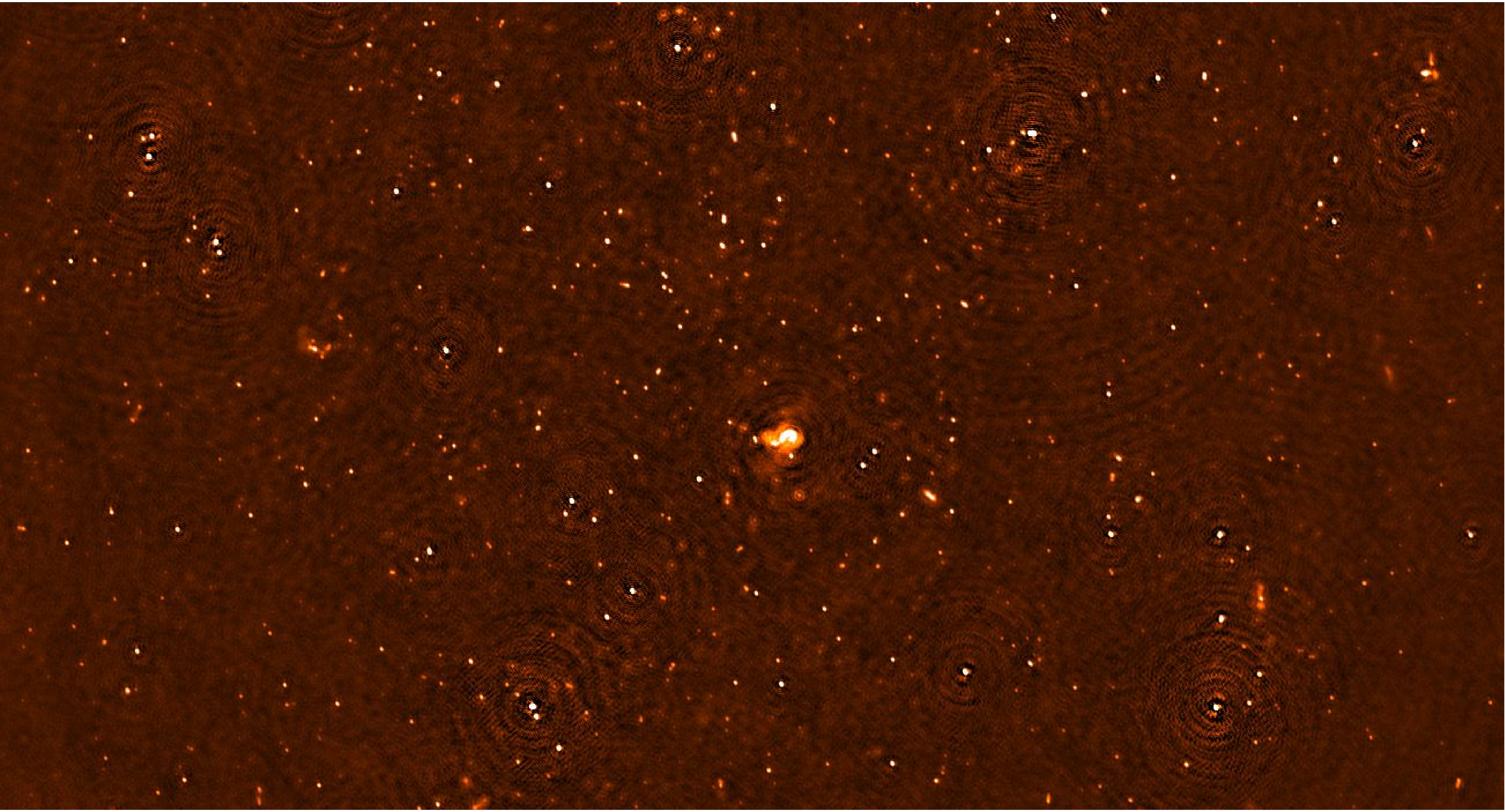
Prefactor tutorial

Pipeline calibration (and imaging) of P23



Wendy Williams

LOFAR data school 2018

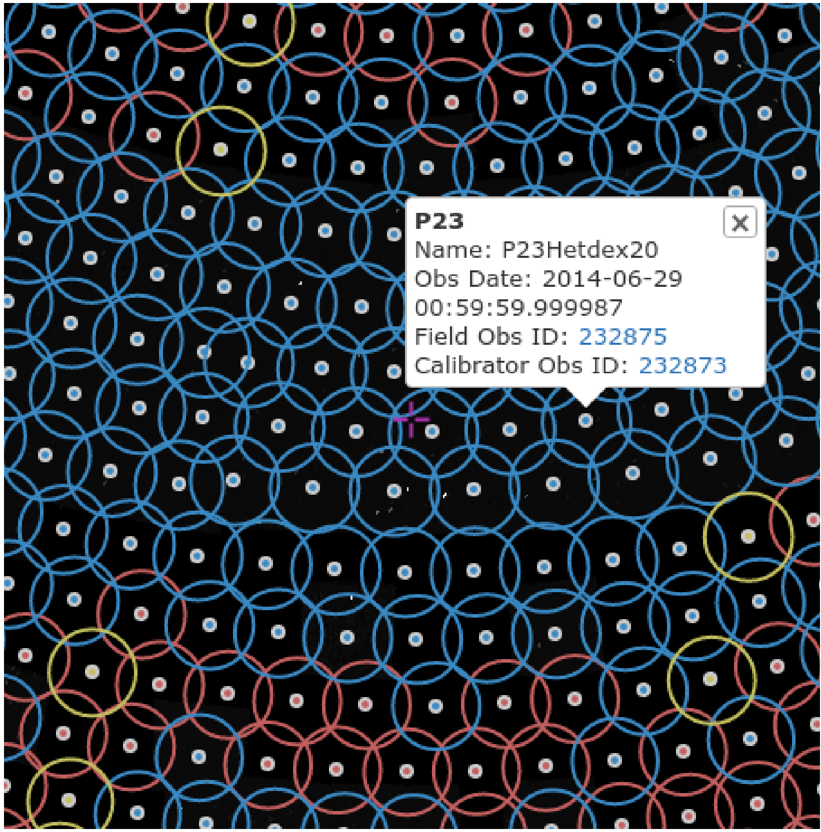
Aims

- In this tutorial we will use prefactor, built in the genericpipeline framework, to:
  - Calibrate the calibrator
  - Transfer the calibrator solutions to the target field
  - ‘Self’-calibrate the target field against a global sky model (from TGSS)
  - Make an image of the target field (not using prefactor)

This tutorial is written in bash (equivalents can be done in tcsh) to be run on CEP3

The Data

- The data for this tutorial come from a standard LOFAR surveys observation
  - 8hr with two target pointings in two simultaneous beams
  - 48 MHz bandwidth
  - bookended by two 10 min calibrator observations
- we will
  - use one of the calibrator observations for the calibration
  - look at one target beam (P23)
  - consider only a subset of the data (in the interest of processing times)



The Data

- The RO preprocessed data can be found on CEP3
  - The calibrator is 3C196 with obsid L232873
    - /data010/scratch/wwilliams/lds18_pft/data/L232873
    - containing 100 subbands (25GB)
  - The target is field P23 with obsid L232875
    - /data010/scratch/wwilliams/lds18_pft/data/L232875
    - containing 20 subbands (228GB)

setup

- Activate a dummy session using the reservation id and the your node:
  - > srun -A lofar_school2018 –reservation=lofar_school2018_114 -N 1 -w lof0XX -u bash -i
- Log into the node in a new terminal tab (you may wish to log in in a second terminal tab as well to monitor/inspect things while the pipeline is running)
  - > ssh -AYCt lodsXX@portal.lofar.eu "ssh -AYCt lodsXX@lhdhead"
  - > ssh -Y lofXX
- make a directory for working in
  - > mkdir -p /data/scratch/<username>/pf_tutorial/
  - > cd /data/scratch/<username>/pf_tutorial/
- make a directory for the pipeline to run in
  - > mkdir pipeline
  - > cd pipeline

Generic pipeline framework

- Part of the LOFAR software stack
  - > module load lofar
- See http://www.astron.nl/citt/genericpipeline/
- To run the pipeline:
  - > genericpipeline.py <parset_file> -v -d -c <config_file>
    - -v for verbosity
    - -d for debug output
      - Produces a LOT of output, but useful for debugging
    - The config file contains some global pipeline settings
    - The workflow (list of tasks/steps) is all contained in the parset

Generic pipeline config

- you can find where the particular lofar installation you are using is located with the LOFARROOT environment variable
  - > echo $LOFARROOT
  - /opt/cep/lofar/lofar_versions/LOFAR-Release-3_1_2/lofar_build/install/gnucxx11_opt
- make a copy of the default pipeline config file
  - > cp $LOFARROOT/share/pipeline/pipeline.cfg .

pipeline.cfg

```
[DEFAULT]
lofarroot = /opt/cep/lofar/lofar_versions/LOFAR-Release-3_1_2/lofar_build/install/gnucxx11_opt
casaroot = /opt/cep/casacore/builds/casacore-2.3.0/build/gnucxx11_opt
pyraproot = /opt/cep/casacore/builds/python-casacore-2.1.2-2.3.0
hdf5root =
wcsroot =
aoflaggerroot=/opt/cep/aoflagger/aoflagger-2.10.0/build
pythonpath = /opt/cep/lofar/lofar_versions/LOFAR-Release-3_1_2/lofar_build/install/gnucxx11_opt/lib64/python2.7/site-packages
runtime_directory = %(lofarroot)s/var/run/pipeline
recipe_directories = [%(pythonpath)s/lofarpipe/recipes]
working_directory = /data/scratch/lofarbuild
task_files = [%(lofarroot)s/share/pipeline/tasks.cfg]

[layout]
job_directory = %(runtime_directory)s/%(job_name)s

[cluster]
clusterdesc = %(lofarroot)s/share/cep2.clusterdesc

[deploy]
engine_ppath = %(pythonpath)s:%(pyraproot)s/lib:/opt/cep/pythonlibs/lib/python/site-packages
engine_lpath = %(lofarroot)s/lib:%(casaroot)s/lib:%(pyraproot)s/lib:%(hdf5root)s/lib:%(wcsroot)s/lib

[logging]
log_file = %(lofarroot)s/var/log/pipeline-%(job_name)s-%(start_time)s.log
xml_stat_file = %(lofarroot)s/var/log/pipeline-%(job_name)s-%(start_time)s-statistics.xml

[feedback]
# Method of providing feedback to LOFAR.
# Valid options:
#    messagebus    Send feedback and status using LCS/MessageBus
#    none          Do NOT send feedback and status
method = messagebus
```

[DEFAULT]
lofarroot = /opt/cep/lofar/lofar_versions/LOFAR-Release-3_1_2/lofar_build/install/gnucxx11_opt
casaroot = /opt/cep/casacore/builds/casacore-2.3.0/build/gnucxx11_opt
pyraproot = /opt/cep/casacore/builds/python-casacore-2.1.2-2.3.0
hdf5root =

pipeline.cfg

Since we are sharing CEP3 nodes (5 pairs on each node with 40 cpus) we need to make sure that the pipeline restricts it's use of the cpus available (in addition to settings in the pipeline parsets we will be using). Add the lines:

```
[remote]
method = local
max_per_node = 8
```

- The method is local to work on a local/single machine (default).
- Other methods that can be used when running on other clusters, e.g. pbs_ssh for multinode jobs using the torque system (provided the data are accessible via a shared filesystem).

[feedback]
# Method of providing feedback to LOFAR.
# Valid options:
#   messagebus   Send feedback and status using LCS/MessageBus
#   none   Do NOT send feedback and status
method = messagebus

pipeline.cfg

```
[DEFAULT]
lofarroot = /opt/cep/lofar/lofar_versions/LOFAR-Release-3_1_2/lofar_build/install/gnucxx11_opt
casaroot = /opt/cep/casacore/builds/casacore-2.3.0/build/gnucxx11_opt
pyraproot = /opt/cep/casacore/builds/python-casacore-2.1.2-2.3.0
hdf5root =
```

Change the feedback method to none
[feedback]
method = none

Also change the logging lines so they don't point to the lofarroot area where you can't write anything...
[logging]
log_file = %(runtime_directory)s/%(job_name)s/logs/%(start_time)s/pipeline.log
xml_stat_file = %(runtime_directory)s/%(job_name)s/logs/%(start_time)s/statistics.xml

```
log_file = %(lofarroot)s/var/log/pipeline-%(job_name)s-%(start_time)s.log
xml_stat_file = %(lofarroot)s/var/log/pipeline-%(job_name)s-%(start_time)s-statistics.xml

[feedback]
# Method of providing feedback to LOFAR.
# Valid options:
#    messagebus    Send feedback and status using LCS/MessageBus
#    none          Do NOT send feedback and status
method = messagebus
```

pipeline.cfg

[DEFAULT]
lofarroot = /opt/cep/lofar/lofar_versions/LOFAR-Release-3_1_2/lofar_build/install/gnucxx11_opt
casaroot = /opt/cep/casacore/builds/casacore-2.3.0/build/gnucxx11_opt
pyraproot = /opt/cep/casacore/builds/python-casacore-2.1.2-2.3.0
hdf5root =

Set the working and runtime directories:

working_directory = /data/scratch/<username>/pf_tutorial/pipeline
runtime_directory = /data/scratch/<username>/pf_tutorial/pipeline

Note that these can be separate directories, if you like, but for simplicity let's keep them the same.
The working directory stores things like the pipeline logs (in a logs subdirectory) and the mapfiles (in a mapfiles subdirectory) that are used to tell the pipeline where data is stored.

engine_lpath = %(lofarroot)s/lib:%(casaroot)s/lib:%(pyraproot)s/lib:%(hdf5root)s/lib:%(wcsroot)s/lib

[logging]
log_file = %(lofarroot)s/var/log/pipeline-%(job_name)s-%(start_time)s.log
xml_stat_file = %(lofarroot)s/var/log/pipeline-%(job_name)s-%(start_time)s-statistics.xml

[feedback]
# Method of providing feedback to LOFAR.
# Valid options:
#   messagebus   Send feedback and status using LCS/MessageBus
#   none   Do NOT send feedback and status
method = messagebus

Generic pipeline outputs

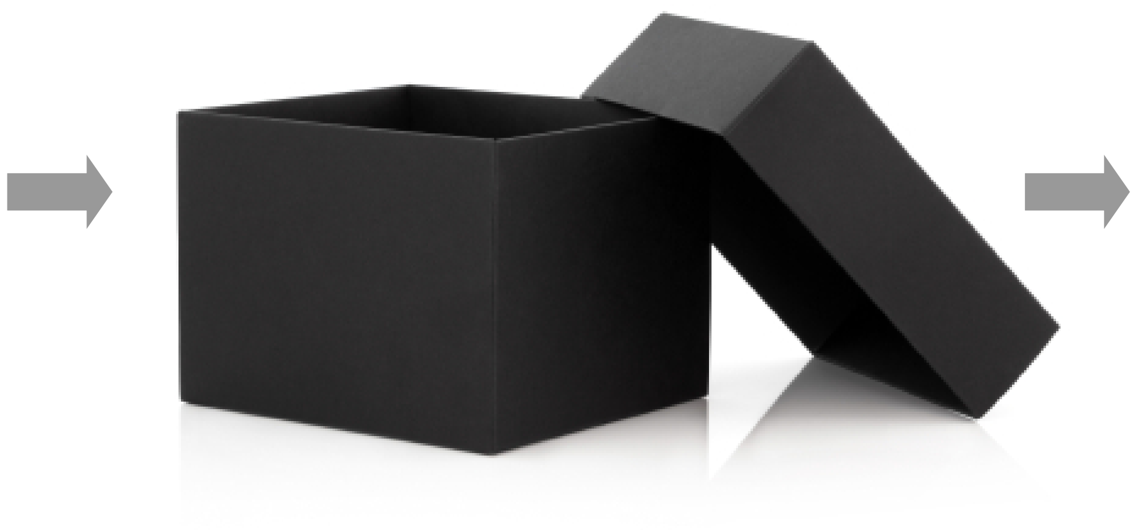
- With no other options, the generic pipeline will create subdirectories with the name of the parset in the runtime and working directories
  - working_directory/job_name
    - mapfiles: contains the files that carry information between pipeline steps or point to data to be used in each step
    - logs: writes a pipeline.log file in a timestamped directory each time the pipeline is run
    - statefile: saves the state of the pipeline run
      - There is a tool in prefactor to manipulate this file: bin/statefile_manipulation.py
      - Can be used to revert to an earlier step
    - parsets: saves some pipeline parsets
  - runtime_directory/job_name
    - Intermediate data products are written here
  - Note we have set these to be the same directory

Prefactor

- Prefactor is…
  - the set of workflows defining the calibration strategy for standard imaging observations
  - a set of genericpipeline parsets (and some additionally defined pipeline steps) and scripts
  - the result of a lot of hard work and development by a number of people
  - under further development
  - easy to use, harder to develop
  - awesome!

