```
# get the total integration time
- bms.totalintegrationtime();
[value=2044.13995, unit=s]
```

1.4 bimacalibrater - Module

Module for calibrating BIMA data
include "bimacalibrater.g"

Description

The `bimacalibrater` module provides functions for calibrating BIMA data. The facilities of the `bimacalibrater` module are made available in Glish by issuing the following command:

```
- include 'bimacalibrater.g'
T
```

where a hyphen precedes user input. The Glish response is indicated without the prompt.
A `bimacalibrater` tool is created and attached to a specified set of measurement sets as indicated in the following example:

```
- bc := bimacalibrater(['sgrb2.ms','sgra.ms'],'1733-130.ms',
+ targetnames=['sgrb','sgra'],phcalnames='1733')
```

Many functions of `bimacalibrater` are wrappers around corresponding functions in `calibrater`. One important exception is `transfer()` which transfers the solutions from one calibration table (usually the one written by `solve()`) to another table (generally the one that will be applied to the target source with `correct()`). This is generally a necessary step because, for BIMA data, the calibration solutions are determined for a small set of spectral windows (e.g., the two sideband averages [the wideband data in miriad parlence]), but these solutions must be applied to all the windows in the target datasets. The only way to do this in AIPS++ is to create a new calibration table which reflects the number of spectral windows in the target sources. Essentially, a `bimacalibrater` tool creates (on demand) and stores `calibrater` tools attached to the Measurement Sets specified in the `bimacalibrater` tool constructor. This tool's various functions manipulate these `calibrater` tools. In general, the functions take a parameter called `sources` which tells the function which `calibrater` tools it should manipulate. The `sources` parameter takes a vector of strings. The strings are source nicknames (see below) and/or source roles. Recognized roles are `'all'` (all sources), `'targets'` (target sources), `'phcals'` (phase calibrators),

'pbcals' (passband calibrators), and 'fcals' (flux calibrators). The source roles are specified by the user when the tool is constructed (see below).
Here is an example of what a typical calibration session using bimacalibrater might look like:

```
bc := bimacalibrater('sgrb2.ms','1733-130.ms',targetnames='narrow',
   phcalnames='wide');
bc.setjy(fluxdensity=[5,0,0,0]);
bc.setsolve(type='G',refant=4,table='gcal',t=600);
bc.solve();
bc.fit();
bc.transfer('gcal.xfer','gcal');
bc.setapply();
bc.correct();
```

Here, a bimacalibrater tool is constructed by specifying a list of target sources ('sgrb2'), nicknames for these sources ('narrow'), a list of phase calibrators ('1733-130.ms'), and a list of nicknames for these calibrators ('wide'). The nicknames are optional and are meant to provide an easy way of referring to datasets (sometimes the names of measurement sets can be long, especially when fully qualified paths must be specified). The nicknames need not be provided, in which case they are set to the names of the corresponding measurement sets.
The flux density scale of the calibrater is set with setjy (which is actually a wrapper of the imager function of the same name). This function will optinally look up the flux density of the calibrator from the BIMA calibrator flux density table and write the flux desnity nearest in time to the observations to the measurement set. Next, the solver is set up using setsolve() and solutions are generated using solve(). Both are wrappers around calibrater functions of the same names.
The fit() function launches the gainpolyfitter GUI which allows one to interpolate the solutions by fitting polynomials to them.
The transfer() function is the reason bimacalibrater exists. The usual procedure of calibrating BIMA data is to generate solutions for the sideband average data of the calibrator. Solutions from these two windows are then applied to all the windows of target source. However, in AIPS++ the number of windows in the calibration table must be the same as the number in the dataset to which it is being applied. Therefore, the solutions in the gcal table, which contains only two windows, must be transferred to a new table gcal.xfer which has the same number of windows as the target source dataset (e.g. 10 [2 sideband average windows + 8 channel data windows]). Once this is done, further information on how to apply the solutions can be provided with setapply() and finally the solutions can be applied with correct().

Tools

bimacalfluxtable	Tool to retrieve information from the BIMA calibrator tables
bimacalibrater	Calibration of BIMA data

Functions

bimacalibratertest	Test bimacalibrater

1.4.1 bimacalibrater - Tool

Package bima
Module bimacalibrater

Calibration of BIMA data

Description

A `bimacalibrater` tool performs all the necessary steps for calibrating BIMA data. It is meant to be a "track-based" tool; that is, it can hold all the datasets observed in a track and use them appropriately in various calibration steps.

