linboxing

Kernel-level access to LinBox exact linear algebra routines

Version 0.5.1

23 April 2008

Paul Smith

Paul Smith
— Email: paul.smith@nuigalway.ie
— Homepage: http://www.maths.nuigalway.ie/~pas
— Address: Department of Mathematics,
National University of Ireland, Galway
Galway,
Ireland.

Copyright

Acknowledgements

The linboxing package is supported by a Marie Curie Transfer of Knowledge grant based at the Department of Mathematics, NUI Galway (MTKD-CT-2006-042685)

Contents

Chapter 1

Introduction

The LinBox C++ library (http://www.linalg.org) performs exact linear algebra and provides a set of routines for the solution of linear algebra problems such as rank, determinant, and the solution of linear systems. It provides representations for both sparse and dense matrices over integers and finite fields. It has a particular emphasis on black-box matrix methods (which are very efficient over sparse matrices), but increasingly also provides elimination-based routines for dense matrices using the industry-standard BLAS numeric routines.

GAP (http://www.gap-system.org) is a system for computational discrete algebra, with particular emphasis on Computational Group Theory. It provides good implementations of exact linear algebra routines on dense matrices over all common fields and the integers. Typically, GAP's versions are faster than LinBox for small finite fields (i.e. order less than 256), but LinBox is much faster for larger finite fields and the integers.

The linboxing (LinBox-in-GAP) package provides an interface to the LinBox C++ library from GAP. It provides alternative versions of GAP linear algebra routines which are mapped through to the equivalent LinBox library routines at the GAP kernel level. The result is linear algebra routines in GAP that are, in the majority of cases, considerably faster than the native GAP versions, and which scale better with matrix size. See the linboxing website at http://www.maths.nuigalway.ie/~pas/CHA/linboxing/index.shtml for timing examples and speedup estimates. As is typical, this speed is at the expense of memory, since the GAP matrices and vector must be copied into a memory format that LinBox can use.

The functions provided by the linboxing package are named the same as the GAP equivalents, but are all contained within the LinBox record, and so are prefixed with 'LinBox.'. The functions provided are

- LinBox.Determinant (3.1.1)
- LinBox.Rank (3.1.2)
- LinBox.Trace (3.1.3)
- LinBox.SolutionMat (3.1.4)

over the integers and prime fields.

Chapter 2

Installation and Use

Before you can use the linboxing package in GAP, there are several things that you must do. You must install a compatible version of the LinBox library (Section 2.1), and you must build the linboxing package's kernel module (Section 2.2). Finally, you will most likely want to run GAP with special command-line parameters (Section 2.3). This chapter covers all of these technical details.

2.1 Installing the LinBox library

Before you can install the linboxing package, you need to have built and installed the LinBox library on your machine. At least version 1.1.5 of LinBox is required to use the linboxing package.

2.1.1 Downloading LinBox

LinBox can be downloaded as source code (a `.tar.gz` archive) from the LinBox website at `http://www.linalg.org/download.html`. The linboxing package supports version 1.1.5 of LinBox (released 3 April 2008), and newer versions.

The following sections give a brief summary of how to build and install the LinBox library. Full installation instructions come with the downloaded LinBox source archive, or are available from `http://www.linalg.org/linbox-html/install-dist.html`.

2.1.2 Before building LinBox

Before you can build LinBox library you will need the standard tools for building a package from source code (including a C++ compiler such as `g++`). The LinBox library itself requires that you already have installed two further libraries:

- the GNU Multiprecision Arithmetic (GMP) Library (see `http://gmplib.org/`)
- a library providing the BLAS linear algebra routines (see `http://www.netlib.org/blas/`)

These libraries may already be installed on your system or be available from your standard package repositories, so you will not usually need to build these from source code (although if you are interested in performance, you should build your own BLAS library, for example using ATLAS `http://math-atlas.sourceforge.net/`). If you are unsure whether or not you have either the GMP or BLAS libraries installed, the `configure` scripts for both LinBox and linboxing check for them in the standard locations and will tell you if they can't find them.

Further details about setting up these prerequisites is available in LinBox's own installation instructions. None of the optional packages that LinBox can also use (such as NTL or Givaro) are required for the linboxing package.

2.1.3 Building the LinBox library

After downloading the LinBox source archive, it can be extracted using

```
tar -xzf linbox-x.x.x.tar.gz
```

which will create a new directory called `linbox-x.x.x` (where `x.x.x` is the current version number).

Then change to the `linbox-x.x.x` directory and type

```
./configure
make
sudo make install
```

The last command runs `make install` with administrator privileges to install the LinBox library in the default location, `/usr/local/`.

