MaLI and DEcalioc Documentation

A guide to calibrating, submitting and obtaining results for a LIGGGHTS DEM simulation run on the Hypnos computing cluster

Tim Churchfield

December 10, 2019

Version 2.0

Contents

List of Figures

List of Tables

1 Introduction

The purpose of this document is to provide the necessary information to successfully calibrate a LIGGGHTS discrete element method (DEM) simulation, submit a simulation to the computing cluster (Hypnos) and obtain the final results in a usable format. Section 2 explains how to use the PuTTY program for communicating with the cluster from a computer running Microsoft Windows and Section 3 focuses on the installation and basic set up of LIGGGHTS, GNU Octave and DEcalioc on the cluster. An overview of the LIGGGHTS scripts that control the simulation of the rotating drum are explained in Section 4. The additional MATLAB scripts and functions written to improve the ease of submitting a LIGGGHTS simulation to the cluster and prepare the resulting output for easy data analysis are covered in Section 5. To calibrate the rotating drum, a forked version of DEcalioc was created. Due to time constraints, the simulation of the rotating drum could not be calibrated; the additional user input required for DEcalioc before calibration can be completed can be found in Section 6. A number of additional pieces of software were used for the creation and visualisation of the simulation of the rotating drum. Although not necessary for its core functionality, if changes are to be made to the simulation or further post-processing is required, the software outlined in Section 7 may be of use. Although not an exhaustive list, the final section (Section 8) summarises some of the more specialised commands used in this document in a single 'cheat sheet' for use after the initial set up.

2 PuTTY

PuTTY is a client program that uses the SSH (Secure Shell) network protocol to run a remote session on a local computer over the network [1]. In this particular use case, it serves two purposes: it provides a terminal interface to interact with the cluster remotely and the ability to connect to the cluster without having to enter a password every time. Although the latter is not strictly necessary, it is a requirement for the MATLAB scripts and functions designed to initialise simulations and download the associated results from the cluster. Several additional utilities are bundled with PuTTY that are used to achieve these particular aims.

2.1 Initial Set-Up

PuTTY can be obtained from `https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html`. Please make sure to download one of the *Package files*, rather than the *Alternative binary files* to obtain all the additional utilities. Download and install PuTTY.

Once installed, open PuTTY and navigate through the PuTTY Configuration window using the Category toolbar on the left hand side. Update the following settings outlined below. If no shortcut was created on installation, the default installation location for PuTTY is *C:\Program Files\PuTTY*.

- Session
  - Host Name (or IP address) $\Longrightarrow$ hypnos5.fz-rossendorf.de
  - Port $\Longrightarrow$ 22
  - Connection Type $\Longrightarrow$ SSH
  - Saved Sessions $\Longrightarrow$ Provide a catchy name for your session
- Connection $\rightarrow$ Data
  - Auto-login username $\Longrightarrow$ your username

Return to the Session category and save the session settings. By loading these settings and selecting `Open`, you should be greeted with a terminal interface requesting your password. Enter your password (it will appear as if nothing is being typed, this is normal when using Linux) to log in to the cluster. For further information, see: `https://help.dreamhost.com/hc/en-us/articles/215464538-How-do-I-configure-PuTTY-`.

2.2 Passwordless Log In

To log in to the cluster without using a password (useful when using PuTTY and required for submitting simulations using MATLAB), a private/public key pair must be generated using the PuTTY Key Generator utility (PuTTYgen). An excellent guide on how this can be achieved can be found at: `https://help.dreamhost.com/hc/en-us/articles/215464758-How-do-I-set-up-passwordless-login-in-PuTTY-`. If further information is required, please see: `https://ssd.eff.org/en/module/deep-dive-end-end-encryption-how-do-public-key-encryption-systems-work`

NOTE: From a security point of view, it is recommended to save the private key with a passphrase to protect it, otherwise anyone with access to your computer will be able to log in to the cluster using your account.

By using the Pageant utility, an SSH authentication agent, the passphrase to your private key only needs to be entered once per session at start-up as it is retained in system memory for the duration of the session. To begin, load Pageant, right click on its icon in the Systems Tray and select `Add Key`. Navigate to your private key (PPK file) and open it; you will be prompted to enter your passphrase for the key. As the MATLAB scripts used to initialise and retrieve results from a LIGGGHTS simulation on the cluster use the Plink and PSFTP utilities included with PuTTY, the private key loaded into Pageant can be used without the need to enter a password each time a command is passed to the cluster by MATLAB.

3 Hypnos

With PuTTY and its associated utilities correctly set up, the focus can move to preparing the cluster. Section 3.1 explains how the DEM software, LIGGGHTS, should be set up to make sure all the necessary features required for the simulation and calibration of the rotating drum are enabled. As the calibration procedure is written using Octave, Section 3.2 covers the modules and additional packages needed when using GNU Octave on both the cluster or a local Ubuntu computer. Section 3.3 is also broken-down into two parts: the set up of the original version of DEcalioc and the forked version for the calibration of the rotating drum. The former may not be necessary, but the minimal working example included with DEcalioc provides a relatively quick way to check that LIGGGHTS and GNU Octave have been set up correctly without conflicting with each other. For all the Linux commands given, it is assumed that the working directory has been set as the default `/home/USERNAME/` directory.

3.1 LIGGGHTS

Installing LIGGGHTS on a Linux computer is a reasonably straight forward process with excellent documentation on how to do so provided on the official LIGGGHTS website: `https://www.cfdem.com/media/DEM/docu/Section_start.html`. Unfortunately it is a little more involved when setting up on the cluster. Some of the steps are blatantly obvious or are not necessary to run a simple DEM simulation; however, it is recommended to follow the following steps to allow for the additional MATLAB code for submitting and importing simulations to function as expected.

3.1.1 Preparation

To begin, log in to the cluster, and set the language from German to English (or any other language).

```
1  ssh -X USERNAME@hypnos5.fz-rossendorf.de
2  LANG=ENG
```

The latest version of LIGGGHTS can be obtained from its GitHub repository and cloned into the home directory. In the case of using multiple versions of LIGGGHTS, it is worth renaming the primary directory to *MYLIGGGHTS*.

```
1  git clone https://github.com/CFDEMproject/LIGGGHTS-PUBLIC.git
2  mv LIGGGHTS-PUBLIC/ MYLIGGGHTS
```

A number of environmental modules need to be loaded when compiling or using LIGGGHTS. To view all the available modules, the `module avail` command can be used. Although the specific version of each module may not be important when using LIGGGHTS by itself, to prevent incompatibilities with other software that is used at the same time as LIGGGHTS, the specific versions listed below for each module should be used.

```
1  module avail
2  module load cmake/3.10.1
3  module load gcc/7.2.0
4  module load openmpi/1.8.6
5  module load jpeg/9c
```

3.1.2 Compiling LIGGGHTS

Compiling LIGGGHTS correctly without issues is dependent on the order of operations. Please follow the instructions below in the order specified to guarantee a working executable. To begin, navigate to the *src* directory of *MYLIGGGHTS* and attempt to compile LIGGGHTS.

```
cd MYLIGGGHTS/src
make auto
```

The above command **will** fail; however, it creates a file called *Makefile.user* that contains numerous settings and paths required for compiling LIGGGHTS. Before editing this file, return to the *src* directory and clean-up the code produced by removing all the compiled object files.

```
make clean-all
```

When accessing the cluster via Putty or one of its derivatives on a Windows operating system, it is not possible to edit files in an external text editor as it is with a Linux-based operating system (although alternative ways are possible, see Section 7). Thus, text files must be edited within the terminal when using Windows. To begin, open the *Makefile.user* file using the GNU nano text editor.

```
nano MAKE/Makefile.user
```

Alternatively, if when logging in to the cluster using a Linux-based operating system and the `ssh` command was given with the `-X` option, files on the cluster can be edited using a local text editor (for example, gedit).

```
gedit MAKE/Makefile.user
```

Several lines within *Makefile.user* must be updated to allow particle data output as VTK files and to produce images of the simulation as JPEG files. If using nano, use the arrow keys to navigate through the file and update the following lines numbers to read the following:

```
Line 12:   AUTOINSTALL_VTK = "ON"
Line 18:   USE_JPG = "ON"
Line 143:  JPG_INC_USR=-I/opt/pkg/filelib/jpeg/9c/include
Line 145:  JPG_LIB_USR=-L/opt/pkg/filelib/jpeg/9c/lib
```

In case of future updates to the *Makefile.user* file by the developers of LIGGGHTS, it should be reasonably obvious which line numbers must be edited to read the above as the lines changed are amended rather than written from scratch. To exit and save the file, type `Ctrl+X` (brings up the menu to save changes), `Y` (save changes, could be `J` if the language was not changed from German) and finally `Enter` to overwrite the existing file.

Due to a bug, a core LIGGGHTS file must also be edited to allow LIGGGHTS to compile with JPEG output support. Within the *src* directory, open the *image.cpp* file and change the following lines to capitalise the option `true`:

```
1    Line 1042: jpeg_set_quality(&cinfo,85,TRUE);
2    Line 1043: jpeg_start_compress(&cinfo,TRUE);
```

Within LIGGGHTS, only a selection of the possible contact model variations are compiled in an optimised manner. Although the following changes are not necessary to run a LIGGGHTS simulation, the contact model used for the DEM simulation of the rotating drum is not optimised, reducing performance by 5–20 percent. To add the contact model to the whitelist of models to be optimised, a new file must be created in the *src* directory called *style_contact_model_user.whitelist*, containing a line describing the contact model to be added. The following commands can be used to create the file and add the line describing the contact model. Once finished, close the document using `Ctrl+D`. It is also worth deleting the old executable, *lmp_auto*, to prevent any potential conflict or confusion.

```
1    cat > style_contact_model_user.whitelist
2    GRAN_MODEL(HERTZ, TANGENTIAL_HISTORY, COHESION_OFF, ROLLING_EPSD2,
         SURFACE_DEFAULT)
3    Ctrl+D
4    rm lmp_auto
```

Another whitelist, *style_contact_model.whitelist*, is automatically created during compilation . This file must be deleted before recompiling LIGGGHTS, otherwise, it will not be updated.

```
1    rm style_contact_model.whitelist
```

NOTE: If at a later date, another contact model is used that is not optimised, *style_contact_model_user.whitelist* can be appended with the new contact model. A full list of all the possible contact models available can be found in the *style_contact_model.h* file found in the *src* directory. Run the `make clean-all` command, append *style_contact_model_user.whitelist* with the new contact model using nano and delete *style_contact_model.whitelist* from the *src* directory before recompiling LIGGGHTS.

