 Introduction

A small piece of software for receiving an artificial light curve from a simulated density map of a gas disk around a binary star.

This manual explains certain procedures with bash-commands, which exists in Linux/UNIX-based systems (including Apple OS). `CirBinDis` should work in Windows as well, but some bash-commands may be different.

## 2 Installing

### 2.1 Software requirements

You need the current software installed before you can use `CirBinDis`. If you are using Linux/UNIX most likely you have all of these installed already, except maybe `Astropy`. The version numbers is what is *known* to work from testing, but some older (and newer) versions will probably also work. If you get an error when using `CirBinDis` and you have an older version of any of these prerequisites, try updating them before you do any other troubleshooting.

- Python (3 > version  $\geq$  2.7.9)
- NumPy (2 > version  $\geq$  1.9.2)
- SciPy (0.16 > version  $\geq$  0.15.1)
- Matplotlib (2 > version  $\geq$  1.4.3)
- AstroPy (1.1 > version  $\geq$  1.0.2)

#### 2.1.1 Installing AstroPy

**Alternative 1:** I recommend using the Anaconda Python distribution. It installs the latest version of Python including very many libraries (all the ones you need for `CirBinDis`). It is also easier to install new libraries and update existing ones with Anaconda. Get Anaconda here: <http://continuum.io/downloads>

**Alternative 2:** If you want a more quick and easy approach just type this in a terminal to get `AstroPy` immediately:  
> `pip install astropy`

**Alternative 3:** If that does not work, then download the latest version from <https://pypi.python.org/pypi/astropy/>, unpack it, and run this inside the unpacked folder:  
> `python setup.py install`

**Alternative 4:** You can also consult the **AstroPy** website: <http://www.astropy.org/>

## 2.2 Downloading and updating

The source code is available at this GitHub repository:

<https://github.com/PaulMag/cirbindis>

The updated version of this manual is contained within the repository, so make sure to always consult the newest version after installing/updating **CirBinDis** .

**Alternative 1:** Provided that Git is installed on you computer (<https://git-scm.com>) you can easily get all the source files by running the following command at the location where you want the repository (recommended):

```
> git clone https://github.com/PaulMag/cirbindis.git
```

To update **CirBinDis** type this inside the repository folder:

```
> git pull origin master
```

**Alternative 2:** You can download the source files as a zip-archive from here:

<https://github.com/PaulMag/cirbindis>

Click “Download ZIP” on the right side of the interface, unpack the archive, and place it wherever you want.

To update **CirBinDis** you have to download the zip-archive again and replace all the old files with the new ones. In other words, make a fresh install.

## 2.3 Make an alias

We recommended to make the alias “cirbindis” for the command

```
python ~/path.to.repository.folder/circumbinarydisk/src/cirbindis.py.
```

F. ex. place this in your `.bashrc` or `.bash_aliases`:

```
alias cirbindis="python /GitHub/circumbinarydisk/src/cirbindis.py"
```

This alias will be assumed for the rest of this manual.

## 3 Preparing your data

The format of the input data must be an ASCII/CSV-file with three columns where each line represents a datapoint in space (or a pickle-file made by **CirBinDis** ). The two first columns of each line represent the position of a datapoint.  $(x, y)$  if using cartesian coordinates and  $(r, \theta)$  if using polar coordinates. The last column represents the density in this position.

Any units can be used for the input data. How to specify units are covered in the section **Configuring and running CirBinDis** .

## 4 Algorithm

`CirBinDis` produces artificial lightcurves by analysing the provided dataset according to given configurations. In this section the process for extracting the lightcurve from the dataset is explained. You do not have to understand the algorithm to use `CirBinDis` , but it can be an advantage for interpreting the results. For a quick summary, see section 4.8 on page 9.

### 4.1 Loading data

TODO

### 4.2 Cropping

The space covered by the dataset may represent a larger area than the disk you want to analyse. The dataset is cropped to an inner and outer radius such that the shape of the remaining datapoints resembles a donut. The outer radius represents the size of the disk and makes sure that the disk is circular. The inner radius is necessary to avoid treating the stars themselves as dust, and the density of the dust is very low close to the stars anyway.

### 4.3 Rotating

The coordinates of all datapoints are rotated stepwise with the rotation matrix  $R_z$  for  $\theta = [0, 360)^\circ$ .

$$R_z = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

This rotation simulates the physical orbital rotation of the dircumbinary disk. The reason we get a variation in the lightcurve is because when the disk rotates we see the stars through different areas of the disk with different densities.

A rotation also happens around the  $y$ -axis due to the inclination angle  $\phi$ .  $R_y$  is the rotation matrix which would perform this rotation. However,  $R_y$  is *not* used, and the  $y$ -rotation is never performed directly. It is implicitly done in a very different manner, see section 4.6 on page 7.

$$R_y = \begin{bmatrix} \cos(\phi) & 0 & \sin(\phi) \\ 0 & 1 & 0 \\ \sin(\phi) & 0 & \cos(\phi) \end{bmatrix}$$

## 4.4 Cylinder

A section of the datapoints are cropped out, which represents only the cylinder of gas that is between an observer on Earth and the star. These are the datapoints that fall within the cylinder whose base area is defined by the stellar surface and which extends from the stellar surface and infinitely along the  $x$ -axis in positive direction (the de facto limit is the outer radius of the disk). In other words, the observer's position is assumed to be  $(\infty, 0, 0)$ . A cylinder like this is made once for *each* azimuthal rotation of all the points. Thus, each cylinder will be a little different from the previous one (if  $d\theta$  is small). If there are two (or even more) stars a cylinder will be created for the line of sight of each star, so there can be two (or even more) cylinders at the same time.

## 4.5 Binning

### 4.5.1 Algorithm

Each cylinder is sliced up into  $n_{steps}$  bins along the line of sight, where  $n_{steps}$  is given by the field `radiussteps` in `input.xml`.  $N_{cylinder}$  is the number of datapoints contained within a cylinder. For each bin the mean density is computed. The binning algorithm works like this:

1. Sort all datapoints in cylinder according to  $x$ -component.
2. Find  $N_{bin} = N_{cylinder} / n_{steps}$ .
3. First  $N_{bin}$  (sorted) datapoints goes in the first bin, next  $N_{bin}$  datapoints go in the second bin, etc.
4. Create corresponding `dr` array, where the `dr` corresponding the each bin is the difference between the  $x$ -component of the first and last datapoint in that bin.

### 4.5.2 Reasoning

An alternative way this could be done is have a static  $\Delta r$  and check which points fall within  $[r, r + dr]$  for  $r$  in  $[0, 1, 2, 3 \dots] \cdot \Delta r$ , but this requires a boolean test on the entire cylinder for each radius. It is much faster to sort the datapoints in the cylinder once and then just slice it with indices. There could be even smarter ways to do it, but this has worked well for now. A side effect of this method is that  $\Delta r$  is smaller in areas where there are more datapoints. If the grid of datapoints is spaced denser in central areas where the most interesting features are this is a bonus compared to a static  $\Delta r$ .

## 4.6 Mean density of bins, weighted and integrated

### 4.6.1 Mass integral

For each bin a mean density is produced from all the datapoints in that bin. This is done by dividing the total mass of the cylinder with its volume. The mass of a cylinder can be calculated from the following integral.

$$M_{bin,j} = \int \int \int_{V_{bin,j}} \rho(x, y, z) \, dx \, dy \, dz$$

All our density datapoints are in the  $xy$ -plane, so the  $x$  and  $y$  part of the integral can be evaluated by summing the density of each datapoint  $\rho_{0,i}$  multiplied with its respective discrete  $dx_i \, dy_i$ . We will assume that the datapoints in the grid is spaced evenly. This is not necessarily true, but it should be approximately true for most cases, especially if the size of the bin is much smaller than the whole dataset. In this case  $dx_i$  and  $dy_i$  is the same for every datapoint.

We have no data for density variation in the  $z$ -direction. Instead we assume a gaussian decrease of density with increasing distance from the midplane.  $\rho(z)$  is the assumed density at a point with altitude  $z$  above a point  $i$  in the midplane with density  $\rho_0$ .

$$\rho(x_i, y_i, z) = \rho_0(x_i, y_i) \cdot \exp\left(-\frac{z^2}{2H^2}\right)$$

Thus the mass integral has a discrete part and an analytical part. The limits of the discrete part is the area of the midplane  $S_{bin}$  contained in the cylinder and is such that  $N_{bin} \, dx_i \, dy_i = S_{bin}$ . The limits for  $z$  are explained in the following subsection.

