```
sk.image ('observed', image='3C273XC1.dirty');
sk.clean (niter=1000, threshold='3mJy', model='3C273XC1.clean.model');
#
# Fourier transform the model to the uv-plane
#
sk.ft (model='3C273XC1.clean.model');
#
# Close the imager tool
#
sk.close();
sk.done();
#
# Create a calibrater tool
#
c:=calibrater('3C273XC1.MS');
#
# Select solution for electronic gain (G) and atmospheric phase (T)
#
c.setsolve ("G", 300.0, F, 3, "gcal_out", F);
c.setsolve ("T", 30.0, T, 3, "tcal_out", F);
#
# Solve for the selected G and T components
#
c.solve();
#
# Close the calibrater tool
#
c.close();
```

Tools

calibrater	Synthesis calibration (self- and cross-)

1.2.1 calibrater - Tool

Package	synthesis
Module	calibrater

Synthesis calibration (self- and cross-)

Description

A calibrater tool allows synthesis calibration operations within AIPS++. A separate calibrater tool needs to be created for each separate measurement set for which processing is to be performed.

Constructors

calibrater	Construct a calibrater tool

Functions

accumulate	Accumulate incremental calibration solutions into a cumulative calibration table
calave	Average calibration solutions over fields, spectral windows, and/or time
close	Close the calibrater tool
correct	Apply calibration information
done	Destroy the calibrater tool
fluxscale	Bootstrap the flux density scale from standard calibrators
initcalset	Re-initialize the calibration scratch columns
linpolcor	Correct the gain table for linear polarization of the calibrator
modelfit	Fit single-component source models to the uv-data
plotcal	Plot a calibration table
posangcal	Apply position angle calibration to an existing cal table
reset	Reset the selected apply and/or solve components
setapply	Arrange to apply calibration
setdata	Set the data selection for subsequent processing
setsolve	Arrange to solve for calibration
setsolvebandpoly	Specialization of setsolve for polynomial B (bandpass) solving
setsolvegainpoly	Specialization of setsolve
setsolvegainspline	Specialization of setsolve for cubic spline G (time-dependent gain) solving
smooth	Produce a smooth and/or interpolated table
solve	Solve for the selected calibration components
state	Request the apply/solve state of the calibrater tool

calibrater - Constructor

Package	synthesis
Module	calibrater
Tool	calibrater

Construct a calibrater tool

Synopsis
calibrater(filename, compress)

Arguments

filename	in	MeasurementSet file name. No default AllowedString
compress	in	Compress calibration columns? AllowedBool DefaultF

Returns
calibrater tool

Description

Create a calibrater tool, and attach it to the specified Measurement Set.

Example

```
c:=calibrater('3C273XC1.MS')
c.solve()
c.close()
```

---

calibrater.setdata - Function

Package	synthesis
Module	calibrater
Tool	calibrater

Set the data selection for subsequent processing

Synopsis
setdata(mode, nchan, start, step, mstart, mstep, uvrange, msselect)

Arguments

mode	in	Type of data selection: channel or velocity Allowed: String from 'none'|'channel'|'velocity' Default: 'none'
nchan	in	Number of channels to select (mode='channel') Allowed: Int Default: 1
start	in	Start channel (1-relative) (mode='channel' Allowed: Int Default: 1
step	in	Step in channel number (mode='channel') Allowed: Int Default: 1
mstart	in	Start velocity (e.g. '20Km/s') Allowed: Quantity Default: '0km/s'
mstep	in	Step in velocity (e.g. '100m/s' Allowed: Quantity Default: '0km/s'
uvrange	in	UV range (in klambda) (e.g. [0, 50]). Default is no restriction. Allowed: Vector of Doubles Default: 0
msselect	in	TAQL selection string. Default is no selection. Allowed: String Default: ' '

Returns
Bool

Description

This function selects which data will be treated by subsequent execution of the solve and correct functions. Note that data selection is not cumulative; any selection made in a previous call is replaced by the selection from the current call.
The arguments are used as follows:

mode indicates how you select the desired channels. A value of 'none' means select all data (make no channel selection); all channel- or velocity-dependent arguments are ignored in this case. A value of 'channel' indicates that you select by channel number. A value of 'velocity' indicates that you select by radio velocity and a value 'opticalvelocity' indicates that you select by optical velocity.

nchan is the number of selected channels for mode='channel'. It defaults to 1; to select all channels, set to 0.

start is active only when mode='channel' and specifies the first channel to be selected. It defaults to 1.

step is active only when mode='channel' and specifies the increment between selected input channels. It defaults to 1.

The channels are centered on velocities: start, start+step, start+2*step, etc.

mstart is active only when mode='velocity' or mode='optical velocity'. It specifies the start velocity to select as a quantity string. The default is 0.

mstep is active only when mode='velocity' or mode='optical velocity'. It specifies the increment between selected channels as a quantity string. The default is 0 which is not very useful...

The channels are centered on velocities: mstart, mstart+mstep, mstart+2*mstep, etc.

msselect is used to specify a subselection of data according to field, spectral window, time, etc., and should be specified as a valid TaQL expression.

Example

```
c:=calibrater('3C273XC1.MS');
```

Select according to channel:

```
c.setdata(mode='channel',nchan=512,start=1,step=1,msselect='FIELD_ID==1')
```

Select according to velocity:

```
c.setdata(mode='velocity', nchan=64, mstart='20km/s', mstep='-100m/s');
```

Reset selection to the entire dataset

```
c.setdata()
```

---

calibrater.setapply - Function

Package	synthesis
Module	calibrater
Tool	calibrater

Arrange to apply calibration

Synopsis

```
setapply(type, t, table, interp, select, spwmap, unset, opacity,
rawspw)
```

Arguments

type	in	Component type Allowed String from 'B','BPOLY','G','GSPLINE','D','P','T','TOPAC','GAINCURVE','M','MF','K'
t	in	Interpolation interval (seconds) Allowed Float Default 0.0
table	in	Calibration table name Allowed String Default ' '
interp	in	Interpolation type (in time) Allowed String from 'nearest', 'linear', 'aipslin' Default 'nearest'
select	in	TAQL selection string. Default is no selection Allowed String Default ' '
spwmap	in	Spectral windows to apply Allowed Vector of Int Default -1
unset	in	If T, unset apply of this type Allowed Bool Default F
opacity	in	Array-wide zenith opacity (for type='TOPAC') Allowed Float Default 0.0
rawspw	in	Spectral window with data phases for application Allowed Float Default -1