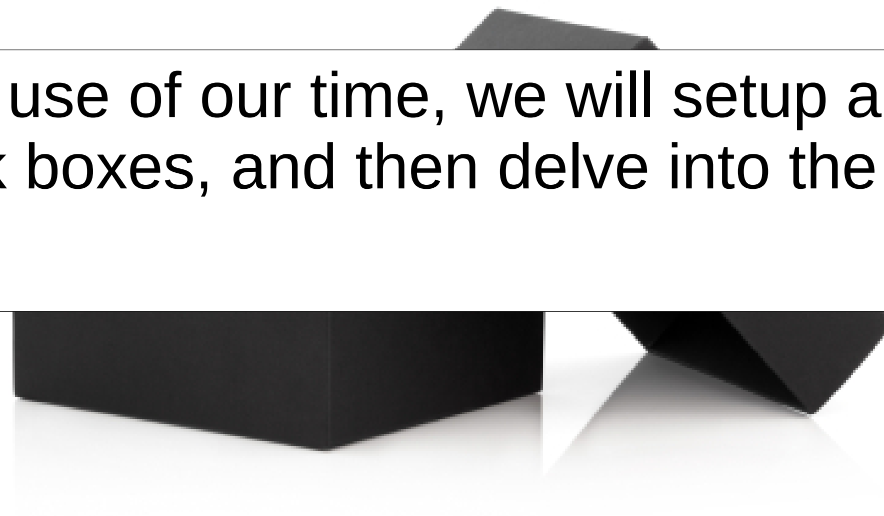
Prefactor

Raw data



Calibrated data
(ready for FACTOR
or DDF)

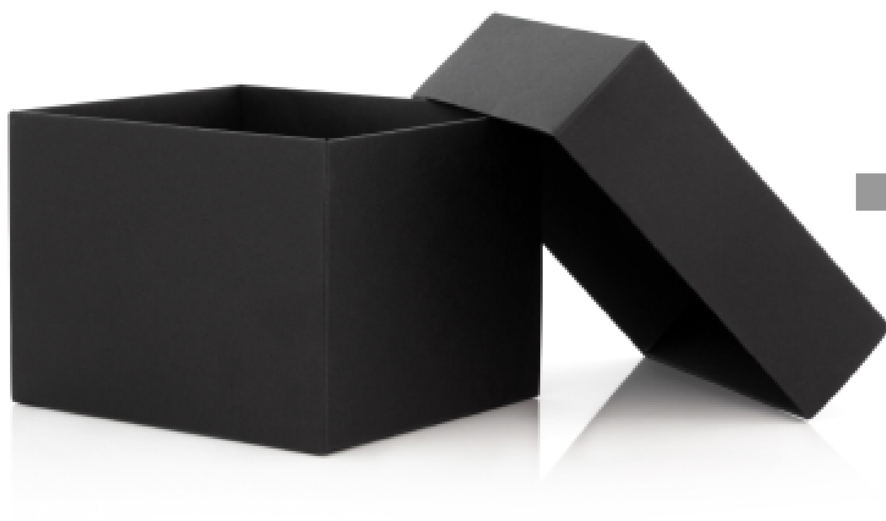
Prefactor



To make the best use of our time, we will setup and start the prefactor pipelines as black boxes, and then delve into the details while they are running...

WARNING!

Garbage



Garbage

Search or jump to… /

Pull requests Issues Marketplace Explore

lofar-astron / prefactor

Unwatch 25 | Unstar 11 | Fork 20

Code | Issues 29 | Pull requests 0 | Projects 0 | Wiki | Insights

Pre facet calibration pipe

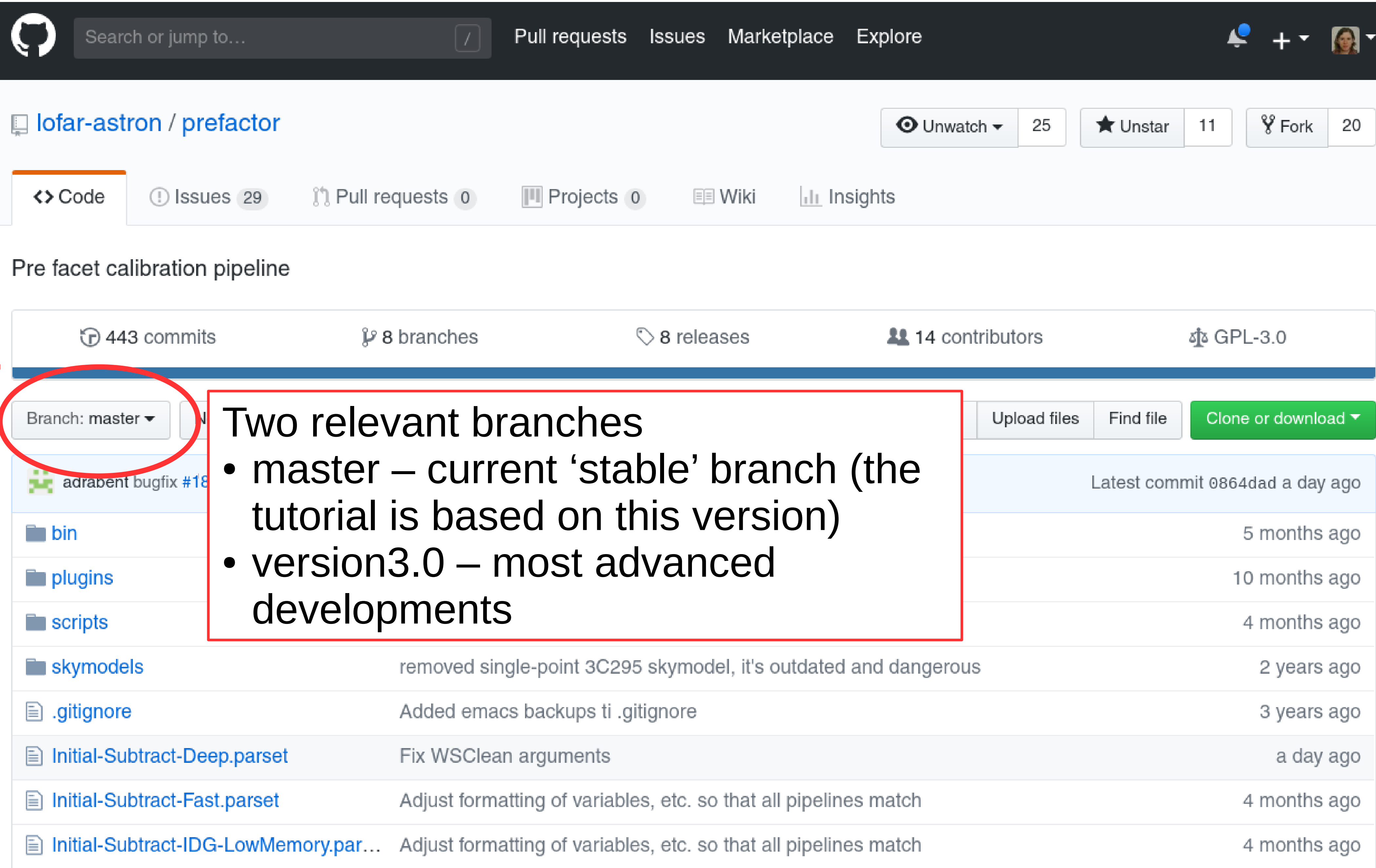
https://github.com/lofar-astron/prefactor

443 commits | 8 branches | 8 releases | 14 contributors | GPL-3.0

Branch: master | New pull request | Create new file | Upload files | Find file | Clone or download

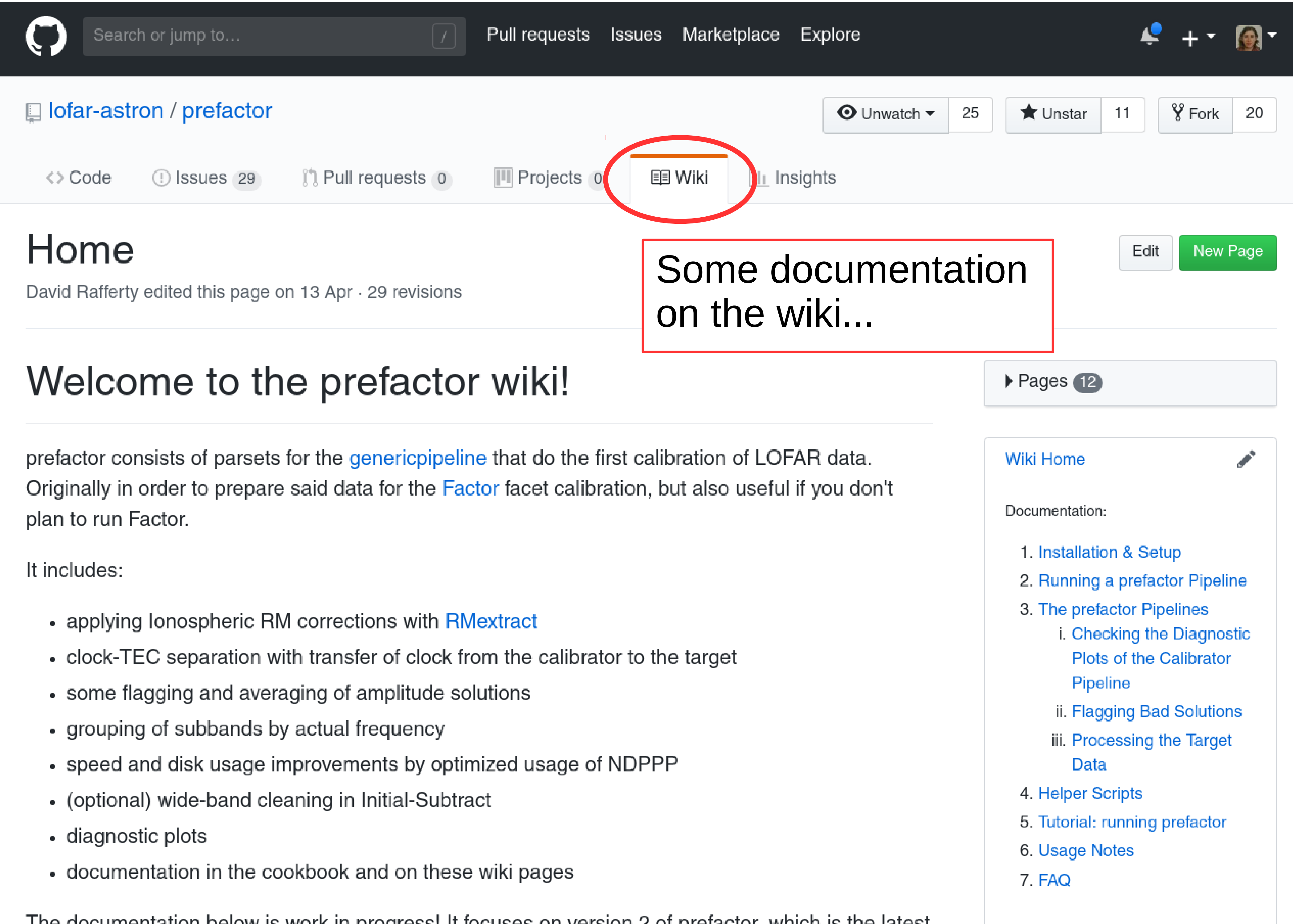
adrabent bugfix #182 — Latest commit 0864dad a day ago

bin	Add header (fixes #148)	5 months ago
plugins	update prefactor for the use of h5parms only	10 months ago
scripts	Use taql + mscal instead of virtual columns	4 months ago
skymodels	removed single-point 3C295 skymodel, it's outdated and dangerous	2 years ago
.gitignore	Added emacs backups ti .gitignore	3 years ago
Initial-Subtract-Deep.parset	Fix WSClean arguments	a day ago
Initial-Subtract-Fast.parset	Adjust formatting of variables, etc. so that all pipelines match	4 months ago
Initial-Subtract-IDG-LowMemory.par…	Adjust formatting of variables, etc. so that all pipelines match	4 months ago



Two relevant branches

- master – current 'stable' branch (the tutorial is based on this version)
- version3.0 – most advanced developments



Search or jump to... / Pull requests Issues Marketplace Explore

lofar-astron / prefactor

Unwatch 25 | Unstar 11 | Fork 20

Code | Issues 29 | Pull requests 0 | Projects 0 | Wiki | Insights

Home

David Rafferty edited this page on 13 Apr · 29 revisions

Edit | New Page

Some documentation on the wiki...

Welcome to the prefactor wiki!

prefactor consists of parsets for the genericpipeline that do the first calibration of LOFAR data. Originally in order to prepare said data for the Factor facet calibration, but also useful if you don't plan to run Factor.

It includes:

- applying Ionospheric RM corrections with RMextract
- clock-TEC separation with transfer of clock from the calibrator to the target
- some flagging and averaging of amplitude solutions
- grouping of subbands by actual frequency
- speed and disk usage improvements by optimized usage of NDPPP
- (optional) wide-band cleaning in Initial-Subtract
- diagnostic plots
- documentation in the cookbook and on these wiki pages

The documentation below is work in progress! It focuses on version 2 of prefactor, which is the latest

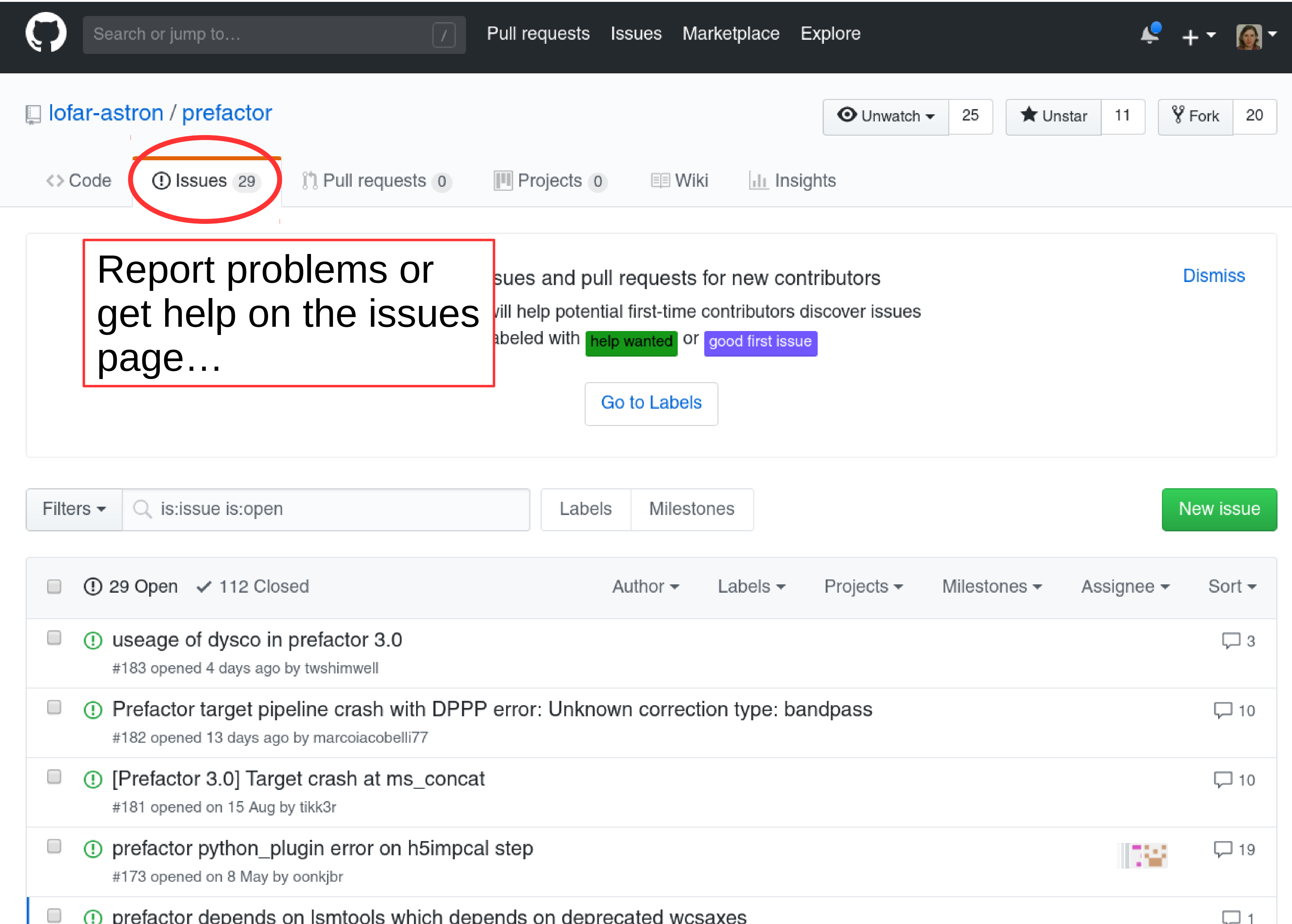
Pages 12

Wiki Home

Documentation:

1. Installation & Setup
2. Running a prefactor Pipeline
3. The prefactor Pipelines
   i. Checking the Diagnostic Plots of the Calibrator Pipeline
   ii. Flagging Bad Solutions
   iii. Processing the Target Data
4. Helper Scripts
5. Tutorial: running prefactor
6. Usage Notes
7. FAQ

Development:



Search or jump to…
Pull requests
Issues
Marketplace
Explore
lofar-astron / prefactor
Unwatch 25
Unstar 11
Fork 20
Code
Issues 29
Pull requests 0
Projects 0
Wiki
Insights
Report problems or get help on the issues page…
sues and pull requests for new contributors
vill help potential first-time contributors discover issues
abeled with help wanted or good first issue
Dismiss
Go to Labels
Filters
is:issue is:open
Labels
Milestones
New issue
29 Open
112 Closed
Author
Labels
Projects
Milestones
Assignee
Sort
useage of dysco in prefactor 3.0
#183 opened 4 days ago by twshimwell
3
Prefactor target pipeline crash with DPPP error: Unknown correction type: bandpass
#182 opened 13 days ago by marcoiacobelli77
10
[Prefactor 3.0] Target crash at ms_concat
#181 opened on 15 Aug by tikk3r
10
prefactor python_plugin error on h5impcal step
#173 opened on 8 May by oonkjbr
19
prefactor depends on lsmtools which depends on deprecated wcsaxes
1

Prefactor and the pipeline.cfg

- Add the prefactor recipes directory to the generic pipeline pipeline.cfg file in the line
  - recipe_directories = [%(pythonpath)s/lofarpipe/recipes,/home/williams/lds18/git/prefactor-master/prefactor]
- this allows the additional pipeline steps defined in the prefactor plugins directory to be used.