Constructors

bimacalibrater	Construct a bimacalibrater tool from a set of MeasurementSets

Functions

addtargets	Add target sources
close	Close the calibrater tools associated with the specified measurement sets.
correct	Apply calibration
done	Done with this tool
fit	Interpolate gain solutions by fitting polynomials
getsourcenames	get a list of source nicknames
plotcal	Plot a calibration table(s)
setapply	Set how solutions should be applied
setdata	Set the data parameters selection for subsequent processing
setjy	Set the flux density of the specified sources
setsolve	Set up what calibration component is to be solved for and how the solution should be determined.
solve	Solve for the selected calibration components
summary	Print a summary of this tool.
transfer	Transfer calibration solutions from one table to another.
type	Get the type of this tool, namely 'bimacalibrater'

bimacalibrater - Constructor

Package bima
Module bimacalibrater
Tool bimacalibrater

Construct a bimacalibrater tool from a set of MeasurementSets

Synopsis
bimacalibrater(targets, phcals, pbcals, fcals, targetnames, phcalnames, fcalnames, pbcalnames)

Description
Create a `bimacalibrater` tool.

Arguments

targets	Target source measurement set names Allowed:Vector of strings Default: ' '
phcals	Phase calibrator measurement set names Allowed:Vector of strings Default: ' '
pbcals	Passband calibrator measurement set names Allowed:Vector of strings Default: ' '
fcals	Flux density calibrator measurement set names Allowed:Vector of strings Default: ' '
targetnames	Target source nicknames Allowed:Vector of strings Default: unset (will set to targets)
phcalnames	Phase calibrator nicknames Allowed:Vector of strings Default: unset (will set to phcals)
fcalnames	Flux density calibrator nicknames Allowed:Vector of strings Default: unset (will set to fcals)
pbcalnames	Passband calibrator nicknames Allowed:Vector of strings Default: unset (will set to pbcals)

Returns
bimacalibrater tool

Example

```
# the '+' is printed by glish, not typed by you
- bc := bimacalibrater(['sgrb2.ms','sgra.ms'],
+ '1733-130.ms',targetnames=['sgrb','sgra'],phcalnames='1733');
```

---

bimacalibrater.getsourcenames - Function

Package	**bima**
Module	**bimacalibrater**
Tool	**bimacalibrater**

get a list of source nicknames

Synopsis
getsourcenames(roles)

Description

The `getsourcenames` function returns a list of source nicknames associated with the specified roles. The `roles` parameter should be a vector of strings, recognized values being `'all'`, `'targets'`, `'phcals'`, `'pbcals'`, `'fcals'` for all sources, target sources, phase calibrators, passband calibrators, and flux density calibrators, respectively. This function is meant to help users remember how they have named their sources (it is also used internally by other functions of `bimacalibrater`).

Arguments

roles	Return nicknames of sources of these roles Allowed: Vector of strings. Recognized values are 'all' (all nicknames), 'targets' (target source nicknames), 'phcals' (phase calibrators), 'pbcals' (passband calibrators), and 'fcals' (flux density calibrators) Default: none

Returns
array of corresponding nicknames or fail

Example

```
- bc := bimacalibrater(['sgrb2.ms','sgra.ms'],
+ '1733-130.ms',targetnames=['sgrb','sgra'],phcalnames='1733');
- bc.getsourcenames('all')
sgrb sgra 1733
- bc.getsourcenames(['phcals','targets'])
1733 sgrb sgra
- bc.getsourcenames('targets')
sgrb sgra
- bc.getsourcenames('phcals')
1733
```

---

bimacalibrater.setjy - Function

Package	**bima**
Module	**bimacalibrater**
Tool	**bimacalibrater**

Set the flux density of the specified sources

Synopsis
setjy(sources, fieldid, spwid, fluxdensity)

Description

The `setjy` function is a wrapper around the `imager` tool's setjy function. It sets the flux density (by modifying the MODEL column) of the specified measurement sets. A list of nicknames (as well the the special groupings allowed in `getsourcenames`) can be given. If the specified flux density is <0, this function will attempt to find the calibrator in the BIMA calibrator flux density table and if successful, will use the flux density nearest in time (no attempt to adjust the flux density for frequency is currently made, nor is any interpolation in time attempted). In this case, there can only be a single source per measurement set. In the case where the user sets the flux density explicitly by passing a > 0 value to this function, only single model can be specified (it will be used for all the specified sources). Thus, you must run this function multiple times if you have calibrators of different flux densities when running in this mode.

Arguments

sources	Nicknames of sources to which to set the flux densities Allowed:Vector of strings (nicknames or roles of sources) Default: 'phcals'
fieldid	Field Id (1-relative) Allowed:Int Default: -1 (all)
spwid	Spectral Window Id. (1-relative) Allowed:Int Default: -1 (all)
fluxdensity	Specified flux density (I,Q,U,V) in Jy, if < 0, attempt to look up flux density in BIMA calibrator flux density table Allowed:Vector of Doubles Default: -1

Returns
T or fail

Example

```
- bc := bimacalibrater(['sgrb2.ms','sgra.ms'],
+ '1733-130.ms',targetnames=['sgrb','sgra'],phcalnames='1733');
- # all of these are equivalent for setting the I flux density to 5 Jy.
- bc.setjy(fluxdensity=[5,0,0,0]);
T
- bc.setjy('1733',fluxdensity=[5,0,0,0]);
T
- bc.setjy('phcals',fluxdensity=[5,0,0,0]);
T
- bc.setjy(spwid=-1,fluxdensity=[5,0,0,0]);
T
# here we allow a lookup in the BIMA calibrator flux density table
- bc.setjy();
NORMAL: Using 2.79999995 Jy at 86.1999969 GHz measured on 1998/01/04/00:00:00.000
```

---

bimacalibrater.setdata - Function

Package	**bima**
Module	**bimacalibrater**
Tool	**bimacalibrater**

Set the data parameters selection for subsequent processing

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

Description
This function is a wrapper around the `calibrater` tool's `sedata` function. The difference being that multiple sources can be specified, although the specified data selection will be the same for all of them. The `sources` parameter takes a list of source nicknames and/or a list of roles (acceptable roles are `'all','targets','phcals','pbcals','fcals'`; see `getsourcenames` for details). The `sourcenames` parameter allows selection of sources in multisource datasets. These names are read from the SOURCE subtable of the Measurement Set and the appropriate FIELD_ID selection is appended to the `msselect` string.

Arguments

sources	Nicknames of sources on which to run calibrater.setdata() Allowed: Vector of strings (nicknames or roles of sources) Default: 'phcals'
mode	Type of processing: channel or velocity Allowed: 'channel'|'velocity'
nchan	Number of channels to select Allowed: Int Default: 1
start	Start channel (1-relative) Allowed: Int
step	Step in channel number Allowed: Int
mstart	Start velocity (e.g. '20Km/s') Allowed: Quantity
mstep	Step in velocity (e.g. '100m/s' Allowed: Quantity
uvrange	UV range (in klambda) (e.g. [0, 50]) Allowed: Vector of Doubles Default: 0 (all data)
sourcenames	List of sources to select (only useful for multisource datasets; see above Allowed: Vector of strings Default: ' '
msselect	TAQL selection string Allowed: String

Returns
T or fail

Example

```
- bc := bimacalibrater(['sgrb2.ms','sgra.ms'],
+ '1733-130.ms',targetnames=['sgrb','sgra'],phcalnames='1733');
- bc.setjy(fluxdensity=[5,0,0,0]);
T
- bc.setdata('1733',mode='channel',uvrange=[10,50]);
T
```

---

bimacalibrater.setsolve - Function

Package **bima**
Module **bimacalibrater**
Tool **bimacalibrater**

Set up what calibration component is to be solved for and how the solution should be determined.

Synopsis

setsolve(sources, type, t, preavg, phaseonly, refant, table, append)

Description

The `setsolve` function is a wrapper around the `calibrater` tool's `setsolve` function. It allows the user to specify what calibration component is to be solved for as well as how the solution should be determined.

If the default value for the solution is specified, the following algorithm is used. The table is given the name `<solution type>cal` (e.g. `gcal` for `G` solutions). If this table already exists, then a number is appended (e.g., `gcal.2`). The table name is stored internally in the tool for use with other functions.

Arguments

sources	Nicknames of sources on which to run calibrater.setsolve() Allowed:Vector of strings (nicknames or roles of sources) Default: 'phcals'
type	Component type Allowed:"G","T","B","D" Default: "
t	Solution interval (in sec) Allowed:Float Default: 0.0
preavg	Pre-averaging interval (in sec) Allowed:Float Default: 0.0
phaseonly	Phase only solution ? Allowed:Bool Default: False
refant	Reference antenna Allowed:Int Default: -1
table	Output calibration table Allowed:Table name Default: " (see above for default naming scheme)
append	Append flag (set to T if you have multiple calibraters and want solutions in a single table) Allowed:Bool Default: False

Returns

Bool or fail

Example

```
# a single calibrater
- bc := bimacalibrater(['sgrb2.ms','sgra.ms'],
+ '1733-130.ms',targetnames=['sgrb','sgra'],phcalnames='1733');
- bc.setjy(fluxdensity=[5,0,0,0]);
T
- bc.setdata('1733',mode='channel');
T
- bc.setsolve('1733','G',60,refant=4,table='gcal');
T
```

```
# for two calibraters, putting all solutions in a single table
- bc := bimacalibrater('g34.3.ms',['1751+096.ms','2148+069.ms'],
  targetnames='src',phcalnames=['cal1','cal2']);
T
- bc.setjy(fluxdensity=[5,0,0,0]);
T
# solutions for both calibraters will go in table gcal2
- bc.setsolve(type='G',refant=4,table='gcal2',t=600,append=T);
T
```

---

bimacalibrater.solve - Function

Package	**bima**
Module	**bimacalibrater**
Tool	**bimacalibrater**

Solve for the selected calibration components

Synopsis
solve(sources)

Description

The `solve` function wraps the `calibrater` tool's `solve` function. Solutions are generated and a calibration table is written for the specified sources. In addition, a keyword called `SPW_INFO` is written to the calibration table. This keyword contains information on the spectral window configuration of the source being solved for. This information is then read by the `transfer` function to help it make an educated guess about how the windows in the calibration written by `solve` map to the spectral windows in the sources to which the calibration will be applied.

Arguments

sources		Nicknames of sources on which to run calibrater.solve()
	Allowed:	Vector of strings (nicknames or roles of sources)
	Default:	'phcals'

Returns
Bool or fail

Example

```
- bc := bimacalibrater(['sgrb2.ms','sgra.ms'],
+ '1733-130.ms',targetnames=['sgrb','sgra'],phcalnames='1733');
- bc.setjy(fluxdensity=[5,0,0,0]);
T
- bc.setdata('1733',mode='channel');
T
- bc.setsolve('1733','G',60,refant=4,table='gcal');
T
- # bc.solve('phcals') or bc.solve('1733') are equivalent to the following
- bc.solve()
T
```

---

bimacalibrater.fit - Function

Package	**bima**
Module	**bimacalibrater**
Tool	**bimacalibrater**

Interpolate gain solutions by fitting polynomials

Synopsis
fit(table)

Description

The `fit()` function creates a `gainpolyfitter` tool which allows the user to interpolate calibration solutions by fitting polynomials to them. This function lauches the GUI and then returns, which means you must be sure to complete the fits you want before proceeding to the next calibration step.

Arguments

table	The calibration table for which solutions are to be interpolated Allowed:Table name Default: unset (uses the table specified in setsolve())

Returns
T or fail

Example

```
- bc := bimacalibrater(['sgrb2.ms','sgra.ms'],
+ '1733-130.ms',targetnames=['sgrb','sgra'],phcalnames='1733');
- bc.setjy(fluxdensity=[5,0,0,0]);
T
- bc.setdata('1733',mode='channel');
T
- bc.setsolve('1733','G',60,refant=4,table='gcal');
T
- bc.solve()
T
- bc.fit()
T
```

---

bimacalibrater.transfer - Function

Package	**bima**
Module	**bimacalibrater**
Tool	**bimacalibrater**

Transfer calibration solutions from one table to another.

Synopsis
transfer(outtable, intable, spwmap, calibratees, forcecopy)

Description

The `transfer()` function is the reason `bimacalibrater` exists. It transfers solutions from one calibration table with `m` spectral windows to another table with `n` spectral windows (in general, `m != n`).
Here's how the function determines how to map the windows in the input table to those in the output table. It first checks the (optional) user specified `spwmap` parameter. With this parameter, the user describes the window mapping. For example, assume the input table has two windows representing solutions for the lower and upper sidebands (windows 1 and 2). Assume the target source has the following spectral window configuration:

```
window   description
--------------------------------
  1      lower sidebnad average
  2      upper sideband average
```

```
3-6      lower sideband channel data
7-10     upper sideband channel data
```

In order to calibrate these data correctly, the user would specify `spwmap=[1,2,1,1,1,1,2,2,2,2]`. Note that for this example, `transfer` is smart enough to determine this mapping without having the `spwmap` parameter specified (see below). Furthermore, the user can opt not to have solutions for some or all windows copied by specifying a value of `0` in the correct window slot of `spwmap`. If `spwmap` is not specified, `transfer` uses the following algorithm. If there is only a single spectral window in the input table, its solutions are copied to all windows of the output. If there are two windows, `transfer` assumes they represent lower and upper sideband solutions. It tries to determine which sideband corresponds to which window by reading the `SPW_INFO` keyword written by the `solve()` function of `bimacalibrater`. If it cannot find this information (e.g. if the input calibration table was written by `calibrater.solve()`), it assumes the first window contains the lower sideband solutions and the second contains the upper sidenbad solutions. It then reads the `SPECTRAL_WINDOW` table of the dataset to be corrected to determine in which sidebands its windows lie. For three or more spectral windows in the input calibration table with `spwmap` unspecified, this function will fail since there is no normal way to transfer solutions in this case. It is up to the user in this case to specify how the solutions are to be mapped.