Returns

Bool

Description

This function is used to specify the calibration components which should be applied during subsequent execution of the solve and correct functions. This function will be executed as many times as necessary to specify all desired calibration components. Each calibration component represents a separate calibration matrix correction included in the measurement equation. The different types correspond to different instrumental and atmospheric effects. Calibration components are available as calibration tables generated by previous solve executions (types 'B','BPOLY','G','GSPLINE','D','T','M','MF','K'), or are calculated analytically on the fly (types 'P', 'TOPAC', 'GAINCURVE'). See the setsolve section for more information about the solvable types.
Use the interp parameter to specify the desired type of time-dependent interpolation. Use interp='nearest' to calibrate each datum with the calibration value nearest in time. Use interp='linear' to calibrate each datum with calibration phases and amplitudes linearly interpolated from neighboring (in time) values. In the case of phase, this mode will assume that phase jumps greater than 180 degrees between neighboring points indicate a cycle slip, and the interpolated value will follow this change in cycle accordingly. Use interp='aipslin' to emulate the basic interpolation mode used in classic AIPS, i.e., linearly interpolated amplitudes, with phases derived from interpolation of the complex calibration values. While this method avoids having to track cycle slips (which is unstable for solutions with very low SNR), it will yield a phase interpolation which becomes increasingly non-linear as the spanned phase difference increases. The non-linearity mimics the behavior of interp='nearest' as the spanned phase difference approaches 180 degrees (the phase of the interpolated complex calibration value initially changes very slowly, then rapidly jumps to the second value at the midpoint of the interval). If the uncalibrated phase is changing this rapidly, a 'nearest' interpolation is not desirable. Usually, interp='linear' is the best choice.
The interp parameter is applicable to any calibration type, as long as there are sufficient solutions available to perform the interpolation. Note that calibration solutions which have been determined for only one timestamp will default to 'nearest'. More interpolation options (e.g., 'cubic') will be added in the near future.
Currently, the interpolation interval is forced to t=0 for solved-for components, which means that the interpolation (or extrapolation) will extend indefinitely, as circumstances dictate. A future version of calibrater will use this parameter to limit the range in time over which interpolation/extrapolation

will be permitted.
The interpolation type and timescale are, by definition, irrelevant for analytically calculated calibration types (e.g., the parallactic angle, P).
For solved-for components, the filename specified in table parameter must contain a calibration table of the type specified in type.
Use the select parameter to specify selection of a subset of calibration measurements from the table to be applied to the visibility data. Arbitrary cross-calibration is possible by combining this function with the setdata function. The string specified must be a valid TaQL expression.
The spwmap parameter is used to indicate how solutions derived from different spectral windows should be applied to other spectral windows. Nominally, data in each spectral window will be corrected by solutions derived from the same spectral window. This is the default behavior of spwmap, i.e., if spwmap is not specified, calibrater will insist that data be corrected by solutions from the same spw. Otherwise, spwmap takes a vector of integers indicating which spectral window *solutions* to apply to which spectral window *data*, such that spwmap[j]=i causes solutions derived from the i-th spectral window to be used to correct the j-th spectral window. For example, if (say) bandpass solutions are available for spws 1 & 3, and it is desired that these be applied to spws 2 & 4 (as well as 1 & 3), respectively, use spwmap=[1,1,3,3]. Even if some spws do not require an explicit spwmap setting, yet one or more does, it is safest to specify it explicitly for all, e.g., spwmap=[1,2,4,4] indicates that spw 3 will be corrected with solutions from spw 4, and the others will behave nominally. Note that if no solutions exist for any of the spws specified in spwmap, an error message will result.
For type='TOPAC', an elevation-dependent opacity correction will be applied according to the zenith opacity value supplied in the opacity parameter. Currently, only one zenith opacity value can be supplied, and it is used for all antennas.
For type='GAINCURVE', an elevation-dependent correction will be applied using parameters read from the data repository. Currently, this is only supported for the VLA.
Use the rawspw parameter to specify a spectral window from which the phase will be copied for application (in a baseline-based manner) to the data selected in setdata. The phases will be scaled by the appropriate frequency ratio. Note that this option currently only works when solving for or applying type='GSPLINE' solutions.
Currently, only single-polarization data will work with

type='BPOLY' or type='GSPLINE'.
Use the state function to review the list of calibration components that have been set for application. To deselect a single component, run setapply with unset=T and type accordingly. Use the reset function to delect all components.
Pending improvements:

- Enable variety of interpolation modes and timescales
- Allow for antenna- and time-dependent opacities
- Generalize BPOLY and GSPLINE to dual polarization
- Provide more user-friendly selection

Example

```
c:=calibrater('3C273XC1.MS');
c.setdata();
c.setapply (type='G', table='gcal2', select='FIELD_NAME==FIELD1')
c.setapply (type='P' t=10.0)
c.correct();
c.close();
```

In this example, 'G' solutions from FIELD1 in table gcal2, and 'P' solutions calculated on a 10-second timescale are applied to the entire visibility dataset.

---

calibrater.smooth - Function

Package	synthesis
Module	calibrater
Tool	calibrater

Produce a smooth and/or interpolated table

Synopsis
smooth(infile, outfile, append, select, smoothtype, smoothtime, interptype, interptime)

Arguments

infile	in	Input calibration table Allowed String Default No default
outfile	in	Output calibration table Allowed String Default No default
append	in	Append to an existing output table? Allowed Bool Default F
select	in	TAQL selection string Allowed String Default No selection
smoothtype	in	The smoothing filter to be used Allowed String from 'mean', 'median', 'smean' (sliding mean), 'smedian' (sliding median) or 'none' Default 'mean'
smoothtime	in	Smoothing filter time constant (sec) Allowed Float Default 0.0
interptype	in	The type of interpolation desired Allowed String from 'nearest','spline','linear','cubic',or 'none' Default 'spline'
interptime	in	Sampling interval for interpolation (sec) Allowed Float Default 0.0

Returns
Bool

Description

This function is intended to be used for interpolating and/or smoothing Jones matrices derived from one or more sources (calibraters). The time-stamps at which the interpolating function is sampled is defined by the selection (via the select argument) and the Measurement Set associated with the input calibration table. The interpolated values are then written to the output calibration table. A smoothing filter can be optionally applied to the input table and the smoothed table then used to derive the specified interpolation function.
A value of 'none' for either interptype or smoothtype switches off interpolation or smoothing respectively. A value of 'none'

for both these would therefore just copy the input table to the output table.
The size of the 'mean' and 'median' smoothing filters is truncated (as against rounded) to the nearest value smaller than or equal to the specified smoothing time constant. This means that the smoothing filter size can be smaller than the specified time constant by some fraction of the sampling interval in the input table. The 'smean' and 'smedian' are sliding versions of the 'mean' and 'median' filters. The sliding step is always one pixel, with the filter centered on each pixel (gain solutions in the input table). At the boundaries, the filters expand (from the starting edge) or shrink (for the ending edge) from a size of one pixel (for the last and first point) till they are of the size smaller than or equal to the specified time-constant.
The filter output is used for deriving the parameters of the specified interpolation function. The interpolated function is then sampled at the sampling interval specified by interptime.
The output table can then be applied to the data as usual (via the setapply()->correct() sequence).

Example

```
c:=calibrater('3C273XC1.MS');
c.smooth(infile='in.gcal',outfile='out.gcal', append=F, select='',smoothtype='median',smoothtime=10,interptype='spline');
c.setapply (type='G', table='out.cal', select='FIELD_NAME==FIELD1')
c.correct();
c.close();
```

In this example, 'G' solutions in the table 'in.gcal' are first smoothed using the median filter with a time-constant of 10sec and then interpolated on timestamps from the associated MS using the spline interpolation function. The output is written into 'out.gcal' and applied to a field named 'FIELD1'.

---

calibrater.setsolve - Function

Package	synthesis
Module	calibrater
Tool	calibrater

Arrange to solve for calibration

Synopsis
setsolve(type, t, preavg, phaseonly, refant, table, append, unset)

Arguments

type	in	Component type Allowed String from 'G','T','B','D','M','MF','K'
t	in	Solution interval (in sec) Allowed Float Default 60.0
preavg	in	Pre-averaging interval (in sec) Allowed Float Default 60.0
phaseonly	in	Solve only for phase? Allowed Bool Default F
refant	in	Reference antenna. Default is none. Allowed Int Default -1
table	in	Output calibration table name Allowed String
append	in	Append to existing table? Allowed Bool Default F
unset	in	If T, unset solve of this type Allowed Bool Default F

Returns
Bool

Description

This function specifies the calibration component will be solved for by the solve function. Currently, only one type should be solved for at one time.
Each calibration component represents a separate calibration matrix correction included in the measurement equation. The different types correspond to different instrumental and atmospheric effects. Currently, the solvable calibration components are types 'G','T','B', and 'D', which are antenna-based, and, 'M', 'MF', and 'K' which are baseline-based.

