Line-Stacker Documentation

Release beta

Jean-Baptiste Jolly

June 11, 2019

CONTENTS

Line-Stacker is a new open access tool for stacking of spectral lines. Line-Stackeris 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

main

Library to stack interferometric images.

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.

setWeightToX(*X*, *chanWidth=0*)
sets the weight of the image to X,

setZeroWeightLeft(*numberOfZeroLeftF*, *chanWidth*)
function used to ignore spectral bins outside of the stacking spectral window

setZeroWeightRight(*numberOfZeroRightF*, *chanWidth*)
function used to ignore spectral bins outside of the stacking spectral window

class LineStacker.**CoordList**(*imagenames=[]*, *coord_type='physical'*, *unit='rad'*)
Extended list to contain list of coordinates.

LineStacker.**getPixelCoords**(*coords*, *imagenames*)
Creates pixel coordinate list from a physical coordinate list and a list of images. :param coords: a list of stacker.Coord coordinates :param imagenames: a list of images' paths

LineStacker.**randomCoords**(*imagenames*, *ncoords=10*)
Randomize a set of coordinates anywhere on any images :param imagenames: a list of images paths :param 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 :param coords: list of original coordinates (stacker.Coord instances) :param beam: beam size is radians,

new random coordinates will be at a minimum distance beam from the original coordinates

Parameters **maxBeamRange** – maximum distance from original coordinates at which new coordinates can be located, in units of beams default is 5

LineStacker.**readCoords**(*coordfiles*, *unit='deg'*, *lineON=True*)
Reads coordinate files from disk and produces a list. /!To each image should be associated one single coord file. :param coordfiles: List of path to coordinate files. Files in csv format. x and y should

be in J2000. If using to stack line, redshift should be in the third column. A weight may be added in the last column to weight positions for mean stacking. If set to None stacking position is defined as the center of each cube

Parameters

- **unit** – Unit of input coordinates. Allows two values, 'deg' and 'rad'.
- **lineON** – either True or False, should be set to True if doing line stacking default 0

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.**stack**(*coords*, *outfile='stackResult'*, *stampsize=32*, *imagenames=[]*, *method='mean'*, *spectralMethod='z'*, *weighting=None*, *maskradius=0*, *psfmode='point'*, *primarybeam=None*, *fEm=0*, *chanwidth=30*, *plotIt=False*)

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** – How 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, additionaly 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** – only for method 'mean', if set to None will use weights in coords.
- **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,
- **plotIt** – direct plot option):

OneD_Stacker

LineStacker.OneD_Stacker.**Stack**(*Images*, *chansStack='full'*, *method='mean'*, *center='lineCenterIndex'*, *velOrFreq='vel'*)

Main (one dimmensional) stacking function requieres list of Image objects :param Images: list of images, images have to objects of the Image class (LineStacker.OneD_Stacker.Image) :param chansStack: number of channels to stack

set to 'full' to stack all channels from all images user input (int) otherwise

Parameters

- **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 dirrectly defined by the user (int)
- **velOrFreq** – 'vel' or 'freq', frequency or velocity mode

analysisTools

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

  default is 32
- **imagenames** – Name of images to extract cubes from
- **method** – stacking method, 'mean' or 'median'

  default is 'mean'
- **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

  default is None
- **maskradius** – radius of the mask used to blank the centre pixels in weight calculation

  default is 0
- **fEm** – rest emission frequency of the line
- **chanwidth** – number of channels of the resulting stack

- **nRandom** – number of boostrap itterations

  default is 1000

- **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 (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

  default is 'all'

LineStacker.analysisTools.**bootstraping_OneD**(*Images*, *nRandom=1000*, *chansStack='full'*, *method='mean'*, *center='z'*, *velOrFreq='vel'*, *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

  default is 1000

- **chansStack** – number of channels to stack, either a fixed number or 'full' for entire spectra

  default is 'full'

- **method** – stacking method, 'mean' or 'median'

  default is 'mean'

- **center** – method to center spectra

  possible values are 'z', 'fit', 'zero_vel', 'center' or user input (must be int)

  see stacker.OneD_Stacker.stack for further information on centering methods default is 'z'

