LASER INTERFEROMETER GRAVITATIONAL WAVE OBSERVATORY

-LIGO-

CALIFORNIA INSTITUTE OF TECHNOLOGY

Modification and Additions to GRASP: PEM Characterization and General Analysis Tools

Publication: #T010009-00-L 2/04/01

by

Anthony Rizzi

Distribution:

all

This is a publication of the LIGO Project.

1 PEM Characterization and General Analysis Tools

These tools consists of modifications and additions to previous chapters of GRASP that:

1. Allow use of stored latest LIGO frames and frames from shared memory at LHO or LLO with ability to analyze an arbitrary channel.
2. Allow characterization of Physics Environmental Data (PEM). For example, these tools can be used to establish "chirpiness" benchmarks. For definition and further explanation of how this is done see section 1.5.
3. Allow creation 1-D and 2-D histograms directly from matched filter program output.
4. Give new scripts and c-programs that allow one to make histograms directly from output of program multifilter*.c series. They allow one to impose parametric tests before binning. A matlab program to display is also included
5. Allow matched filtering of ASCII files. It also allows injection of signal from ASCII file into data.
6. Allow software injection of multiple chirps of varied type in banked matched filter runs.
7. Scripts have been written to facilitate changes to GRASP structure to allow easy modification of software for other related uses.
8. Script has been made to facilitate use of GRASP itself.
9. ASCII files that give trees that breaks out a few of the GRASP functions by file and directory location.
10. Function (snapshotF) added for static viewing of a portion of a channel in a frame.

These tools are described in more detail below. First, general purpose tools are discussed including setgrasp and "tree" files. Next, functions that allow viewing data are discussed. Third, modifications to optimalF.c (the programs that run frames through a single matched filter template) are discussed. Then, modifications to multiprocessor (mpi) code that run frames through multiple filters (multifilterF.c) are covered. Following this is a description of the code that allows histogram making from multifilter* series output. Then comes a description of how to make some of the changes found in the new code in unchanged code. An explanation of benchmarking of PEM channels comes next. For example, the script make_new_tail is explained. Lastly, a summary of all modules that must be changed to accomplish changes and additions of the type found herein.

1.1 General purpose tools

We begin with two general purpose files that facilitate use of GRASP:

- `setgrasp`
  - A script that allows one to set environmental variables by question driven menu. The script needs to be modified for your directory choices. Once this is done the process of running GRASP becomes much easier. I recommend putting the line "source setgrasp" in the .cshrc file of your GRASP login.
- `Tree files`
  - These files are found in GRASP_HOME/tree and can be searched in vi or emacs or your favorite editor. One often finds in reading a particular GRASP code file that he does not know what module a function is defined in. Further, one does not usually know where that module is. For animateF.c, snapshotF.c and optimalF.c, the first problem is solved by using the tree files below. Simply search for the desired function, the file has an arrow that emanates from the module (c-file) that contains the function. The second problem can be solved, among other ways by using GRASP_tree which lists the directories followed by the files in that directory.
  - The tree current files are given below:
    - animateF_tree
    - snapshotF_tree
    - optimalF_tree
    - GRASP_tree contains a list of all directories in GRASP

1.2 Frame access tools

The following changed functions (originally in chapter 3 of the GRASP manual) and one function not in GRASP previously, are functions that allow viewing data.

- `animateF.c`
  - It now allows use of shared memory frames at either LIGO site. For more details see optimalF.c below. It is found in GRASP_HOME / src/ examples/ examples_frame. The execution format is as previously.
- `snapshotF.c`
  - This function has been added to allow static viewing of some portion of the data. It is very useful for more closely inspecting a portion of a frame when one suspects something interesting (e.g. after doing a matched filter test). It is found in the same directory as animateF.c. This function can also view shared memory frames at either LIGO site. For more details see optimalF.c below. The execution format is: "snapshotF 0 2 | xmgr -pipe &". The first number is an integer in seconds indicating when to start and the second number is an integer in seconds indicating when to end. Next, there follows a pipe to xmgr so that the results can be displayed.
- `calibrateF.c`
  - This function has been modified in a similar way to optimalF.c below. It allows use of shared memory frames, arbitrary channel and transfer function.

1.3 Filtering data with a single matched template

The following changes have been made to the program module optimalF.c, chapter 6 of the GRASP manual. The optimalF program takes frames as input and runs a single matched filter across the data. If a given segment of data has an signal to noise ratio (SNR) greater than a given value the segment is then subject to two further tests: gaussianity and a test that returns the probability that the segments frequency distribution is gaussian noise plus a chirp of the prescribed type. The optimalF.c module and modules that are derived from it are described below. Note that the naming convention is that "F" in optimalF.c means optimal filtering applied to frame files. Whereas the "ASCII" in optimalAscii.c means optimal filtering applied to an ASCII file. The "matlab" nomenclature below is based on the fact that the ASCII files used by us were generated by matlab; one can of course generate the ASCII files in any manner that respects the rule that the file be a column of numbers(cf. the gaussian white noise file: GRASP_HOME/ src/ examples/ examples_inspiral/ matlab_noise3 as an example). In all the below, please set the path to find your transfer function. To do this, search for filep (the file pointer to the transfer function file) in the code and change the argument of fopen to the desired path and file name. Two transfer functions are included in the directory GRASP_HOME/ src/ examples/ examples_inspiral/:

- - ssones.ascci-which is a unity transfer function
  - SeismicStack2.ascii-which is the transfer function of the seismic stack using a model in given PEM document written by Ed Daw. Also, included in this transfer function is the transfer function of the pendulum's that the test masses are hung on. Also, the transfer function has been multiplied by an over all constant.
  - The format of these files is three columns: 1-frequency 2-real part of transfer function 3-imaginary part of transfer function. One must set LIGOfrinum at beginning of file to the total number of points (nb. $LIGOfrinum = numberoflines * 3$).
- `optimalF.c`
  - This function can now grab LIGO frame data *.frame (take care if you have frames that are newer than version 4.11; this release has been tested with 4.11 but no later version).
  - It can now grab a channel from an environmental variable called GRASP_CHANNEL. For example, one might set this variable to L0:PEM_BSC4_ACCX which gives the accelerometer reading in the x-direction at the west end station. As per usual takes it data from GRASP_FRAMEPATH.
  - It can now collect frames from shared memory at either LIGO site. If one wants to collect data from live frames at LLO (LHO) on decatur (fortress), one creates a directory and puts an empty file

called LLO_shared_mem (LHO_shared_mem) into it. As long as shared memory has been correctly setup with the handle being /online/LLO_Online (/online/LHO_Online) (for example, in LLO the environmental variable LIGOSMPART must be set to /online/LLO_Online). Run "FrDump" to see that frames are indeed being put into shared memory. Also, be careful that enough shared memory has been allocated to keep frames for long enough time that calculations can be done. If one is using computers that are being used by others, be sure to remember to run GRASP only at times when the load put by GRASP will not interfere with others work. If one has to interrupt the program during frame collection, type "smrepair" to keep from tying up shared memory for other users; this is also a good thing to do if one is doing a lot of short runs.

  * In order to accomplish the interface with the DMT, I had to compile the program with a c++ compiler rather than just a c compiler. This was done two ways: the way it was done here with optimalF did not involve changing the file but added another file called main_assign.c. The second method involves introduces changes in the files themselves GRASP_HOME/ src / examples/ examples_template_bank/ multifilterF.c for an example of this.

- It now handles transfer functions. The transfer function is applied to the data before it is sent to the optimal filtering and veto coding. This function can be any "frequency filtering" one wants to apply. One manually enters a transfer function file that will be applied to the raw data to take one from the output of the measuring device to motion of test mass due to specified PEM channel (SeismicStack2 is an ASCII file that contains conversion from the output of large Wilcoxan (model #731A) accelerometer (through seismic stack and test mass pendulum– note a particular scalar factor has been applied to the original transfer function to emphasize a certain frequency range). There is also a function optimalF_with_Pem_transFun_env_var.c that allows one to set path to transfer function using an environmental variable rather than hard coding.
- It now has debug functionality to check for unity response function by using the difference. This code is present in optimalF.c and is also recommended as a template for other types of testing and functionality.
- It now puts the mean power spectrum into a file after "decay" (nominally 15) calls to avg_inv_spec. Two files are made: one, called seismicN.dat, contains just power spectrum numbers for each bin; the other, called seismic.dat, gives the frequency in first column and power in second
- It now handles PEM sized data. In particular, an argument of is_gaussian has been increased.