Arrange to pre-apply any existing calibration components (of types other than the solved-for one) using the setapply function. The solution interval t, if > 0.0, specifies the duration of data used for each calibration solution. In general, the solution intervals are measured from the beginning of data segments for each field and spectral window. If t is large enough, a single solution may encompass data from more than one scan or even the whole observation (per field and spectral window). The solution interval represents a coherence time, not an integration time w.r.t. any gaps in the time series; in effect, such gaps are ignored, and the latest time in the solution is never more than t seconds after the earliest time. If t $=$ 0.0, one solution per scan will be computed regardless of the (possibly variable) scan durations. If t $=$ -1, one solution will be obtained for each unique timestamp in the data.
For types B, G, and T, the Measurement Equation is always assumed stationary on the timescale of the solution (t), and so the data are appropriately normalized and automatically averaged within the solution interval before solving. For type D, there are systematic variations (e.g., parallactic angle, P) in Measurement Equation terms (even after pre-applied) on timescales shorter than the expected variation in the D terms. In this case, it is necessary to specify a pre-averaging interval which is shorter than the solution interval. This is done using the preavg parameter, which defaults to 60 seconds.
Supported calibration types:

G Antenna-based, multiplicative complex gains

B Antenna-based complex bandpass; a frequency-dependent version of G

T Antenna-based polarization-independent (tropospheric) gain; a polarization-independent version of G

D Antenna-based instrumental polarization leakage

M Baseline-based complex gains (cf, G)

MF Baseline-based complex bandpass (cf, B)

K Baseline-based fringe-fitting (phase,delay,delay-rate; currently limited to delays and delay-rates in the central ambiguity)

Pending improvements:

- Fix multi-setsolve (serial) mode

- Improve phase-only mode (per type), internal refant usage
- Add fixed mean gain modulus option
- Enable generalized solution modes, e.g., per-field, etc. (and rationalize t=0 and t=-1 accordingly).

Example

```
c:=calibrater('3C273XC1.MS');
c.setapply (type='P',t=5.0);
c.setsolve (type='G',t=300.0, refant=3, table='gcal');
c.solve();
c.done();
```

In this example, analytic (non-solvable) parallactic angle corrections are applied on a 5-second timescale before G solutions are obtained on a timescale of 300 seconds. The resulting solutions are phase-referenced to antenna 3, and stored in a calibration table called 'gcal'.

```
c.reset();
c.setapply (type='P',t=5.0);
c.setapply (type='G',table='gcal');
c.setsolve (type='D',t=86400.0, preavg=60.0, refant=3, table='dcal');
c.solve();
c.done();
```

In this example, the solve/apply state of the calibrater tool is reset and then the P and G corrections (from above) are applied before solving for D solutions on a diurnal timescale. Note that the data will be averaged only to 60 seconds before the solution. The resulting D solutions are stored in a table called 'dcal'.

---

calibrater.setsolvegainpoly - Function

Package	synthesis
Module	calibrater
Tool	calibrater

Specialization of setsolve

Synopsis

setsolvegainpoly(table, append, mode, degree, refant, unset)

Arguments

table	in	Output calibration table name Allowed: String Default: ' '
append	in	Append to existing table? Allowed: Bool Default: F
mode	in	Phase or Amplitude mode? Allowed: String from 'PHAS', 'AMP', or 'PHASAMP' Default: 'PHAS'
degree	in	Order of polynomial) Allowed: Int Default: 3
refant	in	Reference antenna. Default is none. Allowed: Int Default: -1
unset	in	If T, unset solve of this type Allowed: Bool Default: F

Returns
Bool

Description

This function is a specialization of the setsolve method.

---

calibrater.setsolvegainspline - Function

Package	synthesis
Module	calibrater
Tool	calibrater

Specialization of setsolve for cubic spline G (time-dependent gain) solving

Synopsis

setsolvegainspline(table, append, mode, preavg, splinetime, refant, npointaver, phasewrap, unset)

Arguments

table	in	Output calibration table name Allowed: String Default: ' '
append	in	Append to existing table? Allowed: Bool Default: F
mode	in	Phase or Amplitude mode? Allowed: String from 'PHAS', 'AMP', or 'PHASAMP' Default: 'PHAS'
preavg	in	Pre-averaging interval (in sec) Allowed: Float Default: 0.0
splinetime	in	Spline timescale (sec) Allowed: Float Default: 10800
refant	in	Reference antenna. Default is none. Allowed: Int Default: -1
npointaver	in	Allowed: Int Default: 10
phasewrap	in	Allowed: Float Default: 250
unset	in	If T, unset solve of this type Allowed: Bool Default: F

Returns
Bool

Description

This function is a specialization of the setsolve method which should be used when cubic spline G solutions are desired, e.g., when SNR on calibrators is very low. Currently, this solving mode is available only for single polarization data.

The visibility data are averaged in frequency (for multi-channel data) prior to the solution. The spline timescale (time between knots) is specified in the splinetime paramter, which defaults to 10800 seconds (3 hours).
The solution is performed for phase and/or amplitude according to the mode parameter. For phase solutions only, use mode='PHAS'. For amplitude solutions only, use mode='AMP'. If both are desired, use mode='PHASAMP', and both will be solved for using the same spline timescale (this mode also assumes that all calibrators have the correct relative flux densities). If solving for phase and amplitude separately (usually in this order), be sure run setapply to specify application of the first one when solving for the second one. The combined solution will be stored in the table specified for the second solve. If it is desired that the first solution table be preserved, use a new table name.
Different spline timescales may be used for phase and amplitude. If multiple fields are present, they will be used together to obtain a single solution. In mode='AMP', it is important that the model flux densities of each fields be correct, so that a single, consistent gain function can be properly determined.
The parameters npointaver and phasewrap tune the phase-unwrapping algorithm when mode = 'PHAS'. Cycle slips are detected (and removed before the spline solve) when the median phase a sequence of length npointaver (in integrations) differs by more than phasewrap degrees from the previous sequence.
When solve is run, the solutions are obtained, and postscript plot of baseline data and solutions is generated and deposited in your working directory. Its name is the same as table, with .ps appended.
At this time, it is not yet possible to plot the *antenna-based* solutions themselves using plotcal.
After obtained, these solutions should be applied using type='GSPLINE' in setapply.
Pending improvements:

- Generalize to dual polarization
- Enable plots of antenna-based solutions (currently, only plots of baseline data and solutions are produced)

Example

```
c:=calibrater('3C273XC1.MS');
c.setsolvegainspline (table='gcal',mode='PHAS');
```

```
c.solve();

c.reset();
c.setsolvegainspline (table='gcal',mode='AMP');
c.solve();
c.done();
```

In this example, a spline solution is first found for phase, then for amplitude.

