# The GALFA-HI Standard Reduction pipeline version 2.6: ad astra per aspera

Joshua E. G. Peek (né Goldston)

July 14, 2009

## 1 Introduction

The purpose of this document is to acquaint anyone who intends on spending a good bit of time processing, calibrating and gridding GALSPECT data with the details of the GALFA-HI Standard Reduction (GSR) package, as written in IDL. The *GALFA User's Guide* has more information about observations and other aspects of the data pipeline. Certainly it would do a new user good to examine *The GALFA User's Guide* before delving into this document, although at this writing, this document is significantly more up-to-date. On the other end of the practicality to theory spectrum, Peek & Heiles (2008; http://arxiv.org/abs/0810.1283) contains a thorough description of the principles behind the data reduction process and explains the scale and type of residual systematics and pitfalls. It is somewhat out of date at present, as the code has been revised since its publication, though all of the principles discussed in that work do still hold true.

A note on IDL: Almost the entire codebase is written in IDL, a proprietary language licensed by ITT. Operating this reduction software without a rudimentary knowledge of IDL would be difficult. Like performing a musical in a language you don't speak, you can memorize the words, but if you get lost, well, it gets ugly. One need not be an IDL ninja, but understanding procedures, functions, keywords, arrays and structures would be very useful. For a good IDL tutorial, follow links from `http://astron.berkeley.edu/~ heiles`.

## 2 The General Overview

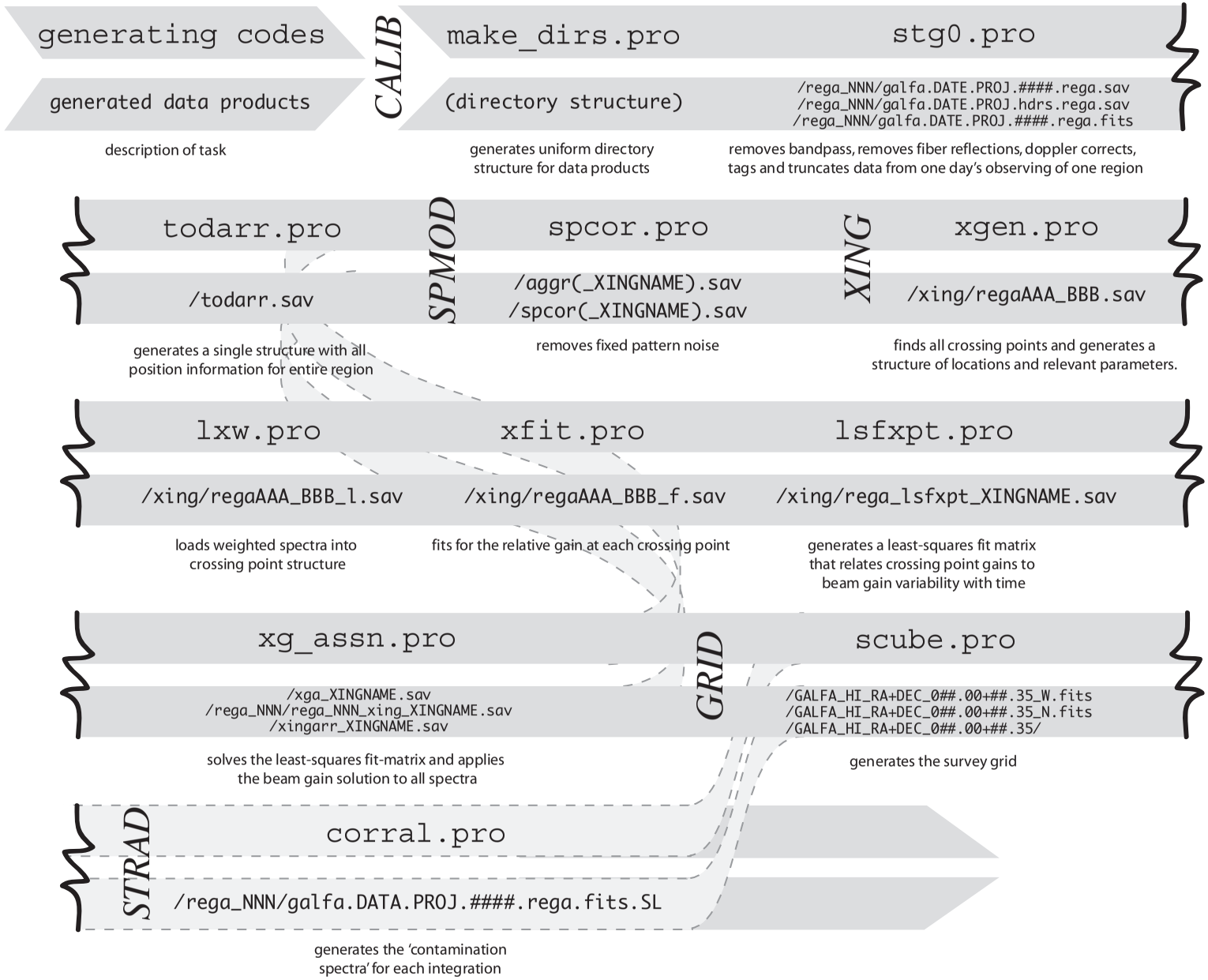
The goal of this data processing package is to take the time-ordered data (TOD) that comes out of GALSPECT, the dedicated GALFA-HI spectrometer, over a *single region* and turn it into a calibrated, gridded spectral (PPV) data cube. Note that this version of the data reduction pipeline is functional for GALSPECT data taken with any observing mode, and is no longer restricted to modes monotonic in RA. This includes Basketweave, Drift, Boustrophodonic (RA/Dec) winking-cal/TOGS II modes. The guts of the first steps in this process are handled by a suite of programs designed by C. Heiles. These programs deal with the Least-Squares Frequency Switching (LSFS)

and header (HDR) areas of the data reduction; they do a lot of corrections to individual data sets. These programs are explained in gory detail in *GSR/PROCS/INIT/HDR AND /INIT/LSFS SOFTWARE* as well as *GENERATE MH AND LSFS FILES* and I refer the interested reader to those documents. The pipeline being described in *this* document calls upon the C. Heiles suite of codes to do the first step of reduction, but is mainly concerned with reducing blocks of data together to make maps. Note, then, that it is not necessary to do any reduction before running the suite of codes described in this document, although the by-product of data checking routines, such as the lsfs and mh files, can be used in this reduction - they are identical to some of the products of stg0, and so some computer time can be saved by copying these to the appropriate directory. Also note that the Archiver (see documents on this topic by M. Krco), will generate mh files that can be used in this process as well. At Arecibo these can be found in */share/galfa/galfamh/*. Because of the many software designers there are many paths to producing LSFS and MH files, but all versions are equally valid.

A flow chart for the data reduction is provided below in Fig 2. The chart goes from upper left to lower right, like a page of text. Each entry on the top of the arrow is a program that one who is reducing the data would directly call, and below each of these entries is the data product associated with each of these. All capital letters in the data products are stand-ins for numbers or names that are attached to specific products. The dashed-line arrows refer to data reduction paths that are optional. Each of these programs call on data products generated earlier in the pipeline, and many of the call on directly antecedent products. The reduction package contains many other programs, but these are the only ones that need be directly called by the user.

