

LASER INTERFEROMETER GRAVITATIONAL WAVE OBSERVATORY
- LIGO -
CALIFORNIA INSTITUTE OF TECHNOLOGY
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Document Type	LIGO-T040022-04-Z	2004/03/03
Notes for DASWG on MATLAB as a tool for LIGO data analysis		
J. Creighton, L. S. Finn, S. Marka, P. Shawhan		

Distribution of this draft:

LIGO Scientific Collaboration

California Institute of Technology
LIGO Project - MS 51-33
Pasadena CA 91125
Phone (626) 395-2129
Fax (626) 304-9834
E-mail: info@ligo.caltech.edu

Massachusetts Institute of Technology
LIGO Project - MS 20B-145
Cambridge, MA 01239
Phone (617) 253-4824
Fax (617) 253-7014
E-mail: info@ligo.mit.edu

WWW: <http://www.ligo.caltech.edu/>

Abstract

We present an overview of MATLAB and discuss its suitability for use by the LSC for performing complex, possibly computing-intensive data analysis pipelines, in addition to its current widespread use for “high-level” analysis tasks such as calculating statistical summaries and generating plots.

Contents

1 Overview	2
2 Availability	3
3 Potential for Use as an LSC Analysis Environment	4
4 Summary and Recommendation	4
A Example of Wrapping a LAL Function for Use Within MATLAB	6
B Notes on Some MATLAB Quality Engineering Practices	9

1 Overview

MATLAB is simultaneously a high-level programming language, a development environment, and an extensive set of function libraries (called “toolboxes”) of meticulously tested and documented programs for numerical computations covering a wide range of technical disciplines. The language is specially developed for numerical linear algebra, which includes the principal operations required for time-series data analysis. The development environment includes a built-in editor coupled to a source-level debugger, a source-level profiler, and a data inspector. Most of MATLAB is written in MATLAB: *i.e.*, there are a small number of “primitive” operations in the language and the rest of the functionality (including the toolboxes) is provided by programs written in the MATLAB language. Users can perform complex data processing tasks by writing scripts and custom functions to suit their needs.

MATLAB is used extensively in the LIGO Lab and the LSC for many purposes, including detector characterization and mission-critical operations such as generating science run calibrations. It is used in the context of some gravitational-wave searches for trigger “post-processing”: calculating statistical summaries, curve fitting, histogramming, and producing plots (both for personal inspection and for publications).

In their basic form, MATLAB programs are interpreted and run within the interactive MATLAB environment. However, there is a “MATLAB Compiler” product which translates MATLAB programs into C or C++ code, which is then compiled and linked to a freely-distributable run-time library. The generated code consists principally of calls to run-time library functions. There are no license restrictions on the binaries generated by the MATLAB Compiler or on the MATLAB run-time libraries which are required to execute them.

MATLAB includes a reasonably straightforward API (application-programmer interface) that allows it to interface with C, C++ or FORTRAN programs. A new MATLAB function can be created by writing code in one of these languages, then using the “mex” program to compile the code into a relocatable object (called a “MEX-file”) that MATLAB can load dynamically.¹ MEX-file code can call functions in external libraries, enabling MATLAB to use code originally written for programs written in C, C++ or FORTRAN. The new MEX-file function becomes available in the MATLAB interactive environment, or in compiled MATLAB code, exactly like any built-in MATLAB function.

Documentation, including syntax and argument descriptions, for all MATLAB functions may be viewed within the interactive environment simply by typing “help” followed by the function name. It is straightforward to write documentation for user-written MEX-files which is similarly viewable within the interactive environment. The MATLAB product also includes a full-featured help browser, and many books are available.

MATLAB is relied upon extensively in large number of technical industries and scientific laboratories, often for risky and/or mission critical operations. As a commercial product developed and maintained by The MathWorks (www.mathworks.com), it undergoes extensive testing procedures to ensure correctness; see Appendix B for further notes on this topic. It has a well-developed and active user community that interacts well with The MathWorks. MATLAB is also supported by a large number of 3rd party companies, which provide MATLAB consulting services and/or toolboxes for specific industries or application areas.

2 Availability

MATLAB is supported on AIX, Digital Unix, HP-UX 10, HP-UX 11, IRIX/IRIX64, Linux, Mac OS X, Solaris, and MS Windows (XP, 2000, NT, 98 and Millenium Edition). The MATLAB Compiler is available on all these platforms. Support for Windows Millenium Edition and Windows 98 will be dropped in the next major release.²

Individual academic users can purchase the basic MATLAB product, which includes the interactive environment, standard mathematical operations, and the “mex” program, for \$500. LSC members will most likely want to add on a number of toolboxes, which cost \$200 each for individual academic users. Individual needs will vary, but toolboxes which are used by LSC members for data analysis include Signal Processing, Filter Design, Statistics, Curve Fitting, and Wavelets. The MATLAB Compiler is another \$500 if purchased individually. There are quantity discounts for these products, as well as alternative licensing arrangements, which are based on the maximum number of concurrent users rather than the number of computers on which the software is installed. (For instance, a single MATLAB Compiler license may be shared by multiple people at one LSC institution, as long as they don’t try to use it at the same time.) The purchase price includes one year of technical support and free product upgrades; after that, one can purchase Software Maintenance Service (technical support and free upgrades) for a yearly fee which is 20% of the current purchase price. (Without Software Maintenance Service, product upgrades may be obtained at a

¹Note that the “mex” program is included with the basic MATLAB product; the MATLAB Compiler product (which produces standalone programs for use *outside* of the MATLAB environment) is not needed for this purpose.

²For more details see <http://www.mathworks.com/products/matrix/test.shtml> and <http://www.mathworks.com/access/helpdesk/base/relnotes/ch112.shtml>.

discounted price which depends on how much time has elapsed since purchase or since Software Maintenance Service was discontinued, up to a maximum of 4 years.) For full pricing information, visit the MathWorks web site (www.mathworks.com) and click on “store”, then on the “Price list” link.

3 Potential for Use as an LSC Analysis Environment

MATLAB will certainly continue to be used by LSC members for “high-level” analysis tasks such as histogramming and visualization. Additionally, its built-in functionality and extensibility make it attractive as a general analysis environment, capable of carrying out sophisticated analysis pipelines end to end. If suitably structured, a MATLAB-based analysis can be significantly easier to debug and to review than an analysis written in a lower-level language such as C, because of the high level of the MATLAB language, because of the excellent existing documentation, and because the interactive environment allows easy inspection and visualization of intermediate results—acting as a built-in debugger.

The standard mathematical operations and signal processing functions built into MATLAB and its toolboxes may be sufficient to implement some gravitational-wave searches. Others will benefit from using MEX-files to build on the existing base of gravitational-wave data analysis code in LAL and other libraries. For instance, the widely-used “`frextract`” and “`frgetvect`” functions are MEX-files which call functions in the VIRGO Frame library to read data from a frame file into a MATLAB array. See Appendix A for an example of “wrapping” a LAL function so that it can be executed from within MATLAB.

The Penn State group has explored how to take advantage of the MATLAB Compiler to run search code in parallel on their large computing cluster. There is little distinction between compiled and interpreted MATLAB: both use the same run-time library for computations and most MATLAB programs can be converted by the Compiler to stand-alone binaries. Compiled MATLAB does not require a license to run, so any number of instances of a compiled MATLAB program can be run. In this way, an analysis program which has been developed and debugged in the interactive MATLAB environment can be run in parallel on a large amount of data with very little extra effort (other than the bookkeeping required to launch the parallel jobs with appropriate parameters). Experience at PSU over the last 6 months has found no instance of code generation errors or unexpected behavior of compiled code.

4 Summary and Recommendation

The MATLAB environment offers powerful built-in functions and is readily extensible to interface with LAL and other libraries, providing a means for performing a complete gravitational wave search within the MATLAB framework. The high level of the language, available documentation, and built-in inspection and visualization tools ease the burden of writing, debugging, and reviewing analysis software. For searches which require large amounts of CPU time, the MATLAB Compiler allows the code (developed and debugged within the interactive environment) to be converted into stand-alone binaries, which can run in parallel on large computing clusters.

We recommend that the LSC embrace MATLAB as a full-fledged analysis environment and support those who choose to develop and execute gravitational-wave searches in that environment by committing to allow MATLAB-based analyses on a permanent basis.³ Given that many LSC members are already skilled at using MATLAB, this should increase the number of scientists actively and constructively contributing to LIGO data analysis.

Of course, use of MATLAB is subject to the understanding that MATLAB-based analyses must satisfy the LSC requirements for documentation, version control, and software review. In particular, functions (above some complexity threshold) which are of general use for gravitational-wave data analysis generally should reside within LAL, to avoid duplication of effort for writing, maintaining, and reviewing software. Some coordination will be necessary to establish guidelines for constructing MATLAB-based analyses⁴ and to facilitate integration with LAL.

³We do not suggest that LSC members be *required* to purchase or use MATLAB, only that it be an available option.

⁴For one thing, relying on functions from an obscure toolbox may limit the ability of other LSC members to reproduce the analysis. At a minimum, the documentation for an analysis will need to specify what toolbox(es) it requires.

A Example of Wrapping a LAL Function for Use Within MATLAB

It took Peter (who has done this kind of thing before) a little under two hours to write and debug the following code. Most likely, LAL functions will be wrapped on an as-needed basis, though it is not out of the question to envision a sophisticated script which would automate the process of wrapping LAL functions in bulk.

```

/*-----
CreateInspiralBank.c
Matlab MEX-file which calls LALInspiralCreateCoarseBank in a particular way.
Written by Peter Shawhan, 17 Feb 2004
Stole code from lalapps/src/inspiral/tmplbank.c (release 3.0)
-----*/

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "mex.h"
#include <lal/LALConfig.h>
#include <lal/LALStdio.h>
#include <lal/LALStdlib.h>
#include <lal/LALError.h>
#include <lal/LALDatatypes.h>
#include "lal/LALInspiralBank.h"

void mexFunction( int nlhs,          mxArray *plhs[],
                  int nrhs, const mxArray *prhs[])
{
    int debug = 0;
    int badargs = 0;
    double mMin, mMax, mm;
    int ndim, dims[10];
    int i, j;
    double freq;
    double *ptr;

    LALStatus          status;
    InspiralCoarseBankIn bankIn;
    InspiralTemplateList *coarseList = NULL;
    INT4               numCoarse = 0;
    double              *pBankCount;
    double              *pBankMasses;

    /*----- Initialize LAL status structure -----*/
    memset( &status, 0, sizeof(LALStatus) );

    /*----- Check validity of arguments -----*/
    if ( debug ) {
        mexPrintf( "Number of arguments (on right-hand side) = %d\n", nrhs );
        mexPrintf( "Number of outputs (on left-hand side) = %d\n", nlhs );
    }
}

