LAPACK++ v. 1.1 Release Notes and Installation Manual

Roldan Pozo

April 1996

Contents

1 About LAPACK++

LAPACK++ is a software package for high performance linear algebra computations in C++ across various computer architectures. The design scope of LAPACK++ includes direct solvers for linear equations, linear least-squares, and various matrix factorizations for dense and banded systems.

LAPACK++ is the product of several ongoing research projects: See [6] for a complete design overview of LAPACK++ and object-oriented issues, and [3], [5] for issues of LAPACK++ on distributed memory architectures.

The framework of LAPACK++ is based on the LAPACK [1] library. Both LAPACK and LAPACK++ are freely available in source code form from the `http://www.netlib.org`. LAPACK++ also provides C++ interface to the Basic Linear Algebra Subprograms (BLAS++); this ensures portability and performance across a wide range of computational platforms, including shared and distributed memory parallel systems.

1.1 What's new

- Version v. 1.1

  The last few years have seen a tremendous change in the C++ language as well as supporting compilers. This version of Lapack++ fixes several bugs and includes modifications for changes in the C++ language and compilers since the inital release of Lapack++ in 1992. Due to the transitory nature of complex number support among various compilers, we have disabled complex support in this release. As C++ compilers move towards the templated ANSI C++ representation, this situation will improve.

  No new features are introduced in version 1.1, nor has the basic functionality been changed. New developments and ongoing efforts for such codes have shifted to the Template Numerical Toolkit (TNT) for Linear Algebra project (`http://math.nist.gov/tnt`), based on the C++ Standard Template Library (STL).

- What's new in v. 1.0

  - No longer depends on Fortran 77 – can use C-LAPACK
  - Eigenvalue examples
  - basic support for `complex`
  - support for multiple platforms:
    - RS/6000, (g++ 2.7.0+[1])
    - Sun (CC and g++ 2.7.0+),
    - MS-DOS (Matrix classes only: Turbo C++ v. 3.0, in Windows 3.1 and MS-DOS 6.0)
  - A=B has copy (rather than .inject()) semantics
  - various improvements and fixes
    - shorter filenames to avoid name truncation in linkers and portability to MS-DOS systems, e.g. `#include LA_GEN_MAT_DOUBLE_H` expands to `#include "gmd.h"`.
    - `Index` class name changed to `LaIndex` to match documentation
    - `LaGenMatDouble::shallow_` flag fixed
    - `::addr()` method returns type* rather than `int`
    - `Int` vectors and matrices changed to `LongInt` to integrate with `INT*4` f2c declarations.

[1] Beware that some versions of g++ may be unstable. In particular, g++ 2.3.3 will seg fault during compilation of some source codes.

2 Included in this release

LAPACK++ supports various matrix classes for vectors, non-symmetric matrices, SPD matrices, symmetric matrices, banded, triangular, and tridiagonal matrices; however, This version does not include all of the capabilities of original f77 LAPACK. Emphasis is given to routines for solving linear systems consisting of non-symmetric matrices (using LU factorizations), symmetric positive definite systems (using Cholesky factorizations) and solving linear least-square systems (using QR factorizations). Support for generalized eigenvalue problems and singular value decompositions, although straightforward to add, are not included in this release.

3 Obtaining and installing LAPACK++

3.1 Installation in a nutshell

1. download the LAPACK++ source from `http://www.netlib.org/c++/lapack++` or `http://math.nist.gov/pozo/lap`
2. edit the `makefile.def` file to specify the home directory of LAPACK++
3. edit the appropriate machine-dependent `makefile`XXX`.def` (XXX = `SUN4`, `RS6K`, etc.) file to describe locations of system-specific libraries and environment options. This includes
   - location LAPACK and BLAS libraries
   - whether or not your system has `ranlib`
   - where to put the resulting LAPACK++ libraries
4. type `make libs`, for creating LAPACK++, Matrix++, and BLAS++ libraries, `make test` for run LAPACK++ tests, or `make all` to perform both.
5. type `make clean` to remove `*.o` and extraneous files.

3.2 Testing

There are various LAPACK++ test modules to the matrix classes, and the linear solvers.

- `matrix_testing`: tests C++ matrix and vector classes
- `blas++_testing`: tests C++ interface to the BLAS
- `lapack++_testing`: tests linear equation drivers

3.3 Overview

You will need three components to successfully install LAPACK++:

- the LAPACK++ v. 1.0 C++ source code
- the C or f77 LAPACK library, `lapack.a`
- Basic Linear Algebra Subprograms (BLAS) library, `blas.a`

LAPACK++ 1.0 matrix classes have been tested with the following compilers: Sun CC v. 3.0.1, and Gnu g++ v. 2.7.0.

