Revised Schedule

9:00 ~ 10:30 **session 1**
10:30 ~ 11:00 coffee
11:00 ~ 12:30 **session 2**
12:30 ~ 13:30 lunch
13:30 ~ 15:00 **session 3**
15:00 ~ 15:30 coffee
15:30 ~ 17:30 **session 4**
17:30 ~ 9:30 beer & homework

ME1: Measurement Equation Of a (Polarized) Point Source

Objectives:
- **Aligning our terminology!**
- Mapping your existing intuition about interferometry onto ME concepts
- Implementing some MEs using MeqTrees.

The Measurement Equation: Putting the "Meq" into MeqTrees!

- The Measurement Equation tells you what you can expect to observe with an interferometer, given a sky and the properties of your instrument.
- Absolutely crucial for simulating and calibrating the next generation of radio telescopes; everything literally revolves around it.

Therefore: no-one gets any beer tonight until we achieve full harmony and understanding!

Survey Results...

1. Radio interferometry...
   [1] heard of it
   [11] basic knowledge
   [6] do it all the time
   [3] I was doing it when Oleg was in diapers

Therefore...
...you know practically everything about the Measurement Equation!

On The Other Hand...

3. The Measurement Equation...
   - [1] never heard of it
   - [2] heard of it
   - [10] know what it looks like, never actually used it
   - [6] know it pretty well
   - [2] I use Jones matrices to do my taxes

- Conclusion:
  The ME is no longer one of these unknown knowns -- the things we don't know we know -- that Donald Rumsfeld didn't know he knew.

A Wafer-Thin Slice of Physics: EM Field Propagation

Pick an *xyz* frame with *z* along the direction of propagation.

The EM field can be described by the complex vector $\vec{e} = \begin{pmatrix} e_x \\ e_y \end{pmatrix}$

The fundamental assumption is **LINEARITY**:

1. Propagation through a medium is linear

   ⇒ can be fully described by a 2x2 complex matrix:

   $$\vec{e}\,' = \boldsymbol{J}\vec{e} \quad \text{i.e.} \quad \begin{pmatrix} e'_x \\ e'_y \end{pmatrix} = \begin{pmatrix} ❑ & ❑ \\ ❑ & ❑ \end{pmatrix} \begin{pmatrix} e_x \\ e_y \end{pmatrix}$$

2. Receptor voltages $\vec{v} = \begin{pmatrix} v_x \\ v_y \end{pmatrix}$ are also linear w.r.t. $\vec{e}$

   $$\Rightarrow \quad \vec{v} = \boldsymbol{J}\vec{e}$$

A Wafer-Thin Slice of Physics: Jones Matrices

* $\boldsymbol{J}$ is called a *Jones matrix.*

* $\boldsymbol{J}$ is obviously cumulative:

$$\vec{v} = \boldsymbol{J}_n(\boldsymbol{J}_{n-1}(\ldots\boldsymbol{J}_1\vec{e})) = (\prod_{i=n}^{1}\boldsymbol{J}_i)\vec{e} = \boldsymbol{J}\vec{e}$$

where $\boldsymbol{J}_1 \ldots \boldsymbol{J}_n$ describes the full signal path.

* Do remember that matrices, in general, do not commute.

NB: What if something is non-linear?..

...we can also write down an equation: $\vec{v} = \mathcal{J}(\vec{e})$

...but this is to be avoided if at all possible.

A Wafer-Thin Slice of Physics: Correlations & Visibilities

An interferometer measures *correlations* btw voltages $\vec{v}_p, \vec{v}_q$:

$$V_{xx} = \langle v_{px} v^*_{qx} \rangle, V_{xy} = \langle v_{px} v^*_{qy} \rangle, V_{yx} = \langle v_{py} v^*_{qx} \rangle, V_{yy} = \langle v_{py} v^*_{qy} \rangle$$

It is convenient to represent these as a matrix product:

$$\boldsymbol{V}_{pq} = \langle \vec{v}_p \vec{v}_q^{\dagger} \rangle = \langle \begin{pmatrix} v_{px} \\ v_{py} \end{pmatrix} (v^*_{qx} \; v^*_{qy}) \rangle = \begin{pmatrix} V_{xx} & V_{xy} \\ V_{yx} & V_{yy} \end{pmatrix}$$

($\langle\,\rangle$: time/freq averaging; $\dagger$: conjugate-and-transpose)

$\boldsymbol{V}_{pq}$ is also called the *visibility matrix.*

Now let's assume that all radiation arrives from a single point, and designate the "source" E.M. vector by $\vec{e}$.

