LB-Colloids Documentation

Release 0.1

Joshua D Larsen

Oct 13, 2018

CONTENTS:

CHAPTER

ONE

INTRODUCTION TO LB-COLLOIDS

Lattice Boltzmann colloids is an object oriented python package that builds and simulates two dimensional, nine fluid node (D2Q9) computational fluid dynamic models. Lattice Boltzmann is able to represent fluid domains defined by complex geometries ehich are observed in natural porous media. Simulated fluid flow is slightly compressible and has been shown to return an approximation of the Navier-Stokes equation (*Benzi et al 1992*). The lattice Boltzmann code used for this software is based on the single relaxation time Bhantager Gross Krook formulation of lattice Boltzmann. Simple bounceback boundary conditions are implemented at fluid boundaries to preserve mass balance. As a result of the bounceback boundary conditions no-slip conditions develop at pore-solid interfaces.

Binarized imagery is used to represent the fluid flow domain in this implementation of lattice Boltzmann. Distinct advantages of using binarized imagery are: fluid flow in natural porous media can be simulated through the use of segmented thin sections, and synthetic porous media can easily be generated as an image and used for fluid dynamic simulations.

Colloid particle tracking simulations can be parameterized and simulated for steady state lattice Boltzmann computational fluid dynamic models using this software. Colloid transport through the soil environment is of great interest and importance to soil development processes through the translocation of clays, contaminant transport (*Saiers 1996, Jaisi et al 2008*), filtration and transport of bio-colloids (*Harter 2000, Redman 2004, Foppen et al 2005*), and soil nutrient dynamics. Microscale colloid particle tracking simulations allow the user to gain additional insight into physio-chemical processes influencing colloid transport. Coupling macroscopic breakthrough curve information with the microscopic distribution of colloids during a simulation provides insight into attachment and colloid retention mechanisms for simulations. These insights have the potential to illuminate retention and transport mechanisms for column and field scale studies of colloid and contaminant transport.

Parameterization of an LB-Colloids model is possible in two ways. Formatted text documents can be supplied to the LB-Colloids simulation software and results will be stored in an hdf5 file and a human readable text document. Advanced users may prefer to parameterize simulations through a python interface. This gives the advanced user the ability to easily calibrate models by adjusting parameters within python code blocks, run multiple models, and perform sensitivity analysis. Currently output readers and built in analysis methods are only available to the advanced user. API documentation serves as a guide to the advanced user on importing and using these python objects.

This document serves as the user guide to the LB-Colloids python package. For more details pertaining to the development of this package see *Larsen and Schaap TBD*. The user guide follows the structure:

Chapter 2: Table of mathematical symbols

Chapter 3: Installation of LB-Colloids

Chapter 4: Formatted text input

Chapter 5: Parameterization of LB-Colloids

Chapter 6: Lattice Boltzmann API

Chapter 7: Colloid Simulation API

Chapter 8: Penetrable-Sphere (PSHPERE) API

CHAPTER

TWO

TABLE OF MATHEMATICAL SYMBOLS

2.1 Lattice Boltzmann

$\boldsymbol{e_i}$: lattice Boltzmann eigenvector array

f_i : distribution function

f_{eq} : equilibrium distribution function

ρ : macroscopic fluid density

τ : relaxation time

$\boldsymbol{u}$: macroscopic fluid velocity

v : fluid viscosity

w_i : fluid link weights

2.2 Colloid Simulation

A_H : Hamaker constant

a_c : colloid radius

D_0 : diffusivity

ϵ_0 : dielectric permativity of a vacuum

ϵ_r : dielectric permativity of water

e : electron charge

F^b : bouyancy force

F^B : brownian force

F^D : drag force

F^G : gravity force

f_1 : hydrodynamic correction factor

f_2 : hydrodynamic correction factor

f_3 : hydrodynamic correction factor

f_4 : hydrodynamic correction factor

$G(0,1)$: random gaussian distribution

g : acceleration due to gravity

h : gap distance

$\bar{h}$: non-dimensional gap distance

I^* : Two times fluid ionic strength.

k : Boltzmann constant

κ : Debye length

M : molarity

N_A : Avagadro's number

Φ^A : Attractive interaction energy

Φ^{EDL} : Electric double layer interaction energy

ψ_c : colloid potential

ψ_s : surface potential

ρ_c : colloid density

ρ_w : fluid density

u : fluid velocity

μ : fluid viscosity

V : colloid velocity

T : Fluid temperature

Z : cation charge

ξ : $6\pi\mu a_c$

CHAPTER

THREE

INSTALLATION OF LB-COLLOIDS

LB-Colloids is currently avaialable for python 2.7.6 - 2.7.12 installations. Support is currently not available for python 3, however most source code can be converted to python 3 with little trouble. Future support is planned for python 3.5

Recommended installation of LB-Colloids is performed using the python tool pip. If pip is not included with your python distribution an installation script can be found at https://bootstrap.pypa.io/get-pip.py. It is recommended that the user adds pip as a path variable for easy installation of python packages.

An installation of gfortran must be present on the user's computer to properly install LB-Colloid's. Mathematic modules are compiled locally in FORTRAN upon install and called by python for computational efficiency. Gfortran should be added as a path variable to your computer.

LB-Colloids can be downloade from *pypi* and/ort *tbd*. Move the LB-Colloids package to your prefered source code location.

Open a terminal and navigate to the base directory of the LB-Colloids source code. You should see setup.py in this directory. Install LB-Colloids using pip.

```
>>> cd Desktop/LB-Colloids
>>> pip install .
>>> # or for the developer use
>>> pip install -e .
```

Congratulations! LB-Colloids is now installed on your machine.

CHAPTER

FOUR

FORMATED TEXT INPUT

LB Colloids computational fluid dynamic models can be parameterized through the use of formatted text files that are parsed by internal input control modules. Five separate data types are supported in the input reader modules. Data types used during the *LB-Colloids* input parameterization are strictly enforced. It is highly recommended that the user familiarize themselves with data types before attempting to parameterize and run *LB-Colloids.*

4.1 Description of Data Types

STRING: String type data within *LB-Colloids* is limited to file names.

Example usage:

LBMODEL: mymodel.png

INTEGER: Integer type data.

Example usage:

NCOLS: 200

FLOAT: A floating point value or real number must be supplied.

Example usage:

AC: 1e-6

BOOLEAN: A value of True or False must be supplied.

Example usage:

PLOT: True

LIST: List type data combines data types to allow the user to specify multiple values in the parameterization process. This data type is used to set image boundary conditions using the solid and void keywords.

Example usage:

SOLID: 255 223 200

DICTIONARY: Dictionary type data combines data types to use related pairs of data in the parameterization process. This data type is limited in usage to the optional concentration and valence keywords.

Example usage:

VALENCE: Na 1 Ca 2 Mg 2 Al 3

4.2 NAM File Inputs

The *LB-Colloid NAM file* is a simple formatted text file that is read by run_model.py. The NAM file contains relative or absolute file paths to a Lattice Boltzmann configuration file and colloid transport configuration files. The program run_model.py looks within its current directory for files ending with the suffix .nam, reads them, and delegates the input and output control to the appropriate *LB-Colloids* modules at runtime.

4.2.1 Input Structure:

Input uses a block type structure. Block keywords begin a parameterization section. When the parameterization section is complete, the keyword end must be supplied for *LB*-Colloids to close the block input reader. Within the input structure description, brackets surrounding a parameter indicates that it is an optional parameter.

LBMODEL (STRING): Block keyword to designate the beginning of a lattice Boltzmann parameterization block. This keyword is used to inform run_model.py that a lattice Boltzmann simulation will be performed.

> **LBCONFIG (STRING):** File name of the lattice Boltzmann configuration file that is used for simulation.

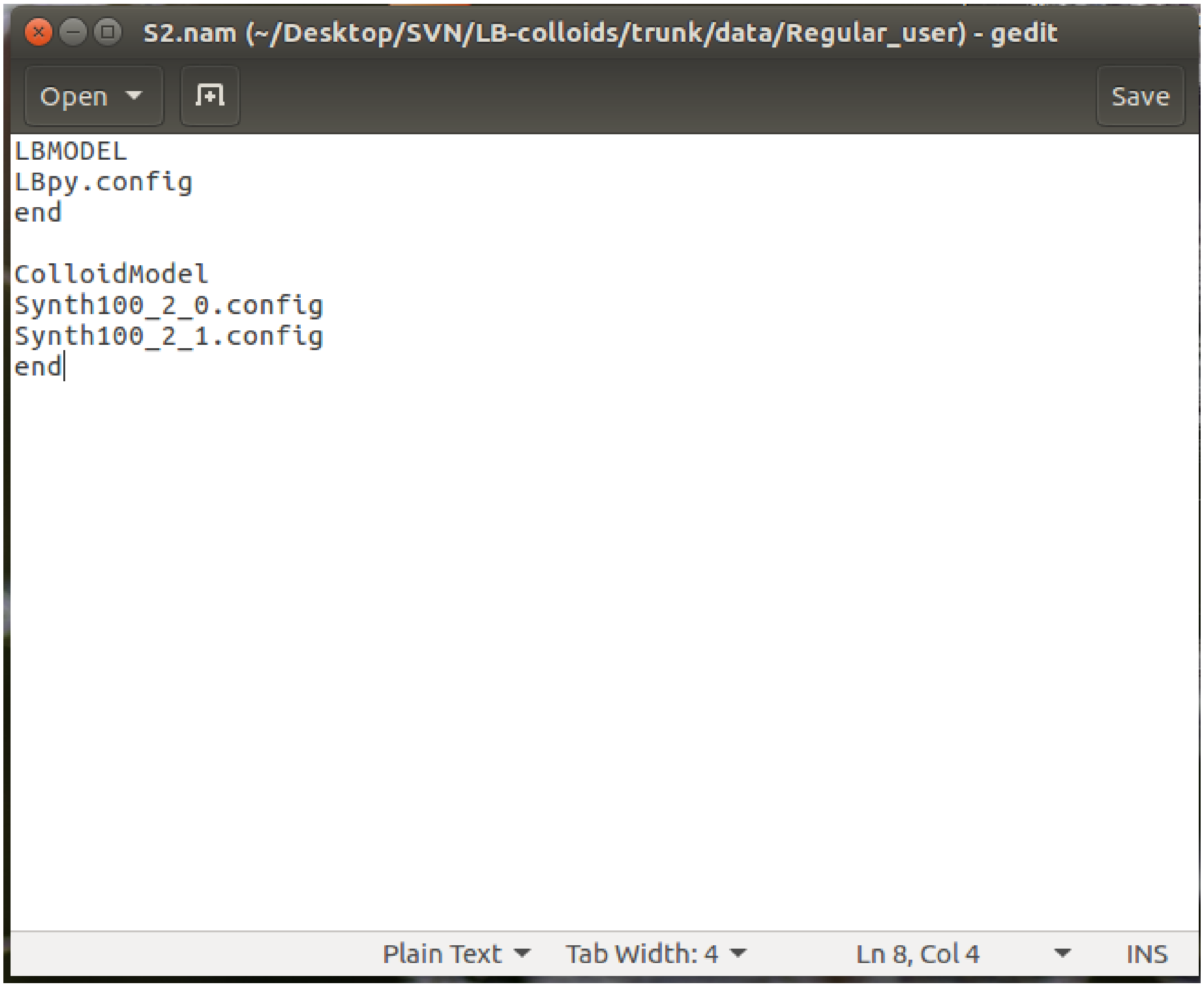
END: Block keyword to inform run_model.py that it has finished gathering parameters for the LBMODEL data block.

[COLLOIDMODEL] (STRING): Optional block keyword to designate the beginning of a colloid transport parameterization block. The keyword is used to inform run_model.py that a colloid transport simulation will be performed.

> **[COLLOIDCONFIG]*n (STRING):** Optional file name of the colloid transport configuration file that is used for the simulation. Multiple colloid transport configuration files can be supplied for a multiple ionic strength model (MISM) and each filename must occupy a new line within the NAM file. This parameter is required if the COLLOIDMODEL keyword is used

[END] (STRING): Optional block keyword to inform run_model.py that it has finished gathering parameters for the COLLOIDMODEL data block. This parameter is required if the COLLOIDMODEL keyword is used

4.2.2 Example NAM file:



S2.nam (~/Desktop/SVN/LB-colloids/trunk/data/Regular_user) - gedit
Open
Save
Plain Text
Tab Width: 4
Ln 8, Col 4
INS

```
LBMODEL
LBpy.config
end

ColloidModel
Synth100_2_0.config
Synth100_2_1.config
end
```

4.3 Lattice Boltzmann Configuration File

The lattice Boltzmann configuration file uses a block input format to parameterize D2Q9 lattice Boltzmann simulations. Four separate blocks are available to the user. The MODEL PARAMETERS and IMAGE PARAMETERS block must be supplied for basic model configuration. The PERMEABILITY PARAMETERS block is optional, however the NITERS value must be updated for a permeability model to be able to run to steady state conditions. Configuration blocks can be added to the lattice Boltzmann configuration file in any order as long as the proper formatting requirements are fulfilled. Parameters within a configuration block may be listed in any order as long as parameter requirements are fulfilled. Only one parameter may be listed per line within the configuration file.

4.3.1 Input Structure:

START MODEL PARAMETERS (STRING): Required block keyword to begin parameterization of the lattice Boltzmann model parameters data block.

LBMODEL (STRING): Hdf5 file path and name corresponding to the lattice Boltzmann model. Lattice Boltzmann simulation results will be written to this Hdf5 file.

LBRES (FLOAT): The lattice Boltzmann simulation resolution parameter corresponds to the image resolution, in meters, of the synthetic porous media or soil thin section for D2Q9 lattice Boltzmann simulations.

[KERNEL] (STRING): Lattice Boltzmann debug option that allows users to switch between simulating models with Python or FORTRAN as the base computational code. This option defaults to FORTRAN (recommended) if it is not supplied by the user. The FORTRAN kernel is approximately 100x faster than the Python kernel. Valid options are 'python' and 'fortran'.

[PHYSICAL_VISCOSITY] (FLOAT): Optional parameter relating to the physical viscosity of the simulated fluid within a model run. This parameter is used in the dimensionalization process from non-dimensional lattice units to SI units. Default value is 8.9e-4 Pa S which corresponds to the viscosity of water at 25^o C.

[PHYSICAL_RHO] (FLOAT): Optional parameter relating to the physical density of the simulated fluid within a model run. This parameter is used in the dimensionalization process from non-dimensional lattice units to SI units. Default value is 997 Kg/m^3 which corresponds to the viscosity of water at 25^o C.

END MODEL PARAMETERS (STRING): Required block keyword to end parameterization of the lattice Boltzmann model parameters block

START IMAGE PARAMETERS (STRING): Required block keyword to begin the parameterization of model domain boundary conditions using image properties of the supplied thin section.

IMAGE (STRING): Filename of the image representing the lattice Boltzmann simulation domain

SOLID (LIST [INTEGER]): List of solid phase greyscale values as integers corresponding to the supplied model thin section

VOID (LIST [INTEGER]): List of pore/void phase greyscale values as integers corresponding to the supplied model thin section

[BOUNDARY] (INTEGER): Number of boundary layers to add to the top and bottom of the simulation domain. This parameter helps avoid local compression effects due to dead end pores connected to the top and bottom of the simulation domain. Default is 10 boundary layers.

[PLOT] (BOOLEAN): Keyword parameter to display a plot of the binarized model domain before simulating fluid flow. This option is recommended during initial parameterization runs to ensure that fluid and solid phases are being properly represented within the domain. Default is False.

END IMAGE PARAMETERS (STRING): Required block keyword to end the parameterization of the image parameters data block.

START PERMEABILITY PARAMETERS (STRING): Required keyword to begin the lattice Boltzmann permeability model parameterization block.

[NITERS] (INTEGER): Number of simulation time steps (number of iterations) applied to the lattice Boltzmann permeability simulation. Defaults to 1. It is highly recommended to change this value as one time step will not reach equilibrium conditions.

[RHO] (FLOAT): Initial fluid density for lattice Boltzmann simulation. Defaults to 1.

[TAU] (FLOAT): Lattice Boltzmann BGK relaxation time parameter. Defaults to 1. The acceptable range of TAU is from 0.5 to 1.5.

[GRAVITY] (FLOAT): Gravity force applied to fluid as a body force. Gravity drives fluid flow in the simulation. Default is 1e-3.

END PERMEABILITY PARAMETERS (STRING): Required keyword to end the lattice Boltzmann permeability model parameterization block

[START OUTPUT CONTROL] (STRING): Keyword to begin the lattice Boltzmann output control parameterization block. This block contains optional parameters that write updates to the terminal and save imagery of lattice Boltzmann simulation progression.

[VERBOSE] (INTEGER): Integer flag that indicates the number of time steps between lattice Boltzmann print statements to the terminal. This option updates user of the model's progression. Default is 100.

[IMAGE_SAVE_INTERVAL] (INTEGER): Integer flag that indicates the time step interval to save model fluid velocity to a matplotlib image. Default is None and no images are saved to disk.

[IMAGE_SAVE_NAME] (STRING): Base name of images to save as output. This parameter is ignored unless the IMAGE_SAVE_INTERVAL parameter is used. Default IMAGE_SAVE_NAME is "LB".

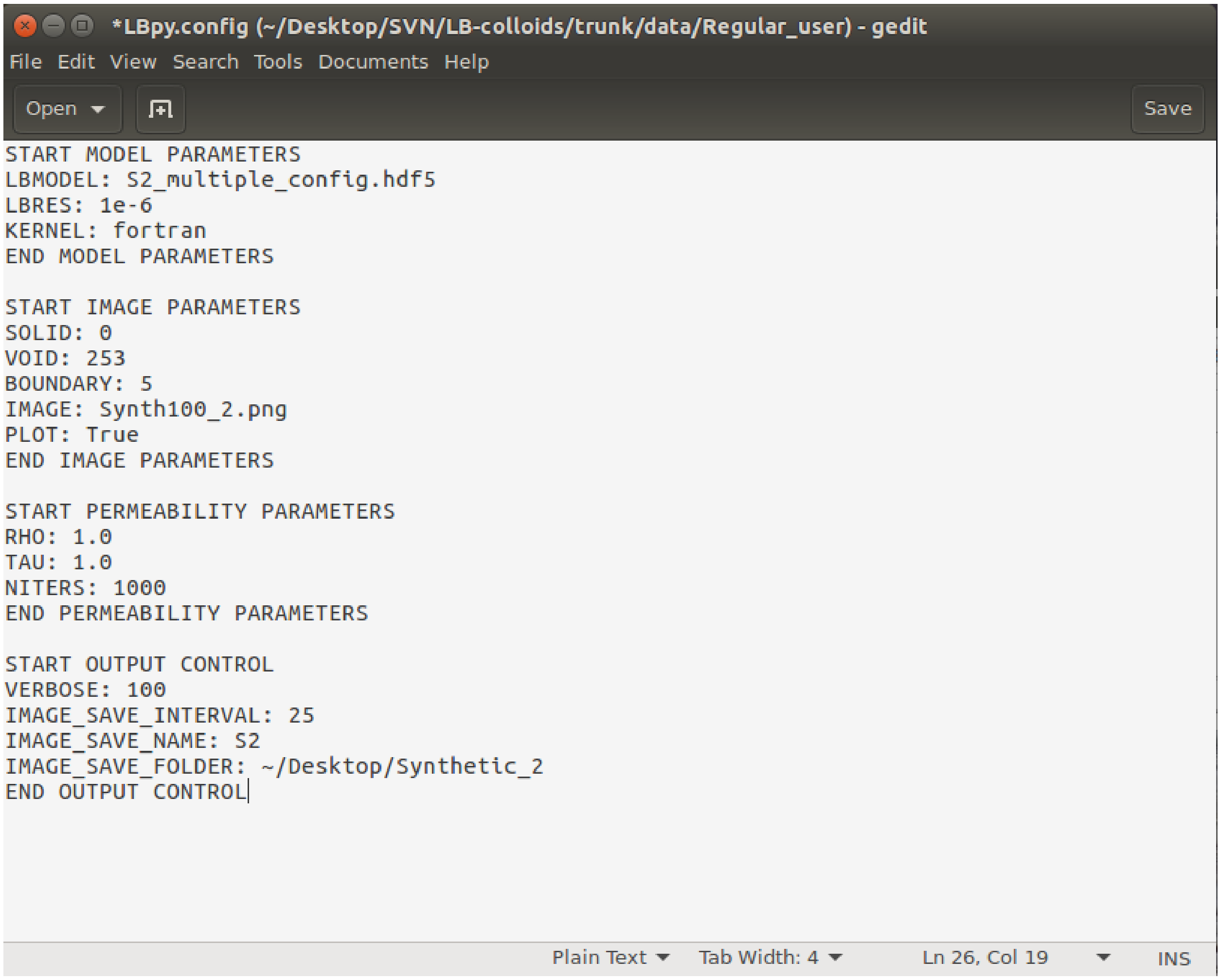
[IMAGE_SAVE_FOLDER] (STRING): Directory location to save simulation velocity images. This parameter is ignored unless the IMAGE_SAVE_INTERVAL parameter is used. Default IMAGE_SAVE_FOLDER is "~/Desktop/LBimages".

[VMIN] (FLOAT): VMIN controls the minimum boundary of velocity to plot on a lattice Boltzmann output image. This parameter is used to adjust the color scale for plotting only. VMIN can only be used if IMAGE_SAVE_INTERVAL is used. Default value is -0.010.

[VMAX] (FLOAT): VMAX controls the maximum boundary of velocity to plot on a lattice Boltzmann output image. This parameter is used adjust the color scale for plotting only. VMAX can only be used if IMAGE_SAVE_INTERVAL is used. Default value is 0.0.

[END OUTPUT CONTROL] (STRING): Keyword to end the lattice Boltzmann output control parameterization block.

4.3.2 Example Lattice Boltzmann configuration file:


*LBpy.config (~/Desktop/SVN/LB-colloids/trunk/data/Regular_user) - gedit
File Edit View Search Tools Documents Help
Open
Save
Plain Text
Tab Width: 4
Ln 26, Col 19
INS

```
START MODEL PARAMETERS
LBMODEL: S2_multiple_config.hdf5
LBRES: 1e-6
KERNEL: fortran
END MODEL PARAMETERS

START IMAGE PARAMETERS
SOLID: 0
VOID: 253
BOUNDARY: 5
IMAGE: Synth100_2.png
PLOT: True
END IMAGE PARAMETERS

START PERMEABILITY PARAMETERS
RHO: 1.0
TAU: 1.0
NITERS: 1000
END PERMEABILITY PARAMETERS

START OUTPUT CONTROL
VERBOSE: 100
IMAGE_SAVE_INTERVAL: 25
IMAGE_SAVE_NAME: S2
IMAGE_SAVE_FOLDER: ~/Desktop/Synthetic_2
END OUTPUT CONTROL
```

4.4 Colloids Simulation Control File

The colloids simulation control file uses a block input structure to parameterize colloid simulations within the *LB-Colloids* system. Four model parameter blocks are available to the user to parameterize colloid models. The MODEL PARAMETERS block must be supplied in the colloid simulation control file to run a model. The PHYSICAL PARAMETERS and CHEMICAL PARAMETERS configuration blocks are optional, however it will be necessary to specify some parameters within each of these blocks to properly simulate experimental conditions for individual models. Defaults within these blocks pertain to glass bead media and kaolinite colloids. The OUTPUT PARAMETERS configuration block is optional and contains useful parameters for later data analysis. Configuration blocks can be added to the colloids simulation control file in any order as long as the proper formatting requirements are fulfilled. Parameters within a configuration block may be listed in any order as long as parameter requirements are fulfilled. Only one parameter may be listed per line within the configuration file.

4.4.1 Input Structure:

START MODEL PARAMETERS (STRING): Keyword to begin the model parameters input block. This input block contains necessary parameterization information to run a basic colloid simulation.

> **LBMODEL (STRING):** HDF5 file name containing results from a steady state lattice Boltzmann simulation
>
> **LBRES (FLOAT):** Lattice Boltzmann grid resolution.
>
> **GRIDREF (FLOAT):** Grid refinement option which creates a bilinear interpolation of the lattice Boltzmann model domain. Colloid grid resolution is defined by
>
> $$COLRES = \frac{\text{LBRES}}{\text{GRIDREF}}$$
>
> **ITERS (INTEGER):** Number of time steps the colloid simulation will run for
>
> **TIMESTEP (FLOAT):** Time step length. Reduction of time step length creates increased stability and greater accuracy, but longer model run times. A recommended starting point is 1e-06 s.
>
> **NCOLS (INTEGER):** Number of colloids to introduce into the system in the initial time step.
>
> **[CONTINUOUS] (INTEGER):** Continuous is a flag that indicates multiple releases of colloids throughout the simulation. The value of continuous is the interval when additional colloids are released into the colloid simulation. Default is 0 (a single pulse of colloids released at the beginning of time step 1).
>
> **[AC] (FLOAT):** The colloid radius parameter defaults to 1e-6 m.
>
> **[RHO_COLLOID] (FLOAT):** Colloid density parameter is optional and adjustable based on type of colloidal particle being simulated. Default value is 2650. Kg/m^3
>
> **[TEMPERATURE] (FLOAT):** Fluid temperature parameter defaults to 298.15 K.

END MODEL PARAMETERS (STRING): Keyword to end the model parameters input block.

[START PHYSICAL PARAMETERS] (STRING): Keyword to that indicates the beginning of the physical parameters input block.

> **[RHO_WATER] (FLOAT):** Physical density of water. Default is 997. Kg/m^3
>
> **[RHO_COLLOID] (FLOAT):** Colloid density parameter. It is unnecessary to parameterize if RHO_COLLOID has been added in the model parameters input block. Defaults to 2650 Kg/m^3
>
> **[VISCOSITY] (FLOAT):** Dynamic viscosity of the fluid phase within a colloid simulation. Default is 8.9e-4 Pa S.
>
> **[SCALE_LB] (FLOAT):** Lattice Boltzmann velocity scaling factor. Exercise extreme caution in using this option.

[END PHYSICAL PARAMETERS] (STRING): Keyword to end the physical parameters input block. This is a required parameter if START PHYSICAL PARAMETERS keyword is used.

[START CHEMICAL PARAMETERS] (STRING): Keyword to that indicates the beginning of the chemical parameters input block.

> **[I] (FLOAT):** The ionic strength of the fluid phase is calculated by the equation
>
> $$I = \sum_{i=1}^{n} Z_i^2 M_i$$

where Z_i is the cation charge and M_i is the molarity of each solution component i. The default ionic strength is set to 1e-3 M.

[ZETA_SOLID] (FLOAT): Bulk zeta potential of the solid phase within a colloid simulation. Default value is for glass bead media -60.9e-3 mV.

[ZETA_COLLOID] (FLOAT): Bulk zeta potential of colloids introduced into a colloid simulation. Default value is for kaolinite colloids -40.5e-3 mV.

[CONCENTRATION] (DICTIONARY [STRING, FLOAT]): Optional parameter that allows the user to specify cation molarity pairs to parameterize the model instead of using ionic strength. This parameter must be supplied if VALENCE is used.

[VALENCE] (DICTIONARY [STRING, INTEGER]): Optional parameter that allows the user to specify cation valence pairs to parameterize the model instead of using ionic strength. This parameter must be supplied if CONCENTRATION is used.

[LVDWST_WATER] (FLOAT): The van der Waals surface tension of the simulation fluid which is used to parameterize van der Waals interactions within the colloids simulation. Default value is 21.8e-3 J/m^2 which corresponds to water.

[LVDWST_COLLOID] (FLOAT): The van der Waals surface tension of the simulated colloidal material which is used to parameterize van der Waals interactions within the colloids simulation. Default value is 39.9e-3 J/m^2 which corresponds to a kaolinite colloid.

[LVDWST_SOLID] (FLOAT): The van der Waals surface tension of the simulated solid phase which is used to parameterize van der Waals interactions within the colloids simulation. Default value is 33.7e-3 J/m^2 for glass bead porous media.

[PSI+_WATER] (FLOAT): The electron-acceptor parameter of Lewis Acid Base surface tension for the simulation fluid. Default is 25.5e-3 J/m^2 for water

[PSI+_COLLOID] (FLOAT): The electron-acceptor parameter of Lewis Acid Base surface tension for the colloid material. Default is 0.4e-3 J/m^2 for a kaolinite colloid.

[PSI+_SOLID] (FLOAT): The electron-acceptor parameter of Lewis Acid Base surface tension for the solid phase. Default is 1.3e-3 J/m^2 for a glass bead porous media.

[PSI-_WATER] (FLOAT): The electron-donor parameter of Lewis Acid Base surface tension for the simulation fluid. Default is 25.5e-3 J/m^2 for water.

[PSI-_COLLOID] (FLOAT): The electron-donor parameter of Lewis Acid Base surface tension for the colloid material. Default is 34.3e-3 J/m^2 for a kaolinite colloid.

[PSI-_SOLID] (FLOAT): The electron-donor parameter of Lewis Acid Base surface tension for the solid phase. Default is 62.2e-3 J/m^2 for glass bead porous media.

[SHEER_PLANE] (FLOAT): Equivalent to the thickness of one layer of water molecules. Also referred to as thickness of the stern layer. Default is 3e-10 m.

[EPSILON_R] (FLOAT): The dielectric constant of water at the simulation temperature. The default value is 78.3 which corresponds to 25^o C.

[END CHEMICAL PARAMETERS]: Keyword to end the chemical parameters input block. This is a required parameter if START CHEMICAL PARAMETERS keyword is used.

[START OUTPUT CONTROL] (STRING): Keyword to that indicates the beginning of the output control input block.

[PRINT_TIME] (INTEGER): Integer flag that indicates the number of time steps between colloid simulations print statements to the terminal. Updates user of the model's progression. Default is equal to the parameter ITERS.

[STORE_TIME] (INTEGER): Integer flag that indicates the number of time steps between internal storage functions within the colloid model. Increasing the value of this flag reduces memory consumption. This flag is also used to specify the interval for saving to a TIMESERIES file and output plotting.

[ENDPOINT] (STRING): String flag that indicates an endpoint file should be saved. This option is highly recommended for use. The endpoint variable should correspond to the name of the endpoint file the user wishes to save.

[TIMESERIES] (STRING): String flag that indicates a timeseries file should be saved. The save interval is indicated by the STORE_TIME variable. The timeseries variable corresponds to the name of the timeseries file the user wishes to save.

[PATHLINE] (STRING): String flag that indicates a pathline file should be saved. Colloid position is written to output at every time step using this option. The pathline variable corresponds to the name of the pathline file the user wishes to save.

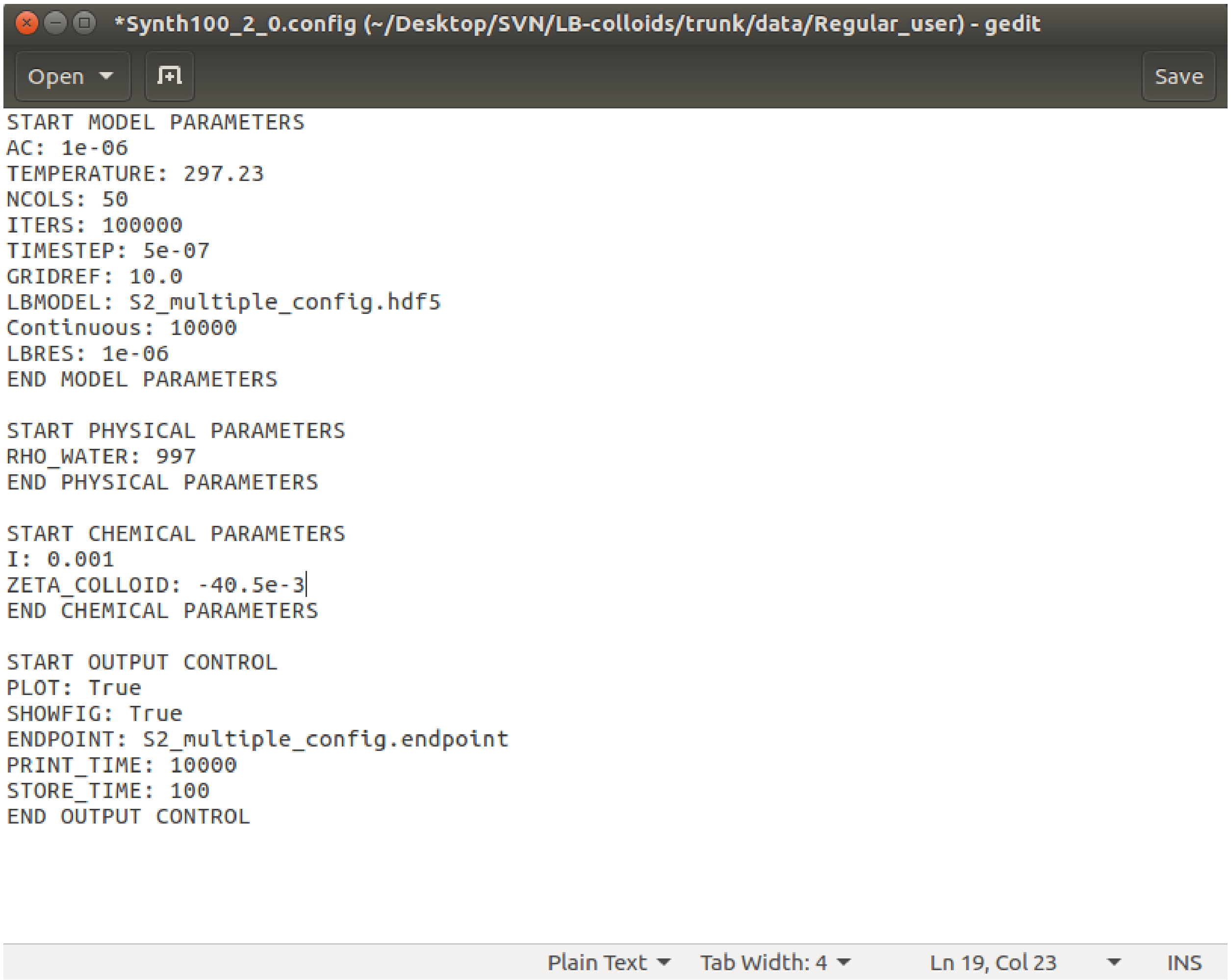
[PLOT] (BOOLEAN): Boolean flag that indicates a plot of colloid positions within the model domain be produced upon successful model completion. The plotting interval of colloid positions is set with the STORE_TIME flag. The plotted image will be saved as a <.png> file with the same base name as was provided for ENDPOINT. If no endpoint file was provided the figure will be displayed for the user to save manually. Default is False.

[SHOWFIG] (BOOLEAN): Boolean flag to indicate that the user wants to examine the figure before manually saving to file. SHOWFIG will only work when PLOT is True. Default is False.

[OVERWRITE] (BOOLEAN): Boolean flag to overwrite data on existing HDF5 file. This flag is useful if the user does not wish to re-run lattice Boltzmann simulations while optimizing a colloid simulation.

[END OUTPUT CONTROL] (STRING): Keyword to end the output control input block. This is a required parameter if START OUTPUT CONTROL keyword is used.

4.4.2 Example colloid model configuration file:



*Synth100_2_0.config (~/Desktop/SVN/LB-colloids/trunk/data/Regular_user) - gedit
Open
Save

```
START MODEL PARAMETERS
AC: 1e-06
TEMPERATURE: 297.23
NCOLS: 50
ITERS: 100000
TIMESTEP: 5e-07
GRIDREF: 10.0
LBMODEL: S2_multiple_config.hdf5
Continuous: 10000
LBRES: 1e-06
END MODEL PARAMETERS

START PHYSICAL PARAMETERS
RHO_WATER: 997
END PHYSICAL PARAMETERS

START CHEMICAL PARAMETERS
I: 0.001
ZETA_COLLOID: -40.5e-3
END CHEMICAL PARAMETERS

START OUTPUT CONTROL
PLOT: True
SHOWFIG: True
ENDPOINT: S2_multiple_config.endpoint
PRINT_TIME: 10000
STORE_TIME: 100
END OUTPUT CONTROL
```

CHAPTER

FIVE

LATTICE BOLTZMANN

5.1 Lattice Boltzmann boundary conditions

The lattice Boltzmann boundary condition modules provide classes and methods to prepare model domains, supplied to LB-Colloids as .png images, for use in a lattice Boltzmann simulation. Before a lattice Boltzmann simulation can take place, it is necessary to convert the model domain imagery into a boolean representation of that media. The boolean conversion allows for easy recognition of pore-solid interface nodes and the representation of geometrically complex domains. Boundary layers can also be added to the upper and lower boundaries of the lattice Boltzmann model. Boundary layers are applied to reduce the potential development of local pressure maxima and compresssion effects due to periodic boundary conditions at the inlet and outlet of the system. Bounceback boundary conditions are applied to all other surfaces within the system.

5.1.1 API documentation

`lb_colloids.`**`LBImage`**

alias of *`lb_colloids.LB.LB_2Dimage`*

Lattice boltzmann image preparation and domain setup module

This module contains an image reading utility and a binarization utility. The LB_2Dimage module is aliased in LB-Colloids as LBImage.

