SIR-MIG

Mike Moser

Apr 29, 2024

CONTENTS:

CHAPTER

ONE

INDICES AND TABLES

- genindex
- modindex
- search

CHAPTER

TWO

WELCOME TO SIR-MIG'S DOCUMENTATION!

SIR-MIG is an algorithm based on the inversion code SIR. It parallelises SIR and implements functionalities to use multiple random guesses. There are three different modes selectable:

- 1C : 1 Component Inversion
- 2C : 2 Component Inversion
- MC : Monte-Carlo Simulation

It is also possible to use it without random guesses.

Check out the *Usage* section for further information.

Check out the *Functions* section for a list of functions.

Check out the *Classes* section for a list of classes which are used to read and write the Stokes profiles and the models.

Note: This project is under active development.

2.1 Contents

2.1.1 Usage

Installation

To use SIR-MIG, first install the necessary libraries as defined in the text file requirements.py. The code might also work for other versions as mentioned there but it is not tested.

The necessary files with the right version can be installed by executing any of the two following lines in the directory of SIR-MIG:

```
(.venv) $ pip install pipreqs
(.venv) $ conda install --yes --file requirements.txt -c conda-forge
```

2.1.2 Functions

File sir.py

Library for repeating functions such as reading the config, writing SIR files.

sir.**list_to_string**(*temp*, *let=','*)

Convert a list to a string

Parameter

temp : list let : str

Letter which is added as a separation

Return

string with the information from the list

sir.**option**(*text1*, *text2*)

Print an option in a help page

Parameter

text1
[str] First text

text2
[str] Second text

Return

None

sir.**read_chi2**(*filename*, *task=''*)

Reads the last chi value in a inv.chi file

Parameter

filename
[string] Path of the chi file

task
[string, optional] Prints out in which folder the chi2 file does not exist. Default: ‘’

Return

chi2
[float] Best chi2 value of the fit

sir.**read_chi2s**(*conf*, *tasks*)
Reads all the chi2 from the inversion

Parameter

config
[dict] Config parameters

tasks
[dict] Dictionary with the used folders

Return

chi2
[numpy array] Numpy array with all chi2 values

sir.**read_grid**(*filename*)
Reads the grid file

Parameter

filename
[string] File to be read

Return

dict
[Dictionary] Dict. with 'Line', 'min', 'step' and 'max' in it

sir.**read_line**(*filename*)
Reads the line file

Parameter

filename
[string] File to be read

Return

dict
[Dictionary] Dict. with 'Line', 'Ion', 'wavelength', 'factor', 'Exc_Pot', log_gf', 'Transition', 'alpha' and 'sigma' in it

sir.**read_config**(*filename*, *check=True*, *change_config=False*)

Reads a config file for the inversion

Parameters

filename
[string] Path of the control file

check
[bool, optional] Check if file exists (Default: True)

change_config
[bool, optional] config file is read to be changed (=> Do not try to load anything) (Default: False)

Returns

Dict
[dict] Dict containing all the information from the config file

sir.**read_control**(*filename*)

Reads a control file in the scheme SIR expects it.

Parameter

filename
[string] Path of the control file

Return

Dict
[dict] Dict containing all the information from the control file

sir.**read_model**(*filename*)

Reads a model file and returns all parameters

Parameter

filename
[string] String containing the path of the file

Return

log_tau
[numpy.array] Log tau

T
[numpy.array] Temperature in K

Pe
[numpy.array] Electron pressure in dyn/cm^2

v_micro
[numpy.array] Microturbulence velocity in cm/s

B
[numpy.array] Magnetic field strength in Gauss

vlos
[numpy.array] Line-of-sight velocity in cm/s

inc
[numpy.array] Inclination in deg

azimuth
[numpy.array] Azimuth angle in deg

z
[numpy.array, optional] Height in km

Pg
[numpy.array, optional] Gas pressure in dyn/cm^2

rho
[numpy.array, optional] Density in g/cm^3

`sir.`**`read_profile`**(*filename*, *num=0*)

Reads the first LINE data from a profile computed by SIR

Parameter

filename
[string] String containing the path of the file

num
[int, optional] Number of the line which is loaded. Default: 0 (use first one from line)

Return

ll
[numpy.array] Wavelengths in A

I
[numpy.array] Stokes I

Q
[numpy.array] Stokes Q

U
[numpy.array] Stokes U

V
[numpy.array] Stokes V

sir.**write_config_1c**(*File*, *conf*)
Writes a config file with the information provided as a dictionary for the mode 1C

