SparseLab Architecture

David Donoho, Victoria Stodden, and Yaakov Tsaig
Stanford University [1]

Version 2.0
March, 2007

[1] **Acknowledgment of Support.** This work was partially supported by NSF DMS-05-05303 and by other sponsors.

Chapter 1

Introduction

Changes and Enhancements for Release 2.0: 4 papers have been added to Sparselab 2.0: "Fast Solution of l1-norm Minimization Problems When the Solutions May be Sparse"; "Why Simple Shrinkage is Still Relevant For Redundant Representations"; "Stable Recovery of Sparse Overcomplete Representations in the Presence of Noise"; "On the Stability of Basis Pursuit in the Presence of Noise."

This document describes the architecture of SparseLab version 2.0. It is designed for users who already have had day-to-day interaction with the package and now need specific details about the architecture of the package, for example to modify components for their own research.

For an introduction to SparseLab at an elementary level, see *About SparseLab*. This document may be accessed via WWW through the SparseLab Home Page: http://sparselab.stanford.edu.

Before beginning, we mention the main components of the SparseLab package, to standardize terminology. First, there are the basic "system components":

1. *Source.* There is source code, in Matlab, TEX, Perl.

2. *Build.* The source code is assembled into a standard release. The current release is 2.0.

3. *Archives.* Compressed archives of the standard release available for three platforms, Mac, Unix and PC, which users can download and install on their machines.

4. *Web Documents.* A web home page (which can be viewed using any web browser) and a series of postscript and pdf files which explain what

SparseLab is and how to get it. The URL is http://sparselab.stanford.edu.

Next there are the basic "user components" of an installed system:

1. *SparseLab Main Directory.* A subdirectory /Sparselab200 of the `Matlab/work` directory, containing the currently released version of SparseLab software, datasets and documentation.

2. *Papers.* A directory /Sparselab200/Papers/ in /Sparselab200 containing scripts reproducing figures in various articles and technical reports.

3. *Examples.* A directory /Sparselab200/Workouts/ in /Sparselab200 containing pedagogical examples that exercise various aspects of SparseLab.

4. *Solvers.* A directory /Sparselab100/Solvers/ in /Sparselab200 containing the various solver engines of SparseLab.

5. *Documentation.* Both pdf and Postscript files available by WWW access.

6. *Datasets.* The largest are included as separate downloads: Sparselabvers_DataSupplementExtCS and Sparselabvers_DataSupplementStOMP, where vers is replaced by the current SparseLab version. Numerical and image data used to illustrate various aspects of sparse analysis by the scripts and workouts.

The following document describes all these various components from a systems-level point of view. An individual needing to modify SparseLab or add to it would be interested in this information.

Chapter 2

Papers

We briefly describe the contents and architecture of the /Sparselab200/Papers/ subdirectory of SparseLab.

2.1 Script Philosophy

The makeup of /Sparselab200/Papers/ is the whole raison d'être of the SparseLab package. The idea is that, when doing research in a computational science, one works to develop reproducible knowledge about the results of computational experiments. The `/Papers` directory is the end product of such an effort. It makes available to researchers around the world, via the Internet, the computations that produced figures which have been published in hardcopy form as technical reports at Stanford University and in forthcoming journal articles. Other researchers can obtain the Matlab code which generated these figures, and can reproduce the calculations that underly the figures. They can, if they wish, modify the calculations by editing the underlying Matlab code. They can use the algorithms on other datasets. They can try their own favorite methods on the same datasets.

Our idea is that, when doing research, long before we write an article, we prepare ourselves with the thought that *what we do on the computer will ultimately be made available to others, for their inspection, modification, reuse and criticism.* This implies several things. First, that the work product which we are aiming to create will be a subdirectory of SparseLab containing a series of scripts that will generate, from scratch, all the figures of the corresponding article. Second, that our work product is *not* the printed figures that go into the article, but the underlying algorithms and code which generate those figures, and which will be made available to others.

