CASA Overview and Main Features (w/emphasis on VP/PB & A-Proj. implementation)

- Implements Measurement Equation (Hamaker, Bergman & Sault, 1996; later extended by Noordam & Cornwell) (see next page)
- Internal data format is Measurement Set (Kemball & Wieringa, 2000): Table for radio telescope data (visibilities) & auxilliary sub-tables; CASA Tables
- 1.5 Million lines of Code (Mostly C++)

User interface, higher-level analysis routines, Viewers = Casa non-Core

- Tools: data access, display, science analysis

Examples:

- Simulator tool: simulator.xml, Simulator.cc (C++ Class)

python> sm.setvp(dovp=T, usedefault=F, ...): turn on/implement VP/PB for simulation

→ programmable command-line interface & scripting: Python (augmented by IPython)

- imager tool: imager.xml, Imager.cc, Imager2.cc

ftmachine = 'pbwproject'

- Tasks → high-level (user-level) analysis procedures:
  - Tasks (implemented in Python) → Tools (implemented in C++)
  - clean (python, xml): task_clean.py, clean.xml, cleanhelper.py

→ gridmode = 'aprojection' ⟷ type of gridding kernel for FFT-based Transforms

ftmachine = 'pbwproject'

Primary Beam correction (image domain, A-Projection)

python> im.setoptions (ftmachine = 'pbwproject')

clean(vis=msname, imagename=nameimag, mode='mfs', gridmode='aprojection', ...)

↕

General physical & astronomical utilities, infrastructure ≡ Casacore

Examples:

- MeasurementSets(.h) module handles storage of telescope data and access to it. MeasurementSet is class that gives access to data.
- MSSimulator: Create empty MeasurementSet from observation and telescope descriptions.
  - Simulator refers to generation of 'fake' data from set of parameters for instrument and sources. The application "Simulator" uses this class to create a true simulated MS with perfect or corrupted data

CASA Measurement Equation → CASA C++ Classes and Modules

- Toolkit for radio astronomical calibration, imaging, & simulation built around Measurement Equation

observed Visibility

Antenna VP, i.e., PB effects

geometry

$$\vec{V}_{ij} = \underbrace{\overleftrightarrow{M}_{ij}\,\overleftrightarrow{B}_{ij}\,\overleftrightarrow{G}_{ij}\,\overleftrightarrow{D}_{ij}}\int \underbrace{\overleftrightarrow{E}_{ij}\,\overleftrightarrow{P}_{ij}\,\overleftrightarrow{T}_{ij}\,\overleftrightarrow{F}_{ij}}\, S\,\vec{I}_\nu(l,m)\, e^{\overbrace{i2\pi(u_{ij}l+v_{ij}m)}^{\text{geometry}}}\, dl\, dm + A_{ij}$$

VisEquation C++ Class
Visibility-Plane effects (Direction independent effects; const. across F.O.V.)
$J_{ij}^{vis}(\nu,t) = M_{ij} B_{ij} G_{ij} D_{ij}$

Image-plane or Sky-plane part (Direction-Dependent Effects)
SkyEquation C++ Class
$J_{ij}^{sky}(\nu,t) = E_{ij} P_{ij} T_{ij} F_{ij}$

Image to be derived. SkyModel C++ class

Additive baseline-based error component

FTMachine

CubeSkyEquation C++ Class → nPBWProjectFT C++ Class

Imager2.cc

implements A-projection algorithm

fpbwproject.f (Fortran): gridding convolution function used by underlying gridder

VLAIllumination ConvFunc C++ class: computes autocorrelation of ideal antenna illumination pattern for VLA imaging.

IlluminationConvFunc, ConvolutionFunction C++ Classes

used to compute convolution functions for convolutional gridding

- $\overleftrightarrow{E}_{ij}$

PBMath

SkyJones

BeamSkyJones

VPSkyJones

- Classes that handle Fourier Trasform part of M.E. (e.g., SkyEquation) use $\boxed{\text{FTMachine}}$ (including $\boxed{\text{nPBWProjectFT}}$ when A-Projection is implemented) to carry out forward and inverse Fourier transforms.

- i, j = telescope I.D. pairs = baseline

CASA Imaging & Deconvolution (with and without Direction-Dependent Effects (DDEs)).

- $\boxed{\text{Imager.cc, Imager2.cc}}$: CASA C++ classes that contain functions needed for the $\boxed{\text{imager tool}}$.
- $\boxed{\text{imager (imager.xml)}}$: Tool that accomplishes synthesis processing.
- $\boxed{\text{clean}}$ task (clean.xml, cleanhelper.py, clean-pg.py): Can set 'gridmode' parameter (to, e.g., 'aprojection') and ftmachine (to, e.g., 'pbwproject') here.
- $\boxed{\text{Clark CLEAN algorithm}}$ can be implemented via $\boxed{\text{ClarkCleanLatModel}}$ and $\boxed{\text{ClarkCleanImageModel}}$ C++ classes (for example)

✱ Classical/Traditional Description of Imaging & Deconvolution without Consideration of DDEs : Linear Optimization View of Deconvolution

- Measurement Equation Describing Interferometer can be written as

$$\vec{V}^{o} = [A]\vec{I}^{o} + \vec{N} \qquad (1), \qquad \text{where}$$

$\vec{V}^{o}$ = true visibilities

$[A]$ = measurement matrix operator : linear transformation from image to visibility domain

$\vec{I}^{o}$ = true image (vector)

$\vec{N}$ = Gaussian random variable (in data domain), or, independent random noise vector.

- Model visibilities ($\vec{V}_{ij}^{M}$) for baseline i-j are calculated from existing model image $\vec{I}^{M}$:

$$\vec{V}^{M} = [A]\vec{I}^{M} \qquad (2), \qquad \text{where}$$

$$I^{M} = \sum_{k} P_k , \quad P_k = \text{Pixel Model} = F_K\,\delta(x - x_K, y - y_K) = \left\{\begin{array}{l}\text{collection of delta}\\ \text{functions of amplitude}\\ F_K \text{ at each pixel location}\end{array}\right\}$$

- χ^2 is optimal estimator

- Generalized dirty image is update direction for iterative deconvolution :

$$\Delta\vec{I}^{dirty} = -C\Delta\chi^2 , \qquad \text{where} \longrightarrow$$

$\chi^2 = |\vec{V}^0 - [A]\vec{I}^M|^2$, C = covariance matrix

Note: $\frac{\partial \chi^2}{\partial P_k} \equiv$ [Dirty Image]

Typically, model image is iteratively improved as:

$$I_i^M = T(I_{i-1}^M, [I_i^R]) = I_{i-1}^M + \alpha\, max \cdot [\Delta I_i^{dirty}] \quad (3)$$

$$\longrightarrow I_{i-1}^M + \alpha \Delta \chi^2 \quad (\text{where } 0<\alpha<1),$$

where

$\Delta I^M \equiv$ [update to existing model image] $= -C' \frac{\partial \chi^2}{\partial I^M}$, ($C'$ = scaling term, either const. or inverse of Hessian)

$I_i^R = \mathcal{F}\{\vec{V}^R\} \equiv FT\{\vec{V}^R\} \equiv$ (Fourier transform of $\vec{V}^R$),

where residual visibilities $\vec{V}^R = \vec{V}^{obs} - \vec{V}^M$, $\vec{V}^{obs}$ = observed visibilities

I_i^M = cumulative model of ith iteration

T = operator that selects part of gradient image and includes conversions between signal domain, polarization domain, and stokes frame using appropriate transform operator.

- Major Cycle can be broken down into two calculations:

(1) Forward: In forward step, model visibilities (V_{ij}^M) for baseline i-j are calculated from existing model image $\vec{I}^M$, using Eq. (2).

- When using FFT algorithm for computing Fourier Transform, gridded visibilities are interpolated from regular grid and re-sampled at measured (u, v, w) points as

$$\vec{V}^{M'}(u_{ij}, v_{ij}, w_{ij}) = \Big([G]\underbrace{[F\vec{I}^M]^g}_{\text{gridded visibilities}}\Big)(u_{ij}, v_{ij}, w_{ij}) \quad (4), \text{ where}$$

$[G]$ = interpolation operator

$[F]$ = Fourier Transform operator

g: indicates data on a regular grid

(2) Backward: In backward calculation, residual visibilities $\vec{V}^R = \vec{V}^{obs} - [A]\vec{I}^M$ are propagated backwards to image plane using $\longrightarrow$

$$\vec{I}^R = [A^\dagger A]^{-1} A^\dagger \vec{V}^R = FT[\vec{V}^R] = [\text{residual image}] \quad (5)$$

- When using FFT algorithm for computing FT, backward calculation will correspond to application of $[GF]^\dagger$ (see Eq. (4)), where operator $[F]$ is unitary.

- If $[G]$ is at least approximately unitary, $[G]^\dagger$ can be used as interpolation operator for re-sampling data on a regular grid to correct for effects of $[G]$ in image.

- An approx. inverse operator with finite support for our purposes can be constructed by using $G^\dagger$ for re-sampling data (l.h.s. of Eq. (4)) and then dividing resulting image by $\det(FG)$.

