1 LoSoTo: LOFAR Solution Tool[1]

The LOFAR Solution Tool (LoSoTo) is a Python package which handles LOFAR solutions in a variety of ways. The data files used by LoSoTo are not in the standard parmdb format used by BBS. LoSoTo uses instead an innovative data file, called H5parm, which is based on the HDF5 standard[2].

WARNING: LoSoTo is still in a beta version! Please report bugs to fdg@hs.uni-hamburg.de and drafferty@hs.uni-hamburg.de. LoSoTo will be soon integrated in the LOFAR environment but up to that moment to use it on cep1 users have to:

```
source /cep1home/fdg/scripts/losoto/tools/lofarinit.csh
```

1.1 H5parm

H5parm is simply a list of rules which specify how data are stored inside the tables of an HDF5 compliant file. We can say that H5parm relates to HDF5 in the same way that parmdb relates to MeasurementSet. The major advantage of using HDF5 is that it is an opensource project developed by a large community of people. It has therefore a very easy-to-use Python interface (the `tables` module) and it has better performance than competitors.

1.1.1 HDF5 format

There are three different types of nodes used in H5parm:

Array: all elements are of the same type.

CArray: like Arrays, but here the data are stored in chunks, which allows easy access to slices of huge arrays, without loading all data in memory. These arrays can be much larger than the physically available memory, as long as there is enough disk space.

Tables: each row has the same fields/columns, but the type of the columns can be different within each other. It is a database-like structure.

The use of tables to create a database-like structure was investigated and found to be not satisfactory in terms of performance. Therefore LoSoTo is now based on CArrays organized in a hierarchical fashion which provides enough flexibility but preserves performance.

1.1.2 Characteristics of the H5parm

H5parm is organized in a hierarchical way, where solutions of multiple datasets can be stored in the same H5parm (e.g. the calibrator and the target field solutions of the same observation) into different *solution-sets* (solset). Each solset can be thought as a container for a logically related group of solutions. Although its definition is arbitrary, usually there is one solset for each beam and for each scan. Each solset can have a custom name or by default it is called sol### (where ### is an increasing integer starting from 000).

Each solset contains an arbitrary number of *solution-tables* (soltab) plus a couple of Tables with some information on antenna locations and pointing directions. Soltabs also can have an arbitrary name. If no name is provided, then it is by default set to the solution-type name (amplitudes, phases, clock, tec...) plus again an increasing integer (e.g. amplitudes000, phase000...). Since soltab names are arbitrary the proper solution-type is stated in the *parmdb_type* attribute of the soltab node. Supported values are: amplitude, phase, scalarphase, rotation, clock, tec, and tecscreen.

Soltabs are also just containers; inside each soltab there are several CArrays which are the real data holders. Typically there are a number of 1-dimensional CArrays storing the *axes* values (see Table 1) and two *n*-dimensional (where *n* is the number of axes) CArrays, "values" and "weights", which contain the solution values and the relative weights.

Soltabs can have an arbitrary number of axes of whatever type. Here we list some examples:

aplitudes : time, freq, pol, dir, ant

[1] This section is maintained by Francesco de Gasperin (`fdg@hs.uni-hamburg.de`).

[2] `http://www.hdfgroup.org/HDF5/`

phases : time, freq, pol, dir, ant

clock : time, ant

tec : time, ant, dir

foobar : foo, bar...

Theoretically the values/weights arrays can be only partially populated, leaving NaNs (with 0 weight) in the gaps. This allows to have e.g. different time resolution in the core stations and in the remote stations (obviously this ends up in an increment of the data size). Moreover, solution intervals do not have to be equally spaced along any axis (e.g. when one has solutions on frequencies that are not uniformly distributed across the band). The attribute *axes* of the "values" CArrays states the axes names and, more important, their order.

Axis name	Format	Example
time (s)	float64	[4.867e+09, 4.868e+09, 4.869e+09]
freq (Hz)	float64	[120e6,122e6,130e6...]
ant	string (16 char)	[CS001LBA]
pol	string (2 char)	[XX, XY, RR, RL]
dir	string (16 char)	[3C196,pointing]
val	float64	[34.543,5345.423,123.3213]
weight (0 = flagged)	float32 [from 0 to 1]	[0,1,0.9,0.7,1,0]

Table 1: Default names and formats for axes values.

1.1.3 Example of H5parm content

Here is an example of the content of an H5parm file having a single solset (sol000) containing a single soltab (amplitude000).

```
# this is the solset
/sol000 (Group) ''

# this is the antenna Table
/sol000/antenna (Table(36,), shuffle, lzo(5)) 'Antenna names and positions'
  description := {
  "name": StringCol(itemsize=16, shape=(), dflt='', pos=0),
  "position": Float32Col(shape=(3,), dflt=0.0, pos=1)}
  byteorder := 'little'
  chunkshape := (2340,)

# this is the source Table
/sol000/source (Table(1,), shuffle, lzo(5)) 'Source names and directions'
  description := {
  "name": StringCol(itemsize=16, shape=(), dflt='', pos=0),
  "dir": Float32Col(shape=(2,), dflt=0.0, pos=1)}
  byteorder := 'little'
  chunkshape := (2730,)

# this is the soltab
/sol000/amplitude000 (Group) 'amplitude'

# this is the antenna axis, with all antenna names
/sol000/amplitude000/ant (CArray(36,), shuffle, lzo(5)) ''
  atom := StringAtom(itemsize=8, shape=(), dflt='')
```

```
  maindim := 0
  flavor := 'numpy'
  byteorder := 'irrelevant'
  chunkshape := (36,)

# direction axis, with all directions
/sol000/amplitude000/dir (CArray(2,), shuffle, lzo(5)) ''
  atom := StringAtom(itemsize=8, shape=(), dflt='')
  maindim := 0
  flavor := 'numpy'
  byteorder := 'irrelevant'
  chunkshape := (2,)

# frequency axis, with all the frequency values
/sol000/amplitude000/freq (CArray(5,), shuffle, lzo(5)) ''
  atom := Float64Atom(shape=(), dflt=0.0)
  maindim := 0
  flavor := 'numpy'
  byteorder := 'little'
  chunkshape := (5,)

# polarization axis
/sol000/amplitude000/pol (CArray(2,), shuffle, lzo(5)) ''
  atom := StringAtom(itemsize=2, shape=(), dflt='')
  maindim := 0
  flavor := 'numpy'
  byteorder := 'irrelevant'
  chunkshape := (2,)

# time axis
/sol000/amplitude000/time (CArray(4314,), shuffle, lzo(5)) ''
  atom := Float64Atom(shape=(), dflt=0.0)
  maindim := 0
  flavor := 'numpy'
  byteorder := 'little'
  chunkshape := (4314,)

# this is the CArray with the solutions, note that its shape is the product of all axes shapes
/sol000/amplitude000/val (CArray(2, 2, 36, 5, 4314), shuffle, lzo(5)) ''
  atom := Float64Atom(shape=(), dflt=0.0)
  maindim := 0
  flavor := 'numpy'
  byteorder := 'little'
  chunkshape := (1, 1, 10, 2, 1079)

# weight CArray, same shape of the "val" array
/sol000/amplitude000/weight (CArray(2, 2, 36, 5, 4314), shuffle, lzo(5)) ''
  atom := Float64Atom(shape=(), dflt=0.0)
  maindim := 0
  flavor := 'numpy'
  byteorder := 'little'
  chunkshape := (1, 1, 10, 2, 1079)
```

1.1.4 H5parm benchmarks

For a parmdb of 37 MB the relative H5parm with no compression is about 89 MB. Using a maximum compression, the H5parm ends up being 18 MB large. Reading times between compressed and non-compressed H5parms are comparable within a factor of 2 (compressed is slower). Compared to parmdb the reading time of the python implementation of H5parm (mid-compression) is a factor of a few (2 to 10) faster.

