BLAS-RMD User Manual

Kristjan Jonasson, Sven Sigurdsson, Hordur Freyr Yngvason,
Petur Orri Ragnarsson, Pall Melsted

August 2019

Contents

1 Introduction

This short user manual contains information on the organization of the folders of the BLAS-RMD pacage, how to install it, check its correctness and run associated example programs. The information here is for the most part duplicated in files named `README` in each (sub)folder of the package and in comments in the `Makefile` in the BLAS-RMD root folder.

2 Overview of the subfolders of BLAS-RMD

Apart from subfolders, the root folder of BLAS-RMD contains three files, `Makefile`, `make.inc` and `README`. The subfolders are the following:

src	F90 sources for single precision RMD-subroutines. Double versions are auto-generated here
demo	Two example programs
tools	Tools for testing, timing, double precision generation and doc generation
test	Test programs
time	Timing programs. Contains subfolder `reports` with results
refblas	Those Fortran routines from Netlib BLAS/LAPACK version 3.7.0 that are needed
evidence	Evidence of portability, standard conformance and test passing
article	Latex sources for the article [1] submitted to TOMS
doc	Latex sources for building the reference manual and this user manual
matlab	Matlab/Octave implementation of the example programs and a few adjoints
vauto	A VARMA time series Matlab package, used by the Matlab programs

In addition the following folders are automatically generated:

lib	For the libraries `libblasrmd.a` and `librefblas.a`
test/more	Contains additional test programs

3 Software packages needed for BLAS-RMD

The following supporting software packages are needed to install and test the BLAS-RMD package and remake its documentation.

- A Fortran compiler
- A BLAS/LAPACK library
- GNU Make
- Awk
- Python
- Latex

The file `README` in the root folder of BLAS-RMD contains a few short comments about how to obtain and install (many of) these packages on Linux and Mac computers.

4 Installation and compilation of BLAS-RMD

The first steps, after unzipping the package delivery file, is to install necessary supporting software. A Fortran compiler, GNU Make, Awk and Python are needed as a minimum. Next, the Fortran compiler and the BLAS library to use must be made known to Make, as well as the location of the BLAS library. One must also decide on the build type: optimized, debug-enabled, or neither. There are three ways to convey this information to Make:

- with an environment variable
- with a variable set on the make command line
- by editing `make.inc`

The compiler is specified with the variable `FC`, the BLAS library with the variable `BLAS` and the build type with the variables `debug` and `optimize`.

After making (compiling) a library from the source files one should run the testing and timing programs to ascertain that the installation was successful. Issuing the command "make" in the root directory will make the RMD-library and run the examples. To run testing and timing, refer to the next subsection. There is additional information in the file `README` in the package root directory. The settings file `make.inc` should work as delivered on both Linux and Macintosh, compiling with gfortran and using OpenBLAS.

Main make targets

The package employs a fairly simple non-recursive make, where all makefiles are in or below the BLAS-RMD root folder. Making from the root folder will ensure that all dependencies are made, whereas making in a subfolder assumes that they are already up to date. Issuing `make <folder>` from the root makes the default target of the specified folder. The following targets are the most important (assumes issuing from the root):

make	makes the rmd-library `lib/libblasrmd.a` and makes and runs demos
make src	just makes the rmd-library
make all	makes everything (including the reference BLAS) and runs tests and demos
make testrun	makes and runs all tests. No output means that test passed
make timerun	makes and runs the timing program with default settings
make doc	remakes the documentation
make clean	removes everything makeable except pdf-documentation
make clean-all	removes also pdf-doc

See more details in the comments at the front of `Makefile` in the root folder.

Specifying the Fortran compiler

The supplied makefiles directly support four Fortran compilers, those of GNU, the Numerical Algorithms Group (NAG), Intel and PGI, specified by setting `FC` to `gfortran`, `nagfor`, `ifort` and `pgfortran` respectively. For example one could make `lib/libblasrmd` using the Intel compiler by issuing `make src FC=ifort`, or by issuing `export FC=ifort` at a Bash command prompt followed by `make src`. After switching compilers one must issue `make clean` before compiling with the new one.

Specifying the BLAS library

The supplied makefiles directly support four BLAS libraries: The Fortran reference BLAS from Netlib, OpenBLAS (the open source continuation of Goto BLAS), Intel's Math Kernel Library (MKL) and Apple's Accelerate BLAS.

