Marlics User Guide

Renato Ferreira de Souza, Eric Koudhi Omori and Rafael Soares Zola

November 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 a liquid crystal order parameters by means of finite differences. Our software is mainly intended to be used to simulate non-curvilinear geometries were finite differences works very well. We have provided two of such geometries: bulk and slab. Non-curvilinear geometries can also be implemented if they are well approximated by Cartesian grids. We have provided one of these geometries: sphere. Since we are providing the source code under a GPL license, the user can implement their own geometries if needed.

The software usage is as follows: the user fills an input file choosing the values of simulation parameters and the stage variables. This script is passed to the software as a call variable. For ease of use we have provided an example input file with the code, which can also be visualized in section4. Also, we have provided a table with all variables for reference in section 4.2.

2 Download and Compilation

The marlics source code can be downloaded in Renato Ferreira de Souza(Belanji) git-hub page (**https://github.com/Belanji**
If you use an Unix system, the source code provides a makefile [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. In Unix system the folder searched by the system when can be found in the variable $PATH.

3 Execution

Executing Marlics is very straightforward, you call the program and pass a input file as an argument. For example, 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):

$ marlics input_file.txt

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

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

[1] thanks to Eric Koudhi Omori

will create a new file if it does not exist, otherwise it will overwrite the existing file with the same name. If you would like to keep the previous content in the file you are overwriting, you can use append operator >>, however, we strongly advise against its use, 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 are passed to marlics via the input file. In this section we will describe how to set up your simulation and how to chose the necessary parameters.

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 followed by the parameter value (or values depending on the parameter informed) separated by any number of spaces:

```
parameter_name   value
```

Notice that any excess space will be ignored, also anything written after the required parameter values (this is useful to let some annotations about the parameters informed). For ease of the use, the parameter names on marlics are not case sensitive.

Comments line can be placed in the input file initiating the line with the symbol "#". We strong recommend the user to place comments in the input file in order to document it.

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

4.2 Marlics parameters

In marlics, some parameters are mandatory while others are optional, some have standard values predefined or depends on others parameters. In order to guide the input file construction, we present a table containing a list of all defined parameters. In the list you can find the parameters names, their units (the column is left in blank when the parameter has no units or its adimentional) and whenever it is mandatory or not. Note that whenever a parameter has an standard value it will be informed in the '"mandatory/standard value". Also, whenever the parameter usage don't fit in any of the previous cases, there is a reference to the section where you can find more information.

Parameter name	variable type	units	mandatory/standard value
Geometry	string		Yes
Nx	integer		Yes
Ny	integer		Yes
Nz	integer		Yes
dx	real	nm	Yes
dy	real	nm	Yes
dz	real	nm	Yes
integrator	string		Yes
facmin	real		0.4
facmax	real		3
prefac	real		0.8
Atol	real		0.001

Rtol	real		0.001
a	real	MJ/(m^2K)	Yes
b	real	MJ/m^2	Yes
c	real	MJ/m^2	Yes
K1	real	pN	See 4.5
K2	real	pN	See 4.5
K3	real	pN	See 4.5
L1	real	pN	See 4.5
L2	real	pN	No
L3	real	pN	No
Ls	real	pN	No
Lq	real	pN/m	No
p0 or q0	real	nm or 1/nm	No
T	real	K	No
Mu or gamma	real	Pa/s	Yes
Mu_s or gamma_s	real	nm Pa m/s	See 4.5
ti	real	μs	0.0
tf	real	μs	Yes
dt	real	μs	Tf/1e6
timeprint	real	μs	Tf/20
timeprint_type	string		Linear
timeprint_increase_factor	real		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		See 4.8
theta_i	real	degrees	See 4.8
phi_i	real	degrees	See 4.8
anchoring_type	int + string		See 4.9
W01	int + real		See 4.9
theta_0	int + real		See 4.9
phi_0	int + real		See 4.9

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

4.3 Geometry parameters

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 the distance between them is set by the parameters di.

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

In **bulk** geometry, the liquid crystal is placed in an square box with periodic boundary conditions in all directions, and therefore, no boundary conditions to be defined by the user.

In **slab** geometry, 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 i.e. there is two boundary conditions to be defined by the user.

Finally, in the **sphere** geometry the liquid crystal is placed inside an sphere with radius $R = dx * Nx/2 = dy * Ny/2 = dz * Nz/2$. Here it is imperative that you define the same number of points for Nx, Ny and Nz, also it is important that the grid spacing is equal in any direction i.e. $dx = dy = dz$. This geometry has one

boundary condition has to be defed and no periodic boundary conditions. Take note that, since we are using a regular grid to define a curvilinear geometry, the geometry resolution greatly depends on the grid sizes.

An example of geometry section is:

```
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         */
```

Which 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 three integrator implemented in marlics: explicit euler (EULER), explicit second order Runge-Kutta (RK2) and the Dormand-Prince 5(4) (DP5), a time adaptive 5th order Runge-Kutta method.

The use of EULER and RK2 do not require any special parameters, you just have to provide the line "integrator" and they are set to use. Both integrators keep the time-step dt constant during the whole simulation.