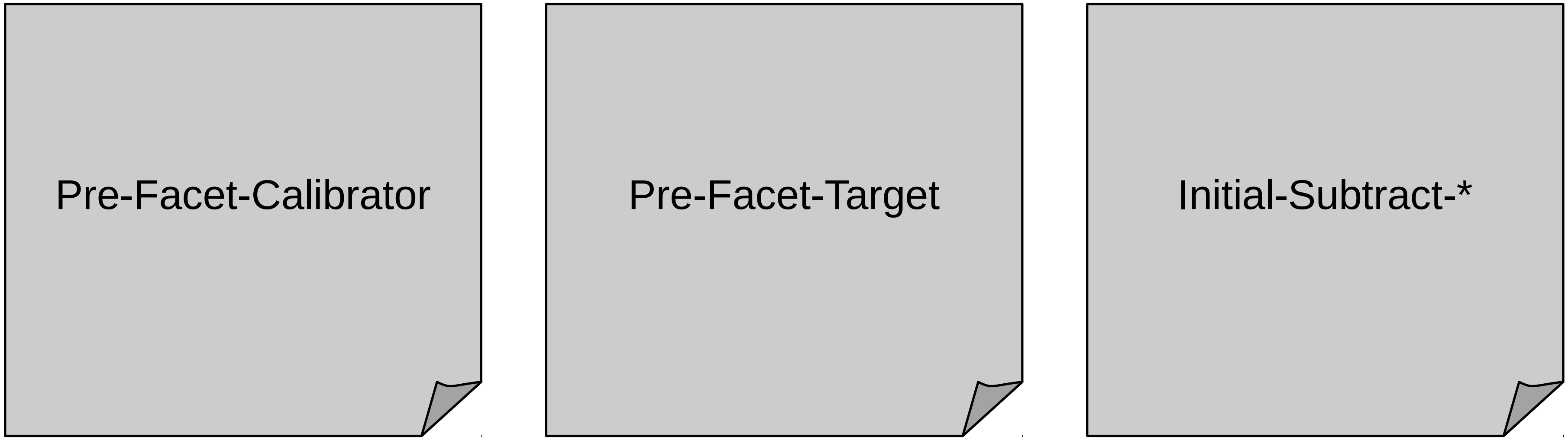
dependencies

- make sure you load the required packages
  - > module load lsmtool # the lofar solution tool
  - > module load lofar
- prefactor, losoto and rmextract are available on CEP3 but are frequently updated, so we will use a standard set of these packages which are located in
  - /home/williams/lds18/git/
- for prefactor we use a version sourced from the master branch in the git repo (with one or two minor changes). Instead of specifying these directly in the pipeline parsets, we will set some environment variables (Note no trailing slashes!!)
  - > export PREFACTOR_PATH=/home/williams/lds18/git/prefactor-master/prefactor
  - > export LOSOTO_PATH=/home/williams/lds18/git/losoto
  - > export PYTHONPATH=/home/williams/lds18/git/losoto/lib/python2.7/site-packages:${PYTHONPATH}
- rmextract needs to be added to your PYTHONPATH
  - > export PYTHONPATH=/home/williams/lds18/git/Rmextract/lib64/python2.7/site-packages:${PYTHONPATH}

Pipeline parset

- Variables
  - are defined in the parset with
    - ! Varname = value
  - And referred to as (Note the spaces!)
    - {{ varname }}
- Comments
  - Defined with # (inline or start of line)
- Steps
  - pipeline.steps = [comma separated list of steps]
  - And defined as
    - Stepname.control.<parameter> = value

parsets



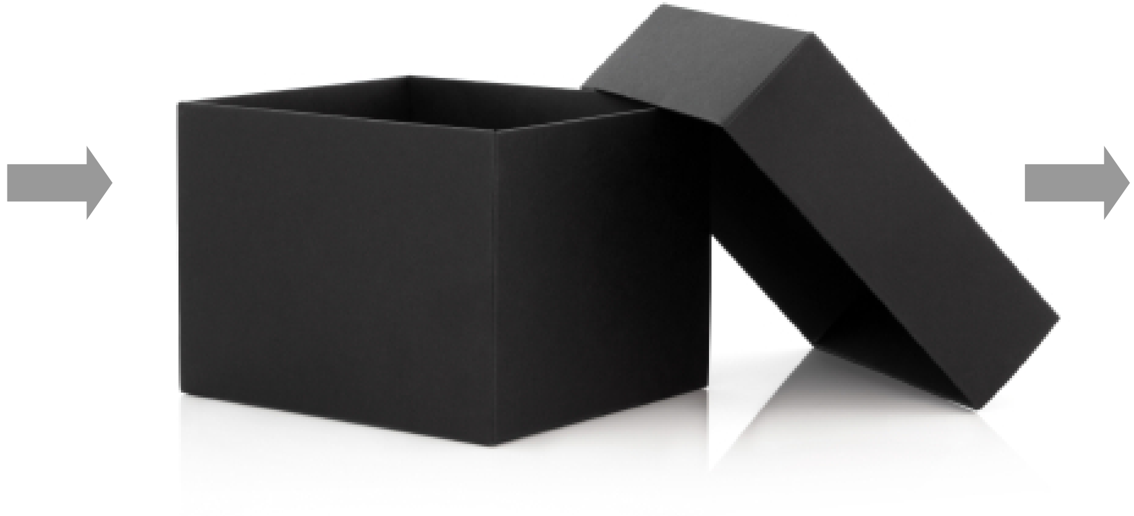
Start the calibrator pipeline

- Get a copy of the calibrator parset
  - > cp /home/williams/lds18/parsets/Pre-Facet-Calibrator-L232873.parset .
  - this is a slightly modified version of the one in the prefactor master branch
  - Note that there are many interesting advances in the version3 branch for the calibrator parset that are beyond the scope of this tutorial

- Then start the pipeline… and sit back have a coffee or write a paper
  - > genericpipeline.py Pre-Facet-Calibrator-L232873.parset -v -d -c pipeline.cfg
  - This should take about 30 min to run
  - realistically though, check that it starts off and doesn't crash and, for now, let's investigate what it is doing

Prefactor – what's inside the box?

Raw data



Calibrated data
(ready for FACTOR
or DDF)

Calibrator pipeline parset

- This parset is already set to point to the correct data and requires no further modification

```
## information about the calibrator data
! cal_input_path          =  /data010/scratch/wwilliams/test_tutorial/L232873
## specify the directory where your calibrator data are stored
! cal_input_pattern       =  L232873*.MS            ## regular expression
pattern of all your calibrator files
```

- And uses our pre-defined environment variables

```
## location of the software
! prefactor_directory      =  $PREFACTOR_PATH        ## path to your prefactor
copy
! losoto_directory         =  $LOSOTO_PATH           ## path to your local
LoSoTo installation
```

  - *Usually one has to change some parameters*

Calibrator pipeline steps

1. createmap_cal
2. ndppp_prep_cal
3. combine_data_cal_map
4. sky_cal
5. make_sourcedb
6. expand_sourcedb
7. calib_cal_parmmap
8. calib_cal
9. cal_apply
10. h5_imp_cal_map
11. h5imp_cal
12. prepare_losoto
13. process_losoto
14. mk_cal_values_dir
15. copy_cal_h5

Calibrator pipeline steps

1. createmap_cal
2. ndppp_prep_cal
3. combine_data_cal_map
4. sky_cal
5. make_sourcedb
6. expand_sourcedb
7. calib_cal_parmmap
8. calib_cal
9. cal_apply
10. h5_imp_cal_map
11. h5imp_cal
12. prepare_losoto
13. process_losoto
14. mk_cal_values_dir
15. copy_cal_h5

Prepare each calibrator SB
Run NDPPP to flag and average

Calibrator pipeline steps

1. createmap_cal
2. ndppp_prep_cal
3. combine_data_cal_map
4. sky_cal
5. make_sourcedb
6. expand_sourcedb
7. calib_cal_parmmap
8. calib_cal
9. cal_apply
10. h5_imp_cal_map
11. h5imp_cal
12. prepare_losoto
13. process_losoto
14. mk_cal_values_dir
15. copy_cal_h5

Prepare required skymodel and mapfiles for calibration

Calibrator pipeline steps

1. createmap_cal
2. ndppp_prep_cal
3. combine_data_cal_map
4. sky_cal
5. make_sourcedb
6. expand_sourcedb
7. calib_cal_parmmap
8. calib_cal
9. cal_apply
10. h5_imp_cal_map
11. h5imp_cal
12. prepare_losoto
13. process_losoto
14. mk_cal_values_dir
15. copy_cal_h5

CALIBRATE! And apply solutions…

Calibrator pipeline steps

1. createmap_cal
2. ndppp_prep_cal
3. combine_data_cal_map
4. sky_cal
5. make_sourcedb
6. expand_sourcedb
7. calib_cal_parmmap
8. calib_cal
9. cal_apply
10. h5_imp_cal_map
11. h5imp_cal
12. prepare_losoto
13. process_losoto
14. mk_cal_values_dir
15. copy_cal_h5

Use losoto to process solutions: smoothing, bandpass and clock/tec separation

Calibrator pipeline steps

1. createmap_cal
2. ndppp_prep_cal
3. combine_data_cal_map
4. sky_cal
5. make_sourcedb
6. expand_sourcedb
7. calib_cal_parmmap
8. calib_cal
9. cal_apply
10. h5_imp_cal_map
11. h5imp_cal
12. prepare_losoto
13. process_losoto
14. mk_cal_values_dir
15. copy_cal_h5

Save the final products

createmap_cal

```
# generate a mapfile of all the calibrator data
createmap_cal.control.kind                          =   plugin
createmap_cal.control.type                          =   createMapfile
createmap_cal.control.method                        =   mapfile_from_folder
createmap_cal.control.mapfile_dir                   =   input.output.mapfile_dir
createmap_cal.control.filename                      =   createmap_cal.mapfile
createmap_cal.control.folder                        =   {{ cal_input_path }}
createmap_cal.control.pattern                       =   {{ cal_input_pattern }}
```

Makes a ‘mapfile’ (text file) containing a list of the input calibrator Mss

- Looking in directory defined by the cal_input_path variable
    - /data010/scratch/wwilliams/lds18_pft/data/L232873
- All files matching the glob pattern
    - L232873*.MS
- The mapfile can be found in
    - /data/scratch/<username>/pf_tutorial/pipeline/Pre-Facet-Calibrator-L232873/mapfiles/createmap_cal.mapfile

ndppp_prep_cal

```
# run NDPPP on the calibrator data
ndppp_prep_cal.control.type                          =   dppp
ndppp_prep_cal.control.max_per_node                  =   {{ num_proc_per_node_limit }}
ndppp_prep_cal.control.error_tolerance               =   {{ error_tolerance }}
ndppp_prep_cal.argument.numthreads                   =   {{ max_dppp_threads }}
ndppp_prep_cal.argument.msin                         =   createmap_cal.output.mapfile    # The input
data.
ndppp_prep_cal.argument.msin.datacolumn              =   DATA
ndppp_prep_cal.argument.msin.baseline                =   CS*&; RS*&; CS*&RS*
ndppp_prep_cal.argument.msout.datacolumn             =   DATA
ndppp_prep_cal.argument.msout.writefullresflag       =   False
ndppp_prep_cal.argument.msout.overwrite              =   True
ndppp_prep_cal.argument.steps                        =   [flag,filter,avg,flagamp]
ndppp_prep_cal.argument.flag.type                    =   preflagger
ndppp_prep_cal.argument.flag.baseline                =   {{ flag_baselines }}
```

Run NDPPP on all calibrator SBs

- Steps are
  - Flag – flag any stations/baselines specified by flag_baseline variable
  - Filter – remove international stations
  - Avg – average in frequency and time to the specified avg_timeresolution and avg_freqresolution
  - Flagamp – clip small amplitudes
- Note the restriction on the number of simultaneously run processes
  - max_per_node is set to num_proc_per_node_limit used for disk intensive processes or multithreaded processes (max_dppp_threads)

combine_data_cal_map

```
# combine all entries into one mapfile (just for the find_skymode
combine_data_cal_map.control.kind                    =   plugin
combine_data_cal_map.control.type                    =   createMapfile
combine_data_cal_map.control.method                  =   mapfile_all_to_one
combine_data_cal_map.control.mapfile_dir             =   input.output.mapfile_dir
combine_data_cal_map.control.filename                =   combine_data_cal_map.mapfile
combine_data_cal_map.control.mapfile_in              =   createmap_cal.output.mapfile
```

Some tweaking of the data mapfiles

sky_cal

```
# find automatically the calibrator sky model
sky_cal.control.type                            =   pythonplugin
sky_cal.control.executable                      =   {{ scripts }}/find_skymodel_cal.py
sky_cal.control.error_tolerance                 =   {{ error_tolerance }}
sky_cal.argument.flags                          =   [combine_data_cal_map.output.mapfile]
sky_cal.argument.DirSkymodelCal                 =   {{ calibrator_path_skymodel }}
```

Uses the prefactor script find_skymodel_cal.py to lookup a skymodel to use for the calibrator

- No need to specify the name of the calibrator – the script will determine it
- The pipeline provides an easy way to include your own python scripts with the pythonplugin type

make_sourcedb

```
# make the sourcedb
make_sourcedb.control.kind                          =   recipe
make_sourcedb.control.type                          =   executable_args
make_sourcedb.control.executable                    =   {{ lofar_directory }}/bin/makesourcedb
make_sourcedb.control.error_tolerance               =   {{ error_tolerance }}
make_sourcedb.control.args_format                   =   lofar
make_sourcedb.control.outputkey                     =   out
make_sourcedb.control.mapfile_in                    =   sky_cal.output.SkymodelCal.mapfile
make_sourcedb.control.inputkey                      =   in
make_sourcedb.argument.format                       =   <
make_sourcedb.argument.outtype                      =   blob
```

Run makesourcedb on the skymodel to turn it into sourcedb format for DPPP calibration

- executable_args to run an executable and pass it arguments

expand_sourcedb

```
# expand the sourcedb mapfile so that there is one entry for every file, l
expand_sourcedb.control.kind                        =   plugin
expand_sourcedb.control.type                        =   expandMapfile
expand_sourcedb.control.mapfile_in                  =   make_sourcedb.output.mapfile
expand_sourcedb.control.mapfile_to_match            =   ndppp_prep_cal.output.mapfile
expand_sourcedb.control.mapfile_dir                 =   input.output.mapfile_dir
expand_sourcedb.control.filename                    =   expand_sourcedb.mapfile
```

Some mapfile magic to map one sourcedb to many calibrator SBs

calib_cal_parmap

```
# generate mapfile with the h5parm names to be used in the calib_cal steps
calib_cal_parmmap.control.kind                          =   plugin
calib_cal_parmmap.control.type                          =   createMapfile
calib_cal_parmmap.control.method                        =   add_suffix_to_file
calib_cal_parmmap.control.mapfile_in                    =   ndppp_prep_cal.output.mapfile
calib_cal_parmmap.control.add_suffix_to_file            =   /instrument.h5
calib_cal_parmmap.control.mapfile_dir                   =   input.output.mapfile_dir
calib_cal_parmmap.control.filename                      =   calib_cal_h5parms.mapfile
```

More mapfile magic to make a mapfile listing the name of the calibration table that will be produced for each SB

- The name will be the SB name with an additional suffix /instrument.h5

calib_cal

```
# now run NDPPP on the averaged calibrator data
calib_cal.control.type                          =   dppp
calib_cal.control.max_per_node                  =   {{ num_proc_per_node_limit }}
calib_cal.control.inplace                       =   True
calib_cal.control.error_tolerance               =   {{ error_tolerance }}
calib_cal.argument.numthreads                   =   {{ max_dppp_threads }}
calib_cal.argument.msin                         =   ndppp_prep_cal.output.mapfile    # The input
data.
calib_cal.argument.msin.datacolumn              =   DATA
calib_cal.argument.msin.baseline                =   CS*&; RS*&; CS*&RS*
calib_cal.argument.steps                        =   [solve]
calib_cal.argument.solve.type                   =   ddecal
calib_cal.argument.solve.mode                   =   rotation+diagonal
calib_cal.argument.solve.nchan                  =   1
calib_cal.argument.solve.solint                 =   1
```

Run DPPP to do the calibration for each SB

- Input sky catalogue (sourcedb)
  - Mapfile from expand_sourcedb
- Output solutions (h5parm)
  - Mapfile from calib_cal_parmmap

cal_apply

```
# apply the calibration solutions
cal_apply.control.type                              =   dppp
cal_apply.control.error_tolerance                   =   {{ error_tolerance }}
cal_apply.control.inplace                           =   True
cal_apply.control.max_per_node                      =   {{ num_proc_per_node_limit }}
cal_apply.argument.numthreads                       =   {{ max_dppp_threads }}
cal_apply.argument.msin                             =   ndppp_prep_cal.output.mapfile
cal_apply.argument.msin.datacolumn                  =   DATA
cal_apply.argument.msout.datacolumn                 =   CORRECTED_DATA
cal_apply.argument.msout                            =   .
cal_apply.argument.steps                            =   [applycal1,applycal2,applycal3]
cal_apply.argument.applycal1.type                   =   applycal
cal_apply.argument.applycal1.correction             =   amplitude000
cal_apply.argument.applycal1.parmdb                 =   calib_cal_parmmap.output.mapfile
cal_apply.argument.applycal2.type                   =   applycal
```

Run NDPPP to do apply the calibration for each SB

- solutions (parmdb)
  - Mapfile from calib_cal_parmmap
  - Solve for:
    - Amplitude
    - Phase
    - Rotation angle

h5_imp_cal_map

```
# generate a mapfile with all files in a single entry
h5_imp_cal_map.control.kind                          =   plugin
h5_imp_cal_map.control.type                          =   compressMapfile
h5_imp_cal_map.control.mapfile_in                    =   calib_cal_parmmap.output.mapfile
h5_imp_cal_map.control.mapfile_dir                   =   input.output.mapfile_dir
h5_imp_cal_map.control.filename                      =   h5_imp_cal_map.mapfile
```

- Mapfile magic to make a list of all the solution tables (parmdbs) produced by DPPP

h5_imp_cal

```
# collect all instrument tables into one h5parm
h5imp_cal.control.kind                         =   recipe
h5imp_cal.control.type                         =   executable_args
h5imp_cal.control.executable                   =   {{ losoto_directory }}/bin/H5parm_collector.py
h5imp_cal.control.error_tolerance              =   {{ error_tolerance }}
h5imp_cal.control.outputkey                    =   outh5parm
h5imp_cal.argument.flags                       =   [-c,h5_imp_cal_map.output.mapfile]
h5imp_cal.argument.outh5parm                   =   outh5parm
```

