

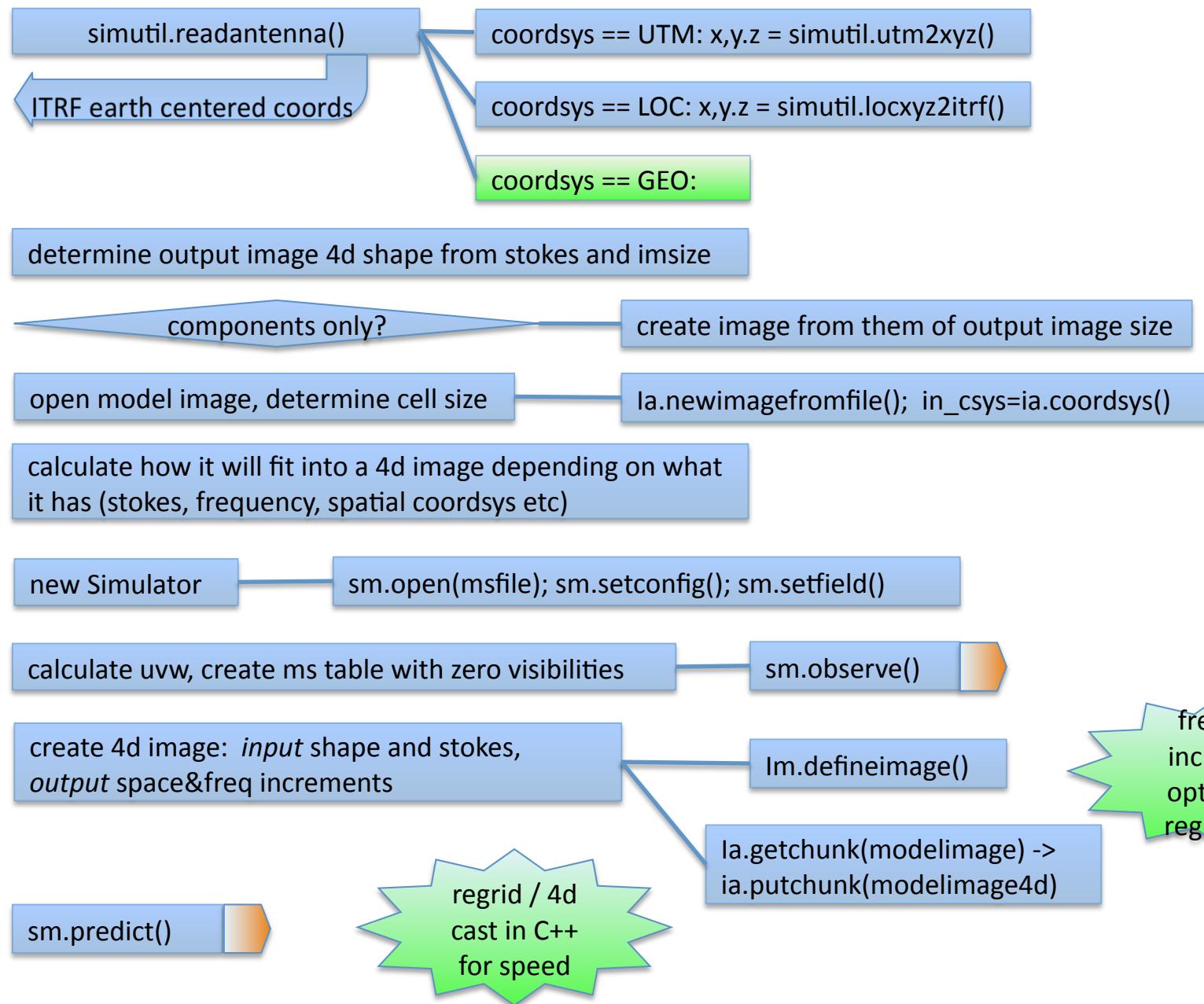
python

C++

Stub/unimplemented code

Places for  
upgrades

## simdata



simdata

sm.openfrommms(noisyms); sm.setnoise()

sm.corrupt()

call clean task

calculate moment zero *input* image

la.open(modelimage4d);  
la.moments(mements=[-1])