Parameters

File
[string] Save path

conf
[dict] Dictionary with all the informations

sir.**write_config_2c**(*File*, *conf*)
Writes a config file with the information provided as a dictionary

Parameters

File
[string] Save path

conf
[dict] Dictionary with all the informations

sir.**write_config_mc**(*File*, *conf*)
Writes a config file with the information provided as a dictionary

Parameters

File
[string] Save path

conf
[dict] Dictionary with all the informations

sir.**write_config**(*File*, *conf*)
Writes a config file with the information provided as a dictionary for the mode 1C

Parameters

File
[string] Save path

conf
[dict] Dictionary with all the informations

sir.**write_control_1c**(*filename*, *conf*)

Writes a control file in the scheme SIR expects it.

Parameter

filename
[string] Save filename of the control file. Typically it is inv.trol

config
[dict] Dictionary with the information from the config file

sir.**write_control_2c**(*filename*, *conf*)

Writes a control file in the scheme SIR expects it.

Parameter

filename
[string] Save filename of the control file. Typically it is inv.trol

conf
[dict] Dictionary with the information from the config file

sir.**write_control_mc**(*filename*, *conf*, *Type='inv'*)

Writes a control file in the scheme SIR expects it.

Parameter

filename
[string] Save filename of the control file. Typically it is inv.trol

config
[dict] Dictionary with the information from the config file

Type
[string] which type of control file is created ('syn' for synthesis, 'inv' for inversion)

sir.**write_grid**(*conf*, *waves*, *filename='Grid.grid'*)

Writes the Grid file with data from the config file

Parameter

config
[dict] Dictionary containing all the information from the config file

filename
[string, optional] String containing the name of the Grid file. Default: Grid.grid

sir.**write_grid_mc**(*conf*, *filename='Grid.grid'*)
Writes the Grid file with data from the config file

Parameter

config
[dict] Dictionary containing all the information from the config file

filename
[string, optional] String containing the name of the Grid file. Default: Grid.grid

sir.**write_model**(*filename*, *Header*, *log_tau*, *T*, *Pe*, *v_micro*, *B*, *vlos*, *inc*, *azimuth*, *z=None*, *Pg=None*, *rho=None*)
Write a model with the given data in a specific format. Note that negative values have one white space less

Parameter

filename
[string] Name of the saved file

Header
[string] Header of the model

log_tau
[numpy.array] Log tau

T
[numpy.array] Temperature in K

Pe
[numpy.array] Electron pressure in dyn/cm^2

v_micro
[numpy.array] Microturbulence velocity in cm/s

B
[numpy.array] Magnetic field strength in Gauss

vlos
[numpy.array] Line-of-sight velocity in cm/s

inc
[numpy.array] Inclination in deg

azimuth
[numpy.array] Azimuth angle in deg

z
[numpy.array, optional] Height in km

Pg
[numpy.array, optional] Gas pressure in dyn/cm^2

rho
[numpy.array, optional] Density in g/cm^3

Return

None

sir.**write_profile**(*filename*, *profiles*, *pos*)
Write a profile for a specific model number to a file

Parameter

filename
[string] Name of the saved file

profiles
[list] List containing all the profiles

atoms
[list] List containing the number of the line from the Line file

pos
[int] Position which model is saved

Return

None

File create_config.py

Functions to create the config files for the different modes.

create_config.**config_MC**()
Creates a config file for the Monte-Carlo simulation by asking questions.

Parameters

None

Returns

None

create_config.**config_1C**()

Creates a config file for the 1 Component Inversion by asking questions.

Parameters

None

Returns

None

create_config.**config_2C**()

Creates a config file for the 2 Components Inversion by asking questions.

Parameters

None

Returns

None

File main.py

Main file to start the program

main.**main**()

Function which executes the programme

Parameters

None

Returns

None

File model.py

Function to read a binary model file with all the physical parameter

model.**read_model**(*filename*)

File misc.py

Miscellaneous functions .. autofunction:: misc.initial

File obs.py

Functions related to the observation

obs.**read_profile**(*profile*, *grid*, *line_file*, *waves*)

obs.**write_psf**(*conf*, *filename*)

Writes the spectral point spread function with the value from the spectral veil correction

Parameter

config
[dict] Dictionary containing all the information of the config file

filename
[string] Filename under which the file is saved

Return

None

File profile_stk.py

Function to read a binary profile file with the four Stokes Parameter

2.1.3 Classes

File model.py

Class model with all the physical parameter

class model.**Model**(*nx=0*, *ny=0*, *nval=0*)

Class containing the models of a simulation

Variables are:

- log_tau
- T
- Pe
- vmicro

- B
- vlos
- gamma
- phi
- z
- pg
- rho

There are several functions:

- read: Read a numpy file containing models and stores it into the class variables
- read_results: Reads the results from the inversion of SIR
- write: Writes a Model to a SIR readable file
- save: Saves the models as a numpy file
- set_limit: Cuts the data to a specific log_tau value (used for plotting)

`__init__`(*nx=0*, *ny=0*, *nval=0*)

Initialisation of the class with the models

Parameter

filename
[str, optional] File name of the model to be loaded. Default: None (= no reading)

filename_fill
[str, optional] File name of the filling factor

`correct_phi`()

SIR can give you any value for the azimuth, e.g. -250 deg, and therefore, to compute the standard deviation those values should be corrected so that I have values from 0 to 360 degrees.

Parameter

None

Return

class

`cut_to_map`(*Map*)

Cut the data to a map [xmin, xmax, ymin, ymax]

Parameters

Map
[list] List with the ranges in pixel in x and y direction

`get_attribute`(*string*)

Returns a specific physical parameter. This can be used if ones only wants a specific parameter depending on a string/input

Parameter

string
[str] Determines which physical parameter is returned Options are: tau, T, Pe, vmicro, B, vlos, gamma, phi, z, Pg, rho, nx, ny, npar, fill

Return

The wished physical parameter

`read`(*fname*, *fmt_type=<class 'numpy.float64'>*)

`read_results`(*task*, *filename*, *path*, *nx*, *ny*)

Reads all the errors from the inversion

Parameter

task
[dict] Dictionary with all the task folders, x and y positions

filename
[str] Filename of the file in each task folder

path
[str] Path where all the files are

nx
[int] how many results are read in x

ny
[int] how many results are read in y

Return

class with the parameters

`set_dim`(*nx*, *ny*, *npars*)

Sets the dimensions if no data is loaded yet

Parameter

nx
[int] Number of Models in x

ny
[int] Number of Models in y

npars
[int] Number of values per physical parameter

`set_limit`(*lim*)
Cuts the arrays to a specific log tau value (should only be used for plotting)

Parameter

lim
[int] log tau value to which the values are cut

`write`(*fname*, *fmt_type=<class 'numpy.float64'>*)
Write data into a binary fortran file

Parameter

fname
[str] File name

fmt_type
[type] type which is used to save it => numpy.float64 used

`write_model`(*filename*, *header*, *x*, *y*)
Write a model with the given data in a specific format.

Parameter

filename
[string] Name of the saved file

Header
[string] Header of the model

x
[int] Integer determining which model

y
[int] Integer determining which model

Return

None

`correct_phi()`

SIR can give you any value for the azimuth, e.g. -250 deg, and therefore, to compute the standard deviation those values should be corrected so that I have values from 0 to 360 degrees.

Parameter

None

Return

class

`cut_to_map(Map)`

Cut the data to a map [xmin, xmax, ymin, ymax]

Parameters

Map
[list] List with the ranges in pixel in x and y direction

`get_attribute(string)`

Returns a specific physical parameter. This can be used if ones only wants a specific parameter depending on a string/input

Parameter

string
[str] Determines which physical parameter is returned Options are: tau, T, Pe, vmicro, B, vlos, gamma, phi, z, Pg, rho, nx, ny, npar, fill

Return

The wished physical parameter

`read_results(task, filename, path, nx, ny)`

Reads all the errors from the inversion

Parameter

task
[dict] Dictionary with all the task folders, x and y positions

filename
[str] Filename of the file in each task folder

path
[str] Path where all the files are

nx
[int] how many results are read in x

ny
[int] how many results are read in y

Return

class with the parameters

`set_dim`(*nx*, *ny*, *npars*)
Sets the dimensions if no data is loaded yet

Parameter

nx
[int] Number of Models in x

ny
[int] Number of Models in y

npars
[int] Number of values per physical parameter

`set_limit`(*lim*)
Cuts the arrays to a specific log tau value (should only be used for plotting)

Parameter

lim
[int] log tau value to which the values are cut

`write`(*fname*, *fmt_type=<class 'numpy.float64'>*)
Write data into a binary fortran file

Parameter

fname
[str] File name

fmt_type
[type] type which is used to save it => numpy.float64 used

`write_model`(*filename*, *header*, *x*, *y*)
Write a model with the given data in a specific format.