With everything set-up correctly, LIGGGHTS can now be compiled. Due to the time required, compiling should not be carried out on the head node (hypnos5), but instead run on one of the compute nodes (laserXXX) of the cluster, submitted as a job. The `qsub` command can be used to request the necessary processor cores for the specified job.

```
qsub -N makeauto -q default -l nodes=1:ppn=8 -l walltime=4:00:00 -I
```

Where:

- `-N makeauto` $\Longrightarrow$ Set name of job to *makeauto*.
- `-q default` $\Longrightarrow$ Select default queue (see: `https://www.hzdr.de/db/Cms?pOid=29813&pNid=852`).
- `-l nodes=1:ppn=8` $\Longrightarrow$ Request one node and 8 processors per node (PPN).
- `-l walltime=4:00:00` $\Longrightarrow$ Set job wall time to 4 hours.
- `-I` $\Longrightarrow$ Interactive job (allow for input from the command window).

Upon successful initialisation, the terminal will read `USERNAME@laser123:~$` (or something similar). Depending on the current usage of the cluster, it may take a while before the job is successfully initialised. To cancel the job and try again use `Ctrl+C` (although it is said that patience is a virtue).

With the job request successfully submitted, navigate back to the *src* directory and compile LIGGGHTS using the 8 processor cores requested.

```
LANG=ENG
cd ~/MYLIGGGHTS/src/
make -j 8 auto
```

During the compiling process, the VTK library necessary for producing the post-processing files is also installed. This can take some time so please be patient. Once finished, the job can be terminated.

```
exit
```

The path to the new VTK library needs to be added to the *.bashrc* file, a shell script that is run on log in to the cluster. Return to the home directory (`cd ~/`) and open the *.bashrc* file. Add the following line, updated with your username, directly below the similar command for the TORQUE libraries (*/opt/torque/lib*). Once added, save and close the file.

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/home/USERNAME/MYLIGGGHTS/lib/vtk/
    install/lib/
```

To save time, the necessary modules can be automatically loaded during log in. In the home directory, open the *own.modules* file and add the following commands below the line `module purge`. If a different version of a module is already being loaded on log in, please comment it out along with any other modules that are not necessary (as this can potentially create conflict between modules). These changes will not take effect until the next log in.

```
1  module load cmake/3.10.1
2  module load gcc/7.2.0
3  module load openmpi/1.8.6
4  module load jpeg/9c
```

Exit the cluster using the `exit` command and log in again. To make sure the core LIGGGHTS features are working as expected, one of the example simulations provided with LIGGGHTS can be used as a test case (for example: *chute_wear*). Navigate to the chosen example simulation directory.

```
1  cd ~/MYLIGGGHTS/examples/LIGGGHTS/Tutorials_public/chute_wear/
```

In its current format, the simulation will not run as it is assumed that a symbolic link for LIGGGHTS has been created. Without administrative rights, this is not possible [2]; thus, it is easier to copy the LIGGGHTS executable, *lmp_auto* from the *src* directory into the simulation directory.

```
1  cp ~/MYLIGGGHTS/src/lmp_auto ./
```

To view the contents of the current directory and confirm that the executable has indeed been copied across, the `ls` command can be used. To run the sample simulation, a job must be submitted to the cluster.

```
1  qsub -N DEM -q default -l nodes=1:ppn=4 -l walltime=4:00:00 -I
```

Navigate back to the simulation directory and initialise the simulation. The `mpirun` command allows for the simulation to be run in parallel across multiple processor cores, improving performance. As four processor cores were requested in the job submission, run the chute wear example with the same number of processor cores.

```
1  cd ~/MYLIGGGHTS/examples/LIGGGHTS/Tutorials_public/chute_wear/
2  mpirun -np 4 lmp_auto < in.chute_wear
```

Where:

- `-np 4` $\Longrightarrow$ Run the simulation using four cores.
- `lmp_auto < in.chute_wear` $\Longrightarrow$ Run the LIGGGHTS simulation script: *in.chute_wear* using the LIGGGHTS executable.

The progress of the simulation can be tracked from the terminal output itself or by tailing the log.liggghts file produced by the simulation in a separate terminal.

```
1    tail -n +1 -f log.liggghts
```

Where:

- `-n +1` $\Longrightarrow$ Output all lines beginning with line 1. Alternatively, `-n 5` would only output the last 5 lines of the file (default is the last 10 lines).
- `-f` $\Longrightarrow$ The tail command will run forever, automatically printing new data to the terminal when found.
- `log.liggghts` $\Longrightarrow$ File to be tailed.

3.1.3 Optimising LIGGGHTS for a New Contact Model

NOTE: The process to optimise LIGGGHTS for a new contact model is somewhat temperamental with no guarantee that the following instructions will work. If unsuccessful, it in recommended to delete the *MYLIGGGHTS* directory (`rm -r ~/MYLIGGGHTS`) and start from scratch, remembering to add the new contact model to the *style_contact_model_user.whitelist* file at the appropriate point (see Section 3.1.2). Please remember to also back-up any simulation data you may wish to keep beforehand.

A performance improvement of up to 20% can be achieved by adding the contact model specified in the DEM simulation to the contact model whitelist (assuming the model is not already present in the list). To determine if a contact model is present on the whitelist, simply run the simulation as normal with your chosen contact model and view the *log.liggghts* file produced during the simulation. If the following can be found in the log file, the whitelist needs to be updated:

```
WARNING:
    The contact model you specified is not located in any whitelist.
    Because of this the model will be run in an unoptimized version (increasing
        runtime by up to 20%).
    In order to optimize this model you have the following options:
            (i)  Run the genAutoExamplesWhitelist.sh script in your LIGGGHTS(R)
                source folder to automatically parse the input script
            (ii) Add the model combination by hand to your
                style_contact_model_user.whitelist that can be found in your
                LIGGGHTS(R) source folder
    If you perform one of the steps above LIGGGHTS(R) needs to be recompiled to
        generate the optimized code.
```

Navigate to the *src* directory and run the *genAutoExamplesWhitelist.sh* script.

```
cd ~/MYLIGGGHTS/src
./genAutoExamplesWhitelist.sh
```

If the script *genAutoExamplesWhitelist.sh* fails to update the whitelist, the model can be added manually instead. First, run the `make clean-all` command in the *src* directory of *MYLIGGGHTS* to remove all the compiled object files. The old LIGGGHTS executable should also be deleted from the *src* directory using the command `rm lmp_auto`. Then, add the new contact model to the end of the whitelist file *style_contact_model_user.whitelist*, also found in the *src* directory (for reference, the *style_contact_model.h* file contains all the possible contact model combinations for LIGGGHTS). With *style_contact_model_user.whitelist* appended, delete the old *style_contact_model.whitelist* file from the *src* directory. Submit a new job to the clus-

ter and recompile LIGGGHTS as before. As the VTK library has already been added to the cluster, compiling LIGGGHTS should be significantly faster than the first time.

```
1    qsub -N makeauto -q default -l nodes=1:ppn=8 -l walltime=4:00:00 -I
2    cd ~/MYLIGGGHTS/src/
3    make -j 8 auto
```

In theory, the whitelist has been successfully updated to include the new contact model.

3.2 GNU Octave

GNU Octave is an open-source programming language written for scientific computing and designed with MATLAB compatibility in mind [3]. The calibration procedure, DEcalioc, is written for Octave and creates multiple LIGGGHTS DEM simulations during optimisation; thus, the main challenge with using DEcalioc on the cluster is making sure that the overlapping environmental module dependencies for Octave match those of LIGGGHTS. This is explained in Section 3.2.1. To test specific Octave functions, it is often easier to use the GUI interface for GNU Octave available on a personal computer, rather than the terminal of the cluster. Although Octave is available for Windows, some of the packages are not natively supported for the version of Octave required and so using a Linux-based operating system is recommended. The installation of GNU Octave for Ubuntu can be found in Section 3.2.2. Additional information on running Ubuntu as a virtual machine can be found in Section 7.4.

3.2.1 Octave on the Cluster

To prevent module conflict between LIGGGHTS and Octave, the following module versions should be used. Please add the modules not already listed in the *own.modules* file to it in the following order as previously explained in Section 3.1.1. These modules are everything needed to run LIGGGHTS, Octave and DEcalioc correctly. To load the new modules, exit and log back into the cluster.

```
1  module load gcc/7.2.0
2  module load openmpi/1.10.2
3  module load jpeg/9c
4  module load gnuplot/5.2.0
5  module load glpk/4.64
6  module load octave/4.2.2
7  module load graphicsmagick/1.3.33
```

The following packages for Octave 4.2.2 need to be installed on the cluster for the simulation to work correctly. Due to a recent change in the parallel package, DEcalioc will not work with the default installation of each package, available using the `pkg install -forge package_name` command; thus, the packages required must be installed manually. Due to their dependencies, it is recommended to install the packages in the order they are listed below. These are not the latest versions of each package and therefore must be obtained from the following link rather than the home page for each package: `https://sourceforge.net/projects/octave/files/Octave%20Forge%20Packages/Individual%20Package%20Releases/`.

```
1  struct-1.0.14.tar.gz
```

```
    stk-2.5.0.tar.gz
    io-2.4.10.tar.gz
    statistics-1.3.0.tar.gz
    optim-1.5.2.tar.gz
    parallel-3.1.1.tar.gz
```

Further information on Octave packages can be found at the following links:

- List of all Octave packages $\Longrightarrow$ `https://octave.sourceforge.io/packages.php`
- Installing and removing packages $\Longrightarrow$ `https://octave.org/doc/v4.2.1/Installing-and-Removing-Packages.html`

Without additional software, the easiest way to add the packages to the cluster is to use the network drives as both the cluster and the local computers at HZDR have access to them. Download each package and add them to the *P* drive of the computer (the personal network drive). Log in to the cluster and navigate to the same network drive (the *HOME* directory). The *tar.gz* files can be copied across to the *home* directory. **NOTE:** *HOME* and *home* are different directories; this is not a typo.

```
cd ~/HOME/
cp struct-1.0.14.tar.gz stk-2.5.0.tar.gz io-2.4.10.tar.gz statistics-1.3.0.tar.
   gz optim-1.5.2.tar.gz parallel-3.1.1.tar.gz ~/
```

To check if the files have successfully been copied across, navigate to the *home* directory (`cd ~/`) and view its contents (`ls`). Start Octave and install each of the packages in turn. To view a list of the packages installed, the Octave command `pkg list` can be used.

```
octave
cd ~
pkg install struct-1.0.14.tar.gz
```

NOTE: If utilising the shared network storage is not possible, an SFTP client, such as WinSCP, can be used to transfer files to the cluster (see Section 7.1 for more information).

3.2.2 Octave on a Personal Ubuntu Computer

As DEcalioc was written for Octave 4.2.2, it is recommended to install and use this version of Octave to prevent incompatibilities. For most Ubuntu-based distributions, Octave 4.2.2 should be available to download and install from the Ubuntu Software Store; if this is not the case, the process becomes somewhat more convoluted. The following instructions to build and install Octave 4.2.2 without the Ubuntu Software Store are known to work for Ubuntu 18.04.3 LTS (Bionic Beaver). There is absolutely no guarantee that this process will work for a different combination of Octave version, Ubuntu version or Linux distribution.

To build Octave, there are a number of packages (dependencies) that must first be installed. Open a terminal, navigate to the home directory and install the following packages.

```
1  cd ~/
2  sudo apt-get install gcc g++ gfortran make libblas-dev liblapack-dev libpcre3-
       dev libarpack2-dev libcurl4-gnutls-dev epstool libfftw3-dev transfig libfltk1
       .3-dev libfontconfig1-dev libfreetype6-dev libgl2ps-dev libglpk-dev
       libreadline-dev gnuplot-x11 libgraphicsmagick++1-dev libhdf5-serial-dev
       openjdk-8-jdk libsndfile1-dev llvm-dev lpr texinfo libgl1-mesa-dev libosmesa6
       -dev pstoedit portaudio19-dev libqhull-dev libqrupdate-dev libqscintilla2-dev
        libsuitesparse-dev texlive texlive-generic-recommended libxft-dev zlib1g-dev
        autoconf automake bison flex gperf gzip icoutils librsvg2-bin libtool perl
       rsync tar qtbase5-dev qttools5-dev qttools5-dev-tools libqscintilla2-qt5-dev
```

Download the Octave 4.2.2 release, extract its contents and enter the newly created directory.

```
1  wget https://ftpmirror.gnu.org/octave/octave-4.2.2.tar.gz
2  tar -xzf octave-4.2.2.tar.gz
3  cd octave-4.2.2
```

The next set of commands are used to configure, build, check and install Octave. By changing line three, the installation directory for Octave can be changed (`./../configure --prefix=$/path/for/new/directory`). Depending on the number of processor cores available or assigned to the virtual machine, the build process can be sped up by increasing the number of parallel jobs specified in line four.

```
1  mkdir .build
2  cd   .build
3  ./../configure --prefix=$HOME/my_octave
4  make -j 2
5  make check
6  make install
```

The final step is to create an alias that allows for the new build of Octave to be started directly from the terminal. This is effectively a shortcut, in this case `myoctave`, that is used to reference the command to start Octave. The second command reloads the *.bashrc* shell script without having to log out and log back in again.

```
1    echo "alias myoctave='$HOME/my_octave/bin/octave'" >> ~/.bashrc
2    source $HOME/.bashrc
```

To start Octave in your current directory, use the newly created alias command `myoctave`. Alternatively, Octave can be run directly from the terminal without a GUI using the command `myoctave -W`.

To uninstall Octave, remove the *.build directory* from the *octave-4.2.2* directory and delete the installation directory for Octave. Remember to also remove the alias entry in the *.bashrc* shell script (`nano ~/.bashrc`).

```
1    cd ~/octave-4.2.2/.build
2    make uninstall
3    cd ~/
4    rm -rf $HOME/my_octave
```

3.3 DEcalioc

Discrete **E**lement **ca**libration using **LI**GGGHTS and **Oc**tave (DEcalioc) is a collection of functions, scripts and packages written for GNU Octave to automatically calibrate a DEM simulation with minimal user preparation and input [4]. The user provides the results obtained from an experiment (e.g., the bulk density) that are to be replicated in a DEM simulation of the same experiment. A surrogate model describing the relationship between the unknown interaction parameters (e.g., the coefficient of rolling friction) and the results of interest is developed, returning the set of parameters that minimises the differences between the results obtained from the experiment and the simulation [4]. As DEcalioc was initially designed to work with LIGGGHTS, it makes the ideal method to calibrate the simulation of the rotating drum. Please make sure Sections 3.1 and 3.2 have been completed first before continuing. It is also recommended to read the documentation for DEcalioc beforehand as it provides an excellent overview of the core functionality of DEcalioc, as well as how to integrate a new DEM model for calibration. The *documentation.pdf* file can be found in the GitHub repository for DEcalioc: `https://github.com/DECALIOC/DEcalioc`.

3.3.1 Sample DEcalioc Set Up and Test

To test if Octave and LIGGGHTS have been configured correctly to work together, the minimal working example included with DEcalioc can be used. With the default settings specified, this process will take approximately five hours. To begin, download DEcalioc from its GitHub repository and rename the primary directory to prevent any confusion with the forked version of DEcalioc.

```
1  cd ~/
2  git clone https://github.com/DECALIOC/DEcalioc.git
3  mv DEcalioc/ DecaliocOG
```

To initialise the DEM simulation included with the minimal working example, DEcalioc calls the file *runscript* found in the example simulation directory (*~/DecaliocOG/DEcalioc/DEMmodels/Lift100/*). By default, the *runscript* file contains the following command:

```
1  liggghts < in.Lift100
```

If a symbolic link for LIGGGHTS could be created, this would work correctly; however, as stated in Section 3.1, this is not possible on the cluster. To circumvent this problem, edit the *runscript* file to read:

```
1  #!/bin/bash
2  ~/MYLIGGGHTS/src/lmp_auto < in.Lift100
```

Where,

- `#!/bin/bash` $\implies$ Specifies that the bash interpreter should be used to interpret the commands given in the shell script.
- `~/MYLIGGGHTS/src/lmp_auto < in.Lift100` $\implies$ Runs the *in.Lift100* script using the LIGGGHTS executable without having to copy across the executable to the simulation directory.

With the *runscript* file updated, the minimal working example can be calibrated. The primary Octave script for DEcalioc that begins the calibration procedure is called *DEcalioc.m*. Create a new job on the cluster, open GNU Octave and initialise the calibration procedure.

```
1  qsub -N octave_Cali -q default -l nodes=1:ppn=8 -l walltime=50:00:00 -I
2  octave
3  cd DEcaliocOG/DEcalioc/
4  DEcalioc
```

If everything is working correctly, the calibration procedure for the minimal working example should take a few hours to finish, providing a set of calibrated parameters for the simulation that best replicate the real-world results.

3.3.2 Forked DEcalioc

A forked version of DEcalioc was created that is designed to calibrate a rotating drum using experimentally determined dynamic angles of repose. Besides the integration of a new model to be calibrated, DEcalioc was altered to improve support for running on Hypnos. Rather than submitting the calibration procedure as a single job, each DEM simulation is submitted as a separate job, greatly increasing the number of parallel simulations that can be run on the cluster at the same time.

The download, set up and submission of the calibration procedure is outlined below. Unfortunately, due to time constraints, the DEM simulation of the rotating drum could not be calibrated; thus, further input by the user is required. This includes: adding the experimentally obtained dynamic angles of repose for each filling degree and drum speed, optimisation tolerances, parameters to be calibrated and parallelisation settings. The alterations required are explained in more detail in Section 6 below. Unfortunately, the standard deviation of the output from the *threeAngles* algorithm used to obtain the non-linear dynamic angle of repose was too large to be useful for calibration and was turned off. Section 6.2 outlines how the algorithm can be reimplemented.

To begin, download the repository to the cluster and once again rename the primary directory to prevent confusion between different versions of DEcalioc.

```
cd ~/
git clone https://github.com/PurpleCrumpets/DEcalioc.git
mv DEcalioc/ DecaliocDrum
```

Submit a job to the long queue on Hypnos for running the primary DEcalioc script. Open Octave and run the *DEcalioc.m* script.

```
qsub -N CalibrationDrum -q long -l nodes=1:ppn=1 -l walltime=1200:00:00 -I
octave
cd ~/DEcaliocDum/DEcalioc/
DEcalioc
```

This will submit the calibration procedure as an interactive job, with the output displayed directly to the terminal. Although useful to track the progress of the calibration procedure and check the values assigned to specific variables, shutting down the computer or closing the terminal **will** cancel the interactive job. To prevent this from occurring, the bash script *submitCalibration.sh* can be used instead to submit the calibration procedure to the cluster. A log file, *log.octave*, is created in */DEcaliocDrum/DEcalioc* which can be tailed instead if desired.