The matrix classes contain support for the basic construction, assignments, indexing, and referencing operations. These matrix and vector objects can be used independently of LAPACK++ (see section 6.5).

This version of LAPACK++ relies on support routines found in the f77 or C version of LAPACK. If the f77 `liblapack.a` has not been built at your site, you will need the f77 or C LAPACK source code (also available from `http://www.netlib.org`). See [2] for details.

The directory structure of LAPACK++ is shown in figure 1. Its contents include

- main `LAPACK++` directory: C++ matrix libraries `liblamatrix++.a` and LAPACK++ library `liblapack++.a`.
- `MATRIX` subdirectory: Source and testing for LAPACK++ matrices and vectors, including General, Symmetric, SPD, Banded, Triangular cases. This includes only access, assignment, construction, and other non-arithmetic operations.
- `BLAS++` subdirectory: object-oriented extensions to the Basic Linear Algebra Subprograms (BLAS). These call either Fortran or assembly computational kernels of Level 3 BLAS for operations such as matrix multiply, triangular solve, and *rank-k* updates.
- `SRC` subdirectory: main LAPACK++ routines for solving linear systems and linear least-squares. Includes support for symmetric, SPD, banded, tridiagonal, and triangular matrices.
- `TESTING` subdirectory: main testing module call drivers for solving linear systems and linear least squares, together with various matrix decomposition modules for LU, Cholesky (LL^T) and QR factorizations. (See section 4 for details.)
- `LIBS` subdirectory: this is where the LAPACK++ libraries (`lapack.a, blas++.a, matrix++.a`) are for each architecture (e.g. `SUN4`, `RS6K`, etc.).

3.4 Building LAPACK++

3.4.1 Editing the Makefiles

After unpacking the source code from `netlib` one will have the directory structure in figure 1. You will first need to modify the machine-dependent `makefile.def` file to describe essential information of your environment. There are several machine dependent makefiles (figure 2 included in the distribution. In these, four macro variables need to be specified:

- `LAPACK_LIB` this is where the C or f77 LAPACK library exists on your system. Typical places are `/usr/local/lib/lapack.a`. If this library does not exist you must first built it. The source for both C and Fortran versions are also available via netlib. See [2] for details.
- `BLAS_LIB` this is where the Basic Linear Algebra Subprogram (BLAS) library exists. On some machines this may be supplied by the vendor. If there already exists an LAPACK library on your system, most likely the BLAS are already installed. The BLAS package is also available via netlib.
- `LAPACKPP_LIB` where to intall the LAPACK++ library. Typically under `~/lapack++/LIBS/`).
- `HASRANLIB` either 't' or 'f', depending on whether the current platform supports the `ranlib` command.
- `ARCH` platform architecture used in `arch.h`.

  This is used to determine the correct linkage between Fortran, C, and C++ function names.

Optional variables that can also be set include

- `TMGLIB` Set this only if you wish to run the test modules **and** the `tmglib.a` library is not available on your system. (It should reside in the same place where the LAPACK library was found.) This is an optional module of LAPACK used for the generation of test matrices.

