CirBinDis

Circumbinary disk analyser

version 0.4.1

Paul Magnus Sørensen-Clark
Jerome Bouvier

August 2, 2015

Contents

1 Introduction **3**

2 Installing **3**
2.1 Software requirements . . . 3
2.1.1 Installing `AstroPy` . . . 3
2.2 Downloading and updating . . . 4
2.3 Make an alias . . . 4

3 Preparing your data **5**

4 Configuring and running `CirBinDis` **5**
4.1 Input parameters . . . 5
4.2 Executing the code . . . 9
4.3 Output . . . 9
4.4 The plotting environment . . . 9

5 Algorithm **10**
5.1 Loading data . . . 10
5.2 Cropping . . . 10
5.3 Density scaling . . . 10
5.4 Rotating . . . 10
5.5 Sylinder . . . 11
5.6 Binning . . . 11
5.6.1 Algorithm . . . 11
5.6.2 Reasoning . . . 12
5.7 Mean density of bins, weighted and integrated . . . 12
5.7.1 Mass integral . . . 12
5.7.2 The z-limits . . . 13
5.8 Integrating intensity . . . 15
5.9 Full algorithm summary . . . 15

6 Troubleshooting **16**
6.1 Contact the author . . . 16

7 Acknowledgments **16**

8 Source code summary **16**
8.1 `input.xml` . . . 16
8.2 `cirbindis.py` . . . 16
8.3 `DensityMap.py` . . . 17
8.4 `Star.py` . . . 17
8.5 `Functions.py` . . . 17
8.6 `plot.py` . . . 17
8.7 `make_testdata.py` . . . 17

Contact:

Paul Magnus Sørensen-Clark: paulmag91@gmail.com

Jerome Bouvier: 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 is version 0.4.1 of the usermanual. If your `CirBinDis` is in a different version, make sure to update either the software or the manual, see section 2.2 on the following page.

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-commads 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 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, θ) 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 section 4.1.

4 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.

4.1 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 diffenert 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 ACII file with 3 columns or a pickle file. If it is ASCII you must also specify if the coordinates are cartesian (x, y, ρ) or polar (r, θ, ρ). x, y, r will be in the distance unit you chose. θ (if you use polar coordinates) will be in the angle unit you chose. The density ρ 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 ≈ 0.

system Write "`cartesian`" if your data is (x, y, ρ) or "`polar`" if your data is (r, θ, ρ).

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 (θ,flux).

lightcurves-show_plot yes/no: Do you want to show the interactive Matplotlib plotting interface with the lightcurve plots when the analysis is complete (ee section 4.4 on page 9)? If you want the chance to manipulate the plots before saving them you should do this. Or if you want to see the plots, but not save them, you should do this.

lightcurves-save_plot yes/no: Do you want to save the lightcurve plots directly as an image file with the default axes and labels, etc. when the analysis is complete?

lightcurves-save_csvtable yes/no: Do you want to save the actual output lightcurve data as CSV-files? Use this if you f.ex. want to plot the results with another program.

densityprofiles-show_plot yes/no: Do you want to show the interactive Matplotlib plotting interface with the density profile plots when the analysis is complete? If you have several stars in your model density profiles are only made for the first one by default (first in the input XML-file). Note: One densityprofile subplot will be made for *each* azimuthstep, so when making densityprofiles you should not use a very large number of azimuthsteps. F.ex. 4 or 9 is ok.

densityprofiles-save_plot yes/no: Do you want to save the lightcurve plots directly as an image file with the default axes and labels, etc. when the analysis is complete?

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 it 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 ° (edge-on) will be chosen.

diskmass The total mass 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-mass. A typical value is 0.01 solMass or smaller. This number is used to calculate a density scaling factor. You can provide several diskmasses by separating them with spaces to perform several analysis with different densities.

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. κ is always provided in units of cm^2/g. A typical value is between 5 and 100.

H0 The semi-thickness of the disk at radius R_0. The number you provide will be in the units you decided in unit-distance. H_0 determines the density at a height from the midplane of the disk: $\rho(x, y, z) = \rho_0(x, y) \cdot \exp(-z^2/H_0^2)$. If H_0 is small there can be less obstruction at higher inclinations. You can provide several values for H_0 by separating them with spaces it you want to analyse different thicknesses.

R0 The radius where $H = H_0$. R_0 only matters if $H_{\text{power}} \neq 0$. The number you provide will be in the units you decided in unit-distance.

H_power $H(R) = H_0(R/R_0)^{H_{\text{power}}}$
$H_{\text{power}} = 0 \Rightarrow$ The disk has uniform thickness.
$H_{\text{power}} = 1 \Rightarrow$ The thickness is proportional to radius.

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 sylinder.

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. $\mathrm{d}\theta = 360\,^\circ/\text{azimuthsteps}$. 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 sylinders 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 sylinder an error will happen.

sylindergridz How many line-of-sights to divide each sylinder in (in z-direction only). If H is small compared to the star radius then this number should be larger, to be able to resolve the disk's thickness. This number greatly affects the run time of the code.

4.2 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.

4.3 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 (ϕ), 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 θ 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
```

4.4 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.

5 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 5.9 on page 15.

5.1 Loading data

TODO

5.2 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.

5.3 Density scaling

TODO:
This section should explain how arbitrary dimensionless input density is converted to physical density units.

5.4 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 ϕ. $\phi = 90\,^\circ$ means that we see the disk edge-on (this is the unrotated y-state). 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 5.7 on the following page.

$$R_y = \begin{bmatrix} \cos(90\,^\circ - \phi) & 0 & \sin(90\,^\circ - \phi) \\ 0 & 1 & 0 \\ \sin(90\,^\circ - \phi) & 0 & \cos(90\,^\circ - \phi) \end{bmatrix}$$

5.5 Sylinder

A section of the datapoints are cropped out, which represents only the sylinder of gas that is between an observer on Earth and the star. These are the datapoints that fall within the sylinder 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). The sylinder is also inclined with the angle ϕ. In other words, the observer's position is assumed to be $[d\sin(\phi), 0, d\cos(\phi)]$, where $d \to \infty$. A sylinder like this is made once for *each* azimuthal rotation of all the points. Thus, each sylinder will be a little different from the previous one (if $\mathrm{d}\theta$ is small). If there are two (or even more) stars a sylinder will be created for the line of sight of each star, so there can be two (or even more) sylinders at the same time.

5.6 Binning

5.6.1 Algorithm

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

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

5.6.2 Reasoning

An alternative way this could be done is have a static Δr and check which points fall within $[r, r + \mathrm{d}r]$ for r in $[0, 1, 2, 3 \ldots] \cdot \Delta r$, but this requires a boolean test on the entire sylinder for each radius. It is much faster to sort the datapoints in the sylinder 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 Δr 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 Δr.

5.7 Mean density of bins, weighted and integrated

5.7.1 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 sylinder with its volume. The mass of a sylinder can be calculated from the following integral.

$$M_{bin,j} = \int\int\int_{V_{bin,j}} \rho(x, y, z)\ \mathrm{d}x\ \mathrm{d}y\ \mathrm{d}z$$

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 $\mathrm{d}x_i\ \mathrm{d}y_i$. 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 $\mathrm{d}x_i$ and $\mathrm{d}y_i$ is the same for every datapoint.

Whe have no data for density variation in the z-direction. Instead we assume a gaussian decrese 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 ρ_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 edges of the bin in the midplane and is such that $N_{bin}\ \mathrm{d}x_i\ \mathrm{d}y_i = S_{bin}$. We assume that the datapoints within one sylinder bin is evenly spaced such that $\mathrm{d}x_i$ and $\mathrm{d}y_i$ are the same for all points. This is not always exactly the case, but probably an ok approximation. The limits for z are explained in section 5.7.2.

The gaussian's integral has no analytical solution, but it can be approximated by the error function (erf), which is a power series. erf is provided in the Python library SciPy (`scipy.special.erf`) and can handle arrays, which means that this integral can be performed for all the datapoints in the sylinder bin very quickly in

a vectorized manner.

$$\begin{aligned} M_{bin} &= \int_{-r_{star}}^{r_{star}} \int_{r_{bin}}^{r_{bin}+\Delta r} \rho_0(x_i, y_i)\ \mathrm{d}x_i\ \mathrm{d}y_i \cdot \int_{z_{i,a}}^{z_{i,b}} \exp\left(-\frac{z^2}{2H^2}\right)\ \mathrm{d}z_i \\ &= \sum_i \left(\rho_{0,i}\ \mathrm{d}x_i\ \mathrm{d}y_i \cdot \frac{\sqrt{\pi}}{2}\sqrt{2H^2}\left[\mathrm{erf}\left(\frac{z_{i,b}}{\sqrt{2H^2}}\right) - \mathrm{erf}\left(\frac{z_{i,a}}{\sqrt{2H^2}}\right)\right]\right) \\ &= S_{bin} \sum_i \left(\rho_{0,i} \cdot \frac{\sqrt{\pi}}{2}\sqrt{2H^2}\left[\mathrm{erf}\left(\frac{z_{i,b}}{\sqrt{2H^2}}\right) - \mathrm{erf}\left(\frac{z_{i,a}}{\sqrt{2H^2}}\right)\right]\right) \end{aligned}$$

$$\begin{aligned} \rho_{bin} &= \frac{M_{bin}}{V_{bin}} \\ &= \frac{M_{bin}}{S_{bin} \cdot \sum_i \int_{z_{i,a}}^{z_{i,b}}\ \mathrm{d}z_i} \\ &= \frac{\sum_i \left(\rho_{0,i} \cdot \frac{\sqrt{\pi}}{2}\sqrt{2H^2}\left[\mathrm{erf}\left(\frac{z_{i,b}}{\sqrt{2H^2}}\right) - \mathrm{erf}\left(\frac{z_{i,a}}{\sqrt{2H^2}}\right)\right]\right)}{\sum_i \int_{z_{i,a}}^{z_{i,b}}\ \mathrm{d}z_i} \end{aligned}$$

ρ_{bin} is thus the mean density of one bin of the sylinder, and it has a corresponding Δr. A ρ_{bin}, Δr pair is calculated for each of the N_{bin} bins in each sylinder.

5.7.2 The z-limits

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

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

The factor $1/\sin(\phi)$ adjusts the radius of the sylinder if it is inclined so that it is always shaped like a circular sylinder. See figure 2.

To get the density inside the entire area of the slice of the sylinder and the variations in density from different altitudes we integrate the density for each point, projected from the bottom (z_a) to the top (z_b) of the sylinder. The distance to integrate is $2W_i$ for each point, centered around z_i.

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

(If a star is positioned at origo $x_{star} = y_{star} = 0$, which is typically the case if you have only one star and thus one sylinder.)

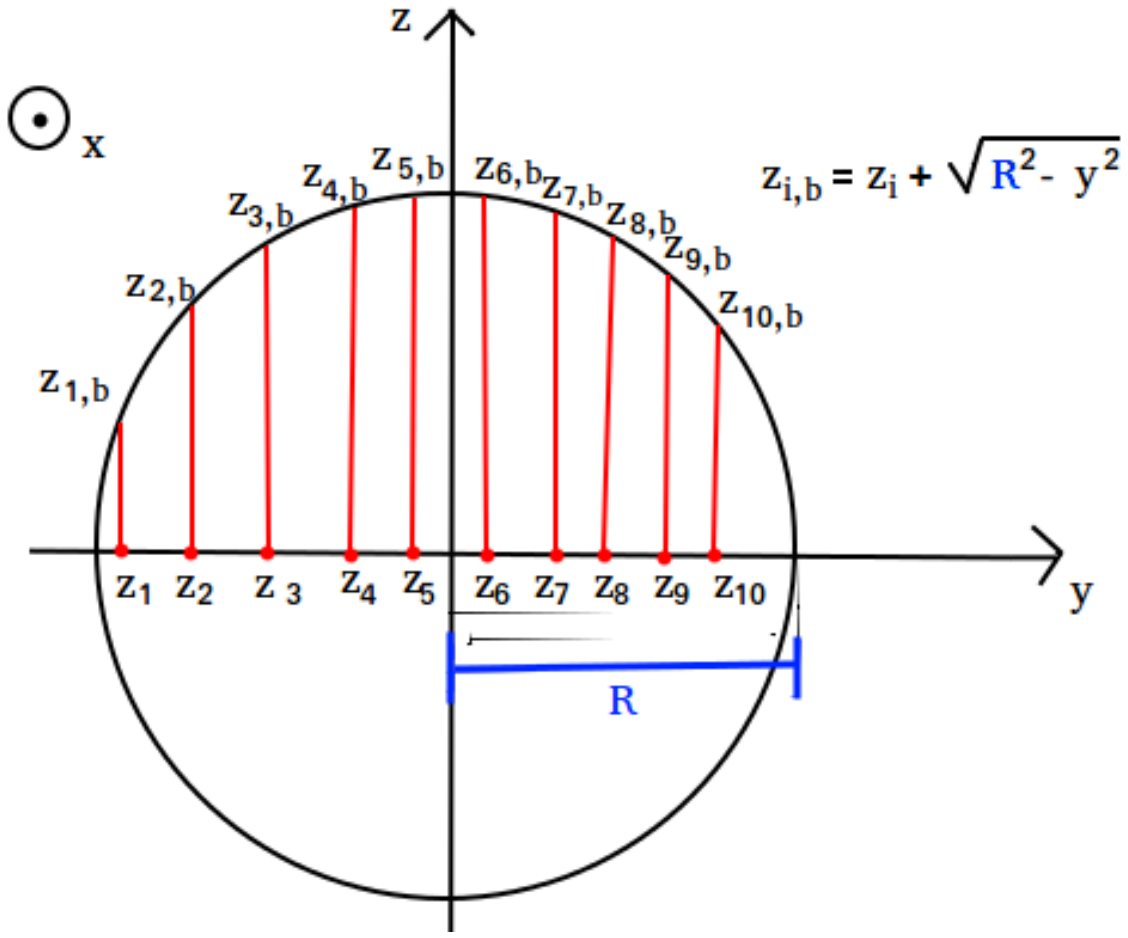

Figure 1: A cross-section of the sylinder. The figure shows how the limits $z_{i,a}$ and $z_{i,b}$ for each datapoint depend on that point's y-component y_i. The position of $z_{i,a}$ in the figure is the same as for $z_{i,b}$, but mirrored to the lower half of the circle.

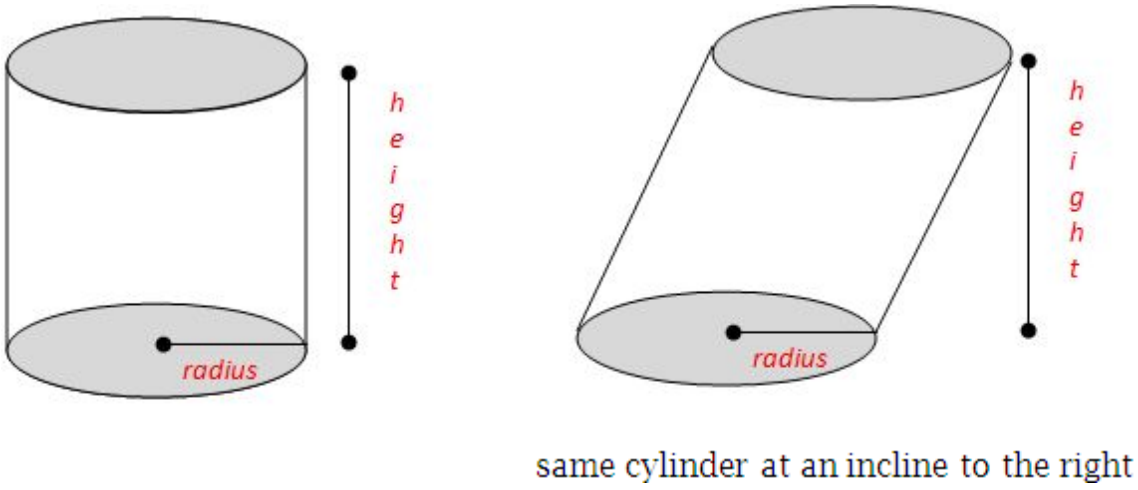

Figure 2: This is how the sylinder becomes inclined. The base radius (and volume) is the same, but from the line-of-sight point of view it is flattened and no longer spherical. From the point of view of the observer the radius (in one direction only) becomes $\sin(\phi) \cdot \text{radius}$.

5.8 Integrating intensity

For each bin j in each sylinder, 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.

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

The resulting intensity passed on to 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 sylindres, the total perceived intensity is simply the sum of the $I_{end,k}$ for each sylinder k.

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

5.9 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-parallell for efficiency).

for each θ in $[0, \ldots, 2\pi]$ **do**
  rotate density datapoints angle θ
  rotate stars angle θ (stars move with disk)

for each star k **do**
extract sylinder
bin sylinder
for each bin j in sylinder **do**

$$\rho_{bin,j} = \frac{\int\int\int_{V_{bin,j}} \rho(x,y,z)\ \mathrm{d}x\ \mathrm{d}y\ \mathrm{d}z}{\pi r_{star,k}^2 \Delta r_j}$$

$\tau_j = \kappa \cdot \rho_{bin,j} \cdot \Delta r_j$
$I_{k,j+1} = I_{k,j} \exp(-\tau_j)$
end for
end for
$I_{\theta,total} = \sum_k I_{k,end}$
end for

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`).