

Documentation of the project FA $\mu$ ST  
"Flexible Approximate Multi-Layer Sparse Transform"



Adrien Leman; Nicolas Bellot

October 5, 2016

# Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Installation on Unix platform</b>	<b>3</b>
2.1	Getting Started . . . . .	3
2.1.1	Required tools . . . . .	3
2.1.2	Required packages . . . . .	3
2.1.3	Basic build and installation . . . . .	4
2.2	Custom - Advanced Installation . . . . .	5
2.3	Build & Install using Code Block . . . . .	6
2.4	Build & Install using Xcode (for MAC OS) . . . . .	6
<b>3</b>	<b>Installation on Windows platform</b>	<b>8</b>
3.1	Getting Started . . . . .	8
3.1.1	Required tools . . . . .	8
3.1.2	Required packages . . . . .	9
3.1.3	Basic build and Installation . . . . .	9
3.2	Custom - Advanced Installation . . . . .	12
<b>4</b>	<b>QuickStart</b>	<b>13</b>
4.1	construct a faust . . . . .	14
4.1.1	construct a faust from a cell-array . . . . .	14
4.1.2	construct a faust from a saved one . . . . .	15
<b>5</b>	<b>Example</b>	<b>16</b>
5.1	Brain Sources Localization . . . . .	16

# Chapter 1

## Introduction

**Presentation:** FA $\mu$ ST is a C++ toolbox, useful to decompose a given dense matrix into a product of sparse matrices in order to reduce its computational complexity (both for storage and manipulation). In the figure 1.1, the matrix  $A$  represents the dense matrix and  $S_j$  correspond to the sparse matrices as  $A = \prod_{j=1}^J S_j$ .

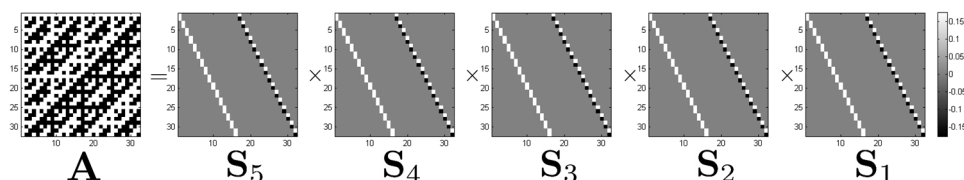


Figure 1.1: presentation

FA $\mu$ ST can be used to speed up iterative algorithms commonly used for solving high dimensional linear inverse problems. The algorithms implemented in the toolbox are described in details in Le Magoarou [1]. The FA $\mu$ ST toolbox is delivered with a Matlab wrapper. For more information on the FAuST Project, please visit the website of the project: <http://faust.gforge.inria.fr>.

**License:** Copyright (2016) Luc Le Magoarou, Remi Gribonval INRIA Rennes, FRANCE

<http://www.inria.fr/>

The FAuST Toolbox is distributed under the terms of the GNU Affero General Public License. This program is free software: you can redistribute it and/or modify it under the terms of the GNU Affero General Public License as published by the Free Software Foundation. This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Affero General Public License for more details. You should have received a copy of the GNU Affero General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

**Organization:** First chapter 2 explains how to install the library FA $\mu$ ST for UNIX platform and second chapter 3 corresponds to the Windows installation. The chapter 4 shows quickly how to use this library and finally an example is proposed in chapter 5.

## Chapter 2

# Installation on Unix platform

The FA $\mu$ ST project is based on an C++ library available for both UNIX and Windows environments. The proposed toolbox provides a Matlab wrapper. CMake has been choose to build the project FA $\mu$ ST because it is an open-source, cross-platform family of tools designed to build, test and package software.

This chapter presents the steps to install the FA $\mu$ ST tools on the Unix platform (both Linux and Mac OS). First section 2.1 presents the basic installation of FA $\mu$ ST, second section 2.2 corresponds to the advanced installation. Third 2.3 and fourth 2.4 sections explain how to used the common Integrated Development Environment (codeBlock and Xcode) in the build and installation process.

## 2.1 Getting Started

### 2.1.1 Required tools

- CMake : (tested with version 3.4.3 cf. website <https://cmake.org/>)
- Matlab : (tested with version 2014 and 2015) The use of the mex function in Matlab requires that you have a third-party compiler installed on your system. The latest version of Matlab (2016a in our case) only supports up to GCC 4.7 (see <http://fr.mathworks.com/support/compilers/R2016a/index.html?sec=glnxa64> for more detail). Please adjust your version of GCC compiler in order to run the installation properly. You must too have matlab in your environment PATH. If not please add.
- *OPTIONAL* The use of GPU process in FAUST project required the drivers for NVIDIA and CUDA install. You must have nvcc in your environment PATH. If not please add.

Please export following variable:

- CC with gcc (example: `"export CC='/usr/lib64/ccache/gcc'"`)
- CXX with g++ (example: `"export CXX='/usr/lib64/ccache/g++'"`)

### 2.1.2 Required packages

Here is a list of packages used in the FA $\mu$ ST project. The installation of this packages are automatically done. There are nothing to do. (see the directory `./externals`).

- Library eigen <http://eigen.tuxfamily.org>
- Library openBlas <http://www.openblas.net>

- Library xml2 <http://xmlsoft.org>
- Library matio <https://sourceforge.net/projects/matio>

### 2.1.3 Basic build and installation

When prerequisites listed in precedent sections are checked, the FA $\mu$ ST installation can be done :

- Download the FA $\mu$ ST package on the website : <http://faust.gforge.inria.fr/>
- Open a command terminal
- Place you in your local FA $\mu$ ST directory (NOTE: don't use any special character in your FAUST directory path, for example the character  $\mu$ )
- Type the following commands :

```
1 mkdir build
2 cd build
3 cmake ..
4 make
5 sudo make install % run with administrator privilege
```

When using the `cmake` command to generate the build system, Cmake performs a list of tests to determine the system configuration and manage the build system. If the configuration is correct then the build system is generated and written. In this case the three last lines of the console log of cmake command should be:

```
...
- Configuring done
- Generating done
- Build files have been written to: <./build>
```

The command `make` will compile the build files.

The command `sudo make install` will install the library and others components in default directory. For that, you must have administrator privilege because the library file `libfaust.a` is copied in your root path directory. If you don't have administrator privilege, you can make a local install using the following option (see 2.2):

```
cmake .. -DCMAKE_INSTALL_PREFIX="<Home/Local/Project/Output>"
```

Similarly, the Matlab wrapper is installed in the default Matlab Directory `/"HOMEDIR"/Documents/MATLAB/Faust`. But you can make a local install for the wrapper using the following option 2.2) :

```
cmake .. -DCMAKE_INSTALL_MATLAB_PREFIX="<Home/Local/Project/Output>"
```

## 2.2 Custom - Advanced Installation

The project FA $\mu$ ST can be configured with optional parameters, for example if you want to install FA $\mu$ ST in a different folder or to enable the parallel computing using multithread capacities provided by the OS. This build system can be parametrized using the Cmake Graphical User Interface, or the Cmake command line tools.

The Cmake Graphical User Interface `ccmake` allows you to selected option input. When using the `ccmake` command in your build directory, the Cmake GUI appears in the console (see fig. 2.1).

