Getting started

The steps below explain how to get your first simulation running with Slide. Some steps are different for Windows or Linux users.

Goal of Slide

The main goal of Slide (Simulator for Lithium-Ion Degradation) is to allow fast simulation of degradation of li-ion batteries. Simulating 5000 full 1C cycles with a time resolution of 2 seconds (1C charge, 1C discharge) for one cell takes about 40 seconds. Including a CV phase on charge to a current limit of 0.05C increases the calculation time to 85 seconds.

The C++ code writes its results in csv files. Matlab functions to read these results have been implemented as well. E.g. to read the results of the check-ups of the pre-defined 'calendar ageing' function, the user has to run 'readCalendarAgeing.m', which will plot the outcomes.

Installing Eclipse (windows users only)

Slide is mostly written in C++ and requires a programming interface. Any c++ programming environment will work, below are the steps to install the environment from Eclipse.

1) Install a Java Development Kit (JDK) or a Java Runtime Environment (JRE): Eclipse needs a JDK or JRE to install and run. If you haven't installed a JDK or JRE yet, it can be downloaded from Oracle (http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html). If you already have a JDK or JRE, you can skip this step. If you are unsure, skip this step, and the Eclipse installer will notify you that you first have to install a JDK or JRE. You might have to restart your computer after installing JDK or JRE before the Eclipse installer will recognize it.
   Please not that not all versions of Eclipse support all versions of JDK or JRE. The Eclipse installer will inform you which Java version to install.
2) Download a version of Eclipse. This project is made using Eclipse Neon, but later versions should work as well. Eclipse can be downloaded from https://www.eclipse.org/downloads/packages/ . You can either download only 'Eclipse IDE

for C/C++ Developers' or the full Eclipse-version (on the right). Make sure to download the 32-bit version if your computer has a 32 bit version of Windows. (note: 64 bit is identified by '64' or 'x64' while the 32 bit version is often identified by '86' or 'x86'. The JDK or JRE version must have the same bit-version as the Eclipse version you are installing. Sometimes the Eclipse installer doesn't recognise the JDK or JRE you have installed. You can try the following steps

   a. Try restarting the computer.
   b. Double check that you have the same bit-version (i.e. 32-bit Java and 32-bit Eclipse or 64-bit Java and 64-bit Eclipse), and your Java version is the one needed by Eclipse (not the latest version of Java available on the Oracle website)
   c. If that doesn't work, uninstall the Java you had installed in step 1 and try re-installing it, potentially try a different version (7/8 and JDK/JRE). Usually Eclipse finds the Java installation after a few re-installations.

3) Execute the installer and if you downloaded the full Eclipse-version, you have to select the 'Eclipse IDE for C/C++ Developers' when asked which Eclipse program to install.

To check afterwards whether a 32 or 64 bit version is installed, open Task Manager and find the Eclipse process (in the tab 'Processes'). If the process is called 'eclipse.exe', it is the 64-bit. If it is called 'eclipse.exe *32' it is a 32-bit version.
can be observed from Task manager (ctrl-shift-esc). In case it is the 32 bit version, behind 'eclipse.exe' is written '(32 bit)'

When you run Eclipse, it will ask you to 'select a directory as a workspace'. This directory will be the folder where all your C++ projects are stored, so choose an empty folder you can easily find back.

Installing a C++ compiler (Windows users only)

Some computers and operating systems come with a c++ compiler. If you don't have one yet, you won't be able to 'build the code' (see below, you will get an error message in Eclipse that the command 'g++' or 'minGW' couldn't be found). In that case, you have to install a C++ compiler. Any C++ compiler should work, but this code has been developed using g++ (from mingw). The steps below are to install the compiler from TDM, which can do both 32 and 64 bit.

