# CONTENTS:

1 Introduction 3
   1.1 Implementation .............................................. 3

2 Installation 5
   2.1 Prerequisites ................................................... 5

3 Workflow 7
   3.1 How to run using cae ........................................... 7
   3.2 How to run using scripts ...................................... 16
   3.3 General instructions ........................................... 21

4 Plugins 23
   4.1 Plugin registration commands .................................. 23
   4.2 Plugin commands ................................................ 23
   4.3 Plugin form design .............................................. 23

5 Abaqus scripts 25
   5.1 Create rail ...................................................... 25
   5.2 Create wheel ..................................................... 25
   5.3 Create rollover .................................................. 25
   5.4 Make rail mesh periodic ....................................... 25
   5.5 Reload modules .................................................. 25

6 Python scripts 27
   6.1 Initial setup .................................................... 27
   6.2 Compiling user subroutines ................................... 27

7 Rollover library 29
   7.1 Rail (3d) ........................................................ 29
   7.2 Wheel (3d) ....................................................... 29
   7.3 Utilities (3d) ..................................................... 29
   7.4 General utilities ................................................. 32

8 Indices and tables 39

Python Module Index 41

Index 43
The rollover simulation methodology is published in Tribology International. If you use the tools or the entire package for your work, please cite [Meyer, K.A., R. Skrypnyk, M. Pletz (2021) “Efficient 3d finite element modeling of cyclic elasto-plastic rolling contact” Tribology International]
The rollover simulation package provides scripts and user subroutines to Abaqus Standard (i.e. quasi-static analysis) for simulating an elastic wheel rolling over a rail. The rail can have complex geometry and material behavior, but the mesh at the end faces must be periodic.

As shown to the right in the figure, the simulation effectively considers an infinite number of periodic cells, where the wheel spacing equals to the simulated rail length. The purpose of this setup is to obtain accurate displacements for much shorter rail lengths compared with not using the periodic boundary conditions.

Between each simulation, the wheel is mapped back to a new starting position. The wheel is modeled as a linear elastic super-element (substructure), removing redundant degrees of freedom. The wheel must be rotationally symmetric.

### 1.1 Implementation

The implementation consists of (1) a python library with functions to setup the Abaqus simulations and (2) fortran user subroutines for mapping back the wheel, as well as for modeling the wheel itself.

The python library can be called either via plugins from within the Abaqus CAE (*How to run using cae*), or from the command line using various input files (*How to run using scripts*). Modifications of the rail can be done from within Abaqus CAE, making it possible to use multiple materials, adding geometric features, changing the mesh, and so on.

Scripts are provided for combining the included fortran subroutines with additional subroutines, such as material models (see *Compiling user subroutines*).
In order to install (or rather setup) the library, the following steps are required. Before attempting to do so, please verify that your system meets all the Prerequisites.

- Download the repository using git: `git clone --recurse-submodules https://github.com/KnutAM/AbaqusRolloverSimulation.git`
- From the top level in the repository, run `scripts_py/setup.py`. This will, amongst others, create `abaqus_v6.env`, which you might modify in the next step.
- To make the plugins work, Abaqus needs to find them. Unfortunately, multiple locations are not supported. Therefore, you have two options:
  1. Add the following to `abaqus_v6.env`: `plugin_central_dir = '<path_to_rollover_directory>/plugins'` This can only be done for one folder, therefore, make sure you don’t require this for other plugins.
  2. Alternatively, copy the `plugins/rollover_plugin.py` to `%HOME%/abaqus_plugins` (Windows) or `~/abaqus_plugins` (Linux). This approach has the downside that any fetched updates from the online repositories containing changes to `plugins/rollover_plugin.py` will require manually repeating this step.
- Copy `abaqus_v6.env` to `%HOME%` or `~`. If you already have an environment file in your home directory, you need to manually merge the changes (most likely you can just append the contents of this new file to the old).

2.1 Prerequisites

The following programs must be installed:

- Abaqus, setup to compile and link user subroutines
- Python, version 2.7 or later

To verify that Abaqus works with user subroutines, run the following command: `abaqus verify -user_std`. Note, on Windows running user subroutines from within CAE might be a problem even if the above command works. In order to setup abaqus to work on Windows, you typically have to add something like the following to the `abaqus.bat` file:

```bash
@call ifortvars.bat intel64 vs2013
@call "C:\Program Files (x86)\Microsoft Visual Studio 12.0\VC\bin\amd64\vcvars64.bat"
```

But when opening a new Abaqus CAE session, `abaqus.bat` might not be called. If you have problems running from within CAE, you could add those lines to the file `launcher.bat` (used when opening Abaqus CAE) as well. To locate this file, right-click on the Abaqus CAE start menu item, and choose “Open file location”. This will likely
take you to a shortcut. Repeat for that shortcut, and you should come to the `launcher.bat`. Add the above code block to this file, before the call to `ABQLauncher.exe`. 
WORKFLOW

In order to create a simulation, three objects are required:

1. An Abaqus Model Database (.cae) file containing the rail
2. A folder with data for a wheel super element
3. A user subroutine, that includes the subroutines supplied in the present repository.

The two former, as well as the simulation itself, can be created using either plugins in Abaqus CAE or via scripts. Details for each method, in addition to some general instructions, are provided:

3.1 How to run using cae

The plugins commands are available from within CAE as:

- A toolbar, activate by Plug-ins → Toolboxes → Rollover
- Menu items, in Plug-ins → Rollover
3.1.1 Creating the rail

A basic rail is created with the command \texttt{Create rail...}, opening the following form.