⊛ Imaging and Deconvolution with D.D.E. (including use of A-Projection Algorithm)

- Recall that full Measurement Equation can be written as:

$$\underbrace{V_{ij}^{obs}(\nu)}_{\text{Data}} = \underbrace{J_{ij}^{Vis}(\nu,t) \int J_{ij}^{sky}(s,\nu,t)}_{\text{Corruptions}} \underbrace{I(s,\nu)}_{\text{sky (Image)}} \underbrace{e^{i\vec{s}\cdot\vec{b}_{ij}}}_{\text{geometry (w-term)}} d\vec{s} \quad (6)$$

where,

Direction cosines: $\vec{s} = (l,m,n) = (l, m, \sqrt{1-l^2-m^2}-1)$

Baseline Vector: $\vec{b}_{ij} = (u_i - u_j,\ v_i - v_j,\ w_i - w_j) = (u_{ij}, v_{ij}, w_{ij})$

- $J_{ij}^{Vis} \equiv J_i^{Vis} \otimes J_j^{*Vis}$ (Direction independent effects; const. across F.O.V.): handled by CASA C++ Class $\boxed{\text{VisEquation}}$ ⟹ Visibility Measurement Equation expressing visibility-plane part of M.E.; Pre-requisite: Measurement Components module

- $J_{ij}^{sky} = J_i^{sky} \otimes J_j^{*sky}$ (Direction-Dependent Effects; e.g., Antenna PB): Image-Plane or Sky-Plane part of M.E., handled by CASA C++ Classes SkyEquation as well as CubeSkyEquation, BeamSkyJones, VPSkyJones, PBMath, PBMath1D, etc ---

- **Relevant C++ Inheritance Diagrams**:

casa::SkyEquation → encapsulates equation between sky brightness and visibility measured by generic interferometer, and it is responsible for image/sky-based part (e.g., VP/PB effects) of M.E. Principle use of SkyEquation is that gradients of χ^2 may be calculated and returned as an image.

↑

casa::CubeSkyEquation

Following components can be plugged into SkyEquation:
- Antenna-based DDE terms: SkyJones
- SkyBrightness Model : SkyModel
- Fourier Transfor Machine : FTMachine

Implements specialized Fourier Transform Machine nPBWProjectFT used for forward and inverse transforms involved with A-projection algorithm and in dealing with pointing offsets.

→ e.g., ft = new nPBWProjectFT(static_cast<nPBWProjectFT &(ft));

ftm_p[K] = new nPBWProjectFT(static_cast<nPBWProjectFT &>(*ft_));

casa::SkyJones → Model sky-plane instrumental effects for SkyEquation. Describes an interface for Components to be used in SkyEquation. It's an Abstract Base Class: most methods must be defined in derived classes.

↑

casa::BeamSkyJones → Beam-like sky-plane effects for SkyEquation. Pre-requisite: SkyEquation, SkyJones, PBMath Inheritance classes. -Derived from SkyJones; describes interface to beam-based SkyJones objects. Like SkyJones, it too has Abstract Base class, but implements beam-related methods. This class encapsulates antenna beam-based aspects which are present in at least a few other specific SkyJones (e.g., VPSkyJones & DBeamSkyJones)

Use PBMath

↑

casa::DBeamSkyJones | casa::VPSkyJones

casa::VPSkyJones → Pre-requisite: SkyEquation, BeamSkyJones. This class only deals with the diagonal elements of Voltage pattern Jones Matrices

casa::PBMathInterface → Virtual base class defining PB interface. Defines interface to PB Math objects, encapsulations of PB math functioning. → defines applyPB and applyVP methods

↑

casa::PBMath1D etc... → PBMath types do mathematical operations of PB's or VP's. This is base class for 1D (i.e., rotationally symmetric) PB's. Can deal with beam squint, which rotates on sky with parallactic angle.

↑

PBMath1DAiry

etc...

$I(s,\nu)$:

$\frac{\partial \chi^2}{\partial P_K}$: Dirty Image

$\Delta I^M = -C' \frac{\partial \chi^2}{\partial I^M}$

$I^M = \sum_K P_K$ → Pixel Model

casa::SkyModel → Model (of) sky brightness for SkyEquation. Contains a number of separate models. Interface to SkyEquation is via an image per model. SkyEquation uses this image to calculate Fourier Transforms, etc.

casa::ImageSkyModel → Image-based model for sky brightness. Describes interface for models to be used in SkyEquation.

casa::CleanImageSkyModel

casa::ClarkCleanImageSkyModel

– $\boxed{\text{A-Projection Algorithm}}$:

- An approximately unitary operator E_{ij} can be constructed as FT of full direction-dependent sky Mueller Matrix J_{ij}^{sky},

$$E_{ij} = FT[J_{ij}^{sky}] \qquad (7)$$

⟹ Image plane corrections can be incorporated as part of deconvolution process by using E_{ij} and $E_{ij}^{\dagger}$ as part of forward and backward (reverse) transforms between visibility and image domains for baseline i-j.