If the GMP and BLAS libraries (see Section 2.1.2) are not in standard locations, or you do not wish to install LinBox in `/usr/local/` (or you cannot), then please refer to the LinBox installation instructions (`http://www.linalg.org/linbox-html/install-dist.html`) for the necessary `configure` options.

2.2 Installing the linboxing package

To install the linboxing package, you should first unpack the linboxing archive file in a directory in the `pkg` hierarchy of your version of GAP. For example, for a package with the extension `.tar.gz`, type

```
tar -xzf linboxing-0.5.1.tar.gz
```

This will extract all of the files into a directory called `linboxing-0.5.1`.

All of the useful functionality of the linboxing package is provided through a compiled GAP kernel module that uses the LinBox library directly. Change to the `linboxing-0.5.1` directory and build the kernel module using the commands

```
cd linboxing-0.5.1
./configure
make
```

The `configure` script runs lots of checks and will search for the locations of all of the required other packages, such as GMP, BLAS, LinBox and GAP itself. If there are any problems, it should report them, and if not then `make` should proceed with no errors. Note that `make install` is not required for linboxing: `make` does all that is needed.

If the required packages are not in the standard locations, you can tell `configure` where they are using the following command-line switches:

--with-blas=<lib> specify the name of the BLAS library, or the linker flags needed to use it

--with-gmpprefix=<prefix> specify the prefix to which GMP library is installed

--with-linboxprefix=<prefix> specify the prefix to which the LinBox library is installed

--with-gaproot=<path> specify the path to GAP's root directory

For example, you may need to use these switches in the following common case. If you do not have root access, you may have installed the LinBox library in your home directory at `/home/pas/software/`. To do this, you will have configured the LinBox build process using `--prefix=/home/pas/software` and when you did `make install`, it would have copied the LinBox library and header files into `/home/pas/software/include` and `/home/pas/software/lib` respectively. You now wish to build this linboxing package. To tell it where to find the LinBox library, you run `configure` with the same prefix that you gave to LinBox, i.e. `--with-linboxprefix=/home/pas/software`.

2.3 Starting GAP with LinBox-friendly memory management

GAP and the LinBox library use different methods for allocating memory, and these do not work well together. GAP needs all of its memory to be contiguous, and so needs to have free space at the end of its current allocation if it ever wants to expand its workspace. The LinBox library allocates its memory using `malloc`, and allocates memory wherever it feels like it. Because of this, if you run GAP and use the linboxing package, then there is a good chance that when GAP needs more memory it will find that some LinBox-allocated memory gets in the way of it expanding the workspace. In this case, GAP will simply exit (without warning!) with the error `cannot extend the workspace any more`.

There are two current solutions to this problem, both of which require GAP to be run with a command-line switch:

Pre-allocate some `malloc` memory for LinBox to use The `-a` command-line option (**Reference: Advanced Features of GAP**) tells GAP to pre-allocate some memory that LinBox should, on most systems, use in preference to getting in the way of the GAP workspace. If you set this sufficiently large (i.1. larger than the largest amount of LinBox memory than you are likely to need at one time), then GAP should be able to expand its workspace as much as it likes. For example, to allocate 50Mb of memory to LinBox (enough for 100,000 small integer matrix elements), use

```
gap -a 50M
```

Allocate GAP a big enough workspace that it will not need extending The `-m` command-line option (**Reference: Command Line Options**) tells GAP to allocate a set number of bytes for the GAP workspace when it starts up. If you set this sufficiently large then GAP will never need to expand its workspace and LinBox can allocate its matrices wherever it likes in the remaining memory. For example, to allocate 256Mb of memory to GAP, use

```
gap -m 256M
```

If you are unsure as to how much memory you might need, refer to (**Reference: Global Memory Information**) for various GAP commands to let you see how much memory your GAP workspace is using. Running GAP with the `-g` (or `-g -g`) command-line switch (**Reference: Command Line Options**) can also help you keep track of memory usage.

You can use both of these solutions at the same time, which may be a safe 'belt and braces' approach. If you intend to regularly use the linboxing package, you can add these options to the `gap.sh` shell script, if you are using it. Future versions of GAP may modify GASMAN storage manager to allow the happy co-existance of GAP memory with `malloc`, which would mean that these switches may eventually not be needed.

2.4 Loading and testing the linboxing package

The linboxing package is not loaded by default when GAP is started. To load the package, type the following at the GAP prompt:

Example

```
gap> LoadPackage( "linboxing");
```

If linboxing isn't already in memory then it is loaded and the author information is displayed. If you are a frequent user of the linboxing package, you might consider putting this line in your `.gaprc` file.

