The MeqTrees Software System And Its Use For Third-Generation Calibration Of Radio Interferometers — The "Samizdat" Version

J.E. Noordam & O.M. Smirnov

Abstract

The formulation of the radio interferometer measurement equation (RIME) for a generic radio telescope by Hamaker et al. has provided us with an elegant mathematical apparatus for better understanding, simulation and calibration of existing and future instruments. The calibration of the new radio telescopes (LOFAR, SKA) would be unthinkable without the RIME formalism, and new software to exploit it.

The MeqTrees software system is designed to implement numerical models, and to solve for arbitrary subsets of their parameters. It may be applied to many problems, but was originally geared towards implementing Measurement Equations in radio astronomy for the purposes of simulation and calibration. The technical goal of MeqTrees is to provide a tool for rapid implementation of such models, while offering performance comparable to hand-written code. We are also pursuing the wider goal of increasing the rate of evolution of radio astronomical software, by offering a tool that facilitates rapid experimentation, and exchange of ideas (and scripts).

MeqTrees is implemented as a Python-based front-end called the meqbrowser, and an efficient (C++-based) computational back-end called the meqserver. Numerical models are defined on the front-end via a Python-based Tree Definition Language (TDL), then rapidly executed on the back-end. The use of TDL facilitates an extremely short turn-around time (hours rather than weeks or months) for experimentation with new ideas. This is also helped by unprecedented visualization capabilities for all final and intermediate results. A flexible data model and a number of important optimizations in the back-end ensures that the numerical performance is comparable to that of hand-written code.

MeqTrees is already widely used as the simulation tool for new instruments (LOFAR, SKA) and technologies (focal plane arrays). It has demonstrated that it can achieve a noise-limited dynamic range in excess of a million, on WSRT data. It is the only package that is specifically designed to handle what we propose to call *third-generation* calibration (3GC), which is needed for the new generation of giant radio telescopes, but can also improve the calibration of existing instruments.

1 Introduction

The MeqTrees software system has been designed to implement an arbitrary Measurement Equation (i.e. a numerical model of an instrument and/or process), and to solve for arbitrary subsets of its parameters. In this paper we will concentrate on the simulation and calibration of data taken with radio telescopes. After all, that is the subject for which MeqTrees was developed originally, and for which it is most urgent. It is also an excellent subject for demonstrating the special capabilities of MeqTrees.

Until recently, radio interferometers like WSRT, VLA, ATCA, GMRT were designed so that they could be approximated by a relatively simple instrumental model. Their *stations*[1] were carefully designed so that the shape of their spatial response beams could be assumed to be 'identical' at all times. In addition, the instrumental error associated with a station can be represented in this model by only two complex gain factors (one for each polarization), which may vary in time and frequency. Because their stations are parabolic dishes that are pointed towards the observed field, instrumental polarization effects can be treated (or not) as small 'leakage' terms.

Before 1980, first-generation calibration (1GC) was based on 'open-loop' methods, making separate calibrator observations before and after, and relying on instrumental stability in between. The result was a dynamic range[2] of about 100:1. However modest, this was enough for a plethora of important discoveries.

Around 1980, the discovery of *self-calibration* (Cornwell & Wilkinson 1981, see also summary by Ekers 1983) ushered in the era of second-generation calibration (2GC). Selfcal is a "closed-loop" method which continuously estimates the complex station gain factors with the help of one or more bright sources in the field of view. In this process, the utilized Sky Model is improved also. Selfcal has been spectacularly successful, and has led to a blossoming of techniques, software packages, and beautiful results. It allows astronomers to achieve dynamic ranges in excess of $10^4 - 10^5$ as a matter of routine, depending on how well the instrument approximates its simplified model. Record dynamic ranges of well over 10^6 have been achieved at the WSRT by de Bruyn (2006) (see also Sect. 2).

Radio astronomy is currently going through a remarkable worldwide burst of building new telescopes and upgrading existing ones[3]. These instruments present a new, two-pronged calibration challenge. On the one hand, they are much more sensitive, so more subtle instrumental effects will have to be taken into account to reach the thermal noise. On the other hand, the use of new technology like phased arrays complicates the instrumental model. Therefore, what we propose to call *third-generation calibration* (3GC) will require a more complex and able Measurement Equation, and a generalized form of selfcal.

In this context, we are fortunate that the explicit Measurement Equation of a generic radio telescope (RIME) was formulated 'in the nick of time' by Hamaker et al. (1996), after almost 50 years of radio astronomy. Further work by Hamaker (2000) led to a fully 2×2 matrix formulation of the RIME, which provides the mathematical underpinnings for 3GC. Without this full-polarization formalism, calibration of the new telescopes would be unthinkable. However, although the RIME is widely recognized as being correct, complete and universal, its actual adoption has been slow. This is caused to a large extent by the sustained success of the existing 2GC data reduction packages (AIPS, NEWSTAR, MIRIAD, DIFMAP), and the resulting disappearance of the function of user-developer. The ensuing low rate of evolution of calibration techniques could be a risk factor for the new telescopes.

One of the most important challenges of 3GC is dealing with Direction-Dependent Effects (DDE), i.e. instrumental effects that can no longer be assumed to be constant over the field of view. The most important DDEs are typically caused by the ionosphere (mostly phase and Farady rotation), and by station beamshapes that differ substantially from each other, and/or vary individually in frequency and time. Tackling DDEs implies that one has to solve for a much larger number of RIME parameters than before. Besides the practical problem of extra processing (which may well turn out to be a major bottleneck, but will not be discussed here), this raises some more fundamental issues about whether there is enough information available for 3GC. And, last but not least, methods are needed to correct for DDEs once they are known, which is non-trivial. All this is discussed in Sect. 9.

At this moment, it is not yet clear how the new telescopes will be calibrated exactly. The development of 3GC will probably take several years of experimentation, by many talented people. The MeqTrees software system is a tool that can play a role in that process at several levels. First of all, it is firmly based on the explicit RIME, which should be the new Common Language of radio astronomy. Secondly, it has many built-in features for *generalized selfcal*, like allowing for arbitrary RIMEs with

[1]Throughout this paper, we will use the generic term *station* for an element of an interferometer array. A station can be a parabolic dish or an aperture array, or something more exotic like a parabolic cylinder. Each station has two output signals, one for each polarization.

[2]Dynamic range is the ratio between the flux of the brightest source in the field, and either the thermal noise or the calibration artefacts, whichever is higher.

[3]An important incentive is the preparation for the building of the multi-billion Euro Square Kilometer Array (SKA) later in this decade.

arbitrary parameterizations, and solving for arbitrary subsets of RIME parameters, including source parameters. Thus, the art of modelling (in Python) is separated from the complex and efficient numerical machinery (in C++) 'under the hood'. Thirdly, MeqTrees is highly modular, so many different people can contribute to it, building on each other's results without getting in each other's way. Fourthly, new ideas can be implemented (and shared!) very quickly, often within a few hours. Very importantly, it is then directly available to the astronomical user. Rapid progress is also greatly helped by the many possibilities for visualization of intermediate results, enabling one to see what is actually going on. And finally, MeqTrees is fun.

In the next section, we present some recent results that give a taste of what MeqTrees can do. This is followed by the heart of this paper, a detailed description of how MeqTrees works in Sects 3-7. Sect. 8 gives a brief description of the RIME, and explains why it is such a powerful formalism. Sect. 9 discusses the modelling, application and estimation of DDEs, and the MeqTrees features that support all this. Sect. 10 sketches how MeqTrees could be the basis of a worldwide collaborative network to address the challenges of 3GC together, and thus help to increase the rate of evolution of radio astronomical data reduction software. After all, the new telescopes are already being built, while neither the software nor the users are ready for them.

2 Some encouraging results

Before describing MeqTrees in detail, we show some rather impressive results, as a kind of appetizer for what it can do. The image in Fig. 1 has been reduced by Smirnov (2010), and has a virtually noise-limited dynamic range of 1.6 million. It is an improvement over the result obtained with NEWSTAR by de Bruyn (2006) with the same data.

The WSRT has been 'world champion' dynamic range for the last few decades because of its favourable characteristics. It has equatorial mounts (so the station beams do not rotate on the sky), excellent pointing, almost identical station beams, on-axis receivers (which minimizes instrumental polarization), and very small closure errors (so the selfcal assumption of antenna-based errors is not violated very much). Thus, the WSRT closely approxi-

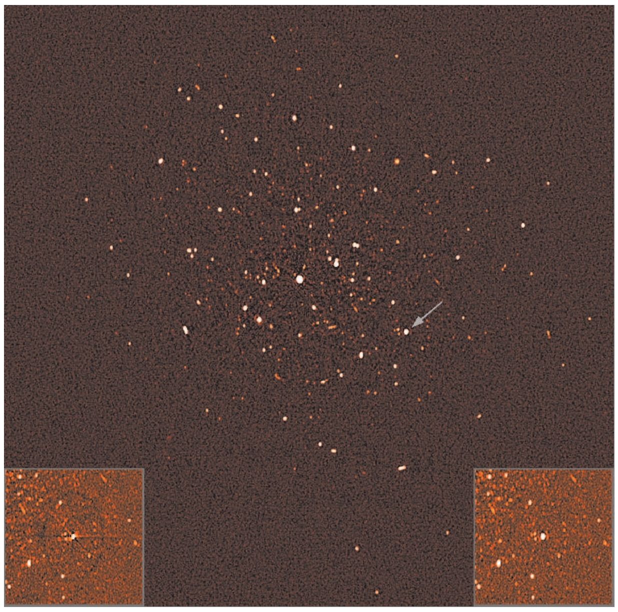


Figure 1: This WSRT 21cm image of the field around the bright radio source 3C147 is virtually noise-limited, and has a dynamic range of 1.6 million. The inset shows a close-up of one off-axis source with and without a differential gain solution.

mates the 2GC instrumental model, which explains why NEWSTAR does so well with it. MeqTrees does better because it can solve for time-varying DDEs (see Fig. 2), which manifest themselves as subtle residual structures in the image around the brighter off-axis sources (see inset of Fig. 1).

Any other telescope (as well as some observational modes of the WSRT) will need 3GC to achieve its full performance. *We have not even mentioned polarization yet.* We will soon repeat the exercise with an observation with deliberate pointing errors. After that, we propose to try our hand[4] with more challengeing telescopes, for instance high-dynamic-range wide-field full-polarization imaging with the (E)VLA. In the meantime, Yatawatta et al. (2010a) has used MeqTrees to make all-sky images with the first LOFAR stations.

Figure 2 shows the differential phase solutions for 5 of the 14 WSRT telescopes as a function of time in the direction of six moderately bright sources in the field. The values are relative to the phase solution in the direction of the dominating 22 Jy source 3C147. With an integration time of 30 minutes, the S/N of the differential phases is clearly large enough to detect slowly varying real effects. The next question is how to interpret these. As outlined in Sect. 9, MeqTrees makes it easy to test various conjectures like pointing errors or differential refraction, by fitting parametrized models to the differential gains. In this case, visual inspection of the plots strongly suggests a linear phase gradient over the 3 km WSRT array, and this was quickly verified with MeqTrees. Possible underlying causes will be dealt with in a follow-up paper.

In addition to beating the world champion, this result also addresses one of the urgent (and fundamental) problems of 3GC, namely the availability of calibration beacons. A very important conclusion that can be drawn from Fig. 2 is that sources as faint as a few mJy are bright enough to allow solving for differential gains, as long as the latter vary smoothly in time (and frequency). *This suggests that there will probably be enough calibration beacons in a typical field to solve for DDE parameters*. This should be verified for other telescopes than the WSRT, including the various SKA concepts.

[4]Or rather, we hope that more directly interested parties will be sufficiently motivated to try their hand, using the available scripts, and their own Jones matrices...

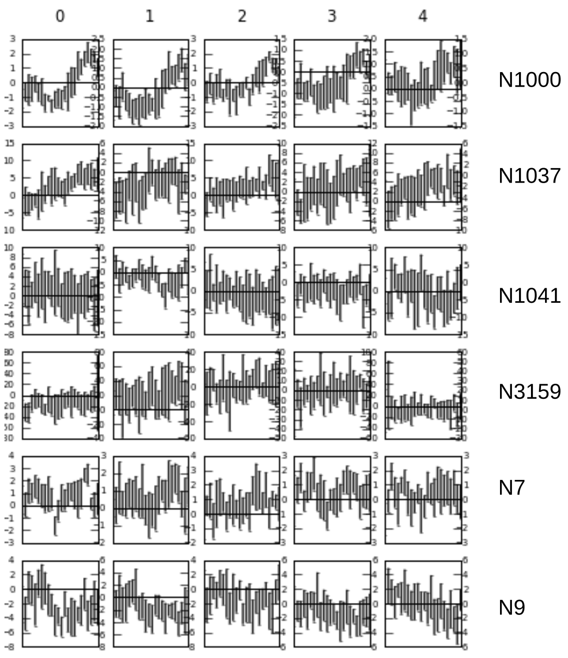


Figure 2: Differential phase solutions in the direction of 6 different sources (per 5 antennas), as a function of time. The integration time is 30 minutes. Note that the S/N is large enough to show slow variations, which suggests that sources as faint as a few mJy may be used to solve for DDEs.

3 Nodes, trees, forests

According to Donald Knuth, “trees have been in existence since the third day of creation, and perhaps earlier.” The use of *trees* as information structures is ubiquitous throughout computing science; Knuth (1973) provides a good introduction. MeqTrees uses trees to represent mathematical expressions. This particular idea dates back to very early work on compilers (Hopper 1952), and in fact was the first application of trees to computer science.

A tree is a type of graph whose *nodes* are connected in a parent-child hierarchy. A tree node can be parent to zero or more child nodes, and can itself be a child of zero or more other nodes.[5] Cycles are not allowed. A node having no children is traditionally called a *leaf*, a node with no parents is called a *root*. A *forest* contains multiple trees, which may be interlinked.

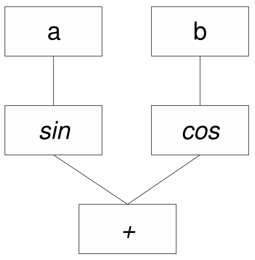


Figure 3: An expression tree.

A traditional expression tree corresponding to the expression $\sin a + \cos b$ is shown in Fig. 3. This illustrates the main concepts of a tree: a and b are *leaf* nodes (having no children), representing atomic components of the expression such as constants or variables; “sin” and “cos” are unary function nodes (performing some mathematical operation on their children), and “+” is a binary function node. To compute the value of the expression, we start at the leaves and propagate the values through their parents, performing the appropriate mathematical operations along the way, until we get to the root node (the “+” node, in this case), the result of which is the value of the expression.

In this traditional view, the result of the computation at any one node is a single value, the value of some expression. MeqTrees goes a step further by making the results *functions*. A typical MeqTree implements a real- or complex-valued function of N real variables, $f(x^{(1)}, ..., x^{(n)})$. The most common variables – at least in radioastronomy – are time t and frequency ν. For simplicity, we’ll just use t and ν in all further examples, with the understanding that everything we describe can be generalized to N-dimensional variable space. MeqTrees calls time and frequency *axes of variability*, or simply *axes*.[6] In the example above, if we imagine that a is a function of time, $a = a(t)$, and b is a function of frequency, $b = b(\nu)$, then the result of the tree becomes a function of time and frequency, $f(t, \nu) = \sin a(t) + \cos b(\nu)$.

MeqTrees represents functions as samples on a grid. To do this, we pick a *domain* in t, ν, and define a gridding over that domain – essentially, two vectors $(t_1, ..., t_n)$ and $(\nu_1, ..., \nu_m)$. The function f can then be represented by a two-dimensional array of samples, $\{f_{ij} = f(t_i, \nu_j)\}$. The result of the root node above – the function $f(t, \nu)$ – is then the array $\{f_{ij}\}$.

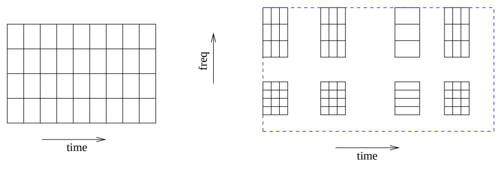


Figure 4: A node computes the value of some function on a grid (which we call the *cells*) within some domain. The domain on the left is two-dimensional (t, ν) and has identical and contiguous cells on a regular grid. It is also possible to have irregular grids of non-contiguous cells with different sizes, as illustrated on the right (cell *size* is only relevant for some rather obscure operations.) Higher-dimensioned domains are also possible.

This then, in a nutshell, is the way MeqTrees work. A a *request* object is created that contains (among other things) the two vectors, $(t_1, ..., t_n)$ and $(\nu_1, ..., \nu_m)$. These are called *cells* in MeqTree parlance, since they represent the rectangular cells of a two-dimensional grid. Note that

[5]By the strict conventional definition of a tree graph, a node may have at most one parent. MeqTrees allows for multiple parents; the proper term for such an structure is *directed acyclic graph*. We use *tree* for brevity.

[6]The current implementation allows up to 16 arbitrarily-named axes, though this number may easily be increased as necessary.

the grid stepping does not need to be regular – see Fig. 4 for an example. The request object is a request to compute a function (whatever function happens to be defined by this tree) on a certain grid. This request is passed to a node of the tree (usually the root node). To compute the function, a typical node will pass the request on to its children, and then perform some mathematical operation on the returned results.

This also describes the interface to a node – a node is given a request (representing a gridding), and returns a result (representing a function sampled on that grid). Indeed, since the interface to any tree is via its root node, operationally a tree is indistinguishable from the root node. When parent nodes deal with child nodes, they have no knowledge (nor do they need any) of whether those child nodes are individual leaf nodes, or have whole subtrees hiding behind them.

The result of a parent node is (usually) determined by performing some mathematical operation on the results of its children. The exact operation is determined by a node's *class*. Let's say a parent of class H implements the binary operation (or function) $H(a, b)$. If the results of its two child nodes (or, equivalently, the subtrees rooted at their nodes) correspond to the functions $f(t, \nu)$ and $g(t, \nu)$, then the result of the parent corresponds to $h(t, \nu) = H(f(t, \nu), g(t\nu))$.

A leaf node has no children, and can compute its result (i.e. the function f that it implements) in a self-contained way. This is also determined by its class, for example:

MeqConstant nodes return a constant value, $f(t, \nu) \equiv c$.

MeqFreq nodes return the frequency, $f(t, \nu) = \nu$.

MeqParm nodes compute, e.g., a polynomial $f(t, \nu) = c_{00} + c_{10}\nu + c_{01}t$, where c_{ij} are read from an external parameter table (these are typically solvable parameters, as will be described below.)

MeqFITSImage nodes return, e.g., an image of the sky read from a FITS file. In mathematical terms, an image is a sampled brightness distribution, $I(\nu, l, m)$. The sky coordinates l, m are another example of axes of variability.

MeqSpigot nodes interface to a Measurement Set (MS) used to store observational data in radioastronomy, and return visibilities sampled by a particular interferometer, $V(t, \nu)$.

To sumarize, a typical tree computes a function defined on a grid in N-dimensional variable space. A forest of trees computes a collection of functions, which together constitute a numerical model. The model may contain solvable parameters which may be optimized in various interesting ways (Sect. 6).

3.1 Specifying trees in TDL

Fig 5 gives a (very simple) example of how to specify a tree, using a script written in *TDL* called *myFirstTree.py*. Part of it is displayed in the middle section of the meqbrowser (see Sect. 3.3). TDL (Tree Definition Language, Smirnov 2008) is basically regular Python, plus some operator overloading that allows one to quickly specify nodes and their connections by means of class, name, any children, and other options. The TDL code shown in fig 5 demonstrates the basic syntax for specifying nodes. It contains a few required functions like *_define_forest()* and *_tdl_job()*, but otherwise it is just a Python script.

Rather than always building trees from individual nodes, it is often far more efficient to manipulate higher-level Python objects and frameworks which, while presenting a simplified interface, cause trees to be constructed behind the scene. This can hide a lot of unnecessary detail from the non-expert user, and accelerate development of complicated trees. Python as a languauge is very well-suited for developing hierarchical object-oriented frameworks. We have developed a number of such frameworks in the context of radioastronomy: a lower-level framework called "Meow" (Measurement Equation Object frameWork), and two higher-level frameworks for simulation ("Siamese") and calibration ("Calico"). More will surely follow.

3.2 Meqbrowser and meqserver

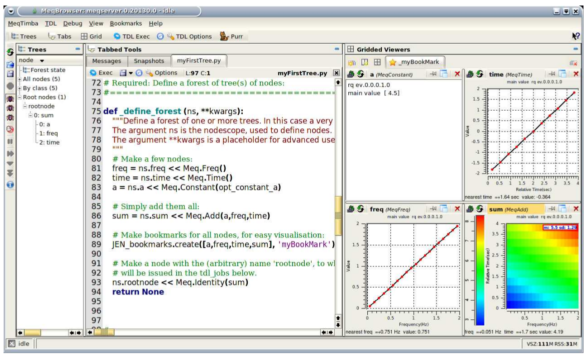
Being a semi-interpreted language, Python offers wonderful flexibility and ease of programming, but computing efficiency is not one of its stronger points. The actual computations in MeqTrees are performed by a fast, optimized back-end called the *meqserver*, which is written mostly in C++. A GUI front-end called the *meqbrowser* (written in Python) provides a rich interface to the computational back-end. Specifically, operations are divided as follows:

- The user loads a TDL script into the front-end (meqbrowser). This script – along with any associated option settings – specifies the exact structure of a tree (essentially, the structure of a computation).
- Meqbrowser executes the TDL code, which results in a string of instructions on how to assemble the tree together to be sent to the back-end (meqserver).
- Meqserver constructs its internal tree representation based on the supplied instructions.
- The user operates the GUI to specify external data (e.g. a Measurement Set to be calibrated, etc.); references to this data (pathnames, etc.) are also passed to the meqserver.
- The user operates the GUI to instruct the meqserver to start processing data.
- Meqbrowser monitors progress and (optionally, upon the user's instruction) fetches and visualizes intermediate results (see Sect. 3.4).
- It is also possible to bypass the GUI front-end, and operate the meqserver noninteractively (i.e. in batch mode.)

Such an architecture allows for great flexibility in specifying how computations are to be carried out (since all the specification is done in TDL/Python), yet avoids the computational inefficiency associated with scripting languages. A lot of thought has been put into making the computational engine as efficient as possible (see Sect. 7). And while in principle a MeqTrees-based computation cannot match the theoretical performance of hand-optimized compiled code, in real-life testing its performance has proven to be either equivalent, or (in the worst case) within a factor of 2–3 of hand-optimized implementations.

3.3 The meqbrowser GUI

MeqTrees has a graphical user interface called the *meqbrowser*. It has three main sections, and a choice of



```
# Required: Define a forest of tree(s) of nodes:
#=====================================

def _define_forest (ns, **kwargs):
   """Define a forest of one or more trees. In this case a very
   The argument ns is the nodescope, used to define nodes.
   The argument **kwargs is a placeholder for advanced use
   """
   # Make a few nodes:
   freq = ns.freq << Meq.Freq()
   time = ns.time << Meq.Time()
   a = ns.a << Meq.Constant(opt_constant_a)

   # Simply add them all:
   sum = ns.sum << Meq.Add(a,freq,time)

   # Make bookmarks for all nodes, for easy visualisation:
   JEN_bookmarks.create([a,freq,time,sum], 'myBookMark')

   # Make a node with the (arbitrary) name 'rootnode', to wl
   # will be issued in the tdl_jobs below.
   ns.rootnode << Meq.Identity(sum)
   return None
```

rq ev.0.0.0.1.0
main value [4.5]

nearest time =+1.64 sec value: -0.364

nearest freq =+0.751 Hz value: 0.751

freq =+0.051 Hz time =+1.7 sec value: 4.19

Figure 5: *The meqbrowser GUI.*

menus and other buttons (Fig. 5). In this example, the contents of the TDL script *myFirstTree.py* has been loaded by means of the TDL menu at the top, and is displayed in the middle section. The TDL code in the *_define_forest()* function defines the nodes that make up this very simple tree. The Python code may be edited directly in the browser, or with an external editor. After compilation (i.e. generation of nodes on the meqserver side), the tree is displayed in the left section, where it may be browsed by opening and closing branches. Clicking on a node displays its contents in various ways in a panel in the right section, which may then be inspected interactively. The tree is executed by issuing a request to a named node (here called 'rootnode'). The execution is done by means of a *_tdl_job()* function in *myFirstTree.py*, which may be called via the *TDL Exec* button. Upon execution, the various display panels on the right will come alive, showing the node results.

A TDL script may have compile-time options, whose values may be set interactively in a popup window under the *TDL Options*'button. The script may also have run-time options, which may be set in a popup window under the *TDL Exec* button. It also offers a choice of available "TDL jobs" to be executed. Clicking on one of them executes the forest, or performs some other function like invoking the AIPS++ imager. Currently selected values of TDL options are retained in a configuration file, which greatly increases the ease of using MeqTrees in practice. The configuration file also comes in handly when a TDL script needs to be executed in batch mode, without the meqbrowser.

Obviously, we have described only the basic functionality of the MeqBrowser, which has many powerful features to make it easy (and fun!) for the user to execute trees and to inspect node results. For instance, it has a bookmark button that offers shortcuts to the display of predefined groups of nodes. The Purr button ("Purr is Useful for Remembering Reductions") activates a rather useful scheme to save and describe all intermediate steps of a data reduction project. Debugging functionality (stop, step, resume etc) is available along the left edge. It also has a profiler, which measures the processing of all nodes, either individually or by class. Execution progress messages are displayed along the botton, and any errors may be inspected in detail.

3.4 Visualization

One of the cornerstones of MeqTrees is its emphasis on visualization at all levels. This is based on the conviction that the quickest way to develop and debug an algorithm is to be able to see what is going on. It cannot be stressed enough that all this visualization is *optional*, and does affect computational efficiency when it is not used. We expect that this capability will be very popular, and will soon become the norm. Therefore we envisage a steady increase in sophistication of both standard and application-specific visualization.

In addition to making visualization possible, the meqbrowser also makes it easy. Clicking on any node in the meqbrowser will display either the node status record (very instructive!), or a specific field, or a plot of the latest result in its cache. Various different types of plots or representations may be selected by right-clicking on a node. When displaying images, middle-clicking on the plot itself will produce a vertical and horizontal cross-section through that point. Optionally, flags may be indicated in the plots. As illustrated in fig 5, the plotter will adapt automatically to the dimensions of the Result: frequency, time, both, etc. If the domain has more than 2 dimensions, different cross-sections may be selected. There is also a multidimensional plotter.

Some nodes display their results in specific ways. The MeqSolver node produces plots that indicate the quality of the solution, and its evolution over successive iterations. The ultimate goal is some kind of visualization of the χ^2 surface. The MeqComposer node produces a so-called *inspector* plot, i.e. time-tracks of its multiple Results, side-by-side in the same panel. Such time-tracks are obtained by averaging along the non-time axes, usually just the frequency axis. The inspector is a cumulative plot in the sense that the time-tracks just grow in length with succesive Results. The MeqDataCollect node also displays the Results of multiple nodes in the same plot, but is refreshed each time. It offers two modes, spectra or real vs. imaginary, and is hierarchical: the results of various MeqDataCollect nodes may be combined in the same plot, e.g. with different colors or symbols.

The ease of use is greatly increased by a menu of meqbrowser *bookmarks*. Clicking on a bookmark conjures up the visualization of a specific node, or a *bookpage* of associated nodes. Bookmarks are defined by the tree designer, to highlight particular aspects of what the tree is doing. Selected bookpages remain close at hand by means of tabs (see Fig. 5). Nodes make their information available for display by *publishing* it, i.e. sending it off into the void, to be picked up by another program. A node may be induced to publish every time it gets a new result, which is the default for bookmarked nodes. This makes it easy (and fun) to watch intermediate results while the tree is executing a sequence of Requests.

Although the standard visualization of MeqTrees offers substantial functionality, we expect that many users will develop specialised visualisation nodes for their specific application areas. This will be encouraged and supported, for instance by offering nodes and other tools (like a result object) to make this easier. For the moment, the user-definable nodes PyNode and PrivateFunction should do.

4 Data Model

We have described how MeqTrees uses trees to represent mathematical expressions that comprise a numerical model. This is only half of the story; the other half is the actual computation, i.e. what kind of data can be fed into these expressions, and how efficiently can it be processed. The capabilities of MeqTrees are in large part determined (and limited) by this underlying data model. This section describes the data model in more details.

4.1 Grids and functions

As stated previously, the atomic unit of computation in MeqTrees is a *function* represented by a set of samples over an N-dimensional grid, e.g., a function of frequency and time $f(t, \nu)$. Internally, this is represented by a *cells* object specifying the grid – containing vectors of, e.g., times (t_i) and frequencies (ν_j) – and an N-dimensional array of samples, e.g. ($f_{ij} = f(t_i, \nu_j)$). For historical reasons the latter object is called a *vells*. A vells object is placed into a container called a *vellset* (the rationale for this will be explained in Sect. 6.) A cells and a vellset together then constitute a *result* object (Fig. 6). A result object is the unit of data that is passed between nodes.

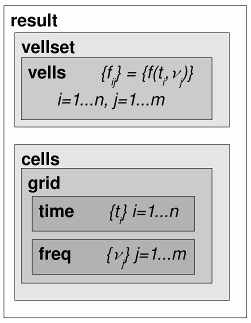


Figure 6: The layout of a basic result object.

If a function does not vary over a given axis, then the dimension of its vells along that axis can be equal to 1, i.e. it can be represented by fewer actual data points. For example, given a cells of M times and N frequencies (and assuming time is the first axis and frequency is the second axis in our grid – the ordering of axes is fixed prior to computation), a function can be represented by vells with dimensions of:

$M \times N$, for a function variable both in time and frequency;

$M \times 1$, (or, equivalently, just an M-vector) for a function variable only in time;

$1 \times N$, (this is **not** equivalent to an N-vector, since frequency is the **second** axis); for a function variable only in frequency;

1, for a constant value.

All mathematical operations transparently handle vells of different dimensions. For example, if the ”+” node in Fig. 3 receives an $M \times 1$ array from one child and a $M \times 1$ array from the other child, it will perform M additions and return a $M \times 1$ array (i.e. a function of time only). If, on the other hand, it receives an $M \times 1$ array (function of time) and a $1 \times N$ array (function of frequency), then it will perform $M \times N$ operations and the result will be an $M \times N$ array (function of time and frequency). These decisions are made directly in the tree at runtime, so the exact same tree can be used to compute an expression involving only constants, or an expression involving functions of time, frequency, etc. In the latter case the computation is automatically optimized in the sense that the tree keeps track of what values depend on what axes, and only executes the mimimum number of calculations.

This is actually one of the most powerful features of MeqTrees. It is often the case in numerical modeling that one starts with a simple model (i.e. constant parameters), and adds complexity (e.g. parameters with time dependence) later. With traditional code, adding a time dependency to a particular branch of a calculation requires that arrays be resized, for-loops added to iterate over time, bugs introduced and squashed again, etc. Depending on the complexity of a piece of code, this can become quite a daunting task, if not an insurmountable barrier to experimentation. With MeqTrees, you only need to change the property of a parameter up in the tree, and time dependence is then automatically propagated throughout all calculations.

4.2 Scalars and tensors

Scalar functions have a single real or complex value at each grid point. A tensor function of type (dimensionality) $n_1, n_2, ..., n_k$ has $n_1 \times ... \times n_k$ values for every grid point. For example, the 2×2 cohærency and Jones matrices used in the RIME are type-2,2 tensor functions. An efficient representation of tensor functions is therefore important for efficient implementations of the RIME.

MeqTrees represents tensors by a result object containing a list of $n_1 \times ... \times n_k$ vellsets, and a vector of integers $(n_1, ..., n_k)$ describing the dimensionality. Figure 7 shows a tensor result corresponding to a 2×2 matrix. Each matrix element is represented by a separate vellset (though all share a common cells object, thus a common grid definition.) This means, among other things, that each element of a matrix can have independent axes of variability. Operationally, it is quite common to see diagonal matrices where the diagonal elements are functions of frequency and/or time, while the off-diagonal elements are zero (and thus constant.) Such tensors can be most efficiently represented with this scheme.

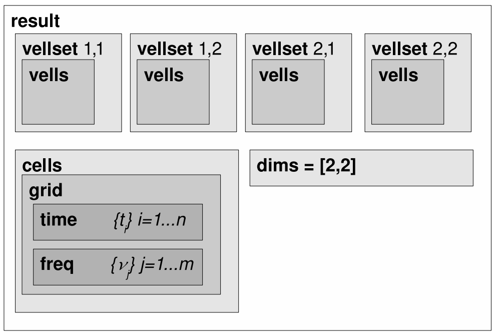


Figure 7: The layout of a tensor result object.

Most node classes can transparently accept tensor arguments. The normal convention is to perform the corresponding mathematical operation element-by-element. All arguments must then have the same tensor dimensions (or, as a special case, be scalar, in which case the scalar value is reused for all elements), and an error is reported otherwise.

MeqTrees also provides a few specialized tensor nodes. The MatrixMultiply node is used to multiply matrices (as well as vectors). The MatrixInvert22 node inverts 2×2 matrices.[7] The Composer can combine the results of multiple nodes into a single tensor, and the Selector extracts individual tensor elements.

5 Data flags

Radio astronomy has to operate in an environment in which there are many external and internal sources of Radio Frequency Interference (RFI). It is therefore important to be able to flag bad data values, and propagate these flags throughout all calculations, so that results derived from bad data are also properly flagged and ignored as necessary.

[7]General matrix inversion is not yet implemented, since 2×2 matrices are sufficient for RIME purposes. It could be added as needed.

MeqTrees can associate a *flag vells* with each vellset of a result. A flag vells is an array of integer flagwords that follows the same dimensionality rules as for normal vells arrays (see Sect. 4.1), i.e. given a cells of N time and M frequency points, a flag vells may have dimensions of 1, N, $1 \times M$, or $N \times M$. The latter case associates a separate flagword with every time/frequency point – a raised bit in the flagword w_{ij} indicates bad data at t_i, ν_j. The intermediate cases correspond to entire times or frequencies being flagged at once, while the first (and admittedly not very useful) case has a single flagword for the entire domain. The different vellsets of a tensor result may have different flag vells, or may share flags by reference (see Fig. 8).

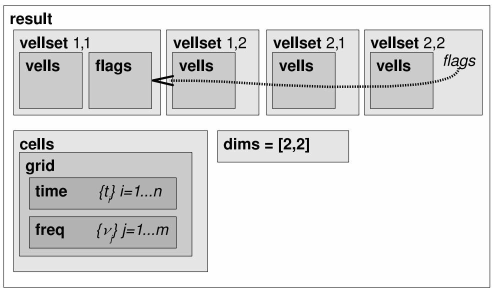


Figure 8: The layout of a result object with flags. Note that vellsets 1,1 and 2,2 share the same flags, while vellsets 1,2 and 2,1 have no flags at all.

Flag vells are automatically propagated through tree nodes in a mathematically sensible manner. For example, when an Add node receives flagged results from its children, the flag vells associated with its result is a bitwise-OR of all the child flags. This means that at the root of the tree, the final result will have flags for every t, ν point where the resulting value is derived from something that was flagged. Statistical and reduction operations (such as taking the mean over a certain set of axes) will also ignore flagged values when computing the statistics.

The flagword contains 32 individual bits, which allows for some very versatile flag management. Since flags can be built up via different procedures (automatic flagging algorithms, heuristics based on metadata describing system status during measurement, manually raised flags), it is exteremely useful to associate these different *flagsets* with different bit positions in the flagword. The user then has the option of activating or ignoring specific flagsets via a bitmask of currently active flags.

Flags can be preserved under storage, if the data format supports them. For radio astronomical data, a standard storage type is the Measurement Set (Kemball & Wieringa 1996), which defines standard columns for boolean flags. MeqTrees extends this by specifying an additional column of bitwise flags.

5.1 Flagging

Flags can also be generated directly inside a tree. MeqTrees offers a very simple but versatile scheme for this. The MeqZeroFlagger node sets flags in its child result based on a comparison to zero. The user then has to supply some ubtree that produces a result that can be used as a discriminator, e.g. in which "bad" data corresponds to values greater than zero. The MeqMergeFlags node is then used to merge the new flags with those of the original data.

Since the discriminator expression is supplied by an arbitrary subtree, any type of flagging can be implemented, from simple data clipping, to flagging based on the value of some completely unrelated expression defined over the same domain (such as the value of a solution, see Sect. 6).

6 Parameters and model fitting

Consider a mathematical model, i.e. some function $M(\nu, t)$ that depends on a number of parameters $(p_1, ..., p_k) = \vec{p}$. We denote this as $M(\nu, t; \vec{p})$. *Model fitting* is the process of finding a value of $\vec{p}$ that minimizes the difference (according to some predefined metric) between measured data $D_{ij} = D(\nu_i, t_j)$ and $M_{ij}(\vec{p}) = M(\nu_i, t_j; \vec{p})$. We also call this *solving* for $\vec{p}$. In radio astronomy, the model M is given by some parameterization of the RIME, and the process of model fitting is called *calibration*. This is explained in more detail in Sect. 8, here we first want to describe the general approach to model fitting employed in MeqTrees.

6.1 Solving in MeqTrees

The MeqTrees approach to solving is as follows (Fig. 9). A set of subtrees of arbitrary complexity implements the model M. The model is not necessarily a single function, but can be a whole set of functions, e.g. $\{M^{(pq)}\}$, giving the visibilities per baseline $p - q$ (see Sect. 8). These model trees are quite similar to the simulation trees discussed above, in Certain parts of the model are the unknown parameters, these are represented by MeqParm nodes. MeqParms are initialized with best guesses (or previous solutions read from disk, if available.)

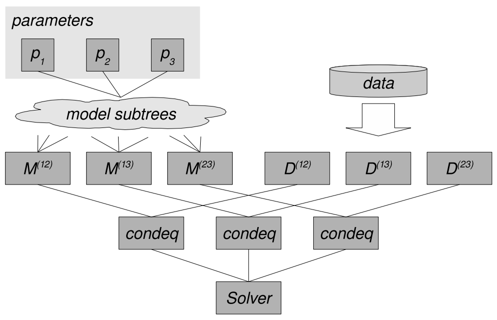


Figure 9: A schematic layout of a solver tree.

A parallel set of subtrees provides the data $D^{(pq)}$. These subtrees can be (and usually are) as simple as a single MeqSpigot node (which interfaces to a Measurement Set containing visibility data), but can also containpreprocessing steps as needed. The two sets of subtrees are linked up via MeqCondeq nodes, which are in turn all children of a MeqSolver.

At the beginning of a solution, the MeqSolver issues a special request that designates a subset of the MeqParms as solvable. Subsequently, trees that contain solvable MeqParms, when asked to compute a result, automatically augument it with partial derivatives w.r.t. solvable parameters $\partial M^{(pq)}/\partial p_k$. Note that this is completely transparent to the user – any tree that can compute a function can also compute the derivatives of that function (see below).

MeqCondeq nodes then form up the difference $\Delta^{(pq)} = M^{(pq)} - D^{(pq)}$, and also the partial derivatives w.r.t. each solvable parameter, $\partial\Delta^{(pq)}/\partial p_k$, and return these to the solver. The solver uses some algorithm to determine a set of incremental parameter updates Δp, and sends these updates back up the tree to the MeqParms, at which point the procedure is repeated until convergence has been achieved, or a maximum number of iterations has been reached.

6.2 Estimating the derivatives

MeqTrees currently estimates first derivatives via finite differencing. If a subtree implementing the function $f(\nu, t)$ depends on the solvable parameters p_1 and p_2, then the vellset in its result (Fig. 10) will actually contain three vells: the "main" value $f(\nu, t; p_1, p_2)$, plus two "perturbed" values $f(\nu, t; p_1 + \delta_1, p_2)$, $f(\nu, t; p_1, p_2 + \delta_2)$.

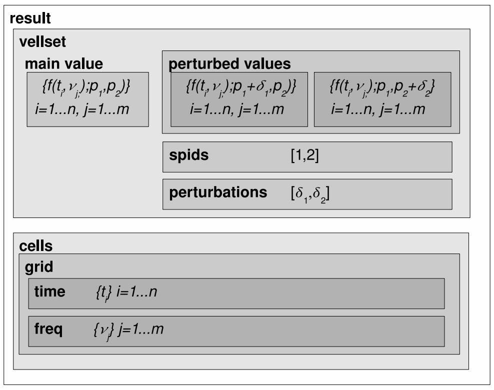


Figure 10: The layout of a result object with perturbed values.

If another subtree for $g(\nu, t)$ depends on solvable parameters p_2 and p_3, then its result will likewise contain three values: $g(\nu, t; p_2, p_3)$, $g(\nu, t; p_2 + \delta_2, p_3)$, $g(\nu, t; p_2, p_3 + \delta_3)$. Note that each vellset also contains a vector of *spids* (solvable parameter identifiers), which indicate what solvable a particular perturbed value is associated with.

Now consider how this information propagates down the tree. If f and g are the child nodes of a MeqAdd node returning $h(\nu, t) = f(\nu, t) + g(\nu, t)$, then the addition must be executed four times:

$$\begin{aligned}
h(\nu, t; p_1, p_2, p_3) &= f(\nu, t; p_1, p_2) + g(\nu, t; p_2, p_3) \\
h(\nu, t; p_1 + \delta_1, p_2, p_3) &= f(\nu, t; p_1 + \delta_1, p_2) + g(\nu, t; p_2, p_3) \\
h(\nu, t; p_1, p_2 + \delta_2, p_3) &= f(\nu, t; p_1, p_2 + \delta_2) + g(\nu, t; p_2 + \delta_2, p_3) \\
h(\nu, t; p_1, p_2, p_3 + \delta_3) &= f(\nu, t; p_1, p_2) + g(\nu, t; p_2, p_3 + \delta_3)
\end{aligned}$$

Note that the left-hand side contains the "main" value of h plus three perturbed values of h w.r.t. p_1, p_2, p_3, while the right-hand side contains a mix of main and perturbed values of f and g, combined according to a simple looping algorithm. This kind of loop over perturbed values is automatically executed inside every MeqTrees node, thus ensuring that the root of the tree computes perturbed values for every solvable parameter in the tree.

Given a main and a perturbed value, the actual derivative is estimated as:

$$\frac{\partial f}{\partial p_k}(\nu, t; p_k) \approx \frac{f(\nu, t; p_k + \delta_k) - f(\nu, t; p_k)}{\delta_k}$$

Note that we keep writing ν, t here to emphasize the fact that both f and its derivatives are potentially functions of many variables such as frequency and time. That is, a separate value and a separate derivative are computed at every time-frequency point.

6.3 Least-squares solver

A (weighted) least-squares solver minimizes the difference $D - M$ in a least-squares sense, i.e. finds a $\vec{p}$ that minimizes the chi-squared sum:

$$\chi^2(\vec{p}) = \sum_{ij} w_{ij}^2 (D_{ij} - M_{ij}(\vec{p}))^2 \qquad (1)$$

where w_{ij} are (optional) weights. Different methods of minimizing χ^2 are known; the suitability of a particular method depends on properties of M such as degree of linearity w.r.t. $\vec{p}$, etc. Since MeqTrees can implement models of arbitrary complexity, our initial designs have tended towards a "policy-free" solving scheme that works adequately in most cases but is not necessarily the optimal one for each particular case. We have therefore decided to use the AIPS++/CASA solver (Brouw 1996), which implements the Levenberg-Marquard algorithm (Madsen et al. 2004). The LM algorithm is a type of gradient descent method which is particularly well-suited to nonlinear problems.

An important problem with any solver is the handling of ill-conditioned problems (i.e. when there's not enough information to solve for all unknowns.) The AIPS++/CASA solver works by accumulating normal equations (received from the MeqCondeqs), then inverting the solution matrix. The inversion is done by SVD, which detects ill-conditioning and handles it by effectively reducing the number of unknowns (this is called reducing the rank of the solution.) A number of solver diagnostics, including rank and condition numbers, is automatically generated by the MeqSolver node, and may be visualized by the user to get an indication of the quality of the fit.

6.4 Alternative approaches

Other kinds of solvers are being actively considered for inclusion in MeqTrees. These may require different ways of calculating the derivatives:

- Analytic derivatives are known to produce more stable solutions. MeqTree nodes can be adapted to compute and propagate analytic derivatives via the chain rule (and fall back to finite differencing should a node be encountered that cannot compute analytic derivatives).

- Second derivatives may allow for better solvers. Second derivatives may be computed via double-differencing, or analytically.

- Bayesian solvers, rather than using derivatives, sample the function over a “cloud” in parameter space. That is, they generate a random set of vector offsets $\vec{\delta}_1, ..., \vec{\delta}_n$, and compute the perturbed model at each offset $M(\vec{p} + \vec{\delta}_k)$. Note that our current scheme of computing perturbed values w.r.t. each solvable parameter can be considered a special case of this, each vector offset $\vec{\delta}_k$ being a simple shift along axis k in parameter space, orthogonal to all the other offsets.

In principle the MeqTree code and internal data structures can be easily adapted to any of the approaches listed here.

6.5 MeqParms

A MeqParm node represents solvable parameters of the model. One of the most powerful features of MeqTrees is that each parameter can be a function of ν, t (and, naturally, any other dimensions.) The current implementation provides a polynomial, so the actual solvables are coefficients of the polynomial. The degree of the polynomial may be specified separately for each MeqParm, and for each solution. Other smooth functions of ν, t may of course be obtained by combining polynomial MeqParms into subtrees.

MeqParms can store their solutions to *MEP tables*. The solutions are identified by domain (e.g. in ν, t). A typical calibration procedure involves solving for one subset of parameters, storing these solutions, then going on to solve for another subset, while using the stored solutions of the first set when evaluating the model. MEP tables have a Python interface, so the stored solutions can be analyzed, plotted and/or reprocessed with external tools.

6.6 Continuity and solve domains

In many problems, radio astronomy not excepted, data volumes preclude processing an entire observation at once. Instead, data has to be broken up into chunks along the e.g. time and/or frequency axes, and processed sequentially. MeqTrees calls such chunks *tiles*. When solving for parameters, solutions are by necessity generated on a tile-by-tile basis. On the other hand, physical considerations can often provide continuity constraints between tiles, and it is important to take advantage of these constraints.

In the simplest case, a MeqParm will generate one solution per tile, and use that solution as the starting value for the next tile. No explicit continuity constraint is imposed. In this case one makes the tile size small, in accordance to how quickly a parameter is expected to vary. It is possible that the expected variation in a parameter is slower than the largest practical tile size. An extreme case of this is when trying to solve for a single value across the entire measurement. If the data for the entire measurement does not fit in memory, obtaining a global solution requires multiple runs through the data, which may impose unacceptable I/O penalties. In general this is a very thorny problem. One (rather inelegant) way around this is doing tile-by-tile solutions with the largest possible tile, and smoothing the solutions afterwards. Other options are averaging the data, or extracting a strided subset of the data.

A more complicated case relates to scenarios where we want to simultaneously solve for parameters that have different degrees of variability. A typical example from radio astronomy would be receiver phases (which are almost constant in frequency, but vary rapidly in time) and bandpasses (which vary very slowly in time, but have very complex behaviour in frequency). MeqParms address this via a technique called *subtiling*. Each MeqParm may be setup with its own subtile size, and an independent solution is then done within each subtile of the larger overall tile. In the example here, phases would have a subtile of size 1 in time, and bandpasses would have a subtile of size 1 in frequency. A separate phase solution per timeslot (constant across all frequencies) and a separate bandpass per frequency (constant across all times in the tile) would then be obtained.

The combination of tile size, subtiling and polynomial degrees makes for a very flexible way to specify parameter behaviour. It is in fact a challenge to present all these options to the user in a non-bewildering fashion.

7 Performance considerations

Given the large data volumes produced by radio astronomical instruments (and the even bigger volumes required to simulate instruments of the future such as the SKA), computational performance is always going to be an important issue, both in terms of processing speed and memory use.

With any software, there is generally a tradeoff between flexibility and efficiency. Highly optimized code is by its nature difficult to revise and extend, and vice versa. MeqTrees tries to get around this problem by providing highly optimized building blocks (i.e. nodes), while offering maximum flexibility in putting them together.

There is always some overhead associated with navigating a tree and passing results around, but this overhead is independent of domain size, whereas actual computational cost increases linearly with the number of, e.g., time and frequency points. This implies that MeqTrees is at its least efficient when using single-cell domains, where housekeeping overhead dominates, and at its most efficient when using large domains, where computational cost dominates. In practice, when using domains of 500–1000 cells, the performance of MeqTrees becomes comparable with that of hand-optimized code. This is achieved through a number of generic optimization techniques, which will be described below.

7.1 Optimal use of axes

Recall from Sect. 4.1 that a vells represents the value of a function using the minimum required number of axes of variability. Values with only a time or only a frequency dependence are passed around as vectors, and are expanded to arrays only when both a time and frequency dependence arises. This avoids redundant computation. It is also possible to explicitly structure trees to take advantage of this, i.e. to introduce extra axes as “late” in the computation as possible.

7.2 Result caching

Each node maintains an optional result cache, which allows computations to be reused. A straightforward but very powerful scheme of *dependency tracking* allows a node to figure out exactly when a result may be usefully cached. For example:

- A node with multiple parents should cache its result until all parents have retrieved it.
- When solving, a result with no dependence on solvable parameters can be cached until the next iteration. Results that do depend on solvable parameters are never cached, since they’re updated with each iteration.
- A result with no dependence on time can be cached until the next tile (assuming we’re iterating over time.)
- If all parents have cached their results, a child may discard cache.

In practice, this means that only the minimum necessary part of the tree is reevaluated when going from one solver iteration to the next, or from one tile to the next. It is also possible to fine-tune the caching policies to trade off computing time vs. memory footprint, etc.

7.3 Parallelism

In these days of cheap computing, parallel processing is the obvious approach to large computational problems. MeqTrees has been designed and implemented with this in mind. The tree paragigm provides ample opportunities for parallelisation, since different branches of the tree (and different trees of the forest) may be executed concurrently. On the other hand, the fact that trees may (and usually do) share branches towards the top makes for interesting scheduling problems – it's not much use to execute branches in parallel if they are going to spend most of their time waiting for the result of a single shared branch.

Meqserver has supported multithreading for a long time, so multiple-core machines may execute different branches of a tree concurrently. It employs a worker thread pool scheme, which avoids shared-branch bottlenecks. If a thread becomes stuck waiting for the result of a shared branch, another worker thread is woken up and assigned to a different part of the tree. In practice, this means that on a modern four-core machine, MeqTrees will happily keep all four cores fully occupied (as long as there's sufficient paralellism in the tree itself.)

Parallelism across a cluster is a far trickier proposition – one has to consider not only scheduling problems, but also cost of data transport between cluster nodes. In 2008, the first MPI version of MeqTrees was tested on a cluster in Oxford, with an eye on large-scale simulations for the SKA radio telescope.[8] This version allows parts of the tree to be distributed across a cluster, and uses MPI to pass results between children and parents that reside on different nodes. This version was tested across 8 cluster nodes and scaled reasonably well, albeit on a rather simple simulation problem. It is clear that the biggest amount of thinking needs to be put into the problem of how to distribute a given tree across a cluster efficiently.

[8]For the time being, it will only be made available to users with special capabilities.

8 The Measurement Equation of a Generic Radio Interferometer (RIME)

Since the development of 3GC may follow various routes, we will not presume to discuss specific calibration schemes here. Important issues are the dynamic range limitations caused by bright sources, the application of DDEs, and the imaging and deconvolution of residuals. It is clear that the Measurement Equation (RIME) has a vital role to play in this. In this section, we will discuss the general structure of the RIME, and some of its properties. We hope to make clear how elegant it is, and thereby to smooth the path to its wider adoption by the radio astronomical community. We will also indicate how the universality and modularity of the RIME opens the way for MeqTrees to offer a set of generic TDL processing scripts, which may be quickly adapted to a wide range of experiments with any radio telescope. The latter is done by means of the simple expedient of plugging in different Jones matrices, or a different sky model. Moreover, as discussed in Sect. 10, such pluggable Jones matrices and sky models may be contributed for sharing by anyone, anywhere.

Note that the description of the RIME given here is by necessity qualitative and brief, since we will concentrate on how the RIME pertains to MeqTrees rather than give a full formal exposition. For a more thorough treatment of the subjects touched upon here, see **?**.

The RIME was formulated by Hamaker et al. (1996) following preparatory work by Morris et al. (1964). Hamaker (2000) then rewrote it in 2×2 matrix form, which is the one we follow here.[9] Note that, since all the existing 2GC packages were written before the RIME was formulated, they *implicitly* implement a limited and approximate form of the RIME, usually optimized for a specific telescope.

The RIME is a *formalism* rather than one specific equation, and so it may be written down in many forms. A particularly elegant and simple form describes a "mostly empty" sky of discrete sources. In this form of the RIME, the predicted value of the visibility sample $\vec{V}_{pq}$ is given

[9]Some versions of the RIME still use 4×4 Mueller matrices. This is entirely equivalent, but much less transparent for our purposes.

by:

$$\vec{V}_{pq} = \vec{G}_p \left(\sum_{k=1}^{N} \vec{E}_{pk} \vec{X}_k \vec{E}^{\dagger}_{qk} \right) \vec{G}^{\dagger}_q \qquad (2)$$

where $\vec{V}_{pq}$ is the 2×2 visibility (also called *cohærency*, or *uv-data*) matrix measured by the interferometer formed by stations p and q. The sum is taken over the contributions $\vec{X}_k$ from N discrete sources in the field, at positions l_k, m_k. They are corrupted by instrumental effects that are represented by so-called Jones matrices (Jones 1941). All the terms of eq. (2) are 2×2 matrices, and "†" represents the Hermitian (or conjugate) transpose operator. The $\vec{E}_{pk}$ term is itself product of a number of Jones matrices corresponding to *direction-dependent effects (DDEs)* associated with station p and direction l_k, m_k, while $\vec{G}_p$ is a product of Jones matrices for the *direction-independent effects (DIEs)* associated with station p.

Note that eq. (2) assumes that all instrumental effects are station-based, i.e. can be fully described by Jones matrices, each associated with a particular station p. This is called the *selfcal assumption*. It is crucial because it increases the ratio between the number of equations (given by measured uv-data) and independent unknowns to the level where selfcal generates non-trivial solutions, as discussed in Sect. 9. In principle the observed data is also corrupted by *interferometer-based* errors (also called closure errors), which are conventionally modelled via extra multiplicative and additive terms on the elements of $\vec{V}_{pq}$. These can then be solved for (with some care), assuming the errors are sufficiently smooth in time.

The 4 elements of V_{pq} represent the 4 possible correlations between the two pairs of output signals from the two stations of an interferometer. These signals are usually labelled X and Y for linearly polarized receptors, or L and R for circularly polarized receptors[10]:

$$V_{pq} = \begin{pmatrix} v_{XX} & v_{XY} \\ v_{YX} & v_{YY} \end{pmatrix} \text{ or } \begin{pmatrix} v_{LL} & v_{LR} \\ v_{RL} & v_{RR} \end{pmatrix}$$

in which element v_{YX} predicts the value of the correlation between the Y signal of station p with the X signal of station q etc.

[10]A "linearly polarized" receptor is sensitive to linearly polarized radiation, e.g. a dipole.

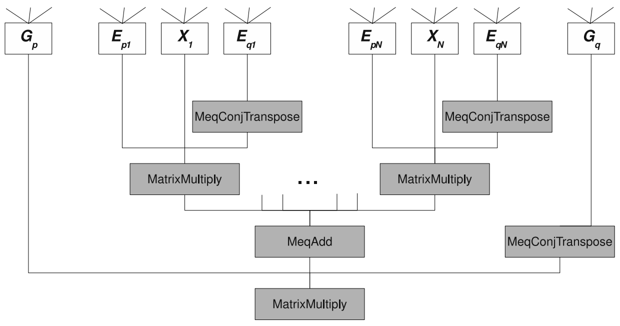


Figure 11: A subtree implementing the RIME given by eq. (2).

8.1 Implementing the RIME in MeqTrees

Implementing an equation like (2) in MeqTrees is very straightforward. On the top level, eq. (2) is just a small subtree composed of MeqAdd, MeqMatrixMultiply and MeqConjTranspose nodes (see Fig. 11). Such a subtree is constructed for every p, q pair. Note that the children of the subtree – nodes returning the constituent matrices of eq. (2), shown in lighter colour in the figure – can themselves be represented by arbitrary subtrees.

Figure 11 captures the essense of the modularity and flexibility of MeqTrees. The ability to plug in arbitrary subtrees (including arbitrary solvable parameters) to represent the source contributions $\vec{X}_k$ and the Jones matrices $\vec{E}_{pk}$ and $\vec{G}_p$ effectively allows for **arbitrary parameterizations** of the sky and instrumental models.

8.2 The Local Sky Model

The RIME predicts the visibilities observed by a particular radio telescope, given a particular source distribution. In eq. (2), $\vec{X}_k$ is the intrinsic cohærency that represents source k. The exact form of this matrix, and the corresponding subtree of Fig. 11, depends on the source model. In principle, $\vec{X}$ is a function of u, v coordinates $\vec{X}(u, v)$ (which, per each baseline pq, are themselves a function of time and frequency) that is a Fourier Transform (F.T.) of the brightness distribution $\vec{\mathcal{B}}(l, m)$, relative to source center. In the case of a point source (i.e. a delta function), this is trivial:

$$\vec{X} = \begin{pmatrix} I+Q & U+iV \\ U-iV & I-Q \end{pmatrix} \text{ or } \begin{pmatrix} I+V & Q+iU \\ Q-iU & I-V \end{pmatrix}, \tag{3}$$

depending on whether a linear (i.e. orthonormal) or circular polarization basis is used.[11] For extended sources, more complicated forms of $\vec{X}(u, v)$ may be provided via their own subtrees. The following forms have been implemented in MeqTrees at time of writing:

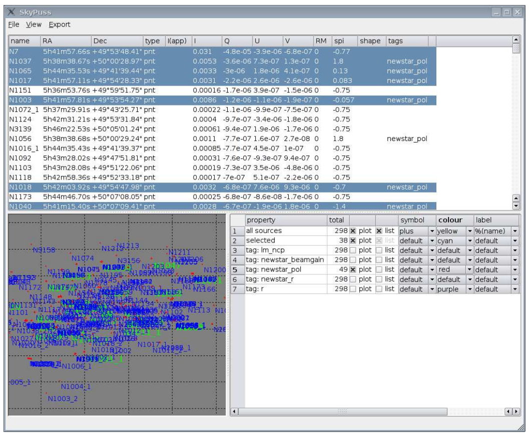
Gaussian components. Slightly extended sources may be approximated by a two-dimensional Gaussian distribution in the *lm*-plane, as is done in the NEWSTAR package. The F.T. of this is a Gaussian in the *uv*-plane, which is provided by a simple subtree with flux, extent and orientation parameters.

Images. Most 2GC packages use images (i.e. a gridded $\vec{\mathcal{B}}(l, m)$ representation) as the their standard sky model. For images, an FFT followed by degridding provides a computationaly effective way of estimating $\vec{X}(u, v)$ (see also Sect. 9). MeqTrees implements this approach via a combination of MeqFFTBrick and MeqUVInterpol nodes (Abdalla 2009).

Shapelets are another way to efficiently model extended source structure. $\vec{\mathcal{B}}(l, m)$ is decomposed into shapelets in the *lm*-plane; this can be efficiently evaluated in the *uv*-plane. The use of this in MeqTrees has been pioneered by Yatawatta et al. (2010b).

The above source representations may be freely mixed, simply by plugging in different kinds of subtrees for the different $\vec{X}_k$ terms in Fig. 11. Note how this differs from the traditional 2GC view of a single sky image. An image has the advantage of modelling arbitrary source structure, but it is limited by gridding errors (and distortions introduced by DDEs, see Sect. 9). To maximize dynamic range, a mixed sky model may be required. Such a mixed model may consist of, e.g. point sources and shapelets for the brightest sources, and an image for the faint background. This is in principle straightforward to implement via different subtrees.

[11]Some formulations include a factor of 1/2 in the definition of $\vec{X}$. See **?** for a discussion of these issues. Note also that it is common to use $\vec{B}$ instead of $\vec{X}$, calling it the source *brightness* (and indeed, the brightness of a source at the phase centre is equivalent to its coherency.) In this paper we'll use $\vec{X}$, reserving $\vec{B}$ for the bandpass Jones, below.



name	RA	Dec	type	I(app)	I	Q	U	V	RM	spi	shape	tags
N7	5h41m57.66s	+49°53'48.41"	pnt		0.031	-4.8e-05	-3.9e-06	-6.8e-07	0	-0.77		
N1037	5h38m38.67s	+50°00'28.97"	pnt		0.0053	-3.6e-06	7.3e-07	1.3e-07	0	1.8		newstar_pol
N1065	5h44m35.53s	+49°41'39.44"	pnt		0.0033	-3e-06	1.8e-06	4.1e-07	0	0.13		newstar_pol
N1017	5h41m57.11s	+49°54'28.33"	pnt		0.0031	-2.2e-06	2.6e-06	-2.6e-06	0	0.083		newstar_pol
N1151	5h36m53.76s	+49°59'51.75"	pnt		0.00016	-1.7e-06	3.9e-07	-1.5e-06	0	-0.75		
N1003	5h41m57.81s	+49°53'54.27"	pnt		0.0086	-1.2e-06	-1.1e-06	-1.9e-07	0	-0.057		newstar_pol
N1072_1	5h37m29.91s	+49°43'25.71"	pnt		0.00022	-1.1e-06	-9.9e-07	-7.5e-07	0	-0.75		
N1124	5h42m31.21s	+49°53'31.84"	pnt		0.0004	-9.7e-07	-3.4e-06	-1.8e-06	0	-0.75		
N3139	5h46m22.53s	+50°05'01.24"	pnt		0.00061	-9.4e-07	1.9e-06	-1.7e-06	0	-0.75		
N1056	5h38m38.68s	+50°00'29.24"	pnt		0.0011	-7.7e-07	1.6e-07	2.7e-08	0	1.8		newstar_pol
N1016_1	5h44m35.43s	+49°41'39.37"	pnt		0.00085	-7.7e-07	4.5e-07	1e-07	0	-0.75		
N1092	5h43m28.02s	+49°47'51.81"	pnt		0.00031	-7.6e-07	-9.3e-07	9.4e-07	0	-0.75		
N1103	5h43m28.08s	+49°51'22.06"	pnt		0.00019	-7.3e-07	3.5e-06	-4.8e-06	0	-0.75		
N1118	5h42m58.36s	+49°52'33.18"	pnt		0.00017	-7e-07	5.1e-07	-2.2e-06	0	-0.75		
N1018	5h42m03.92s	+49°54'47.98"	pnt		0.0032	-6.8e-07	7.6e-06	9.3e-06	0	-0.7		newstar_pol
N1173	5h44m46.70s	+50°07'08.05"	pnt		0.00025	-6.8e-07	-8.6e-08	-1.7e-06	0	-0.75		
N1040	5h41m15.40s	+50°07'09.41"	pnt		0.0028	-6.7e-07	-1.9e-06	1.8e-06	0	-1.4		newstar_pol

	property	total			symbol	colour	label
1	all sources	298	☒ plot	☒ list	plus	yellow	%(name)
2	selected	38	☒ plot	☒ list	default	cyan	default
3	tag: lm_ncp	298	☐ plot	☐ list	default	default	default
4	tag: newstar_beamgain	298	☐ plot	☐ list	default	default	default
5	tag: newstar_pol	49	☒ plot	☐ list	default	red	default
6	tag: newstar_r	298	☐ plot	☐ list	default	default	default
7	tag: r	298	☐ plot	☐ list	default	purple	default

Figure 12: The user interface to the MeqTrees Local Sky Model

The definitions of the various sources that are relevant for a particular observation are stored in a Local Sky Model (LSM). MeqTrees provides an end-user tool for managing this information (see Fig. 12). Once the user has supplied an LSM, the relevant subtrees are constructed programmatically.

Last but not least, we should note that the parameters of the different source parameterizations are automatically functions of, e.g., frequency and time, and may in principle be solved for just like any other parameter in a tree (always provided the observational data is sufficient to constrain the problem, of course.) MeqTrees makes no distinction between instrumental and source parameters: it is possible to solve for *any* subset of RIME parameters.

8.3 Jones matrices

In eq. (2), the intrinsic LSM source cohærency matrices $\vec{X}_k$ are "corrupted" by means of 2×2 Jones matrices. These are the real heart of the RIME. They represent the DDEs ($\vec{E}_{pk}$) associated with station p and direction l_k, m_k, and DIEs ($\vec{G}_p$) associated with station p.

$\vec{E}_{pk}$ and $\vec{G}_p$ are in principle themselves matrix products of a series of Jones matrices corresponding to individual physical effects.[12] Matrix multiplication does not com-

[12]Note that, because the signal path to each station is completely described by its own series of Jones matrices, the RIME is valid for arrays with very dissimilar stations, as is the case for LOFAR, and will proba-

mute, so the individual Jones terms must be placed into the equation in the correct order, corresponding to their physical order in the signal propagation path (but see below.)

It is very useful to have a common letter-based nomenclature for the standard Jones matrices. Below we will give a by no means exhaustive list, with nomenclature mostly following Noordam (1996). Where appropriate, we will mention what form the Jones matrix usually takes. Three important forms are the *diagonal* matrix, the *rotation* matrix, and the *scalar* matrix:

$$\begin{pmatrix} d_{11} & 0 \\ 0 & d_{22} \end{pmatrix} \qquad \mathrm{Rot}\,\phi = \begin{pmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{pmatrix} \qquad d \equiv \begin{pmatrix} d & 0 \\ 0 & d \end{pmatrix}$$

Note that diagonal or rotational form is subject to choice of basis. For example, a rotation in an orthonormal basis becomes a special kind of diagonal matrix in the circular polarization basis. (Below, we will assume an orthonormal basis unless otherwise noted.) Scalar matrices, on the other hand, are scalar regardless of basis. Within formulae, we shall use normal-weight capitals (e.g. A as opposed to $\vec{A}$) to distinguish scalar matrices.

These three forms are important due to matrix commutation. Scalar matrices commute with everything, while diagonal matrices commute among themselves, as do rotations. This allows us to commute certain Jones terms around the RIME in order to derive new forms of the equation and/or for purposes of either computational efficiency.

Common DDEs, in order of appearance in the signal path, are:

K-Jones: The phase term, accounting for the geometric delay and fringe stopping associated with station p and direction l, m. This is a scalar matrix of the form $K_p = \exp i(u_p l + v_p m + w_p n)$, so it may be commuted to any part of the RIME (in fact, phase itself is a sum of contributions from different parts of the signal path, which are commuted together into the overall K-Jones.) Commuting K_p and $K_q^\dagger$ next to each other, then multiplying them gives the Fourier Transform kernel – it could be said that K-Jones is at the heart of all interferometery!

Z-Jones: Ionospheric phase and amplitude effects. The latter are usually small enough to be ignored. The phase delay ζ has a known frequency dependence ($\propto \nu^{-1}$). Z is a scalar matrix, and so may be commuted anywhere. (This latter is rather fortunate for LOFAR, because it means that the large ionospheric phase effects may be calibrated at a convenient point early in the process. See also Sect. 9.) For narrow fields, Z can be treated as a DIE (and, for calibration purposes, is absorbed in G-Jones, see below).

F-Jones: Ionospheric Faraday rotation. This is a rotation matrix; the angle has a ($\propto \nu^{-2}$) dependence.

T-Jones: Tropospheric phase delay and extinction, another scalar matrix. For narrow fields, T can be treated as a DIE, and also absorbed in G-Jones during calibration. Alternatively, its values may be provided externally by water-vapour radiometers, as with mm-wave telescopes like ALMA.

E-Jones: The station beamshape (i.e. primary beam gain/phase in direction l, m). This is the most telescope-specific Jones matrix of them all, and the least well understood. For 2GC, it is usually assumed that E-Jones is time-independent and identical across stations, which means that it can be incorporated into the local sky model (in the form of apparent rather than intrinsic fluxes.) That this is not the case can severly limit imaging fidelity at instruments such as the VLA (Bhatnagar et al. 2008). Indeed, it can be argued that every radio telescope has or will have an E-Jones problem.

P-Jones: Projection matrix, corresponding to the projected position angle of the two receptors of a station on the sky. For dishes with narrow FoVs, this is a DIE, and is a simple rotation (constant for equatorial mounts, and offset by the time-variable parallactic angle for alt-az mounts.) For a horizontal dipole array like a LOFAR station, $\vec{P}$ becomes a more complicated expression (which can also be incorporated into an E-Jones model.)

Commonly used DIEs are:

bly be the case for the SKA.

D-Jones: "On-axis" polarization leakage[13] between the two receptors. This may be parameterized in terms of dipole orientation error (for linear receptors) and ellipticity. Note that the concept of "leakage" itself is a carry-over from 2GC packages, and is only a first-order approximation to the rather complicated polarization behaviour that can be more generally described by using proper E-Jones models. D-Jones is usually an almost-unity matrix, with small non-zero off-diagonal terms.

G-Jones: Complex gain, the staple of 2GC. Nominally, this corresponds to the electronic gain of the receivers, but for calibration purposes it cannot always be distinguished from the Z- and T-Jones terms, and so ends up subsuming all three effects. It is a diagonal matrix (unless electronic cross-talk is also incorporated, in which case the off-diagonal terms take on small non-zero values) with rapid variation in time, but little to none in frequency.

B-Jones: Electronic bandpass. This is a diagonal matrix like G-Jones, but it has considerable structure in frequency, and only slow (if any) variation in time. Since D-Jones varies on similar timescales, it may be useful to combine the two into a full 2×2 matrix.

8.4 Simulation vs. calibration

Real-life applications of the RIME in MeqTrees have, to date, fallen into two broad categories, *simulation* and *calibration*.

Simulation has become an increasingly important field in the past decade, due to the large number of new radio telescopes being designed and built. For simulation, the RIME (plus an optional noise term) predicts the output of a [real or theoretical] telescope observing a model sky. Given a sky model, and a set of Jones matrices for the DDE and DIE components (see e.g. Fig. 14), MeqTrees constructs a set of per-baseline trees corresponding to a RIME such as that in eq. (2), and evaluates them for a series of times and frequencies specified by a Measurement Set (MS). The resulting similated visibility data is then written out to the MS.

[13]In 2GC practice, D-Jones is the leakage in the direction of the dominating source, which is usually at the field centre.

Calibration is, essentialy, model fitting, as described in Sect. 6. The RIME is used to predict model visibilities (similar to the simulation case), but the outputs of the subtrees are then treated as the model functions $M^{(pq)}$ of Sect. 6, and fitted (by solving for their parameters) to the observed data $D^{(pq)}$, which is read from an MS. The resulting residuals are also written out to the MS. If the parameters being solved for are the complex station gains (the G-Jones term), this procedure is the equivalent of traditional 2GC selfcal. However, since MeqTrees in principle allows for arbitrary parameterizations, the same approach can be used to solve for, e.g., coefficients of beam models, ionospheric models, etc.

These two applications of the RIME are superficially similar, in that in both cases the RIME is used to predict model visibilities. However, the kinds of Jones matrices employed can be significantly different. For simulations, we are usually interested in a realistic representation of the underlying physics, so we use some of the Jones terms listed above, as relevant to a particular simulation. For example, a simulation may usefully incorporate separate Z-Jones, T-Jones and G-Jones terms, employing different numerical models for their matrix elements. During calibration, on the other hand, we may be unable to solve for these effects separately, since they all add up into one rapidly varying per-station phase term (assuming a narrow frequency band and FoV, where the different frequency behaviour of Z-, T- and G-Jones cannot be distinguished, and Z- and T-Jones become direction-independent.) Nor do we need to, since it is only the overall phase term that we need to calibrate in order to image the data properly. Therefore, for calibration purposes, all three effects can be captured by a single G-Jones term with solvable (complex) diagonal elements.

We call such forms of the RIME *phenomenological*, since they are meant to provide sufficient degrees of freedom (i.e. solvable parameters) to capture the *effect* of the instrument on observed data, with little regard to the underlying physics. Here's an example of a real-life phenomenological RIME for polarization calibration of the WSRT:

$$\vec{V}_{pq} = \vec{B}_p \vec{G}_p \vec{P} \left(\sum_{k=1}^{N} \vec{E}_k K_{pk} \vec{X}_k K^{\dagger}_{qk} \vec{E}^{\dagger}_k \right) \vec{P}^{\dagger} \vec{G}^{\dagger}_q \vec{B}^{\dagger}_q \qquad (4)$$

Here, $\vec{B}_p$ is a solvable full 2×2 matrix to capture bandpass and leakage (highly variable in frequency, but only slowly variable in time), $\vec{G}_p$ is a solvable diagonal matrix to capture rapidly variable amplitude and phase effects (no variation in frequency, rapid variation in time), $\vec{P}$ is a dipole orientation matrix (known, constant, and same for all stations), $\vec{E}_k = \vec{E}(l_k, m_k)$ is an apriori primary beam model (same for all stations), and K_p is the usual phase term.

At very high ($> 10^5$) dynamic ranges, short-term low-level instability of the WSRT bandpass makes it impossible to separate the $\vec{B}$ and $\vec{G}$ solutions, so a simpler form of the RIME may be used instead. This form is roughly equivalent to per-channel selfcal in 2GC terminology:

$$\vec{V}_{pq} = \vec{G}_p\vec{P}\left(\sum_{k=1}^{N} \vec{E}_k K_{pk}\vec{X}_k K^\dagger_{qk}\vec{E}^\dagger_k\right)\vec{P}^\dagger\vec{G}^\dagger_q \qquad (5)$$

Here, $\vec{G}_p$ is solvable, with rapid variation in frequency and time on the diagonal, and slow variation in time on the off-diagonal. Another version of this equation was used by Smirnov (2010) to produce the result in Fig. 1:

$$\vec{V}_{pq} = \vec{G}_p\vec{P}\left(\sum_{k=1}^{N} \Delta\vec{E}_{pk}\vec{E}_k K_{pk}\vec{X}_k K^\dagger_{qk}\vec{E}^\dagger_k\Delta\vec{E}^\dagger_{qk}\right)\vec{P}^\dagger\vec{G}^\dagger_q \qquad (6)$$

Here, *differential gain* $\Delta\vec{E}_{pk}$ is a phenomenological Jones term capturing the source-dependent complex gain variations (which can be due to any combination of physical effects). It is diagonal, and solvable on large time and frequency scales.

8.5 RIME and software modularity

Early thinking about implementations of the RIME in software (Noordam 1996) tended towards a “one equation to rule them all” doctrine. A particular sequence of Jones terms was chosen and carefully elaborated on, with the implicit expectation that it would be appropriate to all telescopes and scenarios, if only all the required Jones matrices were properly implemented. This thinking also had a strong influence on the AIPS++/CASA calibration modules.

Our subsequent work on the subject has convinced us that a more flexible approach is vital. The discussion in the previous section suggests that no specific set of Jones terms can be appropriate to all cases – not even if we restrict ourselves to one relatively simple instrument such as the WSRT, as is clear from eqs. (4–6). The need to build *arbitrary* measurement equations drove the design of MeqTrees from the beginning.

Fortunately, the structure of the RIME itself encourages just such a flexible and modular approach. Any specific knowledge about an instrument can be encoded in the form of Jones matrices, and all Jones matrices work the same way. This has the very considerable advantage that each instrumental effect can be modelled in isolation, while it will still be taken into account correctly within the RIME. *This is a far cry from the intractable and approximate mathematical expressions in older data reduction packages, which are often not implemented explicitly, but scattered in poorly documented bits and pieces throughout the code.*

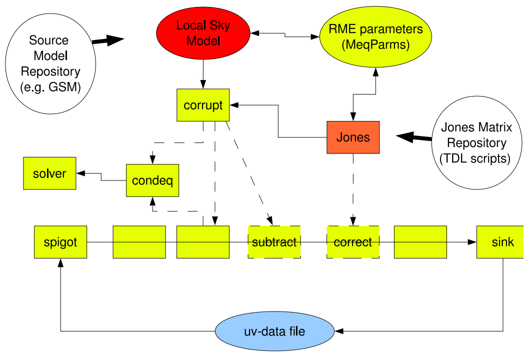

Source Model Repository (e.g. GSM)
Local Sky Model
RME parameters (MeqParms)
corrupt
Jones
Jones Matrix Repository (TDL scripts)
solver
condeq
spigot
subtract
correct
sink
uv-data file

Figure 13: Schematic structure of a typical TDL processing script. Thanks to the RIME, with its Jones matrices and Local Source Model, such scripts are highly modular, and may quickly be adapted to different telescopes and experiments. This greatly facilitates collaboration and rapid experimentation. See also the text.

Figure 13 shows schematically how the modularity of the RIME can be exploited to generate a set of generic processing scripts that can be adapted for use with any radio telescope. Rather than being written in basic TDL from scratch, these scripts are based on a set of generic Python frameworks called the Cattery, which ships with MeqTrees 1.1. The user selects a Cattery script for a particular operation (e.g. simulation, or visualization, or some sort of calibration), and perhaps customizes it by changing a few lines of Python code. He selects a suitable sequence of Jones modules from a “Jones repository”.[14] Finally, he selects a Local Sky Model, for which a number of model formats are supported. All these modules are free to define their own custom processing options (solvable parameters, etc.) MeqTrees automatically extracts the relevant option set from the selected modules, and presents it to the user via a GUI (Fig. 14). When used in batch mode, it can load these options from configuartion files.

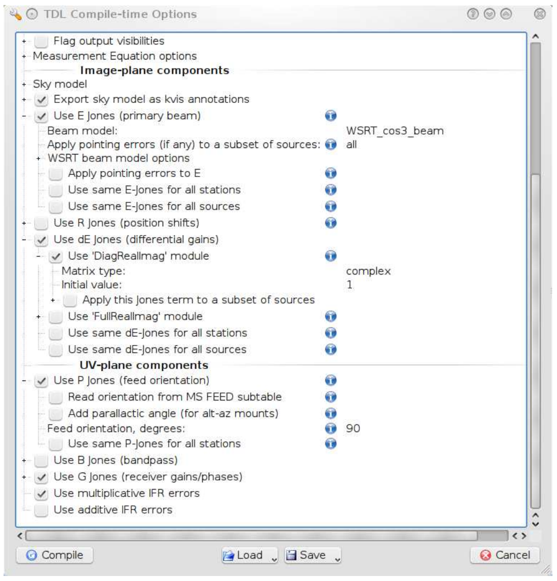

```
TDL Compile-time Options
+ [ ] Flag output visibilities
+ Measurement Equation options
        Image-plane components
+ Sky model
+ [✓] Export sky model as kvis annotations
- [✓] Use E Jones (primary beam)
    Beam model:                                          WSRT_cos3_beam
    Apply pointing errors (if any) to a subset of sources:  all
  + WSRT beam model options
    [ ] Apply pointing errors to E
    [ ] Use same E-Jones for all stations
    [ ] Use same E-Jones for all sources
+ [ ] Use R Jones (position shifts)
- [✓] Use dE Jones (differential gains)
  - [✓] Use 'DiagRealImag' module
      Matrix type:                                       complex
      Initial value:                                     1
    + [ ] Apply this Jones term to a subset of sources
  + [ ] Use 'FullRealImag' module
    [ ] Use same dE-Jones for all stations
    [ ] Use same dE-Jones for all sources
        UV-plane components
- [✓] Use P Jones (feed orientation)
    [ ] Read orientation from MS FEED subtable
    [ ] Add parallactic angle (for alt-az mounts)
    Feed orientation, degrees:                           90
    [ ] Use same P-Jones for all stations
+ [ ] Use B Jones (bandpass)
+ [✓] Use G Jones (receiver gains/phases)
  [✓] Use multiplicative IFR errors
  [ ] Use additive IFR errors
```

Figure 14: The Options GUI of a typical processing script. A number of Jones modules may be selected for inclusion in the tree. Each Jones module can implement its own custom option set, which is displayed in submenus.

9 Direction Dependent Effects (DDE)

In view of the importance of DDEs for 3GC, and the relative paucity of available experience, it seems justified to devote a few remarks to their treatment as part of the RIME, and the special features that MeqTrees brings to bear on the subject. However, in view of the difficulty, and urgency, of exploring the *terra incognita* of DDEs, perhaps the most important features of MeqTrees are the possibilities for rapid experimentation, visualization and collaboration.

9.1 Application and deconvolution

Since radio interferometers tend to have sparsely filled apertures, the images they produce suffer from a point-spread-function (PSF) with relatively high sidelobes. In order to study the [usually faint] objects of interest, the image has to be deconvolved to remove the PSF around the bright sources. Accurate deconvolution is complicated by the gridding, and by the fact that the PSF varies over the field of view, especially in the presence of DDEs. Therefore, all high-dynamic-range imaging schemes subtract the contributions of at least the brightest sources in the source model from the uv-data itself, before Fourier transforming the uv-residuals to a residual image.

[14]At time of writing, a number of standard Jones modules were included in the MeqTrees 1.1 binary release. More modules can be checked out manually from a designated area on our Subversion server. We intend to set up a more formally structured repository as more diverse Jones modules are implemented.

It is always possible to apply DDEs (and DIEs) in the *forward* direction,[15] i.e. when predicting corrupted uv-data from a sky model. This can be done with arbitrary precision in the trivial case of a discrete source, using an equation similar to (5). This is called the *DFT* approach, since the K-Jones terms in eq. (5) correspond to the kernel of a Discrete Fourier Transform. DFT is *de rigueur* for the brightest sources, where gridding would introduce unacceptable inaccuracies. We call these *Category I* sources. The DFT approach is accurate but relatively slow (scaling as the number of sources.)

For the fainter (*Category II*) sources in the LSM, less accurate but more efficient FFT methods may be used. For this, MeqTrees offers the FFTBrick/UVInterpol node suite (collectively known as the *UVBrick*). Any selection of LSM sources, with any parameterization, may be gridded onto up to four ($IQUV$) 3D (l, m, ν) image cubes, to be collectively Fourier transformed and then de-gridded onto the actual *uvw* coordinates. An implementation by Abdalla (2009) does this with remarkable accuracy, and for large patches of the sky. Bhatnagar et al. (2008) have developed an algorithm for efficient application of DDEs during the de-gridding process, using *convolutional functions* (essentially, the F.T. of the Jones term) as the interpolation operator. We plan to incorporate this algorithm into a future version of the UVBrick.

The Bhatnagar method can also be used in the *backward* direction, i.e. when making an image from uv-data. Unlike the case of DIEs, it is not possible to correct for DDEs in the uv-data: the latter can only be corrected for a single point in the sky[16]. DDEs can be applied while gridding, though, using the inverse of the relevant Jones matrix[17] for interpolation.

FFT-based methods come with inherent inaccuracies (both due to gridding, and the approximate nature of the convolutional approach.) There's a complex four-way trade-off between attainable dynamic range, the number of sources to be subtracted via DFT, the resolution of the FFT, and available computing power, which is beyond the scope of this paper.

[15]The nomenclature is consistent with other authors, and is determined by the direction of signal propagation.

[16]Thus, in the presence of DDEs, corrected uv-data do not exist.

[17]Bhatnagar et al. (2008) require the Jones matrix to be approximately unitary for the backward application of DDEs, though it is not yet clear whether this is a fundamental requirement.

In the Bhatnagar et al. (2008) paper, the convolutional approach is used to correct for effects of the primary beam pattern (E-Jone) on VLA data. It is shown to work perfectly for simulated data, but only to first order for actual observations. Our interpretation of this discrepancy is that the accuracy of their E-Jones model is the limiting factor. This can probably be improved by using a parametrized E-Jones model and solving for the parameters continuously during the observation with the help of calibration beacons in the field (see below).

9.2 Modelling DDEs

When approaching the problem of modelling DDEs, we might formulate *Instrumental Design Rule No. 1:* instrumental effects must behave smoothly in all relevant dimensions (ν, t, l, m).[18] Only then can we expect to model them with a sufficiently small number of solvable parameters, and use interpolation between a minimum number of sampling points. This is very important in the light of the fundamental question whether there is enough information available for 3GC (see Sect. 9.3 below).

Next, we suggest that DDE models should be expressed mathematically as a series expansion (such as e.g. shapelets), whose coefficients may then be treated as RIME parameters. This approach makes it simple to minimize the number of parameters by using only as many terms as are needed to fit the available information. Thus, the art of modelling DDEs hinges on finding suitable basis functions. As an illustration, we will briefly discuss the cases of the two most important DDEs, the ionosphere (Z-Jones) and station beamshapes (E-Jones):

Modelling the ionosphere. The **Z-Jones** term incorporates the ionospheric phase delay and amplitude (extinction and defocusing). Since this has the same effect on both polarizations, Z-Jones is a scalar matrix, so we only need a single expression: $a_{pk} \exp^{2\pi i \zeta_{pk}}$, in which ζ_{ik} is the ionospheric phase as seen from station p, in the direction of (l_k, m_k). (The amplitude effect is outside the scope of this paper, and is indicated here for completeness only.) The ionosphere is, of course, not part of telescope "de-

[18]This rule has been often intoned, but not always heeded, during the design of LOFAR. This will not be different for the other telescopes, including the SKA.

sign" per se, but fortunately it is smooth on at least the large scales.

Especially at low (LOFAR) frequencies, the phase can vary by many radians, at a rate of as much as 0.1 rad/s. Under such conditions, the calibration system has to maintain 'phase-lock' at all times, i.e. to track the phase excursions. For this reason, it is fortunate that the Z-Jones matrix is scalar, and so may be calibrated for independently (by commuting it to a convenient place in the RIME.)

This can be represented in a frequency-independent way by means of the Total Electron Content along the propagation path: $\phi \approx -25\lambda\mathrm{TEC}$ (rad). Since an interferometer only sees relative phase, we only need to measure or model the relative TEC. Using the TEC makes it easy to include GPS information for sampling the ionosphere, in addition to celestial sources.

The Minimum Ionospheric Model (MIM) is a way to model the ionosphere with minimal assumptions about its 3D structure, and a minimum number of parameters. Assuming *only* that the TEC distribution varies smoothly in all dimensions, one postulates the existence of a smooth multi-term function like a low-order 6-dimensional polynomial[19] $\mathrm{TEC}(x, y, z, l, m, t)$. One then uses as many terms as necessary to fit the available information, i.e. the self-cal phases of the calibration beacons. The required number of terms depends on the ionospheric conditions, of course, so the system should ideally be able to adapt itself dynamically to changing conditions.

Modelling station beamshapes. The **E-Jones** term represents station beamshape. This is the most telescope-specific Jones matrix of them all, and the least well understood.

The E-Jones matrix will in general require four separate expressions for its elements, which may share a common set of RIME parameters. Only in particularly well-behaved cases like the WSRT can the E-Jones be approximated by a diagonal matrix. The E-Jones of other telescopes are increasingly more complicated, ranging from the VLA, with its beam-squint due to off-axis receivers, to the horizontal dipole arrays of LOFAR.

Since relatively little work has been done in this area up to now, parametrized beamshape expressions are urgently needed for all radio telescopes. Suitable multi-term functions (e.g. polar shapelets, Bessel functions) are required. Since this is often quite a specialized job, it should be done *on paper* by EM experts, and implemented in MeqTrees by others. Obviously, fewer terms will be sufficient if one is only interested in a small part (e.g. the main lobe) of the beam.

9.3 Solving for DDE parameters

For the best results, DDE parameters must be estimated continuously in a "closed-loop" manner, using suitable sources in the field, which we call *calibration beacons*. For the ionospheric phase, two approaches to this have been proposed: *field-based calibration* (FBC; Cotton et al. 2004), which fits source positions in the image plane, and *source peeling and atmospheric modelling* (SPAM; Intema et al. 2009), which fits a global screen to direction-dependent phase solutions obtained by peeling. These can be seen as steps towards a fully general MIM-based approach (see above). For station beamshapes, closed-loop estimation has not yet been attempted very often because there was no software to make use of the faint sources in the field. MeqTrees has changed that (Sect. 2).

Tackling DDEs implies that we have to solve for a much larger number of RIME parameters than before. Besides the practical problem of extra processing (which may well turn out to be a major bottleneck, but will not be discussed here), this raises the fundamental issue or whether we actually measure enough data to constrain the problem.

Selfcal works because it is an overdetermined problem. An interferometer composed of N_{st} stations generally yields $n_{\mathrm{eq}} = O(N_{\mathrm{st}}^2)$ measurements (per time/frequency bin), while traditional 2GC selfcal equations have $n_{\mathrm{uk}} = O(N_{\mathrm{st}})$ unknowns. For 3GC, we are adding many extra parameters, so we must be careful that the problem remains well-determined.

In general, the extra parameters are associated either with per-station Jones matrices, which adds a constant scaling factor to n_{uk}, or with global models such as the MIM, which just adds some fixed number of unknowns to the total. For large N_{st} (e.g. the SKA), the scaling law for n_{eq} then ensures that the problem easily remains overdetermined. However, many instruments operating today

[19] Any multi-term smooth function will do. Alternatives are Zernike polynomials or Karhunen Loeve (KL) functions.

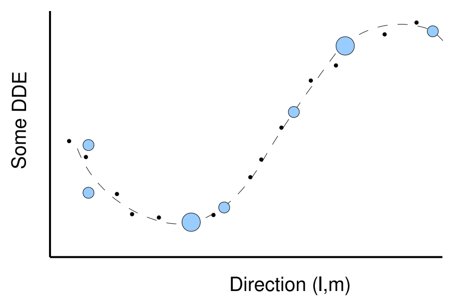


Figure 15: The parameters of the smooth functions that represent Direction-Dependent Effects (DDE) are estimated with the help of bright sources (calibration beacons, big dots). These functions (dotted line) may then be interpolated to calculate the DDE values in the directions (l, m) of fainter sources (small dots).

have a relatively small N_{st}, so we must still look for ways to reduce the number of additional unknowns per station.

One way to do it is to take advantage of the relative smoothness of DDEs in the time, space and frequency domains. As a simple example, the differential gain solutions in Fig. 2 consist of one complex parameter per source, per station, per 60×30 time-frequency bins. This represents only a fractional increase in the number of unknowns per time-frequency bin (i.e. per equation). As a more complicated example, if it is possible to represent the station E-Jones by a series expansion such as polar shapelets, whose coefficients are themselves smooth functions of frequency and time, then the actual number of extra unknowns added to the problem is also << 1 per bin. MeqTrees is designed to exploit these approaches with minimum fuss.

The $n_{\mathrm{eq}} > n_{\mathrm{uk}}$ condition is necessary, but not sufficient. We must also measure enough signal to constrain the problem. When solving for DDE, this requires a sufficient number of calibration beacons in the observed field. This is not a problem at low frequencies (e.g. LOFAR), where the field is large, and the sources relatively bright. But at higher frequencies this could be a problem. Therefore, 3GC software must be able to solve for slowly varying *differential* effects, using long integration times, and wide frequency bands, so that relatively faint sources can be used effectively. MeqTrees is specifically designed to make this possible (see Sects. 4.1 and 6.6). In this context the results discusssed in Sect. 2 are highly encouraging.

We expect that a lot of experimentation will be necessary before the various DDEs are modelled correctly. Based on our experience described in Sect. 2, we recommend the following procedure for modelling DDEs experimentally, using real data:

1. Solve for the rapidly varying instrumental effects in the direction of the dominating sources in the field, and correct the uv-data.
2. Solve for slowly varying *differential* effects (e.g. complex gain) in the direction of some of the fainter sources in the field, as shown in Fig. 2.
3. Use the above to solve for the parameters of experimental DDE functions, and study the quality of the fit.

As illustrated in Sect. 2 this procedure offers considerable scope for experimentation with different models between steps 2 and 3. In the spirit of Fig. 13, MeqTrees supports such a “cottage industry” by offering processing scripts that take care of steps 1 and 2, and allow experimental Jones matrices to be plugged in for step 3. The only thing that the experimenter has to do is to devise a parametrized analytical model (on paper) that might fit the results of step 2, implement this in the form of a Jones matrix (which is very quick, if adapting an existing one), and then see how well the model can be made to fit. Once the DDE models are known, it is also possible to solve directly for their parameters from the uv-data, without going through the intermediate step of differential gains.

10 The Rate of Evolution

Assuming that it is true that the new telescopes need 3GC to achieve their advertized performance, there is a lot of work to be done. Even if we roughly know what to do, the new algorithms and tools must be fully developed, and tested in practice. This will require trial and error, and the involvement of many talented user-developers around the world. In addition, versions of 3GC must be implemented efficiently on the powerful production machines that handle the huge amounts of data that are generated by the new telescopes. And because the data volumes are so large, new ways have to be found to allow users to interact with the data reduction process, and to develop their own processing techniques.

In the meantime, the new telescopes are already being built, while neither the software nor the users are ready for them. Everyone is under tremendous pressure to get things to work, and there is very little room for experimentation. As a result, the new telescopes, and the upgraded old ones, will not reach their full potential for some time to come[20]. Fortunately, the much greater capabilities of the new hardware will compensate for this to some extent.

However, the question presents itself whether the rate of evolution of radio astronomical data reduction software might not be increased somehow. Due to the sustained success of the existing 2GC packages, this rate has been low in the two decades after the "creative eighties", especially if we only count software that has actually reached the end user.

One of the lessons from History is that rapid progress occurs when many people get involved in a process that allows them to compete without holding each other back. This is only possible if those people share a common language, and have easy access to each other's results. Progress is even faster if they can easily build on each other's results, using what works and ignoring the rest. MeqTrees can help to create such conditions.

First of all, the question of a Common Language is an easy one: The RIME is a big improvement over earlier instrumental models, and it is complete in the sense that it is valid for all existing and planned radio telescopes. Therefore, the RIME is an excellent, and uncontroversial candidate. Unfortunately, for all kinds of valid reasons, its adoption by the radio astronomical community has been slow. The use of MeqTrees offers an opportunity to redress this.

In terms of easy access to each other's results, MeqTrees offers the following:

- All processing scripts, Jones matrix definitions and LSM source models are TDL scripts, which are just Python, so they can be easily exchanged by email or internet.

- The Purr tool (released with MeqTrees 1.1, but can be installed and used independently) provides a highly effective medium for reporting on a data reduction process with a minimum of effort. It semi-automatically creates *purrlogs* that contain things like images, plots, input parameters, and of course text. Purrlogs are directly available as HTML, for easy exchange via the web[21].

[20]History is full of precedents. It took the WSRT 36 years to reach the thermal noise on 3C84, in all 4 Stokes parameters.

As to building on each other's results, Fig. 13 illustrates that the modularity of the RIME makes it very easy to take an existing processing script, and then to adapt it by plugging in other Jones matrix definitions or LSM source models. Modules may be exchanged between individuals, but we could also think in terms of repositories to which anybody, anywhere can add:

- A repository of standard TDL scripts for different data processing operations. Such scripts can be simply cannibalized with a text editor.

- A repository of Jones matrix definitions, with versions for all existing and future radio telescopes.

- A repository of LSM source models, or tools that can convert subsets of surveys like NVSS or WENSS into an LSM that can be used and updated with MeqTrees.

- Specialized experts can develop highly sophisticated C++ nodes, without knowing anything more than the simple interface between nodes. An example is the UVBrick discussed in Sect. 9.

An important side-effect of this TDL approach is that new algorithms are directly available to the astronomical user also[22]. The resulting feedback should be a very important stimulus for rapid progress. It also creates more knowledgeable users.

[21]Purrlogs play an important role in the Oxford/SKADS Set of Standard Challenges (SSSC), a website for demonstrating new algorithms, and for educating the new generation of radio astronomers.

[22]And wouldn't it be nice if the software running on production systems could be reconfigured by means of TDL scripts, some of which could be prepared and debugged by the user himself?

But the MeqTrees features that will probably have the greatest impact on the rate of evolution are the following:

- Rapid experimentation: The Tree Time Unit (TTU)[23] is defined as the time it takes to implement a new idea in MeqTrees, and obtain a result to look at. It is less than a day. This should be compared to the weeks, months, years or even decades to get a new idea implemented in any of the 2GC packages, which have all been virtually frozen for decades. And anyone can do it.

- MeqTrees offers unprecedented visualisation of intermediate results. It is possible to inspect the result of *every* node in the tree, at all times, and it is very easy to define customized "PyNodes" that display the results of multiple nodes in the same plot in a particular way. Being able to see what is going on, together with rapid experiments, is the engine of progress.

With all this in mind, we have started an internet-based "Creative Commons" for the 3GC community. It has loosely knit groups that pursue major themes like the ionosphere or station beamshapes, and others that work on new concepts like shapelets or the UVBrick. Perhaps the most beneficial effect of such a collaborative network will be that it enables clever, hungry but relatively isolated people to join the fun, without luring them away from their home institutes. *This is an untapped and very welcome source of extra manpower to swell the ranks of the new generation of users and developers*.

11 Conclusions

MeqTrees raises the art of instrumental modelling to the level where user-developers can concentrate on the physics of the problem, while the complex numerical machinery, e.g. for solving for arbitrary subsets of parameters, is 'hidden under the hood'. In the special case of radio astronomy, the correct treatment of various instrumental effects is greatly facilitated by the elegant matrix formalism of the Measurement Equation of a generic radio telescope (RIME). The latter is well on its way to become the new Common Language of radio astronomy.

In a separate paper (Smirnov 2010), we have demonstrated that MeqTrees can exceed the results of the best 2GC package (NEWSTAR) by achieving a dynamic range of almost 2 million with WSRT data, while also dealing with DDEs (see also Sect. 2). Very importantly, we tentatively conclude that, by using the kind of features offered by MeqTrees, it is possible to use relatively faint (a few mJy) sources in the field as calibration beacons for the estimation of DDE parameters. This is a major step towards addressing the fundamental problems of availability of sufficient information for 3GC.

The present collection of MeqNode classes offers basic functionality, with bias towards radio astronomy (see Appendix A). There are various TDL (Python) frameworks to help the user in building complex trees; these in fact evolve much more rapidly than the binary release cycle. We also intend to offer a MeqWizard tool to help both novices and experts to find their way in the multiverse of possibilities. The MeqTrees kernel is robust and efficient, and has been tested thoroughly on real data (see Sect. 2).

MeqTrees is (slowly) beginning to find its place, propelled by the increasingly urgent need for 3GC simulation and calibration software for radio astronomy. It plays its designated role as pathfinder for LOFAR calibration, as illustrated by impressive all-sky LOFAR images (Yatawatta et al. 2010a). It has been used as the main education tool in several international workshops, to train the the new generation of radio astronomers in the use of the RIME and 3GC.

The short turnaround time and extended visualization offered by MeqTrees has the potential of greatly increasing the *rate of evolution* of data reduction software, by enabling a (much) larger number of people to experiment (much) faster. Therefore, much of our effort in the near future will be concentrated on nurturing the growth of a user community, as discussed in Sect. 10. We feel that such an investment will be vastly more effective than trying to implement the long list of desirables with our tiny core team, using only our own ideas. To use a popular platitude: "Rather than giving a man a fish, so he eats for a day, we will teach him to fish, so he eats all his life".

Encouragingly, there are indications that such a collaborative network is already forming. A good example

[23]The TTU is inspired by the Brouw Time Unit (BTU), which is rumoured to be half a quiet afternoon.

is the recent re-implementation of a complex node like the UVBrick by Filipe Abdalla at UCL, with input from widely separated others in Canada, US, UK and Europe. Another example is role that MeqTrees plays in the multi-faceted interaction between the far-flung participants of the LOFAR project.

Acknowledgements

Up to now, the following people have made direct contributions to MeqTrees, either by implementing and testing nodes, or by writing and exercising TDL scripts: Sarod Yatawatta, Tony Willis, Maaijke Mevius, Ger van Diepen, Ronald Nijboer, Filipe Abdalla, Rob Assendorp, Ilse van Bemmel, Ian Heywood, Hans-Rainer Kloeckner, Rense Boomsma, Michiel Brentjens, Joris van Zwieten, Alessio Magro. Many more should follow, and will be acknowledged, as our collaborative network grows.

Others have contributed to the distribution of MeqTrees and its operation on different platforms: Chris Williams, Stef Salvini, Mike Sipior, James Anderson and George Heald. Helpful comments were extracted from Wim Brouw. MeqTrees uses various modules from AIPS++/CASA, particularly those written by Ger van Diepen, Wim Brouw and Tim Cornwell. Ger van Diepen and Malte Maquarding have also been instrumental in making the CASA installation more robust. For the application of DDEs we have been influenced by the work of Sanjay Bhatnagar and Steve Gull. The MeqTrees Forum website is kindly hosted by University College London, and operated by Julien Girard of the Observatoire de Paris.

The authors are particularly grateful for the faith and patience of a succession of ASTRON managers over the years: Marco de Vos, Mike Garrett, Ronald Nijboer, Arnold van Ardenne, Kjeld van der Schaaf, Harvey Butcher. And finally, the very substantial support of Steve Rawlings (Oxford Astrophysics, UK) and Peter Dewdney and Sean Dougherty (DRAO, Canada) has made all the difference.

References

Abdalla, F. B. 2009, in MCCT SKADS Mixed Workshop: Towards third-generation calibration in radio astronomy

Bhatnagar, S., Cornwell, T. J., Golap, K., & Uson, J. M. 2008, A&A, 487, 419

Brouw, W. N. 1996, AIPS++ Note 224: AIPS++ Least Squares background, Tech. rep.

Cornwell, T. J. & Wilkinson, P. N. 1981, MNRAS, 196, 1067

Cotton, W. D., Condon, J. J., Perley, R. A., et al. 2004, in Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series, Vol. 5489, Ground-based Telescopes, ed. J. M. Oschmann, Jr., 180

de Bruyn, A. G. 2006, in SKA Calibration and Imaging Workshop 2006

Ekers, R. D. 1983, in Serendipitous Discoveries in Radio Astronomy, ed. K. I. Kellermann & B. Sheets, 154

Hamaker, J. P. 2000, A&ASS, 143, 515

Hamaker, J. P., Bregman, J. D., & Sault, R. J. 1996, A&ASS, 117, 137

Hopper, G. M. 1952, in ACM '52: Proceedings of the 1952 ACM national meeting (Pittsburgh) (New York, NY, USA: ACM), 243

Intema, H. T., van der Tol, S., Cotton, W. D., et al. 2009, A&A, 501, 1185

Jones, R. C. 1941, J. Opt. Soc. America, 31, 488

Kemball, A. J. & Wieringa, M. H. 1996, AIPS++ Note 229: MeasurementSet definition version 2.0, Tech. rep.

Knuth, D. E. 1973, The Art of Computer Programming, Vol. 1, Fundamental Algorithms, 2nd edn. (Reading, Massachusetts: Addison-Wesley)

Madsen, K., Nielsen, H. B., & Tingleff, O. 2004, Methods for Non-Linear Least Squares Problems (2nd ed.)

Morris, D., Radhakrishnan, V., & Seielstad, G. A. 1964, ApJ, 139, 551

Noordam, J. E. 1996, AIPS++ Note 185: The Measurement Equation of a Generic Radio Telescope, Tech. rep.

Smirnov, O. M. 2008, The TDL Bedside Companion

Smirnov, O. M. 2010, A&A, in prep.

Yatawatta, S., Brentjens, M. A., de Bruyn, A. G., et al. 2010a, Experimental Astronomy, Special LOFAR issue, in press

Yatawatta, S., de Bruyn, G., Brentjens, M., Nijboer, R., & Noordam, J. 2010b, A&A, in prep.

A Available MeqNode classes

Perhaps the most concise description of the capabilities of MeqTrees for the discerning user is an overview of the node classes that are available already, and the ones that are desirable in the near future:

Leaf nodes: Constant, Parm, Freq, Time, Grid, GaussNoise, RandomNoise, Spigot, FITSImage (and several other FITS interface nodes)

Unary operations: Exp, Log, Abs, Invert, Negate, Sqrt, Pow2(8), Sin, Cos, Tan, Acos, Asin, Atan, Cosh, Sinh, Tanh, Norm, Arg, Real, Imag, Conj, Ceil, Floor, Identity. See also Sect. 4.2.

Binary operations: (two children): Subtract, Divide, Pow, Mod, ToComplex(real,imag), Polar(ampl,phase). See also Sect. 4.2.

Accumulation: (one or more children): Add, Multiply, WSum, WMean. The last two need a vector of weights. See also Sect. 4.2.

Reduction nodes: Sum, Mean, Product, StdDev, Rms, Min, Max, NElements. These reduce a result along selected axes. The default is all axes, in which case the result is a scalar. See also Sect. 4.2.

Tensor operations: Composer, Selector, Paster. Tensor nodes are nodes with multiple vellsets in their Result (see Sect. 4.2).

Matrix operations: Transpose, ConjTranspose, MatrixMultiply, MatrixInvert22. The latter operates on 2x2 matrices only, which is sufficient for the RIME (see Sect. 8).

Flow control: ReqSeq, ReqMux, Sink, VisDataMux. They regulate the order in which their children get Requests, and which Result to pass on. They also synchronize the flow of Requests and Results in parallel trees.

Domain Control: ModRes, Resampler, CoordTransform. ModRes modifies the Request before it is passed on, Resampler modifies the Result itself. CoordTransform modifies the Request passed to its second child.

Flagging: ZeroFlagger, MergeFlags (see Sect. 5).

Solving: Condeq, Solver, Parm (see Sect. 6). For the moment, MeqTrees offers only a Levenberg-Marquard non-linear solver.

Visualization : All nodes, DataCollect, Inspector(=Composer) (see Sect. 3.4).

Coordinates: (mostly radio astronomy): UVW, LMN(radec,radeq0), AzEl(radec,xyz), RaDec(azel,xyz), LMRaDec(lm), ObjectRaDec(name), LST(domain,xyz), ParAngle(radec,xyz), LongLat(xyz). The vector *xyz* is an Earth position in IRTF coordinates.

Transforms: FFTBrick, UVInterpol (collectively known as UVBrick) (see Sect. 8).

User-definable nodes: PyNode, Functional, PrivateFunction. These allow the user to insert his own function, written in Python or C++, to read the Results from one or more child nodes, and to generate a customized Result.

A full description of these nodes is outside the scope of this paper. The present collection has an obvious bias towards radio astronomy. This may change as MeqTrees is used in other application areas. We envisage a core collection of nodes that offer basic functionality, surrounded by collections of more specialised nodes (and Python frameworks) for use in specific areas. The latter should be mostly contributed by users – we strive towards an Open Source developement model where everybody contributes, while a small core team keeps the mainline development on track. Some of the contributed nodes will eventually find their way into the core collection, while some of the present nodes will be moved out. In the meantime, the various types of user-definable nodes (such as the PyNode) allow experimentation with new node classes before they are implemented as a regular C++ node class. Obviously, we will have to solve the technical problem of linking such contributed nodes into MeqTrees with a minimum of fuss.

B Joining the MeqTrees family

MeqTrees is freely distributed under the terms of the GNU General Public License. The 1.1 release is available as of March 2010. The release consists of binary packages for the major Linux distros, so installation is relatively painless (and users of unsupported platforms always have the option of building from source.) An experimental Mac OSX port is also running, but not officially released. MeqTrees is natively multi-threaded to take full advantage of multi-core machines common today, and an MPI-based cluster version was developed at Oxford, and is currently being tested by Tony Willis at DRAO (Canada.)

Although the centre of MeqTrees development remains at ASTRON, a dedicated person has been appointed at Oxford to oversee the distribution to other institutes.

MeqTrees reuses several AIPS++/CASA modules and other tools. On the one hand such dependencies present a problem, because it makes it more difficult to distribute the package for multiple platforms. On the other hand, the reuse of other people's modules opens the way (eventually) to a belated fullfillment of the original dream of AIPS++: A World Processing Package, which offers common tools based on a Common Language (the RIME), while still having the capability to renew itself continuously through the inclusion of new ideas from the entire community of user-developers.

For further information on downloading and installing the software, please refer to the MeqTrees Wiki: `http://www.astron.nl/meqwiki`. You can also join the MeqTrees forum hosted at UCL: `https://great08.projects.phys.ucl.ac.uk/meqtrees/`.