The integrator DP5 uses a 5th order Runge-Kutta to propagate the solution and a 4th order Runge-Kutta to estimate the error err. We implemented the time-step control as presented by **cite the nordicks here later**. This implementation requires 5 additional parameters: facmin, facmax, Atol, Rtol and prefac. facmax gives the maximum increase factor of the time-step dt, while facmin gives its maximum reduction. The values Atol and Rtol gives the the absolute and relative error tolerances respectively. Finally prefac is safety factor which multiplies the increase/decrease factor (we advise the using prefac= 0.9 or prefac= 0.8).

An example of integrator section would be:

```
integrator  DP5
atol  0.005
rtol  0.005
facmax  3.0
facmin  0.4
prefac  0.8
```

4.5 Liquid crystal Parameters

In Marlics, we simulate the elastic behavior of a 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 which are set in the input file.

The parameters a, b and c are the thermodynamic parameters together with the temperature T defines: the equilibrium value of the scalar order parameter S and energy necessary to shift its value. It is obligatory to define all of them and the they need to fulfill the following constraints: c can not be 0 and aT has to be lower than 0 to define a liquid crystal system. The temperature T is taken relative to virtual phase transition temperature T^*, for example:

```
a    0.182
b   -2.12
c   1.73
T  -1                  Kelvin
```

defines a liquid crystal at 1 degree lower than the virtual transition temperature.

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 $\{K_{11}, K_{22}, K_{33}, K_{24}\}$ and let marlics calculates the values of L_i as proposed in [] (note that this method uses the equilibrium vale of S to calculate the values of L_i). Example:

```
k11   16.7         /*   pN    */
k22   7.8          /*   pN    */
k33   18.1         /*   pN    */
k24   0            /*   pN    */
```

As explained before, If you define both K_i and L_i, or if you insert multiple instances of these parameters, 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).

In order to simulate a chiral nematic you need to define the helix pitch, p_0, as the space necessary to the helix execute a 2π turn (in nanometers). For example:

```
p0   500
```

Sets the pitch to be 500 nanometers. In a alternative way, you can define the chiral constant, q_0, given by $q_0 = 2\pi/p_0$.

Finally, the liquid crystal viscosities can be passed in two different ways. You can pass constant the μ_1 value directly, or pass the parameter γ as used in the Frank-Osen theory and let marlics calculate the value of its value using $\mu_1 = \gamma/S_{eq}$. The same can be done with the surface viscosity μ_{1s} which can be passed directly or by means of γ_s. For example:

```
mu_1       0.3          %/*      Pa  s              */
mu_1_s     30.0          %/*      10 ^-9 Pa  s       */
```

defines the values of $\{\mu_1, \mu_{1s}\}$, or by passing

```
gamma      0.3          %/*      Pa  s              */
gamma_s    30.0          %/*      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. For example:

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

set-up a simulation that starts at $t = 0$ and finishes at $t = 5ms$. If using a fixed time-step integrator the time steps will be taken with an interval $dt = 0.001\mu s$, otherwise it will start the simulation with a time-step $dt = 0.001$ but this value will be adapted as the simulation evolves.

4.7 output parameters

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

The *timeprint_type* defines if the snapshots will be taken in linear or logarithmic rate. If *timeprint_type* is set to **linear**, then the value of *timeprint_increase_factor* is the time interval between snapshots. If *timeprint_type* is set to **logarithmic**, then the the snapshot number $s+1$ will be taken at:

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

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

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

sets the first snapshot to be taken at $t = 50\mu s$, while the following ones will be taken at logarithmic rate with an increase factor of 1.16.

The *output_folder* can be used to define a output directory (the standard value is".\", also known as "current folder" in Unix system). 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 be substituted for the snapshot number. For example:

```
output_folder   csv
output_fname    output_$$.csv
initial_output_file_number   0
```

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

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 $S = 0.15S_{eq}$ (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 $P = 0$ while all the directors are set in the same direction. This direction is given by two parameters *theta_i* and *phi_i*, the polar and azimuthal 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, each boundary has a number which must be informed together with the parameter yu want to set. In more details:

slab: you must define two boundaries in this geometry, the top and the bottom wall containing the liquid crystal. The bottom wall is referred by 0 while the top wall is referred by 1.

sphere: This geometry has one boundary to be defined, the outer layer of the sphere, which is refereed by the number 0.

bulk: this geometry has no boundaries.

Actually there is 3 kinds of boundaries conditions implemented in marlics: Rapini-Papoular(Nobili-Durant), Fournier-Galatola and strong boundary conditions. Each boundary conditions requires some parameters to be defined.

Boundary conditions that rely on a penalty function, like Rapini-Papoular and Founier-Galatola, requires an anchoring intensity Wo1. Boundary condtions that rely on an easy axis for the energy calculation, *i.e.* Rapnini-Papoular and strong boundary, 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 of an sphere, you need to include:

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

and if you want to implement an Fournier and Galatola on the top surface you must include:

```
anchoring_type  1   Fournier-Galatola
W01             1   1000.0
```

4.10 Complete input file:

Here you can find the complete input file: