LAPACK for Windows

LAPACK FOR WINDOWS

- What do you need?
- Prebuilt libraries
- Running LAPACK under Windows
- Running LAPACKE from VS Studio (just C code, no Fortran!!!)
- Easy Windows Build

What do you need?

To run LAPACK on Windows?

- If you have INTEL compilers installed on your machine, please download the Prebuilt static libraries using INTEL Compilers libraries
- If you **DO NOT** have INTEL compilers installed on your machine, you will need to install **MinGW 32 bits** or **MinGW-w64** and then download the Prebuilt dynamic libraries using Mingw
- Call LAPACK directly from C using the LAPACKE C Interface. You will need to install **MinGW 32 bits** and then download the Prebuilt dynamic libraries using Mingw or even better download a VS Studio Solution with everything ready (BLAS, LAPACK and LAPACKE lib and dll) and two simple LAPACKE examples), you will just need to unzip and build. Do not forget to consult also the LAPACKE User Guide.

To run a program calling a LAPACK routine under Windows?

Please follow this extensive guide provided by one of our user.

To build LAPACK libraries under Windows?

You will need to install **CMAKE** on your machine and please refer to the build section .

Prebuilt libraries for Microsoft Visual Studio Projects

Prebuilt dynamic libraries using Mingw

Requirement: Mingw 32 bits or 64 bits
Information: Those libraries were built with CMAKE for Visual Studio 2010 and Mingw compilers and correspond to **LAPACK 3.5.0**.

Instructions:

- Download the BLAS and LAPACK dll and lib that correspond to your need. See table below
- Link your C application built with MSVC with the BLAS and LAPACK libraries (the lib files) you just downloaded. In your project properties, change the properties "Linker > General > Additional Library Directory" to tell Visual Studio where the libraries are, and also add the name of your BLAS and LAPACK libraries in "Linker > Input > Additional Dependencies", just put "liblapack.lib;libblas.lib"
- Once your application compiled correctly, do not forget to copy the liblapack.dll and libblas.dll where your executable is or the make sure that the dll are on your system path or put them in the WINDOWS\system32 folder, else binary won't run
- Your application will also require the GNU runtime DLLs (both libgfortran-3.dll and libgcc_s_dw2-1.dll are needed.) from MinGW to be available. Just put the GNU runtime directory (for example, for 32 bits C:\MinGW\bin) in your PATH, you should be good to go

Instructions for LAPACKE:

- Download the BLAS, LAPACK and LAPACKE dll. At the moment only Win32 Release available (but you can build your own flavor with CMAKE) See table below
- Link your C application built with MSVC with the BLAS,LAPACK and LAPACKE libraries (the lib files) you just downloaded. In your project properties, change the properties "Linker > General > Additional Library Directory" to tell Visual Studio where the libraries are, and also add the name of your BLAS, LAPACK and LAPACKE libraries in "Linker > Input > Additional Dependencies", just put "liblapacke.lib;liblapack.lib;libblas.lib"
- Specifically for LAPACKE, you need to add ADD_;HAVE_LAPACK_CONFIG_H;LAPACK_COMPLEX_STRUCTURE;

in "C/C++ > Preprocessor > Preprocessor Definitions"
- Once your application compiled correctly, do not forget to copy the liblapacke.dll, liblapack.dll and libblas.dll where your executable is or the make sure that the dll are on your system path or put them in the WINDOWS\system32 folder, else binary won't run
- Your application will also require the GNU runtime DLLs (both libgfortran-3.dll and libgcc_s_dw2-1.dll are needed.) from MinGW to be available. Just put the GNU runtime directory (for example, for 32 bits C:\MinGW\bin) in your PATH, you should be good to go
- Do not forget to consult also the LAPACKE User Guide.

	Ref BLAS	LAPACK	LAPACKE
Release	x64 dll x64 lib win32 dll win32 lib	x64 dll x64 lib win32 dll win32 lib	x64 dll x64 lib win32 dll win32 lib LAPACKE header file LAPACKE mangling header file LAPACKE config header file LAPACKE utils header file

Prebuilt static libraries using INTEL Compilers

Requirement: Intel Compiler for Windows
Information: Those libraries were built with CMAKE for Visual Studio 2010 and INTEL compilers and correspond to **LAPACK 3.5.0**.

