Marlics User Guide

Renato Ferreira de Souza and Eric Koudhi Omori

September 16, 2019

1 Introduction

Welcome to Marlics (**Mar**ingá **li**quid **c**rystal **s**imulator). Marlics is a software developed in C++ to simulate the dynamics of the liquid crystal tensorial order parameter by means of[e] finite differences. Finite differences works very well in non-curvilinear geometries, and in a lesser ~~less~~[e] extent, in curvilinear geometries as well. We already provided three of such geometries (slab, bulk and sphere), but since we are providing the source code under a GPL license, the user can implement their own geometries if needed (please see developer guide).

The software usage is simple: the user fills an input file chosing the values of simulation parameters and the stage variables. This script is passed to the software as the main input ("<" in Unix systems) which in turns set up and run the simulation. For ease of use we have provided an example input file with the code. In the section 4 we are going to describe all the available parameters and options.[e]

In the following sections we will shown how to obtain and execute the MarLiCS, along with the source code.~~In the following sections we will shown describe which are all the possible parameters and options.~~[e]

2 Download and Compilation

The marlics source code can be downloaded in Renato Ferreira de Souza(Belanji) git-hub page (**Fill here the gitub page later**). If you use an Unix system, the source code provides a makefile (thanks to Eric Koudhi Omori)[1] to easy up the process of compilation. Marlics needs the following libraries installed on your system and reachable by the compiler: gsl (Gnu scientific library) and BLAS. We provide two examples in the make file, one we use the GSL BLAS, and other we use the mkl library to provide the necessary functions. After setting up you just have to type:

$ make

Now, Marlics is already functional. If you prefer you can add an shortcut (or a copy of the program) in a folder where your system can reach automatically, known as binary directories in Unix system. A list of these directories can, and must, be found under the variable $PATH.~~In Unix system the folder searched by the system when can be found in the variable $PATH.~~[e]

3 Execution

Executing Marlics is very straightforward, you call the program and pass a input file as the standard input. For example, assuming that you have some version (or a link) of the marlics in one binary folder, you can use the command:~~if you are using an Unix system and your marlics has an shortcut/copy in a directory shown in the $PATH. If your input file is named "input_file.txt" you will launch Marlics typing (we provide an example input file if you would like to try it in your console):~~[e]

$ marlics < input_file.txt

[1] stop misspelling koudhi, please, just please. and don't forget to erase this =D

to launch a instance of marlics using the "input_file.txt" as input file.[e]

You will notice that the software display several lines of information on your display, if you would like to save these lines to keep an log of the simulation execution (which we strongly advise you to do so), you can redirect the output direct to a file. In Unix systems this can be done with the redirection operator >, which will create a new file if it does not exist, or it will overwrite an existing file. If you would like to keep the previous content in the file you can use append operator >>, however, we strongly advise against the use of >>, since it can leave the output files very difficult to understand.

Again as an example, if your settings is in a file named "input_file.txt" and you would like to place the log information in a new/overwritten output file named "simulation.log", you can do this by typing:

$ marlics < input_file.txt >simulation.log

4 Setting up your input file

All parameters and simulation setup can be ~~are~~[e] passed to marlics via the input file. In this section we will describe how to set up your simulation and how chose the parameters necessary.

To help the user we provided an example input file with the program.

4.1 Filling the input file

To fill in an parameter value in the input file you need to include a line with the parameter name and the parameter value separated by space: ~~type de parameter name followed by any number of spaces and the parameter value afterwards:~~ [e]

```
parameter_name   value
```

notice that any excess space will be ignored. For ease of the users, the parameter names on marlics isn't case sensitive.

Be carefull since marlics can accept multiple instances of the same parameter, but will consider just the last one.[e]

If you want to write a text to help you remember something our to help others understand you input file, you can place a comment on the file. Comments are lines placed in the input file that are ignored by marlics. In marlics comments start with the character # and goes until the end of the line.

4.2 Marlics parameters

In marlics some parameters are mandatory and others are optional, have some standard values predefined or depends of others parameters. In order to guide the input file construction, here is a list of the actual defined parameters:

Parameter name	variable type	mandatory/standard value
Geometry	string	Yes
Nx	integer	Yes
Ny	integer	Yes
Nz	integer	Yes
dx	integer	Yes
dy	integer	Yes
dz	integer	Yes
integrator	string	Yes
facmin	double	0.4
facmax	double	3
prefac	double	0.8

Atol	double	0.001
Rtol	double	0.001
A, b and c	double	Yes
K1, K2 and K3	double	if L1 is not defined
L1	double	if K1, K2 and K3 are not defined
L2, L3, Ls and Lq	double	No
p0 or q0	double	no
T	double	no
Mu or gamma	double	Yes
Mu_s or gamma_s	double	Except in bulk geometry
ti	double	0.0
tf	double	Yes
dt	double	Tf/1e6
timeprint	double	Tf/20
timeprint_type	string	Linear
timeprint_increase_factor	double	Tf/20
output_folder	string	.
output_fname	string	director_field_$$.csv
initial_output_file_number	int	0
initial_conditions	string	yes
initial_file_name	string	in read_from_file ic
theta_i	double	in homogeneous ic
phi_i	double	in homogeneous ic
anchoring_type	int + string	1 for sphere and 2 for slab geometry
W01	int + double	1 for each Founier-Galatola or Rapini-Papoular anchoring
theta_0	int + double	1 for each Rapini-Papolar or strong anchoring
phi_0	int + double	1 for each Rapini-Papolar or strong anchoring

For more information about these parameters, check the following sections.

