2 Installation and Getting Started

2.1 Prerequisites

Smooth Emulator programs should run on UNIX, Mac OS or Linux, but is not supported for Windows OS. The software is largely written in C++, though there are some python scripts included for graphing results. In addition to a C++ compiler (CMake files assume you can accommodate the C++ 20 standard), the user needs the following software installed.

- git

- CMake

- Eigen3 (Linear Algebra Package)

- Python/Matplotlib (only for generating plots)

CMake is an open-source, cross-platform build system that helps automate the process of compiling and linking for software projects. Hopefully, CMake will perform the needed gymnastics to find the Eigen3 installation. To install CMake, either visit the CMake website (https://cmake.org/), or use the system's package manager for the specific system. For example, on Mac OS, if one uses *homebrew* as a package manager, the command is

```
% brew install cmake
```

Eigen is a C++ template library for vector and matrix math, i.e. linear algebra. The user can visit the Eigen website (https://eigen.tuxfamily.org/dox/), or use their system's package manager. For example on Mac OS with *homebrew*,

```
% brew install eigen
```

2.2 Downloading the Repository and Copying Files to Working Directories

The software requires downloading the BAND framework software repository into some directory. Should that be in the User's home directory, the User might enter

```
/Users/CarlosSmith% git clone https://github.com/bandframework/bandframework.git
```

Within the repository, there will be a directory
/Users/CarlosSmith/bandframework/software/SmoothEmulator/. All the relevant functionality of *Smooth Emulator* is contained within this directory. Throughout the manual the phrase ${SMOOTH_HOME} will refer to this directory.

The User should create a personal directory from which the User would perform a project. This is easiest accomplished by copying the analysis template directory from the *Smooth* distribution. For example

```
% cp -r ${SMOOTH_HOME}/AnalysisTemplate /Users/CarlosSmith/MY_ANALYSIS
```

The User may choose any other name besides `MY_ANALYSIS`. Hence forth, `${MY_ANALYSIS}` is a name chosen by the User, and will refer to this directory, including the path, from which the User will perform most of the analysis. Thus, the User may wish to have several such directories, located according to the User's preference. Any time a new analysis is performed with new parameters, and if the User wishes to save the previous analysis, a new `${MY_ANALYSIS}` directory should be created.

The User should create a second personal directory from which they will store their main programs and executables. This should be copied from the repository as well.

```
% cp -r ${SMOOTH_HOME}/LocalTemplate /Users/CarlosSmith/MY_LOCAL
```

The User may choose any other name besides `MY_LOCAL`, and henceforth, `${MY_LOCAL}` will refer to this directory, including its path. `${MY_LOCAL}/software/` contains the source code and CMake files for compiling the main programs. Executables will be stored in `${MY_LOCAL}/bin/`. Because the User might wish to edit the main programs, or to add similar programs, this provides the User a space to make such edits, all while leaving an original copy of the directory in the main repository. The User may find it convenient to add `${MY_LOCAL}/bin/` to their path.

The main software source codes and libraries, i.e. all but the main programs, which tend to be quite short, remain in the repository. If the User believes that they might be editing such files, they might want to copy the entire repository to a location without any fear of having a `git pull` command overwriting their work. The philosophy behind this structure is that it allows the User to devise and build simple programs making calls to *Smooth Emulator* libraries without altering the source code and libraries which are kept in the `${SMOOTH_HOME}/` directory structure. If the User wished to make modifications to this code, it would make sense to copy the `${SMOOTH_HOME}/` directory to a new location.

The directory structure is Fig. 2.1. As stated above, for the remainder of this manual, `${SMOOTH_HOME}/`, `${MY_LOCAL}/` and `${MY_ANALYSIS}/` will be used to denote the location of these directories.

2.3   Compiling the Software

First, change into software directories, then create the makefiles with cmake, then compile them.

```
% cd ${SMOOTH_HOME}/software
% ${SMOOTH_HOME}/software% cmake .
% ${SMOOTH_HOME}/software% make
```

There seems to be a common problem that `cmake` misreports the path of the `Eigen` installation. If the User should get an error stating that the Eigen header files cannot be found, the User can set the environmental variable,

```
% export EIGEN3_INCLUDE_DIR=/usr/local/include/eigen3
```
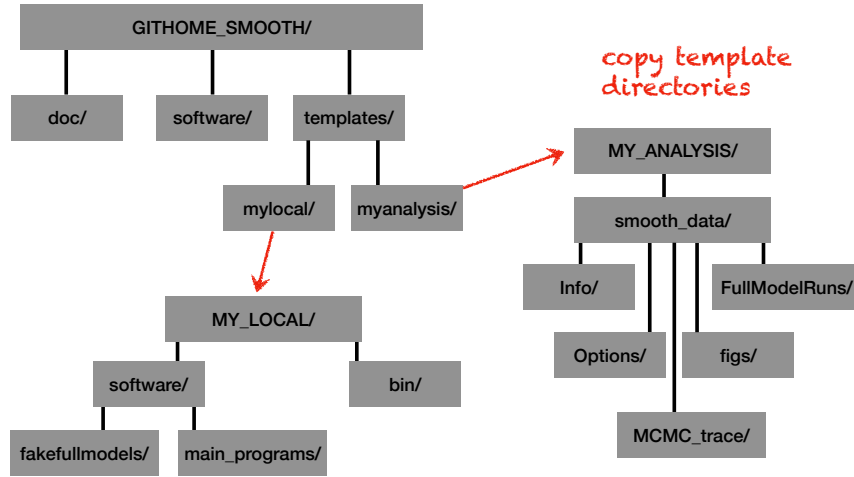
GITHOME_SMOOTH/

doc/   software/   templates/

mylocal/   myanalysis/

copy template directories

MY_ANALYSIS/

smooth_data/

Info/   FullModelRuns/

Options/   figs/

MCMC_trace/

MY_LOCAL/

software/   bin/

fakefullmodels/   main_programs/

Figure 2.1: The directory structure: The User clones the repository into some location, which will be referred to as ${SMOOTH_HOME}. The User copies the `LocalTemplate` and `AnalysisTemplate` directories to two new locations, ${MY_LOCAL} and ${MY_ANALYSIS}. The User will likely make multiple copies of the analysis directory for different projects or for different analyses with different options. The main source code and compiled libraries will be built and kept within the ${GITHOME_SMOOTH}/ tree. Source code for simple main programs can be found in ${MY_LOCAL}/software/ and executables are stored in ${MY_LOCAL}/bin/. The software is designed to be run from the command line in the ${MY_ANALYSIS} directory. Data used and created by *Smooth Emulator* software is stored in ${MY_ANALYSIS}/smooth_data/.

The final arguments may need to be changed depending on the User's location of the packages. If the User wishes to choose a specific C++ compiler, the `cmake` command should be replaced with `cmake -D CMAKE_CXX_COMPILER=g++-11`, or whichever compiler is preferred.

At this point all the libraries are built, but this does not include the main programs. The main programs are short, and are meant to serve as examples which the User might copy and edit at will. Before compiling, one needs to set an environmental variable so that the compilation can find the libraries needed for compilation.

```
% export SMOOTH_HOME=/Users/CarlosSmith/bandframework/software/SmoothEmulator
```

The first part of the path needs to be replaced with the location of the `bandframework` git repository. If one prefers not to set an environmental variable, one can instead edit ${MY_LOCAL}/software/CMakeLists.txt, and replace the line where the variable ${SMOOTH_HOME} is set to the shell's environmental variable with one where it is set explicitly.

Below, this illustrates how to compile the programs used for generating training points with Simplex and for tuning the emulator with *Smooth Emulator*:

```
% cd ${MY_LOCAL}/software
```

```
% ${MY_LOCAL}/software% cmake .
% ${MY_LOCAL}/software% make
  .
  .
```

The `cmake` command will also recompile the main libraries in `${SMOOTH_HOME}/software/` if necessary. Several source codes for main programs can be found in `${MY_LOCAL}/software/MainPrograms/`. If you build your own main programs (probably using these as examples), you can edit the `CMakeList.txt` file in `${MY_LOCAL}/software/MainPrograms/`, using the existing entries as an example. The executables should appear in `${MY_LOCAL}/bin/`.

2.4 The `${MY_ANALYSIS/smooth_data}` Directory

Within `${MY_ANALYSIS}/smooth_data` there are several sub-directories (assuming it was created from the template). The first is `smooth_data/Info/`. Information about the model parameters, and their priors is stored in `smooth_data/Info/prior_info.txt`, and information about the observables is stored in `smooth_data/Info/observable_info.txt`. The file `smooth_data/Info/experimental_info.txt` stores information about the measurements for each observable, and is not needed for building the emulator, but is used by the MCMC investigation of the posterior. The `smooth_data/Options/` directory stores text files used to set options for the training-point optimization, the emulator and the MCMC programs. The `smooth_data/FullModelRuns` data has subdirectories `run0/`, `run1/` ⋯. Each subdirectory stores information describing a full model run. The file `smooth_data/FullModelRuns/runX/` stores the model parameters used for run "X". This file can be provided by the User, or it can be created by the training-point optimization codes. Once those files are created, it is the User's responsibility to run the full model and record the observables in `smooth_data/FullModelRuns/runX/obs.txt`. Given those observables, the User can then run the emulator software. If the User runs the MCMC software, the MCMC traces are stored as text files in `smooth_data/MCMC_Trace/`. Examples of PYTHON scripts for graphing using MATPLOTLIB can be found in `smooth_data/figs/`.