Parameter

filename
[string] Name of the saved file

Header
[string] Header of the model

x
[int] Integer determining which model

y
[int] Integer determining which model

Return

None

File profile_stk.py

Class profile_stk with the four Stokes Parameter

`class profile_stk.Profile`(*nx=None*, *ny=None*, *nw=0*)
Class containing the models of a simulation

Variables are:

- wave
- stki
- stkq
- stku
- stkv

There are several functions:

- read: Read a numpy file containing models and stores it into the class variables
- read_results: Reads the results from the inversion of SIR
- write: Writes a Model to a SIR readable file
- save: Saves the models as a numpy file
- set_limit: Cuts the data to a specific log_tau value (used for plotting)

`__init__`(*nx=None*, *ny=None*, *nw=0*)

Initialisation of the class with the Profiles

Parameter

nx
[int] Integer of pixels in x direction

ny
[int] Integer of pixels in y direction

nw
[int] Number of wavelength points

`__read_grid`()

Reads the grid file

Parameter

filename
[string] File to be read

Return

dict
[Dictionary] Dict. with 'Line', 'min', 'step' and 'max' in it

`__read_profile_sir`()

Reads the first LINE data from a profile computed by SIR

Parameter

filename
[string] String containing the path of the file

Return

ll
[numpy.array] Wavelengths in A

I
[numpy.array] Stokes I

Q
[numpy.array] Stokes Q

U
[numpy.array] Stokes U

V
[numpy.array] Stokes V

__read_profile_sir_mc()

Reads the first LINE data from a profile computed by SIR

Parameter

filename
[string] String containing the path of the file

Return

ll
[numpy.array] Wavelengths in A

I
[numpy.array] Stokes I

Q
[numpy.array] Stokes Q

U
[numpy.array] Stokes U

V
[numpy.array] Stokes V

cut_to_map(*Map*)

Cut the data to a map [xmin, xmax, ymin, ymax]

Parameters

Map
[list] List with the ranges in pixel in x and y direction

cut_to_wave(*range_wave*)

Cut the data to the range in wavelengths

Parameters

range_wave
[list] List with the ranges from the config file

read(*fname*, *fmt_type=<class 'numpy.float32'>*)

read_results(*task*, *filename*, *path*, *nx*, *ny*)

Reads all the errors from the inversion

Parameter

task
[dict] Dictionary with all the task folders, x and y positions

filename
[str] Filename of the file in each task folder

path
[str] Path where all the files are

nx
[int] how many results are read in x

ny
[int] how many results are read in y

Return

class with the parameters

read_results_MC(*path*, *tasks*, *filename*)

Reads all the profiles of the synthesis or inversion

config
[dict] Config information

tasks
[dict] Dictionary with the folder names

Type
[string, optional] Indicating if synthesis or inversion. Default: 'syn'

set_dim(*nx*, *ny*, *nw*)

Sets the dimensions if no data is loaded yet

Parameter

nx
[int] Number of Models in x

ny
[int] Number of Models in y

nw
[int] Number of wavelength points

write(*fname*, *fmt_type=<class 'numpy.float32'>*)

Write data into a binary fortran file

Parameter

fname
[str] File name

fmt_type
[type] type which is used to save it => numpy.float64 used

`write_profile`(*filename*, *x*, *y*, *Grid*)

Writes data to profiles as described in the config file Note to call cut_to_wave, otherwise it writes the wrong profiles!