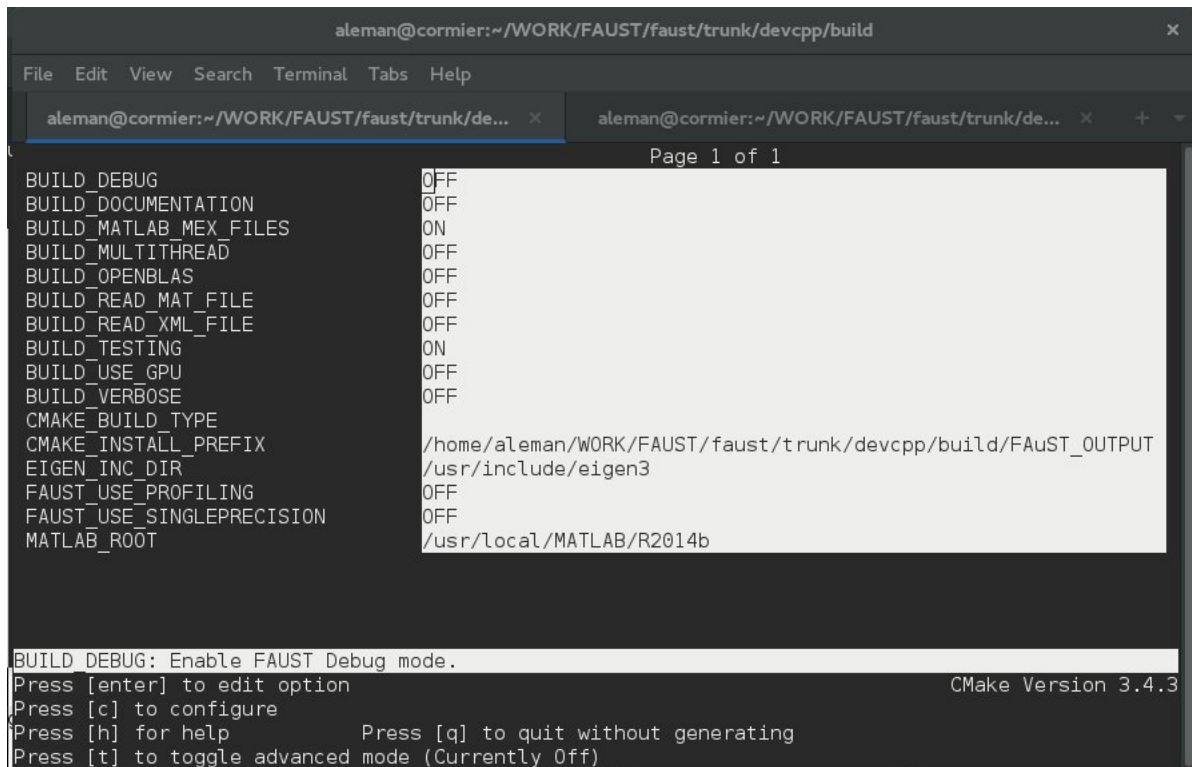


Figure 2.1: ccmake GUI

When scrolling on a value and pressing [enter], this value can be edited, the black underlaid row displays some information about the option and required path to create the build system. In the case of an option press [enter] to toggle the ON/OFF values. You can edit option by press [enter]. For example, press [enter] to edit option `CMAKE_INSTALL_PREFIX` to modify the install directory.

After choosing options for the build and setting the required fields, press [c] to configure. The configuration of the build system is checked again by Cmake, at the end of this check if the build settings are correct, you can press [g] in order to generate the build system.

Instead the `ccmake` GUI, an other possibility to configure and generate the project is to use the command line `cmake` which can take the option input. Here is the list of available options:

```
cmake .. -D < BUILD_NAME >=< value >
```

- `CMAKE_INSTALL_PREFIX` : Install directory

- CMAKE\_INSTALL\_MATLAB\_PREFIX : Install directory for the Matlab wrapper (mex-function, demo ...)
- BUILD\_TESTING : Enable the ctest option (default value is ON)
- BUILD\_DOCUMENTATION : Generating the doxygen documentation (default value is OFF)
- BUILD\_MULTITHREAD : Enable multithread with OpenMP Multithreading (default value is OFF)
- BUILD\_VERBOSE : Enable verbose option when compile (-v) (default value is OFF)
- BUILD\_DEBUG : Enable FAUST Debug mode (default value is OFF )
- BUILD\_USE\_GPU : Using both CPU and GPU process ( default value is OFF)
- BUILD\_MATLAB\_MEX\_FILES : Enable building Matlab MEX files (default value is ON)
- BUILD\_OPENBLAS : Using openBLAS for matrix and vector computations (default value is OFF )
- BUILD\_READ\_XML\_FILE : Using xml2 library to read xml files (default value is OFF)
- BUILD\_READ\_MAT\_FILE : Using matio library to read mat files (default value is OFF)

Following the selected option, the cmake installer automatically checks the dependent component (library OpenBlas, eigen, matio, libxml2).

## 2.3 Build & Install using Code Block

progress...