---

calibrater.setsolvebandpoly - Function

Package	synthesis
Module	calibrater
Tool	calibrater

Specialization of setsolve for polynomial B (bandpass) solving

Synopsis

setsolvebandpoly(table, append, degamp, degphase, visnorm, bpnorm, maskcenter, masksedge, refant, unset)

Arguments

table	in	Output calibration table name Allowed: String Default: ' '
append	in	Append to existing table? Allowed: Bool Default: F
degamp	in	Polynomial degree for amplitude solution Allowed: Int Default: 3
degphase	in	Polynomial degree for phase solution Allowed: Int Default: 3
visnorm	in	Normalize data prior to solution Allowed: Bool Default: F
bpnorm	in	Normalize result? Allowed: Bool Default: T
maskcenter	in	Number of channels to avoid in center of each band Allowed: Int Default: 1
masksedge	in	Fraction of channels to avoid at each band edge (in %) Allowed: Int Default: 5
refant	in	Reference antenna Allowed: Int Default: -1
unset	in	If T, unset solve of this type Allowed: Bool Default: F

Returns
Bool

Description

This function is a specialization of the setsolve method which should be used to arrange for bandpass solving when polynomial solutions for B are desired, e.g., when SNR on calibrators is very low. Currently, this solving mode is available only for single polarization data.
Prior to the solution, the visibility data are averaged in time,

and if multiple spectral windows are present (after running setdata), they are concatenated in frequency and one polynomial solution is delivered.

The solution is performed for both phase and amplitude, to the polynomial degrees specified in degamp and degphase (they can be different), and the result will be stored in the same table. The frequency domain of the solutions is limited to only the range of frequencies selected in setdata. When correcting data with these solutions (for other solves or with correct), only data within this domain will be corrected. Data outside (e.g., edge channels avoided in setdata for the solve), will not be corrected. Therefore, the same (or narrower) channel selection is recommended for all operations using solutions produced by this function and solve().

Like other solve types, the solve() function actually performs the solve according to the parameters set in this function, and for this type, it generates a plot file which shows the baseline spectra delivered to the solver (i.e., corrected according to any setapply() invocations, and time-averaged), with the derived solution (products of antenna-based bandpass polynomials) overlaid.

After solving, these solutions should be applied using type='BPOLY' in setapply.

The visnorm parameter is used to normalize the assembled spectral data, in a per baseline manner. If visnorm=T is used, this will have the effect of removing any non-frequency-dependent closure errors (e.g., as caused by source structure, or introduced by the instrument) from the data, and should be used with caution. The resulting solutions will be effectively normalized as well. When visnorm=F is used, closure errors in the data (as supplied to the solver) may be visible in the form of offsets between the data and solutions. For bandpass calibration, this is usually ok, as the *shape* of the bandpass is the most important aspect of the solution.

The bpnorm parameter controls normalization of the resulting solutions *after* the solve. Note that the baseline spectra plot does not show normalized solutions, even if bpnorm=T (the plot shows the data as provided to the core solver, and is generated before the solutions are normalized).

Note that use of bpnorm=T or visnorm=T will remove or prevent the storage of scaling information in the bandpass solutions. Sometimes it is desirable to store absolute scaling information in the bandpass solution; in these cases, bpnorm and visnorm must both be F.

The maskcenter and maskedge parameters control how many channels are ignored on-the-fly, at the center and edges of each input spectral window, respectively. To avoid edge channels, it is almost always better to flag these channels directly, or select against them in setdata. Aggressive use of maskedge (large values), will yield polynomial solutions which will tend to diverge at the edges (especially when the polynomial degree is also high), because maskedge does not change the frequency domain of the solutions. Such solutions should be used with caution in subsequent operations. (It is best to avoid use of maskedge.)

Example

```
c:=calibrater('ngc5921.ms');
c.setsolvebandpoly(table='bpoly',degamp=5,degphase=7);
c.solve();
c.done();
```

In this example, amplitude (degree 5) and phase (degree 7) Chebychev polynomial bandpasses are determined using the default parameters.

---

calibrater.state - Function

Package	synthesis
Module	calibrater
Tool	calibrater

Request the apply/solve state of the calibrater tool

Synopsis
state()

Returns
Bool

Description

Request the apply/solve state of the calibrater tool. A listing of all calibration components that have been set for application or solving is written to the logger.

Example

```
c:=calibrater('3C273XC1.MS');
c.setapply ('P', 5.0);
c.setsolve ('G', 300.0, F, 3, 'gcal_1', T);
c.state();
```

---

calibrater.reset - Function

Package	synthesis
Module	calibrater
Tool	calibrater

Reset the selected apply and/or solve components

Synopsis
reset(apply, solve)

Arguments

apply	in	If T, unset all apply settings Allowed: Bool Default: T
solve	in	If T, unset all solve settings Allowed: Bool Default: T

Returns
Bool

Description

Resets the apply and/or solve components previously set by setapply and setsolve.

Example

```
c:=calibrater('3C273XC1.MS');
c.setapply ('P', 5.0);
c.setsolve ('G', 300.0, F, 3, 'gcal_1', T);
c.state();
c.reset(apply=T,solve=F);

c.reset()
```

---

calibrater.initcalset - Function

Package	synthesis
Module	calibrater
Tool	calibrater

Re-initialize the calibration scratch columns

Synopsis
initcalset(calset)

Arguments

calset	in	Not yet used. Allowed:Int Default:1

Returns
T

Description

This function re-initializes the calibration scratch columns: MODEL_DATA to unity (in total intensity, and unpolarized), and CORRECTED_DATA to (observed) DATA.

Example

```
c:=calibrater('3C273XC1.MS');
c.setapply ('P', 5.0);
```

```
c.setsolve ('G', 300.0, F, 3, 'gcal_1', T);
c.state();
c.initcalset();
c.solve();
```

---

calibrater.solve - Function

Package	synthesis
Module	calibrater
Tool	calibrater

Solve for the selected calibration components

Synopsis
solve()

Returns
Bool

Description
Execution of this function initiates a solve for the calibration component specified in a previous setsolve execution. Existing calibration components (as specified in one or more setapply executions) will be appropriately applied to the observed and model data according to their position in the Measurement Equation, and their commutation properties.

Example

```
c:=calibrater('3C273XC1.MS');
c.setapply ('P', t=10)
c.setsolve ('G', 300.0, F, 3, 'gcal_1', T);
c.solve();
c.close();
```

---

calibrater.correct - Function

Package	synthesis
Module	calibrater
Tool	calibrater

Apply calibration information

Synopsis
correct()

Returns
Bool

Description

This function applies the calibration components specified via the setapply function to the observed visibility data and writes the result to the CORRECTED_DATA column of the Measurement Set.

Example

```
c:=calibrater('3C273XC1.MS');
c.setapply ('G', 10.0, 'gcal_1')
c.correct();
c.close();
```

---

calibrater.modelfit - Function

Package	synthesis
Module	calibrater
Tool	calibrater

Fit single-component source models to the uv-data

Synopsis
modelfit(niter, type, par, vary, file)

Arguments

niter	in	Number of fitting iterations to execute Allowed: Int Default: 0
type	in	The component type Allowed: String: "P", "G", or "D" Default: "P"
par	in	Starting guess for component parameters Allowed: Vector of Int Default: [1,0,0]
vary	in	Control which parameters to let vary in fit Allowed: Vector of Bool Default: []
file	in	Optional output component list table Allowed: String Default: ""

Returns
Vector containing resulting fit parameters

Description

This function enables fitting single component source models to the uv-data selected by a prior run of setdata. If the selected data span multiple spectral windows, a single model will be fit to them jointly.

The function is in the calibrater tool because it is desirable to fit a resolved calibrator with a resolved component shape after phase calibration but before amplitude calibration, to improve the quality of the amplitude calibration solution.

The supported component types are points, elliptical Gaussians, or elliptical Disks, and their integrated flux density, direction offset (from the phase center), and (if applicable) their shape parameters are derived using a non-linear fitting algorithm. A subset of their parameters may optionally be held fixed.

