BBarolo Documentation

Release 1.7

Enrico Di Teodoro

Oct 23, 2023

BBAROLO:

3D-Barolo (3D-Based Analysis of Rotating Objects via Line Observations) or **BBarolo** is a tool for fitting 3D tilted-ring models to emission-line datacubes. The Python wrapper is **pyBBarolo**.

For a detailed description of the algorithms used in this code, please refer to BBarolo's main paper. The documentantion is also available in PDF here.

Table of contents:

CHAPTER

ONE

INSTALLING BBAROLO

1.1 Pre-compiled binaries

Pre-compiled executable files for the GUI and the command line utility are available at this page. Binaries are for Linux x86_64 and Mac OS X (> 10.6).

The command line executable (`BBarolo`) is also included in the GUI binary packages and can be found at:

- **Linux**: same directory of BBaroloGUI
- **MacOS**: BBaroloGUI.app/Contents/MacOS/BBarolo

The pre-compiled executables should work on most systems. If they don't, please compile BBarolo from source following the instructions below.

1.2 Compiling from source

The best way to exploit BBarolo functionalities is to compile the code directly on your computer. BBarolo compiles and runs on Unix machines only.

1.2.1 Requirements

To compile the code, all you need is:

- a C++ compiler supporting C++11 standard, like the GNU compiler. OpenMP support is required for multi-threading.
- CFITSIO library.
- FFTW library.
- WCS library.
- QT toolkit (> 4.0), only if you would like to use the GUI.
- Gnuplot and Python (> 2.6) with the Astropy package.

Most of these libraries and packages should already be installed on scientific machines. Otherwise, you can easily install them through the terminal commands of the various package managers, i.e. *apt-get* on Ubuntu-based, *pacman* on Arch-based, *yum* on RPM-based distros, *brew* or *port* on Mac OS X. Note that the QT toolkit is only needed to compile the GUI, which is optional. Gnuplot and Python are not needed to successfully compile the code, but without them BBarolo will not produce any outputs.

1.2.2 Compiling

If your machine satisfies the above requirements, compiling BBarolo will hopefully be a piece of cake.

1. **Download** the latest stable release. From a terminal:

```
> wget https://github.com/editeodoro/Bbarolo/archive/X.Y.tar.gz
```

   where *X.Y* is the release version. If you are brave, you can also try the latest (non stable) source code from here.

2. **Uncompress** it and enter the BBarolo directory:

```
> tar -xvf X.Y.tar.gz
> cd Bbarolo-X.Y
```

3. **Configure** running autoconfigure script:

```
> ./configure
```

   If the script is not executable: `> chmod +x configure`. The configure script takes a number of optional arguments. For instance, it is possible to specify installation and library directories, or the compiler to use:

```
> ./configure CXX=icpc --prefix=/dir/to/install --with-cfitsio=/dir/of/
↪cfitsio --with-wcslib=/dir/to/wcslib
```

   If the configuration fails, follow the suggestions given by the script to manually set the path of libraries.

4. **Compile** the source:

```
> make
```

   To compile in parallel: `> make -j N`, where N is the number of processors. If the compilation succeeds, the executable **BBarolo** will appear in the current directory.

Optional steps

5. Install, e.g. copy the executable in the installation path (default is /usr/local/bin):

```
> make install
```

6. Compile the GUI (QT > 4 needed):

```
> make gui
```

   This can fail for a number of reasons.

7. Compile BBarolo as a library:

```
> make lib
```

8. Clean up unnecessary files:

```
> make clean
```

1.3 Installing via Homebrew (MAC only)

If you use Homebrew package manager for MacOSX, there is a simpler way of compiling and installing the code:

```
> wget https://raw.github.com/editeodoro/Bbarolo/master/bbarolo.rb
> brew install [--HEAD] bbarolo.rb
```

The first command downloads a ruby installing script, the second command installs BBarolo in /usr/local/Cellar/bbarolo and symlinks the executable to /usr/local/bin. Homebrew takes care of all dependencies. After installation, you will be able to run the code from any directory just by typing `BBarolo`. The optional argument `--HEAD` install the latest non-stable version rather than the stable release.

To uninstall BBarolo:

```
> brew uninstall bbarolo
```

CHAPTER

TWO

RUNNING BBAROLO

2.1 Graphical User Interface

BBarolo comes with a Graphical User Interface (GUI) that can help the user in setting up the input parameters. Running BBarolo through the GUI should be quite straightforward: you do not have to learn the annoying list of *available parameters*: the user needs just to fill the required fields in the GUI and this will create a text file with the correct parameters and run BBarolo.

N.B.: Although the GUI allows the user to set the main parameters, many options can only be enabled through the command line tool. Moreover, I stress that the GUI has not been updated together with the last releases of BBarolo. If you experience any issues with the GUI or want to have full control of the code, I recommend you to use the command line.

2.2 Command line

BBarolo is mainly meant to be run from the command line. For a very quick guide and to appreciate the biggest achievement of my PhD, just type `BBarolo` on your keyboard in a terminal window.

Execution with a parameter file: BBarolo takes input parameters specified through a parameter file, provided at the runtime. This is a text file containing a list of parameter names and values:

```
PARAM1      VALUE1
PARAM2      VALUE2
PARAM3      VALUE3
...         ...
```

All available parameters are described in the *task documentation*. In the input file, parameter names are not case-sensitive and lines starting with # or // are not read in. The order in which parameters are listed is unimportant, but, if a parameter is listed more than once, only the last value is considered.

Some parameters are mandatory, some others are optional and have default values which are assumed when not explicitly set. A template parameter file for the 3DFIT task can be obtained with the command `> BBarolo -t`. A list of all parameters with their default values can be printed with `> BBarolo -d`. An example of parameter file can be found here, full runnable instances can be downloaded from this page. The command `> BBarolo -v` will return information about the code version and compiler flags.

After your parameter file is ready, BBarolo can be run with the following:

```
> BBarolo -p paramfile
```

where *paramfile* is the name of the user-defined input file. Since version v1.6, individual parameter values can also be overridden from the command line directly, without changing the parameter file. For example:

```
> BBarolo -p paramfile INC=60 PA=120
```

will set a INC of 60 degrees and a PA of 120 degrees and ignore the values listed in the *paramfile*.

Execution with command-line arguments: BBarolo can alternatively read all parameters directly from the command line using the `-c` option. Parameters are given in the form `PARAM=VALUE` with no blank spaces:

```
> BBarolo -c PARAM1=VALUE1 PARAM2=VALUE2 PARAM3=VALUE3
```

If `VALUE` must contain white spaces, just include it in between quotation marks (e.g. `FREE="VROT VDISP"`). Names of parameters are not case-sensitive. This way is very convenient when the user only needs to run a task with few parameters. For example, to run the source finder with default parameters:

```
> BBarolo -c fitsfile=yourfits.fits search=true
```

Automated execution: BBarolo can otherwise be run in a completely automated way, i.e. providing no parameters but the input FITS datacube. In this case, the code uses the source finder to identify the galaxy in the datacube, tries to guess initial values for the rings and fits a 3D model to the data. Although this procedure might work and return nice best-fit models if used with high resolution and high S/N data, it should be used carefully. In particular, the algorithm for guessing initial values for the fit is still quite coarse. Wrong initial guesses may lead to completely inappropriate models.

If you still want to try the automated execution:

```
> BBarolo -f fitsfile
```

where *fitsfile* is the name of the FITS file of the galaxy to analyse.

CHAPTER

THREE

LIST OF TASKS AND PARAMETERS

BBarolo's main algorithm for fitting 3D kinematic models to emission line data (*3DFIT*) makes use of a number of utilities. These tasks include, for example, the disk modeling (*GALMOD*), the source finder (*SEARCH*) and the smoothing utility (*SMOOTH*), and can be conveniently used outside the main algorithm as well.

In this page, I list the main tasks and related input parameters available in BBarolo. Parameter names are in **boldface**, default values are in brackets. The names of parameters are not case-sensitive.

3.1 General parameters

In the following, a list of general parameters (e.g., not task-specific).

3.1.1 Input/output parameters

- **FITSFILE** [none]. The name of the input FITS file. This is a mandatory parameter for **all tasks**.
- **OUTFOLDER** [./output]. The directory where the output files will be written.
- **VERBOSE** [true]. Enable all the output messages.
- **THREADS** [max CPUs]. Number of CPUs to use for task execution. All BBarolo's tasks have shared-memory parallelization. The code needs to be compiled with OpenMP support. If you encounter any problem with multi-thread execution, switch back to single-thread mode and signal the problem.
- **PLOTS** [true]. If true, output plots will be produced (Python/Gnuplot needed).
- **SHOWBAR** [true]. Whether to show progress bars.
- **STATS** [false]. If true, calculate and print statistics of input FITS file.

3.1.2 Beam parameters

Following parameters can be used to specify the size and shape of the Point Spread Function (PSF or beam). These parameters are ignored if beam information is written in the header of the input FITS, either through BMAJ, BMIN and BPA keywords or in the HISTORY. The code defines the beam following the priority order: header -> bmaj,bmin,bpa params -> beamfwhm param -> default to 30 arcsec.

- **BMAJ** [none]. The FWHM of the major axis of the elliptical Gaussian beam in *arcsec*.
- **BMIN** [none]. The FWHM of the minor minor axis of the elliptical Gaussian beam in *arcsec*.
- **BPA** [none]. The position angle of the major axis of the elliptical Gaussian beam in *degrees*, counter-clock from the North direction.

- **BEAMFWHM** [none]. The FWHM of a circular Gaussian beam in *arcsec* .

3.2 3DFIT task

3DFIT is the main BBarolo's routine: it fits a 3D tilted-ring model to an emission-line data-cube. Algorithms used are described in this paper.

3.2.1 Parameters

- **3DFIT** [false]. This flag enables the 3D fitting algorithm. Can be *true* or *false*. The old flag GALFIT is now deprecated and will be no more supported in future BBarolo's releases.

Rings input

Following parameters are used to define the initial set of rings used for the fit. All parameters are allowed to vary ring-by-ring or they can just be fixed to their initial value.

All parameters listed below (except NRADII and RADSEP) can be given in the form of a single value valid for all rings or through a text file containing values at different radii. In this second case, the syntax to be used is *file(filename,N,M)*, where *filename* is the name of the file with values, *N* is the column number (counting from 1) and *M* is the starting row (all rows if omitted). A subset of rows can also be selected using the syntax *file(filename,N,start:stop)*, where *start* and *stop* are the first and last row to be considered.

If any of the following parameters is not explicitly specified, BBarolo will estimate an appropriate initial value for that parameter.

- **NRADII** [none]. The number of rings to be used and fitted. If not given, BBarolo will determine it from the radial extension of the emission line.
- **RADSEP** [none]. The separation between rings in *arcsec*. If N radii have been requested, the rings will be placed at N*RADSEP + RADSEP/2. If not given, it will be equal to the FWHM of the beam major axis.
- **RADII** [none]. This parameter can be used as an alternative to NRADII and RADSEP. RADII can be 1) a text file (see above) with ring specified or 2) a list of radii or 3) a string in the format *Rmin~Rmax:step* (for example *60~180:60* will center rings at 60, 120 and 180 arcsec).
- **XPOS** [none]. X-center of rings. Accepted format are in *pixels* (starting from 0, unlike GIPSY) or in WCS coordinates in the format +000.0000d (*degrees*) or +00:00:00.00 (*sexagesimal*). If not specified, it is determined from the centroids of the emission.
- **YPOS** [none]. Like XPOS, but for the y-axis.
- **VSYS** [none]. Systemic velocity in *km/s*. If not given, it is estimated as the central velocity in the global line profile.
- **VROT** [none]. Rotation velocity in *km/s*.
- **VDISP** [8]. Velocity dispersion in *km/s*.
- **VRAD** [0]. Radial velocity in *km/s*.
- **INC** [none]. Inclination in *degrees*. If not given, it is guessed from the total map.
- **PA** [none]. Position angle in *degrees* of the receding side of the galaxy, measured anti-clockwise from the North direction. If not given, it is estimated from the velocity field.
- **Z0** [0]. Scale-height of the disc in *arcsec*.

- **DENS** [1]. Gas surface density in units of *1E20 atoms/cm2*. Fit of this parameter is not currently implemented and its value is not relevant if a normalization is used.

Main options

Some important parameters that can be used to control 3DFIT. All following parameters have default values and are therefore optional.

- **FREE** [VROT VDISP INC PA]. The list of parameters to fit. Can be any combination of VROT, VDISP, VRAD, VSYS, INC, PA, Z0, XPOS, YPOS.
- **MASK** [SEARCH]. This parameter tells the code how to build a mask to identify the regions of genuine galaxy emission. Accepted values are *SMOOTH*, *SEARCH*, *SMOOTH&SEARCH*, *THRESHOLD*, *NONE* or a FITS mask file:
  - *SMOOTH*: the input cube is smoothed according to the *smooth parameters* and the mask built from the region at S/N>BLANKCUT, where **BLANKCUT** is a parameter representing the S/N cut to apply in the smoothed datacube. Defaults are to smooth by a FACTOR = 2 and cut at BLANKCUT = 3.
  - *SEARCH*: the source finding is run and the largest detection used to determine the mask. The *source finding parameters* can be set to change the default values.
  - *SMOOTH&SEARCH*: first smooth to a lower resolution and then scan the smoothed data for sources. Parameters for smoothing and source-finding are the same as the *SMOOTH* and *SEARCH* tasks.
  - *THRESHOLD*: blank all pixels with flux < THRESHOLD. A **THRESHOLD** parameter must be specified in the same flux units of the input datacube.
  - *NONE*: all regions with flux > 0 are used.
  - *file(fitsname.fits)*: A mask FITS file (i.e. filled with 0s and 1s).
- **NORM** [AZIM]. Type of normalization of the model. Accepted values are: *LOCAL* (pixel by pixel), *AZIM* (azimuthal) or *NONE*.
- **TWOSTAGE** [true]. This flag enables the second fitting stage after parameter regularisation. This is relevant just if the user wishes to fit parameters other than VROT, VDISP and VRAD. The inclination and the position angle are regularised by polynomials of degree POLYN or a Bezier function (default), while the other parameters by constant functions.
- **REGTYPE** [auto]. Type of regularisation to use for second fitting stage. Accepted values are *auto* (the code will choose), *bezier* (Bezier interpolation), *median* (take the median), or a positive integer *n* for a nth-degree polynomial interpolation. It is possible to choose different types for INC, PA, VSYS, XPOS, YPOS and Z0 with a list of keyword-value pairs separated with whitespaces. A single value is used only to set INC and PA together. For example:

```
REGTYPE   bezier                  # Bezier function for INC and PA, 'auto' for␣
↪others
REGTYPE   INC=1 PA=median         # A line for INC, median for PA
REGTYPE   INC=2 PA=0 VSYS=bezier # A parabola for INC, a constant for PA, bezier␣
↪for VSYS
```

- **POLYN** [-1]. **DEPRECATED**. It will be discontinued after v1.6, use REGTYPE instead.
- **LINEAR** [0.85]. This parameter controls the spectral broadening of the instrument. It is in units of channel and it represents the standard deviation, not the FWHM. The default is for data that has been Hanning smoothed, so that FWHM = 2 channels and LINEAR = FWHM/2.355.
- **SIDE** [B]: Side of the galaxy to be fitted. Accepted values are: A = approaching, R = receding and B = both (default)

- **FLAGERRORS** [false]. Whether the code has to estimate the errors. This heavily slows down the run.
- **ADRIFT** [false]. If true, calculate the asymmetric drift correction. First regularize velocity dispersion and density profile and then compute the correction following classical prescription (see e.g. Iorio et al. 2017).
- **ADRIFTPOL1** [3]. Degree of polynomial function used to regularize the velocity dispersion in the computation of the asymmetric drift correction. If set to -1, it will not regularize it and just use the ring-by-ring best fit.
- **ADRIFTPOL2** [3]. Degree of polynomial function used to regularize the function log(VDISP**2*SIGMA) in the computation of the asymmetric drift correction.

Advanced options

Additional optional parameters to refine the fit for advanced users.

- **DELTAINC** [5]. This parameter fixes the boundaries of parameter space at [INC-DELTAINC, INC+DELTAINC]. It is not advisable to let the inclination varying over the whole range [0,90].
- **DELTAPA** [15]. This parameter fixes the boundaries of parameter space at [PA-DELTAINC, PA+DELTAPA]. It is not advisable to let the position angle varying over the whole range [0,360].
- **DELTAVROT** [inf]. This parameter fixes the boundaries of parameter space at [VROT-DELTAVROT, VROT+DELTAVROT]. Default is no limit.
- **MINVDISP** [0]. Minimum gas velocity dispersion allowed.
- **MAXVDISP** [1000]. Maximum gas velocity dispersion allowed.
- **FTYPE** [2]. Function to be minimized. Accepted values are: 1 = chi-squared, 2 = |mod-obs|, (default) and 3 = |mod-obs|/(mod+obs)).
- **WFUNC** [2]. Weighting function to be used in the fit. Accepted values are: 0 = uniform weight, 1 = |cos()| and 2 = cos()^2, default), where is the azimuthal angle (= 0 for galaxy major axis). Negative values can be used to set a sin() weight: -1 = |sin()| and -2 = sin()^2.
- **LTYPE** [1]. Layer type along z. Accepted values are: 1 = Gaussian (default), 2 = sech^2, 3 = exponential, 4 = Lorentzian and 5 = box.
- **CDENS** [10]. Surface density of clouds in the plane of the rings per area of a pixel in units of *1E20 atoms/cm^2* (see also GIPSY GALMOD).
- **NV** [nchan]. Number of subclouds in the velocity profile of a single cloud (see also GIPSY GALMOD). Default is the number of channels in the datacube.
- **BWEIGHT** [1]. Exponent of weight for blank pixels. See Section 2.4 of reference paper for details. Large numbers mean that models that extend further away than observations are severely discouraged.
- **STARTRAD** [0]. This parameter allows the user to start the fit from the given ring. Indexing from 0.
- **NOISERMS** [0] If > 0, Gaussian noise with rms = NOISERMS will be added to the final model cube.
- **NORMALCUBE** [true]. If true, the input cube is normalized before the fit. This usually helps convergence and avoids issues with very small flux values.
- **BADOUT** [false]. If true, it writes also unconverged/bad rings in the output ringfile (with a flag identifying them).
- **VELDEF** [AUTO]. Velocity definition to convert frequency/wavelength axis into velocity. Accepted values are *RADIO*, *OPTICAL* or *RELATIVISTIC*. If *AUTO* (default), the code will use radio definition if spectral axis is frequency and relativistic definition if spectral axis is wavelength.
- **PLOTMASK** [false]. If true, the mask contour is overlaid on the channel maps and PVs plots.

- **PLOTMINCON** [-1]. Minimum flux contour to use in channel map and PV plots. Must be positive. If not given or negative, the code will estimate an appropriate contour. Contour levels will be set at [1,2,4,8,...] x PLOTMINCON.

Parameters for high-z galaxies

For high-z galaxies the following additional parameters are available.

- **REDSHIFT** [0]. The redshift of the galaxy.
- **RESTWAVE** [none]. The rest wavelength of the line you want to fit, if the spectral axis of the data is wavelength. Units must be the same of the spectral axis of the cube. For example, if we want to fit the H-alpha line and CUNIT3 = "angstrom", set a value 6563. It can be a single value, or a list of values for fitting multiple lines at the same time.
- **RESTFREQ** [none]. The rest frequency of the line you want to fit, if the spectral axis of the data is frequency. Units must be the same of the spectral axis of the cube. The rest frequency value is often read from the FITS header and does not need to be explicitly set by the user. If set, the RESTFREQ value overrides the header value. It can be a single value, or a list of values for fitting multiple lines at the same time.

These parameters are used to calculate the conversion from wavelengths/frequencies to velocities. The velocity reference is set to 0 at RESTWAVE*(REDSHIFT+1) or RESTFREQ/(REDSHIFT+1). VSYS has to be set to 0, but can be also used to fine-tune the redshift. Finally, if these two parameters are not set, BBarolo will use the CRPIX3 as velocity reference and the proper VSYS has to be set based on that.

- **RELINT** [1]. A list of line ratios for multiple line fitting. The number of ratios must be the same of given **RESTFREQ** or **RESTWAVE**.

3.2.2 Outputs

The 3DFIT task produces several outputs to check the goodness of the fit. In the following *NAME* is the name of the galaxy and *NORM* is the kind normalization used.

- A FITS file *NAMEmod_NORM.fits*, containing the best-fit model datacube.
- A FITS file *mask.fits*, containing the mask used for the fit.
- FITS files of position-velocity cuts taken along the average major and minor axes for the data and the best-fit model. In particular:
  - *NAME_pv_a.fits*: P-V of the data along the major axis.
  - *NAME_pv_b.fits*: P-V of the data along the minor axis.
  - *NAMEmod_pv_a_NORM.fits*: P-V of the model along the major axis.
  - *NAMEmod_pv_b_NORM.fits*: P-V of the model along the minor axis.
- FITS files of the moment maps for the data and the model. These can be found in the *maps* subdirectory:
  - *NAME_0mom.fits*, *NAME_1mom.fits*, *NAME_2mom.fits:* 0th, 1st and 2nd moment maps of the data.
  - *NAME_NORM_0mom.fits*, *NAME_NORM_1mom.fits*, *NAME_NORM_2mom.fits:* 0th, 1st and 2nd moment maps of the model.
- Text files *rings_final1.txt* and *rings_final2.txt*, containing the ring best-fit parameters for the first and second fitting steps. The file *rings_final2.txt* is only produced if **TWOSTAGE** is true.
- A text file *densprof.txt*, with the radial intensity profiles along the best-fit rings.
- If **ADRIFT** is true, a text file *asymdrift.txt* with the asymmetric drift correction parameters.

- Plotting scripts to produce output plots with Gnuplot/Python can be found in the *plotscripts* subdirectory.
- A PDF file *NAME_chanmaps_NORM.pdf* with a channel-by-channel comparison of data and model cubes.

- A PDF file *NAME_pv_NORM.pdf* with a comparison of data and model P-Vs taken along the average major and minor axes.