$$\begin{aligned} M_{bin} &= \int \int_{S_{bin}} \rho_0(x_i, y_i) \, dx_i \, dy_i \cdot \int_{z_{i,a}}^{z_{i,b}} \exp\left(-\frac{z^2}{2H^2}\right) \, dz \\ &= \sum_i \left( \rho_{0,i} \, dx_i \, dy_i \cdot \frac{\sqrt{\pi}}{2} \sqrt{2H^2} \left[ \operatorname{erf}\left(\frac{z_{i,b}}{\sqrt{2H^2}}\right) - \operatorname{erf}\left(\frac{z_{i,a}}{\sqrt{2H^2}}\right) \right] \right) \end{aligned}$$

### 4.6.2 The $z$ -limits

Each density point is given a weight according to where it is in the cylinder. Points closer to the middle of the cylinder gains larger weight because they represent its full height and thus a larger volume than points near the edges. This is illustrated in figure 1 on the following page.

$$W_i(y) = \frac{\sqrt{r_{star}^2 - (y_i - y_{star})^2}}{\cos(\phi)}$$

The factor  $1/\cos(\phi)$  adjusts the height of the cylinder if it is inclined so that it is always shaped like a circular cylinder.

To get the density inside the entire area of the slice of the cylinder and the variations in density from different altitudes we integrate the density for each point, projected



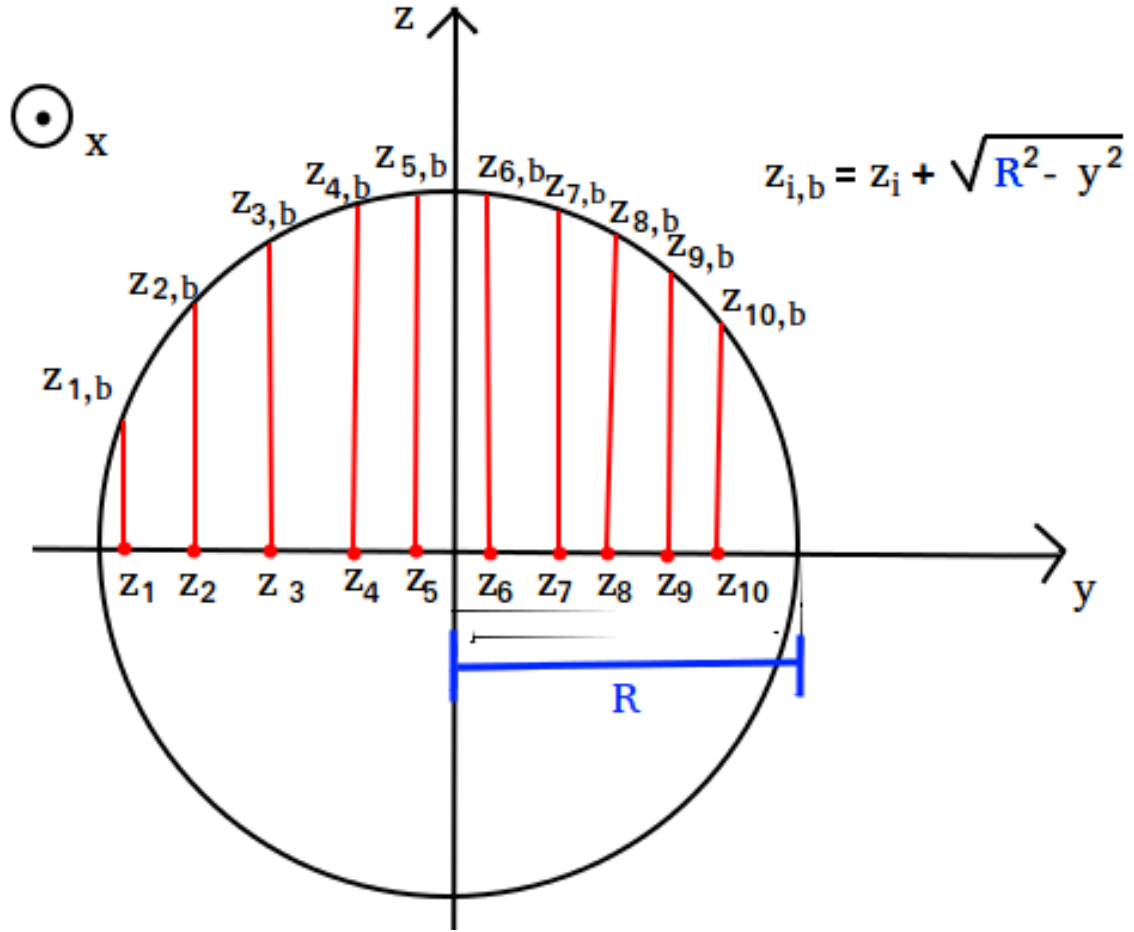


Figure 1: A cross-section of the cylinder. The figure shows how the limits  $z_{i,a}$  and  $z_{i,b}$  for each datapoint depend on that point's  $y$ -component  $y_i$ . The position of  $z_{i,a}$  in the figure is the same as for  $z_{i,b}$ , but mirrored to the lower half of the circle.

from the bottom ( $z_a$ ) to the top ( $z_b$ ) of the cylinder. The distance to integrate is  $2W_i$  for each point, centered around  $z_i$ .

$$\begin{aligned} z_i &= (x_i - x_{star}) \cdot \tan(\phi) \\ z_{i,a} &= z_i - W_i \\ z_{i,b} &= z_i + W_i \end{aligned}$$

## 4.7 Integrating intensity

For each bin  $j$  in each cylinder, from the inside to the outside of the disk, the ratio of intensity transferred from one bin to the next is given by the following expression.

$$\begin{aligned} \tau_j &= \kappa \cdot \rho_{bin,j} \cdot \Delta r_j \\ I_{j+1} &= I_j \exp(-\tau_j) \end{aligned}$$

The resulting intensity passed on by the outermost bin  $I_{end}$  is the intensity of the star's radiation that escapes the disk and is observed by the observer on the current line of sight. If there are several stars and thus several cylinders, the total perceived intensity is simply the sum of the  $I_{end,k}$  for each cylinder  $k$ .

$$I_{total} = \sum_k I_{k,end}$$

## 4.8 Full algorithm summary

This is how one analysis is performed, and the product is one lightcurve. If providing different values for certain parameters, like different inclination angles or different outer radii then this analysis will be performed once for each different value of each parameter (different inclinations are actually analysed in quasi-parallel for efficiency).

```

for each  $\theta$  in  $[0, \dots, 2\pi]$  do
  rotate density datapoints angle  $\theta$ 
  rotate stars angle  $\theta$  (stars move with disk)
  for each star  $k$  do
    extract cylinder
    bin cylinder
    for each bin  $j$  in cylinder do
      
$$\rho_{bin,j} = \frac{\int \int \int_{V_{bin,j}} \rho(x, y, z) \, dx \, dy \, dz}{\pi r_{star,k}^2 \Delta r_j}$$

      
$$\tau_j = \kappa \cdot \rho_{bin,j} \cdot \Delta r_j$$

      
$$I_{k,j+1} = I_{k,j} \exp(-\tau_j)$$

    end for
  end for
  
$$I_{\theta,total} = \sum_k I_{k,end}$$

end for
```

## 5 Configuring and running CirBinDis

How to make necessary configurations and then run `CirBinDis` to perform an analysis.

This is the most practical and maybe the most important section, as it explains how to actually use the software.

### 5.1 Input parameters

The input parameters for each run of `CirBinDis` is configured in an XML file with a predetermined layout. Inside the repository you will find `/xml/input.xml`. Copy this file, save it together with your dataset, and modify the values of the fields as required for your dataset (do not blindly use the default values).

Specifically, this is where you provide the filename of the dataset to analyse. If `input.xml` is in another folder than the dataset you need to write the relative or absolute pathname of the datafile.

You can save your copy of the XML-file with whatever name you wish, which can be useful to link separate XML-files to specific datasets that are in the same folder, or if you have different sets of parameters that you want to reuse on the same dataset.

Here follows a detailed description of what all the fields in the input file means.

**unit-mass** Here you state which mass unit you want to use. This unit is interpreted by `AstroPy`'s Unit-module and it can be almost any mass unit or quantity of units you can think off. The full list of possibilities exists here: <http://astropy.readthedocs.org/en/latest/units/#module-astropy.units.si> The most practical unit is often the solar mass, which is written as `"solMass"`. You could use `"geoMass"` (Earth's mass) or `"1e10 kg"`.

**unit-distance** Same rules as for unit-mass, but this is your distance unit. Typical units can be `"au"`, `"11 solRad"`, or `"1.7e10 m"`.

