

GALFA Codes, Explanations and Examples
Samantha Hoffman
June 2007

1. Before You Begin the Data Reduction Process

Before beginning any data reduction, it is important to organize the data by date in a word processing file, such as an emacs file. This will save you lots of time in the future - trust me. To open a new file, type: **emacs name_of_file &**. Inside this new file create a table, that includes columns labeled: date, mh files, slst, elst, scan #, and done/saved. Once this is done, you need to find out if all of the mh files exist for a particular day. To do so, log into Arcibo and type: **cd /share/galfa/galfamh**. Then type: **ls *date*project*** (fill in the appropriate date); this will bring up a list of all of the mh files for that particular day. Place an x or another symbol in the mh file column in your table to let you know that a particular file has an existing mh file. Not doing so may cause you problems later, as any missing mh file must be generated.

Next, find the scan numbers and the object name. This information can be found in the all of the mh files. In IDL, type: **restore, 'file.sav', /ver** (filling in the appropriate file name). Then type: **help, mh, /st** and look for a parameter called OBJECT. It will look like this: ms_00_00. The first word and/or numbers is the object name, and the middle number(s) is the scan number. For instance, this would be scan 0.

The next step is to determine the scan start and end times, or slst and elst. This can be done in two ways. The first way involves looking at the mh files again. In IDL, open the first mh file for that scan/day with: **restore, 'file.sav', /ver** (fill in the appropriate numbers). Then type: **help, mh, /st** and look for LST_MEANSTAMP DOUBLE. This is the slst time. Or, you can type: **print, mh.lst_meanstamp**, and take the smallest number.

To find the elst, open up the last mh file of the scan in IDL and type: **restore, 'file.sav', /ver**. Then type: **print, mh.lst_meanstamp**, and the last number is the elst.

The second way to find the slst and elst is a bit more complicated. It involves using a handy program called endfinder. Open a new emacs file, and call it endfinder.idl. Type:

```
endfinder, year, month, day, proj, root, lsts, mhdir=mhdir, th
```

Above this, define your parameters. Here, root = '/share/galfa'. Lsts is just what the outputs are called. Mhdir = '/share/galfa/galfamh'. Th refers to thickness of the dots, and can be set from 2 to 4. After you have saved this file, bring up IDL and type: **addpath, '~goldston/AUX'**. Then type: **.r opixwin**. This gets the program running properly. Then call the file by typing: **@endfinder.idl**.

Endfinder is tricky to learn at first (especially when you aren't given any directions). It requires a 3-button mouse. The program will bring up an image of the scan in color-coded dots. You need to zoom in to find the best starting point. To do so, use the middle mouse button. First click on a region near where you want to zoom in. Then click the middle button again, and draw a box around the area. You can continue to zoom in like this until you've isolated a small region of blue dots. When you want to choose a particular dot, click the left mouse button. A red circle will appear. Choose a specific dot by clicking the middle mouse button. The image will then zoom back out again and allow you to repeat the process to find a "good" end point. When you

have finished, the program will take you back to IDL. BEFORE you exit IDL, type: **print, lsts**. These are you start and end times.

Once you have finished all of this and entered all of the necessary information into your table, it is time to start the data reduction process!

2. CALIB and stg0.pro

Snežana Stanimirović wrote the following version of the code.

Stage zero is the first major step in data reduction, and is also the longest part of the reduction process. Stage zero is designed to calibrate the data's fits files into temperature units, to remove excess telescopic noise by removing the IF bandpass, and to perform a Doppler correction. It also organizes the data by scan (or by day) and stores the .fits and .sav files into appropriate subdirectories.

In your home directory, open a new file in emacs by typing: emacs name_of_file.idl &. It is important to add the "&" sign after the name of the file so that you can continue to type in your xterm. It is also appropriate to name the file after the code. For example, I'd name this code's file stg0_a2032.idl. In this new file type:

```
stg0, year, month, day, proj, region, root, startn, endn, slst, $
elst, scan, /nomh, calfile=calfile, $
fitsdir=fitsdir, mhdir=mhdir, $
caldir=caldir

stop
```

Above this code, define your parameters. The defined parameters given here are what were used when running stage zero on data from Arecibo project a2032. Make sure everything is defined as a string, unless otherwise stated.

root = this should be your home directory, or the directory where all of the data can be found.
Ex: '/share/galfa/madison/Samantha/'

proj = Arecibo project name; ex: 'a2032'

region = this "name" is found inside of all of the mh files that correspond to the data. Ex: 'ms'

slst = the first time-stamp of the scan; take this number directly from your table. This is defined as a **number**.

elst = the last time-stamp of the scan; take from your table. Also defined as a **number**.

/nomh = also can be typed in as nomh=nomh. This is a way of telling the code that it does not need to generate new mh files.

calfile = this is a particular location where the code can find a specific calibration file. This parameter does **not** need to be used unless the code keeps breaking/complaining. In the case that this does happen, you can use this parameter to specify the location of a "good" file from the

previous scan for the code to use instead. Check with your professor/advisor before putting this parameter to use.

fitsdir= this is where the “raw” fits files are kept; This is always: ‘/share/galfa/’

mhdir = this is where all of the mh files are located. This is always: ‘/share/galfa/galfamh/’.

caldir = this is where the code can find all of the lsfs files. Ex: ‘/share/galfa/madison/Samantha/a2032/ms/lsfs/’

Define the year, month, day, and scan inside of the code. Remember to change these each time you run stage zero!

Outputs: The output files will be in the directory that matches the name of your object. For instance, my files were located in /share/galfa/madison/Samantha/a2032/ms. There will be two files for each scan; one that is .fits and one that is .sav.

Note: from now on, neglect the parameters of tdf and odf, which can be seen in every code in the packet, because they refer to a two-digit version of the code that was used in the past.

3. SPMOD and spcor.pro

Spcor.pro is contained in the SPMOD software. This code is very simple, but it takes a significant amount of time. This gets rid of all of the major ripples in the spectral baseline caused by wave reflections inside of the Gregorian dome. Because this addition to the baseline ripple is relatively constant from day to day in each receiver, this information can be used with some fancy HI modeling to decrease the contribution to the spectra by the noise.

Note: the GALFA pipeline document states that spcor.pro needs to be run twice, once before XING and once after XING. I have been instructed otherwise by Josh Goldston himself to not run spcor.pro a second time.

Also, if you are reducing a project that does not require the XING section, you only need to run this program once.

To begin, open up another new emacs file and type:

```
spcor, root, region, scans, proj
```

Define root, region, and proj the same way that you did for stg0.pro. Scans refers to the number of days in the project and is defined as a **number**, not a string.

Outputs: This code will produce two files, aggr.sav and spcor_##.sav, both located in the object directory.

4. XING and xgen.pro

This is the first step in the XING data reduction set. Xgen.pro finds all of the crossing points in a basketweave scan dataset. This is particularly useful because it will find and use the best data point for the final map.

In an emacs file, type:

```
xgen, root, region, scans, proj, goodx=goodx, xday=xday, blankfile=blankfile
```

Define root, region, scans, and proj as before. You do not need to define anything for goodx or xday; goodx refers to the output 7x7 matrix and xday is a matrix that is equal to (scans x scans). The blankfile is a user-generated file that contains all of the “bad seconds” from a particular scan. I personally did not create my own blankfile, but I do know that the program is called edblanks.pro. As I’ve never used the program, I’m afraid I can’t give any proper advice on how to use it. But if one is generated, define the path to the blankfile as a string.

Outputs: The output files appear in the xing directory, and look like: xing/regAAA_BBB.sav.

5. lxw.pro

Lxw.pro uses the file from xgen.pro, and loads all of the spectra into the chosen crossing point files. Be aware that this can take a long time!