In absence of antenna pointing errors, operator E_{ij}^{P} is auto-correlation of ideal antenna illumination patterns of polarization product P. In presence of antenna pointing errors, E_{ij}^{P} is different for each baseline i-j.

- Now, can re-write M.E. (Eq. (6)) as (simplified):

$$V_{ij}^{obs} = E_{ij}^{P} * [V^{\circ}] = E_{i}^{*P} * E_{j}^{P} * [V^{\circ}] \qquad (8), \quad \text{where}$$

E_i^P = Antenna Aperture Illumination Pattern $= FT[J_i^{sky}]$

- Here V^{obs} is equal to the true visibilities V° convolved with autocorrelation of antenna aperture illumination pattern.

- In $\boxed{\text{backward application of DDEs}}$, conversion from image plane to u-v plane involves application of DDE's to predicted visibilities. This can be realized by using $E_{ij}^{p\dagger}$ as an interpolation operator for re-sampling $V^{p}(u_{ij}, v_{ij})$ (visibilities for polarization product p) on regular grid at pixels by indices (n,m) as:

$$V^{p,G}(n\Delta u, m\Delta v) = \left(E_{ij}^{p\dagger} V^{p}(u_{ij}, v_{ij})\right)(n\Delta u, m\Delta v) \qquad (9),$$ where

G is used to indicate on a regular grid.

- In $\boxed{\text{forward application of DDE's}}$ (converting from uv plane to image plane, i.e., imaging), DDE's are applied by means of convolution during uv-plane gridding stage, and images corresponding to gridded visibilities are then computed as

$$I^{dirty} = det\left(F^{\dagger}[E_{ij}^{p\dagger}]\right)^{-1} F^{\dagger} V^{p,G} \qquad (10)$$

We can also describe and explain Eqs. (9) & (10) using following notation and in following way:

If there exists a function K_{ij} such that $K_{ij}^{T} * E_{ij} \sim$ Delta Function, then:

- <u>Gridding</u>: $V_{ij}^{G} = K_{ij}^{T} * V_{ij}^{obs} = K_{ij}^{T} * E_{ij} * [V^{o}] \approx [V^{o}]_{ij} \qquad (11)$

- <u>Imaging</u>: $FFT[V^{o}] \longrightarrow I^{dirty} \qquad (12)$

- <u>Prediction</u>: $V_{ij}^{M} = K_{ij} * FFT[I^{M}] \qquad (13)$

A-Projection provides accurate method to apply known DDEs in forward direction, during degridding step, when predicting visibilities from a model image, and approximate method to correct for them in reverse direction (when gridding visibilities for imaging).

Overall Deconvolution Algorithm with A-Projection can be summarized as follows:

1. *Initialize*: Set initial model to 0 or to model using apriori knowledge of sky emission (e.g., model obtained with conventional techniques).
2. *Major Cycle*
   - *Forward Calculation*: Compute residual *visibilities* $V^{obs} - V^{M}$ using observed visibilities V^{obs} for each polarization product.
   - *Backward Calculation*: Compute residual *image* using Eqs. (9) & (10) above.
3. *Minor Cycle*: Update model image applying some operator T (see Eq. (3)).
4. Go to 2 until convergence reached, typically quantified by suitable stopping criteria (noise level, distribution of residuals, etc—)
5. Smooth deconvolved image by resolution element and add back residuals.

- In terms of CASA C++ Classes and inheritance diagrams, implementation of A-Projection in CASA can be roughly illustrated as follows:

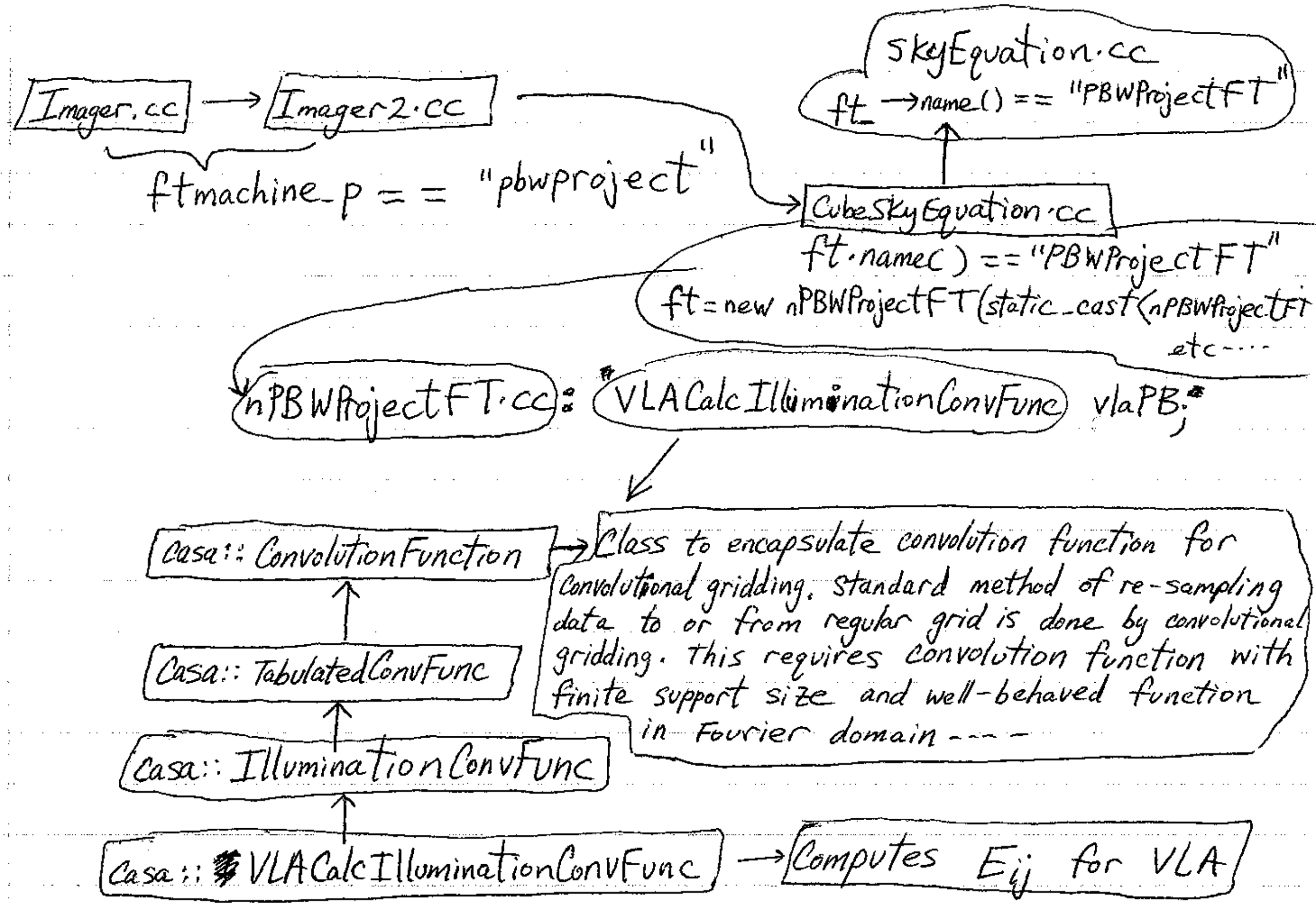
Casa:: FTMachine

↑

Casa:: nPBWProjectFT

→ defines interface to Fourier Transform machine. Pre-requisite: *SkyModel* module, *SkyEquation* module, *VisBuffer* module. SkyEquation needs to be able to perform FTs on visibility data. FTMachine allows efficient FT processing using VisBuffer which encapsulates chunk of visibility (typically all baselines for one time) together with all info. needed for processing (e.g., UVW coordinates.

→ nPBWProjectFT is an FTMachine for Gridded FTs, including effects of PB & pointing offsets. Pre-requisite classes are FTMachine, SkyEquation, & VisBuffer. In the implementation of A-projection algorithm, FTs of full direction-dependent Sky Mueller matrix for baselines i-j are evaluated using PBMath class & VLACalcIlluminationConvFunc class (which involves calculation of autocorreleation of ideal antenna illumination pattern for VLA imaging), which is derived from base class ~~Convolution Function~~ ConvolutionFunction. {GCF used by underlying gridder is written in Fortran and handled by code fpbwproj.f}



References:

- Bhatnagar, S. et al., A&A, 487, 419, 2008; EVLA Memo #100 (2006)
- Rau, U. et. al., Proc. IEEE, Vol. 97, No. 8, 2009
- Smirnov, O., A&A, 527, A107, 2011
- http://www.aoc.nrao.edu/~sbhatnag
- CASA Class Hierarchy: http://casa.nrao.edu/active/docs/doxygen/html/hierarchy.html