Refactoring SCHED from Fortran to Python step-by-step.

Bob Eldering, Joint Institute for VLBI ERIC (JIVE)

JIVE Joint Institute for VLBI ERIC

SCHED

SCHED is a program for scheduling Very Long Baseline Interferometry (VLBI) observations. The input to SCHED is a text file, which can be as simple as:

```
setinit = eg1024-1024 /
  band = '6cm'
  nchan    = 16
  bits     = 2
  bbfilter = 16.0
  pol      = dual /
stations = eflsberg, jodrell1, cambg32m,
  onsala85, wb, noto, medicina, torun
setup = eg1024-1024
source='J0045+4555'  dur=1:30 gap=1:00 /
source='M31'         dur=3:30 gap=0    /
```

SCHED writes several different types of output files, some of which are useful for the scheduler and some of which are meant for the systems at the stations and the correlator.

SCHED is written in Fortran 77 and developed mainly by Craig Walker at the National Radio Astronomy Observatory (NRAO). Continuous developments at the stations require updates to SCHED. With the deployment of the Digital Base Band Converter (DBBC) system in the European VLBI Network (EVN), we at JIVE decided to work on the SCHED code.

URL: `www.aoc.nrao.edu/~cwalker/sched`

F2PY

Instead of continuing working on the Fortran code at JIVE, we decided to do new developments in Python. However, we do not want to start from scratch and rewrite everything in Python in one go, as SCHED is quite a large program (82K lines of code). This is where F2PY comes in. The purpose of the F2PY *-Fortran to Python interface generator-* is to provide a connection between the Python and Fortran languages. F2PY is a part of NumPy (numpy.f2py), it facilitates creating/building Python C/API extension modules that make it possible:

- to call Fortran 77 subroutines
- to access Fortran 77 COMMON blocks

from Python.

When we rewrite a Fortran subroutine to Python, any calls to other Fortran subroutines can be handled by the extension module. This allows us to rewrite parts of the SCHED program in Python, while still re-using other parts.

URL: `docs.scipy.org/doc/numpy/f2py`

Author

Bob Eldering
`eldering@jive.eu`

pySCHED

We call the Python fork of SCHED pySCHED. The main reason for the switch to Python, is that it is a modern language. This will make it easier to maintain pySCHED in the future. And since the astronomy programming community seems to be more familiar with Python than with Fortran, the switch to Python will allow more people to change and contribute to the code.

Also, using Python made it easy to add a number of features to SCHED:

- With `setuptools` and the Python Package Index (PyPI) it is easy to install pySCHED, simply type: `pip3 install pythonSCHED` in a shell.
- SCHED has a large amount of data files describing for examples sources and stations. The distribution of these data files is closely tied to the distribution of the program itself. In pySCHED, these files are updated on start-up, using the Python module `git` and a repository on GitHub.
- Using Matplotlib we get plots that automatically allow zooming, panning, printing, etc.

GitHub URL: `github.com/jive-vlbi/sched`
PyPI URL: `pypi.org/project/pythonSCHED`

Applying F2PY to SCHED

Fortran

```
      SUBROUTINE GEOCHK( JSCN, ISCN, STARTB, TGEOEND, OKGEO, USEGEO,
     1                   SEGELEV )
      DOUBLE PRECISION  STARTB, TGEOEND
      DOUBLE PRECISION  SRCSEP, REQSEP, TFREQ
      INTEGER           JSCN, ISCN, USEGEO(*)
      LOGICAL           OKGEO(*)
      REAL              SEGELEV(MAXSTA,MGEO)
```

Subroutine arguments in Fortran 77 are passed by reference. This does not match Python method calling. This is solved by marking arguments as input and/or output. The easiest way to do this is by adding special comments in the code. An example is shown here, the original on the left and the added comments on the right.

Python

```
      SUBROUTINE GEOCHK( JSCN, ISCN, STARTB, TGEOEND, OKGEO, USEGEO,
     1                   SEGELEV )
Cf2py intent(in) JSCN, ISCN, STARTB, TGEOEND
Cf2py intent(in, out) OKGEO, USEGEO
Cf2py intent(out) SEGELEV
      DOUBLE PRECISION  STARTB, TGEOEND
      DOUBLE PRECISION  SRCSEP, REQSEP, TFREQ
      INTEGER           JSCN, ISCN, USEGEO(*)
      LOGICAL           OKGEO(*)
      REAL              SEGELEV(MAXSTA,MGEO)
```

```
      CALL GEOCHK( JSCN, ISCN, STARTB, TGEOEND, OKGEO, USEGEO,
     1             SEGELEV )
C
      DO ITRIAL = 1, GEOTRIES
         IF( GEOPRT .GE. 1 ) THEN
            CALL WLOG( 0, ' ' )
            MSGTXT = ' '
            WRITE( MSGTXT, '( A, I5, A, I5 )' )
     1           'GEOMAKE starting to construct trial segment',
     2           ITRIAL, '.  First scan: ', ISCN
            CALL WLOG( 0, MSGTXT )
         END IF
```

Running the example above through F2PY, we get a Python module (called `schedlib` in this example), which allows us to call Fortran code from Python. On the left is the original Fortran code calling `GEOCHK` with some context and on the right is the Python translation of the same code.

```
    ok_geo = np.empty(shape=schedlib.schsou.geosrci.shape,
                      dtype=bool)
    use_geo = np.empty(shape=schedlib.schsou.geosrci.shape,
                       dtype=int)
    ok_geo, use_geo, seg_elevation = schedlib.geochk(
        j_scan, scan_index, start_time, end_time, ok_geo, use_geo)

    for trial in range(1, schedlib.schsou.geotries + 1):
        if schedlib.schsou.geoprt >= 1:
            schedlib.wlog(0, "GEOMAKE starting to construct "
                          "trial segment {}.  First scan: {}".\
                          format(trial, scan_index))
```

```
      INTEGER          NLGP, NRESTFQ(MAXLGP)
      DOUBLE PRECISION RESTFREQ(MAXLCH,MAXLGP)
      CHARACTER        LINENAME(MAXLGP)*8
C
      COMMON  /SCHLIN/ RESTFREQ, NLGP, NRESTFQ
      COMMON  /SCHCLI/ LINENAME
```

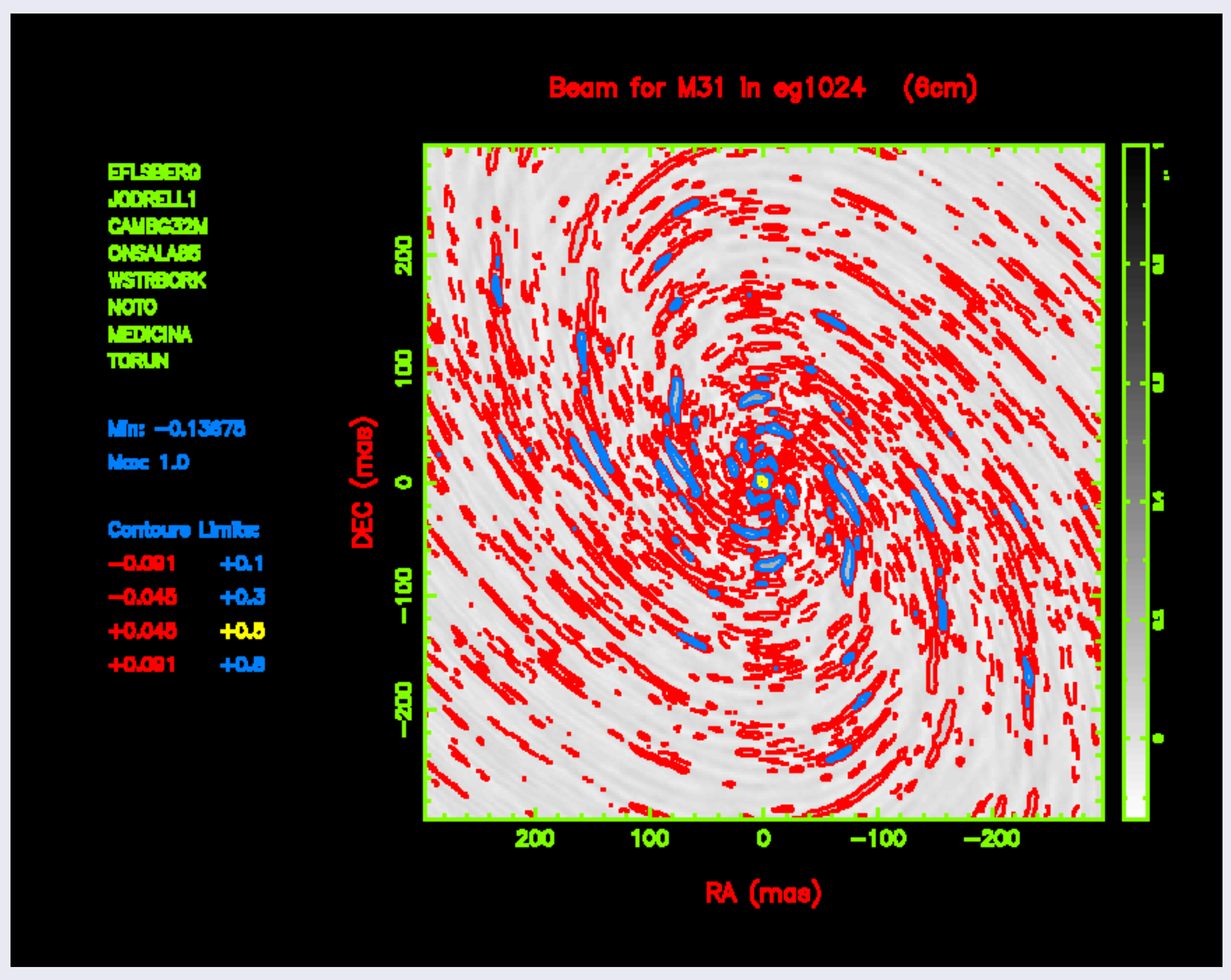
```
      DO IGP = 1, NLGP
         PRINT *, LINENAME(I), RESTFREQ(I)
      END DO
```

COMMON blocks are also made available in `schedlib`. SCHED does not use data structures, instead lists of data structures are represented using arrays: members are grouped together by using the same index. In pySCHED we formalize this convention by actually grouping these COMMON block arrays into classes. The base Python class `Catalog` has all the logic to get access to the COMMON block variables. Classes derived from `Catalog` only have to define which members are to be grouped together. Here an example is shown of such a derived class with a basic usage snippet.

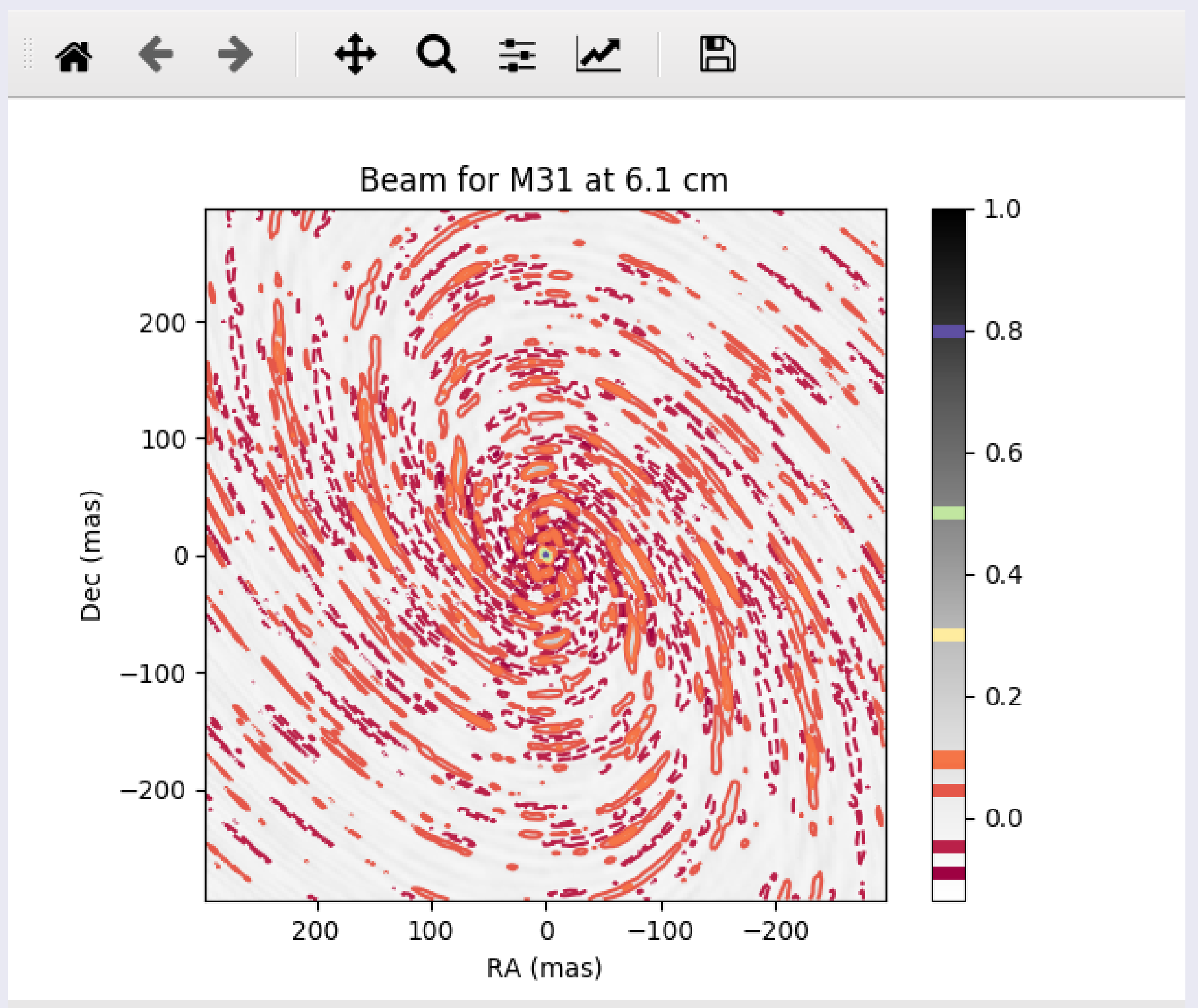
```
class LineRestFrequencyCatalog(Catalog):
    block_items = {
        schedlib.schlin : [
            "restfreq",
            "nrestfq"
        ],
        schedlib.schcli : [
            "linename"
        ]}

    def scheduled_slice(self):
        return slice(s.schlin.nlgp)
```

```
catalog = LineRestFrequencyCatalog()
for entry in catalog.entries:
    print(entry.linename, entry.restfreq)
```



Of course a poster is not complete without at least some pictures. On the left and right a beam plot is shown using PGPLOT in SCHED and using Matplotlib in pySCHED respectively.


Beam for M31 at 6.1 cm
Dec (mas)
RA (mas)
200
100
0
−100
−200
1.0
0.8
0.6
0.4
0.2
0.0