Although effort has been made to try to make the best guesses possible, the best way to make sure you are getting what you want is to specify `spwmap`.

If the default value of `outtable` is specified, the output table name will be called <`intable`>`.xfer` (e.g., `gcal.xfer` if the input table is named `gcal`.

Arguments

outtable	Name of the calibration table to which to write solutions Allowed:Any valid table name Default: ”
intable	Name of the calibration table from which to transfer solutions Allowed:Any valid table name Default: ” (will use the table written by solve)
spwmap	Array describing the spectral window mapping from the input to the output (see above). The spectral windows are 1-based, and a value of 0 means not to transfer solutions for the specified window (GAINS are filled in with 0’s) Allowed:Vector of Ints >= 0 Default: unset (transfer will do its best to guess how this should be done; see above)
calibratees	Nicknames or roles of sources to which the solutions will be applied (if more than one source is specified they all should have the same spectral window configuration) Allowed:Vector of strings Default: ’targets’
forcecopy	Force a copy of the input to output table when the input could be used to apply solutions Allowed:Boolean Default: T

Returns
T or fail

Example

```
- bc := bimacalibrater(['sgrb2.ms','sgra.ms'],
+ '1733-130.ms',targetnames=['sgrb','sgra'],phcalnames='1733');
- bc.setjy(fluxdensity=[5,0,0,0]);
T
- bc.setdata('1733',mode='channel');
T
- bc.setsolve('1733','G',60,refant=4,table='gcal');
T
- bc.solve()
T
- bc.transfer('gcal.xfer','gcal',[1,2,1,1,1,1,2,2,2,2],'sgrb');
T
```

bimacalibrater.setapply - Function

Package	**bima**
Module	**bimacalibrater**
Tool	**bimacalibrater**

Set how solutions should be applied

Synopsis
setapply(sources, type, t, table, select)

Description

The `setapply` function is a wrapper around the `calibrater` tool's `setapply` function. It is used to set calibration component information for applied calibration. Each calibration component represents a separate calibration matrix correction included in the measurement equation.

Arguments

sources	Nicknames of sources on which to run calibrater.setapply() Allowed: Vector of strings (nicknames or roles of sources) Default: 'targets'
type	Component type Allowed: "G","T","B","D" Default: 'G'
t	Interpolation interval (in sec) Allowed: Float Default: 0.0
table	Input calibration table Allowed: String Default: unset (will use table written by transfer(), or if transfer() hasn't been run, the table written by solve()
select	TAQL selection string Allowed: String Default: " "

Returns
Boolean or fail

Example

```
- bc := bimacalibrater(['sgrb2.ms','sgra.ms'],
+ '1733-130.ms',targetnames=['sgrb','sgra'],phcalnames='1733');
- bc.setjy(fluxdensity=[5,0,0,0]);
T
- bc.setdata('1733',mode='channel');
T
- bc.setsolve('1733','G',60,refant=4,table='gcal');
T
- bc.solve()
T
- bc.transfer('gcal.xfer','gcal',[1,2,1,1,1,1,2,2,2,2],'sgrb');
T
- # bc.solve() is equivalent to the next line for this example
- bc.solve('sgrb','G',0,'gcal.xfer');
T
```

---

bimacalibrater.correct - Function

Package	**bima**
Module	**bimacalibrater**
Tool	**bimacalibrater**

Apply calibration

Synopsis
correct(sources)

Description

This function is a wrapper around the `calibrater` tool's `correct()` function. It applies existing calibration data to the specified measurement sets, with initial table names and interpolation intervals set by the `setapply` function.