A Wafer-Thin Slice of Physics: The M.E. Emerges

Antennas p,q then measure: $\vec{v}_p = \boldsymbol{J}_p \vec{e}, \quad \vec{v}_q = \boldsymbol{J}_q \vec{e}$

where $\boldsymbol{J}_p, \boldsymbol{J}_q$ are Jones matrices describing the signal paths from the source to the antennas.

Then $\boldsymbol{V}_{pq} = \langle (\boldsymbol{J}_p \vec{e})(\boldsymbol{J}_q \vec{e})^\dagger \rangle = \langle \boldsymbol{J}_p (\vec{e}\,\vec{e}^\dagger) \boldsymbol{J}_q^\dagger \rangle = \boldsymbol{J}_p \langle \vec{e}\,\vec{e}^\dagger \rangle \boldsymbol{J}_q^\dagger$

(making use of $(\boldsymbol{AB})^\dagger = \boldsymbol{B}^\dagger \boldsymbol{A}^\dagger$, and assuming $\boldsymbol{J}_p$ is constant over $\langle\rangle$)

The inner quantity is known as the *source coherency*:

$$\boldsymbol{B} = \langle \vec{e}\,\vec{e}^\dagger \rangle = \frac{1}{2} \begin{pmatrix} I+Q & U \pm iV \\ U \mp iV & I-Q \end{pmatrix} \leftrightarrow (I, Q, U, V)$$

which we can also call the *source brightness.* Thus:

$$\boldsymbol{V}_{pq} = \boldsymbol{J}_p \boldsymbol{B} \boldsymbol{J}_q^\dagger$$

And That's The Measurement Equation!

$$\boldsymbol{V}_{pq} = \boldsymbol{J}_p \boldsymbol{B} \boldsymbol{J}_q^\dagger$$

- Or in more pragmatic terms:

$$\overbrace{\begin{pmatrix} XX & XY \\ YX & YY \end{pmatrix}}^{\text{measured}} = \overbrace{\begin{pmatrix} j_{xx(p)} & j_{xy(p)} \\ j_{yx(p)} & j_{yy(p)} \end{pmatrix}}^{\boldsymbol{J}_p} \overbrace{\frac{1}{2} \begin{pmatrix} I+Q & U+iV \\ U-iV & I-Q \end{pmatrix}}^{\text{source}} \overbrace{\begin{pmatrix} j^*_{xx(q)} & j^*_{yx(q)} \\ j^*_{xy(q)} & j^*_{yy(q)} \end{pmatrix}}^{\boldsymbol{J}_q^\dagger}$$

- NB: it is also possible to write the ME with a circular polarization basis (RR, LL, etc.) We'll use linear polarization throughout.

Accumulating Jones Terms

If $\boldsymbol{J}_p, \boldsymbol{J}_q$ are products of Jones matrices:

$$\boldsymbol{J}_p = \boldsymbol{J}_{pn} \cdots \boldsymbol{J}_{p1}, \quad \boldsymbol{J}_q = \boldsymbol{J}_{qm} \cdots \boldsymbol{J}_{q1}$$

Since $(\boldsymbol{AB})^\dagger = \boldsymbol{B}^\dagger \boldsymbol{A}^\dagger$, the M.E. becomes:

$$\boldsymbol{V}_{pq} = \boldsymbol{J}_{pn} \ldots \boldsymbol{J}_{p2} \boldsymbol{J}_{p1} \boldsymbol{B} \boldsymbol{J}_{q1}^\dagger \boldsymbol{J}_{q2}^\dagger \ldots \boldsymbol{J}_{qm}^\dagger$$

or in the "onion form":

$$\boldsymbol{V}_{pq} = \boldsymbol{J}_{pn} (\ldots (\boldsymbol{J}_{p2} (\boldsymbol{J}_{p1} \boldsymbol{B} \boldsymbol{J}_{q1}^\dagger) \boldsymbol{J}_{q2}^\dagger) \ldots) \boldsymbol{J}_{qm}^\dagger$$

Jones' Anatomy

- $\boldsymbol{J}_p$ is “cumulative”: more effects correspond to additional multiplicative Jones terms.
- Therefore, the “total” $\boldsymbol{J}_p$ is a matrix product of a *“Jones chain”* of individual effects:

  $$\boldsymbol{J}_p = \boldsymbol{J}_{pn} \boldsymbol{J}_{pn-1} \ldots \boldsymbol{J}_{p1}$$
- The order of the $\boldsymbol{J}$ terms corresponds to the physical order of the effects. In general, the matrices don't commute!