The following table describes the options:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rail sketch file</td>
<td>The path to the rail sketch, see \textit{Creating a profile sketch} and \textit{The data folder and how to specify paths}</td>
</tr>
<tr>
<td>rail cae name</td>
<td>Name of the Abaqus Model Database (.cae) file to be created</td>
</tr>
<tr>
<td>rail length</td>
<td>The extrusion length of the rail</td>
</tr>
<tr>
<td>mesh size</td>
<td>The fine and coarse mesh size, separated by a comma</td>
</tr>
<tr>
<td>refine p1</td>
<td>The first point (x,y) defining the refine rectangle</td>
</tr>
<tr>
<td>refine p2</td>
<td>The second point defining the refine rectangle</td>
</tr>
<tr>
<td>sym_sign</td>
<td>If symmetry about the yz-plane is used, specify the x-direction (-1 or +1), pointing away from the material. Set to 0 if symmetry is not used.</td>
</tr>
</tbody>
</table>

This will create a rail part, which could look like the following.
The red region marks the refine rectangle. This rectangle is used to create a cell by partitioning the rail by extrusion in the z-direction. The generated cell serves two purposes. Firstly, it controls where the fine mesh is used. Secondly, the external faces belonging to the cell, except the end faces with normal in the z-direction, make up the contact region.

The rail can now be modified to suit the needs of the simulation, e.g. changing the geometry, mesh and material definitions. The requirements on the rail are given in *Modifying the basic rail*. Note that the tool to make a TET mesh periodic is available as a plugin:

*Plug-ins → Rollover → Tools → Periodize mesh* or on the Rollover toolbar:

### 3.1.2 Creating a wheel

A wheel is created with the command *Create wheel...*, opening the following form
The following table describes the options:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wheel sketch file</td>
<td>The path to the wheel sketch, see <em>Creating a profile sketch</em> and <em>The data folder and how to specify paths</em></td>
</tr>
<tr>
<td>wheel save folder</td>
<td>Name of the folder in which the wheel super element files should be saved.</td>
</tr>
<tr>
<td>mesh</td>
<td>The fine and coarse mesh size, separated by comma</td>
</tr>
<tr>
<td>use quadratic</td>
<td>0 for linear elements, 1 for quadratic elements</td>
</tr>
<tr>
<td>angle interval</td>
<td>The angular interval in which to retain wheel nodes. Measured in radians around x, relative the negative y-axis.</td>
</tr>
<tr>
<td>x contact interval</td>
<td>The x-interval in which to retain wheel nodes.</td>
</tr>
<tr>
<td>partition line y</td>
<td>The y-coordinate (in the sketch, typically negative) outside which the fine mesh should be applied.</td>
</tr>
</tbody>
</table>

Upon pressing OK, a wheel substructure is calculated. This can take considerable time, especially for fine meshes. It is therefore recommended to test first with a bit coarser mesh. It is particularly the fine mesh size that determines the size, as this is used to determine the angular interval to mesh the wheel. For the default settings with a very coarse mesh, the full wheel mesh looks like...
Note, however, that it is currently not supported to manually edit the wheel mesh. The motivation is that once the wheel only needs to be calculated once, and it is therefore not required to optimize the mesh.
3.1.3 Creating the simulation

A simulation is created with the command **Create simulation...** opening the following form:

![Create wheel form](image)

This form has multiple tabs, which are described by the following tables:

<table>
<thead>
<tr>
<th>Rail</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.cae file</td>
<td>The path to the rail Abaqus Model Database file (.cae)</td>
</tr>
<tr>
<td>shadow extents</td>
<td>Name of the folder in which the wheel super element files should be saved.</td>
</tr>
<tr>
<td>use ref pt.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Wheel</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>folder</td>
<td>The folder containing the wheel super element files</td>
</tr>
<tr>
<td>translation</td>
<td>The vector (x,y,z) which the wheel should be translated on import. Initially, the wheel center is at (0,0,0). The rail sketch determines the (x,y) position of the rail, and it starts at z=0 and ends at z=L, where L is the rail length specified above.</td>
</tr>
<tr>
<td>use ref pt.</td>
<td>If rail extension should be used, a reference point is required. Otherwise, fewer constraints are added creating a slightly more efficient simulation. Set to 0 for no reference point, 1 otherwise.</td>
</tr>
<tr>
<td>Contact</td>
<td></td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------------------------------------------------------</td>
</tr>
<tr>
<td>friction coeff</td>
<td>The friction coefficient for the contact</td>
</tr>
<tr>
<td>contact stiff</td>
<td>The constant contact stiffness used (penalty method)</td>
</tr>
</tbody>
</table>

### Loading

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>initial depression</td>
<td>How much to move the wheel control point in negative y-direction using displacement control, before switching to load control, in the first cycle.</td>
</tr>
<tr>
<td>time inbetween</td>
<td>Which step time to use for the initial steps and the steps when mapping back the wheel in each cycle.</td>
</tr>
<tr>
<td>inbetween max incr</td>
<td>Maximum number of increments for the above steps.</td>
</tr>
<tr>
<td>rolling length</td>
<td>The rolling length, should match the rail length.</td>
</tr>
<tr>
<td>rolling radius</td>
<td>The rolling radius used to convert slip to rotation speed.</td>
</tr>
<tr>
<td>max increments</td>
<td>Maximum number of increments for one rolling step</td>
</tr>
<tr>
<td>min increments</td>
<td>Minimum number of increments for one rolling step</td>
</tr>
<tr>
<td>num cycles</td>
<td>Number of cycles to simulate. Please read <em>Adding rolling cycles</em>.</td>
</tr>
<tr>
<td>cycles spec</td>
<td>The cycles for which a change in loading conditions are specified. Given as csv, matching “cycles spec”.</td>
</tr>
<tr>
<td>wheel load</td>
<td>The force applied to the wheel control point in negative y-direction. Given as csv, matching “cycles spec”.</td>
</tr>
<tr>
<td>speed</td>
<td>The linear speed for the wheel control point. Given as csv, matching “cycles spec”.</td>
</tr>
<tr>
<td>slip</td>
<td>The wheel slip s, such that ( \dot{\theta} x = (1 + s) \frac{v}{R} ) where ( \dot{\theta} x ) is the wheel control point rotation speed around x, ( v ) is the speed and ( R ) is the “rolling radius” Given as csv, matching “cycles spec”</td>
</tr>
<tr>
<td>rail ext</td>
<td>The rail extension at the end of the rolling cycle, varying linearly to this value. Given as csv, matching “cycles spec”</td>
</tr>
</tbody>
</table>

### Output

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The name of the field output request to be created</td>
</tr>
<tr>
<td>set</td>
<td>The rail set name to take field output data for. Additionally, the names “FULL_MODEL” (both rail and wheel) and “WHEEL_RP” (wheel reference point) are supported.</td>
</tr>
<tr>
<td>variables</td>
<td>Which variables to output, comma separated, to find the correct variables, see the string created when setting up a field output request from within CAE.</td>
</tr>
<tr>
<td>frequency</td>
<td>How often (in terms of increments) to save data</td>
</tr>
<tr>
<td>cycle</td>
<td>How often (in terms of cycles) to save data. If e.g. 25 if specified, output will occur at cycle 1, 26, 51, and so on.</td>
</tr>
</tbody>
</table>

The form can be run with the default settings, except changing the paths to the generated `rail_example.cae` and folder `wheel_example`, or moving them to the default path specified. This action will create the following mesh, where the wheel is modeled using membrane elements.

The default settings do not add any field output. In that case, Abaqus’ default field outputs will be used. Note that this choice can result in very a large output database file (.odb) if many cycles are simulated.

### 3.1. How to run using cae
3.1.4 Running the simulation from CAE

The standard user subroutine is added to the job, allowing to run the created job directly inside CAE. If running via the command line from a different folder (e.g. a computational cluster), please see Running simulation to ensure all required files are available. Using the command line is required if the input file was modified according to Adding rolling cycles.
3.1.5 Result

After successfully run with the default settings, the von Mises stresses in the rail at the middle of the rolling cycle become as shown here:

Note that the simulation time is rather long for this example, because the mesh on the rail has not been optimized. It is usually beneficial to use hexagonal elements.
3.2 How to run using scripts

This document describes the overall workflow for how to setup and run a rollover simulation.

The rollover simulation consists of an elastic wheel rolling over a rail. To apply the “wormhole” boundary condition for the wheel, a set of user subroutines (usub) are used. So in order to setup and run the simulation, the user must do the following steps

1. Create the rail (“Rail .cae file”)
2. Create the wheel (“Wheel folder”)
3. Compile user subroutines (“usub (.obj/.o file)”). Normally not required each time
4. Create the rollover simulation (“Input files”)

Most of the scripts rely on a .json formatted settings file, for further details on this format see The json format.

3.2.1 Create the rail

The rail is (typically) created by first calling the create_rail_3d.py abaqus script (see Create a basic rail). This creates a .cae file. This file can then be edited to adapt to the specific simulation (see Modifying the basic rail). The end product will be a .cae file that may be convenient to store in the data/rails folder in the repository (see The data folder and how to specify paths).
Create a basic rail

A basic rail is created by calling the abaqus script `create_rail_3d.py` from a folder containing a file `rail_settings.json`. This file should contain the following settings:

- "material" (optional): Describes the material model and parameters to be applied to the entire rail. See `Material specification`. If not given, an elastic material will be set.
- "rail_profile" (mandatory): Path to an Abaqus sketch, saved as a `.sat` file, describing the profile of the rail in the xy-plane. See `Creating a profile sketch` for further details.
- "rail_length" (mandatory): The length (z) of the rail to be created
- "rail_name" (mandatory): The name of the `.cae` file to be created
- "refine_region" (optional): `[[xmin, ymin], [xmax, ymax]]` Describes a rectangle within which the fine mesh will be applied and from which the contact surface will be defined. If not given, the entire rail will be finely meshed, and the entire surface will be set as contact surface.
- "fine_mesh" (mandatory): The element size for the fine mesh
- "coarse_mesh" (mandatory): The element size for the coarse mesh.
- "sym_dir" (optional): The direction of the symmetry plane. If not given, no symmetry is assumed. If given, should be `[1,0,0]` or `[-1,0,0]`.

**Material specification**

The material specification should contain the following settings:

- "material_model": Name of the material model to use
- "mpar": Material parameters, given with new keywords.

The supported material models are described below. Note, however, that after the rail `.cae` file has been created, you can freely edit the material model specification.

### elastic

```json
"material_model": "elastic",
"mpar": {"E": <Elastic modulus>, "nu": <Poissons ratio>}
```

### chaboche

```json
"material_model": "chaboche",
"mpar": {"E": <Elastic modulus>, "nu": <Poissons ratio>,
  "Y0": <Initial yield>, "Qinf": <Isotropic saturation>,
  "biso": <Isotropic hardening rate>,
  "Cmod": [<Kinematic modulus 1>, ..., <Kinematic modulus N>],
  "gamma": [<Kinematic saturation speed 1>, ..., <Kinematic saturation speed N>]
}
```
Modifying the basic rail

To script all details of how the rail should be meshed, and if there should be inclusions, cracks, etc., is rather cumbersome and not time efficient. Therefore, it is chosen to allow the user to edit the rail part as an intermediate step. In general, creating the basic rail above is not necessary, but highly recommended as it ensures that correct names are given to sets and surfaces. When modifying the rail part, it is therefore important not to change set names etc. With large geometric modifications, it might also be necessary to redefine these sets to capture the correct parts. A summary of the requirements for the rail part that is used later when generating the rollover is given here.

- The model should be named “RAIL”
- The part should be named “RAIL”
- Sets
  - “BOTTOM_NODES” should contain all nodes at the bottom of the rail
  - “SIDE1_SET” should contain all nodes on the face at z=0
  - “SIDE2_SET” should contain all nodes on the face at z=L where L is the length of the rail.
  - “RAIL_CONTACT_SET” should contain the face where potential contact with the wheel can occur.
  - If present, “SYMMETRY” should contain all nodes on the symmetry face at x=0
- Surfaces
  - “RAIL_CONTACT_SURFACE” should be the surface where potential contact with the wheel can occur.
- Mesh
  - The rail must be meshed, and no constraints should be added (i.e. one cannot use incompatible meshes because this introduces constraints between the nodes).
  - The mesh in “SIDE1_SET” and “SIDE2_SET” must match. I.e. the mesh in “SIDE2_SET” should be a translation from the mesh in “SIDE1_SET”.
- Sections, including material definitions, must be defined on cells of the part.

When working with TET elements, the script `make_rail_mesh_symmetric.py` can be used to ensure a periodic mesh. Otherwise, if HEX meshes are used as a mapped mesh, this will also give the same mesh on both sides.

### 3.2.2 Create the wheel

A wheel super element is created by calling the abaqus script `create_wheel_3d.py` from a folder containing a file `rail_settings.json`. This file should contain the following settings:

- "wheel_name" (mandatory): Name of the folder where the wheel data are placed
- "wheel_angles" (mandatory): [min_ang, max_ang], the angular interval containing the retained wheel nodes. Wrt. the negative y-axis, positive rotation around the x-axis. In radians.
- "wheel_profile" (mandatory): Path to an Abaqus sketch, saved as a .sat file, describing the profile of the wheel in the xy-plane. See *Creating a profile sketch* for further details.
3.2.3 Compile user subroutines

The python script `create_usub.py` (in the `scripts_py` folder) is used to compile the user subroutines. To compile the default subroutine, run this script without any arguments.

If you have additional user subroutines that you wish to use, give the path to the fortran source file (`<your_subroutines_file>`) for these subroutines. You can use include statements as long as all source files reside in the same folder (or subfolders) as the main file. In general, the subroutines should compile with `abaqus make library=<your_subroutines_file>` from their specific folders.

The result will be (1) a folder `tmp_src_dir` and (2) a file `usubs_combined-std.o` (Windows/Linux) The `tmp_src_dir` will contain all sources and a log file describing the compilation process (in case you have any issues). If it works successfully, you can delete this folder. The `usubs_combined-std` file should be copied (and probably renamed to a more descriptive name). It can be convenient to put it in the `data/usubs` directory in the repository (see The data folder and how to specify paths).

3.2.4 Create the rollover simulation

A rollover simulation is created by running the Abaqus script `create_rollover_3d.py`. It reads in the file `rollover_settings.json` which should contain the following settings:

- "rail"
  - model_file: Path to the rail .cae file to use
  - shadow_extents: [ext_at_z=0, ext_at_z=L] How far out to create shadow regions in each end of the rail.
  - use_rail_rp: Boolean if rail reference point should be used or not.

- "wheel"
  - "folder": Path to the folder describing the wheel super element
  - "translation": How to translate the wheel (this depends on both the rail and wheel geometry. Typically, the wheel origin is in the wheel center and the rail origin is at the bottom of the rail.
  - "stiffness": The stiffness (elastic modulus) of the wheel. Its Poissons ratio is fixed at 0.3 from the wheel generation.
  - "symmetric" (optional): Should symmetry in the yz-plane be applied, defaults to false.

- "loading"
  - "initial_depression": Amount of displacement controlled depression before changing to force control on the normal load.
- "inbetween_step_time": Step time to use for the dummy steps (the initial depression, first loading, moving back, reapply load, and release nodes)
- "inbetween_max_incr": Max allowed increments during the dummy steps. To do it in a single increment is always attempted, except for the initial depression and first loading where \( \min(5, \text{inbetween_max_incr}) \) steps are used.
- "rolling_length": The rolling length (must be equal to rail length!).
- "rolling_radius": The effective rolling radius (used to convert slip to wheel rotation).
- "max_incr": Max number of increments to use during rolling.
- "min_incr": Minimum (and initial) number of increments during rolling.
- "num_cycles": Number of rollover cycles to calculate (see also Adding rolling cycles).
- "cycles": \([1, c_{\text{spec}_2}, \ldots, c_{\text{spec}_N}]\), for which cycles that loading parameters are changed. See also Specifying load parameters.
- "vertical_load": \([F_1, F_2, \ldots, F_N]\) Amount of force pushing the wheel onto the rail.
- "speed": \([v_1, v_2, \ldots, v_N]\) The speed at which the wheel is rolling over the rail.
- "slip": \([s_1, s_2, \ldots, s_N]\) The amount of slip as the wheel rolls over the rail.
- "rail_ext": \([e_1, e_2, \ldots, e_N]\) The rail extension at the end of the cycle

**Specifying load parameters**

All load parameters, "cycles", "vertical_load", "speed", "slip", "rail_ext" are specified as lists with equal length. The "cycles" list describe at which cycles the load parameters in the other categories shall be applied. If no specific setting exists for a given cycle, the values from the previous cycle are used. Hence, the minimum requirement is to specify for the first cycle, and then this will be used for all subsequent cycles.

The "slip" = \( s \) is defined such that
\[
\dot{\phi} = (1 + s) \frac{v}{R}
\]
where \( \dot{\phi} \) is the wheel rotation speed, \( v \) is the linear wheel velocity ("speed") and \( R \) is the wheel radius ("rolling_radius").

**Field output description**

The key under "field_output" gives the name of the specific field output request created, e.g. "<field_output_1>". And for each of these keys the following keys should be specified:

- "set": The set in the rail part for which the output should be saved. Note that there are two special names: * "FULL_MODEL": All parts of the model (wheel and rail) * "WHEEL_RP": The wheel reference point.
- "var": \(["VAR_1", "VAR_2", \ldots, "VAR_N"]\). The variables to be saved. Supported variables can be found when creating field outputs in Abaqus CAE. But typical examples are “U” (displacements and rotations), “S” (stresses)
- "freq": How many increments between each time the variables should be saved in the active steps of the field output request.
- "cycles": How many cycles between each time the variables should be saved (i.e. between the active steps of the field output request). If e.g. 25 is specified, output will occur on cycle 1, 26, 51, etc.
3.3 General instructions

This section describes information common to multiple methods, or steps in the methods.

3.3.1 Adding rolling cycles

When adding many 100 steps, Abaqus CAE is rather slow. Therefore, a script is provided to extend a simulation by adding cycles with the same content repeated. Typically, if e.g. 25 is specified as the "cycle" above, then it suffices to generate 26 cycles, and repeat these. The first cycle is not repeated, hence “doubling” the number of cycles will then give 51 cycles in total.

