Documentation

This package contains several python codes to run simulations of rigid bodies made out of *blob particles* rigidly connected near a single wall (floor). These codes allow the user to calculate the mobility of complex shape objects, solve mobility or resistance problems for suspension of rigid bodies or run deterministic or stochastic dynamic simulations.

We explain in the next sections how to use the package. For the theory consult the references:

1. **Brownian Dynamics of Confined Rigid Bodies** S. Delong, F. Balboa Usabiaga and A. Donev. The Journal of Chemical Physics, **143**, 144107 (2015). DOI arXiv
2. **Hydrodynamics of suspensions of passive and active rigid particles: a rigid multiblob approach** F. Balboa Usabiaga, B. Kallemov, B. Delmotte, A. Pal Singh Bhalla, B. E. Griffith and A. Donev. arXiv

1. Prepare the package

The codes are implemented in python and it is not necessary to compile the package to use it. However, we provide alternative implementations in *C* (through the Boost Python library) and *pycuda* for some of the most computationally expensive functions. You can skip to section 2 but come back if you want to take fully advantage of this package.

1.1 Prepare the mobility functions

The codes use functions to compute the blob mobility matrix **M** and the matrix vector product **Mf**. For some functions we provide a *C* implementation which can be around 5 times faster than the python version. We also provide *pycuda* implementations which, for large systems, can be orders of magnitude faster. To use the *C* implementation move to the directory `mobility/` and compile `mobility_ext.cc` to a `.so` file using the Makefile provided (which you will need to modified slightly to reflect your Python version, etc.).

To use the *pycuda* implementation all you need is *pycuda* and a GPU compatible with CUDA; you don't need to compile any additional file in this package.

1.2 Blob-blob forces

In dynamical simulations it is possible to include blob-blob interactions to, for example, simulate a colloid suspension with a given steric repulsion. Again,

we provide versions in *python*, *C* and *pycuda*. To use the *C* version move to the directory `multi_bodies/` and compile `forces_ext.cc` to a `.so` file using the Makefile provided (which you will need to modified slightly to reflect your Python version, etc.).

To use the *pycuda* implementation all you need is *pycuda* and a GPU compatible with CUDA; you don't need to compile any additional file in this package.

2. Rigid bodies configuration

We use a vector (3 numbers) and a quaternion (4 numbers) to represent the location and orientation of each body, see Ref. 1 for details. This information is saved by the code in the `*.clones` files, with format:

```
number_of_rigid_bodies
vector_location_body_0 quaternion_body_0
vector_location_body_1 quaternion_body_1
.
.
.
```

For example, see the file `multi_bodies/Structures/boomerang_N_15.clones` to see the representation of a boomerang-like particle with location (0, 0, 10) and orientation given by the quaternion (0.5, 0.5, 0.5, 0.5).

The coordinates of the blobs forming a rigid body in the default configuration (location (0, 0, 0) and default quaternion (1, 0, 0, 0)) are given to the codes with `*.vertex` files. The format of these files is:

```
number_of_blobs_in_rigid_body
vector_location_blob_0
vector_location_blob_1
.
.
.
```

For example, the file `multi_bodies/Structures/boomerang_N_15.vertex` gives the structure of a boomerang-like particle formed by 15 blobs.

3. Run static simulations

We start explaining how to compute the mobility of a rigid body close to a wall. First, move to the directory `multi_bodies/` and inspect the input file `inputfile_bodies_mobility.dat`:

```
# Select problem to solve
scheme                                    bodies_mobility

# Select implementation to compute the blobs mobility
# Options: python and boost
mobility_blobs_implementation             python

# Set fluid viscosity (eta) and blob radius
eta                                       1.0
blob_radius                               0.25

# Set output name
output_name                               data/run.bodies_mobility

# Load rigid bodies configuration, provide
# *.vertex and *.clones files
structure   Structures/boomerang_N_15.vertex Structures/boomerang_N_15.clones
```

Now, to run the code, use

```
python multi_bodies_utilities.py --input-fule inputfile.dat
```

The code writes the results in files with the prefix `data/run.bodies_mobility`. You can inspect the file `*.bodies_mobility.dat` to see the 6x6 rigid body mobility.