- Use LoSoTo utility H5parm_collector.py to gather all the solution tables into a single h5parm file

prepare_losoto

```
# create losoto v2 parset file
prepare_losoto.control.kind                          =   plugin
prepare_losoto.control.type                          =   makeLosotoParset
prepare_losoto.control.steps                         =   [clocktec, clock_smooth, bandpass, xyoffset,
plot_phases, plot_amps, plot_clock, plot_TEC, plot_bandpass, plot_xyoffset]
prepare_losoto.control.filename                      =   input.output.job_directory/losoto.parset
prepare_losoto.control.global.ncpu                   =   0
prepare_losoto.control.clocktec.soltab               =   [sol000/phase000]
prepare_losoto.control.clocktec.operation            =   CLOCKTEC
prepare_losoto.control.clocktec.combinePol           =   True
prepare_losoto.control.clocktec.flagBadChannels      =   False
prepare_losoto.control.clocktec.fit3rdOrder          =   False
prepare_losoto.control.clock_smooth.soltab           =   [sol000/clock000]
prepare_losoto.control.clock_smooth.operation        =   SMOOTH
prepare_losoto.control.clock_smooth.axestosmooth     =   [time]
```

- Write a parsset file for LOSOTO
- Losoto will be used to
  - Separate clock and TEC
  - Smooth the clock solutions
  - Solve for the bandpass
  - Calculate the phase offset between XX and YY polarisations
  - Make some inspection plots of the solutions
    - In the {{ inspection_directory }}
      - {{ results_directory }}/inspection

process_losoto

```
# do the processing on the LoSoTo file
process_losoto.control.kind                 =   recipe
process_losoto.control.type                 =   executable_args
process_losoto.control.executable           =   {{ losoto_directory }}/bin/losoto
process_losoto.control.max_per_node         =   {{ num_proc_per_node }}
process_losoto.argument.flags               =   [h5imp_cal.output.mapfile,
input.output.job_directory/losoto.parset]
```

- Run LoSoTo
  - Calls the executable in the {{ losoto_directory}}/bin directory
  - Arguments are the h5 parm file and the LoSoTo parset just created

mk_cal_values_dir

```
# create the cal_values_directory if needed
mk_cal_values_dir.control.kind                          =   plugin
mk_cal_values_dir.control.type                          =   makeDirectory
mk_cal_values_dir.control.directory                     =   {{ cal_values_directory }}
```

- Plugin to create a directory for the final solutions

copy_cal_h5

```
# copy the cal h5parm to the cal-values directory
copy_cal_h5.control.kind                                    =   recipe
copy_cal_h5.control.type                                    =   executable_args
copy_cal_h5.control.executable                              =   /bin/cp
copy_cal_h5.control.max_per_node                            =   {{ num_proc_per_node_limit }}
copy_cal_h5.control.mapfile_in                              =   h5imp_cal.output.mapfile
copy_cal_h5.control.inputkey                                =   source
copy_cal_h5.control.arguments                               =   [source,{{ cal_values_directory }}]
```

- Uses executable_args type to call OS command cp to copy the final solutions to the final solutions directory

Monitoring the pipeline

- In another terminal on the node you can run OS commands to see what processes are running, how much memory and cpus they are using
  - top
  - htop
  - Ps - ef | grep <username>
- You can inspect the data products being produced in the runtime_directory
- Browse the pipeline log file
  - > less pipeline/Prefacet-Calibrator-L /logs/<timestamp>/pipeline.log
  - Familiarity with the log structure will help you navigate it better
    - e.g. ;grep Beginning <logfile>’ will give you the times each step started
    - You can also find the outputs from each step (useful when things go wrong)

Success!

- The pipeline run will end with

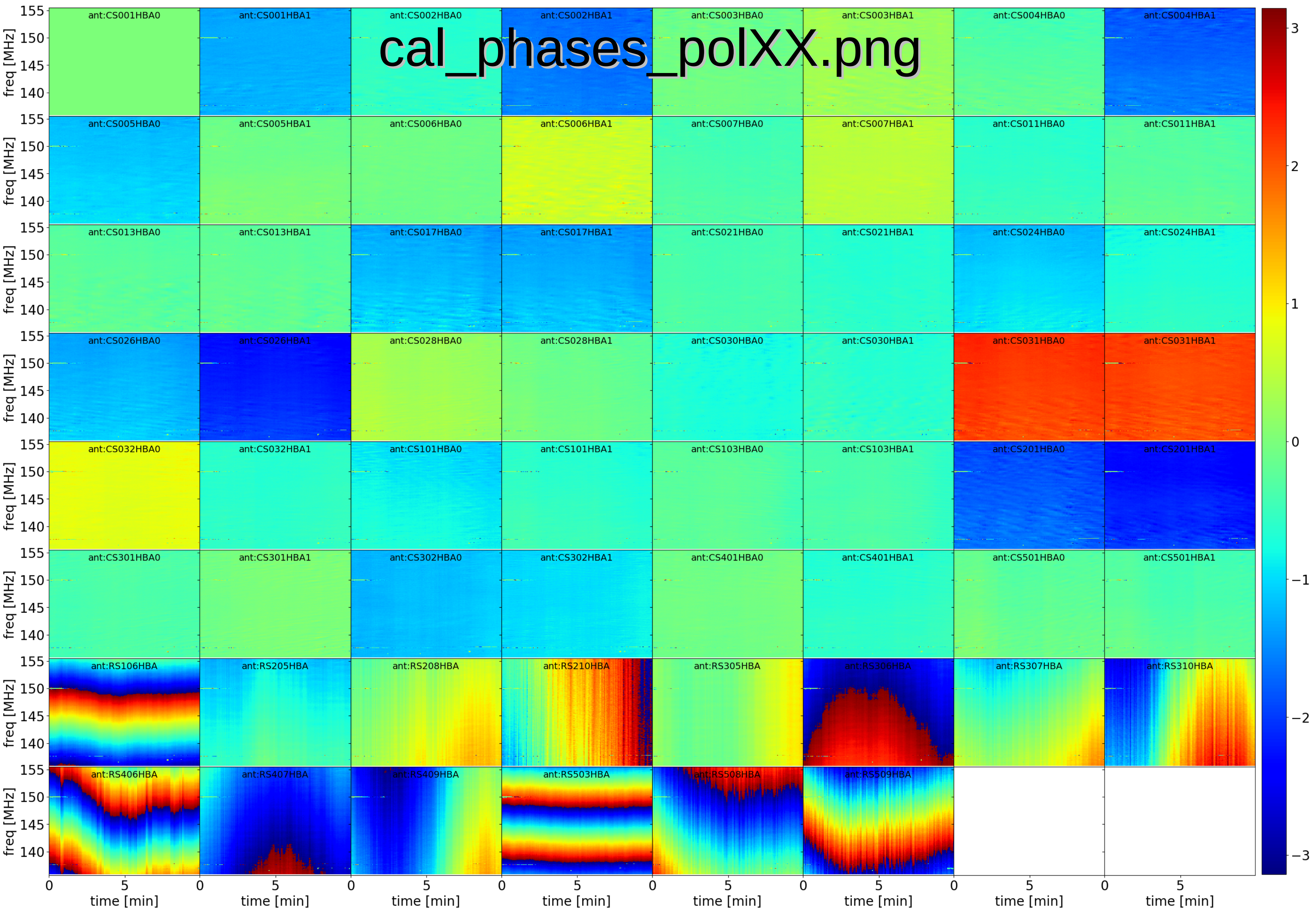
2018-09-17 15:47:12 INFO   genericpipeline: LOFAR Pipeline finished succesfully.

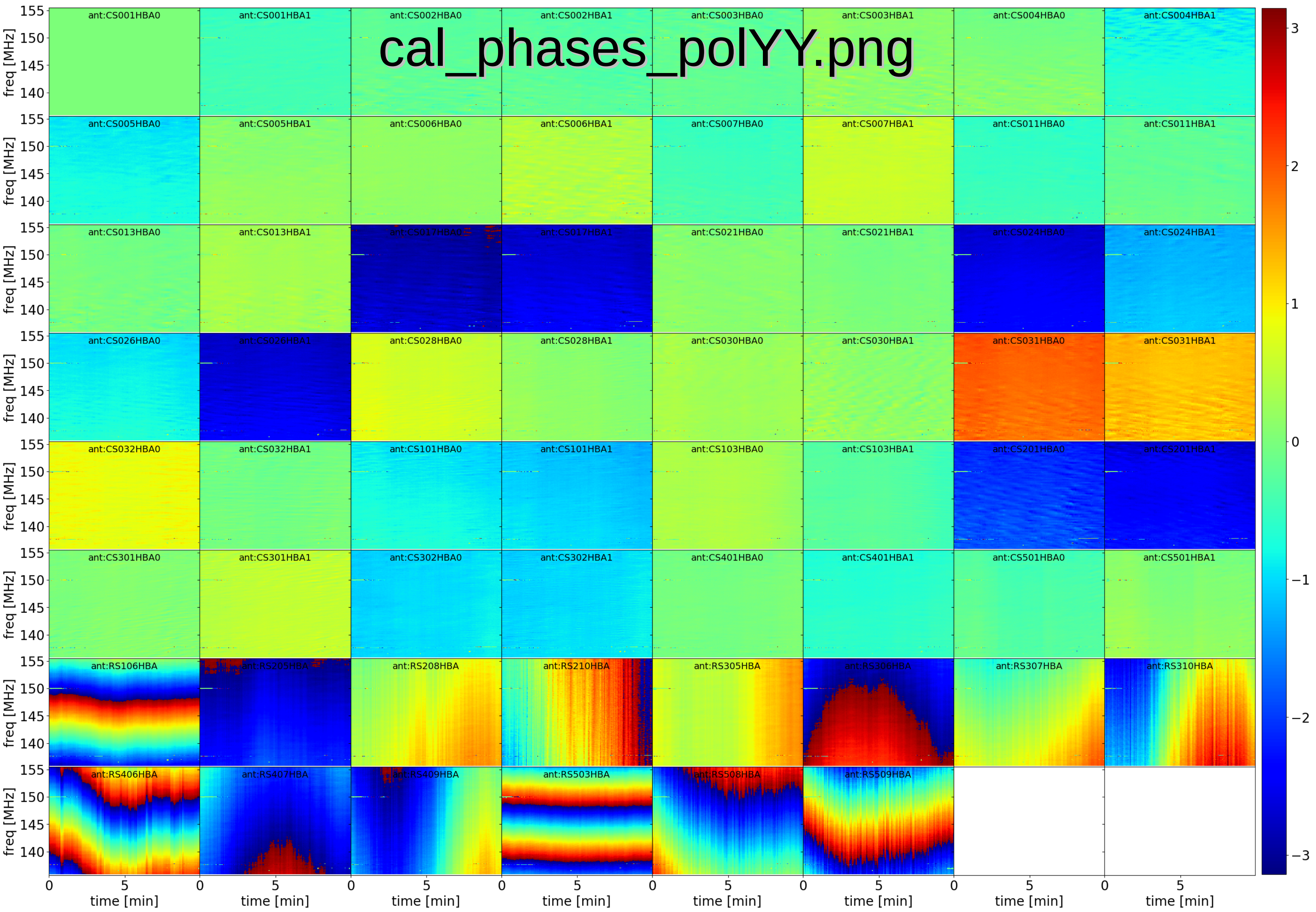
2018-09-17 15:47:12 INFO   genericpipeline: recipe genericpipeline completed

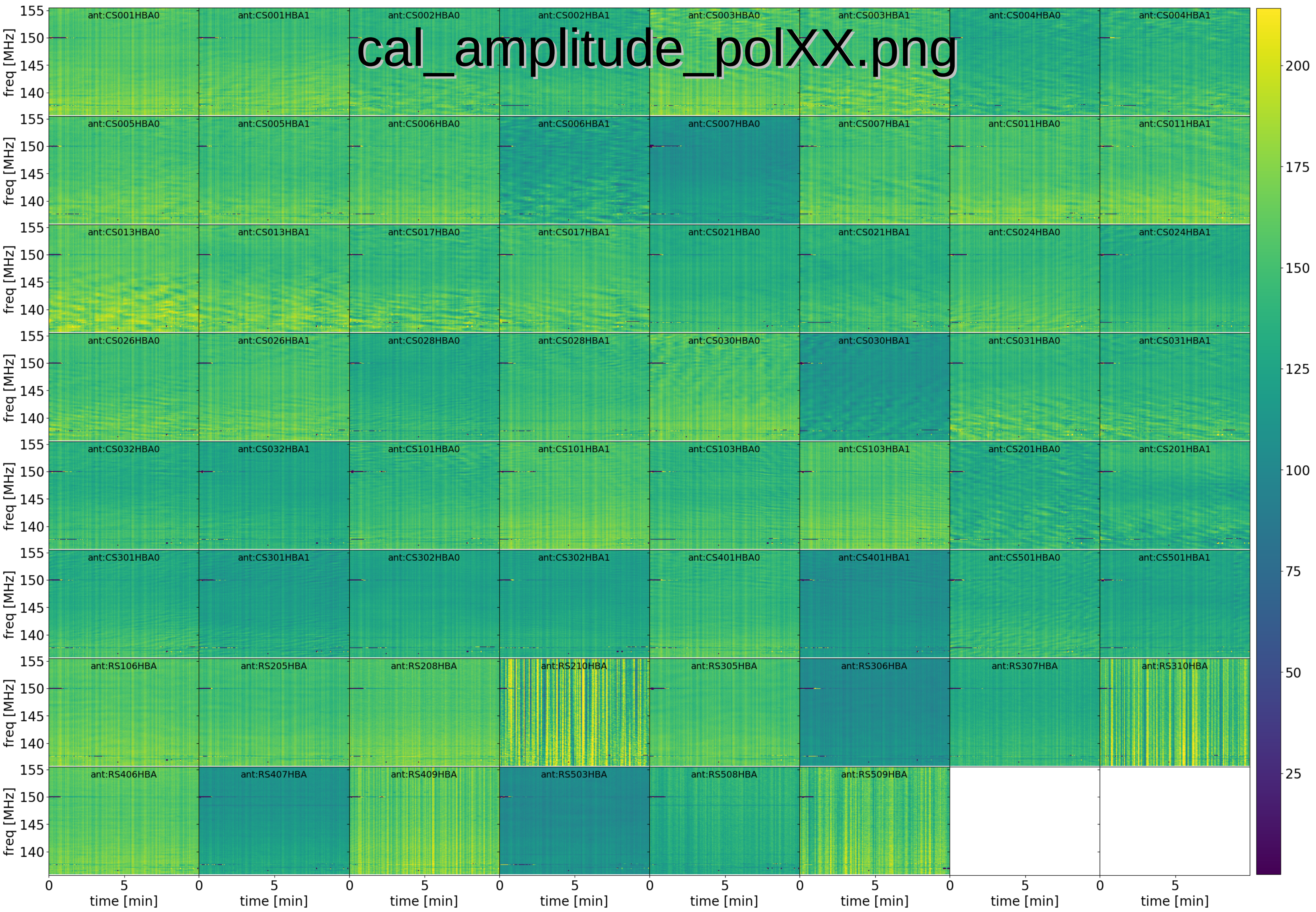
- … after about 30 min

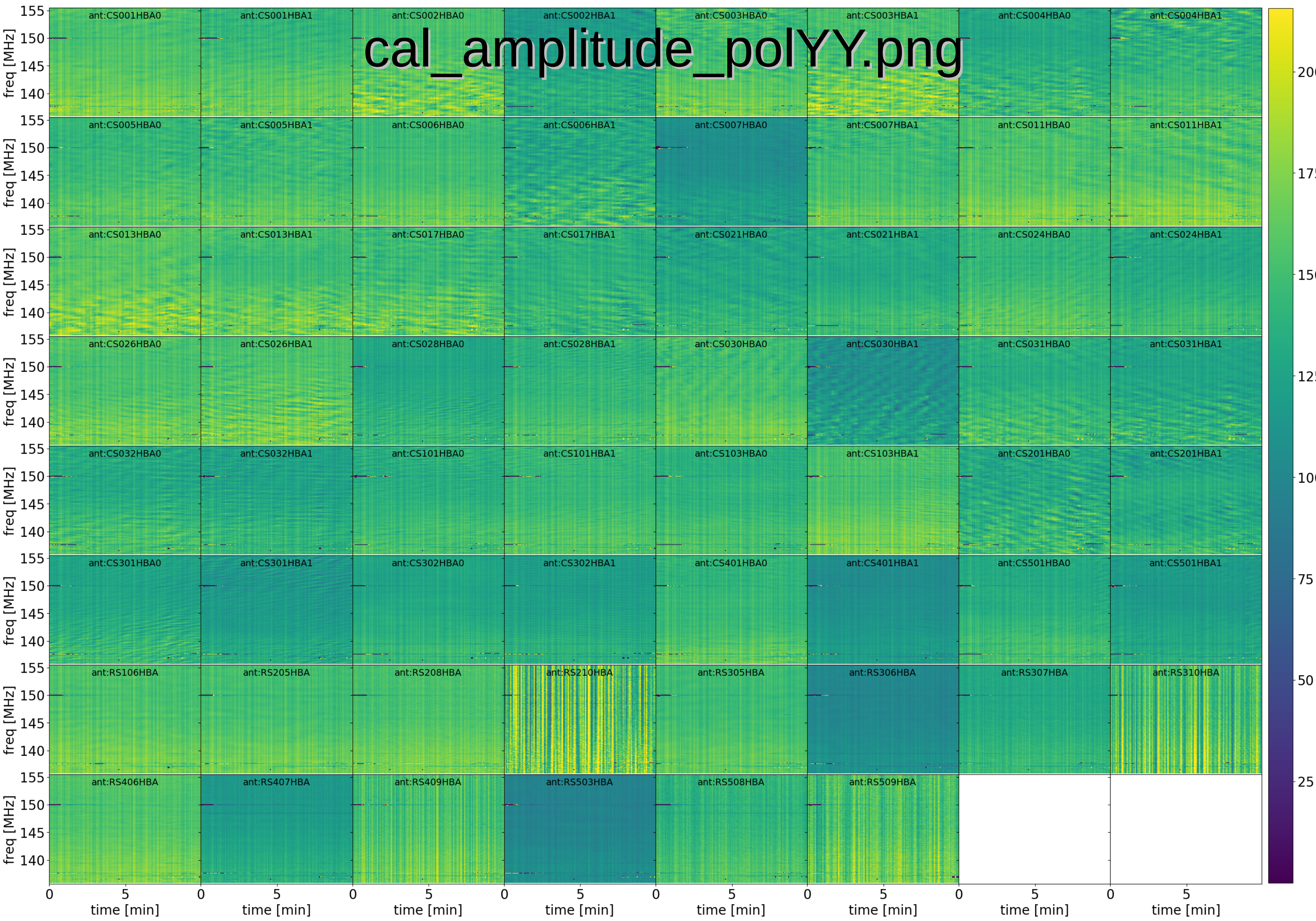
Inspect the results

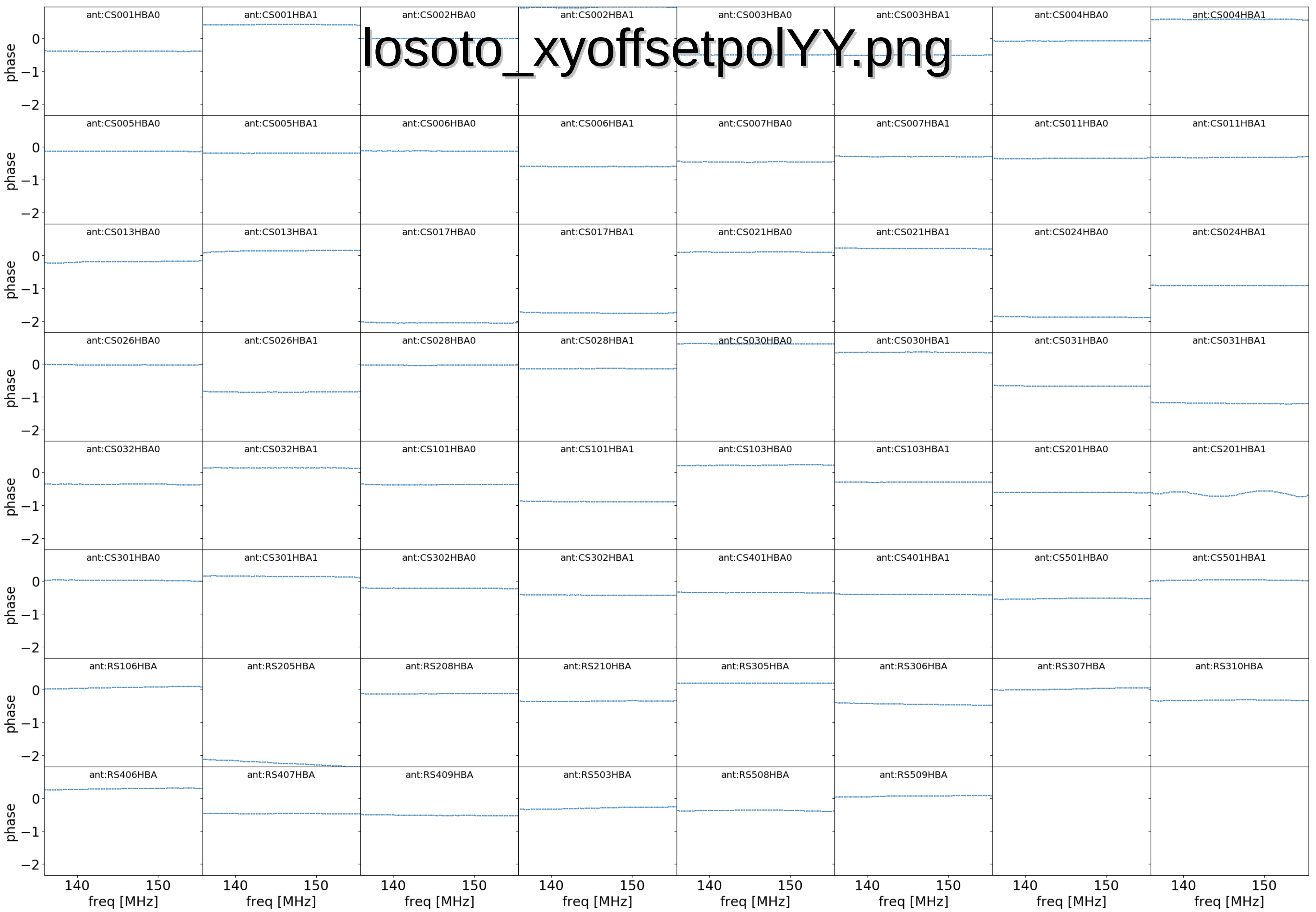
- Calibration plots are in
  - /data/scratch/<username>/pf_tutorial/pipeline/Pre-Facet-Calibrator-L232873/results/inspection/
    - cal_phases_polXX.png
    - cal_phases_polYY.png
    - cal_amplitude_polXX.png
    - cal_amplitude_polYY.png
    - losoto_clock.png
    - losoto_tec.png
    - losoto_bandpasspolXX.png
    - losoto_bandpasspolYY.png
    - losoto_xyoffsetpolYY.png


cal_phases_polXX.png
ant:CS001HBA0
ant:CS001HBA1
ant:CS002HBA0
ant:CS002HBA1
ant:CS003HBA0
ant:CS003HBA1
ant:CS004HBA0
ant:CS004HBA1
ant:CS005HBA0
ant:CS005HBA1
ant:CS006HBA0
ant:CS006HBA1
ant:CS007HBA0
ant:CS007HBA1
ant:CS011HBA0
ant:CS011HBA1
ant:CS013HBA0
ant:CS013HBA1
ant:CS017HBA0
ant:CS017HBA1
ant:CS021HBA0
ant:CS021HBA1
ant:CS024HBA0
ant:CS024HBA1
ant:CS026HBA0
ant:CS026HBA1
ant:CS028HBA0
ant:CS028HBA1
ant:CS030HBA0
ant:CS030HBA1
ant:CS031HBA0
ant:CS031HBA1
ant:CS032HBA0
ant:CS032HBA1
ant:CS101HBA0
ant:CS101HBA1
ant:CS103HBA0
ant:CS103HBA1
ant:CS201HBA0
ant:CS201HBA1
ant:CS301HBA0
ant:CS301HBA1
ant:CS302HBA0
ant:CS302HBA1
ant:CS401HBA0
ant:CS401HBA1
ant:CS501HBA0
ant:CS501HBA1
ant:RS106HBA
ant:RS205HBA
ant:RS208HBA
ant:RS210HBA
ant:RS305HBA
ant:RS306HBA
ant:RS307HBA
ant:RS310HBA
ant:RS406HBA
ant:RS407HBA
ant:RS409HBA
ant:RS503HBA
ant:RS508HBA
ant:RS509HBA
freq [MHz]
155
150
145
140
time [min]
0
5
3
2
1
0
−1
−2
−3


cal_phases_polYY.png
ant:CS001HBA0
ant:CS001HBA1
ant:CS002HBA0
ant:CS002HBA1
ant:CS003HBA0
ant:CS003HBA1
ant:CS004HBA0
ant:CS004HBA1
ant:CS005HBA0
ant:CS005HBA1
ant:CS006HBA0
ant:CS006HBA1
ant:CS007HBA0
ant:CS007HBA1
ant:CS011HBA0
ant:CS011HBA1
ant:CS013HBA0
ant:CS013HBA1
ant:CS017HBA0
ant:CS017HBA1
ant:CS021HBA0
ant:CS021HBA1
ant:CS024HBA0
ant:CS024HBA1
ant:CS026HBA0
ant:CS026HBA1
ant:CS028HBA0
ant:CS028HBA1
ant:CS030HBA0
ant:CS030HBA1
ant:CS031HBA0
ant:CS031HBA1
ant:CS032HBA0
ant:CS032HBA1
ant:CS101HBA0
ant:CS101HBA1
ant:CS103HBA0
ant:CS103HBA1
ant:CS201HBA0
ant:CS201HBA1
ant:CS301HBA0
ant:CS301HBA1
ant:CS302HBA0
ant:CS302HBA1
ant:CS401HBA0
ant:CS401HBA1
ant:CS501HBA0
ant:CS501HBA1
ant:RS106HBA
ant:RS205HBA
ant:RS208HBA
ant:RS210HBA
ant:RS305HBA
ant:RS306HBA
ant:RS307HBA
ant:RS310HBA
ant:RS406HBA
ant:RS407HBA
ant:RS409HBA
ant:RS503HBA
ant:RS508HBA
ant:RS509HBA
freq [MHz]
155
150
145
140
time [min]
0
5
3
2
1
0
−1
−2
−3

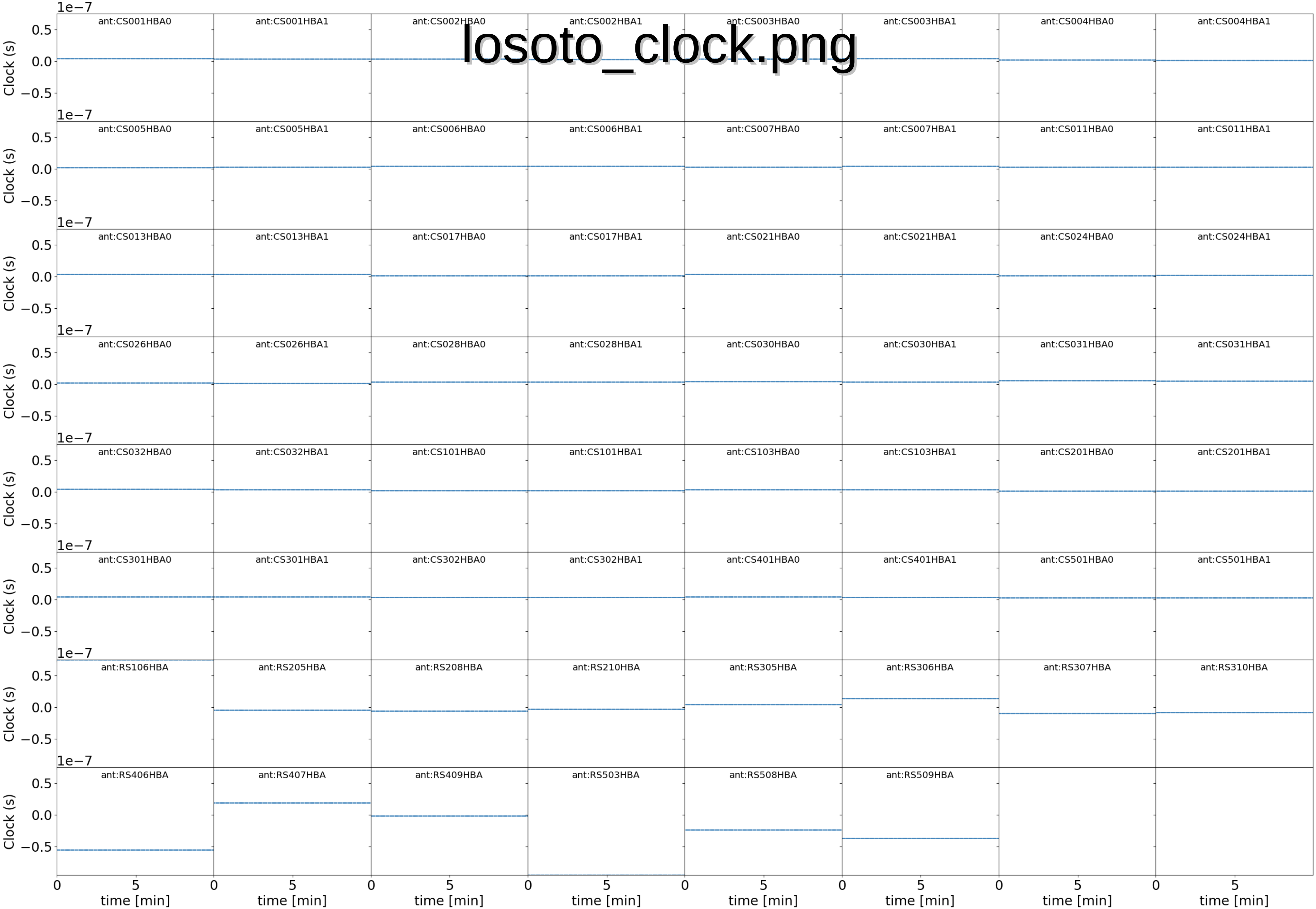

cal_amplitude_polXX.png
ant:CS001HBA0
ant:CS001HBA1
ant:CS002HBA0
ant:CS002HBA1
ant:CS003HBA0
ant:CS003HBA1
ant:CS004HBA0
ant:CS004HBA1
ant:CS005HBA0
ant:CS005HBA1
ant:CS006HBA0
ant:CS006HBA1
ant:CS007HBA0
ant:CS007HBA1
ant:CS011HBA0
ant:CS011HBA1
ant:CS013HBA0
ant:CS013HBA1
ant:CS017HBA0
ant:CS017HBA1
ant:CS021HBA0
ant:CS021HBA1
ant:CS024HBA0
ant:CS024HBA1
ant:CS026HBA0
ant:CS026HBA1
ant:CS028HBA0
ant:CS028HBA1
ant:CS030HBA0
ant:CS030HBA1
ant:CS031HBA0
ant:CS031HBA1
ant:CS032HBA0
ant:CS032HBA1
ant:CS101HBA0
ant:CS101HBA1
ant:CS103HBA0
ant:CS103HBA1
ant:CS201HBA0
ant:CS201HBA1
ant:CS301HBA0
ant:CS301HBA1
ant:CS302HBA0
ant:CS302HBA1
ant:CS401HBA0
ant:CS401HBA1
ant:CS501HBA0
ant:CS501HBA1
ant:RS106HBA
ant:RS205HBA
ant:RS208HBA
ant:RS210HBA
ant:RS305HBA
ant:RS306HBA
ant:RS307HBA
ant:RS310HBA
ant:RS406HBA
ant:RS407HBA
ant:RS409HBA
ant:RS503HBA
ant:RS508HBA
ant:RS509HBA
freq [MHz]
155
150
145
140
time [min]
0
5
200
175
150
125
100
75
50
25


cal_amplitude_polYY.png
ant:CS001HBA0
ant:CS001HBA1
ant:CS002HBA0
ant:CS002HBA1
ant:CS003HBA0
ant:CS003HBA1
ant:CS004HBA0
ant:CS004HBA1
ant:CS005HBA0
ant:CS005HBA1
ant:CS006HBA0
ant:CS006HBA1
ant:CS007HBA0
ant:CS007HBA1
ant:CS011HBA0
ant:CS011HBA1
ant:CS013HBA0
ant:CS013HBA1
ant:CS017HBA0
ant:CS017HBA1
ant:CS021HBA0
ant:CS021HBA1
ant:CS024HBA0
ant:CS024HBA1
ant:CS026HBA0
ant:CS026HBA1
ant:CS028HBA0
ant:CS028HBA1
ant:CS030HBA0
ant:CS030HBA1
ant:CS031HBA0
ant:CS031HBA1
ant:CS032HBA0
ant:CS032HBA1
ant:CS101HBA0
ant:CS101HBA1
ant:CS103HBA0
ant:CS103HBA1
ant:CS201HBA0
ant:CS201HBA1
ant:CS301HBA0
ant:CS301HBA1
ant:CS302HBA0
ant:CS302HBA1
ant:CS401HBA0
ant:CS401HBA1
ant:CS501HBA0
ant:CS501HBA1
ant:RS106HBA
ant:RS205HBA
ant:RS208HBA
ant:RS210HBA
ant:RS305HBA
ant:RS306HBA
ant:RS307HBA
ant:RS310HBA
ant:RS406HBA
ant:RS407HBA
ant:RS409HBA
ant:RS503HBA
ant:RS508HBA
ant:RS509HBA
freq [MHz]
155
150
145
140
time [min]
0
5
200
175
150
125
100
75
50
25

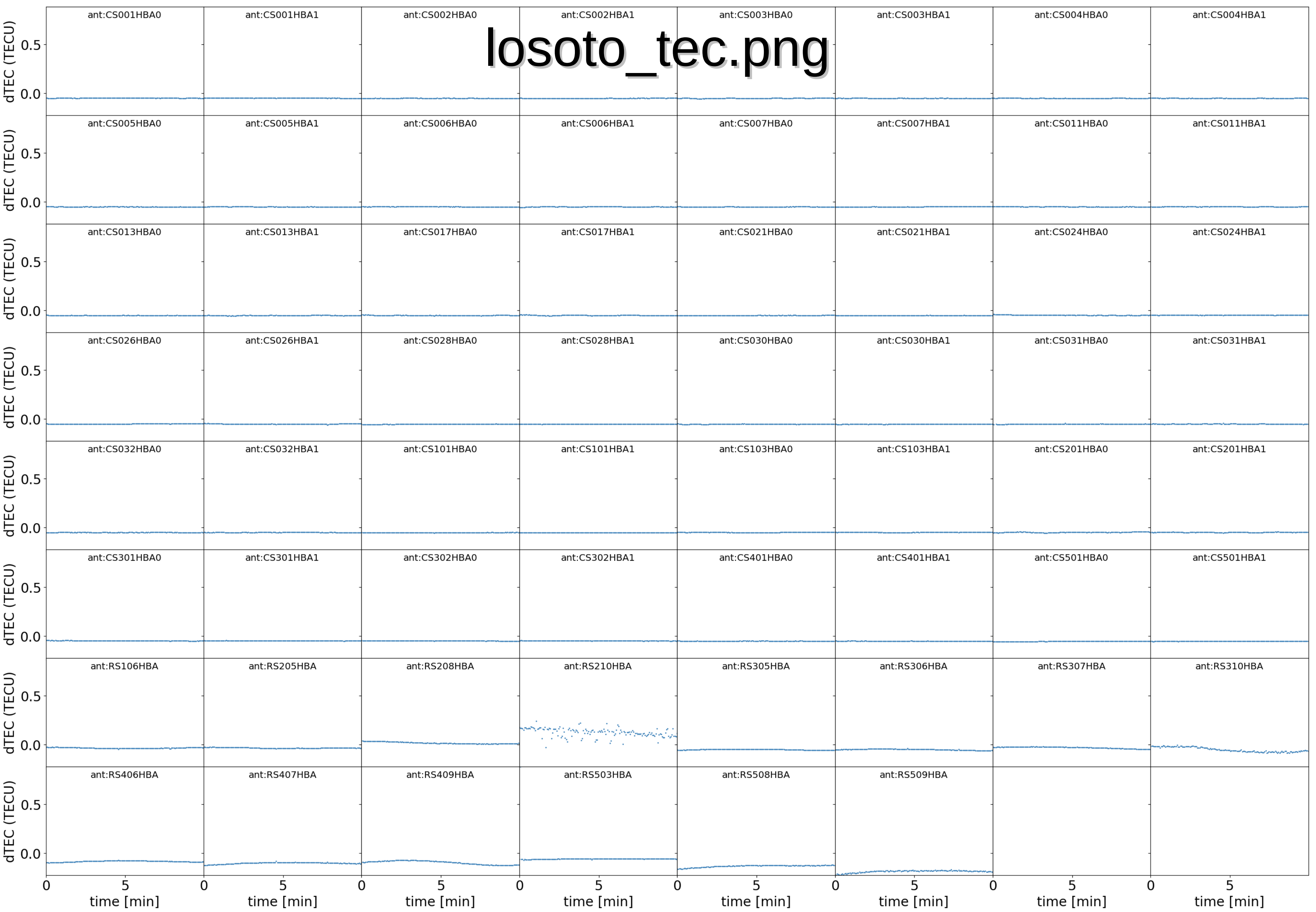


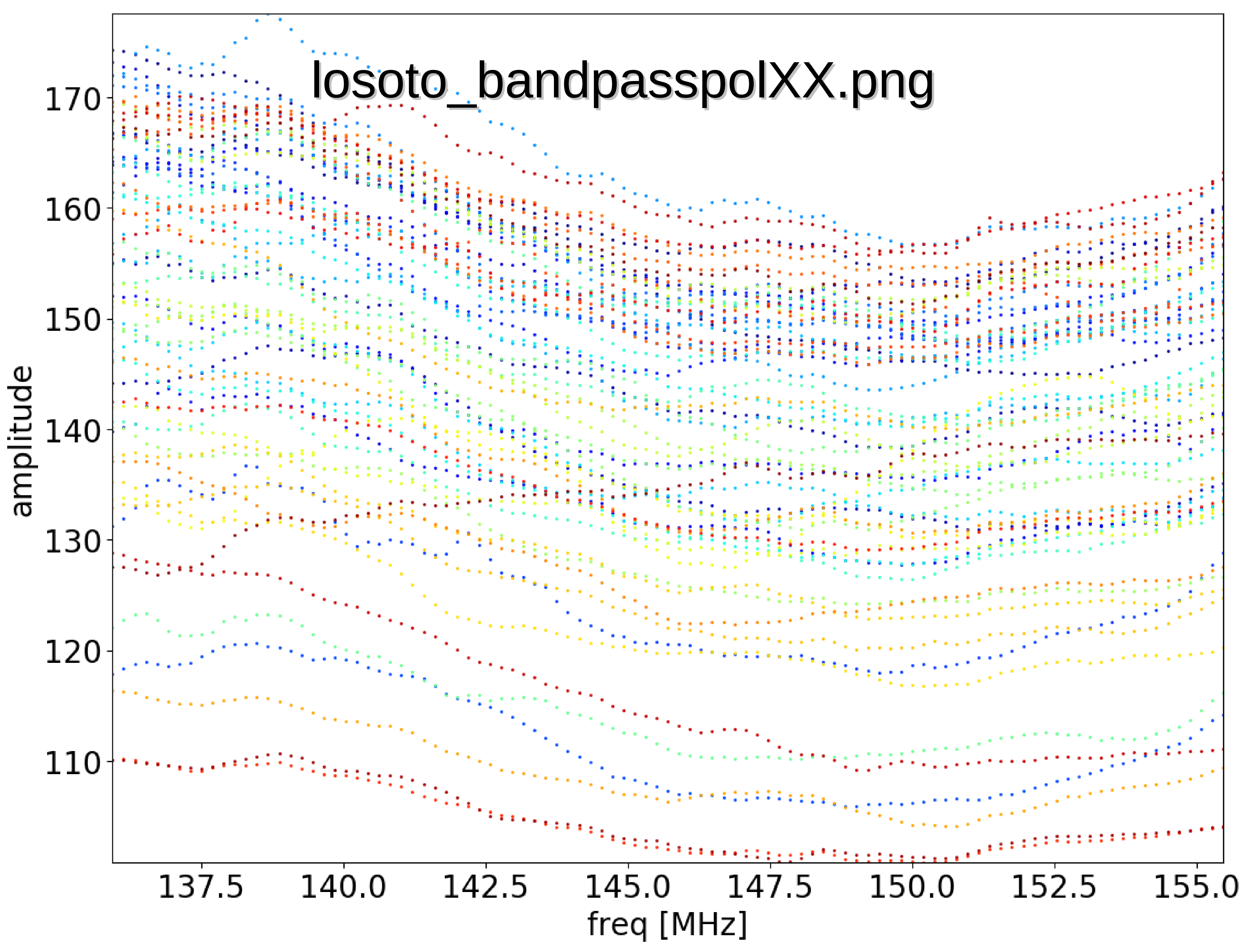
losoto_xyoffsetpolYY.png
ant:CS001HBA0
ant:CS001HBA1
ant:CS002HBA0
ant:CS002HBA1
ant:CS003HBA0
ant:CS003HBA1
ant:CS004HBA0
ant:CS004HBA1
ant:CS005HBA0
ant:CS005HBA1
ant:CS006HBA0
ant:CS006HBA1
ant:CS007HBA0
ant:CS007HBA1
ant:CS011HBA0
ant:CS011HBA1
ant:CS013HBA0
ant:CS013HBA1
ant:CS017HBA0
ant:CS017HBA1
ant:CS021HBA0
ant:CS021HBA1
ant:CS024HBA0
ant:CS024HBA1
ant:CS026HBA0
ant:CS026HBA1
ant:CS028HBA0
ant:CS028HBA1
ant:CS030HBA0
ant:CS030HBA1
ant:CS031HBA0
ant:CS031HBA1
ant:CS032HBA0
ant:CS032HBA1
ant:CS101HBA0
ant:CS101HBA1
ant:CS103HBA0
ant:CS103HBA1
ant:CS201HBA0
ant:CS201HBA1
ant:CS301HBA0
ant:CS301HBA1
ant:CS302HBA0
ant:CS302HBA1
ant:CS401HBA0
ant:CS401HBA1
ant:CS501HBA0
ant:CS501HBA1
ant:RS106HBA
ant:RS205HBA
ant:RS208HBA
ant:RS210HBA
ant:RS305HBA
ant:RS306HBA
ant:RS307HBA
ant:RS310HBA
ant:RS406HBA
ant:RS407HBA
ant:RS409HBA
ant:RS503HBA
ant:RS508HBA
ant:RS509HBA
phase
0
−1
−2
140
150
freq [MHz]

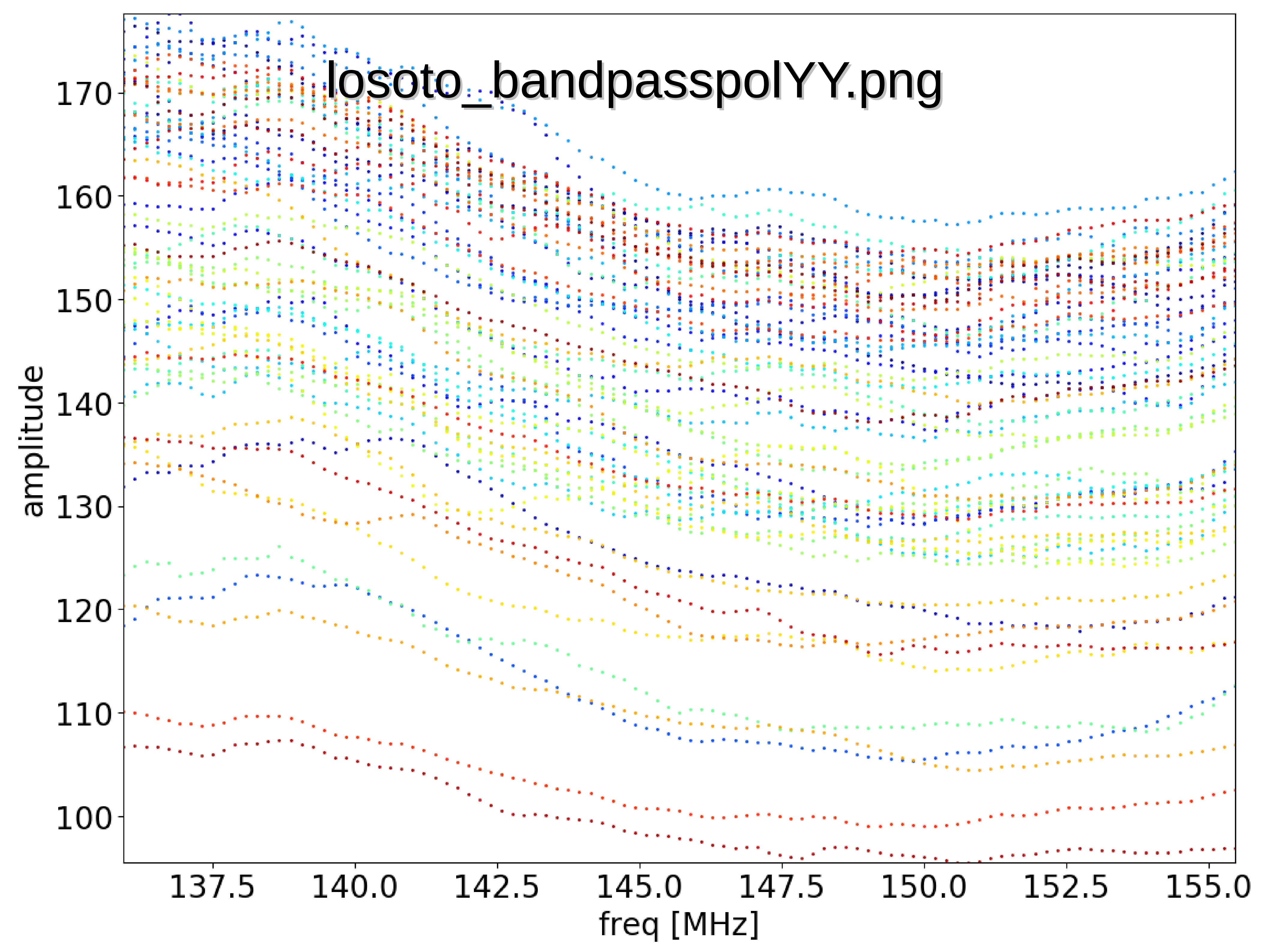
losoto_clock.png



losoto_tec.png




losoto_bandpasspolXX.png
amplitude
freq [MHz]
170
160
150
140
130
120
110
137.5
140.0
142.5
145.0
147.5
150.0
152.5
155.0



losoto_bandpasspolYY.png
amplitude
100
110
120
130
140
150
160
170
137.5
140.0
142.5
145.0
147.5
150.0
152.5
155.0
freq [MHz]

Start the target pipeline

- Get a copy of the targer parset
  - > cp /home/williams/lds18/parsets/Pre-Facet-Target-L232875.parset .
  - this is a slightly modified version of the one in the prefactor master branch
- Edit the cal_values_directory path to point to your results from the calibrator pipeline
  - ! cal_values_directory = /data/scratch/<username>/pf_tutorial/pipeline/Pre-Facet-Calibrator-L232873/results/cal_values

Start the target pipeline

- Then start the pipeline…
  - > genericpipeline.py Pre-Facet-Target-L232875.parset -v -d -c pipeline.cfg
  - realistically though, check that it starts off and doesn't crash and, for now, let's investigate what it is doing

Target pipeline steps

1. mk_inspect_dir
2. createmap_target
3. combine_data_target_map
4. get_ion_files
5. trans
6. ndppp_prep_target
7. create_ateam_model_map
8. make_sourcedb_ateam
9. expand_sourcedb_ateam
10. predict_ateam
11. ateamcliptar
12. combine_target_map
13. sortmap_target
14. do_sortmap_maps
15. dpppconcat
16. check_unflagged
17. check_unflagged_map
18. sky_tar
19. create_target_model_map
20. make_sourcedb_target
21. expand_sourcedb_target
22. gsmcal_parmmap
23. gsmcal_solve
24. gsmcal_apply
25. h5_imp_gsmsol_map
26. h5imp_gsmsol
27. plot_gsm_phases
28. gsmcal_antmap
29. make_structurefunction
30. old_plot_gsmphases
31. createmap_plots
32. copy_plots
33. mk_results_dir
34. make_results_mapfile
35. move_results

Target pipeline steps

1. mk_inspect_dir
2. createmap_target
3. combine_data_target_map
4. get_ion_files
5. trans
6. ndppp_prep_target
7. create_ateam_model_map
8. make_sourcedb_ateam
9. expand_sourcedb_ateam
10. predict_ateam
11. ateamcliptar
12. combine_target_map
13. sortmap_target
14. do_sortmap_maps
15. dpppconcat
16. check_unflagged
17. check_unflagged_map
18. sky_tar

Prepare mapfiles and directories

23. gsmcal_solve
24. gsmcal_apply
25. h5_imp_gsmsol_map
26. h5imp_gsmsol
27. plot_gsm_phases
28. gsmcal_antmap
29. make_structurefunction
30. old_plot_gsmphases
31. createmap_plots
32. copy_plots
33. mk_results_dir
34. make_results_mapfile
35. move_results

Target pipeline steps

1. mk_inspect_dir
2. createmap_target
3. combine_data_target_map
4. get_ion_files
5. trans
6. ndppp_prep_target
7. create_ateam_model_map
8. make_sourcedb_ateam
9. expand_sourcedb_ateam
10. predict_ateam
11. ateamcliptar
12. combine_target_map
13. sortmap_target
14. do_sortmap_maps
15. dpppconcat
16. check_unflagged
17. check_unflagged_map
18. sky_tar
19. create_target_model_map
20. make_sourcedb_target
21. expand_sourcedb_target

Get ionosphere information and add to calibration table

26. h5imp_gsmsol
27. plot_gsm_phases
28. gsmcal_antmap
29. make_structurefunction
30. old_plot_gsmphases
31. createmap_plots
32. copy_plots
33. mk_results_dir
34. make_results_mapfile
35. move_results

Target pipeline steps

1. mk_inspect_dir
2. createmap_target
3. combine_data_target_map
4. get_ion_files
5. trans
6. ndppp_prep_target
7. create_ateam_model_map
8. make_sourcedb_ateam
9. expand_sourcedb_ateam
10. predict_ateam
11. ateamcliptar
12. combine_target_map
13. sortmap_target
14. do_sortmap_maps
15. dpppconcat
16. check_unflagged
17. check_unflagged_map
18. sky_tar
19. create_target_model_map
20. make_sourcedb_target
21. expand_sourcedb_target
22. gsmcal_parmmap
23. gsmcal_solve

28. gsmcal_antmap
29. make_structurefunction
30. old_plot_gsmphases
31. createmap_plots
32. copy_plots
33. mk_results_dir
34. make_results_mapfile
35. move_results

Preprocess target Sbs
Averaging, flagging

Target pipeline steps

1. mk_inspect_dir
2. createmap_target
3. combine_data_target_map
4. get_ion_files
5. trans
6. ndppp_prep_target
7. create_ateam_model_map
8. make_sourcedb_ateam
9. expand_sourcedb_ateam
10. predict_ateam
11. ateamcliptar
12. combine_target_map
13. sortmap_target
14. do_sortmap_maps
15. dpppconcat
16. check_unflagged
17. check_unflagged_map
18. sky_tar

Use an Ateam skymodel to predict the Ateam visibilities and flag based on these

19. create_target_model_map
20. make_sourcedb_target
21. expand_sourcedb_target
22. gsmcal_parmmap
23. gsmcal_solve
24. gsmcal_apply
29. make_structurefunction
30. old_plot_gsmphases
31. createmap_plots
32. copy_plots
33. mk_results_dir
34. make_results_mapfile
35. move_results

Target pipeline steps

1. mk_inspect_dir
2. createmap_target
3. combine_data_target_map
4. get_ion_files
5. trans
6. ndppp_prep_target
7. create_ateam_model_map
8. make_sourcedb_ateam
9. expand_sourcedb_ateam
10. predict_ateam
11. ateamcliptar
12. combine_target_map
13. sortmap_target
14. do_sortmap_maps
15. dpppconcat
16. check_unflagged
17. check_unflagged_map
18. sky_tar
19. create_target_model_map
20. make_sourcedb_target
21. expand_sourcedb_target
22. gsmcal_parmmap
23. gsmcal_solve
24. gsmcal_apply
25. h5_imp_gsmsol_map
26. h5imp_gsmsol
27. plot_gsm_phases
28. gsmcal_antmap
29. make_structurefunction

Group subbands into bands and concatenate and flag

34. make_results_mapfile
35. move_results

Target pipeline steps

1. mk_inspect_dir
2. createmap_target
3. combine_data_target_map
4. get_ion_files
5. trans
6. ndppp_prep_target
7. create_ateam_model_map
8. make_sourcedb_ateam
9. expand_sourcedb_ateam
10. predict_ateam
11. ateamcliptar
12. combine_target_map
13. sortmap_target
14. do_sortmap_maps
15. dpppconcat
16. check_unflagged
17. check_unflagged_map
18. sky_tar
19. create_target_model_map
20. make_sourcedb_target
21. expand_sourcedb_target
22. gsmcal_parmmap
23. gsmcal_solve
24. gsmcal_apply
25. h5_imp_gsmsol_map
26. h5imp_gsmsol
27. plot_gsm_phases
28. gsmcal_antmap
29. make_structurefunction
30. old_plot_gsmphases
31. createmap_plots
32. copy_plots
33. mk_results_dir