Why is this great?

- A complete and mathematically elegant framework for describing all kinds of signal propagation effects.
- ...including those at the antenna, e.g.:
  - beam & receiver gain
  - dipole rotation
  - receptor cross-leakage
- Effortlessly incorporates polarization:
  - think in terms of a ***B*** matrix and never worry about polarization again.
- Applies with equal ease to heterogeneous arrays, by using different Jones chains.

Why is this even greater?

- Most effects have a very simple Jones representation:

$$\text{gain: } \boldsymbol{G} = \overbrace{\begin{pmatrix} g_x & 0 \\ 0 & g_y \end{pmatrix}}^{\textit{diagonal matrix}} \quad \text{phase delay: } \overbrace{\begin{pmatrix} e^{-i\phi} & 0 \\ 0 & e^{-i\phi} \end{pmatrix}}^{\textit{scalar matrix}} \equiv e^{-i\phi}$$

$$\text{rotation: } \begin{pmatrix} \cos\gamma & -\sin\gamma \\ \sin\gamma & \cos\gamma \end{pmatrix} \equiv \text{Rot}(\gamma) \quad \textit{(rotation matrix)}$$

$$[\ \text{e.g. Faraday rotation: } \boldsymbol{F} = \text{Rot}(\frac{RM}{\nu^2})\ ]$$

$$\text{receptor cross-leakage: } \boldsymbol{D} = \begin{pmatrix} 1 & d \\ -d & 1 \end{pmatrix} \quad (\text{or Rot}(d)?)$$

Three Layers Of Intuition

- **Physical:** e.g. beam gain, parallactic angle
  - beam pattern of X and Y dipoles different, causes polarization of off-center sources
  - P.A. rotates polarization angle
- **Geometrical:** stretching, rotation
  - do not commute...
- **Mathematical:** matrix properties

$$\boldsymbol{G} = \begin{pmatrix} g_x & 0 \\ 0 & g_y \end{pmatrix} \text{ and } \boldsymbol{P} = \begin{pmatrix} \cos\gamma & -\sin\gamma \\ \sin\gamma & \cos\gamma \end{pmatrix} \text{ do not commute;}$$

$$\text{m.e. is: } \boldsymbol{V}_{pq} = \boldsymbol{G}_p \boldsymbol{P}_p \boldsymbol{B} \boldsymbol{P}_q^\dagger \boldsymbol{G}_q^\dagger$$

ME ME ME

- The general formulation above is *"<u>The</u> Measurement Equation"* *(of a generic radio interferometer...)*
- When we want to simulate a specific instrument, we put specific Jones terms into the ME, and derive *<u>a</u> measurement equation* for that instrument.
- We then *implement* that specific m.e. in software (e.g. with MeqTrees)
- Existing packages implicitly use specific m.e.'s of their own.

Observing a point source with a perfect instrument

Even w/o instrumental effects, we still have geometry, so:

$$\boldsymbol{V}_{pq} = \boldsymbol{K}_p \boldsymbol{B} \boldsymbol{K}_q^\dagger$$

$\boldsymbol{K}_p$ is the *phase shift* term, a ***scalar*** Jones matrix:

$$\boldsymbol{K}_p = \begin{pmatrix} e^{-i\phi_p} & 0 \\ 0 & e^{-i\phi_p} \end{pmatrix} \equiv e^{-i\phi_p}$$

Antenna phase ϕ_p accounts for the pathlength difference:

$$\phi_p = 2\pi(u_p l + v_p m + w_p(n-1))$$

where u_p, v_p, w_p are *antenna coordinates* (in wavelengths),

and l, m, n are the direction cosines for the source.

$(n=\sqrt{1-l^2-m^2}$, for "small" fields $n \to 1$.)

The (familiar?) Scalar Case

'Classic' (scalar) visibility of a source:

$$V_{pq} = I e^{-i\phi_{pq}}$$

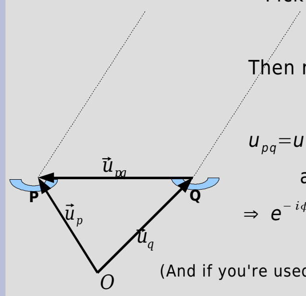
where ϕ_{pq} is the *interferometer phase difference:*

$$\phi_{pq} = 2\pi(u_{pq} l + v_{pq} m + w_{pq}(n-1))$$

Baseline coordinates $\vec{u}_{pq} = (u_{pq}, v_{pq}, w_{pq})$

have a very simple relationship to antenna coordinates.

Antenna UVWs and Antenna Phase



Pick an arbitrary reference point *O*.

$$\vec{u}_p \equiv \vec{OP},\ \vec{u}_q \equiv \vec{OQ},\ \vec{u}_{pq} \equiv \vec{QP}$$

Then regardless of which *O* we picked,

$$\vec{u}_{pq} = \vec{u}_p - \vec{u}_q, \text{ i.e.}$$

$$u_{pq} = u_p - u_q,\ v_{pq} = v_p - v_q,\ w_{pq} = w_p - w_q$$

and for phases: $\phi_{pq} = \phi_p - \phi_q.$

$$\Rightarrow\ e^{-i\phi_{pq}} = e^{-i(\phi_p - \phi_q)} = e^{-i\phi_p} e^{i\phi_q} = e^{-i\phi_p}(e^{-i\phi_q})^*$$

(And if you're used to thinking in terms of closure phases:

$\phi_{pq} + \phi_{qr} + \phi_{rp} = \phi_p - \phi_q + \phi_q - \phi_r + \phi_r - \phi_p = 0$)

The (familiar?) Scalar Case

We can decompose each *interferometer* phase term

into a pair of *antenna* phases:

$$V_{pq} = e^{-i\phi_p} I (e^{-i\phi_q})^*$$

compare this to:

$$\boldsymbol{V}_{pq} = \boldsymbol{K}_p \boldsymbol{B} \boldsymbol{K}_q^\dagger,$$

with $\boldsymbol{B} = \frac{1}{2}\begin{pmatrix} I & 0 \\ 0 & I \end{pmatrix}$

K: Breaking The Coherency Barrier

- The interferometer phase term is traditionally considered separately; however, it fits the Jones formalism like a glove.
- ***K*** combines two physical effects:
  - pathlength difference
  - time delays (i.e. fringe stopping)
- Scalar matrices commute with everything, so we're allowed to “merge” these two effects and shift the resulting ***K***'s around.
- ***K*** is scalar only for co-located receivers:
  - moral: keep your dipoles together!
- Forget about the Fourier Transform for now...

Building a Tree: Matrix Multiplication

$$\boldsymbol{V}_{pq} = \boldsymbol{K}_p \boldsymbol{B} \boldsymbol{K}_q^{\dagger}$$

- See `ME1/demo1-predict-ps.py`
- The **Meq.MatrixMultiply** node implements matrix multiplication.
- We repeat this for all interferometers (all *p-q* pairs), in a **for** loop.

Building a Tree: Creating a B Matrix

$$\boldsymbol{B} = \frac{1}{2}\begin{pmatrix} I+Q & U+iV \\ U-iV & I-Q \end{pmatrix}$$

- The **Meq.Matrix22** shortcut creates a matrix from four children (using a **Meq.Composer** node).
- *IQUV*'s will be hardwired constants (for now)

Building a Tree: VisPhaseShift

- The **Meq.VisPhaseShift** node computes the a phase term as follows:

$$\Phi(u,v,w,l,m,n;\nu) = \exp\left(-\frac{2\pi i \nu}{c}(ul+vm+wn)\right)$$

- Takes two “vector” children for *uvw*'s (in meters) and *lmn*'s.
- The **Meq.ConjTranspose** shortcut implements the “†” operation.
- We can explicitly form up an *(l,m,n-1)* vector using a **Meq.Composer** node.
- But where do we get the *uvw's*?

Building a Tree: Where to get meta-data?

Where do *uvw*'s come from?

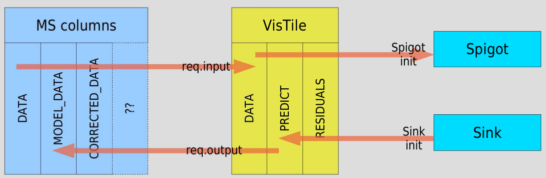
- *uvw*'s can be computed by a **Meq.UVW** node.
- This requires antenna positions, time, and the phase centre RA/Dec:
  - time: comes from the MS grid (via the request)
  - phase centre RA/Dec: from MS sub-tables
  - antenna positions: from MS sub-tables
- Don't want to hard-code this stuff, else our script will be tied to a particular MS.
- Need a way to get observational parameters from the MS and put them into the tree.