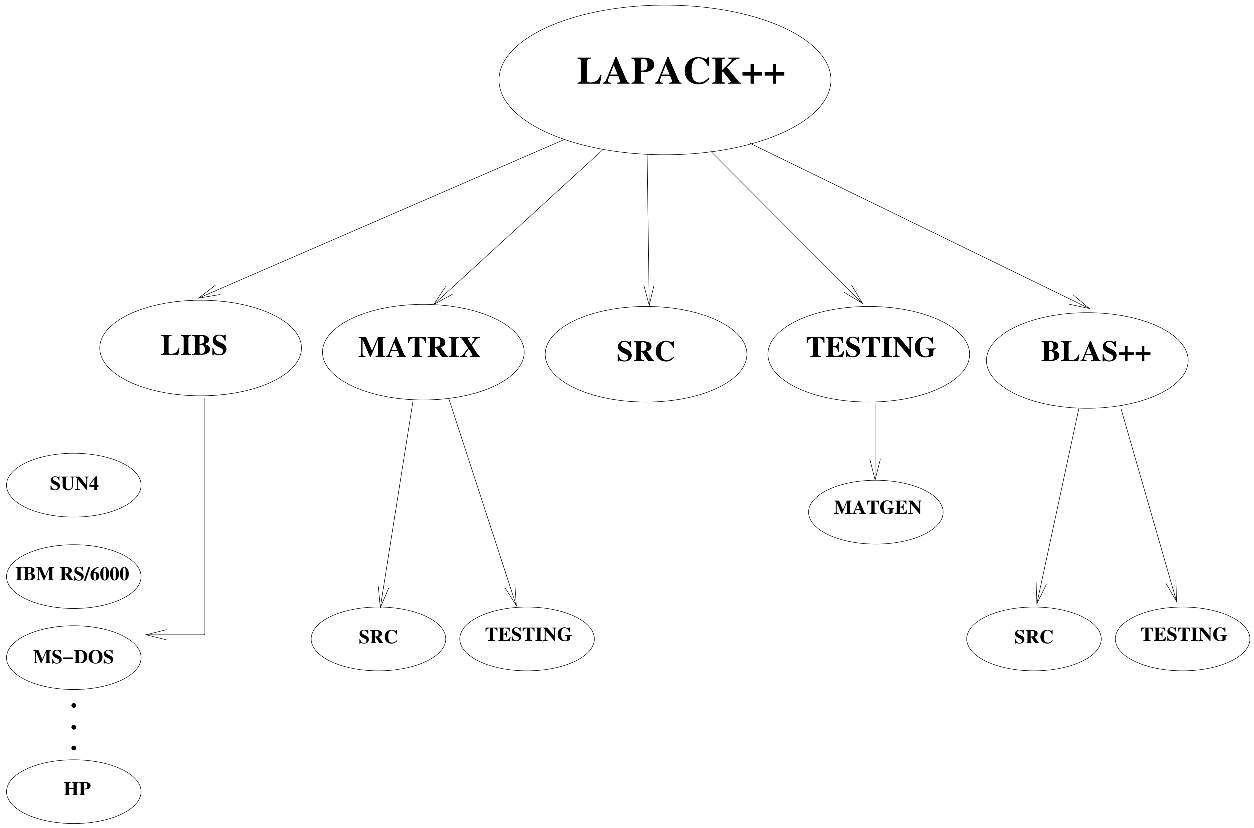

Figure 1: Directory structure of LAPACK++ source code.

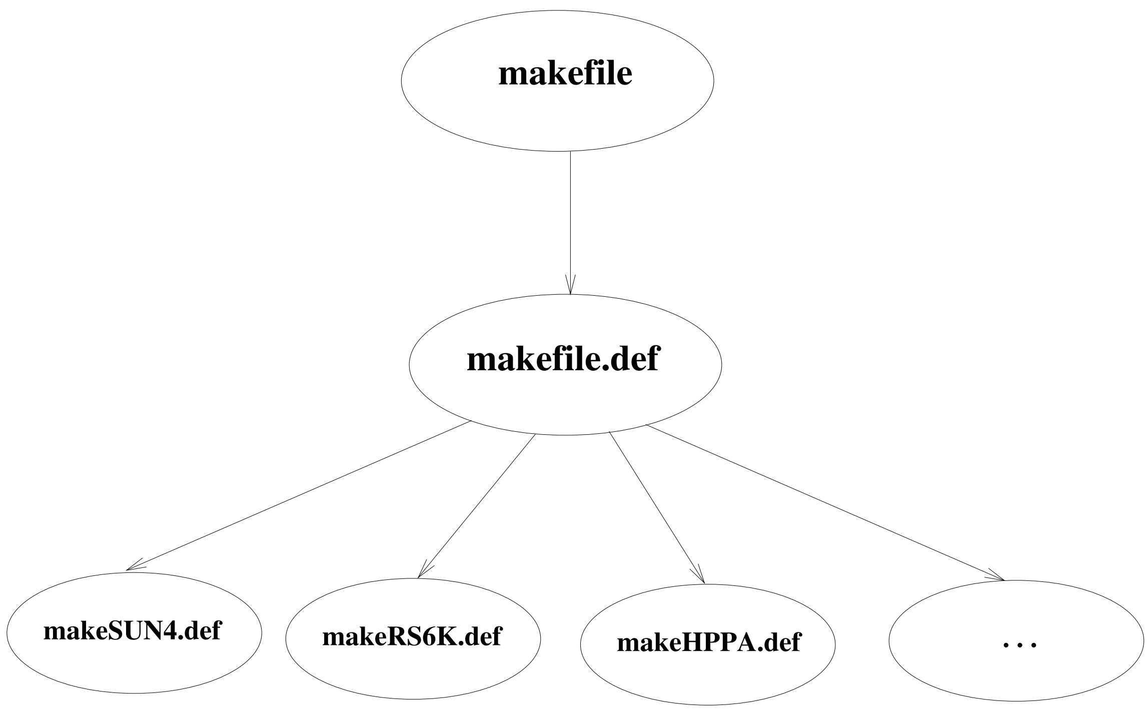

Figure 2: Makefile structure of LAPACK++ source codes.

- LAPACKPP_LIB the name and path of the newly created LAPACK++ library, (typically $(LAPACKPP_DIR)/liblapack++.a).
- LAMATRIX_LIB the name and path of the newly created C++ Matrix library (typically $(LAPACKPP_DIR)/liblamatrix++).

Now one can specify in the main makefile of the LAPACK++ root directory the exact packages to build:

- C++ Matrix class
- C++ Testing Module
- LAPACK++ library
- LAPACK++ Testing Module

4 Testing LAPACK++

Since the f77 library liblapack.a has already been installed, much of the testing of numerical algorithms for accuracy and robustness has already been performed. The tests in LAPACK++ focus on verifying the correctness of matrix structures, and the treatment of argument parameters between the f77 and C++ interfaces.

Each LAPACK++ subdirectory (see figure 1) has its own testing module. The testing routines in LAPACK++/MATRIX/TESTING run through various access, initialization, and referencing methods of the basic vectors and matrices and compare the results with pre-generated scripts.

The testing routines in LAPACK++/TESTING verify acceptable solutions to various linear problems $Ax = b$ by performing the residual check

$$\frac{||Ax - b||_\infty}{N||A||_\infty ||x||_\infty \epsilon} < 1$$

For solving over-determined systems, the linear least square solution utilizes a similar residual

$$\frac{||A^T(Ax - b)||_\infty}{N||A||_\infty ||x||_\infty \epsilon} < 1$$

for verification.

4.1 Test Matrix Generation Library

The LAPACK++ testing routines rely on various modules for the generation of random test matrices of particular structures (symmetric, SPD, banded, etc.) These generation routines are found in the Fortran library tmglib.a. It is an optional part of LAPACK, and should be in the same place where liblapack.a was found. If not, the Fortran source code for this matrix generation library is included as part of LAPACK++ in the MATRIX/MATGEN subdirectory.

If you wish to build it, go in this subdirectory and type make. The library will be created and reside in the main LAPACK++ directory.

4.2 How to report bugs

Report bugs, comments, questions and suggestions to lapackpp@cs.utk.edu.

5 Tips and Suggestions

5.1 Reducing Compilation time

5.1.1 Minimize the number of included header files

LAPACK++ header files can be quite large, particularly when one takes into the various data types and different matrix types (e.g. symmetric, upper triangular, tridiagonal).

If your compiler does not support the ability pre-compile header files, it could be spending a significant portion of its time processing LAPACK++ object and function declarations.

The main header `lapack++.h` is a "catch-all" file containing complete declarations for over fifty matrix classes and several hundred inlined matrix functions. While this file is most general, it is unlikely that one will need all of its declarations. It is much more efficient to include a specific header file for a matrix storage class or matrix data type.

5.1.2 Header file madness

The LAPACK++ header files will tipical only include those function declarations for the matrix types currently defined. For example, if `LaSpdMatDouble` matrices are used, then the compiler macro `_LA_SPD_MAT_DOUBLE_H_` is defined. Thus, the LAPACK++ can select to include only those header files which are relevant.

The alternative is to include every possible matrix type and LAPACK++ function, for every program, even a tiny program which uses a small portion of the `LAPACK++/MATRIX++` classes.

Therefore, one should declare all of their matrix and vector types **before** including BLAS++ and LAPACK++ header files.

5.2 Performance Considerations

5.2.1 Array Bounds Checking

LAPACK++ allows the application developer to determine the level of run-time index range checking performed when accessing individual $A(i,j)$ elements. It is strongly recommend for developers to use the `LA_BOUNDS_CHECK` whenever possible, thus guarding against out-of-bounds references. This is typically used in the design and debugging phases of the program development, and can be turned off in production runs.

5.2.2 Improving $A(i,j)$ efficiency

The LAPACK++ matrix classes were optimized for use with BLAS and LAPACK routines. These routines require matrices to be column-ordered and adhere to Fortran-like accessibility.

The code for `A(i,j)` inlines integer multiplications to compute the address offset; however, some C++ compilers cannot optimize this out of a loop. An alternate implementation uses an indirect-addressing scheme similar to $A[i][j]$ to access individual elements. Tests have shown this to be more successful on various C++ compilers: Sun, Borland.

5.3 Exception Handling

Although exception handling has been officially accepted into the language by the ANSI committee it is not yet reliably supported by all C++ compilers. For sake of portability, LAPACK++ uses a macro mechanism to simulate `throw` expressions so that the library code will work correctly when exception handling becomes supported:

```
#define throw throw_
inline void throw(const char *message)
{
```

```
    cerr << "Exception: " << message << endl;
    exit(1);
}
```

Transfer of control does not correspond to a "try" block, but rather exits the program with the proper diagnostic message. This is a similar behavior to the real exception handling if there was no explicit user-supplied try block.

