Documentation

Jean-Baptiste Jolly

May 5, 2020

Beta release 1.5.3

CONTENTS

Line-Stacker is a new open access tool for stacking of spectral lines. Line-Stacker is an ensemble of both CASA tasks and native python tasks, and can stack both 3Dcubes or already extracted spectra. Additionaly a set of tools are included to help further analyse stacked spectra and stacked sample.

Some example, showing how to use of Line-Stacker, can be found in the example section.

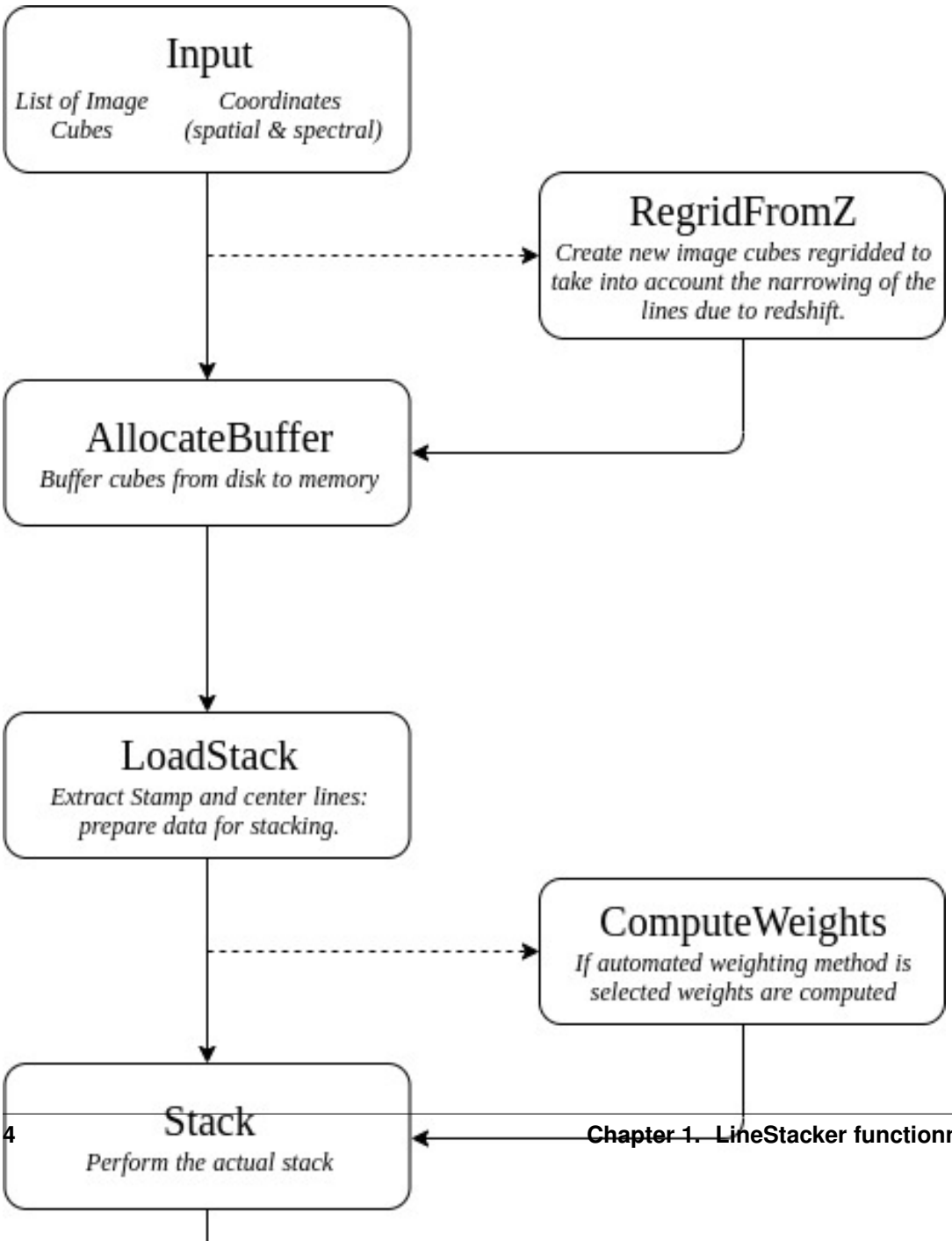
,

CHAPTER

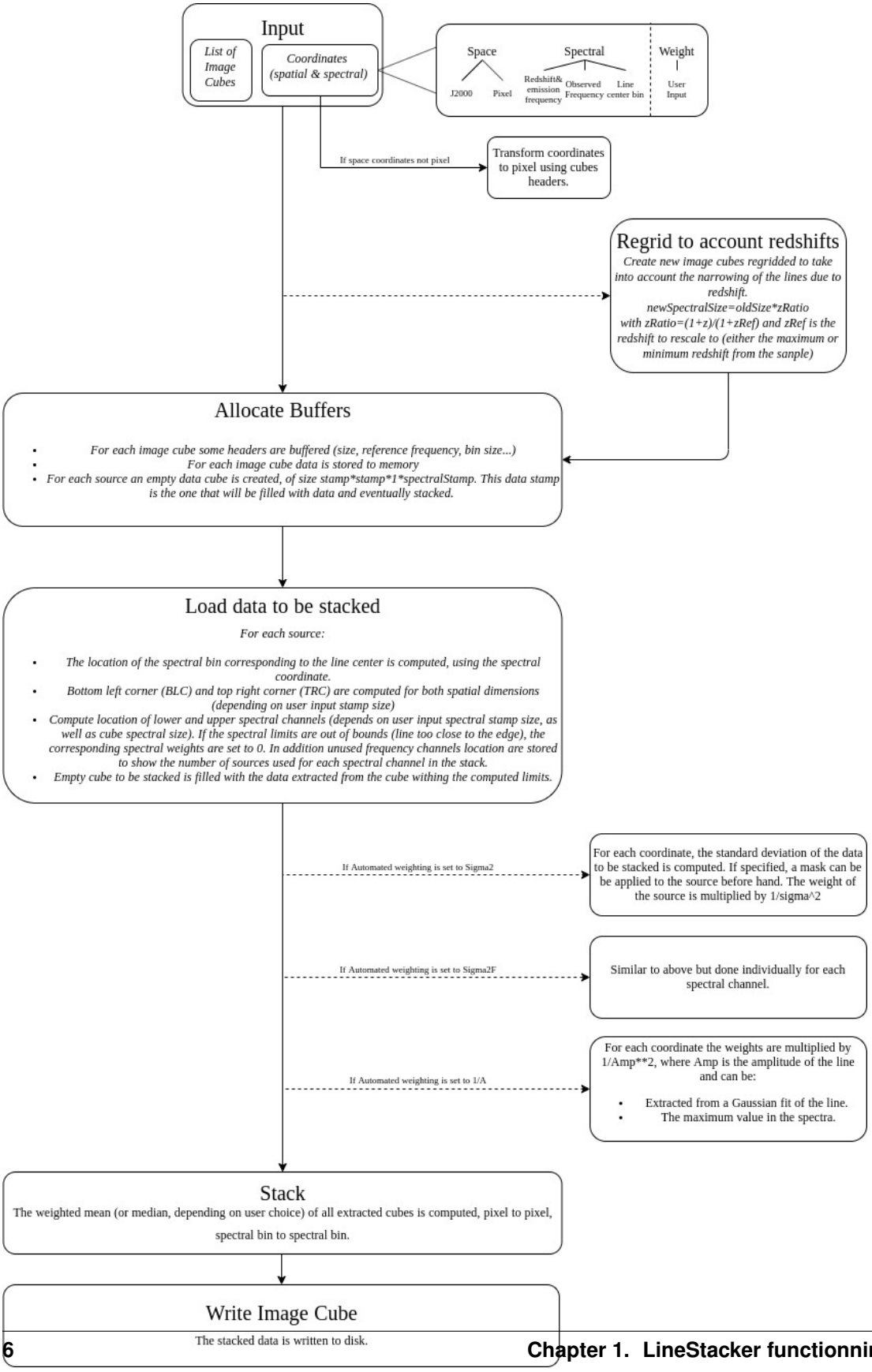
ONE

LINESTACKER FUNCTIONNING PRINCIPLE

Basic



Extended


Input
List of Image Cubes
Coordinates (spatial & spectral)
Space
J2000
Pixel
Spectral
Redshift& emission frequency
Observed Frequency
Line center bin
Weight
User Input
If space coordinates not pixel
Transform coordinates to pixel using cubes headers.
Regrid to account redshifts
Create new image cubes regridded to take into account the narrowing of the lines due to redshift.
newSpectralSize=oldSize*zRatio
with zRatio=(1+z)/(1+zRef) and zRef is the redshift to rescale to (either the maximum or minimum redshift from the sanple)
Allocate Buffers
For each image cube some headers are buffered (size, reference frequency, bin size...)
For each image cube data is stored to memory
For each source an empty data cube is created, of size stamp*stamp*1*spectralStamp. This data stamp is the one that will be filled with data and eventually stacked.
Load data to be stacked
For each source:
The location of the spectral bin corresponding to the line center is computed, using the spectral coordinate.
Bottom left corner (BLC) and top right corner (TRC) are computed for both spatial dimensions (depending on user input stamp size)
Compute location of lower and upper spectral channels (depends on user input spectral stamp size, as well as cube spectral size). If the spectral limits are out of bounds (line too close to the edge), the corresponding spectral weights are set to 0. In addition unused frequency channels location are stored to show the number of sources used for each spectral channel in the stack.
Empty cube to be stacked is filled with the data extracted from the cube withing the computed limits.
If Automated weighting is set to Sigma2
For each coordinate, the standard deviation of the data to be stacked is computed. If specified, a mask can be be applied to the source before hand. The weight of the source is multiplied by 1/sigma^2
If Automated weighting is set to Sigma2F
Similar to above but done individually for each spectral channel.
If Automated weighting is set to 1/A
For each coordinate the weights are multiplied by 1/Amp**2, where Amp is the amplitude of the line and can be:
Extracted from a Gaussian fit of the line.
The maximum value in the spectra.
Stack
The weighted mean (or median, depending on user choice) of all extracted cubes is computed, pixel to pixel, spectral bin to spectral bin.
Write Image Cube
The stacked data is written to disk.

CHAPTER

TWO

LINESTACKER

main

LineStacker Module:

Main Stacker module. Contains all basic functions.

class LineStacker.**Coord**(*x*, *y*, *z=0*, *obsSpecArg=0*, *weight=1*, *image=0*)

Describes a stacking position on an image.

Class used internally to represent coordinates. May describe a physical coordinate or a pixel coordinate.

Init: Creates a coordinate. A pixel coordinate should always specify to which image it belongs. Physical coordinates should be in J2000 radians.

Parameters

- **x** – x coordinate of stacking target
- **y** – y coordinate of stacking target
- **z** – redshift of stacking target
- **obsSpecArg** – argument of spectral bin on which to center the stack, will be computed from **z** and **fEm** if not specified
- **weight** – Weight of the. source in case of mean stacking. Default is 1.
