A DNS simulator for the dynamics of particles in fluids KAPSEL Version 5.12

User's Manual

KAPSEL Development Team

kapsel.dev@gmail.com

February 1, 2024

Contents

Chapter 1

Introduction

1.1 What is KAPSEL?

KAPSEL[1] is a software package for simulating disperse particle systems (systems in which particles move freely through a fluid). In simulating the motion of particles dispersed in a fluid, we must account not only for the impact of *direct* interactions between particles, but also for *indirect* interactions—known as *hydrodynamic interactions*—mediated by the motion of the fluid medium. Thus, reproducing the complicated trajectories of fluid-dispersed solid particles requires understanding how the fluid moves between them. One common computational strategy is direct numerical simulation (DNS), in which the Navier-Stokes equations are solved numerically to yield an accurate description of fluid motion on length scales smaller than the particle sizes, after which the hydrodynamic interactions between particles at a given time step may be determined from local fluid stresses. However, DNS calculations typically entail high computational costs, and this has spurred the development of alternative strategies that avoid numerically solving the Navier-Stokes equations. Examples include the methods of dissipative particle dynamics (DPD) and multi-particle collision dynamics (MPC), which model fluids as systems of particles satisfying the law of momentum conservation, and the Brownian dynamics (BD) and Stokesian dynamics (SD) approaches, in which hydrodynamic interactions between particles are determined without explicitly solving for fluid motion. In comparison to these and other simplified approaches, DNS has the great advantage of accurately describing fluid motion based on the fundamental governing equations; for this reason, the method is not restricted to modeling simple Newtonian fluids, but may be extended to handle more complex fluids.

In DNS, particles are represented not by infinitesimal points but by finite-volume bodies, and momentum exchanges between particles and fluids through the solid-fluid interface are properly accounted for. The task of modeling both the trajectories of arbitrarily-shaped solid bodies moving freely through a fluid and the flow of the fluid around those bodies is a coupled multiphysics problem; the most common strategy for solving this problem via numerical computation is the finite-element method (FEM), in which finite bodies are represented by discretized non-uniform meshes that are re-generated at each time step. Consequently, FEM models of disperse systems with many solid particles involve enormous numbers of mesh nodes that must be reconfigured at each computational step, resulting in exorbitant computational costs. To circumvent this difficulty while retaining the strengths of DNS modeling for multi-particle disperse systems, we independently developed a *smoothed-profile* (SP) method, in which sharp, discontinuous interfaces between particles and fluids are replaced by finite-thickness transition regions characterized by profile functions. The SP method, which allows simulations to achieve both high accuracy and high computational efficiency, is implemented in KAPSEL (Kyoto Advanced Particle Simulator for ELectrohydrodynamics), a simulator we developed for DNS modeling of disperse particle systems. This manual discusses the basic principles of KAPSEL, explains how to install the package, and presents a number of detailed sample computations. A detailed discussion of the SP method and its background may be found in Ref. [1].

1.2 What are KAPSEL's capabilities?

KAPSEL offers a wide range of analytical capabilities, which users may invoke and configure by preparing UDF input

[1]KAPSEL is a software package developed by members of the Soft Matter Engineering Laboratory in the Department of Chemical Engineering at Kyoto University . Early versions of KAPSEL, up to and including version 4, were distributed as open-source freeware, with users granted the freedom to define their own specifications subject to the provisions of a license agreement. Since version 5, KAPSEL has been closed-source commercial software, distributed only in the form of executable files available for purchase; viewing or editing the source code requires a separate non-disclosure agreement (NDA). The lead developer responsible for the current version and all previous versions of the KAPSEL core is Professor Ryoichi Yamamoto, and the copyright holders are Kyoto University and Professor Ryoichi Yamamoto.

files (typically given file names like `input.udf`). This is discussed in detail in later sections of this manual; here we briefly outline KAPSEL's primary simulation capabilities.

1.2.1 Simulating particles dispersed in a Newtonian fluid

Particles dispersed in a Newtonian fluid may be simulated by selecting `Navier_Stokes` as the `constitutive_eq`. This is useful for purposes such as predicting the trajectories of arbitrarily-shaped particles sinking due to gravity and characterizing the thermal-diffusion and agglomeration behavior of dispersed particles due to Brownian motion. See Section 3 for further details. Set `constitutive_eq` to `Navier_Stokes_FDM` to perform the analysis using a fluid solver based on the finite-difference method (FDM) instead of the default pseudo-spectral solver.

1.2.2 Simulating a disperse particle system in the presence of a shear flow

Particles dispersed in a Newtonian fluid under shear flow may be simulated by selecting `Shear_Navier_Stokes_Lees_Edwards` as the `constitutive_eq`. This is useful for predicting the trajectories of arbitrarily-shaped particles under shear flows or characterizing the rheological properties of disperse particle systems. See Section 4 for further details. Set `constitutive_eq` to `Shear_Navier_Stokes_Lees_Edwards_FDM` to perform the analysis using an FDM-based fluid solver.

1.2.3 Simulating charged colloidal particles dispersed in electrolytic solutions

Charged colloidal particles dispersed in an electrolytic solution may be simulated by selecting `Electrolyte` as the `constitutive_eq`. This is useful for predicting stable structures of charged colloidal systems and characterizing the electrophoretic properties of charged colloidal particles in steady-state (DC) or oscillating (AC) electric fields. See Section 5 for further details.

1.2.4 Simulating particles dispersed in two-component phase-separated fluids

Particles dispersed in two-component phase-separated fluids may be simulated by selecting `Navier_Stokes_Cahn_Hilliard_FDM` as the `constitutive_eq`. This is useful for purposes such as predicting fluid phase-separation structures or studying how particle distributions are affected by changes in fluid components or in factors such as the affinity between particles and fluid phase interfaces. To simulate particles dispersed in two-component phase-separated fluids under shear flow, set `constitutive_eq` to `Shear_NS_LE_CH_FDM`. This is useful for predicting phase-separation structures in the presence of shear flows or characterizing rheological properties of systems of particles dispersed in two-component phase-separated fluids. See Section 6 for further details.

1.2.5 Simulating microswimmers

Microswimmer particles dispersed in Newtonian liquids may be simulated by selecting `Navier_Stokes` as the `constitutive_eq`. This is useful for purposes such as predicting collective behavior resulting from the interaction among large numbers of microswimmers. See Section 7 for further details. Set `constitutive_eq` to `Navier_Stokes_FDM` to perform the analysis using an FDM-based fluid solver. Set `constitutive_eq` to `Navier_Stokes_Cahn_Hilliard_FDM` to simulate microswimmer particles dispersed in two-component phase-separated fluids.

1.2.6 Simulating Quincke rollers

Quincke rollers—self-propelled particles produced by rolling on a slab immersed in a Newtonian fluid—may be simulated by selecting `Navier_Stokes` as the `constitutive_eq`. This is useful for purposes such as predicting collective behavior resulting from the interaction among large numbers of Quincke rollers. See Section 8 for further details. Set `constitutive_eq` to `Navier_Stokes_FDM` to perform the analysis using an FDM-based fluid solver.

1.3 What is covered by this manual?

As a first step, Section 2 provides detailed instructions for installing and building the software and libraries needed to run KAPSEL.

The remaining six sections describe various types of simulations supported by KAPSEL. Each of these sections begins with a thorough review of the theoretical background and basic equations relevant to the given type of simulation, then provides a concise explanation of how to create UDF input files to set simulation parameters; finally, each section concludes with a series of detailed computational examples. The structure and format of the UDF input file is documented comprehensively in Appendix A.

Chapter 2

Installing and running KAPSEL

2.1 System-dependent procedures for installing and running KAPSEL

2.1.1 The KAPSEL runtime environment

The procedure for installing and running KAPSEL depends slightly on whether you are running on a Windows, Linux, or Mac system:

Linux

If you are using KAPSEL on a Linux system, you may skip to section 2.1.2 below.

Windows

To use KAPSEL on a Windows system, you must first install the Windows Subsystem for Linux (WSL)[1]. Open PowerShell or Windows Command Prompt in administrator mode by right-clicking and selecting ”Run as administrator”, enter the following command, then restart your machine. After installing WSL, skip to section 2.1.2 below.

```
    $ wsl --install
```

Mac

To use KAPSEL on MacOS, you must first install Xcode and the command line tools.

2.1.2 Installing OCTA

For tasks such as processing input parameters and visualizing output data, KAPSEL relies on an external user-interface module known as Gourmet. Gourmet is distributed as an internal component of the open-source package OCTA (a universal simulator for soft materials), which must be installed before using KAPSEL.

To install OCTA, simply visit `http://octa.jp/` to download and run an appropriate installer for your system.[2] The instructions below assume that `OCTA8.#` has been installed in `/usr/local/OCTA8#` (for Linux or Mac) or in `C:\OCTA8.#` (for Windows).[3]

2.1.3 Installing KAPSEL

First download the latest version of the KAPSEL package; this will be an archive file named `kapsel#.#.zip`.[4] Then extract the content of the archive and enter the top-level directory:

```
    $ unzip kapsel#.#.zip
    $ cd kapsel#.#
```

The top-level directory contains the following subdirectories:

./bin/ KAPSEL executable files for various platforms.

[1] https://learn.microsoft.com/en-us/windows/wsl

[2] Answers to questions regarding OCTA/GOURMET are available upon registering as a member of the OCTA-BBS website.

[3] Here the pound symbol (“#”) is to be replaced with numerals to indicate the version of your OCTA installation.

[4] Here the pound symbol (“#”) is to be replaced with numerals to indicate the version of your KAPSEL installation.

./Documents/ User manuals (including this document).

./Examples/ Example calculations illustrating various KAPSEL features.

./UDF/ UDF files (`define.udf`/`input.udf`) for using all features in the latest KAPSEL release.

Next, within the `./bin/` directory, find the executable file appropriate for your runtime environment and create a symbolic link in the KAPSEL installation directory.

Linux

```
    $ ln -s ./bin/linux/* .
```

Windows

From Ubuntu console window (not Windows command line)

```
    $ ln -s ./bin/linux/* .
```

Mac (Intel or Arm CPU)

```
    $ ln -s ./bin/macOS/* .
```

2.1.4 Validating your KAPSEL license

Commercial KAPSEL binaries will require a valid license to run. Two type of licenses are available, node-locked and cluster licenses, both of which support offline / online environments. Node-locked licenses, issued on an individual basis, will require a unique (anonymized) machine id before they can be activated. Likewise, cluster licenses require a unique environment id, together with the list of allowed user groups. These identifiers can be obtained by running the `key` command and checking the `MACHINE FINGERPRINT` code. The first time you run you should see output similar to the following

```
    $ ./key
    [INFO] System name : ...
    [INFO] Machine     : ...
    [INFO] Node name   : ...
    [INFO] Release     : ...
    [INFO] Version     : ...
    [INFO] Network interface (en0) : ...
    [INFO] Network interface (en1) : ...
    ...
    [INFO] Host domain : ...
    [INFO] Machine UUID: ...
    # [Machine Fingerprint]
 ↪  8bzm9b3xae00898539e754a3adba83bb11cdcea441d06401f5f3df350fcc7705
    # [Environ Fingerprint]
 ↪  22cca0e1fc66357d0aa8d238e7gg675a6d7ebcf323fdbbcef7e09c46c6259f17
    # [User Groups]  user_group_1 user_group_2 ...
    [ERROR] Environment variable KAPSEL_PUBLIC_KEY is missing
```

For node locked licenses, you will be asked to provide the `MACHINE FINGERPRINT` for all machines that will run KAPSELto the license provider. For cluster licenses (Linux only), you will be asked to provide the `ENVIRON FINGERPRINT`, together with the list of user groups you wish to enable on the cluster (e.g., user_group_1, user_group_2). To determine the appropriate `ENVIRON FINGERPRINT`, be sure to run the `key` program on a compute node, and not on the login node (i.e., you should submit the job as you would a KAPSEL job). You will receive a KAPSEL license key, together with an encrypted machine/license file from the license provider. The license key uniquely identifies your license, the license files are linked to individual machines/clusters and allow us to support offline environments. Note that both the key and the license are required to run KAPSEL . A license keys is like a personal password for KAPSEL, it should be treated as such, and should not be shared.

To allow KAPSEL to perform the license verification, you should set the appropriate environment variables. This can be done by editting and sourcing the `setvars.sh` script found in the `./bin` directory

```
$ source ./bin/setvars.sh
```

This will set the KAPSEL_ACCOUNT_ID, KAPSEL_PUBLIC_KEY, KAPSEL_LICENSE_KEY, and KAPSEL_LICENSE_PATH variables. Note that you should edit the script to use the KAPSEL license key you were issued as the KAPSEL_LICENSE_KEY variable, and set the KAPSEL_LICENSE_PATH to point to your license file. The format for the license key will be of the form key/your_kapsel_license_key. The license files are distributed as plain text files, with an encoded payload contained between the BEGIN/END MACHINE FILE header/footer

```
-----BEGIN MACHINE FILE-----
eyJlbmMiOiJsSTc4N0QwcGZua1RvRDVOSjFpRXlaU093Q09QQ0NOdktKZHpC
...
TlJWWGp6Ym5DRkF6V3lOU3NUeG9xZm9MV2FlWlhITEZnR21Ub2VBdz09Iiwi
YWxnIjoiYWVzLTI1Ni1nY20rZWQyNTUxOSJ9
-----END MACHINE FILE-----
```

You should never edit the KAPSEL_ACCOUNT_ID or KAPSEL_PUBLIC_KEY variables, nor the license files, as doing so will make it impossible to validate your license.

Alternatively you can execute the following directly in the console

```
$ export KAPSEL_ACCOUNT_ID="85e8531e-915a-4795-a287-2bec0d90ae5e"
$ export
↪ KAPSEL_PUBLIC_KEY="5fae6bb532c12ef70e1ce5f69b33ecd4fea8a522658b1204a6203ee5f101f608"
$ export KAPSEL_LICENSE_KEY="key/your_kapsel_licence_key"
$ export KAPSEL_LICENSE_PATH="./license.lic"
```

With a valid/activated license, and properly defined environment variables, you can now run KAPSEL. You should see the following message upon successful validation of your license and machine.

```
$ ./key
# [KAPSEL LICENSE VERIFICATION]
# [Machine Fingerprint]
↪ 8bzm9b3xae00898539e754a3adba83bb11cdcea441d06401f5f3df350fcc7705
# [Environ Fingerprint]
↪ 22cca0e1fc66357d0aa8d238e7gg675a6d7ebcf323fdbbcef7e09c46c6259f17
# [User Groups]  user_group_1 user_group_2 ...
[INFO] Try as Node license
[INFO] Importing license file
[INFO] Searching for file './license.lic' : YES
[OK] License file successfully imported!
[INFO] Verifying...
[OK] License file successfully verified!
[INFO] Decrypting...
[OK] License file successfully decrypted!
[INFO] Parsing...
[OK] License successfully parsed!
# [INFO] Node license imported
# [INFO] Attempting to connect to license server...
...
# [OK] Signature is valid!
# [OK] License key is valid!  (code = "VALID")
# [License ID] g9aa321f-a19f-424b-99a6-a1x9f393d69g
#
OK!
```

2.1.5 Testing KAPSEL

If executing KAPSEL produces the following command-line output, then KAPSEL is properly installed on your system.

```
$ cd UDF
$ ../kapsel -Iinput.udf -Ooutput.udf -Ddefine.udf -Rrestart.udf
#
# OMP RUNTIME :
# Number of processors        : 0
# Number of threads           : 8
# Max OMP threads             : 8
# Dynamic thread enabled?     : 0
# Nested parallelism enabled? : 1
#
#using input.udf as input
#using output.udf as output
```

```
    #using define.udf as definition
    #using restart.udf as restart
    # [KAPSEL LICENSE VERIFICATION]
    # [Machine Fingerprint]
↪   8bzm9b3xae00898539e754a3adba83bb11cdcea441d06401f5f3df350fcc7705
    # [Environ Fingerprint]
↪   22cca0e1fc66357d0aa8d238e7gg675a6d7ebcf323fdbbcef7e09c46c6259f17
    # [User Groups]  user_group_1 user_group_2 ...
    # [INFO] Node license imported
    # [INFO] Attempting to connect to license server...
    # [INFO] Online validation OK
    # [OK] Signature is valid!
    # [OK] License key is valid!  (code = "VALID")
    # [License ID] g9aa321f-a19f-424b-99a6-a1x9f393d69g
    #
       ...
    #output.udf end.
    #restart.udf end.
    #Simulation has ended!
    #Total Running Time (s):      24.77
    #                   (m):       0.41
    #                   (h):       0.01
    #Average Step Time  (s):       0.02
    #                   (m):       0.00
    #                   (h):       0.00
```

The `input.udf` file used in the above sample describes a simulation involving 5 heavy particles and 5 light particles sedimenting in a Newtonian fluid on a $32\times64\times32$ CFD[5] computational mesh. The simulation should complete within 1 minute; if it ran correctly, a file named `output.udf` will be produced.

When using the multi-core executable file, you may set environment variables before launching KAPSEL to specify the number of CPU cores used. In the following example, we request 8 CPU cores:

```
    $ export OMP_NUM_THREADS=8
```

2.1.6 Methods for visualizing simulation data

The use of GOURMET with python scripts allows visualization of various types of simulation data. Below we present some simple examples.

- Launch GOURMET
  - On Windows: `(OCTA install dir)\GOURMET\gourmet`
  - On Linux: `/usr/local/OCTA8#/GOURMET/gourmet`
  - On Mac: `/usr/local/OCTA8#/GOURMET/gourmet`

- Open the `output.udf` file

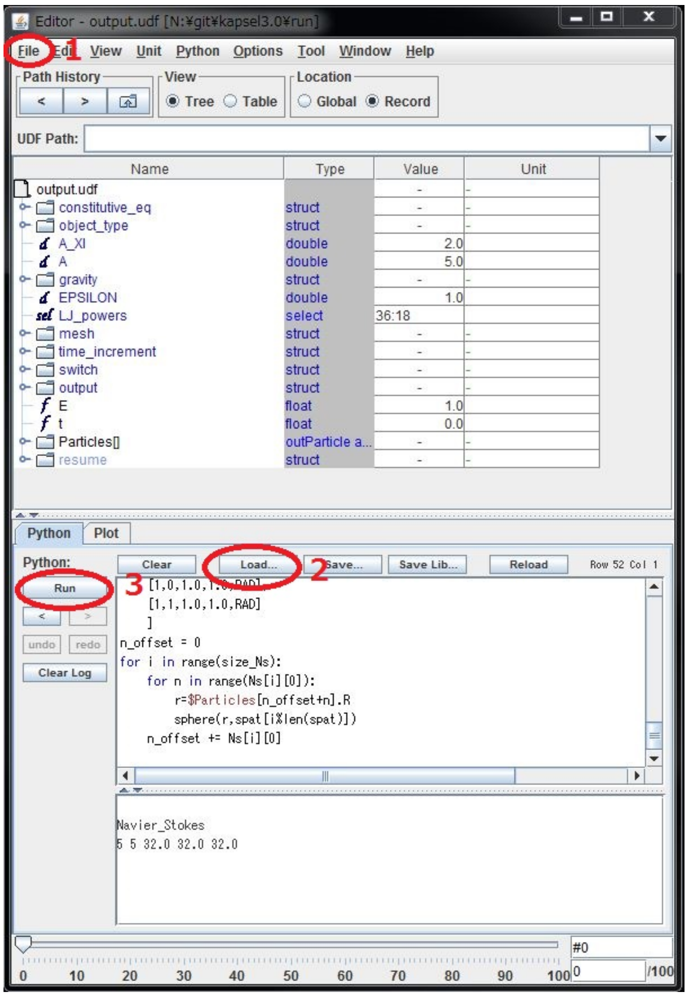
  `File -> Open -> output.udf` [1]

  Positions and velocities for all particles at each time step are stored in the `Particle[]` variables. Use the slide bar at the bottom of the GOURMET viewer window to view variables at other time steps.

- Create animations in GOURMET
  1. Click the `Load` button within the `Python` panel at the bottom of the GOURMET viewer window. [2]
  2. Open the `particleshow.py` file and click the `Run` button. [3]
  3. A new window opens. Click the Play button in this window. [4]

- Plot data in GOURMET (gnuplot)
  1. Click the `Load` button in the `Python` panel at the bottom of the GOURMET viewer window. [2]

[5]Computational Fluid Dynamics

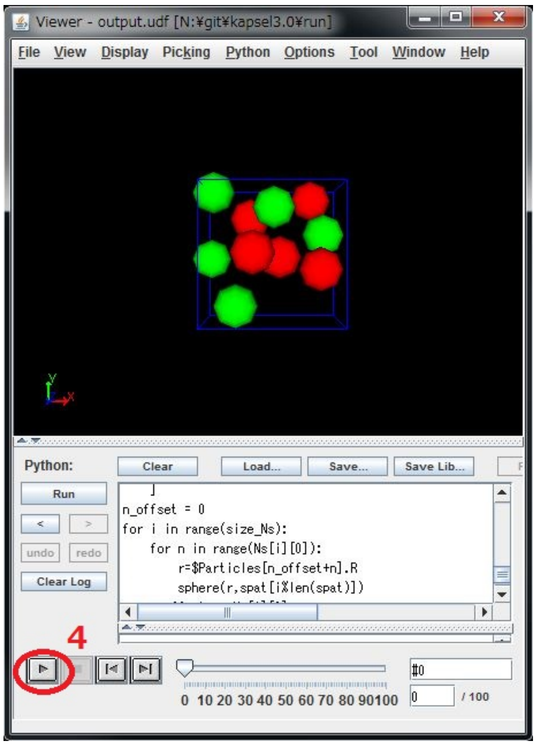

Editor - output.udf [N:¥git¥kapsel3.0¥run]
File Edit View Unit Python Options Tool Window Help
1
Path History
View
Tree Table
Location
Global Record
UDF Path:
Python Plot
Python:
Clear Load... Save... Save Lib... Reload
Row 52 Col 1
2
3
Run
undo redo
Clear Log
#0
/100

View: [✓ Tree] Table — Location: Global [✓ Record]

Name	Type	Value	Unit
output.udf		-	-
constitutive_eq	struct	-	-
object_type	struct	-	-
A_XI	double	2.0	
A	double	5.0	
gravity	struct	-	-
EPSILON	double	1.0	
LJ_powers	select	36:18	
mesh	struct	-	-
time_increment	struct	-	-
switch	struct	-	-
output	struct	-	-
E	float	1.0	
t	float	0.0	
Particles[]	outParticle a...	-	-
resume	struct	-	-

```
    [1,0,1.0,1.0,RAD]
    [1,1,1.0,1.0,RAD]
    ]
n_offset = 0
for i in range(size_Ns):
    for n in range(Ns[i][0]):
        r=$Particles[n_offset+n].R
        sphere(r,spat[i%len(spat)])
    n_offset += Ns[i][0]
```

```
Navier_Stokes
5 5 32.0 32.0 32.0
```


Viewer - output.udf [N:¥git¥kapsel3.0¥run]
File View Display Picking Python Options Tool Window Help
Y
X
Python:
Clear Load... Save... Save Lib...
Run
undo redo
Clear Log
4
0 10 20 30 40 50 60 70 80 90 100
#0
/ 100

```
    ]
n_offset = 0
for i in range(size_Ns):
    for n in range(Ns[i][0]):
        r=$Particles[n_offset+n].R
        sphere(r,spat[i%len(spat)])
```

2. Open the `plot.py` file and click the `Run` button. [3]
3. From the `View` box at the top of the window, click `Table`. [4]
4. Select `Graph Sheet[]` from the left side of the window. [5]
5. Select the `Plot` panel at the bottom of the window and enter the following commands into the command box. [6,7]
6. Click the `Plot` button to display the time evolution of `Vx` for a particle (particle #1) [8]

```
    plot 'plot.dat' using 2:7 with lines
```

2.1.7 Analyzing simulation data

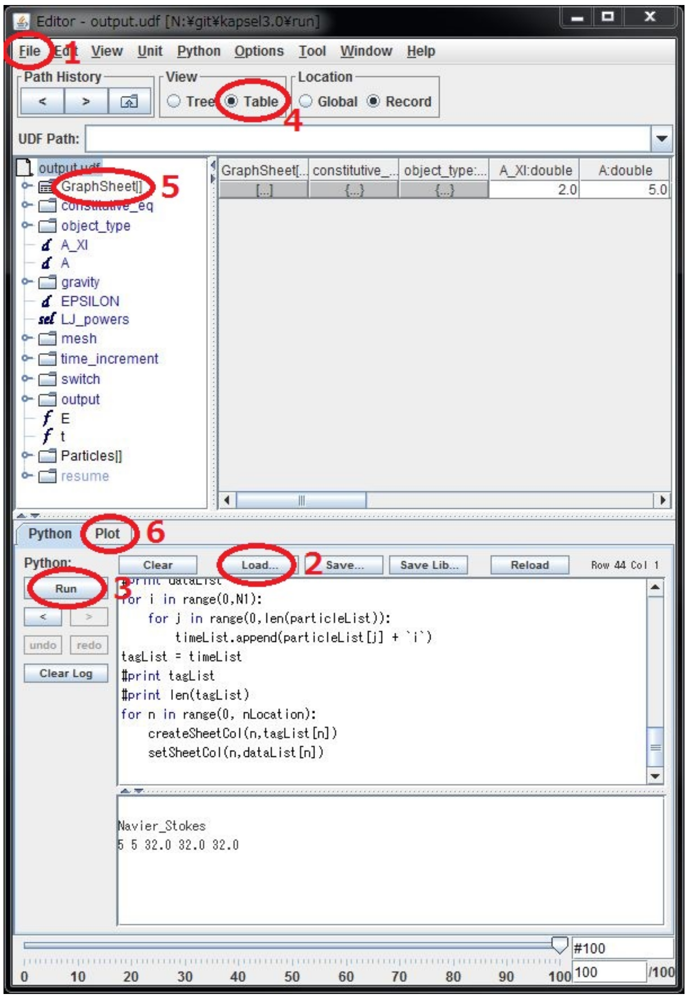
Computational data (particle coordinates and velocities at each time step) produced by running a simulation are saved to the file `output.udf`. The data in this file may be accessed by any of the following methods.

- Using python codes to analyze data

  To use python codes, import the `UDFManager.py` module[6] to access data in the UDF files. The attached file `sk.py` is a python script that computes static structure factors $S(k)$ from time-series data on particle positions stored in `output.udf`.

  Before using this script, you must set up the `numpy` package. Then you may edit the script as appropriate to analyze simulation data for various purposes.

[6]For further details on this module, see the section named "GOURMET PYTHON Script reference manual" in the documented titled "OCTA: A comprehensive simulator for soft materials."



GraphSheet[...	constitutive_...	object_type:...	A_XI:double	A:double
[...]	{...}	{...}	2.0	5.0

```
#print dataList
for i in range(0,N1):
    for j in range(0,len(particleList)):
        timeList.append(particleList[j] + `i`)
tagList = timeList
#print tagList
#print len(tagList)
for n in range(0, nLocation):
    createSheetCol(n,tagList[n])
    setSheetCol(n,dataList[n])
```

```
Navier_Stokes
5 5 32.0 32.0 32.0
```

To use the `sk.py` script, first launch GOURMET:

- On Windows: `Start Menu > All Programs > OCTA > StartGourmetTerm`
- On Linux/Mac: `/usr/local/OCTA8#/GOURMET/gourmetterm`

Then run the following commands:

```
python sk.py
gnuplot
>> plot 'sk.dat' w line
```

- Analyzing data in Jupyter Notebook

  Proceed as follows to repeat the above analysis in Jupyter Notebook[7]

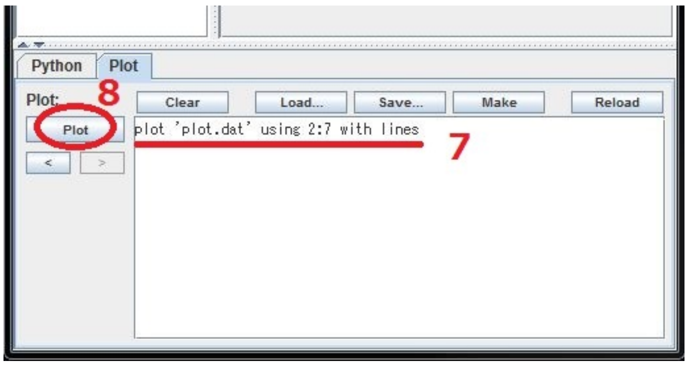
```
$ . $PF_FILES/bin/gourmet_profile.sh
$ . $PF_FILES/bin/platform_env.sh
$ jupyter notebook sk.ipynb
```

- Running analyses in Fortran or C

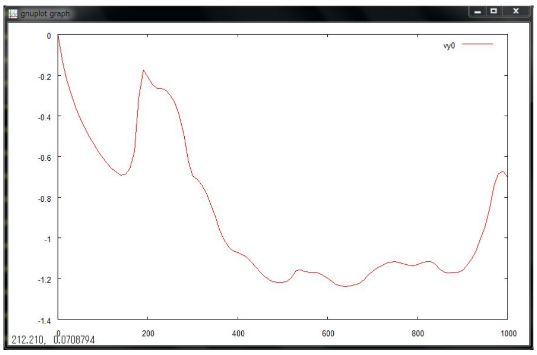
  To use Fortran or C, use the `libplatform` library[8] to access data in UDF files.

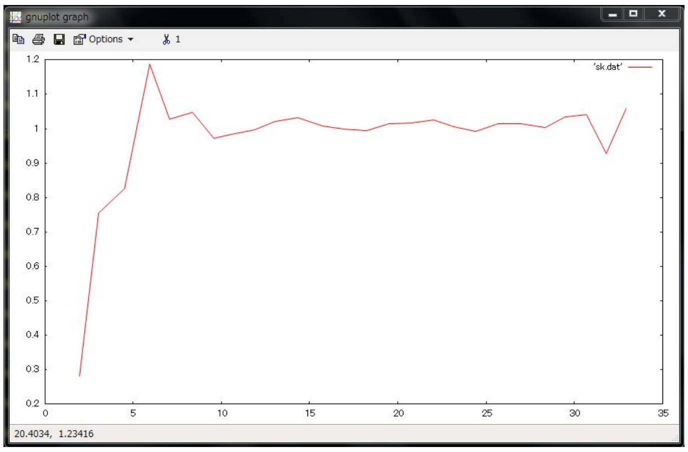
[7]Distributed with Anaconda and other distributions.

[8]For further details on this library, see the section named "libplatform: A platform interface library reference manual" in the documented titled "OCTA: A comprehensive simulator for soft materials."



```
plot 'plot.dat' using 2:7 with lines
```





KAPSEL writes many important data quantities (such as pressures at each time step) to `stderr`. Typically these data items will be displayed on the command line, but you may also proceed as follows to redirect `stderr` to a file:

- For `csh` or `tcsh`:

```
$ ../kapsel  -Iinput.udf -Ooutput.udf -Ddefine.udf -Rrestart.udf >& out
```

- For sh, bash, or Windows command prompt:

```
$ ../kapsel  -Iinput.udf -Ooutput.udf -Ddefine.udf -Rrestart.udf 2> out
```

2.2 Input UDF and definition UDF files

The input UDF file (conventionally named `input.udf`) is used to configure KAPSEL settings appropriate for a given simulation. The structure of `input.udf` is specified in the definition UDF file (conventionally named `define.udf`), and thus the `input.udf` and `define.udf` files must be of the same version. `input.udf` consists of multiple configuration sections; these are described in the next few subsections, and more detailed explanation may be found in Appendix A.

2.2.1 Fluid settings

Each choice of `constitutive_eq` involves its own distinct set of fluid settings. These are described in the "Input UDF file" subsections of Sections 3-8.

2.2.2 Object (particle) settings

The property `object_type.type` specifies the type of particle. The possible settings are `spherical_particle`, `chain` (for flexible chains), and `rigid` (for rigid bodies).

A repulsive interaction between particles, based on the excluded volume of the particles, may be introduced by specifying a truncated Lennard-Jones (LJ) potential. This potential takes the form

$$U_{LJ}(r) = \begin{cases} 4\epsilon\left[\left(\frac{\sigma}{r}\right)^{2n} - \left(\frac{\sigma}{r}\right)^{n}\right] + \epsilon & (r < 2^{1/n}\sigma), \\ 0 & (r > 2^{1/n}\sigma) \end{cases} \tag{2.1}$$

with $\sigma = d = 2a, \epsilon$, and r denoting the particle diameter, the interaction strength, and the distance between the ith and jth particles, respectively. The parameter n may be set to 6, 12, or 18, with larger values yielding a more sharply varying potential. To introduce only a repulsive interaction based on the excluded volume, set `swithch.LJ_truncate` to `ON`. If you wish to introduce an attractive interaction between particles, setting `swithch.LJ_truncate` to `OFF` enables computation of the usual Lennard-Jones potential with a cut-off distance $r_{\mathrm{cut}} = 2.5\sigma$.

When configuring the exponents of the LJ potential, the choice `DLVO` selects the DLVO potential, an interaction between charged particles i and j, having charge valences of z_i and z_j, separated by distances that are large compared to atomic length scales [2].

$$V_{\mathrm{DLVO}}(r) = \begin{cases} V_{\mathrm{Coul}}(r) + V_{\mathrm{vdW}}(r) & (r \geq r^*) \\ V_{\mathrm{Coul}}(r) + V_{\mathrm{excv}}(r) & (r < r^*) \end{cases} \tag{2.2}$$

Here,

$$V_{\mathrm{Coul}}(r) = \frac{e^2}{4\pi\varepsilon_r\varepsilon_0} z_i z_j \left(\frac{\exp(\kappa a)}{\kappa a + 1}\right)^2 \frac{1}{r}\exp(-\kappa r) \tag{2.3}$$

represents the screened Coulombic potential between charged particles,

$$V_{\mathrm{vdW}}(r) = -\frac{A}{12}\left[\frac{d^2}{r^2 - d^2} + \frac{d^2}{r^2} + 2\ln\left(\frac{r^2 - d^2}{r^2}\right)\right] \tag{2.4}$$

is the van der Waals potential acting between spherical particles, at distances much larger than atomic length scales, and

$$V_{\mathrm{excv}}(r) = \frac{K}{2}(r - d)^2 - V_m \tag{2.5}$$

mimics the excluded volume potential to avoid the occurrence of overlapping particles. V_{vdw} and V_{excv} are to be smoothly switched at an interparticle distance $r = r^*(> d \equiv 2a)$, thus the following two parameters are automatically set given the value of A

$$K = \frac{A}{6}\frac{r^* d^2}{r^* - d}\left(\frac{1}{(r^{*2} - d^2)^2} + \frac{1}{r^{*4}} + \frac{r^{*2} - d^2}{r^{*6}}\right) \tag{2.6}$$

$$V_m = \frac{K}{2}(r^* - d)^2 + \frac{A}{12}\left[\frac{d^2}{r^{*2} - d^2} + \frac{d^2}{r^{*2}} + 2\ln\left(\frac{r^{*2} - d^2}{r^{*2}}\right)\right] \tag{2.7}$$

Figure 2.1 shows the schematic plots for these three potentials. The values for z_i, r^*, κa, $C \equiv \frac{e^2}{4\pi\varepsilon_r\varepsilon_0}$, A are set through the following variables defined in `input.udf`.

- `object_type.spherical_particle.Particle_spec[i].Surface_charge` = z_i
- `DLVO.n` = r^*/d
- `DLVO.kappa_a` = κa
- `DLVO.vdw_coeff` = A
- `DLVO.coulomb_coeff` = C

The choice `electro_osmotic_flow` for the exponents of the LJ potential selects an interaction potential mediated by electro-osmotic flow, relevant for charged particles in the vicinity of a slab. This is only used for the Quincke rollers discussed in Section 8.

Flexible-chain particles and rigid-body particles are implemented as secondary particles consisting of aggregates of multiple spherical primary particles. Thus, initial configurations for these systems must be specified in `input.udf`

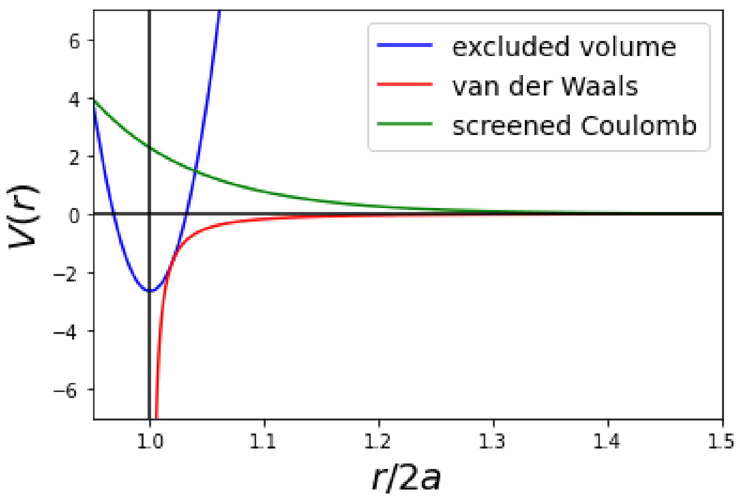


Figure 2.1: Schematic plots for the three contributions to the DLVO potential. The green, red, and blue lines represent the screened Coulombic $V_{\mathrm{Coul}}(r)$, van der Waals (Hamarker) $V_{\mathrm{vdW}}(r)$, and excluded volume $V_{\mathrm{excv}}(r)$ potentials, respectively. We set $d = 1$, $r^* = 1.02d$, $\kappa a = 5$, $\frac{e^2}{4\pi\varepsilon_r\varepsilon_0} z_i z_j \left(\frac{\exp(\kappa a)}{\kappa a+1}\right)^2 = 50000$, $A = 1$ in this figure.

by setting `switch.INIT_distribution=user_specify`. In a flexible chain, pairs of neighboring beads interact through a finitely extensible non-linear elastic (FENE) potential of the form

$$U_F(r) = -\frac{1}{2} k_c R_0^2 \ln\{1 - (r/R_0)^2\}, \tag{2.8}$$

which is responsible for binding. Here r is the distance between neighboring beads and the parameters have values $k_c = 30\epsilon/\sigma^2$, $R_0 = 1.5\sigma$, If a rigid body is selected, the initial particle structure is preserved by an implicit constraint force.

➤`object_type.spherical_particle.Particle_spec[]`: Properties of spherical particles

`object_type.spherical_particle.Particle_spec[].Particle_number`
Number of particles.

`object_type.spherical_particle.Particle_spec[].MASS_Ratio`
Ratio of particle density to fluid density.

`object_type.spherical_particle.Particle_spec[].Surface_charge`
Surface charge (not used for simulations of two-component phase-separated fluids).

`object_type.spherical_particle.Particle_spec[].janus_axis`
Orientation of the Janus axis in a body-fixed coordinate system.

`object_type.spherical_particle.Particle_spec[].janus_propulsion`
Propulsive motion of Janus particles. Possible values: `OFF` (disabled), `TUMBLER` (particle propelled along propulsion axis by fixed external force), `SQUIRMER` (squirmer particle propelled along propulsion axis by slip boundary conditions), `OBSTACLE` (particle forming a fixed obstacle).

`object_type.spherical_particle.Particle_spec[].janus_force.x`
x component of propulsive force.

`object_type.spherical_particle.Particle_spec[].janus_force.y`
y component of propulsive force.

`object_type.spherical_particle.Particle_spec[].janus_force.z`
  z component of propulsive force.

`object_type.spherical_particle.Particle_spec[].janus_torque.x`
  x component of propulsive torque.

`object_type.spherical_particle.Particle_spec[].janus_torque.y`
  y component of propulsive torque.

`object_type.spherical_particle.Particle_spec[].janus_torque.z`
  z component of propulsive torque.

`object_type.spherical_particle.Particle_spec[].janus_slip_vel`
  The parameter B_1 determining the surface-slip velocity for self-propelled particles.

`object_type.spherical_particle.Particle_spec[].janus_slip_mode`
  The parameter B_2/B_1 determining the type of slip motion for squirmer particles: pusher, neutral, or puller.[9]

`object_type.spherical_particle.Particle_spec[].janus_rotlet_C1`
  The parameter C_1 describing unipolar rotation about the propulsion axis.

`object_type.spherical_particle.Particle_spec[].janus_rotlet_dipole_C2`
  The parameter C_2 describing bipolar rotation about the propulsion axis.

➤`object_type.chain.Chain_spec[]`: Properties of flexible chains

`object_type.chain.Chain_spec[].Beads_number`
  Number of beads associated with a single chain.

`object_type.chain.Chain_spec[].Chain_number`
  Number of chains.

`object_type.chain.Chain_spec[].MASS_RATIO`
  Ratio of bead density to fluid density.

`object_type.chain.Chain_spec[].Surface_Charge`
  Surface charge (not used for simulations of two-component phase-separated fluids).

`object_type.chain.Chain_spec[].janus_axis`
  Orientation of the Janus axis in a bead-fixed coordinate system.

➤`object_type.rigid.Rigid_spec[]`: Properties of rigid bodies

`object_type.rigid.Rigid_spec[].Beads_number`
  Number of beads comprising the rigid body.

`object_type.rigid.Rigid_spec[].Rigid_number`
  Number of rigid bodies.

[9]More specifically, when this parameter is {negative, zero, positive} we have a {pusher, neutral, puller} [3, 4].

`object_type.rigid.Rigid_spec[].MASS_RATIO`
: Ratio of bead density to fluid density. The mass of overlapping beads is not double-counted.

`object_type.rigid.Rigid_spec[].Surface_charge`
: Total surface charge: z (not used for simulations of two-component phase-separated fluids).

`object_type.rigid.Rigid_spec[].Rigid_motion`
: Selects the type of motion executed by rigid bodies: either `free` (free motion) or `fix` (motion with fixed translational and rotational velocities). For `fix`, the (lab-frame) velocities are as specified by the `Rigid_velocity` and `Rigid_omega` variables defined below. Note that, by default, this will fix all 6 degrees of freedom. To constrain individual translational/rotational degrees of freedom, use the `switch.free_rigid` options described below.