The programs covered are split into 6 groups - CALIB, SPMOD, XING, GRID, STRAD and AUX. CALIB is responsible for the zeroth order calibration of the data, and its organization by day of observing or 'scan'. SPMOD is responsible for dealing with minimization of baseline ripple in the data set. XING is responsible for the 'crossing-point' reduction - using the information gleaned by comparing the relative strengths of lines observed at the same position on different days to constrain the gain of each beam. GRID is responsible for taking fully corrected spectra and turning them into a data cube. STRAD is our first attempt to correct for the sidelobes of the ALFA beams. AUX is a set of auxiliary codes that may be employed in reduction, and will be explained last, though the codes may be useful in other steps of the reduction. AUX also allows for some more sophisticated maneuvers, such as combining data from different regions and projects into single data cubes or renaming the paths to data when changing file systems. AUX also contains a swath of interactive codes for inspecting and flagging data. When AUX codes could be useful in a step I will refer to the interested reader to the AUX section.

The plan here is to talk about each of the programs that are called in the flow chart sequence and most of the sub-programs that they call. I heavily recommend looking at this document while examining the source code, side-by-side; the code is (typically) well documented, and many questions on subtleties can be understood by reading the annotated code itself. For those not familiar, IDL codes typically have commented headers that explain inputs and outputs, and I recommend that a user spend the time to at least take a look at these headres. That being said, some of the code involves some mind-bending array gymnastics, (and occasionally calling of C routines) and so may not be totally transparent to the reader (or, for that matter, the author). This code, though rather well refined at this point, may still be difficult for those not familiar with IDL to use, particularly when exploring new observational and reduction parameter space.


generating codes
generated data products
description of task
CALIB
make_dirs.pro
(directory structure)
generates uniform directory structure for data products
stg0.pro
/rega_NNN/galfa.DATE.PROJ.####.rega.sav
/rega_NNN/galfa.DATE.PROJ.hdrs.rega.sav
/rega_NNN/galfa.DATE.PROJ.####.rega.fits
removes bandpass, removes fiber reflections, doppler corrects, tags and truncates data from one day's observing of one region
todarr.pro
/todarr.sav
generates a single structure with all position information for entire region
SPMOD
spcor.pro
/aggr(_XINGNAME).sav
/spcor(_XINGNAME).sav
removes fixed pattern noise
XING
xgen.pro
/xing/regaAAA_BBB.sav
finds all crossing points and generates a structure of locations and relevant parameters.
lxw.pro
/xing/regaAAA_BBB_l.sav
loads weighted spectra into crossing point structure
xfit.pro
/xing/regaAAA_BBB_f.sav
fits for the relative gain at each crossing point
lsfxpt.pro
/xing/rega_lsfxpt_XINGNAME.sav
generates a least-squares fit matrix that relates crossing point gains to beam gain variability with time
xg_assn.pro
/xga_XINGNAME.sav
/rega_NNN/rega_NNN_xing_XINGNAME.sav
/xingarr_XINGNAME.sav
solves the least-squares fit-matrix and applies the beam gain solution to all spectra
GRID
scube.pro
/GALFA_HI_RA+DEC_0##.00+##.35_W.fits
/GALFA_HI_RA+DEC_0##.00+##.35_N.fits
/GALFA_HI_RA+DEC_0##.00+##.35/
generates the survey grid
STRAD
corral.pro
/rega_NNN/galfa.DATA.PROJ.####.rega.fits.SL
generates the 'contamination spectra' for each integration

# 3 CALIB Programs

## 3.1 `make_dirs.pro`

To simplify the reduction process we set up a directory structure that is unique to each project – this helps the code be sure where everything is, and is the basis for the rest of the reduction. It is *completely* mandatory to set up this file structure in this way - otherwise the code will fail. The directory structure is regularized by `make_dirs`, which is run as follows:

```
IDL> make_dirs, root, project, regions, scans, tdf=tdf
```

`root` is the directory the whole thing falls under, `project` is the project name that GALSPECT used to write the fits files (e.g. a1234), `regions` are the names of the different regions done in the one project (can be a single entry), and `scans` are the number of days each one takes (again, can be an array or a single number). **NB:** It is recommended to name your regions with a short string of lowercase letters - names that use underscores, hyphens, capital letters or, heaven forfend, *spaces* may not be supported in the software. It is also not advisable to name `project` or `region` as a subset of each other or any part of the `root`. The directory structure it generates is shown graphically in Fig. 1. A note on directory specification: in this pipeline, any time you specify a directory, specify it with a trailing `/`, e.g. `/share/galfa/galfamh/`. The `tdf` keyword allows for reverse compatibility to the original two-digit formatting that was the standard for versions of the pipeline before v2.2. The current version uses three digit formatting to label directories by day in the scan. Note that these names, `root`, `project`, and `scans` are standardized throughout the pipeline, along with `region`. A note on 'scans': In this GSR pipeline, and this document, 'scan' and 'day' are used interchangeably. In both cases what is being referred to is a uninterrupted session of data taking with GALSPECT and the associated TOD. Typically, one uses only a single scan for each contiguous data-taking session, though one may wish to break up a given session of data taking into many separate scans/days, for instance in the case of leapfrog scanning, which the code will happily interpret. On the other hand, any non-contiguously taken data cannot be put into the same scan/day. The unfortunate and redundant day/scan nomenclature is an artifact of the version history of this code, for which we apologize. If day and scan are both used in a procedure call, day refers to the calendar day of the observation.

Note that this stage of the reduction, making the directory tree, needs only to be run a single time for a project. The rest of the reduction process will refer to a single *region* within this project, and would have to be run multiple times for multiple regions. It is also important to realize that the reduced data itself will be imprinted with the original root directory you choose to run the code in. It is advisable to run this reduction in one directory, and not move it from place to place, as this will confuse the reduction. There are codes to deal with moving directory trees (see §???), but if it is recommended that the user attempt to avoid this situation.

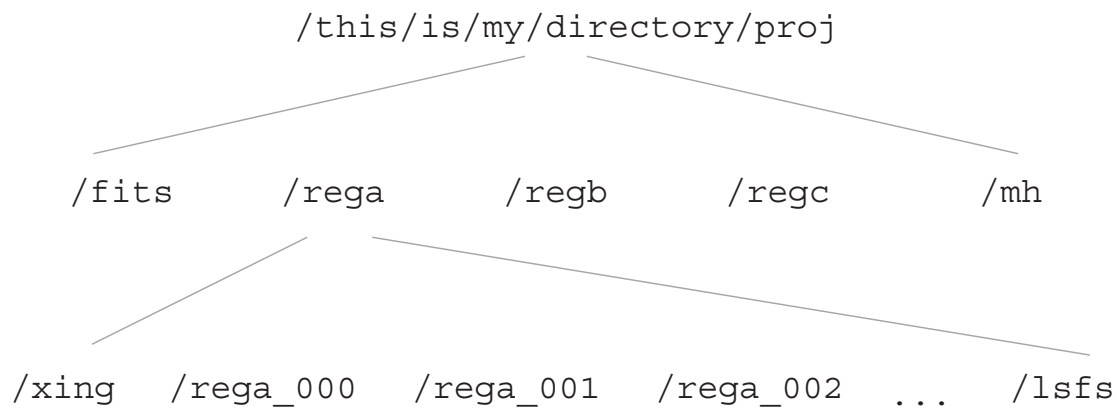



Figure 1: The file structure generated by `make_dirs, '/this/is/my/directory/', 'proj', ['rega', 'regb', 'regc'], [5,6,8]`

### 3.2 `stg0.pro`

The stage zero (stg0) data reduction takes the data in raw form (fits files) and

- removes the IF bandpass (see Heiles 2005, GALFA Technical Memo 2005-01 on how exactly this is done)
- does a rough gain calibration to put the data into temperature units
- removes bright, single channel radio frequency interference
- removes cable echo artifacts
- does a doppler correction to the Local Standard of Rest (LSR) frame
- finds the data that are during your day's scan
- saves these data to an appropriate folder, for a single scan's observing of a single region.