**unit-angle** This is your distance unit. It can be `"rad"`, `"deg"`, `"arcmin"`, `"arcsec"`.

**datafile** The pathname to the dataset to be analysed. The pathname must be relative to where you run the `CirBinDis` from, f.ex. `"data/mydiskdata.dat"`, or it can be an absolute path. It can be an ASCII file with 3 columns or a pickle file. If it is ASCII you must also specify if the coordinates are cartesian ( $x, y, \rho$ ) or polar ( $r, \theta, \rho$ ).  $x, y, r$  will be in the distance unit you chose.  $\theta$  (if you use polar coordinates) will be in the angle unit you chose. The density  $\rho$  will be scaled to a suitable unit "behind the scenes", depending on `radius_in`, `radius_out`, `diskradius`, and `diskmass`.

**dataname** A name/identifier for the current dataset. This name will be included in the filenames of the output lightcurves and in the title of the plots. If left blank the datafilename will be used instead.

**resave\_as** If you provide a filename/pathname here the cropped version (according to `radius_in`, `radius_out`) of the input datafile will be saved. This cropped version of the data can then be used as the input datafile later. This is useful because it can reduce the size of the datafile. Loading millions of lines of data takes a while. If you use `".p"` or `".pickle"` as filename extension it will be saved as a pickle file. If left blank there will *not* be any saving. If several filenames are provided (separated with spaces) several copies will be made with different names. Usually you will want to resave each of your datasets as a pickle-file once, and after that you should leave this field blank.

**normalization** How to normalize the output data (lightcurve). There are 3 options, as listed below. You can choose several of them at the same time by writing several words separated by spaces, or you can write `"all"`.

**mean** Divide each lightcurve with its mean value.

**max** Divide each lightcurve with its maximum value.

**stellar** Divide each lightcurve with its unobscured stellar value. By that we mean the flux it would have if there was no extinction. This is the only normalization method where we see the relation between the different curves. If some curves are much fainter than others they will appear as  $\approx 0$ .

**system** Write `"cartesian"` if your data is  $(x, y, \rho)$  or `"polar"` if your data is  $(r, \theta, \rho)$ .

**outfolder** The pathname to the folder to contain the output files. The pathname must be relative to where you run the program from, or it can be an absolute path. If the folder does not already exist it will be created automatically. Inside this folder there will be created 2 more folders: `"plots"` for the plot images and `"csvtables"` for CSV-files for each lightcurve on the format  $(\theta, \text{flux})$ .

**radius\_in** Define the inner and out radius for where the part of the disk which causes extinction exists. The number you provide will be in the units you decided in unit-distance. The dataset will be cropped to these limits, so it becomes a "donut". You can provide several inner and outer radiuses by separating them with spaces if you want to analyse different sized disks at the same time. If `radius_in` is left blank it will default to just outside the position of the stars.

**radius\_out** If `radius_out` is left blank it will default to the smallest radius that contains the entire dataset

**inclination** Which inclinations to analyse the system in. Several inclinations can be separated with spaces. When you provide several inclinations their respective lightcurves will be displayed in the same plot. If no inclinations is provided a default of  $90^\circ$  (edge-on) will be chosen.

**diskmass** The total mass of the entire circumstellar/binary disk, including the vast outer part which doesn't cause extinction. The number you provide will be in

the units you decided in unit-mass. A typical value is 0.01 solMass or smaller. This number is used to calculate a density scaling factor.

**diskradius** The total radius of the entire circumstellar/binary disk, including the vast outer parst which doesn't cause extinction. The number you provide will be in the units you decided in unit-distance. A typical value is 50 AU. This number is used to calculate a density scaling factor.

**kappa** Opacity constant.  $\kappa$  is always provided in units of  $cm^2/g$ . A typical value is between 5 and 100.

**H0** The semi-thickness of the disk. The number you provide will be in the units you decided in unit-distance.  $H_0$  determines the density at a distance from the midplane of the disk. If  $H_0$  is small there can be less obstruction at higher inclinations.  $\rho(x, y, z) = \rho_0(x, y) \cdot \exp(-z^2/H_0^2)$

**star** The star contains several subfields, which determines the parameters for one particular stars. If you want 2 (or more) stars, copy-paste the entire star section with all its fields and fill in the parameters for each star individually.

**x,y** The cartesian position of the star in unit-distance. Leave  $< r, theta >$  blank or delete them entirely if you use these.

**r, theta** The polar position of the star in unit-distance and degrees(!). Leave  $< x, y >$  blank or delete them entirely if you use these.

**radius** The radius of the star in unit-distance. This determines the radius of the line-of-sight cylinder.

**intensity** The intensity of the star. This is entirely unitless, because the lightcurves are always normalized anyway. What matters is only the relative intensity between the stars, if you have several stars.