- **velOrFreq** – velocity or frequency mode (chose according to your spectral axis type)

  possible values are 'vel' or 'freq'

  default is 'vel'

- **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

default is 'all'

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

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

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

Randomly ressamples spectra and grades them accordingly to a given grade function

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
- **other argument for LineStacker.OneD_Stacker.Stack** (*Any*) –

tools

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

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

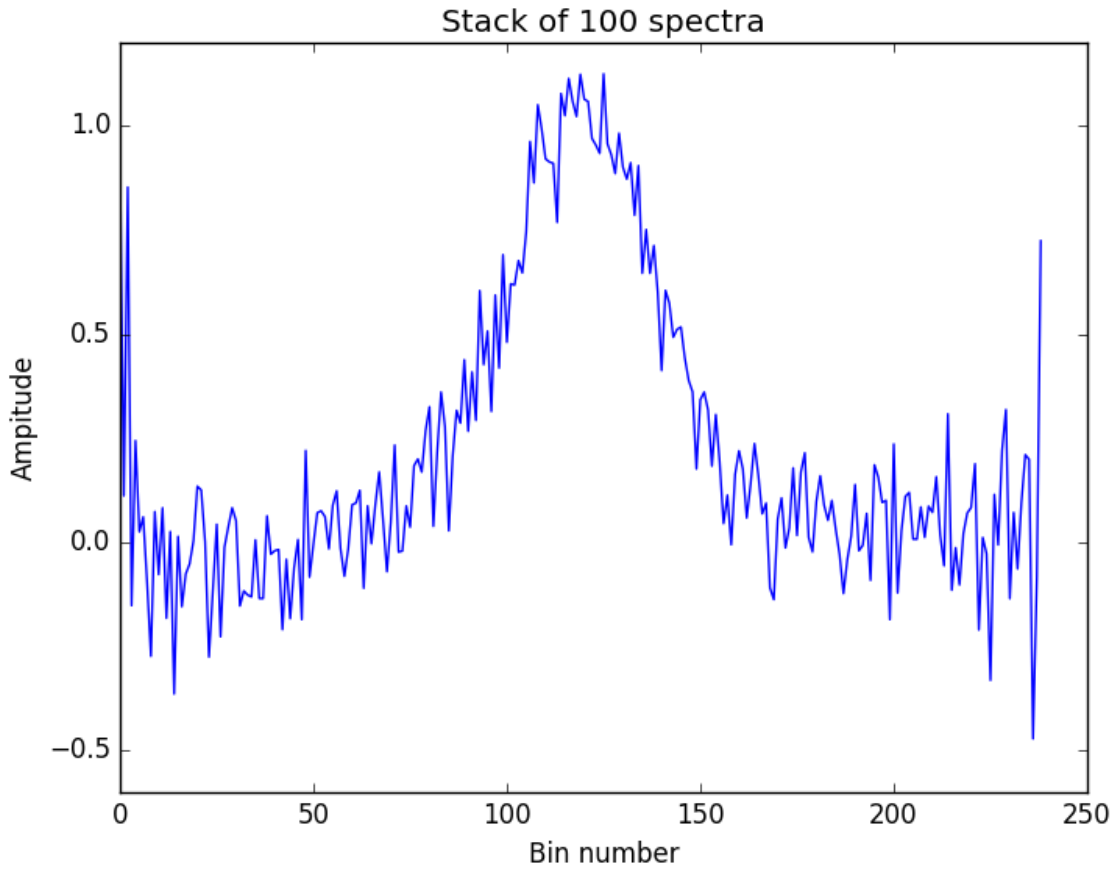
Example

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)
#plotting
import matplotlib.pyplot as plt
fig=plt.figure()
ax=fig.add_subplot(111)
ax.plot(stacked[0])
ax.set_xlabel('Bin number')
ax.set_ylabel('Ampitude')
ax.set_title('Stack of '+str(numberOfSpectra)+' spectra')
fig.show()
```

With a resulting plot looking like this:



CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

L

B

C

G

L

P

R

S

W