- `optimalF-noise_make_file.c`
  - This function is a modified version of the optimalF program described above.
  - Its main feature is that it generates the two noise files described below. These files allow one to use N segments of data to create a simple average background noise that is then used for the optimal (matched) filtering in the program; this, for example, provides a way of comparing the results of optimal filtering with local exponential averaging (which is what is done in optimalF.c) to optimal filtering with simple arithmetic average of a ”typical” portion of the data (which is what is done in optimalF-noise_use-file.c below). The two files are put into the directory that optimalF-noise_make_file is executed (to change search for “fptwice” in the code and change the path accordingly). The two files created are:
    - data-mean_pow_spec (this one starts from icut and goes to npoint/2-1) and
    - data-twice_inv_noise (this one starts at zero and goes to npoint/2)
  - These files contain respectively the arithmetic average of successive data segments of mean and twice inverted noise (i.e. of data). The average is done by a function, simple_avg_inv_spec, (instead of avg_inv_spec) found in matched.c in $GRASP_HOME/ src/ inspiral. The net effect is to give average mean $power = (P1 + P2 + P3....PN)/N$. The files created by this program are then to be used by optimalF-noise_use_file to perform the optimal filtering (one filter) on the given data channel.
  - It has the capability of injecting chirps from ASCII files. The chirp or other desired injected signal is inserted into an array by this code and then can be put into the data stream using time_inject_chirp.

- `optimalF-noise_use_file.c`
  - This function uses a simple average of N segments of ”typical” data for the optimal filtering. It does this by making use of the files data-mean_pow_spec and data-twice_inv_noise generated by optimalF-noise_make-file.c for the optimal filtering on frame data. It looks for the files it needs in the current directory (to change this directory search for “fptwice” in the code).

- `optimalAscii-noise_make_file.c`
  - This function is identical to the above function (optimalF-noise_make_file) except that it takes its data input from an ASCII file (this file is specified in code –search for “fpmatlab_noise”; currently it points to matlab_noise3) rather than from frames. It creates two noise

files: mean_pow_spec and twice_inv_noise as done above in the current directory. If one wants to use this for creating noise files for use with optimalAscii-noise-use-file.c set NPOINT=16384 (for multifilterAscii-noise-use-file.c (see below) set NPOINT=65536). There is also a hard coded option (currently commented out) that puts mean_pow_spec in two files:

  * noise_gauss-whtN.dat and noise_gauss-wht.dat. The first file has only spectral power Numbers while the other includes frequency as well as power spectral numbers.

- `optimalAscii_noise-use_file.c`

  - This function uses a simple average of N segments of "typical" data for the optimal filtering. It is the ASCII version of optimalF-noise-use-file.c. The ASCII is accessed as above. It uses the files data-mean_pow_spec and data-twice-inv noise generated by optimalAscii-noise_make_file.c for optimal filtering on an ASCII file. It looks in the current directory for these file (to change look for file pointer "fptwice" in code).

1.4 Filtering data with a bank of matched templates

The following changes have been made to program multifilterF.c, described in section 9 of the GRASP manual. The multifilterF.c program takes frames as input and runs a bank of matched filters across the data. If a given segment of data has an SNR greater than a given value the segment is then subject to two further tests: gaussianity and a test that returns the probability that the segments frequency distribution is gaussian noise plus a chirp of the prescribed type. The multifilterF.c module and the modules that are derived from it are described below; many of them are parallel versions to the optimal*.c series described above. As for the optimal*.c series, the naming convention is that ”F” in multifilterF.c means optimal filtering applied to frame files. Whereas the ”ASCII” in multifilterAscii.c means optimal filtering applied to an ASCII file. Make sure transfer function is set to correct location before running (for more details see description preceding the optimal*.c series).

- `multifilterF.c`
  - This function now allows all the functionality described above for optimalF.c. The file `readme_multifilter_running` contains the command line for running multifilterF; one will need to alter the path to “mpirun” to agree with your installation setup. The number in the file is the number of processors one wants to use. As per usual, one must set the machines in /mpi/mpich-1.2.0/util/machines.

- `multifilterF-noise_make_file.c`
  - Analogous to multifilterF.c, this function makes the two noise spectrum files mean_pow_spec and twice_inv_noise by averaging the first N data segments of the GRASP_FRAMEPATH data in Channel GRASP_CHANNEL. The files are created in the directory that the function is executed in (to change this search for “fptwice_inv_noise” in the code). These files are then used by multifilter-noise_make_file.c. To do averaging, one needs to make use of simple_avg_inv_spec instead of avg_inv_spec (both are in matched.c in GRASP_1.9.8/ src/ inspiral). Both ”make” and ”use” are currently set up to use transfer function of the seismic tack; see optimalF-noise_make_file for a few details of how calculation are done.