- **image** – Index of the image associated to the source. Automatically set with **LineStacker.readCoords**.

class LineStacker.**CoordList**(*imagenames=[]*, *coord_type='physical'*, *unit='rad'*)

Extended list to contain list of coordinates.

Requires an image list in case of pixel coordinates to work properly.

Parameters

- **imagenames** – A list of image names, requiered for pixle coordinates to work properly.
- **coord_type** – **'physical'** or **'pixel'**. **'physical'** coordinates should be converted to pixels using **LineStacker.getPixelCoords** before stacking.
- **unit** – can be **'rad'**, **'deg'** or **'pix'**.

LineStacker.**getPixelCoords**(*coords*, *imagenames*)

Creates pixel coordinate list from a physical coordinate list and a list of images.

Parameters

- **coords** – A list of stacker.Coord coordinates.

- **imagenames** – A list of images' paths.

LineStacker.**randomCoords**(*imagenames*, *ncoords=10*)
Randomize a set of coordinates anywhere on any images.

Parameters

- **imagenames** – A list of images paths.
- **ncoords** – Number of random coordinates. Default is 10.

LineStacker.**randomizeCoords**(*coords*, *beam*, *maxBeamRange=5*)
Randomize a new set of coordinates at a distance [beam, maxBeamRange*beam] of the original coordinates

Parameters

- **coords** – list of original coordinates (stacker.Coord instances)
- **beam** – beam size is radians, new random coordinates will be at a minimum distance beam from the original coordinates.
- **maxBeamRange** – maximum distance from original coordinates at which new coordinates can be located, in units of beams. Default is 5.

LineStacker.**readCoordsNamesGUI**()
Open GUI to select coordinates files.

Returns path of selected files.

LineStacker.**writeCoords**(*coordpath*, *coords*, *unit='deg'*)
Write coordinates to a file

Parameters

- **coordpath** – absolute path to coordinate file to be created
- **coords** – list of coordinates (stacker.Coord instances) to write to file

line_image

LineStacker.line_image Module:

Module for cube line stacking.

LineStacker.line_image.**calculate_amp_weights**(*coords*, *fit=False*)
Sets weights of each coord to one over the line amplitude.

/!\ only use if lines are visible pre-stacking

Parameters

- **coords** – A coordList object of all target coordinates.
- **fit** – If set to **True**, spectra will be extracted from each coordinate pixel and fited with a gaussian. The gaussian's extracted amplitude will be used as the line's amplitude.

  If set to **False**, the line's amplitude is set to the value of the brightest spectral bin of each coordinate pixel.

LineStacker.line_image.**calculate_sigma2_weights**(*coords*, *maskradius=0.0*)
Computes standard deviation of data cubes and sets weights to one over sigma**2.

Parameters

- **coords** – A coordList object of all target coordinates.

- **maskradius** – Radius (in pixel) of the mask, centered on coordinate center, to avoid including pixels close to source in noise computation.

LineStacker.line_image.**calculate_sigma2_weights_spectral**(*coords*, *maskradius=0.0*)

Computes standard deviation of data cubes in every spectral channel and sets weights to one over sigma**2.

Parameters

- **coords** – A coordList object of all target coordinates.
- **maskradius** – Radius (in pixel) of the mask, centered on coordinate center, to avoid including pixels close to source in noise computation.

LineStacker.line_image.**stack**(*coords*, *outfile='stackResult.image'*, *stampsize=32*, *imagenames=[]*, *method='mean'*, *spectralMethod='z'*, *weighting=None*, *maskradius=0*, *psfmode='point'*, *primarybeam=None*, *fEm=0*, *chanwidth='default'*, *plotIt=False*, *regridFromZ=False*, *regridMethod='scaleToMin'*, *saveSubCubes=False*, ***kwargs*)

Performs line stacking in the image domain. returns: Estimate of stacked flux assuming point source.