## 2.4 Build & Install using Xcode (for MAC OS)

The Faust tools can be generated using XCode on MAC OS. To do this, please follow this instructions :

- Download the FA $\mu$ ST package on the website : <http://faust.gforge.inria.fr/>
- Open a command terminal
- Place you in your local FA $\mu$ ST directory (NOTE: don't use any special character in your FAUST directory path, for example the character  $\mu$ )
- Type the following commands :

```
1 mkdir build
2 cd build
3 cmake .. -G "Xcode"
```

- Then, open your project FAUST.xcodeproj with Xcode and build the targets.

**NOTE:** You can generated the target using the terminal command `xcodebuild` :

```
1 mkdir build
2 cd build
3 cmake .. -G "Xcode"
4 %% list all target of the project
5 xcodebuild -list -project FAUST.xcodeproj
6 %% Build the targets
7 xcodebuild -configuration "Release" -target "ALL_BUILD" build
8 %% performs the "make install"
9 xcodebuild -configuration "Release" -target "install" build
```



## Chapter 3

# Installation on Windows platform

The FA $\mu$ ST project is based on an C++ library available for both UNIX and Windows environments. The proposed toolbox provides a Matlab wrapper. CMake has been choose to build the project FA $\mu$ ST because it is an open-source, cross-platform family of tools designed to build, test and package software.

This chapter presents the steps to install the FA $\mu$ ST tools on the Windows platform. First section 3.1 presents the basic installation of FA $\mu$ ST and second section 3.2 corresponds to the advanced installation.

### 3.1 Getting Started

#### 3.1.1 Required tools

The installation of the FA $\mu$ ST tools depends on other components to be installed in order to run properly.

1. **Install CMake** for building the FA $\mu$ ST tools. From <https://cmake.org/download/>, download Binary distributions correspond to your environment (in our case cmake-3.6.1-win64-x64.zip). The directory of binary must be add to the environment PATH of your system if you want to use the cmake command line tool.
2. **Install Matlab** if not already done (MATLAB R2015b in our case). The builder of the FA $\mu$ ST tools automatically checks your Matlab root directory if your `matlab.exe` is present in your environment Path and/or if your Matlab installation has been performed in a default directory like "C:/Program Files/MATLAB/<R2015b>/bin/matlab.exe" or "C:/Program Files (x86)/MATLAB/<R2015b>/bin/matlab.exe". In case of several versions of Matlab installed in your system, you can force the directory of your preferred version of Matlab using the following system variable :  
`MATLAB_EXE_DIR="C:/Program Files/MATLAB/<R2015b>/bin/matlab.exe"`

*Note for the case of using the compiler MinGW :* In Matlab, you must install MinGW version 4.9.2 from MATLAB using the **ADDON menu**. For more detail, please follow the instruction given in following link : [http://fr.mathworks.com/help/matlab/matlab\\_external/install-mingw-support-package.html](http://fr.mathworks.com/help/matlab/matlab_external/install-mingw-support-package.html). For that, you must have a id session for Mathwork. It is easy to create. Current this latest step, an environment variable called MW\_MINGW64\_LOC is automatically generated.

3. **Install C++ Compiler:** Both **Microsoft visual C++** and **MinGW "Minimalist GNU for Windows"** compiler have been tested. If you are more familiar with the graphical user interface, prefer the **Microsoft visual C++** compiler. Otherwise, if you

are friendly with Unix tools and command line terminal, preferd **MinGW "Minimalist GNU for Windows"** C++ compiler. The version of the C++ compiler must be coherent with the version of your Matlab version. In this documentation, the version of our C++ compiler corresponds to Matlab 2014 and 2015. If you use an other version of Matlab, please refer to the Mathworks website <http://fr.mathworks.com/support/compilers/<R20XXa>>.

For **Microsoft visual C++** installation :

- Download and install Microsoft Visual C++ 2013 professional from <https://www.microsoft.com/en-US/download/details.aspx?id=44916>

For **MinGW** installation :

- Download Mingw in <https://sourceforge.net/projects/mingw/files/latest/download?source=files>
- Launch install file and choose MINGW version 4.9.2 for mexFunction compatibility
- The directory of binary must be add to the environment PATH.
- *Note for make tool* : In a terminal command, type **make**. if it doesn't exist, please check if **make.exe** file is present in MINGW install directory. if not, you can copy and rename mingw32-make.exe to make.exe

### 3.1.2 Required packages

Here is a list of packages used in the FA $\mu$ ST project. Eigen and OpenBlas library are automatically installed : there are nothing to do (see the directory `./externals/win/`).

- **Eigen** is a C++ template library for linear algebra: matrices, vectors, numerical solvers, and related algorithms (see <http://eigen.tuxfamily.org>).
- **OpenBLAS** is an optimized BLAS library based on GotoBLAS2 1.13 BSD version. (see <http://www.openblas.net>). To install OpenBlas, refer to <https://github.com/xianyi/OpenBLAS/wiki/Installation-Guide>. You can directly download precompiled binary here <https://sourceforge.net/projects/openblas/files/v0.2.14/>

### 3.1.3 Basic build and Installation

When prerequisites listed in previous section 3.1.1 are checked, the FA $\mu$ ST installation can be done. First download the FA $\mu$ ST package on the website <http://faust.gforge.inria.fr/>.

Depending to your C++ compiler (MinGW or Visual studio), refer to the right part and follow the given instructions.

#### Using MinGW compiler:

- Open a command terminal
- Place you in your local FA $\mu$ ST directory (NOTE: don't use any special character in your FAUST directory path, for example the character  $\mu$ )
- Type the following commands :

```

1 mkdir build
2 cd build
3 cmake -G "MinGW Makefiles" ..
4 make
5 make install % run with administrator privilege

```

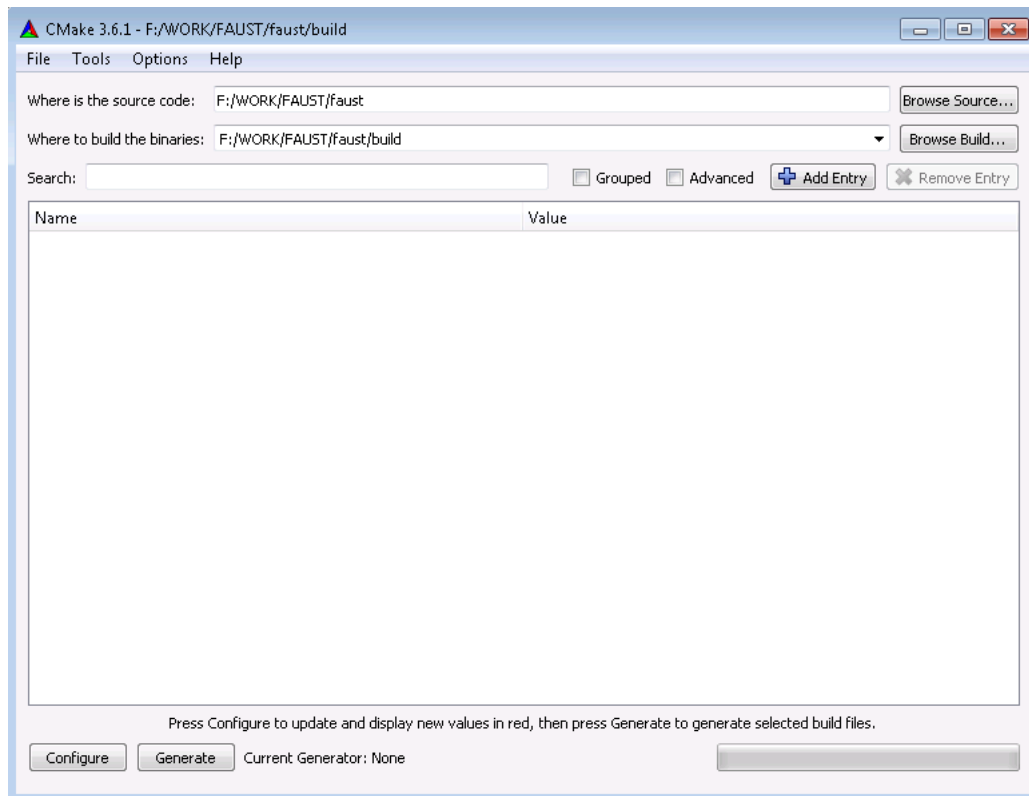
**NOTE:** You can generate the CodeBlocks project with the following command :

```
cmake -G "CodeBlocks - MinGW Makefiles" ..
```