```
cd ~/DEcaliocDum/
/opt/torque/bin/qsub submitCalibration.sh
tail -f -n +1 log.octave
```

4 LIGGGHTS Scripts

For the DEM simulation of the rotating drum, there are four separate LIGGGHTS scripts:

- *data.head*
- *in.rotatingdrum*
- *in.rotatingdrumRestart*
- *imaging.data*

The most important script for the end-user is *data.head*. Here, the primary variables that control the simulation are defined and can be edited. This is discussed in more detail in Section 4.2 below. The LIGGGHTS script *in.rotatingdrum* contains the commands used to define the simulation, from the filling of the drum to the final mixing of particles. Restart files are created at key points throughout the simulation. These files can be used by the *in.rotatingdrumRestart* script to continue the simulation, potentially with a different set of properties as defined by *data.head* (see Section 4.4). The final script *imaging.data* is only invoked if specified by the user in *data.head*. During the rotation of the drum, images of the particles can be taken from the end of the drum. These are primarily used for calibration of the drum using DEcalioc. For better quality images and greater flexibility in their composure, ParaView can be used instead (see Section 7.6). The format of the output data from a LIGGGHTS simulation can be found in Section 5.

4.1 in.rotatingdrum and in.rotatingdrumRestart

For the simulation of the rotating drum, the user should not need to change the *in.rotatingdrum* and *in.rotatingdrumRestart* scripts. In case this is required (e.g., change the contact model), a brief overview of each stage in the simulation is provided below. As the *in.rotatingdrumRestart* script uses virtually the same settings as the *in.rotatingdrum* script and shares the same commands for the rotation of the drum, it will not be discussed directly. **NOTE:** Every time geometry is added or removed from the simulation, the contact model is redefined. If the contact model is to be changed, please remember to change all occurrences of the contact model command.

The *in.rotatingdrum* script is divided into three distinct sections: the initialisation of the simulation, preparation of the drum for rotation and the rotation of the drum. These are labelled Section I1–I7, E1–E11 and E12–E14 respectively.

4.1.1 Initialisation

I1 Import the parameters and settings defined in the *data.head* script.

I2 Set the preliminary simulation settings. This includes: atom properties, processor layout and communication, units, periodicity of the domain and the size of the neighbours list for contact detection.

I3 Set the limits used to control the size of the domain throughout the simulation and create the domain. The limits are dynamically set based on the settings defined in the *data.head* script.

I4 Set the material and interaction properties for each material and material combination. The properties are set based on the settings defined in the *data.head* script.

I5 Fix the settings for the insertion of the particles to the simulation domain. Based on the chosen drum size (3D or quasi-2D) the insertion rate is changed to prevent particles bouncing out of the drum during the filling process. The geometry used to define the region of particle insertion is added to the simulation.

I6 Import the initial simulation geometry (drum, divider and bottom lid of drum with the cut-out for the removal of the divider). The geometry for a 3D or quasi-2D simulation are inserted and rotated into position. The contact model for the simulation geometry is also set.

I7 The final simulation settings are applied. This includes: the particle contact model and the direction and magnitude of gravity. The time step is also set and compared against

the Rayleigh time step. The frequency of the output to the terminal, writing the drum rotational speed to the *drum_rotational_velocity.txt* file, producing images for calibration and the creation of the VTK and STL files for post-processing is set (the latter two if specified in *data.head*).

4.1.2 Rotation Preparation

E1 The filling of the drum is started. Once all the particles are below a certain height, the top lid of the drum is added to the simulation and the geometry used to specify the area of particle insertion removed. The y-direction limits of the simulation domain are adjusted to accordingly.

E2 Once the drum is filled, the particles are allowed to settle for 1.5 seconds. A restart file is created (*1_filled_drum.restart*).

E3 The direction that gravity is acting is rotated 90 degrees from the axial direction of the drum (positive y-direction) to the vertical (negative z-direction). This is less computationally expensive than rotating the geometry. The time taken for rotation is defined in the *data.head* file.

E4 The particles are allowed to settle for one second after the pseudo-rotation of the drum. A restart file is created (*2_pre_vibrate.restart*).

E5 Begin to level the particles by rotating the drum around its axis. The drum rotational speed and number of rotations are specified in the *data.head* input script.

E6 The particles are allowed to settle for 0.5 seconds after the initial levelling by rotation of the drum.

E7 The particles in the drum are levelled by vibrating the drum. The amplitude, frequency and number of oscillation for vibration are specified in the *data.head* input file. Based on the amplitude of oscillation, the simulation domain size is increased for vibration and returned to its original size afterwards.

E8 After levelling, the particles are allowed to settle for 1.5 seconds. The simulation domain is increased in the y-direction in preparation for the removal of the divider from the drum. A restart file is created (*3_post_vibrate.restart*).

E9 The divider is withdrawn from the drum and removed from the simulation. The speed of removal is defined in the *data.head* input file. The end caps of the drum may be removed and the simulation domain size adjusted accordingly depending on the periodicity settings and the drum size specified in the *data.head* file.

E10 The particles are allowed to settle for one second after the removal of the divider.

E11 The simulation is run until the current time step is a multiple of the frequency of output defined in Section I7. This is to make sure that first output file produced (if set in *data.head*) describes the initial conditions of the rotation of the drum. A restart file is created for the simulation ready to begin rotation (*2D_periodic.restart*, *2D_non_periodic.restart* or *3D.restart*).

4.1.3 Rotation

E12 Obtain the set of initial conditions for the rotation of the drum specified in the *rpm_drum.txt* file.

E13 Loop for the rotation of the drum. The time and rotational speed for each loop are specified in *rpm_drum.txt*. The *imaging.data* script is called for each loop and 2D images of the end of the drum during mixing created if specified in *data.head*.

E14 The rotation of the drum is stopped. If initialised, the imaging command is also stopped and a final restart file is created (*END_2D_periodic.restart*, *END_2D_non_periodic.restart* or *END_3D.restart*).

4.2 data.head

The *data.head* file allows the user to change key settings and properties for the other three LIGGGHTS scripts, and the submission of the simulation to the cluster. Five separate materials are defined in the simulation, each given a shorthand name. They are defined below in Table 1. The number assigned to each material is used when inserting particles or geometry into the simulation domain as a particular material type. The shorthand name is used when defining certain variables (for example: density or friction coefficients). An overview of each variable specified in *data.head* can be found below.

Table 1: Simulation Materials

Number	Type	Material	Shorthand
1	Plastic	Polypropylene (PP)	PP
2	Glass	Soda-lime glass	GL
3	Drum wall	Poly(methyl methacrylate) (PMMA), Aluminium (Al)	DW
4	Drum imaging end without opening	Poly(methyl methacrylate) (PMMA)	DE1
5	Drum end with opening	Polyvinyl chloride (PVC)	DE2

NOTE: To reduce the number of design variables to be calibrated and due to the difficulty in distinguishing between the effects of each of the end materials for a quasi-2D drum, the fifth material, DE2 is not in use. Instead, both ends of the drum currently are defined as the same material, DE1. Updating the scripts in the future to include the fifth material should not be too difficult.

Simulation Settings

- *simtype* stores the simulation type (i = initialisation, r = restart). This is used by the MATLAB script *liggghtsInitialisationScript* to upload the required files to either start a new simulation or restart an existing one (see Section 4.4).
- *queue* stores the queue to submit the job to on the cluster. The choice of queue affects the number of processor cores that can be assigned to a simulation, as well as the maximum wall time. Please see `https://www.hzdr.de/db/Cms?pNid=23` for more information.
- *walltime* stores the maximum wall time for the job submitted to the cluster. It is in the format: hhhh:mm:ss.
- *simdim* stores the dimensions of the simulation domain. A value of 3 indicates a 3D simulation with the full-length drum, while a value of 2 specifies a quasi-2D simulation with the shorter drum.

- *periodicbb* specifies the periodicity of the simulation domain in the y-direction (0 = non-periodic, 1 = periodic). This is only applicable for a quasi-2D simulation (*simdim* has a value of 2).

- *proc* stores the number of processor cores assigned to the simulation.

- *nodes* stores the number of processing nodes assigned to the simulation. **NOTE:** This setting does not work at the moment and should be left equal to 1.

- *autoproc* specifies if the processor layout should be automatically set by LIGGGHTS, overriding the variables *(x/y/z)proc* (0 = don't auto-assign, 1 = auto-assign).

- *(x/y/z)proc* stores the number of CPUs assigned in the x/y/z direction. The product of *(x/y/z)proc* must equal *proc*. The exception to this is when the variable is assigned a value of 0. In this case LIGGGHTS will auto-assign the number of processor cores in that direction based on the value of *proc* and the remaining *(x/y/z)proc* variables. All three of *(x/y/z)proc* can be set to 0. Unlike the Premium version of LIGGGHTS, the public version does not support the dynamic allocation of processors in 3D space as the simulation progresses. Please choose the processor layout wisely as it may otherwise hinder the performance of the simulation. To maximise the efficiency of a multi-threaded simulation, the number of particles in each region assigned to a processor should be the same.

- *produceVTK* specifies if VTK and STL files, containing particle information and the simulation geometry respectively, are produced.

- *imaging* specifies if the LIGGGHTS script *imaging.data* is called during the mixing of the particles in the drum (0 = no images produced, 1 = images produced for the full mixing, 2 = images produced only during the calibration period specified by *caliStart*, *caliLength* and *caliSkip*).

- *caliStart* stores the loop count at which calibration imaging begins. This is only applicable if calibration imaging is selected for the *imaging* variable. Each additional time-rotational speed pair after the initial conditions pair specified in *rpm_drum.txt* defines one loop of the rotation of the drum. See Section 4.3 below for more information.

- *caliLength* stores the number of loops that calibration imaging is carried out for.

- *caliSkip* stores the number of loops skipped after imaging has stopped before it resumes. For the calibration of the rotating drum, *caliStart*, *caliLength* and *caliSkip* are given values of 3, 1 and 2 respectively. For the sample input for *rpm_drum.txt* above, the loops for which *imaging.data* is invoked can be seen in Figure 1 below.

DEM Parameters

- *ts* stores the step size of the simulation (s).
- *neighdim* stores the size of the neighbour list (m).
- *thermostep* stores the frequency that the output files are produced from LIGGGHTS (images, particle information, geometry files, record of drum rotational speed) (s).

Drum Rotation

- *rotDirection* stores the direction of drum rotation, relative to the initial starting positions of each particle type (0 = glass particles lead rotation, 1 = polypropylene particles lead rotation).

Particle Insertion

- *insertionrate* stores the rate at which particles of one type are added to the simulation domain during the filling of the drum ($particles/s$).
- *volfracGL* stores the volumetric fraction of glass particles in the bed ($-$).
- *volfracPP* stores the volumetric fraction of polypropylene particles in the bed ($-$).
- *fillingdegree* stores the combined volumetric filling degree of both particle types in the drum ($-$).
- *porosity* stores the porosity of the particle bed ($-$).

Material Properties

- *radiusXX* stores the radius of particles of material type XX (m).
- *densityXX* stores the density of particles of material type XX (kg/m^3).
- *ymXX* stores the Young's modulus of material type XX (Pa). **NOTE:** The values listed are two orders of magnitude less than their actual values. It has been shown that reducing the Young's modulus from a very, very large number to a very large number does not affect the simulation in any significant way [4]. By reducing the Young's modulus, the Rayleigh criterion is increased; thus, a larger time step can be used to improve the performance of the simulation.
- *prXX* stores the Poisson's ration of material type XX ($-$).

- *CoR_XX_YY* stores the coefficient of restitution between materials *XX* and *YY* (−).
- *CoF_XX_YY* stores the coefficient of friction between materials *XX* and *YY* (−).
- *CoRF_XX_YY* stores the coefficient of rolling friction between materials *XX* and *YY* (−).

NOTE: The coefficients between materials that represent the geometry have been set as low as possible. As these materials cannot interact with each other, their value is ultimately arbitrary.

Geometry

It is important to input the correct dimensions for the drum and its components as they are used to control the size of the simulation domain. An excessively large simulation domain may cause unequal assignment of particles per processor, reducing performance. A domain that is smaller than the geometry will cause the simulation to fail.

- *lengthdrum3D* stores the length of the full-sized '3D' drum (m).
- *lengthdrum2D* stores the length of the 'quasi-2D' drum (m).
- *widthdrum* stores the outer diameter of the drum (m).
- *iddrum* stores the inner diameter of the drum (m).
- *thickcap* stores the thickness of the end caps/lids of the drum (m).
- *rmpMix* stores the rotational speed of the drum for the levelling of the particles within the drum (RPM). This is not used for the mixing of the particles.
- *numrotMix* stores the number of complete rotations of the drum during the levelling of particles ($-$).
- *(x/y/z)amplitude* stores the amplitude of the vibration of the drum in the x/y/z direction during the levelling of the particles within the drum (m).
- *frequencyvib* stores the frequency of the oscillation of the drum (Hz). **NOTE:** This must be a whole number to prevent the geometry from leaving the simulation domain or the off-axis rotation of the drum.
- *cyclevib* stores the number of vibration cycles ($-$). **NOTE:** To prevent the geometry leaving the simulation domain and the off-axis rotation of the drum, *cyclevib* must be a whole number.
- *rotTime* stores the time taken to change the direction that gravity acts from a horizontal to a vertical direction (s).
- *dividerspeed* stores the linear speed of the divider when it is removed from the drum (m/s).

4.3 rpm_drum.txt

Further to the four LIGGGHTS scripts, there is one additional text file that is used to control the LIGGGHTS simulation: *rpm_drum.txt*. Here, the input to the loop controlling the rotation of the drum during mixing is provided by specifying the rotational speed of the drum at particular instances in time. For each loop, a constant acceleration is assumed to occur between each time specified. An example *rpm_drum.txt* file is shown below, alternating between time (seconds) and the rotational speed (RPM) of the drum at that time. The output is shown graphically in Figure 1 below.