Thus, it is no good to print a hardcopy of a figure that we see on the screen and save that for photocopying into a final version of the paper. Once we are happy with a figure we see on the screen, we must save the code that generated the figure, and then edit the code to make it part of a system that automatically reproduces all the figures of an article.

The philosophy we are adopting can be traced to Jon Claerbout and Martin Karrenbach's article *Electronic Documents Give Reproducible Research New Meaning* (`http://sepwww.stanford.edu`). We especially like a thought of theirs which we paraphrase as follows:

> *A traditionally published article is not the end product of scholarship; it is the advertisement for the scholarship. The working software environment that produced the figures in the article is the actual end product of the scholarship.*

To work in accordance with the philosophy, we must adopt a discipline of how we structure our computational experiments in Matlab. A benefit of this discipline is, hopefully, to avoid the sloppiness and errors that are ubiquitous in computational science.

2.2 Script Architecture

The architecture of the `/Papers` directory is as follows. At present, it contains these subdirectories, recreating figures in published articles:

```
ExtCSDemo    - ‘‘Extensions of Compressed Sensing’’
HDCPNPDDemo  - ‘‘High-Dimensional Centrosymmetric Polytopes with Neighborliness
Dimension’’
MSNVENODemo  - ‘‘Breakdown Point of Model Selection when the Number of Variable
Number of Observations’’
NPSSULEDemo  - ‘‘Neighborly Polytopes and Sparse Solutions of Underdetermined L
NRPSHDDemo   - ‘‘Neighborliness of Randomly-Projected Simplices in High Dimensi
SNSULELPDemo - ‘‘Sparse Nonnegative Solutions of Underdetermined Linear Equatio
Programming’’
StOMPDemo    - ‘‘Sparse Solution of Underdetermined Linear Equations by Stagewi
Matching Pursuit’’
```

These subdirectories have been created following several rules, which should be followed in making future additions.

1. Each article gets one subdirectory of /Sparselab200/Papers/.

2. Each subdirectory contains: (a) the paper itself, (b) a subdirectory housing a demo.

3. The files in a subdirectory have stylized names, so that the name indicates the function of the file.

4. Stylized names are based on a *name* and a *short prefix*. The name should be short but descriptive, for example, `Adapt` for scripts associated with the paper *Adapting to Unknown Smoothness via Wavelet Shrinkage* and the prefix should be a related tag, just two-characters long, for example `ad`.

5. The subdirectory name reflects the name you have chosen, for example `/Sparselab200/Papers/Adapt`.

2.2.1 Demos

The Demo subdirectory contains (a) meta-routines that run the whole figure-generating process, (b) scripts that generate individual figures, (c) datasets the scripts draw on, and (d) specialized tools, not present in SparseLab proper, for generating the figures.

2.2.2 Specialized Tools

There are several tools available in the `Utilities` directory to help you with writing scripts. For example, when creating a display through several `Plot` calls, it is preferable to use SparseLab functions like `LockAxes` and `UnLockAxes` rather than to use the low-level Matlab routine `hold`. See Chapter 6 below.

2.2.3 Scripting Rules

I. One script creates one complete figure, not a series of figures, and not just a subplot of a figure.

II. If several scripts need to work with the same variables – for example, if you want a variable to be created in one script and then used in later scripts – these variables must be made global (see section 4 below).

III. No `pause`'s, `print`'s, of `figure` calls in a script.

IV. As far as possible try to use the tools in the SparseLab `Utilities` directory to perform actions like setting axes.

Inspection of existing scripts will help in following these rules. If you obey these rules, then your scripts should be upwardly compatible with script-running engines making more sophisticated use of the Matlab user interface.

2.2.4 Documenting Individual Figures

Each .m file for an individual figure contains a help header which is displayed in the command window at the time the figure is generated in the plot window. This provides a kind of on-line narrative, or caption. Here is an example from `ExtCSDemo`:

```
% GenFig1 -- ExtCSDemo Figure 1: Error of reconstruction versus
% number of samples for signals with a controlled number of nonzeros.
%
% Data files used: DataL0_20.mat, DataL0_50.mat, DataL0_100.mat
%
% See also: GenDataL0.m
```

2.3 Adding New Scripts

To add new demonstration scripts to /Sparselab200/Papers/, having the same format and effect as `ExtCSDemo`:

1. Decide on a *name* for your demo and a *short prefix* for files implementing your demo. For example, `MyOwnDemo` and `mo`.