The default is to use OpenBLAS so for that there is no need to set the `BLAS` variable, however the variable `OPENBLAS` must be set to point to the OpenBLAS root folder (the parent folder of the `lib`-folder where the library itself resides). With Bash this is most easily accomplished by setting the variable in `.bashrc` e.g. with `export OPENBLAS=$HOME/opt/openblas`. To use the reference BLAS provided with BLAS-RMD, simply give the BLAS variable the value refblas (e.g. with `make src BLAS=refblas`). No variable is needed to point at its location. The Mac BLAS, Accelerate, also doesn't need a location variable. To use it, it is enough to set `BLAS=accelerate`. Using the BLAS from MKL is a little more involved, but after proper installation and setup of environment variables, as outlined in the file `README` file in the BLAS-RMD root folder, it is sufficient to set `BLAS=mkl`.

Specifying the build type

Two Make variables may be used to specify compile options for the build type, `debug`, and `optimize`. Set `debug=yes` to make a debug-enabled build and `optimize=yes` for an optimized build. Additional options may be set with the `FOPTS` variable. For example, to make `nagfor` detect the use of uninitialized variables, one may set `FOPTS=-C=undefined`. Note that this last setting is only possible with the reference BLAS. The BLAS-RMD test programs have been run with the `C=undefined` option, as seen in the file `evidence/nagfor`.

5 Running the example programs

As outlined in the BLAS-RMD article [1] there are two example programs, and these reside in the subfolder `demo`. To make and run both examples using default settings issue:

```
make demo.
```

The first example computes and displays the function value and corresponding adjoint for a groundwater model sum of squares at two points. To carry out the numerical minimization of this sum of squares one may use the Matlab program `matlab/demo_run_groundwater.m` (using Matlab or Octave). To run only exampe 1 issue `./demo1_driver` from within the demo folder.

The second example computes the likelihood and corresponding adjoint for a one step ($p = 1$) vector autoregresive (VAR) time series model of dimension r for a series of length n. The data may be read from a file named `var_<r>_<n>.dat` or generated inside the program. Four example files are provided for the pairs $(r, n) = (1, 4)$, $(2, 3)$, $(2, 4)$ and $(2, 40)$. New datafiles may be constructed with `matlab/init_demo_var.m`. To see a summary of running example 2 issue `./demo2_driver` from the `demo` folder, with more details given by comments in `demo/demo2_driver.f90`. The demo-folder also contains programs to check the correctness of the adjoints computed in the examples which may be run by issuing `make testdemo`, as well as a shell script for timing demo 2 explained shortly in `demo/README`.

6 Test programs

The test programs accompanying the package consist of two groups, as detailed in the readme-file in the test folder. The first group contains tests to check that the computed derivatives match numerical derivatives for random matrices of various sizes and shapes (general, band, triangular, symmetric). All possible combinations of the control character options, `side`, `uplo`, `diag` and `trans_`) are also tested. The read-me file also contains some information on how the analytical and numerical derivatives are computed.

The second group of test programs tests that arrays retained from a corresponding BLAS call are not changed by an RMD-routine, that adjoints that are not selected via the SEL parameter also remain unchanged, that selected adjoints are computed the same regardless of the SEL value, and that adjoints that should be updated are indeed added to.

There are also tests for the two example programs in the demo folder, as well as for the Lyapunov equation solver in that folder.

To compile and link all the provided test programs, issue `make test`. To (compile, link and) run all the tests, issue

```
make testrun.
```

The running of `testdemo` was explained in the last section. The output from running the tests on an Apple Macintosh, demonstrating that all tests have passed, is in the files `test.log` and `testdemo.log` in the evidence directory. Several more make targets exist in `test/Makefile`, and these are described in `test/README`.

7 Running the timing program

The BLAS-RMD timing program is in `timermd.f90`. Once the executable is made via `make time`, from the root folder, cd into the time folder and issue the command

```
./timermd
```

to obtain help on its parameters and examples on how to run it. The subfolder `time/reports` contains results of timing with several combinations of matrix sizes, number of threads, compiler, platform, and BLAS. That folder contains its own readme-file, explaining briefly how such a report could be made. See the file `time/README` for some more information.

8 References

[1] Kristjan Jonasson, Sven Sigurdsson, Hordur Freyr Yngvason, Petur Orri Ragnarsson, Pall Melsted, *Algorithm xxx: Fortran subroutines for reverse mode algorithmic differentiation of BLAS matrix operations*, ACM Transactions on Mathematical Software (TOMS), xx, xx, 2020.