	Ref BLAS	LAPACK	LAPACKE
Release	x64 win32	x64 win32	x64 win32
Debug	x64 win32	x64 win32	x64 win32

Running LAPACK under Windows

Thanks to Olumide, Evgenii Rudnyi and Mark Hoemmen at http://groups.google.com/group/matrixprogramming their early suggestions and the reading the draft of this HOWTO. Any inaccuracies in this document are mine. Use with care.

Part 1: Getting a BLAS library

Preamble
LAPACK is designed as a two-tiered Fortran library, comprising higher level subroutines and "lower-level Basic Linear Algebra Subprograms (BLAS) in order to effectively exploit the caches on modern cache-based architectures" (http://en.wikipedia.org/wiki/LAPACK). For reference purposes, the LAPACK installation provides a(n untuned) version of the BLAS which is not optimized for any architecture. This **reference** BLAS implementation may be orders of magnitude slower than optimized implementations, for matrix factorizations and other computationally intensive matrix operations. Optimized implementations the BLAS are available from a number of vendors and projects such as: **Intel** (commercial), **AMD**, **ATLAS**, and **GotoBLAS**.

Part 1-a: Using the Reference BLAS

The Reference BLAS for Windows can be downloaded here .

Part 1-b: Using the MKL BLAS

MKL - http://software.intel.com/en-us/intel-mkl/ This is a good match to go with INTEL Fortran compiler of course.

Part 1-b: Using the ACML BLAS

AMD (AMCL) -- http://developer.amd.com/cpu/Libraries/ ... fault.aspx

Part 1-c: Using ATLAS BLAS (requires compilation)

-- Cygwin or MinGW required --
ATLAS is aim to, by self-discovery, automatically generate an optimized BLAS library. For a step by step procedure please refer to the ALTAS Forum

Part 1-c: Using GotoBLAS (requires compilation)

-- MinGW and MSYS required (MinGW has to be installed first) --

The GotoBLAS source is available from here (there's short registration form to fill), and can be compiled for Windows with

MinGW. No changes need to be made to GotoBLAS' config file Makefile.rule, unless a particular compiler is preferred. Happily, the config file automatically enables multithreading if more than one processor is available.

1. Download and extract the GotoBLAS source to any directory of choice, and make any desired changes to the config file (the default option should also work well).
2. "cd" to the top-level directory containing the source, and type "make"
3. The result of this process should be a file libgoto_-r.a, and a (symbolic) link libgoto.a pointing to this file. (For example, libgoto_banias-r1.26.a) but also the Windows library (*.lib) and dll are generated by default. .

Final note:
* Windows needs to be told where to find this dll, else you will get a serious error when you try to run your program. There are several ways to do accomplish this. One, is to add the location of the dll to the PATH environment variable. Another is to simply copy the dll to the Windows\system32 folder. I did the later.
For more information, refer to Microsoft guidelines on Search Path Used by Windows to Locate a DLL

Part 2: Using LAPACK subroutines in a Visual Studio INTEL FORTRAN Project

1. Download the LAPACK precompiled binaries. File names of the precomputed debug libraries end with the letter "d" e.g. BLASd.lib and LAPACKd.lib (in comparison to the release versions BLAS.lib and LAPACK.lib).
2. Locate your BLAS libraries for your machine. (You may want to choose the Debug config if you choose GOTOBLAS)
3. Download the LAPACK-VS-Example Visual Studio project and unzip
4. Move or Copy the libraries from step 1 in the LAPACK-VS-Example folder.
5. If you are not using the Reference BLAS, you will need to modify change the properties "Linker > General > Additional Library Directory" to tell Visual Studio where the libraries are, and also add the name of your BLAS library in "Linker > Input > Additional Dependencies"
6. Compile the project and run the resulting executable. You should get the output:

```
Hello World
INFO =           0
  3.00000000000000       0.333333333333333        4.00000000000000
 0.666666666666667
 -4.00000000000000        4.50000000000000
END OF PROGRAM...
```

Part 2: Using LAPACK subroutines in a Visual Studio C/C++ Project without a FORTRAN compiler

1. Download the LAPACK precompiled binaries for MinGW. You should have a two files: **liblapack.lib** and **liblapack.dll** (libblas.lib libblas.dll if you also want the Reference BLAS) .
2. OPTIONAL: Obtain a tuned version of BLAS for your machine (refer to "Compiling GotoBLAS").
3. Create a Visual Studio project with the following sample C program:
4. For C++ program, rename the prototypes in the above program to

```
void dgesv_( )
void dgels_( )

to
extern "C" void dgesv_( )
extern "C" void dgels_( )
```

5. Add the the BLAS and LAPACK libraries to the Visual Studio project settings,
   under Linker -> General -> Additional Library Directories: the directory where your liblapack.lib is.
   under Linker -> Input -> Additional Dependencies: libblas.lib;liblapack.lib (For example, on my machine, I am using the Reference BLAS)
   Note: because BLAS libraries commonly provide faster versions of some LAPACK subroutines, the BLAS library must be listed before before LAPACK library.
   Note: make sure that all the dll (BLAS, LAPACK, MinGW dlls) are on your system path or copy them in the WINDOWS\system32 folder, else binary won't run.
6. Compile the project and run the resulting executable. You should get the output: The solution is

```
-0.661082 9.456125 -16.014625
```

Prologue:
Part 3 of this HOWTO will briefly explain what "dgesv" means and how to call it and other LAPACK subroutines with the appropriate arguments.

Part 3: More about LAPACK subroutines (example: dgesv)

In the previous section, I explained how to call a LAPACK subroutine e.g. dgesv_ from a C or C++ program, but I did not explain what the dgesv meant as well as its arguments. This is the purpose of this part of the HOWTO. In doing so, I will refer to the LAPACK documentation and hopefully show how easy it is to find an appropriate LAPACK subroutine and create the corresponding C/C++ function prototype for it.

Understanding dgesv

Prefix -- "dge"

From the LAPACK naming scheme -- http://www.netlib.org/lapack/lug/node24.html, it is plain to see that:

- d in "dgesv" means: double precsion data
- ge in "dgesv" means: general as in unsymmetric matrix

As such we can deduce that "dge" refers to the sort of matrix we have -- a general/unymmetric matrix containing double precision data

Suffix -- "sv"

This refers to the type of driver routine (solver in lay speak) to be used to solve the linear system. There are two kinds on drivers: simple drivers (suffixed with "sv") and "expert" drivers (suffixed with "svx"). Refer to http://www.netlib.org/lapack /lug/node26.html .

Therefore dgesv is simple driver routine for a general/unymmetric matrix containing double precision data.

Subroutine arguments

From the page http://www.netlib.org/lapack/double/dgesv.f , we can see that the subroutine dgesv has 8 arguments.

- The first argument is N, an integer. This is marked as an input (meaning argument will not be modified, as opposed to an input argument or an input/output argument) in the documentation. In C/C++ speak we can therefore refer to argument 1 as a constant integer i.e. "const int". However, because in Fortran all ALL arguments, *without exception* are passed by address, the type of N in C/C++ is: const *int . (Same goes for argument 2.)
- Argument 3, marked in the documentation as an input/output double precision array. In C/C++ terms input/output means NOT-constant. Therefore, because arguments are passed by reference, the type of argument 3 is: double * .
- Argument 5, marked in the documentation as an output integer array. In C/C++ terms this means the argument is not a const. Therefore argument 5 is of type int* . Same goes for argument 8, even though the argument is not an array (remember, all Fortran arguments are passed by address).

It should now be clear why the C/C++ prototype for dgesv is CODE: SELECT ALL extern "C" void dgesv_(const int * , const int * , double * , const int * , int * , double * , const int * , int *);

Prologue:

A pattern for using directly LAPACK subroutines should now be clear.

- First find the appropriate subroutine from the list of available drivers here.
- Look up the driver in the index of routines here.
- Create a the appropriate C/C++ prototype for the driver.

Using LAPACKE subroutines in a Visual Studio C/C++ Project

1. Download the Visual Studio Solution LAPACKE examples and unzip.The Solution contains all the includes, libraries and dlls you need.
2. Build Solution (only Win32/Release available)
3. Open a cmd prompt (Click Run.. then enter cmd)
4. Go to your LAPACKE_examples/Release folder using the cd command
5. You have two examples you can run: example_DGESV_rowmajor.exe and example_ZGESV_rowmajor.exe (The solution is written in the source file)
6. Optional: send your feedback to the lapack team at lapack@eecs.utk.edu