2. Create a new subdirectory of /Sparselab200/Papers/. For example, `MyOwn`.

3. Create the following m-files:

   ```
   MyOwnDemo    - starts the Demonstration, invokes Choices
   MyOwnInit    - sets up data structures
   MyOwnFig     - called from Choices
   MyOwnIntro   - help file, explains contents of directories
   ```

   Suggestion: copy the corresponding files in one of the other subdirectories of `/Papers` into your new subdirectory, giving them these names; then edit these files to refer to your own new scripts.

4. Create the scripts which implement your demo: `mofig1.m`, `mofig2.m`, etc. The scripts need to follow thee rules mentioned above in sections 2.2, 2.3 and 2.4.

2.4 Modifying Existing Scripts

You might want to modify an existing script for several reasons:

- To try it out on a different dataset;
- To try it out with different parameters;
- To insert a different method in place of the existing method, using the same dataset.

Our rules for script creation should help make this possible. Some issues to keep in mind:

First, the script that generates a certain figure might be dependent on computations done in the process of generating earlier figures. Therefore, the script cannot be assumed to work correctly in stand-alone mode. If the script refers to any global variables then, at a minimum, the corresponding `Init` script has to be run before the indicated script in order to set global variables up.

Second, in order to generate a certain effect, it might therefore be necessary to change earlier scripts, not just the script formally associated with the figure you are interested in. The change might have to be in the `Init` script (to affect global variables), and might possibly have to be in other scripts as well.

Third, when a set of scripts has been well-written, it should be necessary *only* to change the `Init` script to produce most changes of the type users will want.

Chapter 3

Examples

Here we describe the contents and architecture of the `/Examples` subdirectory of SparseLab. We've included a number of pedagogical examples in SparseLab, so that the user can familiarize himself or herself with the software and with our intentions in providing it. Currently we include:

`nnfEx`	`Non-negative Matrix Factorization`
`reconstrutionEx`	`Signal Reconstruction`
`RegEx`	`Model Selection in Regression`
`TFDecompEx`	`Time Frequency Decomposition`

Each example is documented on the SparseLab website and can be run by running the correspondingly named .m file in each directory.

3.1 Examples Philosophy

`/Examples` is a subdirectory of /Sparselab200 that is much like `Papers` in that it contains a variety of subdirectories, each of which contains a sequence of scripts generating figures. However, `Examples` is different in that its primary motivation is *not* to reproduce figures in our own articles. Instead, its motivation is for more informal, exploratory purposes

3.2 Existing Examples

In the current release, version 2.0, we distribute the following Examples:

`/nnfEx`	`Non-negative Matrix Factorization`

`/reconstrutionEx`	`Signal Reconstruction`
`/RegEx`	`Model Selection in Regression`
`/TFDecompEx`	`Time Frequency Decomposition`

3.3 Examples Architecture

It is a good idea to follow the same naming practices and file organization as in the directory /Sparselab200/Papers/.

3.3.1 Naming

In the Regression example, we use the filenames `RegEx01.m`, `RegEx02.m` after the name of the example directory for the main script. We try to number figures in an obvious way and to stick with names no longer than eight characters.

3.3.2 Script Contents

Each file should generate one figure, and should avoid the use of `clg`, `figure`, `print` and `pause`. This is the same set of rules that we adhere to in /Sparselab200/Papers/.

3.3.3 Meta Routines

By following the above rules it is easy to write wrapper code to print all figures or to cycle through all figures. Such wrapper code typically has suggestive names like `BBPrintAllFigs` or `BBShowAllFigs`.