`object_type.rigid.Rigid_spec[].Rigid_velocity.x`
: x component of rigid body velocity (lab frame).

`object_type.rigid.Rigid_spec[].Rigid_velocity.y`
: y component of rigid body velocity (lab frame).

`object_type.rigid.Rigid_spec[].Rigid_velocity.z`
: z component of rigid body velocity (lab frame).

`object_type.rigid.Rigid_spec[].Rigid_omega.x`
: x component of rigid body angular velocity (lab frame).

`object_type.rigid.Rigid_spec[].Rigid_omega.y`
: y component of rigid body angular velocity (lab frame).

`object_type.rigid.Rigid_spec[].Rigid_omega.z`
: z component of rigid body angular velocity (lab frame).

2.2.3 Common simulation settings

➤Standalone UDF properties for particle radius and interface thickness

`A_XI`
: Interface thickness ξ.[10]

`A`
: Particle radius.

➤`gravity`: Properties of gravitational forces

`gravity.G`
: Gravitational acceleration.

`gravity.G_direction`
: Direction in which gravitational forces are applied.

➤Standalone UDF properties describing the character of interparticle forces

[10] For simulations of electrolytes or two-component phase-separated fluids, we must compute the gradient of ϕ for the GL free energy F, for which purpose it is best to choose $\xi \geq 2$.

`EPSILON`
Defines the units of energy in the Lennard-Jones potential.

`LJ_powers`
Selects one of the various supported forms for the interparticle potential. Allowed values:

`12:6`	Lennard-Jones potential with the given exponents
`24:12`	
`36:18`	
`macro_vdw`	macroscopic interparticle potential
`electro_osmotic_flow`	used only for Quincke rollers

➤`mesh`: Computational mesh sizes

`mesh.NPX`
Base-2 logarithm of mesh size in x-direction ($L_x = 2^{\texttt{NPX}}$)

`mesh.NPY`
Base-2 logarithm of mesh size in y-direction ($L_y = 2^{\texttt{NPY}}$)

`mesh.NPZ`
Base-2 logarithm of mesh size in z-direction ($L_z = 2^{\texttt{NPZ}}$)

➤`time_increment`: Simulation timestep settings

`time_increment`
Specifies how KAPSEL determines the simulation timestep. Allowed values:

`auto`	KAPSEL will automatically set the timestep to its maximum allowable value, given by $T_{step} = \rho/\eta k_{max}^2$, where k_{max} is the maximum wavenumber determined by the lattice spacing Δ.
`manual`	A fixed timestep will be specified by the user.

`time_increment.auto.factor`
Specifies a multiplicative scale factor for auto-determined timesteps. The simulation timestep is given by $\Delta t = \texttt{factor} \times T_{step}$.

`time_increment.manual.delta_t`
User-specified timestep Δt.

2.2.4 Configuring selectable features

➤`switch`: Miscellaneous simulation parameters

`switch.ROTATION`
Set to `ON` to solve the equations of motion for the rotational motion of particles.

`switch.LJ_truncate`
Selects the form of the Lennard-Jones potential acting between particles. Set to `OFF` for the usual form of the potential, including the attractive term. Set to `ON` to exclude the attractive term, retaining only the repulsive term. Set to `NONE` to exclude both terms, i.e. to disable the interparticle potential entirely.

`switch.INIT_distribution`
: Specifies the initial particle configuration. Possible values: `uniform_random` (random), `random_walk` (particles randomly moved from the sites of a square lattice), `FCC` (FCC lattice), `BCC` (BCC lattice), `user_specify` (user-specified coordinates and velocities).[11]

`switch.INIT_distribution.random_walk.iteration`
: Number of trial iterations for the `random_walk` setting of `INIT_distribution`.

`switch.INIT_orientation`
: Initial particle orientation. Possible values: `user_specify` (orientations specified by the user), `random` (random), `space_align` (body-axis of particles aligned with space/lab axis).[12]

`switch.SLIP_tol`
: Convergence criterion for iterative calculation of fluid-flow fields when introducing slip velocities in the tangential direction at the interfaces of self-propelled particles.

`switch.SLIP_iter`
: Maximum iteration count for iterative calculation of fluid-flow fields when introducing slip velocities in the tangential direction at the interfaces of self-propelled particles.

`switch.FIX_CELL.x`
: Set to `ON` to zero out the DC component of the total velocity in the *x* direction.

`switch.FIX_CELL.y`
: Set to `ON` to zero out the DC component of the total velocity in the *y* direction.

`switch.FIX_CELL.z`
: Set to `ON` to zero out the DC component of the total velocity in the *z* direction.

`switch.pin.type`
: Set to `YES` to fix particle positions.

`switch.pin.YES.pin[]`
: Specify the indices of any particles for which translational motion is to be prohibited.

`switch.pin.YES.pin_rot[]`
: Specify the indices of any particles for which rotational motion is to be prohibited.

`switch.free_rigid.type`
: Configure individual translational/rotation degrees of freedom (DOFs) for rigid bodies. These option are meant to be used together with the `object_type.rigid.Rigid_spec[].Rigid_motion` options, which fix all six DOF to have specified (lab-frame) translational/rotational velocities.

To allow the six DOF to be configured independently for each rigid species, set this option to `YES`, and use the `vel.x|y|z` and `omega.x|y|z` options decribed below to selectively free (`YES`) or fix (`NO`) the corresponding DOF. The fixed value of each velocity component is the one specified by the `Rigid_velocity` and `Rigid_omega` variables defined via the `object_type.rigid.Rigid_spec[]` options.

For example, to specify a rigid particle with fixed `vel.y` and `omega.z`, you should specify the following `free_rigid` options below: `vel.x`=`YES`, `vel.y`=`NO`, `vel.z`=`YES` and `omega.x`=`YES`, `omega.y`=`YES`, `omega.z`=`NO`; i.e., the convention here is to specify which degrees of freedom are to be set free.

[11]If `user_specify` is chosen, particle positions and velocities are initialized to the values specified for `user_specify.Particles[].R` and `user_specify.Particles[].v`. If the number of items in the input list is less than the value specified for `Particle_number`, increase the length of `user_specify.Particles[]`, either by directly editing the UDF file or by using `Edit->Add an array Element` in GOURMET.

[12]If `user_specify` is selected, particle orientations will be initialized to the values specified for `user_specify.Particles[].q`.

`switch.free_rigid.YES.DOF[].spec_id`
Specify the indices of rigid bodies to configure.

`switch.free_rigid.YES.DOF[].vel.x`
Configure degrees of freedom for *x*-directed translational motion of rigid bodies. Set to `YES` to allow motion or `NO` to fix the position.

`switch.free_rigid.YES.DOF[].vel.y`
Configure degrees of freedom for *y*-directed translational motion of rigid bodies. Set to `YES` to allow motion or `NO` to fix the position.

`switch.free_rigid.YES.DOF[].vel.z`
Configure degrees of freedom for *z*-directed translational motion of rigid bodies. Set to `YES` to allow motion or `NO` to fix the position.

`switch.free_rigid.YES.DOF[].omega.x`
Configure degrees of freedom for *x*-directed rotational motion of rigid bodies. Set to `YES` to allow motion or `NO` to fix the orientation.

`switch.free_rigid.YES.DOF[].omega.y`
Configure degrees of freedom for *y*-directed rotational motion of rigid bodies. Set to `YES` to allow motion or `NO` to fix the orientation.

`switch.free_rigid.YES.DOF[].omega.z`
Configure degrees of freedom for *z*-directed rotational motion of rigid bodies. Set to `YES` to allow motion or `NO` to fix the orientation.

`ns_solver.OBL_INT`
Set to `linear` or `spline` to select the approximation function used for coordinate transformations in shear-flow simulations.

`switch.wall.type`
Set to `FLAT` to specify planar walls.

`switch.wall.FLAT.axis`
Set to `X`, `Y`, or `Z` to specify the direction normal to planar walls.

`switch.wall.FLAT.DH`
Thickness of planar walls, measured in units of lattice spacing.

`switch.wall.FLAT.LJ_Params`
Specifies how the potential acting at planar walls is chosen. If set to `AUTO`, the potential will be the same as the interparticle potential. If set to `MANUAL`, the potential is user-specified via the `switch.wall.FLAT.MANUAL` property.

`switch.wall.FLAT.MANUAL.truncate`
Specifies the presence or absence of attractive forces at planar walls. If set to `ON`, the potential is purely repulsive. If set to `OFF`, the potential is a sum of repulsive and attractive contributions.

`switch.wall.FLAT.MANUAL.powers`
Specifies the exponents in the Lennard-Jones potential at planar walls. Possible values: `12:6`, `24:12`, `36:18`.

`switch.wall.FLAT.MANUAL.EPSILON`
: Determines the strength of the Lennard-Jones potential at planar walls.

`switch.quincke.type`
: Set to `ON` when simulating Quincke rollers.

`switch.quincke.ON.e_dir`
: Set to `X`, `Y`, or `Z` to specify the direction of the external electric field.

`switch.quincke.ON.w_dir`
: Set to `X`, `Y`, or `Z` to specify the direction of the angular-velocity vector associated with rotation induced by the Quincke effect.

`switch.quincke.ON.torque_amp`
: Magnitude of rotational torque.

`switch.multipole.type`
: Set to `ON` to run simulations using the Ewald method.

`switch.multipole.ON.Dipole`
: Set to `ON` to use dipolar Quincke particles.

`switch.multipole.ON.Dipole.ON.magnitude`
: Magnitude of dipole moment.

`switch.multipole.ON.Dipole.ON.type`
: Set to `FIXED` or `QUINCKE` to specify the type of dipole.

`switch.multipole.ON.Dipole.ON.FIXED.dir`
: Set to `X`, `Y`, or `Z` to specify the direction of a fixed dipole.

`switch.multipole.ON.EwaldParams.alpha`
: Screening parameter for Ewald method.

`switch.multipole.ON.EwaldParams.delta`
: Convergence criterion for determination of $k_{\max}$ via the Ewald method.

`switch.multipole.ON.EwaldParams.converge`
: Convergence parameter for Ewald method.

`switch.multipole.ON.EwaldParams.epsilon`
: Dielectric permittivity at boundaries for Ewald method.

2.2.5 Data output settings

➤`output`: Data output settings

`output.GTS`
: Data output interval measured in number of steps.

`output.Num_snap`
: Number of data outputs. The total number of timesteps is given by `GTS` $\times$ `Num_snap`.

`output.AVS`
: Set to `ON` to output data in AVS format.
Reference: `https://ja.overleaf.com/project/5fe7389e29fad9c51e06569e`

`output.AVS.ON.Out_dir`
: Specifies the directory in which AVS-format data output files are written.[13]

`output.AVS.ON.Out_name`
: Specifies the filename prefix for AVS-format data output files.

`output.AVS.ON.FileType`
: Specifies the format of AVS data output files. Possible values: `Binary`, `ASCII`, `EXTENDED`

`output.ON.EXTENDED.Driver.Format`
: Specifies the extended output data format. At present, the only supported setting is `HDF5`.

`output.ON.EXTENDED.Print_field.Crop`
: Set to `YES` to reduce the number of output data fields.

`output.ON.EXTENDED.Print_field.YES.Slab_x.start`
: Index of the first lattice point in the x direction to output.

`output.ON.EXTENDED.Print_field.YES.Slab_x.count`
: Number of lattice points in the x direction to output.

`output.ON.EXTENDED.Print_field.YES.Slab_x.stride`
: Interval between output lattice points in the x direction.

`output.ON.EXTENDED.Print_field.YES.Slab_y.start`
: Index of the first lattice point in the y direction to output.

`output.ON.EXTENDED.Print_field.YES.Slab_y.count`
: Number of lattice points in the y direction to output.

`output.ON.EXTENDED.Print_field.YES.Slab_y.stride`
: Interval between output lattice points in the y direction.

`output.ON.EXTENDED.Print_field.YES.Slab_z.start`
: Index of the first lattice point in the z direction to output.

`output.ON.EXTENDED.Print_field.YES.Slab_z.count`
: Number of lattice points in the z direction to output.

`output.ON.EXTENDED.Print_field.YES.Slab_z.stride`
: Interval between output lattice points in the z direction.

[13] For example, if this property is set to `data`, then the directories `./data` and `./data/avs` must be created in advance. The AVS field file will be written to a file named `data.fld` in the `./data` directory. Data files will be written in the directory `./data/avs` and will have filenames of the form `data_*.dat`, where * will be replaced by the step count.

`output.ON.EXTENDED.Print_field.Vel`
Set to YES to output velocity-field values.

`output.ON.EXTENDED.Print_field.Phi`
Set to YES to output the SP function ϕ.

`output.ON.EXTENDED.Print_field.Charge`
Set to YES to output the charge density distribution (not used for simulations of two-component phase-separated fluids).

`output.ON.EXTENDED.Print_field.Pressure`
Set to YES to output the pressure field (currently not implemented).

`output.ON.EXTENDED.Print_field.Tau`
Set to YES to output the stress tensor.

`output.UDF`
Set to ON for UDF output.[14]

2.2.6 Restart settings

➤`resume`: Handling interrupted calculations

`resume.calculation`
Specifies how KAPSEL proceeds in the event of an interrupted calculation. Allowed values:

NEW	Start new calculation.
CONTINUE	Read data saved upon termination of previous calculation and resume that calculation.
CONTINUE_FDM	
CONTINUE_FDM_PHASE_SEPARATION	

Which of the three CONTINUE options to use depends on the type of simulation you are running:

- Use CONTINUE for simulations using spectral methods.[15]
- Use CONTINUE_FDM for simulations of single-component fluids via finite-difference methods.[16]
- Use CONTINUE_FDM_PHASE_SEPARATION for simulations of two-component fluids via finite-difference methods.[17]

2.2.7 GOURMET display settings

➤`Unit_Parameter`: Display settings for GOURMET. Does not affect simulation results.

`Unit_Parameter.Name`
Name of UDF file.

`Unit_Parameter.Comment`
Comment for UDF file.

[14]Coordinates and velocities for each particle will be written to the Particles[] section of the output UDF file. The number of data records saved will be the value specified for Num_snap.
[15]This includes the following choices for constitutive_eq: Navier_Stokes, Shear_Navier_Stokes, Shear_Navier_Stokes_Lees_Edwards, or Electrolyte.
[16]This includes the following choices for constitutive_eq: Navier_Stokes_FDM or Shear_Navier_Stokes_Lees_Edwards_FDM.
[17]This includes the following choices for constitutive_eq: Navier_Stokes_Cahn_Hilliard_FDM or Shear_NS_LE_CH_FDM.

`Unit_Parameter.Temperature`
Temperature unit, used as a reference when simulation parameters are reported in actual units in GOURMET.

`Unit_Parameter.Length`
Length unit, used as a reference when simulation parameters are reported in actual units in GOURMET.

`Unit_Parameter.rho`
Density unit, used as a reference when simulation parameters are reported in actual units in GOURMET.

2.3 Output and restart UDF files

The output UDF file (`output.udf`) accumulates data reported by KAPSEL simulations at fixed simulation-time intervals. The interval (number of timesteps) between data reports, and the total number of timesteps in a simulation, may be configured via the following properties in input UDF files:

`output.GTS`
Data output interval measured in number of steps.

`output.Num_snap`
Number of data outputs. The total number of timesteps is given by `GTS` $\times$ `Num_snap`.

The restart UDF file (`restart.udf`) stores all data written at the end of a simulation.[18] To use `restart.udf` to resume a completed calculation, follow the steps below.

1. Rename `restart.udf` to `input2.udf`.
2. Open `input2.udf` in Gourmet and set `resume.Calculation` to one of the following values as appropriate for the simulation in question: `CONTINUE`, `CONTINUE FDM`, `CONTINUE FDM PHASE SEPARATION`
3. Increase the value specified for `output.Num_step` by the number of additional timesteps you wish to compute. Thus, the new value of `output.Num_step` should be (number of timesteps previously computed) + (number of additional timesteps to compute).
4. The values of various parameters and the number of particles can also be changed on restart. If particles are to be added/changed, the `objects_type` block should be set appropriately and the additions/changes should be applied to the `Particles` in the `resume` block. `INIT_distribution/INIT_orientation` in the `switch` block is ignored on restart.
5. Execute the following command line:

   ```
   % kapsel -Iinput2.udf -Ooutput.udf -Ddefine.udf -Rrestart.udf
   ```

   This will read data from `input2.udf` and resume the calculation starting at the endpoint of the previous calculation.
6. For both `output.udf` and AVS data files, output data from the resumed simulation may be appended to the existing set of output data from the previous simulation.
7. When the resumed calculation is complete, all data will again be written to `restart.udf`, and the above procedure may be repeated to extend the calculation yet again.

[18]In restart UDF files, results are saved under the item `resume.CONTINUE.Saved_Data`. Consequently, continuation of a previous calculation may be initiated by using the `restart.udf` file written by the previous calculation as the input UDF file for a new KAPSEL run.

Chapter 3

Simulating particles dispersed in Newtonian fluids

Systems consisting of colloidal particles dispersed in liquids (solvents), commonly known as colloidal "suspensions", are ubiquitous in many sectors of modern daily life: foodstuffs, paints, pigments and dyes, cosmetics, slurries, and more. Experience with microparticle suspensions shows that any hope of understanding the properties and behavior of these systems depends crucially on having a firm grasp of the nature of the motion of both the colloidal particles and the solvent.

3.1 Theoretical background and basic equations

KAPSELconducts simulations for several types of dispersed particle systems using a direct numerical approach known as the *smoothed profile* (SP) method [5, 6], which takes into account both the solid particles and the fluid, and solves their equations of motion simultaneously. The method is not restricted to simulations of Newtonian fluids, but may be applied to systems in which the solvents are complex fluids with arbitrary Reynolds numbers.

3.1.1 Basic equations for disperse particle systems

The SP method allows us to solve for the motion of solid particles dispersed in a continuous medium [6]. In principle, the SP method is applicable to solid particles of arbitrary shapes.

Also, the method is not restricted to simulating simple liquids, but can also handle complex multi-phase fluids and other complicated systems, for which it makes liberal use of appropriate constitutive equations to enable direct computational modeling.

To illustrate how the motion of particles and fluids ties in to the SP method, in this chapter we consider the motion of a disc-shaped rigid (solid) body moving freely in a Newtonian fluid [7]. As shown in Figure 3.1, the solid body is described as an assembly of spherical beads; in what follows we will simply refer to this body as the particle.

The equations governing incompressible Newtonian fluids are the continuity equation and the Navier-Stokes equations:

$$\nabla \cdot \boldsymbol{u}_f = 0, \tag{3.1}$$

$$\left(\partial_t + \boldsymbol{u}_f \cdot \nabla\right) \boldsymbol{u}_f = \rho_f^{-1} \nabla \cdot \boldsymbol{\sigma}, \tag{3.2}$$

$$\boldsymbol{\sigma} = -p\boldsymbol{I} + \eta \left[\nabla \boldsymbol{u}_f + \left(\nabla \boldsymbol{u}_f\right)^t\right] \tag{3.3}$$

Here $\boldsymbol{u}_f, \rho_f$, and η are the fluid velocity, density, and viscosity, while $\boldsymbol{\sigma}$ is the stress field (tensor); p is the pressure, $\boldsymbol{I}$ is the identity tensor, and $(\cdots)^t$ denotes matrix transposition. Unless stated otherwise, henceforth we assume that the particle and fluid densities are equal, $\rho_f = \rho_p = \rho$. At solid-fluid interfaces we impose no-slip boundary conditions, i.e., at solid surfaces we enforce $u_f = u_p$ where u_p is the particle velocity at the solid surface. Denoting by M_I and I_I the mass and inertial moment of the Ith particle ($I = 1, \cdots, N$), the motion of a solid particle (here comprised of N rigid spheres) is described by the Newton-Euler equations [8]:

$$\dot{\boldsymbol{R}}_I = \boldsymbol{V}_I, \qquad \dot{\boldsymbol{Q}}_I = \mathrm{skew}(\boldsymbol{\Omega}_I) \cdot \boldsymbol{Q}_I, \tag{3.4}$$

$$M_I \dot{\boldsymbol{V}}_I = \boldsymbol{F}_I^H + \boldsymbol{F}_I^C + \boldsymbol{F}_I^E + \boldsymbol{G}_I^V, \tag{3.5}$$

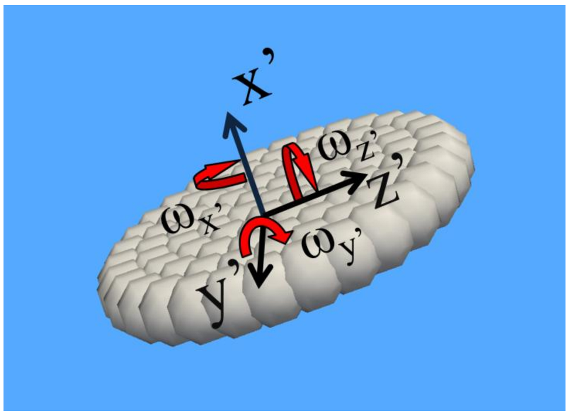


Figure 3.1: Schematic illustration of a disc-shaped solid body moving in a three-dimensional space. The motion of the body is characterized by its three translational and three rotational degrees of freedom.

$$\dot{\boldsymbol{J}}_I = \boldsymbol{N}_I^H + \boldsymbol{N}_I^C + \boldsymbol{N}_I^E + \boldsymbol{G}_I^\Omega, \tag{3.6}$$

Here $\boldsymbol{R}_I$ and $\boldsymbol{V}_I$ are the position and velocity of the center of mass, $\boldsymbol{Q}_I$ and $\boldsymbol{\Omega}_I$ are the corresponding orientation matrix and angular velocity, and $\boldsymbol{J}_I = \boldsymbol{I}_I \cdot \boldsymbol{\Omega}_I$ is the angular momentum. The orientation matrix relates the basis vectors $\{\boldsymbol{e}_i\}$ $(i = 1, 2, 3)$ of a fixed orthogonal coordinate system (the laboratory system) to the basis vectors $\{\tilde{\boldsymbol{e}}_i\}$ of a time-dependent coordinate system co-rotating with the particle according to $\tilde{\boldsymbol{e}}_i = \sum_{j=1}^3 \boldsymbol{e}_j Q_{ji}$. The temporal evolution of the rotation matrix is governed by the (skew-symmetric) angular-velocity matrix, which is determined by the angular-velocity vectors $[\text{skew}(\boldsymbol{\Omega})]_{ij} = -\sum_{k=1}^3 \epsilon_{ijk}\Omega^k$ (with ϵ_{ijk} the Levi–Civita symbol) according to

$$\text{skew}(\boldsymbol{\Omega}) = \begin{pmatrix} 0 & -\Omega^z & \Omega^y \\ \Omega^z & 0 & -\Omega^x \\ -\Omega^y & \Omega^x & 0 \end{pmatrix}. \tag{3.7}$$

Particles experience both forces $\boldsymbol{F}$ and torques $\boldsymbol{N}$ from their surroundings. There are three contributions to these forces and torques: a contribution $\boldsymbol{F}^C$ arising from direct interparticle interactions, a contribution $\boldsymbol{F}^E$ due to gravity or other external fields, and a contribution from the solvent. The contribution from the solvent is the sum of two terms: one term $\boldsymbol{F}^H$ representing hydrodynamic interactions between particles, and a second term $\boldsymbol{F}^V$ representing thermal fluctuations. Note that the most accurate choice of notation for the various force and torque contributions differs depending on the system in question; we will discuss each case individually. We also introduce random forces and torques $\boldsymbol{G}^V$ and $\boldsymbol{G}^\Omega$, acting on particles due to thermal fluctuations, which we model as white noise. That is, we have $\langle \boldsymbol{G}_I^V \rangle = \langle \boldsymbol{G}_I^\Omega \rangle = 0$ and

$$\langle \boldsymbol{G}_I^V(t) \cdot \boldsymbol{G}_J^V(0) \rangle = 3k_BT\alpha^V\delta(t)\delta_{IJ}, \tag{3.8}$$
$$\langle \boldsymbol{G}_I^\Omega(t) \cdot \boldsymbol{G}_J^\Omega(0) \rangle = 3k_BT\alpha^\Omega\delta(t)\delta_{IJ} \tag{3.9}$$

where $\langle\rangle$ denotes statistical averaging and α^V and α^Ω are adjustable parameters used to ensure that the particle temperature T accurately tracks the specified value. The particle temperature T is determined in advance by simulating a single particle fluctuating in a solvent and adjusting T until the translational and rotational diffusion coefficients D^V and D^Ω agree with the theoretical values.

We now explain how the SP method is used to solve for the motion of a system described by the above equations. At every point $\boldsymbol{x}$ in the spatial region of the simulation—including both solvent regions and solid regions in the particle domains—the SP approach defines a velocity field according to

$$\boldsymbol{u}(\boldsymbol{x}, t) = (1 - \phi)\,\boldsymbol{u}_f(\boldsymbol{x}, t) + \phi\boldsymbol{u}_p(\boldsymbol{x}, t), \tag{3.10}$$

where

$$\phi\boldsymbol{u}_p(\boldsymbol{x}, t) = \sum_{I=1}^{N} \phi_I\left[\boldsymbol{V}_I + \boldsymbol{\Omega}_I \times \boldsymbol{r}_I\right], \tag{3.11}$$

expresses the rigid-body motion of solid particles in terms of the velocity field. Here $\boldsymbol{r}_I = \boldsymbol{x} - \boldsymbol{R}_I$ is the distance from the Ith particle's center of mass and $\phi_I = \phi_I(\boldsymbol{x}; \boldsymbol{R}_I, \boldsymbol{Q}_I) \in [0, 1]$ is a phase-field function that distinguishes between solid regions inside particles ($\phi \simeq 1$) and fluid regions exterior to particles ($\phi \simeq 0$).

The SP method replaces the usual sharp solid-liquid interface with a blurred finite-thickness interface region, with the interface thickness ξ incorporated into the phase-field function ϕ; in this document we refer to this as a smoothed-profile (SP) function. Replacing discontinuous solid-fluid interfaces with finite-width transition regions yields two key advantages for numerical calculations. First, the positions of solid-fluid interfaces may be tracked using only a limited number of lattice points. Second, interactions between the solid and fluid—such as hydrophilic or hydrophobic behavior—may be properly specified as appropriate for the objectives of a simulation.

3.1.2 Outline of the SP method

Applying the continuity equation and the Navier-Stokes equations to the full velocity field $\boldsymbol{u}(\boldsymbol{r})$—which includes both particle and fluid contributions—yields

$$\nabla \cdot \boldsymbol{u} = 0, \tag{3.12}$$

$$(\partial_t + \boldsymbol{u} \cdot \nabla)\, \boldsymbol{u} = \rho^{-1} \nabla \cdot \boldsymbol{\sigma} + \phi \boldsymbol{f}_p, \tag{3.13}$$

where $\phi \boldsymbol{f}_p$ acts as a penalty force to satisfy the rigid-body constraints in the particle regions. Also, whereas equation (3.3) expresses the local stress tensor in terms of the fluid contribution to the velocity ($\boldsymbol{u}_f$), in the SP method stress is determined by the total velocity ($\boldsymbol{u}$), defined even within the solid particle domains.

The quantities $\phi \boldsymbol{f}_p$, $\boldsymbol{F}_I^H$, and $\boldsymbol{N}_I^H$ are defined for the interval between discrete timesteps ($n \to n+1$). We denote by $\boldsymbol{u}^n$ the velocity field at the nth timestep, i.e., at time $t_n = nh$ (with h the timestep).

We begin by neglecting the $\phi \boldsymbol{f}_p$ terms in (3.13) and (3.4) and integrating those equations from t_n to $t_n + h$, yielding the following updates to the particle position and direction:

$$\boldsymbol{u}^* = \boldsymbol{u}^n + \int_{t_n}^{t_n+h} \mathrm{d}s \nabla \cdot \left[\rho^{-1} \boldsymbol{\sigma} - \boldsymbol{u}\boldsymbol{u}\right], \tag{3.14}$$

$$\boldsymbol{R}_I^{n+1} = \boldsymbol{R}_I^n + \int_{t_n}^{t_n+h} \mathrm{d}s \boldsymbol{V}_I, \tag{3.15}$$

$$\boldsymbol{Q}_I^{n+1} = \boldsymbol{Q}_I^n + \int_{t_n}^{t_n+h} \mathrm{d}s\, \mathrm{skew}(\boldsymbol{\Omega}) \cdot \boldsymbol{Q}_I. \tag{3.16}$$

At this point we must update the particle velocity field, even if the particle velocities have not changed. Assuming that the conservation of momentum holds for the velocity $\boldsymbol{u}^*$ in the absence of the $\phi \boldsymbol{f}_p$ term, the hydrodynamic forces and torques acting on a particle are determined by the balance of momentum received by the fluid from the particle:

$$\left[\int_{t_n}^{t_n+h} \mathrm{d}s \boldsymbol{F}_I^H\right] = \int \mathrm{d}\boldsymbol{x} \rho \phi_I^{n+1} \left(\boldsymbol{u}^* - \boldsymbol{u}_p^n\right), \tag{3.17}$$

$$\left[\int_{t_n}^{t_n+h} \mathrm{d}s \boldsymbol{N}_I^H\right] = \int \mathrm{d}\boldsymbol{x} \left[\boldsymbol{r}_I^{n+1} \times \rho \phi_I^{n+1} \left(\boldsymbol{u}^* - \boldsymbol{u}_p^n\right)\right]. \tag{3.18}$$

The result is that the particle velocities are updated to $\boldsymbol{V}_I^{n+1}$ and $\boldsymbol{\Omega}_I^{n+1}$. After these updates, equation (3.11) determines the final particle velocity field $\phi^{n+1} \boldsymbol{u}_p^{n+1}$, i.e. ϕ_I^{n+1}.

The last step is to determine the velocity field for the entire system at time t_{n+1}, which we do by considering the quantity $\phi \boldsymbol{f}_p$ introduced to preserve particle rigidity:

$$\boldsymbol{u}^{n+1} = \boldsymbol{u}^* + \left[\int_{t_n}^{t_n+h} \mathrm{d}s \phi \boldsymbol{f}_p\right], \tag{3.19}$$

$$\left[\int_{t_n}^{t_n+h} \mathrm{d}s \phi \boldsymbol{f}_p\right] = \phi^{n+1} \left(\boldsymbol{u}_p^{n+1} - \boldsymbol{u}^*\right) - \frac{h}{\rho} \nabla p_p, \tag{3.20}$$

The pressure due to the rigidity constraint is obtained from the continuity equation $\nabla \cdot \boldsymbol{u}^{n+1} = 0$. Because viscous stresses are applied throughout the entire spatial region, the no-slip boundary conditions at relevant particle surfaces are satisfied in the interior of the finite interface width ξ.

3.1.3 Smoothed Profile (SP) functions

In this section we describe the construction of SP functions for particles of arbitrary shapes. An SP function is a function ϕ that takes the value $\phi = 1$ in the interior of particles and the value $\phi = 0$ in fluid regions outside of particles, interpolating smoothly between these limits over a finite-thickness interface region of width ξ. For the particular case of spherical particles, several varieties of SP function with closed-form analytical expressions have been proposed [5]; one example is [9, 10]

$$\phi_I = \frac{1}{2}\left[\tanh\left(\frac{a - |\boldsymbol{r}_I|}{\xi}\right) + 1\right], \tag{3.21}$$

with a denoting the sphere radius. The interface width ξ is a freely adjustable parameter, but for numerical stability it is generally advisable to choose ξ to be one or two times the spacing between points in the computational mesh.

The cross-sectional particle image in Figure 3.2 shows an interface region of width ξ separating the solid material region inside the particle and the fluid region outside the particle.

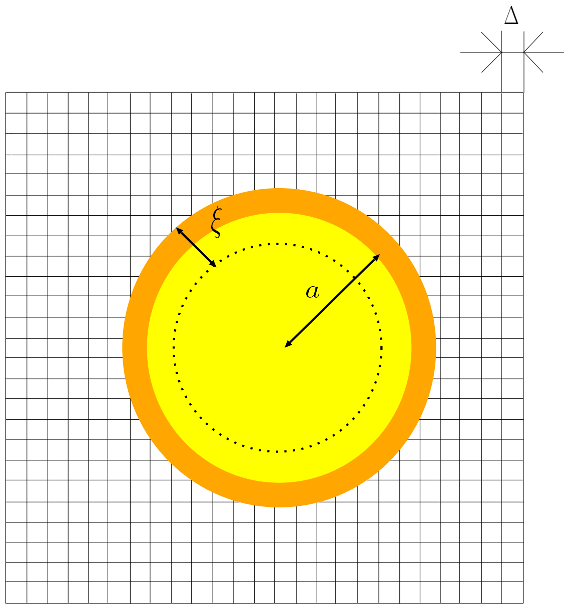


Figure 3.2: Cross-sectional view of a particle in a fluid to illustrate the notion of an SP function. Δ is the lattice spacing, a the particle radius, and ξ the width of the interface region. In this case the interface region occupies a volume of approximately $\sim \pi a^2 \xi$ at the particle surface. Reproduced from Phys. Rev. E 71, 036707[5], Copyright 2005, with permission from the American Physical Society.

Non-spherical particles of complex shapes may be represented as aggregates of spherical beads. These beads may overlap with each other, allowing arbitrary particles to be represented as assemblies of beads. Denoting by n_I the number of spherical beads used for the Ith particle, an SP function appropriate for the overall configuration is

$$\Phi_I(\boldsymbol{x}, t) = \sum_{i=1}^{n_I} \phi_{I,i}(\boldsymbol{x}, t), \tag{3.22}$$

$$\phi_I(\boldsymbol{x}, t) = \frac{\Phi_I}{\max(\Phi_I, 1)}, \tag{3.23}$$

where $\phi_{I,i}$ is the SP function for the ith spherical bead in the bead assembly for the Ith particle.

Once an SP function has been defined for a particle—*any* particle, including particles of complicated shapes—the hydrodynamic impulses on the particle and the associated momentum impulse to the fluid are given by equations (3.17), (3.18), and (3.20).

Because the inertia of the fluid is determined by the SP algorithm discussed above, added mass effects are captured by the hydrodynamic impulse in equation (3.17). To clarify the nature of the associated inertial effects, it is convenient to decompose the integrand of expression (3.17), for the hydrodynamic impulse, in the form

$$\begin{aligned}\rho\phi_I^{n+1}\left(\boldsymbol{u}^* - \boldsymbol{u}_p^n\right) &= -\rho\phi_I^{n+1}\left(\boldsymbol{u}_p^{n+1} - \boldsymbol{u}^*\right) + \rho\phi_I^{n+1}\left(\boldsymbol{u}_p^{n+1} - \boldsymbol{u}_p^n\right)\\ &= -\rho\int_{t_n}^{t_n+h} \mathrm{d}s\phi_I\boldsymbol{f}_p - h\nabla p_p\end{aligned}$$

$$+\rho\phi_I^{n+1}\left[\boldsymbol{V}_i^{n+1}-\boldsymbol{V}_i^{n}+\left(\boldsymbol{\Omega}_I^{n+1}-\boldsymbol{\Omega}_I^{n}\right)\times\boldsymbol{r}_I^{n+1}\right], \tag{3.24}$$

Spatial integration subject to $\int \mathrm{d}\boldsymbol{x}\phi_I\boldsymbol{r} = 0$ yields

$$\boldsymbol{F}_I^H = -\int \mathrm{d}\boldsymbol{x}\rho\phi_I\boldsymbol{f}_p - \int \mathrm{d}\boldsymbol{x}\phi_I\nabla p_p + M_I\frac{\rho}{\rho_{p,I}}\dot{\boldsymbol{V}}_I, \tag{3.25}$$

with $\rho_{p,I}$the mass density of the Ith particle. From this equation we see that the hydrodynamic force in (3.17) consists of the counterreaction to the action of the rigidity constraint plus added mass effects.

The accuracy of SP calculations depends on both the timestep h and the interface width ξ. Luo et al. [11] reported that computational accuracy does not increase monotonically as h decreases, and that optimal performance is attained when h is of the same order of magnitude as $\xi^2\rho/\eta$ (the time required for momentum to propagate a distance ξ via viscous diffusion). This is a reflection of the fact that, in order to study the viscous Stokes layer resulting from the impulse [equation (3.20)] of the rigidity constraints, it is necessary to select an appropriately commensurate value of the interface width.

3.1.4 Procedure for determining particle temperature

1. Fix values for α^V and α^Ω and simulate a one-particle system at thermal equilibrium.

2. From your simulation results, compute values for the translational and rotational diffusion coefficients D_{sim}^V and D_{sim}^Ω by considering mean-square displacements or equivalent quantities.

3. Compare your values for D_{sim}^V and D_{sim}^Ω to the analytical expressions for the diffusion coefficients in an infinitely dilute one-particle system. (These expressions are: $D_0^V = k_BT^V/6\pi\eta a$ for translational motion, $D_0^\Omega = k_BT^\Omega/8\pi\eta a^3$ for rotational motion.) Based on this comparison, compute the temperatures T^V and T^Ω associated with the translational and rotational motion of the particle.

4. If $T^\Omega \neq T^V$, adjust the values of α^V and α^Ω and repeat the calculation. Continue in this way until you have found values of α^V and α^Ω for which $T^\Omega = T^V(= T)$.

Further discussion of this temperature-determination strategy may be found in Refs.[12–14].

3.2 Input UDF settings

3.2.1 Fluid settings

The motion of colloidal particles in a Newtonian fluid may be simulated by setting `constitutive_eq` to `Navier_Stokes`. The settings that may be configured via input UDF files for this case are described below.

➤`constitutive_eq.Navier_Stokes`: Solvent properties

`constitutive_eq.Shear_Navier_Stokes.DX`
The lattice spacing Δ, which defines the length unit.

`constitutive_eq.Shear_Navier_Stokes.RHO`
Density of solvent.

`constitutive_eq.Shear_Navier_Stokes.ETA`
Viscosity of solvent.

`constitutive_eq.Shear_Navier_Stokes.kBT`
Particle temperature.

`constitutive_eq.Shear_Navier_Stokes.alpha_v`
Correction term for particle temperature associated with translational motion.[1]

`constitutive_eq.Shear_Navier_Stokes.alpha_o`
Correction term for particle temperature associated with rotational motion.

3.2.2 Configuring objects (particles)

The `object_type.type` setting specifies the type of particle. The possible values are `spherical_particle` for spherical particles, `chain` for flexible chains, or `rigid` for rigid bodies.

3.2.3 Choice of length and time units

The length unit is given by the lattice spacing Δ. The time unit τ_0 is determined by the fluid density ρ, the fluid viscosity coefficient η, and the lattice spacing Δ according to $\tau_0 = \rho\Delta^2/\eta$.

- For the Navier-Stokes equations, we choose a system of units in which $\rho = \eta = \Delta = 1$.
- Assuming the input UDF file specifies RHO= A, ETA= B, and DX= C, the maximum wavenumber k_{max} may be determined from C and the upper bound on the timestep is determined from the momentum diffusion time in the form $T_{step} = (A/B)/k_{max}^2$. The actual simulation timestep is related to this bound through a multiplicative scale factor, i.e. $\Delta t = \texttt{factor} \times T_{step}$.
- Consider the correspondence between physical quantities in simulations and in reality. A grid spacing of 1 µm is reasonable for the length scales we wish to consider. Assuming we use water ($\eta = 1 \times 10^{-3}$ Pa s, $\rho = 1 \times 10^3$ kg m^{-3}) as a solvent, the time unit is then $\tau_0 = 1 \times 10^{-6}$ s.

[1]When adding thermal diffusion of microparticles to simulations, you must specify a particle temperature. In principle, particle temperatures may be estimated from the diffusive motion of particles at thermal equilibrium; however, this approach may yield incorrect temperatures due to numerical factors such as interface thickness. In such cases, `alpha_v` may be used to correct temperature errors. If the measured temperature is too low by a factor of n (that is, the measured temperature is $1/n$ times the correct temperature), then setting `alpha_v` to n will reproduce the correct temperature (and similarly with `alpha_o` for the rotational temperature). For a particle radius of 4 or 5 and interface thickness $\xi = 2$, setting `alpha_v=alpha_o=1` seems to reproduce specified temperatures.

3.3 Computational examples

3.3.1 Sedimentation of particles

The input UDF files for this example are `gravity.udf` and `gravity100000.udf` in the `Examples/02/` folder. These files describe simulations of particles suspended in fluid sedimenting under the influence of gravity; the number of particles is 3204 (`gravity.udf`) or 100000 (`gravity1000000.udf`). To run these simulations, first confirm that the folders `./avs_g1/` and `./avs_g1/avs/` have been created on your system,[2] then execute the following commands:

```
    $ ../../kapsel -Igravity.udf -Ooutput.udf -Ddefine.udf -Rrestart.udf
    $ ../../kapsel -Igravity100000.udf -Ooutput.udf -Ddefine.udf -Rrestart.udf
```

For this example (Fig. 3.3), we choose a $256 \times 256 \times 512$ computational mesh. In the `gravity.udf` file we specify the following values for simulation parameters: number of particles $N_p = 3204$, particle diameter $D = 10$, interface thickness $\xi = 2$. With these settings, the fractional volume occupied by the particles is $\varphi = 0.05$. Figure 3.3 shows a snapshot of the 3204-particle suspension sedimenting due to gravity. of gravity.

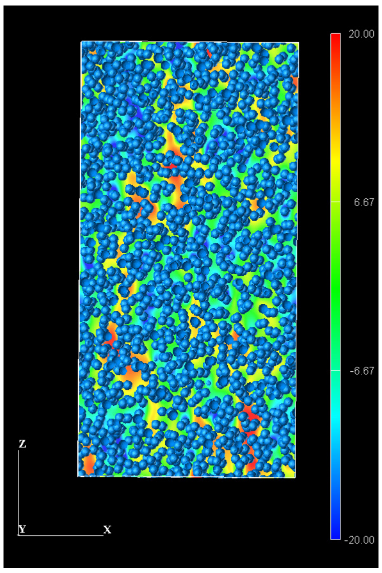


Figure 3.3: A fluid suspension of 3204 particles sedimenting under the influence of gravity. Gravity acts in the z direction. Colors indicate fluid field velocity in the z direction.