You can test the installation of the linboxing package by running the GAP command `TestLinboxing` (3.2.3):

Example

```
gap> TestLinboxing();
```

2.5 Recompiling this documentation

This documentation is written using the GAPDoc package, and should be available in PDF, HTML and text formats. It should not normally be necessary to rebuild the documentation (if you are reading this!). However, rebuilding the documentation can be done from within GAP when running on a standard UNIX installation by using the GAP command `MakeLinboxingDoc` (3.2.2).

Chapter 3

Function Reference

3.1 Replacements for GAP functions

3.1.1 LinBox.Determinant

◊ `LinBox.Determinant(M)` (function)
◊ `LinBox.DeterminantMat(M)` (function)
◊ `LinBox.DeterminantIntMat(M)` (function)

Returns the determinant of the square matrix `M`. The entries of `M` must be integers or over a prime field.

Example

```
gap> LinBox.DeterminantMat([[1,2,3],[4,5,6],[7,8,9]]);
0
```

3.1.2 LinBox.Rank

◊ `LinBox.Rank(M)` (function)
◊ `LinBox.RankMat(M)` (function)

Returns the maximal number of linearly-independent rows of the matrix `M`. The entries of `M` must be integers or over a prime field.

Example

```
gap> LinBox.RankMat([[1,2,3],[4,5,6],[7,8,9]]);
2
```

3.1.3 LinBox.Trace

◊ `LinBox.Trace(M)` (function)
◊ `LinBox.TraceMat(M)` (function)

For a square matrix `M`, returns the sum of the diagonal entries. The entries of `M` must be integers or over a prime field. Note that this version (unlike the others in this package) is typically slower than the GAP equivalent, but is provided for completeness.

Example
```
gap> LinBox.TraceMat([[1,2,3],[4,5,6],[7,8,9]]);
15
```

3.1.4 LinBox.SolutionMat

◊ LinBox.SolutionMat(*M*, *b*) (function)

Returns a row vector *x* that is a solution of the equation $xM = b$, or `fail` if no solution exists. If the system is consistent, a random solution *x* is returned. The entries of *M* must be integers or over a prime field, but if they are integers then the solution may include rationals.

Example
```
gap> LinBox.SolutionMat([[1,2,3],[4,5,6],[7,8,9]], [2,1,0]);
[ -1, -1, 1 ]
gap> LinBox.SolutionMat([[1,2,3],[4,5,6],[7,8,9]], [2,1,0]);
[ -5/4, -1/2, 3/4 ]
```

3.2 Miscellaneous functions

3.2.1 LinBox.SetMessages

◊ LinBox.SetMessages(*on*) (function)

Turns on or off the printing of the LinBox library commentator messages. If the boolean argument *on* is set to `true` then messages will be printed, else no messages will be displayed.

3.2.2 MakeLinboxingDoc

◊ MakeLinboxingDoc(*[make-internal]*) (function)

Builds this documentation from the linboxing package source files. This should not normally need doing: the current documentation is built and included with the package release.

If the optional boolean argument *make-internal* is `true` then the internal (undocumented!) functions are included in this manual.

3.2.3 TestLinboxing

◊ TestLinboxing(*[num-tests]*) (function)

Test the installation of linboxing and print profiling information. This tries all the linboxing package's algorithms with random vectors and matrices over random fields (covering all the distinct supported field types). The results are compared with the equivalent GAP functions, and the relative times displayed.

The optional argument *num-tests* specifies how many times to run each test: the default is 5 times.

Chapter 4

Implementation

The linboxing package consists three parts. The first part is written in GAP, and this consists of test routines and wrappers for functions in the linboxing kernel module. The second part is the kernel module's interface to GAP, which is written in C. This handles the interface between GAP and the third part, which is the C++ code which calls functions in the LinBox library.

In the C++ part of the kernel module, the GAP objects such as vectors, matrices and their elements are converted into the corresponding LinBox data types. The requested LinBox function is then called, and the result converted back onto GAP objects.

Currently, all GAP matrices are converted into dense matrices in the LinBox library. LinBox provides good support for sparse matrices, but at present there is no standard way in GAP to represent sparse matrices. There are plans to overhaul matrix objects in GAP, and once sparse matrix objects are provided in GAP, these should be converted into LinBox sparse matrices.

For more details of the implementation, please refer to the source code documentation. The C and C++ source code can be found in the `src` directory of the linboxing package, and contains comments which can be converted into HTML documentation using `doxygen` (which must therefore be available on your system). Create this documentation using the following command:

```
cd src
doxygen Doxyfile
```

Point your web browser at `src/html/index.html` to browse the documentation.

Index