Chapter 4

Datasets

The scripts we have just discussed make use of several datasets, which are made available in the directory /Sparselab200/DataSets/. In this chapter we describe the architecture of our dataset library.

4.1 Dataset Philosophy

We make available datasets through *centralized readers*. The idea is that the knowledge of how to access a dataset should be concentrated in a single place, and that the access to any dataset should be made in a stereotyped way, through a simple function call, not through explicit input and output routines.

In this way, if a dataset is available in the system because it has been used for one script, it automatically becomes available throughout the system for any other purpose one would wish, without others needing to know the format or location of the data.

If, in the future, the dataset needs to be moved to some other location in the file system, or if it needs to be stored in some other format, no scripts that use the data for demonstrations will need to change. Instead, one changes only the code implementing the access method rather than the scripts which want to use the dataset.

(The alternative is, of course, that any such changes in the future require rewriting all existing scripts!)

The same philosophy applies for datasets which are synthetic – those created by Matlab formulas. They are accessed in a stereotyped way through access to a *centralized synthesizer*. In this way, a synthetic signal designed for one use in one script automatically becomes available for other purposes.

4.2 Dataset Directory

The Contents.m file in the `Datasets` directory contains the following information. It shows that there are several tools for accessing data, 1-d datasets and 2-d datasets.

It is possible that at some time in the future, we will also have 3-d datasets (probably movies) or collections of still images.

```
%          Data Fabricators
%
%   MakeBlocks        -   Make artificial blocky signal
%   MakeBumps         -   Make artificial bump signal
%   MakeMatrix        -   Make artificial random matrix
%
%
```

4.3 Dataset Documentation

Each dataset in the system has a documentation file, with suffix .doc. Here is an example of a documentation file for a 1-d signal:

```
caruso.asc -- Digital signal of Caruso singing

Access
    Enrico = ReadSignal('Caruso');

Size
    50,000 by 1

Sampling Rate
    8192 Hz

Description
    In MATLAB, the command sound(Enrico,8192) will play this sound
    back at the right pitch.

Source
    Obtained by anonymous FTP from the xwplw package
    developed by R.R. Coifman and Fazal Majid at Yale University.
    You can get this X-windows adapted waveform analysis
```

```
    package by anonymous FTP to math.yale.edu.
```

Here is an example of a documentation file for a 2-d image:

```
canaletto.raw -- Gray-scale image of Canaletto painting

Access
    Canal = ReadImage('Canaletto');

Size
    512 by 512

Gray Levels
    256

Description
    This image was used in an article by P. Perona and J. Malik,
    "Scale-Space Filtering by Anisotropic Diffusions," IEE PAMI.

Source
    Obtained from John Canny and Jitendra Malik, of EECS at
    U.C. Berkeley.
```

You will notice the following fields in the documentation:

1. *Title.* A one-line header at the start of the file, giving the filename, and, after two hyphens, descriptive text.
2. *Access.* A code fragment indicating the stereotyped access method.
3. *Size.* The size of the signal or image.
4. *Gray Levels.* Applicable for Images only.
5. *Sampling Rate.* Applicable for Sounds only.
6. *Source.* Indication of the original source of the data.
7. *Description.* Additional description of the data.

4.4 Adding New Datasets

To add new datasets to SparseLab, do the following:

1. *Installation.* Place the dataset, in stereotyped format, in the `Datasets` directory. Modify one of the existing access functions to read in the dataset. (You can, in a pinch, place the dataset elsewhere, or keep it in a nonstandard format).
2. *Documentation.* Insert a .doc file in the `Datasets` directory to explain the dataset.

To add a new synthetic matrix type to SparseLab, simply modify the function `MakeMatrix`, by inserting a new case in the "compound if"; the new case tests for a new, previously unused name, and contains a formula defining the signal in that case. Add a separate function, similar to `UniformSphericalMatrix` for example, with the build instructions. It is best if the formula is designed to work the same way the other formulas work – to produce an output at any given signal length or image extent.