The file `gravity100000.udf` specifies the following parameter values: number of particles $N_p = 100000$, particle diameter $D = 4$, interface thickness $\xi = 2$. With these settings, the fractional volume occupied by the particles is $\varphi = 0.10$. Figure 3.4 shows a snapshot of the 100000-particle suspension sedimenting due to gravity.

3.3.2 Particle diffusion

The input UDF file for this example is `repulsive.udf` in the `Examples/03/` folder. This UDF file specifies a simulation of diffusion phenomena for a system of Brownian particles with repulsive interparticle forces but no other interactions.

```
    $ ../../kapsel -Irepulsive.udf -Ooutput.udf -Ddefine.udf -Rrestart.udf
```

For this example (Fig. 3.5), we choose a $128 \times 128 \times 128$ computational mesh. The fractional volume occupied by the particles in this case is $\varphi = 0.064$. We specify the following values for the simulation parameters: number of particles $N_p = 500$, particle diameter $D = 8$, interface thickness $\xi = 2$, temperature $k_B T = 5$. The solvent has a density $\rho = 1$ and a viscosity coefficient $\eta = 1$. Figure 3.5 shows a snapshot of this suspension of particles diffusing under the influence of repulsive interparticle interactions.

[2]These folders do not need to be created if you do not need AVS data; set the AVS switch to OFF in that case. This applies for all other sample calculations as well.

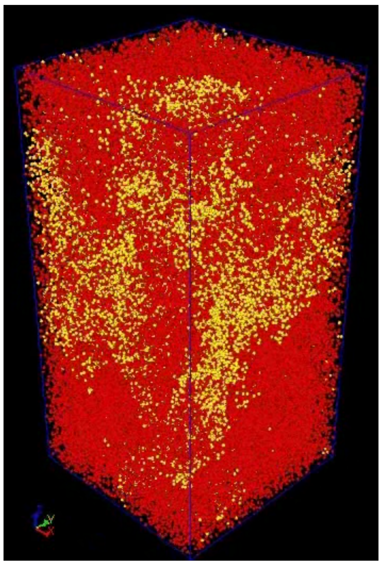


Figure 3.4: The same as in the previous figure, but for a suspension of 100000 particles. Colors are for visualization purposes only, all particles are identical.

3.3.3 Particle aggregates

The input UDF files for this example are `aggregate.udf`, `aggregate_LE.udf`, and `aggregate_LE_2.udf` in the `Examples/05/` folder. `aggregate.udf` describes a simulation of aggregation and diffusion phenomena in a system of particles with attractive interparticle interactions.

```
$ ../../kapsel -Iaggregate.udf -Ooutput.udf -Ddefine.udf -Rrestart.udf
```

For this example (Fig. 3.6), we choose a $128 \times 128 \times 128$ computational mesh. The fractional volume occupied by the particles in this case is $\varphi = 0.064$, and we specify the following values for the simulation parameters: number of particles $N_p = 500$, particle diameter $D = 8$, interface thickness $\xi = 2$, temperature $k_B T = 5$. The solvent has density $\rho = 1$ and viscosity coefficient $\eta = 1$. Figure 3.6 shows aggregation and diffusion phenomena for this suspension of particles with attractive interparticle interactions.

The alternative UDF input files `aggregate_LE.udf` and `aggregate_LE_2.udf` describe the same simulation, but now in the presence of shear flow.

```
$ ../../kapsel -Iaggregate_LE.udf -Ooutput.udf -Ddefine.udf -Rrestart.udf
$ ../../kapsel -Iaggregate_LE_2.udf -Ooutput.udf -Ddefine.udf -Rrestart.udf
```

For this example (Fig. 3.7), we use the same values of computational and simulation parameters used for the simulation of Fig. 3.6: $128 \times 128 \times 128$ computational mesh, fractional volume occupation by particles $\varphi = 0.064$, number of particles $N_p = 500$, particle diameter $D = 8$, interface thickness $\xi = 2$, temperature $k_B T = 5$. The solvent has a density $\rho = 1$ and a viscosity coefficient $\eta = 1$.

The files `aggregate_LE.udf` and `aggregate_LE_2.udf` specify shear-flow velocities of $\dot{\gamma} = 0.005$ and $\dot{\gamma} = 0.05$, respectively. Simulation snapshots for these two cases are shown in the left and right panels of Figure3.7.

3.3.4 Motion of a particle chain

The input UDF files for this example are `s000_t010_free.udf`, `s001_t010_free.udf`, and `s001_t000_free.udf` in the `Examples/06` folder.

`s000_t010_free.udf` describes a simulation of a single particle chain in a static fluid. `s001_t010_free.udf` and `s001_t000_free.udf` describe similar calculations, but now in the presence of zig-zag shear flows.

```
$ ../../kapsel -Is000_t010_free.udf -Ooutput.udf -Ddefine.udf -Rrestart.udf
$ ../../kapsel -Is010_t010_free.udf -Ooutput.udf -Ddefine.udf -Rrestart.udf
```

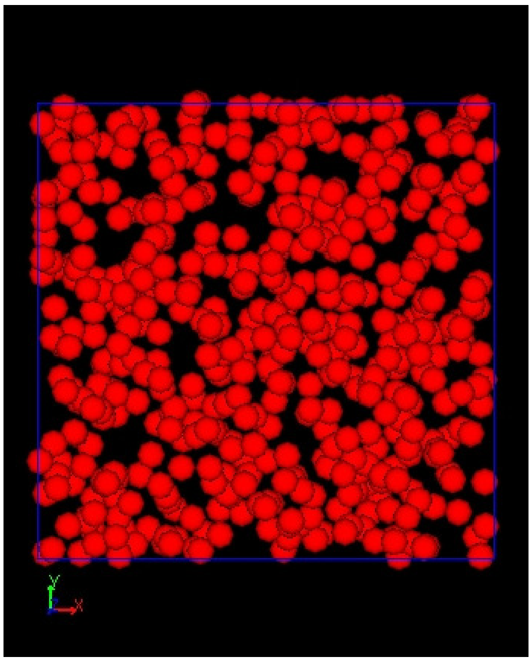


Figure 3.5: Snapshot of a suspension of particles with repulsive interparticle forces but no other interactions.

```
$ ../../kapsel -Is010_t000_free.udf -Ooutput.udf -Ddefine.udf -Rrestart.udf
```

For this example (Fig. 3.8), we choose a 32 × 64 × 16 computational mesh. The fractional volume occupied by the particles in this case is $\varphi = 0.005$. We specify the following values for the simulation parameters: number of particles per chain $N_p = 5$, particle diameter $D = 4$, interface thickness $\xi = 2$. `s000_t010_free.udf` and `s001_t010_free.udf` specify a temperature of $k_B T = 1.0$. `s001_t000_free.udf` specifies a temperature of $k_B T = 0.0$. `s001_t010_free.udf` and `s001_t000_free.udf` specify a shear velocity of $s = 0.01$.

Fig. 3.8 shows snapshots from the simulations described by these files. The left snapshot is for a particle chain in a static fluid at temperature $k_B T = 1.0$. The center snapshot is for a particle chain in a zig-zag shear flow at temperature $k_B T = 1.0$. The right snapshot is for a particle chain in a zig-zag shear flow at temperature $k_B T = 0.0$.

3.3.5 Motion of an arbitrarily-shaped rigid particle

The input UDF files for this example are `N30k3.0p52.0_vy.udf`, `N30k3.0p52.0_omegay.udf`, `N30k3.0p52.0_gravity.udf`, `helixes_gravity.udf`, and `helixes_shear0005.udf` in the `Examples/08/` folder.

`N30k3.0p52.0_vy.udf` describes a simulation of a helical particle chain exhibiting translational motion at fixed velocity.

```
$ ../../kapsel -IN30k3.0p52.0_vy.udf -Ooutput.udf -Ddefine.udf -Rrestart.udf
```

For this example we choose a 256 × 256 × 256 computational mesh. The fractional volume occupied by the particles in this case is $\varphi = 0.00048$. We specify the following values for the simulation parameters: number of particles per chain $N_p = 30$, particle diameter $D = 4$, interface thickness $\xi = 1$, velocity $V = -0.005$

`N30k3.0p52.0_omegay.udf` describes a simulation of a helical particle chain exhibiting rotational motion at fixed angular velocity.

```
$ ../../kapsel -IN30k3.0p52.0_omegay.udf -Ooutput.udf -Ddefine.udf -Rrestart.udf
```

For this example we choose a 256 × 256 × 256 computational mesh. The fractional volume occupied by the particles in this case is $\varphi = 0.00048$. We specify the following values for the simulation parameters: number of particles per chain $N_p = 30$, particle diameter $D = 4$, interface thickness $\xi = 1$, angular velocity $\Omega = 0.00041$.

`N30k3.0p52.0_gravity.udf` describes a simulation of a helical particle chain sedimenting under the influence of a fixed external force (gravity).

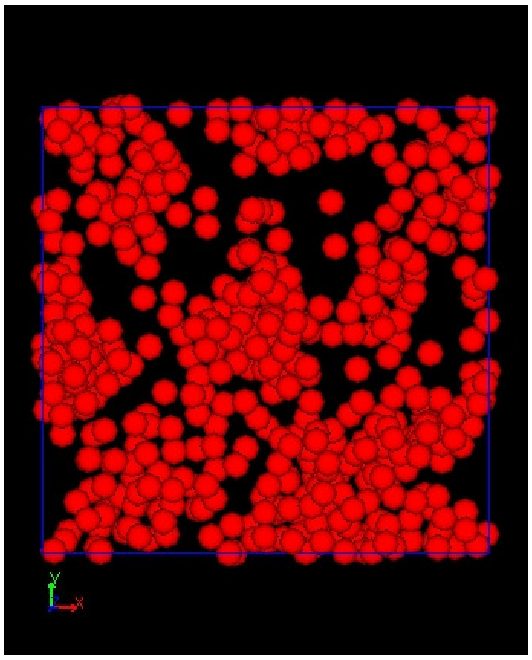


Figure 3.6: Snapshot illustrating aggregation and diffusion phenomena in a suspension of particles with attractive interparticle interactions.

```
$ ../../kapsel -IN30k3.0p52.0_gravity.udf -Ooutput.udf -Ddefine.udf -Rrestart.udf
```

For this example we choose a 256 × 256 × 256 computational mesh. The fractional volume occupied by the particles in this case is $\varphi = 0.00048$. We specify the following values for the simulation parameters: number of particles per chain $N_p = 30$, particle diameter $D = 4$, interface thickness $\xi = 1$, gravity $g = -0.01$.

`helixes_gravity.udf` describes a simulation of three helical particle chains sedimenting due to gravity.

```
$ ../../kapsel -Ihelixes_gravity.udf -Ooutput.udf -Ddefine.udf -Rrestart.udf
```

For this example (Fig. 3.9) we choose a 256 × 256 × 256 computational mesh. The fractional volume occupied by the particles in this case is $\varphi = 0.0016$. We specify the following values for the simulation parameters: number of particles per chain $N_p = 100$, particle diameter $D = 4$, interface thickness $\xi = 1$, gravity $g = -0.01$. Figure 3.9 shows a snapshot of the three sedimenting helical particle chains.

`helixes_shear0005.udf` describes a simulation of three helical particle chains in a shear flow.

```
$ ../../kapsel -Ihelixes_shear0005.udf -Ooutput.udf -Ddefine.udf -Rrestart.udf
```

For this example we choose a 256 × 256 × 256 computational mesh. The fractional volume occupied by the particles in this case is $\varphi = 0.0016$. We specify the following values for simulation parameters: number of particles per chain $N_p = 100$, particle diameter $D = 4$, interface thickness $\xi = 1$, shear velocity $\dot{\gamma} = 0.005$.

3.3.6 Sedimentation of arbitrarily-shaped rigid particles

The input UDF file for this example is `input_rigid_free1.udf` in the `Examples/10/` folder. This file describes a simulation of a body falling through a fluid at Reynolds number $Re = 800$.

```
$ ../../kapsel -Iinput_rigid_free1.udf -Ooutput.udf -Ddefine.udf -Rrestart.udf
```

For this example (Fig. 3.10) we choose a 64 × 128 × 64 computational mesh. The density of the falling body is 1.5 times the density of the fluid, and the gravity strength is $g = -50$. Figure 3.10 shows a snapshot of the falling body at $Re \simeq 800$.

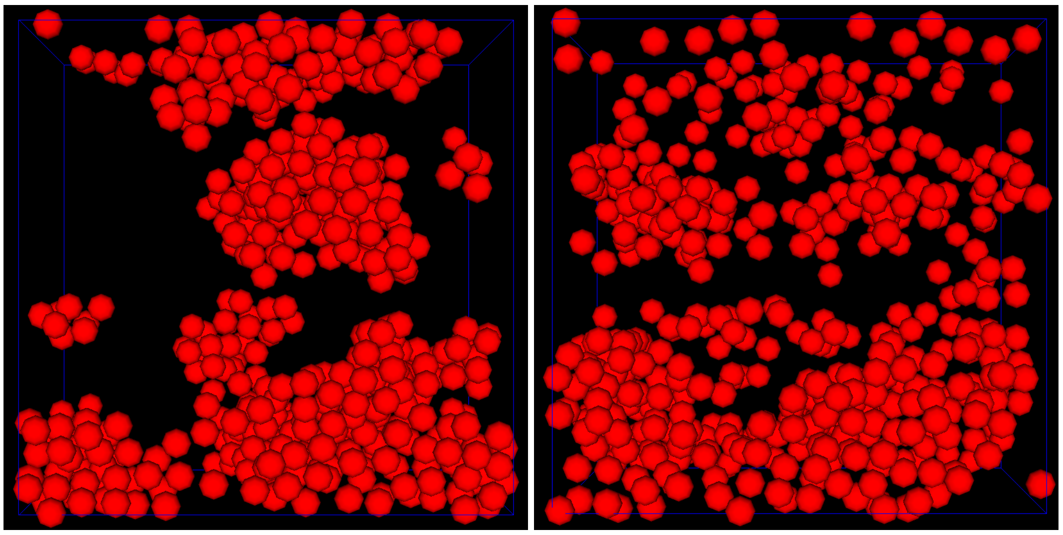


Figure 3.7: Snapshots from simulations like that of Fig. 3.6, but now in the presence of weak (left) or strong (right) shear flow.

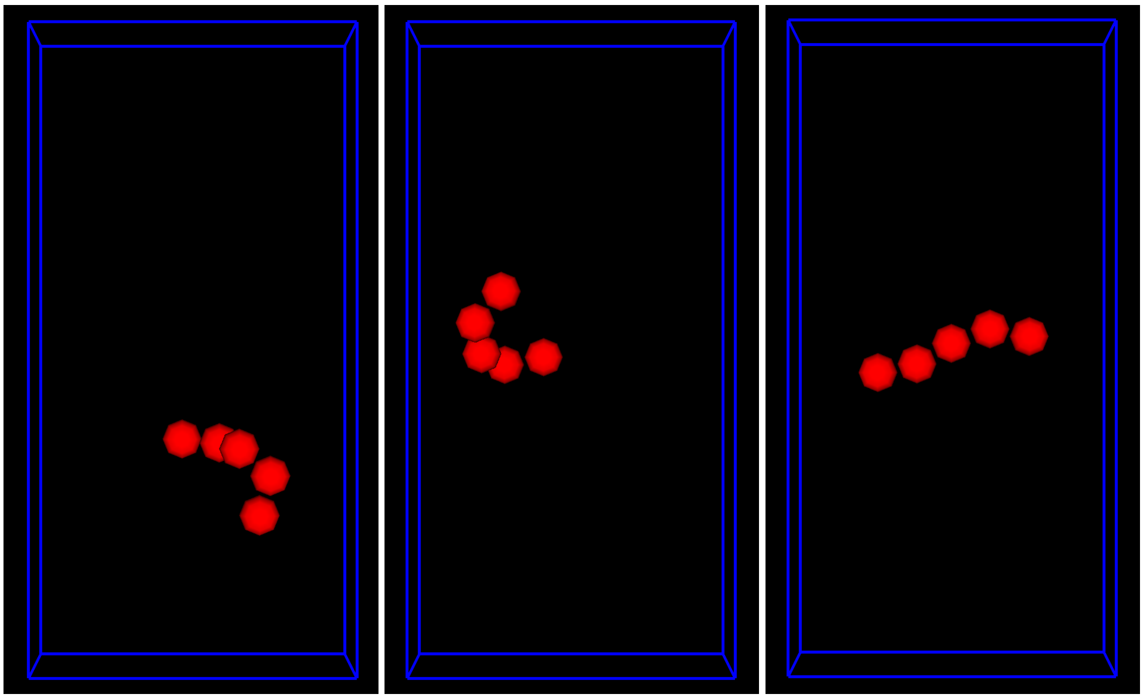


Figure 3.8: Left: Snapshot of a single flexible particle chain in a static fluid at temperature $k_B T = 1.0$. Center: Snapshot of a particle chain in a zig-zag shear flow at temperature $k_B T = 1.0$. Right: Snapshot of a particle chain in a zig-zag shear flow at temperature $k_B T = 0.0$.

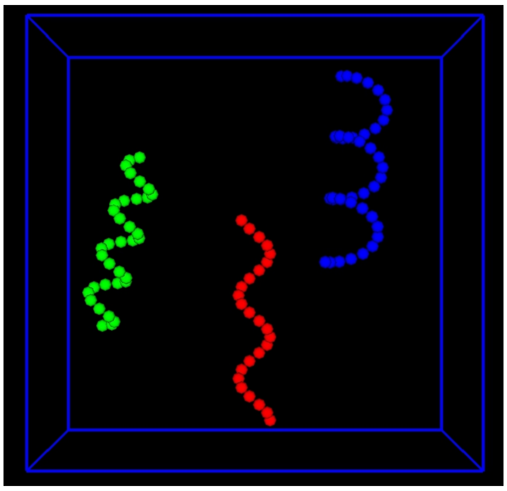


Figure 3.9: Snapshot of three helical particle chains sedimenting due to gravity.

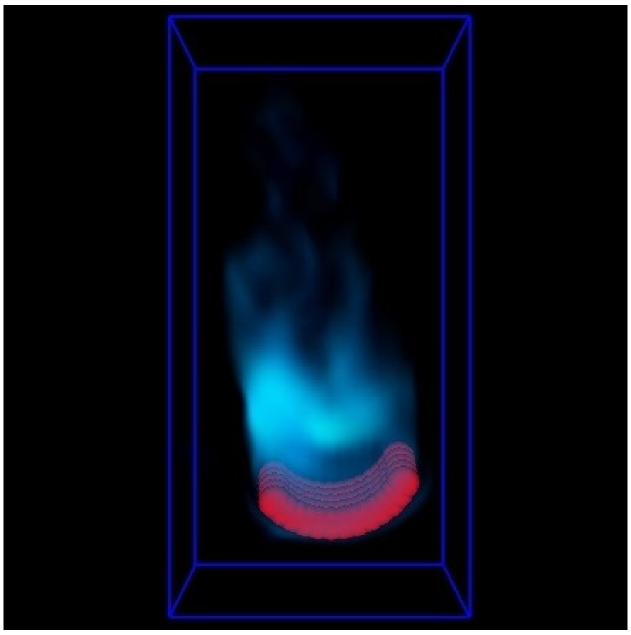


Figure 3.10: Snapshot of a body falling through a fluid at Reynolds number *Re* = 800. Light blue shading indicates fluid velocity magnitude.

Chapter 4

Simulating disperse particle systems in shear flows

4.1 Theoretical background and basic equations

Colloidal suspensions—collections of colloidal particles dispersed in liquids (solvents)—are important and widely-studied systems in chemical engineering, mechanical engineering, physics, and many other fields of science and engineering. Applications of these systems span a wide range of products—from foodstuffs to paints, pigments, cosmetics, slurries, and more—and a proper understanding of their fluid-mechanical behavior is important for many industrial production and treatment processes. However, the dynamical behavior of microparticle suspensions can be extremely complicated due the effects of interparticle interactions, thermal fluctuations, and other factors, and accurately modeling the fluid-mechanical behavior of these systems is a difficult challenge. Microparticle suspensions exhibit a wide variety of behavior in various circumstances. For example, studies of transport phenomena in these systems reveal that their viscosities and other fluid-mechanical properties depend strongly on factors such as dispersed particle concentrations, fluid-flow strength, and the nature of particle-fluid interfaces. The task of elucidating the relationship between such macroscopic fluid-mechanical properties and their microscopic mechanisms ranks among the central themes of basic research on colloidal suspensions.

Despite the great practical importance of microparticle-suspension rheology, constructing a general theoretical framework for these systems remains an extremely complex task (discussed, for example, in textbooks [15] and relatively recent (Japanese-language) reviews [16, 17]). For a solvent of viscosity η, the viscosity η^{app} of a suspension of particles in regions of low particle concentration (characterized by the particle volume fraction φ), is known on theoretical grounds to vary linearly with φ in the form

$$\frac{\eta^{app}}{\eta} = 1 + \frac{5}{2}\varphi. \tag{4.1}$$

This is Einstein's famous viscosity relation (see, for example, [15]). However, Einstein's formula holds only for dilute suspensions with particle volume fractions of $\varphi \ll 10\%$ or so. At higher particle concentrations, one must account for the effects of interparticle interactions and the associated changes in particle structure, as well as the emergence of phenomena such as glass transitions and crystallization; these complications pose grave challenges for the task of theoretically predicting particle-suspension rheologies from basic physical laws. Nonetheless, despite the difficulty of this task, its practical importance has motivated many experimental studies, whose findings have been organized in the form of various proposed empirical and quasi-empirical formulas for the viscosity of suspensions (such as the Doughherty-Krieger equation [18]; see also Refs. [17, 19] and references within).

As the particle concentration increases, phenomena begin to be affected by a variety of factors beyond the colloid volume fraction, and the goal of establishing a unified theory must be set aside in favor of separately analyzing individual phenomena. The SP method allows for direct computational simulation of solvent-induced multibody interactions among colloids, enabling studies of the rheology of many types of suspensions. The key strengths of this direct numerical approach to rheology analysis are its ability to account for *hydrodynamic interactions between particles*, *thermal fluctuations of particles*, and *shear flows*. KAPSEL allows simulation and rheology measurement (steady-state flow and dynamic viscoelasticity) for monodisperse systems in Newtonian fluids, including both simple and oscillatory shear flows.

KAPSEL uses Lees-Edwards periodic boundary conditions to represent uniform shear flows. The advantages of this approach include (a) the accurate imposition of shear velocities without the use of walls or other specialized boundaries, and (b) the absence of singular points (kinks) at which the velocity gradient changes sign within the system, as in zigzag flows. With Lees-Edwards periodic boundary conditions, one solves the constitutive equations in an oblique coordinate system in which the shear flow induces advection of lattice points with the passage of time. The coordinate transformations used to represent shear flows with Lees-Edwards periodic boundary conditions are depicted schematically in Figure 4.1. If there is no shear flow at time $t = t_0$, the collection of solid particles dispersed in the solvent is scattered over a rectangular lattice [Figure 4.1(a)]. If a shear flow develops at later times ($t > t_0$), both particles and lattice points are advected by the flow; however, although the computational mesh is deformed by the shear flow, the shapes of the particles remain unchanged. More specifically, the computational mesh has two representations—one in a fixed coordinate system [Figure 4.1(b)] and one in an oblique coordinate system distorted by the shear flow [Figure 4.1(c)]—where the shapes of the particles are distorted in the latter. The algorithm as we have described it resembles the SLLOD algorithm used to model uniform dispersion-free flows in non-equilibrium molecular-dynamics (MD) simulations, but implementing the algorithm in lattice-based systems requires extra caution [20–22]. Within the oblique coordinate system, which deforms as time passes due to the shear flow, we denote the velocity by $\boldsymbol{\xi} = \boldsymbol{u} - \boldsymbol{U}$. Then, using basis vectors $\hat{\boldsymbol{E}}_1 = \boldsymbol{e}_x + \gamma(t)\boldsymbol{e}_y, \hat{\boldsymbol{E}}_2 = \boldsymbol{e}_2, \hat{\boldsymbol{E}}_3 = \boldsymbol{e}_3$, the appropriate contravariant form of the Navier-Stokes equations with respect to $\boldsymbol{\xi}$ reads

$$\rho\left(\partial_{\hat{t}} + \hat{\xi}^j \hat{\nabla}_j\right)\hat{\xi}^i = \hat{\nabla}_j \hat{\sigma}^{ji} + \hat{\phi}\hat{f}^i - 2\dot{\gamma}(\hat{t})\hat{\xi}^2\delta^{i,1} \tag{4.2}$$

$$\hat{\nabla}_i \hat{u}^i = \hat{\nabla}_i \hat{\xi}^i = 0 \tag{4.3}$$

where hats ($\hat{\cdot}$) indicate tensor components in the oblique coordinate system. After solving these equations for the flow, we compute particle-fluid interactions in the fixed (laboratory) coordinate system to solve for the particle dynamics. Under shear flow, the spatial average of the instantaneous stress on the entire disperse particle system may be expressed

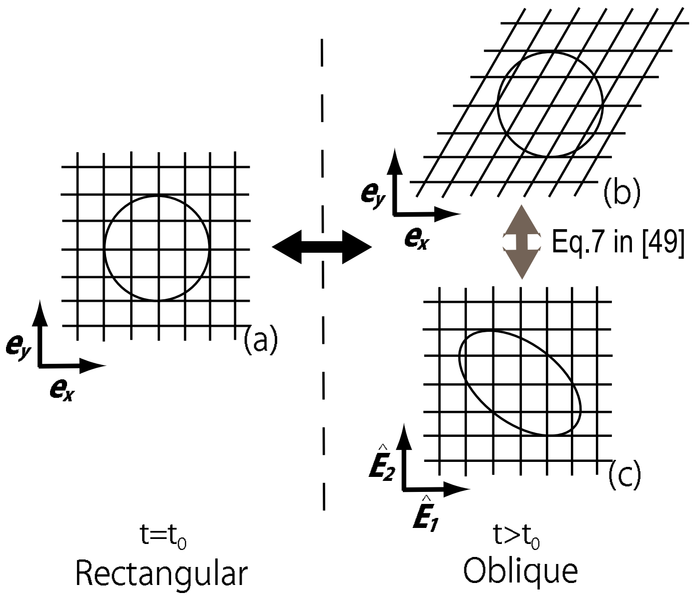


Figure 4.1: Schematic depiction of the coordinate transformations used to represent shear flows with Lees-Edwards periodic boundary conditions. (a) At time $t = t_0$, solid particles are scattered over a rectangular lattice. The emergence of shear flow at later times ($t > t_0$) induces advection of both the particles and the solvent (lattice points); however, only the computational mesh deforms, with particle shapes remaining unchanged. In other words, while the computational mesh deforms in the fixed (laboratory) coordinate system (b), in the oblique coordinate system (c) particles deform in response to the shear flow. The transformation between coordinate systems (b) and (c) may be found in equation (7) of Ref. [22]. Reproduced from J. Chem. Phys 134, 064110[22], Copyright 2011, with the permission of AIP Publishing.

in terms of the particle binding force $\rho\phi\boldsymbol{f}_p$ in the form

$$\boldsymbol{\Sigma} = \frac{1}{V}\int \mathrm{d}\boldsymbol{x}\left[\boldsymbol{\sigma} - \boldsymbol{x}\rho\phi\boldsymbol{f}_p + \boldsymbol{x}\boldsymbol{u}\cdot\boldsymbol{\nabla}(\rho\boldsymbol{u})\right]. \tag{4.4}$$

Subtracting from this the (spatially-averaged) contribution of the host fluid leaves just the particle contribution to the stress, $\boldsymbol{s} = \boldsymbol{\Sigma} - \langle\boldsymbol{\sigma}\rangle$.

4.1.1 Steady-state shear flow

For a steady-state shear flow, the apparent and intrinsic viscosities η_{app} and $[\eta]$ of the suspension are given by

$$\eta_{app} = \frac{\langle \Sigma^{xy} \rangle}{\dot{\gamma}} = \frac{\langle \sigma^{xy} \rangle}{\dot{\gamma}} + \frac{\langle s^{xy} \rangle}{\dot{\gamma}} = \eta + \frac{\langle s^{xy} \rangle}{\dot{\gamma}} \tag{4.5}$$

$$[\eta] = \frac{\eta_{app} - \eta}{\eta\varphi} = \frac{\langle s^{xy} \rangle}{\eta\varphi\dot{\gamma}}. \tag{4.6}$$

4.1.2 Oscillatory shear flow

Introducing an oscillatory shear flow allows measurement of the dynamic viscoelasticity of the suspension. In KAPSEL, a time-varying shear flow in a system is formed by using, as a control parameter, the shear velocity $\dot{\gamma}$, given by

$$\dot{\gamma}(t) = \dot{\gamma}_0 \cos(\omega t) \tag{4.7}$$

where $\dot{\gamma}_0$ is the shear-velocity amplitude. The maximum strain amplitude γ_0 may be evaluated in the form

$$\gamma_0 = \int_0^{\pi/2\omega} \dot{\gamma}(s) ds = \frac{\dot{\gamma}_0}{\omega}. \tag{4.8}$$

Small strains correspond to the regime of linear viscoelasticity, while for larger strains we are in the nonlinear viscoelasticity regime.

For an oscillatory shear flow $\dot{\gamma}(t)$, the time-dependent stress $\Sigma^{xy}(t)$ in the suspension may, within the linear-response regime, be expressed in the form

$$\Sigma^{xy}(t) = \sigma_0 \cos(\omega t - \delta) \tag{4.9}$$

where σ_0 is the stress amplitude and δ is the phase shift between the shear velocity $\dot{\gamma}$ and the shear stress Σ^{xy}. In this case, the storage modulus G' and loss modulus G'', which are functions of the dynamic viscoelasticity, may be expressed in the form

$$G'(\omega) = \frac{\omega\sigma_0 \sin\delta}{\dot{\gamma}_0}, \; G''(\omega) = \frac{\omega\sigma_0 \cos\delta}{\dot{\gamma}_0}. \tag{4.10}$$

G' and G'' respectively characterize the elastic and viscous behavior of the suspension. KAPSEL takes the shear velocity $\dot{\gamma}(t)$ of equation (4.7) as an input and computes the shear stress $\Sigma^{xy}(t)$ of equation (4.9) as the corresponding response (output). The moduli $G'(\omega)$ and $G''(\omega)$ may be determined by conducting simulations for various values of ω and fitting the temporal evolution of variables to the above expressions. The derivations of these relations may be found in textbooks on rheology.

4.2 Input UDF files

4.2.1 Fluid settings

The motion of colloidal particles under shear flow in a Newtonian fluid may be simulated by setting `constitutive_eq` to `Shear_Navier_Stokes_Lees_Edwards`. The settings that may be configured via input UDF files for this case are described below.

➤`constitutive_eq.Shear_Navier_Stokes`: Solvent properties

`constitutive_eq.Shear_Navier_Stokes.DX`
The lattice spacing Δ, which defines the length unit.

`constitutive_eq.Shear_Navier_Stokes.RHO`
Solvent density.

`constitutive_eq.Shear_Navier_Stokes.ETA`
Solvent viscosity.

`constitutive_eq.Shear_Navier_Stokes.kBT`
Particle temperature.

`constitutive_eq.Shear_Navier_Stokes.alpha_v`
Correction term for particle temperature associated with translational motion.[1]

`constitutive_eq.Shear_Navier_Stokes.alpha_o`
Correction term for particle temperature associated with rotational motion.

`constitutive_eq.Shear_Navier_Stokes.External_field.type`
Set to `DC` (steady-state shear flow) or `AC` (oscillatory shear flow).

`constitutive_eq.Shear_Navier_Stokes.External_field.DC.Shear_rate`
Shear velocity $\dot{\gamma}$.

`constitutive_eq.Shear_Navier_Stokes.External_field.AC.Shear_rate`
Oscillatory shear velocity amplitude $\dot{\gamma}_0$.

`constitutive_eq.Shear_Navier_Stokes.External_field.AC.Frequency`
Shear velocity frequency ω.

4.2.2 Object (particle) settings

The `object_type.type` setting specifies the type of particle. The possible values are `spherical_particle` for spherical particles, `chain` for flexible chains, or `rigid` for rigid bodies.

[1] In principle, particle temperatures may be estimated from the diffusive motion of particles at thermal equilibrium; however, this approach may yield incorrect temperatures due to numerical factors such as interface thickness. In such cases, `alpha_v` may be used to correct temperature errors. If the measured temperature is (say) too low by a factor of 1.2, then setting `alpha_v` to 1.2 will reproduce the correct temperature (and similarly with `alpha_o` for the rotational temperature). For a particle radius of 4 or 5 and interface thickness $\xi = 2$, the choice `alpha_v=alpha_o=1` reproduces correct temperatures.

4.3 Computational examples

4.3.1 Rheology of particle suspensions under steady-state shear flow

The input UDF files for this example are `colloid_LE_v31_t01.udf` and `colloid_LE_v55_t50.udf` in the `Examples/07/` folder.

```
$ ../../kapsel -Icolloid_LE_v31_t01.udf -Ooutput.udf -Ddefine.udf -Rrestart.udf
$ ../../kapsel -Icolloid_LE_v55_t50.udf -Ooutput.udf -Ddefine.udf -Rrestart.udf
```

For this example (Fig. 4.2), we choose a 64×64×64 computational mesh. We specify the following values for the simulation parameters: shear velocity $\dot{\gamma} = 0.001$, particle radius $a = 4$, interface thickness $\xi = 2$. `colloid_LE_v31_t01.udf` specifies $N_p = 300$ particles at temperature $k_BT = 0.1$; for this case the fractional volume occupied by the particles is $\varphi = 0.307$. `colloid_LE_v55_t50.udf` specifies $N_p = 540$ particles at temperature $k_BT = 5.0$; for this case the fractional volume occupied by the particles is $\varphi = 0.552$.

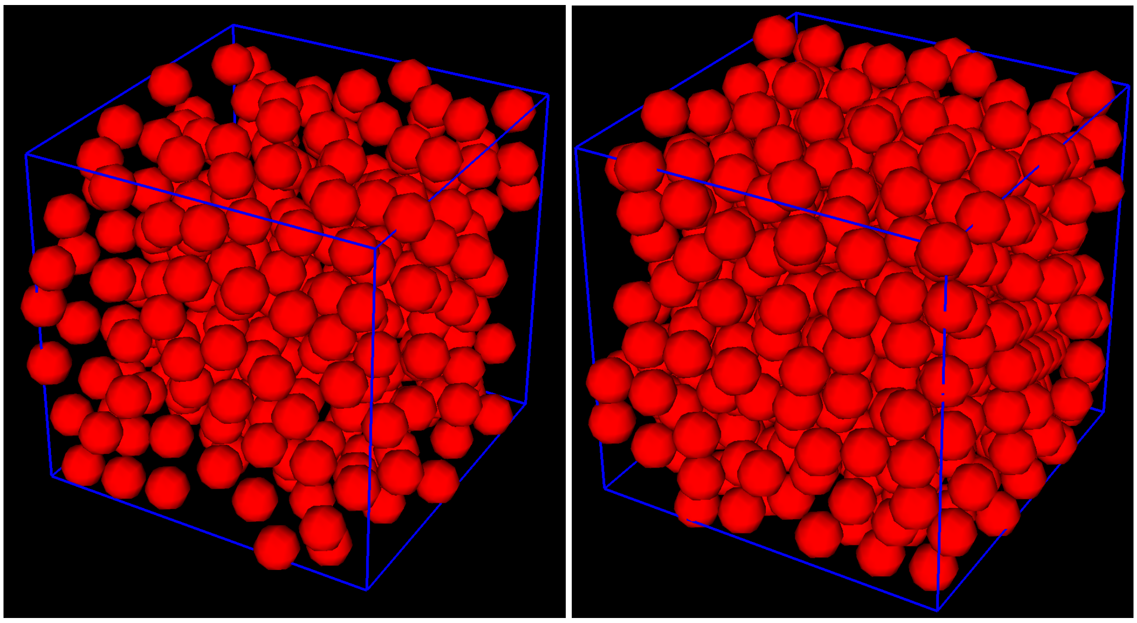


Figure 4.2: Flow snapshots for a particle suspension under steady-state shear flow using Lees-Edwards periodic boundary conditions. Left: $N_p = 300$ particles, temperature $k_BT = 0.1$. Right: $N_p = 540$ particles, temperature $k_BT = 5.0$.

Setting `constitutive_eq` to `Shear_Navier_Stokes_Lees_Edwards` and `External_field.type` to `DC` results in the following output written to standard error:

```
#1:time 2:shear_rate 3:degree_oblique 4:shear_strain_temporal 5:lj_dev_stress_temporal 6:
0.0748173 -0.00588609 -0.000440381 ...
```

⋮

The significance of this output stream is as follows.

- `1:time`... Elapsed time
- `2:shear_rate`... Shear velocity
- `3:degree_oblique`... Strain in oblique coordinate system[2]
- `4:shear_strain_temporal`... Applied strain ($\dot{\gamma}t$)

[2]KAPSEL represents uniform shear flows by performing calculations in an oblique coordinate system that distorts in the shear direction with the passage of time. For computational stability, when the strain in the oblique coordinate system reaches a certain threshold, its value is remapped into the orthogonal coordinate system. The value of `3:degree_oblique` is the strain with this remapping taken into account. Note that this differs from the applied strain ($\dot{\gamma}t$) on the system. For further details, see Ref. [23].

- `5:lj_dev_stress_temporal`... Shear stress due to interparticle potential
- `6:shear_stress_temporal_old`... Shear stress due to particle-fluid interactions
- `7:shear_stress_temporal_new`... Shear stress due to particle-fluid interactions[3]
- `8:reynolds_stress`... Reynolds shear stress
- `9:fluid_stress`... Shear stress due to bulk fluid
- `10:interfacial_stress`... Shear stress due to fluid-fluid interface
- `11:apparent_stress`... Shear stress on suspension[4]
- `12:viscosity`... Viscosity of suspension[5]

Fig. 4.3 plots the temporal evolution of the stress on the suspension, while Fig. 4.4 plots the relationship between viscosity and strain.

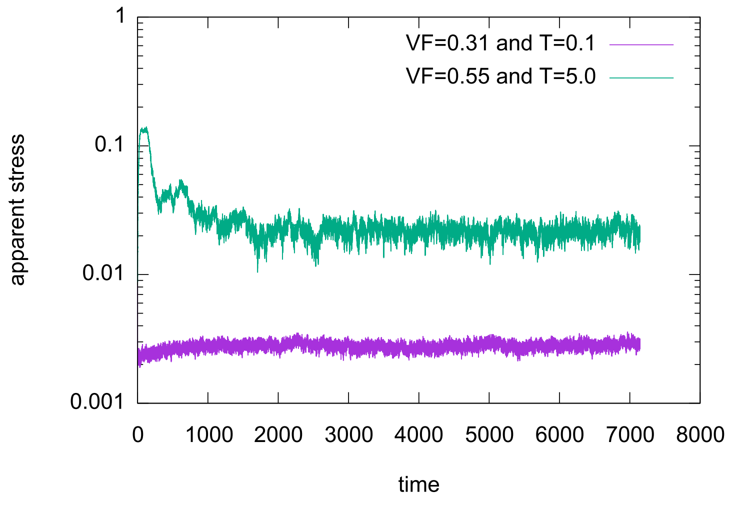


Figure 4.3: Temporal evolution of the stress for a suspension in steady-state shear flow. The x and y axes represent columns 1 and 11 in the output stream.

4.3.2 Rheology of particle suspensions under oscillatory shear flow

The input UDF files for this example are `colloid_LE_v31_t01_AC.udf` and `colloid_LE_v55_t50_AC.udf` in the `Examples/07` folder.

```
$ ../../kapsel -Icolloid_LE_v31_t01_AC.udf -Ooutput.udf -Ddefine.udf -Rrestart.udf
$ ../../kapsel -Icolloid_LE_v55_t50_AC.udf -Ooutput.udf -Ddefine.udf -Rrestart.udf
```

For this example (Fig. 4.5), we choose a 64×64×64 computational mesh. We specify the following values for the simulation parameters: shear velocity $\dot{\gamma} = 0.001$, particle radius $a = 4$, interface thickness $\xi = 2$. `colloid_LE_v31_t01_AC.udf` specifies $N_p = 300$ particles at temperature $k_BT = 0.1$; for this case the fractional volume occupied by the particles is $\varphi = 0.307$. `colloid_LE_v55_t50_AC.udf` specifies $N_p = 540$ particles at temperature $k_BT = 5.0$; for this case the fractional volume occupied by the particles is $\varphi = 0.552$.

Setting `constitutive_eq` to `Shear_Navier_Stokes_Lees_Edwards` and `External_field.type` to `AC` results in the following output written to standard error:

[3] KAPSEL implements two methods (`old` and `new`) for computing the shear stress due to particle-fluid interactions. In most cases the `new` method is the best choice.

[4] This is the sum of `5:lj_dev_stress_temporal`, `7:shear_stress_temporal_new`, `8:reynolds_stress`, `9:fluid_stress`, and `10:interfacial_stress`.