The data are organized by day numbers – if there are 11 days in your scan pattern your days will go from 000 to 010, the data for each day will reside in folders in */this/is/my/directory/proj/regx/regx*_000, .../*regx*_001, etc. In this case you would end up running some version of the stage zero reduction 11 times, one for each day. As a byproduct, each .fits file will have a corresponding .mh file that will live in */this/is/my/directory/proj/mh/* (unless you set nomh) and each day will have .lsfs file that will live in */this/is/my/directory/proj/regx/lsfs*. Note that if you are unfamiliar with 'zero-indexing', now is the time to become familiar: it is crucial to remember that if you have N scans, your scans will be labeled from 0 to N-1. This convention is used throughout the GSR pipeline.

This is how to call `stg0.pro`:

```
IDL>   stg0, year, month, day, proj, region, root, slst, $
         elst, scan, fd, v25, nomh=nomh, calfile=calfile, $
         mhdir=mhdir, caldir=caldir, tdf=tdf, odf=odf, deblip=deblip
```

Some of these are a little subtle. `scan` is the day number you are interested in reducing - it is typical to number your days of observations with scan numbers in increasing chronological order, so that your observation on November 23rd would be scan 0, your next observation on November 26th would be scan 1 and your next one on November 24th would be scan 2. `slst` are the starting LSTs of the observation. `elst` is the ending LST of the observation. If your observation was a standard BW observation (now obsolesced by the smart basketweave codes in CIMA 3.0), the relevant LSTs can be found in the output of `BW_fm.pro`, unless the observations were terminated early and the spectrometer was left running for some time; then it is best to put in the LST at which the observation was terminated, to avoid adding data to your map taken in some alien observing mode. `fd` is the directory in which the code should look for the raw fits file data.

`nomh` allows the user to specify that no new mh files need to be generated, `mhdir` specifies that mh files can be found in a specific non-standard directory. This tends to be this situation, as mh files are generated by the archiver ($/share/galfa/galfamh/$). `caldir` similarly allows one to set a specific non-standard directory for the LSFS file, and `calfile` allows the user to specify a specific LSFS file to use, rather than generating one anew. `tdf`, if set, uses the old two-digit format for directories (not recommended unless reverse-compatibility is crucial) and `odf` saves the reduced data in the .sav data format, as in the previous version, rather than in the .fits format employed in GSR 2.2. This is also strongly advised against, particularly in areas with many crossing points, as it will make XING very, very slow.

In the current version, 2.6, a few things have been changed. In particular, the old `startn` and `endn` inputs have been removed. To keep the code backward compatible (and add the capacity to use these inputs in awkward situations), an input called `v25` has been added. When using the code normally, this input is left *unset*, which is to say the number of inputs in the code one fewer than is called for. To use the code with version 2.5 inputs, simply call it as if it were vesion 2.5, and the `v25` input will tell the code to resort the inputs correctly. *Do not set `v25` unless you really know what you are doing!*

We have also added a `deblip` flag, which removes a peculiar form of RFI which we call 'blips'. They are intermittent, with a near 12-second period. They have a width of a few hundred channels, and appear in all beams simultaneously. They seem to be composed of 4 or 5 Gaussian components, and do not change in spectral shape. The amplitude of the blips does change rapidly from feed to feed and second to second. If the `deblip` flag is set, the code will attempt to remove this particular, pernicious form of RFI from the data before performing the LSR correction.

The stage zero reduction code calls a sequence of relevant codes to do the work of the reduction. First, the mh files are generated. These are called through a code called `mh_wrap.pro`, which is a code in the `/procs/hdr` suite. Unless you provide it with the name of a previously generated LSFS file, through the calfile keyword, it will call a routine `lsfs_wrap.pro` which, given a list of

file names, will generate as many LSFS files as their are SMARTF runs. Only the first of these LSFS files each day will be used to reduce the data for that day. After these two file sets have been generated, the code calls the `calcor_gs.pro` code.

`calcor_gs.pro` is responsible for applying the bandpass correction (through `lsfs_wrap.pro`) the temperature correction (again, through `lsfs_wrap.pro`), the doppler correction (through the .mh files), un-swapping any swapped receivers, and tagging the data with other bits of useful information, such as the the temperatures of the cals being applied. It also is responsible for only including data that are part of the BW scan, so that the rest of the reduction is not confused by extra data. To do these things `calcor_gs` calls a bunch of other yet smaller programs. For the bandpass correction `m1polycorr.pro` is invoked and for doppler correction, `dcs_wrap.pro` is called, which in turn calls `dop_cor_spect.pro`. On top of all this, a code called `deflect.pro` gets rid of the effect of any bad impedance matching in the cables that run from ALFA downstairs. This single fourier component 'reflection' can dominate any other ripple in the spectra. A new code has been added that removes single-channel RFI, with the help of a code called `despike.pro`.

The stage zero code generates three data products. One is the reduced data, saved in a fits file with the format *reg_NNN/galfa.DATE.PROJ.####.abc.fits*. The second is an associated .sav file, containing mh data, information on the wide-band spectrum and useful tags. These are saved in files with the format *reg_NNN/galfa.DATE.PROJ.####.abc.sav*, each in a folder associated with the day that it was observed. These two files contain all the data that was originally contained in a single .sav file in the previous versions of this pipeline; one can revert to this method, with the use of the `odf` keyword, though it is not advised. The last data product is a single list of the information for a given day, including a über mh file that spans the entire day's worth of time. These are in the format *reg_NNN/galfa.DATE.PROJ.hdrs.reg.sav*.

