Re-factoring of the `calibrater` related classes

S. Bhatnagar and G. Moellenbrock
NRAO, Socorro

October 02, 2004

Abstract

The general form of the equation representing the measurement by an aperture synthesis telescope along with the corruptions of the observations by various errors is $\boldsymbol{V}^{obs}_{ij} = \mathsf{J}_i \otimes \mathsf{J}^*_j \boldsymbol{V}^{True}_{ij}$ where $\boldsymbol{V}^{obs}$ and $\boldsymbol{V}^{True}$ are the observed and the true visibilities, J_i is the antenna based Jones matrix representing the corruptions and $\otimes$ is the matrix outer product operator. Given a model visibility, it is possible to solve for the J_is by minimizing a Goodness-of-fit (GOF) criterion (e.g. χ^2) using one of the standard function minimization algorithms. Software implementation of such a solver can be decomposed into three distinct objects: (1) the `VisEquation` object which implements the above linear equation given a set of corrupting Visibility Jones (VJ) matrices, (2) the `VisJones` object which represents the detailed physics of the corruptions via the VJ matrices, and (3) the `Solver` object which, given a `VisEquation` and a `VisJones` object, implements the numerical algorithm to solve for the Jones matrices.

In this memo, we discuss the re-factoring of the existing implementation of the `VisEquation` and the `VisJones` classes to create the new `Solver` class. This re-factoring will encourage and enable code reuse as well as make it easier to implement new more complicated solvers in AIPS++, built on existing solvers. This will also be a requirement for the near future scientific computing we need to do. We discuss the implementation of a solver to solve for the antenna based pointing errors using the re-factored `VisEquation`, `VisJones`, and the `Solver` classes.

1 Introduction

The measurement equation for a connected element aperture synthesis telescope, along with the various corruptions due to the imperfections in the electronics, mechanical control of the antennas, atmosphere, ionosphere, etc. can be elegantly described by a linear equation of the type:

$$\boldsymbol{V}^{obs} = \mathsf{J}_{ij} \boldsymbol{V}^{True} + \boldsymbol{N} \tag{1}$$

where $\boldsymbol{V^{True}}$ and $\boldsymbol{V^{obs}}$ are the true (uncorrupted) and the observed visibility vectors, J_{ij} is the Muller matrix representing the corruption of the measurement from the baseline $i-j$ and $\boldsymbol{N}$ is the noise vector (Hamaker et al. 1996; Sault et al. 1996). Wherever J_{ij} can be factored into antenna based Jones matrices J_{i} and J_{j}, it can be expressed as $\mathsf{J}_{\mathsf{i}} \otimes \mathsf{J}^*_{\mathsf{j}}$ (where $\otimes$ is the matrix outer product operater). Such a description forms the mathematical basis of the imaging and calibration modules in AIPS++ (Cornwell 1995a,b; Noordam 1995, 1996).

Given a reasonably accurate model visibility ($\boldsymbol{V^M}$), the measurement equation represents an over determined set of equations which can be solved for the antenna based Jones matrices by defining a Goodness-of-fit (GOF) criterion and minimizing it with respect to the parameters of the Jones matrices. The additive noise in the visibility domain can be shown to have a gaussian probability distribution. The natural GOF criterion is therefore the χ^2. The appropriate solver for such a problem can be any of the numerous numerical least-squares minimization algorithms. Such algorithms are invariably iterative in nature. The basic numerical computations needed in each iteration are the computation of the difference between $\boldsymbol{V^{obs}}$ and $\mathsf{J}_{\mathsf{ij}}\boldsymbol{V^M}$, and the derivative of $\mathsf{J}_{\mathsf{ij}}\boldsymbol{V^M}$. The solver then updates the current estimates of the parameters being solved for at each iteration and terminates the iterations when convergence is achieved (or some other user defined termination criterion is satisfied).