```
0
0
0.5
5
15.5
5
25.5
5
26
20
41
20
51
20
51.5
40
66.5
40
76.5
40
```

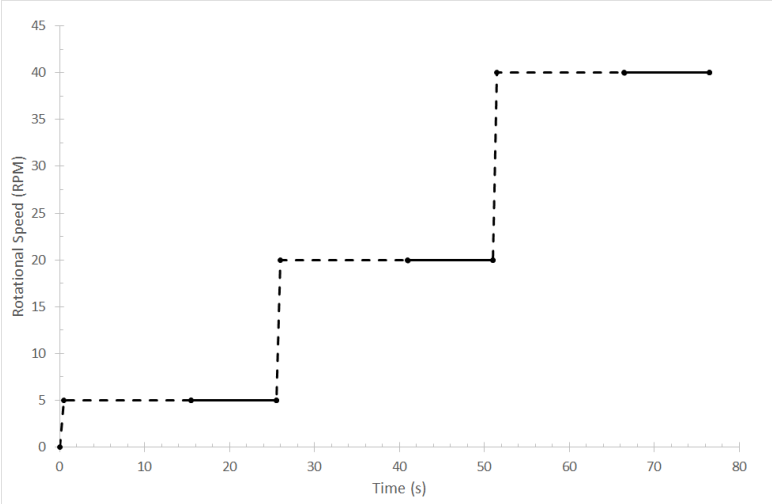

Figure 1: Sample Profile of rpm_drum.txt. The solid line segments refer to rotation loops for which *imaging.data* was invoked for calibration purposes (see Simulation Settings in Section 4.2).

4.4 Using Restart Files

As discussed in Section 4, *in.rotatingdrum* produces restart files at key points that can be utilised by the *in.rotatingdrumRestart* script to continue a simulation, with the option to use a different set of interaction properties as defined in *data.head*. The latter half of the *in.rotatingdrumRestart* script containing the rotation of the drum is nearly identical to *in.rotatingdrum*; the main differences between the two are the initial commands for preparing the simulation [5]. Although basic incompatibility between the simulations is checked when submitting a LIGGGHTS restart simulation using MATLAB (see Section 5.2.1), to prevent any issues, it is generally recommended to use the same input *data.head* file as the simulation that created the restart file. Restart files are produced at five different stages throughout the simulation:

1. *1_filled_drum.restart*
2. *2_pre_vibrate.restart*
3. *3_post_vibrate.restart*
4. *2D_periodic.restart*, *2D_non_periodic.restart*, *3D.restart*
5. *END_2D_periodic.restart*, *END_2D_non_periodic.restart*, *END_3D.restart*

The current *in.rotatingdrumRestart* script is designed to use one of the restart files produced during the latter two stages. To help differentiate between compatible restart files, these restart files are given different file names depending on the periodicity of the simulation and the length of the drum used. As *in.rotatingdrumRestart* is currently configured to read the fifth set of restart files, Section I2 (Initialisation - Load Restart File) of the script will need to be updated if a different restart file is to be utilised. The same directory (*restart*) is used for the restart files used for and produced by the simulation (see Figure 2 below). As a result, any existing restart file **will** be overwritten by a newly created restart file of the same name. Please remember to keep a back-up of the original restart file separate if it is to be used for multiple simulations.

4.5 Syntax Highlighting

The best way to edit a LIGGGHTS scripts is to use the text editor gedit as a plugin exists that allows for the syntax of LIGGGHTS scripts to be highlighted as appropriate. For Windows, gedit can be installed from: `https://wiki.gnome.org/Apps/Gedit#Download`. For Linux, the command `sudo apt-get install gedit` can be used.

With gedit installed, the necessary language definition file can be found at: `https://www.cfdem.com/gnome-syntax-highlighting`. On Windows, use 7-Zip to extract the contents of *liggghts.lang_.tar_1.gz* **twice** and place the resultant *liggghts.lang* file in the *language-specs* directory of gedit. In a default Windows installation, this is: *C:\Program Files\gedit\share\gtksourceview-3.0\language-specs*. On Linux the following commands can be used instead.