### 3.2.1 `todarr.pro`

`todarr.pro` simply puts all the mh data together in one über-über mh-like file, for easy access. This structure is actually called 'mht' and it contains only the ras, decs, UTC stamps, scans in which the data are found, and the associated file names. These data are used in many places throughout the code. This code needs to be run only once per reduction - even if later steps fail, the data product from this step need not be regenerated. It is called as follows:

```
IDL> todarr, root, region, scans, proj, tdf=tdf
```

and generates *todarr.sav*

## 3.3 SPMOD

The purpose of the `SPMOD` software is to get rid of ripples generated by reflections inside the geodetic dome and in the Arecibo superstructure. Our tests have shown that the largest contribution to baseline ripple (after fiber reflections) is fixed over the course of a day in each receiver, when the

receiver does not move in AZ, ZA or rotation angle. We use this information, along with some sophisticated modeling of the HI we observe, to greatly decrease the contribution of baseline ripple. A subtlety with this approach is that we need to know the relative gains of the receivers before we can disentangle the HI from the background ripple, but was need to be able to remove the background ripple so that we can accurately determine the relative gains! Our solution is to jump-start the process by making a guess as to the relative gains of the beams first, by just comparing the HI from beam-to-beam each day. We then take that, use it to get rid of the ripple (SPCOR), use the ripple-removed data to do an accurate gain calibration (XING), and then go back and re-run the ripple removal process (SPCOR). So the true process is CALIB, SPCOR, XING, SPCOR, GRID, STRAD, GRID (if you also include the sidelobe correction). A few things must be mentioned about this approach. Firstly, it is not always fruitful to iterate on this step, and it is unknown as to exactly why iteration works on some data set and not others. Secondly, it is crucial to mention that it is not yet known whether these techniques have any useful effect upon data where the ALFA rotation angle changes substantially during the run or upon data where the telescope points significantly off of the meridian during the run. It is the author's guess that this is a useful reduction to run for the latter situation, and not the former, but at this point this is only educated speculation. Also, the code will happy run GRID at this point, using the data from CALIB. It will be very stripy, but if you want to know whether things are basically working, it is recommended that you skip to GRID before running this section.

### 3.3.1 `spcor.pro`

`spcor.pro` is responsible for doing this processing and calling all the relevant sub-codes.

```
IDL> spcor, root, region, scans, proj, $
noaggr=noaggr, badrxfile=badrxfile, xingname=xingname, $
tdf=tdf, odf=odf, old_zg=old_zg, rficube=rficube, fn=fn, $
v1=v1, force1=force1, decrng=decrng
```

The first four inputs are in the standard format. The `noaggr` keyword is for if you have already run the code and trust your aggregate spectrum file (aggr(_XINGNAME).sav) - aggregating the spectra takes some time, so this a good keyword to use if you have already generated your aggr.sav file. `userrxfile` is the same as is `stg0.pro`, and should be set if you have a badrx file. `xingname` should not be set the first time through SPCOR - it is the name of a previous XING reduction to apply, and if set will instruct the code to read XING and old SPCOR data. The second time through SPMOD it should be set to the the name used in the previous run of your XING reduction (the `xingname` variable in `lsfxpt.pro` and `xg_assn.pro` ).The `odf` and `tdf` keywords are the same as in INIT. `old_zg` uses an older style of zero-order gain correction, which is not reccomended. `rfi_cube` is a preliminary keyword to stop RFI from interfering with the baseline ripple fit. It is a [8192, 2, 7, scans] vector that is mostly 1s, but is zero where you wish to zero-out any RFI. Typically the keyword is implemented if the fits from SPCOR seem to be corrupted by RFI. Note that this keyword only functions when using the `v1` keyword. `fn` can be set to files containing non-standard equations of condition solutions to the fixed-pattern noise. This is an engineering mode.

GSR version 2.6 saw a major upgrade in `spcor.pro`, as reflected in the following keywords. The code now accounts for variations in the the declination of the observation, as the ripple will change as dec changes for a fixed hour angle. This modeling of declination changes means the code must make a guess as to the range of declination that the useful part of the observation covers, which necessitates the addition of the keywords `force1` and `decrng`, in case the code has trouble with this guess. The code also now tries to use data with limited HI intensity, so as to limit the contamination of the deduced ripple spectra by the HI. It also corrects for the effect of the LSR shift over the course of the day, so that the ripple removed matches the ripple observed. The `v1` keyword reverts to the original version of the code, which does not implement these changes. The `force1` keyword can be set to an array of 1s and 0s, with length equal to the number of scans. For any scan where `force1` is set to 1, only one declination region is set, regardless of the range in decs is in the raw data set. This is useful for drift scans, which occupy only a single dec, but might have extra data at other decs we wish to ignore. The `decrng` keyword is similar, but it allows the user to force a particular declination range to use, if a basketweave scan is expected to run over a certain range, but includes extra data that would confuse the software.

`spcor.pro` will call three main codes to do the analysis - `aggr_spect.pro`, which aggregates spectra over the whole data set, `zogain.pro`, which determines the zeroth order gain corrections and `find_fpn.pro`, which finds the so-called 'fixed-pattern noise', the dominant part of the baseline ripple which we are attempting to mitigate.

The data products from this code are *aggr(_XINGNAME).sav* and *spcor(_XINGNAME).sav*. Note that in the new version the data in *spcor(_XINGNAME).sav* is substantially changed and more complex.

## 3.4 XING

### 3.4.1 `xgen.pro`

`xgen.pro` is responsible for finding all the crossing points. This is a relatively fast procedure, as the code only reads the .hdrs files, rather than the entire data sets. `xgen.pro` takes a standard set of inputs:

```
IDL> xgen, root, region, scans, proj, goodx=goodx, $
         xday=xday, tdf=tdf, blankfile=blankfile
```

`xgen.pro` is primarily a wrapper for a more core piece of code, in this case `newx.pro`. `xgen.pro` calls `newx.pro` for all possible combinations of scans and beams crossed with all other combinations of scans and beams. It first does the auto section, which is to say beam-to-beam crossing within a single day, and then does the main section, for beam-to-beam crossing on days that are not alike. Note that on a given day we do not wish to allow all beams to cross each other - some beams never cross, and some beams cross in awkward points, when cals may be firing or when the telescope is slewing quickly. This is taken care of by a 7x7 matrix, `goodx`, which is set up for gear 6, normal basketweave scanning and can be superseded with a keyword of the same name. Note that only

the upper triangle, `goodx[i,j]`, with i < j, is relevant. `xday` can be set to an equivalent matrix, but for days. If you do not wish to look for crossing points between specific days, set this to be a $N_{days}$x$N_{days}$ array, with 1s at [n,m] where you wish to compute crossing points between day n and day m and 0s where you do not. `blankfile` can be set to the full path to a file containing blanking data - a range of UTCs for a given beam to not include in the crossing point information. `getx.pro`, the core code, when handed a the appropriate mh files, will generate a structure (xarr) that contains a lot of different information about the crossing points. This information includes the day (or 'scan') number and beam number of each of the tracks that crossed, as well as time information, in which files the spectral data can be found and the positions of the crossing point. It also includes weights, which is to say how much emphasis to put on the data point before the crossing and how much to put on the one after the crossing. Currently this is just a linear weighting by distance from the crossing point. The structure also contains blanks for spectra to be loaded in and relative gains to be determined, that will be filled in later steps. `xgen.pro` serves to feed this program the correct information to make crossing point structures and save them with the appropriate names. The data products are called *xing/regAAA_BBB.sav*. Note: This needs to be run *only once*, and does not need to be rerun for future times through the XING process.

