

MUESLI

Build and Installation Guide

Fortran implementation

Release 2.23.5

Édouard CANOT*

May 4, 2026



*IPR/CNRS, Rennes, France

Contents

1	Introduction	3
2	Working environment	4
2.1	Operating System	4
2.2	Communication protocol: X11 and Wayland	4
2.3	Screen interaction using mouse and keyboard	5
2.4	Compilers	5
2.5	Hardware counter	5
2.6	Third-party libraries	5
2.6.1	Included libraries	5
2.6.2	Mandatory commands and libraries	5
2.6.3	Recommended libraries	6
2.6.4	Optional libraries or tools	7
2.7	Fonts	7
3	Installation	8
3.1	Getting MUESLI	8
3.2	Unpacking the distributed package	8
3.3	Configuring MUESLI	8
3.3.1	Launching the <code>configure</code> script	8
3.3.2	What to do when the <code>configure</code> script fail?	8
3.3.3	After configuring	9
3.4	Compiling MUESLI	10
3.5	Running MUESLI tests (Optional)	11
3.6	Installing MUESLI	11
3.7	Verifying MUESLI installation	11
3.8	Matlab mex files installation (Optional)	12
A	Use of BLAS and LAPACK in MUESLI	13
A.1	BLAS and LAPACK implementations	13
A.2	Choosing or switching between implementations	14
B	Installing the HDF5 Fortran 90 interface (optional)	15

1 Introduction

This document describes the MUESLI Fortran library installation.

MUESLI is freely available at the following address: <https://perso.univ-rennes1.fr/edouard.canot/muesli>

More information can be found in the following documents:

- *MUESLI User's Guide*
- *MUESLI Reference Manual*
- *MUESLI Inside*

Copyright © 2003-2026, Édouard CANOT, IPR/CNRS, Rennes, France.

Bugs reports or comments: <mailto:Edouard.Canot@univ-rennes.fr>

About the name

Muesli: loose mixture of mainly rolled oats and often also wheat flakes, together with various pieces of dried fruit, nuts, and seeds. There are many varieties, some of which also contain honey powder, spices, or chocolate. (from <https://en.wikipedia.org/wiki/Muesli>)

Credits

Cover photograph: the photo on the cover is copyrighted by Simon Krzic. It can be used under a Limited Royalty Free License (see <http://www.dreamstime.com/muesli-imagefree1964928>).

2 Working environment

2.1 Operating System

Linux (32-bit as well as 64-bit) is the platform which has been used for developping and testing MUESLI. The whole library (including the two main components, FML and FGL) should work on any UNIX platform.

For Windows users, the Muesli library may be built under the mingw-w64 chaintool (see <http://mingw-w64.sourceforge.net/>), using the GCC compiler. You must have a complete mingw-w64 installed, providing the usual UNIX tools (shell interpreter, make, ar, etc.). These last years, MinGW may appear to be difficult to install, so a good alternative is to use the IDE *Simply Fortran* (see <http://simplyfortran.com/>) for which we provides some binary archives under the MUESLI web page (<https://perso.univ-rennes1.fr/edouard.canot/muesli/mingw.html>). Be aware that you cannot create interactive figures on the screen (the process relies on X11) but you can create EPS and PDF images of your figures during the computation.

Recently, support for Mac OS X has been improved. The full MUESLI library (both FML and FGL parts) may run under Darwin (the name of the Mac Operating System); Yosemite (Darwin 10.10) and Mojave (Darwin 10.14) versions have been fully tested, after installing XQuartz (see <https://www.xquartz.org/>). Many Mac OSes include Blas and Lapack libraries, but the pretty old version 3.2 (at least from Yosemite to Mojave) should be avoided. As specified on the MUESLI web page (see above), this version leads to problems, especially run-time errors and segmentation faults. Therefore, it is strongly recommended to compile and install a recent, supported Lapack version. We recommend the usage of *Homebrew* as package manager for Mac OS (see <https://brew.sh/>).

2.2 Communication protocol: X11 and Wayland

Historically, graphical applications on Linux relied on the *X11* protocol (implemented by the *Xorg* server). Nowadays, most modern desktop environments use the newer *Wayland* protocol as their primary display system.