Building a Tree: Attaching an init-script

- An *init-script* is a Python script that is executed *on the kernel side* by the **VisDataMux** node. The script name is specified as part of the I/O request.
- It can provide a *MS header handler* (among other things).
- The MS header contains all observational parameters.
- The header handler can put the required values into “placeholder” nodes (finding them by name).
- The same script can be used for all trees, as long as our placeholder nodes follow the same naming convention.
  - i.e. “ra”, “dec” for phase center
  - “x0”, “y0”, “z0” for array center
  - “x:*p*”, “y:*p*”, “z:*p*” for position of antenna #*p*.
- Placeholders are created as constants, e.g. “`ns.ra<<0`”.

Building a Tree: Writing the data

- **VisTile** is an intermediate *uv* data layer (don't want to be locked into MSs forever...)
- The I/O record maps MS columns to VisTile
- Sinks & Spigots map tile columns to trees



Building a Tree: Ready!

- Load up the “K Jones” and “Inspector” bookmarks.
- Run the tree by selecting “test forest”
- Now we want to make an image.

We'll use the AIPS++ imager...

2. AIPS++...
   [5] heard of it
   [2] tried to run it once
   [9] succeeded in running it once
   [5] have used it in anger
   [0] invented it

- We have enough expertise in this room...
- ...no-one to blame this time though.

TDL Jobs: Doing other useful stuff

- _test_forest() is a “TDL job”.
- More jobs can be added by defining functions called _tdl_job_*foo*(), all these will be automatically placed into the “Exec” menu.
- Jobs can contain arbitrary Python code...
  - ...including calling the shell...
  - ...e.g. to run Glish and call the AIPS++ imager.
- See `ME1/demo2-predict-ps-image.py`

Introducing (Complex) Gain Errors

$$\boldsymbol{V}_{pq} = \boldsymbol{G}_p \boldsymbol{K}_p \boldsymbol{B} \boldsymbol{K}_q^\dagger \boldsymbol{G}_q^\dagger$$

$$\boldsymbol{G}_p = \begin{pmatrix} g_{x,p} & 0 \\ 0 & g_{y,p} \end{pmatrix} \qquad (g_x, g_y \text{ may be complex})$$

or in scalar form:

$$v_{xx,pq} = g_{x,p} g_{x,q}^* e^{-i\phi_{pq}} (I+Q)/2$$
$$v_{yy,pq} = g_{y,p} g_{y,q}^* e^{-i\phi_{pq}} (I-Q)/2$$
$$v_{xy,pq} = g_{x,p} g_{y,q}^* e^{-i\phi_{pq}} (U+iV)/2$$
$$v_{yx,pq} = g_{y,p} g_{x,q}^* e^{-i\phi_{pq}} (U-iV)/2$$

Building a Tree With Gains

- See `ME1/demo3-predict-ps-gain.py`
- It's trivial to add extra Jones terms to **Meq.MatrixMultiply.**
- The biggest effort is actually figuring out what numbers to plug in.
  - depends on your simulation objectives
  - we'll set up subtrees to compute these
  - could also come from a parameter DB, or from FITS images/tables.
- For now, let's assign a random, time-variable gain-phase to each antenna.

Time Variability On the Cheap

Something like: $g=e^{iA\sin(Bt+C)}$

- Remember that we get a “time grid” with each request.
- The **Meq.Time** node returns *f(t) = t*, combine it with a **Meq.Sin** node to compute *A**sin*(Bt+C)*
- Generate random *A,B,C*'s per dipole (using Python's **random** module)
- **Meq.Polar** builds x*exp(*iy*)
- Run the tree (load up the bookmarks).
- Make a map.
  - note that we can now select a column to image, the new simulation is in DATA, the old one is in MODEL_DATA.

Some Performance Considerations

$$V_{pq}=G_p K_p B K_q^{\dagger} G_q^{\dagger}$$

- This does N_{time}xN_{freq}x4 individual matrix multiplications.
- ***B*** is constant, ***G***'s are variable in time, and ***K***'s are variable in time-freq.
- If we reorder the terms as follows (***K*** commutes):

$$V_{pq}=K_p (G_p B G_q^{\dagger}) K_q^{\dagger}$$

we end up with N_{time}x2 + N_{time}xN_{freq}x2 ops.

- Is this a good idea?