- A PDF file *NAME_maps_NORM.pdf* with a comparison of data and model moment maps.

- A PDF file *NAME_parameters.pdf* with the best-fit parameters.

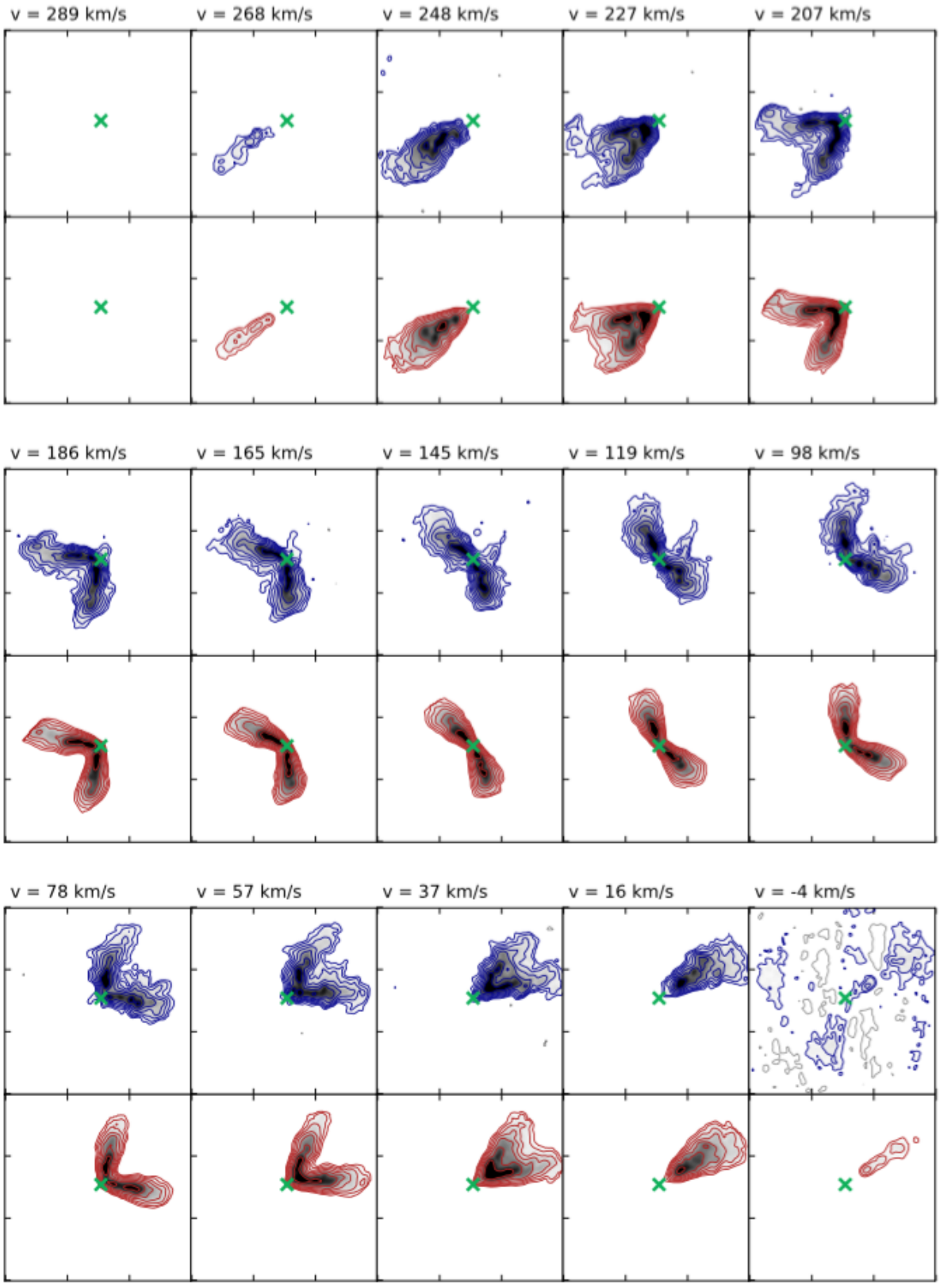
- If **ADRIFT** is true, a PDF file *asymmetricdrift.pdf* with the asymmetric drift correction.

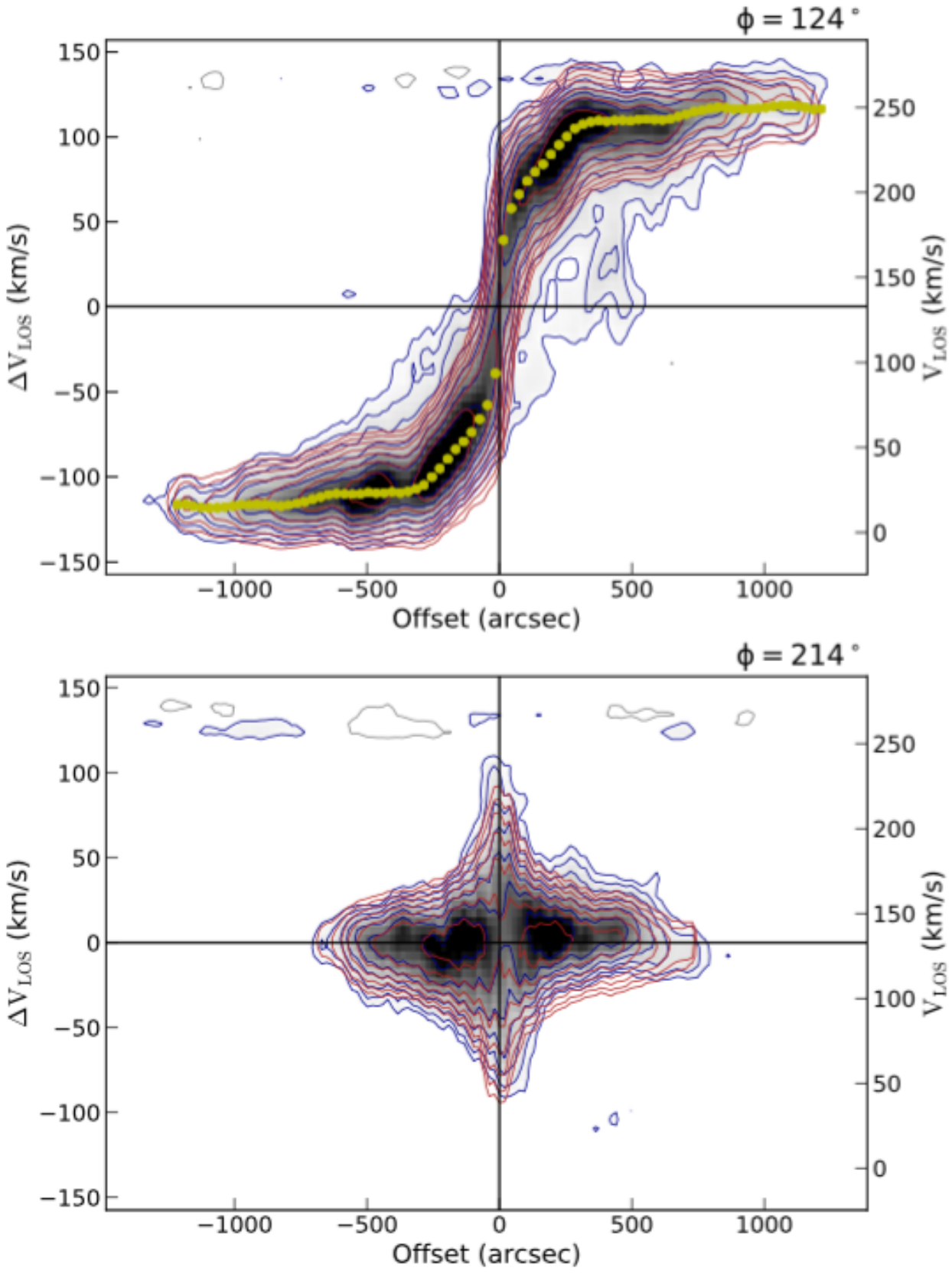
3.2.3 Example

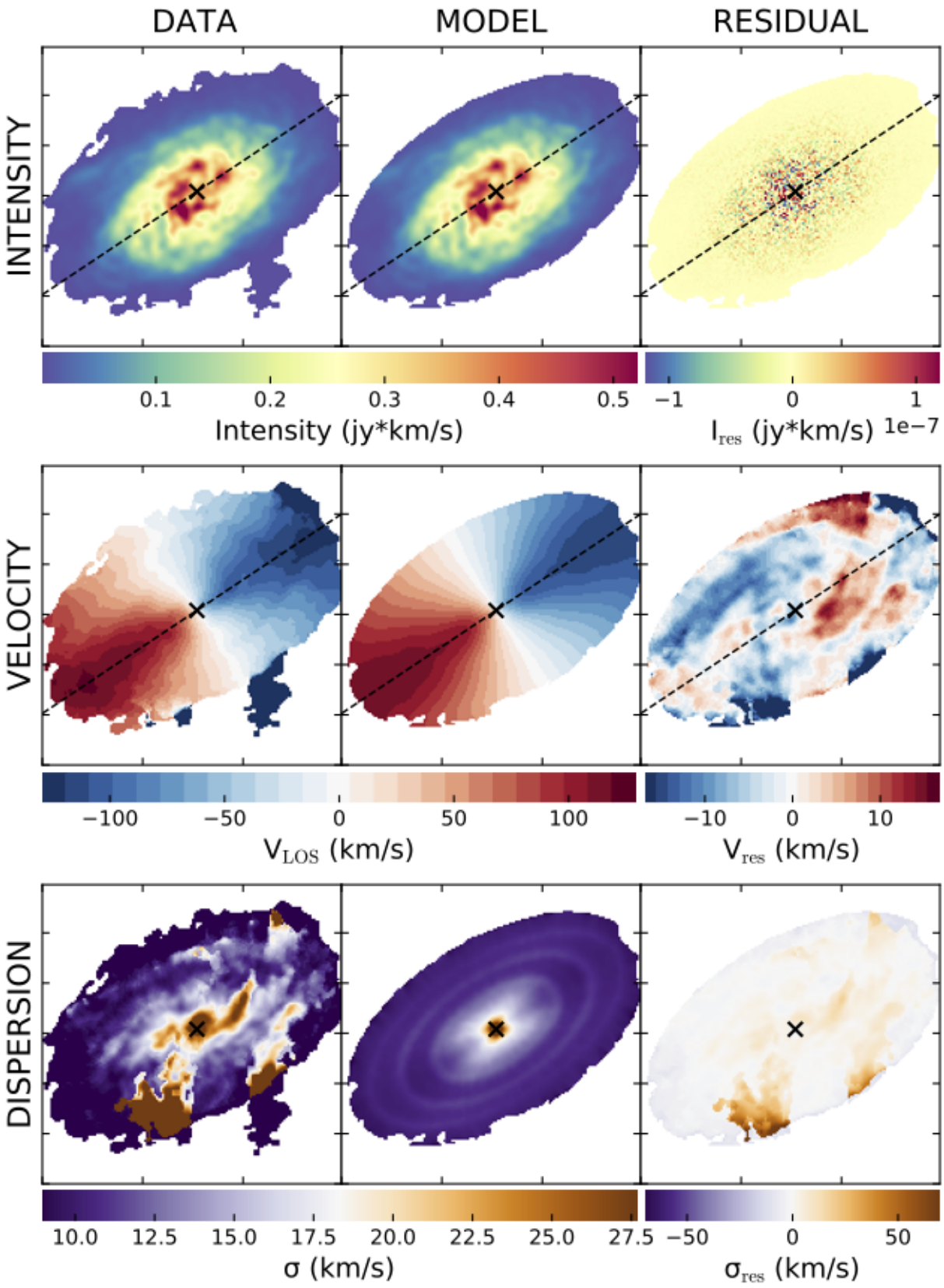
Above outputs can be obtained with the following `parameter` file and the usual example `datacube`.

