`fringe_code_iris` : User Manual

Simon Moine (simon.moine@ens-lyon.fr),
Stéphane Basa (stephane.basa@lam.fr)

June 2023

Contents

Part I
Manual

1 Introduction

Here we present a program to automatically remove fringing on the images taken by the IRiS telescope. The following document explains how to use the program.

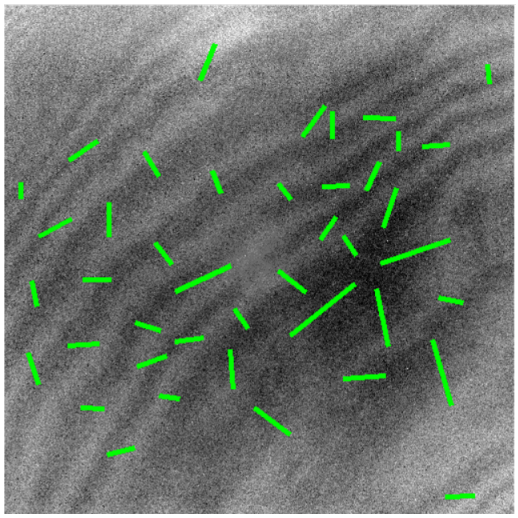

Figure 1: Example of a set of control pairs on a fringe model. The control pairs are defined by the end of the lines, the first end is on a bright fringe and the second end on a dark fringe.

2 Overall method

Fringing is due to two things : the atmospheric emission of light by OH molecules in the sky and the CCD's defaults. As a consequence, the fringe pattern does not change with time. However, the intensity of the fringes depends on the atmospheric conditions and can strongly change during a night.

To remove the fringing on IRiS images, we use the method proposed by Snodgrass C. and Carry B. in [2]. First, we have to make a fringe model by taking the median of a great number of image taken by the telescope.
Then, to remove the fringing on the images, we used a certain number of control pairs, which are pairs of pixels: one of the two pixels is on a bright fringe and the other is on a dark fringe. An illustration of this is showed on the figure 1. Once we have our control pairs, we calculate the difference of flux between the bright and dark pixels for each pair on the model and on the image we want to clean. Then, we calculate the ratio between the flux differences on the image and on the model, and take the median. Finally, we subtract the model to the image, weighted by the median ratio.

All these steps are summarized in the flowchart of figure 2.
To remove fringing from the IRiS images, there are three different codes to run :

- `gather_data.py`, which is used to gather all the calibrated images taken by IRiS in a given period of time,
- `model.py`, which is used to create a fringe model, which will be subtracted to the images we want to reduce later. The algorithm used here is inspired by the algorithm of Snodgrass C. and Carry B. for EFOSC Images (cf. [2]) and the `fringez` code from Medford et al. (cf. [1]),
- `remove_fringing.py`, which is used at the end to remove the fringes on a given image or image folder. The algorithm used is taken from [2].

The detailed functioning of each code is available in appendixe at the end of this document.

3 Use of the program

3.1 Input and output

The script works only on `fits` images with one header and one body.

It generates images reduced from fringing. For an image `image.fits`, the reduced image will be `image_fringecor.fits`. All the information about the reduction parameters are written in the header's history.

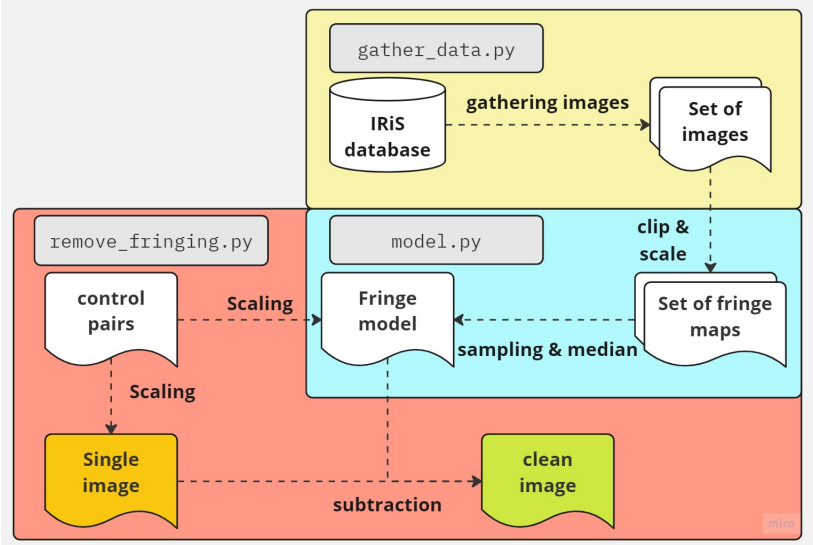

Figure 2: Flowchart of the IRiS fringe code.

3.2 Setup files

All three codes need a setup file, which is a text file. You must write only one parameter per line. Here is the template for a line:

```
parameter   value
```

`parameter` and `value` are separated by a tabulation. Additional spaces and empty lines are ignored. Comments begin with the character `"#"`. Some parameters take a default value if they are not specified. The list of all parameters with their possible default values and their type are specified in the section **Parameters**.

All the setup files and the codes must be in the same directory. Plus, all the file extensions must be specified in their name.

3.3 Gathering data

First, you will a need set of images to make the fringe model. To do this, we use the `gather_data.py` code, which directly downloads all the images from the IRiS database taken between 2 dates and save them in a folder on the computer. The folder will be created in the path you indicate in the setup file.

For the creation of the model, it is recommended to take images from different parts of the sky. For more information, see the parameter `far target`).

First, create the setup file. Then, open a terminal in the directory in which you are, and write this command:

```
python gather_data.py -f gather_data_setup.txt
```

If you want more information about how the parameters are given to the code (for debugging for example), write instead:

```
python gather_data.py -f gather_data_setup.txt -v
```

3.4 Making the model

Once you have the data, you can make the fringe model. To do this, we use the code `model.py`. It will create a `fits` image of the model in the path you indicate in the setup file.

As the data are quite big, it is possible to divide your image set in multiple samples to avoid an overflow error. (see the parameter `number of samples`). In this case, the code will make a model per sample, and then will take the median of all these models to create the final model.

As the fringe pattern does not change with time, you only need to make the fringe model once. It is also recommended to use only images without large light sources (galaxies and nebulas) to make the model : it will improve the quality of the model.

First, create the setup file, in which you will give the name of the image folder you just created before. Then, open a terminal in the directory in which you are, and write this command :

```
python model.py -f model_setup.txt
```

If you want more information about how the parameters are given to the code (for debugging for example), write instead :

```
python model.py -f model_setup.txt -v
```

3.5 Defining the control pairs

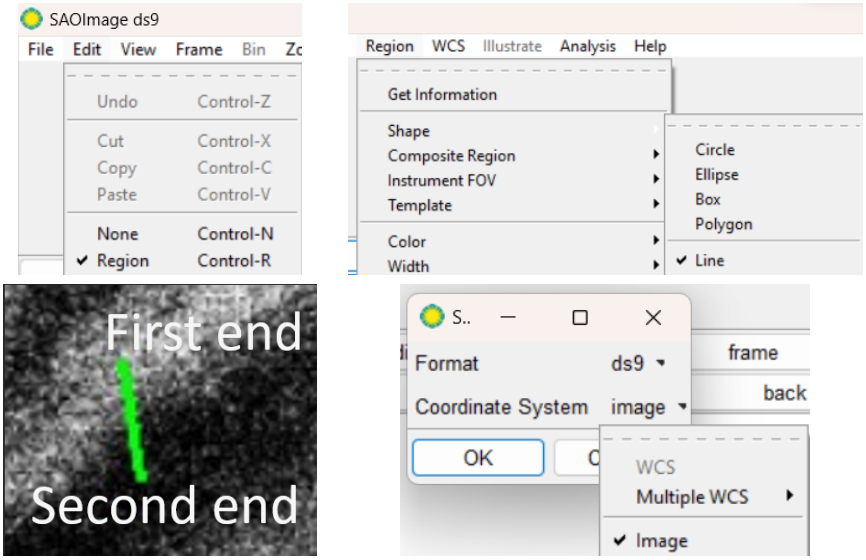

Figure 3: Steps to create your own control pairs file : first (top left), select `region` in the `edit` tab, then (top right) choose the `line` shape and then (bottom left) draw your lines, beginning by the bright end. Finally, (bottom right) save the file with the `image` coordinate system.

In the script zip, you will find a file named `pairs.xml`, which contains all the information about the control pairs we can see on figure 1. You can use these control pairs in what is coming next. It is a region file made with DS9.

If you want, you can create your own control pairs file from scratch by using DS9. All the following steps are summarized in figure 3.
To do so, first open one of the fringe models you have in DS9. Then, go to the `edit` tab and activate the `region` mode. Then, go to the `region` tab, then to `shape`, and select `line`. From there, drag and click on the image to build your lines. **It is important to note that the first end of a pair is considered to be the bright end and the second the dark end by the program, so be careful when you are doing your control pairs file.** When you have finished with drawing the control pairs, you can save them in a region file by going to the `region` tab. When the software asks the coordinate system, make sure to select `image` to save the pixel coordinates. You can save the region file with the extensions `.reg` or `.xml`.