This is a benchmark example:

```
INFO: ### Read all frequencies for a pol/dir/station
INFO: PARMDB -- 1.0 s.
INFO: H5parm -- 0.35 s.
INFO: ### Read all times for a pol/dir/station
INFO: PARMDB -- 0.98 s.
INFO: H5parm -- 0.34 s.
INFO: ### Read all rotations for 1 station (slice in time)
INFO: PARMDB -- 0.99 s.
INFO: H5parm -- 0.53 s.
INFO: ### Read all rotations for all station (slice in time)
INFO: PARMDB -- 12.2 s.
INFO: H5parm -- 4.66 s.
INFO: ### Read all rotations for remote stations (slice in ant)
INFO: PARMDB -- 4.87 s.
INFO: H5parm -- 1.91 s.
INFO: ### Read all rotations for a dir/station and write them back
INFO: PARMDB -- 1.33 s.
INFO: H5parm -- 1.08 s.
INFO: ### Read and tabulate the whole file
INFO: parmdb -- 0.94 s.
INFO: H5parm -- 0.02 s.
```

1.2 LoSoTo

LoSoTo is made by several components. It has some tools used mostly to transform parmdb to H5parm and back (see Sec. 1.2.1). A separate program (`losoto.py`) is instead used to perform operations on the specified H5parm. `losoto.py` receives its commands by reading a parset file that has the same syntax of BBS/NDPPP parsets (see Sec.1.2.3).

1.2.1 Tools

There are currently four tools shipped with LoSoTo:

`parmdb_collector.py` fetches parmdb tables from the cluster

`H5parm_importer.py` creates an h5parm file from an instrument table (parmdb) or a globaldb created with `parmdb_collector.py`

`H5parm_merge.py` copy a solset from an H5parm files into another one

`H5parm_exporter.py` export an H5parm to a pre-existing parmdb

The usage of these tools is described in Sec. 1.3.

1.2.2 Operations

These are the operations that LoSoTo can perform:

RESET : reset the solution values to 1 for all kind of solutions but for phases which are set to 0.

PLOT : plot solutions in 1D/2D plots or plot TEC screens.

SMOOTH : smooth solutions using a multidimensional running median. The n-dimensional surface generated by multiple axis (e.g. time and freq) can be smoothed in one operation using a different FWHM for each axis.

CLIP : clip all solutions a certain factor above/below the median value.

ABS : take the absolute value of the solutions (probably most meaningful for amplitudes).

FLAG : iteratively remove a general trend from the solutions and then perform an outlier rejection. This operation is still to be implemented.

NORM : normalize solutions of an axis to have a chosen average value.

INTERP : interpolate solutions along whatever (even multiple) axis. Typically one can interpolate in time and/or frequency. This operation can also simply rescale the solutions to match the median of the calibrator solution on a specific axis.

CLOCKTEC : perform clock/tec separation.

TECFIT : fit TEC values per direction and station to phase solutions

TECSCREEN : fit TEC screens to TEC values

EXAMPLE : this is just an example operation aimed to help developing of new operations.

Beside these operations which require activation through the LoSoTo parset (see Sec. 1.2.3), one can call `losoto.py` with the "-i" option and passing an H5parm as argument to obtain some information on it. Information on a specific subset of solsets can be obtained with "-i -f solset_name(s)".

```
$ losoto.py -i single.h5

Summary of single.h5

Solution set 'sol000':
======================

Directions: pointing

Stations: CS001LBA   CS002LBA   CS003LBA   CS004LBA
          CS005LBA   CS006LBA   CS007LBA   CS011LBA
          CS017LBA   CS021LBA   CS024LBA   CS026LBA
          CS028LBA   CS030LBA   CS031LBA   CS032LBA
          CS101LBA   CS103LBA   CS201LBA   CS301LBA
          CS302LBA   CS401LBA   CS501LBA   RS106LBA
          RS205LBA   RS208LBA   RS305LBA   RS306LBA
          RS307LBA   RS310LBA   RS406LBA   RS407LBA
          RS409LBA   RS503LBA   RS508LBA   RS509LBA

Solution table 'amplitude000': 2 pols, 2 dirs, 36 ants, 1 freq, 4314 times

Solution table 'rotation000': 2 dirs, 36 ants, 1 freq, 4314 times

Solution table 'phase000': 2 pols, 2 dirs, 36 ants, 1 freq, 4314 times
```

1.2.3 LoSoTo parset

This is an example parset for the interpolation in amplitude:

```
LoSoTo.Steps = [interp]
LoSoTo.Solset = [sol000]
LoSoTo.Soltab = [sol000/amplitude000]
LoSoTo.SolType = [amplitude]
LoSoTo.ant = []
LoSoTo.pol = [XX,YY]
LoSoTo.dir = []

LoSoTo.Steps.interp.Operation = INTERP
LoSoTo.Steps.interp.InterpAxes = [freq, time]
LoSoTo.Steps.interp.InterpMethod = nearest
LoSoTo.Steps.interp.MedAxes = []
LoSoTo.Steps.interp.Rescale = F
LoSoTo.Steps.interp.CalSoltab = cal000/amplitude000
LoSoTo.Steps.interp.CalDir = 3C295
```

In the first part of the parset "global" values are defined. These are values named LoSoTo.val_name. In Table 2 the reader can find all the possible global values.

Var Name	Format	Example	Comment
LoSoTo.Steps	list of steps	[flag,plot,smoothPhases,plot_again]	sequence of steps names
LoSoTo.Solset	list of solset names	[sol000, sol001]	restrict to these solsets
LoSoTo.Soltab	list of soltabs: "solset/soltab"	[sol000/amplitude000]	restrict to these soltabs
LoSoTo.SolType	list of solution types	[phase]	restrict to soltab of this solution type
LoSoTo.ant	list of antenna names	[CS001_HBA]	restrict to these antennas
LoSoTo.pol	list of polarizations	[XX, YY]	restrict to these polarizations
LoSoTo.dir[a]	list of directions	[pointing, 3C196]	restrict to these pointing directions

[a] it is important to notice that the default direction (e.g. those related to BBS solving for anything that is not "directional": Gain, CommonRotationAngle, CommonScalarPhase...) have the direction: "pointing".

Table 2: Definition of global variables in LoSoTo parset.

For every stepname mentioned in the global "steps" variable the user can specify step-specific parameters using the syntax: LoSoTo.Steps.stepname.val_name. At least one of these options must always be present, which is the "Operation" option that specifies which kind of operation is performed by that step among those listed in Sec. 1.2.2. All the global variables (except from the "steps" one) are also usable inside a step to change the selection criteria for that specific step. A list of step-specific parameters is given in Table 3.

1.3 Usage

This is a possible sequence of commands to run LoSoTo on a typical observation:

1. Collect the parmdb of calibrator and target:

```
~fdg/scripts/losoto/tools/parmdb_collector.py -v -d "target.gds" -c "clusterdesc" -g globaldb_tgt
~fdg/scripts/losoto/tools/parmdb_collector.py -v -d "calibrator.gds" -c "clusterdesc" -g globaldb_cal
```

where "[target | calibrator].gds" is the gds file (made with combinevds) of all the SB you want to use. You need to run the collector once for the calibrator and once for the target. "Clusterdesc" is a cluster description file as the one used for BBS (not stand-alone).

2. Convert the set of parmdbs into an h5parm:

```
~fdg/scripts/losoto/tools/H5parm_importer.py -v tgt.h5 globaldb_tgt
~fdg/scripts/losoto/tools/H5parm_importer.py -v cal.h5 globaldb_cal
```

3. Merge the two h5parms in a single file (this is needed if you want to interpolate/rescale/copy solutions in time/freq from the cal to the tgt):

Var Name	Format	Example	Comment
Operations	string	RESET	An operation among those defined in Sec. 1.2.2
RESET			
Weight	bool	0 | 1	True: reset also the weights[a]
PLOT			
PlotType	1D | 2D | TECScreen	1D	Type of plot
Axes	list of axes names	[time]	For 1D plots is one axis name, two axes for 2D plots
MinMax	[float, float]	[0,100]	Force a min/max value for the dependent variable
Prefix	string	images/test_	Give a prefix to all the plots
SMOOTH			
Axes	list of axes names	[freq, time]	Axis name on which to smooth, may be multiple
FWHM	list of float	[10, 5]	FWHM, one for each axis
ABS			
CLIP			
Axes	list of axes names	[freq, time]	Axis name to take medians over, may be multiple
ClipLevel	float	5	factor above/below median at which to clip
FLAG (TBI)			
NORM			
NormVal	float	1.	the value to normalize the mean
NormAxis	axis name	time	the axis to normalize
INTERP			
CalSoltab	soltab name	cal001/amplitude000	The calibrator solution table
CalDir	dir name	3C196	Use a specific dir from CalSoltab instead that the same of the target
InterpAxes	list of axes names	[time, freq]	The axes along which to interpolate, can be multiple
InterpMethod	nearest | linear | cubic	linear	Type of interpolation method
Rescale	bool	0 | 1	Just rescale to the median value of CalSoltab, do not interpolate
MedAxis	axis name	time	rescale to the median of this axis
CLOCKTEC (TBI)			
TECFIT			
Algorithm	algorithm name	sourcediff	The algorithm to use in TEC fitting
MinBands	int	4	Minimum number of bands a source must have to be used
MaxStations	int	26	Maximum number of stations to use
OutSoltab	soltab name	ion000/tec000	the output solution table
TECSCREEN			
Height	float	200e3	The height in meters of the screen
Order	int	15	The maximum order of the KL decomposition
OutSoltab	soltab name	ion000/tecscreen000	the output solution table

[a] note that weights are currently not propagated back to parmdb

Table 3: Definition of step-specific variables in LoSoTo parset.

```
~fdg/scripts/losoto/tools/H5parm_merge.py -v cal.h5:sol000 tgt:cal000
```

4. Run LoSoTo using e.g. the parset given in Sec. 1.2.3:

```
~fdg/scripts/losoto/losoto.py -v tgt.h5 losoto-interp.parset
```

5. Convert back the h5parm into parmdb:

```
~fdg/scripts/losoto/tools/H5parm_exporter.py -v -c tgt.h5 globaldb_tgt
```

6. Redistribute back the parmdb tables into globaldb_tgt that are now updated (inspect with parmdbplot), there's no automatic tool for that yet.

1.4 Developing in LoSoTo

LoSoTo is much more than a stand alone program, the user can use LoSoTo to play easily with solutions and to experiment. The code is freely available and is already the result of the collaborative effort of several people. If interested in developing your own operation, please have a look at: `https://github.com/revoltek/losoto/`.

In the "tools" directory the user can find all the tools described in Sec. 1.2.1 plus some other program. All these programs are stand-alone. `losoto.py` is the main program which calls all the operations (one per file) present in the "operation" directory. It relays on the `h5parm.py` library which deals with reading and writing from an H5parm file and the `operations_lib.py` library which has some functions common to several operations.

An example operation one can use to start coding its own, is present in the "operation" directory under the name "`example.py`". That is the first point to start when interested in writing a new operation. The most important thing shown in the example operation is the use of the H5parm library to quickly fetch an write back solutions in the H5parm file.