List of options for `multi_bodies_utilities.py`:

- `scheme`: Options: `mobility, resistance and bodies_mobility`. Select the problem to solve. `mobility` computes the velocities of a suspension of rigid bodies subject to external forces and torques (see below). `resistance` computes the forces and torques on rigid bodies moving with given velocities (see below). `bodies_mobility` computes the mobility of one rigid body as in the above example.
- `mobility_blobs_implementation`: Options: `python and boost`. This option indicates which implementation is used to compute the dense blobs mobility matrix **M**. See section 1 to use the boost version.
- `mobility_vector_prod_implementation`: Options: `python, boost and pycuda`. This option select the implementation to compute the matrix vector product **Mf**. See section 1 to use the boost or pycuda implementation.

- `eta`: (float) the fluid viscosity.
- `blob_radius`: (float) the hydrodynamic radius of the blobs.
- `solver_tolerance`: (float) the tolerance for the iterative mobility solver.
- `output_name`: (string) the prefix used to save the output files.
- `velocity_file`: (string) name of the file with the velocities of the rigid bodies used in the `resistance` problem. The format of the file is one line per body and six floats per line corresponding to linear (first three) and angular velocities (last three).
- `force_file`: (string) name of the file with the forces and torques used in the `mobility` problem. The format of the file is one line per body and six floats per line corresponding to force (first three) and torque (last three).
- `structure`: (two strings) name of the vertex and clones files with the rigid bodies configuration, see section 2. To simulate bodies with different shapes add to the input file one `structure` option per each kind of body and give their `vertex` and `clones` files.

The output files are:

- `.inputfile`: It is a copy of the input file.
- `.bodies_mobility.dat`: the 6x6 rigid body mobility computed with the option `scheme bodies_mobility`. When this mobility is apply to the vector (force, torque) generates the vector (velocity, angular_velocity).
- `.force.dat`: force and torques on the bodies computed with the option `scheme resistance`. The format of the file is one line per body and six floats per line corresponding to force (first three) and torque (last three).
- `.velocities.dat`: velocities of the rigid bodies computed with the option `scheme mobility`. The format of the file is one line per body and six floats per line corresponding to linear (first three) and angular velocities (last three).

4. Run dynamic simulations

We have two python codes to run dynamic simulations. The first, in the directory `boomerang/`, allows to run Brownian simulations for a single body. See the instruction in `doc/boomerang.txt`. Here, we explain how to use the other code which allows to run deterministic simulations for many bodies. In the future we will extend this code to allow for stochastic simulations of many bodies.

First, move to the directory `multi_bodies/` and inspect the input file `data.main`:

```
# Select integrator
scheme                                  deterministic_adams_bashforth

# Select implementation to compute M and M*f
mobility_blobs_implementation           python
mobility_vector_prod_implementation     python

# Select implementation to compute the blobs-blob interactions
blob_blob_force_implementation          python

# Set time step, number of steps and save frequency
dt                                      0.1
n_steps                                 10
n_save                                  1

# Set fluid viscosity (eta), gravity (g) and blob radius
eta                                     1.0
g                                       1.0
blob_radius                             0.25

# Set parameters for the blob-blob interation
repulsion_strength                      1.0
debey_length                            1.0

# Set interaction with the wall
repulsion_strength_wall                 0.0
debey_length_wall                       1.0

# Set output name
output_name                             data/run

# Load rigid bodies configuration, provide
# *.vertex and *.clones files
structure Structures/boomerang_N_15.vertex Structures/boomerang_N_15.clones
structure Structures/shell_N_42_Rg_0_225.vertex Structures/shell_N_42_Rg_0_225.clones
```

With this input we can run a simulation for three rigid bodies, one with a boomerang shape and two with a spherical shape; see structures given in the options `structure`. To run the simulation just use

```
python multi_bodies --input-file data.main
```

Now, you can inspect the output, `ls data/run.*`. The output files are:

- `.bodies`: it contains the number of bodies and blobs in the simulation. Also how many types of bodies (i.e. different shapes) and how many bodies of each type.
- `.clones`: For each time step saved and each structure type the code saves a file with the location and orientation of the rigid bodies. The name format is (output_name + structure_name + time_step + .clones) The format of the files is the same that in the input .clones files.
- `inputfile`: a copy of the input file.
- `seed`: it saves the state of the random generator.
- `time`: the wall-clock time elapsed during the simulation, in seconds.

