LOFAR synthesis data handling
TaQL

Ger van Diepen
ASTRON

diepen astron nl

astron.nl/~gvd/tutorial4.pdf

Why TaQL?

One often wants to:

- Look at values in a MeasurementSet or other Casacore tables
- Check if data are correct
- Make small modifications
- Do some calculations
- Convert values to another reference frame or to readable text

Can be done in a Python script using pyrap.tables

- Can take some effort

Can often be done more easily using TaQL (Table Query Language)

- Very similar to SQL
- Easy to use for simple problems
  (although some people think it is very complex)
- Can handle complex problems, but requires some more knowledge (and thought)
  you have to know the function names, etc.

Fully described in http://casacore.github.io/casacore-notes/199.html

MeasurementSet

A structured set of tables

- Main table contains the visibility data, flags, UVW, weights, …

  A collection of rows/columns where each row contains the data of a single baseline and timeslot (and band/field):

- DATA	2D Complex array (nfreq*4); visibilities (XX,XY,YX,YY)
- FLAG	2D Bool array; flag per visibility (True=bad)
- UVW	U,V,W coordinate (meters)
- WEIGHT_SPECTRUM	weight per visibility
- TIME	time in MJD (seconds)
- ANTENNA1	first station of baseline (index in ANTENNA subtable)
- ANTENNA2	second station of baseline
- Several more columns	

- Subtables contain meta data
  - ANTENNA
    - NAME, POSITION, …
  - FIELD
    - PHASE_DIR, …
  - SPECTRAL_WINDOW
    - REF_FREQUENCY, CHAN_FREQ, CHAN_WIDTH, …
  - Several more subtables

Show MS info, columns and subtables:

```
showtable in=a.ms dm=f             # showtable –h for help
msoverview in=a.ms                 # msoverview –h for help
```

See http://casacore.github.io/casacore-notes/229.html for details

showtable in=a.ms dm=f

```
Structure of table /Users/diepen/data/LOFAR_L33277_SB010_uv.MS
------------------ Measurement Set
10353 rows, 23 columns (using 1 data managers)

  UVW             double   shape=[3] unit=[m,m,m] measure=uvw,J2000 directly stored
  FLAG            Bool     ndim=2
  FLAG_CATEGORY   Bool     ndim=3
  WEIGHT          float    ndim=1
  SIGMA           float    ndim=1
  ANTENNA1        Int      scalar
  ANTENNA2        Int      scalar
  ARRAY_ID        Int      scalar
  DATA_DESC_ID    Int      scalar
  EXPOSURE        double   scalar unit=[s]
  FEED1           Int      scalar
  FEED2           Int      scalar
  FIELD_ID        Int      scalar
  FLAG_ROW        Bool     scalar
  INTERVAL        double   scalar unit=[s]
  OBSERVATION_ID  Int      scalar
  PROCESSOR_ID    Int      scalar
  SCAN_NUMBER     Int      scalar
  STATE_ID        Int      scalar
  TIME            double   scalar unit=[s] measure=epoch,UTC
  TIME_CENTROID   double   scalar unit=[s] measure=epoch,UTC
  DATA            Complex  ndim=2
  WEIGHT_SPECTRUM float    ndim=2

 SubTables:
    /Users/diepen/data/LOFAR_L33277_SB010_uv.MS/ANTENNA
    /Users/diepen/data/LOFAR_L33277_SB010_uv.MS/DATA_DESCRIPTION
    /Users/diepen/data/LOFAR_L33277_SB010_uv.MS/FEED
    /Users/diepen/data/LOFAR_L33277_SB010_uv.MS/FLAG_CMD
    /Users/diepen/data/LOFAR_L33277_SB010_uv.MS/FIELD
    /Users/diepen/data/LOFAR_L33277_SB010_uv.MS/HISTORY
    /Users/diepen/data/LOFAR_L33277_SB010_uv.MS/OBSERVATION
    /Users/diepen/data/LOFAR_L33277_SB010_uv.MS/POINTING
    /Users/diepen/data/LOFAR_L33277_SB010_uv.MS/POLARIZATION
    /Users/diepen/data/LOFAR_L33277_SB010_uv.MS/PROCESSOR
    /Users/diepen/data/LOFAR_L33277_SB010_uv.MS/SPECTRAL_WINDOW
    /Users/diepen/data/LOFAR_L33277_SB010_uv.MS/STATE
    /Users/diepen/data/LOFAR_L33277_SB010_uv.MS/LOFAR_STATION
```

Some basic examples

```
taql 'select from a.ms where ANTENNA1!=ANTENNA2 giving cross.ms'
taql 'select from a.ms where ANTENNA1!=ANTENNA2 giving cross.ms as plain'
```

Select the cross-correlations and store the result in another table.
The first one as a RefTable (takes < 1 second); the second one makes a true copy (much slower).

```
taql 'SELECT * FROM my.ms OFFSET 1e20'
```

Do an empty selection to show the column names only.

```
taql 'select INTERVAL from my.ms limit 1'
```

Show the integration time (is constant in a LOFAR MeasurementSet)

```
    select result of 1 rows
1 selected columns:  INTERVAL
4.00556
```

```
taql 'select from my.ms orderby unique TIME'
taql 'select from my.ms orderby unique ANTENNA1,ANTENNA2'
taql 'select gcount() from my.ms'
```

Show the number of time slots, baselines, and total number of rows.

```
taql 'select ANTENNA1,ANTENNA2,UVW from my.ms
      orderby descending sumsqr(UVW[:2] limit 1'
```

Show the stations forming the longest baseline. Later we'll see how to get the names of the stations.

```
taql 'insert into a.ms/STATE set SIG=True, REF=False, CAL=0, LOAD=0,
        SUB_SCAN=0, OBS_MODE="", FLAG_ROW=False
```

Add a row to the STATE subtable and write the given column values in it.

TaQL commands

SQL-like data selection and manipulation

can often replace a python script

- `SELECT`
  - select columns, select rows, sort rows
- `UPDATE`
  - update data in one or more rows and columns
- `INSERT`
  - add rows and fill them
- `DELETE`
  - delete rows
- `CREATE TABLE`
  - create a new table
- `CALC`
  - calculate an expression, possibly using table data
- `COUNT`
  - count number of rows for table subsets (e.g., per baseline)

SELECT

Selects rows and /or columns and creates a new table

- result is normally a so-called RefTable (references the selection in the original table)
- Most parts are executed in the order given below, but SELECT after GROUPBY

SELECT columns	columns or expressions to select (default all)
FROM tables	the input table(s) to use
WHERE expression	which rows to select (default all); must result in bool scalar
GROUPBY columns	scalar columns or expressions to group and aggregate on
HAVING expression	groups to select
ORDERBY columns	sort the result on scalar columns or expressions
LIMIT N	maximum number of result rows (default all) (<0 is from end)
OFFSET M	skip first M result rows (default 0); useful with ORDERBY (<0 is from end)
GIVING table	persistent output table (default none)

TaQL functionality

- Support of sets/arrays and many array functions
- Support of glob patterns and regex
- Support of units
- Support of date/time (UTC)
- Support of cone search
- Support of user defined functions (measure handling)
- Advanced interval support
- Support of nested queries
- Aggregation (GROUPBY, HAVING)
- Limited joins (only implicit equi-join on rownumber or id)
- Case-insensitive (except column names and string constants)

Looks a bit overwhelming, but simple selections can be expressed simply, especially in pyrap.tables

May require careful thinking how to express a query

Some SQL knowledge is advantageous

See http://casacore.github.io/casacore-notes/199.html

Where can TaQL be used?

- in C++ casacore using function tableCommand
- in pyrap using function taql
  - indirectly in functions t.query, t.sort, t.select, and t.calc
- on command line using the program taql
  - use ‘taql -h’ to see how to use it
  - can also be used interactively (with command recall)

- Most important commands:
  - select
  - update
  - insert
  - calc

TaQL styles

TaQL indexing can have different styles.
The standard way resembles Python

- array indices and row numbers start counting at 0
- end is exclusive
  - [0:5] is [0,1,2,3,4]
  - [1:3:0.5] is [1., 1.5, 2., 2.5]
- array axes order is C-style (row major)
  - first axis varies slowest
  - e.g., DATA column in an MS has axes [freq,pol]

Opposite is the old Glish style.