4.3 Geometry parameters

The geometry parameter sets up the confinement of your liquid crystal. Actually there are 3 different types of geometries available in marlics: bulk, slab and sphere(experimental).

In bulk simulations, the liquid crystal is placed in an square box with periodic boundary conditions in all directions.

In slab the liquid crystal is placed inside a box with periodic boundary conditions in the x and y direction, and with boundary conditions chosen by the user in both ends of the z axis.

Finally, in the sphere geometry the liquid crystal is placed inside an ellipsoid with radius equals half of the Grid size i.e. $Nx/2, Ny/2$ and $Nz/2$. Here, one boundary condition has to be defined as the ellipsoid outer layer, giving no space to an periodic boundary condition. Take note that, since we are using a regular grid to define a curvilinear geometry, the geometry resolution greatly depends on the Grid Size. ~~Finally in the sphere geometry the liquid crystal is places inside an sphere with one boundary condition setup-ed by the user and no periodic boundary conditions at all. Since here we use regular grids and the sphere has an curvilinear geometry, the fit of the liquid crystal inside the sphere is ....~~ [e]

The grid size and spacing is controlled with 6 parameters: Nx, Ny, Nz ,dx ,dy and dz. The number of points in each axis directions is defined ny the parameters Ni, while de distance between them is set by the parameters di. For example:

```
geometry   slab
Nx   200                %   /*      grid size       */
Ny   200                %   /*      grid size       */
```

```
Nz   100          %   /*    grid size      */
dx   10.0         %   /*    10^-9 m        */
dy   10.0         %   /*    10^-9 m        */
dz   10.0         %   /*    10^-9 m        */
```

Sets a slab with 200 points in the x and y axis and 100 in the z one, and every point is separated by a distance of 10 nanometers in each direction.

4.4 Integrator

We have now just one integrator implemented in marlics, the Dormand-Prince 5(4), a time adaptive 5th order Runge-Kutta method. Our implementation accepts 5 parameters to control how the time-adaption is performed, but all of them have standard values (we refer the interested reader to read the reference guide for more details).

An example line would be:

```
integrator   DP5
```

4.5 Liquid crystal Parameters

In Marlics, we simulate the elastic behavior of chiral, or non-chiral, nematics by solving the dynamic equations given by Landau-de Gennes model (please, see reference xx). This model needs a set of parameters: a, b, c, $T - T*$, L_1, L_2, L_3, L_q, L_s, p_0, mu, mu_s.

The parameters a, b and c are the thermodynamic parameters and defines the equilibrium liquid crystal order parameter, together with energy necessary to shift its value. It is obligatory to define all 3 of them, c can not be 0 and a has to be bigger to 0 to define a liquid crystal system. Example:

```
A    0.182
b   -2.12
c    1.73
```

The parameters L_i can be filled in two different ways. You can pass the values of L_i directly , for example:

```
L1   7.26        %   \* pN *\
L2   18.8        %   \* pN *\
L3   1.91        %   \* pN *\
Ls   1.0         %   \* pN *\
Lq   3.42        %   \* pN/nm *\
```

Or you can provide the values of the Frank elastic constant and let marlics calculates the values of L_i for you. Example:

```
k11   16.7       %   \* pN *\
k22   7.8        %   \* pN *\
k33   18.1       %   \* pN *\
```

If you , for some reason, define both ~~miss the use of~~[e] K_i and L_i, or insert multiple instances[e], the software will use only the last mentioned and ignore the others (We strong advise against it, since it can make your input file very confusing).

To simulate a chiral nematic you need to define the helix passe, p_0,~~If you want to simulate a chiral nematic you define the helix passe p_0~~[e] as the space necessary to the helix execute a 2π turn (distance being set in nanometers), under the parameter name p0 or p_0[e]. For example:

```
p0   500
```

Sets the passe to be 500 nanometers. In a alternative way, you can (but you shouldn't) define the quiral constant, q_0, given by $q_0 = 2\pi/p_0$.[e]

The temperature T is taken relative to virtual phase transition temperature T^*, for example:

```
T  -1
```

means that $(T - T^*) = -1$ Kelvin.