Example usage of how to create a binary image with boundary conditions applied and save the base model for usage in with LB_permeability

```
>>> from lb_colloids import LBImage
>>> image = LBImage.Images("my_thin_section.png")
>>> binary = LBImage.BoundaryCondition(image, fluidvx=[0], solidvx=[233, 255],␣
↪nlayers=3)
>>> binary.binarized
>>> HDF5_write(binary.binarized, binary.porosity, binary.boundary, 'LBModel.hdf5')
```

`class` `lb_colloids.LB.LB_2Dimage.`**`BoundaryCondition`**(*data*, *fluidvx*, *solidvx*, *nlayers*, *bottom=False*)

Class to instatiate open boundary layers at the top and bottom of the lattice boltzmann image, closed boundary layers at either side of the image, and binarize the array into a boolen type.

Parameters

- **`data`** (*`np.ndarray`*) – NxN array of image data relating to the input file
- **`fluidvx`** (*`list`*) – list of fluid voxel grey values
- **`solidvx`** (*`list`*) – list of solid voxel grey values

- **nlayers** (*int*) – number of open boundary layers to add to the top and bottom of the image array.

Attributes

binarized
return – Binary image with boundary conditions applied

fluid_voxels
return – user defined fluid volxels

grey_values
return – unique image grayscale values

nlayers
return – Number of boundary condition layers

porosity
return – image porosity

solid_voxels
return – user defined solid voxels

class lb_colloids.LB.LB_2Dimage.**HDF5_write**(*arr*, *porosity*, *boundary*, *output*)
Write class for LB2d_image. Writes a HDF5 file that includes the binary image, porosit &, number of boundary layers,

Parameters

- **arr** (*np.ndarray*) – binarized image data
- **porosity** (*float*) – porosity of the image
- **boundary** (*int*) – number of boundary layers
- **output** (*str*) – output hdf5 file name

class lb_colloids.LB.LB_2Dimage.**Images**(*infile*)
Convience class to open and adjust RGB images to BW if necessary.

Parameters **infile** (*string*) – binary input file name (image file domain)

lb_colloids.LB.LB_2Dimage.**run**(*image*, *solid*, *fluid*, *output*, *boundary=5*)
Funcitonal approach to run the LB-Colloids script

Parameters

- **image** (*str*) – image file name
- **solid** (*list*) – list of interger grey scale values corresponding to the solid phase
- **fluid** (*list*) – list of interger grey scale values corresponding to the fluid phase
- **output** (*str*) – output hdf5 file name
- **boundary** (*int*) – number of boundary layers for the model

```
>>> LBImage.run('my_thin_section.png', fluid=[0], solid=[233, 255],
>>>              output="LBModel.hdf5", nlayers=3)
>>>
```

5.2 Lattice Boltzmann Permeability

The lattice Boltzmann permeability modules contain the principle methods and subroutines for simulating fluid flow within a model domain. Two possible programatic styles can be used for simulating lattice Boltzmann fluid domain models. The python "kernel" is primarily used for delevopment and debugging purposes. The default FORTRAN "kernel" provides a compiled version of this code, which is executes at speeds of approximately 100x faster than the python version. The FORTRAN code is dynamically compiled to the local machine upon installation of LB-Colloids. If FORTRAN compilation fails, it is reccomended that the user use the python kernel until the compiling issue is resolved. Documentation of the lattice Boltzmann permeability modules does not include documentation of the FORTRAN subroutines or the python mathematical code. For an understanding of the mathematics behind the lattice Boltzmann method an examination of source code and additional reading into lattice Boltzmann literature is required.

5.2.1 API documentation

D2Q9 lattice Boltzmann simulations are performed using classes and methods contained in this module. The LB2DModel class is the main class the user will interact with within this module. LB2DModel calls a series of subroutines to run and save the lattice Boltzmann simulation

Basic mathematical relationships have been implemented from relevant academic literature. For a complete handling of the mathematics please Chen and Doolen 1996 is a great starting point. A listing of mathematical relationships are provided here for completeness.

$$\rho = \sum_{i=1}^{n} f_i$$

$$\rho \boldsymbol{u} = \sum_{i=1}^{n} f_i \boldsymbol{e_i}$$

$$v = \frac{1}{6}(\frac{2}{\tau} - 1)$$

$$f_i(x + e_i, t + \Delta t) = f_i(x, t) - \frac{f_i - f_i^{eq}}{\tau}$$

$$f_i^{eq} = \rho w_i[1 + 3\boldsymbol{e_i} \cdot \boldsymbol{u} + \frac{9}{2}(\boldsymbol{e_i} \cdot \boldsymbol{u})^2 - \frac{3}{2}u^2$$

Although lattice Boltzmann mathematics are included within the python methods contained within this module, it is highly reccomended that the user use the default Fortran Kernal option to run LB models. The python kernal is approximately 100x slower than Fortran.

Example showing the build and run of a D2Q9 LB2DModel

```
>>> from lb_colloids import LB2DModel
>>> from lb_colloids import LBImage
>>>
>>> image = LBImage.Images("my_thin_section.png")
>>> binary = LBImage.BoundaryCondition(image, fluidvx=[0], solidvx=[233, 255],␣
↪nlayers=3)
>>> model = LB2DModel(img=binary.binarized)
>>> model.niters = 2000
>>> model.rho = 1.0
>>> model.tau = 0.8
>>> result = model.run(output="LBModel.hdf5")
```

class `lb_colloids.LB.LB_2Dpermeability.`**`HDF5_write`**(*mrho, tau, u, f, rho, output, mean_uy, mean_ux, pore_diameter, reynolds_number, velocity_factor, img=None, porosity=None, boundary=None*)

Hdf5 model write class to save simulation results from a LB Permeability model run

Parameters

- **`mrho`** (*`float`*) – mean fluid density
- **`tau`** (*`float`*) – lb relaxation time
- **`u`** (*`np.ndarray`*) – fluid velocity array
- **`f`** (*`np.ndarray`*) – distribution function
- **`rho`** (*`np.ndarray`*) – density array
- **`output`** (*`str`*) – hdf file name
- **`mean_uy`** (*`float`*) – mean fluid velocity y direction
- **`mean_ux`** (*`float`*) – mean fluid velocity x direction
- **`pore_diameter`** (*`float`*) – mean pore diameter
- **`reynolds_number`** (*`float`*) – calculated reynolds number
- **`velocity_factor`** (*`float`*) – non-dimensional to dimensional velocity conversion factor
- **`img`** (*`float`*) – binary image array
- **`porosity`** (*`float`*) – porosity
- **`boundary`** (*`float`*) – nlayers boundaty condition

class `lb_colloids.LB.LB_2Dpermeability.`**`LB2DModel`**(*img, kernel='fortran'*)

object oriented method to instantiate and run a Two-Dimensional lattice boltzmann model. Calls upon either fortran or python kernels to run a model.

Please reference documentation for *lb_colloids.LB.LB_2Dpermeability.***LB2DModel** for a full listing of attributes and methods.

Uses protective programming to ensure user supplied data fits within normal model parameters.

All attributes can be reset before a model run by passing a valid value to them

Parameters

- **`img`** (*`np.ndarray`*) – binarized image array from LBImage
- **`kernel`** (*`str`*) – the simulation kernel. Default is fortran

run: method to run the lb model and return a distribution function

Attributes

Methods

`cs`

return – Lattice speed of sound

cs2
return – Lattice speed of sound ** 2

get_mean_pore_size()

Returns the mean pore diameter of the domain

get_reynolds_number()

Returns the model's reynolds number after simulation

get_velocity_conversion()

Returns the conversion factor from LB velocity to Phys.

gravity
return – Body force applied to simulation

img
return – Binarized lattice boltzmann fluid domain

kernel
return – Kernel type (Python or Fortran)

niters
return – Number of lattice Boltzmann time steps to be simulated

nlayers
return – Calculated number of boundary layers applied to the simulation domain

nx
return – Model size in the x-direction

ny
return – Model size in the y-direction

physical_rho
return – Physical density of the simulation fluid

physical_viscosity
return – Physical viscosity of the simulation fluid

porosity
return – calculated porosity of the simulation domain

q
return – number of simulation fluid nodes

resolution
return – Model resolution in physical units (meters)

rho
return – Non-dimensional fluid density

run(*output='LBModel.hdf5', image_int=None, image_folder=None, image_name='LB_', vmax=0, vmin=-0.01, verbose=None*)
user method to run the lattice Boltzmann model and return the resulting distribution function.

tau
return – Simulation relaxation time

viscosity
return – Calculated lattice Boltzmann viscosity for the simulation

`lb_colloids.LB.LB_2Dpermeability.`**`darcy_velocity`**(*x, img, nbound*)

Method to get the darcy velocity of the steady state lb model based on outflow velocity :param np.ndarray x: macroscopic velocity in the y-direction :param np.ndarray img: image array corresponding to model domain :param int nbound: number of boundary layers applied to model domain

Returns darcy velocity

`lb_colloids.LB.LB_2Dpermeability.`**`get_mean_pore_size`**(*img, nx*)

Finds the mean pore diameter of the domain

Parameters

- **`img`** (*`np.ndarray`*) – binary image array of domain
- **`nx`** (*`int`*) – number of pixels in x direction of fluid domain

Returns

return Model domain mean pore size

`lb_colloids.LB.LB_2Dpermeability.`**`get_reynolds_number`**(*pore_diameter, uy, porosity, rho, viscosity*)

Calculate the model's reynolds number after simulation based off of mean velocity and mean fluid density

Parameters

- **`pore_diameter`** (*`float`*) – calculated lb pore diameter
- **`uy`** (*`float`*) – lb mean fluid velocity in y direction
- **`porosity`** (*`float`*) – porosity of the medium
- **`rho`** (*`float`*) – lb mean fluid density
- **`viscosity`** (*`float`*) – lb fluid viscosity

Returns Simulation Reynolds number

`lb_colloids.LB.LB_2Dpermeability.`**`get_velocity_conversion`**(*reynolds_number, uy, rho, pore_diameter, viscosity*)

Calculates the physical velocity conversion factor from LB reynolds number and user supplied physical parameters.

Parameters

- **`reynolds_number`** (*`float`*) – Fluid simulations reynolds number
- **`uy`** (*`float`*) – mean y velocity from LB model
- **`rho`** (*`float`*) – physical density of the fluid
- **`pore_diameter`** (*`float`*) – physical pore diameter
- **`viscosity`** (*`float`*) – physical fluid viscosity

Returns Non-dimensional fluid velocity to physical fluid velocity conversion factor

`class` `lb_colloids.`**`LB2DModel`**(*img, kernel='fortran'*)

object oriented method to instantiate and run a Two-Dimensional lattice boltzmann model. Calls upon either fortran or python kernels to run a model.

Please reference documentation for *lb_colloids.LB.LB_2Dpermeability.***LB2DModel** for a full listing of attributes and methods.

Uses protective programming to ensure user supplied data fits within normal model parameters.

All attributes can be reset before a model run by passing a valid value to them

Parameters

- **img** (*np.ndarray*) – binarized image array from LBImage
- **kernel** (*str*) – the simulation kernel. Default is fortran

run: method to run the lb model and return a distribution function

Attributes

Methods

5.3 Lattice Boltzmann Input Output

Lattice Boltzmann input/output (*IO*) modules have been developed to facilitate end user functionality, such as reading configuration files and creating lattice Boltzmann fluid velocity images. The lattice Boltzmann IO module structure is a late addition to the project and is still in it's infancy, although the packaged code is completely stable. All lattice Boltzmann input and output read and write objects will eventually be migrated to this set of modules.

5.3.1 API documentation

lb_colloids.**lbIO**

alias of *lb_colloids.LB.LBIO*

lbIO is the lattice Boltzmann input control module. This module contains a Config class that reads in the lattice Boltzmann configuration file and parses the parameters contained within that file. HDF5 write options will eventually be moved from other portions of the lattice Boltzmann modeling system to LBIO.py

Example import of a lattice Boltzmann configuration file is shown below

```
>>> from lb_colloids import lbIO
>>> config_file = "LB_model.config"
>>> config = lbIO.Config(config_file)
>>> model_dict = config.model_parameters()
>>> image_dict = config.image_parameters()
>>> permeability_dict = config.permeability_parameters()
>>> output_dict = config.output_parameters()
```

class lb_colloids.LB.LBIO.**Config**(*fname*)

Class to handle the input output control of the lattice botzmann modeling system. Can be added to, and uses consistancy checks to ensure correct types and required values are present for LBModel to run.

Parameters **fname** (*str*) – configuration file name

Methods

check_if_valid(*blockname*, *pname*, *validparams*)

Method checks if a specific parameter is valid for the block it was supplied in the configuration file.