### Using Visual Studio compiler:

In the case of **Microsoft Visual Studio 2013** compiler using the **Graphical Users Interfaces**:

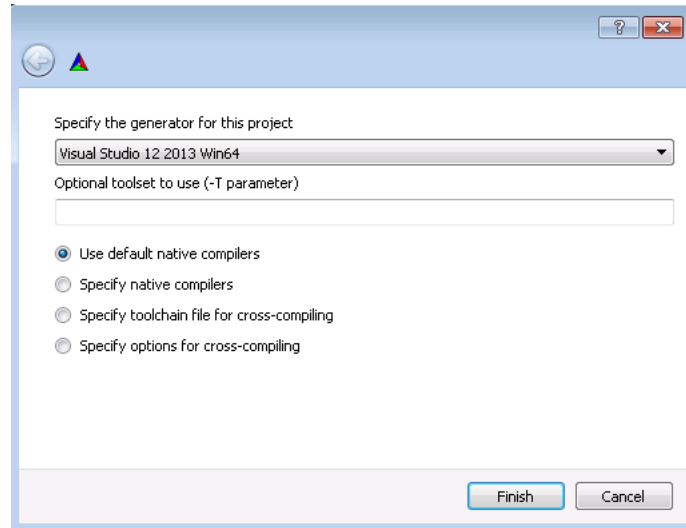
1. Open application `cmake-GUI.exe` from the program menu or from your cmake install binaries directory to launch the CMake configuration application:



**Figure 3.1:** cmake GUI

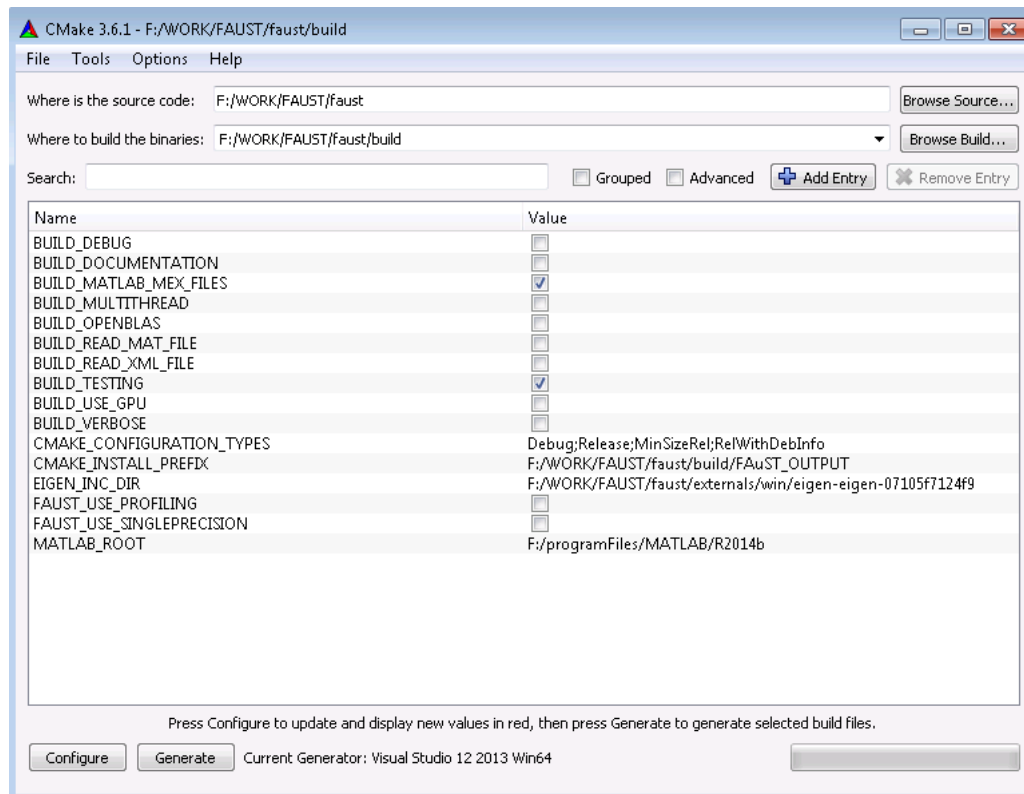
2. Set the "Where is the source code:" text box with the path of the directory where the source files are located (F:/WORK/FAUST/faust) and the "Where to build the binaries:" with the path of the directory where you want to build the library and executable files (F:/WORK/FAUST/faust/build). (see fig. 3.1).

When clicking for the first time on the [Configure] button, CMake will ask for the build tool you want to use. The build system type depends on the builder you want to use, in our case this is the Visual Studio X (X depending the version of Visual installed on the computer) chain tools. (see fig. 3.2).



**Figure 3.2:** cmake GUI

3. When pressing again the [Configure] button to configure the build system, CMake performs a list of tests to determine the system configuration and manage the build system. If the configuration is correct then no pop-up will appear during the tests and CMake finally shows the various options of the build underlaid in grey. In case of a configuration issue, a pop up window warns you about this issue indicating which test has failed, in this case the build option in the CMake application software will be underlaid in red. We will discuss in Section 3.2 what to do in such a case, but let us for the moment assume that everything ran smoothly. (see 3.3).



**Figure 3.3:** cmake GUI

4. Once the build system configured then generated, you have to actually build FAUST, using Visual Studio.
5. Open file "faust.sln" with visual studio
6. Click right on Target ALL\_BUILD and select generated
7. Click right on Target INSTALL and select generated
8. Click right on Target CTEST and select generated

**NOTE:** In the case of **Microsoft Visual Studio 2013** compiler using the command terminal :

- Open a command terminal
- Place you in your local FA $\mu$ ST directory (NOTE: don't use any special character in your FAUST directory path, for example the character  $\mu$ )
- Type the following commands :

```
1 mkdir build
2 cd build
3 cmake ..
4 cmake --build . --config "Release" --target "install"
```

## 3.2 Custom - Advanced Installation

progress...

## Chapter 4

# QuickStart

A Matlab wrapper is delivered with the FA $\mu$ ST C++ library. It provides a user friendly new class of matrix **Faust** efficient for the multiplication with matlab built-in dense matrix class.

As much as possible, a **Faust** object is handled as a normal matlab matrix, here is listing of matlab builtin function that can be applied to a Faust A :

```
1 >> % considering A is a Faust of size 10x3
2 >>
3 >> % get the size of the faust
4 >> [dim1,dim2] = size(A);
5 >>
6 >> % transpose a faust
7 >> A_trans = A';
8 >>
9 >> % multiplication by A
10 >> x1 = rand(3,3);
11 >> y1 = A*x1;
12 >>
13 >> % multiplication by A'
14 >> x2 = rand(10,5);
15 >> y2 = A'*x2;
16 >>
17 >>
18 >> % get the 2-norm (spectral norm) of the faust A
19 >> norm_A = norm(A); % equivalent to norm(A,2);
20 >>
21 >> % get the coefficient i,j and slicing for reading purpose
22 >> coeff=A(i,j);
23 >> col_2=A(:,2);
24 >> submatrix_A=A(3:5,2:3);
25 >> submatrix_A=A(2:end,3:end-1);
26 >> % Warning : A(i,j)=3 will not modify A, writing is not allowed
27 >>
28 >> % get the number of non-zeros coefficient
29 >> nz = nnz(A);
```

## 4.1 construct a faust

A **Faust** object can be constructed from several ways.

### 4.1.1 construct a faust from a cell-array

First, you can build a Faust from a cell-array of matlab matrix (sparse or dense) representing its factors.

The following example shows how to build a random faust of size 5x3 with 3 factors :

```
1 >> nb_factor = 3;
2 >> dim1 = 5;
3 >> dim2 = 3;
4 >>
5 >> % list of factors
6 >> factors = cell(nb_factor);
7 >>
8 >> factors{1}=rand(dim1,dim2); % first factor is rectangular
9 >> for i=2:nb_factor
10 >>     factors{i}=rand(dim2,dim2);
11 >> end
12 >>
13 >> % build the faust
14 >> A=Faust(factors);
```

An optional multiplying scalar argument can be taken into account :

```
1 >> % multiplicative scalar
2 >> lambda = 3.5
3 >> % build the faust
4 >> A=Faust(factors,lambda);
```