\$project.\$modelimage\$.flat0

calculate moment zero *output* image

\$project.\$modelimage\$.flat

regrid flat input to flat output shape, add clean components to regridded flat input

\$project.clean.flat

ia.convolve2d()

\$project.convolved.im

ia.imagecalc()

\$project.diff.im, \$project.fidelity.im

simutil.statim( model, clean, diff, etc)

calculate stats, plot using matplotlib

Simulator::observe(source,spw,startTime,stopTime)

NewMSSimulator::observe(source,spw,startTime,stopTime)

get antXYZ from antenna subtable, feed, spw, source info from their subtables

add nIntegrations rows and extend hypercube and subtables

NewMSSimulator::calcAntUVW()

Put UVW values in new rows

If autoCorrelationWt\_p > 0 anslo add AC rows

flag based on elevation and shadowing

add exact phase center to Pointing subtable

Pointing errors?

Simulator:: predict( modellImage, compList)

Simulator:: createSkyEquation( modellImage, compList)

sm\_p = new CleanImageSkyModel(); sm\_p->add() sets pointers and inits vars

SkyEquation:: predict()

SkyEquation:: predictComponents()

SkyEquation:: get(VisBuffer& result, ComponentList& compList)

SkyEquation:: applySkyJones(Skycomponent& corruptedComponent, vb, row)

e.g. BeamSkyJones:: apply(SkyComponent& ....)

PBMath.applyPB()

PBMath1D.apply(SkyComponent& ....)

compFlux(pol) \*= taper;

Pointing errors

SimpleComponentFTMachine:: get(vb, component);

rotateUVW(vb.uvw() ...)

modelData = component.visibility()

SkyComponent:: visibility()

SkyCompRep:: visibility()

e.g. GaussianShape::visibility(uvw, itsFT(-uvw(0)\*wavenumber, uvw(1)\*wavenumber))

copy visibilities to desired column (Model or Data)

Simulator:: predict( modellImage, compList)

Simulator:: createSkyEquation( modellImage, compList)

sm\_p = new CleanImageSkyModel(); sm\_p->add() sets pointers and inits vars

SkyEquation:: predict()

SkyEquation:: predictComponents()

copy visibilities to desired column of VB (Model or Data)

SkyEquation:: inititalizeGet()

SkyEquation:: applySkyJones(vb, row, ImageInterface&

e.g. BeamSkyJones:: apply()

PBMATH.applyPB()

Apply atmosphere TJones here?

Pointing errors

SkyEquation:: get(VisBuffer& result, Int model)

e.g. GridFT::get(

rotateUVW(vb.uvw ...; refocus(vb.uvw ....

FTMachine::getInterpolateArrays(

fgridft.f : dgrid()

copy visibilities to desired column of VB (Model or Data)

copy visibilities from VB back to VI

Stolen from George Moellenbrock's Synthesis Summer School lecture

# From Idealistic to Realistic

- Formally, we wish to use our interferometer to obtain the visibility function, which we intend to invert to obtain an image of the sky:

$$V(u, v) = \int_{\text{sky}} I(l, m) e^{-i2\pi(ul+vm)} dl dm$$

- In practice, we correlate (multiply & average) the electric field (voltage) samples,  $x_i$  &  $x_j^*$ , received at pairs of telescopes ( $i, j$ ) and processed through the observing system:

$$V_{ij}^{\text{obs}}(u_{ij}, v_{ij}) = \langle x_i(t) \cdot x_j^*(t) \rangle_{\Delta t} = J_{ij} V(u_{ij}, v_{ij})$$

- Averaging duration is set by the expected timescales for variation of the correlation result (typically 10s or less for the VLA)
- $J_{ij}$  is an *operator* characterizing the net effect of the observing process for baseline  $(i, j)$ , which we must *calibrate*
- Sometimes  $J_{ij}$  corrupts the measurement irrevocably, resulting in data that must be *edited*

Stolen from George Moellenbrock's Synthesis Summer School lecture

# The Scalar Measurement Equation

12

$$V_{ij}^{obs} = \int_{sky} J_i J_j^* I(l, m) e^{-i2\pi(u_{ij}l + v_{ij}m)} dl dm$$