Arguments

sources	Nicknames of sources on which to run calibrater.correct() Allowed: Vector of strings (nicknames or roles of sources) Default: 'targets'

Returns
Boolean or fail

Example

```
- bc := bimacalibrater(['sgrb2.ms','sgra.ms'],
+ '1733-130.ms',targetnames=['sgrb','sgra'],phcalnames='1733');
- bc.setjy(fluxdensity=[5,0,0,0]);
T
- bc.setdata('1733',mode='channel');
T
- bc.setsolve('1733','G',60,refant=4,table='gcal');
T
- bc.solve()
T
- bc.transfer('gcal.xfer','gcal',[1,2,1,1,1,1,2,2,2,2],'sgrb');
T
- bc.solve('sgrb','G',0,'gcal.xfer');
T
# bc.correct() is equivalent to the next line
- bc.correct('sgrb');
T
```

---

bimacalibrater.type - Function

Package	**bima**
Module	**bimacalibrater**
Tool	**bimacalibrater**

Get the type of this tool, namely 'bimacalibrater'

Synopsis
type()

Description

This function simply returns the string `'bimacalibrater'`.

Returns
'bimacalibrater'

Example

```
- bc := bimacalibrater(['sgrb2.ms','sgra.ms'],
+ '1733-130.ms',targetnames=['sgrb','sgra'],phcalnames='1733');
- bc.type();
bimacalibrater
```

---

bimacalibrater.addtargets - Function

Package	**bima**
Module	**bimacalibrater**
Tool	**bimacalibrater**

Add target sources

Synopsis
addtargets(mss, names)

Description

This function adds target sources to the list of targets.

Arguments

mss	Names of the target MSs Allowed:Vector of strings Default: None
names	Nicknames for the target MSs Allowed:Vector of strings Default: unset (will use the values supplied in the mss parameter

Returns
Boolean or fail

Example

```
- bc := bimacalibrater([,,'1733-130.ms',phcalnames='1733');
- bc.setjy();
T
- bc.setdata('1733',mode='channel');
T
- bc.setsolve('1733','G',60,refant=4,table='gcal');
T
- bc.solve()
T
- bc.addtargets('sgrb2.ms','sgrb2');
T
```

---

bimacalibrater.done - Function

Package	**bima**
Module	**bimacalibrater**
Tool	**bimacalibrater**

Done with this tool

Synopsis
done()

Description

Release resources used by this tool.

Returns
Boolean T

Example

```
- bc := bimacalibrater(['sgrb2.ms','sgra.ms'],
+ '1733-130.ms',targetnames=['sgrb','sgra'],phcalnames='1733');
- bc.done();
T
```

---

bimacalibrater.close - Function

Package	**bima**
Module	**bimacalibrater**
Tool	**bimacalibrater**

Close the calibrater tools associated with the specified measurement sets.

Synopsis
close(sources)

Description

This function is a wrapper around the `calibrater` tool's `close()` function. It closes the calibrater tools (if they exist) associated with the specifed measurement sets.

Arguments

sources	Nicknames of sources on which to run calibrater.close() Allowed: Vector of strings (nicknames or roles of sources) Default: none

Returns
Boolean T or fail

Example

```
- bc := bimacalibrater(['sgrb2.ms','sgra.ms'],
+ '1733-130.ms',targetnames=['sgrb','sgra'],phcalnames='1733');
- bc.setjy(fluxdensity=[5,0,0,0]);
T
- bc.setdata('1733',mode='channel');
T
- bc.setsolve('1733','G',60,refant=4,table='gcal');
T
- bc.solve()
T
- bc.transfer('gcal.xfer','gcal',[1,2,1,1,1,1,2,2,2,2],'sgrb');
```

```
T
- bc.solve('sgrb','G',0,'gcal.xfer');
T
- bc.correct('sgrb');
T
- bc.close('sgrb');
T
```

---

bimacalibrater.plotcal - Function

Package	**bima**
Module	**bimacalibrater**
Tool	**bimacalibrater**

Plot a calibration table(s)

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

Description

This function is a wrapper around the `calibrater` tool's `plotcal()` function which plots calibration tables. `calibrater.plotcal()` is run seperately on each source specified by the `sources` parameter.
The following plottype choices are currently available 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 following plottype choices are currently available for D solutions:

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.