6 Frequently asked questions

6.1 What is the performance of LAPACK++?

For medium-to-large large matrices (say $n > 100$) where performance is a consideration, LAPACK++ performance is identical to the native Fortran LAPACK codes. For example, a recent test on IBM RS/6000 workstation showed both packages obtaining speeds of 50 Mflops for matrices of size 300x300 and larger. This is not surprising, since the both utilize the same optimized Level 3 BLAS routines for the computation-intensive sections. See [6] for more performance details.

6.2 Do I need a Fortran compiler to use LAPACK++?

No, you can use the C version of LAPACK available from `netlib`.

6.3 Why are the LAPACK++ matrix classes not templated?

There is a templated vector example `template_v.h` in the `LAPACK++/INCLUDE` subdirectory. This should provide provide an example for those who wish to create vectors and matrices of arbitrary types.

LAPACK++ does not support generic matrix types, since the underlying BLAS supports only support real and complex floating point numbers.

Further work into fully templated support is being conduncted under the Template Numerical Toolkit (`http://math.nist.gov/tnt`) project.

6.4 Can LAPACK++ work with built-in C/C++ arrays and other matrix classes?

Yes. LAPACK++ routines utilize contiguous storage for optimum data reuse (Level 3 BLAS) and C++ arrays must be explicitly transformed into this data storage before integrating with LAPACK++. This is easily accomplished with the various matrix constructors, e.g. `LaGenMatDouble::LaGenMatDouble( double **, int, int )`.

For integrating with external C++ matrix classes, all that is required of a user-specific matrix class is the ability to access individual elements, i.e. something similar to `A(i,j)`. Any C++ matrix library should support this. If your matrix class uses column-major ordering, then the conversion can be as simple as a copying a pointer to the first element. Consult the *LAPACK++ Users' Manual* for more details.

6.5 Can the matrix and vector objects be used independently of BLAS++ and LAPACK++?

Yes, the MATRIX++ subdirectory `LAPACK++/MATRIX++` subdirectory (see figure 1) can be used as an independent package. This may be useful if to develop matrices and vectors of user-defined types. The codes in this subdirectory define only access functions, assignments, construction, and basic non-arithmetic relationship between matrices.

6.6 Is there an existing standard for C++ matrix classes?

ANSI C++ defines a valarray class, as well as a templated vector class in the Standard Template Library (STL). Neither of these classes in their current form, however, are ideal for numerical matrix classes.

Linear algebra matrices, particularly sparse matrices, have little resemblence to conventional "dense" arrays. Furthermore, the types of operations and optimizations are quite different than one or two-dimensional arrays. For example, an expression such as `A(i,j) = 1.0` may involve a hash-table lookup, or linked-list traversal. (The classes in Lapack++ have been designed to avoid such scheme, but this is just one formulation – there are others.)

For more info on matrix classes built using STL components, see the Template Numerical Toolkit, `http://math.nist.gov/tnt`.

References

[1] E. Anderson, Z. Bai, C. Bischof, J. W. Demmel, J. J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen, *LAPACK User's Guide*, SIAM, Philadelphia, 1992.

[2] E. Anderson, J. J. Dongarra, S. Ostrouchov, *Installation Guide for LAPACK*, LAPACK Working Note 41, University of Tennessee, Computer Science Technical Report CS-92-151, Feb., 1992.

[3] J. Choi,J. J. Dongarra, R. Pozo, D. W. Walker, *ScaLAPACK: A Scalable Linear Algebra Library for Distributed Memory Concurrent Computers*, Frontiers of Massively Parallel Computing, McLean, Virginia, October 19-21, 1992.

[4] J. Choi and J. J. Dongarra and D. W. Walker, *PB-BLAS : Parallel Block Basic Linear Algebra Subroutines on Distributed Memory Concurrent Computers*, Oak Ridge National Laboratory, Mathematical Sciences Section, in preparation, 1993.

[5] J. J. Dongarra, R. Pozo, D. W. Walker, *An Object Oriented Design for High Performance Linear Algebra on Distributed Memory Architectures*, Object Oriented Numerics Conference (OONSKI), Sunriver, Oregon, May 26-27, 1993.

[6] J. J. Dongarra, R. Pozo, D. W. Walker, *LAPACK++: A Design Overview of Object-Oriented Extensions for High Performance Linear Algebra*, Computer Science Technical Report, University of Tennessee, 1993.