It is important to understand that Wayland does **not** remove support for X11 applications. Instead, a compatibility layer named *XWayland* is almost always provided. This component implements a minimal X11 server running on top of Wayland, allowing legacy X11 applications to run unmodified.

The graphical part of MUESLI (*FGL*) relies on X11 libraries (*libX11*, *Xrender*, etc.¹). Therefore:

- on a native X11 session, FGL works as expected ;
- on a Wayland session, FGL runs through *XWayland*, which ensures compatibility without requiring any change in user code.

In practice, all graphical components of MUESLI have been successfully tested on recent Linux distributions using Wayland (via *XWayland*).

However, due to the more asynchronous rendering model of modern display systems (especially under Wayland), very short-lived graphical programs may occasionally exhibit issues such as blank windows if they terminate immediately after issuing drawing commands. This is not a limitation of X11 itself, but a consequence of the underlying display pipeline.

This issue has been recently addressed in MUESLI (2.23.5 release) by ensuring proper synchronization with the display system.

For information, you can check your current session type with:

```
$ echo $XDG_SESSION_TYPE
```

which typically returns `x11` or `wayland`.

Although it is still possible on some systems to select an X11 session at login time, this option is becoming

¹Installing development packages such as `libx11-dev` remains necessary even on Wayland-based systems, since applications are still compiled against the X11 client libraries.

less common. The recommended and supported configuration is therefore to run MUESLI on Wayland using *XWayland*.

2.3 Screen interaction using mouse and keyboard

In some situations, you may choose to interact with the Figure (to pan and/or zoom inside it); in this case, you must have obviously a mouse and a keyboard. Moreover, some shortcuts use the [CTRL] key (see the *Muesli Reference Manual*); take care that the [CTRL] key is not caught by some other feature of your desktop environment, for example the one which is named *"Highlights the current location of the pointer when the Control key is pressed and released"*.

2.4 Compilers

MUESLI is written mainly in the Fortran language (modern dialect however), but some parts need a C/C++ compiler.

On Linux, MUESLI is usually tested with the following compilers (all are good ones, and a review of their specs — or even a comparison between them — would be out of the scope of this manual), listed below in the alphabetical order:

- **GNU Compiler Collection**, versions ≥ 4.9 : **gfortran** (Fortran compiler), **gcc** (C compiler) and **g++** (C++ compiler) [*version 4.5 to 4.8 may or may not work, since they are no longer supported by Muesli*]
- **INTEL professional compilers**, versions ≥ 13 : **ifort** (Fortran compiler), **icc** (C compiler) and **icpc** (C++ compiler) [*versions ranged from 11 and 12 may or may not work, since they are no longer supported by Muesli*]
- **INTEL OneAPI LLVM compilers**, versions ≥ 2021 : **ifx** (Fortran compiler), **icx** (C compiler) and **icpx** (C++ compiler)

The well known problem of compilation cascades were eliminated using various ways (see the *MUESLI Inside* documentation).

NOTE: Since the end of 2007, PGI f95 is no more supported, for many reasons (one of them is that it is difficult to maintain the code for many compilers); likewise, NAG f95, OPEN64 and IBM XL support have been removed resp. in october 2008 and august 2013. SOLARIS (Oracle) and G95 (FSF) support have been removed in june 2014.

2.5 Hardware counter

MUESLI is able to count the number of floating-point operations (flops). This way has been chosen because it is more reliable than measuring CPU times. However, it requires the PAPI library which counts flops via hardware counter. Under linux, hardware counting is available via patching the kernel for the PERFCTR library.

Be aware that multicore processors cannot count easily the exact number of floating-point operations... therefore, the PAPI installation is recommended only for moncore processors.

2.6 Third-party libraries

2.6.1 Included libraries

MUESLI already includes all or some parts of the following numerical/graphical libraries: ARPACK, SLATEC, SUITESPARSE, METIS, TRIANGLE and MFPLLOT. However, some additional libraries are required while others are only optional (see below).