Parameters

- **blockname** (*str*) – the name of the configuration block
- **pname** (*str*) – parameter name
- **validparams** (*tupele*) – tuple of valid parameters for the specific configuration block

check_model_parameters(*ModelDict*)

Check for required parameters. If not present inform the user which parameter(s) are needed

Parameters **ModelDict** (*dict*) – Model Input block dictionary

get_block(*blockname*)

Method to locate the beginning and end of each input block

Parameters **blockname** (*str*) – blockname of a parameters block in the config file.

Returns (list) returns a list of parameters contained within the block

image_parameters()

reads the IMAGE PARAMETERS block of the configuration file and creates the ModelDict which contains the required parameters to run LB-Colloid

Returns (dict) Dictionary containing image paramaters for simulation

model_parameters()

reads the MODEL PARAMETERS block of the configuration file and creates the ModelDict which contains the required parameters to run LB-Colloid

Returns (dict) Dictionary containing model parameter block information

output_parameters()

reads the OUTPUT CONTROL block of the configuration file and creates the ModelDict which contains the required parameters to run LB-Colloid

Returns (dict) Dictionary containing output control parameters

parametertype(*parameter*)

Method takes a parameter string, splits it, and sets the parameter type

Parameters **parameter** (*str*) – string of "<pname>: <param>"

Returns (tuple) parameter name, parameter associated with pname

permeability_parameters()

reads the PERMEABILITY PARAMETERS block of the configuration file and creates the ModelDict which contains the required parameters to run LB-Colloid

Returns (dict) Dictionary containing parameters from the Permeability input block

lb_colloids.LB.**LB_pretty**

alias of *lb_colloids.LB.LB_pretty*

LB_pretty contains methods to format and generate matplotlib images of lattice Boltzmann velocity results. These methods are called internally in LB2DModel if the user specifies plotting within the output dictionary.

example code to save an image of fluid magnitude within a domain from hdf5 output

```
>>> from lb_colloids.LB.LB_pretty import velocity_image
>>> from lb_colloids import ColloidOutput
>>>
>>> model = ColloidOutput.Hdf5Reader("LB_model.hdf5")
>>> u_y = model.get_data("lb_velocity_y")
>>> u_x = model.get_data("lb_velocity_x")
>>> image = model.get_data("image")
>>> u = np.array([u_y, u_x])
>>> velocity_image(u, image, "LB", 1000, vel=False, vmin=0.0001, vmax=0.1)
```

`lb_colloids.LB.LB_pretty.`**`set_colormap`**(*vel*)

Method to set the matplotlib colormap option based on python version and datatype

Parameters **vel** (*bool*) – Are we plotting a yvelocity image

Returns (object) Matplotlib colormap object

`lb_colloids.LB.LB_pretty.`**`velocity_image`**(*u, img, name, numit, vel, vmin, vmax*)

Lattice Boltzmann fluid velocity image generation routine. Can return plot velocity or fluid magnitude.

Parameters

- **u** (*np.ndarray*) – Lattice Boltzmann macroscopic velocity array
- **img** (*np.ndarray*) – LBImage binarized image array
- **name** (*str*) – base name to save figure to
- **numit** (*int*) – time step that image is produced at
- **vel** (*bool*) – velocity flag, if false magnitude is plotted
- **vmin** (*float*) – Matplotlib vmin
- **vmax** (*float*) – Matplotlib vmax

CHAPTER SIX

COLLOID SIMULATION

6.1 LB Colloid simulations

The lattice Boltzmann colloids simulation module, LB_Colloid, contains the base simulation classes for colloid models. The LB_Colloid module is general and can be adapted for use with other computational fluid dynamic methods that produce velocity arrays within a discernable porous media. Colloid streaming, updating, and internal data storage are managed within this module. Simulation time and physical time are also tracked within the colloids simulation module.

6.1.1 API documentation

`lb_colloids.`**`ColloidModel`**

alias of *`lb_colloids.Colloids.LB_Colloid`*

The LB_Colloid module contains base classes to simulate colloid transport for colloid simulation. This module acts as the control center. The class Colloid is the base representation of a colloid and contains streaming and updating rules. The class TrackTime is the simulation timer, which tracks both number of time steps and the time step length. Also of importance is the run() method. This method initiates a colloid simulation from a IO.Config object.

A user can initiate a model run with a Colloid_IO.Config() object. Please see the Input Output section for details on building the Colloid_IO.Config() object

```
>>> from lb_colloids import ColloidModel
>>>
>>> config = IO.Config()   # We assume that the Colloid_IO.Config() object is already␣
↪built. See the Colloid_IO section for details
>>> ColloidModel.run(config)
```

`class` `lb_colloids.Colloids.LB_Colloid.`**`Colloid`**(*xlen*, *ylen*, *resolution*, *tag=0*)

Primary colloid class to instantiate and track colloids through a colloid simulation

Parameters

- **`xlen`** (*`int`*) – grid length after interpolation in the x-direction
- **`ylen`** (*`int`*) – grid length after interpolation in the y-direction
- **`resolution`** (*`float`*) – grid resolution after interpolation
- **`tag`** (*`int`*) – colloid number for output writing

Methods

reset_master_positions()

Resets the master position storage mechanism for all colloids. Master position storage is used to later generate colloid-colloid DLVO fields.

store_position(*timer*)

Method to store colloid position and update the the time of storage

Parameters **timer** (*float*) – current model time

strip_positions()

Memory saving function to strip unused, save colloid position information

update_position(*xvelocity*, *yvelocity*, *ts*)

grid index method to update continuous colloid system using the discete grid forces. idxry must be inverted because we assume (0,0) at top left corner and down is negitive.

Parameters

- **xvelocity** (*np.ndarray*) – colloid simulation velocity array in the x domain
- **yvelocity** (*np.ndarray*) – colloid simulation velocity array in the y domain
- **float** (*ts*) – physical time step in seconds

update_special(*irx*, *iry*, *flag*)

Special updater class for colloids that have exited the model domain or experienced an internal error

Parameters

- **irx** (*int*) – grid index in the x domain
- **iry** (*int*) – grid index in the y domain
- **flag** (*int*) – number indicator of special condition. 3 = normal breakthrough condition

class lb_colloids.Colloids.LB_Colloid.**Singleton**

Singleton object to hold Colloid positions. Potential object to hold calculated arrays to avoid passing and reduce memory requirements

class lb_colloids.Colloids.LB_Colloid.**TrackTime**(*ts*)

TrackTime class is the model timer. This class enables stripping stored time steps which is useful to free memory after writing the data to an external file. Is necessary for output class functionality!

Parameters **ts** (*int*) – model time step set by user (physical time)

Methods

print_time()

Method prints time in a standard format to the terminal

strip_time()

Memory saving method that removes unused information from timer arrays

update_time()

Method to update the current model time and store it lb_colloids.Colloids.LB_Colloid.**run**(*config*)

Model definition to setup and run the LB_Colloids from config file. This is the also the preferred user interaction method to run simulations when working with python.

Parameters **config** (*colloid_IO.config*) – object or list of colloid_IO.Config objects

6.2 LB Colloid mathematics

The colloid mathematics module is responsible for calculating forces that define colloid-surface and colloid-colloid interactions. Objects exist within this module for each physical and physio-chemical interaction force that is simulated within a colloid particle tracking simulation. Forces are converted to a 'velocity-like' representation to account for the change in position due to applied forces. Calculations are performed for Brownian motion, drag forces, gravity, bouyancy, electrostatic repulsion, and attrative chemical interaction energies. A detailed review of the mathematical theory involved will not be described in this document. The base equations for each interaction are listed within the user documentation. The python version of colloid interactions will be migrated to FORTAN for computational efficiency. User functionality and object calls are not expected to change after migration.

6.2.1 API documentation

lb_colloids.**ColloidMath**

alias of *lb_colloids.Colloids.Colloid_Math*

ColloidMath is the primary mathematics module for Colloid Simulations. This module contains both Physical and Chemical formulations of colloid forces within a porous media. The DLVO and ColloidColloid classes contain complex formulations of chemical interaction forces. Other classes contain physical force calculations or provide mathematical conversion from Force to a Velocity like unit that can be used to recover the change in colloid position. Users should not have to call these classes directly when running a model.

Basic examples of how these modules are called assume that a user has already provided input to the lbIO.Config() module and the appropriate dictionaries have been built. For more information on required parameters and keywords please inspect API documentation for each respective class. Docstrings also provide basic mathematical relationships for each class.

```
>>> from lb_colloids import ColloidMath as cm
>>>
>>> grav = cm.Gravity(**PhysicalDict)
>>> grav.gravity  # returns the gravity force on a colloid
>>> bouy = cm.Bouyancy(**PhysicalDict)
>>> bouy.bouyancy  # returns the bouyancy force on a colloid
>>> gap = cm.Gap(xarr, yarr, **PhysicalDict)
>>> brownian = cm.Brownian(gap.f1, gap.f2, **PhysicalDict)
>>> brownian.brownian_x  # returns brownian force in the x-direction on a colloid
>>> brownian.brownian_y
>>> drag = cm.Drag(ux, uy, gap.f1, gap.f2, gap.f3, gap.f4, **PhysicalDict)
>>> drag.drag_x  # returns an array of drag forces in the x-direction
>>> dlvo = cm.DLVO(xarr, yarr, **ChemicalDict)
>>> dlvo.EDLx  # returns an array of electric double layer forces in the x-direction
>>> dlvo.LewisABy  # returns an array of lewis acid base forces in the y-direction
>>> dlvo.LVDWx  # returns an array of lifshitz-van der waals forces in the x-direction
>>> colcol = cm.ColloidColloid(xarr, **ChemicalDict)
>>> colcol.x_array  # returns an array of dlvo forces for colloid-colloid interactions
>>> colcol.update(Singleton.positions)  # updates the class to generate new colloid-
↪colloid interaction arrays
```

class `lb_colloids.Colloids.Colloid_Math.`**`Bouyancy`**`(`***kwargs*`)`

Class to estimate the gravitational force experienced by a colloid. Gravity is applied as a positive value to maintain vector direction.

$$F^b = \frac{4\pi a_c^3 \rho_w g}{3}$$

Keyword Arguments

- **`rho_water`** (*`flaot`*) – density of water kg/m^3. Default is 997.
- **`rho_colloid`** (*`float`*) – particle density of a colloid in kg/m^3. Default is 2650.
- **`ac`** (*`float`*) – colloid radius in m. Default is 1e-6.

Returns bouyancy (float) Bouyancy force that a colloid experiences

class `lb_colloids.Colloids.Colloid_Math.`**`Brownian`**`(`*f1*, *f4*, ***kwargs*`)`

Class to estimate brownian forces on colloids. Uses the relationships outlined in Qui et. al. 2010 where

$$F_x^B = \xi \sqrt{\frac{2D_0}{f_1 dt}} G(0,1)$$

$$F_y^B = \xi \sqrt{\frac{2D_0}{f_4 dt}} G(0,1)$$

Parameters

- **`f1`** (*`np.ndarray`*) – Drag force correction term [Gao et. al. 2010. Computers and Math with App]
- **`f4`** (*`np.ndarray`*) – Drag force correction term [Gao et. al. 2010]

Keyword Arguments

- **`ac`** (*`float`*) – Colloid radius. Default 1e-6
- **`viscosity`** (*`float`*) – Dynamic viscosity of water. Default 8.9e-4 Pa S.
- **`T`** (*`float`*) – Absolute temperature in K. Default is 298.15

Returns brownian_x: (np.ndarray) array of browian (random) forces in the x direction [Qiu et. al 2011.]

Returns brownian_y: (np.ndarray) array of browian (random) forces in the y direction [Qiu et. al 2011.]

Attributes

class `lb_colloids.Colloids.Colloid_Math.`**`ColloidColloid`**`(`*arr*, ***kwargs*`)`

The ColloidColloid class is used to calculate colloid-colloid interaction forces using the formulations presented in Liang 2008, Qui 2012, and Israelichevi 1996. Attractive forces are based on the Liang & Israelichevi formulation. Electric doulbe layer forces are calculated using Qui et. al. 2012.

The ColloidColloid object also provides methods to update ColloidColloid force array fields during model streaming.