4.5 Dataset Sources

We would like to take this opportunity to thank the sources of our datasets. We reprint here from the file `THANKS.m` in /Sparselab200/Documentation/.

```
%  Contributors of Data
%       Yaakov Tsaig
```

Chapter 5

Documentation

There has been extensive concern for the documentation of the code in SparseLab. We try to use all the features of Matlab as well as other features to produce a coherent, understandable system.

5.1 Help Headers

Each function in the SparseLab system has documentation contained inside the .m file with its Matlab code. This documentation can be accessed on-line by typing `help Name` where `Name` is the name of the function. For example, typing `help SolveMP` gives:

```
% SolveMP: Matching Pursuit (non-orthogonal)
% Usage
%   [sol iters activationHist] = SolveMP(A, b, maxIters, NoiseLevel, verbose)
% Input
%   A           dictionary (dxn matrix), rank(A) = min(d,n) by assumption
%   y           data vector, length d.
%   maxIters    number of atoms in the decomposition
%   NoiseLevel  estimated norm of noise, default noiseless, i.e. 1e-5
%   verbose     1 to print out detailed progress at each iteration, 0 for
%               no output (default)
% Outputs
%    sol             solution of MP
%    iters           number of iterations performed
%    activationHist  Array of indices showing elements entering
%                    the solution set
```

```
% Description
%   SolveMP implements the greedy pursuit algorithm to estimate the
%   solution of the sparse approximation problem
%      min ||x||_0 s.t. A*x = y
% See Also
%   SolveOMP
% References
%   Matching Pursuit With Time-Frequency Dictionaries (1993) Mallat, Zhang
%   IEEE Transactions on Signal Processing
%
```

This illustrates the main components of the format we have adopted: a one-line *help header*, and sections for *Usage*, *Inputs*, *Outputs*, *Side Effects*, *Description*, *Examples*, *Algorithm*, *See Also* and *References*.

1. *Header*. The first line of the help header is called the `H1` line by the Matlab folks. It is special to Matlab, and to SparseLab. When you use the `lookfor` command, Matlab examines this line for each .m file in its path to find text matching the request. When a release of SparseLab is built, these lines are compiled and sorted in alphabetical order to make files in the documentation directory. Format: a percent sign, a single blank, the name of the function, a blank followed by double hyphens and a blank, and a short description of the function. The description should contain as many helpful keywords as possible.

2. *Usage*. Here, indicate the calling prototype. Format: the output argument(s) (enclosed within square brackets if there is more than one output argument), an equals sign, the function name followed by the input argument(s) enclosed within parentheses. Optional input arguments are enclosed within square brackets.

3. *Inputs*. Here, one line per input variable, indicating the name of the variable, the formal data type and the interpretation. Also, indicate if the input is optional by enclosing it within square brackets.

4. *Outputs*. Here, one line per output variable, indicating the name of the variable, the formal data type and the interpretation.

5. *Side Effects*. Here, indicate any side effects the routine may have (graphics, sound, etc.). Omit if the function has no side effects.

6. *Description*. Here, describe what the function does in as much detail as possible.

7. *Examples.* Here, list examples of how the function is called in practice. This field is optional.

8. *Algorithm.* Here, describe the algorithm used by the function. This field is optional.

9. *See Also.* Here, mention other routines which this routine calls or which call this one, or routines with a special relationship to this function. This field is optional.

10. *References.* Here, list references from which the user may obtain further information about the function. This field is optional.

5.2 Documentation Directory

The directory /Sparselab200/Documentation/ contains a variety of information about SparseLab. There are a number of general files, which describe various terms and conditions and goals. The contents of any of these files may be examined by typing its name.

```
% ADDINGNEWFEATURES       -  How to Add New Features to SparseLab
% BUGREPORT               -  How to report bugs about SparseLab
% COPYING                 -  SparseLab Copying Permissions
% DATASTRUCTURES          -  Basic data structures in SparseLab
% FEEDBACK                -  Give feedback about SparseLab
% GETTINGSTARTED          -  Ideas for getting started with SparseLab
% INSTALLATION            -  Installation of SparseLab
% LIMITATIONS             -  SparseLab known limitations
% PAYMENT                 -  No Charge for SparseLab Software
% REGISTRATION            -  SparseLab Registration
% SUPPORT                 -  SparseLab Support
% THANKS                  -  Thanks to contributors
% VERSION                 -  Part of SparseLab Version v$VERSION$
% WARRANTY                -  No Warranty on SparseLab software
```

To add or modify the first group of files, very little is required. Simply add new files. The second group of files, being automatically generated at build time, should not ordinarily be modified. Instead, modify the source from which they are automatically compiled.