In a new emacs file, type:

```
lxw, root, region, scans, proj, $  
/no_over, xday=xday
```

Root, region, scans, and proj are the same as before. The /no_over parameter is used for the sake of time. If the program should get interrupted and you need to run it over, the code will not reload data that had been previously loaded. Xday=xday calls the matrix that you made in xgen.pro.

Outputs: This code produces a file that looks very similar to the one produced by xgen.pro. It takes the form: xing/date_1.sav. There should be at least one of these files for every scan.

6. xfit.pro

Xfit.pro determines the relative “point-to-point” gain for each crossing point. This will help the image look smoother by further reducing noise from the telescope’s beam gain. It sounds confusing, but all it does is plots two spectra against each other and fits a line to their slope. This program does not take long to run.

Type, in a new emacs file:

```
xfit, root, region, scans, proj
```

All parameters are the same as before.

Outputs: The output file looks like: xing/date_f.sav, similar to the previous two. There should be a corresponding “f file” to each “l file” from lxw.pro.

7. lsfxpt.pro

This code is rather complicated, but don't be afraid! This creates a matrix that will fit all of the crossing points by using a predetermined ratio for the beam gains. It is important to note that this particular code can take a very long time to run, depending on how many files you have. Make sure that you are using a computer that no one else is using or you may run out of memory!

In emacs:

```
lsfxpt, root, region, scans, proj, $
degree, xarrall, yarrall, name, $
fourier=fourier, daygain=daygain
```

Define name, as a string, as whatever you want to call the output file. This parameter will be used in another XING code, so make sure it is easily identifiable to you. Ex: ‘first’

degree = 0; this should be defined as a number, and should be set to the degree of polynomial fit desired. Set to -1 is you want a polynomial that has no dependence on right ascension.

fourier = [1,3]

Note: I defined degree and fourier as suggested in the GALFA pipeline document.

I also defined the daygain parameter, even though you don't have to. You can choose to set the beamgain parameter instead. The beamgain parameter is for the creation of independent zeroeth order fits for the gain of each beam.

daygain=1; this creates independent fits for each day's gain. Define this as a number.

xarrall= the x parameter in the least square fits (LSF); you don't have to set this to anything, just type in this parameter in the input code.

yarrall = the y parameter in the least square fits (LSF); this does not need to be set to anything either.

Outputs: This saves an easily identifiable file in your xing directory. It looks like: /xing/reg_lsfxpt_name.sav.

8. xg_assn.pro

Xg_assn.pro takes the file produced by lsfxt.pro and assigns the resulting gains to each point in the data set.

```
xg_assn, root, region, scans, proj, fitsvars, name
```

Root, region, scans, proj are still the same as always.

fitsvars is an output for the “variables that get generated while solving the matrix-inversion problem.” You do not need to define anything for this parameter; just enter it into the code.

name = the same name that was given to the output file in lsfxt.pro.

Outputs: xg_assn.pro produced 3 output files. They take the form of: “/reg_NNN/reg_NNN_xing_name.sav,” “/xingarr_name.pro,” and “/xga_name.sav,” respectively. These can be found either in the xing directory or the object directory.

9. GRID and todarr.pro

This is the first of the gridding codes! And you’ll be happy to know that this code does not take very long to run!

```
todarr, root, region, scan, proj
```

All of the parameters are the same as the ones previously defined.

Outputs: The output file will appear in your object directory as todarr.sav.

10. sdgw.pro

This code will make the final data cube! It takes all of the previous data and uses it to make a visual representation.

In a new emacs file, type:

```
sdgw, root, region, proj, gridname, lon0, lat0, spmax, $  
spmin, spres, imsize, $  
spname=spname, name=name, vel=vel, $  
norm=norm
```

gridname = ‘map1’, or whatever you want to call your final data cube. Make it easily identifiable, and be sure to change the name every time that you make a new map.

lon0 = center of longitude in degrees. This will take some calculations, as well as some trial and error. You can determine this number by using a rough estimation of the middle time stamp of all of your scans, and multiplying that number later. You can move the map over in another data cube if the number is not correct. Enter in this parameter as a **number**.

lat0 = center of latitude in degrees. To get this number, open an mh file and look for the declination parameter, abbreviated as DEC_HALFSEC DOUBLE. Print the minmax of this and take the middle point. Enter in this parameter as a **number**.