Arguments

sources	Nicknames of sources on which to run calibrater.plotcal() Allowed:Vector of strings (nicknames or roles of sources) Default: none
plottype	Plot type Allowed:String Default: 'AMP'
tablename	Calibration table name Allowed:String Default: ''
antennas	Antennas to plot Allowed:Vector of Ints Default: False
fields	Fields to plot Allowed:Vector of Ints Default: False
polarization	Polarization to plot Allowed:Integer Default: 1
spwids	Spectral windows id.'s to plot Allowed:Vector of Ints Default: False
timeslot	Timeslot index for B Jones Allowed:Integer Default: 1
multiplot	Turn on separate antenna plots Allowed:Bool Default: F
nx	If multiplot=T, number of plots on horizontal axis Allowed:Integer Default: 1
ny	If multiplot=T, number of plots on vertical axis Allowed:Integer Default: 1
psfile	name of output PostScript file to write plot to Allowed:String Default: empty string, causing plot to be sent to the screen

Returns
Boolean or fail

Example

```
- bc := bimacalibrater(['sgrb2.ms','sgra.ms'],
+ '1733-130.ms',targetnames=['sgrb','sgra'],phcalnames='1733');
- bc.setjy(fluxdensity=[5,0,0,0]);
T
- bc.setdata('1733',mode='channel');
T
- bc.setsolve('1733','G',60,refant=4,table='gcal');
T
- bc.solve()
T
- bc.plotcal('1733',plottype='phase',table='gcal',antennas=[1,3]);
T
- bc.close('1733');
T
```

---

bimacalibrater.summary - Function

Package	**bima**
Module	**bimacalibrater**
Tool	**bimacalibrater**

Print a summary of this tool.

Synopsis
summary()

Description

This function prints a summary of the current state of the tool.

Returns
Boolean T

Example

```
- bc := bimacalibrater(['sgrb2.ms','sgra.ms'],
+ '1733-130.ms',targetnames=['sgrb','sgra'],phcalnames='1733');
- bc.summary()
```

```
<lots of state information gets printed>
T
```

1.4.2 bimacalfluxtable - Tool

Package	**bima**
Module	**bimacalibrater**

Tool to retrieve information from the BIMA calibrator tables

Description

A `bimacalfluxtable` allows for retrieval of information from the BIMA calibrator flux tables in the AIPS++ data repository.

Constructors

bimacalfluxtable	Construct a bimacalfluxtable tool

Functions

close	Close the table tools associated with this tool
contains	does the table contain the specified calibrator?
done	Done with this tool
fluxlist	Get the flux history for the specified calibrator
fluxtable	Get the table containing the flux histories of calibrators
nearestflux	Get the flux nearest in time to the specifed date of the specified calibrator.
positiontable	Get, as a table tool, the position table (J2000)
type	Get the type of this tool, namely 'bimacalfluxtable'

bimacalfluxtable - Constructor

Package	**bima**
Module	**bimacalibrater**
Tool	**bimacalfluxtable**

Construct a bimacalfluxtable tool

Synopsis

bimacalfluxtable()

Description
Create a `bimacalfluxtable` tool.

Arguments

Returns
bimacalfluxtable tool

Example

```
# the '+' is printed by glish, not typed by you
- bcft := bimacalfluxtable();
```

---

bimacalfluxtable.contains - Function

Package	**bima**
Module	**bimacalibrater**
Tool	**bimacalfluxtable**

does the table contain the specified calibrator?

Synopsis
contains(calname)

Description

This function returns true if the specified calibrator is contained in the position (J2000) table.

Arguments

calname	The calibrator to determine the existance of Allowed:String Default: none

Returns
Boolean

Example

```
- bcft := bimacalfluxtable();
- bcft.contains('1733-130');
T
- bcft.contains('somedummycal');
F
```

---

bimacalfluxtable.fluxlist - Function

Package	**bima**
Module	**bimacalibrater**
Tool	**bimacalfluxtable**

Get the flux history for the specified calibrator

Synopsis
fluxlist(calname)

Description

This function retrieves, as a record, the flux history for the specified calibrator. The fields in the resulting record are `obsdate` (in fiducial days), `flux` (in Jy), `rms` (in Jy), and `freq` (in GHz).

Arguments

calname	Name of the calibrator for which to get the flux history Allowed:String Default: none

Returns
a record containing the history or fail

Example

```
- bcft := bimacalfluxtable();
- r := bcft.fluxlist('1733-130');
- r.flux;
[5.5999999 6.30000019 5.5 5.5999999 6 4.5999999 6 5.69999981 6.4000001
6.80000019 8.19999981 8.69999981 11.6000004 11 9.5 10 11 10 10.1999998
10.3000002 11.6000004 6.5 2.79999995 3.20000005 2.5999999 2.5
2.20000005 2.0999999 2.29999995 2.5 2.5999999 2.9000001 2.79999995
3.4000001 3.5 3.20000005 4.30000019 4.30000019]
- r.obsdate[5];
49250
```

---

bimacalfluxtable.fluxtable - Function

Package	**bima**
Module	**bimacalibrater**
Tool	**bimacalfluxtable**

Get the table containing the flux histories of calibrators

Synopsis
fluxtable()

Description

This function returns as a table tool the table containing calibrator flux density history.