Finally, the liquid crystal viscosities can be passed in two different ways. You can pass the μ (set in *Pa s* ~~*sPas*~~[e]) value directly, or pass the parameter γ (set in *Pa s* ~~*sPanms*~~[e]) as used in the Frank-Osen theory and let marlics calculate the value of ~~μmu_s~~[e] for you. The same can be done with the surface viscosity. For example:

```
mu_1      0.3          %/*      Pa  s          */
mu_1_s    3.0          %/*      10 ^-9 Pa  s   */
```

4.6 Time parameters

There is 3 time parameters to be set in the input file.The parameter t_i and t_f (in microseconds) defines respectively the start and the end time of the simulation. The parameter dt (in microseconds) defines the time step, for a fixed time-step integrator, and the initial one for an adaptive time integrator.~~The parameter t_i (in microseconds) defines the start time of the simulation. The parameter t_f (in microseconds) defines the time the simulation finished and dt (in microseconds) defines the initial time step for a fixed time-step integrator and the initial for an adaptive time integrator~~[e]. For example:

```
dt   0.001        %/*      10^-6  s       */
ti   0.0          %/*      10^-6  s       */
tf   5000.0       %/*      10^-6  s       */
```

4.7 output parameters

There are 3 parameters to control how snapshots are taken and 3 controlling the file names in Marlics. The parameters controling the snapshot frequency are: *timeprint*, *timeprint_type*, *timeprint_increase_factor*. The *timeprint* sets when the first snapshot is taken (in microseconds).

The *timeprint_type* defines the interval function between snapshots, as linear or logarithmic. In the "linear" *timeprint_type*, the value of *timeprint_increase_factor* is the time interval between snapshots. In the "logarithmic" *timeprint_type*, the value of *timeprint_increase_factor* is factor which we multiply the snapshots times. Mathematicaly, it can be expressed as:
~~set if the interval between snapshots will be linear or logarithmic. The parameter *timeprint*_increase_factor sets the time interval between snapshots. If *timeprint*_type is set to "linear", the value of *timeprint*_increase_factor is the time interval between snapshots. If *timeprint*_type is set to "logarithmic", the value of *timeprint*_increase_factor is factor which we multiply the snapshots times. Mathematicaly, it can be expressed as:~~[e]

$$t_{s+1} = t_s * \text{timeprint\_increase\_factor} \tag{1}$$

where t_s is the Sth snapshot time and t_{s+1} is the $(S+1)$th snapshot time.

The *output_folder* can be used to define a output directory (the standard value is".\", also known as "current folder"). For naming your output files, you can use the parameter *output_fname* to define a pattern (the standard value is "director_field_$$.csv"). The given name must have a "$$" to define the file number position. for example:

```
output_folder   csv
output_fname    output_$$.csv
```

will produce the sequence "output_0.csv, output_1.csv, output_2.csv,..." and so on, inside the folder "csv".

Finally, there is the parameter *initial_output_file_number* which sets the number of the first snapshot taken (the standard value is 0)[e]. For example if *initial_output_file_number* is set to 3, them, the first snapshot will be *director_field_3.csv*.

One example of the setup would be:

```
time_print_type                logarithmic
timeprint                      50.          % /*  10^-6  s   */
timeprint_increase_factor      1.16         % /*  10^-6  s   */
initial_output_file_number     0
```

4.8 Initial Condtions

Marlics provides 4 types of initial conditions: Random, homogeneous, random_bulk_homogeneous_easy_-axis and read_from_file.

The *Random* initial conditions gets random directors, however, the nematic order parameter is set to $S = 0.15 S_{eq}$ (there would be no meaning leaving the order parameter fluctuating with high variance).

The *homogeneous* initial conditions set the liquid crystal order parameter at $S = S_{eq}$ and $P = 0$ and all the directors are set in the same direction. The direction is given by two parameters *theta_i* and *phi_i*, the polar and azimutal angles, both in degrees units.

The *random_bulk_homogeneous_easy_axis* sets the liquid crystal as *random* in the bulk and as *homogeneous* in the boundaries, in this case, the liquid crystal will be oriented along the easy axis in the boundary, see section for more information

Finally, the *read_from_file* take an marlics snapshot and use it as initial conditions. The file name is passed by the variable *ic_file*.

One example of the initial condition setup would be:

```
initial_conditions   homogeneous
theta_i              45.0
phi_i                0.0
```

4.9 Boundary Conditions:

For each geometry you must setup a defined number of boundaries. For example, in the slab geometry you must define two boundaries, the top and the bottom wall containing the liquid crystal. Each boundary region has a number that can be used to identify which boundary are you working with. For example, in the slab the bottom wall is referred by 0 while the top wall is referred by 1.

Actually there is 3 kinds of boundaries conditions implemented in marlics: Rapini-Papoular(Nobili-Durant), Fournier-Galatola and strong boundary conditions. Each boundary conditions needs its sets a set(?)[e] of parameters defined.

Boundary conditions that rely on a penalty function, like Rapini-Papoular and Founier-Galatola,[e] needs to defines an anchoring intensity Wo1. Boundary condtions that rely on an easy axis for the energy calculation, *i.e.* Rapnini-Papoular and strong boundary,[e] also need to define the easy axis by means of the polar angle theta_0 and azimutal angle phi_0.

If you want to define a boundary condition, for example, of Rapini-Papoular at the 0 wall, you need to iclude:~~type:~~[e]

```
anchoring_type  0   Rapini-Papoular.
W01             0   1000.0
theta_0         0   45.0
phi_0           0   45.0
```

4.10 Complete input file: