Sleeping with the LIGGGHTS on

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

Tim Churchfield

September 30, 2019

Version 1.0

Contents

List of Figures

List of Tables

1 Introduction

The purpose of this document is to provide the necessary information and steps to successfully calibrate a LIGGGHTS DEM simulation, submit it to the computing cluster (Hypnos) and obtain the final results in a usable format. Sections 2 and 3 focus on the set-up required locally and on the cluster respectively. An overview of the various LIGGGHTS scripts used for the simulation of the rotating drum are covered in Section 4 and how these can be used in combination with MATLAB in Section 5. Although not an exhaustive list, Section 6 is intended to be a basic guide to some of the more useful Linux commands for navigating around the cluster and submitting jobs to the cluster. The final Section (Section 7) provides an overview of some additional pieces of software; although not necessary, they may be useful if changes are to be made to the simulation or further post-processing is required.

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 no strictly necessary for using PuTTY, it is a requirement to use the MATLAB scripts and functions designed to initialise and download simulation results from the cluster. Several additional utilities are bundled with PuTTY that are used to achieve these particular aims. 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. Run the installer file.

To begin, open PuTTY (if no shortcut was created on installation, the default installation location is: `C:\Program Files\PuTTY`). Navigate through the PuTTY Configuration window using the Category toolbar on the left hand side and change the following settings:

- 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 - Data
  - Auto-login username $\Longrightarrow$ your username

Return to the Session category and save the session settings. By loading these settings and clicking Open, you should be greeted with a terminal interface requesting your password. For further information, see: `https://help.dreamhost.com/hc/en-us/articles/215464538-How-do-I-configure-PuTTY-`.

2.1 Passwordless Log In

To allow for passwordless log in to the cluster when using PuTTY or the MATLAB scripts, a private/public key pair must be generated using the PuTTY Key Generator utility (PuTTYgen). An explanation on how public key encryption works can be found here: `https://ssd.eff.org/en/module/deep-dive-end-end-encryption-how-do-public-key-encryption-systems-work`. 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-`. **NOTE:** From a security point of view, it is highly 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 memory for the duration of the session. To begin, load Pageant and 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 directly used to negate the need for entering 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. This includes installing LIGGGHTS, Octave and DEcalioc to be able to run and calibrate LIGGGHTS simulation without conflict between each program.

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 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 due to a lack of administrative rights. 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 to function as expected.

3.1.1 Preparation

Log in to cluster

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

Set language

```
LANG=ENG
```

Obtain the latest version of LIGGGHTS from it GitHub repository

```
git clone https://github.com/CFDEMproject/LIGGGHTS-PUBLIC.git
```

Change the name of the default LIGGGHTS folder

```
mv LIGGGHTS-PUBLIC/ MYLIGGGHTS
```

Check for available environmental modules and loading relevant ones

```
module avail
module load cmake/3.10.1
module load gcc/7.2.0
module load openmpi/1.8.6
```

3.1.2 Compiling LIGGGHTS

Compile LIGGGHTS to produce the executable: *lmp_auto*

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

The above command **will** fail. This is to be expected. It creates a file called *Makefile.user* that contains numerous settings for compiling LIGGGHTS that can be edited. As far as I'm aware, 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. To begin, open Makefile.user using nano:

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

Using the arrow keys, navigate the line for `AUTOINSTALL_VTK`. Remove the comment symbol and edit it to be the following:

```
AUTOINSTALL_VTK = "ON"
```

To exit and save the file, type: `ctrl+X` (brings up menu to save changes), `Y` (save changes) and finally `Enter` to overwrite the existing file.

Clean-up the code by removing all the compiled object files from the source code before running the `make auto` command again.

```
make clean-all
```

With everything set-up, LIGGGHTS can now be compiled. Due to the length of time required to set everything up, it is faster to submit the compiling process to the cluster as a parallel job. To create the job and request the necessary CPUs:

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

Where:

- `-N makeauto` $\Longrightarrow$ Setting name of job to *makeauto*.
- `-q default` $\Longrightarrow$ Selecting default queue (see: `https://www.hzdr.de/db/Cms?pOid=29813&pNid=852`).
- `-l nodes=1:ppn=8` $\Longrightarrow$ Requesting one processing node and 8 processing cores.
- `-l walltime=4:00:00` $\Longrightarrow$ Setting job wall time to 4 hours.

- `-I` $\Longrightarrow$ Interactive job (allows for input to the command window).

If this command fails to bring up the following line: `USERNAME@laser043:~$` (or something similar), use `ctrl+c` to cancel the job and try again. If this continues, it is likely that there is a very large queue of job submissions for the cluster. They say patience is a virtue...

With the job request successfully submitted, LIGGGHTS can be compiled using the 8 processing cores requested.

```
cd ~/MYLIGGGHTS/src
make -j 8 auto
```

During the compiling process, the VTK library is also installed. This can take a while to run so please be patient.Once finished, the job can be killed.

```
exit
```

The path to the new VTK library needs to be added to *.bashrc*, a shell script that is run every time you log in to the cluster. Return to you home directory (`cd \~`) and open *.bashrc* using nano in the manner shown above. Add the following line updated with your username. Save and close the file once finished.

```
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 you r home directory, open the file *own.modules* in the normal manner 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 you don't need (As this can potentially create conflict between modules).

```
module load cmake/3.10.1
module load gcc/7.2.0
module load openmpi/1.8.6
```

To reap the rewards of your labour, exit the cluster using the command `exit` and log in again. To make sure everything is work as expected, one of the example simulations provided with LIGGGHTS can be used as a test case (in this case *chute_wear*).

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

Unfortunately, since we do not have administrative rights, we cannot create a symbolic link for LIGGGHTS. Thus, it is easier to copy the LIGGGHTS executable, *lmp_auto* from the *src* directory into the simulation directory.

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

To double check 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, the simulation directory returned to and the simulation initialised. The `mpirun` command allows for the simulation to be run in parallel across multiple CPUs, improving performance.

```
qsub -N DEM -q default -l nodes=1:ppn=4 -l walltime=4:00:00 -I
cd ~/MYLIGGGHTS/examples/LIGGGHTS/Tutorials_public/chute_wear/
mpirun -np 4 lmp_auto < in.chute_wear
```

Where:

- `-np 4` $\Longrightarrow$ Running simulation using four CPUs.
- `lmp_auto < in.chute_wear` $\Longrightarrow$ Running 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:

```
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.

3.1.3 Adding the Contact Model to the Whitelist

NOTE: Unfortunately, the following instructions do not work for the model used in the DEM simulation of the rotating drum. Other users of LIGGGHTS have posted similar issues on the LIGGGHTS forums. Please feel free to try anyway as the performance improvement could be significant.

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 and view the log.liggghts file produced during the simulation. If the following is found in the log file when the contact model is applied, 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.
```

If using the script genAutoExamplesWhitelist.sh fails, the contact model can be added manually to the whitelist. Add the contact model to the bottom of the whitelist file *style_contact_model_user.whitelist*, found in the *src* directory of *MYLIGGGHTS*. The entry for the rotating drum DEM simulation is the following:

```
GRAN_MODEL(HERTZ, TANGENTIAL_HISTORY, COHESION_OFF, ROLLING_EPSD2,
    SURFACE_DEFAULT)
```

With the whitelist updated, LIGGGHTS needs to be recompiled. As the VTK library has already been added to the cluster, it is not necessary to submit the compilation of LIGGGHTS as a job as the process should be significantly quicked than the first time.

```
rm lmp_auto
make clean-all
make -j 8 auto
```

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

3.2 DEcalioc

Discrete **E**lement **ca**libration using **LI**GGGHTS and **Oc**tave (DEcalioc) is a piece of software written to automagically calibrate a DEM simulation with minimal user preparation and input. DEcalioc is written using Octave and is designed to work with LIGGGHTS, both of which are free and open-source pieces of software. Several additional modules specific to Octave are required for DEcalioc, preventing MATLAB being used instead. In the following sections, the set-up of Octave and DEcalioc is explainedm, such that the minimal working example provided with DEcalioc can be run without issues. For further information regarding DEcalioc, please refer to its GitHub Repository: `https://github.com/DECALIOC/DEcalioc`.

3.2.1 Octave Set-Up

The main challenge with using DEcalioc is making sure that the overlapping environmental module dependencies for Octave match those of LIGGGHTS. For this reason, the following module versions **must** be used. Please add those not already listed in *own.modules* to the file in the following order as previously explained in Section 3.1.1. These modules are everything needed to run LIGGGHTS, Octave and DEcalioc correctly. You will need to exit and log back into the cluster for the newly listed modules to be loaded.

```
module load gcc/7.2.0
module load openmpi/1.10.2
module load gnuplot/5.2.0
module load glpk/4.64
module load octave/4.2.2
```

The following packages for Octave need to be installed on the cluster for the simulation to work correctly. Due to their dependencies, it is recommended to install the packages in the order they are listed. The packages can be found at: `https://octave.sourceforge.io/`. 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/`.

```
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`

Once downloaded, the *tar.gz* files should be placed in your home directory (`cd ~`) on the cluster using an SSH program or copied from the HOME directory. Octave can then be started and the packaged installed:

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

To view a list of the packages installed the command `pkg list` can be used when Octave is running.

3.2.2 DEcalioc Set-Up

With Octave correctly set-up with the necessary packages, DEcalioc can be downloaded from its GitHub Repository and renamed (if you wish).

```
cd ~/
git clone https://github.com/DECALIOC/DEcalioc.git
mv DEcalioc/ MYDecalioc
```

DEcalioc comes with a minimal working example to test if everything has been set-up correctly. To initialise the DEM simulation during calibration, DEcalioc calls the file *runscript* found in the directory of the minimal working example (`~/MYDEcalioc/DEcalioc/DEMmodels/Lift100/`). In its default state, *runscript* contains the following command:

```
liggghts < in.Lift100
```

If a symbolic link for LIGGGHTS could be created, this would work correctly; however, as stated in Section 3.1, a symbolic link for LIGGGHTS cannot be created. As a result, *runscript* must be edited to read the followijng command below.

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

Where,

- `#!/bin/bash` $\Longrightarrow$ Convention so the shell knows what kind of interpreter to run.

- `~/MYLIGGGHTS/src/lmp_auto < in.Lift100` $\implies$ Runs the *in.Lift100* using the LIGGGHTS executable previously compiled and located at: `~/MYLIGGGHTS/src/`.

With the *runscript* file updated, the minimal working example can be run. The head Octave script for DEcalioc that begins the calibration procedure is called *DEcalioc.m*.

```
qsub -N octave_Cali -q default -l nodes=1:ppn=8 -l walltime=49:00:00 -I
octave
cd MYDEcalioc/DEcalioc/
DEcalioc
```

If everything is working correctly, the calibration procedure for the minimal working example should take a few hours to finish. For any other calibration simulation, please make sure the *runscript* file has also been edited appropriately such that the *lmp_auto* executable can be found during the calibration process. Please refer to the DEcalioc documentation for how to set-up a new DEM simulation to be calibrated.

3.3 Calibration of the Rotating Drum

The documentation provided with DEcalioc provides an excellent explanation on how to integrate a new DEM model with DEcalioc and should be the starting place if changes are to be made to DEcalioc. For the calibration of the rotating drum, a brief overview of the changes made are provided below.

NOTE: As of 30/09/2019, the full list of changes has not been finalised.

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* is the primary script containing the commands used to define the simulation, from the filling of the drum to the rotation of the drum to mix the particles. Before the final rotation of the drum is initialised, a restart file is created. This file is used by the *in.rotatingdrumRestart* script to continue the simulation, potentially with a different set of properties as defined by *data.head* (see Section 5.2). 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 an end-on viewpoint of the drum. These are primarily used for calibration of the drum using DEcalioc as the quality of the final image is somewhat lacking. For better quality images and greater flexibility in their composure ParaView can be used instead (see Section 7.5).

4.1 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 rotation of the drum during mixing is controlled by specifying the RPM of the drum at particular instances in time. 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 RPM of the drum at that time. This is shown graphically in Figure 1 below.

0
0
1
40
3
40
5
40
6
0
7
0

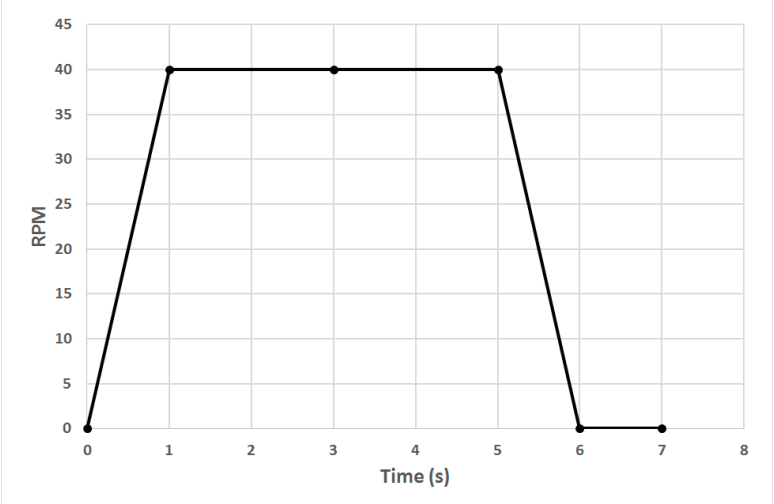

Figure 1: Sample Profile of rpm_drum.txt

4.2 data.head

The *data.head* file allows the user to change key settings and properties for *in.rotatingdrum* and *in.rotatingdrumRestart* as well as the job submission on the cluster. Five separate materials are defined in the simulation, each given a shorthand name. They are defined below in 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 for each material type 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 in the subsequent sections.

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

Simulation Settings

- *simtype* specifies the simulation type (i = initialisation, r = restart). This is used by the MATLAB script *liggghts_initialisation_script* to either upload *in.rotatingdrum* (*simtype* = *i*) or *in.rotatingdrumRestart* (*simtype* = r) to the cluster to be used as the primary LIGGGHTS simulation script (See Section 5.2)‘.
- *walltime* stores a string that defined the maximum wall time when submitting the simulation job to the cluster. It is in the format: hhhh:mm:ss.

- *simdim* specifies 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 CPUs assigned to the simulation. NOTE: If the number of CPUs exceeds 16, the bash script *job.script* must be edited to increase the number of nodes to greater than 1. With the number of particles in the simulation, this is unlikely to be necessary but is worth pointing out.

- *autoproc* specifies if the processor layout should be automatically set by LIGGGHTS, overriding *(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 CPUs 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. For the public version of LIGGGHTS, this does not update as the simulation progresses so please choose wisely as it may otherwise hinder the performance of the simulation. Ideally, the number of particles per CPU should be equal across all CPUs.

- *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 calibration period). The images produced are primarily for calibration purposes using DEcalioc.

- *caliCount* stores the loop count at which calibration imaging begins. This is only applicable if calibration imaging is selected in *imaging*. Based on *rpm_drum.txt*, the rotation of the drum is defined by points in time where the RPM of the drum is specified. Each additional time-RPM pairs after the initial conditions pair defines one loop of the rotation of the drum. See Section 4.1 above for more information.

DEM Parameters

- *ts* stores the time-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, drum rotational speed) (s).

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 [2]. 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 ($-$).
- *CoSF_XX_YY* stores the coefficient of static 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 three through five and their same-material coefficients have been set as low as possible. Theoretically, as these materials are only used to represent the geometry in the simulation, their value is arbitrary.

Geometry

- *lengthdrum3D* stores the length of the full-sized '3D' drum (m).
- *lengthdrum2D* stores the length of the 'quasi-2D' drum (m).
- *widthdrum* stores the outside diameter of the drum (m).
- *iddrum* stores the inside 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).
- *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 by vibration (m).
- *frequencyvib* stores the frequency of the oscillation of the drum (Hz).
- *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 gravity from a horizontal to a vertical direction (s).
- *dividerspeed* stores the linear velocity of the divider when it is removed from inside the drum (m/s).

4.3 Syntax Highlighting

The best way to edit LIGGGHTS Scripts is to use the text editor gedit as plugin exists that allows for the syntax of LIGGGHTS scripts to be highlighted as appropriate. Gedit can be installed from the following link: `https://wiki.gnome.org/Apps/Gedit#Download` for your favourite operating system.

With gedit installed, the necessary language definition file can be found at the following link: `https://www.cfdem.com/gnome-syntax-highlighting`. Extract the contents of *liggghts.lang_.tar_.gz* and place the *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` and for Linux: `/usr/share/gtksourceview-3.0/language-specs`. To utilise this feature, open a LIGGGHTS script in gedit and go `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 in the subsequent sections.

5.1 Overview of Directories and Scripts

Their are two primary MATLAB scripts, *liggghts_initialisation_script* and *liggghts_import_script*, each with several additional supporting functions.

There are two distinct folder structures The folder structure for the MATLAB code and relevant directories can be found in Figure 2 and the MATLAB function dependencies in Figure 3. An explanation of the colour of each folder of function can be found 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	-

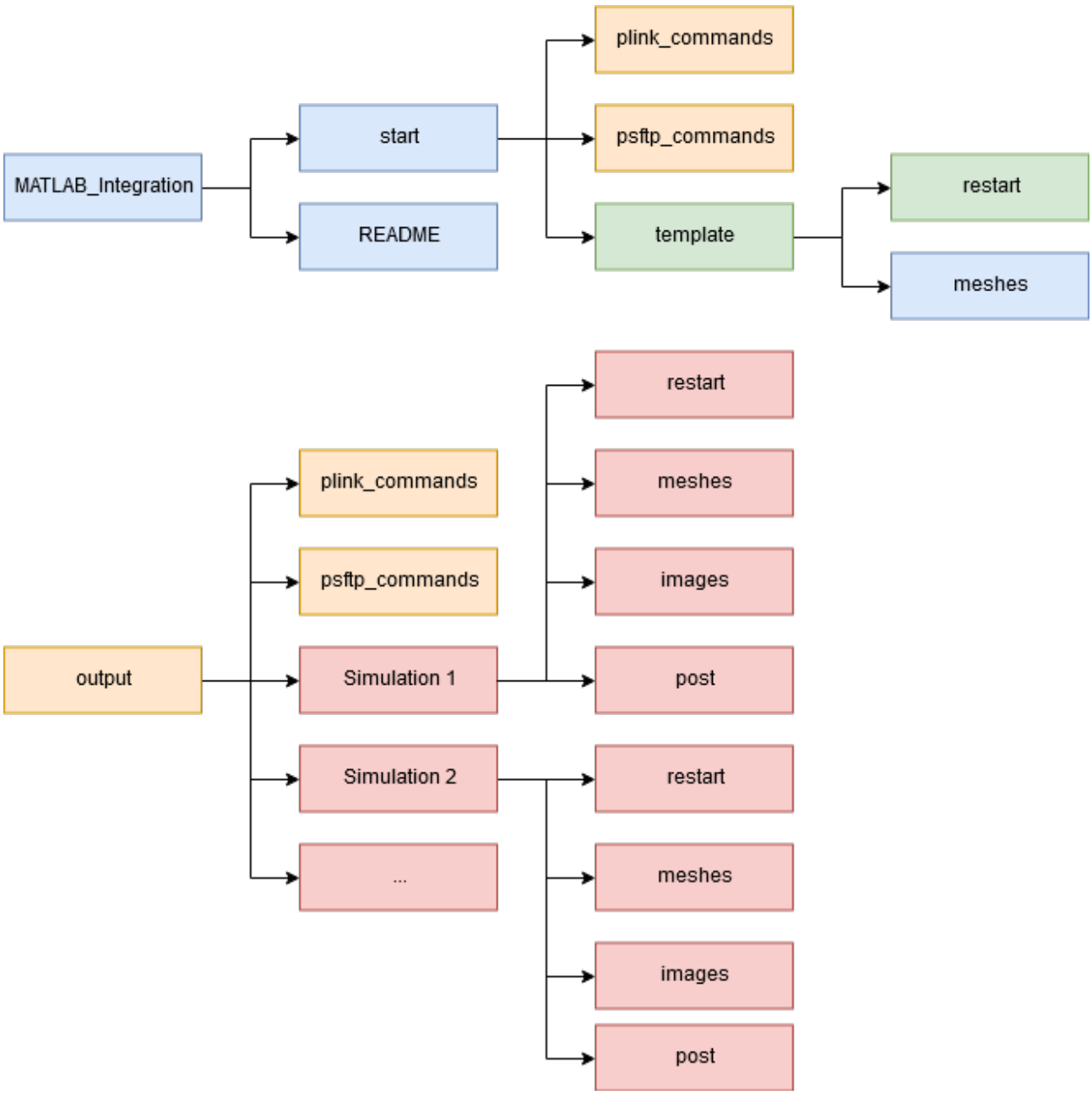

Figure 2: Folder structure for the MATLAB integration with LIGGGHTS

5.2 Using Restart Files

As discussed in Section 4, *in.rotatingdrum* produces a restart file that is utilised by *in.rotatingdrumRestart* to continue a simulation with the option to use a different set of properties as defined in *data.head*. Depending on the simulation settings, one of three different restart files are created:

- *2D_periodic.restart*
- *2D_non_periodic.restart*

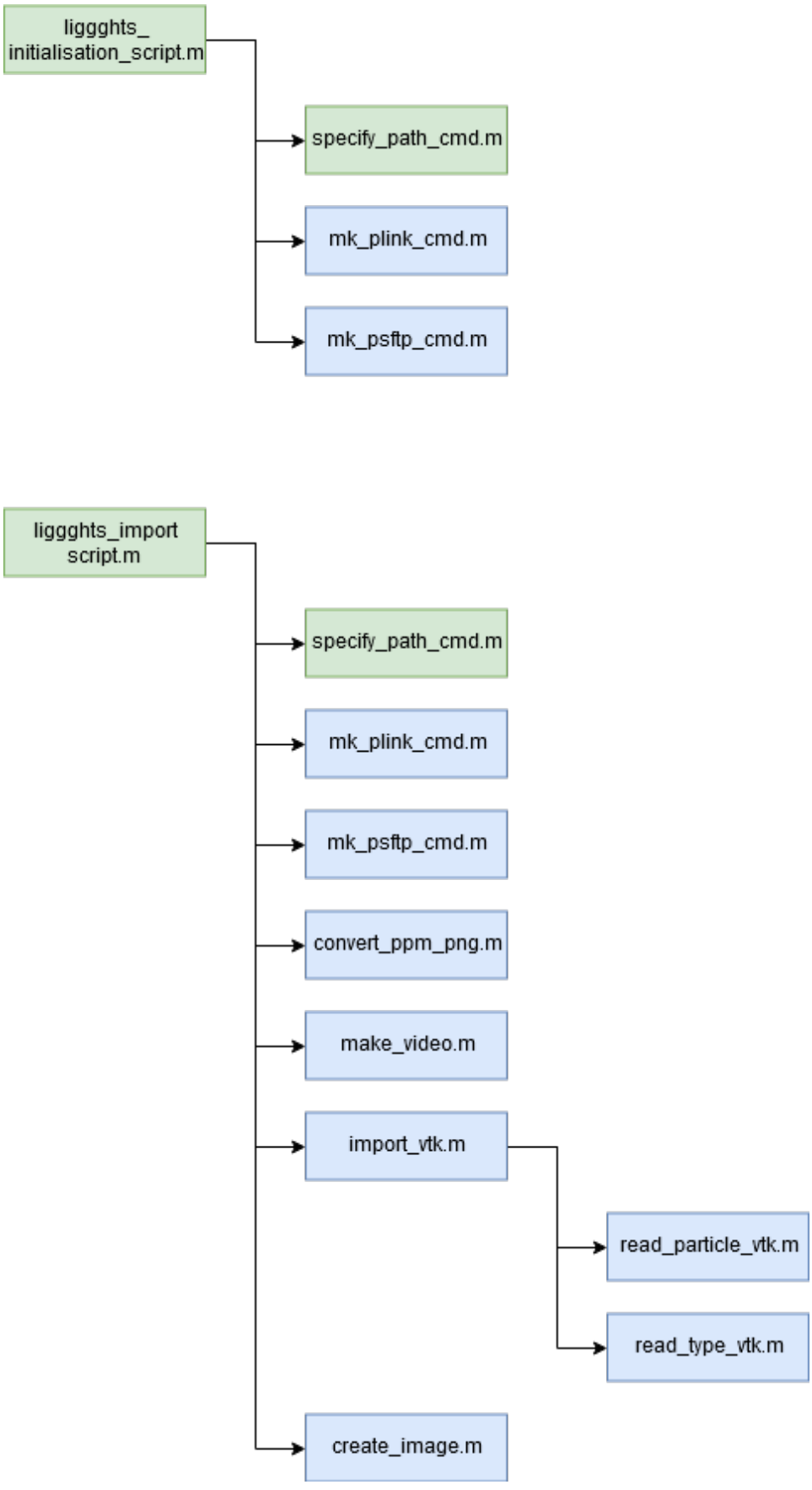

Figure 3: MATLAB Function Dependencies

- *3D.restart*

The restart files produced throughout the simulation can be found in the *restart* directory of the simulation directory (see Figure 2). Any existing restart file is overwritten when a new one is created by the simulation.

5.3 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 *MATLAB_Integration* directories on the local computer and finally the absolute path to the *LIGGGHTS-PUBLIC* (or equivalent) directory on the cluster. A more detailed explanation and formatting requirements are provided within the function itself.

5.3.1 Submitting Simulation to the Cluster

To submit a LIGGGHTS simulation to the cluster the *liggghts_initialisation_script* is used. There are two additional variables that can be defined as the start of the script: *projectname* and *tail*.

The variable *tail* stores a string that specifies whether the *log.liggghts* file that . 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.

5.3.2 Obtaining results from LIGGGHTS Simulation on the Cluster

6 Useful Commands

Throughout this document, many Linux commands were used without providing any explanation on their use. As there already exists a plethora of excellent guides online for how to use the Linux terminal [3], the basic Linux commands will not be covered here; instead, some for managing and submitting jobs to the cluster. Some of thi is a repeat but it is useful to have it all in one place Beyond submitting a job to the cluster or tailing (both of which were discussed in detail in Section Insert Text Here.

- `qstat -u $USER` $\Longrightarrow$ View details of all active job submissions.
- `qselect -u $USER | xargs qdel` $\Longrightarrow$ Delete all submitted jobs at once.
- `#!/bin/bash` $\Longrightarrow$

log in to cluster remotely

6.1 Submit Job to the Cluster

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 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/`.

7.4 Oracle VM VirtualBox

Rather than running LIGGGHTS simulations on the cluster, a virtualised Linux machine can be used instead. Oracle VM VirtualBox 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.virtualbox.org/` and `https://xubuntu.org/`.

7.5 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 individual time-step. The software is fairly straightforward to use, but visualising the particles as spheres, rather than discrete points is not obvious. How to change this is described below in Section 7.5.1. Additional information on ParaView can be found at: `https://www.paraview.org/`.

7.5.1 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* must

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 (polypropylene) or red (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*, 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*.

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] 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).

[3] 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).