MH AND LSFS: BLACK-BELT DETAILS

Carl Heiles (October 25, 2005)

Contents

1. INTRODUCTION

This document is intended for people who know their way around the first two stages of GSR reduction, which are the generation of `mh` and `lsfs` files. It presents details that one might wish or need to know to understand how things are done and what the potential problems and difficulties are, or need to modify the software. In other words, it's for the *black-belt*. Accordingly, we don't spend time defining a lot of things whose definitions can be found elsewhere, or even definitions that are in the documentation of the procedures. In other words, use this in combination with the existing documentation within the procedures. One convention: "nb" and "wb" are short for "narrowband" and "wideband", respectively.

2. THE FITS FILE AND ITS $m1$ STRUCTURE

The fits files are read by the Goddard procedure `mrdfits`. *Important note:* a related procedure is `mrd_hread.pro`, the Goddard version of which failed on one of our files. To avoid this failure mode we must use a modified version which resides in `...gsr/procs/init/hdr`, (putting it first on our IDL path).

Convention: in this document we often specify paths by writing three dots before `gsr`. The subdirectory `gsr` is the top of the tree that contains all software used in the GALFA data reduction, and it might be installed on any machine (such as your own or one at Arecibo). At Arecibo, the `gsr` directory tree has a version number suffix, and it is located in the `/share/galfa/`, so at the present time of writing (25 Oct 2005), at Arecibo what we mean by `...gsr/` is `/share/galfa/gsr1.1/`.

We read the fits file and create a structure called `m1`. This is an array of length equal to the number of records in the fits file, which are taken once per second. Normally the fits file is 10 minutes long and no datapoints are skipped, so normally it's `m1[600]`. However, sometimes there are fewer elements because datapoints are skipped and sometimes the file is cut short because GALSPECT was stopped.

The details of the `m1` structure are given in Jeff Mock's code that writes it; there's a copy in `...gsr/cl-1.0.0/src/gdiag/gfits.c`. A *complete* set of Jeff's software is on `mofongo` in `/mnt/disk2/jmock`. Some of the more important tags in the `m1` structure include the following:

1. `data` and `g_wide` are the 7679 and 512 channel nb and wb spectra, in long integers with offset of -2^{31}.

2. `g_seq` is a sequence number. I'm not sure of the zero point. The sequence number is incremented for each datapoint. Sometimes the net is too busy to write a datapoint, which is skipped. If so, the recorded sequence number will skip. That is, the sequence number applies to the datapoint whether or not is was recorded, so you can check for missing datapoints by checking the sequence number.

3. `crval2a, crval3a, crval2b, crval3b` are ra, dec, az, and za respectively. There are time delays so that they are not absolutely accurate. Correcting these inaccuracies is one of the main reasons for the `mh` files.

4. `alfa_ang` is the rotation angle of the ALFA turret.

5. `g_mix` is the internal narrowband mixer frequency (lo3), known as `digitalmix`. The GALFA memo "GALFA Spectrometer: Setup, Operations, Basics" explains how to set `g_mix`.

6. `g_lo1` and `g_lo2` are the first and second LO frequencies. See GALFA memo "GALFA Spectrometer: Setup, Operations, Basics" for an explanation of the mixing system.

7. `g_postm` is ra, dec and `g_azzatm` the az, za, but not at the exact 1 sec tick; they are corrected in the `mh` header.

3. GSR/PROCS/INIT/HDR

These programs generate `mh` files, which contain the `mh` and `mx` structures. The `mh` structures contain accurate time and position information and the `mx` structures contain statistical information on data quality. The primary function of the `mh` structure is to define times and positions accurately. The values in the original data from the fits file are not accurate because of delay problems in obtaining and transmitting the data over the net.

Generating the `mh` file is normally done using the shell `mh_wrap.pro`, which reads the list of fits files, invokes the workhorse `m1_hdr.pro` for each one, and writes the associated `mh` save file for each one. See the document "HOW TO GENERATE MH AND LSFS FILES".

3.1. Obtaining the RF and IF frequencies

The procedure `bbifdftprops.pro` calculates rf and if frequencies for each channel of the wb and nb spectra. It also returns the theoretical bandpasses of those spectra. The inputs are all in the `m1` and `mh` structures except for the sidebands, which you must specify. See this procedure's documentation for details. You have to specify sidebands. Call the procedure this way:

```
sb1= -1.d
sb2= 1.d
sb_bb= -1.d
lo2= m1[ 0, nspLO].g_lo2/1.d6
digitalmix= m1[ 0, 0].g_mix/1.d6
lo1= m1[ 0,nspLO].g_lo1/1.d6
bbifdftprops, sb1, sb2, sb_bb, lo1, lo2, digitalmix, $
  rffrq_wb, if1frq_wb, bbfrq_wb, rffrq_nb, if1frq_nb, bbfrq_nb, bbgain_dft_nb
```

Here, `m1` is the structure that is read from the fits file; alternatively, you could get the l.o. frequency from the `mh` structure.

3.2. The *mh* structure

The `mh` structure is defined by `mhdefine.pro`. The important values are calculated in `m1_hdr.pro`. The important tags include:

1. Various tags with the embedded word "stamp". These are accurate values calculated from the sequence number `m1.g_seq` (a running count of the number of 1-second GALSPECT records written since midnight AST on a given day) using a complicated and involved least squares fit. These include:
   (a) `utcstamp`, the UTC since the beginning of 1971 in seconds;
   (b) `julstamp`, the Julian date at UTCSTAMP (Julian days are for UTC in Greenwich);
   (c) `lstmeanstamp` and `lstappstamp`, the mean and apparent LST for the particular `utcstamp`.
2. Four positions with the embedded word "halfsec". GALSPECT records data once each second. During this second the telescope usually moves; we average the positions at the beginning and end of each second to obtain the mean position, equal to the position halfway through the one second interval; thus the term "halfsec". These positions are accurate, having been corrected for sample jitter. Units are degrees for az, za, and dec; and hours for ra.
3. `vlsr` and `vbary`, the velocity of the telescope wrt LSR and the barycenter, respectively. Calculated using `chdoppler.pro`.
4. `pwr_wb` and `pwr_nb`, the spectrum-integrated wb and nb powers in the original data units.
5. `errs`, a set of decoded (human-readable) errors from the original `m1.g_err` (which is not human-readable). `errs` is a [6,2,7] array for each datapoint, 6 values of error for each of the 2 pols and 7 beams. `mh.errs` is generated by `error_decode.pro`, whose documentation describes the meanings in detail.
6. `versiondate`, the date of the software version (yyyymmdd). BE SURE TO CHANGE THIS IF YOU MODIFY THE SOFTWARE!

In addition, most of the m1 header array data are repeated in the `mh` structure.

3.3. The *mx* structure

To *interpret* and *examine* the quantities in the `mx` structure, you can use the programs discussed at length in the document "DOES EVERYTHING WORK PROPERLY? DO THESE CHECKS ON EVERY DAY'S DATA!!". There's also a first attempt at a printed version for the diagnostics in `listmx.pro` . This also discusses the diagnostics, and should be read before going further here.

The mx structure is defined by mxdefine.pro. The important values are calculated in rxdiagnostics.pro. It analyzes the time series of pwr_wb and pwr_nb. The important tags include:

1. julstamp, as defined above for the mh structure
2. ccfwb and ccfnb, the CCF between all pairs of the 14 receivers
3. feedbadwb and feedbadnb, an analyzed version of ccfwb and ccfnb to provide a simply-interpretable result; this is currently not reliable or useful.
4. rmwratiowb and rmsrationb. for each receiver, use a 19-second median filter to remove drifts; remove data whose residual exceeds 3σ; select only the records not in calibrations. Then, for each receiver calculate its rms divided by its mean and divide by the mean of that modified data stream.
5. rxradarwb, rxradarnb, for each receiver a 2-element array, the first element is the period in seconds and the second is the amplitude of that Fourier component divided by the mean power.
6. sjuwv and sjunb, the crosscorrelation peak of a 12-s pulse train with the median-filtered pwr_wb and pwr_nb, divided by the mean values of those pwr arrays. Negative values are preserved in case the radar power saturates the receiver gain.
7. versiondate, the versionddate of the software.

4. GSR/PROCS/INIT/LSFS—GENERATING THE LSFS FILE

This section generates lsfs save files, which use the SMARTF frequency switching calibration to generate the bandpass shapes and also the factors to approximately convert spectral numbers to temperature. The theory and practical details of the Least-Squares-Frequency-Switching (LSFS, a.k.a. SMARTF) are given in the galTechMemo_2005_01 entitled "Least Squares Frequency Switching".

We generate lsfs files using the lsfs_wrap procedure. Its inputs include the path and array of fits files to be treated. See the document "HOW TO GENERATE MH AND LSFS FILES".

4.1. Manual selection of files within a given day for LSFS processing

One problem that can occur: the software selects the LSFS files automatically, and these files are big. If it selects too many the machine's memory will be exceeded. If this happens you need to intervene by hand and restrict the number of fits files by using a shorter, hand-determined list.

You may need to manually select input files for LSFS if there are long stretches of contiguous LSFS calibration; loading all these fits files would exceed the computer's memory. If you need to select the input files for LSFS manually, you produce a list of a few files that contain contiguous LSFS calibration data that will be all treated as one group and averaged together. You use these as input to `lsfs_shell`, which oversees the LSFS calibration by invoking `lsfs`, which is the "boss" for the LSFS reduction; invoking `ggnb_recon`, which reconstructs the nb bandpass; and writing out the save file.

4.2. The LSFS save files

The save file is of the form
`lsfs.yyyymmdd.tttttttttt.proj.nnnn.sav`
where `tttttttttt` is the `utcstamp` for the first LSFS record (that's the time in seconds since 1971), and the other parameters come from the fits file that contains the first LSFS record. The `utcstamp` is included to make it easier for future programs to select the nearest LSFS calibration.

4.3. The procedure *lsfs.pro*

`lsfs.pro` has as input the set of fits files containing one group's worth of SMARTF data. It returns both wb and nb i.f. filter shapes (`ggwb[512,14,2]`, `ggnb[480,14,2]`), the r.f. spectra at each of the 7 l.o. frequencies and cal on/off (`rf4wb[512,14,7,2]`[1], `rf4nb[480,14,7,2]`), the derived r.f. spectra over all frequencies covered by the 7 l.o. frequencies (`rfwb[543,14,2]`, `rfnb[543,14,2]`), calibration factors, the nb baseband frequencies for each channel (`bbfrq_nb[7679]`), the nb theoretical bandpass shape (`bbgain_dft_nb[7679]`), the r.f. frequencies for each channel (`rffrq_wb[512]`, `rffrq_nb[7679]`), and the cal deflections in original recorded units.

The astute reader will have noticed that the nb spectra have 7679 channels but we return only 480 channels for `ggnb` and `rf4nb`. The reason is that the smallest l.o. separation in the series of 7 l.o. LSFS frequencies is about 195 kHz, which corresponds to 224 nb channels. There is no information on the frequency structure within this 224-channel range. We assume that there is no structure on that frequency range and in `ggnb_recon` do a fancy-dancy interpolation to recover the nb i.f. gain for all 7679 channels. **THIS INTERPOLATION NEEDS TO BE REVISITED**

As the boss, `lsfs.pro` invokes the following procedures:

[1]The index order is [channels, 14 receivers, 7 lo frequencies, and cal on/off].

4.3.1. m1_to_m1s_s0.pro

.

This takes the entire array of `m1` structures in the designated input files and filters them to make sure which are *really* LSFS, creating the new array `m1s`. In doing this it discards all leading datapoints with CALON, all trailing ones with CALON in excess of 3 minutes (180 datapoints), and incorporates an estimated time delay for the l.o. to change. It returns the `m1s` structure for processing.

All of this would be better done in the program `find_smartf` because memory is an issue and we would be saving fewer unused `m1` datapoints in memory. This change should be incorporated sometime.

4.3.2. lsfs1.pro

This performs the awful job of deciding which records are indeed valid. It has to find 7 l.o. frequencies that are used in LSFS; determine which have cal ON and OFF; discard datapoints near the transitions, because the transitions don't occur on one-second ticks; and check to make sure that data really are LSFS and don't simply have a bunch of different l.o. frequencies that were used for a different purpose. I've tried to document the different tests within the code. These tests are quite involved and complicated—if you have to work with them I hope the descriptions are adequate, but you will probably need to sample some parameter values when reducing real data to figure out what's really happening.

One slightly tricky part in processing the wb and nb spectra is arithmetically using the long integers, which are in the `m1` structure from the fits file. You can't add many of these together without overflowing the long integers. And if you convert them individually to floats you lose accuracy; if you convert them to doubles it costs memory. See the code for the best technique.

One required operation that occurs in this program is averaging a number of one-second records together. To accomplish this in IDL we use the `total` function. You have to be careful with memory and computing speed, and the fact that the arrays are in long integers. To use `total` the best way: `total( long64( m1s.g_wide))`. Note the conversion from `long` to `long64`—important!!

4.3.3. carl9.pro

`carl9.pro` does the actual LSFS solution, using the SVD technique described in `galTechMemo_2005_01`. One important set of its inputs is the set of matrices and vectors involved with the SVD matrix math. Those are called `xmatrix`, `wgts`, `wgt_inv`, and `xxinv_svd`, which correspond to the quantities $\mathbf{X}$, $[\mathbf{W}]$, $\left[\frac{1}{\mathbf{W}}\right]$, and $\boldsymbol{\alpha} \cdot \boldsymbol{X^T}$ in equations 28 and 29 of that memo. These quantities are read from files on disk because they are computationally very expensive to calculate. If you ever want to generate new ones, which would occur if you change the relative separations, number, or sequence of the LO frequencies used in the calibration, you need to generate new ones using `xmatrixgen.pro`. Here we don't give the details on how to use it. In fact, it's not documented either; this needs work!

One important optional input is `quiet`. If this is not set, you can watch the convergence of the iterations; it's kind of fun (for a while). It probably takes longer to plot than to calculate, so...

5. GSR/PROCS/INIT/LSFS—CORRECTING DATA FOR IF BANDPASSES ETC

After having derived the i.f. bandpasses we want to use them to get bandpass-corrected data—after all, that's the whole rationale for this mess! We do this with `polycorr.pro`, which is normally invoked by `m1polycorr.pro`. Inputs to `m1polycorr.pro` include the lsfs file name, which file contains the derived bandpasses, and the structure m1 which contains the spectra you want to bandpass-correct. The important outputs that need explaining are generated by `polycorr.pro` and are detailed below.

`polycorr.pro` does the work. It does the following:

1. An important function is to use the wb spectra to baseline-correct the nb spectra. We do this by polyfitting the wb spectra. The polyfit coefficients need to carry over exactly from the wb to the nb spectra. Each wb channel has the width of about 224 nb channels. To match things up, we need to lump together 224 nb channels for each wb channel and be very careful about getting the centering right. We do this by binning the rf frequencies of the nb spectra (called `rffrq_nb`) to exactly (almost) match those of the wb spectra (called `rffrq_wb`). After this binning operation, the nb spectra match the shapes of the wb spectra to a truly remarkable degree of accuracy (which they should—because the nb spectra are digitally derived from the same digital data stream as the wb spectra, and in particular the nb filter shape is defined digitally).
2. Each wb and nb raw spectrum is incremented by the long integer offset 2^{31}, converted to float, and divided by its appropriate i.f. bandpass.
3. There is a d.c. spike in channel 256 of each of the 512-channel wb spectra; we interpolate over it.
4. We bin the nb spectra according to the above prescription, extract the ~ 33 nb binned channels that correspond to the wb ones, and derive the ratio of powers, $fctr = \frac{nb}{wb}$. We multiply all wb spectra by this factor to get it on the same power scale as the nb spectra.

5. `fctr` is the factor by which the original wb spectra (after bandpass correction) were multiplied to get them on the same scale as the nb spectra. That is, the output spectra are called `swb_c` and `snb_c`; `swb_c` and its pre-polyfit channel-integrated power `pwb_uc` have been scaled by `fctr`, so you can recover the original unscaled spectra and powers by dividing the returned `swb_c` and `pwb_uc` by their appropriate `fctr`.

6. We fit an n^{th} degree polynomial (input parameter `degree`) to the wb spectrum and use the very same coefficients to correct the nb spectrum. The wb spectrum is 100 MHz wide, the narrow about 7 MHz wide. At the moment the default degree is 18. This is a problem—instead of using a large degree we should use a smaller range in the wb spectrum. **THIS NEEDS INVESTIGATION–HIGH PRIORITY**.

This research was supported in part by NSF grant AST 04-06987 and by the NAIC.