INTERNSHIP REPORT

MASTER 1 CSMI

June 26, 2015

FEEL++

THE FEELPP-BOOK

1 Introduction

Feel++ is a unified C++ implementation of Galerkin methods (finite and spectral element methods) in 1D, 2D And 3D to solve partial differential equations. This part of the report focuses on the documentation of the basic tools we need for feel++ and feelpp-book.To clearly describe each tool, l will make reference to a simple dummy mini project called operation. This mean project required me to write simple codes of basic arithmetic operations and then document the codes using these tools. Below are the files from the project :

i. main.cpp

ii. operation.cpp

iii. operation.h

Now we can discuss in details how to use each of these tools for a feel++ project making reference to a simple dummy mini project.

2 BASIC TOOLS FOR FEEL++

2.1 GITHUB

GitHub is a web-based Git repository hosting service, which offers all of the distributed revision control and source code management (SCM) functionality of Git as well as adding its own features.

At the heart of GitHub is Git, an open source project started by Linux creator Linus Torvalds . Matthew McCullough, a trainer at GitHub, explains that Git, like other version control systems, manages and stores revisions of projects. Although it ?s mostly used for code, McCullough says Git could be used to manage any other type of file, such as Word documents or Final Cut projects. Think of it as a filing system for every draft of a document.

GitHub offers free public repositories and collaborations for open-source software creation and charges small and large businesses for private repositories and collaboration. From a social perspective, GitHub allows users to change, adapt and improve software each project is called a "fork" in its public repositories. Users can work together in teams or alone. Public users create profiles which display repositories and public activity and help programmers find projects.

GitHub has become the industry-standard version control and publishing platform for web developers, but it's great for designers too.Some people have referred to GitHub as a publishing tool. While others might refer to it as a version control system and still other people might describe it as a collaboration platform. Well, actually GitHub is all of those things in one form or another.

If you are new to github,these are the basic things you need to know to get started with GitHub :

I. CREATING A PERSONAL ACCOUNT The first step to using GitHub is to set up a personal GitHub account.If you already have an account, visit github.com and sign in.

II. GItHub DOCUMENTATION

Like most hosted services, GitHub has a wealth of online documentation, that can help you get set up and it can serve as a great reference for you as you're learning. You'll notice, for example, that the first time you come to GitHub after setting up an account, you've got these four information blocks right in front of you, front and center, and they're designed

to help you set up Git. Create repositories, so, where you'll work on your projects. For existing repositories, so you can work on some other people's projects. And show off some of the social tools that are baked into GitHub as well. For quick refferences visit GitHub guides. This will give you the basic knowledge to get started into GitHub.

III. CREATING A REPO Once Git is installed, using it is just a matter of navigating to the directory that you want to manage, and then initializing it. And this process is called creating a Repository or repo for short. A single installation of Git, can track as many repos as you'd like. So, once it's installed you can just start using it.

IV. ADDING collaborators I want to talk about collaborating with GitHub. GitHub has some amazingly powerful collaboration tools and you can use them in a variety of interesting ways and workflows. To learn more about how collaborating works on GitHub, click here

V. ADDING files Now that we have a Git repo up and running, we need to add some files for it to track. Now remember, git doesn't just automatically track every file in the directory as soon as you put it there. It only tracks the files that you tell it to, a file can be either tracked or untracked. Now untracked files or a file that have been added since the last commit. Now since we don't have a commit as soon as we add some files to this folder automatically they are going to come in as untracked files. To add files to a repo, in the command line type :

```
git add file_Name
```

VI. MAKING A COMMIT After we have added some files to our empty Git repo,(we used the Git add Cmd to then stage those files). So, of course, the next step that once you have all the files staged that you want for your next commit, the next step is to go ahead, and do the commit itself. And that's taking that sort of snapshot of the project. At that moment in time.To add a commit message, in the command line, write :

```
git commit -m "write the commit message"
```

NOTE : A commit must be made each time a file is modified or updated.

VII. Checking differences Most of the time you're going to want to automatically add all of your modified files for your next commit. But there are times when you might want to check to see what's been changed before you make your commit. For example maybe you're working with a team member and you're not exactly sure what has changed. Or maybe you've been working for a couple of days and you're not sure whether you made a change to a file or not. Well to do that,in terminal write :

```
git diff
```

Inside the index file, it'll show you the changes in two ways.