- Go to https://sourceforge.net/projects/tdm-gcc/ and download the installer
- Run the installer, and select 'Create: create a new TDM-GCC installation)
- Select the 'MinGW-w64/TDM (32-bit and 64-bit)' installation
- Install it in the standard directory. DO NOT MODIFY THE INSTALLATION DIRECTORY, else Eclipse might not find the compiler.
- Use any 'mirror' you like (this is the server from which the code will be downloaded)
- In 'Choose Components' you don't have to change anything. Normally, the required packages are installed (the essential package is gcc -> c++ but you need the other packages to make the installation work)
- Click install. It might take a couple of minutes

Installing Cmake (Linux users only)

Please download and install cmake: https://cmake.org/download/

- Download the zipped file (.gar.gz) and unzip it
- In the terminal, navigate to the folder where you downloaded it using cd

- Type the following commands in the terminal:
    - ./configure
    - make
    - make install

Downloading and running the code for Windows users

Download the zip file of Slide, and extract it. Move the project to the folder where you had put the Eclipse working space.

You then have to import the project in Eclipse.

- In Eclipse, click 'file', 'import'
- Under 'General', select 'Existing projects into Workspace' (and click on 'next')
- Click on 'browse' (next to 'Select root directory'), and brose to the workspace (or the folder where you had downloaded and extracted this project). Select the folder of this project (i.e. the unzipped folder) and click 'ok'.
- Click on 'Finish', and the project should appear in the 'project explorer window' (the left of the Eclipse screen).
- You can expand folders, the c++ code is in the subfolder 'src'. By double clicking on 'Main.cpp' you will open the code of the main-function, where you can select what you want to simulate by uncommenting the line which calls the function. You uncomment by removing the double backslash ('\\') at the start of a line. You comment something in by adding a double backslash in front of it. E.g. to simulate a few CCCV cycles, uncomment the line starting with '//CCCV(M, pref, deg, cellType, verbose) in the code block 'cycling function calls'. Save the code after you have done so. An elaborate explanation about this is given in the next chapters.
- Then you have to ensure the correct settings are enabled. If you have downloaded the entire project, these should already be correct. If you have only downloaded the source code files and are making your own project, do the following (note that these settings are to get the GNU g++ compiler, if you want to use a different compiler you have to make your own settings). Right click on the project folder (in eclipse) and select 'properties' all the way at the bottom. In the pop-up window on the left, extend 'C/C++ Build' (click on the white triangle) and in the drop-down menu 'Settings'. On the tab 'tool settings' check the following:
    - GCC C++ Compiler: in the box called 'Command' write the following text (without the quotation marks): ' g++ -std=c++11 ' (this calls the g++ compiler for C++ and uses C++ version 11)
    - GCC C++ Compiler -> Miscellaneous: in the box called 'other flags' type the following text (without the quotation marks): ' -c -fmessage-length=0 -std=c++11 '
    - GCC C Compiler: in the box called 'Command' write the following text (without the quotation marks): ' gcc -std=c++11 ' (this calls the gcc compiler for C and uses C version 11)
    - GCC C Compiler -> Miscellaneous: in the box called 'other flags' type the following text (without the quotation marks): ' -c -fmessage-length=0 -std=c++11 '
    - MinGW C++ linker: in the box called 'Command' type following text (without the quotation marks): ' g++ '
    - MinGW C++ Linker -> Miscellaneous: in the box called 'Linker flags' type the following text (without the quotation marks): ' -Wl,--stack,8000000000 '(this option increases the amount of RAM that the simulation can use. If your computer doesn't have much RAM you might have to reduce this number; if the simulation runs out of RAM, it will stop working but no error message will be printed, i.e. you have no clue why it stopped).

- Then you have to build the code by clicking on the hammer-icon in Eclipse (or press 'CTRL' + 'b'). The first time you do this, it might take a while (over a minute). But because builds are incremental, it should go much faster from then onward (a few seconds). If you get an error message that the command 'g++' or 'mingw' could not be found, it means your computer doesn't have a c++ compiler. In that case, follow the steps from the section 'installing a C++ compiler' above.
- Then you can run the code by clicking on the run-button (the green circle with a white triangle inside it). Or you can run it by expanding the subfolder 'Binaries', right-clicking on the .exe-file in there, selecting 'run as' and '1 Local C/C++ Application'
    - As said, a C++ always starts executing the 'main' function in Main.cpp. So if all function calls are still commented, nothings is going to happen. If you have uncommented all function calls, many simulations will be done after each other.
- Then the simulation should start.
- The results are written in csv files, often in subfolders of the project folders. The name of the folder is an identifier indicating which degradation models were used for the simulation.
- Various Matlab functions are written to read the csv files and plot the outcomes. E.g. to read the results from the simulation of the CCCV cycles, execute the script readCCCV.m in Matlab.

Every time you do a simulation, one or more subfolders are created and the results of the simulation are written in those folders. The name of the subfolder consists of three parts (in this order)

- the prefix: in *main* you have to define a string called *prefix*. The name of all subfolders will start with the value of this string, followed by an underscore
- the degradation identification: a series of numbers will indicate which degradation models were used during the simulation (the string is generated by the function *print_DEG_ID* in Degradation.cpp or by the Matlab script printDEGID.m). identifiers of the same mechanism are separated by a hyphen (-), while identifiers of different mechanisms are separated by an underscore ('_'). E.g. 1-0_2-3-1_1-4_1 means we use
    - 1-0 SEI model 1, no porosity changes due to SEI
    - 2-3-1 CS models 2 and 3, decrease the diffusion constant according to model 1
    - 1-4 LAM models 1 and 4
    - 1 Li lating model 1
- the identification string specified in the function you are running (e.g. *CCCV* if you simulate the CCCV cycles).

The code forbids overwriting of previous results. If you do a second simulation, you have two options: either you remove the folders with the old data, or you change the value of 'prefix' (e.g. from '0' to '1') such that the data will be written in folders with a different name.

Downloading and running the code for Linux users

Download the zip file of Slide, and extract it.

In the terminal, navigate to the project folder (the main folder, not src). The first time you build the code, you need to type three commands

```
cmake.          (configure using Cmake and CMakeLists.txt)
make            (build)
./slide         (run)
```

You might get a seg fault because the program can not access enough memory to store all its data. If you don't know how to solve this, you can try typing

```
ulimit -S -s 8000000000
```

However, this command defines the amount of memory ALL programs can use, not just Slide. Therefore, use this with care

Every time you do a simulation, one or more subfolders are created and the results of the simulation are written in those folders. The name of the subfolder consists of three parts (in this order)

- the prefix: in *main* you have to define a string called *prefix*. The name of all subfolders will start with the value of this string, followed by an underscore
- the degradation identification: a series of numbers will indicate which degradation models were used during the simulation (the string is generated by the function *print_DEG_ID* in Degradation.cpp or by the Matlab script printDEGID.m). identifiers of the same mechanism are separated by a hyphen (-), while identifiers of different mechanisms are separated by an underscore ('_'). E.g. 1-0_2-3-1_1-4_1 means we use
    - 1-0 SEI model 1, no porosity changes due to SEI
    - 2-3-1 CS models 2 and 3, decrease the diffusion constant according to model 1
    - 1-4 LAM models 1 and 4
    - 1 Li lating model 1
- the identification string specified in the function you are running (e.g. *CCCV* if you simulate the CCCV cycles).

Various Matlab functions are written to read the csv files and plot the outcomes. E.g. to read the results from the simulation of the CCCV cycles, execute the script readCCCV.m in Matlab.

The code forbids overwriting of previous results. If you do a second simulation, you have two options: either you remove the folders with the old data, or you change the value of 'prefix' (e.g. from '0' to '1') such that the data will be written in folders with a different name.
To run later simulations, you only need the last two commands from the terminal

```
make
./slide
```