For the algorithm to work well, the experience shows that at least 20 pairs are required. Also note that the pixel coordinates can be floats : the program will convert them into integers if this happens.

3.6 Removing the fringing on IRiS images

Now you can remove fringing on your images using the `remove_fringing.py` code, which will subtract correctly the model which was done just before. The reduced images will be created in the same directory as the original images.

You can either reduce a single image or all the images in a folder, depending on the parameter you give in the setup file. If the two parameters are given, the script will run on the folder.
In order not to be bothered by the noise in the signal while calculating the flux differences on the images, the bright and dark pixels values of each control pairs (which are needed to scale the model to the image) are obtained by calculating the mean of the pixel values in a small box centered on the end of each pair (see the parameter `box width`). This is showed on figure 4.

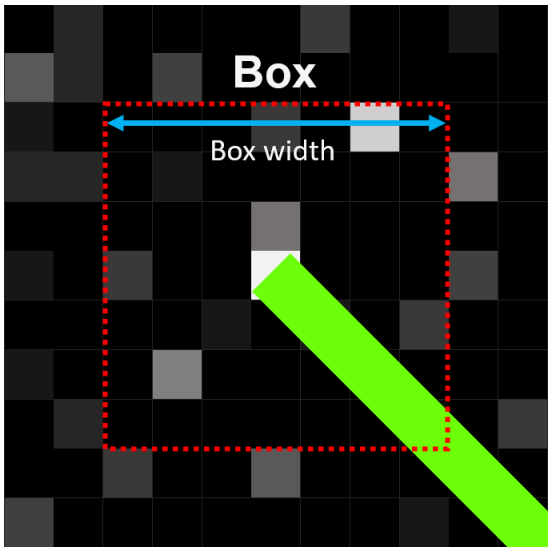

Figure 4: Diagram which explains how the bright and dark pixel values are calculated : we do the mean of the pixel values in a little box centered in the end of the pair.

First, create the setup file in which you will specify the model and (and maybe the control pairs file) you have done before. Then, open a terminal in the directory in which you are, and write this command:

```
python remove_fringing.py -f remove_fringing_setup.txt
```

If you want more information about how the parameters are given to the code (for debugging for example), write instead:

```
python remove_fringing.py -f remove_fringing_setup.txt
```

4 Parameters

All the dates are UTC dates. The date type is a string with the format `YYYY/MM/DD`. The shape type is a couple of integers with the format `int1xint2`. When a parameter is a file name (or a folder name), you can either give the path of the file (e.g. `~\python\fringe_code\iris_model.fits`) or just the name of the file (e.g. `iris_model.fits`). If you just give the name of an input file, make sure that the file is in the same directory as the codes. If you just give the name of an output file, it will be created in the same directory as the codes. All parameters without a default value (`None` in the **Default value** column) must be specified, except for `remove_fringing.py` (in which you can specify only one of the two parameters `image name` and `folder name`) and `number of samples`.

Name	Type	Default value	Description / Remarks
gather_data.py			
band	string	i	Band of observation / The possible values are : u, g, r, i, z, OIII, CH4 and Halpha
beg date	date	{year - 1}/{month}/{day}	Date after which the images have been taken
end date	date	{year}/{month}/{day}	Date before which the images have been taken
folder name	string	iris_{band}_band{beg date}_{end date}	The folder in which the images are saved
far target	bool	True	If set to True, the code will ignore a new target if is too close from an already seen target ($< 2'$)
model.py			
image folder	string	None	Folder containing the images for the model
number of samples	int	None	Number of samples in which the image set is divided / If None, the image set is not divided is samples
shape	shape	2048x2048	The shape of the images selected
model name	string	iris_model_{year}_{month}-{day}_{hour}_{minute}_{second}.fits	Name of the model
remove_fringing.py			
image name	string	None	Image to reduce
folder name	string	None	Folder in which are the images to reduce
model name	string	None	Model used for reduction
control pairs	string	pairs.xml	Control pairs file used for reduction
box width	int	11	Box width for the mean of pixel values / It must be an odd integer, otherwise it will be "rounded" to the superior odd integer

A Full architecture of the program

The program is divided into 4 codes, which are :

- `utils.py`, which contains all the basic functions needed for the three other codes,
- `gather_data.py`, which is used to gather all the calibrated images taken by IRiS in a given period of time,
- `model.py`, which is used to create a fringe model, which will be subtracted to the images we want to reduce later. The algorithm used here is inspired by the algorithm of Snodgrass C. and Carry B. for EFOSC Images (cf. [2]) and the `fringez` code from the Astronomical Society of the Pacific (cf. [1]),
- `remove_fringing.py`, which is used at the end to remove the fringes on a given image or image folder. The algorithm used is taken from [2].