VIII. Fetching and Pulling from Your Remotes To get data from your remote projects, you can run :

```
git fetch [remote-name]
```

The command goes out to that remote project and pulls down all the data from that remote project that you don ?t have yet. After you do this, you should have references to all the branches from that remote, which you can merge in or inspect at any time. If you clone a repository, the command automatically adds that remote repository under the name origin. So, git fetch origin fetches any new work that has been pushed to that server

since you cloned (or last fetched from) it. Itś important to note that the git fetch command pulls the data to your local repository , it doesnt́ automatically merge it with any of your work or modify what you are currently working on. You have to merge it manually into your work when you ?re ready. If you have a branch set up to track a remote branch, you can use the git pull command to automatically fetch and then merge a remote branch into your current branch. This may be an easier or more comfortable workflow for you ; and by default, the git clone command automatically sets up your local master branch to track the remote master branch (or whatever the default branch is called) on the server you cloned from. Running git pull generally fetches data from the server you originally cloned from and automatically tries to merge it into the code you ?re currently working on.

IX. Pushing to Your Remotes When you have your project at a point that you want to share, you have to push it upstream. The command for this is simple :

```
git push [remote-name] [branch-name].
```

If you want to push your master branch to your origin server (again, cloning generally sets up both of those names for you automatically), then you can run this to push any commits you ?ve done back up to the server : git push origin master This command works only if you cloned from a server to which you have write access and if nobody has pushed in the meantime. If you and someone else clone at the same time and they push upstream and then you push upstream, your push will rightly be rejected. You ?ll have to pull down their work first and incorporate it into yours before you ?ll be allowed to push. NOTE :

```
git remote rename file_name
```

and

```
git remote rm file_name
```

will rename and remove the given file respectively from a repo.

X. GIT PULL REQUEST Pull requests let you tell others about changes you've pushed to a repository on GitHub. Once a pull request is sent, interested parties can review the set of changes, discuss potential modifications, and even push follow-up commits if necessary.The request, printed to the standard output, summarizes the changes and indicates from where they can be pulled.

XI. FORK AND PULL The fork and pull model lets anyone fork an existing repository and push changes to their personal fork without requiring access be granted to the source repository. The changes must then be pulled into the source repository by the project maintainer. This model reduces the amount of friction for new contributors and is popular with open source projects because it allows people to work independently without upfront coordination. Pull requests are especially useful in the fork and pull model because they provide a way to notify project maintainers about changes in your fork. However, they're also useful in the shared repository model where they're used to initiate code review and general discussion about a set of changes before being merged into a mainline branch.

```
git request-pull' [-p] <start> <url> [<end>]
```

Generate a request asking your upstream project to pull changes into their tree. The request, printed to the standard output, summarizes the changes and indicates from where they can be pulled. The upstream project is expected to have the commit named by ¡start¿

and the output asks it to integrate the changes you made since that commit, up to the commit named by ¡end¿, by visiting the repository named by ¡url¿.

OPTIONS

```
-p : include patch text in the output.
<start>  : Commit to start at.
           This names a commit that is already in the upstream history.
<url>  : The repository URL to be pulled from
<end>
```

XII. ISSUES Issues are a great way to keep track of tasks, enhancements, and bugs for your projects. They are kind of like emails, except they can be shared and discussed with the rest of your team. Most software projects have a bug tracker of some kind. GitHubś tracker is called Issues, and has its own section in every repository.See how to create an issue here. A title and description describe what the issue is all about.

XIII. Notifications Notifications are GitHub ?s way to keep up to date with your Issues. You can use them to find out about new issues on repositories, or just to know when someone needs your input to move forward on an issue. There are two ways to receive notifications : via email, and via the web. You can configure how you receive notifications in your settings. If you plan on receiving a lot of notifications, we like to recommend that you receive web + email notifications for Participating and web notifications for Watching.

XIV. MILESTONES Milestones are used to track the progress of similar issues and pull requests as they're opened and closed over time. At a glance, you can easily see the progress of work in a milestone's lifetime. Once you have collected a lot of issues, you may find it hard to find the ones you care about. Milestones, labels, and assignees are great features to filter and categorize issues.