### 3.4.2 `lxw.pro`

After `xgen` is run, all of the spectra must be loaded into the crossing point files, with a code called `lxw.pro`. `lxw.pro` is called similarly:

```
IDL>  lxw, root, region, scans, proj, $
         no_over=no_over,  xday=xday, badrxfile=badrxfile, $
         tdf=tdf, odf=odf,no_auto=no_auto,$
         xingname=xingname, keepold=keepold, $
         no_spcor=no_spcor, parallel=parallel
```

with all the inputs identically formatted. The keyword `xingname` allows the user to input the *previous* run through's XINGNAME, so as to load appropriate spcor data and XING corrections. This is a little confusing, so I will try to be as clear as possible. The first time through XING, *do not* set this parameter, until `lsfxpt` and `xg_assn`, wherein you will name the XING reduction you are doing. The second time though, set this value to the value you set in the previous round for `xg_assn` and `lsfxpt`, and make a new name once you get to `lsfxpt` and `xg_assn` the second time around. The `badrxfile` keyword here refers to any badrx file you might wish to use to avoid loading corrupted spectra. The `no_auto` keyword allows the user to skip loading crossing points within a day (usually used for engineering) and `no_over` allows the user to not reload data that has already been loaded, if the program was interrupted. `keepold` allows the user to keep the old `_1` files from a previous time through the SPMOD-XING loop. This is worth setting in case the reduction runs into errors, but once the data make a good image it is a good idea to delete these as they can take significant disk space. `parallel` allows the user to paralllelize this code across any number of machines -many copies of this command can be run at once with this keyword set. This will increase the speed of the code of almost N-fold. Note that the implementation of this keyword has changed since GSR2.5. `no_spcor` allows `lxw` to run without any spcor data.`lxw.pro` takes a significant chunk of time, because of the amount of file reading and writing required. As above, this

is only a wrapper code. The core code that is being run is a code called `loadxfits.pro`, a much simplified version of `loadx.pro`. Note that `loadx.pro` is used when the data are stored in the older data format, which can be *excruciatingly* slow. As it is, this code can take hours to run. Note that this code calls a multipurpose code called `fixrx.pro`, which takes any dataset that has bad receivers and overwrites the offending data with its beam-pair. Since the spectra we are interested in are an average of these two polarizations, we don't want to average in any bad data. Of course, this reduces our signal-to-noise, but so be it. This procedure produces a similar data product to the last one, the only difference being that the slots for spectra are now filled with correctly weighted spectra. They are written in the format *xing/regAAA_BBB_l.sav*.

### 3.4.3 `xfit.pro`

The relative point-to-point gains must now be determined.

```
IDL>  xfit, root, region, scans, proj, no_auto=no_auto, $
         conrem=conrem, tdf=tdf, $
         keepold=keepold, xingname=xingname
```

`conrem` would only be set if the data had not had their continuum subtracted in the previous stages - this should be considered an 'engineering' mode that should never need to be invoked.

`xfit.pro` otherwise follows the same structure as the previous code, but does not (for some reason) call a core code. It effectively plots the two spectra against each other and fits a line to the slope, thus determining the relative gain. It records this as well as any detected offset in the baseline, which is currently ignored. On occasion the fitting program flips out, with some part of the fit non-converging, so there is an error trap to deal with this - usually this comes from user error. All of the original data, plus the gain and zero-point, less the spectra themselves, are saved in a structure called 'outx'. They are save in files called *xing/regAAA_BBB_f.sav*

### 3.4.4 `lsfxpt.pro`

`lsfxpt.pro` is the code responsible for engineering the 'equations-of-conditions' (X) matrix that fits all of the of the crossing point. It is based upon the idea that the ratio of the gains can be approximated as

$$R = \frac{G_{BD}(t)}{G_{B'D'}(t)} = \frac{1+\delta_{BD}(t)}{1+\delta_{B'D'}(t)} \simeq 1 + \delta_{BD}(t) - \delta_{B'D'}(t), \tag{1}$$

where B and B' are some arbitrary beams and D and D' are some arbitrary days. Note that the approximation here induces errors of $2\delta^2/(1-\delta)$, which get to be 10% when $\delta$ nears 20%. This allows us to set up our Y in the equation

$$Y = X \cdot C \tag{2}$$

as just

$$Y = \delta_A - \delta_B = R - 1, \tag{3}$$

which is linear in the $\delta_{BD}(t)$, and so can be solved with a set of linear equations. The C is a set of coefficients that determine the varying gain of each beam. The 'equations-of-conditions' matrix, that connects our data (Y) to the parameters we wish to fit (F) can be set up in a variety of different ways, controlled by the various keywords.

```
IDL> lsfxpt, root, region, scans, proj, $
degree, xarrall, yarrall, xingname,$
fourier=fourier, daygain=daygain, beamgain=beamgain, $
big=big, tdf=tdf, time=time
```

`root`, `region`, `scans` and `proj` are all standard inputs. `degree` is the degree of polynomials to fit to the varying gains of each beam and day. Set it to -1 to have no polynomial fits and to 0 and higher to have polynomial fits of those orders. `xarrall` and `yarrall` are the output matrices that are generated, and are only output here for diagnostics. `name` is the name of the fit, which allows the user to keep track of different attempts to fit the varying gains. The `fourier` keyword allows the user to fit with sines and cosines by setting it to [a,b], where a is the lowest order (1 is a single period across the domain) and b is the highest order. Currently 1 is the lowest value that works for fourier (the zero-order fourier component, e.g. DC offset, can be done with zeroth order polynomials). The `daygain` and `beamgain` keywords allow one to fit overall gains for each beam (which are equivalent to the cal values for that beam) and for an overall day. It is not yet known how much the optimum parameters will vary from region to region, but a good first guess might be to set `degree` to 0, to get the basic overall numbers, and `fourier` to [1, 3], to get a little bit of higher-order correction. Three new keywords were implemented in gsr 2.3 . The first is `big`, which allows the code to handle situations where the number of crossing points in so large that the X matrix cannot be handled in memory. In this case, for each crossing point file $X^T X$ and $X^T Y$, are generated and then co-added to all following files. This is recommended. The second is the `time` keyword, which, if set, parameterizes the gain variations in UTC time, rather than RA. It is not clear to the author if there is ever a reason not to set this keyword, but it certainly must be set for any data set non monotonic in RA. `lsfxpt.pro` calls a piece of code called `makdom.pro` which, for each scan, generates a range over which the fourier components and/or polynomial coefficients can be evaluated. This range is expressed as a structure (mdsts) that can be read by `locdom.pro`, which is used to evaluate the elements of the X matrix. The X matrix must also contain constraints on the gains; since the gains are relative, the fits are equally good if all the gains are evaluated to be huge or tiny. We do this through a 'pinpoints' constraint. At a bunch of specific RAs (or times), the total of all the fits for each beam and day is forced to be zero. These pinpoints are regularly spaced throughout the domain, and are proportional in number to the highest order (in fourier or polynomial) fit coefficients. The X and Y matrices that are generated by the program, along with the the mdsts structure, the length of the data (non-constraint) part of the Y array (ndata), the locations of the crossing points and the pinpoints are all saved in a file of the form */xing/reg_lsfxpt_NAME.sav*