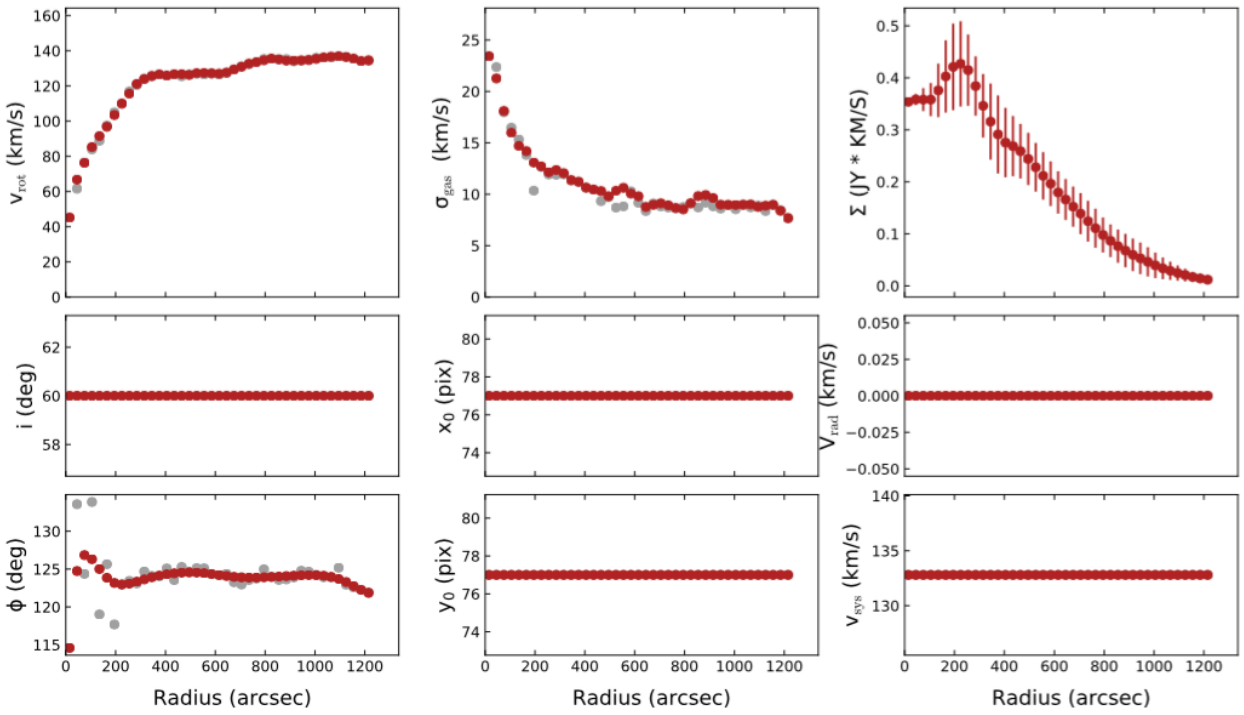
```
FITSFILE    ngc2403.fits
THREADS     4

////////// 3DFIT parameters ////////////
3DFIT       true
// Input rings
NRADII      41
RADSEP      30
VSYS        132.8
XPOS        77
YPOS        77
VROT        120
VDISP       8
INC         60
PA          123.7
Z0          10
// Free parameters
FREE        VROT VDISP PA
// Normalization type
```

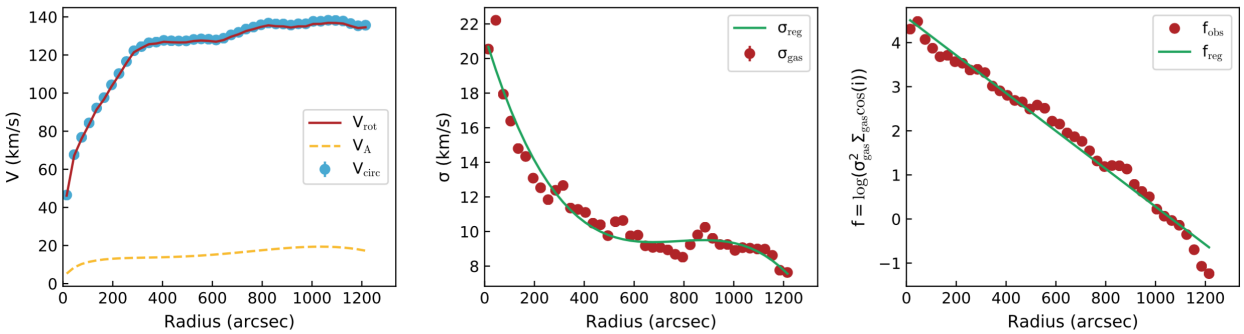
(continues on next page)


v = 289 km/s
v = 268 km/s
v = 248 km/s
v = 227 km/s
v = 207 km/s
v = 186 km/s
v = 165 km/s
v = 145 km/s
v = 119 km/s
v = 98 km/s
v = 78 km/s
v = 57 km/s
v = 37 km/s
v = 16 km/s
v = -4 km/s


ϕ = 124°
ΔV_LOS (km/s)
V_LOS (km/s)
Offset (arcsec)
ϕ = 214°
ΔV_LOS (km/s)
V_LOS (km/s)
Offset (arcsec)


DATA
MODEL
RESIDUAL
INTENSITY
0.1
0.2
0.3
0.4
0.5
Intensity (jy*km/s)
−1
0
1
I_{res} (jy*km/s) 1e−7
VELOCITY
−100
−50
0
50
100
V_{LOS} (km/s)
−10
0
10
V_{res} (km/s)
DISPERSION
10.0
12.5
15.0
17.5
20.0
22.5
25.0
27.5
σ (km/s)
−50
0
50
$σ_{res}$ (km/s)


v_rot (km/s)
σ_gas (km/s)
Σ (JY * KM/S)
i (deg)
x0 (pix)
V_rad (km/s)
ϕ (deg)
y0 (pix)
v_sys (km/s)
Radius (arcsec)
Radius (arcsec)
Radius (arcsec)


V (km/s)
V_rot
V_A
V_circ
σ (km/s)
σ_reg
σ_gas
f = log(σ²_gas Σ_gas cos(i))
f_obs
f_reg
Radius (arcsec)
Radius (arcsec)
Radius (arcsec)

(continued from previous page)

```
NORM        LOCAL
// Mask
MASK        SEARCH
// Other options
LTYPE       2
FTYPE       2
DISTANCE    3.2
BWEIGHT     1
WFUNC       2
TWOSTAGE    true
ADRIFT      true
////////////////////////////////////////
```

3.2.4 Guidelines for a successful fit

To obtain a good fit with very low resolution data, I usually follow some basic steps:

1. *Observational parameters*. Check that the header of your datacube has information on the PSF/beam of your observations. These are usually stored in the BMAJ, BMIN and BPA keywords. If these are not present in the header, you can use the *beam parameters* to specify them. Be sure also to set the correct spectral broadening with the LINEAR parameter. These infos are fundamental to properly account for observational biases.

2. *Obtain a mask*. A good mask is important for a nice fit because it tells the code which regions are real emission and which are just noise. The mask should be as large as possible to include faint emission, but not too large to include lots of noise. Try the different algorithms available in BBarolo and compare the produced masks with your data. When you find a mask you're happy with, keep that configuration.

3. *Initial guesses*. The parameter space can be quite degenerate and it's very important to provide the code with reliable initial guesses for parameters. In particular, the code is very sensitive to the initial value of the inclination angle. BBarolo comes with some algorithms to automatically estimate initial values when these are not explicitly given in the parameter file. However these algorithms are simple and may fail in a number of situations, depending also on the quality of your data. If you let the code estimating the initial values, always check that these make sense before going on with the fit. I usually prefer to provide my initial guesses. I extract the moment maps using the mask just obtained and use them to get the initial guesses. Center position and inclination (XPOS, YPOS, INC) can be obtained from the intensity map. If higher resolution observations are available (like HST), I would rather use them for the galaxy center and inclination. The velocity field can be used to estimate the kinematic position angle (PA). The midpoint of the global spectrum can be used as systemic velocity (VSYS). I stress that the code is quite good in estimating the VSYS, so it is quite safe to let it unset (but always check!). The initial values of rotation velocity and velocity dispersion are not very important, so you can give some random sensible value.

4. *Fit*. Depending on the data, you can decide to fit several parameters at the same time or keep some of them fixed. If your data have very low resolution, it may be wise to keep the geometry fixed and fit only the kinematics (VROT and VDISP). Check the outputs and if you are not happy with the model, try to change the initial parameters and/or the *fit options*.

3.2.5 Fitting several galaxies at the same time

An experimental function of BBarolo 1.5 allow the user to fit several galaxies at the same time. This can be useful, for example, when a large sample needs to be analysed on a supercluster. BBarolo launches a number of MPI processes and each process takes care of a galaxy at a time.

To use this function, you need to compile BBarolo with MPI:

```
> make mpi
```

If you have an MPI interface (OpenMPI, MPICH, etc. . .), this command will create an executable BBarolo_MPI in the working directory. You need to prepare a text file with a list of parameter files *params.list* and then run BBarolo_MPI through mpirun:

```
> mpirun -np NPROC BBarolo_MPI -l params.list
```

where NPROC is the number of MPI processes. Each MPI process can be also run in multi-thread mode with the usual **THREADS** parameter. This is basically the same of running NPROC instances of BBarolo, each with a single parameter file.

3.3 SPACEPAR task

SPACEPAR allows the user to explore the full parameter space for any pair of *3DFIT* parameters in a given range. It is useful to check that 3DFIT is converging to a good enough minimum of the parameter space. It can be very slow to run, but it is particularly recommended for low quality observations and small data-cubes.