Because of the automatic build process, it is important to maintain the integrity of certain files. These include:

- Contents files. Every directory should have a Contents.m file. When adding a new function to a directory, be sure to add it to the directory's Contents file as well.

- H1 Lines of Help documents. Every .m file should contain a help header, and the `H1` line of the help header should follow the rules specified above.

- $VERSION$ marker. Every Contents.m file has, in the `H1` line, a description of what the directory contains, as well as a version marker. The text $VERSION$ is replaced, automatically upon build, by the current version number.

5.3 Examples Directory

Another useful component of the system documentation is the `/Examples` directory, which contains scripts that exercise the software in various ways.

The user can look through the graphics generated by this documentation and, upon seeing something interesting, inspect the corresponding script to see how the graphic was created.

Currently, the `/Workouts` directory contains three subdirectories:

```
nnfEx            Non-negative Matrix Factorization
reconstrutionEx  Signal Reconstruction
RegEx            Model Selection in Regression
TFDecompEx       Time Frequency Decomposition
```

5.4 TEX Documents

The system also comes with several documents, written in TEX, which function as manuals for system-maintenance people.

The file `SparseMacros.tex` within `SparseLab Documentation` contains macros that define the current version of SparseLab, filenames, file sizes, file locations, etc. This file should be modified appropriately for new releases of SparseLab. It is included by all the documents described below.

5.4.1 About SparseLab

About SparseLab helps a new user with installing and getting started with SparseLab. The corresponding pdf and postscript documents are available

at:
`http://sparselab.stanford.edu`. The source is written in LaTeX. It is contained within the `About SparseLab` folder `Documentation`.

5.4.2 Architecture

You are currently reading the *SparseLab Architecture* document. It contains system-level information about the SparseLab distribution. The corresponding postscript document is available via `http://sparselab.stanford.edu`. The source is written in LaTeX. It is contained within the `SparseLab Architecture` folder in `SparseLab Documentation`.

Chapter 6

Utilities

Several utilities are available in SparseLab mainly for the purpose of centralizing various programming idioms. If SparseLab is ever to be ported to Octave, for example, these allow one to modify only the utilities to the new platform and achieve the desired effect of platform-independent scripts.

The current Contents.m file for /Sparselab200/Utilities/ goes as follows:

```
%   Contents.m            -   This file
%   aconv.m               -   Convolution Tool for Two-Scale Transform
%   AutoImage.m           -   Automatic Scaling for Image Display
%   DownDyadHi.m          -   Hi-Pass Downsampling operator (periodized)
%   DownDyadLo.m          -   Lo-Pass Downsampling operator (periodized)
%   dyad.m                -   Index entire j-th dyad of 1-d wavelet xform
%   dyadlength.m          -   Find length and dyadic length of array
%   FWT_PO.m              -   Forward Wavelet Transform (periodized, orthogonal)
%   FWT_TI.m              -   translation invariant forward wavelet transform
%   iconv.m               -   Convolution Tool for Two-Scale Transform
%   IWT_PO.m              -   Inverse Wavelet Transform (periodized, orthogonal)
%   IWT_TI.m              -   translation invariant forward wavelet transform
%   LockAxes.m            -   Version-independent axis command
%   lshift.m              -   Circular left shift of 1-d signal
%   MakeONFilter.m        -   Generate Orthonormal QMF Filter for Wavelet Transform
%   MirrorFilt.m          -   Apply (-1)^t modulation
%   Noisemaker.m          -   Add Noise to Signal
%   NormNoise.m           -   Estimates noise level, Normalize signal to noise level 1
%   packet.m              -   Packet table indexing
%   PlotSpikes.m          -   Plot 1-d signal as baseline with series of spikes
%   PlotWaveCoeff.m       -   Spike-plot display of wavelet coefficients
```

```
%   RegisterPlot.m      -   Add legend with file name, date, flag
%   reverse.m           -   Reverse order of elements in 1-d signal
%   rshift.m            -   Circular right shift of 1-d signal
%   ShapeAsRow.m        -   Reshape 1d vector as row
%   ShapeLike.m         -   Reshape first argument like second argument
%   TIDenoise.m         -   Translation invariant denoising of a 1-D signal
%   twonorm.m           -   Computes ||v||_2
%   UnlockAxes.m        -   Version-independent axis command
%   UpDyadHi.m          -   Hi-Pass Upsampling operator; periodized
%   UpDyadLo.m          -   Lo-Pass Upsampling operator; periodized
%   UpSample.m          -   Upsampling operator
```