```
cd ~/Downloads/
tar xvzf liggghts.lang_.tar_1.gz
sudo mv liggghts.lang /usr/share/gtksourceview-3.0/language-specs/
rm liggghts.lang_.tar_1.gz
```

To enable syntax highlighting, close all open sessions of gedit and open a LIGGGHTS script in a new instance of gedit. Syntax highlighting should be available under `Menu -> View -> Highlight Mode -> LIGGGHTS`.

5 MATLAB-LIGGGHTS Integration

To allow for the easy set up, submission and tracking of LIGGGHTS simulations to the cluster, as well as the preparation results for further analysis, several scripts and functions were written using MATLAB. An overview of their use is provided below. The MATLAB functions, LIGGGHTS scripts and additional files can be obtained from the following GitHub repository: `https://github.com/PurpleCrumpets/MaLI.git`.

5.1 Overview of Directories and Scripts

The directory structure for the project and the two main MATLAB scripts' function dependencies can be found in Figures 2 and 3, respectively. The significance behind the colour of each folder or function is explained below in Table 2.

Table 2: Directory and Function Colour Codes

Colour	Directory	Function
Orange	Created during runtime	-
Green	Contains files requiring user input	Require user input
Blue	Fixed components of code	Fixed components of code
Red	Obtained from cluster	-

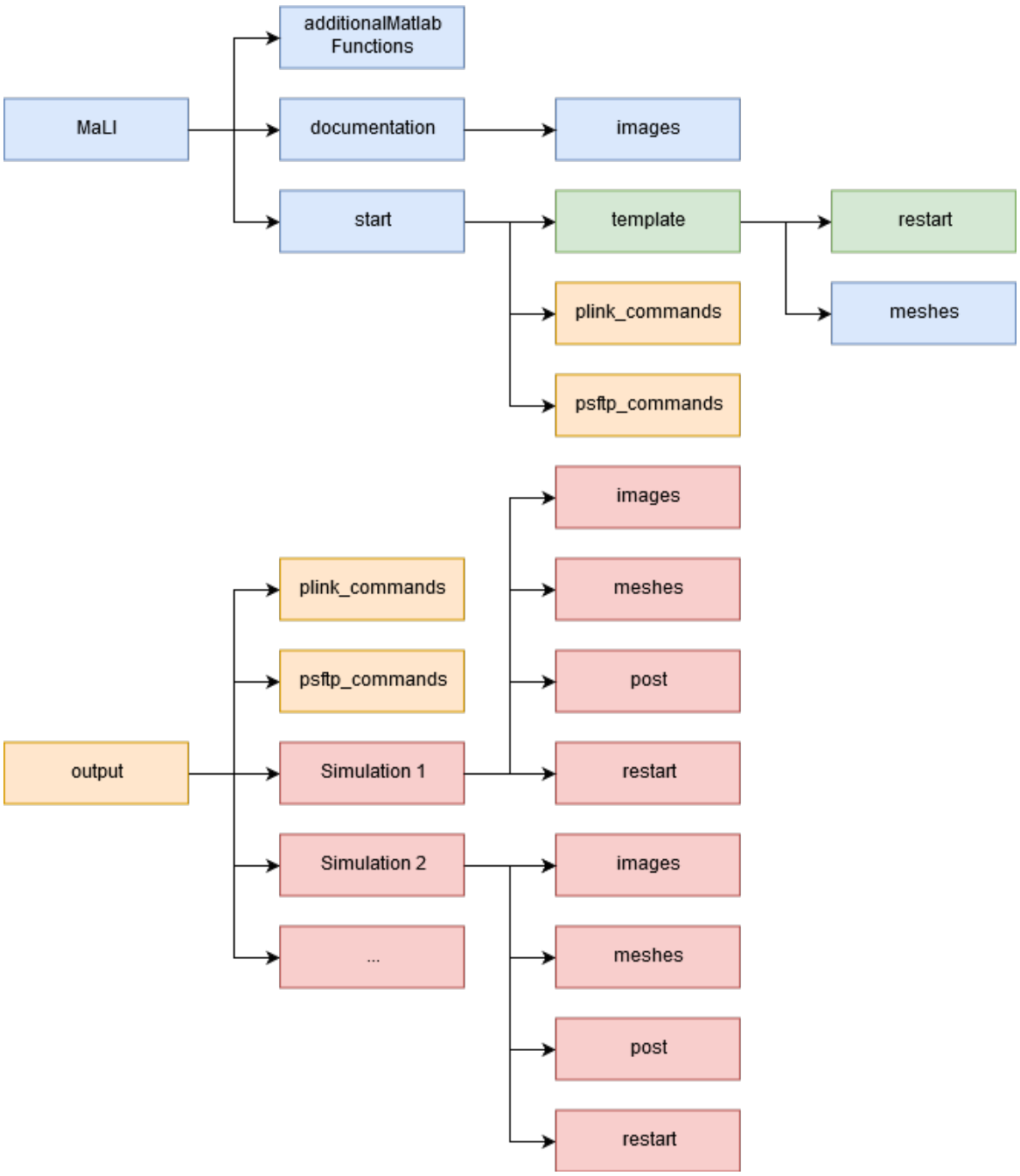

Figure 2: The folder structure for the MATLAB integration with LIGGGHTS (MaLI) repository.

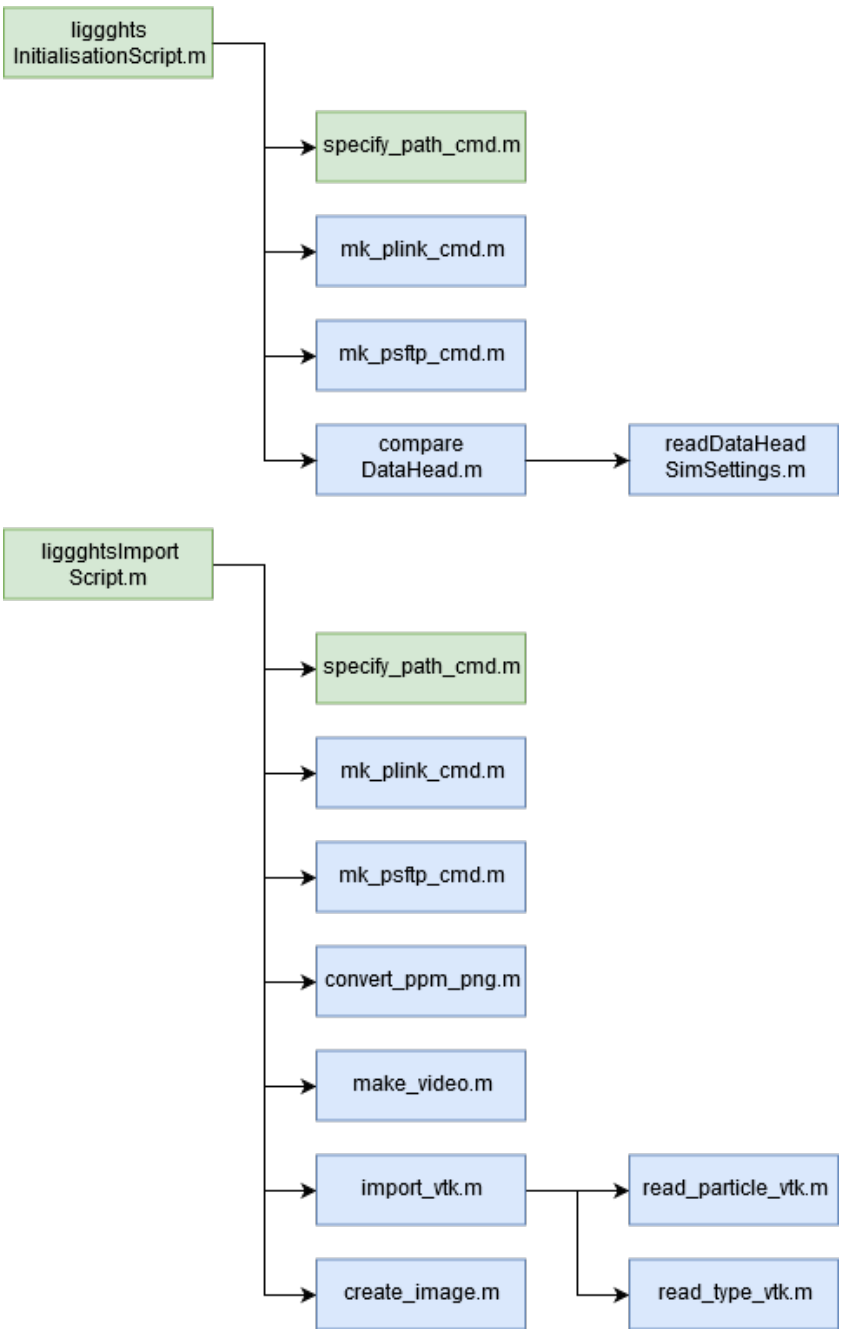

Figure 3: The MATLAB function dependencies for the *liggghtsInitialisationScript.m* and *liggghtsImportScript.m* scripts.

The main input directory, *MaLI*, contains the primary MATLAB functions for simulation submission and results obtainment; additional MATLAB functions for secondary tasks can be found in the *additionalMatlabFunctions* directory, but are not discussed in any detail as they are unlikely to be needed or used. The documentation for the project and the forked version of DEcalioc can be found in the *documentation* directory. The final directory, *start*, contains all the LIGGGHTS scripts, bash scripts, mesh and restart files for a LIGGGHTS DEM simulation. The directories *plink_commands* and *psftp_commands* are created dynamically during the submission of a simulation and contain text files with commands for the cluster. The mesh files that describe the geometry of the simulation should be stored in the *meshes* directory. Depending on the user settings specified in the *liggghtsInitialisationScript.m* and *data.head* scripts, restart files placed in the *restart* directory are used for the given simulation.

The *output* directory also contains a similar set of dynamically created *plink_commands* and *psftp_commands* directories. The remaining directories are the downloaded simulation directories from the cluster, modified by the *liggghtsImportScript.m* script. The contents of each simulation directory will change depending on the settings defined in the *data.head* input file. Any images produced during the LIGGGHTS simulation can be found in the *images* directory. The images are stitched together to produce a rudimentary video of the particles played back in real time. A final file, *image_ts.txt* records the time steps that imaging was started and finished. There can be more than one pair of time steps if imaging is turned on and off during the simulation (as required for calibration). The *post* directory contains the output geometry (STL), particle files (VTK) and drum rotational speed log (*drum_rotational_velocity.txt*) produced every *thermostep* simulation seconds defined in *data.head* (see Section 4.2). The initial and final time steps for the rotational of the drum are also recorded to *start_rotating.txt*. The geometry and particle files produced can be visualised using ParaView (see Section 7). Any new restart files created during the simulation are stored in the *restart* directory. Existing restart files of the same name will be overwritten in the process. The final *meshes* directory contains the same initial mesh files that were used for the simulation.

5.2 Using MATLAB

Before the MATLAB initialisation and output scripts can be used, public key authentication with the cluster must be set up (see Section 2) and *specify_path_cmd.m* must be edited on a per-user basis. The *specify_path_cmd.m* function requires the end-user to specify their username for the cluster, the absolute paths for both the *output* and *MaLI* directories on the local computer and finally the absolute path to the *MYLIGGGHTS* (or equivalent) directory on the cluster. A more detailed explanation and formatting requirements are provided within the function itself.

5.2.1 Submitting a Simulation

To submit a LIGGGHTS simulation to the cluster, the *liggghtsInitialisationScript.m* script is used. There are three additional variables that can be defined as the start of the script: *projectname*, *restartSource* and *tail*. All of these variables are optional; if they are not in use, please leave them as empty character arrays. The remaining simulation settings are defined in the *data.head* LIGGGHTS input script (see Section 4.2).

As the variable name suggests, the name of the simulation can be assigned to the variable *projectname*. If this project name already exists on the cluster, the user will be prompted to either overwrite the existing simulation or to provide a new project name. If left blank, the user will be prompted to provide a name for the project. Please make sure that the name provided does not contain any spaces; an underscore may be used instead if so desired.

If a restart simulation is defined by *data.head*, the *restartSource* variable is used, otherwise it is ignored. The user can specify the name of a project found in the *output* directory to obtain the restart files from or leave it blank to input the project name via a prompt. By specifying *'useCurrentFile'*, the restart files found in the *template* directory are used instead. To prevent the failure of a simulation due to core incompatibilities (e.g, number of processors), the *data.head* for the current simulation is checked against the corresponding input file for the simulation that produced the restart file.

The variable *tail* stores a string that specifies whether the *log.liggghts* file that is created by the given simulation should be output live to the Command Window of MATLAB. Upon finishing the simulation or an error is encountered, the tail command will automatically terminate, returning control of the MATLAB Command Window to the end-user.

5.2.2 Obtaining Results

To download and prepare the results from a simulation for analysis, the *liggghtsImportScript.m* script is used. Three user defined variables are available: *projectname*, *download* and *vtk_num*.

The *projectname* variable is used to specify which project to download from the cluster. If this project already exists in the *output* directory, the user will be prompted if they wish to overwrite the existing downloaded project or to create a new directory. If a new directory is created, the current date and time will be appended to the name of the new project directory.

Sometimes, the user may not want to re-download the results for a given simulation but still enable their formatting for analysis; the *download* variable can be used to control this behaviour. By default, if *download* is not defined by the user, the project will be re-downloaded.

The final user defined variable, *vtk_num*, is used when saving the results of a simulation to a MAT-file. It may be desirable to save all of the results from a simulation (*'all'*), just the data from the rotation of the drum (*'rotating'*) or save both as two separate files (*'both'*). Without being specified by the user, the default option is *'all'*. The MAT-files are saved in the main simulation directory.

5.2.3 Format of Results

The results of a simulation are saved as a single MATLAB table, *vtk_data_out*, subdivided into the headings: *particle_files*, *data*, *type_data* and *particle_properties*. The *particle_files* heading is a structure recording all the VTK particle files imported for the given simulation in natural sort order. The *data* heading contains the majority of the simulation data. For each time step, the position, velocity, force, angular velocity and torque for every particle is recorded, as well as the corresponding drum rotational speed. Each particle in *data* is refereed to by a unique ID number between one and the number of particles in the simulation. The corresponding particle type (1 is polypropylene, 2 is glass) for each particle ID is recorded in *type_data*. The final heading, *particle_properties*, records the type, radius, density and particle count of both particle types for reference.

6 Calibration

An overview of the calibration procedure is provided below. It is recommended to read through the documentation provided with the original version of DEcalioc found at its GitHub repository (`https://github.com/DECALIOC/DEcalioc`), as well as Section 3.3 that describes the set up and submission of DEcalioc on the cluster before continuing. The basic structure of DEcalioc was not changed and follows the same procedure outlined in the documentation included with DEcalioc. Beyond changing the settings outlined in Section 6.1, the main changes made were to the DEM models, the initialisation of each simulation and the functions for obtaining their results.

Three separate DEM models were added, *rotatingdrum20*, *rotatingdrum35* and *rotatingdrum50*, corresponding to a drum filling degree of 20, 35 and 50 percent respectively. Beyond the differences in filling degree, each model was identical. To improve the efficiency of each simulation and minimise the influence of a changing segregation index on the dynamic angle of repose, a restart file was created for each filling degree of a premixed drum. A brief overview of the simulation used to create the restart files (*2D_non_periodic.restart*) can be found in Section 4.1. The contents of the drum was mixed at 40 RPM for 40 seconds and the particles allowed to settle after rotation for a further 6 seconds before the restart file was created.

The cluster has two separate queues that a job can be submitted to: the default and long queues. The former has a limit of 160 processor cores per user and a maximum wall time of 200 hours (8.3 days). The latter is limited to only 16 processor cores but has a maximum wall time of 1,200 hours (50 days). Without making changes to DEcalioc, submitting the calibration procedure to the default queue would lead to the wall time being reached before calibration could finish, while the limited number of processors available on the long queue would also prevent calibration from finishing within its maximum wall time. To avoid this problem, each DEM simulation was submitted as a separate job to the default queue, while the main DEcalioc Octave script, *DEcalioc.m*, that had overall control of the calibration procedure was run as a separate 'master' job on the long queue. The supplementary Octave functions added to obtain the necessary information for the parallelisation of the calibration procedure can be found in the *infrastructure* directory of DEcalicoc.

For each DEM model, the drum was rotated at three different rotational speeds (5, 20 and 40 RPM) for 20 seconds at each speed. Images of the profile of the drum were produced periodically for the latter 10 seconds of each rotational speed. Additional Octave functions were added to read-in each sequence of images and determine the average dynamic angle of repose for a given filling degree and rotational speed combination. Three different algorithms were implemented by Theodore to obtain the dynamic angle of repose, although only the most

basic algorithm, *noModel*, was used (please see Section 6.2 to reimplement one of the more sophisticated algorithms). The additional functions for obtaining the dynamic angle of repose can found in the *OctaveFuns* directory of each DEM model.

6.1 Calibration Settings

The majority of the settings that the user must specify for the calibration of the rotating drum can be found in the *loadInput.m* Octave function. The alterations made to DEcalioc have changed the meaning and purpose of some of the variables and rendered others as simply dummy placeholders. The new definitions, broken down by variable type, are outlined below.

Input

- *Input.model*$\{X\}$ stores the folder name of model X. Currently, there are three different models, *rotatingdrum20, rotatingdrum35* and *rotatingdrum50*, each corresponding to a drum filling degree of 20, 35 and 50 percent respectively.