```

```

if ( nrhs != 3 || nlhs != 2 ) {
    badargs = 1;
} else {
    /*-- Make sure all arguments are numeric --*/
    if ( ! mxIsDouble(prhs[0]) ) { badargs = 1; }
    if ( ! mxIsDouble(prhs[1]) ) { badargs = 1; }
    if ( ! mxIsDouble(prhs[2]) ) { badargs = 1; }
}

if ( badargs ) {
    mexErrMsgTxt( "Usage: [n,bank] = LALInspiralCreateCoarseBank(mMin,mMax,mm)\n"
        " mMin = minimum mass of either body\n"
        " mMax = maximum mass of either body\n"
        " mm = requested minimal match\n" );
    return;
}

/*----- Copy arguments to local variables -----*/
ptr = mxGetPr(prhs[0]); mMin = *ptr;
ptr = mxGetPr(prhs[1]); mMax = *ptr;
ptr = mxGetPr(prhs[2]); mm = *ptr;

/*----- Generate a fake PSD for template bank generation -----*/
bankIn.shf.epoch.gpsSeconds = 731000000;
bankIn.shf.epoch.gpsSeconds = 0;
bankIn.shf.deltaF = 1.0;
bankIn.shf.f0 = 0.0;
bankIn.shf.data = NULL;
LALDCreateVector( &status, &(bankIn.shf.data), 4097 );
memset( bankIn.shf.data->data, 0,
    bankIn.shf.data->length*sizeof(COMPLEX8) );
for ( i=1; i<4097; ++i ) {
    freq = (double) i;
    bankIn.shf.data->data[i] = pow(freq,-4.0) + freq ;
}

/*----- Set remaining parameters (many values hard-coded here!) -----*/
bankIn.massRange      = MinMaxComponentMass;
bankIn.mMin           = mMin;
bankIn.mMax           = mMax;
bankIn.MMax           = bankIn.mMax * 2.0;
bankIn.mmCoarse       = mm;
bankIn.mmFine         = 0.99; /* doesn't matter since no fine bank yet */
bankIn.fLower         = 70.0;
bankIn.fUpper         = 2048.0;
bankIn.iflso          = 0; /* currently not implemented */
bankIn.tSampling      = 4096.0;
bankIn.order          = twoPN;
bankIn.approximant    = TaylorT1;
bankIn.space          = Tau0Tau3;
bankIn.etamin         = bankIn.mMin * ( bankIn.MMax - bankIn.mMin ) /
    ( bankIn.MMax * bankIn.MMax );

/*----- Call the LAL function to create the bank -----*/

```

```

LALInspiralCreateCoarseBank( &status, &coarseList, &numCoarse, bankIn );

/*----- Fill Matlab outputs -----*/
ndim = 1;
dims[0] = 1;
plhs[0] = mxCreateNumericArray(ndim,dims,mxINT32_CLASS,mxREAL);
pBankCount = mxGetPr(plhs[0]);
*pBankCount = numCoarse;

plhs[1] = mxCreateDoubleMatrix(numCoarse,2,mxREAL);
pBankMasses = mxGetPr(plhs[1]);
for ( i=0, j=0; i<numCoarse; ++i, j+=2 ) {
    pBankMasses[j] = coarseList[i].params.mass1;
    pBankMasses[j+1] = coarseList[i].params.mass2;
}

/*----- Report number of templates generated -----*/
mexPrintf( "Laid out a bank of %d templates\n", numCoarse );

/*----- Clean up LAL stuff -----*/
LALFree( coarseList );
LALDDestroyVector( &status, &(bankIn.shf.data) );

return;
}

```

Here is how to compile and link this code, producing a dynamically loadable object:

```

mex CreateInspiralBank.c $LIGOTOOLS/lib/liblal.a -I$LIGOTOOLS/include \
    -output CreateInspiralBank

```

And here is how to use it in a MATLAB session:

```

[n,bank] = CreateInspiralBank(1.0,3.0,.97);
Laid out a bank of 2444 templates
plot(bank(:,1),bank(:,2),'.')

```

B Notes on Some MATLAB Quality Engineering Practices

Significant portion of MATLAB is open code, which greatly simplifies the code review and validation. The MATLAB code is also continuously monitored, tested and verified by a dedicated department called "Quality Engineering" (QE) within the MathWorks and a large pool of independent beta testers. The company puts strong emphasis on the reliability of the product as it is extensively used in high risk or mission critical tasks. Besides extensive internal testing and validation, the code goes through several freezes and external beta testing cycles in the development, validation and endgame phases. Our understanding is that both the performance and correctness aspects of the code are closely monitored. The quality control infrastructure includes comprehensive unit tests by the programmer in collaboration with QE. QE also develops, maintains and relies on a thorough automatic test suite for continuous real time code testing; including a complete rebuild for supported platforms in every 24 hours (48 for weekends). Problems are tracked, described and their resolution is documented via a dedicated problem reporting system. Performance and memory issues are evaluated using dedicated resources. Long periods are allocated for code freezes and product testing before each beta and final releases.

Detailed and convincing information on QE practices used to maintain and improve the MATLAB environment is available from MathWorks. However, the company requests that individuals contact them for such details and very strongly discourage the dissemination of documents obtained in such a way.