Colloid colloid interaction energies are calculated via:

$$\Phi^{EDL} = 32\pi\epsilon_0\epsilon_r a_c(\frac{kT}{Ze})^2 * [tanh(\frac{Ze\psi_c}{4kT})]^2 * exp(-\kappa h)$$

$$A_H = 384\pi\frac{\psi_c^2 hkTI^*}{\kappa^2}exp(-\kappa h)$$

$$\Phi^A = -\frac{A_H}{6}[\frac{2a_c^2}{h^2+4a_ch} + \frac{2a_c^2}{(h+2a_c)^2} + ln(1-\frac{4a_c^2}{(h+2a_c)^2})]$$

Parameters

- **arr** (*np.ndarray*) – A np.ndarray that represents the shape of the colloid domain
- **resolution** (*float*) – Colloid model resolution

Keyword Arguments

- **valence** (*dict*) – Valences of all species in solution. (Optional)
- **concentration** (*dict*) – Concentration of all species in solution (Optional)
- **zeta_colloid** (*float*) – Measured_zeta potential of colloid (Reccomended). Default -40.5e-3 Na-Kaolinite Colloid [Chorom 1995. Eur. Jour. of Soil Science]
- **zeta_surface** (*float*) – Bulk_zeta potential of porous media (Reccomended). Default -60.9e-3 Glass bead media [Ducker 1992, Langmuir V8]
- **I** (*float*) – Ionic strength of simulated solution (Reccomended). Default 1e-3 M
- **ac** (*float*) – Colloid radius in meters. Default 1e-6 m.
- **epsilon_r** (*float*) – Relative dielectric permativity of water. (Optional) Default 78.304 @ 298 K [Malmberg and Maryott 1956. Jour. Res. Nat. Beau. Std. V56(1)
- **sheer_plane** (*float*) – Equivelent to the thickness of one layer of water molecules. (Optional) Default 3e-10 m [Interface Science and Technology, 2008. Volume 16 Chapter 3]
- **T** (*float*) – Temperature of simulation fluid. Default 298.15 k

Attributes

Methods

colloid_potential
Property method that generates colloid potential

debye
Property method to calculate the debye length on the fly

ionic_strength
Property method to calculate ionic_strength on the fly

positions
Property method to generate colloid positions if they are not stored yet

update (*colloids*)
Updates the colloidal positions and force arrays for the system

Parameters **colloids** (*list*) – (list, <class: Colloids.LB_Colloid.Colloid)

x
Property method to generate the x force array for colloid-colloid interaction

x_array
Property method to generate the full x force array for colloid-colloid interaction

x_distance_array
Generates an angular distance array in the x direction.

y
Property method to generate or return the y force array for colloid-colloid interaction

y_array
Property method to generate the full y force array for colloid-colloid interaction

y_distance_array
Generates an angular distance array in the y direction

class lb_colloids.Colloids.Colloid_Math.**DLVO**(*xarr*, *yarr*, ***kwargs*)

Class method to calculate vectorized DLVO force arrays for colloid surface interaction using methods outlined in Qui et. al. 2011 and Liang et. al. 2008? *Check this later*

Parameterization of this class is handled primary through the ChemistryDict by **kwargs

Mathematics used in calcuation of DLVO interaction energies are:

$$\frac{1}{\kappa} = (\frac{\epsilon_r \epsilon_0 kT}{e^2 N_A I^*})^{\frac{1}{2}}$$

$$\Phi^{EDL} = \pi \epsilon_0 \epsilon_r a_c (2\psi_s \psi_c ln(\frac{1 + exp(-\kappa h)}{1 - exp(-\kappa h)}) + (\psi_s^2 + \psi_c^2) ln(1 - exp(-2\kappa h)))$$

Parameters

- **xarr** (*np.ndarray*) – Physical distance from solid boundaries in the x direction
- **yarr** (*np.ndarray*) – Physical distance from solid boundaries in the y direction

Keyword Arguments

- **valence** (*dict*) – Valences of all species in solution. (Optional)
- **concentration** (*dict*) – Concentration of all species in solution (Optional)
- **zeta_colloid** (*float*) – Measured_zeta potential of colloid (Reccomended). Default -40.5e-3 Na-Kaolinite Colloid [Chorom 1995. Eur. Jour. of Soil Science]
- **zeta_surface** (*float*) – Bulk_zeta potential of porous media (Reccomended). Default -60.9e-3 Glass bead media [Ducker 1992, Langmuir V8]
- **I** (*float*) – Ionic strength of simulated solution (Reccomended). Default 1e-3 M
- **ac** (*float*) – Colloid radius in meters. Default 1e-6 m.
- **epsilon_r** (*float*) – Relative dielectric permativity of water. (Optional) Default 78.304 @ 298 K [Malmberg and Maryott 1956. Jour. Res. Nat. Beau. Std. V56(1)
- **sheer_plane** (*float*) – Equivelent to the thickness of one layer of water molecules. (Optional) Default 3e-10 m [Interface Science and Technology, 2008. Volume 16 Chapter 3]
- **T** (*float*) – Temperature of simulation fluid. Default 298.15 k
- **lvdwst_colloid** (*float*) – Lifshits-van der Waals surface tension component from colloid. (Reccomended) Default is 39.9e-3 J/m**2 [Giese et. al. 1996, Jour. Disp. Sci. & Tech. 17(5)]

- **lvdwst_solid** (*float*) – Lifshits-van der Waals surface tension component from solid. (Reccomended) Default is 33.7e-3 J/m**2 [Giese et. al. 1996]
- **lvdwst_water** (*float*) – Lifshits-van der Waals surface tension component from water. (Reccomended) Default is 21.8e-3 J/m**2 [Interface Science and Technology, 2008. V16(2)]
- **psi+_colloid** (*float*) – Lewis acid base electron acceptor parameter. (Reccomended) Default is 0.4e-3 J/m**2 [Giese et. al. 1996]
- **psi+_solid** (*float*) – Lewis acid base electron acceptor parameter. (Reccomended) Default is 1.3e-3 J/m**2 [Giese et. al. 1996]
- **psi+_water** (*float*) – Lewis acid base electron acceptor parameter. (Reccomended) Default is 25.5e-3 J/m**2 [Interface Science and Technology, 2008. V16(2)]
- **psi-_colloid** (*float*) – Lewis acid base electron donor parameter. (Reccomended) Default is 34.3e-3 J/m**2 [Giese et. al. 1996]
- **psi-_solid** (*float*) – Lewis acid base electron donor parameter. (Reccomended) Default is 62.2e-3 J/m**2 [Giese et. al. 1996]
- **psi-_water** (*float*) – Lewis acid base electron donor parameter. (Reccomended) Default is 25.5e-3 J/m**2 [Interface Science and Technology, 2008. V16(2)]
- **xvArr** (*np.ndarray*) – Array of vector directions.This array is applied to properly represent attractive and repulsive forces
- **yvArr** (*np.ndarray*) – Array of vector directions.This array is applied to properly represent attractive and repulsive forces

Returns EDLx (np.ndarray) vectorized np.array of electric-double-layer force values in the x-direction

Returns EDLy (np.ndarray) vectorized np.array of electric-double-layer force values in the y-direction

Returns LVDWx (np.ndarray) vectorized np.array of lifshitz-van-der-walls force values in the x-direction

Returns LVDWy (np.ndarray) vectorized np.array of lifshitz-van-der-walls force values in the y-direction

Returns LewisABx (np.ndarray) vectorized np.array of lewis acid base force values in the x-direction

Returns LewisABy (np.ndarray) vectorized np.array of lewis acid base force values in the y-direction

Attributes

Methods

attractive_x

Calculates the combined attractive force between colloid surface based upon Liang et. al. 2008

Returns np.ndarray

attractive_y

Calculates the combined attractive force between colloid surface based upon Liang et. al. 2008

Returns np.ndarray

ionic(*valence*, *concentration*)

Calculates the 2*I from user supplied valence and concentraitons

$$I^* = \sum_i Z_i^2 M_i$$

Parameters

- **valence** (*dict*) – Dictionary of chemical species, valence
- **concentration** (*dict*) – Dictionary of chemical species, concentration

Returns I (float) 2*ionic stength

k_debye

Method to calculate Debye length

Returns Debye length (float)

class lb_colloids.Colloids.Colloid_Math.**Drag**(*ux*, *uy*, *f1*, *f2*, *f3*, *f4*, ***kwargs*)

Class to calculate colloidal drag forces from fluid velocity arrays. Based from calculations outlined in Gao et, al 2010 and Qui et. al. 2011.

$$F_x^D = \frac{\xi}{f_4}(f_3 u_x - V_x)$$

$$F_y^D = \xi(f_2 u_y - \frac{V_y}{f_1})$$

Parameters

- **ux** (*np.ndarray*) – fluid velocity in the x-direction
- **uy** (*np.ndarray*) – fluid velocity in the y-direction
- **Vx** (*np.ndarray*) – colloid velocity in the x-direction
- **Vy** (*np.ndarray*) – colloid velocity in the y-direction
- **f1** (*np.ndarray*) – Hydrodynamic force correction term [Gao et. al. 2010.]
- **f2** (*np.ndarray*) – Hydrodynamic force correction term [Gao et. al. 2010.]
- **f3** (*np.ndarray*) – Hydrodynamic force correction term [Gao et. al. 2010.]
- **f4** (*np.ndarray*) – Hydrodynamic force correction term [Gao et. al. 2010.]

Keyword Arguments

- **ac** (*float*) – Colloid radius. Default is 1e-6 m
- **viscosity** (*float*) – Dynamic fluid viscosity of water. Default 8.9e-4 Pa S
- **rho_colloid** (*float*) – Colloid particle density. Default $2650 kg/m^3$
- **rho_water** (*float*) – Water density. Default $997 kg/m^3$

Returns drag_x (np.ndarray) non-vectorized drag forces in the x-direction

Returns drag_y: (np.ndarray) non-vectorized drag forces in the y-direction

Attributes

Methods

drag_x

return – drag force array in the x direction

update(*vx, vy*)

Updates the colloid velocity array for producing drag forces :param vx: :param vy:

class lb_colloids.Colloids.Colloid_Math.**ForceToVelocity**(*forces, **kwargs*)

Class that calculates a "velocity-like" value from force arrays

Parameters **forces** (*np.ndarray*) – Array of forces felt by a colloid

Keyword Arguments

- **ts** (*float*) – Physical time step value
- **rho_colloid** (*float*) – Colloid particle density, default $2650kg/m^3$
- **ac** (*float*) – colloid radius, default 1e-6 m

Returns velocity (np.array, np.float) Array of "velocities" calculated from forces

class lb_colloids.Colloids.Colloid_Math.**Gap**(*xarr, yarr, **kwargs*)

Class that calculates the non-dimensional gap distance between colloid and surface.

This class also calculates hydrodynamic force correction terms outlined in Gao et. al. 2010. Note: Passing a np.nan value into here can return an overflow warning!

$$f_1(\bar{h}) = 1.0 - 0.443exp(-1.299\bar{h}) - 0.5568exp(-0.32\bar{h}^{0.75})$$

$$f_2(\bar{h}) = 1.0 + 1.455exp(-1.2596\bar{h}) - 0.7951exp(-0.56\bar{h}^{0.50})$$

$$f_3(\bar{h}) = 1.0 - 0.487exp(-5.423\bar{h}) - 0.5905exp(-37.83\bar{h}^{0.50})$$

$$f_4(\bar{h}) = 1.0 - 0.35exp(-0.25\bar{h}) - 0.40exp(-10\bar{h})$$

Parameters

- **xarr** (*np.ndarray*) – Array of x-distances to nearest solid surface
- **yarr** (*np.ndarray*) – Array of y-distances to nearest solid surface

Keyword Arguments **ac** (*float*) – Radius of a colloid. Default is 1e-6

Returns f1 (np.ndarray) Drag force correction term [Gao et al 2010]

Returns f2 (np.ndarray) Drag force correction term [Gao et al 2010]

Returns f3 (np.ndarray) Drag force correction term [Gao et al 2010]

Returns f4 (np.ndarray) Drag force correction term [Gao et al 2010]

Methods

class lb_colloids.Colloids.Colloid_Math.**Gravity**(***kwargs*)

Class to generate the estimated gravitational force experienced by a colloid

$$F^G = \frac{-4\pi a_c^3 \rho_c g}{3}$$

Keyword Arguments

- **rho_colloid** (*float*) – Particle density of a colloid in kg/m^3. Default is 2650.
- **ac** (*float*) – colloid radius in m. Default is 1e-6

Returns gravity (float) Gravitational force that a colloid experiences

class lb_colloids.Colloids.Colloid_Math.**Velocity**(*LBx*, *LBy*, *velocity_factor*, ***kwargs*)

Class that dimensionalizes LB velocity from non-dimensional lattice Boltzmann units

Parameters

- **LBx** (*np.ndarray*) – Array of Lattice Boltzmann velocities in the x-direction
- **LBy** (*np.ndarray*) – Array of Lattice Boltzmann velocities in the y-direction
- **velocity_factor** (*float*) – LB to physical velocity conversion factor. Default is 1

Keyword Arguments

- **ts** (*float*) – Time step value, default is 1.
- **scale_lb** (*float*) – Scale the dimensionalized velocity from lattice Boltzmann. Use with caution. Default is 1

Returns xvelocity (np.array, np.float) array of dimensionalized velocities in the x-direction

Returns yvelocity (np.array, np.float) array of dimensionalized velocities in the y-direction

6.3 LB Colloid Input Output

Colloid simulation input/output (*IO*) modules have been developed to facilitate end user functionality, such as reading configuration files, building configuration files with python, data processing, and visualization. Output modules have been developed for the user familiar with python. These modules include methods for reading HDF5 output, ascii output, performing statistical analysis on data, recovering macroscopic ADE parameters, and performing data visualization using the matplotlib library.

|todo: find a place to include run_model.py

6.3.1 API documentation

lb_colloids.**cIO**

alias of *lb_colloids.Colloids.Colloid_IO*

Basic Input and Output control for Colloid Simulation models are hosted within Colloid_IO.py. Config and ColloidConfig are classes to set up model dictionaries can be passed along to the main simulation routines. ColloidConfig is a backend method for the super user. This provides a simple overridden dictionary class that is able to build configuration files and a list object that can be passed directly to the Config class.