- First, isolate non-direction-dependent effects, and factor them from the integral:

$$= (J_i^{vis} J_j^{vis*}) \int (J_i^{sky} J_j^{sky*}) I(l, m) e^{-i2\pi(u_{ij}l + v_{ij}m)} dl dm$$

- Next, we recognize that over small fields of view, it is possible to assume  $J^{sky}=1$ , and we have a relationship between ideal and observed Visibilities:

$$= (J_i^{vis} J_j^{vis*}) \int_{sky} I(l, m) e^{-i2\pi(u_{ij}l + v_{ij}m)} dl dm$$

$$V_{ij}^{obs} = (J_i^{vis} J_j^{vis*}) Y_{ij}^{true} = J_i J_j^* V_{ij}^{true}$$

- Standard calibration of most existing arrays reduces to solving this last equation for the  $J_i$

# Solving for the $J_i$

- We can write:  $\frac{V_{ij}^{obs}}{V_{ij}^{true}} - (J_i J_j^*) = 0$
- ...and define chi-squared:  $\chi^2 = \sum_{\substack{i,j \\ i \neq j}} \left| \frac{V_{ij}^{obs}}{V_{ij}^{true}} - (J_i J_j^*) \right|^2 w_{ij}$
- ...and minimize chi-squared w.r.t. each  $J_i$ , yielding (iteration):

$$J_i = \sum_{j \neq i} \left( \frac{V_{ij}^{obs}}{V_{ij}^{true}} J_j w_{ij} \right) / \sum_{j \neq i} (J_j)^2 w_{ij} \quad \left( \frac{\partial \chi^2}{\partial J_i} = 0 \right)$$

- ...which we recognize as a weighted average of  $J_i$ , itself:

$$J_i = \sum_{j \neq i} (J'_i w'_{ij}) / \sum_{j \neq i} w'_{ij}$$

# Calibration and Corruption

- $J_i$  contains many components:
  - $F$  = ionospheric effects
  - $T$  = tropospheric effects
  - $P$  = parallactic angle
  - $X$  = linear polarization position angle
  - $E$  = antenna voltage pattern
  - $D$  = polarization leakage
  - $G$  = electronic gain
  - $B$  = bandpass response
  - $K$  = geometric compensation
  - M, A = baseline-based corrections
- Order of terms follows signal path (right to left)
- In CASA, each term is a VisCal, and their application to visibilities is handled by the VisEquation
- For simulation, we must create VisCals of the the desired types, calculate their terms a priori and store that information in their CalSets

$$\vec{J}_i = \vec{K}_i \vec{B}_i \vec{G}_i \vec{D}_i \vec{E}_i \vec{X}_i \vec{P}_i \vec{T}_i \vec{F}_i$$

```
sm.openfrommms(noisyms); sm.setnoise()
```

Simulator::setnoise(pwv,altitude,etc)

Simulator:: create\_corrupt(simpar.type="ANoise")

svc = createSolvableVisCal(upType,\*vs\_p)

SolvableVisCal::setSimulate()

SolvableVisCal:: sizeUpSim( vs, nChunkPerSim, solTimes)

ANoise:: createCorruptor()

SolvableVisCal:: createCorruptor()

get nSpw etc from VI.msColumns

ANoiseCorruptor:: initialize()

new MLCG, new Normal

pass info down to corruptor like start/stop times, etc

Iterate through VI; for each chunk, determine correct time slot in corruptor;  
set antenna, focusChan, etc in corruptor;

solveCPar()(gpos) = corruptor\_p->simPar(vi,type(),ipar);

ANoiseCorruptor:: simPar()

return Complex((\*nDist\_p())\*amp(),(\*nDist\_p())\*amp());

keep each gain, weight, etc in CalSet

SolvableVisCal:: store() write caltable if desired

add this SVC to pointer block vc\_p

Simulator:: create\_corrupt(simpar.type="MMueller")

-> AtmosCorruptor

Arrange info  
more naturally  
between  
corruptor and  
parent VisCal ?