[5] The result of dividing `11:apparent_stress` by `2:shear_rate`

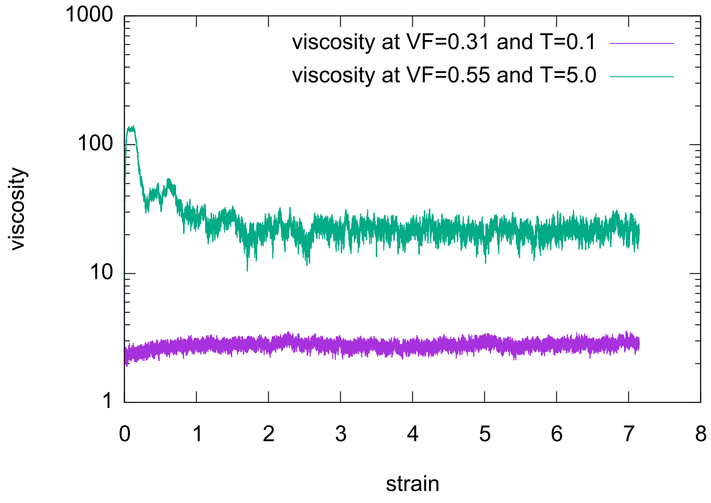
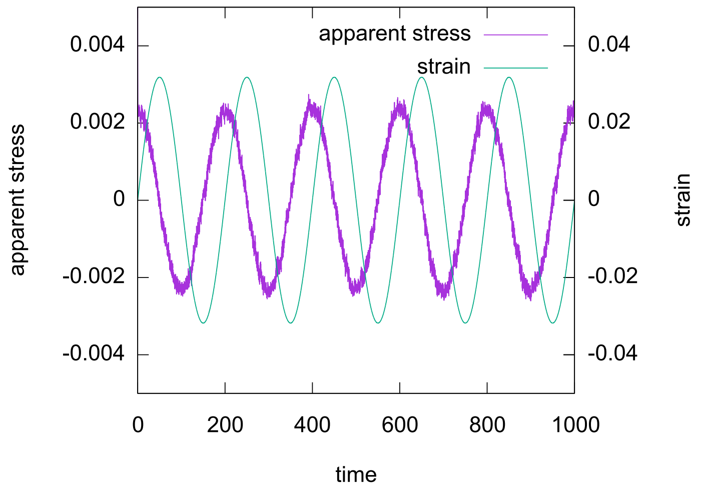


Figure 4.4: Viscosity-strain relationship for a suspension under steady-state shear flow. The *x* and *y* axes represent columns 4 and 12 in the output stream.

```
#1:time 2:shear_rate 3:degree_oblique 4:shear_strain_temporal 5:lj_dev_stress_temporal 6:
0.0748173 -0.00588609 -0.000440381 ...
```

The significance of the various columns in this output stream is the same as that described above for the steady-state case.

Fig. 4.5 plots the temporal evolution of the stress on the suspension.



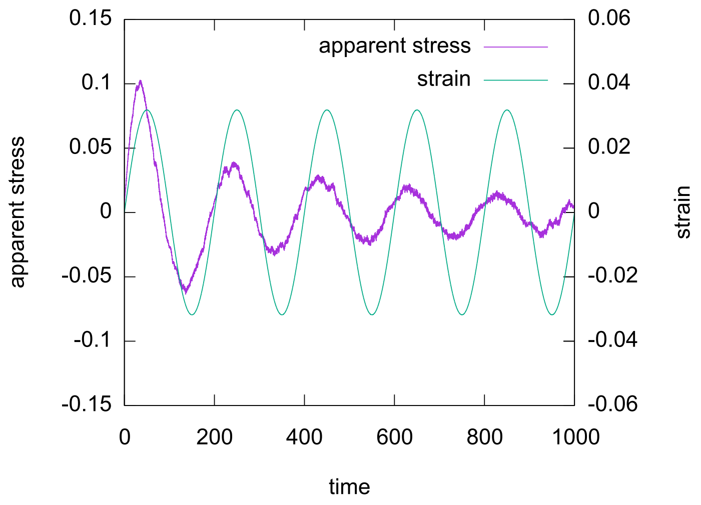


Figure 4.5: Temporal evolution of the stress for a suspension in oscillatory shear flow. The x and y axes represent columns 1 and 11 in the output stream. Upper plot: particle volume fraction $\varphi = 0.307$, temperature $k_BT = 0.1$. Lower plot: particle volume fraction $\varphi = 0.552$, temperature $k_BT = 5.0$.

Chapter 5

Simulating charged particles dispersed in electrolytic solutions

5.1 Electrophoresis of charged colloidal particles

When colloidal particles are dispersed in water or another solvent with extremely high dielectric permittivity, ions are emitted from dissociable groups on the colloid surface, causing the particle surface to acquire a net electric charge. The emitted ions are electrostatically attracted to the particle surface, but also undergo diffusion due to thermal fluctuations, resulting in the formation of a cloud-like ionic atmosphere around the colloid known as an *electric double layer*. The properties of disperse colloid systems at thermal equilibrium are described by the Poisson-Boltzmann equation; linearizing this equation yields the Debye–Hückel approximation, from which valuable insights may be obtained. In contrast, in systems exhibiting electrophoresis—or any of the other related effects known collectively as *interfacial electrokinetic phenomena*—the behavior of particles and ion distributions is governed by a competition between hydrodynamic and electrostatic interactions. As a result of this competition, the ion distribution is unable to keep up with the motion of the particles, causing the electric double layer to distort from a spherically symmetric distribution and yield configurations that differ from the equilibrium states. To analyze such complicated situations theoretically, there is no choice but to introduce simple approximations. In fact, until recently, even numerical simulations were rarely, if ever, capable of reproducing these phenomena correctly.

In applications of the SP method to simulations of electrokinetic phenomena, the colloids are treated as particles—as in the other types of simulation—but the solvent and the ions are treated as a continuous medium and described, after a coarse-graining procedure, by a density field. To make these calculations feasible, we have designed a computational formalism that uses SP functions to ensure smooth interfaces and self-consistently solves a coupled set of equations for three interacting degrees of freedom: the equations of motion for the colloids, the advection-diffusion equation for the ion distribution, and the Navier-Stokes equations for the solvent velocity field [5, 6, 24, 25]. This approach enabled the world's first successful quantitative simulations of electrophoresis [25]. In the following section we describe the basic equations considered internally by KAPSEL and briefly survey the underlying theoretical background.

5.2 Basic equations

We begin by surveying the basic equations of electrohydrodynamics needed for these simulations, following Refs. [6, 25]. Consider N spherical colloidal particles of radius a dispersed in an electrolytic solvent. We assume that the dielectric permittivity ϵ of the solvent is spatially uniform, including in the interior of the colloids. In the SP method, the interface between a particle and the solvent is represented by a smooth function $\phi(\boldsymbol{r}) \in [0, 1]$ of finite width ξ. Working on a fixed orthogonal lattice, the SP function takes the values $\phi = 1$ in the interior of colloids and $\phi = 0$ in solvent regions outside of colloids; the interface region is defined by the condition $0 < \phi < 1$. Other approaches that use $\phi(\boldsymbol{r})$ in an orthogonal lattice include the methods of Tanaka-Araki [26] and Kajishima et al. [27]. The use of interface functions yields an overwhelming advantage in computational efficiency compared to finite-element methods or other approaches based on non-structured lattices. We assume that colloid surfaces are uniformly charged, with a total charge of Ze per particle. In typical calculations involving continuous media, particle surface charge distributions are represented by delta functions; in finite-element and similar approaches this requires the use of an appropriate boundary-conforming computational mesh, significantly degrading computational efficiency. In the SP method, by contrast, the particle surface charge distribution $eq(\boldsymbol{r})$ may itself be represented as a smooth function involving first derivatives of the SP function ϕ:

$$eq(\boldsymbol{r}) = \frac{Ze|\nabla\phi(\boldsymbol{r})|}{4\pi a^2}. \tag{5.1}$$

Just as the SP function $\phi(\boldsymbol{r})$ is designed to approach a step function as $\xi \to 0$, the charge density tends to a delta function as $\xi \to 0$.

5.2.1 The advection-diffusion equation

The density distribution C_α for an ion of species α with charge z_α may be defined at all points in the computational domain by writing

$$C_\alpha(\boldsymbol{r}, t) = (1 - \phi(\boldsymbol{r}, t))C^*_\alpha(\boldsymbol{r}, t). \tag{5.2}$$

The factor $(1 - \phi)$ selects only regions in which ions are present. $C^*_\alpha(\boldsymbol{r}, t)$ is an auxiliary variable, introduced for computational convenience, and defined to be smooth everywhere in the computational domain. In the interior of colloids ($\phi = 1$), the quantity C^*_α has no physical meaning. The total charge distribution, including the colloid surface charge, is

$$\rho_e(\boldsymbol{r}) = e\sum_\alpha z_\alpha C_\alpha(\boldsymbol{r}) + eq(\boldsymbol{r}). \tag{5.3}$$

We constrain the initial distribution by requiring the electro-neutrality condition to be satisfied, $\int \rho_e \mathrm{d}\boldsymbol{r} = 0$. The temporal evolution of the auxiliary ion density C^*_α is governed by the advection-diffusion equation:

$$\partial_t C^*_\alpha = -\nabla \cdot C^*_\alpha \boldsymbol{v} + \Gamma_\alpha \nabla \cdot (C^*_\alpha \nabla\mu_\alpha). \tag{5.4}$$

The two terms on the right-hand side here describe advection by the solvent velocity field $\boldsymbol{v}$ and diffusion due to the gradient of the chemical potential μ_α, respectively. Because C^*_α obeys the advection-diffusion equation, the quantity $\int \mathrm{d}\boldsymbol{r}C^*_\alpha$ is conserved.

To prevent ions from penetrating into the interior of colloids, we impose the constraint that the ion diffusion flow velocity at particle-solvent interfaces has a vanishing component in the direction normal to the interface, i.e., that $\boldsymbol{n} \cdot \nabla\mu_\alpha = 0$ [6, 25]. Here, $\boldsymbol{n}$ is the outward-pointing normal vector at the colloid surface; in terms of the SP function we may write $\boldsymbol{n} = -\nabla\phi/|\nabla\phi|$. Also, Γ_α is the Onsager transport coefficient for ions of species α, related to the ion's friction coefficient and diffusion coefficient according to $f_\alpha = 1/\Gamma_\alpha, D_\alpha = k_B T\Gamma_\alpha$. We take the ion chemical potential to be [28]

$$\mu_\alpha = k_B T \ln C^*_\alpha + z_\alpha e(\Psi - \boldsymbol{E} \cdot \boldsymbol{r}). \tag{5.5}$$

Here $\boldsymbol{E}$ represents an external electric field, and the electrostatic potential $\Psi(\boldsymbol{r})$ solves the Poisson equation:

$$\epsilon\nabla^2\Psi = -\rho_e. \tag{5.6}$$

In an equilibrium state, ions with this chemical potential obey the Poisson-Boltzmann distribution.

5.2.2 The Navier-Stokes equations

The solvent velocity field $\boldsymbol{u}$ is described by the Navier-Stokes equations for an incompressible flow ($\nabla \cdot \boldsymbol{u} = 0$):

$$\rho(\partial_t + \boldsymbol{u} \cdot \nabla)\boldsymbol{u} = -\nabla p + \eta\nabla^2\boldsymbol{u} - \rho_e(\nabla\Psi - \boldsymbol{E}) + \phi\boldsymbol{f}_p \tag{5.7}$$

where ρ, η, and p are the density, viscosity coefficient, and pressure of the solvent. Note that the fluid experiences an electrostatic force $-\rho_e(\nabla\Psi - \boldsymbol{E})$. The external-force term $\phi\boldsymbol{f}_p$ represents a constraint force that preserves the rigidity of the particles. In other words, we consider adhesive boundary conditions at particle surfaces due to $\phi\boldsymbol{f}_p$. Further discussion of this point may be found in Refs. [5, 6].

5.2.3 Equations of motion

If the ith colloidal particle has mass M_p, its time-dependent position and velocity $\{\boldsymbol{R}_i, \boldsymbol{V}_i\}$ evolve in time according to the equations of motion:

$$\dot{\boldsymbol{R}}_i = \boldsymbol{V}_i, \tag{5.8}$$

$$M_p\dot{\boldsymbol{V}}_i = \boldsymbol{F}^H_i + \boldsymbol{F}^{other}_i \tag{5.9}$$

Here $\boldsymbol{F}^H_i$ is the force received from the fluid, representing the balance of momentum between solid and fluid [6]. Also, $\boldsymbol{F}^{other}_i$ represents forces due to interparticle potentials such as Lennard-Jones potentials. Although omitted here, similar considerations hold for the rotational motion of particles [6]. This completes our survey of the basic equations governing the simulations.

5.3 Properties of electric double layers

The basic tool for quantitative analysis of electric double-layer structures is the Poisson-Boltzmann equation.

5.3.1 The Poisson-Boltzmann equations

In the absence of external electric fields ($\boldsymbol{E} = 0$), the equilibrium ion distribution may be obtained from equation (5.5). Assuming the chemical potential is uniform (μ_α = const) the equilibrium ion distribution takes the form

$$C^*_\alpha(\boldsymbol{r}) = \bar{C}_\alpha \exp\left(-\frac{z_\alpha e\Psi(\boldsymbol{r})}{k_B T}\right). \tag{5.10}$$

This is the Boltzmann distribution in the presence of an electrostatic potential Ψ. Combining this with equation (5.6) yields the Poisson-Boltzmann equation.

5.3.2 The Debye-Hückel approximation and the Debye screening length

Consider a single spherical colloidal particle in a $z : z$ symmetric electrolytic solvent. The Poisson-Boltzmann equation reads [15, 29]

$$\nabla^2\Psi(\boldsymbol{r}) = \frac{2ze\bar{C}}{\epsilon}\sinh\left(\frac{ze\Psi(\boldsymbol{r})}{k_B T}\right). \tag{5.11}$$

At infinite distance from the system we impose the boundary conditions $\Psi|_{\boldsymbol{r}=\infty} = 0$ and $C^*|_{\boldsymbol{r}=\infty} = \bar{C}$. The boundary condition at the particle surface is such that the surface charge density remain constant and equal to $\sigma e = Ze/4\pi a^2$, thus requiring

$$\nabla\Psi|_{\text{surface}} = -\frac{\sigma e}{\epsilon}. \tag{5.12}$$

Assuming $ze\Psi/k_B T \ll 1$ and linearizing equation (5.11) yields the *Debye–Hückel* approximation:

$$\nabla^2\Psi(\boldsymbol{r}) = \frac{2z^2e^2\bar{C}}{k_B T\epsilon}\Psi = \kappa^2\Psi, \tag{5.13}$$

where

$$\kappa^{-1} = \frac{1}{\sqrt{8\pi\lambda_B z^2\bar{C}}} \tag{5.14}$$

is a constant with dimensions of length known as the *Debye screening length.* Here $\lambda_B = e^2/4\pi k_B T\epsilon$ is the *Bjerrum length.* For a typical electrolyte we have

$$\kappa^{-1} = \frac{1}{\sqrt{4\pi\lambda_B\sum_\alpha z_\alpha^2\bar{C}_\alpha}}. \tag{5.15}$$

As the system is spherically symmetric, we need only consider variations with respect to the radial coordinate $r = |\boldsymbol{r}|$, yielding the simplified equation:

$$\frac{d^2\Psi}{dr^2} + \frac{2}{r}\frac{d\Psi}{dr} = \kappa^2\Psi \tag{5.16}$$

whose general solution takes the form of a Yukawa potential:

$$\Psi(r) = \Psi_0\frac{a}{r}\exp[-\kappa(r-a)]. \tag{5.17}$$

The electrostatic force due to a charged colloid is screened beyond distances on the order of κ^{-1}. The Debye screening length κ^{-1} may be thought of as the distance at which the Coulomb force attracting oppositely-charged ions to the particle surface is balanced by the tendency of thermal diffusive motion to prevent ions from localizing. Thus κ^{-1} may be interpreted as the thickness of the electric double layer. At high temperatures the thermal energy $k_B T$ is large and κ^{-1} is large. On the other hand, the larger the ion strength ($\sum_\alpha z_\alpha^2\bar{C}_\alpha/2$) the stronger the effect of ionic screening and the shorter the screening length κ^{-1}. As an example, consider the bulk salt concentration $\bar{C}$ in a $z : z$ symmetric electrolytic solvent. Assuming the medium is water at 25 °C, the Bjerrum length is $\lambda_B = 0.72$nm, and inserting actual numerical values in equation (5.14) yields

$$\kappa^{-1} = \frac{0.3}{z\sqrt{\bar{C}}}\ (\text{nm}). \tag{5.18}$$

For $z = 1$, $\bar{C} = 0.1$M we have $\kappa^{-1} = 1$nm, while for $z = 1$, $\bar{C} = 0.001$M we have $\kappa^{-1} = 10$nm.

The surface potential $\Psi(r = a) = \Psi_0$ is determined by the boundary condition $d\Psi/dr(r = a) = -\sigma e/\epsilon$. Ψ is related to the surface charge density σe by

$$\sigma e = \epsilon\kappa\Psi_0(1 + (\kappa a)^{-1}). \tag{5.19}$$

Thus the surface potential increases linearly with increasing surface charge. However, the Debye-Hückel approximation ceases to be valid in regions of high surface potential, and in practice the surface potential eventually tends to increase by smaller and smaller increments in response to increasing surface charge. This behavior may be derived by solving the Poisson-Boltzmann equation, but approximate formulae for 1:1 electrolytic solvents have also been proposed by Loeb-Overbeek-Wiersema [15] and by Ohshima-Healy-White [29, 30].

5.4 Principles of electrophoresis

In the presence of an external electric field E, a colloidal particle with charge Ze feels an electrostatic force ZeE and begins to move. The acceleration of the particle due to this electrostatic force is opposed by a viscous drag force from the fluid, and the balance between these two effects results in constant-velocity motion of the particle at steady-state velocity V. Assuming the viscous drag force is the Stokes drag $6\pi\eta aV$ for a spherical particle of radius a, the force balance reads

$$ZeE = 6\pi\eta aV \tag{5.20}$$

yielding an electrophoretic mobility of

$$\frac{V}{E} = \frac{Ze}{6\pi\eta a}. \tag{5.21}$$

However, actual electrophoretic mobilities are smaller than this estimate. This is because equation (5.21) does not account for the effect of electrostatic forces on the ionic atmosphere surrounding the colloids, which induce motion in that atmosphere. Colloids moving through the fluid drag their ionic atmosphere with them, reducing their velocities correspondingly. Also, deviations from spherical symmetry in electric double layers give rise to additional forces that must be taken into account; this tends to make theoretical analysis prohibitively complicated, and instead a number of simplified models for analyzing electrophoretic mobilities have been considered [15, 29].

5.4.1 Smoluchowski's equation

When the particle radius a is much larger than the electric double-layer thickness κ^{-1}, i.e. $\kappa a \gg 1$, we may apply Smoluchowski's equation. In this limit, the electric double layer is considered to be infinitesimally thin, so we ignore the curvature of the particle surface and model it as a planar slab. Suppose an external electric field E_x is applied parallel to the slab (in the x direction). In a particle-fixed coordinate system, in which we ride atop a particle and observe the motion of the surrounding fluid, the fluid velocity at infinite distance from the particle is $-V$; balancing the viscous drag force and the electrostatic force on the particle yields

$$\eta\frac{\partial^2 v_x}{\partial y^2} + \sum_\alpha eC_\alpha E_x = 0. \tag{5.22}$$

Poisson's equation relates the ion distribution to the second derivative of the electrostatic potential, i.e.

$$\eta\frac{\partial^2 v_x}{\partial y^2} = \epsilon\frac{\partial^2\Psi}{\partial y^2}E_x. \tag{5.23}$$

Integrating this expression, subject to the boundary conditions of vanishing velocity gradient, vanishing potential gradient, and vanishing potential at spatial infinity, we find

$$\eta[v_x(y) + V] - \epsilon E\Psi(y) = 0. \tag{5.24}$$

Because the velocity at the particle surface ($y = 0$) is 0, we have

$$\frac{V}{E} = \frac{\epsilon\zeta}{\eta}. \tag{5.25}$$

This is Smoluchowski's equation. The zeta potential ζ is conventionally defined as the potential at the slip surface, but we have here replaced this with the potential $\Phi(0)$ at the particle surface.

5.4.2 Hückel's equation

In the opposite limit of Smoluchowski's equation, we take the particle radius to be much smaller than the electric double-layer thickness, i.e., $\kappa a \ll 1$. In this limit we have simply a point charge of strength Ze, and Hückel's equation applies. In equation (5.21), we take the potential at the particle surface to be the Coulomb potential, i.e.,

$$\zeta = \frac{Ze}{4\pi\epsilon a} \tag{5.26}$$

whereupon

$$\frac{V}{E} = \frac{2}{3}\frac{\epsilon\zeta}{\eta}. \tag{5.27}$$

This is Hückel's equation.

5.4.3 Henry's equation and the O'Brien-White analysis

For general values of κa, a bridge between equations (5.25) and (5.27) is furnished by Henry's equation:

$$\frac{V}{E} = f(\kappa a)\frac{\epsilon\zeta}{\eta} \tag{5.28}$$

where the Henry coefficient $f(\kappa a)$ is defined by

$$\begin{aligned} f(\kappa a) &= 1 - 5\exp(\kappa a)E_7(\kappa a) + 2\exp(\kappa a)E_5(\kappa a) && (5.29)\\ &= \frac{2}{3} + \frac{(\kappa a)^2}{24} - \frac{5(\kappa a)^3}{72} - \frac{(\kappa a)^4}{144} + \frac{(\kappa a)^5}{144} + \left[\frac{(\kappa a)^4}{12} - \frac{(\kappa a)^6}{144}\right]\exp(\kappa a)E_1(\kappa a). && (5.30) \end{aligned}$$

Here $E_n(\kappa a)$ is the nth exponential integral. Smoluchowski's equation corresponds to $f = 1(\kappa a \to \infty)$ while Hückel's equation corresponds to $f = 2/3(\kappa a \to 0)$ (Fig. 5.1).

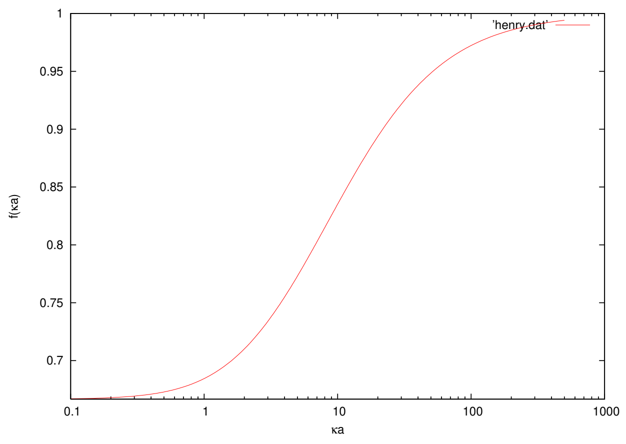


Figure 5.1: The Henry coefficient $f(\kappa a)$.

Henry's equation (5.28) depends linearly on the zeta potential and applies only for small values of this potential. For larger zeta potentials it becomes necessary to account for the impact of deformations in the electric double-layer potential (known as *relaxation effects*). O'Brien-White used numerical analysis to propose a relationship between the zeta potential and the electrophoretic mobility for arbitrary values of κa and ζ [31]. Later, an analytical formula valid for $\kappa a \geq 10$ was proposed by Ohshima-Healy-White [32].

5.5 Input UDF files

5.5.1 Fluid settings

Set `constitutive_eq` to `Electrolyte` to simulate charged-colloid motion or electrophoretic phenomena in the presence of external electric fields.[1] The settings that may be configured via input UDF files for this case are described below.

➤`constitutive_eq.Electrolyte`: Solvent and ion-distribution properties

`constitutive_eq.Electrolyte.DX`
Lattice spacing Δ, which defines the length unit.

`constitutive_eq.Electrolyte.RHO`
Solvent density.

`constitutive_eq.Electrolyte.ETA`
Solvent viscosity.

`constitutive_eq.Electrolyte.kBT`
Particle temperature.

`constitutive_eq.Electrolyte.alpha_v`
Correction term for thermal fluctuations (translational motion) of particles.

`constitutive_eq.Electrolyte.alpha_o`
Correction term for thermal fluctuations (rotational motion) of particles.

`constitutive_eq.Electrolyte.Dielectric_cst`
Dielectric permittivity of solvent.

`constitutive_eq.Electrolyte.Init_profile`
Set to `Uniform` or `Poisson_Boltzmann` to select how the initial ion distribution is chosen.[2]

`constitutive_eq.Electrolyte.Add_salt.type`
Selects the number of ion species. Allowed values:

`saltfree`	One ion species, with a charge opposite to that of the particle surface.
`salt`	Two ion species (one positive, one negative).

`constitutive_eq.Electrolyte.Add_salt.saltfree.Valency_counterion`
Counterion valency for `Add_salt = saltfree`.

`constitutive_eq.Electrolyte.Add_salt.saltfree.Onsager_coeff_counterion`
Counterion Onsager transport coefficient for `Add_salt = saltfree`.

`constitutive_eq.Electrolyte.Add_salt.salt.Valency_positive_ion`
Positive-ion valency for the case `Add_salt = salt`.

[1]Other possible settings for `constitutive_eq` include `Navier_Stokes` or `Shear_Navier_Stokes` to simulate the motion of colloidal particles in Newtonian fluids in the absence or presence of shear flow. With these choices, thermal fluctuations of charged particles will be neglected.

[2]Note that choosing `Poisson_Boltzmann` may be rather time-consuming, particularly for large numbers of particles.

`constitutive_eq.Electrolyte.Add_salt.salt.Valency_negative_ion`
Negative-ion valency for the case `Add_salt = salt`.

`constitutive_eq.Electrolyte.Add_salt.salt.Onsager_coeff_positive_ion`
Positive-ion Onsager transport coefficient for the case `Add_salt = salt`.

`constitutive_eq.Electrolyte.Add_salt.salt.Onsager_coeff_negative_ion`
Negative-ion Onsager transport coefficient for the case `Add_salt = salt`.

`constitutive_eq.Electrolyte.Add_salt.salt.Debye_length`
Debye screening length. If a value is specified, the salt concentration will be chosen to yield the specified Debye length.

`constitutive_eq.Electrolyte.Electric_field.type`
Set to `ON` or `OFF` to enable or disable an external electric field.

`constitutive_eq.Electrolyte.Electric_field.ON.type`
For the case `Electric_field.type=ON`, set to `DC` or `AC` to select a constant or oscillatory external electric field.

`constitutive_eq.Electrolyte.Electric_field.ON.DC.Ex`
x-directed electric field strength for the case `Electric_field.ON.type=DC`.

`constitutive_eq.Electrolyte.Electric_field.ON.DC.Ey`
y-directed electric field strength for the case `Electric_field.ON.type=DC`.

`constitutive_eq.Electrolyte.Electric_field.ON.DC.Ez`
z-directed electric field strength for the case `Electric_field.ON.type=DC`.

`constitutive_eq.Electrolyte.Electric_field.ON.AC.Ex`
x-directed electric field strength for the case `Electric_field.ON.type=AC`.

`constitutive_eq.Electrolyte.Electric_field.ON.AC.Ey`
y-directed electric field strength for the case `Electric_field.ON.type=AC`.

`constitutive_eq.Electrolyte.Electric_field.ON.AC.Ez`
z-directed electric field strength for the case `Electric_field.ON.type=AC`.

`constitutive_eq.Electrolyte.Electric_field.ON.AC.Frequency`
Frequency of oscillatory external electric field.

5.5.2 Object (particle) settings

The `object_type.type` setting specifies the type of particle. The possible values are `spherical_particle` for spherical particles, or `chain` for flexible chains.

5.5.3 Choice of length and time units

The length unit is given by the lattice spacing Δ. The time unit τ_0 is determined by the fluid density ρ, the fluid viscosity coefficient η, and the lattice spacing Δ according to $\tau_0 = \rho\Delta^2/\eta$.

- For the Navier-Stokes equations, we choose a system of units in which $\rho = \eta = \Delta = 1$.

- Assuming the input UDF file specifies RHO= A, ETA= B, and DX= C, the maximum wavenumber k_{max} may be determined from C and the upper bound on the timestep is determined from the momentum diffusion time in the form $T_{step} = (A/B)/k_{max}^2$. The actual simulation timestep is related to this bound through a multiplicative scale factor, i.e., $\Delta t =$ `factor` $\times T_{steo}$.
- Consider the correspondence between physical quantities in simulations and in reality. A grid spacing of 1 µm is reasonable for the length scales we wish to consider. Assuming we use water ($\eta = 1 \times 10^{-3}$ Pa s, $\rho = 1 \times 10^3$ kg m^{-3}) as a solvent, the time unit is then $\tau_0 = 1 \times 10^{-6}$ s.
- For the case `constitutive_eq=Electrolyte`, T_{step} is set to the smallest value between $(A/B)k_{max}^2$ and $(1/k_B T\Gamma_\alpha)/k_{max}^2$, and information regarding which of these two quantities is chosen is written to the console (standard error) during KAPSEL execution.

5.6 Computational examples

5.6.1 Single-particle electrophoresis

The input UDF file for this example is `colloid_1.udf` in the folder `Examples/01/`. Check that the output-file folders `./avs_ch/` and `./avs_ch/avs/` have been created on your system, then run the following command.

```
$ ../../kapsel -Icolloid_1.udf -Ooutput.udf -Ddefine.udf -Rrestart.udf
```

For this example, we choose a $64 \times 64 \times 64$ computational mesh. We consider a 1:1 electrolytic solvent (viscosity coefficient $\eta = 1$, density $\rho = 1$) and set the following values for the simulation parameters: particle radius $a = 5$, interface thickness $\xi = 2$, charge $Z = -100$, electric field strength $E_x = 0.1$, Debye screening length κ^{-1}=10. Fig. 5.2 shows a snapshot from this simulation of single-particle electrophoresis, while Fig. 5.3 plots the time evolution of the particle's velocity; the velocity approaches a constant value determined by the balance of electrostatic and fluid drag forces on the particle. The plot in Fig. 5.3 is generated by the script `plot.py`.

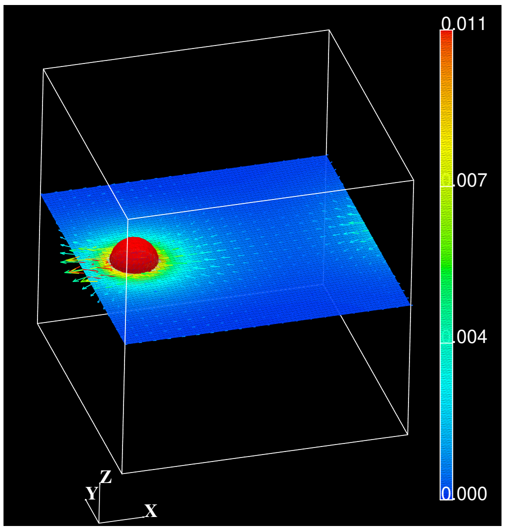


Figure 5.2: Snapshot from simulation of a colloidal particle (red) exhibiting electrophoresis under the influence of a positive x-directed external electric field E. Dark/light colors indicate the charge density distribution; arrows indicate the solvent velocity field.

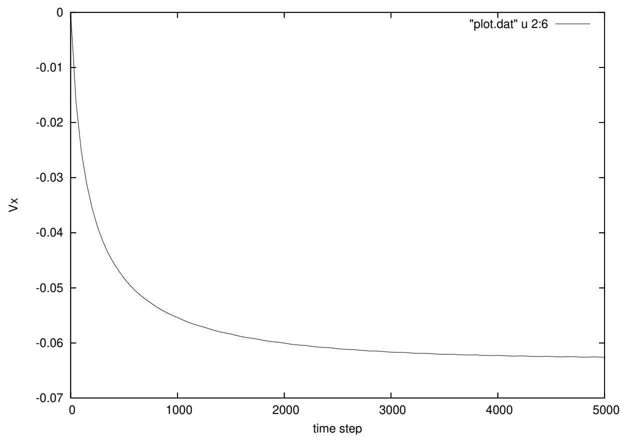


Figure 5.3: Time evolution of electrophoretic velocity of a single colloidal particle.

5.6.2 Many-particle electrophoresis

The input UDF file for this example is `colloid_32.udf` in the `Examples/01/` folder. The simulation is similar to that for the previous example, but now with 32 particles. Initial particle positions are chosen at random, and other parameter values are chosen as in the previous example.

Check that the output-file folders `./avs_ch/` and `./avs_ch/avs/` have been created on your system, and then run the following command.

```
$ ../../kapsel -Icolloid_32.udf -Ooutput.udf -Ddefine.udf -Rrestart.udf
```

Fig. 5.4 shows the results of this simulation of many-particle electrophoresis, while Fig. 5.5 plots the time evolution of the average electrophoretic velocity for $N_P = 16, 32, 64$ particles; the average velocity decreases as the number of particles increases.

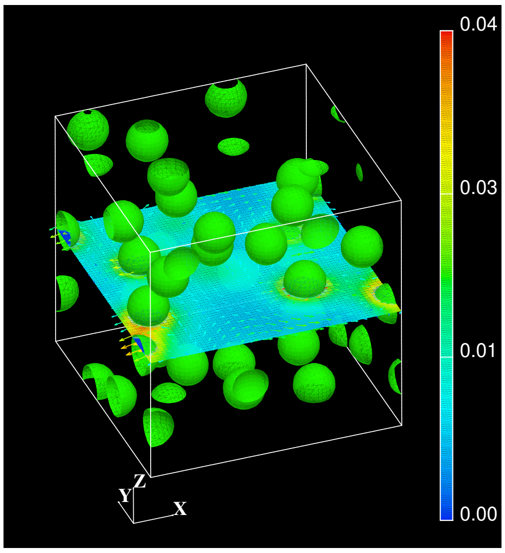


Figure 5.4: Snapshot from a simulation of colloidal particles (green) exhibiting electrophoresis under the influence of a positive x-directed external electric field E. Dark/light colors indicate the charge density distribution; arrows indicate the solvent velocity field.

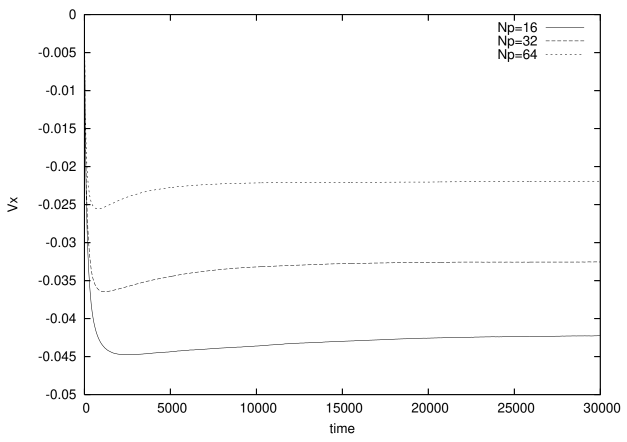


Figure 5.5: Time evolution of the average electrophoretic velocity for multiparticle systems. The upper, middle, and lower plots are for $N_p = 64, 32, 16$ particles.

5.6.3 Electrophoresis in a mixed particle system including both positively and negatively charged particles

The input UDF file for this example is `colloid_p32m32.udf` in the `Examples/01/` folder. This file describes a simulation of electrophoresis in a mixed particle system containing 32 positively-charged colloids and 32 negatively-charged colloids. Other simulation parameters are configured as in the single-particle example above. Check that the output-file folders `./avs_ch/` and `./avs_ch/avs/` have been created on your system, and then run the following command.

```
$ ../../kapsel -Icolloid_p32m32.udf -Ooutput.udf -Ddefine.udf -Rrestart.udf
```

Fig. 5.6 shows a snapshot from this simulation of electrophoresis in a system of 32 positively-charged and 32 negatively-charged colloids.

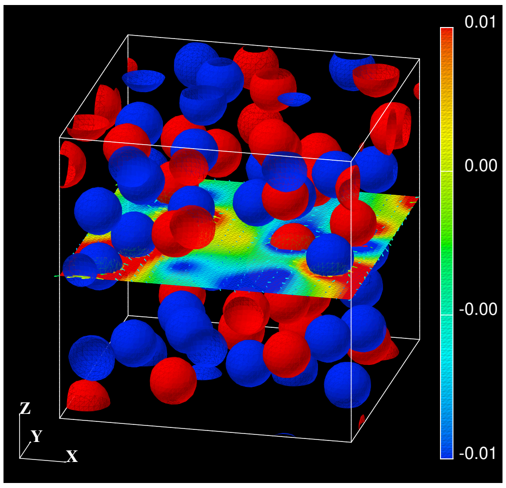


Figure 5.6: Snapshot from a simulation of a mixed particle system containing both positively-charged (red) and negatively-charged (blue) colloids exhibiting electrophoresis under the influence of a positive x-directed external electric field E. Positively and negatively charged colloids move in the positive and negative x directions, respectively.

The input file `colloid_p5km5k.udf` describes a similar simulation, but with the numbers of positively and negatively charged colloids increased to 5,000 each. Other parameter settings remain as in the single-particle example above.

```
$ ../../kapsel -Icolloid_p5km5k.udf -Ooutput.udf -Ddefine.udf -Rrestart.udf
```

Fig. 5.7 shows a snapshot from this simulation of electrophoresis in a system of 5,000 positively-charged and 5,000 negatively-charged colloids.

5.6.4 Using AVS/Express for visualization

Set `output.AVS` to `ON` to enable generation of AVS output files. This will create an output file named `avs_charge.v`, which may be opened in the AVS/Express software package[3] for visualization. In the `Read_field` module, specify the field information file `data.fld` to enable visualization of particles, solvent velocity fields, and charge distributions. The plots in Figs. 5.2, 5.4, and 5.6 above were generated in this way using `avs_charge.v` files.

5.6.5 Using Gourmet for visualization

Launch Gourmet, load the output file `output.udf`, use Gourmet's Python panel to `Load` the Python script `show_field.py` (distributed with KAPSEL), and select `Run` to open a new graphics window in which particles, solvent velocity fields, and charge density distributions may be visualized.[4] Click the **Play** button at the bottom of the screen to start an

[3]http://www.avs.com/

[4]Gourmet_2003 and earlier versions contain bugs that prevent this script from running properly. If you are affected by this problem, use the alternative visualization script `particleshow.py`. In this case the solvent velocity field and the charge density distribution will not be displayed; only the particle visualization will be generated.

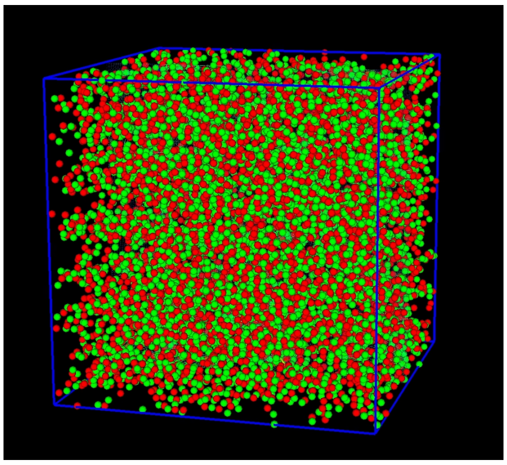


Figure 5.7: Snapshot from a simulation of a mixed particle system containing both positively-charged (red) and negatively-charged (green) colloids exhibiting electrophoresis under the influence of a positive x-directed external electric field E. Positively and negatively charged colloids move in the positive and negative x directions, respectively.

animation. Particle visualizations are based on particle coordinate data written to output UDF files in the form of data records, so the property `output.UDF` must be set to `ON`. Also, visualizations of solvent velocity fields and charge density distributions are produced by reading AVS-format binary output files, so `output.AVS` must be set to `ON` and `output.AVS.ON.FileType` must be set to `Binary`. If you wish only to visualize the motion of particles, use `particleshow.py` with `output.UDF` set to `ON`.

5.6.6 Using gnuplot to plot data

The gnuplot package[5] may be used to plot time-series data for particle coordinates and velocities written to output UDF files in the form of data records [33]. Launch Gourmet, load the output file `output.udf`, use Gourmet's Python panel to `Load` the Python script `plot.py` (distributed with KAPSEL), and select `Run` to create a new plot-generation sheet from particle coordinates and velocities written as data records. To plot the data, in the "View" box select `Table` instead of `Tree`; select `GraphSheet` from the list of variables to browse available `GraphSheet` data. Then, in Gourmet's Plot panel select `Make` followed by `Plot` to generate a plot like that in Fig. 5.3. If the plot is difficult to interpret due to an excess of linestyles, simply use the editor in the Plot panel to edit the `plot` command, removing any unneeded components. Further details on how to invoke gnuplot from Gourmet may be found in Section 3 of Ref. [33].

[5]http://www.gnuplot.info/

Chapter 6

Simulating particles dispersed in two-component phase-separated fluids

6.1 Adding particles to two-component phase-separated fluids

Fluids consisting of mixtures of two immiscible components exhibit a rich variety of phase-separation structures, and techniques for controlling these systems are extremely important for industrial applications. For example, the two-component systems known as *emulsions*—in which one component exists in the form of droplets, with sizes on the order of microns or smaller, dispersed throughout a second fluid component—exploit special properties of low-mutual-affinity fluids in coexistence and have applications spanning a wide range of fields, from foodstuffs, to cosmetics, to petroleum recovery and beyond. However, emulsions are intrinsically thermodynamically unstable, and the stabilization and structural control of these systems pose significant practical challenges. In the early 1900s, Ramsden[34] and Pickering[35] reported that emulsions could be stabilized by the addition of insoluble microparticles. This is due to the tendency of microparticles to adhere to interfaces between two fluid components, where they play a role similar to that of surfactants. Ever since the work of Ramsden and Pickering, an active school of research has studied various microparticle species—including silica, carbon black, and metal complexes—as emulsion additives [36, 37]. More recently, the use of microparticles to stabilize fluid interfaces in two-component phase-separated mixtures with *bicontinuous* structures has led to the creation of new compound materials known as *bijels* that have been a focus of intense interest [38, 39]. Experience with microparticle additives for phase-separation structural control has shown that, in addition to fluid-fluid interactions, fluid-particle interactions and the affinity between particles and fluid interfaces also play key roles in these systems. KAPSEL is capable of taking all of these interactions into account when simulating particle additives to two-component phase-separated fluids, enabling studies of the impact of particle additives on phase-separation structures and rheological properties.

For studies of particles in *single*-component fluids, the use of KAPSEL's Smoothed Profile (SP) method is relatively straightforward: there is only one type of interface to consider, namely the particle-fluid interface. For two-component fluids, on the other hand, we encounter a new type of interface—the fluid-fluid interface—that must be represented as well. Thus, for simulations of particles dispersed in two-component phase-separated fluids we introduce a new interface function ψ that distinguishes between the two fluid components that meet at fluid interfaces, and we seek to describe how ψ evolves in time (Fig. 6.1). KAPSEL handles solid particles immersed in two-component phase-separated fluids by combining the SP method with the well-known *model H* [40] approach to descriptive modeling of the dynamics of two-component phase-separated fluids. Section 6.2 describes how the SP method is extended to handle two-component phase-separated fluids, while Section 6.3 describes various user-configurable simulation parameters used by KAPSEL. Finally, as computational examples exploiting these new KAPSEL capabilities, Section 6.4 describes and presents results of simulations of solid particles dispersed in two-component phase-separated fluids and Pickering emulsions.

6.2 Theoretical background and basic equations

6.2.1 Basic equations for two-component phase-separated fluids

In the model H theory [40], the equations of motion for a two-component phase-separated fluid are the Navier-Stokes (NS) equations augmented by a chemical-potential-gradient term:

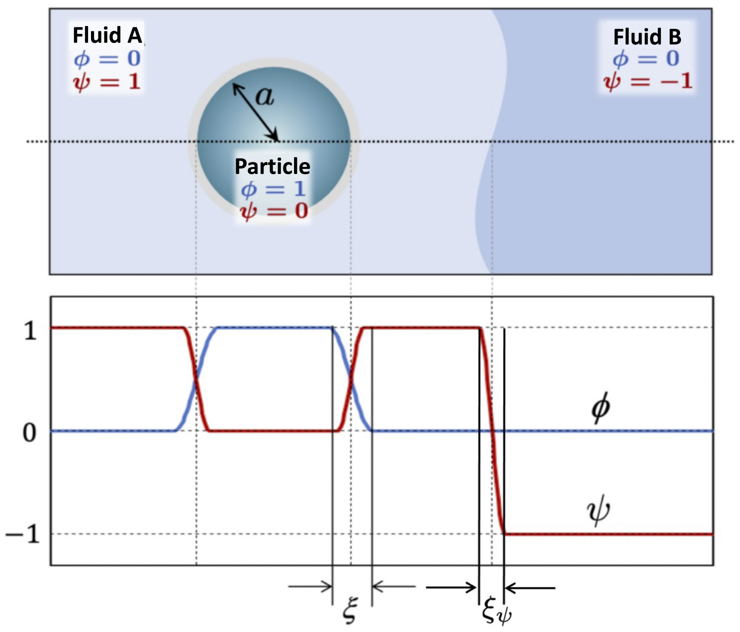


Figure 6.1: Interface functions used to describe particles dispersed in a two-component phase-separated liquid. In addition to the SP function ϕ separating particles from the surrounding fluid, we introduce an interface function ψ that separates the two fluid components from each other. The thicknesses of the particle-fluid and fluid-fluid interfaces are denoted respectively by ξ and ξ_ψ. In this figure, ψ takes values of ± 1 in bulk fluid regions, but this depends on the settings chosen for the double-well potential $f(\psi)$.

$$\frac{\partial \boldsymbol{u}}{\partial t} + (\boldsymbol{u} \cdot \boldsymbol{\nabla})\boldsymbol{u} = -\frac{1}{\rho}\boldsymbol{\nabla} p + \frac{1}{\rho}\boldsymbol{\nabla} \cdot \eta(\boldsymbol{\nabla u} + \boldsymbol{\nabla u}^{\mathrm{t}}) - \frac{\psi}{\rho}\boldsymbol{\nabla}\mu_\psi - \frac{\phi}{\rho}\boldsymbol{\nabla}\mu_\phi + \phi \boldsymbol{f}_{\mathrm{p}}. \tag{6.1}$$

Here the viscosity η is treated as a position-dependent physical quantity defined throughout the two-component fluid. The superscript t indicates matrix transposition, and $\mu_\psi = \delta F/\delta\psi$ and $\mu_\phi = \delta F/\delta\phi$ are the locally-defined chemical potentials with respect to ψ and ϕ, defined as functional derivatives of the Ginzburg-Landau (GL) free energy F. The free energy F in KAPSEL is discussed in further detail below.

Next, the time evolution of the fluid-fluid interface function ψ is described by the Cahn-Hilliard (CH) equation:

$$\frac{\partial \psi}{\partial t} = -\nabla \cdot \boldsymbol{J} \tag{6.2}$$

where $\boldsymbol{J}$ is the mass flux, expressed here in a form that includes an advection term:

$$\boldsymbol{J} = \psi \boldsymbol{u} - \kappa \nabla \mu_\psi. \tag{6.3}$$

Also, κ is the mobility of the two-component fluid. Note that the symbol κ, used in this section to denote mobility, has no relation to the Debye length κ^{-1} introduced in Section 5.

We next discuss the GL free energy F. In KAPSEL we consider phase separation in two-component fluids interacting with particles, and thus F takes the form

$$F = \int d\boldsymbol{r} \left[f(\psi) + \frac{\alpha}{2}(\boldsymbol{\nabla}\psi)^2 + w\xi\psi|\boldsymbol{\nabla}\phi|^2 + d(\psi - \bar{\psi})^2\phi + z\xi_\psi\phi(\boldsymbol{\nabla}\psi)^2 \right]. \tag{6.4}$$

Here the function $f(\psi)$ is a double-well potential governing the local dynamics of the fluid-fluid interface function ψ. This potential may be chosen to be either a Landau potential:

$$f(\psi) = \frac{a}{4}\psi^4 - \frac{b}{2}\psi^2 \tag{6.5}$$

or a Flory-Huggins (FH) potential:

$$f(\psi) = k_{\mathrm{B}}T_0 \left[\frac{\psi}{N_{\mathrm{A}}} \ln \psi + \frac{(1-\psi)}{N_{\mathrm{B}}} \ln(1-\psi) + \chi\psi(1-\psi) \right]. \tag{6.6}$$

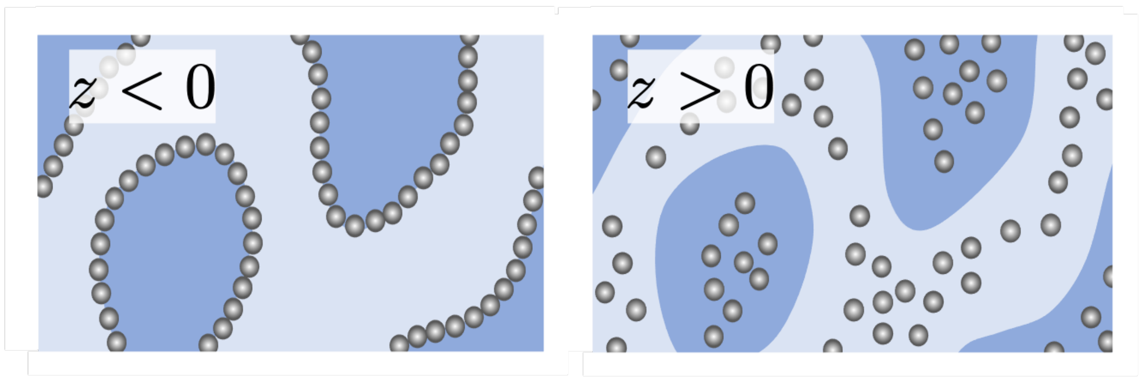


Figure 6.2: Effect of the z parameter on the particle dispersion.

The quantities a and b in equation (6.5) are adjustable parameters used to tune the relative strengths of the two terms in the potential. Also, in equation (6.6) the quantity $k_B T_0$ is the reference unit of thermal energy, N_A and N_B are the degrees of polymerization of the fluid components, and χ is the Flory interaction parameter.

The second term in equation (6.4) represents interfacial energy associated with fluid-fluid interfaces. The third term describes particle-fluid interactions. The fourth term is included to ensure, through the relation $\bar{\psi} = \frac{a\psi_0^3-(b-2d)\psi_0}{2d}$, that ψ takes the arbitrary fixed value ψ_0 at points inside the particles.[1] Finally, the fifth term describes interactions between particles and fluid-fluid interfaces; here z is a coefficient and ξ_ψ is the fluid-fluid interface thickness.[2] Figure 6.2 illustrates how the z parameter affects particle dispersion. For $z < 0$, the free energy is minimized when particles exist at fluid-fluid interfaces, so particles migrate toward those interfaces in that case. In contrast, for $z > 0$ particles distance themselves from fluid-fluid interfaces. The fluid-fluid interface thickness ξ_ψ depends on the function $f(\psi)$; for a Landau double-well potential it is given by [41].

$$\xi_\psi = \sqrt{\frac{\alpha}{2b}}. \tag{6.7}$$

In simulating particles dispersed in two-component phase-separated fluids we successively solve the equations of motion for the fluid and for the particles, equations (6.1) and (3.4)-(3.6), together with equation (6.2) for the time evolution of the fluid-fluid interface function ψ, which distinguishes the two fluid components. KAPSEL solves these equations on a semi-staggered Arakawa B lattice [42] using the marker-and-cell (MAC) method [43, 44]. Combining this approach with the techniques of Section 4 allows for simulations of particles dispersed in two-component phase-separated fluids in the presence of shear flows. Solution methods are discussed in detail in Appendix D.

6.2.2 Viscosity of suspensions

For a suspension of particles in a single-component fluid, the shear stress and viscosity are given by equations (4.5) and (4.6) On the other hand, when particles are added to a two-component phase-separated fluid, there are additional surface-tension terms associated with fluid-fluid interfaces, whose contributions must be taken into account. The xy component of the stress tensor due to such interfaces is given by [45]

$$\sigma_{\text{int}}^{xy} = \frac{1}{V}\int d\boldsymbol{r}\left(-\alpha\frac{\partial\psi}{\partial x}\frac{\partial\psi}{\partial y}\right). \tag{6.8}$$

Consequently, the apparent viscosity of the system is obtained by adding the contribution of equation (6.8) to equation (4.6). Section 6.4.2 presents a computational example involving a viscosity calculation in the presence of steady-state shear flow.

6.3 Input UDF files

6.3.1 Fluid settings

KAPSEL simulations of particles dispersed in two-component phase-separated fluids may be performed (a) with no imposed flow field, or (b) in the presence of a shear flow. Set `constitutive_eq` to `Navier_Stokes_Cahn_Hilliard_FDM`

[1]Setting $d = 0$ must be avoided. To ignore this term, you should set such as $d = 0.0001$.
[2]This term has no effect if $f(\psi)$ is chosen to be of the Flory-Huggins form.

for case (a) or to `Shear_NS_LE_CH_FDM` for case (b).

Settings for simulations with no imposed flow field

➤`constitutive_eq.Navier_Stokes_Cahn_Hilliard_FDM`: Properties of two-component phase-separated fluids

`constitutive_eq.Navier_Stokes_Cahn_Hilliard_FDM.NS_solver.type`
: Specifies the method used to solve the NS equations. The allowed values are `explicit_scheme` or `implicit_scheme`.[3]

`constitutive_eq.Navier_Stokes_Cahn_Hilliard_FDM.NS_solver.implicit_scheme.tolerance`
: Convergence criterion for implicit solution of NS equations.

`constitutive_eq.Navier_Stokes_Cahn_Hilliard_FDM.NS_solver.implicit_scheme.maximum_iteration`
: Maximum number of iterations for implicit solution of NS equations.

`constitutive_eq.Navier_Stokes_Cahn_Hilliard_FDM.NS_solver.implicit_scheme.viscosity_change`
: Set to `ON` to enable simulations of two-component phase-separated fluids (A/B) with different viscosities.

`constitutive_eq.Navier_Stokes_Cahn_Hilliard_FDM.NS_solver.implicit_scheme.ON.ETA_A`
: Viscosity of fluid component A.

`constitutive_eq.Navier_Stokes_Cahn_Hilliard_FDM.NS_solver.implicit_scheme.ON.ETA_B`
: Viscosity of fluid component B.

`constitutive_eq.Navier_Stokes_Cahn_Hilliard_FDM.CH_solver.type`
: Specifies the method used to solve the CH equation. The allowed values are `explicit_scheme` or `implicit_scheme`.[4]

`constitutive_eq.Navier_Stokes_Cahn_Hilliard_FDM.CH_solver.implicit_scheme.tolerance`
: Convergence criterion for implicit solution of CH equation.

`constitutive_eq.Navier_Stokes_Cahn_Hilliard_FDM.CH_solver.implicit_scheme.maximum_iteration`
: Maximum number of iterations for implicit solution of CH equations.

`constitutive_eq.Navier_Stokes_Cahn_Hilliard_FDM.DX`
: Lattice spacing Δ, which defines the length unit.

`constitutive_eq.Navier_Stokes_Cahn_Hilliard_FDM.RHO`
: Fluid density ρ.

[3]KAPSEL implements the MAC implicit method [46] as an implicit solver for the NS equations. The use of an implicit solver increases the computation time, but allows simulations of two-component phase-separated fluids with different viscosities. Solution methods are discussed in detail in Appendix D.1. When `implicit_scheme` is selected, BiCGSTAB [47] is chosen as the default iterative solver for simultaneous systems of linear equations, and several new user-configurable simulation settings, discussed in the table above, become available. When using iterative solvers for implicit methods, the solution is considered converged when the residual falls below the specified `tolerance` value. If the number of iterations exceeds `maximum_iteration`, KAPSEL concludes that no solution was obtained and the simulation is terminated. Iterative solvers other than BiCGSTAB may be used by linking with the Lis library [48]. This is discussed in more detail in Appendix E.

[4]In addition to the explicit solver, methods implemented by KAPSEL for solving the CH equation include a backward-difference method for time derivatives and a quasi-implicit solver [49] based on values extrapolated by the Adams-Bashforth method for nonlinear potential terms. Solution methods are discussed in detail in Appendix D. For the quasi-implicit CH solver, iterative solvers other than BiCGSTAB may be used by linking with the Lis library [48]. This is discussed in more detail in Appendix E.

`constitutive_eq.Navier_Stokes_Cahn_Hilliard_FDM.ETA`
Fluid viscosity η.[5]

`constitutive_eq.Navier_Stokes_Cahn_Hilliard_FDM.kBT`
Particle temperature.

`constitutive_eq.Navier_Stokes_Cahn_Hilliard_FDM.alpha_v`
Correction term for particle temperature associated with translational motion.

`constitutive_eq.Navier_Stokes_Cahn_Hilliard_FDM.alpha_o`
Correction term for particle temperature associated with rotational motion.

`constitutive_eq.Navier_Stokes_Cahn_Hilliard_FDM.Potential.type`
Set to `Landau` or `Flory_Huggins` to select the double-well potential for a two-component phase-separated fluid.

`constitutive_eq.Navier_Stokes_Cahn_Hilliard_FDM.Potential.Landau.composition_ratio`
Fluid composition ratio for Landau potential.

`constitutive_eq.Navier_Stokes_Cahn_Hilliard_FDM.Potential.Landau.initial_fluctuation`
Initial concentration fluctuation for Landau potential.

`constitutive_eq.Navier_Stokes_Cahn_Hilliard_FDM.Potential.Landau.a`
a parameter for 4th-order term in Landau potential.

`constitutive_eq.Navier_Stokes_Cahn_Hilliard_FDM.Potential.Landau.b`
b parameter for 2nd-order term in Landau potential.

`constitutive_eq.Navier_Stokes_Cahn_Hilliard_FDM.Potential.Landau.d`
d parameter used to force the composition of fluid regions inside particles to an arbitrary value ψ_0 when using a Landau potential.

`constitutive_eq.Navier_Stokes_Cahn_Hilliard_FDM.Potential.Landau.w`
Particle-fluid interaction parameter w for Landau potential.

`constitutive_eq.Navier_Stokes_Cahn_Hilliard_FDM.Potential.Landau.z`
z parameter relevant for interactions between particles and fluid-fluid interfaces when using a Landau potential.[6]

`constitutive_eq.Navier_Stokes_Cahn_Hilliard_FDM.Potential.Landau.psi_0`
Arbitrary value ψ_0 for ψ function at points inside particles for Landau potential.

`constitutive_eq.Navier_Stokes_Cahn_Hilliard_FDM.Potential.Landau.alpha`
Fluid-fluid interfacial energy parameter α for Landau potential.

`constitutive_eq.Navier_Stokes_Cahn_Hilliard_FDM.Potential.Landau.kappa`
Mobility κ for Landau potential.

[5]The simulation timestep is chosen based on the viscosity value specified here. Caution: When `viscosity_change` is set to `ON`, the viscosities of the two fluid components do not affect the timestep. Consequently, specifying values for `ETA_A` or `ETA_B` that are significantly larger than the value set here for `ETA` may degrade the accuracy of the simulations or lead to computational instabilities. This is discussed in more detail in Section 6.3.3.

[6]Particle interactions with fluid-fluid interfaces are only active when using a Landau potential.

`constitutive_eq.Navier_Stokes_Cahn_Hilliard_FDM.Potential.Flory_Huggins.composition_ratio`
Fluid composition ratio for FH potential.

`constitutive_eq.Navier_Stokes_Cahn_Hilliard_FDM.Potential.Flory_Huggins.initial_fluctuation`
Initial concentration fluctuation for FH potential.

`constitutive_eq.Navier_Stokes_Cahn_Hilliard_FDM.Potential.Flory_Huggins.na`
Degree of polymerization N_A for fluid component A.

`constitutive_eq.Navier_Stokes_Cahn_Hilliard_FDM.Potential.Flory_Huggins.nb`
Degree of polymerization N_B for fluid component B.

`constitutive_eq.Navier_Stokes_Cahn_Hilliard_FDM.Potential.Flory_Huggins.chi`
Flory interaction parameter χ.

`constitutive_eq.Navier_Stokes_Cahn_Hilliard_FDM.Potential.Flory_Huggins.d`
d parameter used to force the composition of fluid regions inside particles to an arbitrary value ψ_0 when using an FH potential.

`constitutive_eq.Navier_Stokes_Cahn_Hilliard_FDM.Potential.Flory_Huggins.w`
Particle-fluid interaction parameter w for FH potential.

`constitutive_eq.Navier_Stokes_Cahn_Hilliard_FDM.Potential.Flory_Huggins.z`
z parameter relevant for interactions between particles and fluid-fluid interfaces when using FH potential.[7]

`constitutive_eq.Navier_Stokes_Cahn_Hilliard_FDM.Potential.Flory_Huggins.psi_0`
Arbitrary value ψ_0 for ψ function at points inside particles for FH potential.

`constitutive_eq.Navier_Stokes_Cahn_Hilliard_FDM.Potential.Flory_Huggins.alpha`
Fluid-fluid interfacial energy parameter α for FH potential.

`constitutive_eq.Navier_Stokes_Cahn_Hilliard_FDM.Potential.Flory_Huggins.kappa`
Mobility κ for FH potential.

`constitutive_eq.Navier_Stokes_Cahn_Hilliard_FDM.Wall_Potential.type`
Set to `ON` to add planar walls to simulations.

`constitutive_eq.Navier_Stokes_Cahn_Hilliard_FDM.Wall_Potential.ON.w`
Parameter determining affinity of fluid for planar walls.

`constitutive_eq.Navier_Stokes_Cahn_Hilliard_FDM.Wall_Potential.ON.psi_0.magnitude`
Parameter determining magnitude of affinity of fluid for planar walls.

`constitutive_eq.Navier_Stokes_Cahn_Hilliard_FDM.Wall_Potential.ON.psi_0.profile`
Set to `uniform` for uniform fluid affinity on planar walls; otherwise set to `user_specify` and configure manually.

[7]Particle interactions with fluid-fluid interfaces are only active when using a Landau potential.

`constitutive_eq.Navier_Stokes_Cahn_Hilliard_FDM.Wall_Potential.ON.psi_0.user_specify.PSI0[][]`
Manually specify a value for the affinity between fluid and planar walls.

`constitutive_eq.Navier_Stokes_Cahn_Hilliard_FDM.Wall_Potential.ON.DRYING.type`
Set to `ON` to simulate drying.

`constitutive_eq.Navier_Stokes_Cahn_Hilliard_FDM.Wall_Potential.ON.DRYING.ON.psi_dry`
Parameter determining rapidity of drying.

Settings for simulations under shear flow

➤`Shear_NS_LE_CH_FDM`: Properties of two-component phase-separated fluids

`constitutive_eq.Shear_NS_LE_CH_FDM.NS_solver.type`
Specifies the method used to solve the NS equations. The allowed values are `explicit_scheme` or `implicit_scheme`.[8]

`constitutive_eq.Shear_NS_LE_CH_FDM.NS_solver.implicit_scheme.tolerance`
Convergence criterion for implicit solution of NS equations.

`constitutive_eq.Shear_NS_LE_CH_FDM.NS_solver.implicit_scheme.maximum_iteration`
Maximum number of iterations for implicit solution of NS equations.

`constitutive_eq.Shear_NS_LE_CH_FDM.NS_solver.implicit_scheme.viscosity_change`
Set to `ON` to enable simulations of two-component phase-separated fluids (A/B) with different viscosities.

`constitutive_eq.Shear_NS_LE_CH_FDM.NS_solver.implicit_scheme.ON.ETA_A`
Viscosity of fluid component A.

`constitutive_eq.Shear_NS_LE_CH_FDM.NS_solver.implicit_scheme.ON.ETA_B`
Viscosity of fluid component B.

`constitutive_eq.Shear_NS_LE_CH_FDM.CH_solver.type`
Specifies the method used to solve the CH equation. The allowed values are `explicit_scheme` or `implicit_scheme`.[9]

`constitutive_eq.Shear_NS_LE_CH_FDM.CH_solver.implicit_scheme.tolerance`
Convergence criterion for implicit solution of CH equations.

`constitutive_eq.Shear_NS_LE_CH_FDM.CH_solver.implicit_scheme.maximum_iteration`
Maximum number of iterations for implicit solution of CH equations.

`constitutive_eq.Shear_NS_LE_CH_FDM.DX`
Lattice spacing Δ, which defines the length unit.

`constitutive_eq.Shear_NS_LE_CH_FDM.RHO`
Fluid density.

[8] As is true for simulations with no imposed flow field, the MAC implicit method [46] is implemented as an implicit solver for the NS equations.
[9] As is true for simulations with no imposed flow field, a quasi-implicit solver for the CH equation is implemented in addition to the explicit solver. [49].

`constitutive_eq.Shear_NS_LE_CH_FDM.ETA`
: Fluid viscosity.[10]

`constitutive_eq.Shear_NS_LE_CH_FDM.kBT`
: Particle temperature.

`constitutive_eq.Shear_NS_LE_CH_FDM.alpha_v`
: Correction term for particle temperature associated with translational motion.

`constitutive_eq.Shear_NS_LE_CH_FDM.alpha_o`
: Correction term for particle temperature associated with rotational motion.

`constitutive_eq.Shear_NS_LE_CH_FDM.Potential.type`
: Set to `Landau` or `Flory_Huggins` to select the double-well potential for a two-component phase-separated fluid.

`constitutive_eq.Shear_NS_LE_CH_FDM.Potential.Landau.composition_ratio`
: Fluid composition ratio for Landau potential.

`constitutive_eq.Shear_NS_LE_CH_FDM.Potential.Landau.initial_fluctuation`
: Initial concentration fluctuation for Landau potential.

`constitutive_eq.Shear_NS_LE_CH_FDM.Potential.Landau.a`
: a parameter for 4th-order term in Landau potential.

`constitutive_eq.Shear_NS_LE_CH_FDM.Potential.Landau.b`
: b parameter for 2nd-order term in Landau potential.

`constitutive_eq.Shear_NS_LE_CH_FDM.Potential.Landau.d`
: d parameter used to force the composition of fluid regions inside particles to an arbitrary value ψ_0 when using a Landau potential.

`constitutive_eq.Shear_NS_LE_CH_FDM.Potential.Landau.w`
: Particle-fluid interaction parameter w for Landau potential.

`constitutive_eq.Shear_NS_LE_CH_FDM.Potential.Landau.z`
: z parameter relevant for interactions between particles and fluid-fluid interfaces when using a Landau potential.[11]

`constitutive_eq.Shear_NS_LE_CH_FDM.Potential.Landau.psi_0`
: Value for ψ function at points inside particles.

`constitutive_eq.Shear_NS_LE_CH_FDM.Potential.Landau.alpha`
: Fluid-fluid interfacial energy parameter α for Landau potential.

`constitutive_eq.Shear_NS_LE_CH_FDM.Potential.Landau.kappa`
: Mobility κ for Landau potential.

[10] The simulation timestep is chosen based on the viscosity value specified here. Caution: When `viscosity_change` is set to `ON`, the viscosities of the two fluid components do not affect the timestep. Consequently, specifying values for `ETA_A` or `ETA_B` that are significantly larger than the value set here for `ETA` may degrade the accuracy of simulations or lead to computational instabilities. This is discussed in more detail in Section 6.3.3.

[11] Particle interactions with fluid-fluid interfaces are only active when using a Landau potential.

`constitutive_eq.Shear_NS_LE_CH_FDM.Potential.Flory_Huggins.composition_ratio`
Fluid composition ratio for FH potential.

`constitutive_eq.Shear_NS_LE_CH_FDM.Potential.Flory_Huggins.initial_fluctuation`
Initial concentration fluctuation for FH potential.

`constitutive_eq.Shear_NS_LE_CH_FDM.Potential.Flory_Huggins.na`
Degree of polymerization N_{A} for fluid component A.

`constitutive_eq.Shear_NS_LE_CH_FDM.Potential.Flory_Huggins.nb`
Degree of polymerization N_{B} for fluid component B.

`constitutive_eq.Shear_NS_LE_CH_FDM.Potential.Flory_Huggins.chi`
Flory interaction parameter χ.

`constitutive_eq.Shear_NS_LE_CH_FDM.Potential.Flory_Huggins.d`
d parameter used to force the composition of fluid regions inside particles to an arbitrary value ψ_0 when using an FH potential.

`constitutive_eq.Shear_NS_LE_CH_FDM.Potential.Flory_Huggins.w`
Particle-fluid interaction parameter w for FH potential.

`constitutive_eq.Shear_NS_LE_CH_FDM.Potential.Flory_Huggins.z`
z parameter relevant for interactions between particles and fluid-fluid interfaces when using FH potential.[12]

`constitutive_eq.Shear_NS_LE_CH_FDM.Potential.Flory_Huggins.psi_0`
Value for ψ function at points inside particles for FH potential.

`constitutive_eq.Shear_NS_LE_CH_FDM.Potential.Flory_Huggins.alpha`
Fluid-fluid interfacial energy parameter α for FH potential.

`constitutive_eq.Shear_NS_LE_CH_FDM.Potential.Flory_Huggins.kappa`
Mobility κ for FH potential.

`constitutive_eq.Shear_NS_LE_CH_FDM.External_field.type`
Set to `DC` (steady-state shear flow) or `AC` (oscillatory shear flow).

`constitutive_eq.Shear_NS_LE_CH_FDM.External_field.DC.Shear_rate`
Shear velocity $\dot{\gamma}$.

`constitutive_eq.Shear_NS_LE_CH_FDM.External_field.AC.Shear_rate`
Amplitude $\dot{\gamma}_0$ of oscillatory shear velocity.

`constitutive_eq.Shear_NS_LE_CH_FDM.External_field.AC.Frequency`
Frequency ω of oscillatory shear velocity.

[12]Particle interactions with fluid-fluid interfaces are only active when using a Landau potential.

6.3.2 Object (particle) settings

The `object_type.type` setting specifies the type of particle. The possible values are `spherical_particle` for spherical particles, `chain` for flexible chains, or `rigid` for rigid bodies.

6.3.3 Choice of length and time units

The length unit is given by the lattice spacing Δ. The time unit τ_0 is determined by the fluid density ρ, the fluid viscosity coefficient η, and the lattice spacing Δ according to $\tau_0 = \rho\Delta^2/\eta$.

Note that, for simulations of two-component phase-separated fluids with different viscosities, the viscosity η used in computing the unit of time is the value specified for `constitutive_eq.*.ETA`. Note carefully that the viscosities of the individual fluid components, i.e. the values specified for `constitutive_eq.*.NS_solver.implicit_scheme.ON.ETA_A` and `constitutive_eq.*.NS_solver.implicit_`
`scheme.ON.ETA_B` are **not** referenced when determining the simulation timestep. (Here * stands for either `Navier_Stokes_Cahn_Hilli`
or `Shear_NS_LE_CH_`
`FDM`.)

- Assuming the input UDF file specifies RHO= A, ETA= B, and DX= C, the maximum wavenumber k_{max} may be determined from C and the upper bound on the timestep is determined from the momentum diffusion time in the form $T_{step} = (A/B)/k_{max}^2$. The actual simulation timestep is related to this bound through a multiplicative scale factor, i.e. $\Delta t = \texttt{factor} \times T_{step}$.

- Consider the correspondence between physical quantities in simulations and in reality. A grid spacing of 1 µm is reasonable for the length scales we wish to consider. Assuming we use water ($\eta = 1 \times 10^{-3}$ Pa s, $\rho = 1 \times 10^3$ kg m^{-3}) as a solvent, the time unit is then $\tau_0 = 1 \times 10^{-6}$ s.

6.4 Computational examples

We now present examples of simulations using the methods described in the preceding subsections to study systems of particles dispersed in two-component phase-separated fluids. Section 6.4.1 describes a brief simulation that provides an outline of the entire KAPSEL simulation process. This simulation involves particles dispersed in two-component phase-separated fluids with no imposed flow field. Section 6.4.2 describes a simulation of a two-component phase-separated fluid with dispersed particles under shear flow, in order to investigate rheological properties. Finally, Section 6.4.3 describes a simulation of a Pickering emulsion [35].

6.4.1 Motion of particles dispersed in a two-component phase-separated fluid

Running the simulation

The UDF files for the simulations of this section are in the Examples/11a/ folder. Use the following commands to enter this folder and run the simulation.

```
$ cd Examples/11a
$ ../../kapsel -Iinput.udf -Ooutput.udf -Ddefine.udf -Rrestart.udf
```

This simulation uses the Landau double-well potential of equation (6.5) to describe phase-separated two-component fluids. Parameter values are written to the standard output when the simulation begins:

```
##################################
# Landau potential is selected.
# Parameters
# Composition ratio   : 0
# Initial fluctuation : 0.05
# a      : 1
# b      : 1
# d      : 1
# w      : -1
# z      : 0
# psi0_p : 0
# alpha  : 1
# kappa  : 1
#
##################################
```

For this simulation, Composition ratio is set to 0 to indicate that the two fluid components are present in equal quantities in the system. Also, w is set to −1 to indicate that particles have greater affinity for one of the two components (component A). psi0_p is a parameter specifying the value of ψ in the particle interior.

If the simulation completes successfully, runtime information will be reported to the console:

```
#Simulation has ended!
#Total Running Time (s):      11.89
#                   (m):       0.20
#                   (h):       0.00
#Average Step Time  (s):       0.01
#                   (m):       0.00
#                   (h):       0.00
```

Computation times vary depending on runtime environment, but this simulation should complete in a few tens of seconds. Upon completion of the simulation, the following output files will have been written to the Examples/11a/ folder:

- output.udf
- restart.udf
- fluid_phase.xmf
- flux_field.xmf
- particle_coordinate.xmf
- particle_phase.xmf
- velocity_field.xmf

- flux_*.h5
- orderparam_*.h5
- particle_*.h5
- particle_data_*.h5
- velocity_*.h5

Here asterisks (*) in file names represent sequence numbers. `output.udf` is an output UDF file containing simulation results that may be read by GOURMET. (Methods for viewing data in GOURMET are discussed below.) Also, `restart.udf` is a restart UDF file that may be used to restart the simulation from an intermediate time. Files with extension `h5` and sequence numbers in the filename store simulation data at individual timesteps in the HDF5 binary data format. These files may be bundled for shared processing via eXtensible Data Model and Format (XDMF) control files, which are used to manage time-series simulation data. XDMF files have extension `xmf` and are used by KAPSEL to establish rules for working with groups of `h5` files, as follows:

`fluid_phase.xmf`:	Data on the function ψ that distinguishes the two fluid components (`orderparam_*.h5`)
`flux_field.xmf`:	Data on the mass flux $\boldsymbol{J}$, equation (6.3) (`flux_*.h5`)
`particle_coordinate.xmf`:	Particle coordinate data (`particle_data_*.h5`)
`particle_phase.xmf`:	Data on the function ϕ that distinguishes particles: (`particle_*.h5`)
`velocity_field.xmf`:	Data on the fluid velocity field $\boldsymbol{u}$ (`velocity_*.h5`)

XDMF files may be displayed using visualization software such as ParaView[13] or Mayavi[14], and the following section presents an example of ParaView visualization. Data stored in `.h5` files may also be viewed directly using the `h5dump` command-line tool, e.g.:

```
$ h5dump orderparam_0.h5
```

Visualizing simulation data

In this section we describe how to use GOURMET and ParaView to visualize simulation results.

Visualizing simulation data with GOURMET

Reading an output UDF file (`output.udf`) into GOURMET allows browsing of simulation parameters and visualization of simulation results at various output steps [Figure (6.3)(**a**)]. Also, the folder `Examples/11a/` contains a Python script named `gourmet_particle_field_show.py` that may be read into GOURMETto visualize the function ψ that distinguishes between the two fluid components. More specifically, using the Python panel at the bottom of the GOURMET screen to read and execute this Python script will open a new graphical window presenting an animated view of simulation results [Figure (6.3)(**b**)]. In this figure, particles are represented by white spheres and fluid interfaces are displayed as blue constant-value surfaces, but these and other display settings may be modified directly in the Python script.

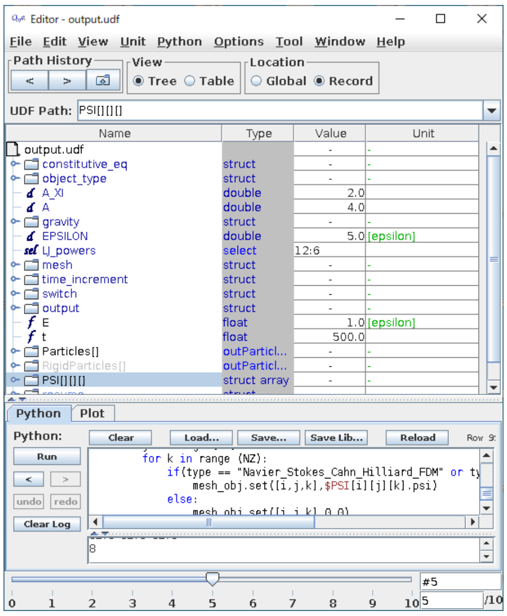
Visualizing simulation data with ParaView

XDMF files may be read into ParaView to visualize or analyze simulation results. The file `pv_sample.pvsm` in the `Examples/11a/` folder is a ParaView State file. Launch ParaView and select `Load State Files` from the `Files` menu at upper left. Read in the file `pv_sample.pvsm` to visualize the function ψ used to distinguish the two fluid components (Figure 6.4).[15] In this figure, particles are represented as white spheres; one fluid component is represented by red-colored (volume rendered) regions, and the other fluid component is displayed using translucent blue-colored regions. ParaView is not only useful for data visualization, but also offers powerful analysis capabilities. For further details, consult the ParaView documentation [50].

Writing Python scripts to analyze simulation data

Simulation data written to output files by KAPSEL may be separately read and analyzed via Python scripts. The `Examples/11a/` folder includes a Python script that computes the static structure factor $S(q)$ [51] for a phase-separation structure. Make sure the python packages `h5py`, `scipy`, and `matplotlib` are installed in your Python

[13]https://www.paraview.org/
[14]http://docs.enthought.com/mayavi/mayavi/
[15]Choose “Search files under specified directory” for ”Load State Data File Options”, and set appropriate path for ”Data Directory”.



View: [✓ Tree] Table — Location: Global [✓ Record]

UDF Path: PSI[][][]

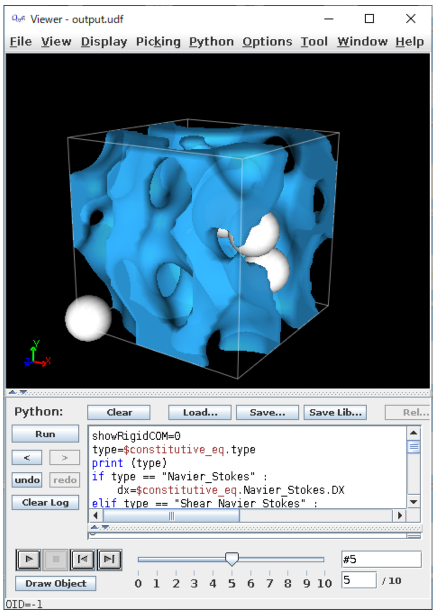
Name	Type	Value	Unit
output.udf		-	-
constitutive_eq	struct	-	-
object_type	struct	-	-
A_XI	double	2.0	
A	double	4.0	
gravity	struct	-	-
EPSILON	double	5.0	[epsilon]
LJ_powers	select	12:6	
mesh	struct	-	-
time_increment	struct	-	-
switch	struct	-	-
output	struct	-	-
E	float	1.0	[epsilon]
t	float	500.0	
Particles[]	outParticl...	-	-
RigidParticles[]	outParticl...	-	-
PSI[][][]	struct array	-	-

```
            for k in range (NZ):
                if(type == "Navier_Stokes_Cahn_Hilliard_FDM" or ty
                    mesh_obj.set([i,j,k],$PSI[i][j][k].psi)
                else:
                    mesh_obj.set([i,j,k],0.0)
```

```
8
```

#5
5 /10

(a) Listing parameter values recorded in output UDF files.

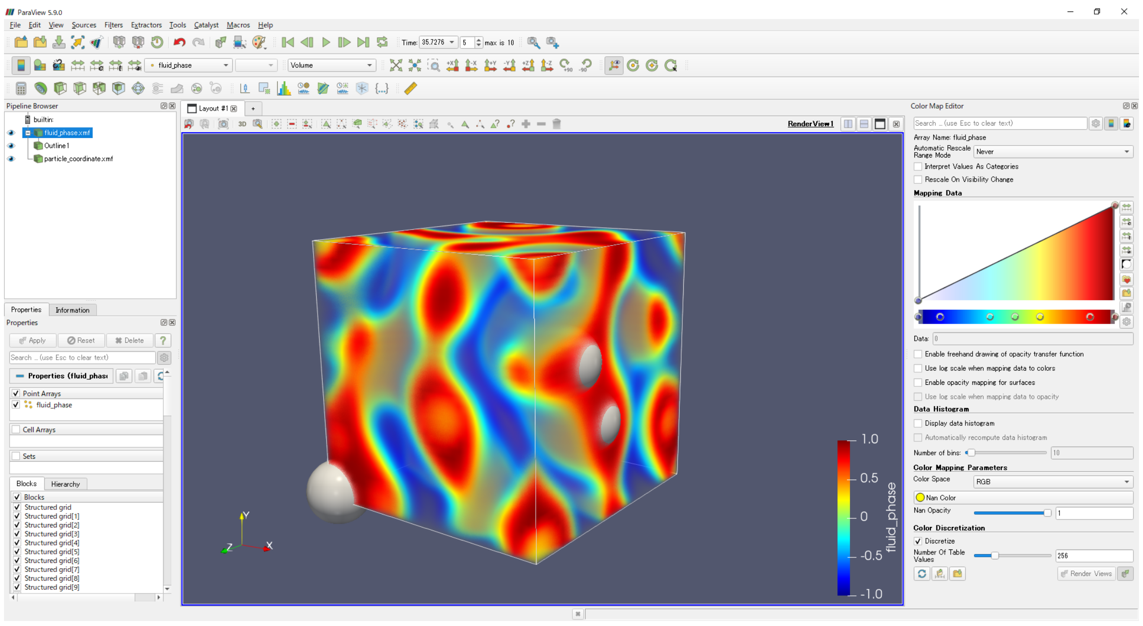


```
showRigidCOM=0
type=$constitutive_eq.type
print (type)
if type == "Navier_Stokes" :
    dx=$constitutive_eq.Navier_Stokes.DX
elif type == "Shear Navier Stokes" :
```

#5
5 / 10
OID=-1

(b) Using a Python script to visualize the function ψ that distinguishes between the two fluid components.

Figure 6.3: Visualizing simulation results in GOURMET.



Time: 35.7276 5 max is 10

Array Name: fluid_phase
Automatic Rescale Range Mode: Never
Color Space: RGB
Nan Opacity: 1
Number Of Table Values: 256

Figure 6.4: Visualizing simulation results in Paraview.