2.6.2 Mandatory commands and libraries

The `configure` script (described in next section) requires the `lsb_release` command to work properly. It is linked to the *Linux Standard Base*, a common runtime environment for third-party packages. This command is usually included for Ubuntu distributions, but this is not always the case when dealing with other distros.

To execute the `configure` script, you will need the `which` and the `make` command.

MUESLI requires at least BLAS and LAPACK version 3 (Fortran linear algebra package, version 3.4 to 3.12 recommended); See also appendix A. These libraries may be included by your compiler (*e. g.*, they are included in the MKL library of INTEL). The archived version of these libraries is mandatory: it can be installed (if not already present) by looking for `-dev` packages of BLAS and LAPACK (for Debian-based distributions) or by looking both for `-devel` and `-static` packages (for RPM-based distros).

To help the user, Table 1 presents the different package names for three common linux distributions (Debian, Fedora and ArchLinux).

	Debian-based (<i>e. g.</i> Ubuntu)	RPM-based (<i>e. g.</i> Fedora)	ArchLinux
Command			
<i>lsb_release</i>	<code>lsb-release</code>	<code>lsb_release</code>	<code>lsb-release</code>
<i>which</i>	<code>debianutils</code>	<code>which</code>	<code>which</code>
<i>make</i>	<code>make</code>	<code>make</code>	<code>make</code>
Library			
<i>BLAS</i>	<code>libblas-dev</code>	<code>blas-devel</code> + <code>blas-static</code>	<code>blas</code>
<i>LAPACK</i>	<code>liblapack-dev</code>	<code>lapack-devel</code> + <code>lapack-static</code>	<code>lapack</code>
<i>ZLIB</i>	<code>zlib1g-dev</code>	<code>zlib-devel</code>	<code>zlib</code>
<i>READLINE</i>	<code>libreadline-dev</code>	<code>readline-devel</code>	<code>readline</code>
<i>LIBX11</i>	<code>libx11-dev</code>	<code>libX11-devel</code>	<code>libx11</code>
<i>FREETYPE</i>	<code>libfreetype-dev*</code>	<code>freetype-devel</code>	<code>freetype2</code>
<i>FONTCONFIG</i>	<code>fontconfig</code> + <code>libfontconfig-dev**</code>	<code>fontconfig</code> <code>fontconfig-devel</code>	<code>fontconfig</code>
<i>XRENDER</i>	<code>libxrender-dev</code>	<code>libXrender-devel</code>	<code>libxrender</code>
<i>XRANDR</i>	<code>libxrandr-dev</code>	<code>libXrandr-devel</code>	<code>libxrandr</code>
<i>IMAGEMAGICK</i>	<code>imagemagick</code>	<code>ImageMagick</code>	<code>imagemagick</code>

Table 1: Packages' name for different commands according to different linux distributions. Note that the packages' name are case sensitive.

Remark: (*) On old Ubuntu releases, it may write `libfreetype6-dev`. (**) The package `libfontconfig-dev` is for recent Ubuntu releases (≥ 21.04); previously, it was `libfontconfig1-dev`.

Remark on X11-related libraries

Even on systems using Wayland as the primary display protocol, the development packages of X11 libraries (such as `libx11-dev`, `libxrender-dev`, etc.) are still required to build graphical applications like MUESLI.

This is because these applications are compiled against the X11 client API. At runtime, they are transparently executed through the *XWayland* compatibility layer.

Note also that some X11-related development packages depend on each other. For example, installing `libxrender-dev` will automatically install `libx11-dev`, since the Xrender extension is built on top of the core X11 library.

2.6.3 Recommended libraries

Verify that the following libraries (together with their development version, providing the headers, when appropriate) are installed on your machine:

- ZLIB (lossless data compression); used to compress data when saving *mfArrays* on disk.

- XTERM: provides the shell command `RESIZE`, allowing the program to know the actual terminal width, and therefore adapt the layout of the `msDisplay` routine.

Notes for some specific Linux distributions:

- On RPM-based systems (like Fedora), you could type for each `<LIB>` library:

```
$ rpm -q <LIB>-devel
```

Note that the ‘devel’ version of these libraries add some necessary include files.

If some library is not present on your system, you must install it with, for example, the `dnf` command:

```
$ sudo dnf install <LIB>-devel
```

- On Debian-based linux distributions (*e.g.* Ubuntu), you should be able to replace the `yum` command above with the `apt` (or `aptitude`) command². Moreover, most of development packages have either the ‘-dev’ or the ‘-headers’ suffix.

2.6.4 Optional libraries or tools

- READLINE (command line editing, with history management).
- LIBX11, along with FREETYPE, FONTCONFIG, XRENDER (the development version, as previously): X11 is the main graphic library for the linux system. Without them, you will not be able to interactively use FGL, the graphic part of MUESLI.
- LIBQT4 (the development version, as previously): it is one of the high level graphic libraries for the linux system. Install it if you plan to use the graphic tool ‘meditor’ (via `msMedit`) for editing matrices. Define the `QT4DIR` environment variable containing the path (without ‘lib’) for the Qt4 library.
- IMAGEMAGICK: it is a package dealing with images; it provides the ‘`convert`’³ command for importing images in FGL; if the ImageMagick package is not available, FGL can import only XPM images.
- HDF5 (Hierarchic data format – for storing scientific data), with the `f90` interface. You will perhaps have also to download the source code of the HDF5-1.6 library in order to build yourself the Fortran 90 interface (see appendix B).

2.7 Fonts

The following fonts (or their usual substitutes⁴) are supposed to be present on the system you are using: **Helvetica**, **Helvetica-Bold**, **Times**, **Times-Bold**, **Times-Italic** and **Times-Bold-Italic**.

Take care that a partial installation of a family (Helvetica or Times) in your local share folder may lead to a bad render by some PDF readers.

²You can issue the ‘`apt search package`’ command to get the exact name of each package.

³A bug in Ubuntu-16.10 prevents to launch the ‘`convert`’ command properly; if you obtain something like ‘`libpng12.so.0: file not found`’ then you should be able to fix the problem by following this page: <http://askubuntu.com/questions/840257/e-package-libpng12-0-has-no-installation-candidate-ubuntu-16-10-gnome/840268>.

⁴Many fonts close to **Helvetica** and **Times** are freely available on Linux. More importantly, it is recommended to use *metrically compatible* fonts, which will not change the layout (*i.e.* the relative position of the glyphs insides words). **Liberation** and **Nimbus** are known to be good substitutions.

3 Installation

3.1 Getting MUESLI

MUESLI is available on the Web at the following URL:

<https://perso.univ-rennes1.fr/edouard.canot/muesli/>

3.2 Unpacking the distributed package

The distributed MUESLI package is a bzip2-ed archive file.

Input the following command to unarchive it:

```
$ tar xvfj muesli-sources-2.21.2_2024-01-26.tar.bz2
```

This will produce a directory named `muesli-sources-2.21.2_2024-01-26`, hereafter called `<MUESLI>`. Please enter this directory:

```
$ cd <MUESLI>
```

3.3 Configuring MUESLI

3.3.1 Launching the configure script

MUESLI comes with a ‘`configure`’ shell script which attempts to check all the requirements and try to discover the path for some used libraries.

Under the directory named `<MUESLI>`, you should find a directory whose name comes from a compiler:

`GNU_GFC` (GNU Compiler Collection)

`INTEL_IFC` (INTEL compiler)

Enter the directory of your choice. The ‘`configure`’ script⁵ must be launched inside one of these directories.

Please first read the available options by typing:

```
$ ./configure --help
```

You may use the `--prefix` option in order to specify the installation directory (default is `/usr/local`):

```
$ ./configure --prefix=$HOME
```

Caveat: when choosing the installation path for installing MUESLI, you cannot choose a path containing any blank character (the installation script will fail).

3.3.2 What to do when the configure script fail?

Usually, the `configure` script should detect all things needed by the MUESLI library. However, the great number of different Linux distributions may lead to unexpected errors, in particular concerning BLAS and LAPACK libraries.