Importing classes from this module follows the notation:

```
>>> from lb_colloids import cIO
>>>
>>> config = cIO.Config("Colloids.config")
>>> cc = ColloidsConfig()
>>> cc['I'] = 0.01
>>> # user must supply all required parameters to the ColloidsConfig dictionary, or␣
↪an AssertionError will be raised
>>> cc_config = cc.config
>>> config = cIO.Config(cc_config)
```

class lb_colloids.Colloids.Colloid_IO.**ColloidsConfig**

OO class to build config files, or build a list that the Config class will recognize and parse Recomended setup method for the super user who is looping many models. Facilitates easy sensitivity analysis, etc. . . .

Class uses a dictionary override to set parameters to the class, and writes them out as a list or as a configuration file

example class usage:

```
>>> from lb_colloids import cIO
>>> cconfig = cIO.ColloidsConfig()
>>> cconfig['I'] = 0.1
>>> cconfig['ncols'] = 500
>>> x = cconfig.config  # returns a formatted list that imitates a colloids␣
↪configuration file
>>> config = cIO.Config(x)
```

Attributes

Methods

chemical_parameters

Current user supplied chemical parameters

config

Property method that creates the config list on the fly for the user from the overridden dictionary

Returns self.__config (list) formatted configuration file list

model_parameters

Current user supplied model parameters

output_control_parameters

Current user supplied output control parameters

physical_parameters

Current user supplied physical parameters

valid_chemical_parameters

List of valid chemical parameters

valid_model_parameters

List of valid model parameters

valid_output_control_parameters

List of valid output control parameters

valid_physical_parameters
List of valid physical parameters

write(*fname*)
Writes a configuration file with user supplied parameters to file.

Parameters **fname** (*str*) – Configuration file name to write

class lb_colloids.Colloids.Colloid_IO.**Config**(*fname*)
Class to open and parse configuration files for LB-Colloids. Many data checks have been implemented to look for consistancy in data type and configuration variable for each input block.

Parameters **fname** (*str*) – Configuration file name ex. Model.config. This class can also accepts

a list of configuration variables that have been set up by cIO.ColloidsConfig()

Returns model_parameters (dict) Dictionary of necessary model parameters

Returns physical_parameters (dict) Dictionary of optional physical parameters

Returns chemical_parmaeters (dict) Dictionary of chemical parameter options

Returns output_control (dict) Dictionary of output control options

Methods

add_universal_parameters(*Dict*)
Add common model parameters to other dictionaries if present. Necessary for parameterization by kwargs of physics and chemistry.

Parameters **Dict** (*dict*) – dictionary to add set of universal paramameters from the required parameters

Returns Dict (dict)

adjust_pname(*pname*)
Adjusts parameter name from configuration file name to LB-Colloids name for a limited number of parameters. Sets all other parameters as lowercase to follow PEP-8

Parameters **pname** (*str*) – parameter name

check_if_valid(*blockname, pname, validparams*)
Method checks if a specific parameter is valid for the block it was supplied to in the configuration file.

Parameters

- **blockname** (*str*) – Name of the input block
- **pname** (*str*) –
- **validparams** (*tuple*) – tuple of valid parameter names for the specific input block

check_model_parameters(*ModelDict*)
Check for required parameters. If not present inform the user which parameter(s) are needed

Parameters **ModelDict** (*dict*) – Checks the model_dict for all required parameters.

Raises **AssertionError** – If all required parameters are not supplied

chemical_parameters()
Reads the CHEMICAL PARAMETERS block of the configuration file and creates the chemical_dict that passes optional parameters to the chemical force calculations in the Colloid_Math.py module

get_block(*blockname*)
Method to isolate an input block from a configuration file for parsing

Parameters **blockname** (*str*) – Blockname of an input block in the configuration file.

Returns (list) returns a list of parameters contained within the block

model_parameters()
Reads the MODEL PARAMETERS block of the configuration file and creates the model_dict which contains the required parameters to run LB-Colloid

output_control()
Reads the OUTPUT CONTROL block of the configuration file and creates the output_dict that passes optional parameters to control model output

parametertype(*parameter*)
Method takes a parameter string, splits it, and sets the parameter type

Parameters **parameter** (*str*) – String of "<pname>: <param>"

Returns (pname, param) (tuple) parameter name, value

physical_parameters()
Reads the PHYSICAL PARAMETERS block of the configuration file and creates the physics_dict that passes optional parameters to the physical force calculations in the Colloid_Math.py module

class lb_colloids.Colloids.Colloid_IO.**HDF5WriteArray**(*ux*, *uy*, *colloidcolloid*, *model_dict*, *chemical_dict*, *physical_dict*)
Class to write chemical and physical force arrays to the Model HDF5 object for later use in data processing and analysis.

Parameters

- **ux** (*np.ndarray*) – Dimensionalized fluid velocity in the x-direction
- **uy** (*np.ndarray*) – Dimensionalized fluid velocity in the y-direction
- **colloidcolloid** (Colloid_Math.ColloidColloid) – ColloidColloid object
- **model_dict** (*dict*) – The supplied model dict from parameterization
- **chemical_dict** (*dict*) – The supplied chemical dict used for parameterization
- **physical_dict** (*dict*) – The supplied physical dict used for parameterization

class lb_colloids.Colloids.Colloid_IO.**Output**(*fi*, ***kwargs*)
Output class for writing formatted ASCII files. This class generates <.endpoint>, <.timeseries> and <.pathline> files. All keywords are required.

Parameters **fi** (*str*) – filename of the output.

Keyword Arguments

- **overwrite** (*bool*) – Determines if file is overwritten, or appended to. useful for generating new pathline and timeseries files
- **ts** (*float*) – Physical time step

- **lbres** (*float*) – Lattice Boltzmann resolution
- **gridref** (*float*) – Grid refinement factor
- **ncols** (*int*) – number of colloids simulated
- **xlen** (*int*) – length of the xdomain in pixels
- **ylen** (*int*) – length of the ydomain in pixels
- **mean_ux** (*float*) – mean fluid velocity in x-direction
- **mean_uy** (*float*) – mean fluid velocity in y-direction
- **continuous** (*int*) – flag for continuous release of colloids

Methods

write_output (*timer, colloids, pathline=True*)

Set up and write colloid streaming output to an ASCII file

Parameters

- **timer** (TrackTime) – Model TrackTime instance
- **colloids** (*list*) – Colloid simulation list containing LB_Colloid.Colloid objects
- **pathline** (*bool*) – Flag to indicate if an endpoint or pathline/timeseries is being written.

write_single_colloid (*timer, colloid*)

Method to write a single colloid to an endpoint file upon breakthrough :param TrackTime timer: Model TrackTime instance :param LB_Colloid.Colloid colloid:

lb_colloids.**ColloidOutput**

alias of *lb_colloids.Colloids.Colloid_output*

The Colloid_output module contains classes to read LB Colloids simulation outputs and perform post processing. Many classes are available to provide plotting functionality. ModelPlot and CCModelPlot are useful for visualizing colloid-surface forces and colloid-colloid forces respectively.

example import of the Colloid_output.py module is as follows

```
>>> from lb_colloids import ColloidOutput
>>> import matplotlib.pyplot as plt
>>>
>>> hdf = "mymodel.hdf5"
>>> mp = ColloidOutput.ModelPlot(hdf)
>>> # model plot accepts matplotlib args and kwargs!!!
>>> mp.plot('edl_x', cmap='viridis')
>>> plt.show()
```

class lb_colloids.Colloids.Colloid_output.**ADE** (*filename, nbin=1000*)

Class to calculate macroscopic advection dispersion equation parameters for field scale model parameterization

Class needs to be re-named and updated to CDE equation

Parameters

- **filename** (*str*) – ascii output file name from colloid model
- **nbin** (*int*) – number of timesteps to bin a pdf for calculation

Methods

reset_pdf(*nbin*, *normalize=False*)
User method to reset values based on changing the pdf bin values

Parameters

- **nbin** (*int*) – number of timesteps to bin a pdf for calculation
- **normalize** (*bool*) – flag to calculate pdf by residence time or end time

solve_jury_1991(*D=0.01*, *R=0.01*, *ftol=1e-10*, *max_nfev=1000*, ***kwargs*)
Scipy optimize method to solve least sqares for jury 1991. Pulse flux.

Parameters

- **D** (*float*) – Diffusivity initial guess. Cannot be 0
- **R** (*float*) – Retardation initial guess. Cannot be 0
- **ftol** (*float*) – scipy function tolerance for solution
- **max_nfev** (*int*) – maximum number of function iterations
- ****kwargs** – scipy least squares kwargs

Returns scipy least squares dictionary. Answer in dict['x']

solve_van_genuchten_1986(*D=0.01*, *R=0.01*, *ftol=1e-10*, *max_nfev=1000*, ***kwargs*)
Scipy optimize method to solve least squares for van genuchten 1986. Miscable displacement.

Parameters

- **D** (*float*) – Diffusivity initial guess. Cannot be 0
- **R** (*float*) – Retardation initial guess. Cannot be 0
- **ftol** (*float*) – scipy function tolerance for solution
- **max_nfev** (*int*) – maximum number of function iterations
- ****kwargs** – scipy least squares kwargs

Returns scipy least squares dictionary. Answer in dict['x']

class lb_colloids.Colloids.Colloid_output.**ASCIIReader**(*filename*)
Class to read in text based output files <endpoint, timestep, pathline> to a pandas dataframe

Parameters **filename** (*str*) – output filename (ie. endpoint, timestep, or pathline)

Methods

read_ascii(*filename*)
Method to read endpoint file data from from ascii files for LB-Colloids Sets data to pandas dataframe

Parameters **filename** (*str*) – colloid model output filename (ie. endpoint, timestep, or pathline)

read_header(*filename*)
Method to read the header from ascii output files for LB-Colloids

Parameters **filename** (*str*) – colloid model output filename (ie. endpoint, timestep, or pathline)

class `lb_colloids.Colloids.Colloid_output.`**Breakthrough**(*filename*)

Class to prepare and plot breakthrough curve data from endpoint files.

Parameters **filename** (*str*) – <>.endpoint file

Variables

- **df** – (pandas DataFrame): dataframe of endpoint data
- ***resolution*** – (float): model resolution
- **timestep** – (float): model timestep
- **continuous** – (int): interval of continuous release, 0 means pulse
- **ncol** – (float): number of colloids per release in simulation
- **total_ncol** – (int): total number of colloids in simulation

Attributes

Methods

breakthrough_curve

Property method that performs a dynamic calculation of breakthrough curve data

plot(*time=True*, **args*, ***kwargs*)

Convience method to plot data into a matplotlib chart.