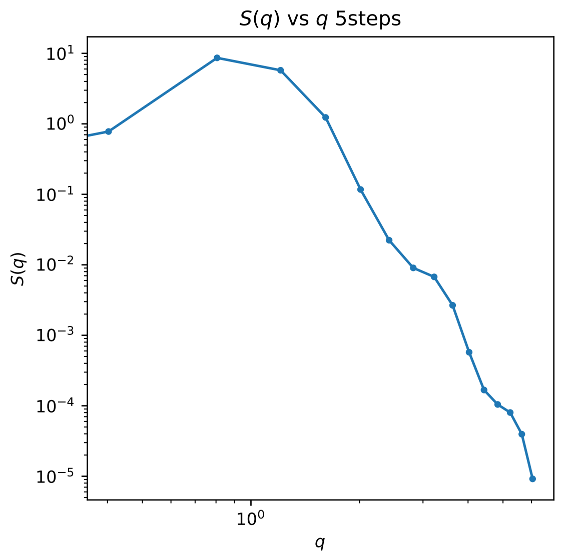


Figure 6.5: Plot of static structure factor $S(q)$ vs wavenumber q, illustrating the use of python scripting for data analysis. Results shown for the 5th simulation timestep.

environment, then execute the command below. This will produce a sequence of $S(q)$ graphs for various timesteps (Figure 6.5).

```
$ python sq_analysis.py
```

6.4.2 Motion of particles dispersed in a two-component phase-separated liquid under shear flow

The input file for this example is in the `Examples/11/` folder. It describes a simulation of a two-component phase-separated fluid, with dispersed particles, in the presence of a steady-state shear flow. Execute the following command to run the simulation:

```
$ cd Examples/11
$ ../../kapsel -Iinput.udf -Ooutput.udf -Ddefine.udf -Rrestart.udf
```

For this example, we consider 128 particles of radius $a = 4$ and interface thickness $\xi = 2$ added to a two-component phase-separated fluid in which a steady-state shear flow with shear velocity $\dot{\gamma} = 0.01$ has been imposed. We set the particle temperature to $k_B T = 1$ and use a $64 \times 64 \times 64$ computational mesh. The two-component fluid exhibits phase separation due to the Landau double-well potential of equation (6.5). Parameter values are written to standard output when the simulation begins:

```
#################################
# Landau potential is selected.
# Parameters
# Composition ratio   : 0
# Initial fluctuation : 0.5
# a      : 1
# b      : 1
# d      : 1
# w      : 0
# z      : 0
# psi0_p : 0
# alpha  : 1
# kappa  : 1
#
#################################
```

For simulations in the presence of shear flow, values for the following data items are written to standard error one line per timestep:

`1:time`	Elapsed time
`2:shear_rate`	Shear velocity
`3:degree_oblique`	Strain in oblique coordinate system [16]
`4:shear_strain_temporal`	Applied strain ($\dot{\gamma}t$)
`5:lj_dev_stress_temporal`	Shear stress due to interparticle potential
`6:shear_stress_temporal_old`	Shear stress due to particle-fluid interactions
`7:shear_stress_temporal_new`	Shear stress due to particle-fluid interactions [17]
`8:reynolds_stress`	Reynolds shear stress
`9:fluid_stress`	Shear stress due to bulk fluid
`10:interfacial_stress`	Shear stress due to fluid-fluid interface
`11:apparent_stress`	Shear stress on suspension [18]
`12:viscosity`	Viscosity of suspension [19]

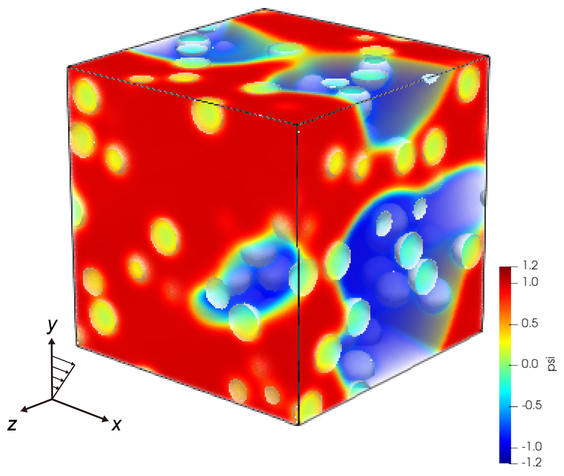


Figure 6.6: Snapshot of fluid interface function ψ captured on the final timestep of the simulation of Example 11 (particles in two-component fluid under shear flow).

Figure 6.6 is a snapshot of the fluid interface function ψ captured in the final timestep (step number 20, 000) of this simulation. In KAPSEL the shear flow is applied with the shear flow in the x direction, the shear gradient in the y direction, and the z direction axis as the neutral direction (the vortex direction). Because the parameter w governing the affinity of particles for the two-component fluid is set to 0, the particles are not preferentially located in one phase or the other, but are instead distributed throughout the full system.

Figure 6.7 plots the time evolution of the xy component of the shear stress, with the red curve indicating the shear stress on the suspension (`11:apparent_stress` in the standard-error output) and the blue curve indicating the shear stress due to the fluid-fluid interface (`10:interfacial_stress`). The results of this simulation indicate that the shear stress due to the fluid-fluid interface contributes significantly to the overall shear stress on the suspension. Also, whereas small fluctuations due to thermal motion of particles are visible in the curve for the overall shear stress on the suspension, no such fluctuations are seen in the curve for the fluid-fluid-interface contribution to the shear stress. This is because KAPSEL introduces thermal fluctuation of particles not through the Navier-Stokes equations for the fluid, but through the Langevin equation for the particles. A detailed discussion of particle temperatures in KAPSEL may be found in Ref. 3.1.1.

6.4.3 Pickering emulsions

The input files for this example are in the `Examples/13/` folder. They define a simulation of a *Pickering emulsion*, an emulsion stabilized by the addition of microparticles as first reported by Pickering [35]. Pickering emulsions are stabilized by the adsorption of microparticles at the fluid-fluid interface. In KAPSEL the affinity of particles for two-component fluid interfaces may be tuned via the z parameter in the 5th term of equation (6.4). This simulation considers 200 particles (radius $a = 4$ and interface thickness $\xi = 2$) dispersed in a two-component fluid, with the particles having neutral affinity ($w = 0$) with respect to the fluid, but nonzero affinity ($z = -0.6$) with respect to the fluid-fluid interfaces . The two-component fluid undergoes a phase separation induced by the Landau double-well potential in equation (6.5), with parameter values $a = b = d = \kappa = 1$ and $\alpha = 3$.

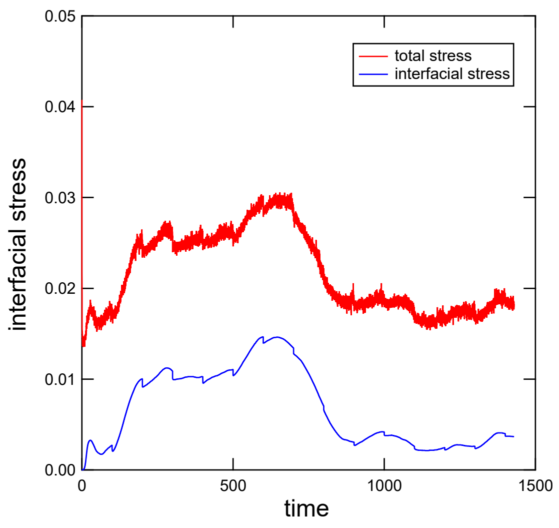


Figure 6.7: Time evolution of the shear stress (xy component) for the simulation of Example 11 (particles in two-component fluid under shear flow). Red: shear stress on the suspension (`11:apparent_stress`). Blue: shear stress due to the fluid-fluid interface (`10:interfacial_stress`).

The following command runs the simulation:

```
    $ cd Examples/13
    $ ../../kapsel -Iinput.udf -Ooutput.udf -Ddefine.udf -Rrestart.udf
```

In the simulation defined by the `input.udf` file in the `Examples/13` folder, particles have nonzero affinity ($z = -0.6$) for interfaces between the particle and the fluid-fluid interfaces.

Figure 6.8 shows a snapshot of the fluid interface function ψ captured in the final timestep (step $200,000$) for this simulation. As we see in this image, the nonzero affinity of particles for fluid-fluid interfaces causes particles to be attracted to these interfaces, where they function as surfactants.

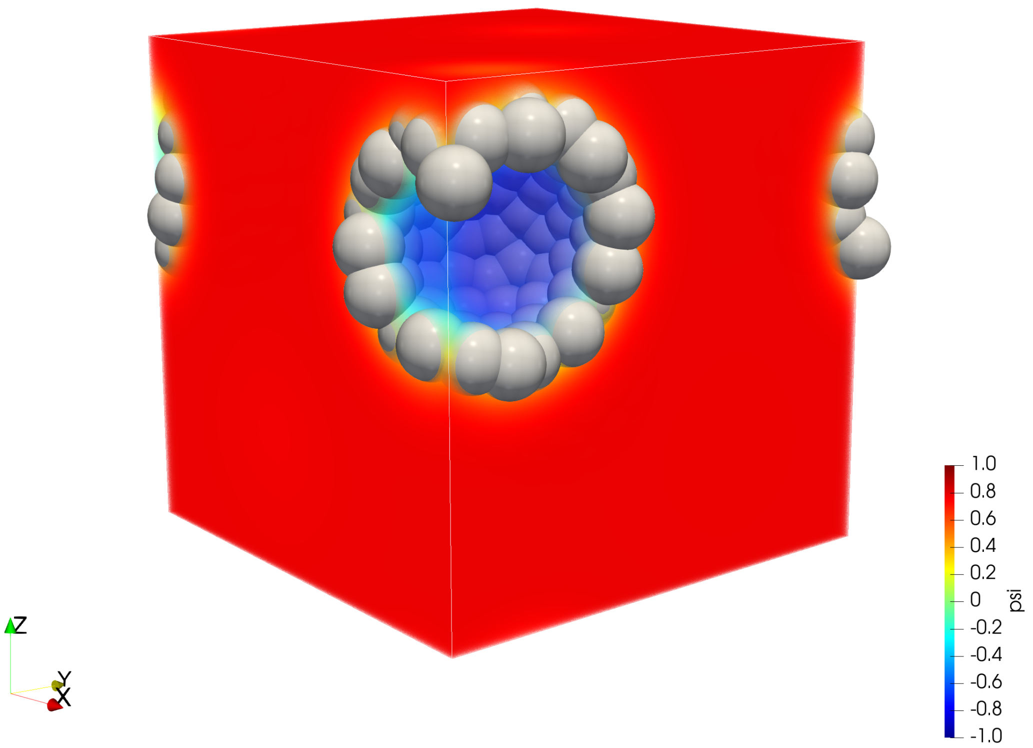


Figure 6.8: Snapshot of the fluid interface function ψ captured on the final timestep of the simulation of Example 13 (Pickering emulsion).

Chapter 7

Simulating microswimmers

7.1 Theoretical background and basic equations

Microswimmers—microscopic bodies moving autonomously through viscous fluids, with examples including spermatozoa, bacteria, and algae—are ubiquitous throughout biology and the life sciences. Microswimmers also constitute one of the most important types of *active matter*, soft matter exhibiting spontaneous self-propelled motion. Studies of active matter are extremely fruitful both for theoretical science—where active-matter research promises to make major contributions to the development and validation of non-equilibrium physics theories—and for practical applications such as smart-drug delivery and other cutting-edge medical innovations. However, the complex role of hydrodynamic interactions in governing the physics of microswimmer systems remains poorly understood, with efforts to elucidate the relevant mechanisms obstructed by difficult and longstanding challenges [52].

7.1.1 The squirmers model

KAPSEL uses a general-purpose model of microswimmers known as the *squirmer* model. This model, developed by Lighthill and Blake [53, 54], is commonly used to investigate hydrodynamic interactions among spheroidal microswimmers.

KAPSEL considers spherical swimmers of radius a. As an alternative to fine-grained modeling of the motion of cilia on particle surfaces, we impose specific boundary conditions at rigid-particle surfaces, which yield similar self-propulsion effects. We take $\{\tilde{\boldsymbol{e}}_i\}$ to be an orthogonal system of basis vectors defining a reference system attached to the particles and fix the direction of particle motion as $\tilde{\boldsymbol{e}}_3 = \tilde{\boldsymbol{e}}$. Points on the particle surface are described by their polar and azimuthal angles θ, λ, and we denote by $\{\boldsymbol{\theta}, \boldsymbol{\lambda}, \boldsymbol{\varrho}\}$, the unit tangent vectors in the angular direction and the unit vector in the radial direction (so that e.g. $\theta = \arccos \boldsymbol{\varrho} \cdot \tilde{\boldsymbol{e}}$). Assuming zero velocity in the radial direction, the slip boundary condition imposed on the squirmers reads [55]

$$\boldsymbol{u}^{sq} = \sum_{n=1}^{\infty} \frac{2}{n(n+1)} B_n P_n'(\cos\theta) \sin\theta \boldsymbol{\theta} + \sum_{n=1}^{\infty} C_n P_n'(\cos\theta) \sin\theta \boldsymbol{\lambda} \tag{7.1}$$

where P_n' is the derivative of the nth Legendre polynomial and B_n, C_n are the nth polar and azimuthal mode coefficients. Neglecting all azimuthal modes ($C_n = 0$) and polar modes of degree 2 or more ($B_n = 0$ where $n > 2$) yields a simple but important model capable of describing both *pushers* (swimmer particles for which propulsive forces arise behind the particle) and *pullers* (swimmers with propulsive forces arising in front of the particle). Denoting the ratio of the first two polar modes as $\alpha = B_2/B_1$, the surface velocity is given by

$$\boldsymbol{u}^{sq}(\theta) = B_1 \left(\sin\theta + \frac{\alpha}{2} \sin 2\theta \right) \boldsymbol{\theta}. \tag{7.2}$$

The parameter B_1 sets the steady-state velocity U of a single particle moving through the fluid according to $U = 2/3B_1$, while B_2 is the stresslet intensity. The ratio of mode coefficients α determines the type of swimmer: puller ($\alpha > 0$), pusher ($\alpha < 0$), or *neutral swimmer* ($\alpha = 0$). Figure 7.1 illustrates the motion of pushers and pullers, and the fluid flow accompanying their propulsion. The character of a swimmer's motion heavily influences the hydrodynamics of the fluid flow induced by its propulsion, which must also be taken into account when considering collective particle behavior.

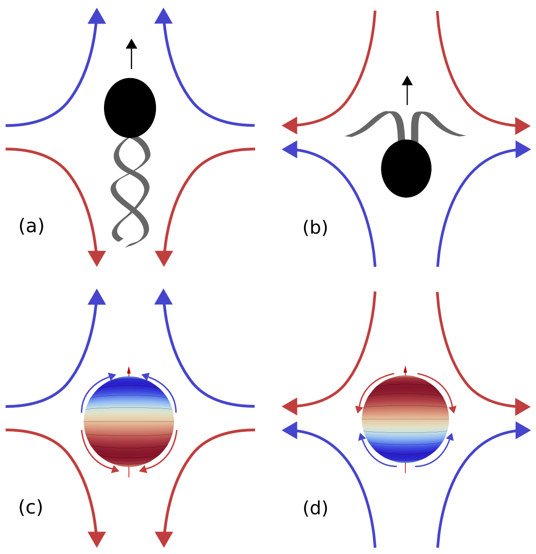


Figure 7.1: Schematic depiction of self-propulsion mechanisms and accompanying fluid flow for (a) pushers, (b) pullers. Describing these swimmers by Blake's squirming model—in which microscopic propulsion mechanisms are replaced by slip boundary conditions at particle surfaces—yields images (c) and (d). Reproduced from Soft Matter 9, 4923-4936[4], Copyright 2013, with permission from the Royal Society of Chemistry.

7.1.2 Basic equations for microswimmers

For SP-method simulations of the fluid mechanics of disperse microswimmer systems, we add a new constraint force term $\phi \boldsymbol{f}_{sq}$ to the continuous-medium equation (3.13) and apply slip boundary conditions at particle surfaces [4, 56]:

$$\rho(\partial_t + \boldsymbol{u} \cdot \boldsymbol{\nabla})\boldsymbol{u} = \boldsymbol{\nabla} \cdot \boldsymbol{\sigma} + \rho\phi \boldsymbol{f}_p + \rho\phi \boldsymbol{f}_{sq}. \tag{7.3}$$

To solve this equation, we add one new step to the usual procedure for SP-method calculations. After updating the advection and viscosity terms, and computing $\boldsymbol{u}^\star$, the term $\phi \boldsymbol{f}_p$ serves to exert rigid-body constraint conditions; at this juncture we compute a surface constraint force $\phi \boldsymbol{f}_{sq}$ and use it to update a velocity field $\boldsymbol{u}^{**}$ that satisfies the squirmer boundary conditions. To implement this strategy, we introduce a new SP function ϕ^{sq}, which takes nonzero values only in the interior of particle interfaces:

$$\phi_I^{sq} = (1 - \phi_I)\frac{|\boldsymbol{\nabla}\phi_I|}{\max\left(|\boldsymbol{\nabla}\phi_I|\right)}. \tag{7.4}$$

The new velocity field takes the form

$$\boldsymbol{u}^{**} = \boldsymbol{u}^* + \left[\int_{t_n}^{t_n+h} \mathrm{d}s\phi \boldsymbol{f}_{sq}\right] \tag{7.5}$$

$$\begin{aligned}\left[\int_{t_n}^{t_n+h} \mathrm{d}s\phi \boldsymbol{f}_{sq}\right] &= \sum_I^N \phi_I^{sq}\left(\boldsymbol{V}_I^\dagger + \boldsymbol{\Omega}_I^\dagger \times \boldsymbol{r}_I + \boldsymbol{u}_I^{sq} - \boldsymbol{u}^*\right) \\ &+ \sum_I^N \phi_I\left(\delta \boldsymbol{V}_I + \delta\boldsymbol{\Omega}_i \times \boldsymbol{r}_I\right) - \frac{h}{\rho}\boldsymbol{\nabla} p_{sq}\end{aligned} \tag{7.6}$$

The first term on the RHS here is a term for actually fixing the slip boundary conditions, while the second term ensures local momentum conservation (specifically, that squirmers pushing/pulling on the fluid at their boundaries experience a counterforce). The third term ensures that the fluid velocity field ultimately remains incompressible. Because the updated particle velocities are not yet known at this point, we solve equation iteratively, with slip boundary conditions applied for $\boldsymbol{V}_I^\dagger$ ($\boldsymbol{\Omega}_I^\dagger$). We compute hydrodynamic forces and torques as described previously (using $\boldsymbol{u}^{**}$ instead of $\boldsymbol{u}^*$), use the results to compute particle velocities, and iterate until the particle velocities $\boldsymbol{V}_I^{n=1}$ for the following time step agree with the velocity $\boldsymbol{V}_I^\dagger$ needed to impose slip boundary conditions. For the motion of a single swimmer, we have validated the efficacy of this computational procedure and confirmed that simulated particle velocities and fluid velocities are in good agreement with theoretical predictions [4].

7.2 Input UDF files

7.2.1 Fluid settings

The following choices are available for `constitutive_eq`.

`Navier_Stokes`:	Newtonian fluid
`Shear_Navier_Stokes`:	Newtonian fluid with zigzag shear flow
`Navier_Stokes_FDM`:	Newtonian fluid
`Navier_Stokes_Cahn_Hilliard_FDM`:	Two-component phase-separated fluid

7.2.2 Configuring objects (particles)

The only available option for `object_type.type` is `spherical_particle`.

7.2.3 Configuring objects (planar walls)

`switch.wall.type` may be set to `NONE` or `FLAT`.

7.3 Computational examples

7.3.1 Motion of microswimmers with periodic boundary conditions

The input UDF files for this example are `squirm_single_a+2.udf` and `squirm_phi0.1_a+2.udf` in the `Examples/09/` folder. The former of these describes a simulation with just a single particle.

```
$ ../../kapsel -Isquirm_single_a+2.udf -Ooutput.udf -Ddefine.udf -Rrestart.udf
```

In this example we use a $64 \times 64 \times 64$ computational mesh with the the following parameter values: number of particles $N_p = 1$, particle diameter $D = 10$, and interface thickness $\xi = 2$, yielding a particle volume fraction of $\varphi = 0.002$. The velocity of microswimmer motion is $V = 2/3B_1 = 0.01$ and the character of the motion is chosen by specifying $B_2 = 2$.

The second file, `squirm_phi0.1_a+2.udf`, describes a similar simulation but with multiple particles:

```
$ ../../kapsel -Isquirm_phi0.1_a+2.udf -Ooutput.udf -Ddefine.udf -Rrestart.udf
```

In this case we have $N_p = 50$ particles of diameter $D = 10$ and interface thickness $\xi = 2$, yielding a particle volume fraction of $\varphi = 0.1$. Other parameters are set as in the single-particle simulation. Fig. 7.2 is a snapshot of the 50-swimmer simulation.

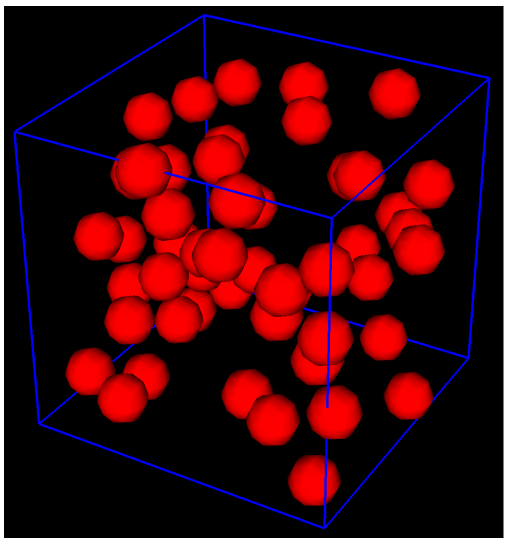


Figure 7.2: Snapshot from simulation of 50 microswimmers.

7.3.2 Motion of microswimmers confined between two parallel slabs

The input UDF file for this example is `squirm_wall.udf` in the `Examples/09/` folder. This file describes a simulation in which microswimmers are confined to the region between two parallel slabs.

```
$ ../../kapsel -Isquirm_wall.udf -Ooutput.udf -Ddefine.udf -Rrestart.udf
```

For this example (Fig. 7.3) we use a $64 \times 256 \times 64$ computational mesh. The thickness of the planar walls in the vertical direction (y axis) is 4. We specify $N_p = 4266$ particles with particle diameter $D = 4$ and interface thickness $\xi = 2$, yielding a particle volume fraction of $\varphi = 0.138$. The velocity of microswimmer motion is $V = 2/3B_1 = 0.25$ and the character of the motion is chosen by specifying $B_2 = 0.5$. We use the python script `particle_show_wall.py` for visualization in GOURMET.

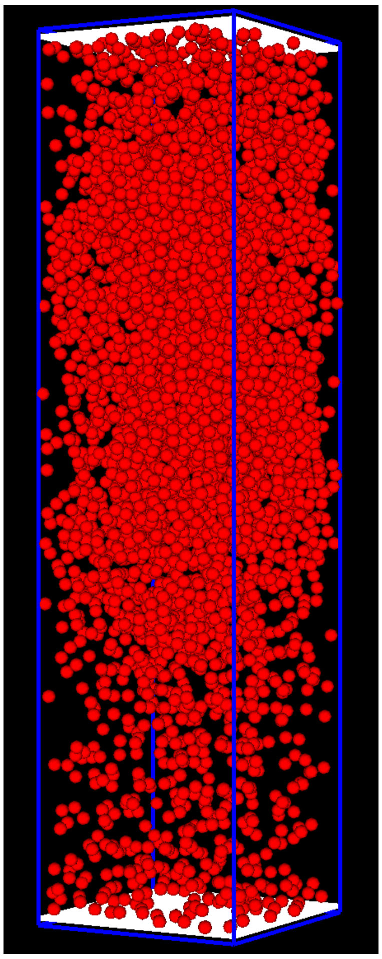


Figure 7.3: Motion of microswimmers confined between two parallel slabs.

Chapter 8

Simulating Quincke rollers

8.1 Theoretical background and basic equations

The study of active matter is not restricted to the motion of bacteria and other living organisms, but encompasses inanimate bodies as well, with systems of self-propelled colloids furnishing an important class of examples. The study of such systems is promising not only for deriving insight into fundamental properties, such as driving mechanisms and collective dynamics, but also for developing practical applications involving control via external fields or other means; among the most important—and difficult—challenges is to understand how self-propelled colloid systems are affected by hydrodynamic interactions.

8.1.1 Quincke rollers

Quincke rollers are spherical colloidal particles dispersed in a viscous fluid that exhibit self-propulsion initiated by rolling over the surface of a planar electrode, as illustrated in Fig. 8.1 [57]. The basic mechanism of this self-propulsion is the Quincke effect, which induces the rotational motion of colloids in an external electrostatic field; if the rotating colloids lie on the surface of an electrode slab, they experience additional fluid-mediated forces due to the non-slip boundary conditions at the electrode surface. The Quincke effect is a phenomenon in which the application of a DC electric field, with a strength above a certain threshold, to a system of dielectric colloidal particles dispersed in a conducting solvent induces spontaneous rotation of the particles [58]. The particles rotate about an axes constrained to lie in a plane perpendicular to the applied electric field, and at rotational speeds proportional to the field strength. Systems of multiple Quincke rollers experience not only hydrodynamic forces, but also electrostatic interactions, excluded-volume effects, and other phenomena that affect the collective dynamics and result in highly complex behavior [57, 59].

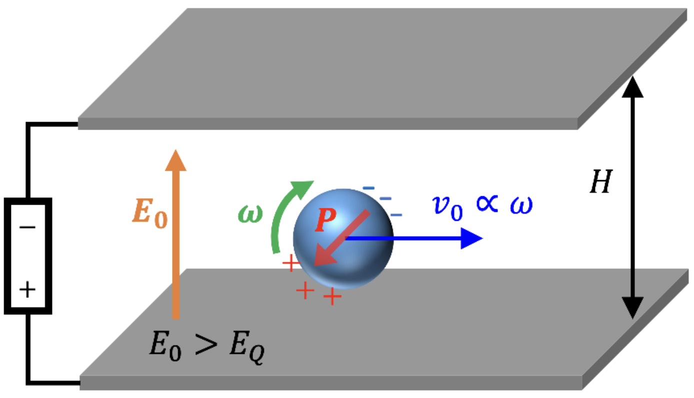


Figure 8.1: Schematic depiction of a Quincke roller. The application of a uniform DC electric field to a system of spherical colloidal particles produces torque on the particles due to the instability of the induced electric dipole moment. If the field strength E_0 exceeds a given threshold E_Q, the colloids begin to rotate. The imposition of no-slip boundary conditions at the electrode surface causes the colloids to move with a translational velocity v_0 proportional to the angular velocity ω of their rotation.

8.1.2 Basic equations for Quincke rollers

To analyze the motion of Quincke rollers using the SP method, we add three new two-body potentials to the equations of motion (3.5) and (3.6):

$$U_{ij}^{DD}(r) = \frac{1}{4\pi\epsilon}\left(\frac{\boldsymbol{P}_i \cdot \boldsymbol{P}_j}{r^3} - 3\frac{(\boldsymbol{P}_i \cdot \boldsymbol{r})(\boldsymbol{P}_j \cdot \boldsymbol{r})}{r^5}\right), \tag{8.1}$$

$$U_{ij}^{LJ}(r) = 4\epsilon^{LJ}\left[\left(\frac{\sigma}{r}\right)^{36} - \left(\frac{\sigma}{r}\right)^{18}\right] + \epsilon, \tag{8.2}$$

$$U_{ij}^{EF}(r) = -\epsilon^{EF}\exp(-r/3\sigma)/r^2. \tag{8.3}$$

Here i, j are particle indices, ϵ is the dielectric permittivity, and U^{DD}, U^{LJ}, U^{EF} describe interaction potentials due to electric dipole moments, excluded-volume effects, and electro-osmotic flows, respectively, with ϵ^{LJ} and ϵ^{EF} setting the magnitudes of the latter two potentials. Assuming the external electric field points in the z direction, dipole moments may be described by their parallel and perpendicular components P_z and P_{xy}. For simplicity, we consider only steady-state Quincke phenomena. The directions of the colloid rotation axes are randomly distributed in a plane perpendicular to the electric-field direction, and the constant rotation speed ω is an input parameter for which the relevant scale is the rotational Reynolds number $\mathrm{Re}_r \equiv \rho_f \omega \sigma^2/\eta$. The xy components of a particle's dipole moment are determined by the cross product of the electric-field direction with the particle's rotation axis. To ensure that colloid rotation axes do not deviate from their plane of definition, we add restorative torques, in the form of harmonic potentials, to correct for any such deviation. Finally, for the magnitudes of the interaction potentials we choose $\epsilon^{LJ} = \epsilon^{EF}$.

8.2 Input UDF files

8.2.1 Fluid settings

The following choices are available for `constitutive_eq`.

`Navier_Stokes`:	Newtonian fluid
`Navier_Stokes_FDM`:	Newtonian fluid

8.2.2 Object (particle) settings

The only available value for `object_type.type` is `rigid`.

8.3 Computational examples

8.3.1 Motion of a single Quincke roller

The input UDF file for this example is `a2_N1_Rer025.udf` in the folder `Examples/12/`.

```
    $ cd Examples/12
    $ ../../kapsel -Ia2_N1_Rer025.udf -Ooutput.udf -Ddefine.udf -Rrestart.udf
```

In this example we use a $64 \times 64 \times 32$ computational mesh and set the following parameter values: particle diameter $\sigma = 4$, interface thickness $\xi = 2$, rotational Reynolds number $\mathrm{Re}_r = 0.25$, gravitational acceleration $g = 1$. The python script `streamplot.py` in the `Examples/12` folder reads data from `h5`-format KAPSEL output files and generates stream plots of fluid flow and simulation snapshot images.

Fig. 8.2 shows a comparison between the simulated self-propulsion velocity of a single Quincke roller and the predictions of lubrication theory [60, 61] and fluid-flow field surrounding the Quincke roller.

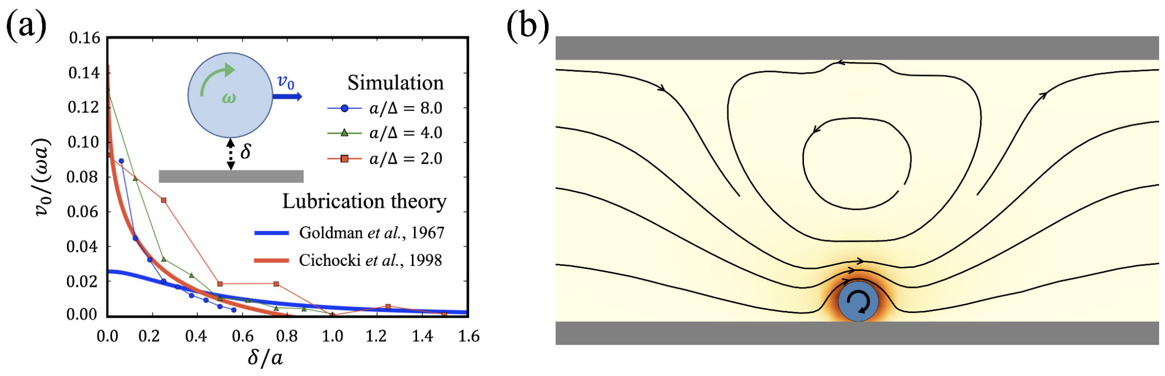


Figure 8.2: Motion of a single Quincke roller. (a) Dependence of particle self-propulsion velocity v_0 on separation distance δ between the particle and planar electrode surfaces, showing both KAPSEL simulation results and the predictions of lubrication theory. The agreement between simulated and theoretical results improves with increasing particle radius a. (b) Fluid-flow field surrounding the Quincke roller, with colors indicating the flow-velocity magnitudes.

8.3.2 Motion of multiple Quincke rollers

The input UDF file for this example is `a2_N200.udf` in the `Examples/12/` folder.

```
    $ cd Examples/12
    $ ../../kapsel -Ia2_N200.udf -Ooutput.udf -Ddefine.udf -Rrestart.udf
```

In this example (Fig. 8.3) we use a $128 \times 128 \times 32$ computational mesh and consider $N_p = 200$ particles, for a particle volume fraction of $\varphi = 0.00568$ (the particle area fraction is 0.15). For the strength of interaction potentials we set $\epsilon^{LJ} = \epsilon^{EF} = 1.0$. For dipole strengths we set $P_{xy}^2/4\pi = 6.0, |P_z|/|P_{xy}| = 3.0$. Other parameter values are set as in the single-particle example above. The initial particle configuration is generated by the python script `init_loc_q.py` in the `Examples/12/` folder. Snapshot images are generated by the python script `snapshot.py`.

Fig. 8.3 shows a simulation snapshot of the 200 Quincke rollers on the planar electrode surface.

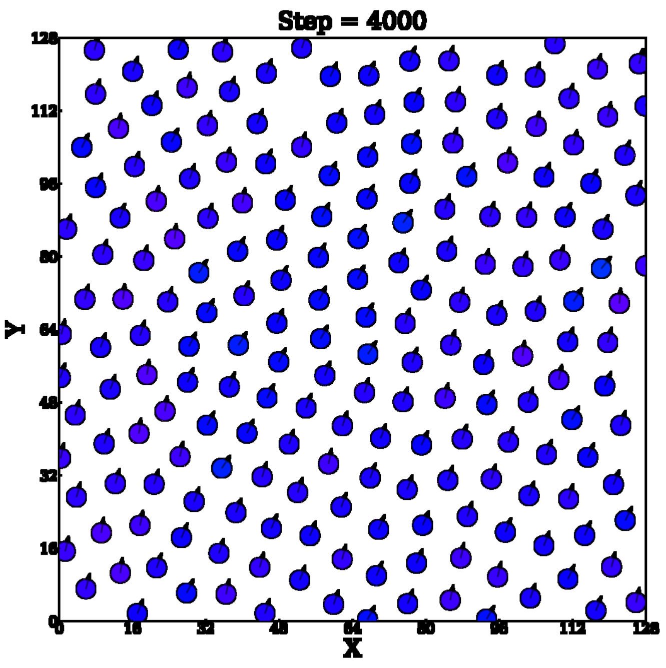


Figure 8.3: Snapshot from a KAPSEL simulation of 200 Quincke rollers. The arrows indicate the rotation axes.

Appendix A

Definition of the input UDF file format

This appendix documents the format of the input UDF files used to define KAPSEL simulations. The structure of input UDF files is defined by a UDF definition file, conventionally named `define.udf`, and includes the following items:

- Fluid settings
- Object settings
- Common simulation settings
- Configuring selectable features
- Data output settings
- Definition of UDF output data classes
- Restart settings
- GOURMET display settings

The following subsections will discuss each of these items in the order listed here, and additional details may be found in the application-specific discussions of Sections 3-8 of this manual.

A.1 Fluid settings

The `constitutive_eq` structure configures the properties of the fluid to be simulated. The type of fluid is specified by selecting one of the 8 available options for the `type` selector:

```
type: select {
  'Navier_Stokes',
  'Shear_Navier_Stokes',
  'Shear_Navier_Stokes_Lees_Edwards',
  'Electrolyte',
  'Navier_Stokes_FDM',
  'Navier_Stokes_Cahn_Hilliard_FDM',
  'Shear_Navier_Stokes_Lees_Edwards_FDM',
  'Shear_NS_LE_CH_FDM'
}
```

Each of these fluid types has its own specific settings, as we now discuss.

<u>`Navier_Stokes`</u>

Set `constitutive_eq` to `Navier_Stokes` to simulate particles dispersed in a Newtonian fluid (Section 3). The `Navier_Stokes` structure has the following fields describing fluid-related simulation parameters:

```
DX:        double [L] "lattice spacing (=1), fixed for all directions"
RHO:       double [rho] "mass density of solvent"
ETA:       double [eta] "shear viscosity of solvent"
kBT:       double [epsilon] "temperature"
alpha_v:   double "correction coefficient of V"
alpha_o:   double "correction coefficient of Omega"
```

Here DX is the lattice spacing, RHO is the fluid density, ETA is the fluid viscosity, kBT is the particle temperature, alpha_v is an adjustable parameter for the particle velocity, and alpha_o is an adjustable parameter for the particle angular velocity.

Shear_Navier_Stokes

Set constitutive_eq to Shear_Navier_Stokes to simulate particles dispersed in a Newtonian fluid in the presence of zigzag shear flow. (This type of simulation is not discussed in this manual.) The Shear_Navier_Stokes structure has the following fields describing fluid-related simulation parameters:

```
DX:        double [L] "lattice spacing (=1), fixed for all directions"
RHO:       double [rho] "mass density of solvent"
ETA:       double [eta] "shear viscosity of solvent"
kBT:       double [epsilon] "temperature"
alpha_v:   double "correction coefficient of V"
alpha_o:   double "correction coefficient of Omega"
```

Here DX is the lattice spacing, RHO is the fluid density, ETA is the fluid viscosity, kBT is the particle temperature, alpha_v is an adjustable parameter for the particle velocity, and alpha_o is an adjustable parameter for the particle angular velocity.

The External_field structure has the following fields describing zigzag shear-flow parameters:

```
External_field: {
  type: select {"DC","AC"}
  DC: {
    Shear_rate:    double [1/tau] "shear rate"
  }
  AC: {
    Shear_rate:    double [1/tau] "shear rate"
    Frequency:     double [1/tau] "alternating frequency"
  }
}
```

Set the type selector to DC for steady-state shear flow. In this case, the Shear_rate field sets the steady-state shear velocity. Alternatively, set the type selector to AC for oscillatory shear flow. In this case, Shear_rate sets the amplitude of the oscillatory shear velocity and Frequency sets the oscillation frequency.

Shear_Navier_Stokes_Lees_Edwards

Set constitutive_eq to Shear_Navier_Stokes_Lees_Edwards to simulate particles dispersed in a Newtonian fluid under shear flow with Lees-Edwards boundary conditions (Section 4). The Shear_Navier_Stokes_Lees_Edwards structure has the following fields describing fluid-related simulation parameters:

```
DX:        double [L] "lattice spacing (=1), fixed for all directions"
RHO:       double [rho] "mass density of solvent"
ETA:       double [eta] "shear viscosity of solvent"
kBT:       double [epsilon] "temperature"
alpha_v:   double "correction coefficient of V"
alpha_o:   double "correction coefficient of Omega"
```

DX is the lattice spacing, RHO is the fluid density, ETA is the fluid viscosity, kBT is the particle temperature, alpha_v is an adjustable parameter for the particle velocity, and alpha_o is an adjustable parameter for the particle angular velocity.

The External_field structure has the following fields describing shear-flow parameters with Lees-Edwards boundary conditions:

```
External_field: {
  type: select {"DC","AC"}
  DC: {
    Shear_rate:   double [1/tau] "shear rate"
  }
  AC: {
    Shear_rate:   double [1/tau] "shear rate"
    Frequency:    double [1/tau] "alternating frequency"
  }
}
```

Set the `type` selector to `DC` for steady-state shear flow. In this case, the `Shear_rate` field sets the steady-state shear velocity. Alternatively, set the `type` selector to `AC` for oscillatory shear flow. In this case, `Shear_rate` sets the amplitude of the oscillatory shear velocity and `Frequency` sets the oscillation frequency.

`Electrolyte`

Set `constitutive_eq` to `Electrolyte` to simulate charged colloidal particles dispersed in an electrolytic solution (Section 5). The `Electrolyte` structure has the following fields describing fluid-related simulation parameters:

```
DX:          double [L] "lattice spacing (=1), fixed for all directions"
RHO:         double [rho] "mass density of solvent"
ETA:         double [eta] "shear viscosity of solvent"
kBT:         double [epsilon] "temperature"
alpha_v:     double "correction coefficient of V"
alpha_o:     double "correction coefficient of Omega"
Dielectric_cst:    double "dielectric constant"
INIT_profile: select {
  "Uniform",
  "Poisson_Boltzmann"
} "Initial condition for density profile of ions"
```

`DX` is the lattice spacing, `RHO` is the fluid density, `ETA` is the fluid viscosity, `kBT` is the particle temperature, `alpha_v` is an adjustable parameter for the particle velocity, `alpha_o` is an adjustable parameter for the particle angular velocity, `Dielectric_cst` is the dielectric permittivity of the solvent, and the `INIT_profile` selector may be set to `Uniform` or `Poisson_Boltzmann` to select the method used to prepare the initial ion distribution.

The `Add_salt` structure has the following fields regarding ions in the electrolytic solution:

```
Add_salt: {
  type:select {"saltfree","salt"}
  saltfree: {
    Valency_counterion:        double "valency of counterion"
    Onsager_coeff_counterion:  double "Onsager coefficient of counterion"
  }
  salt: {
    Valency_positive_ion:      double "valency of positive ion"
    Valency_negative_ion:      double "valency of negative ion"
    Onsager_coeff_positive_ion: double "Onsager coefficient of positive ion"
    Onsager_coeff_negative_ion: double "Onsager coefficient of negative ion"
    Debye_length:              double "Debye screening length in the unit of DX"
  }
}
```

The `type` selector may be set to `saltfree` for a simulation in which only counterions are present, or to `salt` for a simulation containing two species (one positive, one negative) of salt ions in addition to counterions.

For `type=saltfree`, `Valency_counterion` and `Onsager_coeff_counterion` specify the valence and Onsager transport coefficient for counterions.

For `type=salt`, the options `Valency_positive_ion`, `Valency_negative_ion`, `Onsager_coeff_positive_ion`, and `Onsager_coeff_negative_ion` specify valences and Osager coefficients for positive and negative ions, and `Debye_length` specifies the Debye screening length.

The `Electric_field` structure offers the following options to describe external electric fields:

```
Electric_field: {
  type: select {"ON","OFF"}
  ON: {
    type: select {"DC","AC"}
    DC: {
      Ex:  double
      Ey:  double
      Ez:  double
    }
    AC: {
      Ex:  double
      Ey:  double
      Ez:  double
      Frequency:   double
    }
  }
}
```

The `type` selector may be set to `ON` or `OFF` for simulations with or without the application of an external electric field. In the former case, the choice `ON.type=DC` selects a constant electric field with Cartesian vector components specified by the `Ex`, `Ey`, and `Ez` options; the choice `ON.type=AC` selects an oscillating electric field with frequency set by the `Frequency` option and amplitudes given by the `Ex`, `Ey`, and `Ez` options.

`Navier_Stokes_FDM`

Set `constitutive_eq` to `Navier_Stokes_FDM` to simulate particles dispersed in a Newtonian fluid using finite-difference methods (Appendix D). In this case, the `NS_solver` field of the `Navier_Stokes_FDM` structure configures the method used to solve the Navier-Stokes equations:

```
NS_solver: {
  type: select {
    'explicit_scheme',
    'implicit_scheme'
  } "explicit_scheme: explicit MAC scheme, ON: implicit MAC scheme"
  implicit_scheme: {
    tolerance:         double "stopping criteria"
    maximum_iteration: int "number of maximum iteration"
  }
}
```

Set the `type` selector to `explicit_scheme` or `implicit_scheme` to specify an explicit or implicit solver. For the implicit case, use `tolerance` and `maximum_iteration` to specify the convergence criterion and maximum number of iterations.

The `Navier_Stokes_FDM` structure also includes the following fields describing fluid-related simulation parameters:

```
DX:       double [L] "lattice spacing (=1), fixed for all directions"
RHO:      double [rho] "mass density of solvent"
ETA:      double [eta] "shear viscosity of solvent"
kBT:      double [epsilon] "temperature"
alpha_v:  double "correction coefficient of V"
alpha_o:  double "correction coefficient of Omega"
```

Here `DX` is the lattice spacing, `RHO` is the fluid density, `ETA` is the fluid viscosity, `kBT` is the particle temperature, `alpha_v` is an adjustable parameter for the particle velocity, and `alpha_o` is an adjustable parameter for the particle angular velocity.