The functions of these utilities can be loosely classified into the categories: *Graphics*, *Random Numbers*, *Shaping Arrays*, and *Scripting*.

Chapter 7

Source and Build

This chapter describes how SparseLab source is compiled into archives for distribution.

7.1 Development System

1 The source for SparseLab development has several components in different directories:

2 *TeX Source* in a directory named `Documentation` inside the `SparseLab100}` `folder.`

3 *Shell Source* `in a directory named shell_tools inside the SparseLab100}` `folder.`

7.2 Shell Tools

Here is an up-to-date list of the high-level files:

```
append_footer.sh        -   Appends a footer to all non-Contents .m files
SparseLab_Footer.txt    -   the footer that gets appended
```

These can be used outside of the Master build process.

7.3 Standard Release

The process of building a "standard" release involves:

1. Appending copyright notices and date-of-modification information to all files in the library;

2. Adding the Matlab Source files to a zip file. This .zip file is the final version that will be made available on the WWW sites.

7.4 Compiling .ps

The `Documentation` directory within `SparseLab_Master` contains one folder for each of the SparseLab documents: *About SparseLab* and *SparseLab Architecture*. These folders contain the LaTeX code for the documents, which are compiled into .ps and .pdf files. These .ps and .pdf files are then made available on the WWW sites.

7.5 Distribution

The uniform download process is chosen for Sparselab. By uniform download process we mean that all the users, independent of the platforms they are using, download the file SparseLabvers.zip in which vers is replaced by the version of the Sparselab. Then during the installation process Sparselab will recognize their platform. Therefore, there is no need for releasing different files for different platforms anymore. The only thing that the distributor should do is just putting the SparseLabvers.zip on the WWW site.

Because of the size of some of the precomputed data files included in SparseLab, two separate download packages, SparseLabvers_DataSupplementExtCS.zip and Sparselabvers_DataSupplementStOMP.zip (where vers is replaces by the appropriate version number) are created for download. Optimally, all three .zip files should be downloaded and installed together.

Chapter 8

Distribution and Maintenance

This chapter describes how SparseLab is distributed and maintained.

8.1 Archive Directory

The `Archive` directory within `SparseLab` is a depository for old versions of the software and documentation.

8.2 SparseLab Account

An account named `SparseLab` is maintained on the leland system at stanford.edu, as is the website. The account serves several varied purposes:

1. The sub-directory `WWW` holds the files used to maintain our Web page.

3. The current version of SparseLab is always present on this account in the sub-directory `SparseLab`.

4. Feedback – questions, comments, suggestions, etc. – may be sent to the development team by e-mailing sparselab@stanford.edu.

8.3 Web Page

The URL of the SparseLab WWW page is http://sparselab.stanford.edu. The html files for the home page are stored in the Documentation subdirec-

tory of SparseLab100/. The home page is constantly changing and evolving. New versions and updates are always announced on the home page.