Parameters

- **time** (*bool*) – if true x-axis is time, false is nts
- ***args** – matplotlib args for 1d charts
- ****kwargs** – matplotlib keyword arguments for 1d charts

plot_pv(**args*, ***kwargs*)

Method to plot breakthrough data with pore volumes (non-dimensional time)

Parameters

- ***args** – matplotlib args for 1d plotting
- ****kwargs** – matplotlib kwargs for 1d plotting

pore_volume_conversion()

Method to retrieve the pore volume calculation conversion for plotting colloids.

class `lb_colloids.Colloids.Colloid_output.`**CCModelPlot**(*hdf5*)

Class to query colloid-colloid interactions and plot data as 1d or as a meshgrid object More sophisticated than standard ModelPlot

Parameters **hdf5** (*str*) – hdf5 file name

Attributes

Methods

get_data (*key*)
Method to return data by key

Parameters **key** (*str*) – valid model key

get_data_by_path (*path*)
Method to return data by hdf5 path

Parameters **path** (*str*) – valid HDF5 data path

keys
Property method to return valid keys to obtain data

plot (*key*, **args*, ***kwargs*)
Plotting method for 1d colloid-colloid dlvo profiles

Parameters

- **key** (*str*) – valid data key
- ***args** – matplotlib plotting args
- ****kwargs** – matplotlib plotting kwargs

plot_mesh (*key*, **args*, ***kwargs*)
Plotting method for 2d representation of colloid-colloid dlvo profiles.

Parameters

- **key** (*str*) – valid data key
- ***args** – matplotlib plotting args
- ****kwargs** – matplotlib plotting kwargs

class lb_colloids.Colloids.Colloid_output.**ColloidVelocity** (*filename*)
Method to return colloid velocity and statistics relating to colloid velocity for a simulation. Class needs to be rebuilt to work with timeseries and pathline files for a more precise velocity measurement

Parameters **filename** (*str*) – endpoint file name

Attributes

Methods

cv
return – coeficient of variance of colloid velocities

max
return – maximum colloid velocity

mean
return – mean colloid velocity

min
return – minimum colloid velocity

plot (**args*, ***kwargs*)
Method to plot distribution of velocities by colloid for array of velocity.

Parameters :param *args: matplotlib plotting args

:param **kwargs: matplotlib plotting kwargs

plot_histogram (*nbin=10*, *width=0.01*, **args*, ***kwargs*)
User method to plot a histogram of velocities using a bar chart.

Parameters

- **nbin** (*int*) – number of specific bins for plotting
- **width** (*float*) – matplotlib bar width.
- ***args** – matplotlib plotting args
- ****kwargs** – matplotlib plotting kwargs

stdev
return – standard deviation of colloid velocities

var
return – variance of colloid velocities

class lb_colloids.Colloids.Colloid_output.**DistributionFunction** (*filename*, *nbin=1000*)
Class to plot a probablity distribution function of colloid breakthrough from endpoint files.

Parameters

- **filename** (*str*) – <>.endpoint file name
- **nbin** (*int*) – number of bins for pdf calculation

Variables

- **df** – (pandas DataFrame): dataframe of endpoint data
- ***resolution*** – (float): model resolution
- **timestep** – (float): model timestep
- **continuous** – (int): interval of continuous release, 0 means pulse
- **ncol** – (float): number of colloids per release in simulation
- **total_ncol** – (int): total number of colloids in simulation
- **pdf** – (np.recarray) colloid probability distribution function

Methods

plot (*time=True*, **args*, ***kwargs*)
Method to plot data into a matplotlib chart.

Parameters

- **time** (*bool*) – if true x-axis is time, false is nts
- ***args** – matplotlib args for 1d charts
- ****kwargs** – matplotlib keyword arguments for 1d charts

plot_pv (**args, **kwargs*)

Method to plot pdf data with pore volumes (non-dimensional time)

Parameters

- ***args** – matplotlib args for 1d plotting
- ****kwargs** – matplotlib kwargs for 1d plotting

pore_volume_conversion ()

Method to retrieve the pore volume calculation conversion for plotting colloids.

reset_pdf (*nbin, normalize=False*)

Method to generate a probability distribution function based upon user supplied bin size.

Parameters

- **nbin** (*int*) – number of time steps to base bin on
- **normalize** (*bool*) – method to calculate pdf by residence time or end time

class lb_colloids.Colloids.Colloid_output.**Hdf5Reader** (*hdf5*)

Reader object to read in HDF5 stored outputs from colloid models. Contains a data_paths dictionary which allows the user to use keys to access data

Parameters **hdf5** (*str*) – LB-Colloid hdf5 file name

Attributes

Methods

get_data (*key*)

Method to retrieve hdf5 data by dict. key

Parameters **key** (*str*) – valid dictionary key from self.keys

Returns data <varies>

get_data_by_path (*path*)

Method to retrieve hdf5 data by specific hdf5 path

Parameters **path** (*str*) – hdf5 directory path to data

Returns data <varies>

keys

return – list of valid hdf5 data keys

class lb_colloids.Colloids.Colloid_output.**LBOutput** (*hdf5*)

Class to anaylze LB fluid/solid properties

Parameters **hdf** (*str*) – hdf5 output filename

Attributes

Methods

get_data(*key*)
Method to select data from hdf5 file based on key, instead of data path

Parameters **key** (*str*) – lattice boltzmann data key

Returns data

keys
return – Lattice boltzmann data keys

class lb_colloids.Colloids.Colloid_output.**ModelPlot**(*hdf5*)
Class to retrieve Colloid force arrays and plot for data analysis.

Parameters **hdf5** (*str*) – hdf5 file name

Attributes

Methods

get_data(*key*)
Get data method to view and analyze colloid force arrays

Parameters **key** (*str*) – valid dictionary key from self.keys

Returns data <varies>

get_data_by_path(*path*)
Method to retrieve hdf5 data by specific path

Parameters **path** (*str*) – hdf5 directory path to data

Returns data <varies>

plot(*key*, **args*, ***kwargs*)
Hdf array plotting using Hdf5Reader keys

Parameters

- **key** (*str*) – valid dictionary key from self.keys
- ***args** – matplotlib plotting args
- ****kwargs** – matplotlib plotting kwargs

plot_velocity_magnitude(*nbin=10*, **args*, ***kwargs*)
Method to create a quiver plot to display the magnitude and direction of velocity vectors within the system.

Parameters

- **nbin** (*int*) – refinement for quiver plotting
- ***args** – matplotlib plotting args
- ****kwargs** – matplotlib plotting kwargs

class lb_colloids.nam_file.**NamFile**(*nam_file*)

Class to read the nam file instance of a lattice boltzmann model and delegate the IO reading to the applicable lb_colloid config reading classes

Parameters **nam_file** (*str*) – nam file name

6.4 LB Colloid Setup (background classes)

The colloid setup module contains basic utilities to create and enforce boundary conditions. The computation of grid distance to the nearest solid interface is performed within this module. The normal user should not call these objects, as they are called internally by the ColloidModel instance.

6.4.1 API documentation

Colloid_Setup contains background classes and methods to prepare a model domain for colloid simulation. The user should not need to import or call methods directly from this module.

class lb_colloids.Colloids.Colloid_Setup.**GridArray**(*arr*, *gridres*, *gridsplit*, *solid=True*)

Gridarray class creates arrays of distances from pore spaces, corrects for interpolation effects at pore boundaries, and creates vector arrays that are later used to give direction to forces.

Parameters

- **bool) arr** (*(np.array,*) – Array of boolean porous media (segmented image array)
- **gridres** (*float*) – model resolution in meters
- **gridsplit** (*int*) – interpolation factor for refining grid mesh
- **solid** (*bool*) – solid phase boolean identifier, default=True

Attributes

gridx

return – (np.array, np.float) Array of distances from nearest solid phase in the x-direction

gridy

return – (np.array, np.float) Array of distances from nearest solid phase in the y-direction

vector_x

return – (np.array, np.float) Array of specific vector directions in the x-direction (-1 == left, 1 == right)

vector_y

return – (np.array, np.float) Array of specific vector directions in the y-direction (-1 == down, 1 == up)

class lb_colloids.Colloids.Colloid_Setup.**Hdf5Reader**(*hdf_name*)

Hdf5 reader class to grab results from lattice Boltzmann model runs to parameterize the colloid simulation. Consider moving this class to the Colloid_IO module

Parameters **HDF_name** (*str*) – lattice boltzmann hdf file name.

Variables

- **imarray** (*np.ndarray*) – binary image array defining model boundaries

- **uarry** (*np.ndarray*) – velocity array of [y, x]
- **yu** (*np.ndarray*) – y velocity array from lattice Boltzmann simulation
- **xu** (*np.ndarray*) – x velocity array from lattice Boltzmann simulation
- **mean_yu** (*float*) – mean velocity in the y direction
- **mean_xu** (*float*) – mean velocity in the x direction
- **velocity_factor** (*float*) – velocity dimensionalization factor

lb_colloids.Colloids.Colloid_Setup.**InterpV**(*LBv*, *gridsplit*, *img=False*)

Interpolation method for the lattice Boltzmann velocity array

Parameters

- **LBv** (*np.ndarray*) – lattice boltzmann velocity array with pore boundaries enforced
- **gridsplit** (*float*) – interpolation factor
- **img** (*bool*) – flag to indicate boolean image interpolation or velocity interpolation

Returns (np.ndarray) interpolated velocity array

lb_colloids.Colloids.Colloid_Setup.**LBVArray**(*LBv*, *img*)

Method to create a velocity array for use in the colloid simulation model

Parameters

- **LBv** (*np.ndarray*) – lattice boltzmann velocity array
- **img** (*np.ndarray*) – boolean image array

Returns np.ndarray of velocity

CHAPTER

SEVEN

PENETRABLE SPHERE (PSHPERE)

class lb_colloids.**PSphere**(*radius=20*, *porosity=0.5*, *dimension=256*, *sensitivity=0.08*)

Pshpere is a class that allows for the automated generation of synthetic porous media in two-dimensions. This approach can be expanded to three dimensions with some effort.

Parameters

- **radius** (*int*) – grain size radius
- **porosity** (*float*) – target porosity for porous media
- **dimension** (*int*) – the x and y dimension in pixels for the domain
- **sensitivity** (*float*) – a porosity sensitivity target. This is the allowable range of error for PShpere

Methods

The Pshpere module contains a class named PShpere which allows the user to generate synthetic porous media, and to get information about that porous media

A user can instantiate and use the PSphere() object as follows:

```
>>> from lb_colloids import PSphere
>>> img = Psphere(dimension=200, radius=20, porosity=0.375, sensitivity=0.01)
>>> # hydraulic radius can be calculated
>>> rh = img.calculate_hydraulic_radius(resolution=1e-06)
>>> # to get a copy of the porous media use
>>> matrix = img.matrix
>>> # save the image
>>> img.save("test_image.png")
```

class lb_colloids.utilities.psphere.**PSphere**(*radius=20*, *porosity=0.5*, *dimension=256*, *sensitivity=0.08*)

Pshpere is a class that allows for the automated generation of synthetic porous media in two-dimensions. This approach can be expanded to three dimensions with some effort.

Parameters

- **radius** (*int*) – grain size radius
- **porosity** (*float*) – target porosity for porous media
- **dimension** (*int*) – the x and y dimension in pixels for the domain
- **sensitivity** (*float*) – a porosity sensitivity target. This is the allowable range of error for PShpere

Methods

calculate_hydraulic_radius(*resolution*)

Calculates the hydraulic radius of the porous medium

Parameters **resolution** (*float*) – model resolution applied to image

Returns hydraulic radius of the image

check_percolation()

Modified sweep line technique that checks the porous media percolation Pshpere automatically calls this during porous media creation.

generate_plane()

Main method used to generate a porous media plane by PSphere, this should not be called by the user

iround(*val*)

Rounding routine to set index locations.

Parameters **val** (*float*) – floating point value

Return int

patch(*x*, *y*, *radius*)

The patch method is used to set grains into a porous media. Not to be called by the user!

Parameters

- **x** (*int*) – x index location
- **y** (*int*) – y index location
- **radius** (*int*) – grain radius

save_image(*image_name*)

Save method, to save an image to file!

Parameters **image_name** (*str*) – image path and name

static static_hydraulic_radius(*matrix*, *invert=True*)

Static method to calculate the hydraulic radius of a given porous medium

Parameters

- **matrix** (*np.ndarray*) – boolean array corresponding to porous media
- **invert** (*bool*) – inverts model, pore space needs to be set to True

Returns hydraulic radius of the image

static static_porosity(*matrix*, *invert=True*)

Static method to calculate the porosity of a given porous media

Parameters

- **matrix** (*np.ndarray*) – boolean array corresponding to porous media
- **invert** (*bool*) – inverts model, pore space needs to be set to True

Returns porosity of the image

static static_surface_area(*matrix*, *invert=True*)

Static method to calculate the non-dimensional surface area of a given porous medium

Parameters

- **`matrix`** (*`np.ndarray`*) – boolean array corresponding to porous media
- **`invert`** (*`bool`*) – inverts model, pore space needs to be set to True

Returns hydraulic radius of the image

PYTHON MODULE INDEX

INDEX