*Astronomical Data Analysis Software and Systems XV* **P173**
*ASP Conference Series, Vol. XXX, 2005*
*C. Gabriel, C. Arviset, D. Ponz and E. Solano, eds.*

# Implementing Arbitrary Measurement Equations With The MeqTree Module

O.M. Smirnov & J.E. Noordam

*ASTRON, P.O. Box 2, 7990AA Dwingeloo, The Netherlands*

**Abstract.**

A Measurement Equation (ME) is a mathematical model of an instrument (e.g. a radio telescope) and the observed object(s). It can be used to predict the data measured with the instrument. In general, calibration of an observation (e.g. selfcal in radio astronomy) involves solving for parameters of an ME in some shape or form, by comparing observed and predicted data. Most calibration packages implicitly or explicitly implement some specific form of the ME. By sheer necessity, this involves making some simplifying assumptions about the instrument and the observation. As new instruments come on-line (WSRT LFFEs, LOFAR, etc.) we find ourselves pushing the limits of these assumptions. The MeqTree module provides a flexible system to implement MEs of arbitrary structure and complexity, and to solve for arbitrary subsets of their parameters.

## 1. Introduction

Measurement Equations (MEs) play a crucial role in the calibration of interferometric observations, since the latter always involve some sort of model fitting. The ME of an interferometer was derived in its closed form by Hamaker et al. (1996). For practical calibration purposes, however, the ME needs to be expressed in a computable form, employing a limited number of parameters. Calibration with an ME then follows a conventional model-fitting loop: initialize parameters from apriori guesses — use the ME to predict observables — compare to data — adjust parameters — repeat to a (hopefully) satisfactory fit.

All existing radiointerferometry calibration packages implicitly or explicitly implement some simplified and *fixed* form of the ME. The traditional *selfcal* algorithm is a good example. In its simplest formulation, it models the sky by a single point source, and the instrument by a single complex gain/phase term per each antenna. Despite the apparent simplicity of this model, it has served radioastronomers in good stead for over two decades!

However, a fixed ME makes it very hard or impossible to calibrate for effects unaccounted for by the software designer. Even older instruments can push the envelope of existing packages; future instruments such as LOFAR and SKA make the situation worse by magnitudes. These instruments will require sophisticated models with many ME parameters which current packages are simply not equipped to handle. Moreover, at the moment we can only guess

which formulations of the ME will actually allow for calibration, so we will need to continue experimenting with different MEs as these instruments come on-line. Clearly, what is needed is a *toolkit* for constructing and fitting MEs.

## 2. The MeqTree Concept

The *MeqTree* (Measurement Equation Tree) module provides such a toolkit. Any model or ME is, in the final analysis, nothing more than a mathematical expression, and as such can be represented by a tree.

MeqTrees are constructed out of *MeqNodes*. Nodes receive *MeqRequests* from their parents and pass them on to their children, get *MeqResults* in return, perform some operation on them, and return the result of that to their parents. A MeqRequest generally defines a domain and gridding in $N$-D space (for example, time-frequency space), and a MeqResult represents a sampling of some function over that domain, *with optional perturbed values.*

A typical MeqNode performs some mathematical operation on the results of its children. MeqTrees provide a large collection of node classes for most mathematical operations, and new node classes may be added by writing a (usually simple) C++ class. A vital feature of MeqTrees is that tensors are represented in an elegant and economical manner. This allows for a mathematically complete model of polarization effects (Hamaker 2000), which have up to now been notoriously difficult to understand and calibrate for.

Thus, we can build a tree representing any function or model or ME, and evaluate that model simply by giving a request to the root node of the tree, and waiting for a result to come back. *If you can write it down as an expression, you can predict it with a MeqTree.*

### 2.1. Solving For MEs

Building an ME is just half the job, what about fitting it to the observed data? MeqTrees provide some specialized node classes to facilitate this:

**MeqParm** leaf nodes represent ME parameters. These can be atomic, or can in turn be functions of, e.g., frequency and time (Mevius et al. 2006, this volume). Any subset of MeqParms may be designated as solvable; solvable MeqParms will return perturbed values along with their main value.

**CondEq** (Conditioning Equation) nodes compare the results of their two children — e.g., a predict subtree on one side, and observed data supplied by a *Spigot* leaf on the other — and convert perturbed values into numeric derivatives.

A **Solver** node does the actual fitting. It collects residuals and derivatives from all its CondEq children, builds a Jacobian matrix, and executes one step of the Levenberg-Marquant minimization algorithm. This results in a set of *parameter updates*, which are then sent back up to the solvable MeqParms. The cycle is repeated until a satisfactory fit is reached, or for some maximum number of iterations. The Solver node then returns a result.

**Control nodes** such as the **ReqSeq** (Request Sequencer) provide flow control. For example, multiple Solvers may be fired in series to solve for different sets of paremeters in turn, a subtract branch can be activated, subtracting the fitted model from the data, a correction branch may apply derived corrections to the residuals, etc.

**Sink** nodes are found at the root of trees. Sinks write their childrens' results out to disk (e.g., to an AIPS++ MS). Depending on the structure of the tree, these results may represent predicted visibilities, residuals, corrected data, etc.

In this way, MeqTrees allow one to solve for arbitrary parameters of any ME. To elaborate on a previous statement, *if you can write it down as an expression, you can solve for it using MeqTrees.* (Of course, the data needs to provide enough constraints to solve for the given parameters...)

## 3. The MeqTree Module

The MeqTree module is implemented by a software package loosely called *MeqTimba*. MeqTimba consists of the following components:

- A computational *kernel*, mostly implemented in C++ (and using libraries from AIPS++, FFTW, etc.) The kernel provides all MeqNode classes, implements basic facilities for creating MeqNodes and connecting them into trees, and provides a low-level interface for controlling MeqTrees.
- A set of *I/O agents* to feed the kernel with data and to dispose of the results. The current set includes agents for reading/writing AIPS++ Measurement Sets (MSs), and also agents for pipelining data over the network.
- A GUI called the *MeqBrowser*, implemented in Python/PyQt. The browser provides kernel control (bulding trees, attaching MSs, running trees) and visualization. The browser also provides an interface to the stepwise debugging and tree profiling functions of the kernel.
- A Python-based Tree Definition Language (TDL). TDL scripts can be loaded and run by the browser, which feeds them to the kernel to construct and run trees. TDL allows one to define MeqTrees in high-level terms. The typical turnaround time for modifying and re-executing a TDL-described ME is measured in seconds. This makes the system remarkably easy to "play" with.

The design of MeqTimba has a number of important highlights:

**Policy-free:** The kernel is (almost) entirely policy-free; it operates with very basic concepts (nodes and trees), with mimumum assumptions about the problem domain. All policy — and thus the problem domain, complete with MEs, data formats, etc. — is defined from the scripting side via TDL. This makes the system eminently adaptable to new instruments and problems (including those outside radioastronomy per se).

**Data transparency**: each node maintains a *state record* that can be examined from the browser. No MeqTree is ever a black box, and the user can examine the behaviour of the model to any level of depth or detail. Node states can also be published into the browser as a tree runs, providing an execution history.

**Visualize everything:** the browser provides built-in tools for visualizing results and other data structures. Even more importantly, TDL scripts can define *bookmarks* that provide "canned" views of the tree which the user can access with a couple of mouse clicks. In fact, trees can contain *visualization branches* that compute derived quantities that are not used in calibration per se, but do provide additional insight into calibration fidelity.

**Naturally parallelizable:** MeqTimba has been designed with parallelization in mind. The current version of the kernel is single-threaded (although the

browser can control it remotely over a network), but future development will allow for trees to execute in parallel and be distributed across a cluster. The built-in tree profiler will aid in determining optimal parallelization strategies.

## 4. Current Status And Future Directions

Over the past year, MeqTimba has been gradually exposed to WSRT data. One current project (Brentjens 2005) involves custom trees for high-dynamic-range calibration of a complex field (3C343) which is hard to deal with using traditional selfcal due to the presence of off-axis bright sources. Performance on par with existing packages has been demonstrated, and current work aims to "go where no package has gone before" and calibrate for finer effects. A second project aims to provide a set of canned *central point source* trees for online and offline processing of full-polarization WSRT calibrator observations. On a completely different tack, Willis (2005) has been using MeqTrees to simulate observations with the projected CLAR telescope, characterized by a time-variable beam, and show that such a beam may be successfully calibrated for.

In the near future, we will be applying MeqTrees to data from the new WSRT Low-Frequency Front Ends (LFFEs), giving a preview of the LOFAR sky. The WHAT project – a prototype LOFAR station linked up with the WSRT – will provide a very interesting test for MeqTrees, being one of the first examples of a truly *heterogenous* (phased array vs. dish) interferometer.

MeqTrees will play an integral role in LOFAR calibration. The LOFAR *Local Sky Model* (Smirnov & Noordam 2003; Nijboer et al. this volume) will use MeqTrees to represent sky sources; we see this as the only way to get a handle on the complex and overcrowded sky seen by LOFAR. The ionosphere is critical at low frequencies, and will also require a relatively sophisticated model (Noordam 2005). Here again MeqTrees are expected to play a vital role.

## References

Brentjens, M. 2005, presentation at SKA WFI Workshop, Dwingeloo[1]

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

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

Mevius, M. 2006, this volume, [P.78]

Nijboer, R.J., Noordam, J.E. & Yatawatta, S. 2006, this volume, [P.57]

Noordam, J.E. 2005, presentation at SKA WFI Workshop, Dwingeloo[2]

Smirnov, O.M. & Noordam, J.E. 2003, in ASP Conf. Ser., Vol. 314, ADASS XII, ed. F. Ochsenbein, M. Allen, & D. Egret (San Francisco: ASP), 18

Willis, A.G. 2005, presentation at SKA WFI Workshop, Dwingeloo[3]

[1] `http://www.skatelescope.org/pages/news/SKA_WFI2005/scd.pdf`

[2] `http://www.skatelescope.org/pages/news/SKA_WFI2005/MIM%20may%20202005.ppt`

[3] `http://www.skatelescope.org/pages/news/SKA_WFI2005/wfi_talk_full.pdf`