Use of modelfit is straightforward:

The niter parameter is used to specify how many non-linear fitting iterations should be used before quitting. Since the adequacy of the model component choice is not known a priori, modelfit does not attempt to automatically detect convergence; it will do as many iterations as specified (not counting iterations which spuriously increase chi-squared). A few to a few 10s of iterations is usually adequate. If not, then the requested fit is ill-conditioned.

The type parameter is used to indicate which component type to solve for. Use "P" for a point, "G" for an elliptical Gaussian, or "D" for an elliptical disk. modelfit will do its best to fit the selected shape.
The par parameter is used to specify a starting guess for the component parameters. Three parameters are required for a point component: integrated flux density, and RA and Decl direction offsets (arcsec). Three additional parameters are required for the elliptically shaped components: the ellipse major axis size (arcsec), the ellipse axial ratio ($<$ 1), and the major axis position angle (deg, measured from North thru East on the sky).
The vary parameter is a vector of Bools that indicate which of the component parameters specified in par should be varied in the fit. The default is to vary all of the component parameters (all T). If any elements of vary are set F, the corresponding parameters in par will be held fixed in the fitting.
The file parameter is used optionally to specify a table name in which will be created at the conclusion of the fit to contain the result component (as a componentlist). This componentlist table can be used in other applications, e.g., in imager.ft() to specify a model for filling the MODEL_DATA column for subsequent processing.
It is generally very important to give a decent ("close enough") starting guess for the parameters. The way to think about this is that, for a single point component, all peaks (positive or negative) in the dirty image correspond to local minima in chi-square, and the fit can be trapped in any of them. You want the fit to get trapped in the correct (global) one. This generally means that the first guess should be within a factor of a few of the correct flux density, within a (naturally weighted) beamwidth of the correct direction offset, and a factor of a few of the correct size. The other parameters are somewhat less sensitive to the starting guess (being limited in range or cyclical). These estimates of adequacy can be complicated by correlations between the parameters for some sets of values.
Interactive and autmatic means of providing optimal first guesses are under consideration for future versions, which will also support multiple components.
The errors reported by modelfit are robust if the data weights are strictly inverse variance weights with no arbitrary global scaling and the model component choice is appropriate. If these conditions hold, the reduced chi-quared should approach ~ 1, and the parameter error estimates will be solid formal errors. If the weights are arbitrarily scaled, so too will be the chi-squared and error estimates. Interpret these with caution!
An excess in the converged chi-square could indicate either a

global mis-scaling of the weights (hopefully), or that the model component just isn't a good representation of the source. In either case, the reported errors will be wrong. If the weights are mis-scaled *and the model is, in fact, a good fit*, then the errors will be under-estimated by approximately the square root of the reduced chi-squared value (which may be < 1, in which case the errors are actually over-estimated).
Pending improvements:

- Provide interactive (and automatic) means of specifying components. This will be particularly useful when the source is not expected to lie at the phase center.
- Fit multiple components. This is mainly limited by the inadequate means of specifying a priori components. The multiple component option may be limited to a standalone modelfit application, to keep the calibrater.modelfit interface simple.
- Improve error estimation and reporting, including access to the full covariance matrix.
- Permit on-the-fly averaging in frequency and time, to improve performance.
- Fit for component polarization
- Fit for spectral index for multi-spectral window or multi-channel data.
- Support more component shapes, e.g., optically thin ellipsoid, elliptical rings, etc.

Example

```
mycal:=calibrater('uvmod_test.ms');

# an offset point source
mycal.setdata(msselect='FIELD_ID==4');    # select field 4
mycal.modelfit(niter=10,                  # 10 fitting iterations
               type='P',                  # fit for a point
       par=[1.0,3.0,1.0]);        # first guess

# an offset disk
mycal.setdata(msselect='FIELD_ID==6');        # select field 6
mycal.modelfit(niter=15,                      # 15 fitting iterations
```

```
              type='D',                      # fit for disk
              par=[3.0,2.5,-2.5,7.0,0.5,0]); # first guess

# a Gaussian at the phase center, for keeps
mycal.setdata(msselect='FIELD_ID==2');    # select field 2
mycal.modelfit(niter=10,                  # 10 fitting iterations
              type='G',                   # fit for Gaussian
              par=[2.0,0,0,1.0,0.5,0],    # first guess for pars
              vary=[T,F,F,T,T,T] ,        # fix direction (at phase center)
              file='gauss1.cl');          # export a component file

myimgr:=imager('uvmod_test.ms');          # fill resulting model into
myimgr.setdata(fieldid=2);                #  MODEL_DATA column
myimgr.ft(complist='gauss1.cl');
myimgr.done();

....more calibration steps....

mycal.done();
```

---

calibrater.fluxscale - Function

Package	synthesis
Module	calibrater
Tool	calibrater

Bootstrap the flux density scale from standard calibrators

Synopsis
fluxscale(tablein, tableout, reference, transfer, append,
refspwmap, fluxd)

Arguments

tablein	in	Input calibration table name Allowed:String
tableout	in	Output calibration table name. Default is tablein. Allowed:String Default:''
reference	in	Reference calibrator field names or indices Allowed:Vector of strings
transfer	in	Transfer source field names or indices. Default is all other fields Allowed:Vector of strings Default:-1
append	in	Append to existing table? Allowed:Bool Default:F
refspwmap	in	List of alternate spw for referencing Allowed:Vector of Ints Default:-1
fluxd	out	Array of output flux densities Allowed:Double Default:[]

Returns
Bool

Description

This function is used to bootstrap the amplitude scale the calibration solutions according to specified reference calibrator(s) of known flux density. This is necessary when the flux densities of some of your calibrators were unknown (and thus were assumed to be 1 Jy) during G solving.
The bootstrapping is achieved by comparing the mean gain norm of the calibration solutions derived for the calibrators specified in reference (one or more sources with known flux densities at the time of G solving) with that of the calibrators specified in transfer, and enforcing the assumption that the antenna gains are constant, on average. The gain solutions for the transfer sources are then re-scaled accordingly.
If no transfer fields are specified, then the solutions for all non-reference fields in tablein will be re-scaled.
If no tableout is specified the input table will be overwritten with the scaled solutions. Note that the resulting table will only contain solutions for those fields implicit in the reference

and transfer specifications. Use append=T to append the scaled solutions to an existing table.
Use the refspwmap parameter to indicate how data for different spectral windows should be matched in calculating the flux density scale factor for transfer fields. The default behavior for refspwmap is to insist on precisely matching spectral windows for reference and transfer fields. When specified, the refspwmap parameter takes a vector of integers indicating which spectral window solutions to use as the reference for others, such that refspwmap[j]=i causes solutions (from reference fields) observed in the i-th spectral window to be used to reference solutions (from transfer fields) observed in the j-th spectral window. For example, for the case of a total of 4 spectral windows: if the reference fields were observed only in spw=2 & 4, and the transfer fields were observed variously in all 4 spws, specify refspwmap=[2,2,4,4]. This will ensure that transfer fields observed in spws 1,2,3,4 will be referenced to reference field data from spws 2,2,4,4, respectively. Note that if the transfer fields were observed only in spws 1 & 3, the same specification would work, but refspwmap=[2,2,4] would suffice. In this case, nothing need be specified for the 4th spw (there are no transfer fields there), and specifying 2 for the 2nd spw is actually inconsequential (though required so that the specification of 4 for spw 3 is properly interpretted).
The derived flux densities for the transfer fields will be reported in the logger, and returned to the glish variable specified in fluxd. This will be an 2D array of shape [number-of-spectral-windows X number-of-fields].
Note that elevation-dependent gain effects may render the basic assumption used here invalid, and so should be corrected for prior to solving for G, using types 'TOPAC' or 'GAINCURVE' in setapply.
Note that the visibility data itself is not used directly by this function.
Pending improvements:

- Allow antenna and uv-distance selection to improve results for resolved calibrators
- Set the visibility model according to the flux density results
- An option to use the data to derive the relative flux densities

Example

```
c:=calibrater('3C273XC1.MS');
c.fluxscale (tablein='3C273.gcal', tableout='3C273.gcal2',
             reference='1328+307', transfer="0957+561 0917+624");
c.done();
```

This example takes an existing calibration table containing G solutions, and writes a scaled output version, using 3C286 (1328+307) as the reference calibrator, to derive properly scaled amplitude calibration for the transfer sources, 0957+561 and 0917+624.