- *Input.cpu*$\{X,1\}$ is not in use. The number of processor cores assigned to a simulation is instead defined in the *data.head* file of a given model. Please leave equal to 1.
- *Input.maxCPU* stores the number of simulations that are queued to the cluster as individual jobs. The hard limit is 512 jobs. It is recommended that the product of *Input.maxCPU* and the number of processor cores assigned to a given simulation (in this case, 4), does not exceed the number of processor cores available per cluster user (160). It is believed that the serial DEM optimisation initially creates *Input.maxCPU* simulations at the start of each iteration, regardless of the value of *optim.tolfun*.
- *samplesPerVar* stores the number of sample simulations to be run per design variable. It is recommended to specify 5–10 samples per design variable [6].
- *numVar* stores the number of design variables to be calibrated.
- *Input.numOfSam* stores the number of sample simulations run **per** model to create the initial Kriging model.

Optimisation Settings

- *optim.targetVal{X}(Y)* stores the target value Y for the dynamic angle of repose for model X. Per model, three angles of repose are specified, one for each of the three different rotation speeds that are applied to the drum. The order of *optim.targetVal{X}(Y)* and *optim,tolRes{X}(Y)* must match the order of the dynamic angles of repose written to the output *XXXX_aaa_XXXXX_angleRepose.txt* file in the *analysis* directory for a given simulation.

- *optim.tolRes{X}(Y)* stores the relative tolerance used for the residual breaking condition. If all of the residuals for the variables specified by *optim.targetVal* are below their respective tolerances, optimisation will stop. Currently, this has been set to 1 standard deviation (1.5 °) of the dynamic angle of repose for the given drum rotational speed and filling degree. It is expressed as a decimal fraction.

- *optim.tolfun* stores the relative stopping criteria for the cost function. If the sum of the squares of the change in the cost function is below *optim.tolfun*, the optimisation run will stop. It is expressed as a decimal fraction.

- *optim.maxIter* stores the maximum number of iterations allowed for the DEM-based optimisation. It must be a positive integer.

- *optim.maxFunEvals* stores the maximum number of evaluations of the cost function (*costFunction.m*) allowed for the DEM-based optimisation. It must be an integer.

- *optim.WRL* stores the weighting factor of the Rayleigh time step size during residual optimisation. As the Rayleigh time step is largely dependent on the Young's modulus [4], a known property; the weighting factor was set to zero to remove its inclusion. To reinclude this weighting factor, lines 56 and 57 of the *costFunction.m* function will need to be updated.

Model Variables

- *modelVars.poissonsRatioP* stores the fixed value of the Poisson's ratio for particle type P that is written to the input *data.head* file for a given simulation $(-)$. It is not in use.
- *modelVars.radiusP* stores the fixed radius of particle type P that is written to the input *data.head* file for a given simulation (m). It is not in use.
- *modelVars.youngsModulusP* stores the fixed Young's modulus of particle type P that is written to the input *data.head* file for a given simulation (Pa). It is not in use.
- *modelVars.densityP* stores the fixed density of particle type P that is written to the input *data.head* file for a given simulation (kgm^{-3}). It is not in use.
- *modelVars.percentRayleigh* stores the fraction of the Rayleigh time step that is used to define the simulation time step. It is not in use.

Design Variables

- *assign*$\{Z\}$ stores the name of design variable Z. It is important that the variable names specified here match the names of the corresponding variables in the input *data.head* file. Further information on the contents of the input *data.head* file can be found in Section 4.2.
- *paramLims* stores the limits of the design variables. Each column Z of *paraLims* refers to its corresponding design variable *assign*$\{Z\}$. The minimum and maximum limits are specified in rows one and two respectively.

Note: The cost function includes the Rayleigh time step, irrespective of the weighting factor defined by *optim.WRL*. Thus, the variables *poissonRatioP*, *radiusP*, *youngsModulusP*, *densityP* and *percentRayleigh* **must** be defined as either fixed model variables or as design variables. For the calibration of the rotating drum, all of these variables are known and are already defined in the *data.head* input file. These variables are assigned arbitrary values that are still written to the *data.head* file, but have no bearing on the simulation. The side effect is that the residual for the Rayleigh time step is undefined, causing the calibration procedure to fail, even with a weighting factor of zero. Line 56 of the *costFunction.m* function defining the residual for the Rayleigh time step was commented out and replaced with line 57 to prevent this from occurring. If at a later date the Rayleigh time step is to be calibrated, these lines of code will need to be returned to their original state. The Rayleigh time step is calibrated for the minimal working example provided with the original version of DEcalioc and should be used as a basis for reimplementing this feature correctly.

6.2 Reincorporating the *threeAngles* Algorithm

Due to the large fluctuations in the dynamic angle of repose obtained using the *threeAngles* algorithm, only the linear dynamic angle of repose is used for calibration. The few small changes that must be made to the DEcalioc to reimplement this algorithm are outlined below.

The target values for the dynamic angles of repose are defined in the *loadInput.m* function (*DEcaliocDrum/DEcalioc/loadInput.m*). The variables *optim.targetVal* and *optim.tolRes* should be updated to include the experimentally determined non-linear dynamic angle of repose and its associated tolerance for the corresponding filling degree and drum rotational speed respectively. The array indexing for each of these variables will need to be updated accordingly.

The next two functions that must be updated can be found in each of the three model directories (*DEcaliocDrum/DEcalioc/DEMmodels/rotatingdrumXX/OctaveFuns*). Please make sure to update the functions for **all models**. The *makeResults.m* script is called after a simulation has finished to obtain the average dynamic angles of repose for the given simulation and write the results to a text file. The variable *angleModel* sets the algorithm used to obtain the dynamic angles of repose. This should be changed from 'noModel' to 'threeAngles' (unfortunately, the third algorithm, *lineTwoCircles*, does not currently work with Octave). The final function, *getResults.m* is used to read the text file containing the dynamic angles of repose into DEcalioc. The *noModel* algorithm produces three angles of repose per simulation, one for each drum rotational speed, while *threeAngles* produces six (linear and non-linear dynamic angles of repose per drum rotational speed). Thus, the variable *res* should be updated to read all six lines of the output file, rather than just the first three.

7 Additional Software

There are several other additional pieces of software that were used with LIGGGHTS. A brief explanation of their purpose is provided below as well as links to where further information can be found.

7.1 WinSCP

Windows **S**ecure **C**o**p**y (WinSCP) is an open-source, file transfer client, allowing for easy file transfer between a local and remote computer. Additionally, remote files can be edited locally and automatically saved back to the remote computer, negating the need of the somewhat clunky interface of editing files using the GNU nano terminal-based text editor. See `https://winscp.net/eng/index.php` for more information.

7.2 AutoPuTTY

When multitasking, the single terminal that can be open with PuTTY can be somewhat limiting. By connecting through PuTTY, AutoPuTTY allows for multiple terminal instances of the cluster to be open simultaneously, with obvious benefits. AutoPuTTY can be found at: `https://r4di.us/autoputty/`.

7.3 Gmsh

To create the CAD files used to represent the drum geometry, the open-source, finite-element mesh generator Gmsh was used. Points in space are defined and linked together as lines, surfaces and/or volumes to create a model. A mesh representation of the model is then created and saved as an STL files for use in LIGGGHTS. For more information, please see: `http://gmsh.info/`. Basic issues with the resultant mesh file can be repaired at: `https://www.trinckle.com/en/printorder.php`.

7.4 VMware Workstation Player

Rather than running and editing LIGGGHTS simulations on the cluster, a virtualised Linux machine can be used instead. VMware Workstation Player is a piece of platform virtualisation software, capable of emulating a Linux machine at near native speed. On low-end PC hardware, it is recommended to use a lightweight Linux distribution (for example: Xubuntu). For additional information, please see: `https://www.vmware.com/products/workstation-player.html` and `https://xubuntu.org/`.

7.5 GitKraken

Although GitHub repositories can be downloaded as compressed ZIP files, the easiest way to manage a repository is to use a Git client. GitKraken is a free, cross-platform Git GUI client that makes the process of downloading, forking and making changes to a project far easier than using the command line. Please see `https://www.gitkraken.com/` for more information.

7.6 ParaView

Using the output VTK (particle information) and STL (geometry information) files from a LIGGGHTS simulation, the simulation can be visualised in 3D using ParaView at each time step. The software is fairly straightforward to use, but visualising the particles as spheres, rather than discrete points is not immediately obvious. How to change this is described below in Section 7.6.1. Additional information for ParaView can be found at: `https://www.paraview.org/`.

7.6.1 Rendering Particles as Spheres, Sorted by Type

With the output files imported into ParaView, select *particles_** from the *Pipeline Browser* and then click on the Glyph icon found in the toolbar above. This should create a sub-element of *particles_** in the *Pipeline Browser* called *Glyph1*. With *Glyph1* selected, the following properties should be changed:

- Glyph Source
  - Glyph Type $\Longrightarrow$ Sphere
  - Radius $\Longrightarrow$ 1
  - Theta Resolution $\Longrightarrow$ 12 (Improves resolution of spheres at a performance loss.)
  - Phi Resolution $\Longrightarrow$ 12 (Improves resolution of spheres at a performance loss.)
- Scale
  - Scale Array $\Longrightarrow$ radius
  - Scale Factor $\Longrightarrow$ 1
- Masking
  - Maximum Number of Sample Points $\Longrightarrow$ 500000 (If this number is too low, some of the particles will not be represented by a sphere; however, if this number is too large, the performance decrease in the visualisation may lead to ParaView crashing. Please select a value accordingly.)

With the above properties changed, click *Apply*. Grey-white spheres should now be visible on-screen. if this is not the case, The greyed out eye symbol next to each item in the *Pipeline Browser* should be selected to add each element to the visualisation. If there are points representing particles without a masking sphere, the *Maximum Number of Sample Points* should be increased. To separate the particles by type, return to the properties menu of *Glyph1* and change *Coloring* from *Solid Color* to *type*. The particles should now be rendered as either blue (type 1 - polypropylene) or red (type 2 - glass) spheres. To edit the colour of the particles to match their actual colour, select *Edit* under the *Coloring* heading to open up the *Color Map Editor* panel. Make sure that the setting *Interpret Values as Categories* is ticked. Under *Annotations*, the colour and label of each particle type can be changed. As defined in the *in.rotatingdrum* and *in.rotatingdrumRestart* scripts, the polypropylene and glass particles have a *Value* of *1* and *2* respectively; the *Annotation* can be updated accordingly. The colour of each particle type can be updated by double-clicking on the circle of colour to the left of each particle *Value*.

8 Useful Commands (Cheat Sheet)

A brief summary of some of the more useful Linux commands for a LIGGGHTS DEM simulation are collected below as a convenient reference point. As there already exists a plethora of excellent guides online for using Linux [7], basic Linux commands will not be covered.

8.1 Log in to the Cluster