3.3.1 Parameters

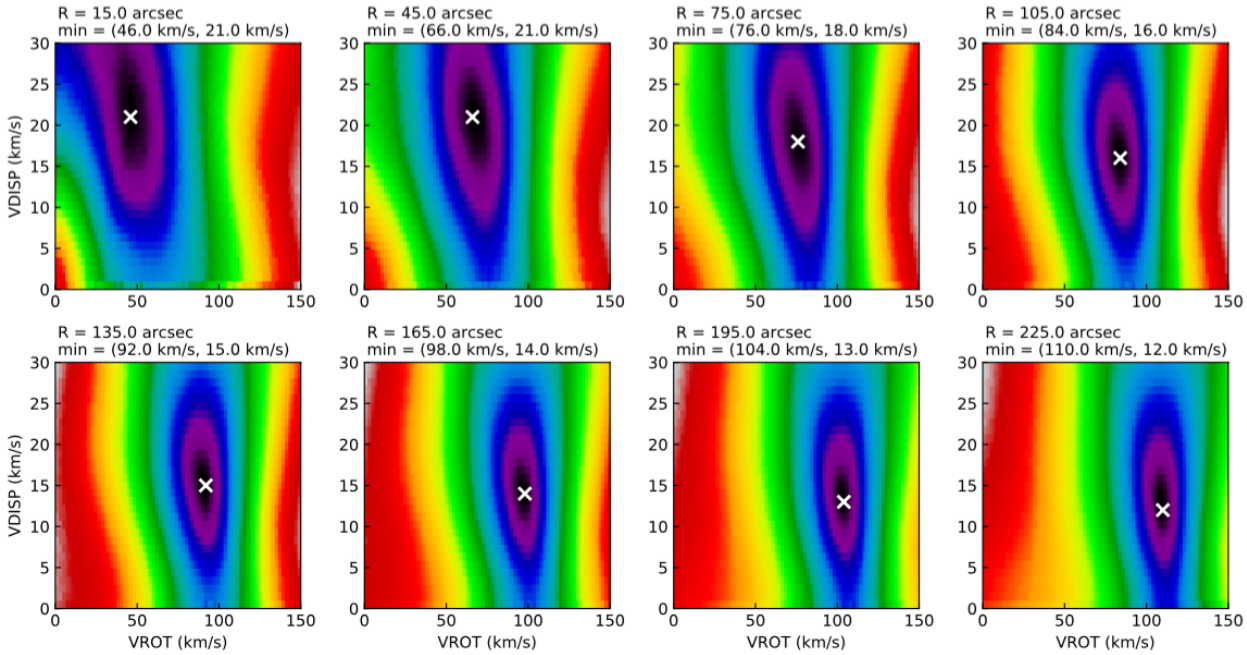
Basic parameters for SPACEPAR are the same of the *3DFIT task*. Additional specific parameters are:

- **SPACEPAR** [false]. This flag enables the SPACEPAR task. Can be *true* or *false*.
- **P1** [none]. First parameter to explore. Can be any of the 3DFIT **FREE** parameters.
- **P1PAR** [none]. A list of three numbers: the minimum value, the maximum value and the step size of the parameter space sampling for **P1**. For example, '0 100 1' samples between 0 and 100 with step size 1.
- **P2** [none]. Second parameter to explore. Can be any of the 3DFIT **FREE** parameters.
- **P2PAR** [none]. A list of three numbers: the minimum value, the maximum value and the step size of the parameter space sampling for **P2**. For example, '1 50 0.5' samples between 1 and 50 with step size 0.5.

3.3.2 Outputs

The task produces the following outputs:

- A FITS file *NAME_spacepar.fits*, containing the full parameter space for each ring.
- A FITS file *mask.fits*, containing the mask used.
- A Python script *spacepar.py* to conveniently plot parameter spaces.
- A PDF file *NAME_spacepar.pdf* with the plots of parameter spaces for each ring (Python required), produced through the *spacepar.py* script. Example below.



3.3.3 Example

Above outputs can be obtained with the following `parameter` file and the usual example `datacube`.

```
FITSFILE     ngc2403.fits
THREADS      1

////////////// SPACEPAR-specific //////////////
SPACEPAR     true
//First parameter
P1           VROT
//Second parameter
P2           VDISP
//First parameter range
P1PAR        0 150 2
//Second parameter range
P2PAR        0 30 1
////////////////////////////////////////////

// OTHER PARAMETERS IN COMMON WITH 3DFIT
NRADII       8
RADSEP       30
VSYS         132.8
XPOS         77
YPOS         77
VROT         120
VDISP        8
INC          60
PA           123.7
Z0           10
FREE         VROT VDISP
LTYPE        2
FTYPE        2
DISTANCE     3.2
```

(continues on next page)

(continued from previous page)

```
MASK        SEARCH
WFUNC       2
```

3.4 GALMOD task

GALMOD is the routine underlying the 3DFIT task. It builds a 3D simulated datacube of a disk galaxy starting from a set of concentric rings with given column density and kinematics. The routine is an updated version of the namesake routine in GIPSY (see also GIPSY GALMOD).

3.4.1 Parameters

Parameters for rings are the same of the *3DFIT task*. Options are LTYPE, CDENS, NV and VELDEF (see *3DFIT options*).

Additional GALMOD-specific parameters are:

- **GALMOD** [false]. This flag enables the 3D disk modelling. Can be *true* or *false*.
- **VVERT** [0]. Vertical velocity in *km/s*.
- **DVDZ** [0]. Gradient of rotation velocity as we move away from the disk plane. This is in *km/s/arcs*.
- **ZCYL** [0]. Height in *arcsec* from the disk plane where the gradient DVDZ begins.
- **SM** [true]. Whether to smooth the model to the same spatial resolution of data.

3.4.2 Outputs

The task produces the following outputs. Here *NAME* is the name of the galaxy and *NORM* is the kind normalization used.

- A FITS file *NAMEmod_NORM.fits*, containing the model datacube.
- A FITS file *mask.fits*, containing the mask used.
- FITS files of position-velocity cuts taken along the average major and minor axes for the input datacube and the model. In particular:
  - *NAME_pv_a.fits*: P-V of the data along the major axis.
  - *NAME_pv_b.fits*: P-V of the data along the minor axis.
  - *NAMEmod_pv_a_NORM.fits*: P-V of the model along the major axis.
  - *NAMEmod_pv_b_NORM.fits*: P-V of the model along the minor axis.
- FITS files of the moment maps for the input data and the model. These can be found in the *maps* subdirectory:
  - *NAME_0mom.fits*, *NAME_1mom.fits*, *NAME_2mom.fits:* 0th, 1st and 2nd moment maps of the data.
  - *NAME_NORM_0mom.fits*, *NAME_NORM_1mom.fits*, *NAME_NORM_2mom.fits:* 0th, 1st and 2nd moment maps of the model.
- A text file *densprof.txt*, with the radial intensity profiles along the chosen rings.

3.4.3 Example

Above outputs can be obtained with the following `parameter` file and the usual example `datacube`.

```
FITSFILE    ngc2403.fits

////////////// GALMOD parameters /////////////
GALMOD      true
// Input rings
NRADII      41
RADSEP      30
VSYS        132.8
XPOS        77
YPOS        77
VROT        120
VDISP       8
INC         60
PA          123.7
Z0          10
DENS        1
VRAD        10
// Normalization type
NORM        NONE
// Mask
MASK        SMOOTH
/////////////////////////////////////////
```

3.5 SEARCH task

BBarolo's search algorithm is derived from Duchamp, a 3D source finder for spectral-line data developed by Matthew Whiting. BBarolo adds a few new functionalities and parallelization. For a comprehensive description of the algorithm and the input parameters, see Duchamp's main paper and user guide.

3.5.1 Parameters

- **SEARCH** [false]. This flag enables the source finding algorithm. Can be *true* or *false*.
- **FLAGROBUSTSTATS** [true]. Whether to use to robust estimators (median and MADFM) instead of normal estimators (mean and standard deviation) when calculating cube statistics.
- **ITERNOISE** [false]. Whether to use an iterative algorithm to estimate the noise level. If true, it will reiterate over the array, masking pixels above 3sigma and re-calculating noise statistics until convergence.
- **SORTSOURCES** [NVOX]. This parameter specifies how to sort detections. Accepted values are *XVALUE*, *YVALUE*, *ZVALUE*, *RA*, *DEC*, *VEL*, *IFLUX* (integrated flux), *PFLUX* (peak flux), *NPIX* (number of pixels), *NVOX* (number of voxels), *W20* (width at 20% peak flux), *W50* (width at 50% peak flux) and *SNR* (average signal-to-noise). By default, sorting is descending, but it can be done ascending by adding a - (e.g. -*VEL*).
- **CUBELETS** [false]. If true, it produces individual cubelets and sub-images for each detected source.
- **SEARCHTYPE** [spatial]. How the search is performed. Accepted values are *spatial* and *spectral*. Spatial search is done in 2D channel maps, spectral search along 1D spectra.
- **SNRCUT** [5]. The primary S/N cut (number of above the mean/median).

- **THRESHOLD** [none]. Alternatively to SNRCUT, the primary threshold can be given in the same flux units of the input datacube. This overrides SNRCUT.
- **FLAGGROWTH** [true]. Whether to grow detected sources to a secondary threshold.
- **GROWTHCUT** [3]. Secondary S/N cut used when growing objects (number of above the mean/median).
- **GROWTHTHRESHOLD** [none]. Alternatively to GROWTHCUT, the secondary threshold can be given in the same flux units of the input datacube. This overrides GROWTHCUT.
- **MINPIX** [beam area]. The minimum number of spatial pixels for a detection to be accepted. Default is the area covered by the observational beam.
- **MINCHANNELS** [2]. The minimum number of channels for a detection to be accepted.
- **MINVOXELS** [none]. The minimum number of voxels for a detection to be accepted. If not set, MINVOXELS = MINPIX*MINCHANNELS.
- **MAXCHANNELS** [none]. The maximum number of channels for a detection to be accepted. Default is no limits.
- **MAXANGSIZE** [none]. The maximum angular size of a detection to be accepted in *arcmin*. Default is no limits.
- **FLAGADJACENT** [true]. Whether to use the adjacent criterion to merge objects. If *false*, the next two parameters are used to determine whether objects are to be merged.
- **THRESHSPATIAL** [2]. The maximum minimum spatial separation in *pixels* for two objects to be merged into a single one. Ignored if FLAGADJACENT is *true*.
- **THRESHVELOCITY** [3]. The maximum minimum channel separation in *channels* for two objects to be merged into a single one. Ignored if FLAGADJACENT is *true*.
- **REJECTBEFOREMERGE** [true]. Whether to reject sources before merging them.
- **TWOSTAGEMERGING** [true]. Whether to do a partial merge during search.

3.5.2 Outputs

The task produces the following outputs. Here *NAME* is the name of the galaxy.

- A FITS file *detections.fits*, a datacube with just the detections.
- A FITS file *DetectMap.fits*, containing a 2D map telling how many channels are detected in each spaxel.
- FITS files of the moment maps for the detections (*NAME_mom0th.fits*, *NAME_mom1st.fits*, *NAME_mom2nd.fits*).
- A text file *detections.txt*, containinig a list of detections and their properties.

3.5.3 Example

With the following `parameter` file, the `datacube` of NGC2403 is searched. Unsurprisingly, the galaxy is detected...

```
FITSFILE    ngc2403.fits
THREADS     4

////////////// SEARCH parameters //////////////
SEARCH      true
// Searching map by map
```

(continues on next page)

(continued from previous page)

```
SEARCHTYPE  spatial
// Primary and secondary cut
SNRCUT      10
GROWTHCUT   3
// Setting some criteria for rejection
MINPIX      30
MINCHANNELS 5
////////////////////////////////////////
```

3.6 SMOOTH task

This task convolves each channel map in a datacube with a given elliptical Gaussian.