To Quote Ancient Wisdom...

“Premature optimization is the root of all evil.”
-- Donald Knuth
“More computing sins are committed in the name of efficiency (without necessarily achieving it) than for any other single reason – including blind stupidity.”
-- W.A. Wulf

- Don't worry about optimizing for calculations until you establish that performance is a problem...
- ...and where the problem lies.
- Jones terms don't always commute, so think carefully before moving them around.
- But if you can reshuffle them, significant CPU savings may in fact result.

Exercise 1: Instrumental Polarization

- Use `ME1/demo3-predict-ps-gain.py` as a starting point.
- Make the source unpolarized, with *I*=1Jy.
- Make the gain amplitudes frequency-dependent (and reset the gain phases to 0):

$$G_p=\begin{pmatrix}1+a_p(\nu-\nu_0) & 0\\ 0 & 1-a_p(\nu-\nu_0)\end{pmatrix}$$

  - pick random a_p's in the range [1e-10,1e-9]
- For ν_0, create placeholder `(ns.freq0<<0)`
- Produce a per-channel map, look at *Q* fluxes.

Exercise 2: Alt-Az Mounts

- A “perfect” instrument has an equatorial mount, i.e. stationary sky.
- With an alt-az mount, the sky rotates relative to each antenna, so we must add a rotation term to our M.E.

$$\boldsymbol{V}_{pq}=\boldsymbol{G}_p\boldsymbol{P}_p\boldsymbol{K}_p\boldsymbol{B}\boldsymbol{K}_q^\dagger\boldsymbol{P}_q^\dagger\boldsymbol{G}_q^\dagger$$

$$\boldsymbol{P}_p=\begin{pmatrix}\cos\gamma_p & -\sin\gamma_p\\ \sin\gamma_p & \cos\gamma_p\end{pmatrix}\equiv\mathrm{Rot}\,\gamma_p$$

γ_p: parallactic angle for antenna p

Exercise 2: Alt-Az Mounts

- Use `ME1/demo3-predict-ps-gain.py` as a starting point.
- Use I=1 Jy, Q=.2 Jy
- Insert the gain terms from Exercise 1.
- Add a ***P*** term to model sky rotation.
- Produce MFS and per-channel maps of *IQUV* flux.
- Hint: **Meq.ParAngle** computes the P.A. as a function of time. It expects a **radec** child (phase center), and an **xyz** child (station position).

$$\boldsymbol{V}_{pq}=\boldsymbol{G}_p\boldsymbol{P}_p\boldsymbol{K}_p\boldsymbol{B}\boldsymbol{K}_q^\dagger\boldsymbol{P}_q^\dagger\boldsymbol{G}_q^\dagger$$

$$\boldsymbol{P}_p=\begin{pmatrix}\cos\gamma_p & -\sin\gamma_p\\ \sin\gamma_p & \cos\gamma_p\end{pmatrix}$$

Exercise 3: UFO

- Use `ME1/demo2-predict-ps-image.py` as a starting point.
- Make the source move:

$$\begin{pmatrix}l\\ m\end{pmatrix}=\begin{pmatrix}l_0\\ m_0\end{pmatrix}+\begin{pmatrix}\dot{l}\\ \dot{m}\end{pmatrix}\frac{\nu-\nu_0}{\nu_1-\nu_0}(t-t_0)$$

$$l_0=m_0=0,\quad \dot{l}=\dot{m}=.5'/\mathrm{hour}$$

- For t_0 ν_0 ν_1 use placeholder constants: `(ns.time0<<0); (ns.freq0<<0); (ns.freq1<<0);` the header script will initialize them for you.
- Produce a per-channel map.

Exercise 4: Amoebas

- Use `ME1/demo2-predict-ps-image.py` as a starting point, put source at l=m=0.
- Insert ionospheric phase (***Z*** jones) that we produced in Intro1/example5 and exercise 3.
  - use one TID (ampl=.1, 50km, 200km/h)
  - for the x,y ionospheric “positions”, use the station x,y (use a Meq.Composer to form up the xy vector...)
- Bonus points: make inspectors for TECs and ***Z***.
- Run script and make a per-channel map.
- Make a time-slice movie:

```
glish -l ~/Workshop2007/make_movie.g
       MODEL_DATA ms=demo.MS
```

Bonus Exercise 5: Simulating a Transient

- Simulate a transient source, make an MFS and a per-channel map.
- For bonus points, make the source narrow-band.