Throw away bands with too little data after flagging

Target pipeline steps

1. mk_inspect_dir
2. createmap_target
3. combine_data_target_map
4. get_ion_files
5. trans
6. ndppp_prep_target
7. create_ateam_model_map
8. make_sourcedb_ateam
9. expand_sourcedb_ateam
10. predict_ateam
11. ateamcliptar
12. combine_target_map
13. sortmap_target
14. do_sortmap_maps
15. dpppconcat
16. check_unflagged
17. check_unflagged_map
18. sky_tar
19. create_target_model_map
20. make_sourcedb_target
21. expand_sourcedb_target
22. gsmcal_parmmap
23. gsmcal_solve
24. gsmcal_apply
25. h5_imp_gsmsol_map
26. h5imp_gsmsol
27. plot_gsm_phases
28. gsmcal_antmap
29. make_structurefunction
30. old_plot_gsmphases
31. createmap_plots
32. copy_plots
33. mk_results_dir
34. make_results_mapfile
35. move_results

Get (or provide) a skymodel for the target field

Target pipeline steps

1. mk_inspect_dir
2. createmap_target
3. combine_data_target_map
4.
5.
6.
7. create_ateam_model_map
8. make_sourcedb_ateam
9. expand_sourcedb_ateam
10. predict_ateam
11. ateamcliptar
12. combine_target_map
13. sortmap_target
14. do_sortmap_maps
15. dpppconcat
16. check_unflagged
17. check_unflagged_map
18. sky_tar
19. create_target_model_map
20. make_sourcedb_target
21. expand_sourcedb_target
22. gsmcal_parmmap
23. gsmcal_solve
24. gsmcal_apply
25. h5_imp_gsmsol_map
26. h5imp_gsmsol
27. plot_gsm_phases
28. gsmcal_antmap
29. make_structurefunction
30. old_plot_gsmphases
31. createmap_plots
32. copy_plots
33. mk_results_dir
34. make_results_mapfile
35. move_results

Phaseonly solve against the skymodel

Target pipeline steps

1. mk_inspect_dir
2. createmap_target
3. combine_data_target_map
4. get_ion_files
5. trans
6. ndppp_prep_target
7.
8.
9.
10. predict_ateam
11. ateamcliptar
12. combine_target_map
13. sortmap_target
14. do_sortmap_maps
15. dpppconcat
16. check_unflagged
17. check_unflagged_map
18. sky_tar
19. create_target_model_map
20. make_sourcedb_target
21. expand_sourcedb_target
22. gsmcal_parmmap
23. gsmcal_solve
24. gsmcal_apply
25. h5_imp_gsmsol_map
26. h5imp_gsmsol
27. plot_gsm_phases
28. gsmcal_antmap
29. make_structurefunction
30. old_plot_gsmphases
31. createmap_plots
32. copy_plots
33. mk_results_dir
34. make_results_mapfile
35. move_results

Make various inspection plots

Target pipeline steps

1. mk_inspect_dir
2. createmap_target
3. combine_data_target_map
4. get_ion_files
5. trans
6. ndppp_prep_target
7. create_ateam_model_map
8. make_sourcedb_ateam
9. expand_sourcedb_ateam
10. predict_ateam
11. ateamcliptar
12. combine_target_map
13.
14.
15.
16.

Copy/move the inspection plots and final calibrated MSs to appropriate places

17. check_unflagged_map
18. sky_tar
19. create_target_model_map
20. make_sourcedb_target
21. expand_sourcedb_target
22. gsmcal_parmmap
23. gsmcal_solve
24. gsmcal_apply
25. h5_imp_gsmsol_map
26. h5imp_gsmsol
27. plot_gsm_phases
28. gsmcal_antmap
29. make_structurefunction
30. old_plot_gsmphases
31. createmap_plots
32. copy_plots
33. mk_results_dir
34. make_results_mapfile
35. move_results

mk_inspect_dir

```
# create the inspection_directory if needed
mk_inspect_dir.control.kind                =  plugin
mk_inspect_dir.control.type                =  makeDirectory
mk_inspect_dir.control.directory           =  {{ inspection_directory }}
```

Make a directory to save the inspection plots

creatmap_target
combine_data_target_map

```
# generate a mapfile of all the target data
createmap_target.control.kind            =   plugin
createmap_target.control.type            =   createMapfile
createmap_target.control.method          =   mapfile_from_fo
createmap_target.control.mapfile_dir     =   input.output.mapfile_dir
createmap_target.control.filename        =   createmap_target.mapfile
createmap_target.control.folder          =   {{ target_input_path }}
createmap_target.control.pattern         =   {{ target_input_pattern }}

# combine all entries into one mapfile, for the sortmap script
combine_data_target_map.control.kind            =   plugin
combine_data_target_map.control.type            =   createMapfile
combine_data_target_map.control.method          =   mapfile_all_to_one
combine_data_target_map.control.mapfile_dir     =   input.output.mapfile_dir
combine_data_target_map.control.filename        =   combine_data_tar_map.mapfile
```

Make a list of all the input target SBs

- In this case 20 SBs matching the regular expression:
  - L232875*SB1[2-3]*.MS

get_ion_files
trans

```
# get ionex files once for every day that is covered by one of the input MSs
get_ion_files.control.type            = pythonplugin
get_ion_files.control.executable      = {{ prefactor_directory }}/bin/download_IONE
get_ion_files.control.max_per_node    =  1
get_ion_files.control.error_tolerance = {{ error_tolerance }}
get_ion_files.argument.flags          = [combine_data_target_map.output.mapfile]
get_ion_files.argument.ionex_server   = {{ ionex_server }}
get_ion_files.argument.ionex_prefix   = {{ ionex_prefix }}
get_ion_files.argument.ionexPath      = {{ ionex_path }}

# generate h5parm with the interpolated calibrator data to apply to the traget
trans.control.type              =  pythonplugin
trans.control.executable        =  {{ scripts }}/add_RMextract_and_times_to_H5parm.py
trans.control.max_per_node      =  {{ num_proc_per_node }}
trans.control.error_tolerance   =  {{ error_tolerance }}
```

Download external ionosphere information

- Note: this sometimes fails to download the file and crashes the pipeline – in this case, simply restart the pipeline until it works…

Calculate and add this the rotation measures to the calibration table

- cal_values_directory = /data010/scratch/wwilliams/pf_tutorial/pipeline/Pre-Facet-Calibrator-L232873/results/cal_values
- ! h5parm = {{ cal_values_directory }}/instrument.h5imp_cal
- Requires Rmextract python module to work

ndppp_prep_target

```
#run NDPPP on the target data to flag, transfer calibrator values, and av
ndppp_prep_target.control.type                          = dppp
ndppp_prep_target.control.max_per_node                  = {{ num_proc_per_node_limit }}
ndppp_prep_target.control.error_tolerance               = {{ error_tolerance }}
ndppp_prep_target.control.mapfiles_in                   = [createmap_target.output.mapfile]
ndppp_prep_target.control.inputkeys                     = [input_file]
ndppp_prep_target.argument.numthreads                   = {{ max_dppp_threads }}
ndppp_prep_target.argument.msin                         = input_file
ndppp_prep_target.argument.msin.datacolumn              = DATA
ndppp_prep_target.argument.msin.baseline                = CS*&; RS*&; CS*&RS*
ndppp_prep_target.argument.msout.datacolumn             = DATA
ndppp_prep_target.argument.msout.writefullresflag       = False
ndppp_prep_target.argument.msout.overwrite              = True
ndppp_prep_target.argument.steps                        =
[flag1,filter,flagamp,applyclock,applybandpass,applyoffset,applybeam,applyrm,count,flag2,count,avg]
```

Run DPPP to flag, apply calibration and average

- Steps are:
  - flag1, filter, flagamp
  - applyclock, applybandpass, applyoffset, applybeam, applyrm
  - count,flag2,count
  - avg
    - Note we are using more averaging than usual here
      - ! avg_timeresolution = 8. ## average to 8 sec/timeslot
      - ! avg_freqresolution = 97.64kHz ## average to 97.64 kHz/ch (= 2 ch/SB)
      - Will result in time-averaging and bandwidth smearing

```
# create a mapfile with the A-Team skymodel, length = 1
create_ateam_model_map.control.kind             =  plugin
create_ateam_model_map.control.type             =  addListMa
create_ateam_model_map.control.hosts            =  ['localho
create_ateam_model_map.control.files            =
[ {{ prefactor_directory }}/skymodels/Ateam_LBA_CC.skymodel
create_ateam_model_map.control.mapfile_dir      =  input.output.mapfile_dir
create_ateam_model_map.control.filename         =  ateam_model_name.mapfile

# make sourcedbs from the A-Team skymodel, length = 1
# outtype = blob, because NDPPP likes that
make_sourcedb_ateam.control.kind                =  recipe
make_sourcedb_ateam.control.type                =  executable_args
make_sourcedb_ateam.control.executable          =  {{ lofar_directory }}/bin/makesourcedb
make_sourcedb_ateam.control.error_tolerance     =  {{ error_tolerance }}
```

create_ateam_model_map
make_sourcedb_ateam
expand_sourcedb_ateam

Prepare a skymodel for the Ateam

- In this case we use a ‘cheap’ version of the skymodel
  - {{ prefactor_directory }}/skymodels/A-Team_lowres.skymodel
  - This has fewer components so is faster, but less precise, in the predict step

predict_ateam ateamcliptar

```
# Predict, corrupt, and predict the ateam-resolution model, length = nfiles
predict_ateam.control.type                          =  dppp
predict_ateam.control.mapfiles_in                   =
[ndppp_prep_target.output.mapfile,expand_sourcedb_ateam.output.mapfile]
predict_ateam.control.inputkeys                     =  [msin,sourcedb]
predict_ateam.control.inplace                       =  True
predict_ateam.control.max_per_node                  =  {{ num_proc_per_node_limit }}
predict_ateam.argument.numthreads                   =  {{ max_dppp_threads }}
predict_ateam.control.error_tolerance               =  {{ error_tolerance }}
predict_ateam.argument.msin.datacolumn              =  DATA
predict_ateam.argument.msout                        =  .
predict_ateam.argument.msout.datacolumn             =  MODEL_DATA
predict_ateam.argument.steps                        =  [predict]
predict_ateam.argument.predict.type                 =  predict
predict_ateam.argument.predict.operation            =  replace
```

Use NDPPP to predict the Ateam visibilities

- In this case we use a ‘cheap’ version of the skymodel

Flag frequencies and times where the Ateam contributes too much

combine_target_map
sortmap_target
do_sortmap_maps

```
# combine all entries into one mapfile, for the sortmap script
combine_target_map.control.kind            =   plugin
combine_target_map.control.type            =   createMapfile
combine_target_map.control.method          =   mapfile_all_to_one
combine_target_map.control.mapfile_dir     =   input.output.mapfile_d
combine_target_map.control.filename        =   combine_tar_map.mapfil
combine_target_map.control.mapfile_in      =   ndppp_prep_target.output.mapfile

# sort the target data by frequency into groups so that NDPPP can concatenate them
sortmap_target.control.type               =  pythonplugin
sortmap_target.control.executable         =  {{ scripts }}/sort_times_into_freqGroups.py
sortmap_target.argument.flags             =  [combine_target_map.output.mapfile]
sortmap_target.argument.filename          =  sortmap_target
sortmap_target.argument.mapfile_dir       =  input.output.mapfile_dir
sortmap_target.argument.target_path       =  input.output.working_directory/input.output.job_name
```

Map all the target SBs into a single group for sorting

Use the sort_times_into_freqGroups.py script (pythonplugin) to calculate the Bands into which to combine the SBs

And some more mapfile magic

dpppconcat

```
# run NDPPP to concatenate the target
dpppconcat.control.type                    =  dppp
dpppconcat.control.max_per_node            =  {{ num_proc_per_node_limit }}
dpppconcat.control.error_tolerance         =  {{ error_tolerance }}
dpppconcat.control.mapfile_out             =  do_sortmap_maps.output.groupmap # tell the pipeline to
give the output useful names
dpppconcat.control.mapfiles_in             =  [do_sortmap_maps.output.datamap]
dpppconcat.control.inputkey                =  msin
dpppconcat.argument.msin.datacolumn        =  DATA
dpppconcat.argument.msin.missingdata       =  True    #\ these two lines will make NDPPP generate dummy
data when
dpppconcat.argument.msin.orderms           =  False   #/ concatenating data
dpppconcat.argument.msout.datacolumn       =  DATA
dpppconcat.argument.msout.writefullresflag =  False
dpppconcat.argument.msout.overwrite        = True
dpppconcat.argument.steps                  =  [flag]     # run the aoflagger (this used to be an extra
```

Run DPPP to concatenate SBs and do additional flagging with AOFlagger

check_unflagged
check_unflagged_map

```
# check all files for minimum unflagged fraction
check_unflagged.control.type           =  pythonplugin
check_unflagged.control.executable     =  {{ scripts }}/check_unfl
check_unflagged.argument.flags         =  [dpppconcat.output.mapfi
check_unflagged.argument.min_fraction  =  {{ min_unflagged_fraction }}
# this step writes hostnames into "check_unflagged.flagged.mapfile" due to a "feature" of the
pythonplugin

# prune flagged files from mapfile
check_unflagged_map.control.kind            =  plugin
check_unflagged_map.control.type            =  pruneMapfile
check_unflagged_map.control.mapfile_in      =  check_unflagged.output.flagged.mapfile
check_unflagged_map.control.mapfile_dir     =  input.output.mapfile_dir
check_unflagged_map.control.filename        =  check_unflagged_map.mapfile
check_unflagged_map.control.prune_str       =  None
```

Use a pythonplugin to calculate the flagged fractions of each band and accept only bands that are not too heavily flagged

… and remove (prune) the heavily flagged bands from the mapfile

sky_tar

```
# if wished, download the tgss skymodel for the target
sky_tar.control.type                 =  pythonplugin
sky_tar.control.executable           =  {{ scripts }}/download_tgss_skymodel_target.py
sky_tar.argument.flags               =  [combine_target_map.output.mapfile]
sky_tar.argument.DoDownload          =  {{ use_tgss_target }}
sky_tar.argument.SkymodelPath        =  {{ target_skymodel }}
sky_tar.argument.Radius              =  5. #in degrees
```

Use a pythonplugin to get a target skymodel for DI calibration

- If no skymodel is specified it will download a skymodel from TGSS
- Here we specify the skymodel and don't download
  - The skymodel we provide comes from TGSS, but over a slightly smaller area (radius of 4 deg instead of the default 5 deg), and includes only the brighter sources
    - ! use_tgss_target = False ## "Force" : always download , "True" download if {{ target_skymodel }} does not exist , "False" : never download
    - ! target_skymodel = /data010/scratch/wwilliams/pf_tutorial/P23-tgss-small.skymode
  - Saves calibration time with some loss of quality

create_target_model_map
make_sourcedb_target
expand_sourcedb_target

```
# create a mapfile with the target skymodel, length = 1
create_target_model_map.control.kind                =  plugin
create_target_model_map.control.type                =  addListM
create_target_model_map.control.hosts               =  ['localh
create_target_model_map.control.files               =  [ {{ tar
create_target_model_map.control.mapfile_dir         =  input.ou
create_target_model_map.control.filename            =  target_model_name.mapfile

# make sourcedbs from the target skymodel, length = 1
# outtype = blob, because NDPPP likes that
make_sourcedb_target.control.kind                   =  recipe
make_sourcedb_target.control.type                   =  executable_args
make_sourcedb_target.control.executable             =  {{ lofar_directory }}/bin/makesourcedb
make_sourcedb_target.control.error_tolerance        =  {{ error_tolerance }}
make_sourcedb_target.control.args_format            =  lofar
```

Run makesourcedb on target skymodel
And relevant formatting of mapfiles

gsmcal_parmap

```
# generate mapfile with the parmDB names to be used in the gsmcal steps
gsmcal_parmmap.control.kind                =  plugin
gsmcal_parmmap.control.type                =  createMapfile
gsmcal_parmmap.control.method              =  add_suffix_to_file
gsmcal_parmmap.control.mapfile_in          =  check_unflagged_map.output.mapfile
gsmcal_parmmap.control.add_suffix_to_file =  /instrument_directionindependent
gsmcal_parmmap.control.mapfile_dir         =  input.output.mapfile_dir
gsmcal_parmmap.control.filename            =  gsmcal_parmdbs.mapfile
```

Prepare a mapfile for the output calibration tables

- will add suffix /instrument_directionindependent to the band Mss
- gsm = global sky model

gsmcal_apply

```
# apply the phase-only calibration solutions
# solve and apply are separate to allow to solve on a subset of baselines but app
gsmcal_apply.control.type                                    = dppp
gsmcal_apply.control.error_tolerance                         = {{ error_tolerance }}
gsmcal_apply.control.inplace                                 = True
gsmcal_apply.control.max_per_node                            = {{ num_proc_per_node_limit }}
gsmcal_apply.argument.numthreads                             = {{ max_dppp_threads }}
gsmcal_apply.argument.msin                                   = check_unflagged_map.output.mapfile
gsmcal_apply.argument.msin.datacolumn                        = DATA
gsmcal_apply.argument.msout.datacolumn                       = CORRECTED_DATA
gsmcal_apply.argument.msout.writefullresflag                 = False
gsmcal_apply.argument.steps                                  = [applycal]
gsmcal_apply.argument.applycal.type                          = applycal
gsmcal_apply.argument.applycal.correction                    = gain
gsmcal_apply.argument.applycal.parmdb                        = gsmcal_parmmap.output.mapfile
```

Use DPPP to do apply the phaseonly solutions to the data

h5_imp_gsmsol_map
h5imp_gsmsol

```
# generate a mapfile with all files in a single entry
h5_imp_gsmsol_map.control.kind               =  plugin
h5_imp_gsmsol_map.control.type               =  MapfileToOne
h5_imp_gsmsol_map.control.method             =  mapfile_all_to_one
h5_imp_gsmsol_map.control.mapfile_in         =  check_unflagged_map.output.mapfile
h5_imp_gsmsol_map.control.mapfile_dir        =  input.output.mapfile_dir
h5_imp_gsmsol_map.control.filename           =  h5_imp_gsmsol_map.mapfile

# import all instrument tables into one LoSoTo file
h5imp_gsmsol.control.type            =  pythonplugin
h5imp_gsmsol.control.executable      =  {{ scripts }}/losotoImporter.py
h5imp_gsmsol.control.error_tolerance =  {{ error_tolerance }}
h5imp_gsmsol.argument.flags          =  [h5_imp_gsmsol_map.output.mapfile,h5imp_gsmsol_losoto.h5]
h5imp_gsmsol.argument.instrument     =  /instrument_directionindependent
h5imp_gsmsol.argument.solsetName     =  sol000
```