To add cycles, call the python script `append_extra_cycles.py` with the multiplication factor as the first argument and the input file as the second argument. The input file defaults to “rollover.inp”. If called with multiplication factor 4 in the above example, 101 cycles would be created.

3.3.2 Running simulation

To run the simulation the following (generated) files are required to be in the simulation directory:

- `rollover.inp` (can have different name): The Abaqus input file
- `load_param.txt` (must have this name): Automatically generated file in the same directory as `rollover.inp` when creating `rollover`. Describes the loading parameters
- `uel_stiffness.txt` (must have this name): File specifying the wheel stiffness matrix. Automatically generated when creating the wheel, automatically copied to the same directory as `rollover.inp` when creating `rollover`
- `rp_coord.txt` (must have this name): File specifying the location of the reference points. Automatically generated in the same folder as `rollover.inp` when creating `rollover`.

In addition, the user subroutine object file must be available, but it does not need to reside in the simulation directory, but can be in a separate directory and its path specified as `<path_to_usub>`.

Run the simulation by

```abaqus job=rollover user=<path_to_usub>```

3.3.3 The json format

The json format is used for the input data. Mostly, the files should be written with a similar formatting as for a Python dictionary. However, there are a few important differences:

- Booleans are written `true` and `false`, as opposed to `True` and `False`.
- All strings (keywords and variables) must be enclosed in double quotes (single quotes are not accepted).
- Exponential formats must be written `A.BeC` (as opposed to `A.eC`) where `A`, `B`, and `C` are integers. E.g. `1.0e-3` is ok, but not `1.e-3`.
- Python’s `None` is written as `null`.
- Comma is not allowed after the last item in a dictionary

To ensure the correct data format, one can write the following code in Python to generate the json file:
import json
filename = 'example.json'  # Give the filename that you want to save to

# Define the parameters you want to save as a Python dictionary
param = {'key1': [1,2,3],  # Example of list data
         'key2': 'this is a string example data' # Example of string data
}

with open(filename, 'w') as fid:
    # Using indent=1 for nicer output, but not required
    json.dump(param, fid, indent=1)

3.3.4 The data folder and how to specify paths

In the repository, there is a folder named “data”. This contains some examples (which are version controlled). However, additional contents added to the subfolders are ignored by the version control and are suitable for adding data that can be reused later. Examples include profile sketches, generated wheels and rails, and compiled user subroutines.

To simplify the use of contents from this folder, path inputs in the *_settings.json files can be relative the data folder. To do this, the path should start by "://", e.g. ":/rails/rail_example.cae". Otherwise, and absolute or relative (to the Abaqus working directory) path can be specified.

3.3.5 Creating a profile sketch

To create a profile sketch in Abaqus CAE, perform the following steps:

1. Open Abaqus CAE
2. Double-click “Sketches” in the model tree
3. Give your sketch a name (this will have no effect later) and press “Continue”
4. Draw a profile and exit the sketch.
5. Go “File”-“Export”-“Sketch…” and choose a location to save the sketch.
6. In the new dialog box, select the sketch you want to export and press “OK”
7. Choose the ACIS version. Just make sure that it can be read by your system, press ok and you are done.

Note: The sketch will only contain the geometry, so if you later want to edit a dimension later, you need to save the .cae file containing the sketch. Then you can edit the sketch in this file later and export it again.

A default subroutine is compiled when running setup.py. For further information on how to compile custom subroutines, please see Compiling user subroutines.
This section describes how the plugins are coded. For information about how to use the plugins, please see “How to run using cae”.

### 4.1 Plugin registration commands

The plugin registration commands are located in the top-level plugin folder, in the file rollover_plugin.py. This file must be available to Abaqus. One option is to specify `plugin_central_dir` in the `abaqus_v6.env` file. Another is to move the file to `%HOME%abaqus_plugins` on Windows and `~/abaqus_plugins` on Linux. Neither option is great. `plugin_central_dir` is limited to one entry, hence multiple locations are not possible. The second option require code to be copied outside the repository, hence loosing the version control.

### 4.2 Plugin commands

The commands called when plugin buttons are pressed are given in the file `rollover/plugins/commands.py`. This file is similar to the scripts in the `scripts_abq` folder.

### 4.3 Plugin form design

For setting up rail, wheel, and rollover simulation, several user inputs are required. Forms are used for this purpose, and these are coded using Abaqus’ GUI codes. Parts that are re-used between different forms are put in `rollover_gui_utils.py`, and the specific forms are coded in `*_form.py`. All located in `rollover/plugins`. 
5.1 Create rail

5.2 Create wheel

5.3 Create rollover

5.4 Make rail mesh periodic

5.5 Reload modules

This module is used to reload all loaded modules from rollover to in case updates have been made. Having this as a separate modules removes unnecessary clutter from the real code.
6.1 Initial setup

setup.py sets up local adaptations by creating the following files that are ignored by git:

- rollover/local_paths.py: This module contains the following variables:
  - rollover_repo_path: Path to the git repository
  - data_path: Path to the data folder (which will contain examples and user data)
- abaqus_v6.env: This file should be added either to the working directory of the simulation, or to the user’s home directory (%HOME% on Windows and ~ on Linux)
- data/usub/usub_rollover.obj or data/usub/usub/usub_rollover.o: Basic user subroutine required to run rollover simulation.

6.2 Compiling user subroutines

The script create_usub.py is used to compile user subroutines by combining multiple input sources.

The usub/usub_3d.for and the contents of that folder are automatically added. On the script input, specify the path to additional fortran source code containing the Abaqus user subroutine. All content of its folder will be added to the temporary directory and the contents of the particular file is added to the combined user subroutine file. Note the following restrictions:

- Only the files containing abaqus subroutines should be given, and maximum one per folder. If you have multiple subroutines, combine these in one file. The remaining files, that do not have abaqus subroutines can be included via include statements. Hence, the req for the compilation to work is that the given subroutine file would compile on its own using abaqus make library=<subroutine_file>
- No files can have the same path relative the copied folder because the contents of the copied folders are put in the same temporary folder.
- No module names may overlap.

Example

You have a user material subroutine called umat.for, that uses a module umat_mod in umat_mod.f90. These routines are located in C:/umats/my_special_material. umat.for should then have the statement include 'umat_mod.f90' before subroutine umat(...). To compile this subroutine together with the required subroutines for rollover, call the present script from some folder on your computer as:

python <path_to_create_usub.py> C:/umats/my_special_material/umat.for
7.1 Rail (3d)

7.1.1 rollover.three_d.rail.basic

7.1.2 rollover.three_d.rail.mesher

7.1.3 rollover.three_d.rail.include

7.1.4 rollover.three_d.rail.shadow_regions

7.1.5 rollover.three_d.rail.constraints

7.2 Wheel (3d)

7.2.1 rollover.three_d.wheel.substructure

7.2.2 rollover.three_d.wheel.three_d_mesh

7.2.3 rollover.three_d.wheel.super_element

7.3 Utilities (3d)

7.3.1 rollover.three_d.utils.contact

Module to setup the rail-wheel contact

rollover.three_d.utils.contact.setup (the_model, contact_stiffness=1000000.0, friction_coefficient=None, elastic_slip_fraction=0.005)

Add a contact property and a surface-to-surface contact in the_model according to the given settings.

Parameters

- **the_model** (Model object (Abaqus)) – The model to which the contact settings should be applied

- **contact_stiffness** (float) – The stiffness used in the normal penalty formulation.

- **friction_coefficient** (float) – The friction coefficient for the tangential behavior. If none, no tangential behavior will be defined and the contact will be frictionless.
• **elastic_slip_fraction** *(float)* – The allowed elastic tangential slip. This will adjust the penalty stiffness for the tangential contact.

**Returns** None

**Return type** None

### 7.3.2 rollover.three_d.utils.fil_output

This module is used to control the output to the Abaqus result (*fil*) file

**Note:** Uses direct editing of input and should be called after all cae options have been set.

```python
rollover.three_d.utils.fil_output.add(the_model, num_cycles)
```

Add .fil output to input file for each rolling step. For the first, add node coordinates and displacements. For the remaining, add only displacements. If rail reference point is used, add node coord and displacements If rail substructure is used, the instance set names change...

**Parameters**

- **the_model** *(Model object (Abaqus))* – The model to which the output is to be added
- **num_cycles** *(int)* – Number of rollover cycles to simulate

```python
rollover.three_d.utils.fil_output.add_to_step(kwb, varstr, step_name, rail_rp=None, use_substr=False)
```

Add output specified to given step.

**Parameters**

- **kwb** *(KeywordBlock object (Abaqus))* – The keyword block that can be used to edit input file directly.
- **varstr** *(str)* – The string specifying which variables to add to output
- **step_name** *(str)* – The name of the step to which output should be added/modified
- **rail_rp** *(str)* – The name of the rail reference point set
- **use_substr** *(bool)* – Is a rail substructure used?

```python
rollover.three_d.utils.fil_output.get_node_file_output_str(nset, varstr, freqency=9999999)
```

Get the string to add to the input file.

**Parameters**

- **nset** *(str)* – Name of node set from which output should be output
- **varstr** *(str)* – comma separated list of variables to output
- **frequency** *(int)* – How often to write output (increments)

**Returns** The string to add

**Return type** str
7.3.3 rollover.three_d.utils.loading

7.3.4 rollover.three_d.utils.mesh_tools

7.3.5 rollover.three_d.utils.odb_output

This module is used to control the output to the Abaqus result (.fil) file

Note: Uses direct editing of input and should be called after all cae options have been set.

rollover.three_d.utils.odb_output.add(the_model, field_output_requests, num_cycles)

Add the user specified field output requests. Default outputs are deleted.

Parameters

- **the_model** (Model object (Abaqus)) – The model to which the output requests will be added
- **field_output_requests** (dict) – A dictionary with field output request specifications. Each field should be a dictionary containing the following fields:
  - **set**: Which set the output applies to. Refers to sets in the rail instance, except special sets:
    - `FULL_MODEL`: The entire model
    - `WHEEL_RP`: Wheel ctrl point
  - **var**: List of variables to save, e.g. (`U`, `S`)
  - **freq**: How often to output during step. I.e. every incr=1. Set to -1 for only last increment.
  - **cycle**: How often to output cycles, i.e. 1 implies every cycle, 10 implies every 10th cycle, etc.

Returns None

Return type None

7.3.6 rollover.three_d.utils.sketch_tools

Tools to work with sketches

rollover.three_d.utils.sketch_tools.import_sketch(the_model, sketch_profile, name='profile')

Import the sketch sketch_profile and add it to the_model.

Parameters

- **the_model** (Model (Abaqus object)) – The model to which the sketch will be added
- **sketch_profile** (str) – Path to an Abaqus sketch profile saved as .sat file (acis)
- **name** (str) – Name of the created sketch, defaults to ‘profile’

Returns The added sketch

Return type ConstrainedSketch (Abaqus object)
7.3.7 rollover.three_d.utils.symmetric_mesh_module

7.4 General utilities

7.4.1 Setup materials

This module contains functions that sets up material models.

rollover.utils.setup_material_mod.add_material(the_model, material_spec, name)
Add a material to the_model according to material_spec with name=name.

Parameters

- **the_model** (Model object (Abaqus)) – The model to which the sketch will be added
- **material_spec** (dict) – Dictionary containing the fields 'material_model' and 'mpar':
  - 'material_model': which material model to use, currently 'elastic', ‘chaboche’, and ‘user’ are supported.
  - 'mpar': Material parameters, please see function corresponding to 'material_model' below for detailed requirements
- **name** (str (max len = 80)) – The name of the material

Returns None
Return type None

rollover.utils.setup_material_mod.setup_elastic(the_material, mpar)
Setup elastic material behavior

Parameters

- **the_material** (Material object (Abaqus)) – The material to which elastic behavior will be added
- **mpar** (dict) – Dictionary containing the fields
  - 'E': Young’s modulus
  - 'nu': Poissons ratio

Returns None
Return type None

rollover.utils.setup_material_mod.setup_chaboche(the_material, mpar)
Setup plastic material behavior with the chaboche model

Parameters

- **the_material** (Material object (Abaqus)) – The material to which elastic behavior will be added
- **mpar** (dict) – Dictionary containing the fields
  - 'E': Young’s modulus
  - 'nu': Poissons ratio
  - 'Y0': Initial yield limit
- 'Qinf': Saturated isotropic yield limit increase
- 'biso': Speed of saturation for isotropic hardening
- 'Cmod': List of kinematic hardening moduli
- 'gamma': List of kinematic saturation parameters

Returns  None
Return type  None

rollover.utils.setup_material_mod.setup_user(the_material, mpar)
Setup user material behavior

Parameters

- **the_material** (Material object (Abaqus)) – The material to which elastic behavior will be added
- **mpar** (dict) – Dictionary containing the fields
  - 'user_mpar_array': List of user material parameters
  - 'nstatv': Number of state variables for user material model

Returns  None
Return type  None

7.4.2 Direct input file editing

This module enable direct editing of input keywords in the input file. Options not available in CAE can therefore be added via the scripting interface.

rollover.utils.inp_file_edit.add_at_end_of_cat(keyword_block, string_to_add, category, name)
Add string_to_add just before the end of the category of type category with name name.

Parameters

- **keyword_block** (KeywordBlock object (Abaqus)) – The Abaqus keyword-Block that contains the keyword to be written to the input file
- **string_to_add** (str) – The string to add to the input file
- **category** (str) – The category to search for (E.g. Part, Step)
- **name** (str) – The name of the category to find (E.g. Part1-1, Step-1)

Returns  None
Return type  None

rollover.utils.inp_file_edit.add_after(keyword_block, string_to_add, find_strings=None)
Add string_to_add after the first line in to keyword_block that contains all strings in find_strings.

Parameters

- **keyword_block** (KeywordBlock object (Abaqus)) – The Abaqus keyword-Block that contains the keyword to be written to the input file
- **string_to_add** (str) – The string to add to the input file
• **find_strings** (*list [ str]*) – List of strings that the line prior after which **string_to_add** should be added must contain. If **find_strings** = None, add in beginning of the input file

Returns None

Return type None

rollover.utils.inp_file_edit.add_before(**keyword_block**, **string_to_add**, **find_strings**=None)

Add **string_to_add** before the first line in to **keyword_block** that contains all strings in **find_strings**.

Parameters

• **keyword_block** (*KeywordBlock object (Abaqus)*) – The Abaqus keyword-Block that contains the keyword to be written to the input file

• **string_to_add** (*str*) – The string to add to the input file

• **find_strings** (*list [ str]*) – List of strings that the line prior after which **string_to_add** should be added must contain. If **find_strings** = None, add in beginning of the input file

Returns None

Return type None

rollover.utils.inp_file_edit.find_strings_in_iterable(**iterable**, **find_strings**, **min_ind**=0)

Find the lowest index >= **min_ind** of a string in **iterable** that contains all strings in **find_strings**

Parameters

• **iterable** (*An iterable of strings*) – An iterable object containing strings. Must support iteration (i.e. for item in **iterable**) and be subscriptable (i.e. **iterable[3:]**).

• **find_strings** – List of strings that the the item in **iterable** must contain to be found.

• **min_ind** – The index from which the search will start

Returns None

Return type None

### 7.4.3 Naming module

Define names to be used throughout the code. All names that are referenced within multiple functions should be defined in this module. Recommended to import as “import naming_mod as names” Hence, the variables will not contain name, and will be written as e.g. names.step0

rollover.utils.naming_mod.cycle_str(**cycle_nr**)

Get string format for cycle number, use to get consistency with how many zeros are padded.

:.param **cycle_nr**: The cycle number

:type **cycle_nr**: int

Returns String with the cycle number

Return type str

rollover.utils.naming_mod.get_step_rolling(**cycle_nr**)

Get the step name for rolling step in cycle **cycle_nr**

:param **cycle_nr**: The cycle number

:type **cycle_nr**: int

Returns The step name
Return type  str
rollover.utils.naming_mod.get_step_return(cycle_nr=2)
    Get the step name for return step in cycle cycle_nr :param cycle_nr: The cycle number :type cycle_nr: int
    Returns  The step name
    Return type  str
rollover.utils.naming_mod.get_step_reapply(cycle_nr=2)
    Get the step name for reapply step in cycle cycle_nr :param cycle_nr: The cycle number :type cycle_nr: int
    Returns  The step name
    Return type  str
rollover.utils.naming_mod.get_step_release(cycle_nr=2)
    Get the step name for release step in cycle cycle_nr :param cycle_nr: The cycle number :type cycle_nr: int
    Returns  The step name
    Return type  str

7.4.4 Abaqus python general tools

rollover.utils.abaqus_python_tools.setup_log_file(log_file='abaqus_python.log')
    Create a new log file, use at beginning to avoid appending to old file. Not required, but makes log output more readable.
    Parameters  log_file (str) – path to log file
rollover.utils.abaqus_python_tools.log(message, log_file='abaqus_python.log')
    Write log message to STDOUT and logfile. If STDOUT redirected by Abaqus, also write to shell (otherwise it can be hidden)
    Parameters
        • message (str) – The message to write to log file/stdout
        • log_file (str) – Path to log file to append to. Created if non-existent.
    Returns  None
rollover.utils.abaqus_python_tools.create_model(model_name)
    Create a model, delete model if already existing in active mdb.
    Parameters  model_name (str) – Name of model to create/overwrite
    Returns  The created model
    Return type  Model object (Abaqus)
7.4.5 General python tools

General utility functions, that are kept here for convenience. If more functions that can be grouped are added, these may be collected in a new file instead.

rollover.utils.general.get_arguments (function, num_first=0)
Given a function, return all its arguments and the mandatory arguments. This can be used to check **kwargs type of input. num_first skips the first num_first arguments (that are mandatory)

Parameters

• function (class 'function') – The function whose arguments should be obtained

• num_first (int) – How many arguments to skip in the beginning

returns: Lists of all arguments and mandatory arguments rtype: list[ list[ str ] ]

rollover.utils.general.extract_function_args (function, arg_dict, num_first=0)
Given a function and a dictionary containing possible arguments, return a new dictionary containing only the arguments accepted by the function (i.e. remove any keywords that are not arguments to the function). Doesn’t give an error if a mandatory argument is missing.

Parameters

• function (class 'function') – The function whose arguments should be obtained

• arg_dict (dict) – Dictionary containing possible function arguments

• num_first (int) – How many arguments to skip in the beginning

Returns Dictionary with acceptable function arguments

Return type dict

7.4.6 Python 2 enhanced support for json

Module for loading and saving using json. Simplifies syntax and removes unicode strings (converts to regular strings)

rollover.utils.json_io.save (filename, contents)
Save contents to filename with the json file format

Parameters

• filename (str) – The name of the file to be saved

• contents (dict) – Dictionary to be saved to json

Returns None

Return type None

rollover.utils.json_io.read (filename)
Load contents from filename with the json file format

For Python 2, unicode strings within the loaded dictionary are converted to regular strings. (In Python 3 unicode is str)

Parameters filename (str) – The name of the file to be loaded

Returns Dictionary to be saved to json

Rtype contents dict
rollover.utils.json_io.u_to_str_in_dict(dict_to_convert)

Convert unicode entries a dictionary

Unicode strings within the loaded dictionary are converted to regular strings. This is only required for Python 2.

**Parameters**

- **filename** (*str*) – The name of the file to be loaded

**Returns**

Dictionary to be saved to json

**Rtype**