Simple queries

Simple queries can be expressed simply using some *pyrap.tables* functions.
After

```
t = pt.table('my.ms')
```

```
t1 = t.query ('ANTENNA1 = ANTENNA2')       # select auto-correlations
```

results in

```
t1 = taql('select from my.ms where ANTENNA1 = ANTENNA2')
```

in fact, in:

```
t1 = taql('select from $1 where ANTENNA1 = ANTENNA2', t)
```

```
t2 = t1.sort ('TIME')                     # sort in time
```

results in

```
t2 = taql('select from $1 orderby TIME', t1)
```

```
t3 = t2.select ('ANTENNA1, ANTENNA2')     # select a few columns
```

results in

```
t3 = taql('select ANTENNA1, ANTENNA2 from $1', t2)
```

Combine as:

```
t3 = t.query ('ANTENNA1=ANTENNA2', sortlist='TIME', columns='ANTENNA1,ANTENNA2')
```

results in

```
t3 = taql('select ANTENNA1,ANTENNA2 from $1 where ANTENNA1=ANTENNA2 orderby TIME', t)
```

Data types

- bool T, F, True, or False
- int64
- double also sexagesimal format
  `12h34m56.78`
  `12d34m56.78 or 12.34.56.78`
- dcomplex 1 + 2i (or 2j) NB. normal addition
  `3*2+2i == 6+2i   not 6+6i`
- string in single and/or double quotes
  `'a"b'"a'c"   means   a"ba'c`
- datetime 10-Nov-2010/12:34:23.98
- regex perl-like m/CS.*/ p/CS*/

Both scalars and arrays of these data types (NOT an array of regex)

A table column or keyword can have any table data type

Operators

On scalar and/or array; in order of precedence:

```
**                          power
!   ~   +   -               unary operators;  ~ is bitwise complement
*   /   //  %               // is integer division result;   % is modulo;   1./2=0.5    1.//2=0
+   -                       + is also string concatenation
&                           bitwise and
^                           bitwise xor
|                           bitwise or
==  !=  >  >=  <  <=        normal comparison
~=  !~=                     about equal (near function with 1e-5 tolerance)
IN INCONE BETWEEN EXISTS
~  !~  LIKE                 pattern matching
&&                          logical and
||                          logical or
```

Operator names are case-insensitive. For SQL compliancy some operators have a synonym.

```
==      =
!=      <>
&&      AND
||      OR
!       NOT
^       XOR
```

Functions

- > 160 standard functions
- Mathematical
  - pi, e, c, sqrt, sin, cos, asin, sinh, exp, pow, log, ...
- Comparison
  - near (operator ~=), nearabs, isnan, isinf, isfinite
- String, Regex
- Date/time
- Array reduction
  - mean, min, sumsqr, median, fractile, ...
  - plural form (mins, …) for reduction of specific axes (e.g. per line)
  - sliding and boxed array functions
- Cone search
- Aggregation
  - gcount, gsum, gmean, gaggr, …
- Miscellaneous
  - angdist, rand
  - iif (condition, val1, val2) (like ternary ?: in C)
  - Type conversion and string formatting
- User defined
  - xxx.func
  - are taken from shared library libcasa_xxx.so or libxxx.so

User defined functions

- derivedmscal library for data in an MS (or CASA calibration table)
  - derived MS quantities like hourangle, AzEl, LAST
    - give hourangle of first station

      `mscal.ha1()`
  - by default PHASE_DIR in FIELD subtable is used
    can give another direction
    - give azimuth/elevation of SUN for this MS

      `mscal.azel('SUN')`
  - CASA selection

    `mscal.baseline("CS*&RS*")`
  - get info from subtables (only in casacore trunk)

    ```
    mscal.ant1name()
    mscal.spwcol('REF_FREQUENCY')
    ```

- meas library for conversion of measures
  - directions, epoch, and positions (knows standard ones)
    - get ITRF position of LOFAR

      `meas.itrf('LOFAR')`
    - get J2000 direction of SUN for WSRT at current time

      `meas.j2000('SUN', datetime(), 'WSRT')`
    - get apparent direction of CygA for next 31 days

      `meas.app('cyga', date()+[0:31], 'LOFAR')`

Units

Units are given implicitly or explicitly and converted if needed.