Parameters

- **coords** – A coordList object of all target coordinates. outfile Target name for stacked image.
- **stampsize** – size of target image in pixels
- **imagenames** – Name of images to extract flux from.
- **method** – 'mean' or 'median', will determined how pixels are calculated
- **spectralMethod** – Method to select the central frequency of the stack. The corresponding value should be found in the 3rd column of the coord file. Possible methods are:

  'z': the redshift of the observed line, if used **fEm** (emission frequency) must be informed as an argument of this Stack function.

  'centralFreq': the (observed) central frequency, or velocity.

  'channel': to dirrectly input the channel number of the center of the stack.

- **weighting** – used if method set to **'mean'**, possible values are:

  'sigma2', weights are set to 1/sigma**2 where sigma is the standard deviation of the corresponding data cube (excluding masked region). See **calculate_sigma2_weights**

  'sigma2F', similar to **sigma2** except weights are individually computed for each spectral bin. See **calculate_sigma2_weights_spectral**

  '1/A', **/!\ only use if lines are visible pre-stacking**, weights are set to one over the line amplitude. See **calculate_amp_weights**

  None, using weights in coords (set to 1 by default).

- **maskradius** – allows blanking of centre pixels in weight calculation
- **psfmode** – Allows application of filters to stacking, currently not supported.
- **primarybeam** – only applies if weighting='pb'
- **fEm** – rest emission frequency of the line,
- **chanwidth** – number of channels of the resulting stack, default is number of channels in the first image.
- **plotIt** – direct plot option

- **regridFromZ** – if set to True new images will be created, regridded to take into account the redshift difference of the different sources. See LineStacker.analysisTools.regridFromZ for a more complete description. NB: ALL IMAGES SHOULD HAVE SAME FREQUENCY BIN originally.
- **regridMethod** – Used if regridFromZ is True. Can be set either to 'scaleToMin' or 'scaleToMax'. In the first case all images are regrided to match the smallest redshift (over-gridding), all images are regridded to match the highest redshift in the other case (under-gridding)
- **saveSubCubes** – If set to True (or to str) sub cubes will be saved as a numpy file. (outSubCubes.npy if set to True, user defined is set to str)

OneD_Stacker

LineStacker.OneD_Stacker Module:

Module for one dimensional line stacking.

class LineStacker.OneD_Stacker.**Image**(*spectrum=[]*, *amp=[]*, *velocities=[]*, *frequencies=[]*, *z=None*, *fEmLine=False*, *centerIndex=None*, *centralVelocity=None*, *centralFrequency=None*, *weights=1*, *name=''*, *fit=False*, *velOrFreq='vel'*)

Image class, each object is a spectrum to stack, containing all necessary information to be stacked they can consist simply of a the flux array, or flux and corresponding spectral values

Parameters

- **coords** – A coordList object of all target coordinates.
- **spectrum** – Spectrum should be of the shape [N,2] or [2,N], where N is the number of spectral bins. The second dimension being the flux and the first the spectral values.
- **amp** – Instead of defining spectrum one can define only the amplitude (flux).
- **velocities=[]** – In the case where amp is defined it is still possible to define the velocites
- **frequencies=[]** – In the case where amp is defined it is still possible to define the frequencies
- **z** – Redshift, one of the possible way to define central frequency. If using redshift **fEmLine** (emission frequency of the line) should be defined. In addition the Image spectra should consist of BOTH amplitudes and spectral information (frequencies or velocites), and not amplitude alone.
- **fEmLine** – Emission frequency of the line, needed if redshift argument is used.
- **centerIndex** – channel index of the line center, an other possible way to define line center.
- **centralFrequency** – (observed) Frequency of the line center, an other possible way to define line center.
- **centralVelocity** – Observed velocity of the line center, an other possible way to define line center.
- **weights** – The weighting scheme to use.

Can be set to **'1/A'** or **'sigma2'**. If **sigma2** is used std of the entire spectra is used UNLESS, **fit** is set to **True**, in which case the central part around the line is excluded from std calculation (central here means one FWHM on each side of the center of the line).

Alternativelly user input can be used, float or list (or array)

- **name** – name of the image, can allow easier identification
- **fit** – If fit is set to **True**, the spectrum will be fitted with a gaussian to try and identify the line center (as well as amplitude if weight is set to 1/A)
- **velOrFreq** – Defining if spectral dimension is velocity or frequency.

freqToVel(*freq*, *z*, *fEmLine*)
Function to go from frequencies to velocites. Requieres rest emission frequency and redshift.

velToFreq(*vel*, *z*, *fEmLine*)
Function to go from velocites to frequencies. Requieres rest emission frequency and redshift.

LineStacker.OneD_Stacker.**Stack**(*Images*, *chansStack='full'*, *method='mean'*, *center='lineCenterIndex'*, *regridFromZ=False*, *regridMethod='scaleToMin'*)
Main (one dimmensional) stacking function.

Requieres list of Image objects.

Parameters

- **Images** – List of images, images have to objects of the Image class (LineStacker.OneD_Stacker.Image).
- **chansStack** – Number of channels to stack. Set to 'full' to stack all channels from all images. User input (int) otherwise
- **method** – stacking method, **'mean'** and **'median'** supported
- **center** – Method to find central frequency of the stack, possible values are:

  "center", to stack all spectra center to center,

  'fit' to use gaussian fitting on the spectrum to determine line center,

  'zero_vel' to stack on velocity=0 bin,

  'lineCenterIndex' use the line center initiated with the image,

  Or dirrectly defined by the user (int)
- **regridFromZ** – if set to True spectra will be regridded to take into account the redshift difference of the different sources. See LineStacker.analysisTools.regridFromZ1D for a more complete description.
- **regridMethod** – Used if regridFromZ is True. Can be set either to **'scaleToMin'** or **'scaleToMax'**. In the first case all spetcra are regrided to match the smallest redshift (overgridding), all spectra are regridded to match the highest redshift in the other case (undergridding).

analysisTools

LineStacker.analysisTools Module:

Module containing statistical/analysis tools for stacking.

,

LineStacker.analysisTools.**bootStraping_Cube**(*coords*, *outfile*, *stampsize=32*, *imagenames=[]*, *method='mean'*, *weighting=None*, *maxmaskradius=0*, *fEm=0*, *chanwidth=30*, *nRandom=1000*, *save='amp'*)

Performs bootstrapping stack of cubes. See stacker.line_image.stack for further information on stack parametres

Parameters

- **coords** – A stacker.coordList object of all target coordinates
- **outfile** – Target name for stacked image
- **stampsize** – size of target image in pixels
- **imagenames** – Name of images to extract cubes from
- **method** – stacking method, 'mean' or 'median'
- **weighting** – weighting method to use if stacking method is mean

  possible values are 'sigma2', 'sigma2F', '1/A', 'None', 1 or user input (float)

  see stacker.line-image for a complete description of weighting methods
- **maskradius** – radius of the mask used to blank the centre pixels in weight calculation
- **fEm** – rest emission frequency of the line
- **chanwidth** – number of channels of the resulting stack
- **nRandom** – number of boostrap itterations
- **save** – data to save at each bootstrap itterations

  possible values are **'all'**, **'amp'**, **'ampAndWidth'** and **'outflow'**

  'all' saves the full stack at each bootstrap itteration /!caution, can be memory expensive

  'amp' saves the amplitude (maximum value of the stack) of the line, at each bootstrap itteration, fastest

  'ampAndWidth' fits the line with a gaussian and saves the corresponding amplitude and width at each bootstrap itteration, can be cpu exppensive

  'ouflow' fits the line with two gaussian components and saves the stack parameters at each bootstrap itteration, can be cpu exppensive

  for 'amp', 'ampAndWidth' and 'ouflow' the line is obtained by summing all pixels inside the stack stamp.

LineStacker.analysisTools.**bootstraping_OneD**(*Images*, *nRandom=1000*, *chansStack='full'*, *method='mean'*, *center='lineCenterIndex'*, *save='all'*)

Performs bootstrapping stack of spectra. See stacker.OneD_Stacker.stack for further information on stack parametres.

Parameters

- **Images** – a list of stacker.OneD_Stacker.Images
- **nRandom** – number of boostrap itterations
- **chansStack** – number of channels to stack, either a fixed number or 'full' for entire spectra
- **method** – stacking method, 'mean' or 'median'

- **center** – Method to center spectra. See stacker.OneD_Stacker.stack for further information on centering methods.
- **save** – data to save at each bootstrap itterations

  possible values are **'all'**, **'amp'** and **'ampAndWidth'**

  'all' saves the full stack at each bootstrap itteration /!caution, can be memory expensive

  'amp' saves the amplitude of the line (maximum value of the stack) at each bootstrap itteration, fastest

  'ampAndWidth' fits the line with a gaussian and saves the corresponding amplitude and width at each bootstrap itteration, can be cpu exppensive.

LineStacker.analysisTools.**noise_estimator**(*imagenames, nRandom=10000, maskradius=0, maskCenter=[0, 0], continuum=True, chanToNoise='random'*)

Estimates noise of image cube.

Parameters

- **imagename** – Path to image
- **nRandom** – Number of itterations maskradius Radius of the mask (in pixels)
- **maskCenter** – Center location of the mask (in pixels). NB: CENTER OF IMAGE IS DEFINED AS [0,0] continuum If set to True points will be taken at random on the any of the spectral channels. Otherwise all channels are probbed independantly.
- **chanTonoise** – If set to a specific (int) value then noise will be extracted from the specified channel. NB: only works if continuum i set to True.

LineStacker.analysisTools.**randomizeCoords**(*coords*, *beam=0*, *lowerLimit='default'*, *upperLimit='default'*)

Function to randomize stacking coordinates, new ramdom position uniformally randomized, centered on the original stacking coordinate

Parameters

- **coords** – A list of stacker.Coord coordinates
- **lowerLimit** – Lower spatial limit (distance from stacking position) to random new random position.

  Default is 5 beams
- **upperLimit** – Upper spatial limit (distance from stacking position) to random new random position.

  Default is 10 beams
- **beam** – beam size, needed if lowerLimit or upperLimit are set to 'default'

LineStacker.analysisTools.**rebin_CubesSpectra**(*coords*, *imagenames*, *regionSize=False*, *widths=False*, *outputName='_SpectralRebinned'*)

Rebin a list of image-cubes so that all width have the same width as the smallest width

/!\ Lines must be visible before stacking to operate rebinning

/!\ Only one coord per image is necessary for rebinning

Parameters

- **coords** – A coordList object of all target coordinates.

- **imagenames** – Name of images to rebin
- **regionSize** – size (in pixels) to extract the spectra from

  If set to False spectra will be extracted solely from the coord pixel
- **widths** – widths of the lines

  If set to 'False' the spectra will be fitted with a gaussian to extract the width
- **outputName** – Suffix of the rebinned cube, to add to the original cube name.

LineStacker.analysisTools.**rebin_OneD**(*images*)

Rebin a list of images so that all width have the same width as the smallest width

/!\ Lines must be visible before stacking to operate rebinning

Parameters **images** – A list of stacker.OneD_Stacker.images

LineStacker.analysisTools.**regridFromZ**(*coords*, *imagenames*, *stampsize=32*, *chanwidth=0*, *fEm=0*, *writeImage=True*, *regridMethod='scaleToMin'*)

Cube regridding function coorecting the observed linewidth change due to redshift. Spectral dimensions are rebinned to a new_size=old_size*zRatio where zRatio=(1+z)/(1+zRef) where z is the observed line redshift and zRef is either the smaller (**scaleToMin**) or bigger (**scaleToMax**) redshift among the studied lines.

Parameters

- **coords** – A coordList object of all target coordinates.
- **imagenames** – Name of images to extract flux from.
- **stampsize** – size of target image in pixels
- **chanwidth** – number of channels of the resulting stack, default is number of channels in the first image.
- **fEm** – rest emission frequency of the line
- **writeImage** – boolean, if set to **True** regridded images will be written to disk (in ./regridedImages)
- **regridMethod** – Method of regridding, can be set either to **'scaleToMin'** or **'scaleToMax'**. In the first case all spetcra are regrided to match the smallest redshift (overgridding), all spectra are regridded to match the highest redshift in the other case (undergridding). **NB**: Using **'scaleToMin'** while having sources at z=0 will lead to errors when writting stacked image to disk.
- **Returns** – List of new regridded image cubes and associated coordinates if writeImage is **True** or list of regridded stamps otherwise.

LineStacker.analysisTools.**regridFromZ1D**(*images*, *regridMethod='scaleToMin'*)

1D regridding function correcting the observed linewidth change due to redshift. Spectra are rebinned to a new_size=old_size*zRatio where zRatio=(1+z)/(1+zRef) where z is the observed line redshit and zRef is either the smaller (**scaleToMin**) or bigger (**scaleToMax**) redshift among the studied lines.

Parameters

- **Images** – List of images, images have to objects of the Image class (LineStacker.OneD_Stacker.Image).
- **regridMethod** – Method of regridding, can be set either to **'scaleToMin'** or **'scaleToMax'**. In the first case all spetcra are regrided to match the smallest redshift (overgridding), all spectra are regridded to match the highest redshift in the other case (undergridding).

- **Returns** – List of new regridded images.

LineStacker.analysisTools.**stack_estimator**(*coords*, *nRandom=100*, *imagenames=[]*, *stampsize=1*, *method='mean'*, *chanwidth=30*, *lowerLimit='default'*, *upperLimit='default'*, ***kwargs*)

Performs stacks at random positions a set number of times.Allows to probe the relevance of stack through stacking random positions as a Monte Carlo process.

Parameters

- **coords** – A coordList object of all target coordinates.
- **nRandom** – Number of itterations
- **imagenames** – Name of imagenames to stack
- **stampsize** – Size of the stamp to stack (because only the central pixel is extracted anyway, this should be kept to default to enhance efficiency)
- **method** – Method for stacking, see LineStacker.line_image.stack
- **chanwidth** – Number of channels in the stack
- **lowerLimit** – Lower spatial limit (distance from stacking position) to randomize new stacking position. Default is 5 beams
- **upperLimit** – Upper spatial limit (distance from stacking position) to randomize new stacking position. Default is 10 beams
- **returns** – Estimate of stacked flux assuming point source.

LineStacker.analysisTools.**subsample_OneD**(*images*, *nRandom=10000*, *maxTest=<function maximizeSNR>*, ***kwargs*)

Randomly ressamples spectra and grades them accordingly to a given grade function. Returns the grade of each spectra as a dictionnary.

Parameters

- **images** – A list of stacker.OneD_Stacker.images
- **nRandom** – Number of itterations of the Monte-Carlo process
- **maxTest** – function test to grade the sources build your own, or use existing: **maximizeAmp**, **maximizeSNR**, **maximizeOutflow**. Default is to maximize amplitude (amplitude being simply maximum value of spectra)

tools

LineStacker.tools.**ProgressBar**(*n*, *total*)

show progress of a process :param n: current step :param total: total number of steps

LineStacker.tools.fit Module:

Basic custom fit module for LineStacker.

LineStacker.tools.fit.**DoubleGauss**(*f=[]*, *amp1=0*, *amp2=0*, *f01=0*, *f02=0*, *sigma1=0*, *sigma2=0*)

Sum of two Gaussian function

,

LineStacker.tools.fit.**DoubleGaussFit**(*fctToFit=[]*, *fullFreq=[]*, *sigma1='default'*, *sigma2='default'*, *ampScale=0.1*, *returnAllComp=False*, *returnInfos=False*, *returnError=False*)

Double Gaussian (sum of two Gaussians) fitting function. Return order: fitting function, (first Gaussian, second Gaussian), (fitting parameters), (error on fitting parameters)

Parameters

- **fctToFit** – One dimensionnal array to be fitted.
- **fullFreq** – Spectral dimension array. **Not requiered**
- **sigma1** – Initial width of the first Gaussian component for fitting, **'default'** is 3 bins.
- **sigma2** – Initial width of the second Gaussian component for fitting, **'default'** is 6 bins.
- **ampScale** – Initial amp ratio (between the two Gaussian components) for fitting, **default** is 0.1.
- **returnAllComp** – If set to **True** function returns both individual components.
- **returnInfos** – If set to **True** function returns fitted parameters.
- **returnError** – If set to **True** function returns error on fitted parameters.

LineStacker.tools.fit.**GaussFit**(*fctToFit=[]*, *fullFreq=[]*, *sigma='default'*, *returnInfos=False*, *returnError=False*)

Simple Gaussian fitting function. Return order: fitting function, (fitting parameters), (error on fitting parameters)

Parameters

- **fctToFit** – One dimensionnal array to be fitted
- **fullFreq** – Spectral dimension array. **Not requiered**
- **sigma** – initial width for fitting, **'default'** is 3 bins
- **returnInfos** – If set to **True** function returns fitted parameters
- **returnError** – If set to **True** function returns error on fitted parameters

LineStacker.tools.fit.**gaussFct**(*x*, *a*, *x0*, *sigma*, *constant=0*)

Basic Gaussian function

CHAPTER

THREE

EXAMPLE

All examples presented bellow, as well as the example data sets used to execute them, are provided in LineStacker/Example. Methods used for plotting can be found in LineStacker/Example.

1D LineStacker

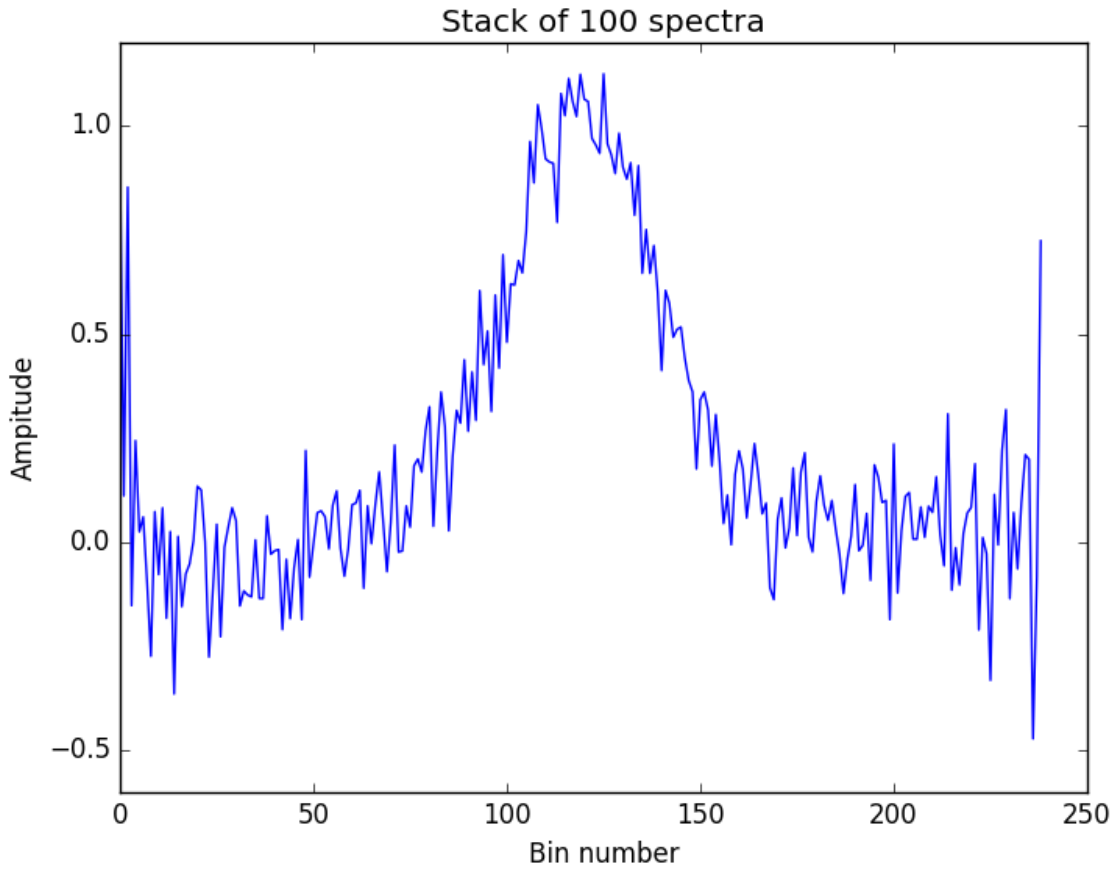
Basic usage

One dimensional example use of Line-Stacker

```
#This files shows basic examplpe use of
#one dimmensional stacking with LineStacker
import numpy as np
import LineStacker.OneD_Stacker

#In this example we stack 100 spectra that are located in the data folder
#and are named spectra_'+str(i) for i in range(100),
#the lines are identified with a central velocity,
#which can be found in the file 'data/central_velocity_'
#lines can be idenfified in many ways however,
#see LineStacker.OneD_Stacker.Stack for more information
numberOfSpectra=100
allImages=([0 for i in range(numberOfSpectra)])
for i in range(numberOfSpectra):
    tempSpectra=np.loadtxt('data/spectra_'+str(i))
    tempCenter=np.loadtxt('data/central_velocity_'+str(i))
    #initializing all spectra as Image class,
    #this is necessary to use OneD_Stacker.Stack
    allImages[i]=LineStacker.OneD_Stacker.Image(spectrum=tempSpectra, centralVelocity=tempCenter)
stacked=LineStacker.OneD_Stacker.Stack(allImages)
```

With a resulting plot looking like this:

,

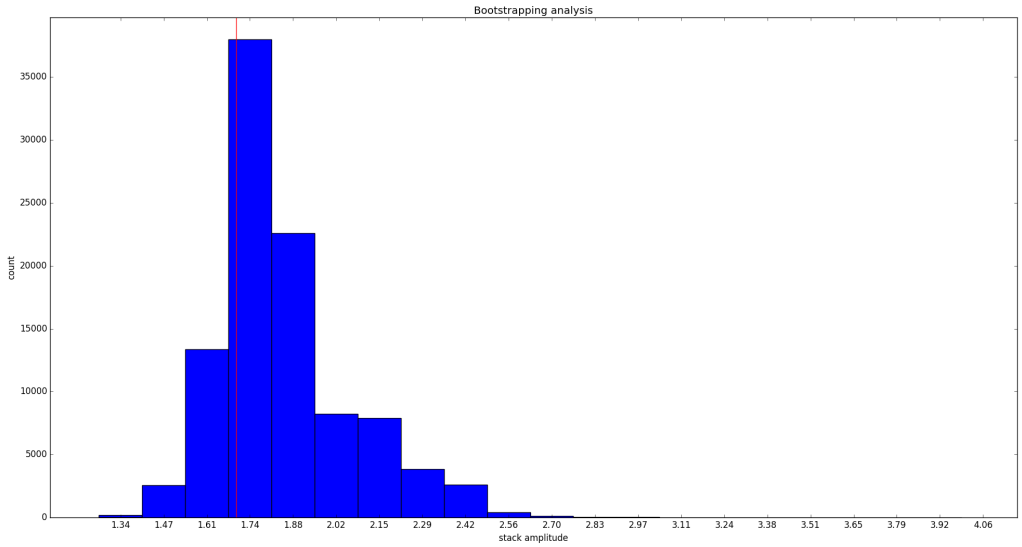


Extended example

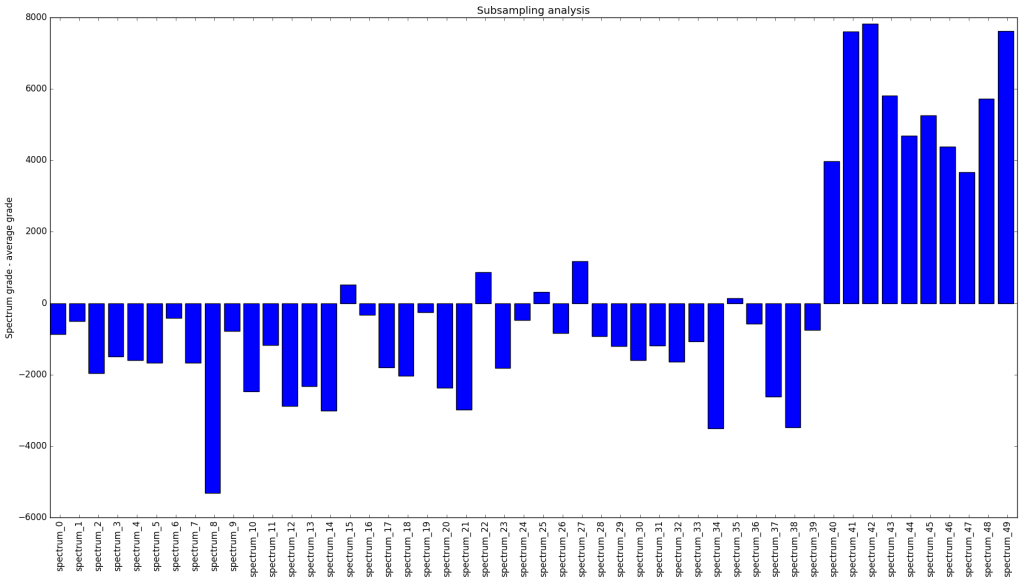
```
#This files shows a more extended examplpe use of
#one dimmensional stacking with LineStacker.
#Showcasing the bootstrapping and subsampling tools.
import numpy as np
import LineStacker.OneD_Stacker
import LineStacker.analysisTools
import LineStacker.tools.fit as fitTools
import matplotlib.pyplot as plt


#==================================================
#                       Stack
#==================================================

#In this example all spectra are stored in the dataExtended folder
#There are 50 spectra, 40 having a line amplitude of 1mJy (number 0 to 39)
#while 10 have a line amplitude of 10mJy (number 40 to 49).
#Spectra are simply called spectra_NUMBER with NUMBER going from 0 to 49.
#In addition to each spectra is associated
#an other file called central_velocity_NUMBER
#containing the associated central velocity.
#
```

```
#In this example we will perform a median stack of all the spectra.
#In a second time we will perform a bootstrapping analysis,
#indicating the presence of outlayers.
#Finally we will perform a subsampling analysis
#to identify these outlayers.

numberOfSpectra=50
allNames=[]
allImages=([0 for i in range(numberOfSpectra)])
#We start by initializing all spectra as LineStacker.OneD_Stacker.Image
#Which is requiered to do one dimensional stacking
for i in range(numberOfSpectra):
    tempSpectra=np.loadtxt('dataExtended/spectra_'+str(i))
    tempCenter=np.loadtxt('dataExtended/central_velocity_'+str(i))
    allImages[i]=LineStacker.OneD_Stacker.Image(    spectrum=tempSpectra,
                                                    centralVelocity=tempCenter,
                                                    name='spectrum_'+str(i))
    #While the name handling is not requiered it allows easier
    #treatment and understanding of the subsampling.
    allNames.append(allImages[i].name)

#Here we perform the actuall stack
stacked=LineStacker.OneD_Stacker.Stack(allImages, method='median')
#The stack amplitude is stored for later vizualization
stackAmp=np.max(stacked[0])

#==================================================
#              bootstrapping analysis
#==================================================

bootstrapped=LineStacker.analysisTools.bootstraping_OneD(   allImages,
                                                            save='amp',
                                                            nRandom=100000,
                                                            method='median')

#plotting
import matplotlib.pyplot as plt
fig=plt.figure()
ax=fig.add_subplot(111)
counts, bins, patches=ax.hist(bootstrapped, bins=20)
#for vizualization purposes...
ax.set_xticks(bins+(bins[1]-bins[0])/2)
ax.set_xticklabels(([str(bin)[:4] for bin in bins]))
#A red vertical line indicates
#the value of the amplitude of the original stack
ax.axvline(x=stackAmp, color='red')
ax.set_xlabel('stack amplitude')
ax.set_ylabel('count')
ax.set_title('Bootstrapping analysis')
fig.show()
```

,




```
#================================================
#               subsampling analysis
#================================================

subSample=LineStacker.analysisTools.subsample_OneD( allImages,
                                                    nRandom=100000,
                                                    method='median')

#plotting
fig=plt.figure()
ax=fig.add_subplot(111)
meanSub=np.mean(([subSample[i] for i in subSample]))
ax.bar(range(len(subSample)), ([subSample[name]-meanSub for name in allNames]))
ax.set_xticks(np.arange(len(allNames))+0.5)
ax.set_xticklabels(allNames, rotation='vertical')
ax.set_ylabel('Spectrum grade - average grade')
ax.set_title('Subsampling analysis')
fig.show()
```



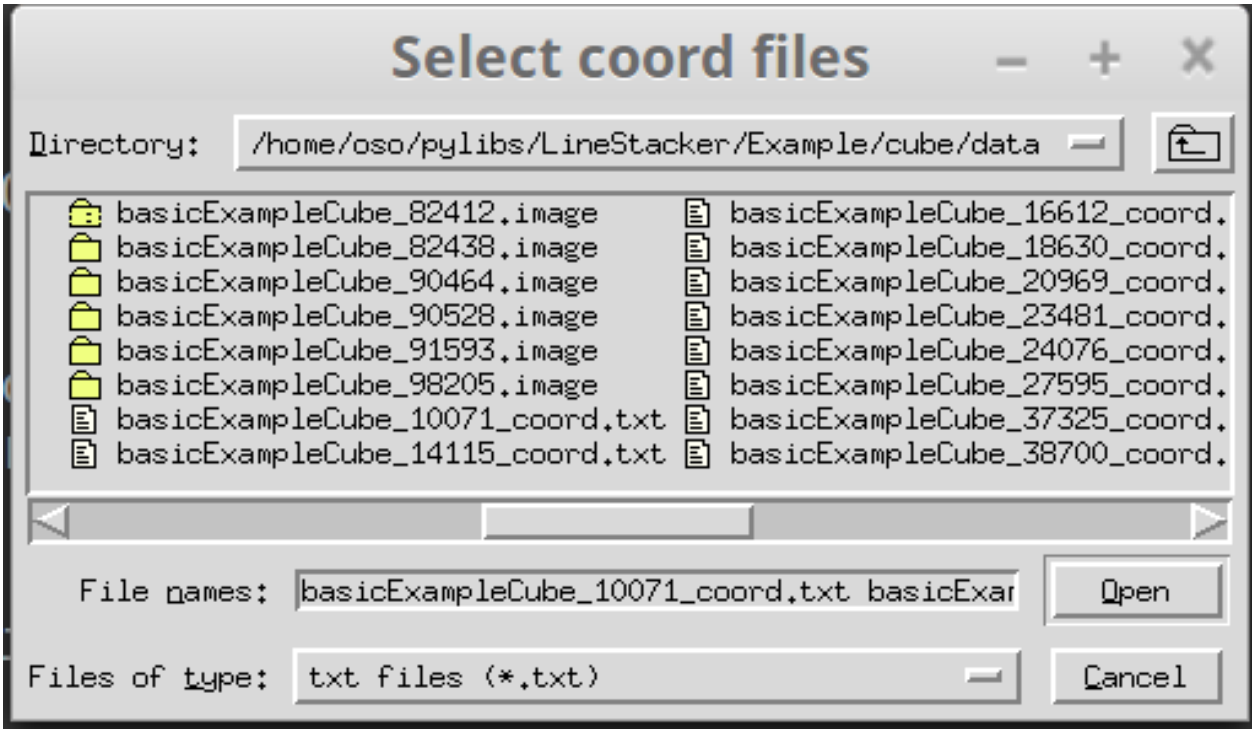
Cube LineStacker

Basic usage

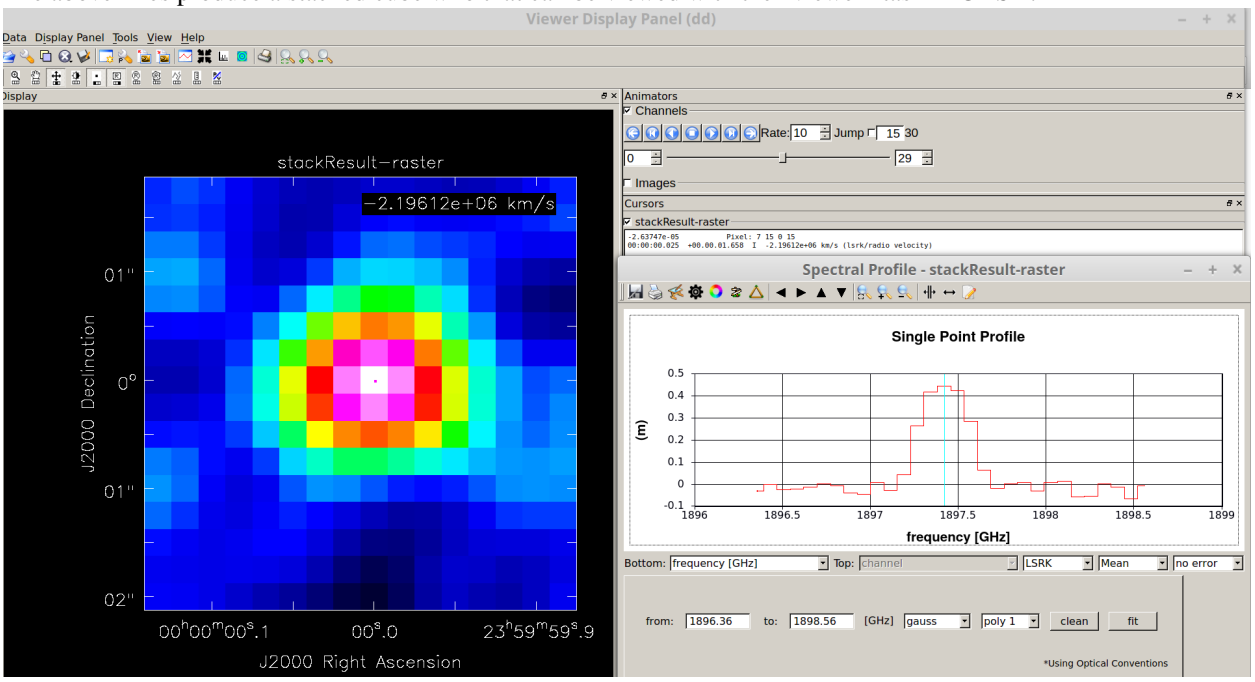
```
import LineStacker
import LineStacker.line_image

#coordinates files are selected using the GUI
coordNames=LineStacker.readCoordsNamesGUI()
coords=LineStacker.readCoords(coordNames)

#image names are identical to coordinates files,
#with '.image' replacing '_coords.txt'
imagenames=([coord.strip('_coords.txt')+'.image' for coord in coordNames])

#because redshift is used to identify the line center,
#the emission frequency is also provided
stacked=LineStacker.line_image.stack(   coords,
                                        imagenames=imagenames,
                                        fEm=1897420620253.1646,
                                        stampsize=16)
```

Here the GUI is used to select the coordinates files, it looks like this:

,



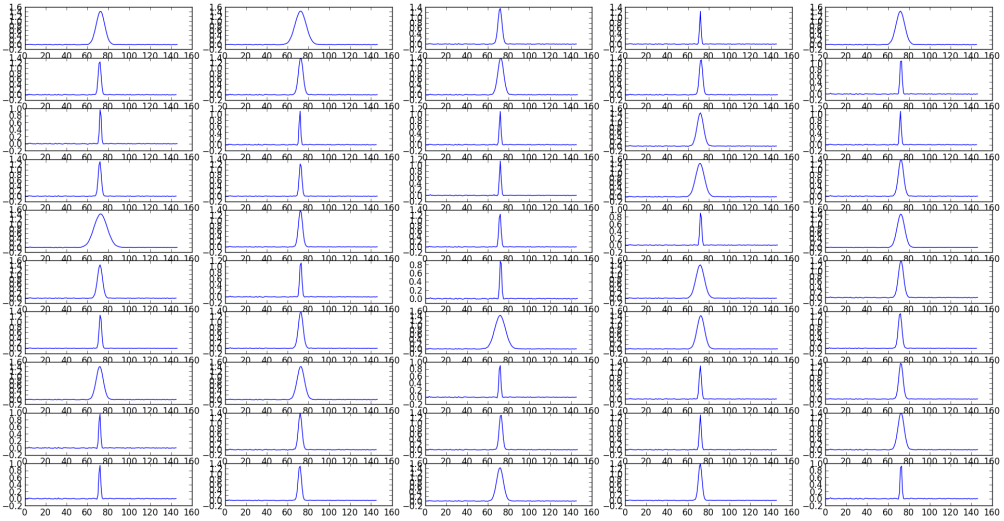
The above lines produce a stacked cube who that can be viewed with the "viewer" task in CASA:



Extended example

```
import LineStacker
import LineStacker.line_image
import LineStacker.analysisTools
import numpy as np
from taskinit import ia
```

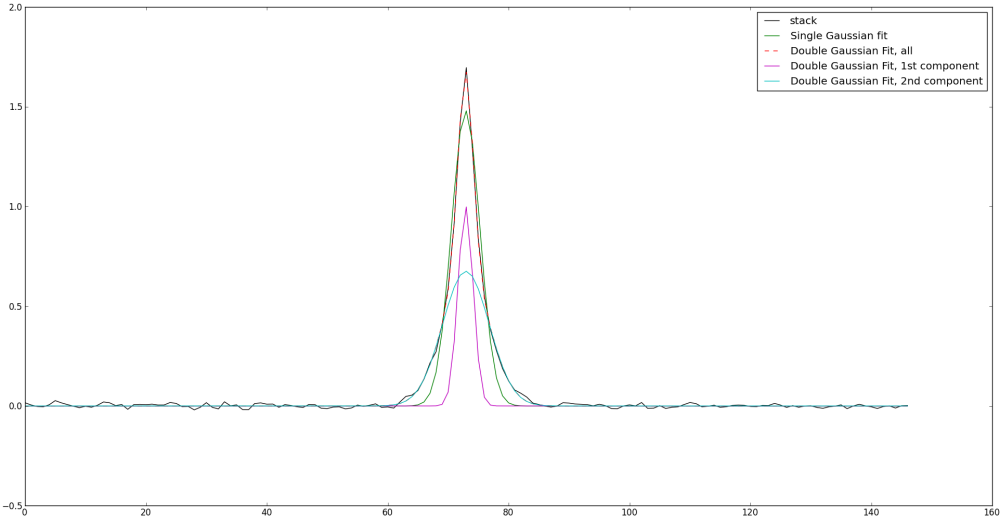
```
import LineStacker.tools.fit as fitTools
import matplotlib.pyplot as plt


#coordinates files are selected using the GUI
coordNames=LineStacker.readCoordsNamesGUI()
coords=LineStacker.readCoords(coordNames)

#image names are identical to coordinates files,
#with '.image' replacing '_coords.txt'
imagenames=([coord.strip('_coords.txt')+'.image' for coord in coordNames])

#because redshift is used to identify the line center,
#the emission frequency is also provided
stacked=LineStacker.line_image.stack(   coords,
                                        imagenames=imagenames,
                                        fEm=1897420620253.1646,
                                        stampsize=8)

#showing every spectra
#(spectra are extracted from the central 5x5 pixels of each cube)
fig=plt.figure()
for (i,image) in enumerate(imagenames):
    ia.open(image)
    pix=ia.getchunk()
    ia.done()
    xlen=pix.shape[0]
    spectrum=np.sum(pix[int(xlen/2)-2:int(xlen/2)+3,
                        int(xlen/2)-2:int(xlen/2)+3,
                        :,
                        :], axis=(0,1,2))
    ax=fig.add_subplot(10,5,i+1)
    ax.plot(spectrum)
fig.show()
```



```
#the stack results are stored in the cube stackResult.image by default
#Here we open and extract the stacked spectra from the stacked cube,
```

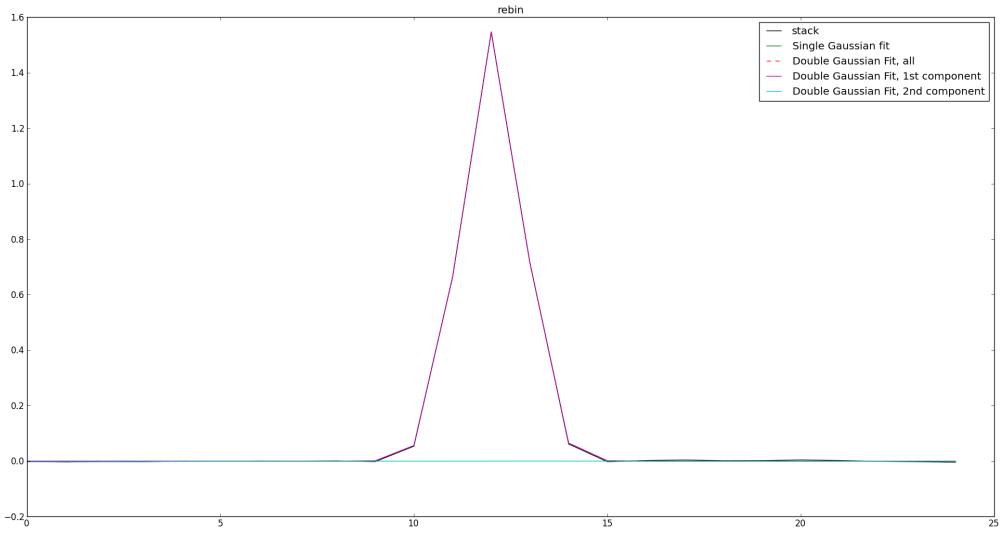
,

```
#and fit it with 1 and 2 Gaussians
#Both the spectrum and the fits are then plotted.
ia.open('stackResult.image')
stackResultIm=ia.getchunk()
ia.done()
integratedImage=np.sum(stackResultIm, axis=(0,1,2))

fited=fitTools.GaussFit(fctToFit=integratedImage)
doubleFited=fitTools.DoubleGaussFit(fctToFit=integratedImage, returnAllComp=True)

fig=plt.figure()
ax=fig.add_subplot(111)
ax.plot(integratedImage,'k', label='stack')
ax.plot(fited, 'g', label='Single Gaussian fit')
ax.plot(doubleFited[0],'r--', label='Double Gaussian Fit, all')
ax.plot(doubleFited[1],'m', label='Double Gaussian Fit, 1st component')
ax.plot(doubleFited[2],'c', label='Double Gaussian Fit, 2nd component')
ax.legend()
fig.show()
```




```
#Cubes are rebinned,
#here the linewidths used for rebenning
#are automatically identified using Gaussian fitting
rebinnedImageNames=LineStacker.analysisTools.rebin_CubesSpectra(    coords,
                                                                    imagenames)

#Rebinned cubes are then stacked
rebinnedStack=LineStacker.line_image.stack(   coords,
                                                    imagenames=rebinnedImageNames,
                                                    fEm=1897420620253.1646,
                                                    stampsize=8)
#Similarly to the previous stack, spectrum is extracted, fited and plotted.
ia.open('stackResult.image')
stackResultIm=ia.getchunk()
ia.done()
integratedImage=np.sum(stackResultIm, axis=(0,1,2))
```

```
fited=fitTools.GaussFit(fctToFit=integratedImage)
doubleFited=fitTools.DoubleGaussFit(fctToFit=integratedImage, returnAllComp=True)

fig=plt.figure()
ax=fig.add_subplot(111)
ax.plot(integratedImage,'k', label='stack')
ax.plot(fited, 'g', label='Single Gaussian fit')
ax.plot(doubleFited[0],'r--', label='Double Gaussian Fit, all')
ax.plot(doubleFited[1],'m', label='Double Gaussian Fit, 1st component')
ax.plot(doubleFited[2],'c', label='Double Gaussian Fit, 2nd component')
ax.set_title('rebin')
ax.legend()
fig.show()
```



,

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

,

L