Parameter

filename
[string] String containing the output path of the profile

x
[int] Position in x

y
[int] Position in y

Grid
[string] Grid file

`write_profile_mc`(*filename*, *x*)

Writes data to profiles as described in the config file

Parameter

filename
[string] String containing the output path of the profile

x
[int] Position in x

y
[int] Position in y

Grid
[string] Grid file

`cut_to_map`(*Map*)

Cut the data to a map [xmin, xmax, ymin, ymax]

Parameters

Map
[list] List with the ranges in pixel in x and y direction

`cut_to_wave`(*range_wave*)
Cut the data to the range in wavelengths

Parameters

range_wave
[list] List with the ranges from the config file

`read_results`(*task*, *filename*, *path*, *nx*, *ny*)
Reads all the errors from the inversion

Parameter

task
[dict] Dictionary with all the task folders, x and y positions

filename
[str] Filename of the file in each task folder

path
[str] Path where all the files are

nx
[int] how many results are read in x

ny
[int] how many results are read in y

Return

class with the parameters

`read_results_MC`(*path*, *tasks*, *filename*)
Reads all the profiles of the synthesis or inversion

config
[dict] Config information

tasks
[dict] Dictionary with the folder names

Type
[string, optional] Indicating if synthesis or inversion. Default: 'syn'

`set_dim`(*nx*, *ny*, *nw*)
Sets the dimensions if no data is loaded yet

Parameter

nx
[int] Number of Models in x

ny
[int] Number of Models in y

nw
[int] Number of wavelength points

write(*fname*, *fmt_type=<class 'numpy.float32'>*)
Write data into a binary fortran file

Parameter

fname
[str] File name

fmt_type
[type] type which is used to save it => numpy.float64 used

write_profile(*filename*, *x*, *y*, *Grid*)
Writes data to profiles as described in the config file Note to call cut_to_wave, otherwise it writes the wrong profiles!

Parameter

filename
[string] String containing the output path of the profile

x
[int] Position in x

y
[int] Position in y

Grid
[string] Grid file

write_profile_mc(*filename*, *x*)
Writes data to profiles as described in the config file

Parameter

filename
[string] String containing the output path of the profile

x
[int] Position in x

y
[int] Position in y

Grid
[string] Grid file

2.1.4 API

create_config	Create a config file as expected for the inversion
sir	Library for repeating functions for analyzing and/or plotting SIR data
model	Class Model with all the tools to read and write the models
profile_stk	Class Profile with all the tools to read and write the Stokes Profiles

create_config

Create a config file as expected for the inversion

Functions

config_1C()	Creates a config file for the 1 Component Inversion by asking questions.
config_2C()	Creates a config file for the 2 Components Inversion by asking questions.
config_MC()	Creates a config file for the Monte-Carlo simulation by asking questions.
help()	

sir

Library for repeating functions for analyzing and/or plotting SIR data

Functions

list_to_string(temp[, let])	Convert a list to a string
option(text1, text2)	Print an option in a help page
read_chi2(filename[, task])	Reads the last chi value in a inv.chi file
read_chi2s(conf, tasks)	Reads all the chi2 from the inversion
read_config(filename[, check, change_config])	Reads a config file for the inversion
read_control(filename)	Reads a control file in the scheme SIR expects it.
read_grid(filename)	Reads the grid file
read_line(filename)	Reads the line file
read_model(filename)	Reads a model file and returns all parameters
read_profile(filename[, num])	Reads the first LINE data from a profile computed by SIR
write_config(File, conf)	Writes a config file with the information provided as a dictionary for the mode 1C
write_config_1c(File, conf)	Writes a config file with the information provided as a dictionary for the mode 1C
write_config_2c(File, conf)	Writes a config file with the information provided as a dictionary
write_config_mc(File, conf)	Writes a config file with the information provided as a dictionary
write_control_1c(filename, conf)	Writes a control file in the scheme SIR expects it.
write_control_2c(filename, conf)	Writes a control file in the scheme SIR expects it.
write_control_mc(filename, conf[, Type])	Writes a control file in the scheme SIR expects it.
write_grid(conf, waves[, filename])	Writes the Grid file with data from the config file
write_grid_mc(conf[, filename])	Writes the Grid file with data from the config file
write_model(filename, Header, log_tau, T, ...)	Write a model with the given data in a specific format.
write_profile(filename, profiles, pos)	Write a profile for a specific model number to a file

model

Class Model with all the tools to read and write the models

Functions

read_model(filename)	

Classes

Model([nx, ny, nval])	Class containing the models of a simulation

profile_stk

Class Profile with all the tools to read and write the Stokes Profiles

Functions

`read_profile`(file)	Reads a profile and returns a class

Classes

`Profile`([nx, ny, nw])	Class containing the models of a simulation

PYTHON MODULE INDEX

INDEX

Symbols

C

G

L

M

O

P

R

S

W