- A table column can have a unit
- Some functions result in a unit (e.g. *asin* results in unit rad)
- A sexagesimal constant has a unit (rad)
- A unit can be given after an expression. Conversion done if needed. Use quotes if a composite unit is used (e.g. 'km/s')

`12 s`	`12 seconds`	
`12 s + 1 h`	`3612 seconds`	
`1 h + 12 s`	`1.00333 hour`	
`(174 lb)kg`	`78.9251 kg`	(in case you use an American scale :-)
`(1 'm/s')'km/h'`	`3.6 km/h`	
`12h34m56.78`	`3.29407 rad`	
`12 m < 1 km`	`True`	
`sin(45 deg)`	`0.707107`	

These expressions can be given directly in taql program (assumes CALC if no command)

Regex

TaQL has rich support for pattern/regex matching (perl-like)

```
NAME ~ p/CS*/              match a glob pattern (as filenames in bash/csh)
NAME ~ p/CS*/i             same, but case-insensitive
NAME !~ p/CS*/             true if not matching the pattern

NAME ~ f/CS.*/             match an extended regular expression

NAME ~ m/CS/               true if part of NAME matches the regex (a la perl)
NAME ~ f/.*CS.*/           is the same

NAME like 'CS%'            SQL style pattern matching (also: not like)

NAME = PATTERN('CS*')      glob pattern using a function (also !=)
NAME = REGEX('CS.*')
NAME = SQLPATTERN('CS%')
```

Advanced

```
NAME ~ d/CS001/1           string distance (i.e., similarity)
```

Arrays

- Arrays can arise in several ways:
  - a table column containing arrays
  - a set results in a 1-dim array

    `[1:4]`   `['str1', 'str2']`
  - function *array* constructs an N-dim array

    `array([1:5], [3,4])`   array with shape [3,4] filled with [1,2,3,4] in each row

    `array(DATA, product(shape(DATA)))`   reshape to a vector
- Slicing is possible as in numpy (no negative values)

  axes can be omitted (yields full axis)

  `DATA[,0]`   only take XX correlation from MS

  `DATA[::2,]`   take every other channel and all correlations
- Full array arithmetic (array-array, array-scalar, scalar-array)
  - all operators, many functions (sin, etc.) available; they work element-wise
  - shapes have to match (no broadcasting like in numpy)
- Reduction functions (also partial for one or more axes)
  - min, median, any, all, ntrue, ..., mins, medians, anys, alls, ntrues, …
- Sliding (running) functions
  - runningmin, runningmedian, …
- Boxed functions
  - boxedmin, boxedmedian, …

Sets and intervals

- The IN operator can be used to test on sets or intervals (or arrays)
  - Interval (numeric, date/time)

    `date(TIME) in 12Jul2010 =:= 12Jul2011`

    - read `'x in start=:=end'` as `'start<=x<=end'`
    - = means closed interval side; use < for open side
    - multiple intervals and/or values can be given in a set
  - Set of values (numeric, string, date/time, interval)

    `ANTENNA1 in [1,2,3,4]` same as `ANTENNA1 IN [1:5]`

`datetime() in [12Nov2014 =:< 12Nov2015]` results in `T`

`datetime() in [12Nov2014 : 12Nov2015]` results in `F` (unless exactly at midnight)

`date() in [12Nov2014 : 12Nov2015]` results in `T`

- Left side can be a set or array; result is similarly shaped Bool array

  `[1,2] in [2,3,4]` results in `[F,T]`
- Right side can be a scalar; then IN is the same as ==

  `v in 1` is same as `v==1`
- A subquery results in a set (discrete values or intervals)

```
ANTENNA1 in [select rowid() from ::ANTENNA where NAME ~ p/CS*/]
TIME in [select from ::WEATHER where WIND_SPEED>10 giving
                          [TIME-INTERVAL/2 =:= TIME+INTERVAL/2]]
```

Grouping and Aggregation

- Use to get aggregated information per group (e.g., baseline)

```
groupby ANTENNA1,ANTENNA2
```

- Normally used with aggregation functions in SELECT

```
select gntrue(FLAG) from my.ms
select gntrue(FLAG) from my.ms groupby TIME
```

- Standard aggregation functions (all start with g)

  gmin, gmax, gsum, gsumsqr, gmean, gstddev, gmedian, gfractile, gntrue, …

- Special aggregation functions

  gfirst, glast — value in first/last row in group

  gaggr — concatenate all rows of group into single array

  `maxs(gaggr(abs(DATA)), 0,1)` — max amplitude per pol

- HAVING can be used to select groups

```
select gntrue(FLAG) as NT from my.ms groupby TIME having NT>0
```

Only selects groups where flags are set

UPDATE

Updates one or more columns in a table for each matching row

UPDATE table
: The table to update

SET column=expression, column=expression, ...
: The columns to update and their new values
: If column contains an array, a slice can be given
: A scalar can be assigned to an array (fills entire array)

FROM table(s)
: Possible other input tables to use

WHERE expression
: Which rows to update (default all); must result in bool scalar

ORDERBY columns
: Sort scalar columns or expressions (default no sorting)

LIMIT N
: Maximum number of matching rows to update (default all)

OFFSET M
: Skip first M matching rows (default 0); useful with ORDERBY

For example, set entire FLAG column to False

```
UPDATE your.ms SET FLAG=False
```

CALC

Calculates an expression; if a table is given, it is calculated for each matching row

CALC expression
: the expression to calculate

FROM tables
: the input table(s) to use (default none)

WHERE expression
: which rows to use (default all); must result in bool scalar

ORDERBY columns
: sort scalar columns or expressions (default no sorting)

LIMIT N
: maximum number of matching rows to update (default all)

OFFSET M
: skip first M matching rows (default 0); useful with ORDERBY

For example:

```
CALC ctod(TIME) from your.MS orderby unique TIME        # format (unique) times
```

Pretty printing using TaQL

- By default program taql pretty prints times

```
taql 'select TIME from ~/GER1.MS orderby unique TIME limit 2'
    select result of 2 rows
1 selected columns:  TIME
28-May-2001/02:26:55.000
28-May-2001/02:27:05.000
```

- and positions

```
taql 'select NAME,POSITION from ~/GER1.MS::ANTENNA'
    select result of 14 rows
2 selected columns:  NAME POSITION
RT0     [3.82876e+06 m, 442449 m, 5.06492e+06 m]
RT1     [3.82875e+06 m, 442592 m, 5.06492e+06 m]
```

- In python script use function ctod

```
# Pretty print TIME like '2001/05/28/02:29:45.000'
# Note the use of $t1 in the TaQL command;
# The function taql substitutes python variables given by $varname
t1 = t.sort ('unique desc TIME', limit=18)
pt.taql('calc ctod([select TIME from $t1])')

# or by passing the times as a python variable; need to tell unit is s
times = t1.getcol('TIME')
pt.taql('calc ctod($times s)')

# or the best way  (cdatetime is a synonym for ctod)
t1.calc ('cdatetime(TIME)')
```

Pretty printing using str(ing)

The function ***str*** can be used

- Converts values to strings
- Optional C-style format string or C++ width.prec
- Can also format date/time and angle using ‘or-ed’ format strings (as defined in class *MVTime*) and optional width

```
str(2rad, ‘angle’)                       # +114.35.30
str(2rad, 'angle|10')                    # +114.35.29.6125
str(4Mar1953, 'DMY|DAY|NO_TIME’)         # Wed-04-Mar-1953
str(1+2i, ’%f + %fj’)                    # 1.000000 + 2.000000j
str(123, ’value=%08d’)                   # value=00000123
str(‘abcdef’, 4)                         # abcd
```

Selection examples

Select all cross-correlations and save result (as a RefTable)

```
select from your.ms where ANTENNA1 != ANTENNA2 giving your_cross.ms
```

Select all cross-correlations and save result (as a PlainTable, thus deep copy); much slower

```
select from your.ms where ANTENNA1 != ANTENNA2 giving your_cross.ms as plain
```

Select all rows where ROW_FLAG does not match FLAG

```
select from your.ms where ROW_FLAG != all(FLAG)
```

Select all rows where some, but not all correlations in a channel are flagged.
Note: ntrues determines #flags per channel; shape(FLAG) gives [nchan,ncorr]; true result if true for any channel

```
select from your.ms where any(ntrues(FLAG,1) in [1:shape(FLAG)[1]])
```

Select some specific baselines (2-3, 2-4, 4-5, and 5-6)
Note: for a row containing e.g. baseline 2-5 you get [TTFF]&&[FFTF] ➔ [FFFF]

```
select from your.ms where any(ANTENNA1=[2,2,4,5] && ANTENNA2=[3,4,5,6])
```

Get the age (in days); could be used to test if an observation is old enough (note: date() is today)
Note use of :: in subtable name

```
calc date() - 4Mar1953
calc 20Jun2011 - date(TIME_RANGE[0]) from your.ms::OBSERVATION
```

Get unique times

```
select from my.ms orderby unique TIME
select unique TIME from my.ms
```

More selection examples

Select baselines (auto and cross) between LOFAR core and remote stations only
Note: this can also be achieved using the program msselect !!

```
select from my.ms where mscal.baseline('[CR]S*&&')    # CASA selection syntax
select from my.ms where all([ANTENNA1,ANTENNA2] in
                            [select rowid() from ::ANTENNA where NAME ~ p/[CR]S*/])
```

Select baselines containing international stations

```
select from my.ms where mscal.baseline('^[CR]S*&&*')
```

Regression test; check if DATA column is as expected (NaNs and rounding errors are possible)
Note: `t1.DATA ~= t2.DATA` is the same as `near(t1.DATA, t2.DATA, 1e-5)`

```
select from test.ms t1, ref.ms t2 where
    not all((isnan(t1.DATA) and isnan(t2.DATA)) or t1.DATA ~= t2.DATA)
```

More selection examples

Show short baselines (< 200 m) with the antenna names (with some kind of join using indexing)

```
 taql ‘select ANTENNA1, ANTENNA2, sqrt(sumsqr(UVW[:2])),
           [select NAME from ::ANTENNA][ANTENNA1] as ANTNAME1,
           [select NAME from ::ANTENNA][ANTENNA2] as ANTNAME2
    from ~/DATA/GER.MS where sumsqr(UVW[:2]) < 200*200 orderby unique ANTENNA1,ANTENNA2’

5 selected columns:  ANTENNA1 ANTENNA2 Col_3 ANTNAME1 ANTNAME2
0   1      73.9286   RT0        RT1
0   2      147.859   RT0        RT2
1   2      73.8642   RT1        RT2
1   3      147.723   RT1        RT3
```

With newest version of casacore (use Loflm)

```
  taql ‘select ANTENNA1, ANTENNA2, sqrt(sumsqr(UVW[:2])),
           mscal.ant1name() as ANTNAME1,
           mscal.ant2name() as ANTNAME2
    from ~/DATA/GER.MS where sumsqr(UVW[:2]) < 200*200 orderby unique ANTENNA1,ANTENNA2’
```

Update examples

Clear all flags in a MeasurementSet or set the flag if corresponding DATA is invalid

```
update your.ms set FLAG=False, ROW_FLAG=False
update your.ms set FLAG=!isfinite(DATA), ROW_FLAG=all(FLAG) # uses new FLAG values!!
```

Update the MOUNT in the ANTENNA table

```
update your.ms/ANTENNA set MOUNT='X-Y'
```

Put CORRECTED_DATA into the DATA column

```
update my.ms set DATA = CORRECTED_DATA
```

Put CORRECTED_DATA from that MS into DATA column of this MS
It requires that both tables have the same number of rows

```
update this.ms, that.ms t2 set DATA = t2.CORRECTED_DATA
```

Subtract background noise from an image using the median in a 51x51 box around each pixel
(updates the image, so one should have made a copy of my.img)

```
update my.img set map = map - runningmedian(map, 25, 25)
```

Flag XX data based on a simple median filter (per row); if set, current flag is kept

```
update my.ms set FLAG[,0]=FLAG[,0] || amplitude(DATA[,0]) >
      3*median(amplitude(DATA[,0])) where isdefined(DATA)
```

Add a line to the HISTORY table of a MeasurementSet (converts the MJD time automatically to sec)

```
insert into my.ms/HISTORY (TIME,MESSAGE) values (mjd(), "historystring")
```

Calculation examples

Count all flags in an MS (uses nested query)

```
CALC sum([select ntrue(FLAG) from my.ms])
```

Get percentage of unflagged data in an MS

```
CALC sum([select nfalse(FLAG) from my.ms]) * 100. /
     sum([select nelements(FLAG) from my.ms])
```

Get the hourangle of the first station (creates a new PlainTable, not RefTable because of expression in columns)

```
SELECT TIME, ANTENNA1, ANTENNA2, mscal.ha1() as HA1 from my.ms giving newtable
```

The same, but return it a as an array

```
CALC mscal.ha1() from my.ms orderby unique TIME,ANTENNA1
```

Angular distance between observation's phase reference direction(s) and a given direction
Do the same for CygA

```
CALC angdist([-3h45m34.95, 10d12m42.5], PHASE_DIR[0,]) FROM your.ms/FIELD
CALC angdist(meas.j2000('CygA'), PHASE_DIR[0,]) FROM your.ms/FIELD
```

Angular distance (in radians) between apparent positions of sun and moon at LOFAR core in coming month
(sun and moon are close on 5-Dec, so it'll be dark for poor Sinterklaas and Zwarte Piet)

```
CALC angdist(meas.app('SUN', 26Nov2013+[0:31], 'LOFAR'),
             meas.app('MOON',26Nov2013+[0:31], 'LOFAR'))
[1.53393, 1.33603, 1.13221, 0.921343, 0.702766, 0.476634, 0.244891, 0.0500374, 0.24891, 0.490751,
0.732487, 0.970688, 1.20361, 1.43043, 1.65101, 1.86568, 2.07504, 2.27972, 2.48028, 2.67696, 2.86884,
3.04371, 3.00427, 2.82496, 2.63905, 2.45208, 2.26415, 2.07456, 1.88225, 1.68595, 1.48433]
```

uvflux example

The Miriad program uvflux estimates the source I flux density and its standard deviation at the phase center without having to make an image.

A single, not too complicated TaQL command (courtesy Dijkema, Heald) provides the same functionality.

```
4.    select gstddev(SUMMED) as STDVALS,
4.           gmean(SUMMED) as MEANVALS,
4.           gcount(SUMMED) as NVALS
3.    from (select gmean(
1.                  sum(iif(FLAG[,0:4:3], 0, abs(DATA[,0:4:3]))) / nfalse(FLAG[,::3])
3.                 ) as SUMMED
            from ~/data/GER.MS
2.          where mscal.baseline('5km~10km') && !all(FLAG)
3.          groupby TIME)
```

A subquery is made to get the mean I flux (= 0.5*(XX+YY)) per time slot in the following way.

1. It first gets the mean of all channels for each baseline. Note that it uses `sum/n` to ignore flagged visibilities. The `iif` function tells to use 0 for them. Also note that XX is the 1st, YY the 4th polarisation, hence [0:4:3] (or [::3]) indexes these polarisations.

   Once masked arrays are supported by TaQL, it could be written as: `mean(DATA[,::3][FLAG[,::3]])`
2. It only uses the baselines with lengths between 5 and 10 km where not all visibilities are flagged.
3. Thereafter the average flux per time slot is determined in the subquery using the `gmean` aggregation and GROUPBY functionality. The result is called SUMMED.
4. Finally, the outer query uses aggregate functions to calculate the overall mean, standard deviation, and number of time slots. The final result is a table with 1 row and 3 columns.

Check StationAdder result

The StationAdder step in DPPP forms a new station ST001. It forms new baselines ST001 with all non-core stations by adding data of the baselines of non-core stations with core stations.

Per timeslot we want to check for each new baseline if the resulting DATA and UVW are as expected.

NewData = sum(OldData * Weight) / sum(Weight) # only use unflagged data points

NewUVW = sum(UVW * SumWeight)) / sum(SumWeight)) # SumWeight is sum of weights of all channels

1. In a LOFAR MS non-core/core baselines have ANTENNA1 as core and ANTENNA2 as non-core!!
2. Per time slot and non-core station we have to combine (group) the data -> use GROUPBY
```
select from a.ms groupby TIME,ANTENNA2
```
3. Only use unflagged data -> use 0 for flagged data points
```
iif(FLAG, 0, DATA)
```
   upcoming version of TaQL will support masked arrays -> DATA[FLAG]
4. Use the non-core/core baselines -> selection of such baselines most easily done with CASA syntax
```
where mscal.baseline("^CS*&CS*")
```
5. Calculate new data by aggregating the data per group and summing over the first axis (i.e., baselines)
```
sums(gaggr(DATA*WEIGHT_SPECTRUM), 0) / sums(gaggr(WEIGHT_SPECTRUM), 0)
```
6. Combine it all, write the result in an output table, and call the output column NDATA
```
select sums(gaggr(iif(FLAG,0,DATA)*WEIGHT_SPECTRUM), 0) /
                 sums(gaggr(iif(FLAG,0,WEIGHT_SPECTRUM)), 0) as NDATA
from a.ms where mscal.baseline("^CS*&CS*)
groupby TIME,ANTENNA2 giving avg.data as plain
```
7. Compare with StationAdder output (thus subset containing ST001 baselines)
```
select from avg.data t1, [select from a.ms where mscal.baseline("ST001&*"] t2
where not all (t1.NDATA ~= t2.DATA)
```

Check StationAdder's UVW

The following taql command calculates the average UVW per new baseline taking the data flags into account.

It was used by Leah Morabito to visually check DPPP's StationAdder results. If the results are stored in a table, another TaQL command could be used to compare with the StationAdder output.

It resembles the previous uvflux example. It uses a subquery to calculate intermediate results.

It also resembles numpy with its whole and partial array operations.

This query takes advantage of the knowledge that in a LOFAR MS the core stations are in ANTENNA1 for the baselines between core and non-core baselines, thus it can sum over ANTENNA2.

```
     select ctod(TIME), ANTENNA2,
4. sums(gaggr(UVW*SUMW),0) / gsum(SUMW) as UVW
1.    from [select TIME,UVW,ANTENNA2,
3.                  sum(iif(FLAG,0,WEIGHT_SPECTRUM)) as SUMW
1.          from a.ms
2.          where mscal.baseline("^CS*&CS*")]
4.    groupby TIME,ANTENNA2
```

1. The subquery selects the required columns.
2. It only uses the rows with cross-correlation baselines between non-core and core (CASA syntax).
3. As part of the subquery output it calculates SUMW, the sum of the weights where the data are not flagged.
   Note: in the next release of casacore one can probably use masked arrays like:

   ```
   sum(WEIGHT_SPECTRUM[FLAG])
   ```

4. The main query aggregates the subquery output to a single array per TIME and ANTENNA2. It sums each array over the first axis, resulting in the weighted sum for U, V, and W. Finally it is divided by the summed SUMW to get the weighted averaged U, V, and W per TIME and ANTENNA2.

Advanced examples 1

Check if an MS is in non-descending time order and check for missing time slots.
It is done by comparing the table subsets of row 0..n-1 and row 1..n (created by a nested query in the FROM).
Note the use of LIMIT -1 to denote the one but last row (a la python indexing).

```
select t1.TIME, t1.TIME-t2.TIME as STEP from
    [select from ~/L33277_SAP000_SB000_uv.MS limit -1] t1,
    [select from ~/L33277_SAP000_SB000_uv.MS offset 1] t2
  where int((t2.TIME-t1.TIME)/t1.EXPOSURE+0.0001) not in [0,1]
```

Check if the QUALITY subtables generated by NDPPP and rficonsole are the same, except for the baselines that were flagged already (because rficonsole counts them, but NDPPP does not).
Note that t3 contains the baselines with some flagged data in the original MS.

```
select from ndppp.ms/QUALITY_BASELINE_STATISTIC t1,
            rficonsole.ms/QUALITY_BASELINE_STATISTIC t2,
            [select unique ANTENNA1,ANTENNA2 from original.ms where any(FLAG)] t3
  where any(t1.VALUE != t2.VALUE) and
  !any(ANTENNA1 = [select ANTENNA1 from t3] && ANTENNA2 = [select ANTENNA2 from t3])
```

Advanced examples 2

Check if the demixing solutions in the old and new way are the same for CygA.
The old one only contains CygA, the new one contains more, so a selection is needed.

```
taql 'select from instrumentold t1,
                  [select from instrument3 where NAMEID in
          [select rowid() from ::NAMES where NAME~m/CygA/]] t2
      where !all(t1.VALUES ~= t2.VALUES)'

    select result of 0 rows
```

Note that above assumes that the table orders are the same. If not,

```
   orderby STARTX,STARTY
```

should be used for both tables to make them the same.

```
taql 'select from [select from instrumentold orderby STARTX,STARTY] t1,
                  [select from instrument3 where NAMEID in
          [select rowid() from ::NAMES where NAME~m/CygA/] orderby STARTX,STARTY] t2
      where !all(t1.VALUES ~= t2.VALUES)'
```

Advanced examples 3

Swap columns ANTENNA1 and ANTENNA2 in a MeasurementSet.
The problem is that TaQL updates in place, thus as soon as one ANTENNA column is set, its original values are lost.

The first solution works fine (think about it), but is some kind of a hack.

```
update my.ms set ANTENNA1 = ANTENNA1+ANTENNA2,
                 ANTENNA2 = ANTENNA1-ANTENNA2, ANTENNA1 = ANTENNA1-ANTENNA2
```

The following solution is neater.
It holds the original values of ANTENNA1 in a temporary table in memory.

```
update my.ms, [select ANTENNA1 from my.ms giving as memory] as orig
            set ANTENNA1 = ANTENNA2, ANTENNA2 = orig.ANTENNA1
```