Manual for the FCS/FCCS simulator software (`diffusion4`)

Dr. Jan W. Krieger <jan@jkrieger.de>

October 22, 2015

Contents

Chapter 1

Availability and Compilation

1.1 Availability

This software is available on GitHub:

`https://github.com/jkriege2/FCSSimulator`

In addition this simulator is also a part (based on the same repository) of the FCS/FCCS data evaluation software QUICKFIT 3.0 (`https://github.com/jkriege2/QuickFit3` and `http://www.dkfz.de/Macromol/quickfit/`), which also offers an integrated editor-GUI for the simulator configuration scripts.

1.2 Download and Setup

DIFFUSION4 is a C++ program, so you will need a working C++ compiler (e.g. the GCC in version $\geq$4.7, for Windows you can use MINGW-BUILDS from `http://sourceforge.net/projects/mingwbuilds/`). The program only relies on the GNU scientific library (GSL), available from `http://www.gnu.org/software/gsl/`, but also integrated in the repository (see below). In addition you will need a working BASH-shell (on Windows use MSYS available from `http://www.mingw.org/wiki/msys`) and GIT to access the repository (if you don't want to download the code manually). In general this program should compile on WINDOWS, LINUX and MACOS X.

In order to install DIFFUSION4, follow these steps:

1. check out the git repository:

```
$ git clone --recursive "https://github.com/jkriege2/FCSSimulator.git"
```

2. ensure that all submodules are checked out:

```
$ cd FCSSimulator
$ git submodule update --init --remote --recursive --force
```

3. if GSL is not available on your system, we'll have to build it now. The repository contains a BASH-script for this purpose:

```
$ cd extlibs
$ . build_dependencies.sh
$ cd ..
```

This script will build a local version of GSL. It is not installed in the system, but only in the directory `./extlibs/gsl/`. It will ask several questions: Generally you should not keep the build-directories (`n`), use as many processor, as you like (e.g. `2` for a dual-core, or `4` for a quad-core machine), you the best optimizations for your local machine, if the programm will only be used locally (twice `y`, or for safe-settings `y` and then `n`). Finally answer `y` when asked whether GSL should be compiled.

1.3 Compilation

the rest of the compilation is done using a `Makefile` in the base directory:

```
$ make
```

1.4 Running the Software DIFFUSION4

After the compilation, an command-line based executable `diffusion4` is created in the base directory. This is the simulator, which can then be started as follows:

```
$ diffusion4 [--spectra SPECTRADIRECTORY] CONFIG_SCRIPT.ini
```

Here `CONFIG_SCRIPT.ini` is a configuration file that tells the simulator what to do and the optional `-spectra SPECTRADIRECTORY` can be used to provide another directory with absorption and emission spectra. A version of this directory is available in the repository under `./spectra/`, which is also loaded automatically, if the option `--spectra` is not given. DIFFUSION4 reads absorption and emission spectra for its simulation from this directory. Note that DIFFUSION4 will not work, if a `spectra`-directory is not provided!

1.5 Running DIFFUSION4 from a Makefile

Often it is useful to run DIFFUSION4 from a Makefile, if several simulations should be done (in parallel) on a computer. here is an example Makefile for this task (see the repository directory `./example_configs/` for further examples:

```
SCRIPTS= gyuri_isat_multigfp0.ini\
         gyuri_isat_multigfp1.ini

SHELL = sh

SCRIPTS_TARGET = $(subst .ini,.target,$(SCRIPTS))

TERMINAL_COMMAND=

ifeq ($(findstring Msys,$(OS)),Msys)
EXE_SUFFIX=.exe
TERMINAL_COMMAND=
else
EXE_SUFFIX=
#TERMINAL_COMMAND=konsole -e
#TERMINAL_COMMAND=x-terminal-emulator -e
endif

all: ${SCRIPTS_TARGET}

%.target: %.ini
	@echo -e "random wait before starting $< ..."
```

```
	@bash random_sleep.sh
	@bash -c "sleep $$[ ( $$RANDOM % 10 ) + 1 ]s"
	@echo -e "starting on $< ..."
	${TERMINAL_COMMAND} ./diffusion4${EXE_SUFFIX} $< > $<.log
	@echo -e "work on $< DONE!"
```

You can run such a Makefile with the command:

```
$ make -f Makefile -j4
```

where `4` specifies the number of processors to use. This makefile also uses the BASH-script `random_sleep.sh`, which waits a random number of seconds (up to 10 or 20) before running the simulation. This ensures that the random number generator of each simulation is initialized with a different seed (the seed is taken from the system time!).

Chapter 2

Introduction to DIFFUSION4

Chapter 3

Reference of the DIFFUSION4-Modules

3.1 Fluorophore Dynamics Modules

3.1.1 Fluorescence Detection Modules

Chapter 4

Extending DIFFUSION4

4.1 Introduction and Registration

You can extend DIFFUSION4 with your own modules by implementing classes of the type `FluorophorDynamics` for fluorophore dynamics modules, or `FluorescenceMeasurement` for fluorescence detection modules. these virtual base-classes each provide virtual functions that you have to implement in order to give you class a function. Finally you will have to register your new class in `main.cpp`:

- `FluorophorDynamics`-classes have to be added near the text

```
////////////////////////////////////////////////
// add you custom dynamics classes here
////////////////////////////////////////////////
```

e.g.:

```
////////////////////////////////////////////////
// add you custom dynamics classes here
////////////////////////////////////////////////
} else if (lgname.find("mydyn")==0 && lgname.size()>5) {
        supergroup="mydyn";
        d=new MyDynamics(fluorophors, oname);
```

Here you custom class is called `MyDynamics` and in the configration files it will have the prefix `mydyn`.

- `FluorescenceMeasurement`-classes have to be added near the text

```
////////////////////////////////////////////////
// add you custom detection classes here
////////////////////////////////////////////////
```

e.g.:

```
////////////////////////////////////////////////
// add you custom detection classes here
////////////////////////////////////////////////
} else if (lgname.find("mydetection")==0 && lgname.size()>14) {
        supergroup="mydetection";
        m=new MyDetection(fluorophors, oname);
```

Here you custom class is called `MyDetection` and in the configration files it will have the prefix `mydetection`.

4.2 Implementing `FluorophorDynamics`-Classes

```
virtual void init();
```

This function is called before the simulation and is used to initialize the simulation object.

```
virtual void propagate(bool boundary_check=true);
```

This function implements the main functionality. It is called in every step and propagates the walkers that are stored in the array `walker_state`. If the parameter `boundary_check` is set `true`, the function body should perform a boundary-check at the borders of the sim-box. Otherwise the sim-box is assumed to be infinite (used for testing).

```
virtual std::string report();
```

This function reports the object-state in human-readable form.

```
virtual void read_config_internal(jkINIParser2& parser);
```

This function reads the object-configuration from the given parser, which is already cd'ed to the group to be read. This function is called twice for each object: Once for the super-group and once for the actual object-group.

Note that there are additional functions that can be used in special cases. See the implemented classes in the repository for details.

4.3 Implementing `FluorescenceMeasurement`-Classes

```
virtual void init();
```

This function is called before the simulation and is used to initialize the simulation object.

```
virtual void propagate();
```

This function implements the main functionality. It is called in every step and propagates the walkers that can be obtained from the dynamics-objects in the array `dyn`.

```
virtual std::string report();
```

This function reports the object-state in human-readable form.

```
virtual void save();
```

This function stores the simulation results.

```
virtual void read_config_internal(jkINIParser2& parser);
```

This function reads the object-configuration from the given parser, which is already cd'ed to the group to be read. This function is called twice for each object: Once for the super-group and once for the actual object-group.

Note that there are additional functions that can be used in special cases. See the implemented classes in the repository for details.