---

calibrater.accumulate - Function

Package	synthesis
Module	calibrater
Tool	calibrater

Accumulate incremental calibration solutions into a cumulative calibration table

Synopsis
accumulate(tablein, incrtable, tableout, field, calfield, interp, t)

Arguments

tablein	in	Input cumulative calibration table name Allowed:String Default:""
incrtable	in	Input incremental calibration table name Allowed:String Default:""
tableout	in	Output cumulative calibration table name. Default is tablein. Allowed:String Default:""
field	in	List of fields to update in input cumulative table. Default is all. Allowed:Vector of strings Default:-1
calfield	in	List of fields in incremental table to use. Default is all. Allowed:Vector of strings Default:-1
interp	in	Interpolation mode to use on incremental solutions Allowed:String Default:'linear'
t	in	Cumulative table timescale when creating from scratch Allowed:Float Default:-1.0

Returns
Bool

Description

This function enables cumulative calibration using calibrater. It is the analog of the task ''CLCAL'' in classic AIPS. The accumulate function is useful when:

- a calibration solution of a particular type already exists,
- an incremental calibration solution *of the same type* is desired (an incremental solution in this context means derived independently from, or determined with respect to, the first)
- the first calibration cannot be implicitly recovered in the course of obtaining the incremental solution

For example, a phase-only ''G'' self-calibration on a target source may be desired to tweak the full amplitude and phase ''G'' calibration already obtained from a calibrator. The initial calibration (from the calibrator) contains amplitude information, and so must be carried forward, yet the phase-only solution itself cannot (by definition) recover this information, as a full amplitude and phase self-calibration would. In this case, the initial solution must be applied while solving for the phase-only solution, then the two solutions combined to form a *cumulative* calibration embodying the net effect of both. In terms of the Measaurement Equation, the net calibration is the *product* of the initial and incremental solutions.

The analog of accumulate in classic AIPS is the use of CLCAL to combine a series of (incremental) SN calibration tables to form successive (cumulative) CL calibration tables.

Cumulative calibration tables also provide a means of generating carefully interpolated calibration, on variable user-defined timescales, that can be examined prior to application to the data with setapply and correct. The solutions for different fields and/or spectral windows can be interpolated in different ways, with all solutions stored in the same table.

The only difference between incremental and cumulative calibration tables is that incremental tables are generated directly from the data via solve or (in the near future) from other ancilliary data (e.g. weather information), and cumulative tables are generated from other cumulative and incremental tables via accumulate. In all other respects (internal format, application to data via setapply and correct, plotting with plotcal, etc.), they are the same, and therefore interchangable. Thus, accumulate and cumulative calibration tables need only be used when circumstances require it.

The accumulate function represents a generalization on the classic AIPS CLCAL model of cumulative calibration in that its application is not limited to accumulation of ''G'' solutions (SN/CL tables classic AIPS are the analog of ''G'' (and, implicitly, ''T'') in aips++). In principle, any basic calibration type can be accumulated (onto itself), as long as the result of the accumulation (matrix product) is of the same type. This is true of all the basic types, except ''D''. Accumulation is currently supported for ''B'', ''G'', and ''T'', and, in future, ''F'' (ionospheric Faraday rotation), ''J'' (generic full-polarization calibration), fringe-fitting, and perhaps others. Accumulation of certain specialized types (e.g., ''GSPLINE'', ''TOPAC'', etc.) onto the basic types will be supported in the near future. The treatment of various calibration from ancilliary data (e.g., system temperatures,

weather data, WVR, etc.), as they become available, will also make use of accumulate to achieve the net calibration.
Note that accumulation only makes sense if treatment of a uniquely incremental solution is required (as described above), or if a careful interpolation or sampling of a solution is desired. In all other cases, re-solving for the type in question will suffice to form the net calibration of that type. For example, the product of an existing ''G'' solution and an amplitude and phase ''G'' self-cal (solved with the existing solution applied), is equivalent to full amplitude and phase ''G'' selfcal (with no prior solution applied), as long as the timescale of this solution is at least as short as that of the existing solution.
Use of accumulate is straightforward:
The tablein parameter is used to specify the existing cumulative calibration table to which an incremental table is to be applied. Initially, no such table exists, and accumulate will generate one from scratch (on-the-fly), using the timescale (in seconds) specified by the parameter t. These nominal solutions will be unit-amplitude, zero-phase (i.e., unit matrix) calibration, ready to be adjusted by accumulation. When t is negative (the default), the table name specified in tablein must exist and will be used.
The incrtable parameter is used to specify the incremental table that should be applied to tablein. The calibration type of incrtable sets the type assumed in the operation, so tablein must be of the same type. If it is not, accumulate will exit with an error message. (Certain combinations of types and subtypes will be supported by accumulate in the future.)
The tableout parameter is used to specify the name of the output table to write. If un-specified (or ''''), then tablein will be overwritten. Use this feature with care, since an error here will require building up the cumulative table from the most recent distinct version (if any).
The field parameter specifies those field names in tablein to which the incremental solution should be applied. The solutions for other fields will be passed to tableout unaltered. If the cumulative table was created from scratch in this run of accumulate, then these solutions will be unit-amplitude, zero-phase, as described above.
The calfield parameter is used to specify the fields to select from incrtable to use when applying to tablein. Together, use of field and calfield permit completely flexible combinations of calibration accumulation with respect to fields. Multiple runs of accumulate can be used to generate a single table with many combinations. In future, a ''self'' mode will be enabled that

will simplify the accumulation of field-specific solutions.
The interp parameter is used to specify the interpolation type to use on the incremental solutions, as in setapply. The currently available interpolation types are ‘‘nearest’’, ‘‘linear’’, and ‘‘aipslin’’. See the setapply URM documentation for more details.
Pending improvements:

- Implement a ‘‘self’’ mode (independent of interpolation type), to simplify or eliminate use of the field and calfield parameters in some contexts (e.g., self-cal)
- Provide spwmap parameter to enable transfer of solutions among spectral windows, as in setapply
- More interpolation modes, e.g., ‘‘cubic’’, and interpolation timescale (timerange to permit interpolation)
- Handle propogation (or not) of bad/flagged solutions
- Support of specialized types (e.g., TOPAC) onto the basic types
- Smoothing (probably a separate function)

Example

```
c:=calibrater('ap366.sim');

# obtain G solutions from calibrator
c.setdata(msselect='FIELD_ID IN [9,11]');
c.setsolve(type='G',table='cal.G0',t=300);
c.solve()

# obtain proper flux density scale
c.fluxscale (tablein='cal.G0', tableout='cal.G1',
             reference='1328+307', transfer="0917+624");

# generate cumulative table for target source on 20s timescale
c.accumulate(tablein='',incrtable='cal.G1',tableout='cal.cG0',
             field='0957+561',calfield='0917+624',
             interp='linear',t=20);

# apply this calibration to target
c.setdata(msselect='FIELD_ID==10');
c.setapply(type='G',table='cal.cG0',interp='linear')
```

```
c.correct();

#    (image target with imager tool)

# phase-selfcal target on 60s timescale
c.setdata(msselect='FIELD_ID==10');
c.setapply(type='G',table='cal.cG0',interp='linear')
c.setsolve(type='G',table='cal.G2',t=60,phaseonly=T);
c.solve();

# accumulate new solution onto existing one
c.accumulate(tablein='cal.cG0',incrtable='cal.G2',tableout='cal.cG1',
             field='0957+561',calfield='0957+561',
             interp='linear');

# apply new cumulative solution to data
c.setapply(type='G',table='cal.cG1',interp='linear')
c.correct();

#   (another round of imaging, etc.)

c.done();
```

---

calibrater.posangcal - Function

Package	synthesis
Module	calibrater
Tool	calibrater

Apply position angle calibration to an existing cal table

Synopsis
posangcal(tablein, tableout, posangcor)

Arguments

tablein	Input calibration table name Allowed:String
tableout	Output calibration table name. Default is input table Allowed:String
posangcor	Position angle corrections (degrees) Allowed:Vector of Floats Default:[]

Returns
Bool

Description

This function is used to apply position angle calibration for observations made using circularly polarized feeds. According to the Measurement Equation formalism, this correction should be applied to a D (instrumental polarization) calibration table.
If no D calibration is performed (and thus no such table is available), the correction can be applied to a G table, but it should NEVER be applied to both, and always applied to a D table if one is available. An input table must be specified. If no output table is specified, then the input table will be modified in place.
Specify, as a vector of values, a position angle adjustment (in degrees) for each spectral window. If only one value is specified, it will be duplicated to all spectral windows; otherwise, the number of values specified must match the number of spectral windows. The sign convention for the position angle adjustment is such that the specified value is the that which, when added to the position angle implied by the data, will yield the correct position angle. For example, if G-, D-, and P-calibrated data for 3c286 suggests a position angle of 45 degrees, the posangcor value should be -12 degrees as this will yield the correct position angle of 33 degrees when added. In general, posangcor equals correct position angle minus observed position angle.
A future version of this function will have an option to recognize standard position angle calibrators and determine the correction automatically.

Example

```
c:=calibrater('3C273XC1.MS');
c.posangcal(tablein='3C286.dcal', tableout='3C286.dpacal',
            posangcor=[-12.0, 54.0]);
c.done();
```

This example takes an existing calibration table containing D Jones matrices, and applies a position angle calibration of 45 and 54 degrees to spectral windows 1 & 2, respectively, writing the result to a new table. The observed position angles for 3C286 must have been 45 and -21 degrees; the corrections specified yield the correct value of 33 degrees when added to the observed values.

---

calibrater.calave - Function

Package	synthesis
Module	calibrater
Tool	calibrater

Average calibration solutions over fields, spectral windows, and/or time

Synopsis

calave(tablein, tableout, fldsin, spwsin, fldsout, spwout, t, append, mode, verbose)

Arguments

tablein	in	Input calibration table name. No default Allowed:String
tableout	in	Output calibration table name. No default Allowed:String
fldsin	in	Input field id(s). No default Allowed:Vector of integers
spwsin	in	Input spectral window id(s). No default Allowed:Vector of integers
fldsout	in	Output field id(s). No default Allowed:Vector of integer
spwout	in	Output spectral window. No default Allowed:Integer
t	in	Averaging time (s). Default is to just copy solutions Allowed:Float Default:-1.0
append	in	Append to existing table? Allowed:Boolean Default:F
mode	in	 Allowed:String from RI or AP Default:'RI'
verbose	in	Write extra information to logger about proceses Allowed:T or F Default:'T'

Returns
Bool

Description

This function can be used to average calibration solutions over fields, spectral windows, and/or time. It is useful for smoothing solutions in time, and for re-identifying solutions for field and spectral window, as may be required in spectral line observations where doppler tracking is used on the target source, but not on the calibrater (and hence the spectral windows for each source are different). This function currently supports averaging of calibration tables containing Jones matrices with diagonal elements only (G, B, T). For B, the averaging is done for each channel separately (there is no channel averaging). Handling of tables with off-diagonal elements (D, F) will be supported in the future.

The fldsin parameter takes a vector of integers; the fldsout parameter must be a single integer (used for all fldsin) or a vector of integers, one for each in fldsin. They must both be specified.
The spwsin parameter takes a vector of integers; currently, only one spwout can be specifed (all spwsin will be averaged to this). They must both be specified. Use append=T to write more than one output spectral window to the same file. All input and output ids must exist in the measurement set.
Use the t paramater to set the interval over which solutions from the input field ids and spectral window ids will be averaged. Note that simple solution time-averaging can be achieved by making fldsin=fldsout and spwsin=spwout. If this parameter is negative (the default) the solutions are just copied.
Use the mode parameter to choose between 'RI'= averaging real and imaginary (coherently) or 'AP'= averaging amplitude and phase (incoherently). Use 'AP' to avoid reduction in amplitude if the solution phases drift appreciably during the averaging interval, t.

Example

```
c:=calibrater('3C273XC1.MS');
c.calave(tablein='3C273.gcal', tableout='3C273.gcal2',
             fldsin=[1,2],spwsin=[2,3],fldsout=7,spwout=13,
             t=300, append=F, mode='AP')
c.done();
```

This example will average solutions from field ids 1 & 2 in spectral windows 2 & 3 in 300 second intervals and identifies the result as field id 7 and spectral window 13. The averaging is incoherent.

---

calibrater.linpolcor - Function

Package	synthesis
Module	calibrater
Tool	calibrater

Correct the gain table for linear polarization of the calibrator

Synopsis

linpolcor(tablein, tableout, fields)

Arguments

tablein	Input calibration table name Allowed: String Default: ''
tableout	Output calibration table name Allowed: String Default: ''
fields	Calibrator field names Allowed: Vector of strings Default: ""

Returns
Bool

Description

This function can be used to correct the gains derived from secondary calibrators with unknown or variable polarization. It should only be used for arrays with linear (X/Y) feeds and an Alt-Az mount for which the observed polarization varies with feed position angle on the sky.
The function fits the gains with a sine and cosine term in feed position angle and extracts the Q and U components of the secondary calibrator. This is only possible if there is sufficient range in the position angle (i.e., minimum of about 6 scans spanning at least 90 degrees in position angle). Check the error of the fit to judge if the fit was succesfull, it should generally be smaller than 0.5%.
Use the fields argument to select calibrator fields to be fitted. The function takes a calibration table as input, and can write the adjusted gain solutions to the same table on output, or create a new table containing these results. The function also prints the derived polarization for each field for each spectral window.

Example

```
c:=calibrater('atca.ms');
c.linpolcor(tablein='atca.gcal', tableout='atca.gcal2',
```

```
            fields='2254-367');
c.done();
```

This example takes an existing calibration table containing G Jones matrices, and writes a corrected output table, correcting only gains derived from 2254-367 for linear polarization.

---

calibrater.plotcal - Function

Package	synthesis
Module	calibrater
Tool	calibrater

Plot a calibration table

Synopsis
plotcal(plottype, tablename, antennas, fields, polarization, spwids, multiplot, nx, ny, psfile)

Arguments

plottype	in	Plot type Allowed:String from 'AMP'|'1/AMP'|'PHASE'|'RLPHASE'|'XYPHASE'|'RI'|'DAMP'|'DPHASE'|'DRI'|'FIT'|'FITWGT'|'TOTALFIT' Default:'AMP'
tablename	in	Calibration table name Allowed:String Default:''
antennas	in	Antennas to plot Allowed:Vector of Ints Default:[]
fields	in	Fields to plot Allowed:Vector of Ints Default:[]
polarization	in	Polarization to plot Allowed:Integer Default:1
spwids	in	Spectral windows id.'s to plot Allowed:Vector of Ints Default:[]
multiplot	in	Turn on separate antenna plots Allowed:Bool Default:F
nx	in	If multiplot=T, number of plots on horizontal axis Allowed:Integer Default:1
ny	in	If multiplot=T, number of plots on vertical axis Allowed:Integer Default:1
psfile	in	name of output PostScript file. Empty string causes plot to be sent to the screen Allowed:string Default:''

Returns
Bool

Description

This function plots a calibration table either to a plotter or to a file.
The argument plottype can take the following values for all types of solutions:

AMP Gain Amplitude vs. Time

1/AMP Inverse Gain Amplitude vs. Time (useful for comparing with classic AIPS)

PHASE Gain Phase vs. Time

RI Gain Real vs. Imaginary

RLPHASE Right/Left Gain phase difference (if polarizations are R,L)

XYPHASE X/Y Gain phase difference (if polarizations are X,Y)

The argument plottype can take the following values for D tables

DAMP Cross-polarized Gain Amplitude vs. Time

DPHASE Cross-polarized Gain Phase vs. Time

DRI Cross-polarized Gain Real vs. Imaginary

The quality of the solutions can be examined with the following plottype choices:

FIT Fit per spectral window

FITWGT Fit weight per spectral window

TOTALFIT Total fit

By default, all antennas (as specified in the antennas argument) will appear on the same plot. Separate plots (all with the same scale) for each antenna can be activated by setting multiplot=T. The multiplot argument only separates plots by antenna (not, e.g., by the field_id(s) specified in the fields argument). If multiplot=T, the nx and ny arguments can be used to specify the number of plots per page.
At the moment, only one polarization can be plotted per execution. This restriction will be relaxed in the near future.
For B solutions, the plotting will loop over timestamps (if more than one).
A hardcopy plot can be created by specifying the psfile argument (which is especially useful for batch processing when a display screen is not available). This will cause the plot to be written to a PostScript file which can be subsequently sent to a printer.

Example

```
c:=calibrater('3C273XC1.MS');
c.plotcal(plottype='PHASE', tablename="gcal", antennas=[1,3], polarization=2);
c.close();
```

calibrater.close - Function

Package	synthesis
Module	calibrater
Tool	calibrater

Close the calibrater tool

Synopsis
close()

Returns
Bool

Description
Close the calibrater tool, which is hardly ever necessary.

Example

```
c:=calibrater('3C273XC1.MS');
c.correct();
c.close();
```

---

calibrater.done - Function

Package	synthesis
Module	calibrater
Tool	calibrater

Destroy the calibrater tool

Synopsis
done()

Returns
Bool

Description
This function destroys the calibrater tool, writing data to disk first.

Example

```
c:=calibrater('3C273XC1.MS');
c.correct();
c.done();
```

1.3 componentmodels - Module

A module for the manipulation of model components
include "componentmodels.g"

Description
The ''component models'' module contains tools and functions which define and manipulate components. Components represent the sky brightness as a function of position on the sky and observing frequency. The main tool in this module is the componentlist tool which provides functions for manipulating groups of components.
Components are a complimentary alternative to representing the sky brightness with an image. They are useful alternative when:

- The field contains weak features in the presence of very bright features. Often residual errors from these bright features prevent the detection or analysis of the weak features. One way to avoid this is to model the bright features using a componentlist and then subtract them in the (u, v) domain.
- The data is limited in quantity and/or contains significant errors or noise. Imaging under these circumstances is difficult and often the user must resort to modeling the sky brightness to reduce the number of free parameters and incorporate additional prior information.

Other tools within aips++, use components and aid in the above tasks. The two most important are:

imager This tool can subtract componentlists in the (u, v) domain.

imagefitter This tool can generate componentlists by fitting to specified regions of an image.

Component Description

Components have a number of properties which can be categorised as relating to either its flux, shape or spectrum.

Flux Properties A fundamental property of a component is its flux or integrated intensity[1]. This can be measured in any units which are dimensionally equivalent to the Jansky ($W/m^2/Hz$). The flux of a component always has all polarisations defined and hence four numbers are used[2] to represent its value. The most common polarisation representation used is the Stokes parameters of I, Q, U, V. Two alternative representations are provided called ''circular'' and ''linear''.
When the circular representation is used the flux values are those that would be seen by detectors that are sensitive to right (R) and left (L) handed circular polarisation. They represent, in order, the RR, RL, LR, LL correlation products.
When the linear representation is used the flux values are those that would be seen if the detectors were sensitive to orthogonal linear polarisations. They represent, in order, the XX, XY, YX, YY correlation products where the parallactic angle is assumed to be zero.
When using the circular or linear representations the flux values will often be complex numbers and are always returned as such. Conversely when specifying the flux using the circular or linear representation four complex numbers are used. Unless some additional constraints are placed on these values e.g, RL = cong(LR) and imag(RR) = imag(LL) = 0, it will be possible to generate components where the flux, when converted to the Stokes representation, is not real. This is not considered an error however such components are not considered ''physical''. The is_physical function can be used to test when a component has values that could not correspond to the actual sky brightness.
When using the Stokes representation the flux values are always returned as real numbers. This value is a truncation, with the imaginary part discarded, of the internal complex value. Because the value is, regardless of the polarisation representation, stored as a complex value no information is lost when inter-converting between different polarisation representations

[1] This should *not* be confused with the peak intensity at the centre of the component.

[2] If you are not interested in polarimetry you can safely ignore three of the values used to represent the flux, provided you do not attempt to convert between different polarisation representations.

Shape Properties The shape properties of a component describe the variation of brightness as a function of the position on the sky. The shape properties are split into ones describing the functional variation and ones describing the position of a reference point on the sky. Components can have one of three shapes; point, Gaussian or disk. For all these shapes the reference point, which is a direction measure, defines the direction of the 'centre' of the component.

Both the disk and Gaussian shapes need additional parameters to completly define their shape. These parameters are the width of the major axis, the width of the minor axis and the position angle of the major axis. All these parameter are specified with angular quantities, and the width is the full-width at half-maximum. The major axis is constrained to being no smaller that the minor axis. The position angle is, following the standard astronomical convention, zero when the major axis is aligned North-South and increases when the northern tip of the major axis rotates to the East.

A point component is a special case of a Gaussian or a disk component with a very small major and minor axis width. It is treated separately both because it is very common in astronomy and because the knowledge that that the widths are infinitesimal allows important assumptions to be made when manipulating these components. The major axis, minor axis and position angle parameters are discarded for point components.

Spectral Properties The spectral properties of a component describe the variation of flux as a function of the observing frequency/wavelength. The spectral properties are split into ones describing the flux variation and ones describing a reference frequency. At the reference frequency the flux of the component is the value defined by the flux properties described above. At other frequencies it may differ. The reference frequency has a reference frame associated with it ie., it is a frequency measure. The frequency variation of a component can either be constant, or vary with a power law ie., it has a spectral index.

If the spectral properties specify that the flux varies depending on a spectral index an additional parameters is required. This is the spectral index (α). Then the spectral variation is then described by:

$$\begin{aligned} I &= I_0 \left(\frac{\nu}{\nu_0}\right)^\alpha \\ Q &= Q_0 \left(\frac{\nu}{\nu_0}\right)^\alpha \end{aligned}$$