This functionality allows you to build a Faust from the factorization algorithm : **mexHierarchical\_fact** or **mexPalm4MSA** :

```
1 >> % factorization step
2 >> [lambda,factors]=mexHierarchical_fact(params);
3 >> % build the faust corresponding to the factorization
4 >> A=Faust(factors,lambda);
```

#### 4.1.2 construct a faust from a saved one

You can also build a Faust from a previously one which is saved into a mat file :

```
1 >> % save the faust A into the file faust.mat
2 >> [lambda,factors]=save(A,'faust.mat');
3 >> % create a new faust from the file faust.mat
4 >> A_loaded=Faust('faust.mat');
```

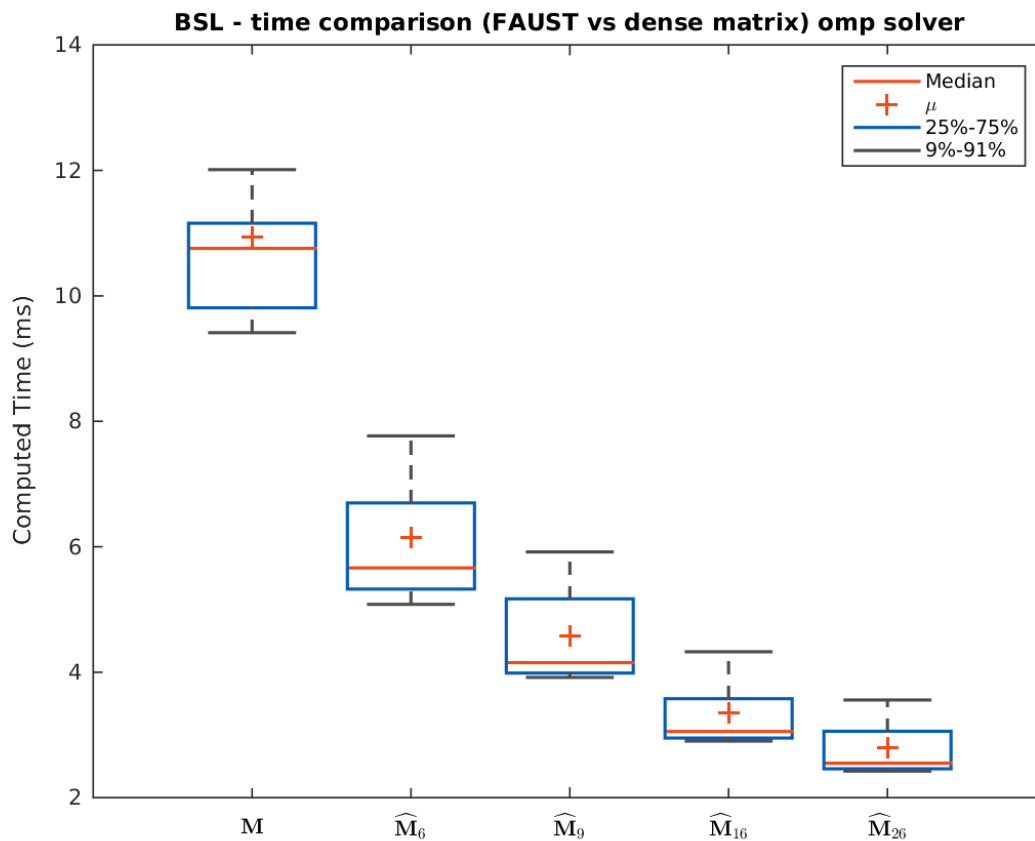


## Chapter 5

# Example

### 5.1 Brain Sources Localization

An experience of Brain Source Localization using several gain matrices including FAuSTs and several solvers is provided. You can execute the matlab script `demo/Brain_source_localization/BSL.m` to run this experiment and `demo/Brain_source_localization/Fig_BSL.m` to display the following pictures illustrating the speed-up using a Fajust.



# Bibliography

- [1] L. Le Magoarou and R. Gribonval. Flexible multilayer sparse approximations of matrices and applications. *IEEE Journal of Selected Topics in Signal Processing*, 10(4):688–700, June 2016.