If the User wishes to perform a different analysis, perhaps using a different set of observables, it is recommended to copy `MY_ANALYSIS` directory and perform the new analysis in that location, so that the previous analysis data is not over-written.

2.5 Setting up "Info" files

The `smooth_data/Info/` directory contains three files which the User must create and edit before running any of the *Smooth Emulator* software. Each of these files is a simple ascii file. The User may add comments to any of the "Info" files by placing the `#` symbol at the beginning of the line. Only the first file, `smooth_data/Info/prior_info.txt` is required for the training-point optimization software. That file and `smooth_data/Info/observable_info.txt` are both necessary for emulation. All three files, including `smooth_data/Info/experimental_info.txt`, which lists the experimental measurements of the observables used to constrain the model-parameter space, are required for running the MCMC software.

2.5.1 `smooth_data/Info/prior_info.txt`

This file defines the model parameters and their priors. The format of the simple ascii file is:

```
parameter_name_1  prior_type_1 xmin_1/centroid_1  xmax_1/Rgauss_1  sensivity_scale_1
parameter_name_2  prior_type_2 xmin_2/centroid_2  xmin_2/Rgauss_2  sensivity_scale_2
.
.
parameter_name_N  prior_type_N xmin_N/centroid_N  xmin_N/Rgauss_N  sensivity_scale_N
```

For each of the N model parameters the file devotes one line. The first string (no spaces) is the parameter name. The second string describes the prior type, and must be either `gaussian` or `uniform`. The next to entries describe the range of the prior. For uniform priors, the numbers represent the boundaries of the distribution. For a gaussian prior, the first entry represents the centroid of the gaussian and the next entry represents the gaussian's width. The last entry is the sensitivity scale. If all the model parameters are expected to contribute similarly, all the values should be set to unity. However, if there are some parameters for which one expects to contribute in a more linear fashion (higher orders are less important) one can enter a smaller number. None of the sensitivity scales should exceed unity. For less impactful model parameters, where the User would be satisfied with a nearly linear fit, one enters a number below unity. In practice, this is the same as assuming that the convergence parameter is larger than those with sensitivity scales set to unity, inversely with the sensitivity scale. For example, if the convergence parameter, $\mathbf{\Lambda}$, is set to 2.5, setting the sensitivity scale to 0.5 for some model parameter is equivalent to assuming that the effective value of $\mathbf{\Lambda}$ for that model parameter is 5.0.

2.5.2 `smooth_data/Info/observable_info.txt`

This file describes the observables. It provides only the $\mathbf{n}$ observable names and the point-by-point uncertainties.

```
observable_name_1    ALPHA_1
observable_name_2    ALPHA_1
.
.
observable_name_n    ALPHA_N
```

The measurement uncertainties, or the uncertainties due to theoretical systematic error, are not provided here – they are provided in `smooth_data/Info/experimental_info.txt`, as they do not come into play until one is comparing the model to measurement or observation. The parameter `ALPHA_i` describes only the point-by-point uncertainty. This class of uncertainty arises when the full model has some sort of noisy contribution, i.e. one where if the model were re-run with identical model parameters the observable values would vary. Examples of such models would be simulations of nuclear collisions using a finite number of events. If the observable is expected to vary by some representative value $\boldsymbol{\sigma_A}$ throughout the prior, `ALPHA_i` is the fraction of $\boldsymbol{\sigma_A}$ describing the additional variation due to noise. For example if one believed that a mean transverse momentum measurement

from a high-energy collider measurement might vary by 100 MeV throughout the prior, and if the noisy contribution was about 1 MeV, then one would set `ALPHA_i` to 0.01. If the noise comes from $N_{\mathbf{sample}}$ measurements, then one would probably simply set ALPHA_i to $1/\sqrt{N_{\mathbf{sample}}}$. This parameter ensures that should the User choose two training points that are nearly adjacent, the emulator will not attempt to exactly reproduce both points, which would lead to wildly varying, non-smooth, behavior.

2.5.3 `smooth_data/Info/observable_info.txt`

The third file describes the measurement and their uncertainties. It is only used when performing the MCMC fit, which occurs after the emulator is trained. The format is

```
observable_name_1    measurement_1  exp_uncertainty_1   theory_uncertainty_1
observable_name_2    measurement_2  exp_uncertainty_2   theory_uncertainty_2
.
.
observable_name_N    measurement_N  exp_uncertainty_N   theory_uncertainty_N
```

The observable names must be identical, or a subset, of those from `smooth_data/Info/observable_info.txt`. The measurement values and uncertainties are typically those taken from the experimental publication. The last entry in each line is the systematical theoretical uncertainty, i.e. that arising from the physics missing from the model. For example, if the model is missing a certain bit of physics, or if that physics is described through an imperfect approximation, then one would expect the model calculations to vary from the experimental values even if the model parameters were perfectly chosen. In practice, the two uncertainties contribute as a single uncertainty, where the two are added in quadrature.