`Navier_Stokes_Cahn_Hilliard_FDM`

Set `constitutive_eq` to `Navier_Stokes_Cahn_Hilliard_FDM` to simulate particles dispersed in a two-component phase-separated fluid (Section 6). In this case, the `NS_solver` field of the `Navier_Stokes_Cahn_Hilliard_FDM` structure configures the method used to solve the Navier-Stokes equations:

```
NS_solver: {
  type: select {
    'explicit_scheme',
    'implicit_scheme'
  } "explicit_scheme: explicit MAC scheme, ON: implicit MAC scheme"
  implicit_scheme: {
    tolerance:         double "stopping criteria"
    maximum_iteration: int "number of maximum iteration"
    viscosity_change: select {'ON','OFF'}
    ON: {
      ETA_A:            double [eta] "shear viscosity of solvent A"
      ETA_B:            double [eta] "shear viscosity of solvent B"
    }
  }
}
```

Set the `type` selector to `explicit_scheme` or `implicit_scheme` to specify an explicit or implicit solver. For the implicit case, use `tolerance` and `maximum_iteration` to specify the convergence criterion and maximum number of iterations. The `viscosity_change` selector allows the two fluid components to be assigned different viscosities. If `viscosity_change` is set to `ON`, then the settings `ETA_A` and `ETA_B` specify separate viscosities for the A and B fluid components. If `viscosity_change` is set to `OFF`, then the single setting `ETA` discussed in detail later specifies the common viscosity of both fluid components.

The `CH_solver` structure describes the method used by KAPSEL to solve the Cahn-Hilliard equations:

```
CH_solver: {
  type: select {
    'explicit_scheme',
    'implicit_scheme'
  } "explicit_scheme: explicit Euler scheme, ON: implicit BDFAB scheme"
  implicit_scheme: {
    tolerance:         double "stopping criteria"
    maximum_iteration: int "number of maximum iteration"
  }
}
```

Set the `type` selector to `explicit_scheme` or `implicit_scheme` to specify an explicit or implicit solver. For the implicit case, use `tolerance` and `maximum_iteration` to specify the convergence criterion and maximum number of iterations.

The `Navier_Stokes_Cahn_Hilliard_FDM` structure also includes the following fields describing fluid-related simulation parameters:

```
DX:       double [L] "lattice spacing (=1), fixed for all directions"
RHO:      double [rho] "mass density of solvent"
ETA:      double [eta] "shear viscosity of solvent"
kBT:      double [epsilon] "temperature"
alpha_v:  double "correction coefficient of V"
alpha_o:  double "correction coefficient of Omega"
```

`DX` is the lattice spacing, `RHO` is the fluid density, `ETA` is the fluid viscosity, `kBT` is the particle temperature, `alpha_v` is an adjustable parameter for the particle velocity, and `alpha_o` is an adjustable parameter for the particle angular velocity.

The `Potential` structure describes the phase-separation potential. Its `type` selector may be set to `Landau` or `Flory_Huggins`:

```
type: select {'Landau','Flory_Huggins'}
```

Selecting `Landau` chooses a Landau double-well potential, in which case the following settings may be configured to specify parameter values:

```
composition_ratio:     double "composition ratio of A and B fluids"
initial_fluctuation:   double "initial fluctuation of concentration"
a:                     double "GL parameter (third order term)"
b:                     double "GL parameter (first order term)"
d:                     double "penalty factor in fictitious particle domain"
w:                     double "penalty factor on particle surface domain"
z:                     double "penalty factor on fluid interface"
psi_0:                 double "psi_0 for particle"
alpha:                 double "surface parameter on fluid-fluid surface"
kappa:                 double "mobility parameter"
```

Here `composition_ratio` sets the composition ratio of the two-component phase-separated fluid, `initial_fluctuation` sets the initial magnitude of concentration fluctuations, `a` is the coefficient of the cubic term, `b` is the coefficient of the linear term, `d` is a parameter governing the magnitude of the term responsible for ensuring that the fluid composition inside the particle domains approaches `psi_0`, `w` is a parameter governing the affinity of particle interfaces, `z` is a parameter governing the affinity between particle interfaces and fluid-fluid interfaces, `psi_0` is the value of ψ in the interior of particles, `alpha` is a fluid interface parameter, and `kappa` is a mobility parameter.

On the other hand, selecting `Flory_Huggins` for `type` chooses a Flory-Huggins double-well potential, in which case the following settings may be configured to specify parameter values:

```
composition_ratio:     double "composition ratio of A and B fluids"
initial_fluctuation:   double "initial fluctuation of concentration"
na:            double "(reduced) number of polymerization of A component"
nb:            double "(reduced) number of polymerization of B component"
chi:           double "Flory's interaction parameter (chi parameter)"
d:             double "penalty factor in fictitious particle domain"
w:             double "penalty factor on particle surface domain"
z:             double "penalty factor on fluid interface"
psi_0:         double "psi_0 for particle"
alpha:         double "surface parameter on fluid-fluid surface"
kappa:         double "mobility parameter"
```

Here `composition_ratio` sets the composition ratio of the two-component phase-separated fluid, `initial_fluctuation` sets the initial magnitude of concentration fluctuations, `na` and `nb` are the degrees of polymerization of the A and B fluid components, `chi` is the Flory interaction parameter, `d` is a parameter governing the magnitude of the term responsible for ensuring that the fluid composition inside the particle domains approaches `psi_0`, `w` is a parameter governing the affinity between fluid and particle interfaces , `z` is a parameter governing the affinity between particle interfaces and fluid-fluid interfaces, `psi_0` is the value of ψ in the interior of particles, `alpha` is a fluid interface parameter, and `kappa` is a mobility parameter.

The `Wall_Potential` structure describes planar walls:

```
type: select {'ON', 'OFF'}
ON:{
  w :        double "penalty factor on wall surface domain"
  psi_0: {
    magnitude:   double "psi_0 magnitude"
    profile : select {'uniform', 'user_specify'}
    user_specify:{
      PSI0[][]:{
        value :  double
      }
    }
  }
  DRYING:{
    type: select {'ON', 'OFF'}
    ON:{
      psi_dry : double "psi_dry"
    }
  }
}
```

Setting the `type` selector to `ON` enables planar walls. In this case, `w` is a parameter governing the affinity between planar walls and fluids, and `psi_0` controls the value of ψ in the interior of planar walls. `psi_0` contains the field's `magnitude`, specifying the magnitude of the affinity between planar walls and fluids, and `profile`, which may be set to `uniform` (uniform affinity over surfaces of planar walls) or `user_specify` (user-specified variation of affinity over planar walls). If `profile` is set to `user_specify`, use `PSI0[][]` to configure values by hand. Within the `DRYING` structure, set the `type` selector to `ON` to specify a drying simulation. In this case, `psi_dry` is a parameter governing the speed of drying.

<u>`Shear_Navier_Stokes_Lees_Edwards_FDM`</u>

Set `constitutive_eq` to `Shear_Navier_Stokes_Lees_Edwards_FDM` for finite-difference simulations of particles dispersed in a Newtonian fluid under shear flow with Lees-Edwards boundary conditions (Appendix D). In this case, the `NS_solver` field of the `Shear_Navier_Stokes_Lees_Edwards_FDM` structure configures the method used to solve the Navier-Stokes equations:

```
NS_solver: {
  type: select {
    'explicit_scheme',
    'implicit_scheme'
  } "explicit_scheme: explicit MAC scheme, ON: implicit MAC scheme"
  implicit_scheme: {
    tolerance:         double "stopping criteria"
    maximum_iteration: int "number of maximum iteration"
  }
}
```

Set the `type` selector to `explicit_scheme` or `implicit_scheme` to specify an explicit or implicit solver. For the implicit case, use `tolerance` and `maximum_iteration` to specify the convergence criterion and maximum number of iterations.

The `Shear_Navier_Stokes_Lees_Edwards_FDM` structure also includes the following fields describing fluid-related simulation parameters:

```
DX:       double [L] "lattice spacing (=1), fixed for all directions"
RHO:      double [rho] "mass density of solvent"
ETA:      double [eta] "shear viscosity of solvent"
kBT:      double [epsilon] "temperature"
alpha_v:  double "correction coefficient of V"
alpha_o:  double "correction coefficient of Omega"
```

Here DX is the lattice spacing, RHO is the fluid density, ETA is the fluid viscosity, kBT is the particle temperature, alpha_v is an adjustable parameter for the particle velocity, and alpha_o is an adjustable parameter for the particle angular velocity.

The External_field structure contains parameters affecting shear flows under Lees-Edwards boundary conditions:

```
External_field: {
  type: select {"DC","AC"}
  DC: {
    Shear_rate:    double [1/tau] "shear rate"
  }
  AC: {
    Shear_rate:    double [1/tau] "shear rate"
    Frequency:     double [1/tau] "alternating frequency"
  }
}
```

Set the type selector to DC for steady-state shear flow. In this case, the Shear_rate field sets the steady-state shear velocity. Alternatively, set the type selector to AC for oscillatory shear flow. In this case, Shear_rate sets the amplitude of the oscillatory shear velocity and Frequency sets the oscillation frequency.

Shear_NS_LE_CH_FDM

Set constitutive_eq to Shear_NS_LE_CH_FDM to simulate particles dispersed in two-component phase-separated fluids under shear flow with Lees-Edwards boundary conditions (Section 6). In this case, the NS_solver field of the Shear_NS_LE_CH_FDM structure configures the method used to solve the Navier-Stokes equations:

```
NS_solver: {
  type: select {
    'explicit_scheme',
    'implicit_scheme'
  } "explicit_scheme: explicit MAC scheme, ON: implicit MAC scheme"
  implicit_scheme: {
    tolerance:         double "stopping criteria"
    maximum_iteration: int "number of maximum iteration"
    viscosity_change: select {'ON','OFF'}
      ON: {
        ETA_A:            double [eta] "shear viscosity of solvent A"
        ETA_B:            double [eta] "shear viscosity of solvent B"
      }
  }
}
```

Set the type selector to explicit_scheme or implicit_scheme to specify an explicit or implicit solver. For the implicit case, use tolerance and maximum_iteration to specify the convergence criterion and maximum number of iterations. The viscosity_change selector allows the two fluid components to be assigned different viscosities. If viscosity_change is set to ON, then the settings ETA_A and ETA_B specify separate viscosities for the A and B fluid components. If viscosity_change is set to OFF, then the single setting ETA discussed in detail later specifies the common viscosity of both fluid components.

The CH_solver structure describes the method used by KAPSEL to solve the Cahn-Hilliard equations:

```
CH_solver: {
  type: select {
    'explicit_scheme',
    'implicit_scheme'
  } "explicit_scheme: explicit Euler scheme, ON: implicit BDFAB scheme"
  implicit_scheme: {
    tolerance:         double "stopping criteria"
    maximum_iteration: int "number of maximum iteration"
  }
}
```

Set the type selector to explicit_scheme or implicit_scheme to specify an explicit or implicit solver. For the implicit case, use tolerance and maximum_iteration to specify the convergence criterion and maximum number of iterations.

The Shear_NS_LE_CH_FDM structure also includes the following fields describing fluid-related simulation parameters:

```
DX:             double [L] "lattice spacing (=1), fixed for all directions"
RHO:            double [rho] "mass density of solvent"
ETA:            double [eta] "shear viscosity of solvent"
kBT:            double [epsilon] "temperature"
alpha_v:         double "correction coefficient of V"
alpha_o:         double "correction coefficient of Omega"
```

Here DX is the lattice spacing, RHO is the fluid density, ETA is the fluid viscosity, kBT is the particle temperature, alpha_v is an adjustable parameter for the particle velocity, and alpha_o is an adjustable parameter for the particle angular velocity.

The Potential structure describes the phase-separation potential. Its type selector may be set to Landau or Flory_Huggins:

```
type: select {'Landau','Flory_Huggins'}
```

Selecting Landau chooses a Landau double-well potential, in which case the following settings may be configured to specify parameter values:

```
composition_ratio:     double "composition ratio of A and B fluids"
initial_fluctuation:   double "initial fluctuation of concentration"
a:     double "GL parameter (third order term)"
b:     double "GL parameter (first order term)"
d:     double "penalty factor in fictitious particle domain"
w:     double "penalty factor on particle surface domain"
z:     double "penalty factor on fluid interface"
psi_0: double "psi_0 for particle"
alpha: double "surface parameter on fluid-fluid surface"
kappa: double "mobility parameter"
```

composition_ratio sets the composition ratio of the two-component phase-separated fluid, initial_fluctuation sets the initial magnitude of concentration fluctuations, a is the coefficient of the cubic term, b is the coefficient of the linear term, d is a parameter governing the magnitude of the term responsible for ensuring that the fluid composition inside the particle domains approaches psi_0, w is a parameter governing the affinity of particle interfaces, z is a parameter governing the affinity between particle interfaces and fluid-fluid interfaces, psi_0 is the value of ψ in the interior of particles, alpha is a fluid interface parameter, and kappa is a mobility parameter.

On the other hand, selecting Flory_Huggins for type chooses a Flory-Huggins double-well potential, in which case the following settings may be configured to specify parameter values:

```
composition_ratio:     double "composition ratio of A and B fluids"
initial_fluctuation:   double "initial fluctuation of concentration"
na:    double "(reduced) number of polymerization of A component"
nb:    double "(reduced) number of polymerization of B component"
chi:   double "Flory's interaction parameter (chi parameter)"
d:     double "penalty factor in fictitious particle domain"
w:     double "penalty factor on particle surface domain"
z:     double "penalty factor on fluid interface"
psi_0: double "psi_0 for particle"
alpha: double "surface parameter on fluid-fluid surface"
kappa: double "mobility parameter"
```

Here composition_ratio sets the composition ratio of the two-component phase-separated fluid, initial_fluctuation sets the initial magnitude of concentration fluctuations, na and nb are the degrees of polymerization of the A and B fluid components, chi is the Flory interaction parameter, d is a parameter governing the magnitude of the term responsible for ensuring that the fluid composition inside the particle domains approaches psi_0, w is a parameter governing the affinity between fluid and particle interfaces , z is a parameter governing the affinity between particle interfaces and fluid-fluid interfaces, psi_0 is the value of ψ in the interior of particles, alpha is a fluid interface parameter, and kappa is a mobility parameter.

The External_field structure has the following fields describing shear-flow parameters under Lees-Edwards boundary conditions:

```
External_field: {
  type: select {"DC","AC"}
  DC: {
    Shear_rate:    double [1/tau] "shear rate"
  }
  AC: {
    Shear_rate:    double [1/tau] "shear rate"
    Frequency:     double [1/tau] "alternating frequency"
  }
}
```

Set the `type` selector to `DC` for steady-state shear flow. In this case, the `Shear_rate` field sets the steady-state shear velocity. Alternatively, set the `type` selector to `AC` for oscillatory shear flow. In this case, `Shear_rate` sets the amplitude of the oscillatory shear velocity and `Frequency` sets the oscillation frequency.

A.2 Object settings

The `object_type` structure describes the properties of objects dispersed in fluids. Its `type` selector defines the type of object:

```
type: select {'spherical_particle','chain','rigid'}
```

<u>`spherical_particle`</u>

Set `object_type.type` to `spherical_particle` to simulate spherical particles dispersed in fluids. In this case the following parameters may be configured:

```
Particle_spec[]:{
  Particle_number: int "number of colloidal particles"
  MASS_RATIO: double "mass density ratio colloid/solvent"
  Surface_charge: double "surface charge of colloid"
  janus_axis: select {'NONE', 'X', 'Y', 'Z'} "janus axis in body_fixed frame"
  janus_propulsion: select{'OFF', 'TUMBLER', 'SQUIRMER', 'OBSTACLE'}
  janus_force: Vector3d "self-propulsion force"
  janus_torque: Vector3d "self-propulsion torque"
  janus_slip_vel: float "Slip velocity coeff B1"
  janus_slip_mode: float "Blake squirmer mode B2/B1"
  janus_rotlet_C1: float "rotlet coefficient C1"
  janus_rotlet_dipole_C2: float  "rotlet dipole C2"
}
```

`Particle_number` sets the number of particles. `MASS_RATIO` specifies the ratio of particle density to fluid density. `Surface_charge` sets the particle surface charge; this is only applicable if `constitutive_eq.type` is set to `Electrolyte`.

Properties of the form `janus_*` specify properties of Janus particles. The `janus_axis` selector may be set to `NONE`, `X`, `Y`, or `Z` to set the orientation of the Janus axis in a particle-fixed coordinate system. The propulsive motion of Janus particles is specified by the `janus_propulsion` selector, which may be set to `OFF` (disabled), `TUMBLER` (particle propelled along propulsion axis by fixed external force), `SQUIRMER` (squirmer particle propelled along propulsion axis by slip boundary conditions), or `OBSTACLE` (particle forming a fixed obstacle). The Cartesian components of the propulsive force and torque on Janus particles may be specified by setting values for `janus_force.x`, `janus_force.y`, `janus_force.z`, `janus_torque.x`, `janus_torque.y`, and `janus_torque.z`. `janus_slip_vel` sets the parameter B_1 determining the surface-slip velocity for Janus particles, while `janus_slip_mode` sets the parameter B_2/B_1 determining the type of slip motion for Janus-particles; finally, `janus_rotlet_C1` and `janus_rotlet_dipole_C2` are the Janus-particle parameters C_1 and C_2 describing rotlet dipole exerted on particles/fluids.

<u>`chain`</u>

Set `object_type.type` to `chain` to simulate flexible particle chains dispersed in fluids. In this case the following parameter settings are available:

```
chain:{
  Chain_spec[]:{
    Beads_number: int "number of beads in a chain"
    Chain_number: int "number of chains"
    MASS_RATIO: double "mass density ratio chain/solvent"
    Surface_charge: double "surface charge of colloid"
    janus_axis: select {'NONE', 'X', 'Y', 'Z'} "janus axis in body_fixed frame"
  }
}
```

`Beads_number` is the number of beads associated with a single chain, while `Chain_number` is the number of chains. `MASS_RATIO` is the ratio of bead density to fluid density. `Surface_charge` sets the particle surface charge; this is only applicable if `constitutive_eq.type` is set to `Electrolyte`. The `janus_axis` selector may be set to `NONE`, `X`, `Y`, or `Z` to specify the orientation of the Janus axis in a bead-fixed coordinate system.

`rigid`

Set `object_type.type` to `rigid` to simulate rigid bodies comprised of spherical beads dispersed in fluids. In this case the following parameter settings are available:

```
rigid:{
  Rigid_spec[]:{
    Beads_number: int "number of beads in a rigid"
    Rigid_number: int "number of rigids"
    MASS_RATIO: double "mass density ratio rigid/solvent"
    Surface_charge: double "surface charge of particle"
    Rigid_motion: select {'fix','free'}
    Rigid_velocity: Vector3d "speed of translation ### fix only ###"
    Rigid_omega: Vector3d "angular velocity ### fix only ###"
  }
}
```

Here `Beads_number` is the number of beads associated with a rigid body, while `Rigid_number` is the number of rigid bodies. `MASS_RATIO` is the ratio of bead density to fluid density. `Surface_charge` sets the particle surface charge; this is only applicable if `constitutive_eq.type` is set to `Electrolyte`. The motion of rigid bodies is described by the `Rigid_motion` selector, which may be set to `free` (for free motion) or `fix` (for motion with fixed translational and rotational velocities). For the choice `Rigid_motion=fix`, all six the Cartesian components of the translational and rotational velocities of rigid bodies (in the lab frame) are specified via `Rigid_velocity.x`, `Rigid_velocity.y`, `Rigid_velocity.z`, `Rigid_omega.x`, `Rigid_omega.y`, and `Rigid_omega.z`. To free/fix individual degrees of freedom use this option together with the `switch.free_rigid` option detailed below 'Configuring selectable features'.

A.3 Common simulation settings

Sections A.1 and A.2 described configurable settings for fluids and objects. In this section we describe common simulation settings applicable in general.

Objects in KAPSEL simulations are comprised of particles. The following parameters affect common properties of these primary particles:

```
A_XI: double "interface thickness in the unit of DX"
A: double "colloid radius in the unit of DX"
```

`A_XI` and `A` respectively specify the particle interface thickness and the particle radius in units of the lattice spacing `DX`.

The `gravity` structure describes gravitational forces on the system simulated by KAPSEL:

```
gravity: {
  G: double [L*tau^-2] "gravitational acceleration constant"
  G_direction: select {'-X','-Y','-Z'} "direction of gravitational acceleration"
}
```

Here `G` is the gravitational acceleration and the `G_direction` selector may be set to `-X`, `-Y`, or `-Z` to specify the direction of gravitational forces.

The following common parameters affect interparticle interaction potentials:

```
EPSILON: double [epsilon] "Lennard-Jones depth"
LJ_powers: select {'12:6','24:12','36:18','macro_vdw','electro_osmotic_flow'} "set of
 ↪  power exponents of LJ potential"
```

`EPSILON` specifies the unit of energy in the Lennard-Jones potential, while `LJ_powers` may be set to one of the following 5 values:

- `12:6`, `24:12` or `36:18` to select a Lennard-Jones potential with the given exponents,
- `macro_vdw` to select a macroscopic interparticle potential, or

- electro_osmotic_flow, a choice used only for Quincke rollers.

The mesh structure describes the computational mesh used for fluid calculations:

```
mesh: {
  NPX: int "number of mesh in x-direction = 2^NPX"
  NPY: int "number of mesh in y-direction = 2^NPY"
  NPZ: int "number of mesh in z-direction = 2^NPZ"
}
```

Here NPX, NPY, and NPZ are the base-2 logarithms of the numbers of mesh points in the x, y, and z directions; that is, the numbers of mesh points are $L_x = 2^{\mathtt{NPX}}$, $L_y = 2^{\mathtt{NPY}}$, and $L_z = 2^{\mathtt{NPZ}}$.

Timestamp settings

The time_increment structure describes simulation timesteps:

```
time_increment: {
  type: select {"auto","manual"}
  auto: {
    factor: double "delta_t = factor * h(determined by system parameters)"
  }
  manual: {
    delta_t: double [tau]
  }
}
```

Set the type selector to auto to set the timestamp to its maximum allowed value, given by $T_{step} = \rho/\eta k_{max}^2$ where k_{max} is the maximum wavenumber determined by the lattice spacing Δ. In this case, factor specifies a multiplicative scale factor, so the actual simulation timestep is given by $\Delta t = T_{step} \times$ factor. Alternatively, set the type selector to manual and set delta_t to your preferred timestep value.

A.4 Configuring selectable features

This section describes features defined in the switch structure.

Set the ROTATION selector to ON or OFF to include or exclude particle rotational motion from consideration in KAPSEL simulations.

```
  ROTATION: select {'ON','OFF'} "OFF: not solve rotation, ON: solve rotation"
```

Use the LJ_truncate selector to specify the form of the Lennard-Jones potential acting between particles. Set to OFF for the usual form of the potential, including the attractive term. Set to ON to exclude the attractive term, retaining only the repulsive term. Set to NONE to exclude both terms, i.e., to disable the interparticle potential entirely.

```
  LJ_truncate: select {'ON','OFF','NONE'} "OFF:normal LJ, ON:WCA, NONE: no-interaction
   ↪  at all"
```

The INIT_distribution structure specifies the initial configuration of particles.

```
INIT_distribution: {
  type: select {
    'uniform_random',
    'random_walk',
    'FCC',
    'BCC',
    'user_specify'
  }
  "uniform_random:distributed uniformly in box, random_walk:distributed uniformly in
   ↪  box, FCC:distributed on FCC lattice, BCC:distributed on BCC lattice,
   ↪  user_specify:configuration and velocity specified by user"
  random_walk: {
    iteration: int
  }
  user_specify: {
    Particles[]: Particle
  }
}
```

Allowed values for the `type` selector are `uniform_random` (random), `random_walk` (particles randomly moved from the sites of a square lattice), `FCC` (FCC lattice), `BCC` (BCC lattice), or `user_specify` (user-specified coordinates and velocities). For `random_walk`, `iteration` specifies the number of random-walk iterations. For `user_specify`, `Particles[]` sets the particle positions and velocities.

Similarly, the `INIT_orientation` selector describes the initial orientation of particles:

```
  INIT_orientation: select {'user_specify', 'random', 'space_align'}
```

The allowed values are `user_specify` (coordinates and velocities specified by user), `random` (random), or `space_align` (particles aligned with the lab-frace axes). For the case `user_specify`, initial particle orientations are specified via `Particles[]`.

`SLIP_tol` and `SLIP_iter` set the convergence criterion and maximum iteration count for iterative calculation of fluid-flow fields when introducing slip velocities in the tangential direction at Janus particle interfaces:

```
  SLIP_tol: float "Tolerance for iterative slip convergence"
  SLIP_iter: int "Maximum number of iterations for iterative slip convergence"
```

The `FIX_CELL` structure configures properties of the simulation cells:

```
FIX_CELL: {
  x: select{'ON','OFF'}"OFF:w/o DC current, ON:with DC current"
  y: select{'ON','OFF'}"OFF:w/o DC current, ON:with DC current"
  z: select{'ON','OFF'}"OFF:w/o DC current, ON:with DC current"
}
```

Set the `x` selector to `ON` to zero out the DC component of the total velocity in the *x* direction, or `OFF` to omit this adjustment (and similarly for the `y` and `z` selectors).

The `pin` structure configures particle degrees of freedom:

```
pin: {
  type: select{"NO","YES"}
  YES:{
    pin[]: int
    pin_rot[]: int
  }
}
```

Set the `type` selector to `YES` to fix ("pin") particle positions. In this case, the `pin[]` / `pin_rot[]` fields lists the indices of particles whose translational/rotational motion is to be prohibited.

The `free_rigid` structure configures individual degrees of freedom (DOFs) for the translational/rotational motion of rigid bodies. This should be used together with the `object_type.rigid.Rigid_spec` option, which will fix all six components of the linear and angular velocities. `free_rigid` can then be used to selectively free any of the DOFs.

```
free_rigid:{
  type: select{'NO', 'YES'}  "Free rigid degrees of freedom"
  YES:{
    DOF[]:{
      spec_id: int           "Rigid body species id"
      vel:{
        x:select{'NO', 'YES'},
        y:select{'NO', 'YES'},
        z:select{'NO', 'YES'}
      } "Free velocity components"
      omega:{
        x:select{'NO', 'YES'},
        y:select{'NO', 'YES'},
        z:select{'NO', 'YES'}
      } "Free omega components"
    }
  }
}
```

Set the `type` selector to `YES` to allow DOFs to be configured separately for each rigid body in the `DOF[]` field. In this case, `spec_id` specifies the index of a rigid body, and the `x`, `y`, and `z` selectors of the `vel` and `omega` structures may be set to `YES` (free) or `NO` (fixed) to enable or disable translational and rotational motion in the various Cartesian directions.

The `ns_solver` structure configures the method used to solve the Navier-Stokes equations (for shear-flow simulations under Lees-Edwards boundary conditions):

```
ns_solver:{
  OBL_INT: select {'linear', 'spline'} "interpolation scheme for Oblique/Rectangular
   ↪  transform"
}
```

The `OBL_INT` selector may be set to `linear` or `spline` to select the approximation method used for the coordinate transformations between the cartesian and oblique grids in the shear-flow simulations.

The `wall` structure specifies properties of planar walls:

```
wall:{
  type: select{'NONE', 'FLAT'}
  FLAT: {
    axis: select{'X', 'Y', 'Z'}"perpendicular axis to flat parallel walls"
    DH: int "wall thickness in number of grid points"
    LJ_Params: select{"AUTO", "MANUAL"}
    MANUAL:{
      truncate: select{'ON', 'OFF', 'NONE'} "Truncate OFF: attractive force, ON: no
       ↪  attractive force"
      powers: select{'12:6','24:12','36:18'} "type of LJ potential"
      EPSILON: double "LJ parameter, default (Basically, a large value.)"
    }
  }
}
```

Set the `type` selector to `FLAT` to specify planar walls. In this case, set the `axis` selector to `X`, `Y`, or `Z` to specify the normal direction to the planar walls. The `DH` field specifies the thickness of the planar walls, measured in units of the lattice spacing. The `LJ_Params` selector describes the potential acting at planar walls; the choice `LJ_Params=AUTO` sets the potential to be the same as the interparticle potential, while the choice `LJ_Params=MANUAL` allows a different potential to be specified. For the `MANUAL` case, the `truncate` selector specifies the presence or absence of attractive forces at planar walls. If set to `ON`, the potential is purely repulsive. If set to `OFF`, the potential is a sum of repulsive and attractive contributions. The `powers` selector may be set to `12:6`, `24:12`, or `36:18` to specify the exponents in the Lennard-Jones potential at planar walls, and `EPSILON` sets the energy unit for the Lennard-Jones potential at planar walls.

The `quincke` structure configures properties of Quincke rollers:

```
quincke:{
  type: select{'ON','OFF'}
  ON: {
    e_dir: select{'X', 'Y', 'Z'}"constraint axis(the direction of external electric
     ↪  field E)"
    w_dir: select{'X', 'Y', 'Z'}"the direction of constant angular velocity vector(body
     ↪  frame)"
    torque_amp: double "the amplitude of constraint torque"
  }
}
```

Set the `type` selector to `ON` to simulate Quincke rollers. In this case, set the `e_dir` selector to `X`, `Y`, or `Z` to specify the direction of the external electric field, set the `w_dir` selector to `X`, `Y`, or `Z` to specify the direction of the angular-velocity vector associated with rotation induced by the Quincke effect, and set `torque_amp` to the torque magnitude.

The `multipole` structure configures Ewald-method simulations:

```
multipole:{
  type: select{'ON', 'OFF'}
  ON:{
    Dipole:{
      type : select{'ON', 'OFF'}
      ON:{
        magnitude: double "the dipole strength"
        type     : select{'FIXED', 'QUINCKE'} "type of dipole"
        FIXED:{
          dir   : select{'X', 'Y', 'Z'} "direction of dipolar axis"
        }
        QUINCKE:{
          type : select{'with_mirror_image','no_mirror_image'} "Mirror image component
           ↪  of electrode surface"
          Pz_factor : double "Pz strength += Pz_factor*magnitude"
        }
      }
    }
    EwaldParams:{
      alpha   : double "Ewald screening parameter"
      delta   : double "Tolerance parameter, used to determine k_max"
      converge: double "Convergence parameter (fraction of k vectors to consider)"
      epsilon : double "Permittivity at boundary ( if negative, set to tinfoil)"
    }
  }
}
```

Set the type selector to ON to run simulations using the Ewald method. The Dipole structure describes properties of electric dipoles; set its type selector to ON to enable dipoles. In this case, magnitude sets the magnitude of the dipole moment, while the type selector may be set to FIXED for fixed dipoles or QUINCKE for Quincke rollers. For type=FIXED, dir may be set to X, Y, or Z to specify the direction of the fixed dipoles. For type=QUINCKE, the type selector may be set to with_mirror_image or no_mirror_image to enable or disable mirror-image components of the electrode surface, while Pz_factor is a parameter specifying the magnitude of the z-component of the dipole moment. Finally, the EwaldParams structure specifies Ewald-method parameters: alpha is the screening parameter, delta is a convergence criterion for determination of $k_{\max}$, converge is a convergence parameter, and epsilon is the dielectric permittivity of the surrounding medium (negative values are used to specify tin-foil boundary conditions, $\epsilon = \infty$).

A.5 Data output settings

The output structure contains fields that may be used to configure KAPSEL's data output settings.

```
output: {
  GTS: int "interval between snapshots"
  Num_snap: int "number of snapshots"
  AVS: select {"ON","OFF"}
  ON:{
    Out_dir: string "directory name"
    Out_name: string "prefix name for data file"
    FileType: select {"BINARY","ASCII","EXTENDED"} "output data type"
    EXTENDED:{
      Driver:{
        Format: select {"HDF5"}
      }
      Print_field:{
        Crop:select{"YES", "NO"}       "Crop Field Data to Hyperslab"
        YES:{
          Slab_x: SlabSelection
          Slab_y: SlabSelection
          Slab_z: SlabSelection
        }
        Vel: select{"YES", "NO"}       "Print velocity field"
        Phi: select{"YES", "NO"}       "Print phi field"
        Charge: select{"YES", "NO"}    "Print charge fields (surface & solute charge &
         ↪  potential)"
        Pressure: select{"YES", "NO"}  "Print pressure field"
        Tau: select{"YES", "NO"}       "Print stress tensor"
      }
    }
  }
  UDF: select {"ON","OFF"}
}
```

GTS sets the data output interval measured in number of steps. Num_snap sets the number of data outputs. Thus, the total number of timesteps in the simulation is given by GTS × Num_snap. Set the AVS selector to ON to enable data output in AVS format. For AVS=ON, the Out_dir and Out_name options set the directory in which AVS-format data output files are written and the filename prefix for those files. The format of AVS data output files is determined by the FileType selector, which may be set to Binary, ASCII, or EXTENDED. The choice EXTENDED specifies HDF5 as the extended data format in KAPSEL. The Print_field structure specifies properties of the output data fields. Set the Crop selector to YES to reduce the number of output data fields. Use Slab_x, Slab_y, and Slab_z to specify the reduction in the x, y, and z directions. Use start, stride, and count to set the first lattice-point index, the interval (number of lattice points) between lattice-point outputs, and the number of lattice points to output. Set the

`Vel` selector to `YES` to output values of velocity fields. Set the `Phi` selector to `YES` to output values of the SP function ϕ. Set the `Charge` selector to `YES` to output the charge density distribution (only applicable if the `type` selector for `constitutive_eq` is set to `Electrolyte`). Set the `Pressure` selector to `YES` to output values of pressure fields. (Currently not implemented). Set the `Tau` selector to `YES` to output values of the stress tensor. Finally, set the `UDF` selector to `ON` to enable UDF output.

A.6 Definition of UDF output data classes

The following UDF output data classes are defined in `define.udf`. As these are not configurable settings, we omit detailed descriptions.

```
\begin{def}
  class outParticle:{
    R:Vector3d [L],
    R_raw:Vector3d [L],
    v:Vector3d [L/tau],
    q:Quaternion,
    omega:Vector3d,
    f_hydro:Vector3d [mass*L*tau^-2],
    torque_hydro:Vector3d,
    f_r:Vector3d [mass*L*tau^-2],
    torque_r:Vector3d,
    f_slip:Vector3d [mass*L*tau^-2],
    torque_slip:Vector3d
  }
  E: float [epsilon] "total kinetic energy of the system"
  t: float "total time"
  Particles[]: outParticle
  RigidParticles[]: outParticle
  PSI[][][]: {
    psi: float
  }
\end{def}

\begin{def}
  class sParticle:{
    R:Vector3d [L],
    R_raw:Vector3d [L],
    v:Vector3d [L/tau],
    v_old:Vector3d [L/tau],
    f_hydro:Vector3d [mass*L*tau^-2],
    f_hydro_previous:Vector3d  [mass*L*tau^-2],
    f_hydro1:Vector3d  [mass*L*tau^-2],
    f_slip:Vector3d  [mass*L*tau^-2],
    f_slip_previous:Vector3d  [mass*L*tau^-2],
    fr:Vector3d  [mass*L*tau^-2],
    fr_previous:Vector3d  [mass*L*tau^-2],
    omega:Vector3d,
    omega_old:Vector3d,
    torque_hydro:Vector3d,
    torque_hydro_previous:Vector3d,
    torque_hydro1:Vector3d,
    torque_slip:Vector3d,
    torque_slip_previous:Vector3d,
    torque_r:Vector3d,
    torque_r_previous:Vector3d,
    q:Quaternion,
    q_old:Quaternion
  }
  class Matrix3d:{
    xx: float,
    xy: float,
    xz: float,
    yx: float,
    yy: float,
    yz: float,
    zx: float,
    zy: float,
    zz: float
  }
  class CTime:{
    ts:int
    time:float [tau]
  }
\end{def}
```

A.7 Restart settings

The `resume` structure contains settings that control restart settings for resuming calculations. The `Calculation` selector describes how to proceed from a previous calculation:

```
Calculation: select {
  'NEW',
  'CONTINUE',
  'CONTINUE_FDM',
  'CONTINUE_FDM_PHASE_SEPARATION'
} "flg in order to specify resumed simulation or not"
```

The choice `NEW` starts a new calculation. The remaining three choices (`CONTINUE`, `CONTINUE_FDM`, and `CONTINUE_FDM_PHASE_SEPARATI`
specifies that data saved upon termination of previous calculation should be read in and the calculation resumed. Which of the three `CONTINUE` options to use depends on the type of simulation you are running:

- Use `CONTINUE` for simulations using spectral methods, i.e., `constitutive_eq` set to `Navier_Stokes`, `Shear_Navier_Stokes`, `Shear_Navier_Stokes_Lees_Edwards`, or `Electrolyte`.
- Use `CONTINUE_FDM` for simulations of single-component fluids via finite-difference methods, i.e., `constitutive_eq` set to `Navier_Stokes_FDM` or `Shear_Navier_Stokes_Lees_Edwards_FDM`.
- Use `CONTINUE_FDM_PHASE_SEPARATION` for simulations of two-component fluids via finite-difference methods, i.e. `constitutive_eq` set to `Navier_Stokes_Cahn_Hilliard_FDM` or `Shear_NS_LE_CH_FDM`.

In each case, results needed to resume calculations are stored under `Saved_Data`:

```
CONTINUE:{
  Saved_Data:{
    jikan: CTime
    Particles[] : sParticle
    GR_body[]   : Vector3d
    GR_masses[] : float
    GR_moments_body[]: Matrix3d
    Zeta[][][]:{
      zeta0: float
      zeta1: float
    }
    uk_dc: Vector3d
    Concentration[][][][]: {ck:float}
    oblique: {
      degree_oblique: float
    }
  }
}
CONTINUE_FDM: {
  Saved_Data: {
    jikan: CTime
    Particles[]: sParticle
    GR_body[]: Vector3d
    GR_masses[]: float
    GR_moments_body[]: Matrix3d
    U[][][]: {
      u0: float
      u1: float
      u2: float
    }
    U_OLD[][][]: {
      u_old_0: float
      u_old_1: float
      u_old_2: float
    }
    oblique: {
      degree_oblique: float
    }
  }
}
CONTINUE_FDM_PHASE_SEPARATION: {
  Saved_Data: {
    jikan: CTime
    Particles[]: sParticle
    GR_body[]: Vector3d
    GR_masses[]: float
    GR_moments_body[]: Matrix3d
    U[][][]: {
      u0: float
      u1: float
      u2: float
    }
    U_OLD[][][]: {
      u_old_0: float
      u_old_1: float
      u_old_2: float
    }
    PSI[][][]: {
      psi: float
    }
    PSI_OLD[][][]: {
      psi_old: float
    }
    STRESS_OLD[][][]: {
      stress_old_0: float
      stress_old_1: float
      stress_old_2: float
    }
    oblique: {
      degree_oblique: float
    }
  }
}
```

As these settings are not configured manually, we omit detailed descriptions.

A.8 GOURMET display settings

The `Unit_Parameter` structure contains fields used to configure settings relevant for GOURMET displays. These settings do not affect simulations or simulation results.

```
Unit_Parameter:{
  Name: string            "Name"
  Comment:string          "Comment"
  Temperature:double  [K] "Temperature"
  Length:double [nm]      "Grid spacing"
  rho:double  [g/cm^3]    "Fluid density"
} "Parameters for unit conversion"
```

`Name` sets the name of the UDF file. `Comment` adds a comment to the UDF file. `Temperature` specifies a temperature unit used as a reference when simulation parameters are reported in actual units in GOURMET. `Length` and `rho` similarly specify length and density units.

Appendix B

Particle calculations

Particle configurations (positions, orientations, velocities, and angular velocities) are updated using a 2-step Adams-Bashforth scheme, with the exception of the initial step for which Euler's method is used. Denoting the dynamic variables of interest by $\boldsymbol{Y}_I$ and putting $\boldsymbol{Y}_I^n = \boldsymbol{Y}_I(t_n)$, we have

$$\boldsymbol{Y}^{n+1} = \boldsymbol{Y}^n + \frac{h}{2}\left(3\dot{\boldsymbol{Y}}^n - \dot{\boldsymbol{Y}}^{n-1}\right) \tag{B.1}$$

In the body-fixed (primary-axis) reference coordinate system (in which the moment-of-intertia tensor $\widetilde{\boldsymbol{I}}$ is diagonal), we solve Euler's equation in the form [8]

$$\begin{pmatrix} \dot{\widetilde{\omega}}^1 \\ \dot{\widetilde{\omega}}^2 \\ \dot{\widetilde{\omega}}^3 \end{pmatrix} = \begin{pmatrix} \widetilde{\tau}^1 + \widetilde{\omega}^2\widetilde{\omega}^3(\widetilde{I}^{22} - \widetilde{I}^{33}) \\ \widetilde{\tau}^2 + \widetilde{\omega}^3\widetilde{\omega}^1(\widetilde{I}^{33} - \widetilde{I}^{11}) \\ \widetilde{\tau}^3 + \widetilde{\omega}^1\widetilde{\omega}^2(\widetilde{I}^{11} - \widetilde{I}^{22}). \end{pmatrix} \tag{B.2}$$

For better precision in numerical calculations, particle orientations are updated using a rotational quaternion q, normalized at each timestep, instead of the rotation matrix R; the dynamical equation for the orientation then reads [62].

$$\dot{\mathsf{q}} = \frac{1}{2}\mathsf{w} \circ \mathsf{q} = \frac{1}{2}\mathsf{q} \circ \widetilde{\mathsf{w}} \tag{B.3}$$

Here $\circ$ denotes quaternion multiplication and the quantities $\widetilde{\mathsf{w}} = (0, \widetilde{\boldsymbol{\omega}})$ and $\omega = (0, \boldsymbol{\omega})$ are quaternions corresponding to the angular velocity in the laboratory system ($\boldsymbol{\omega}$) and in the body-fixed reference coordinate system ($\widetilde{\boldsymbol{\omega}} = R^t \cdot \boldsymbol{\omega}$).