For example, Ubuntu distribution provides three different BLAS libraries (these libraries may or may not be installed on your machine); however, only one of them is available via a soft link in the standard path `/usr/lib/`. See the appendix [A](#) for further information.

In any case, you may tell the `configure` script where it must select the correct BLAS and LAPACK libraries, like:

⁵This script has been designed to display colored information on a “green on black” terminal. Please use such a color configuration or, alternatively, you may definitely remove the colors by setting the `COLORS_ACTIVATED` variable to 0, at the beginning of the ‘`configure`’ script.


```
$ ./configure --blas=/usr/local/MY_LAPACK_PATH \
              --lapack=/usr/local/MY_LAPACK_PATH \
              --ldopt=-Wl,-rpath,/usr/local/MY_LAPACK_PATH
```

3.3.3 After configuring

After configuring⁶, you should have a look to the `Makefile.config`, just created under the `config` directory. This locale configuration file contains pathes for required third party libraries or tools:

- `COLORED_TERM` could be set to "no" for supressing colors in your terminal. This may be the case if you are using a "black on white" color scheme for your terminal.
- `HDF5` must be assigned to "yes" only if you want to use the HDF5 format in file IO. Usually, it should be left to "no".
- `HDF5_1_6` must be assigned to a directory containing the HDF5-1.6 library; it could be something like `/usr/lib`.
- `HDF5_F90_DIR` must be assigned to a directory containing the f90 interface of the HDF5 library (see also appendix B).
- `SHELL` concerns the shell used in the Makefile command.
- `NO_X11` should be set to "yes" if the X11 library is not found (under Windows, or some Mac OS X).
- `DARWIN` should be set to "yes" for Mac OS X.
- `X86_64` must be set to "yes" for a 64-bit linux.
- `ZLIB_DIR` must be assigned to a directory containing the ZLIB library; it could be something like `/usr/lib`.
- `ZLIB_INCL` must be assigned to a directory containing the headers of the ZLIB library; it could be something like `/usr/include`.
- `FREETYPE2` should be left to "yes" if you intend to use antialiased character display on X11 windows.
- `FREETYPE2_DIR` must be assigned to a directory containing the FREETYPE2 library; it could be something like `/usr/lib`.
- `FREETYPE2_INCL` must be assigned to a directory containing the headers of the FREETYPE2 library; it could be something like `/usr/include`.
- `READLINE` must be assigned to "yes" if the readline library is installed. As this library is required, it should be always "yes".
- `NEED_TERMCAP` must be assigned to "yes" if the termcap library must be explicetely added, via `-ltermcap`⁷.
- `X11_DIR` must be assigned to a directory containing the X11 library; `/usr/lib` is a usual place.
- `X11_INCL` must be assigned to a directory containing the X11 headers; `/usr/include` is a usual place.
- `QT4_QMAKE` must be assigned to "yes" if the development version of the Qt4 library is available on your system. Check that the `QT4DIR` environment variable contains the path (without 'lib') to the Qt4 library.
- `INSTALL_DIR` must be assigned to the location you want to have MUESLI installed.

⁶In case where the configure step fails you may try to (i) copy yourself `config/Makefile.config.in` to `Makefile.config` and (ii) set manually all variables.

⁷It seems that is the case for old linux distros (*e.g.* RedHat Fedora ≤ 7) only. If you don't know anything about this point, `NEED_TERMCAP` may be assign to "no": if, during the link, the loader complains about undefined references like `tgetnum`, then you must change to "yes" and perhaps install the `libtermcap-devel` package.

- DOCS_INSTALL_DIR must be assigned to the location you want to have the MUESLI documentation installed.
- MUESLI_CONFIG_DIR must be assigned to the location you want to have the `muesli-config` script installed. It must be a directory included in your `PATH` environment variable.
- BLAS_DIR must be assigned to a directory containing the BLAS libraries⁸, at least in the shared format.
- LAPACK_DIR must be assigned to a directory containing the LAPACK libraries⁸, at least in the shared format.
- ADD_LIB concerns additional libraries. For example, if the BLAS and LAPACK libraries have been compiled with the old g77 Fortran compiler, you have to add the `libg2c` library, by setting `-lg2c` to this variable.
- LD_OPT concerns additional linker flags. May be used to fix multiple definitions of the `xerbla_` symbol; in this case, input `-Wl,-zmuldefs` (for GNU linker).
- BLAS_LAPACK_VENDOR must be assigned to "yes" if you plan to use the versions of BLAS and LAPACK provided by the vendor.
- LIBSTDC++ contains a flag required to link the `libstdc++` library. It is compiler dependent.
- NEED_LRT specified whether the link command needs to contain the `'-lrt'` flag.

3.4 Compiling MUESLI

Under the directory named `<MUESLI>`, you should find a directory whose name comes from a compiler (*e.g.* `GNU_GFC` or `INTEL_IFC`); enter the directory of your choice.

Before compiling, some adjustments are needed in the three following makefiles:

```
./Makefile
./tests/Makefile
./tests/fgl/Makefile
```

Open each of these makefiles and check the following variables, which are located at the beginning of the file:

OPT_FLAGS	optimization flags
LIBSTDC++	(if present and necessary) the full path of the <code>libstdc++</code> library

Do the same thing for the compiler name, and other associated tools:

F90COMP (or F90C)	Fortran 90 compiler name
CC	C compiler name
CPLUSPLUS	C++ compiler name

Then, type:

```
$ make -s distclean
$ make -s
```

the latter command begins the compile process showing only the object file being processed. If you want a full output (verbose mode), type instead:

```
$ make MODE=verbose
```

Other options are possible. See a short explanation by typing:

⁸MUESLI will search a subdirectory, named *F90C_TAG* (either `GNU_GFC` or `INTEL_IFC`), depending on the compiler used (see also the next subsection). The libraries must be located inside this subdirectory.

```
$ make help
```

3.5 Running MUESLI tests (Optional)

Before installing MUESLI, you should verify that all is correct (**WARNING**: under Mac OS X, i.e. Darwin, you must install the library before running the tests – see next section).

Change to the `tests` directory and type:

```
$ make
```

Then, run the shell script:

```
$ ./run_all
```

will outputs all results on your screen. Be aware that you must type return between each module test execution. After a successful run, each module test displays a short message at the end.

You can also further investigate if results are good. So, type:

```
$ make check
```

this will produce as many `.diff` files as executable tests.

A similar procedure can be applied to verify the FGL part, in the `tests/fgl` directory. Usually, you should install the MUESLI library before running the graphic tests. However, you can run them, after defining some environment variables via the command:

```
$ source ./prepa_env
```

Be aware that many testing routines are interactive under *X11*, so that you have to resume manually, most of time by selecting something within the figure or clicking with the mouse; it is also possible to escape these interactive parts by typing the *ESC* key.

If you obtain black EPS files, this may certainly due to an old bugged PostScript library, especially that used in Ubuntu-14.04 from June 2014 (libspectre1). The cure is to define the environment variable `MFPLLOT_ADD_SHOWPAGE_TO_EPS` set to 1. If this doesn't solve the problem, don't hesitate to write an e-mail to the author.

All created EPS and PDF files should show consistent contents when opened. Note that under Ubuntu (at least up to the 18.10 version), *libpoppler* (which is the engine used to make the preview icon in the file manager) doesn't work well for native PDF shading feature. The preview icons may present a difference between EPS and PDF but once opened, the content should be the same. An additional remark concerns the PDF viewer: *okular* is recommended because *evince* doesn't support native PDF gradients nor native PDF transparency.

3.6 Installing MUESLI

```
$ make -s install
```

will copy all necessary files in the appropriate location (choosen during the configuration step, via the `--prefix` option of the configure script).

3.7 Verifying MUESLI installation

Change to the `tests/muesli-config` directory and type:

```
$ make
```

The latter `make` command should build the two following executable files: `test_muesli_config_fml` and `test_muesli_config_fgl`.

The first executable uses only the FML part, whereas the second one uses both the FML and the FGL part.

3.8 Matlab mex files installation (Optional)

Matlab mex files allow to read/write *.mbf files (MUESLI Binary Files) in gzipped/not gzipped state, and in little/big endian format.

Currently, these mex files can be obtained only with the GNU-gfortran compilers. MUESLI must be compiled before building the mex files.

Matlab releases R2011b to R2013a (*i. e.* resp. versions 7.13 to 8.1) have been tested. With some little chance, previous or newer releases should work.

First go to the `mex` directory (under GNU_GFC) and edit the `Makefile` script; choose appropriate values for the two following variables at the beginning of the file:

- `ARCH` must contains either `i686` or `x86_64` for a 32-bit or a 64-bit machine, respectively.
- `CFG` may contains either `debug` or `optim`; however, it should be left to `optim` in order to get performance under Matlab.
- `MUESLI_MEX_INSTALL_DIR` contains the directory containing your mex files; usually, this directory is added in the Matlab path via the ‘`addpath`’ command. (see the appropriate doc under Matlab)

Then, edit also the `mexopts.sh` file, but only the `CC` and `FC` variables which define the compilers which are to be used. Please note, however, that not all versions of GNU GCC are supported. As mentionned at the beginning of the file, see: http://www.mathworks.com/support/compilers/current_OPTIM/ but also http://www.mathworks.com/support/sysreq/previous_OPTIMs.html.

Then type:

```
$ make
$ make install
```

The directory `tests/data/` contains many *.mbf files for testing under Matlab.

A Use of BLAS and LAPACK in MUESLI

A.1 BLAS and LAPACK implementations

Many implementations of the BLAS (Basic Linear Algebra Subroutines) API exist:

- the so-called *Reference* implementation comes from <http://www.netlib.org/lapack>; it is very stable but not as fast as the following (optimized) ones.
- the *ATLAS* implementation (Automatically Tuned Linear Algebra Software); it is known to be faster than the reference one.
- *OpenBLAS* is an optimized BLAS based on GotoBLAS2; it contains many routines written in assembly language which in turn gives to this implementation a very good efficiency.

All the previous implementations are free. Others exist but are not free (INTEL-MKL, AMD-ACML, etc.).

The following table summarizes some characteristics of the three main BLAS implementations:

	Debian-based (e.g. Ubuntu)	RPM-based (e.g. Fedora)
<i>Reference</i>	package name: libblas-dev location: /usr/lib/libblas/ library name: libblas.{a so}	package name: blas-devel location: /usr/lib/ library name: libblas.{a so}
<i>ATLAS</i>	package name: libatlas-base-dev location: /usr/lib/atlas/ library name: libblas.{a so}	package name: atlas-devel location: /usr/lib/atlas/ library name: libatlas.{a so}
<i>OpenBLAS</i>	package name: libopenblas-dev location: /usr/lib/libopenblas/ library name: libopenblas.{a so}	package name: openblas-devel location: /usr/lib/ library name: libopenblas.{a so}

BLAS characteristics for some recent Linux distributions.

Concerning the LAPACK (Linear Algebra PACKage) library, which is a high-level set of routines calling the BLAS library, the situation is as follows:

	Debian-based (e.g. Ubuntu)	RPM-based (e.g. Fedora)
<i>Reference</i>	package name: liblapack-dev location: /usr/lib/lapack/ library name: liblapack.{a so}	package name: lapack-devel location: /usr/lib/ library name: liblapack.{a so}
<i>ATLAS</i>	package name: libatlas-base-dev location: /usr/lib/atlas/ library name: liblapack.{a so}	package name: atlas-devel location: /usr/lib/atlas/ library name: liblapack.{a so}
<i>OpenBLAS</i>	package name: libopenblas-dev location: /usr/lib/libopenblas/ library name: liblapack.{a so}	package name: openblas-devel location: /usr/lib/ library name: liblapack.{a so}

LAPACK characteristics for some recent Linux distributions.

Notes :

- Only the *Reference* implementation provides the whole set of LAPACK routines; indeed, ATLAS provides a subset of the LAPACK routines.
- The path /usr/lib/ may differ from one Linux distribution to the other. Typical pathes are also /usr/lib64/ or /usr/lib/x86_64-linux-gnu/.

A.2 Choosing or switching between implementations

Most of Linux distributions give access to a useful command allowing the user to choose the appropriate implementation:

```
$ update-alternatives --config libblas.so
```

or

```
$ update-alternatives --config libblas.so-<multiarch>
```

where `<multiarch>` is the multiarch path for your architecture (e.g. `x86_64-linux-gnu`). See <https://wiki.debian.org/DebianScience/LinearAlgebraLibraries>.

When choosing a BLAS/LAPACK implementation, take care that the combination should be coherent. For example, the combination: BLAS=*OpenBLAS* and LAPACK=*ATLAS* leads to a configure fail.

Usually, the MUESLI user chooses one BLAS/LAPACK implementation, and then configures and builds the MUESLI library. Even this simple way may lead to difficulties:

- the MUESLI `configure` script may fail to find the BLAS and LAPACK libraries because, as it can be seen in the previous table, the name and/or the location of these libraries may vary.
- again, the `configure` script may fail to test the link of a small program using BLAS and LAPACK (e.g. the *OpenBLAS* implementation is able to handle multithreaded programs, so an additional flag is required during the link step).
- the `muesli-config` script, which is very useful to compile and link the user programs, may be difficult to build.

In the case where the user wants to switch to another implementation, it is not necessary to clean and to compile again the whole MUESLI sources. Actually, the following steps should be followed:

1. launch again the `configure` script, while selecting new options to, for example, localize a different BLAS/LAPACK implementation;
2. re-execute the `make install` command, in order to create a new appropriate `muesli-config` script; check that the following commands return expected results:

```
$ muesli-config --blas_lib --blas-implem
$ muesli-config --lapack_lib --lapack_implem
```

3. (if applicable) rebuild just the shared libraries by doing:

```
$ rm -f lib/libfml.so lib/libfgl.so
$ make <options>
```

4. (optionally) check if the `test_muesli_config_fml` program can be rebuilt in the `tests/muesli-config` folder.

B Installing the HDF5 Fortran 90 interface (optional)

As of 2011, most linux distributions propose the 1.8 version of the HDF5 library. This latter version is not (yet) supported by MUESLI, so you have to install yourself the HDF5-1.6 library (follow the official instructions to build yourself this package).

Follow these steps:

1. get the whole original source code of the HDF5 library `hdf5-1.6.x.tar.gz` at:

<http://www.hdfgroup.org/ftp/HDF5/releases/>

2. put it in a saved directory

3. untar it with the `tar xvfz` command

4. enter the main directory created and type:

```
./configure --prefix=/opt      (or any other directory)
make
make install
```

5. configure now the fortran interface:

```
cd fortran
CC="gcc" CFLAGS="-O2" F9X="gfortran" FFLAGS="-O2" \
./configure --with-szlib=no --prefix=<install_path>
```

[replace `<install_path>` by an adequate path (*e.g.* it should be `"`pwd`/../../f90/GNU_GFC"` if you plan to use MUESLI with different compilers); add `-fPIC` both to the `CFLAGS` and `FFLAGS` variables above if you are using a 64 bits OS⁹]

6. build and install the fortran interface:

```
make lib
make install
```

7. then, do some post-install things: (**only** if `<install_path>` is specific to HDF5)

```
cd <install_path>
mv lib/* .
rm -f -R bin/ doc/ include/ lib/
```

8. lastly, create the shared library:

```
cd <install_path>
mkdir tmp
cd tmp
ar x ../libhdf5_fortran.a
gcc -shared -o ../libhdf5_fortran.so *.o
cd ..
rm -f -R tmp/
```

This is for the GNU-gfortran compiler – adapt step 5 above for another compiler (the `<install_path>` variable should be ended by one of the following tags: `GNU_GFC` or `INTEL_IFC`).

Note also that step 7 must be do only if the chosen `<install_path>` doesn't contains stuff from other libraries, as `/usr/local`.

⁹the addition of the `-fPIC` option prevents to obtain an error like:
 reallocation R_X86_64_32 against a local symbol can not be used when making a shared object during the creation of the shared library.