A Better Interface Between Scientists and Data Reduction Software

B. Nikolic

Astrophysics Group, Cavendish Laboratory, University of Cambridge
http://www.mrao.cam.ac.uk/~bn204/

ALMA Software Development Workshop
NRAO/Charlottesville
October 2011

UNIVERSITY OF CAMBRIDGE

Outline

Introduction

Examples and Derived Proposals
- Operation Folding
- Dependency tracking
- Intermediate product tracking
- Data reduction branches

Theory

Implementing a better interface

Summary

Aims for this talk

This talk will aim to convince you:

- We can make data reduction significantly easier, faster and more reliable
- We can do this relatively <u>easily</u>
- This is likely to be worth doing

I would like to take home:

- Feedback on the ideas and approach presented
- <u>Is it</u> worth doing?
- Good project for the ALMA development programme?

Data reduction, software, and pipelines

- This talk is relevant to data reduction like we currently do for aperture synthesis interferometry:
  1. Environments like CASA or AIPS+Python wrappers
  2. Iterative flagging/calibration/imaging
  3. Large datasets, expensive to move around and expensive to process
  4. Mostly about command interface $\equiv$ Language
- A fully commissioned pipeline that delivers reduced calibrated data will remove the need to develop this for ALMA! (No interaction – no interface needed!)
- However some of the ideas may be useful in development of a pipeline too

Automatised data reduction/Human decision making

We can't solve all data reduction problems – Lets give everybody an easy interface to solve it themselves [1]

[1] This particular paraphrasing inspired by slides of one of David Nolen's presentations

Automatised data reduction/Human decision making

- This is how I would approach reducing a significant quantity of observations from ALMA
  - All the ‘tools’ are there, linking them up is too time consuming/difficult
- Currently mostly just ideas – little implementation yet
- Aiming to develop a small, scaled back, prototype implementation for analysis of WVR testing data
- Likely would require funding from the ALMA development programme for a full implementation

A better interface can be:

1. Faster
   - Less Wall-clock time
   - Less Scientist's time
   - Fewer computational resources
2. More reliable
   - Fewer opportunities for user error
   - Easier to make fully repeatable
   - Easier to review by reading the script
3. More communicable
   - The data reduction script can be used to communicate what needs to be done to other people as well as the computer

Why?

- Much more data/observations/spectral lines/fields per radio astronomer! Can we keep up?
- Barriers to *understanding* and *doing* aperture synthesis must be minimised – *'we'll do it for you'* is not a solution
- In some aperture synthesis experiments there is no single 'right' way of doing the reduction – peers must be able to easily repeat and adjust our reduction
- In new generation of telescopes much cheaper to move data reduction 'scripts'/'recepies' and products rather than the visibility data

Straw-man requirements

1. Commands should be designed to best communicate to other scientists what needs to be done
2. Trying out different parameters/commands should be easy, efficient – should recognise there is no single ‘correct’ result
3. Concise
4. Efficient, fast

Outline

Introduction

Examples and Derived Proposals
- Operation Folding
- Dependency tracking
- Intermediate product tracking
- Data reduction branches

Theory

Implementing a better interface

Summary

Outline

Simple flagging-based example

Note:

- I use flagging here for illustration only
- Similar principles apply to many other operations

Flagging fragment

A fragment of an ALMA data reduction script:

```
# Python/CASA
vis="mydata.ms"
flagdata(vis=vis, autocorr=True)
flagdata(vis=vis, mode='shadow', diameter=12.0)
flagdata(vis=vis, antenna='DV04')
```

This likely causes *three* complete iterations through the data. Why:

- The interface is fully procedural
- Each `flagdata` only knows about itself – it doesn't know it is followed by another similar command

If Input/Output limited $\Rightarrow$ big performance penalty

Operation folding 'by hand'

Compare to following hypothetical command:

```
# Python/Something like CASA
vis="mydata.ms"
flagdata_(vis=vis, [ {'autocorr': True} ,
                     {'mode'='shadow', 'diameter': 12.0},
                     {'antenna'='DV04'}])
```

- All three operations have been 'folded' into a single command
- flagdata_ can execute all of them in a single iteration through the data set

Drawbacks:

1. The user must decide what commands to fold and when
2. Different interaction when doing single commands to script

Folding multiple operations?

But, maybe there is also a benefit of combining application of calibration and flagging?

```
# Python/Something like CASA
vis="mydata.ms"
gencommand_(vis=vis, [ {'op': 'flagdata', 'autocorr': True} ,
                       {'op': 'flagdata', 'mode':'shadow', 'diameter': 12.0},
                       {'op': 'flagdata', 'antenna':'DV04'},
                       {'op': 'applycal', 'caltable': ['myvis.bpass',
                                                        'myvis.W'] }
                       ])
```

It is clear where this is going:

```
# Python/Something like CASA
gencommand_('myscript.py')
```

Back to square one!
⇒ The ‘script’ must be in a non-procedural language

Proposal

Operations automatically re-ordered and folded to optimise performance:

```
# Python/CASA
vis="mydata.ms"
flagdata(vis=vis, autocorr=True)
flagdata(vis=vis, mode='shadow', diameter=12.0)
flagdata(vis=vis, antenna='DV04')
```

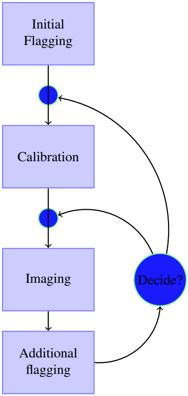
⇒ Automatic translation ('re-writing') ⇒

```
# Python/Something like CASA/User does not see this
vis="mydata.ms"
flagdata_(vis=vis, [ {'autocorr': True} ,
                     {'mode'='shadow', 'diameter': 12.0},
                     {'antenna'='DV04'}])
```

⇒ Execution!

Outline

Restart after additional calibration



Where should be reduction continue after additional flagging:

- Before calibrations if they affected by the new flags
- After calibrations if the new flags only affect the science target
- In each case only the SPWs, fields, etc that can be affected should be redone

Restart – the old fashioned way

```
#Python/CASA
if 1:
   #Initial flag
   flagdata()
     ....
if 1:
   #Calibration
   caltable=root+'.W'
   gaincal(...)
if 1:
   applycal(..., caltable)
   ...
if 1:
   #Imaging
   clean(....)
   ....
```

```
#Python/CASA
#Initial flag
flagdata()
#     ....
#Calibration
#gaincal(...)
#applycal(...)
#    ...
#
# Imaging
clean(....)
#clean(....)
#
```

1. Error prone!
2. The script looks different from the interactive commands

⇒ The computer should decide which steps need to be done and which not!

Proposal

- The user always runs the entire script
- Only the operations which could have different outcome are executed by the computer

Advantages

- The script is always in final version
- No possibility of mistake due to incorrect restart
- Save time by avoiding unnecessary full restarts

Outline

Introduction

Examples and Derived Proposals

- Operation Folding
- Dependency tracking
- Intermediate product tracking
- Data reduction branches

Theory

Implementing a better interface

Summary

Intermediate data product tracking

Integration vs. scan based gain calibration tables:

```
#Python/CASA
gaincal(vis = split1, field = '0', gaintable = root+'bandpass.bcal'
        refant = 'DV02', caltable = root+'intphase.gcal',
        calmode = 'p', solint = 'int' ,
        minsnr=2.0,  minblperant=4)
gaincal(vis = split1, field = '0', gaintable = root+'bandpass.bcal'
        refant = 'DV02', caltable = root+'infphase.gcal',
        calmode = 'p', solint = 'inf' ,
        minsnr=2.0,  minblperant=4)
```

...

```
#Python/CASA
gaincal(vis = split1,
        field = '0',
        gaintable = root+'bandpass.bcal'
        refant = 'DV02',
        caltable = root+'inf-phase-dv02-field0-blperant4.gcal',
        calmode = 'p',
        solint = 'inf' ,
        minsnr=2.0,
        minblperant=4)
```

⇒ eventually the *name* of the table encodes all the parameters!

Proposal

1. The computer should be keeping track of intermediate data products, calibration tables, plots, images etc
2. We should access them by primarily calling the commands that created them!
3. Closely related to dependency tracking, data reduction branches

Outline

Introduction

Examples and Derived Proposals

- Operation Folding
- Dependency tracking
- Intermediate product tracking
- Data reduction branches

Theory

Implementing a better interface

Summary

Data Reduction Branches – script

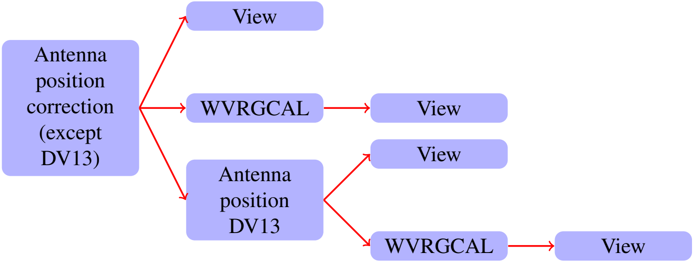
```
default(gencal)
vis = myvis.ms'
caltable␣=␣'antpos_fix'
caltype␣=␣'antpos'
antenna␣=␣'DV02,DV04,DV05,DV06,DV07,DV08,DV10,DV12,DV13,PM01,PM03'
parameter␣=␣[
␣␣␣␣␣0.000228,␣-0.000334,␣-0.000013,
␣␣␣␣␣0.000163,␣0.000239,␣0.000025,
␣␣␣␣␣0.000060,␣-0.000092,␣0.000384,
␣␣␣␣␣0.000053,␣0.000158,␣0.000001,
␣␣␣␣␣0.000103,␣0.000328,␣0.000351,
␣␣␣␣-0.000039,␣-0.000085,␣-0.000041,
␣␣␣␣-0.000331,␣-0.000056,␣0.000246,
␣␣␣␣␣0.000133,␣-0.000210,␣-0.000160,
␣␣␣␣-0.000045,␣0.000104,␣0.000109,␣#␣Not␣sure␣about␣this␣one!␣(BN)
␣␣␣␣␣0.000191,␣0.000010,␣0.000119,
␣␣␣␣␣0.000159,␣0.000005,␣-0.000054
]
gencal()

os.system('../WVRGCAL/bin/wvrgcal --ms myvis.ms \
          --output myvis.W --toffset -1 ')

default(applycal)
vis␣=␣'uid___A002_X219601_X4cd.ms'
#gaintable␣=␣'antpos_fix'
gaintable␣=␣['antpos_fix',␣'uid___A002_X219601_X4cd.W']
applycal()
```

Data reduction branches – notes

Note:

1. The user obviously wants to try with/without the WVR calibration – uses the commenting out technique
2. Also want to try with/without correction for antenna DV13
3. Note also the difficulty of attaching antenna names to position correction values

Data reduction branches – graph



- Recording just the data reduction path that happened to work in one case is not enough
- Decisions need to be made by scientists
- But, all inputs for their decisions prepared automatically

Proposal

1. The user should be able to specify multiple branches of data reduction where different parameters/procedures/options are invoked
2. The computer should keep track of the results of computation and present them to the user
3. The reductions would ideally be parallelised

Outline

Introduction

Examples and Derived Proposals
- Operation Folding
- Dependency tracking
- Intermediate product tracking
- Data reduction branches

Theory

Implementing a better interface

Summary

In advance: Sorry!

Preamble

Python/CASA is a huge improvement on many previous environments. In some cases, only now, that many CASA/Python scripts are available to be analysed can we identify problems.

Two approaches:

1. Look for examples in existing scripts of what can be done better and implement that
2. Look for patterns in the examples and derive more general solutions.

What pattern links the above examples?

Procedural model

Also called the 'von Neumann' model:

1. Key feature are pervasive *states*
2. Program consist of creation and sequential execution of blocks of manipulations of *states* (i.e., procedures)
3. The major part of most compilers today is undoing the von Neumann model behind the scenes
4. Originally states $\equiv$ program variables
5. Our state? The measurement set
   - A very large state
   - Very expensive to manipulate

   Sequentially, blindly executing unoptimised blocks of operations on measurement set is very inefficient

Backus on the Procedural model

Conventional programming languages are growing ever more enormous, but not stronger. Inherent defects at the most basic level cause them to be both fat and weak:
[...]
their inability to effectively use powerful combining forms for **building new programs from existing ones**, *and their lack of useful mathematical properties for reasoning about programs*

John Backus, ACM Turing Award Lecture, **1977**

Solution? (In theory only?)

Applicative language + program transformation

Solution? (In theory only?)

Solution? (In theory only?)

(In fact we can get most of the way there quite easily, see later)

Outline

Implementation choices

- Keep the existing interface / improve the interpreter
  - Hard!
  - Would not satisfy requirements
  - Fragile
- A Python wrapper for the existing interface
  - This talk
- More radical departure – use another language
  - Maybe in the future...

Python wrapper

Advantages

- Keep the existing language foundation
- Incremental development, immediate results
- Easy distribution to users
- Integration with other analysis done in Python
- Feasible!

Disadvantages

- Need to reduce the range of possible programming constructs
- Slightly awkward syntax

How does it work

- *Applicative* on measurement sets – no side effects, each command *transforms* the data
- *Lazy* – results only computed when requested by the user, not as encountered
- No flow control(!)
  Decisions made by scientists
  (Flow control based on original data easily implementable)
- Optimisation/rewriting stage just before execution

Prototype Example

```
def mydata():
    return Vis("uid___A002_X219601_X4cd.ms")

def mostpos(d):
    "Correct for antenna positions that we are sure about"
    d=Antpos(d, "DV02", [0.000228, -0.000334, -0.000013])
    d=Antpos(d, "DV04", [0.000163, 0.000239, 0.000025,])
    d=Antpos(d, "DV07", [0.000103, 0.000328, 0.000351])
    d=Antpos(d, "DV10", [-0.000331, -0.000056, 0.000246])
    d=Antpos(d, "DV12", [0.000133, -0.000210, -0.000160])
    d=Antpos(d, "DV13", [-0.000045, 0.000104, 0.000109])
    d=Antpos(d, "PM01", [0.000191, 0.000010, 0.000119])
    d=Antpos(d, "PM03", [0.000159, 0.000005, -0.000054])
    return d

def maybepos(d):
    "Not sure about this one! Will want to tru with and without"
    d=Antpos(d, "DV08", [-0.000039, -0.000085, -0.000041])
    return d


def antcheckd()
    return Select(mydata(), spw=1)

Plot(VisRaster(mostpos(antcheckd())),
     dims=["time", "phase"])

Plot(VisRaster(maybepos(mostpos(antcheckd()))),
     pdims=["time", "phase"])

go_reduce()
```

Version 0 goals

1. Applicative sub-language of Python
2. Commands for basic calibration, flagging, continuum imaging only

Features

1. Restart/Dependency tracking
2. DR Branching
3. Folding
   - 3.1 Antenna flagging
   - 3.2 BL correction
4. Simple intermediate product cache

Future goals (relatively easy reach)

- Automatic parellelisation multi-thread/SMP/cluster
  - Can parallelise on outermost scale, high efficiency
- Summary reports of all data reduction steps
- Automatic management of cache of intermediate data products

Outline

Summary

1. Data reduction interfaces are important for ALMA today
2. We can make them faster, more efficient, more reliable
3. Making them so is feasible, not a large scale project
4. In my opinion: This is the route to solving the data reduction problem