Appendix C

Fluid calculations by spectral methods

As noted above in our discussion of simulation methods, KAPSEL uses the fractional-step method. To begin, this method determines $\boldsymbol{u}^*$ by time-evolving a modified version of the Navier-Stokes equations in which the advection and viscous-stress terms are present but the $\phi \boldsymbol{f}_p$ term is omitted. To simplify computations, KAPSEL does not solve directly for $\boldsymbol{u}$, but rather for $\boldsymbol{\omega} = \boldsymbol{\nabla} \times \boldsymbol{u}$. In a fixed Cartesian coordinate system with periodic boundary conditions, the vorticity equations satisfying the incompressibility condition ($\boldsymbol{\nabla} \cdot \boldsymbol{u} = 0$) read

$$\begin{aligned}\partial_t \boldsymbol{\omega} &= -\boldsymbol{\nabla} \times \boldsymbol{\nabla} \cdot (\boldsymbol{u}\boldsymbol{u}) + \rho^{-1} \boldsymbol{\nabla} \times \boldsymbol{\nabla} \cdot \boldsymbol{\sigma}, \\ \partial_t \widehat{\boldsymbol{\omega}} &= \boldsymbol{k} \times [\boldsymbol{k} \cdot \mathcal{F}_{\boldsymbol{k}}(\boldsymbol{u}\boldsymbol{u})] - \rho^{-1} \boldsymbol{k} \times [\boldsymbol{k} \cdot \widehat{\boldsymbol{\sigma}}]\end{aligned} \tag{C.1}$$

where $\mathcal{F}_{\boldsymbol{k}}(f) = \widehat{f}(\boldsymbol{k})$ denotes the Fourier transform and $\boldsymbol{k}$ is a wave-vector. The fluid velocity field is obtained from the definition of vorticity and the incompressibility condition:

$$\widehat{\boldsymbol{u}} = i \frac{\boldsymbol{k} \times \widehat{\boldsymbol{\omega}}}{|\boldsymbol{k}|^2} \tag{C.2}$$

The advantage of solving an equation for the vorticity $\boldsymbol{\omega}$ is that there is no need to solve the pressure Poisson equation to ensure incompressibility. Instead, incompressibility is satisfied via a Fourier-space projection of the form

$$\widehat{\boldsymbol{u}} \longrightarrow \left[\boldsymbol{I} - \frac{\boldsymbol{k}\boldsymbol{k}}{|\boldsymbol{k}|^2}\right] \cdot \widehat{\boldsymbol{u}}. \tag{C.3}$$

To reduce the amount of memory needed to store program code, KAPSEL further exploits the fact that the three vorticity components are not linearly independent, whereupon it suffices to consider only two components. Denoting by $\widehat{\boldsymbol{a}} = \mathcal{F}_{\boldsymbol{k}}(\boldsymbol{\nabla} \times \boldsymbol{A}) = i\boldsymbol{k} \times \widehat{\boldsymbol{A}}$ the Fourier transform of the curl of an arbitrary vector field $\boldsymbol{A}$, for all wavevectors (with $\boldsymbol{k} \neq 0$) we define an invertible map ($\dagger : C^3 \to C^2$) of the form

$$(\widehat{\boldsymbol{a}})^\dagger = (\widehat{a}_1, \widehat{a}_2, \widehat{a}_3)^\dagger : \begin{cases} (\widehat{a}_2, \widehat{a}_3) & k_1 \neq 0 \\ & \widehat{a}_1 = -(k_3 \widehat{a}_3 + k_2 \widehat{a}_2)/k_1 \\ (\widehat{a}_3, \widehat{a}_1) & k_1 = 0, k_2 \neq 0 \\ & \widehat{a}_2 = -k_3 \widehat{a}_3 / k_2 \\ (\widehat{a}_1, \widehat{a}_2) & k_1 = k_2 = 0, k_3 \neq 0 \\ & \widehat{a}_3 = 0 \end{cases} \tag{C.4}$$

Because $\widehat{\boldsymbol{a}}(\boldsymbol{k} = 0)$—which corresponds to a volume integral over the entire vector field—cannot be determined from $\widehat{\boldsymbol{a}}^\dagger$, we compute it separately. Applying this mapping to equations (C.1) yields a formula for $\widehat{\boldsymbol{\zeta}} = \widehat{\boldsymbol{\omega}}^\dagger$; this reduces by 1/3 the computational cost of updating the velocity field.

The form of the equations that are integrated depends on the stress tensor $\boldsymbol{\sigma}$ that is used. For a Newtonian fluid with $\boldsymbol{\nabla} \cdot \boldsymbol{\sigma} = \eta \nabla^2 \boldsymbol{u}$, the equation for the vorticity $\boldsymbol{\omega}$ reads

$$\partial_t \widehat{\boldsymbol{\omega}} = -\nu k^2 \widehat{\boldsymbol{\omega}} + \boldsymbol{k} \times [\boldsymbol{k} \cdot \mathcal{F}_{\boldsymbol{k}}(\boldsymbol{u}\boldsymbol{u})]. \tag{C.5}$$

We have expressed this in the form

$$\partial_t a = \mathcal{L} a - \mathcal{N}(t, a(t)) \tag{C.6}$$

where $\mathcal{L}$ is a time-independent linear operator and $\mathcal{N}$ is a nonlinear operator. The general solution of this equation is

$$a(t_n + h) = e^{\mathcal{L}h} a(t_n) - e^{\mathcal{L}h} \int_0^h \mathrm{d}\tau e^{-\mathcal{L}\tau} \mathcal{N}(t_n + \tau, a(t_n + \tau)) \tag{C.7}$$

and the linear portion may be solved exactly. The integral of the nonlinear portion may be approximated by various methods [63, 64]. In particular, the first-order approximation $\mathcal{N}(t_n + \tau, a(t_n + \tau)) = \mathcal{N}(t_n, a(t_n))$ yields

$$a(t_n + h) = e^{\mathcal{L}h} \left[a(t_n) - \mathcal{L}^{-1} \left(1 - e^{-h\mathcal{L}} \right) \mathcal{N}(t_n, a(t_n)) \right] \tag{C.8}$$

$$= a(t_n) + \left(e^{-h\mathcal{L}} - 1 \right) \left(a(t_n) + \mathcal{L}^{-1} \mathcal{N}(t_n, a(t_n)) \right) \tag{C.9}$$

which reduces to Euler's method in the limit $|\mathcal{L}| \to 0$.

Appendix D

Fluid calculations by finite-difference methods (FDM)

For simulations of particles dispersed in two-component phase-separated fluids, discussed in Section 6, KAPSEL uses finite-difference methods for fluid calculations. This appendix discusses the implementations of these methods.

D.1 Solving the Navier-Stokes equations

KAPSEL solves the Navier-Stokes (NS) equations (6.1) using explicit and implicit marker-and-cell (MAC) solvers [43, 46]. Variables are arranged on the computational mesh in a semi-staggered Arakawa-B lattice configuration, with velocity variables associated with lattice points and pressure variables associated with the centers of lattice cells [42]. The explicit and implicit MAC solvers are described in the following subsections.

D.1.1 Explicit MAC solver

First, from equation (6.1), omitting the term describing the volume force exerted by particles on fluids, we discretize the pressure term implicitly and the remaining terms explicitly:

$$\frac{\tilde{\boldsymbol{u}}^{n+1}-\boldsymbol{u}^n}{\Delta t}+(\boldsymbol{u}^n\cdot\boldsymbol{\nabla})\boldsymbol{u}^n+\frac{1}{\rho}\boldsymbol{\nabla}p^{n+1}-\nu\nabla^2\boldsymbol{u}^n+\frac{\psi^n}{\rho}\boldsymbol{\nabla}\mu_\psi^n+\frac{\phi^n}{\rho}\boldsymbol{\nabla}\mu_\phi^n=0. \tag{D.1}$$

Here quantities with superscript ${}^{n+1}$ are unknowns, while quantities with superscript ^{n} are known at step n. The tilde on the unknown flow velocity indicates that the velocity field obtained here is a *predicted* velocity field that must be corrected at a later stage. Assuming the continuity equation holds for the unknown flow velocity, we obtain the following Poisson equation for the pressure:

$$\frac{\Delta t}{\rho}\nabla^2 p^{n+1}=\boldsymbol{\nabla}\cdot\boldsymbol{u}^n-\Delta t\boldsymbol{\nabla}\cdot\left[(\boldsymbol{u}^n\cdot\boldsymbol{\nabla})\boldsymbol{u}^n-\nu\nabla^2\boldsymbol{u}^n+\frac{\psi^n}{\rho}\boldsymbol{\nabla}\mu_\psi^n+\frac{\phi^n}{\rho}\boldsymbol{\nabla}\mu_\phi^n\right]. \tag{D.2}$$

The first term on the right-hand side here is expected to vanish under ordinary circumstances, but we retain it for self-adjustment of cycle errors. Solving this Poisson equation yields the unknown pressure p^{n+1}, which we insert into (D.1) to obtain the unknown flow velocity $\tilde{\boldsymbol{u}}^{n+1}$. The value thus obtained for $\tilde{\boldsymbol{u}}^{n+1}$ fails to distinguish between sites inside particles and in fluid regions, and for this reason it cannot be used in its present form as the value for the next timestep. Instead, we modify the velocity field by applying a constraint force in particle regions to ensure that the velocity field in particle regions properly accounts for the motion of particles:

$$\boldsymbol{u}^{n+1}=\tilde{\boldsymbol{u}}^{n+1}+\phi\boldsymbol{f}_{\mathrm{p}}\Delta t. \tag{D.3}$$

The constraint force applied here is divergence-free, so the resulting velocity field satisfies the continuity equation. This yields the velocity field for the next step, $\boldsymbol{u}^{n+1}$.

D.1.2 Implicit MAC solver

By separating velocities from pressures and applying an implicit solver, we achieve greater computational stability than what is possible with an explicit solver alone, as we now explain [46].

First, we discretize the NS equation in the form

$$\frac{\tilde{\boldsymbol{u}}^{n+1}-\boldsymbol{u}^{n}}{\Delta t}+\boldsymbol{u}^{*}\cdot\boldsymbol{\nabla}\boldsymbol{u}^{n+\frac{1}{2}}+\frac{1}{\rho}\boldsymbol{\nabla}p^{n+1}-\boldsymbol{\nabla}\cdot\frac{\eta^{*}}{\rho}\left(\boldsymbol{\nabla}\boldsymbol{u}^{n+\frac{1}{2}}+\boldsymbol{\nabla}\boldsymbol{u}^{\mathrm{T}n+\frac{1}{2}}\right)+\frac{\psi^{*}}{\rho}\boldsymbol{\nabla}\mu_{p}si^{*}+\frac{\phi^{n+1}}{\rho}\boldsymbol{\nabla}\mu_{\phi}^{*}=0. \tag{D.4}$$

Here quantities with superscript $^{n+1/2}$ are unknowns, at timestep $n+1/2$, and are determined by a Crank-Nicolson scheme, while quantities with superscript * are known quantities, extrapolated using an Adams-Bashforth (AB) method. For example, flow velocities are given by

$$\boldsymbol{u}^{n+\frac{1}{2}}=\frac{1}{2}(\boldsymbol{u}^{n}+\tilde{\boldsymbol{u}}^{n+1}), \tag{D.5}$$

$$\boldsymbol{u}^{*}=\frac{1}{2}(3\boldsymbol{u}^{n}-\boldsymbol{u}^{n-1}). \tag{D.6}$$

Also, here the viscosity η is a position-dependent variable. If the continuity equation holds for the unknown flow velocity $\boldsymbol{u}^{n+1}$ in equation (D.4), we obtain the following Poisson equation for the pressure:

$$\frac{\Delta t}{\rho}\nabla^{2}p^{n+1}=\boldsymbol{\nabla}\cdot\boldsymbol{u}^{n}-\Delta t\boldsymbol{\nabla}\cdot\left[\boldsymbol{u}^{*}\cdot\boldsymbol{\nabla}\boldsymbol{u}^{*}-\boldsymbol{\nabla}\cdot\frac{\eta^{*}}{\rho}\left(\boldsymbol{\nabla}\boldsymbol{u}^{*}+\boldsymbol{\nabla}\boldsymbol{u}^{\mathrm{T}*}\right)+\frac{\psi^{*}}{\rho}\boldsymbol{\nabla}\mu_{p}si^{*}+\frac{\phi^{n+1}}{\rho}\boldsymbol{\nabla}\mu_{\phi}^{*}\right]. \tag{D.7}$$

Note that all unknown terms on the right-hand side here are replaced by known terms obtained via the AB method, so this Poisson equation may be solved independently. The pressure thus obtained is inserted into equation (D.4) to yield a simultaneous system of linear equations that may be solved for the unknown flow velocity. By default, KAPSEL solves this system using BiCGSTAB [47] with no preprocessing; however, as discussed in Appendix E, the parallel iterative solver library Lis may be used as an alternative to allow various preprocessing steps and various iterative solvers.

D.1.3 Solving the Navier-Stokes equations in the presence of shear flow with Lees-Edwards boundary conditions

The combination of explicit and implicit MAC solvers may also be used for computations involving shear flow with Lees-Edwards boundary conditions imposed via coordinate transformations based on tensor analysis [23], as we now discuss. This method is described in detail in Section 4. Consider a shear flow with shear velocity $\dot{\gamma}(t)$ imposed in the $\boldsymbol{e}_x$ direction, where $\boldsymbol{e}_x$ belongs to a set of basis vectors for an orthogonal coordinate system. Then the shear velocity at coordinate y along the $\boldsymbol{e}_y$ axis—the direction of the shear-flow gradient—is given by $\boldsymbol{U}=\dot{\gamma}(t)y\boldsymbol{e}_x$. Next, for a shear flow subject to these conditions we consider transforming to an oblique coordinate system (covariant basis) that varies in time as the system evolves. The basis vectors for this oblique coordinate system are described in the text preceding equation (4.2). For KAPSEL simulations in the presence of shear flows with Lees-Edwards boundary conditions, working in the oblique coordinate system we subtract the shear-flow contribution from the flow velocity to yield $\hat{\boldsymbol{\xi}}=\boldsymbol{u}-\boldsymbol{U}$, then solve an NS equation for the contravariant components $\hat{\xi}^i$:

$$\partial_t\hat{\xi}^i+\hat{\xi}^j\hat{\partial}_j\hat{\xi}^i=-\rho^{-1}G^{ij}\hat{\partial}_j\hat{p}+\nu G^{jk}\hat{\partial}_j\hat{\partial}_k\hat{\xi}^i-2\dot{\gamma}\hat{\xi}^2\delta_{i1}, \tag{D.8}$$

$$\hat{\partial}_j\hat{\xi}^j=0. \tag{D.9}$$

where $\hat{p}$ is the pressure defined in the oblique coordinate system. In equations (D.8) and (D.9) we have adopted the Einstein summation convention; also, ∂_t denotes differentiation with respect to time, while $\hat{\partial}_i$ denotes spatial derivatives in the oblique coordinate system (covariant basis), i.e. $\hat{\partial}_i=\partial/\partial\hat{x}^i$, and G^{ij} are the contravariant components of the metric tensor. Then for the shear flow we have [23]

$$G^{ij}=\begin{pmatrix}1+(\dot{\gamma}t)^2 & -\dot{\gamma}t & 0\\ -\dot{\gamma}t & 1 & 0\\ 0 & 0 & 1\end{pmatrix} \tag{D.10}$$

KAPSEL solves for equations (D.8) and (D.9) via the same procedure used for computations in orthogonal coordinate systems, discretizing time and applying explicit and implicit MAC solvers in the oblique coordinate system; we omit further details.

D.2 Solving the Cahn-Hilliard equation

We next describe KAPSEL's strategy for solving the Cahn-Hilliard (CH) equation given by equation (6.2); as in the cases discussed above, KAPSEL implements both explicit and quasi-implicit solvers for this purpose.

D.2.1 Explicit solver

KAPSEL implements Euler's method as an explicit solver for the CH equation:

$$\frac{\psi^{n+1}-\psi^{n}}{\Delta t}+\boldsymbol{\nabla}\cdot(\psi^{n}\boldsymbol{u}^{n})=\kappa\nabla^{2}\left[f'(\psi^{n})-\alpha\nabla^{2}\psi^{n}+w\xi|\boldsymbol{\nabla}\phi^{n+1}|^{2}+2d\phi^{n+1}(\psi^{n}-\bar{\psi})-2z\xi_{\psi}\nabla\phi^{n+1}\cdot\nabla\psi^{n}\right]. \tag{D.11}$$

Here we have used the conservative form of the convection term in the CH equation to ensure good conservation behavior for ψ. Note that, in this equation, the quantity ϕ^{n+1}, denoting the value of ϕ on the next timestep, is a *known* quantity because the weak-coupling solver evolves ϕ before evolving ψ.

D.2.2 Implicit solver

Numerical approaches to the CH equation involve as many as 4 layers of partial-derivative operators, and are thus prone to computational instability in general. KAPSEL addresses this difficulty by implementing the following quasi-implicit solution strategy [49]:

$$\begin{aligned}\frac{3\psi^{n+1}-4\psi^{n}+\psi^{n-1}}{2\Delta t}+\boldsymbol{\nabla}\cdot(\psi^{n+1}\boldsymbol{u}^{n+1})=\kappa\nabla^{2}\Big[&2f'(\psi^{n})-f'(\psi^{n-1})-\alpha\nabla^{2}\psi^{n+1}+w\xi|\boldsymbol{\nabla}\phi^{n+1}|^{2}+2d\phi^{n+1}(\psi^{n+1}-\bar{\psi})\\&-2z\xi_{\psi}\nabla\phi^{n+1}\cdot\nabla\psi^{n+1}\Big].\end{aligned} \tag{D.12}$$

This formulation uses a backward-difference scheme for time-derivative terms, and also performs an implicit solve by using values extrapolated via the AB method to linearize the nonlinear potential term on the right-hand side. By default, KAPSEL uses BiCGSTAB with no preprocessing to solve the resulting simultaneous system of linear equations; however, as discussed in Appendix E, the parallel iterative solver library Lis may be used as an alternative to allow various preprocessing steps and various iterative solvers.

D.2.3 Solving the Cahn-Hilliard equation in the presence of shear flow with Lees-Edwards boundary conditions

The CH equation in the oblique coordinate system reads

$$\partial_t\hat{\psi}+\hat{\partial}_i(\hat{u}^i\hat{\psi})=\kappa G^{ij}\hat{\partial}_i\hat{\partial}_j\left\{f'(\hat{\psi})-\alpha G^{kl}\hat{\partial}_k\hat{\partial}_l\hat{\psi}+w\xi G^{kl}\hat{\partial}_k\hat{\phi}\hat{\partial}_l\hat{\phi}+2d\hat{\phi}(\hat{\psi}-\bar{\psi})-2z\xi_{\psi}G^{kl}[\hat{\partial}_k\hat{\phi}\hat{\partial}_l\hat{\psi}+\hat{\phi}\hat{\partial}_k\hat{\partial}_l\hat{\psi}]\right\} \tag{D.13}$$

where $\hat{\phi}$ and $\hat{\psi}$ are the SP functions, defined in the oblique coordinate system, that respectively identify particle-fluid interfaces and fluid-fluid interfaces . As in the cases discussed above, both explicit and quasi-implicit solvers are implemented for the oblique coordinate system.

Appendix E

Interfacing with the Lis library

In KAPSEL the default method for solving simultaneous systems of linear equations in finite-difference method (FDM) calculations is BiCGSTAB [47]. BiCGSTAB is a stable and high-speed algorithm that is widely used for fluid calculations and is the best choice for almost all KAPSEL calculations, but the possibility that this method may—in extremely rare cases—fail to achieve successful computations cannot be entirely excluded. To allow BiCGSTAB to be replaced by more robust methods (such as GMRES [65]) in such situations, KAPSEL implements the ability to interface with the Lis [48] library for general-purpose iterative solvers. When building the KAPSEL executable file, setting the `LIS` option in the `Makefile` to `ON` will automatically select Lis as the default choice of implicit solver for the Navier-Stokes (NS) or Cahn-Hilliard (CH) equations; this enables a variety of combinations of preprocessing techniques and iterative solvers. (Note that KAPSEL must be recompiled to use Lis.)

Preprocessing steps and iterative solvers are configured within the program. The `Init_lis` function in the source file `fdm_matrix_solver.cxx` refers to `lis_solver_set_option` in two places, corresponding to implicit-solver settings for the NS and CH equations respectively. Lis settings are briefly discussed in comments at relevant places in the code; for further details, consult the freely available Lis user manual [66].

Appendix F

Compiling KAPSEL

KAPSEL version 5 is distributed in the form of binary executable files, so there is no need for users to compile the code from source. For the benefit of developers interested in the compilation process, this appendix briefly discusses the procedure for compiling KAPSEL. The following description assumes that you are working as the root user. When working with a user account that has administrative rights, use the `sudo` command as appropriate. The most recent information may be found at the KAPSEL website: `https://kapsel-dns.com`.

F.1 The KAPSEL developer's environment

The procedure for compiling and running KAPSEL depends slightly on whether you are running on a Windows(+Cygwin), Linux, or Mac system:

Linux

If you are using KAPSEL on a Linux system, you may skip to section F.2 below.

Windows

To use KAPSEL on a Windows system, you must first install the Cygwin[1] package. In the Select Package section, set View "Category" and be sure that your Cygwin installation includes the following Packages.

- all Packages in the `Devel` category
- all Packages related to `fftw3` in the `Libs` category
- all Packages related to `hdf5` in the `Libs` category
- `python3` Package in the `Python` category

After installing Cygwin, skip to section F.2 below.

Mac

To use KAPSEL on MacOS, you must first install Xcode and the command line tools. By default, the `gcc` command invokes the `clang` compiler; to extract maximal performance from your hardware it is best to install `gcc-12` explicitly using Homebrew.

F.2 Installing OCTA

For tasks such as processing input parameters and visualizing output data, KAPSEL relies on an external user-interface module known as Gourmet. Gourmet is distributed as an internal component of the open-source package OCTA (a universal simulator for soft materials), which must be installed before using KAPSEL.

To install OCTA, simply visit `http://octa.jp/` to download and run an appropriate installer for your system.[2] The instructions below assume that `OCTA8.#` has been installed in `/usr/local/OCTA8#` (for Linux or Mac) or in `C:\OCTA8.#` (for Windows).[3]

[1]http://cygwin.com/

[2]Answers to questions regarding OCTA/GOURMET are available upon registering as a member of the OCTA-BBS website.

[3]Here the pound symbol ("#") is to be replaced with numerals to indicate the version of your OCTA installation.

F.3 Building `libplatform`

`libplatform` is an I/O library used by OCTA to access data stored in UDF files. Depending on your environment, building `libplatform` may require installing additional software tools such as OpenJDK[4] or Python3[5].

Windows

Log on to Windows with administrator privileges, open a Cygwin terminal window, and execute the following commands.

```
$ ln -s /cygdrive/c/OCTA8.# /usr/local/OCTA8#
$ cd /usr/local/OCTA8#/GOURMET/src
$ make distclean
$ ./configure --with-python
$ make
$ make install
```

Linux

- If using gcc:[6]

```
$ cd /usr/local/OCTA8#/GOURMET/src
$ make distclean
$ ./configure --with-python
$ make
$ make install
$ mv ../lib/linux64/libplatform.a ../lib/linux64/libplatform_gcc.a
```

- If using icc:[7]

```
$ cd /usr/local/OCTA8#/GOURMET/src
$ make distclean
$ ./configure CC=icc CXX=icpc --with-python
$ make
$ make install
$ mv ../lib/linux64/libplatform.a ../lib/linux64/libplatform_icc.a
```

Mac

The build process varies depending on the compiler you are using.

- If using clang:[8]

```
$ cd /usr/local/OCTA8#/GOURMET/src
$ make distclean
$ ./configure --with-python
$ make
$ make install
$ mv ../lib/macosx/libplatform.a ../lib/macosx/libplatform_clang.a
```

- If using gcc:[9]

```
$ cd /usr/local/OCTA8#/GOURMET/src
$ make distclean
$ ./configure CC=gcc-12 CXX=g++-12 --with-python
$ make
$ make install
$ mv ../lib/macosx/libplatform.a ../lib/macosx/libplatform_gcc.a
```

[4]AdoptOpenJDK: https://adoptopenjdk.net / For arm64 (Apple silicon): https://www.azul.com/downloads/
[5]Anaconda: https://www.anaconda.com/
[6]gcc may be installed by running `brew install gcc-12` or a similar command.
[7]The Intel C compiler, available as part of Intel oneAPI Toolkits
[8]clang is the default C compiler on macOS
[9]gcc may be installed by running `brew install gcc-12` or a similar command.

After building `libplatform`, there is a possibility that GOURMET will not run correctly for one reason or another. In this case, execute one of the the following commands:

```
$ cd /usr/local/OCTA8#/GOURMET
$ ./Make-All.sh
```

```
$ cd /usr/local/OCTA8#/GOURMET/src
$ ./build-gourmet
```

For further details regarding `libplatform`, consult the OCTA manual.

F.4 Installing FFTW

Windows

The source code must be downloaded and installed manually.

```
$ ./configure --prefix=/opt/fftw/latest.gcc CFLAGS="-O3" FFLAGS="-O3"
↪ --enable-openmp --enable-threads --enable-shared --disable-fortran
$ make
$ make install
```

Linux

The source code must be downloaded and installed manually.

- If using gcc:

```
$ ./configure --prefix=/opt/fftw/latest.gcc CFLAGS="-O3" FFLAGS="-O3"
↪ --enable-openmp --enable-threads --enable-shared --disable-fortran
$ make
$ make install
```

- If using icc:

```
$ ./configure --prefix=/opt/fftw/latest.gcc CC=icc CXX=icpc CFLAGS="-O3"
↪ FFLAGS="-O3" --enable-openmp --enable-threads --enable-shared --disable-fortran
$ make
$ make install
```

Mac

- If using clang:

```
brew install fftw
```

- If using gcc:

```
$ ./configure --prefix=/opt/fftw/latest.gcc CC=gcc-12 CXX=g++-12 CFLAGS="-O3"
↪ FFLAGS="-O3" --enable-openmp --enable-threads --enable-shared --disable-fortran
$ make
$ make install
```

F.5 Installing HDF5

Windows

The source code must be downloaded and installed manually.

```
$ ./configure --prefix=/opt/hdf5/latest.gcc CFLAGS="-O3" FFLAGS="-O3"
↪ --enable-fortran --enable-cxx
$ make
$ make install
```

Linux
The source code must be downloaded and installed manually.

- If using gcc:

  ```
  $ ./configure --prefix=/opt/hdf5/latest.gcc CFLAGS="-O3" FFLAGS="-O3"
  ↪ --enable-fortran --enable-cxx
  $ make
  $ make install
  ```

- If using icc:

  ```
  $ ./configure --prefix=/opt/hdf5/latest.icc CC=icc CXX=icpc CFLAGS="-O3"
  ↪ FFLAGS="-O3" --enable-fortran --enable-cxx
  $ make
  $ make install
  ```

Mac

- If using clang:

  ```
  $ brew install hdf5
  ```

- If using gcc:

  ```
  $ ./configure --prefix=/opt/hdf5/latest.gcc CC=gcc-12 CXX=g++-12
  ↪ --enable-fortran --enable-cxx
  $ make
  $ make install
  ```

F.6 Installing LIS (optional)

Windows
The source code must be downloaded and installed manually.

```
$ $./configure --prefix=/opt/lis/latest.gcc
$ make
$ make install
```

Linux
The source code must be downloaded and installed manually.

- If using gcc:

  ```
  $ $./configure --prefix=/opt/lis/latest.gcc
  $ make
  $ make install
  ```

- If using icc:

  ```
  $ ./configure --prefix=/opt/lis/latest.icc CC=icc CXX=icpc
  $ make
  $ make install
  ```

Mac

The source code must be downloaded and installed manually.

- If using clang :

```
$ ./configure --prefix=/opt/lis/latest.clang CC=clang CXX=clang++
$ make
$ make install
```

- If using gcc:

```
$ ./configure --prefix=/opt/lis/latest.gcc CC=gcc-12 CXX=g++-12
$ make
$ make install
```

F.7 Building the KAPSEL executable file

Download the latest version of the KAPSEL source-code archive file `kapsel#.#.zip`[10] and uncompress the archive:

```
$ unzip kapsel#.#.zip
$ cd kapsel#.#/src
```

To ensure that the library file `libplatform.a` built in Section F.3 is properly linked, modify the `GOURMET_HOME_PATH` variable in your `Makefile`. Depending on your environment, it may also be necessary to modify the `-I` (search path for include files) and/or `-L` (search path for library files) flags. Once you have correctly configured all paths, delete temporary working files.

```
$ make clean
```

Then select and execute one of the following `make` commands to build the KAPSEL binary executable file appropriate for your computing environment:

Windows

Use Cygwin:

```
$ make ENV=CYGWIN
$ make ENV=CYGWIN FFT=FFTW
$ make ENV=CYGWIN_OMP FFT=FFTW
```

Linux

- If using gcc:

```
$ make ENV=GCC
$ make ENV=GCC FFT=FFTW
$ make ENV=GCC_OMP FFT=FFTW
```

- If using icc:

```
$ make ENV=ICC
$ make ENV=ICC FFT=IMKL
$ make ENV=ICC_OMP FFT=IMKL
```

Mac

- If using clang:

[10] Here the pound symbol (“#”) is to be replaced with numerals to indicate the version of your KAPSEL installation.

```
$ make ENV=CLANG
$ make ENV=CLANG FFT=FFTW
$ make ENV=CLANG_OMP FFT=FFTW
```

- If using gcc:

```
$ make ENV=GCC_MAC
$ make ENV=GCC_MAC FFT=FFTW
$ make ENV=GCC_MAC_OMP FFT=FFTW
```

All platforms

Copy the executable `kapsel` to `bin`, and create a symbolic link in the KAPSELinstall directory.

```
$ cd ..
$ cp ./src/kapsel ./bin/kapsel_self_made
$ ln -s ./bin/kapsel_self_made ./kapsel
```

Set appropriate environment variables, and run KAPSEL.

```
$ export DYLD_LIBRARY_PATH = "/opt/fftw/latest.gcc/lib: /opt/hdf5/latest.gcc/lib:
↪ /opt/lis/latest.gcc/lib: DYLD_LIBRARY_PATH"
$ ./kapsel
Usage:
> ./kapsel -I[input UDF] -O[output UDF] -D[define UDF] -R[restart UDF]
```

Acknowledgements

This work was supported by JST PRESTO, JST-CREST, Grant-in-Aid for Scientific Research, New Energy and Industrial Technology Development Organisation (NEDO), and by the National Institute of Advanced Industrial Science and Technology (AIST), Kyoto University and Kyushu University.

References

[1] R. Yamamoto, J. J. Molina, and Y. Nakayama, "Smoothed profile method for direct numerical simulations of hydrodynamically interacting particles", Soft Matter **17**, 4226 (2021).

[2] J. N. Israelachvili, *Intermolecular and Surface Forces, 3rd Edition* (Academic Press, 2011).

[3] J. R. Blake, "A spherical envelope approach to ciliary propulsion", Journal of Fluid Mechanics **46**, 199 (1971).

[4] J. J. Molina, Y. Nakayama, and R. Yamamoto, "Hydrodynamic interactions of self-propelled swimmers", Soft Matter **9**, 4923 (2013).

[5] Y. Nakayama and R. Yamamoto, "Simulation method to resolve hydrodynamic interactions in colloidal dispersions", Physical Review E **71**, 036707 (2005).

[6] Y. Nakayama, K. Kim, and R. Yamamoto, "Simulating (electro)hydrodynamic effects in colloidal dispersions: Smoothed profile method", The European Physical Journal E **26**, 361 (2008).

[7] J. J. Molina and R. Yamamoto, "Direct numerical simulations of rigid body dispersions. I. Mobility/friction tensors of assemblies of spheres", The Journal of Chemical Physics **139**, 234105 (2013).

[8] J. V. José and E. J. Saletan, *Classical Dynamics: A Contemporary Approach* (Cambridge University Press, Aug. 1998).

[9] H. Tanaka and T. Araki, "Simulation Method of Colloidal Suspensions with Hydrodynamic Interactions: Fluid Particle Dynamics", Physical Review Letters **85**, 1338 (2000).

[10] R. Yamamoto, "Simulating Particlçe Dispersions in Nematic Liquid-Crystal Solvents", Physical Review Letters **87**, 075502 (2001).

[11] X. Luo, M. R. Maxey, and G. E. Karniadakis, "Smoothed profile method for particulate flows: Error analysis and simulations", Journal of Computational Physics **228**, 1750 (2009).

[12] T. Iwashita, Y. Nakayama, and R. Yamamoto, "A numerical model for brownian particles fluctuating in incompressible fluids", Journal of the Physical Society of Japan **77**, `10.1143/JPSJ.77.074007` (2008).

[13] T. Iwashita, Y. Nakayama, and R. Yamamoto, "Velocity autocorrelation function of fluctuating particles in incompressible fluids: toward direct numerical simulation of particle dispersions", Progress of Theoretical Physics Supplement **178**, `10.1143/PTPS.178.86` (2008).

[14] T. Iwashita and R. Yamamoto, "Short-time motion of brownian particles in a shear flow", Physical review. E, Statistical, nonlinear, and soft matter physics **79**, 031401 (2009).

[15] D. A. S. W. B. Russel and W. R. Schowalter, *Colloidal dispersions* (Cambridge University Press, Cambridge, England, 1989).

[16] Y. Otsubo, "The present status and prospect of suspension rheology", Nihon Reoroji Gakkaishi **31**, 15 (2003).

[17] T. Matsumoto, "Rheology of colloidal disperse systems", Nihon Reoroji Gakkaishi **32**, 3 (2004).

[18] I. M. Krieger, "Rheology of monodisperse latices", Advances in Colloid and Interface Science **3**, 111 (1972).

[19] C.-H. Hsueh and P. Becher, "Effective viscosity of suspensions of spheres", Journal of the American Ceramic Society **88**, 1046 (2005).

[20] A. Onuki, "Phase transitions of fluids in shear flow", Journal of Physics: Condensed Matter **9**, 6119 (1997).

[21] S. Toh, K. Ohkitani, and M. Yamada, "Enstrophy and momentum fluxes in two-dimensional shear flow turbulence", Physica D: Nonlinear Phenomena **51**, 569 (1991).

[22] H. Kobayashi and R. Yamamoto, "Implementation of Lees–Edwards periodic boundary conditions for direct numerical simulations of particle dispersions under shear flow", The Journal of Chemical Physics **134**, 064110 (2011).

[23] H. Kobayashi and R. Yamamoto, “Implementation of lees–edwards periodic boundary conditions for direct numerical simulations of particle dispersions under shear flow”, The Journal of Chemical Physics **134**, 064110 (2011).

[24] K. Kim and R. Yamamoto, “Efficient simulations of charged colloidal dispersions: a density functional approach”, Macromolecular Theory and Simulations **14**, 278 (2005).

[25] K. Kim, Y. Nakayama, and R. Yamamoto, “Direct numerical simulations of electrophoresis of charged colloids”, Phys. Rev. Lett. **96**, 208302 (2006).

[26] H. Tanaka and T. Araki, “Simulation method of colloidal suspensions with hydrodynamic interactions: fluid particle dynamics”, Phys. Rev. Lett. **85**, 1338 (2000).

[27] T. Kajishima, S. Takiguchi, H. Hamasaki, and Y. Miyake, “Turbulence structure of particle-laden flow in a vertical plane channel due to vortex shedding”, JSME International Journal Series B Fluids and Thermal Engineering **44**, 526 (2001).

[28] J.-L. Barrat and J.-P. Hansen, *Basic concepts for simple and complex liquids* (Mar. 2003).

[29] *A Book in Japanese* (1995).

[30] H. Ohshima, T. W. Healy, and L. R. White, “Accurate analytic expressions for the surface charge density/surface potential relationship and double-layer potential distribution for a spherical colloidal particle”, Journal of Colloid and Interface Science **90**, 17 (1982).

[31] R. O’Brien and L. White, “Electrophoretic mobility of a spherical colloidal particle”, Journal of The Chemical Society, Faraday Transactions 2 **74**, 1607 (1978).

[32] H. Ohshima, T. W. Healy, and L. R. White, “Approximate analytic expressions for the electrophoretic mobility of spherical colloidal particles and the conductivity of their dilute suspensions”, J. Chem. Soc., Faraday Trans. 2 **79**, 1613 (1983).

[33] *A Book in Japanese* (2004).

[34] W. Ramsden, “Separation of solids in the surface-layers of solutions and ‘suspensions’(observations on surface-membranes, bubbles, emulsions, and mechanical coagulation).—preliminary account”, Proceedings of the royal Society of London **72**, 156 (1904).

[35] S. U. Pickering, “CXCVI.-Emulsions”, Journal of the Chemical Society, Transactions **91**, 2001 (1907).

[36] F. Fenouillot, P. Cassagnau, and J.-C. Majesté, “Uneven distribution of nanoparticles in immiscible fluids: morphology development in polymer blends”, Polymer **50**, 1333 (2009).

[37] Y. Yang, Z. Fang, X. Chen, W. Zhang, Y. Xie, Y. Chen, Z. Liu, and W. Yuan, “An overview of pickering emulsions: solid-particle materials, classification, morphology, and applications”, Frontiers in pharmacology **8**, 287 (2017).

[38] M. E. Cates and P. S. Clegg, “Bijels: a new class of soft materials”, Soft Matter **4**, 2132 (2008).

[39] M. N. Lee and A. Mohraz, “Bicontinuous macroporous materials from bijel templates”, Advanced Materials **22**, 4836 (2010).

[40] P. C. Hohenberg and B. I. Halperin, “Theory of dynamic critical phenomena”, Reviews of Modern Physics **49**, 435 (1977).

[41] M. Doi, *Soft Matter Physics* (Oxford University Press, 2013).

[42] A. Arakawa and V. R. Lamb, “Computational design of the basic dynamical processes of the ucla general circulation model”, General circulation models of the atmosphere **17**, 173 (1977).

[43] F. H. Harlow and J. E. Welch, “Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface”, The physics of fluids **8**, 2182 (1965).

[44] J. H. Ferziger and M. Peric, *Computational methods for fluid dynamics* (Springer Science & Business Media, 2012).

[45] D. Jacqmin, “Calculation of two-phase navier–stokes flows using phase-field modeling”, Journal of Computational Physics **155**, 96 (1999).

[46] A. Maruoka, J. Matsumoto, and M. Kawahara, “A fractional step finite element method for incompressible navier–stokes equations using quadrilateral scaled bubble function”, Journal of Structural Engineering A **44**, 383 (1998).

[47] H. A. Van der Vorst, “Bi-cgstab: a fast and smoothly converging variant of bi-cg for the solution of nonsymmetric linear systems”, SIAM Journal on scientific and Statistical Computing **13**, 631 (1992).

[48] *Lis: Library of Iterative Solvers for Linear Systems*, `https://www.ssisc.org/lis/index.ja.html`, Accessed: 2019-01-28.

[49] Y. He, Y. Liu, and T. Tang, "On large time-stepping methods for the Cahn–Hilliard equation", Applied Numerical Mathematics **57**, 616 (2007).

[50] *ParaView Documentation*, `https://docs.paraview.org/en/latest/`, Accessed: 2021-09-30.

[51] C. Domb, *Phase transitions and critical phenomena* (Elsevier, 2000).

[52] M. C. Marchetti, J. F. Joanny, S. Ramaswamy, T. B. Liverpool, J. Prost, M. Rao, and R. A. Simha, "Hydrodynamics of soft active matter", Reviews of Modern Physics **85**, 1143 (2013).

[53] M. J. Lighthill, "Hydromechanics of Aquatic Animal Propulsion", Annual Review of Fluid Mechanics **1**, 413 (1969).

[54] J. R. Blake, "A spherical envelope approach to ciliary propulsion", Journal of Fluid Mechanics **46**, 199 (1971).

[55] O. S. Pak and E. Lauga, "Generalized squirming motion of a sphere", Journal of Engineering Mathematics **88**, 1 (2014).

[56] N. Oyama, J. J. Molina, and R. Yamamoto, "Simulations of Model Microswimmers with Fully Resolved Hydrodynamics", Journal of the Physical Society of Japan **86**, 101008 (2017).

[57] A. Bricard, J.-B. Caussin, N. Desreumaux, O. Dauchot, and D. Bartolo, "Emergence of macroscopic directed motion in populations of motile colloids", Nature **503**, 95 (2013).

[58] G. Quincke, "Ueber rotationen im constanten electrischen felde", Annalen der Physik **295**, 417 (1896).

[59] A. Mauleon-Amieva, M. Mosayebi, J. E. Hallett, F. Turci, T. B. Liverpool, J. S. Van Duijneveldt, and C. P. Royall, "Competing active and passive interactions drive amoebalike crystallites and ordered bands in active colloids", Physical Review E **102**, 032609 (2020).

[60] A. J. Goldman, R. G. Cox, and H. Brenner, "Slow viscous motion of a sphere parallel to a plane wall—i motion through a quiescent fluid", Chemical engineering science **22**, 637 (1967).

[61] B. Cichocki and R. Jones, "Image representation of a spherical particle near a hard wall", Physica A: Statistical Mechanics and its Applications **258**, 273 (1998).

[62] M. P. Allen and D. J. Tildesley, *Computer Simulation of Liquids* (Oxford Science Publications, 1987).

[63] S. M. Cox and P. C. Matthews, "Exponential time differencing for stiff systems", Journal of Computational Physics **176**, 430 (2002).

[64] M. Hochbruck and A. Ostermann, "Exponential integrators", Acta Numerica **19**, 209 (2010).

[65] Y. Saad and M. H. Schultz, "Gmres: a generalized minimal residual algorithm for solving nonsymmetric linear systems", SIAM Journal on scientific and statistical computing **7**, 856 (1986).

[66] *Lis User Guide*, `https://www.ssisc.org/lis/lis-ug-en.pdf`, Accessed: 2019-01-28.