The calibration module of AIPS++ implements the measurement equation in the `VisEquation` class which, apart from other services, provides the services to compute the χ^2 and its derivative via the `gradientsChiSquared()` method. This method internally uses `apply()` and `applyGradients()` methods of the `VisJones` class to apply the Jones matrix and its derivative to the visibility (i.e., compute $\mathsf{J}_{\mathsf{ij}}\boldsymbol{V^M}$ and $\nabla\mathsf{J}_{\mathsf{ij}}\boldsymbol{V^M}$). The `VisJones::solve()` method implements the iterative solver, which calls the `gradientsChiSquared()` method of the `VisEquation` class (which calls `apply()` and `applyGradients()` services of the `VisJones` back again). This circular calling of methods between the `VisJones` and the `VisEquation` classes is an indication that some re-factoring is required to de-couple these classes. Separating the solver from the `VisJones` class will in fact achieve this de-coupling and eliminate the circular calling sequences. In section 2 we do an Object Oriented analysis to find the conceptual objects involved in building a generic solver for the Jones matrices. A similar analysis has been done by Cornwell (1995a) earlier. In section 3 we describe in detail the re-factoring of the `VisEquation` and `VisJones` classes. The existing methods are slightly modified to provide services required by the generalized `Solver` class. Section 4 contains the detailed design of the `Solver` class and section 5 describes the implementation of a new solver to solve for the antenna based pointing offsets.

2 Object Oriented analysis of a generic solver

Equation 1 describes the linear relation between the measured and the true quantities and encapsulates the physical description of the measurement process. $\boldsymbol{V^{obs}}$ represents the observation of the uncorrupted data ($\boldsymbol{V^{True}}$) by an imperfect instrument (the telescope). $\mathsf{J_{ij}}$ encapsulates the physics of the imperfections of the telescope, which in general is a product of a number of matrices each representing a different source of imperfection (antennas, feeds, correlater, etc.). The obvious real-life objects that one can then identify are (1) an object which gives access to the observed data (and the model data), and (2) an object to represent the telescope imperfections in the form of Muller matrices ($\mathsf{J_{ij}}$). These objects, in AIPS++, are the `VisSet` and `VisJones` tree of classes. However note that the `VisJones` object holds the Jones matrices for all the antennas (all the $\mathsf{J_i}$s) and provides services for the operation to construct the Muller matrices ($\mathsf{J_{ij}} = \mathsf{J_i} \otimes \mathsf{J_j^*}$) as well. The measurement equation is implemented in the `VisEquation` classes, which encapsulates the sequence of corrupting elements that the signal passes through in the telescope. The `VisSet`, `VisEquation`, and the `VisJones` classes then represent the data, and the physics of the measurement and the corruptions respectively.

The process of calibrating the instrument involves applying $\mathsf{J_{ij}}^{-1}$ to the observed data. Corruptions which are not known a priori need to be numerically solved for using controlled observations with the instrument. In general $\mathsf{J_{ij}}$ is singular and requires an iterative non-linear algorithm to solve for them (e.g. χ^2 minimization algorithms). The basic requirements for such a solver are:

1. **Computation of the residuals:** Given a model for the true measurements, this is the difference between the actual measurements and the model data corrupted by the current estimates of the corruptions ($\boldsymbol{V^{obs}} - \mathsf{J_{ij}}\boldsymbol{V^{M}}$). When $\boldsymbol{V^{M}}$ and $\mathsf{J_{ij}}$ accurately represents the true observations and the instrumental corruptions, this quantity will be statistically consistent with the noise model. Note that this may require the operators to construct the Muller matrix from a pair of Jones matrices and the operator to apply the Muller matrix to the data. For most corruptions, these operators are the matrix outer-product operator and the matrix multiplication operator respectively, but is not always the case (e.g. for antenna pointing offsets, both these operators are the convolution operator ($\boldsymbol{V^{obs}} = \mathsf{E_i} \star \mathsf{E_j^*} \star \boldsymbol{V^{M}}$)).

2. **Computation of $\nabla \mathsf{J_{ij}} \boldsymbol{V^{M}}$:** In an iterative solver, starting from a reasonable initial guess, the update direction at each iteration is given by $\nabla\chi^2$. It is easy to see that given the observed data and $\nabla \mathsf{J_{ij}} \boldsymbol{V^{M}}$, using the derivative chain rule, the update direction can be computed. Note that since $\mathsf{J_{ij}}$ can be composed of two independent Jones matrices and in general each parameterized by physical parameters, computation of $\nabla \mathsf{J_{ij}}$ with respect to the parameters will be done using the derivative chain rule.

Given these very generic services, one can implement a generic iterative solver which is not specific to the detailed physics of the measurement and the corruptions. This leads to a fourth distinct object, we call the `Solver` object. This encapsulates the numerical details of the iterative algorithms to estimate $\mathsf{J_{ij}}$. Separating the numerical algorithm into an independent class makes a clear distinction between the software representation of the physics of the measurement/corruptions and the numerical solver, independent of the physics. This also is the only place where the `VisEquation` and the `VisJones` classes are coupled, eliminating the back-n-forth call sequences between these classes (see the next section and Fig. 1).

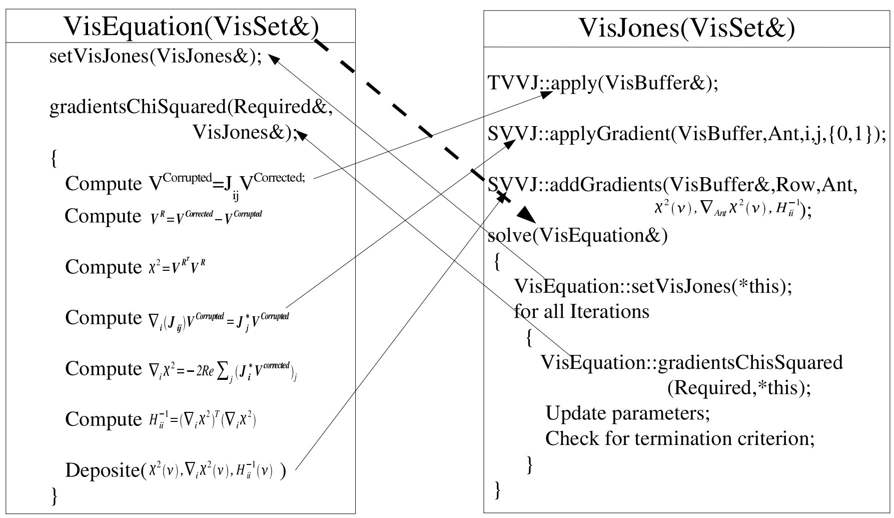

Figure 1: The sequence of events triggered by the iterative solver for multiplicative Jones matrices in the current implementation. The thick dashed line indicates the instantiation of the `VisEquation` object, which is then passed as an argument to the `VisJones::solve()` method. The sequence of calls initiated from the `solve()` method are indicated by the continuous lines.

3 Re-factoring the `VisEquation` and the `VisJones` classes

The `VisEquation` class currently contains the `gradientsChiSquared`() method. This takes a `VisJones` object as an argument, and computes the residual visibilities, the χ^2, $\nabla\chi^2$ and the inverse of the diagonal elements of the Hessian (H_{ii}^{-1}). To compute $\mathsf{J_{ij}}\boldsymbol{V^M}$, it calls the `VisJones::apply()` method, which is responsible for computing the Muller matrix ($\mathsf{J_{ij}}$) from the individual Jones matrices ($\mathsf{J_i}$s). It also computes $\nabla\mathsf{J_{ij}}\boldsymbol{V^M}$ and calls `VisJones::applyGradient()` which is responsible for evaluating the gradient (Eq. 2) as $\mathsf{J_i}\boldsymbol{V^M}$. The χ^2, $\nabla\chi^2$, and H_{ii}^{-1} *per channel* are accumulated in `VisEquation` and it again makes a call to `VisJones::addGradients()` to deposit these quantities in the `VisJones` object where they are required by the iterative numerical solver. The actual iterative solver is implemented in `VisJones::solve()`, which makes a call to `VisEquation::gradientsChiSquared()` and triggers the sequence of calls described above. The sequence of events is shown graphically in Fig. 1.

For the purpose of solving for the Jones matrices the basic services required are:

1. `VisBuffer& apply(VisBuffer& VB, VisMuller& VM)`

   Given $\boldsymbol{V}^{\boldsymbol{Corrected}}$ and a Muller matrix J_{ij}, compute

   $$\boldsymbol{V}^{\boldsymbol{Corrupted}} = \mathsf{J}_{\mathsf{ij}} < \texttt{OP1} > \boldsymbol{V}^{\boldsymbol{Corrected}}$$

   This essentially implements the operator `<OP1>` and possibly should be a global function implemented as part of the `VisMuller` implementation (but not as a method of it). For most Muller types, this will be the inner product operator and will be evaluated as
   `VB.corrupted = apply(VB.corrected,VM)`.

2. `VisMuller& apply(VisJones &Ji, VisJones &Jj)`

   Given two Jones matrices, compute the equivalent Muller matrix as:

   $$\mathsf{J}_{\mathsf{ij}} = \mathsf{J}_{\mathsf{i}} < \texttt{OP2} > \mathsf{J}_{\mathsf{j}}^{*}$$

   This implements the operator `<OP2>`, which is the matrix outer product for most Jones types. This probably should be implemented as a global function as part of the Jones matrix class.

   Should these be global functions?

3. `VisBuffer& VisEquation::getResiduals(VisMuller &Jij)`

   This computes

   $$\boldsymbol{V}^{\boldsymbol{R}} = \boldsymbol{V}^{\boldsymbol{obs}} - \mathsf{J}_{\mathsf{ij}}\boldsymbol{V}^{\boldsymbol{M}}$$

   The second term on the right hand side will be computed using the service of item 1 as `this->residual = this->observed()-apply(Jij,this->model())`.

4. `VisBuffer& VisEquation::applyGradients(VisBuffer& VB, VisMuller& Jij,int i)`

   This computes

   $$\frac{\partial \mathsf{J}_{\mathsf{ij}}}{\partial \mathsf{J}_{\mathsf{i}}}\boldsymbol{V}^{\boldsymbol{M}} = \mathsf{J}_{\mathsf{j}}{}^{*}\boldsymbol{V}^{\boldsymbol{M}} \tag{2}$$

   Partial derivative of J_{ij} is computed by `Jij.gradient(i)`, which for most Jones types, returns $\mathsf{J}_{\mathsf{j}}^{*}$. The operator in item 1 is then used to compute the final result as `apply(Jij.gradient(i),this->model())`.

Most of these services exist in the current implementation - though sometimes not as distinct methods. The `VisMuller` class does not exist but in practice will be a light weight class, more for keeping the code in tune with the mathematical formulation and hence more understandable. It will most likely be a `friend` of the `VisJones` class and hold the indices of the constituent Jones matrices. For most of the Jones types, the `VisMuller::apply(VisJones& Ji, VisJones& Jj)` will return `outerproduct(Ji,conj(Jj))` and `VisMuller::apply(VisMuller& VM, VisBuffer& VB)` will return the inner product of `VM` and `VB`. Similarly `VisMuller::gradient(int i)` for normal Jones types just returns `conj(this->otherAntJones(i))`. This re-factoring is unlikely to impact the run time performance or memory requirement in any adverse way.

4 The `Solver` class

The `Solver` class will provide interface to the various minimization algorithms. E.g., the pseudo code for a simple Conjugate Gradient solver may look like this:

```
Solver::ConjugateGradient(VisEquation &VE, VisJones &J)
 {
    J0 = Make_Initial_Guess(VE,J);
    for all iterations
     {
       VR = VE.getResiduals(J0); // The residuals
       ChiSq = innerproduct(VR,VR);
       gradV = VE.getGradient(J0);
       gradChiSq = -2*real(innerproduct(VR,gradV));
       hDiag = 2*real(innerproduct(gradV,gradV));
       J0 = J0 + gradChiSq/hDiag;
       Check_for_termination_criterion;
     }
    if (Converged) J = J0;
 }
```

This separation of the physical representation of the problem and numerical solver has the advantage that, e.g. the implementation of a new type of Jones matrix which is deemed solvable involves implementation of a specialization of the existing `VisJones` class without the need to know the details of the `VisEquation` and the `Solver` classes. Similarly, implementing a more sophisticated numerical solver, possibly using an external numerical library like the GNU Scientific Library (M. Galassi et al. 2002), involves specialization (or modification) of the `Solver` class alone. Both these scenarios are most likely to be a requirement in the very near future and will encourage and enable easy code re-use.

5 Implementation of the E-Jones solver

The measurement equation for the corruptions due to the antenna based pointing offsets is given by

$$\boldsymbol{V}^{\boldsymbol{Corrupted}} = \mathsf{E_i} \star \mathsf{E_j^*} \star \boldsymbol{V}^{\boldsymbol{Corrected}} \tag{3}$$

where $\mathsf{E_i}$ is the illumination pattern for antenna i and $\star$ is the convolution operator (Bhatnagar et al. 2004). For the design of the `VisEquation`, `VisJones` and the `Solver` classes discussed in this memo, implementing a solver for $\mathsf{E_i}$s requires a specialization of the `VisJones` class, we call the `EJones` class. In such a specialization the `apply(VisEquation&, VisMuller&)` and `apply(EJones&, EJones&)` methods will implement the convolution of $\mathsf{E_{ij}} = \mathsf{E_i} \star \mathsf{E_j^*}$ with the visibility and the convolution of $\mathsf{E_i}$ and $\mathsf{E_j}$ respectively. The `VisMuller::gradient()` method will also need to be overloaded in a specialization to perform the convolution of $\nabla \mathsf{E_{ij}}$ with the corrected visibilities.

The computation of the model visibilities as a function of the observed (u, v) is done by the convolutional de-gridding algorithm. This essentially re-samples the gridded visibility at the measured (u, v) values using a gridding convolution function. Normally, this function is same for all the baselines. The convolution of $\mathsf{E_{ij}}$ with $\boldsymbol{V}^{\boldsymbol{Corrected}}$ can be achieved by modifying the AIPS++ gridder services provided by the `FTMachine` class of the `imager` module to use $\mathsf{E_{ij}}$ while de-gridding the baseline $i - j$. Note that this implies that the de-gridder potentially uses a different gridding convolution function for each baseline.

The `apply(VisEquation&, VisMuller&)` and the `VisMuller::gradient()` methods will then use the modified `FTMachine::put()` method and supply it with $\mathsf{E_{ij}}$ and $\nabla \mathsf{E_{ij}}$ functions for baseline $i - j$ to compute the corrupted visibilities and its derivative respectively.

References

Bhatnagar, S., Cornwell, T. J., & Golap, K. 2004, Solving for the antenna based pointing errors, Tech. rep., EVLA Memo 84

Cornwell, T. J. 1995a, The Generic Interferometer: I Overview of Calibration and Imaging, Tech. rep., AIPS++ Note 183

Cornwell, T. J. 1995b, The Generic Interferometer: II Image Solvers, Tech. rep., AIPS++ Note 184

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

M. Galassi, M., Davies, J. T., Gough, B., et al. 2002, GNU Scientific Library Reference Manual - Second Edition (Network Theory Limited)

Noordam, J. 1995, Some practical aspects of the matrix-based Measurement Equation of a generic radio telescope, Tech. rep., AIPS++ Note 182

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

Sault, R. J., Hamaker, J. P., & Bregman, J. D. 1996, A&AS, 117, 149