2.2 DOXYGEN

Doxygen is a tool for auto-generating API documentation, though you can also use it to generate documentation separate from an API. It can generate documentation from annotated C++ sources, but it also supports other popular programming languages such as C, Objective-C, PHP, Java, Python, IDL (Corba, Microsoft, and UNO/OpenOffice flavors), Fortran, VHDL, Tcl, and to some extent D.The main advantage of Doxygen is that you can write documentation directly within the comments of your source code. Doxygen searches for source code in your tree and generates API documentation for it.

It can generate an on-line documentation browser (in HTML) and/or an off-line reference manual in LaTeX from a set of documented source files. There is also support for generating output in RTF (MS-Word), PostScript, hyperlinked PDF, compressed HTML, and Unix man pages. The documentation is extracted directly from the sources, which makes it much easier to keep the documentation consistent with the source code. You can configure doxygen to extract the code structure from undocumented source files. This is very useful to quickly find your way in large source distributions.Doxygen can also visualize the relations between the various elements by means of include dependency graphs, inheritance diagrams, and collaboration diagrams, which are all generated automatically. You can also use doxygen for creating normal documentation. NOTE :

Doxygen is developed under Mac OS X and Linux, but is set-up to be highly portable. As a result, it runs on most other Unix flavors as well. Furthermore, executables for Windows are

available.For more informations about downloading and installing Doxygen, click here

So, what's the value to Doxygen ?

Some of Doxygen's output is extracted from the semantics of the source programs. Other parts are from special comments you embed in your code. The entire process requires a special configuration file that specifies exactly what you want to do. Just as your development directory usually has a makefile that controls the build process, you also have a Doxyfile that contains the options used by Doxygen. This is simply a free-form text file that has options variables and their values. Doxygen automatically generates an example Doxyfile for you on request.

The source comments can use any of three formats :

Special comments can mimic a Javadoc comment by using an extra asterisk in a multiline comment

```
(/** A special comment */).
```

You can also use an exclamation point instead of the second asterisk

```
(/*! Another special comment */).
```

Two or more single line comments (//) that have at least one extra slash or exclamation point following the comment marker also form a special comment. **Tags :** Special comments can contain Doxygen tags that let you specify particulars about your program. By default, comments refer to the next lexical element, for example :

```
   /***
     *@brief, which provides a brief description of the function.
     *@param, which identifies a single parameter to the function.
          (you may have more than one param tag).
     *@return, which describes the function's return value.
     *@date, which describes the date of documentation
     *@todo
     */
```

We can configure our cmake system to enable Doxygen to automatically generate an API documentation when we run make. To generate a documentation for our simple dummy mini project,we have to add to our CMakeLists.txt the following :

```
# add a target to generate API documentation with Doxygen
#set OFF if you don't want Doxygen to generate the documentation
option(BUILD_DOCUMENTATION
  "Use Doxygen to create the HTML based API documentation" ON)
if(BUILD_DOCUMENTATION)
find_package(Doxygen)
if(NOT DOXYGEN_FOUND)
message(FATAL_ERROR
"Doxygen is needed to build the documentation.
Please install it correctly")
 endif()
configure_file(
CMAKE_CURRENT_SOURCE_DIR/Doxyfile.in{CMAKE_CURRENT_ BINARY_DIR}/Doxyfile ONLY)
add_custom_target(doc
DOXYGEN_EXECUTABLE{CMAKE_CURRENT_BINARY_DIR}/Doxyfile
COMMENT "Generating API documentation with Doxygen"
VERBATIM)
  #Doxygen will be triggered every time we run make
  # IF you do NOT want the documentation to be generated
  # EVERY time you build the project
  # then leave out the 'ALL' keyword from the above command.
```

The base situation is like this : doc/CMakeLists.txt file checks for Doxygen and if found, adds a doc target to the build system. It also generates a doc/Doxyfile in the build folder, which allows cmake to substitute some variables such as version number, project name, source and destination folder etc. In your source tree is somewhere a Doxyfile, which you previously used to generate documentation by running doxygen in this directory. Rename this file to Doxyfile.in After another CMake run, you can type ?make doc ? to have CMake run Doxygen. To keep the source tree clean in out-of-source builds, the documentation is generated in the corresponding build directory.

To view the online html online documentation, we use the following command line in the source directory : open html/index.html To generate the pdf format of the documentation from the Latek files created, go to the Latex directory and then use the command line : pdflatex pdflatex fileName For detailed information about doxygen, please consult the online doxygen documentation

2.3 MAKEFILES

Compiling your source code files can be tedious, especially when you want to include several source files and have to type the compiling command everytime you want to do it.Makefiles are special format files that together with the make utility will help you to auto-magically build and manage your projects.

It is recommended to create a new directory and place all the files in there before applying the make commands.

Creating a Makefile

A Makefile typically starts with some variable definitions which are then followed by a set of target entries for building specific targets (typically .o and executable files in C and C++, and .class files in Java) or executing a set of command associated with a target label.

The basic Makefile

The basic makefile is composed of :

```
target: dependencies
[tab] system command
```

The following is the generic target entry form :

```
# comment
# (note: the <tab> in the command line is necessary for make to work)
target:  dependency1 dependency2 ...
      <tab> command

for example:
#
# target entry to build program executable from program and mylib
# object files
#
program: program.o mylib.o
    gcc -o program program.o mylib.o
```

Below is an example of a simple Makefile in c++ for files named operation.cpp,main.cpp and operation.h of a dummy mini project (operation) :(click here to see the dummy mini project source codes)

```
all: operation
operation: main.o operation.o
    g++ main.o operation.o -o operation
```

```
main.o: main.cpp
    g++ -c main.cpp

operation.o: operation.cpp
    g++ -c operation.cpp

clean:
    rm -rf *.o
```

The make utility

If you run

```
make
```

this program will look for a file named makefile in your directory, and then execute it.

If you have several makefiles, then you can execute them with the command :

```
make -f MyMakefile
```

There are several other switches to the make utility. For more info, man make.

Build Process

Compiler takes the source files and outputs object files Linker takes the object files and creates an executable Visit GNU make for more informations about makefile

2.4 CMAKE

CMake is a tool that helps simplify the build process for development projects across different platforms. CMake automates the generation of buildsystems such as Makefiles and Visual Studio project files. CMake is a 3rd party tool with its own documentation.

CMake is controlled by writing instructions in CMakeLists.txt files. Each directory in your project should have a CMakeLists.txt file. What is nice about CMake is that CMakeLists.txt files in a sub-directory inherit properties set in the parent directory, reducing the amount of code duplication.

creating a cmake file

Native CMake projects that are intended to be used by other projects (e.g. libraries, but also tools that could be useful as a build-utility, such as documentation generators, wrapper generators, etc.) should provide at a minimum a

```
<name>Config.cmake
```

or a

```
 <lower-name>-config.cmake  file.
```

This file can then be used by the

```
 find_package()
```

command in config-mode to provide information about include-directories, libraries and their dependencies, required compile-flags or locations of executables.

Building with CMake

Setting up a bunch of CMakeLists.txt files will not immediately allow you to build your project. CMake is just a cross platform wrapper around more traditional build systems. In the case of linux, this means make. A quick preprocessing step will convert your CMakeLists.txt description into a traditional make build system automatically. One nice and highly recommended feature of CMake is the ability to do out of source builds. In this way you can make all

your .o files, various temporary depend files, and even the binary executables without cluttering up your source tree. For documentation(using doxygen) purposes, we can configure our CMake file so that when we run the cmake and make commands, doxygen automatically geneates the documentation if desired. Lets take a look at a simple CMakeLists.txt file for our simple dummy project.

```
set(CMAKE_CXX_COMPILER /usr/bin/clag++ 3.6.0)
cmake_minimum_required(VERSION 2.8.9 FATAL_ERROR)
project(operation)

# Create a library called "operationlibo" which includes
#the source file "operation.cxx".
# The extension is already found.
#Any number of sources could be listed here.

option(USE_SHARED
            "use a shared library" OFF)

if(USE_SHARED)

add_library(operationlib SHARED operation.cpp operation.h)
# Make sure the compiler can find include files for our operation library
# when other libraries or executables link to operation

target_include_directories (
operationlib PUBLIC
CMAKE_CURRENT_SOURCE_DIR)else()add_library(operationlibSTATICoperation.cppoperation.h)target_include_d
endif()
#
# Add executable called "operatiom" that is built from the source files
# "operation.cxx"". The extensions are automatically found.
#

add_executable(operation main.cpp)

# Link the executable to the operation library.
#Since the operation library has
# public include directories we will use those link
#directories when building operation
#

target_link_libraries(operation LINK_PUBLIC operationlib)

# add a target to generate API documentation with Doxygen
#set OFF if you don't want doxygen to generate a documentation
option(BUILD_DOCUMENTATION
"Use Doxygen to create the HTML based API documentation" ON)
if(BUILD_DOCUMENTATION)
find_package(Doxygen)
 if(NOT DOXYGEN_FOUND)
  message(FATAL_ERROR
  "Doxygen is needed to build the documentation. Please install it correctly")
  endif()
  configure_file(
 CMAKE_CURRENT_SOURCE_DIR/Doxyfile.in{PROJECT_BINARY_DIR}/Doxyfile @ONLY)
  add_custom_target(doc ALL
  COMMAND DOXYGEN_EXECUTABLE{PROJECT_BINARY_DIR}/Doxyfile
   COMMENT "Generating API documentation with Doxygen" VERBATIM)
  #Doxygen will be triggered every time we run make
  # IF you do NOT want the documentation to be generated
  # EVERY time you build the project
```

```
  # then leave out the 'ALL' keyword from the above command.
endif()
```

To use out of source builds, create a build directory in your top-level folder (technically, this can be anywhere, but the top-level project folder seems to be a logical choice). Next, change into your build directory and run cmake pointing it to the directory of the top-level CMakeLists.txt. For example for our dummy project, i have my source codes in the src directory : cd src : takes us to the src directory ls : displays all that is in the src directory mkdir build : Builds the build directory cmake .. : Runs cmake pointing it to the src directory

```
MBP-de-winsy-2:stageM1 user$ cd src
MacBook-Pro-de-winsy-2:src user$ ls
operation.cpp   irma.png
Doxyfile.in        image.png        operation.h
CMakeLists.txt                main.cpp
MacBook-Pro-de-winsy-2:src user$ mkdir build
MacBook-Pro-de-winsy-2:src user$ ls
operation.cpp
Doxyfile.in        image.png        operation.h
Makefile        irma.png
CMakeLists.txt        build            main.cpp
MacBook-Pro-de-winsy-2:src user$ cd build
MacBook-Pro-de-winsy-2:build user$ cmake ..
```

Then run the make command.

Remember to be in your build directory and point cmake only to the directory containing the top-level CMakeLists.txt file, not the file itself. If all goes well, cmake will process your CMakeLists.txt files, find the location of all libraries and include paths and spew a bunch of configuration information including a traditional Makefile in your build directory. You are now ready to build using the traditional make system. Run make in your build directory to compile and link everything. CMake even tosses in some nice colors, progress bars, and suppresses a bunch of gcc output. If you want verbose output, you can type VERBOSE=1 make (helpful if something goes wrong)

If you modify code in your source directory, including even a CMakeLists.txt file and re-run make in the build directory, make and cmake will recompile and rebuild necessary changes. If you are only making changes in a subdirectory, you can simply run make in the corresponding subdirectory in the build tree to process updates. An initial source of confusion with out of source builds is that you basically have two copies of your source tree, one with actual source code, and one with Makefiles and binary executables (in the build tree). It is probably best to keep two windows open with one in the build tree for making and running your programs, and one window in the source tree for modifying source files. One nice thing about out of source builds is that cleaning up object files, makedepend files, binaries, and other miscellaneous build cruft can be done by simply deleting the entire build directory because there is no source. You can also use make clean to clean up the actual object and binary files, but when you are planning to tar up your source or distribute your code to the masses, you can simply do

```
MBP-de-winsy-2:stageM1 user$ cd src
MacBook-Pro-de-winsy-2:src$ rm -rf build
MacBook-Pro-de-winsy-2:src$ ls
operation.cpp    operation.h
Doxyfile.in        image.png
Makefile        irma.png
CMakeLists.txt    main.cpp
```

to clean up and package your code.

This only touches the surface of what CMake can do. Check out the CMake Wiki for more info.

3 REFERENCES

1 GITHUB GUIDES

2 GIT HELP

3 GIT TUTORIALS

4 WIKIPEDIA

5 DOXYGEN MANUAL

6 CMAKE COMMANDS

7 MAKEFILE TUTORIAL

8 CMAKE TUTORIAL