**azimuthsteps** How many different line-of-sights to analyse. This is the resolution of the output lightcurves.  $d\theta = 360^\circ / \text{azimuthsteps}$ . The entire dataset needs to go through a matrix rotation for each azimuthstep, and that is by far the slowest part of the algorithm. It is often wise to choose a low number here first to get a rough idea of what the lightcurves will look like and then increase it to over 100 steps when you want to see fine structures.

**radiussteps** How many bins to divide the line-of-sight cylinders in. This number should be as high as possible to increase accuracy of the flux integration. If you make density profiles this will determine the resolution. A typical value is 100, but if you have enough datapoints, set it even higher. If you try to use more radius steps than there is datapoints in each cylinder an error will happen.

## 5.2 Executing the code

When you have prepared your input XML-file with your dataset, type the command `cirbindis`

(or `python ~/path_to_repository_folder/circumbinarydisk/src/cirbindis.py`) followed by the name of your XML-file in a terminal. F.ex.:

```
> cirbindis input.xml
> cirbindis dataA.xml
> cirbindis dataA_big.xml
> cirbindis data/set1.xml
> python src/cirbindis.py input.xml (without alias)
```

The software will run until it has completed the analysis of your dataset with the parameters you specified, or stop and throw an error message if there is a problem with the configuration.

## 5.3 Output

The output after a `CirBinDis` analysis is a comma-separated-value (csv) file. The output file will be placed in the path you specified in `input.xml`. The filename contains the value of the parameters  $H$ ,  $r_{in}$ ,  $r_{out}$ , and inclination ( $\phi$ ), separated by double underscores “\_\_”. A filename can be f.ex. “`H=0.1__r_in=0.75__r_out=3__inc=5.csv`”. If you perform several analysis of the same dataset at once, f.ex. by providing several values for  $r_{out}$ , then one outfile will be produced for each different value of  $r_{out}$ .

The first line of the output is a header containing all the physical and numerical parameters for the simulation. There are two columns. The first column lists rotation angles  $\theta$  in units of degrees. The second column lists the observed intensities given the respective angles, normalised so the mean intensity is 1. A output file with `azimuthsteps=8` can look like this:

```
#H=0.1,␣kappa=10,␣r_star=0.21-0.19,␣r_in=0.75,␣r_out=3,␣dr=0.75,
dtheta=45deg,␣inc=5deg
0.000000,0.000000
45.000000,0.000000
90.000000,0.000000
135.000000,0.000000
180.000000,7.644186
225.000000,0.000000
270.000000,0.355814
315.000000,0.000000
```

## 5.4 The plotting environment

If you are unfamiliar with the `matplotlib` plotting environment, I recommend that you have a quick look at the following url: <http://matplotlib.org/users/>

`navigation_toolbar.html` Here it is explained how to manipulate the plot, like zooming or changing the axes. You can always save the current state of the plot as png, ps, eps, svg or pdf.

## 6 Troubleshooting

### 6.1 Contact the author

If you cannot find out how to do something and this manual does not explain it, send an email to `paulmag91@gmail.com` and ask. Do this also if you have feedback or suggestions for improvements, as `CirBinDis` is under development.

## 7 Acknowledgments

TODO

## 8 Source code summary

Here is a summary of what the individual files of `CirBinDis` does. The full source code is available at <https://github.com/PaulMag/cirbindis> as explained in section 2.2 on page 4.

### 8.1 `input.xml`

The file for the user to provide input parameters to `CirBinDis` . You can copy and modify it. It is not a part of the source code itself.

### 8.2 `cirbindis.py`

The main file. The script that is called when running `CirBinDis` .

### 8.3 `DensityMap.py`

Contains the class `DensityMap` for making an instance of a dataset representing a circumbinary disk. It contains most methods that can be performed on the data. Also contains a subclass `Sylinder`. `Sylinders` a sub-sets of a full dataset.

## 8.4 `Star.py`

The simple `Star` class whose intention is to hold the physical parameters of each star.

## 8.5 `Functions.py`

A file containing some general functions that are used other places in the program.

## 8.6 `plot.py`

A standalone script that can be used to plot the output of `CirBinDis` . You can just as well use something else, f.ex. TOPCAT. This script may be outdated.

## 8.7 `make_testdata.py`

A standalone script that can be used to generate artificial datasets that can be analysed by `CirBinDis` . You can use it for testing and for generating data according to any analytical function that you would like to analyse (then you need to change the function `density`).