### 3.4.5 xg_assn.pro

xg_assn.pro is designed to assign the gains to each point in the data set, given the output of lsfxpt.pro. The code has rather simple inputs:

```
IDL> xg_assn, root, region, scans, proj, fitsvars, xingname, $
        cutoff=cutoff, big=big, tdf=tdf, time=time
```

fitsvars is an output for all of the variables that get generated while solving the matrix-inversion problem. It is good to examine this data to understand the goodness of your fit. name is the same as the name used in the lsfxpt.pro and names this set of gain files. cutoff allows one to set a cutoff value for the inverse of the weight matrix as determined by lsf_svd.pro. big should be set if it was set on lsfxpt.pro, to do the correct matrix inversion. tdf reverts to the two-digit format. time assumes the crossing points are parameterized by time, rather than RA, and should be set if it was set in lsfxpt.pro. Note that the code can take an *extremely* long time to run and may max out the memory of a computer. More than about 15 fourier coefficient terms in your X matrix (see lsfxpt.pro) may in fact top out a 2 GBs of RAM machine, and may take many hours to run. This procedure generates 3 data products. One is the crossing point gains, separated into their respective directories, in the files */reg_NNN/reg_NNN_xing_NAME.sav*. Another is an amalgamation of all these data in a single file, called */xingarr_NAME.pro*. The last contains all of the fitting data, and is called */xga_NAME.sav*

## 3.5 GRID

### 3.5.1 sdgw.pro

sdgw.pro has a storied lineage; gridzilla, AO_gridzilla, ao_gridzilla_GALFA, gridzalfa and, finally, sdgw, which departs somewhat from the theme. The product of sdgw.pro is the final gridded data in two formats; .fits and .sav files. The complexities of how this code works are way beyond the scope of this document, except to say that it is a wrapper code that calls the brains of the operation, sdgrid.pro, written by Tim Robishaw. It calls the code once to determine which files are needed in a given grid and the again for each data file that needs to be loaded. It is called as follows

```
IDL>  sdgw, root, region, proj, gridname, lon0, lat0, spmax, $
        spmin, spres, imsize, rs=rs, projection=projection, _REF_EXTRA=_extra, $
        tdf=tdf, spname=spname,  badrxfile=badrxfile, xingname=xingname, $
        fwhm=fwhm, vel=vel, allfiles=allfiles, savepath=savepath, $
        gridfunc=gridfunc, odf=odf, norm=norm, blankfile=blankfile, $
        arcminperpixel=arcminperpixel, pts=pts, truecart=truecart, $
        cpp=cpp, nofits=nofits, spp=spp, strad=strad, nan=nan $
```

The first three parameters are as usual. `gridname` is any string you would like to identify the grid with. `lon0` is the center of the grid in longitude space in degrees. `lat0` is the same for latitude. `spmax`, `spmin` and `spres` define the maximum, minimum and binsize for the spectra to grid, which can be specified in spectral channel or in km/s, depending upon how you set the keyword `vel`. `imsize` is a 2-element array that is set to the size of the region in pixels. If the keyword `xingname` is set, crossing point calibrations with that name are applied to the data. `rs` allows the user to change the root of the data file, if the reduction was begun on another file system. Specify `file` to use a specific badrx file to eliminate bad rxs. Specify `savepath` if you wish to have the data save somewhere other than the main directory of the region. To use the spectral correction from SPMOD set `spname` to the name of the last spmod reduction. This is a little confusing - if you've run SPMOD and then XING, set spname to 'null' and xingname to whatever you called your xing reduction. If you've run SPMOD then XING and then SPMOD again, based on your XING, then set both of them to the same xingname. If you've run SPMOD, XING, SPMOD, XING then set spname to your first xingname and xingname to your more recent xingname. You can keep refining this, but it has not yet been shown to have any good effects. `tdf` and `odf` are as usual and `fwhm` specifies the FWHM of the convolving beam, which can just be left to the default 3.35 arcminutes. It is possible that crisper maps may come from decreasing this number, but we have yet to examine this issue in detail.

`allfiles`, if set, skips the step in which the files to be included are verified - this will save a little time if you are certain that all your data files lie within your grid region, and will lose a lot of time if you specify it without this being true. `projection` specifies one of the projections from Calabretta & Greisen 2002, A&A, 395, 1077 - see line 278 of `sdgrid.pro`. If this is not set, the default is the Cartesian projection. Not all of these projections have been carefully tested, so beware. `arcminperpixel` is self explanatory, and is defaulted to 1. `gridfunc` allows the user to specify a gridding kernal, and is defaulted to a Gaussian kernal. `norm` allows the user to specify a normalization of the entire data set to divide by. Setting `norm` eliminates the LDS calibration of the data that would normally happen. It is a good idea to use `norm` if either you are looking at data very far off the static Galactic HI (e.g. HVCs), so that calibrating would be very noisy, or if you are doing many grids you wish to stitch together, where you would like to have a fixed calibration. Automatic normalization can fail entirely if the requested cube is very narrow in velocity range ($< 2$ km/s) or at very high velocities. In all of these cases, it is recommended that the user do a first pass on a normal sized grid that spans the static HI line, retrieve the normalization factor and then apply it to subsequent grids. `blankfile` allows the user to specify a file that contains information about bad seconds, and removes said data. `pts` is an engineering mode allowing the user to highlight points in the map to check registration issues. `truecart` is very important, and it belies a complexity in the word 'Cartesian'. Most people imagine that the word 'Cartesian' applies to maps with equal spacing in RA and Dec and with pixels that increase in y having fixed RA and pixels that increase in x having fixed Dec (or *orthogonal graticules*, for the enthusiasts). From the perspective of Calabretta & Greisen 2002 this is only true if *the projection center is a [180, 0].* If the projection center is at the center of the map, these things will not be true, and you will get a weird map, that does not register. To force the center to be at [180, 0], set `truecart`. Note that though the pixels will be in the right spot, some fits readers will not read these maps correctly, instead assuming the wrong projection center. This problem is still under study. `cpp` forces the C++ implementation of non-square sparese-matrix inversion (long story), but should not be used for most data cubes. `nofits` stops the code from producing fits files and only produces the IDL '.sav' files. The `nan` keyword fills the blank areas with NaN, rather than zero, and is needs when it

is called by `sdgw.pro`.

### 3.5.2 `scube.pro`

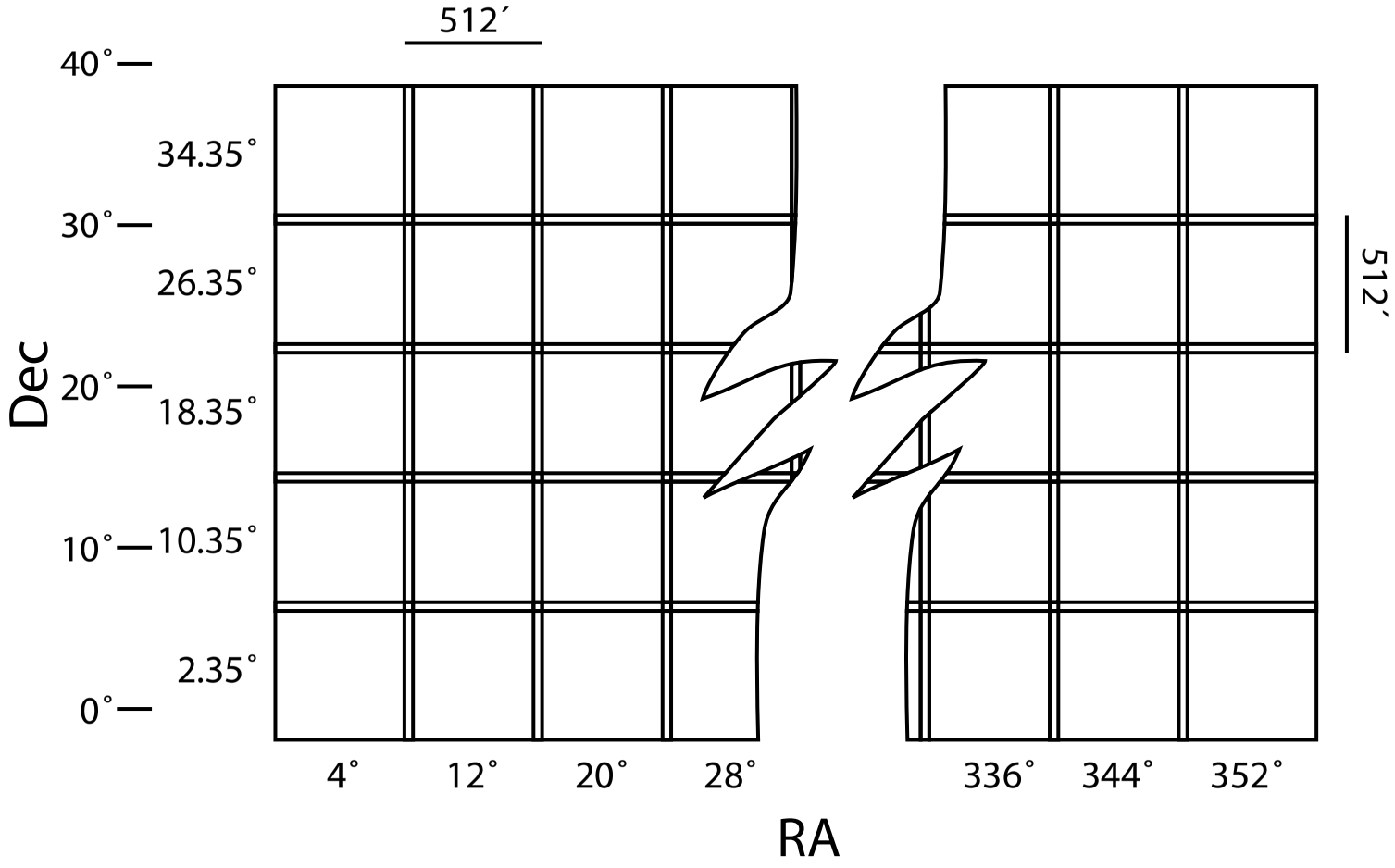
`scube` is a wrapper code for `sdgw` that generates GALFA-HI *survey cubes*. So what is a survey cube? Survey cubes are the agreed upon format for the entire GALFA-HI survey. They are 512 × 512 pixels, in true cartesian projection with 1 (small-circle) arcminute per pixel in both RA and Dec. Four distinct data products are generated over this region. First, the 'time' images, which show the integration time over each pixel. Second the 'Wide' cubes, which average over 4 channels in frequency to build cubes that span the entire velocity range. These are 512 × 512 × 2048 data cubes, stored in fits files as scaled integers. Third, the 'Narrow' cubes, which span only 1/4 of the velocity range to retain the full velocity resolution of 184 m/s, stored similarly to the 'Wide' cubes. Finally, the 'RA-vel slices', 512 individual files for each survey region that are slices through the cube at a fixed Dec. These are stored in their own folder. These follow the format:

*GALFA_HI_RA_+DEC_004.00+02.35_T.fits*
*GALFA_HI_RA_+DEC_004.00+02.35_W.fits*
*GALFA_HI_RA_+DEC_004.00+02.35_N.fits*
*GALFA_HI_RA_+DEC_004.00+02.35/GALFA_HI_RA_+DEC_004.00+02.35_000.fits*
*GALFA_HI_RA_+DEC_004.00+02.35/GALFA_HI_RA_+DEC_004.00+02.35_001.fits*
...

for the survey cubes centered at RA = 4°, Dec = 2.35°. The cubes are spaced by 8° (less than their linear size by 32 pixels) in both RA and Dec, so there are 45 center positions in RA ( 4°, 12°, 20°, ..., 356°) and 5 center positions in Dec (2.35°, 10.35°, 18.35°, 26.35°, and 34.35°). See Figure 3.5.2. `scube` is called as follows:

```
IDL>  scube, root, region, proj, cnx, cny, rs=rs,$
          tdf=tdf, spname=spname, badrxfile=badrxfile, $
          xingname=xingname, odf=odf, blankfile=blankfile, $
          arcminperpixel=arcminperpixel, norm=norm, $
          pts=pts,  cpp=cpp, madecubes=madecubes, $
          noslice=noslice, strad=strad,  nocleanup=nocleanup
```

With all of the calls being the same as those called in `sdgw`, except `cnx` and `cny`, which are the zero-indexed positions of the cubes requested for RA and Dec, respectively. So to make the cube at RA = 20°, Dec = 10.35° one would set `cnx = 2` and `cny = 1`. `strad` and `nocleanup` are addressed below.


512′
40°
34.35°
30°
26.35°
20°
18.35°
10°
10.35°
2.35°
0°
Dec
4°
12°
20°
28°
336°
344°
352°
RA
512′

## 3.6 STRAD

This suite of codes implements a correction to a final data cube for the effects of the first sidelobe of the ALFA beam pattern. It effectively makes a new set of time-ordered data that contains the spectra that contaminate the original spectra from the sidelobes. It does this 're-observing' by assuming a data cube created by `scube` is a good representation of the sky, and then examining that cube with our measured shapes and amplitudes for the sidelobes in question. This generates a set of spectra, saved in files that mirror the original data, e.g. *reg_NNN/galfa.DATE.PROJ.####.abc.fits.SL*. These spectra can then be removed from the orginal data and regridded by re-running `scube` with the `strad` keyword set. Note that the original data cube from which these sidelobe spectra are taken and the final cube must have the same scale factor, otherwise the amplitude of the sidelobes will be incorrect. To implement this stray radiation correction, simply run the `scube` code again, now with the `strad` flag set. Note that at present only the wide cubes are used as the dirtycubes, which will somewhat degrade the spectral resolution in the resulting cleaned narrow cubes and slices. The `nocleanup` flag will, if set, not remove the *reg_NNN/galfa.DATE.PROJ.####.abc.fits.SL* data when the processing is complete. This is primarily for inspecting the data.

## 3.7 AUX

AUX contains a variety of codes that allow the user to do some fancy things with the data, and in particular allow the user to manipulate file structures using symbolic links, to cull and combine data sets. It also has a number of codes that allow the user inspect and flag the data interactively.

### 3.7.1 `merge.pro`

`merge.pro` allows the user to combine two completely separate data sets into a new data set. This allows the user to do crossing point calibrations between two data sets. It is called as follows:

```
IDL> merge, root1, proj1, region1, days1, $
         root2, proj2, region2, days2, $
         newroot, newproj, newreg, odf=odf, tdf=tdf
```

The inputs are all in analogy to the simple `root`, `proj` and `region`, but specify the two original regions (1 and 2) and map to a new region (new). This is all accomplished through symbolic linking.

### 3.7.2 `subday.pro`

`subday.pro` is built on the same chassis as as `merge.pro`, but has a slightly different goal. It extracts a subset of days the user wishes to examine from a single region and symlinks it to a new directory. It is called as follows:

```
IDL> subday, sds, root1, proj1, region1, days1, $
         newroot, newproj, newreg, odf=odf, tdf=tdf
```

and is in complete analogy to `merge.pro`. `sds` is a list of days the user wishes to keep.

### 3.7.3 plotmh.pro

`plotmh` is a simple diagnostic tool that allows the user to plot all of the positions of the data in a region. It is called as follows:

```
IDL>  plotmh, name,_EXTRA=ex, mht, noplot=noplot, $
          mhtfile=mhtfile, ctbl=ctbl, cutoff=cutoff
```

`name` is the name of the project in question. Any keywords can be passed to the plot command, via `_EXTRA`. If `noplot` is set then no plot is made. If `mhtfile` is set, then instead of cobbling all data points together from mh files, the composite file is read from this path, otherwise it will be written to */share/galfa/mhtfile.sav/*. `cutoff` is a way to limit the number of objects shown in the legend - set it to the lowest number of seconds acceptable for an object to get an entry in the legend.

### 3.7.4 endfinder.pro

`endfinder.pro` is a tool to select the correct LST endpoints for a set of observations, and is designed to be used in conjunction with `stg0.pro`. It is important to select these carefully, lest your data be contaminated by data taken in alien observing modes. It is called as follows:

```
IDL> endfinder, year, month, day, proj,  $
         root, lsts, mhdir=mhdir, mht=mht, th=th$
         wt=wt, ct=ct
```

where the first 5 inputs are standard and the last output, `lsts` is a 2 element array with the starting and ending lsts to use in `stg0.pro`. `mhdir` is the directory in which the mh files are located and `mht` is an output structure containing all the mh information read. `th` is the thickness of the data displayed; setting this to 2 or 3 may help the visibility of the plots. `wt` is the wait time between clicks - if you are having trouble accidentally clicking, you may want to set this to 1 or 2. `ct` is the color table (as in the `loadct` command), which defaults to 13. Choose another color table if you wish. The code is a self-explanatory GUI that requires a 3-button mouse.

### 3.7.5 examaggr.pro

`examaggr.pro` is designed to allow the user to look though the aggregate spectra as created at the beginning of `spcor.pro`. Is is called as follows:

```
IDL> examaggr, root, region, proj, wt, _REF_EXTRA=_REF_EXTRA
```

The first 3 inputs are standard and `wt` is the wait time between displaying the data. `_EXTRA` allows the user to add any other plot keywords, to better control the plot outputs.

### 3.7.6 t1.pro

`t1.pro` is a code used for determining relationships between the gridded data and the time-ordered data. If you see a flaw in your gridded data and you want to trace back where that flaw originated from, t1 is the code for you. Essentially, it displays a data cube and then allows you to cursor over the cube. It will overlay the time ordered data associated with where your cursor is, and output all the various information regarding that time ordered data.

```
IDL> t1, fn, todarr, day, beam, $
second, file, slrange=slrange, blfile=blfile
```

`fn` is the name of the image fits file you wish to investigate, an output of sdgw or scube. `todarr` is either the path to and name of the todarr.sav file, or the variable restored from that file, `mht`. `day`, `beam`, `second`, and `file` are the outputs from the code, which tell you everything you might need to know about the position of the time-ordered data you have selected. If the input cube is a survey cube, `slrange` can be set to a two element array to limit the range over which the loaded cube extends in velocity. `blfile` can be set to the path to the blankfile, so that blanked time-ordered data show up in a different color.

### 3.7.7 t2.pro

`t2.pro` is based on the same chassis as `t1.pro`, but it is used exclusively for adding blanks to the blankfile.

```
IDL> t2, fn, todarr, blfile, slrange=slrange, bt=bt
```

The call is the same as the call to `t1.pro`, except `blfile` is an input rather than a keyword, and the `bt` keyword allows you to input a two-element array containing the min and max amplitude you are interested in seeing in the data slice. The code gives you instructions as it runs and requires a three-button mouse.

### 3.7.8 add_blanks.pro

`add_blanks.pro` is a similar piece of code, in that it is designed to allow the user to interactively select regions over which to blank the data. This code works in the position and overall amplitude

domain, rather than overlaying the TOD on a gridded data cube. This can be helpful for culling stray scans, where the telescope pointed in a strange direction, or places where the receivers got out-of-whack temporarily.

```
IDL> add_blanks, root, region, scans, proj, $
blankfile, badrxfile=badrxfile, _REF_EXTRA=_EXTRA, $
badcl=badcl, range=range, $
xsize=xsize, ysize=ysize
```

the only new keywords here are `badcl`, which allows you to set the color of the blanked data, `range`, which allows you to set a range of scans to examine with a two-element array, and `xsize` and `ysize`, which allow you to specify the size of the interactive windows. Again the code is interactive, self-explanatory and requires a three-button mouse.

### 3.7.9 The cube server

There are two pieces of code that implement the serving of GALFA-HI super- and sub- cubes. The idea is that a database is constructed of the current available survey cubes, and then with this data base other cubes can be constructed. The server can currently only serve either 'Wide' or 'Narrow' data, and cannot make high resolution, off-line cubes from the slice data. Also note, this is not technically part of the GSR pipeline, in the sense that it is not necessarily associated with a particular project.

To set up a computer system to do run the cube server, one needs to put a group of survey cubes together in a single directory. Once this is done, the survey cubes must be cataloged. This is done with the code `fill_T_all.pro`. This is a run-time script and can be executed by simply typing

```
IDL> .run fill_T_all
```

in a copy of IDL running in the directory that contains the survey cube data. Once this is run once, the `extract_cube.pro` code will work indefinitely on the server.

`extract_cube.pro` allows you to build new cubes with different dimensions from the original survey cube data set:

```
IDL> extract_cube, ra, dec, minv, maxv, $
norw, ctitle=ctitle, xcs=xcs, ycs=ycs, $
binv=binv, nosav=nosav, nofits=nofits, $
dustdir=dustdir, datadir=datadir, hdrpath=hdrpath
```

`ra` and `dec` are the centers of the requested cube in degrees. `minv` and `maxv` are the velocity edges of the requested cube in km/s. `norw` is set to either 'N' or 'W' to choose Narrow or Wide data cubes.

`ctitle` is the name of the output cube, `xcs` and `ycs` are the size of the cube in arcminutes (pixels), in ra and dec respectively. `binv` is the binning factor; setting this to 3 reduces the resolution in the spectral domain by a factor of 3. `nosav` and `nofits` allow the user to not produces sav files or fits files, respectively. If `dustdir` is set, and contains properly formatted data from IRAS, infrared images on the same scale are produced. `datadir` specifies the directory of the raw data cubes, if you are running IDL in a different directory. `hdrpath` allows the user to specify a new path to the standard header information, rather than the location in which it is typically kept in the GSR file system structure.

### 3.7.10 survey cube helper codes

These are two little pieces of code that allow you to understand where the survey cubes are in the data set.

```
IDL> name = cname(cnx, cny)
```

will print out the name of a survey cube (less the _W.fits, for example), when fed the coordinates equivalent to those that are inputs to `scube.pro`. `cplot.pro` will overplot the boundaries of a cube:

```
IDL> cplot, cnx, cny, _EXTRA=_EXTRA, hrs=hrs
```

where setting the `hrs` keyword specifies that the plot should be in hours RA, rather than degrees RA.