```
1   ssh -X USERNAME@hypnos5.fz-rossendorf.de
```

Where:

- `ssh` $\Longrightarrow$ Command for SSH client.
- `-X` $\Longrightarrow$ Enables X11 forwarding. This allows for securely run graphical applications (e.g., gedit) installed on the cluster to be viewed locally with the appropriate GUI. Without additional software, this will not work with PuTTY on Windows (see `https://sourceforge.net/projects/xming/`).
- `USERNAME` $\Longrightarrow$ Username for the Hypnos server.
- `hypnos5.fz-rossendorf.de` $\Longrightarrow$ Address for the Hypnos cluster.

8.2 Run a Bash Script

```
1   ./path/to/bash/script/bashScript.sh
```

By navigating to the directory of the bash script, the command `./bashScript.sh` will also work. The text file containing the bash script should start with: `#!/bin/bash` to specify that the bash interpreter should be used to interpret to the commands given in the shell script.

8.3 Submit a Job to the Cluster

```
1   qsub -N jobName -q default -l nodes=1:ppn=8 -l walltime=4:00:00 -I
```

Where:

- `-N jobName` $\Longrightarrow$ Set the name of job to *jobName*.
- `-q queueName` $\Longrightarrow$ Select the default queue (see: `https://www.hzdr.de/db/Cms?pOid=29813&pNid=852` for a list of available queues).
- `-l nodes=1:ppn=8` $\Longrightarrow$ Request one node and 8 processors per node (PPN). For a LIGGGHTS simulation, the number of nodes should be kept equal to 1.
- `-l walltime=4:00:00` $\Longrightarrow$ Set job wall time to 4 hours. The maximum wall time possible depends on the job queue (see `https://www.hzdr.de/db/Cms?pOid=29813&pNid=852`).
- `-I` $\Longrightarrow$ Set as an interactive job (allow for input from the command window). Closing the terminal while an interactive job is running will terminate it.

8.4 Run a Single LIGGGHTS Simulation from the Terminal

Rather than using the MATLAB functions to submit a simulation to the cluster, a simulation can be started manually, directly from the cluster.

```
1   cd /path/to/LIGGGHTS/script/
2   mpirun -np 4 ~/MYLIGGGHTS/src/lmp_auto < in.LiggghtsScript
```

Where:

- `-np 4` $\Longrightarrow$ Set the number of processors for the simulation to four. If first submitted as a qsub job submission, the number of processors specified by the `mpirun` command should be equal to or less than the number of processors assigned to the job.
- `~/MYLIGGGHTS/src/lmp_auto` $\Longrightarrow$ Set the path to the LIGGGHTS executable, *lmp_auto*. Alternatively, the executable can be first copied to the directory containing *in.LiggghtsScript* and the simulation started using the command `mpirun -np 4 lmp_auto < in.LiggghtsScript`.
- `lmp_auto < in.LiggghtsScript` $\Longrightarrow$ Run the LIGGGHTS simulation script: *in.LiggghtsScript* using the LIGGGHTS executable.

8.5 Tail a Log File

The progress of the simulation can be tracked from the terminal output itself when the job is run interactively or by tailing the log.liggghts file produced by the simulation.

```
tail -n +1 -f log.liggghts
```

Where:

- `-n +1` $\Longrightarrow$ Output all lines beginning with line 1. Alternatively, `-n 5` would only output the last 5 lines of the file (default is 10).
- `-f` $\Longrightarrow$ The tail command will run forever, automatically printing new data to the terminal when found.
- `log.liggghts` $\Longrightarrow$ File to be tailed.

8.6 Cluster Job Management

View all your active job submissions.

```
qstat -u $USER
```

Delete a single job submission.

```
qdel 1234567.hypnos3
```

To copy and paste the name of a job from the output of `qstat -u $USER`, double left-click the name of the job to highlight it and right-click to paste it to the command line when using PuTTY or middle mouse-click when using Linux.

Delete all your active job submissions.

```
qselect -u $USER | xargs del
```

References

[1] Simon Tatham. *PuTTY FAQ*. 2019. URL: https://www.chiark.greenend.org.uk/~sgtatham/putty/faq.html#faq-what (visited on 09/24/2019).

[2] DCS Computing GmbH. *2. Getting Started — LIGGGHTS v3.X documentation*. 2016. URL: https://www.cfdem.com/media/DEM/docu/Section_start.html (visited on 11/28/2019).

[3] John Eaton W. *GNU Octave*. Jan. 2019. URL: https://www.gnu.org/software/octave/ (visited on 11/28/2019).

[4] Michael Rackl and Kevin J. Hanley. "A methodical calibration procedure for discrete element models". en. In: *Powder Technology* 307 (Feb. 2017), pp. 73–83. ISSN: 00325910. DOI: 10.1016/j.powtec.2016.11.048. URL: https://linkinghub.elsevier.com/retrieve/pii/S0032591016308403 (visited on 08/28/2019).

[5] CFDEMresearch GmbH. *read_restart command — LIGGGHTS v3.X documentation*. 2016. URL: https://www.cfdem.com/media/DEM/docu/read_restart.html (visited on 08/29/2019).

[6] Michael Rackl. *Re: DEcalioc - Samples needed for Creating Design of Experiments*. Personal Email. Sept. 2019.

[7] Dave McKay. *37 Important Linux Commands You Should Know*. en-US. 2019. URL: https://www.howtogeek.com/412055/37-important-linux-commands-you-should-know/ (visited on 09/26/2019).