- `multifilterF-noise_use_file.c`
  - This file replaces the local exponential averaging done by multifilterF with a simple arithmetic average of ”typical” data segments. Uses twice_inv_noise and mean_pow_spec from ”make” file immediately above. It looks in the current directory for these files (to change, search for the file pointer “fptwice_inv_noise” in code).

- `multifilterAscii.c`

  - This code does local exponential averaging (with 1/e at 15 data segments) to get noise used in optimal filtering on an ASCII data file. It does optimal filtering on the ASCII data file with name and path given in the code (to change the file, search for, “fpmatlab”, the file pointer to the ASCII data file).

- `multifilterAscii-noise_make_file.c`

  - This code *would* do the same as multifilterF-noise_make_file.c, except it would take data from an ASCII file referenced in code (look for file pointer “fpmatlab”). However, there is no need for a new function. One uses optimalAscii-noise_make_file.c to create noise files for use by multifilterAscii-noise_use_file.c. One must use NPOINT=65536, when making a noise file for multifilterAscii-noise_use_file. The ASCII file is accessed as in optimalAscii-noise_make_file.c above. One also should use names for the noise files created by optimalAscii-noise_make_file for multifilterAscii-noise_use_file.c that will distinguish it from the files created for optimalAscii-noise_use_file.

- `multifilterAscii-noise_use_file.c`

  - This program looks at an ASCII file given in code and uses a mean_pow_spec and twice_inv_noise from ASCII files created by a make file from the ASCII data file. Make sure to set the correct path in the code (search for “fptwice_inv_noise” to change).

- `multifilterAscii-noise_use_file-hist_make.c`

  - This code is the same as multifilterAscii-noise_use_file.c but also bins SNR and probability to create 2D histogram ASCII file that is named in the code (initially matlab_hist) (search for the file pointer “fphist” and change the file path/name to what you like) . Once the file is created it can be displayed in matlab using the provided program, ThreeDHist.m.

- `multifilterAscii_with_injectedChirps.c`

  - This program is the same as multifilterAscii.c, but before the template bank is run past the data, chirps are injected into the data. It can be used, for example, to determine how accurately one can reconstruct chirp parameters using find_chirp. The following (#define) parameters must be set and explain the operation of the program.
    * INJECT_CHIRPS: This is “1” or “0” variable that one should set to “1” if one wants to inject chirps.
    * INJECT_SINGLE_CHIRP: This is 1 or 0 variable that one should set to “1” if one wants to inject only a single chirp not a series of chirps.

∗ N_TIMES_INJECT: It specifies the number times to inject the same chirp before going on to next chirp. If it were set to 50, with 66 templates and no skips, this requires 3300 data segments; note that the segments after this will have no injected chirps.

∗ DATA_SEGS_WAIT: This variable is currently set to 15. With this value, it says to inject a chirp only into every 15th data segment.

∗ N_TEMPLATES_TO_SKIP: This says to do every N_TEMPLATES_TO_SKIP in the template grid.

1.4.1 Histogram binning and drawing

The following are files that contain programs for binning the output of multifilter* programs.

- `bin_for_hist.c and related scripts for making histograms`
  - This is a program that takes the output c-shell scripts below and creates a histogram in an ASCII file. The multifilter programs above create signals.***** files (each file corresponds to one block of data with results for all matched filter templates) that are used as input by the scripts below. These scripts (found in GRASP_HOME/ src/ examples/ examples_template) are:
    - program_max_prob- script that picks out max probability (prob) in each signal file (signals.* from multifilter) and creates a sorted list in file named max_prob with highest probability at top. The list contains the signal file name that it found along with the probability.
    - program_max_snr – does the same as max_prob but sorts out signal to noise ratio (SNR) instead of probability. The output goes in file named max_snr.
    - program_max_prob-withSNR – same as max_prob but includes SNR in output. The output goes in file named max_prob-withSNR.
    - program_max_snr-withPROB -same as max_snr but includes probability (prob) in output. The output goes in file named max_snr-withPROB.
    - program_prob+snr - same as program_max_prob-wtihSNR but removes ”variance” name that sometimes appears in place of number when using above.
    - ThreeDHist.m-matlab program that allows display of output.

- `signal_test_multi_files.c`
  - This program takes as input the signals.* files which is the output from multifilter programs. Signals.* is an 11 column array with N (number of templates) lines per file. This program outputs 12 columns: the first 11 are as in signals.* the last is the line number (or template number). It outputs only those that have passed a windowing test (i.e. withinrange (maxvalue, minvalue)). The program prompts you for which parameters you would like to use as a window test; answer y (otherwise n) to those you wish to use and it will prompt you for minimum and maximum values. We output this into a file called tested_*.out for example, if one has used test $SNR > 5$ then file is tested_snrgt5.out.

- `histogram1-12.c`
  - It takes as input the output of signal_test_multi_files.c, bins a one dimensional histogram with given min and max values, (any values above max are binned into max any values below min are binned into min) and outputs into a file with a naming convention like hist_snrgt5_snr.out. The "12" in the name of this program refers to the number of columns it accepts in its input file; it is expected that it will be expanded.
- `histogram2-12-col.c`
  - This program is the same as above except it bins a 2D (instead of 1D) histogram. The output file naming convention, for example file, is hist2d_snrgt5_snrprob.out where we are binning SNR and probability and have used the condition in signal_test_multi_files.c that the SNR is greater than 5. That is, only template lines inside signals.***** that pass this SNR test are used. The output format is columns of numbers; this format will not be accepted by Histmat.m. The program below, histogram2-12-hist.c, will output a format that can be displayed using the matlab program Histmat.m.
- `histogram2-12-hist.c`
  - This is the similar to above program, histogram2-12-col.c, except outputs a file that is in the form of a matrix that looks like a 2d-historgram. An example of the output file naming is hist2dform_snrgt5_snrprob where we are binning SNR and prob for values of SNR greater than 5. Such an output file is what Histmat.m expects. The matlab program Histmat.m can be used with this as input to create a pictorial input.
- `Histmat.m`
  - This is a matlab prgram that uses output from programs specified above to draw histogram.

1.5 Making further changes

- In general to make any changes to the code simply make the change and type "make" and any file changes made will be compiled in. To make the changes "permanent" one may like to use the new script :
  - make_new_tail
    - This script is run as "make_new_tail < Makefile". It takes the Makefile and strips out the last part and dumps it into Makefile.tail. It saves the old Makefile.tail before over writing it. The stored Makefile.tail format is Makefile.tail.0 where the "0" can be up to "2".

To convert a file that is not already converted to reading frames from shared memory follow the following procedure. Suppose we are editing a program file called program.c:

- Change the Makefile (and the Makefile.tail (see make_new_tail script above) if you wish the changes to remain when one recompiles GRASP) every $(CC) command in the compile block for program.c to $(CC_PP). This will cause the program to be compiled by your chosen c++ compiler.
- Type: make. The compile will give errors for the items that cannot be found. For example, realft() looks for certain c++ libraries. Make a list of all the functions that cause errors.
- In program.c, comment all declarations for these variables then insert the following lines just below the last preprocessor directive at the top of the file:
  - extern "C" {int FrIOOpenR(int,int,int);void realft(float*, unsigned long,int);}
  - int dumbframe=0
  - At the beginning of the "main" subroutine after the declarations insert the line: while (dumbframe) FrIOOpenR(0,0,0);
  - Run make to compile changes.
- For an example of how this is done, one can look at GRASP_HOME/ src/ examples/ examples_template_bank/ multifilterF.c

1.6 Characterizing the PEM data with GRASP

This section gives a brief account of the use of the code in this chapter for benchmarking PEM data.

The central problem of gravity wave detection is extracting the very small signal from the background noise. Experimentally, this means isolation from the noise and design strategies that emphasize signal and suppress noise. From the data analysis point of view, a significant part of the problem is making sure that an apparent gravity wave detection is not really just noise masquerading as signal. For example, one might "see" a chirp-like waveform in the interferometer output; one might think that a gravity wave from a binary-inspiral has been detected. However, one must be careful, because this signal could also have been induced from the noise background; it may actually come from one of the environmental variables. For instance, the ground could be moving in such a way that, after being transmitted through the mechanical couplings of the seismic stack and the pendulum, it induces a chirp-like motion of the test mass. Hence, one must setup an anti-coincidence between environmental channels and the interferometer output to filter out such false detections. In this regard it is important to characterize each Physics Environmental Monitoring (PEM) channel and indeed all noise sources, to determine the degree to which each channel contains chirp-like (after being referred to the test mass) signals. In short, one would like a benchmark telling the degree of "chirpiness" of each channel. The plots of the noise contribution to the test mass motion from various sources are given in terms of rms power spectrum. Such graphs do not tell the whole story. For example, noise that is very chripy will be harder to find a real binary-inspiral gravity wave signal then noise that is in some sense orthogonal to a chirp; these graphs are opaque to this issue.

