Documentation of the project FAμST
"Flexible Approximate Multi-Layer Sparse Transform"

Adrien Leman; Nicolas Bellot

August 24, 2016

Contents

Chapter 1

Introduction

Presentation: FAμST 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). FAμST 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 FAμST 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`.

Organization: The chapter 2 presents the installation of the library FAμST, the chapter 3 shows quickly how to used this library and finally an example is proposed in chapter 4.

Chapter 2

Installation

The FAμST project is based on an C++ library available for both UNIX and Windows environments. CMake has been choose to build the project FAμST because it is an open-source, cross-platform family of tools designed to build, test and package software (cf. website `https://cmake.org/`).

First section 2.1 explains how to install the project FAμST for UNIX platform and second section 2.2 corresponds to the Windows installation.

2.1 Unix platform

2.1.1 Prerequisities for installation

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.

[OPTIONAL] The use of GPU process in FAUST project required the drivers for NVIDIA and CUDA install.

In your environment PATH, please add following components :

- matlab (example: "export PATH=/usr/local/bin:$PATH")

OPTIONAL nvcc for GPU compiler (example: "export PATH=/usr/local/cuda-X.X/bin:$PATH"

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 FAUST installation

When prerequisities listed in precedent section 2.1.1 are checked, the FAμST installation can be done :

- Download the FAμST package on the website : `http://faust.gforge.inria.fr/`
- Open a command terminal
- Place you in the FAμST directory, and type the following commands :

```
mkdir build
cd build
cmake ..
make
make install
```

2.1.3 Optional - Advanced Installer

The project FAUST can be configured with optional parameters. They can be activated in the principal CMakeList.txt file, or directly using the cmake command line

$cmake\ ..\ -D < BUILD_NAME >=< ON/OFF >$

- 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).

2.2 Windows platform

2.2.1 Prerequisities for installation

The installation of the FAμST project depends on other components to be installed in order to run properly.

1. **Install CMake** for building the FAUST project. In `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.
2. **Install 7-Zip** `http://www.7-zip.org/` . 7-Zip is a file archiver used for external library

3. **Install Matlab** if not already done (MATLAB R2015b in our case).

   Note for the case of the use of 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`. 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.

4. **Install C++ Compiler:** Both **Microsoft visual C++** and **MinGW "Minimalist GNU for Windows"** compiler have been tested. The version of this compilers must be coherent with the version of your Matlab version. Here is the compiler installation corresponding to Matlab 2014 and 2015. If you use an other version of Matlab, please refers to the Mathworks website `http://fr.mathworks.com/support/compilers/<R20XXa>`.

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

   - Download and install Microsoft .NET Framework 4
   - Download and install Microsoft SDK 7.1
   - Download and install Microsoft Visual C++ 2013 professional

   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 is present in MINGW install directory. if not, you can copy and rename mingw32-make.exe to make.exe

5. In your environment PATH, please verify and add following components :

   - matlab.exe (example: "C:\Program Files\MATLAB\R2015b\bin")
   - 7z.exe (example: "C:\prog\7-Zip")
   - cmake.exe (example: "C:\Users\ci\Documents\library\cmake-3.6.0-win64-x64\bin")
   - In case of the use of MinGW compiler : gcc.exe (example: "C:\mingw-w64\mingw64\bin")

2.2.2 FAUST installation

When prerequisities listed in precedent section 2.2.1 are checked, the FAμST installation can be done.

- Download the FAμST package on the website : `http://faust.gforge.inria.fr/`
- Open a command terminal
- Place you in the FAUST directory, and type the following commands :

  In the case of MinGW compiler :

```
mkdir build
cd build
cmake -G "MinGW Makefiles" ..
make
make install
```

In the case of Microsoft Visual Studio 2013 compiler :

```
mkdir build
cd build
cmake -G "Visual Studio 12 2013" ..
cmake --build . --config "Release" --target "install"
```

2.2.3 Optional - Advanced Installer

progress...

Chapter 3

QuickStart

A matlab wrapper is delivered with the FAUST 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 :

```
>> % considering A is a Faust of size 10x3
>>
>> % get the size of the faust
>> [dim1,dim2] = size(A);
>>
>> %  transpose a faust
>> A_trans = A';
>>
>> % multiplication by A
>> x1 = rand(3,3);
>> y1 = A*x1;
>>
>> % multiplication by A'
>> x2 = rand(10,5);
>> y2 = A'*x2;
>>
>>
>> % get the 2-norm (spectral norm) of the faust A
>> norm_A = norm(A); % equivalent to norm(A,2);
>>
>> % get the coefficient i,j and slicing for reading purpose
>> coeff=A(i,j);
>> col_2=A(:,2);
>> submatrix_A=A(3:5,2:3);
>> submatrix_A=A(2:end,3:end-1);
>> % Warning :  A(i,j)=3 will not modify A, writing is not allowed
>>
>> % get the number of non-zeros coefficient
>> nz = nnz(A);
```

3.1 construct a faust

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

3.1.1 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 :

```
>> nb_factor = 3;
>> dim1 = 5;
>> dim2 = 3;
>>
>> % list of factors
>> factors = cell(nb_factor);
>>
>> factors{1}=rand(dim1,dim2); % first factor is rectangular
>> for i=2:nb_factor
>>      factors{i}=rand(dim2,dim2);
>> end
>>
>> % build the faust
>> A=Faust(factors);
```

An optional multiplying scalar argument can be taken into account :

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

This functionality allows you to build a faust from the factorization algorithm : **mexHierarchical_fact** or **mexPalm4MSA** :

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

3.1.2 construct a faust from a saved one

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

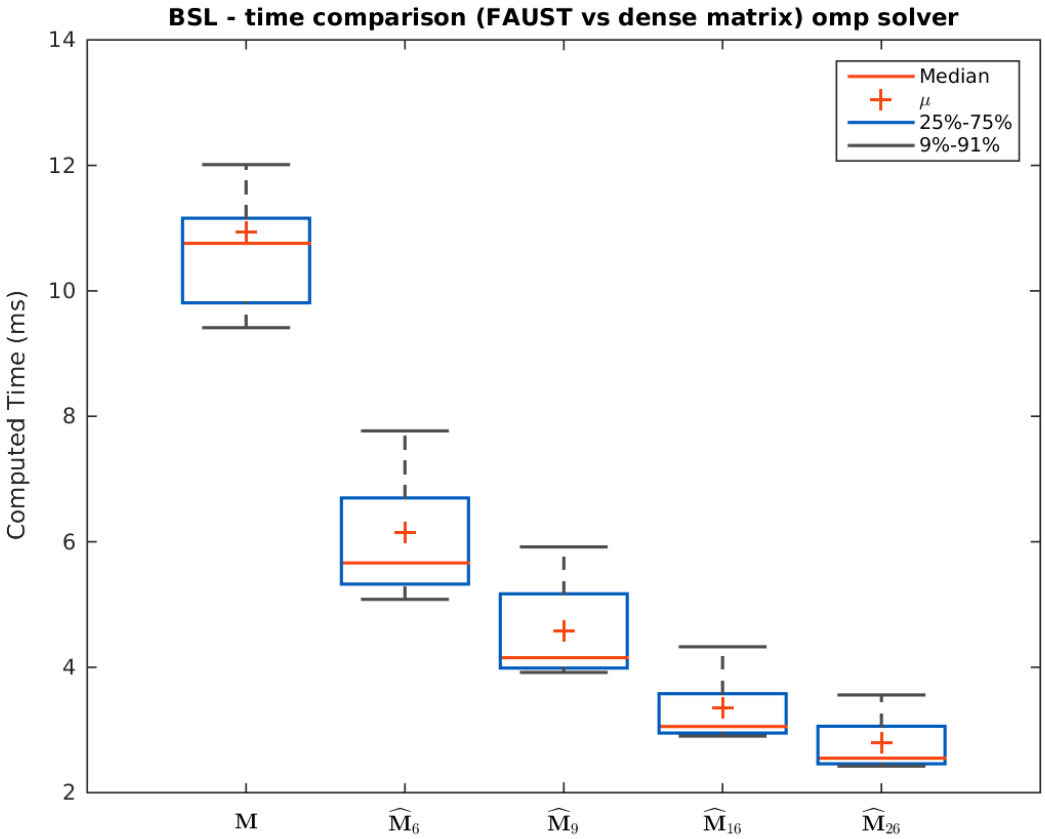
```
>> %  save the faust A into the file faust.mat
>> [lambda,factors]=save(A,'faust.mat');
>> % create a new faust from the file faust.mat
>> A_loaded=Faust('faust.mat');
```

Chapter 4

Example

4.1 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 Faμst.



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.