Gather the output parmdb’s into an h5 file for LoSoTo

- Will be stored in solution set ‘sol000’

plot_gsm_phases

```
# plot the phase solutions from the phase-only calibration of the target
plot_gsm_phases.control.kind                                  =  recipe
plot_gsm_phases.control.type                                  =  executable_args
plot_gsm_phases.control.executable                            =  {{ losoto_directory }}/bin/losoto
plot_gsm_phases.control.max_per_node                          =  {{ num_proc_per_node }}
plot_gsm_phases.control.parsetasfile                          =  True
plot_gsm_phases.control.args_format                           =  losoto
plot_gsm_phases.control.mapfiles_in                           =  [h5imp_gsmsol.output.h5parm.mapfile]
plot_gsm_phases.control.inputkeys                             =  [hdf5file]
plot_gsm_phases.argument.flags                                =  [hdf5file]
plot_gsm_phases.argument.LoSoTo.Steps                         =  [plot]
plot_gsm_phases.argument.LoSoTo.Solset                        =  [sol000]
plot_gsm_phases.argument.LoSoTo.Soltab                        =  [sol000/phase000]
plot_gsm_phases.argument.LoSoTo.SolType                       =  [phase]
plot_gsm_phases.argument.LoSoTo.ant                           =  []
```

Plot the phase solutions with LoSoTo

Note that at the moment this doesn’t actually do anything in the pipeline (it doesn’t crash, but LoSoTo does nothing) – should be updated to reflect changes in LoSoTo. Compare how plotting is done in the calibrator pipeline with the prepare_losoto and process_losoto steps.

gsmcal_antmap
make_structurefunction

```
# generate mapfile with the antenna tables of the concatenated ta
gsmcal_antmap.control.kind                  =  plugin
gsmcal_antmap.control.type                  =  createMapfile
gsmcal_antmap.control.method                =  add_suffix_to_file
gsmcal_antmap.control.mapfile_in            =  dpppconcat.output.mapfile
gsmcal_antmap.control.add_suffix_to_file =  /ANTENNA
gsmcal_antmap.control.mapfile_dir           =  input.output.mapfile_dir
gsmcal_antmap.control.filename              =  gsmcal_antmaps.mapfile

# plot the phase solutions from the phase-only calibration of the target
make_structurefunction.control.kind                  =  recipe
make_structurefunction.control.type                  =  executable_args
make_structurefunction.control.executable            =  {{ scripts }}/getStructure_from_phases.py
make_structurefunction.control.max_per_node          =  {{ num_proc_per_node }}
make_structurefunction.control.mapfiles_in           =
```

Make a mapfile of the ANTENNA tables in the band MSs

And run the getStructure_from_phases.py script

- Produces a plot of the Structure function of the phase solutions for each band

old_plot_gsmphases

```
# plot the phase solutions from the phase-only calibration of the tar
old_plot_gsmphases.control.kind               =  recipe
old_plot_gsmphases.control.type               =  executable_args
old_plot_gsmphases.control.executable         =  {{ scripts }}/plot_solutions_all_stations.py
old_plot_gsmphases.control.max_per_node       =  {{ num_proc_per_node }}
old_plot_gsmphases.control.mapfiles_in        =
[gsmcal_parmmap.output.mapfile,check_unflagged_map.output.mapfile]
old_plot_gsmphases.control.inputkeys          =  [infile,outbase]
old_plot_gsmphases.control.arguments          =  [-p,infile,outbase]
```

More plotting of the phase solutions (different / old way) using plot_solutions_all_stations.py

createmap_plots
copy_plots

```
# generate a mapfile of all the diagnostic plots
createmap_plots.control.kind            =   plugin
createmap_plots.control.type            =   createMapfile
createmap_plots.control.method          =   mapfile_from_folder
createmap_plots.control.mapfile_dir     =   input.output.mapfile_dir
createmap_plots.control.filename        =   diagnostic_plots.mapfile
createmap_plots.control.folder          =   input.output.working_directory/input.output.job_name
createmap_plots.control.pattern         =   *.png

# copy the diagnostic plots to the results_directory
copy_plots.control.kind               =  recipe
copy_plots.control.type               =  executable_args
copy_plots.control.executable         =  /bin/cp
copy_plots.control.max_per_node       =  {{ num_proc_per_node_limit }}
copy_plots.control.mapfile_in         =  createmap_plots.output.mapfile
```

Make a list of the inspection image files and copy them to the inspection directory

mk_results_dir
make_results_mapfile
move_results

```
# create the results directory if needed
mk_results_dir.control.kind                 =  plugin
mk_results_dir.control.type                 =  makeDirectory
mk_results_dir.control.directory            =  {{ results_directory }}

# make mapfile with the filenames of the results that we want
make_results_mapfile.control.kind              =  plugin
make_results_mapfile.control.type              =  makeResultsMapfile
make_results_mapfile.control.mapfile_dir       =  input.output.mapfile_dir
make_results_mapfile.control.filename          =  make_results_mapfile.mapfile
make_results_mapfile.control.mapfile_in        =  check_unflagged_map.output.mapfile
make_results_mapfile.control.target_dir        =  {{ results_directory }}
make_results_mapfile.control.make_target_dir =  True
make_results_mapfile.control.new_suffix        =  .pre-cal.ms
```

Move the calibrated band MSs to a results directory

Success!

- The pipeline run will end with

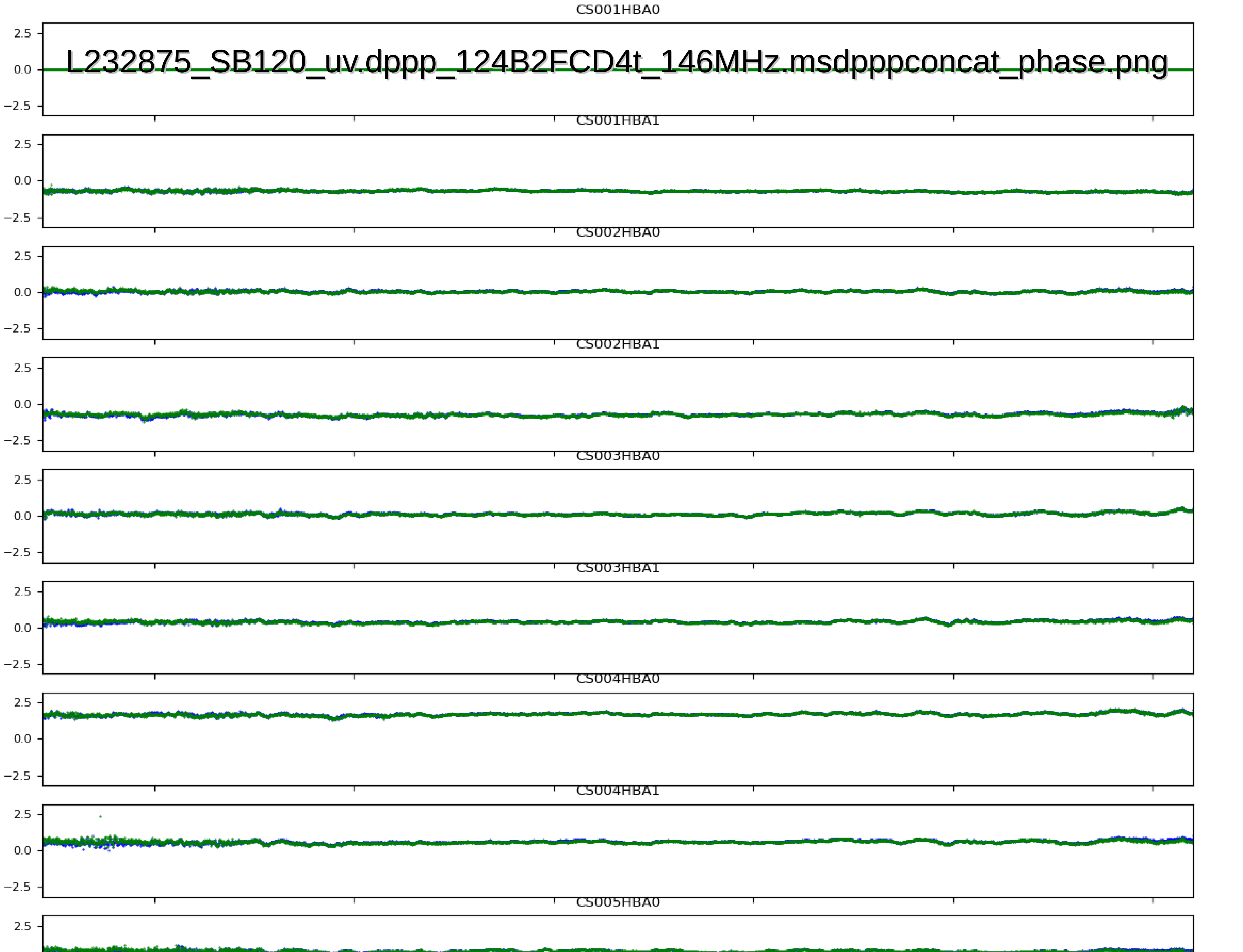
2018-09-17 15:47:12 INFO   genericpipeline: LOFAR Pipeline finished succesfully.

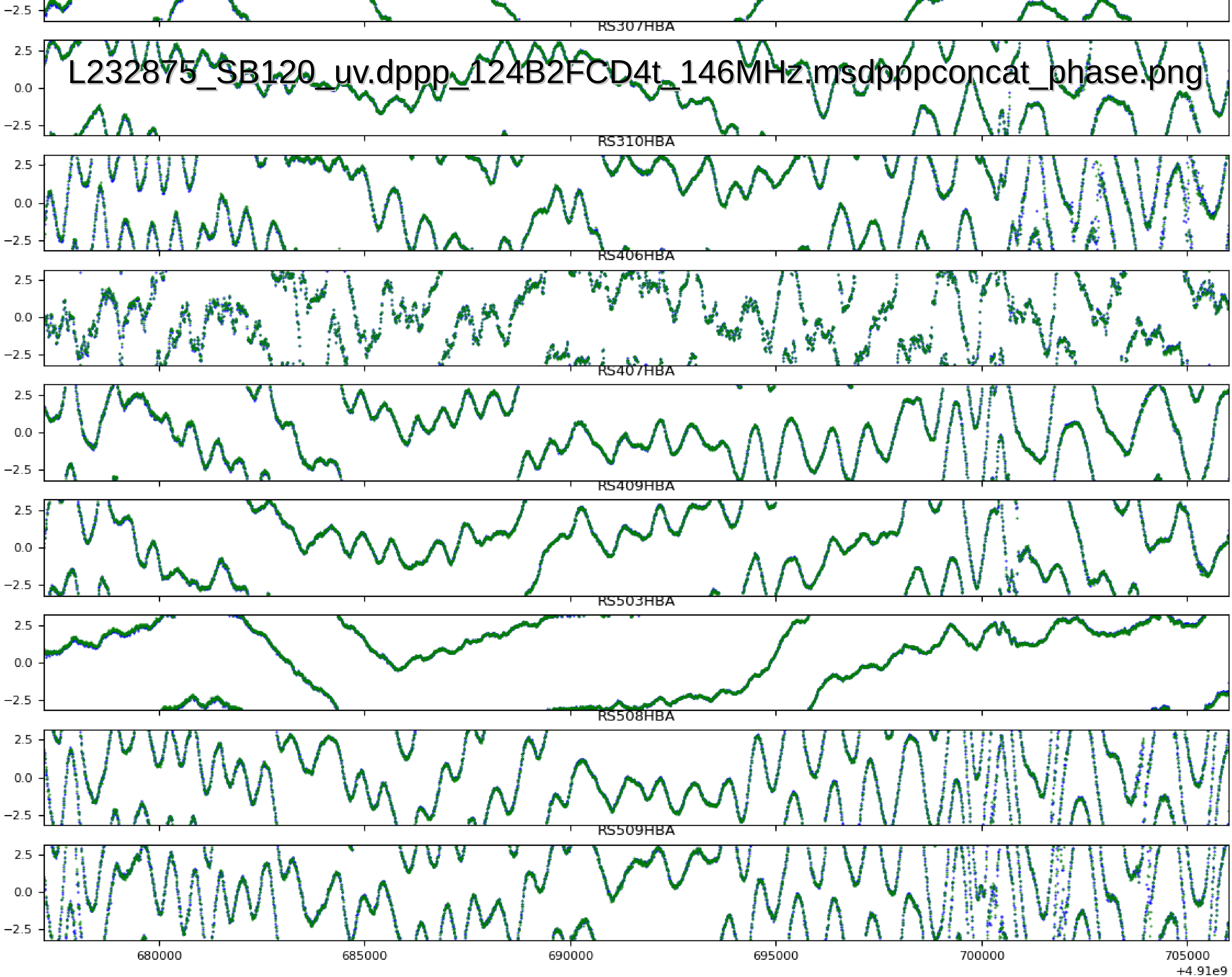
2018-09-17 15:47:12 INFO   genericpipeline: recipe genericpipeline completed

- … after about 3 hrs

Inspect the results

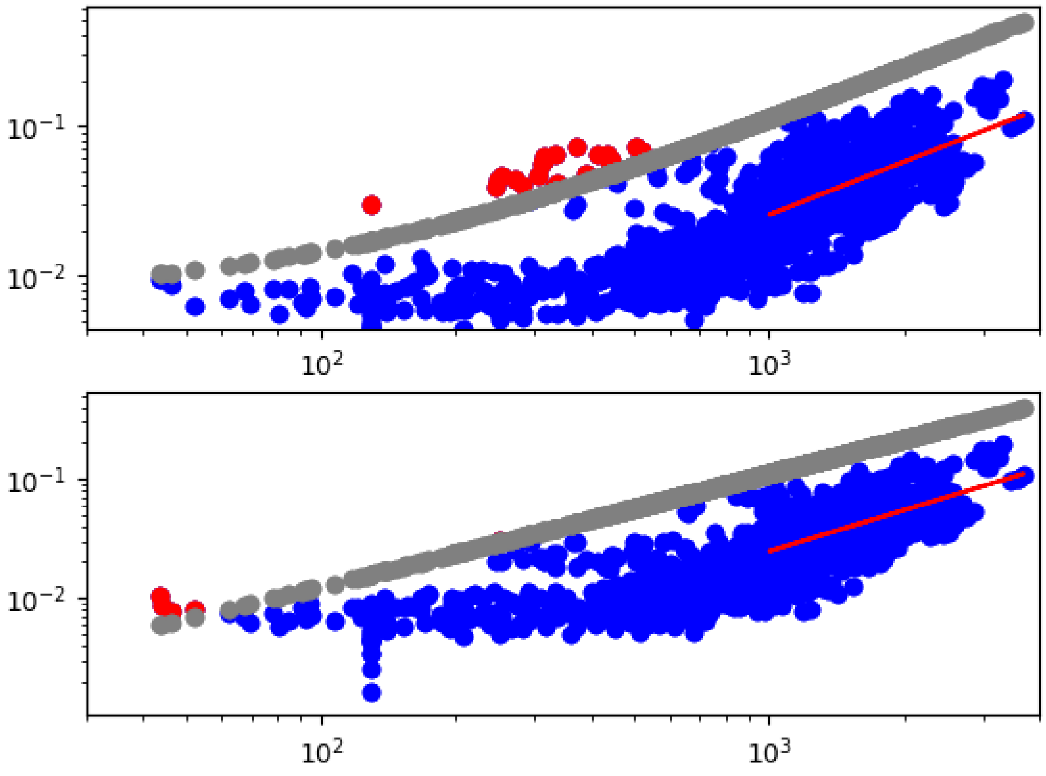
- Calibration plots are in
  - /data/scratch/<username>/pf_tutorial/pipeline/Pre-Facet-Target-L232875/results/inspection/
    - L232875_SB120_uv.dppp_124B2FCD4t_144MHz.msdpppconcat_structure.png
    - L232875_SB120_uv.dppp_124B2FCD4t_146MHz.msdpppconcat_structure.png
    - L232875_SB120_uv.dppp_124B2FCD4t_144MHz.msdpppconcat_phase.png
    - L232875_SB120_uv.dppp_124B2FCD4t_146MHz.msdpppconcat_phase.png


L232875_SB120_uv.dppp_124B2FCD4t_146MHz.msdpppconcat_phase.png
CS001HBA0
CS001HBA1
CS002HBA0
CS002HBA1
CS003HBA0
CS003HBA1
CS004HBA0
CS004HBA1
CS005HBA0
2.5
0.0
−2.5



L232875_SB120_uv.dppp_124B2FCD4t_146MHz.msdpppconcat_phase.png
RS307HBA
RS310HBA
RS406HBA
RS407HBA
RS409HBA
RS503HBA
RS508HBA
RS509HBA
2.5
0.0
−2.5
680000
685000
690000
695000
700000
705000
+4.91e9

L232875_SB120_uv.dppp_124B2FCD4t_146MHz.msdpppconcat_structure.png



Make an image

- If preparing for DDF stop here and precede with the ddf-pipeline
- If preparing for FACTOR run the Init-Subtract prefactor pipeline
  - A few flavours
    - Deep
    - Fast
  - Produces the residual data and sky model needed for FACTOR as well as high and low resolution images of the target field
- Here we will simply use wsclean to make an image of the 20 target SBs we processed

wsclean

- > module load wsclean
- > wsclean -size 4200 4480 -maxuv-l 7000 -baseline-averaging 6.72164158179 -local-rms-method rms-with-min -mgain 0.8 -auto-mask 3.3 -pol I -padding 1.4 -weighting-rank-filter 3 -auto-threshold 0.5 -j 8 -local-rms-window 50 -mem 20 -weight briggs 0.0 -name /data/scratch/<username>/pf_tutorial/P23-wsclean -scale 0.00208 -threshold 0.0 -niter 50000 -no-update-model-required -reorder -local-rms -fit-beam /data/scratch/<username>/pf_tutorial/pipeline/Pre-Facet-Target-L232875/results/L232875_*.pre-cal.ms
  - Should run for about 30 min
- > module load ds9
- > ds9 -scale limits -0.005 0.05 P23-wsclean-image.fits