```
contents dict
```  

### 7.4.7 Reloading modules

This module is used to reload all loaded modules from rollover to in case updates have been made. Having this as a separate modules removes unnecessary clotter from the real code.

rollover.utils.reload_modules.execute()
CHAPTER

EIGHT

INDICES AND TABLES

• genindex
• modindex
• search
PYTHON MODULE INDEX

r
reload_modules, 25
rollover.three_d.utils.contact, 29
rollover.three_d.utils.fil_output, 30
rollover.three_d.utils.odb_output, 31
rollover.three_d.utils.sketch_tools, 31
rollover.utils.abaqus_python_tools, 35
rollover.utils.general, 36
rollover.utils.inp_file_edit, 33
rollover.utils.json_io, 36
rollover.utils.naming_mod, 34
rollover.utils.reload_modules, 37
rollover.utils.setup_material_mod, 32

S
scripts_py.create_usub, 27
scripts_py.setup, 27
INDEX

A
add () (in module rollover.three_d.utils.fil_output), 30
add () (in module rollover.three_d.utils.odb_output), 31
add_after () (in module rollover.utils.inp_file_edit), 33
add_at_end_of_cat () (in module rollover.utils.inp_file_edit), 33
add_before () (in module rollover.utils.inp_file_edit), 34
add_material () (in module rollover.utils.setup_material_mod), 32
add_to_step () (in module rollover.three_d.utils.fil_output), 30

C
create_model () (in module rollover.utils.abaqus_python_tools), 35
cycle_str () (in module rollover.utils.naming_mod), 34

E
execute () (in module rollover.utils.reload_modules), 37
extract_function_args () (in module rollover.utils.general), 36

F
find_strings_in_iterable () (in module rollover.utils.inp_file_edit), 34

G
get_arguments () (in module rollover.utils.general), 36
get_node_file_output_str () (in module rollover.three_d.utils.fil_output), 30
get_step_reapply () (in module rollover.utils.naming_mod), 35
get_step_release () (in module rollover.utils.naming_mod), 35
get_step_return () (in module rollover.utils.naming_mod), 35
get_step_rolling () (in module rollover.utils.naming_mod), 34

I
import_sketch () (in module rollover.three_d.utils.sketch_tools), 31

L
log () (in module rollover.utils.abaqus_python_tools), 35

M
module reload_modules, 25
rollover.three_d.utils.contact, 29
rollover.three_d.utils.fil_output, 30
rollover.three_d.utils.odb_output, 31
rollover.three_d.utils.sketch_tools, 31
rollover.utils.abaqus_python_tools, 35
rollover.utils.general, 36
rollover.utils.inp_file_edit, 33
rollover.utils.json_io, 36
rollover.utils.naming_mod, 34
rollover.utils.reload_modules, 37
rollover.utils.setup_material_mod, 32
scripts_py.create_usub, 27
scripts_py.setup, 27

R
read () (in module rollover.utils.json_io), 36
reload_modules module, 25
rollover.three_d.utils.contact module, 29
rollover.three_d.utils.fil_output module, 30
rollover.three_d.utils.odb_output module, 31
module, 31
rollover.three_d.utils.sketch_tools
    module, 31
rollover.utils.abaqus_python_tools
    module, 35
rollover.utils.general
    module, 36
rollover.utils.inp_file_edit
    module, 33
rollover.utils.json_io
    module, 36
rollover.utils.naming_mod
    module, 34
rollover.utils.reload_modules
    module, 37
rollover.utils.setup_material_mod
    module, 32

S
save() (in module rollover.utils.json_io), 36
scripts_py.create_usub
    module, 27
scripts_py.setup
    module, 27
setup() (in module rollover.three_d.utils.contact), 29
setup_chaboche() (in module
rollover.utils.setup_material_mod), 32
setup_elastic() (in module
rollover.utils.setup_material_mod), 32
setup_log_file() (in module
rollover.utils.abaqus_python_tools), 35
setup_user() (in module
rollover.utils.setup_material_mod), 33

U
u_to_str_in_dict() (in module
rollover.utils.json_io), 36