Verify  
weights are  
correct

Scale noise  
with a Jones

sm.corrupt()

Simulator:: corrupt()

VisEquation:: setApply(vc\_p) puts VCs in order

VisEquation:: setPivot() correct Model with some VCs, corrupt Data with the rest

Iterate VI

VisEquation:: collapseForSim(vb)

Model = Data

VisCal:: corrupt()

e.g. VisMueller:: applyCal(ModelCube)

Data = 0

VisCal:: correct()

e.g. VisMueller:: applyCal(visCube)

vb.visCube()+=vb.modelVisCube();

VisIter:: setWeightMat(vb)

check

copy from VB back to VI

Revised Topics for Tuesday:

- second attempt to explain VisCal corruption scheme
- Which Measurement Eqn Terms are most critical, and in which modes (full vs residual)
- Implementation of T
- Single Dish Issues: update on sdsim, which ME terms are most critical for Sdsim
- [morning in SOC: Bhatnagar]: implementation of Pointing calibration and correction
- Other issues

Lunch Talk

NAASC meeting:

simulation library

coordination of OST, portal, proposal prep etc – most critical functionality

4pm if interest Ted Bergin Astrochemistry Talk

$$V_{ij}^{obs} = J_i V_{ij}^{true} J_j^*$$

$$\vec{J}_i = \vec{K}_i \vec{B}_i \vec{G}_i \vec{D}_i \vec{E}_i \vec{X}_i \vec{P}_i \vec{T}_i \overleftrightarrow{EP}_i \vec{F}_i$$

$$V_{ij}^{obs} = A\text{Noise}_{ij} + M_{ij} J_i (A_{ij} V_{ij}^{true}) J_j^*$$

`visEquation::setPivot()` e.g. `setPivot(B)`

`visEquation::collapseForSim()`

- $F$  = ionospheric effects
- $T$  = tropospheric effects
- $P$  = parallactic angle
- $X$  = linear polarization position angle
- $E$  = antenna voltage pattern
- $D$  = polarization leakage
- $G$  = electronic gain
- $B$  = bandpass response
- $K$  = geometric compensation
- $M, A$  = baseline-based corrections
- $A\text{Noise}$  = baseline-based additive

Corrupt down to pivot:

$$Model_{ij}^{obs} = B_i G_i D_i T_i (A_{ij} Model_{ij}^{pure}) T_j^* D_j^* G_j^* B_j^*$$

Correct up to pivot:

$$Data_{ij} = K_i^{-1} ((A\text{noise}_{ij} + 0) M_{ij}^{-1}) K_j^{-1*}$$

Return Data+Model

Regular correction for calibration:

$$V_{ij}^{corr} = J_i^{-1} ((V_{ij}^{obs}) M_{ij}^{-1}) J_j^{-1*}$$

$$V_{ij}^{obs} = J_i V_{ij}^{true} J_j^*$$

$$\vec{J}_i = \vec{K}_i \vec{B}_i \vec{G}_i \vec{D}_i \vec{E}_i \vec{X}_i \vec{P}_i \vec{T}_i \overleftrightarrow{EP}_i \vec{F}_i$$

$$V_{ij}^{obs} = A\text{Noise}_{ij} + M_{ij} J_i (A_{ij} V_{ij}^{true}) J_j^*$$

`visEquation::setPivot()` e.g. `setPivot(X)`

`visEquation::collapseForSim()`

Corrupt down to pivot:

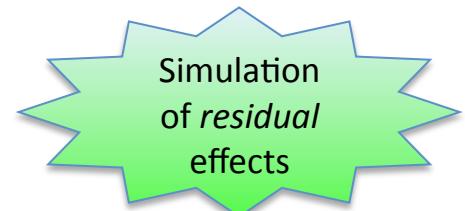
$$Model_{ij}^{obs} = T_i (A_{ij} Model_{ij}^{pure}) T_j^*$$

Correct up to pivot:

$$Data_{ij} = D_i^{-1} G_i^{-1} B_i^{-1} ((A\text{noise}_{ij} + 0) M_{ij}^{-1}) B_j^{-1*} G_j^{-1*} D_j^{-1*}$$

Return Data+Model

- $F$  = ionospheric effects
- $T$  = tropospheric effects
- $P$  = parallactic angle
- $X$  = linear polarization position angle
- $E$  = antenna voltage pattern
- $D$  = polarization leakage
- $G$  = electronic gain
- $B$  = bandpass response
- $K$  = geometric compensation
- $M, A$  = baseline-based corrections
- $A\text{Noise}$  = baseline-based additive



These VisCals represent the *residual* gains after gain calibration