Documention for module stacker

L. Lindroos

September 9, 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.

2 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`. It 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 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`.

4 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.

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.

5 image

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

Also provides functions to calculate local weights for positions from the stamps surrounding them.

The module can handle a list of images as well as an individual image. An individual image should be specified as a one element list.