List of options for the input file:

- `scheme`: Options: `deterministic_forward_euler, deterministic_forward_euler_dense_algebra, deterministic_adams_bashforth`. It selects the scheme to integrate the equation of motion and solve the mobility problem. The `*forward_euler*` schemes are first order accurate while `*adams_bashforth*` is second order accurate. The scheme `*dense_algebra` use dense algebra methods to solve the mobility problem and therefore the computational cost scales like (number_of_blobs)**3. The other schemes use preconditioned GMRES to solve the mobility problem and are more efficient.
- `mobility_blobs_implementation`: Options: `python and boost`. This option indicates which implementation is used to compute the dense blobs mobility matrix **M**. See section 1 to use the boost version.
- `mobility_vector_prod_implementation`: Options: `python, boost and pycuda`. This option select the implementation to compute the matrix vector product **Mf**. See section 1 to use the boost or pycuda implementation.
- `blob_blob_force_implementation`: Options: `None, python, boost and pycuda`. Select the implementation to compute the blob-blob interactions. If None is selected the code does not compute blob-blob interactions.
- `eta`: (float) the fluid viscosity.
- `blob_radius`: (float) the hydrodynamic radius of the blobs.
- `solver_tolerance`: (float) the tolerance for the iterative mobility solver.
- `output_name`: (string) the prefix used to save the output files.
- `dt`: (float) time step length to advance the simulation.

- `n_steps`: (int) number of time steps.
- `initial_step`: (int (default 0)) use value `step` to restart a simulation from the time step `step`. The code will try to load `*.clones` of the form (output_name + structure_name + step + .clones) instead of the name given in the option `structure`.
- `n_save`: (int) it indicates to save the bodies configuration every `n_save` steps.
- `repulsion_strength`: (float) the blobs interact with a Yukawa potential, this is the strength of the potential (see section 4.1 to modified blobs interactions).
- `debey_length`: (float) the blobs interact with a Yukawa potential, this is the characteristic length of the potential (see section 4.1 to modified blobs interactions).
- `repulsion_strength_wall`: (float) the blobs interact with the wall with a hard sphere + Yukawa potential. This is the strength of the Yukawa potential (see section 4.1 to modified blobs interactions).
- `debey_length_wall`: (float) the blobs interact with the wall with a hard sphere + Yukawa potential. This is the characteristic length of the Yukawa potential (see section 4.1 to modified blobs interactions).
- `seed`: (string) name of a file with the state of a random generator from a previous simulation. It can be use to generate the same random numbers in different simulations.
- `structure`: (two strings) name of the vertex and clones files with the rigid bodies configuration, see section 2. To simulate bodies with different shapes add to the input file one `structure` option per each kind of body and give their `vertex` and `clones` files.

4.1 Modified the codes

Right now, the interactions between blobs and between blobs and the wall are hard-coded in the codes. We explain here how the user can change these interactions.

- blob-blob interactions: to modified the *python* implementation edit the function `blob_blob_force` in the file `multi_bodies/multi_bodies_functions.py`. To modified the C implementation edit the function `blobBlobForce` in the file `multi_bodies/forces_ext.cc` and recompile the C code. To modified the *pycuda* version edit the function `blob_blob_force` in the file `multi_bodies/forces_pycuda.py`

- blob-wall interaction: to modified the *python* implementation edit the function `blob_external_force` in the file `multi_bodies/multi_bodies_functions.py`. There are not *C* or *pycuda* versions of this function since it is not an expensive operation.

5. Software organization

- **body/**: it contains a class to handle a single rigid body.
- **boomerang/**: See next section.
- **doc/**: documentation.
- **mobility/**: it has functions to compute the blob mobility matrix **M** and the product **Mf**.
- **multi_bodies/**: the main code to run simulations of rigid bodies.
- **quaternion_integrator/**: it has a small class to handle quaternions and the schemes to integrate the equations of motion.
- **sphere/**: the folder contains an example to simulate a sphere whose center of mass is displaced from the geometric center (i.e., gravity generates a torque), sedimented near a no-slip wall in the presence of gravity, as described in Section IV.C in [1]. Unlike the boomerang example this code does not use a rigid multiblob model of the sphere but rather uses the best known (semi)analytical approximations to the sphere mobility.
- **stochastic_forcing/**: it contains functions to compute the product **M**ˆ{1/2}**z** necessary to perform Brownian simulations.
- **utils.py**: this file has some general functions that would be useful for general rigid bodies (mostly for analyzing and reading trajectory data and for logging).