Another way to find both the center of latitude and longitude is to plot RA and declination points from the mh files to see exactly what region of the sky the scan observed. Just type: **restore, 'file.sav', /ver** and then: **plot, mh.RA_HALFSEC, mh.DEC_HALFSEC**. Then use the graph to determine the central points.

For the next part, make the numbers small for the first couple of maps or else the files will be very large and take up valuable disk space.

spmin: the minimum spectral channel. First open up any fits file in IDL and use: **data=readfits('filename')**. Then type: **plot, data[* , 1, 1, 300** (or whatever number is in the middle of the data set – this printed when you use the data=readfits function) **]**. Look for the main emission peak and pick a channel near the center of the peak. This is your spmin, and is entered in as a number. Later on you can take a number that is 100-500 channels below where you want to graph.

spmax: the maximum spectral channel. Take a number one or two channels away from the number you chose for the minimum. Later on you can take a number that is 100-500 channels above where you want to graph.

spres: 1; this is the spectral resolution. 1 is not too big and should be good for most maps. When you are making maps with 500 to 1000 channels, it is appropriate to set spres to 10.

imsize: This is your image size. This parameter assumes that you are using a pixel size of 1 arcminute (if this is not what you want, there is a parameter that you can set that will change the pixel size to whatever you want).

To determine the size of the final image, you need to know how large of an area you are dealing with. Each hour of right ascension needs to be multiplied by 15 to get degrees. Then, multiply by 60 to get your needed image size. For declination, simply take the number of degrees the map will cover and multiply by 60. Take these two numbers and enter them as an array, i.e. [1000, 800]. *Note: it is best to try a map that is of image size [50, 50] or [500, 500] the first time that you make a data cube.*

name = This is the name that you have used in lsfxt.pro, xg_assn.pro, and todarr.pro.

vel = This will convert the map from being temperature dependent to being velocity dependent, which is much easier to read.

norm = 1; "This allows the user to specify a normalization of the entire data set to divide by." It is good to set this to one as a "safety precaution."

Outputs: The final data cube! It will have two components, /gridname name.fits and /gridname name.sav in your object directory.

11. What to do After the Final Cube is Made

There are few things for you to do after you have made a data cube. The first thing you can do is look at the map using a program called karma. When you are inside of the directory that contains the map, type: `/usr/local/karma/bin/kvis name.fits`. It will bring up the karma program for you to view and play with your map.

If the cube looks good, you should secure copy it over to a special directory on Snežana Stanimirović's computer. When you are located in the directory that contains the file that you want to copy, type:

```
scp name.fits username@leffe.astro.wisc.edu:/d/leffe/sstanimi/UgradResearch/Data/
```

When prompted, enter in the password that you use when you log in to your computer and not the remote GALFA password.

You have completed this part of the research process! Feel free to go back and make more maps using higher channel sizes and image sizes! And do not be afraid to ask questions as you go – this is a learning experience for everyone involved!

Reference

Peek, Joshua E.G. The GALSPECT pipeline version 2.2: GALSPECT with a vengeance. 1-12. 26 August, 2006.

Appendix

In order to make sure that you are using the most recent version of all of these codes, it is best to type into IDL the following before calling a program:

```
addpath, '~goldston'  
addpath, '~goldston/calib'  
addpath, '~goldston/AUX'  
addpath, '~goldston/GRID'  
addpath, '~goldston/XING'  
addpath, '~goldston/SPMOD'
```

*If you have any questions or suggestions about this document, please contact me at shoffman3@wisc.edu
This document was completed on 22 January 2007. It was updated on 26 June, 2007.*