3.6.1 Parameters

- **SMOOTH** [false]. This flag enables the smooth algorithm. Can be *true* or *false*.
- **OBMAJ** [none]. Major axis of the initial beam in *arcsec*. Do not set if you want to use the beam information in the input FITS file (the parameter overrides it).
- **OBMIN** [none]. Minor axis of the initial beam in *arcsec*. Do not set if you want to use the beam information in the input FITS file (the parameter overrides it).
- **OBPA** [none]. Position angle of the major axis of the initial beam in *degrees*. Do not set if you want to use the beam information in the input FITS file (the parameter overrides it).
- **BMAJ** [none]. Major axis of the final beam in *arcsec*.
- **BMIN** [none]. Minor axis of the final beam in *arcsec*.
- **BPA** [none]. Position angle of the major axis of the final beam in *degrees*.
- **FACTOR** [2]. If set, the beam of the output cube is [FACTOR*OBMAJ,FACTOR*OBMIN,OBPA]. Ignored if BMAJ, BMIN, BPA are specified.
- **SCALEFACTOR** [none]. Scaling factor for output datacube. BBarolo will calculate an appropriate one if left unset.
- **FFT** [true]. Whether to convolve by using Fast Fourier Transform or not.
- **REDUCE** [false]. If *true*, BBarolo repixels the output datacube to preserve the number of pixels in a beam.
- **SMOOTHOUTPUT** [none]. Output smoothed FITS file. Default is input file name with a suffix indicating the new beam size.

3.6.2 Outputs

The task writes the smoothed datacubes in the FITS file *NAME_sN.fits*, where *NAME* is the name of the galaxy and *N* is the new beam size.

3.6.3 Example

Below, an example `parameter` file to smooth the usual `datacube` to a coarser spatial resolution.

```
FITSFILE    ngc2403.fits
THREADS     4

////////////// SMOOTH parameters //////////////
SMOOTH      true
// New beam parameters.
BMAJ        150
BMIN        150
BPA         0
// Repixeling
REDUCE      true
/////////////////////////////////////////////
```

3.7 SMOOTHSPEC task

This task convolves each spectrum in a datacube with a given window, i.e. it performs spectral smoothing.

3.7.1 Parameters

- **SMOOTHSPEC** [false]. This flag enables the spectral smoothing algorithm. Can be *true* or *false*.
- **WINDOW_TYPE** [HANNING]. Type of the smoothing window. Implemented windows include *HANNING*, *BOXCAR*, *BARTLETT*, *WELCH*, *BLACKMAN*, *FLATTOP*.
- **WINDOW_SIZE** [3]. Size of the smoothing window in channels.
- **REDUCE** [false]. Whether to do channel resampling. If *true*, output data will be averaged over (WINDOW_SIZE+1)/2 channels.

3.7.2 Outputs

The task writes the smoothed datacube in the FITS file *NAME_hN.fits*, where *NAME* is the name of the galaxy and *N* is the size of the Hanning window.

3.7.3 Example

Below, a `parameter` file to Hanning smooth the usual `datacube`.

```
FITSFILE     ngc2403.fits

////////////// SPECSMOOTH parameters //////////////
SMOOTHSPEC   true
WINDOW_TYPE  HANNING
WINDOW_SIZE  11
/////////////////////////////////////////////////
```

3.8 2DFIT task

The classical 2D tilted-ring modelling of a galaxy: a model velocity field is fitted to the observed velocity field (see, e.g., Begeman 1987). This technique is fast and good for high spatial resolution data, but completely unreliable for low resolution data (no beam smearing correction).

3.8.1 Parameters

- **2DFIT** [false]. This flag enables the 2D fitting of the velocity field.

Parameters and options that control the task are in common with *3DFIT*. In particular, 2DFIT supports the following parameters: **NRADII**, **RADSEP**, **XPOS**, **YPOS**, **VSYS**, **VROT**, **VRAD**, **PA**, **INC**, **FREE**, **SIDE**, **WFUNC**. If **FITSFILE** is a datacube, the velocity field to fit is extracted as 1st moment using a mask for the input datacube defined by the **MASK** parameter (written in the output directory). If **FITSFILE** is a 2D velocity map, this is used to fit the tilted-ring model.

3.8.2 Outputs

The task produces the following outputs. *NAME* is the name of the galaxy.

- A FITS file *NAMEmap_1st.fits* with the velocity field used for the fit.
- A FITS file *NAMEmao_2d_mod.fits*, containing the model velocity field.
- A text file *NAME_2dtrm.txt*, with the best-fit parameters ring-by-ring.

3.8.3 Example

The following example `parameter` fits a 2D tilted-ring model to the usual NGC2403 `datacube`.

```
FITSFILE    ngc2403.fits

////////////// 2DFIT parameters //////////////
2DFIT       true
// Input rings
NRADII      41
RADSEP      30
VSYS        132.8
XPOS        77
YPOS        77
VROT        120
INC         60
PA          123.7
// Free parameters
FREE        VROT INC PA
// Mask for extracting velocity field
MASK        SEARCH
// Which side to fit
SIDE        B
// Weighting function
WFUNC       2
////////////////////////////////////////////
```

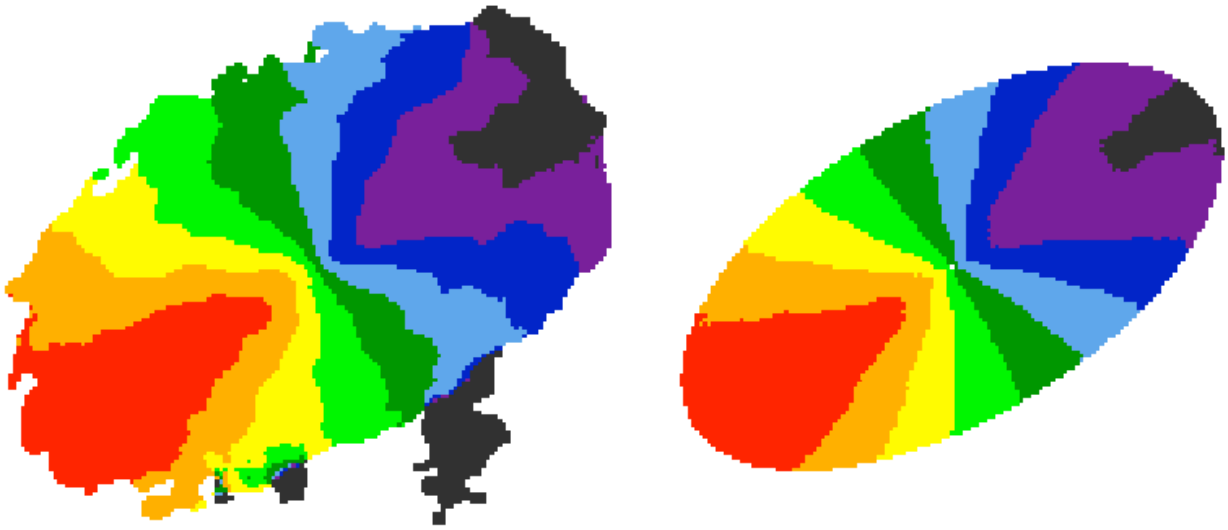


Fig. 1: Observed (left) and modelled (right) velocity field for NGC 2403.

3.9 ELLPROF task

This task can be used to calculate the radial density profile of a galaxy.

3.9.1 Parameters

- **ELLPROF** [false]. This flag enables the radial profile task.

Parameters for the task are: **RADII**, **NRADII**, **RADSEP**, **XPOS**, **YPOS**, **PA**, **INC**, **SIDE** (see *3DFIT*). If **FITSFILE** is a datacube, the profile is calculated from the column density map calculated after masking the cube accordingly to the **MASK** parameter. If **FITSFILE** is a 2D intensity map, this is used to extract the profile.

3.9.2 Outputs

The task produces the following outputs. *NAME* is the name of the galaxy.

- A FITS file *NAME_densmap.fits* with the map used to extract the profile.
- A text file *NAME_densprof.txt*, containing with the radial profiles.

3.9.3 Example

An example `parameter` file to extract the radial profile from the usual NGC2403 `datacube`, given a set of rings.

```
FITSFILE    ngc2403.fits

////////////// ELLPROF parameters /////////////
ELLPROF       true
// Input rings
NRADII      41
```

(continues on next page)

(continued from previous page)

```
RADSEP       30
XPOS         77
YPOS         77
INC          60
PA           123.7
// Mask for extracting intensity map
MASK         SEARCH
// Which side of the galaxy
SIDE         B
////////////////////////////////////////
```

3.10 Moment maps and position-velocity cuts

BBarolo can be used to extract global profiles, moment maps and position velocity diagrams. For moment maps and profile, the input datacube can be masked using the MASK parameter (see *3DFIT*).

3.10.1 Parameters for maps

- **GLOBALPROFILE** [false]. If *true*, calculate the total line profile from a datacube and write it to a text file.
- **TOTALMAP** [false]. If *true*, calculate the total intensity map from a datacube and write it to a FITS file.
- **VELOCITYMAP** [false]. If *true*, calculate the velocity field from a datacube and write it to a FITS file. If spectral axis in the datacube is frequency/wavelength, the velocity definition can be chosen through the **VELDEF** parameter (see *3DFIT advanced options*).
- **DISPERSIONMAP** [false]. If *true*, calculate the velocity dispersion field from a datacube and write it to a FITS file.
- **MAPTYPE** [MOMENT]. It specifies the way the kinematic maps are derived. Can be either *MOMENT* (classical moments) or *GAUSSIAN* (gaussian fit).
- **RMSMAP** [false]. If *true*, calculate the RMS map, i.e. the RMS in each spectrum, from a datacube and write it to a FITS file. The RMS is calculated in an iterative way. RMS is the standard deviation for normal statistics and MADFM/0.6745 for robust statistics (**FLAGROBUSTSTATS** parameter).
- **MASSDENSMAP** [false]. If *true*, calculate a mass surface-density map in units of Msun/pc^2 from a datacube and write it to a FITS file. This is just for HI data and the input datacube is required to have JY/BEAM flux density units.
- **SNMAP** [false]. If *true*, calculate a signal-to-noise map for the masked total map, following the prescriptions by Verheijen & Sancisi (2001) and Lelli et al. (2014) (see their appendixes). It has to be combined with **TOTALMAP** = *true*. The noise and S/N maps are written in individual FITS files.
- **CONTCHANS** [1E06 1E06]. Number of line-free channels at the low/high velocity ends of the data that have been used to calculate and subtract the continuum. If a single number is given, it will assume the same number of channels have been used at both velocity ends. If no continuum subtraction has been done, use large numbers (default). Only relevant if **SNMAP** is *true*.
- **TAPER** [UNIFORM]. Type of online tapering used during data acquisition. Accepted values are *UNIFORM* (uniform taper), *HANNING1* (hanning taper with all channels kept) and *HANNING2* (hanning taper with half channels thrown away). Only relevant if **SNMAP** is *true*.

3.10.2 Parameters for PV slices

- **PVSLICE** [false]. If *true*, extract a position-velocity image from a datacube and write it to a FITS file. The slice can be defined by either a point and an angle (see **XPOS_PV**, **YPOS_PV**, **PA_PV** below), or by two points (see **P1_PV**, **P2_PV** below). The former has priority over the latter.
- **XPOS_PV** [none]. Reference X of the slice. Can be a pixel or a coordinate (see also **XPOS** for *3DFIT*).
- **YPOS_PV** [none]. Reference X of the slice. Can be a pixel or a coordinate (see also **YPOS** for *3DFIT*).
- **P1_PV** [none]. X and Y coordinates of the first point defining a slice. For example: '10 30' for a pixel at X=10 and Y=30.
- **P2_PV** [none]. X and Y coordinates of the second point defining a slice.
- **WIDTH_PV** [0]. Width of the slice in arcsec. The PV cut will be averaged over a rectangular window extending from -width/2 to +width/2 from the slice defined above.
- **ANTIALIAS** [0.5]. It defines how many pixels to use for antialiasing algorithm. It can be an integer or 0.5. The number of pixels used for antialiasing will be (1+2*ANTIALIAS)^2. Set it to 0 for no antialiasing.

3.10.3 Outputs

The required map/P-V is written in a FITS file.

3.10.4 Example

The following `parameter` file extract maps and PV diagrams from the usual `datacube`.

```
FITSFILE      ngc2403.fits
THREADS       4
// Using a threshold mask
MASK          THRESHOLD
THRESHOLD     0.01
// Extracting all maps
TOTALMAP      true
VELOCITYMAP   true
DISPERSIONMAP true
RMSMAP        true
// Extract PV along major axis
PVSLICE       true
XPOS_PV       77
YPOS_PV       77
PA_PV         123
```

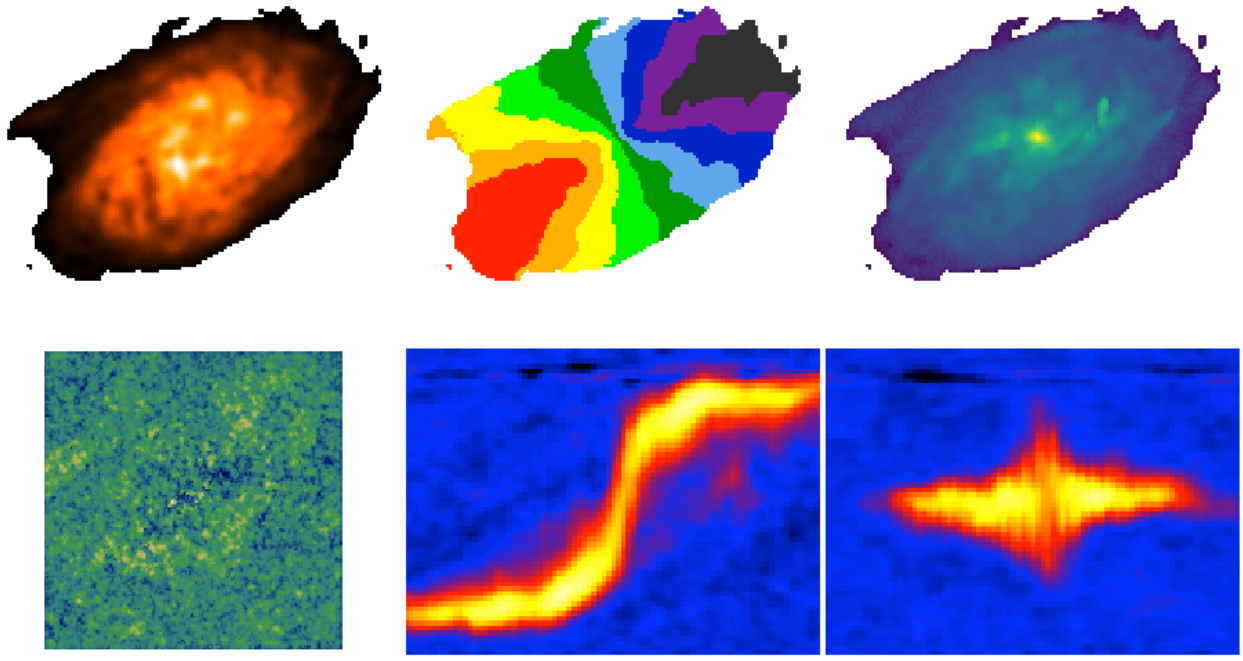


Fig. 2: Moment maps, RMS map and P-Vs along the major and minor axis

CHAPTER

FOUR

TOOLS FOR FITS FILES

Sometimes a FITS file does not conform with BBarolo or keywords in the FITS header can not be understood by the code. A few utilities to easily modify a FITS file and its header are now included in the latest versions of BBarolo. These tools are based on or developed from the NASA's FITS utilities.

A list of implemented tools can be obtained with `BBarolo --fitsutils`. Four utilities are currently available. Each tool is promptly accessible from the command line through `BBarolo --UTILNAME`, where UTILNAME is one of the utilities listed below:

modhead. Modify or add a keyword to a FITS header. Type `BBarolo --modhead` for help:

```
$$ BBarolo --modhead

 BBarolo's MODHEAD FITS utility:

 Write or modify the value of a header keyword.
 If a newvalue is not specified, just print the current value.

 Usage:
   BBarolo --modhead filename[ext] keyword [newvalue]

 Examples:
   BBarolo --modhead in.fits cunit1      (list the CUNIT1 keyword)
   BBarolo --modhead in.fits cunit1 deg  (set CUNIT1 = 'deg')

 NOTE: it may be necessary to enclose the input file name in single
 quote characters on some Unix shells.
```

remhead. Remove a keyword from a FITS header. Type `BBarolo --remhead` for help:

```
$$ BBarolo --remhead

 BBarolo's REMHEAD FITS utility:

 Delete a header keyword from a FITS file.

 Usage:
   BBarolo --remhead filename[ext] keyword

 Examples:
```

(continues on next page)

(continued from previous page)

```
  BBarolo --remhead in.fits object      (remove the OBJECT keyword)

NOTE: it may be necessary to enclose the input file name in single
quote characters on some Unix shells.
```

listhead. List all keywords in a FITS header. Type `BBarolo --listhead` for help.

```
$$ BBarolo --listhead

 BBarolo's LISTHEAD FITS utility:

 List the FITS header keywords in a single extension, or, if [ext] is
 not given, list the keywords in all the extensions.

 Usage:
   BBarolo --listhead filename[ext]

 Examples:
   BBarolo --listhead file.fits      (list every header in the file)
   BBarolo --listhead file.fits[0]   (list primary array header)
   BBarolo --listhead file.fits[2]   (list header of 2nd extension)
   BBarolo --listhead file.fits[GTI] (list header of GTI extension)

 NOTE: it may be necessary to enclose the input file name in single
 quote characters on some Unix shells.
```

fitscopy. Make a copy of a FITS file, optionally selecting a subset. Type `BBarolo --fitscopy` for help:

```
$$ BBarolo --fitscopy

 BBarolo's FITSCOPY FITS utility:

 Copy an input file to an output file, optionally filtering the file in the
 process. Filters may be used to extract a subimage from a larger image,
 select rows from a table, filter a table with a GTI time extension or a SAO
 region file, create or delete columns in a table, create an image by binning
 2 table columns, and convert IRAF format .imh or raw binary data files into
 FITS images (see CFITSIO User's Guide for filtering syntax).

 Usage:
   BBarolo --fitscopy inputfile[filter] outputfile

 Examples:
   BBarolo --fitscopy in.fits out.fits                  (simple file copy)
   BBarolo --fitscopy in.fits[11:50,21:60] out.fits     (copy a subimage)
   BBarolo --fitscopy in.fits[-*,*] out.fits            (mirror reverse axis 0)
   BBarolo --fitscopy iniraf.imh out.fits               (IRAF image to FITS)
```

(continues on next page)

(continued from previous page)

```
  BBarolo --fitscopy in.dat[i512,512] out.fit           (binary file to FITS)
  BBarolo --fitscopy in.fits[events][pi>35] out.fits    (copy rows with pi>35)
  BBarolo --fitscopy in.fits[events][bin X,Y] out.fits  (bin an image)

NOTE: it may be necessary to enclose the input file name in single quote
characters on some Unix shells.
```

fitsarith. Perform arithmetic operations between two FITS files or on a single FITS file. Type `BBarolo --fitsarith` for help:

```
$$ BBarolo --fitsarith

 BBarolo's FITSARITH FITS utility:

 Perform an operation between two images or between an image and a number.
 Supported operators are add, sub, mul, div (first character required).

 Usage:
   BBarolo --fitsarith image1 { image2 | value } oper outimage

 Examples:
   BBarolo --fitsarith in1.fits in2.fits add out.fits  (add the 2 files)
   BBarolo --fitsarith in1.fits 1000.0 mul out.fits    (mult in1 by 1000)

 NOTE: it may be necessary to enclose the file names in single quote
 characters on some Unix shells.
```

CHAPTER

FIVE

INSTALLING PYBBAROLO

pyBBarolo is a python wrapper to the BBarolo main code. It is compatible with both Python 2 (>2.6) and with Python 3. It can be easily installed via either pip or from the main repository.

5.1 From pip

pyBBarolo is available as a package in the Python Package Index (PyPI). The easiest way of installing it is through `pip`:

```
> pip install pyBBarolo
```

This will download the package, compile BBarolo source code and install pyBBarolo in the python library path. Make sure you have permissions to write in the installing directory. Depending on your computer setup, you may need to run pip with superuser privileges (e.g.: `> sudo pip install pyBBarolo`).

N.B.: The above command will compile BBarolo, which means that your machine needs to have pre-installed all the libraries which the C++ code depends from (see *requirements*). If pip fails during compilation, please follow the procedure below.

5.2 Build and install

The python package can be alternatively installed from the main repository. You'll need to compile BBarolo as a library and then install pyBBarolo:

1. Follow steps 1-4 of procedure to *compile BBarolo from source*:

   ```
   > wget https://github.com/editeodoro/Bbarolo/archive/X.Y.tar.gz
   > tar -xvf X.Y.tar.gz && cd Bbarolo-X.Y
   > ./configure
   > make
   ```

   where *X.Y* is the software release. PyBBarolo is only available for BBarolo's release 1.4 and over.

2. Install the python package:

   ```
   > python setup.py install
   ```

If either compilation or installation fail, refer to BBarolo *compiling* and *troubleshooting* pages.

CHAPTER

SIX

QUICKSTART

6.1 Running a task

In pyBBarolo, BBarolo's task are wrapped as python classes. The generic procedure to run a task is as follow:

1. Import the task from pyBBarolo module:

   ```
   from pyBBarolo import Task
   ```

   where Task is one of pyBBarolo *tasks*.

2. Create an object of the task class:

   ```
   bb = Task(fitsname)
   ```

   where *fitsname* is the input FITS file. All tasks need an initial FITS file.

3. Initialize the task:

   ```
   bb.init(args)
   ```

   where *args* are required arguments that depend on the various tasks. Required arguments can be printed with:

   ```
   bb.show_arguments()
   ```

4. Set options:

   ```
   bb.set_options(opts)
   ```

   where *opts* are the task-dependent available options. All options have default values, so this step is not mandatory. To see a list of available options:

   ```
   bb.show_options()
   ```

5. Run the task:

   ```
   bb.compute()
   ```

6.2 Available tasks

- **FitMod3D()**: wrapped class for BBarolo's *3DFIT* task.

- **GalMod**(): wrapped class for BBarolo's *GALMOD* task.
- **Search**(): wrapped class for BBarolo's *SEARCH* task.
- **FitMod2D**(): wrapped class for BBarolo's *2DFIT* task.
- **Ellprof**(): wrapped class for BBarolo's *ELLPROF* task.
- **SpectralSmooth**(): wrapped class for BBarolo's *SMOOTHSPEC* task.

6.3 Example 1: 3D fit of a galaxy

Suppose you have an astonishing observation of your favorite galaxy and you want to fit a 3D kinematic model to your emission line datacube.

Let's fit the HI datacube of NGC 2403, which is available as a part of BBarolo's working examples. We just have to set initial conditions, options and then run the task.

First of all, we import and start FitMod3D:

```
from pyBBarolo import FitMod3D

# FITS file of the galaxy to model
filen = "./examples/ngc2403.fits"
# Initializing a 3DFIT object
f3d = FitMod3D(filen)
```

Secondly, we initialize rings with initial guesses for the fit:

```
# Initializing rings. Parameters can be values or arrays
f3d.init(radii=np.arange(15,1200,30),xpos=77,ypos=77,vsys=132.8,vrot=120,vdisp=8,
↪vrad=0,z0=10,inc=60,phi=123.7)
```

A list of needed arguments can be printed with `f3d.show_arguments()`.

Thirdly, we can change some default options for the fit. For a list of available options: `f3d.show_options()`.

For instance, we can set a mask made through the source-finding algorithm (`mask="SEARCH"`), parameters to fit (`free="VROT VDISP"`), the distance of the galaxy in Mpc (`distance=3.2`) and the directory for outputs (`outfolder='output/ngc2403`):

```
f3d.set_options(mask="SEARCH",free="VROT VDISP",wfunc=2,distance=3.2,outfolder=
↪'output/ngc2403')
```

If the beam information is not available in the FITS header, it is fundamental to set the size of the beam:

```
f3d.set_beam(bmaj=60,bmin=60,bpa=-80)
```

It is now time to run the fit:

```
bfrings, bestmod = f3d.compute(threads=4)
```

This function performs the fit and writes relevant FITS files in the output directory. The function returns a n x m matrix containing the best-fit rings (`bfrings`), where n = number of rings and m = number of parameters, and a FITS astropy object (`bestmod`) containing the best-fit model. These are also written in `bfit` and `outmodel` methods of FitMod3D class.

Finally, we can use BBarolo built-in routines to write plots of data and model, like channel maps, moment maps, position-velocity diagrams and best-fit parameters:

```
f3d.plot_model()
```

6.4 Example 2: 3D model of a galaxy

It is also possible to simply build a 3D model datacube from given parameters. This is accomplished with the GalMod task. The procedure is similar to the one above:

```
from pyBBarolo import GalMod

# FITS file of the galaxy to model
filen = "./examples/ngc2403.fits"
# Initializing a GalMod object
gm = GalMod(filen)
# Initializing rings. Parameters can be values or arrays
gm.init(radii=np.arange(15,1200,30),xpos=74,ypos=74,vsys=132.8,vrot=120,vrad=10,
↪vvert=5,vdisp=8,z0=10,inc=60,phi=123.7)
# Now, let's take a look to the default options (see BB documentation)
gm.show_options()
# Changing some options
gm.set_options(ltype=1)
# Compute the model
mymodel = gm.compute()
# Smooth to the same resolution of data
mymodel = gm.smooth()
# mymodel is an astropy cube and we can do whatever we like with it.
mymodel.writeto("awesome_model.fits",overwrite=True)
```

6.5 Example 3: All tasks

This `script` shows how to use all wrapped classes:

```
import os
import numpy as np
from pyBBarolo import *

if __name__ == '__main__':

    fitsname = "./examples/ngc2403.fits"

    #'''
    ### 3D FIT TUTORIAL ###################################################################
    f3d = FitMod3D(fitsname)
    f3d.init(radii=np.arange(15,450,30),xpos=77,ypos=77,vsys=132.8,\
             vrot=120,vdisp=8,vrad=0,z0=10,inc=60,phi=123.7)
    f3d.show_options()
    f3d.set_options(mask="SEARCH",free="VROT VDISP",wfunc=2,distance=3.2,ltype=2)
    f3d.set_options(outfolder="output/ngc2403")
    f3d.set_beam(bmaj=60,bmin=60,bpa=-80)
    bfrings, bestmod = f3d.compute(threads=4)
    f3d.plot_model()
    ######################################################################################
    #'''
```

(continues on next page)

(continued from previous page)

```
    #'''
    ### GALMOD TUTORIAL ###################################################################
    gm = GalMod(fitsname)
    gm.show_arguments()
    gm.init(radii=np.arange(15,1200,30),xpos=74,ypos=74,vsys=132.8,\
             vrot=120,vdisp=8,z0=10,inc=60,phi=123.7)
    gm.show_options()
    gm.set_options(ltype=1)
    mymodel = gm.compute()
    mymodel = gm.smooth()
    mymodel.writeto("awesome_model.fits",overwrite=True)
    ######################################################################################
    #'''

    #'''
    ### ELLPROF TUTORIAL ##################################################################
    el = Ellprof(fitsname)
    el.show_arguments()
    el.init(radii=np.arange(15,1200,30),xpos=74,ypos=74,inc=60,phi=123.7)
    el.show_options()
    el.set_options(mask="SMOOTH")
    rings = el.compute()
    print (rings['rad'],rings['msurfdens'])
    el.writeto("rings_ellprof.txt")
    ######################################################################################
    #'''

    #'''
    ### 2DFIT TUTORIAL ####################################################################
    rm = FitMod2D(fitsname)
    rm.show_arguments()
    rm.init(radii=np.arange(15,1200,30),xpos=74,ypos=74,vsys=132.8,vrot=120,inc=60,
↪phi=123.7)
    rm.show_options()
    rm.set_options(wfunc=1, free="VROT INC")
    rings = rm.compute(threads=1)
    print (rings['rad'],rings['vrot'])
    rm.writeto("rings_2dfit.txt")
    ######################################################################################
    #'''

    #'''
    ### SEARCH TUTORIAL ###################################################################
    s = Search(fitsname)
    s.set_options(growth=True)
    s.show_options()
    s.search(threads=4)
    ######################################################################################
    #'''

    #'''
    ### SPECTRAL SMOOTHING TUTORIAL #######################################################
    s = SpectralSmoothing(fitsname)
    a = s.smooth(window_type='hanning',window_size=11,threads=8)
    s.writeto("smoothedcube.fits",average=True)
    ######################################################################################
```

(continues on next page)

(continued from previous page)

```
    #'''
```

6.6 pyBBarolo for PROs

Beside the python classes described above, there is probably a better way to run BBarolo from python for users who are familiar with BBarolo's parameter files. A class `BBaroloWrapper` to call conveniently BBarolo C++ executable is implemented in the `pyBBarolo.wrapper` module. This class allows the user to set any parameter supported in BBarolo and it is extremely useful for scripting.

The `BBaroloWrapper` class takes in input a list of BBarolo's parameters, which can be given as either a python dictionary or a list of strings. The class is basically a container of parameters. A working example is below:

```
from pyBBarolo.wrapper import BBaroloWrapper

# Initializing class (doing some spatial smoothing for example)
bb = BBaroloWrapper(fitsfile='./examples/ngc2403.fits',smooth=True,bmaj=120,bmin=120)

# Parameters can alternatively be given as a list of strings or dictionary
# bb = BBaroloWrapper(['FITSFILE=./examples/ngc2403.fits','SMOOTH=True','BMAJ=120',
↪'BMIN=120'])
# bb = BBaroloWrapper(dict(fitsfile='./examples/ngc2403.fits',smooth=True,bmaj=120,
↪bmin=120))

# Parameters can also be added or removed
bb.add_options(threads=8,outfolder='myoutfolder')
bb.remove_option('threads')

# Parameters can be printed or easily be written in a parameter file
print (bb)
bb.write_parameterfile("param.par")

# Now we can run BBarolo
bb.run(exe="/path_to_BBarolo/BBarolo",stdout=None)
```

The above is exactly the same as calling BBarolo with a parameter file, thus it will write all the outputs in the usual output folder. The `run()` function above takes in input two optional parameters, `exe` and `stdout`. The former is your local BBarolo binary (if not given, it will search in your $PATH). The latter is where to write BBarolo's messages: if `None` or not given, BBarolo will print messages on the command line, if `'filename.log'` it will write everything in a file *filename.log* , if `'null'` no output message will be written.

NB 1: Parameter names given to `BBaroloWrapper` are not case sensitive.

NB 2: No check is performed on the given parameters. If a parameter does not exist in BBarolo, it will just be ignored.

CHAPTER

SEVEN

API

CHAPTER

EIGHT

LICENSE AND CITATIONS

BBarolo and **pyBBarolo** are distributed under the terms of the GNU General Public License version 3.0. The text of the license is included in the main directory of the repository as LICENSE.

If you use **BBarolo** or **pyBBarolo** in any published work, please cite the code paper: Di Teodoro & Fraternali, 3D BAROLO: a new 3D algorithm to derive rotation curves of galaxies, 2015, MNRAS, 451, 3021.

CHAPTER

NINE

TROUBLESHOOTING

I will try to write a troubleshooting page while I receive the feedback from BBarolo's users.

In the meanwhile, please report any bug or problem you have with BBarolo and pyBBarolo. If you are a Github user, you can submit an issue ticket at this page. Otherwise you can email me. Please attach any significant error messages and tell me how to reproduce the problem.

I will try to fix the issues as soon as I can. Thank you.

9.1 Frequently Asked Questions

1) The code compiled successfully but when I run BBarolo I get an "error while loading shared libraries: libXXX: cannot open shared object file: No such file or directory".

This error arises when some of BBarolo's dependencies (CFITSIO, WCS or FFTW libraries) have not been installed in a default library path. Solve this problem by adding the library path to the LD_LIBRARY_PATH variable. If you are using Bash shell:

```
export LD_LIBRARY_PATH=/path/to/libXXX:$LD_LIBRARY_PATH
```

If you are using C-shell:

```
setenv LD_LIBRARY_PATH /path/to/libXXX:$LD_LIBRARY_PATH
```

To make it permanent, the above commands can be simply added to your ~/.bashrc or ~/.cshrc files (make sure to start a new shell after doing it).

2) On my Mac OS, BBarolo compiles correctly but then it crashes with no error message.

This may be due to Apple default compiler, Clang. The latest versions of Clang++ does not compile the code correctly, resulting in a SEGSEV error (like "zsh: abort"). To fix this, recompile the code using another compile (for example GNU GCC).

3) BBarolo seems to run smoothly but suddenly it gets "Killed".

The code has been killed by the system kernel for some reason. For example, you can check that with:

```
dmesg | grep -i kill
```

99% will be a memory problem. BBarolo needs to allocate as much memory as three times the size of the input datacube. If you are working with big datasets (>4 GB), you can easily run out of memory. Please use a more powerful machine.