Easy Windows Build

Build Instructions for LAPACK 3.5.0 for Windows with Visual Studio

Requirements: Visual Studio, Intel C and Fortran Compilers, CMAKE 2.8.12

1. Download the **lapack.tgz** from the netlib website and unzip.
2. Download **CMAKE** and install it on your machine.
3. Open CMAKE
    - Point to your lapack-3.5.0 folder as the source code folder
    - Point to a new folder where you want the build to be (not the same is better)
    - Click configure, check the install path if you want to have the libraries and includes in a particular location.
    - Choose Visual Studio Solution.
    - You will have to click "Specify native compilers" and indicate the path to the ifort compiler. On my machine, it is "C:/Program Files (x86)/Intel/Compiler/11.1/048/bin/ia32/icl.exe".
    - You may have to click again configure until everything becomes white
    - Click generate, that will create the Visual Studio Solution and you are done.
    - Close CMAKE

4. Look in your "build" folder, you have your LAPACK Visual Studio Solution, just open it.
5. Build the **"ALL_BUILD"** project, it will build the solution and create the libraries
6. Build the **"INSTALL"**. This will put the libraries and include in your install folder.
7. Build the **"RUN_TESTS"**. The BLAS and LAPACK testings will be run.

Build Instructions to create LAPACK and LAPACKE 3.5.0 dlls for Windows with MinGW

Requirements: MinGW, CMAKE 2.8.12, VS IDEs

1. Download the lapack.tgz from the netlib website and unzip.
2. Download CMAKE and install it on your machine.
3. Download MinGW 32 bits or MinGW-w64 and install it on your machine.
4. Put the GNU runtime directory in your PATH, for me I added C:\MinGW\bin (MinGW 32 bits) in my PATH (right click on your computer icon, go to properties, advanced system settings, Environment Variables, look for the PATH variable and put 'C:\MinGW\bin;' in front of its current value)
5. Open CMAKE
    - Point to your lapack-3.5.0 folder as the source code folder
    - Point to a new folder where you want the build to be (not the same is better)
    - Click configure, check the install path if you want to have the libraries and includes in a particular location.
    - Choose MinGW Makefiles.
    - Click "Specify native compilers" and indicate the path to the Mingw compilers.
        - For Win32, on my machine, the Fortran Compiler is "C:/MinGW/bin/gfortran.exe", and the C compiler is "C:/MinGW/bin/gcc.exe"
        - For x64, on my machine, it is "C:/mingw64/bin/x86_64-w64-mingw32-gfortran.exe" and the C compiler is "C:/mingw64/bin/x86_64-w64-mingw32-gcc.exe"
    - **For x64 build ONLY** , add the variable CMAKE_SIZEOF_VOID_P and set it to 8 (string), this will force CMAKE to create the VCVARSAMD64 variable (see post on forum) Note: CMAKE team corrected the issue, and thus this workaround won't be needed if you are using CMAKE 2.8.13 or above
    - Click "Specify native compilers" and indicate the path to the Mingw compilers. On my machine, it is "C:/MinGW /bin/gfortran.exe"
    - Set the 'BUILD_SHARED_LIBS' option to ON.
    - Set the 'CMAKE_GNUtoMS' option to ON.
    - **if you want to build the LAPACKE library,** set the 'LAPACKE' option to ON.
    - Click again configure until everything becomes white
    - Click generate, that will create the mingw build.
    - Close CMAKE
6. Open a cmd prompt (Click Run.. then enter cmd)
7. Go to your build folder using the cd command
8. Type C:/MinGW/bin/mingw32-make.exe
9. Type C:/MinGW/bin/mingw32-make.exe test if you want to run LAPACK testings to make sure everything is ok
10. Your libs are in the lib folder, the dlls are in the bin folder. The resulting build will provide both GNU-format and MS-format import libraries for the DLLs.
11. Now you should be able to create a C application built with MSVC and linked directly to the MinGW-built LAPACK DLLs
12. NOTE: Your C application built with Microsoft Visual Studio and linked to the MinGW-built lapack DLLs will run but requires the GNU runtime DLLs (both libgfortran-3.dll and libgcc_s_dw2-1.dll are needed.) from MinGW to be available. As you have the GNU runtime directory in your PATH, you should be good to go.
13. Do not forget to consult also the LAPACKE User Guide.

Thank you to the CMAKE guys for providing this build.

Support and Feedback

We are working hard to improve the LAPACK Windows support but it seems that users still have problems. We would like to know how we are doing, and how we could further help you. Your post on the forum will be appreciated.

- Forum : http://icl.cs.utk.edu/lapack-forum/index.php
- Mailing List: lapack@cs.utk.edu