B `utils.py`

As it was said before, the `utils.py` code gathers all the functions which are useful for the next codes. Here are the different functions of this code :

- `pierside`
- `create_fits`
- `update_fits`
- `gather_normalized_images`

You can find a description of each function in the following subsections.

B.1 `pierside`

A little problem of the IRiS telescope is that it can be in two different positions to observe : west pierside or east pierside. Depending on the pierside, the image does not have the same orientation. As a consequence, an image with a west pierside is flipped compared to an image with an east pierside, so is the fringe pattern. This is showed in the figure 5. This information can be found in the header of an image. In order to have the bigger number of images possible to create the model of the fringes, we want to flip the "wrong" images to have a correctly oriented pattern.

The `pierside` function takes the name of a `.fits` image from IRiS and a folder path, then checks the pierside of the image in the header, and flip it if necessary. In all the cases, the function creates in the folder indicated a new `fits` file with the name `{image_name}_pierside.fits`, to indicate that these images had their pierside checked and were flipped if necessary. We chose arbitrarily to set all the images with an **east pierside** here.

However, if an image is flipped, the sky coordinates are set wrong because they are not flipped with the image. So we can't process the removing of the fringes on the flipped image because of this coordinate problem.
Note that this function is only used in the `model.py` code to have the bigger set of images possible to do the fringe model. The `remove_fringing.py` code reduces the original images without a modified pierside to keep the right sky coordinates.

B.2 `create_fits` and `update_fits`

These functions are directly taken from the `fringez` code from [1].

`create_fits` takes an image name, a matrix, and a header. Then, it creates a new `.fits` image `{image_name}.fits`, with the matrix as the image and the header given. The only problem of this function is that if a file with the same already exists, it will overwrite this file, but not correctly.
To avoid this problem, the `update_fits` function exists. It does the same thing as `create_fits` but replaces correctly an existing file.

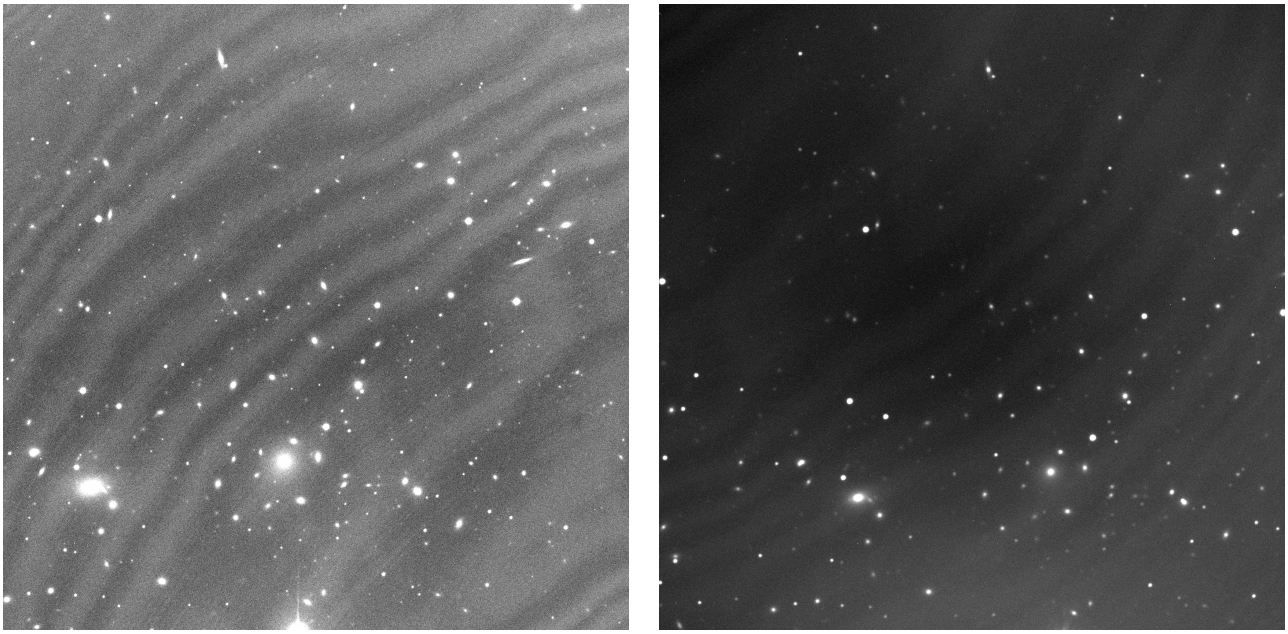

Figure 5: Two images of A1656, taken by IRiS in the SDSS i' band. The left image was taken with an east pierside, and the right image with a west pierside. One can notice that the two images are flipped, and so is the fringe pattern.