To make a "chirpiness" benchmark, we use the same techniques that will be used to detect the gravity wave signals. The techniques must be the same to get a real indication of what level of false detection a given environmental channel will induce. One can characterize each channel by obtaining the number of false detections that would occur assuming that the given channel is the only source of test mass motion. In this chapter, routines and code modifications are discussed for making such chirpiness benchmarks but, of course, similar studies would be useful for ring-down and other sources. Specifically, finding the benchmark for seismic induced noise in the test mass is often mentioned, but again the reasoning applies to any PEM channel.

To be concrete, let's take the case of the seismic channel. First convince oneself that the system is correctly operating from seismometer (or accelerometer) through to GRASP. This can be done, for example, by signal injection. Next, one should obtain the transfer function that takes one from ground motion up through to the in-line motion of the test mass. One only has the output of the seismometer not the ground motion directly so this means dividing by the seismometer's transfer function to get back to ground motion and then multiplying by the transfer function of the seismic stack and the pendulum that the test mass hangs on. Typically, one has experimental numbers and theoretical input that one must use to make a model so he can produce an ASCII file of the format described above. One must pick a frequency of interest and scale the transfer function accordingly; this is only important for transfer functions and variables that have a large dynamic range. One now may want to use make_grid and make_mesh to create an optimal template bank grid; however, these functions have several problems that cause core dumps even if the transfer function is not as steep as that of the seismic stack. Instead, the user may choose to use what seems a reasonable covering of the parameter space; in any case, the benchmark (false events/unit time) obtained with a less than optimal grid will only be smaller than the actual. Using the chosen template bank, one then runs the code. The code inserts the transfer function before the matched filtering bank and then runs the bank across it giving as output SNR and other parameters for each segment. Finally, one analyzes the data to extract a reasonable benchmark in terms of false events per unit time.

1.7 Summary of changes and additions

This section is a summary of the functions that have been changed and added much of it for PEM characterization. All functions have been tested with LINUX and SUN beowulf systems (note dmt frames only exist on SUN platform). GRASP_HOME = GH is the variable set in the SiteSpecific file. The format is directory followed by files changed or added.

- GH
  - setgrasp
  - make_new_tail
  - SiteSpecific
  - libraries added: lsmp.o lsmp_con.o SigFlag.o
- GH/tree
  - animateF_tree
  - GRASP_tree
  - optimalF_tree
  - snapshotF_tree
- GH/src/inspiral/
  - matched.c
- GH/src/utility
  - frameinterface.c
- GH/src/examples/examples_frame/
  - animateF.c
  - snapshotF.c
  - optimalF.c
  - calibrateF.c similar modifications to above: note units will not be correct
  - Makefile.tail
- GH/src/examples/examples_inspiral
  - optimal* series described above
  - ASCII transfer function files
  - setgrasp

- GH/src/examples/examples_template_bank
  - bin_for_hist.c and related scripts for making histograms
  - histogram1-12.c
  - histogram2-12-col.c
  - histogram2-12-hist.c
  - signal_test_multi_files.c
  - multifilter* series described above and Makefile.tail
  - ASCII transfer function files
- GH/src/examples/examples_binary-search/
  - The "binary-search files" list below are not changed in this release. However, files modified in a similar way to the optimal*.c and multifilter*.c series exist; some debugging and testing are not complete so they have not been included. Please contact Dr. A. Rizzi at arizzi@ligo.caltech.edu for more information. In any case, I recommend the mulitifilter package for the PEM operations.
  - binary_get_data.c
  - binary_search.c
  - binary_params.h
  - environment-setup
- GH/parameters
  - The functions listed below are used with make_grid to allow creation of optimized template grid for accelerometer at South end at LLO. They incorporate the seismic stack and the pendulum.
  - detectors.dat.seismic
  - noise_seismic_transFun.dat
- GH/src/examples/examples_template_bank
  - make_grid.seismic.c —contains setup for creating grid appropriate to seismic induced motion of test mass.
- GH/examples/examples_utility
  - translate.c

1.7.1 Acknowledgments:

I would like to thank the many people who have contributed to the effort in modifying GRASP to give the functionality described in this chapter: in particular, Mike Werda, Tom Evans, Dan Fabrycky, John Zweizig, Chethan Parameswariah, Joel Tohline and Mario Diaz.