Arguments

Returns
Table tool

Example

```
- bcft := bimacalfluxtable();
- fluxtable := bcft.fluxtable();
```

---

bimacalfluxtable.nearestflux - Function

Package	**bima**
Module	**bimacalibrater**
Tool	**bimacalfluxtable**

Get the flux nearest in time to the specifed date of the specified calibrator.

Synopsis
nearestflux(calname, date)

Description

This function returns, as a record, information about the specified calibrator closest to the specified date. The field names of the record returned are `obsdate` (in fiducial days), `flux` (in Jy), `rms` (in Jy), and `freq` (in GHz).

Arguments

calname	The calibrator for which to retrieve information Allowed:String Default: None
date	The date to which to get the nearest-in-time record Allowed:Quantity Default: None

Returns
Record or fail

Example

```
- bcft := bimacalfluxtable();
- bcft.nearestflux('1733-130',dq.quantity('today'));
[obsdate=52328, flux=4.30000019, rms=0.600000024, freq=86.1999969]
- bcft.nearestflux('1733-130',dq.quantity('01jan2000'));
[obsdate=51605, flux=2.0999999, rms=0.5, freq=86.1999969]
```

---

bimacalfluxtable.positiontable - Function

Package	**bima**
Module	**bimacalibrater**
Tool	**bimacalfluxtable**

Get, as a table tool, the position table (J2000)

Synopsis
positiontable()

Description

This function returns as a table tool the position table of BIMA calibrators.

Arguments

Returns
Table tool

Example

```
- bcft := bimacalfluxtable();
- postable := bcft.positiontable();
```

---

bimacalfluxtable.type - Function

Package	**bima**
Module	**bimacalibrater**
Tool	**bimacalfluxtable**

Get the type of this tool, namely 'bimacalfluxtable'

Synopsis
type()

Description

This function simply returns the string `'bimacalfluxtable'`.

Returns
'bimacalfluxtable'

Example

```
- bcft := bimacalfluxtable();
- bcft.type();
bimacalfluxtable
```

---

bimacalfluxtable.done - Function

Package	**bima**
Module	**bimacalibrater**
Tool	**bimacalfluxtable**

Done with this tool

Synopsis
done()

Description

Release resources used by this tool.

Returns
Boolean T

Example

```
- bcft := bimacalfluxtable();
- bcft.done();
T
```

---

bimacalfluxtable.close - Function

Package	**bima**
Module	**bimacalibrater**
Tool	**bimacalfluxtable**

Close the table tools associated with this tool

Synopsis
close()

Description

This function closes the table tools associated with this tool.

Arguments

Returns
Boolean T

Example

```
- bcft := bimacalfluxtable();
- bcft.close();
T
```

1.4.3 bimacalibratertest - Function

Package bima
Module bimacalibrater

Test bimacalibrater

Synopsis
bimacalibratertest(verbose, modelcaltable, modeldatatable)

Description
`bimacalibratertest` is a function to test bimacalibrater. It uses datasets in the AIPS++ data repository. The basic strategy of the tests is to fill `Miriad`

datasets (a target and calibrator source) and calibrate them. The resulting calibration table is compared to a fiducial table, and similarly the `MODEL_DATA` and `CORRECTED_DATA` columns of the calibrated target dataset are compared to a fiducial dataset. The function returns `T` if all tests pass, `F` otherwise. The filling, reaveraging, calibration, and tests take 10 minutes on an unloaded Linux system with a 1.5 GHz Pentium 4 processor and a 7200 rpm IDE hard drive (many portions of the function are I/O bound). The function will create a scratch directory in the current directory called `bimacalibratertester_work` where it will write and manipulate datasets. The datasets used are rather large, and so about 350 MB of disk space is required.

Arguments

verbose	in	Controls the amount of logger output, 0=none, higher numbers=more Allowed:Int ¿= 0 Default: 1
modelcaltable	in	Use this table as the fiducial table for comparison with the generated calibration table Allowed:Any valid table name Default: ” (uses table in the data repository)
modeldatatable	in	Use this table as the fiducial table for comparison with the generated MODEL_DATA and CORRECTED_DATA columns Allowed:Any valid table name Default: ” (uses table in the data repository)

Returns
Boolean or fail

Example

```
- include 'bimacalibrater.g';
- bimacalibratertest();
<Lots of messages regarding filling, reaveraging, calibration, and testing>
T
```