Documentation for module stacker

L. Lindroos

October 14, 2014

1 Introduction

This document describes version 1.0 of the module `stacker`. The module is designed to stack interferometric data. Primarily it was designed to allow stacking in the *uv* domain, but supports stacking in image domain. When referencing the usage of this code, please cite Lindroos et al. (2014).

If you have question, or are looking for more information, do not hesitate to contact me at lindroos@chalmers.se .

2 Algorithm

The algorithms used by this code are described in Lindroos et al. (2014). The image based algorithm supports either mean or median stacking, where the flux from sub images are average with either the median or a weighted mean method. It works on a pixel-by-pixel basis in a defined rectangular region (stamp) around the stacking positions.

The *uv*-stacking algorithm is based on aligning the phases of all visibilities to the target sources. Stacked visibilities are calculated as

$$V_{\mathrm{stack}}(u,v,w) = V(u,v,w)\frac{\Sigma_{k=1}^{N} W_k \frac{1}{A_N(l_k,m_k)} e^{\frac{2\pi}{\lambda} i \left(ul_k + vm_k + w(\sqrt{1-l_k^2-m_k^2})\right)}}{\Sigma_{k=1}^{N} W_k} \tag{1}$$

where (l_k, m_k) are the separation of stacking position k from the phase centre, (u, v, w) are the coordinates of the visibility in the *uv* plane, λ is the wavelength, $A_N(l_k, m_k)$ is the primary beam attenuation at stacking position k, and W_k is the weight of the stacking position.

3 Usage

This document provides a quick overview of the available functionalities in `stacker`. The module contains 3 packages: `pb`, `image` and `uv`. The packages `image` and `uv` provides functions to stack in the image and *uv* domain respectively. The package `pb` allows for describing the primary beam model.

For a typical example of how `stacker` can be used see the provided example in "`example/stack_testdata.py`".

3.1 Coordinates

The coordinates are described by a `coordList` object in the module. A `coordList` object can be generated from a csv file with the function `stacker.readCoords`, i.e.,

```
coord = stacker.readCoords(<path to coordinate file>)
```

Syntax of coordinate file should be

```
x1, y1(, weight1)
x2, y2(, weight2)
```

etc. For more info run help(stacker.readCoords) in casapy.

A coords object can also be built from scratch

```
coords = stacker.CoordList()
coords.append(stacker.Coord(x1, y1))
coords.append(stacker.Coord(x2, y2))
```

Note that coordinates here should be give in J2000 radians.

3.2 Primary beam model

The primary beam model can be defined by an instance of `stacker.pb.PrimaryBeamModel` or inherited classes. The c code only support `stacker.pb.MSPrimaryBeamModel` and `stacker.pb.PrimaryBeamModel`. A primary beam model can be generated from a measurement set with the function `stacker.pb.guesspb`.

```
pbmodel = stacker.pb.guesspb(<path to ms file>)
```

If *uv* stacking is run without specifying a primary beam model it will automatically attempt to use `stacker.pb.guesspb`. This has been tested for ALMA and VLA data, and in these cases manually specifiyng the primary beam model is not necessary.

A custom primary-beam model can be used by writing it to a casa image. The image should be centered at $(0., 0.)$. The primary beam is loaded into casa using

```
pbmodel = stacker.pb.MSPrimaryBeamModel(<path to pb file>)
```

3.3 uv

Submodule for stacking in the *uv* domain. Primarily provides two functions `stacker.uv.stack` and `stacker.uv.noise`. The first perform (`stacker.uv.stack`) the actual stacking, and requires an input *uv*-data file and a `coordsList` object as input.

```
import stacker, stacker.uv
coords = stacker.readCoords(<path to coordinate file>)
flux = stacker.uv.stack(coords, <path to ms data file>,
                        <path to save stacked ms file>)
```

For more info on usage run `help(stacker.uv.stack)` in casapy.

The second (`stacker.uv.noise`) calculates noise using a Monte Carlo where random positions are stacked to estimate the noise level. The function will try to recompute weights for the random positions. If you require variable weights which are not simply the primary beam or the noise in a local stamp you will have to re-implement the function.

3.4 image

Submoduls for stacking in the image domain. Provides the same functions as the submodule `uv` except it works fully in the image domain.

```
import stacker, stacker.image
coords = stacker.readCoords(<path to coordinate file>)
flux = stacker.image.stack(coords, <target file>,
                          <stamp size in pixels>, <list of image maps>)
```

The function `stacker.image.stack` writes the stacked image to ¡target file¿, as well as returning a flux estimate which assumes the stacked source is a point source. The input image maps are assumed to have been primary beam corrected. The package can handle a list of images as well as an individual image. An individual image should be specified as a one element list. For more details run help(stacker.image.stack) in casapy.

The `image` package also provides functions to calculate local weights for positions from the stamps surrounding them or from the primary beam. For ALMA data it is most likely best to use the pb weighting mode. This mode requires a primary beam model to calculate the weights, for more details see section 3.2.

4 Some related techniques

This sections describes a few of techniques which are not unique to stacking, but may be useful in the context of stacking.

4.1 Removing bright sources

Bright non-target sources present in the data can significantly impact the stacking result. To reduce this effect, a model of the bright sources should be subtracted from the *uv* data. If the data is imaged with `clean` in casa the model can be subtracted using the task `uvsub`.

The package `modsub` also provides a facility to subtract a model from the *uv* data. As input it takes a casa component list of the model.

```
import stacker.modsub
stacker.modsub.modsub(<path to casa cl file>, <path to in ms file>,
                  <path to out ms file>)
```

This function can also be given a custom primary-beam model defined by the `pb` package.

4.2 Stacking in mosaic data sets

When *uv* stacking in mosaics it is important that the relative weights between the different pointings are scaled correctly. This may not be the case for non contiguous mosaics. In these cases the weights can be recomputed using the casa task `statwt`. This calculates the weight of each visibility using the scatter over nearby visibilities.

If using image-stacking it is most likely best image each pointing separately. The function `stacker.image.stack` can be given a list of all pointings, and will automatically assign proper weights to each position in each pointing.

References

Lindroos L., Knudsen K. K., Vlemmings W., Conway J., Martí-Vidal I., 2014, in prep.