B.3 `gather_normalized_images`

This function is adapted from the `fringez` code from [1].

The `gather_images` function takes a folder name and an optional parameter `N_samples`. For each image in the folder, the function takes its median and subtract it to the original image. Then, the normalized image obtained is put in a list containing all the normalized images from the folder.
Sometimes, the data can be quite big, and it is impossible to make an exhaustive list of all the normalized images without an overflow error during the creation of the model after. To avoid this problem, we have the parameter `N_samples` : it indicates into how many samples we want to divide our set of images in the folder. For each sample, normalized images will be gathered, and then their median will be taken. It is this median which will be added to the final list returned by the function.

C Algorithms of the three other codes

C.1 `gather_data.py`

Here, the algorithm is quite simple :

- first, we load the `HTML` source code of the IRiS observation page,
- then, we memorize all the zip URLs corresponding to dates between the two dates we gave.
- Finally, we download all the zips and extract the calibrated images from them. A calibrated image is an image on which the dark, bias and flat were corrected.

C.2 `model.py`

All the images of a given folder will be processed as it is indicated after :

- First, we check the pierside of each image, flip it if necessary, and finally save the image (flipped or not) in a temporary file,

- then we gather the fringe maps of the images in the temporary file (so the fringe pattern has the same orientation for all the images).
- Finally, we take the median of the fringe map set : this is our fringe model, which will be saved as a `fits` file.

At the end, all the parameters used to create the model and the code version used are written in the header's history.

C.3 `remove_fringing.py`

For the image we want to reduce and the model we use, we take the pixel value at each end of each control pair and for each control pair, we calculate the difference (δF_i for the image and δM_i for the model) between the bright value and the dark value. For the image, we name b_i the bright value of the pair i and d_i the dark value of the pair i. For the model, we name b'_i the bright value of the pair i and d'_i the dark value of the pair i. So we have :

$$\begin{array}{rcl} \delta F_i & = & b_i - d_i \\ \delta M_i & = & b'_i - d'_i \end{array} \tag{1}$$

However, in order not to be bothered by the noise in the signal, b_i and d_i are obtained by calculating the mean of the values taken by the pixels in a small box centered in the end of each pair. This is shown on the figure 4. Once we have done this, we calculate the ratios $r_i = \frac{\delta F_i}{\delta M_i}$, and we take the median, which will be named r.
From there, we obtain our clean image $\mathbf{I_{clean}}$ like this (where $\mathbf{I}$ is the original image and $\mathbf{M}$ is the model) :

$$\mathbf{I_{clean}} = \mathbf{I} - r \times \mathbf{M} \tag{2}$$

In the case in which the pierside of the image is west and not east, we flip the image in the beginning of the algorithm, and flip it again at the end (in order to have the model correctly oriented for the subtraction).

Finally, the cleaned image is saved as `{image_name}_fringecor.fits`. Plus, all the parameters used for the reduction and the code version used are written in the header's history.

D Version

There is also a little version code `version.py` in the program, containing five functions:

- `version_all` : gives the version of the complete script,
- `version_utils` : gives the version of the `utils.py` code,
- `version_data` : gives the version of the `gather_data.py` code,
- `version_model` : gives the version of the `model.py` code,
- `version_remove` : gives the version of the `remove_fringing.py` code,

If you launch directly the code, it will display the five versions. You can manually change the versions in the python code : open the code and change the version wanted inside the corresponding function.

References

[1] Michael S. Medford et al. "Removing Atmospheric Fringes from Zwicky Transient Facility i-band Images using Principal Component Analysis". In: *Publications of the Astronomical Society of the Pacific* 133.1024 (June 2021), p. 064503. DOI: 10.1088/1538-3873/abfe9d. URL: https://dx.doi.org/10.1088/1538-3873/abfe9d.

[2] Colin Snodgrass and Benoît Carry. "Automatic Removal of Fringes from EFOSC Images". In: *The Messenger* 152 (June 2013), pp. 14–16. URL: https://hal.science/hal-00844047.