Battery Single Particle Model

v1.0

Generated by Doxygen 1.9.1

Chapter 1

Half-SPM-Solver

This is a simulation code for a half-cell single particle model (half-SPM model) written in Fortran90 and interfaced with Python. The code calculates the change in concentration in a single, spherically symmetric particle for a set simulation time from which a voltage profile can also be obtained. The relevant input parameters can be set in the Jupyter notebook 'Input_NetCDF.ipynb' file, which saves the parameters as a NetCDF file. This will be read by the programme, which runs the simulation and returns a visualisation of the concentration and (if chosen) of the voltage profile.

1.1 Getting Started

A Jupyter notebook tutorial is provided to guide the user through the the main features of the simulation package. This is most easily accessed by cloning the GitHub repository to the machine that will be used to run the simulation.

```
git clone https://github.com/HubertJN/PX915_Group_Project.git
```

The tutorial notebook can be accessed by moving to the directory containing the cloned repository and opening a new jupyter notebook environment.

```
cd [directory the repository has been cloned to]
jupyter notebook
```

There is the option to either start a new simulation or to continue a simulation from a checkpoint file. In the first case, all the input parameters have to be set and a new input file in NetCDF format will be created. If the simulation is continued from the checkpoint file, all input parameters are kept the same as for the first run except for the simulation length and the number of simultion steps after which a new checkpoint file is created.

1.2 Setting the Input Parameters

The default parameters are for a positive NCM (Nickel-Cobalt-Manganese) electrode and were taken from Chen2020.

The notebook provides slider bars and an input cell that allow the user to change the input parameters within suggested ranges. There is also the option so set the parameters in a unrestricted input cell – however the user is advised to use this sensibly, since it otherwise might lead to unphysical behaviour or failure of the code.

Important consideration are:

- time step: if the time steps are set too high, the simulation might take too long to complete
- SOC and applied current: at 0% or 100% state of charge (SOC), the applied current should be set positive (charging) or negative (discharging/use), respectively
- If an anode material is being simulated, it will still be the positive electrode with respect to lithium in this half-cell SMP model and the parameters need to be set accordingly

1.3 Running the Simulation

Once the input parameters are set and the 'SPM_input.nc' file has been created, the simulation can be run using the provided Makefile.

1.3.1 Serial Code

To run the code in serial execute the following commands:

```
make
```

To run the simulation, the command is:

```
make exe
```

To visualise the output, the command is:

```
make visual
```

1.3.2 Parallel Code

To run in parallel open "Makefile" and set "num_threads" to the desired number of threads to parellise over. Run same commands as above.

1.4 Documention

For more information see our documentation. If the documentation needs to be re-generated, input the following into the command prompt.

```
make docs
```

1.5 Uncertainty Quantification

There are a variety of options to visualise the uncertainties involved with the simulation. The following commands represent the available uncertainty quantification options.

Perform sensitivity analysis and then display the results.

```
make sensitive
```

Display the results from the sensitivity analysis.

```
make vis_sens
```

Perform uncertainty propagation using random latin hypercube sampling and display the results.

```
make uncertain
```

Display the results from random latin hypercube sampling.

```
make uncer_vis
```

Perform sensitivity analysis and perform random latin hypercube sampling. Then calculate uncertainty from the standard deviations and random latin hypercube sampling. Then display results.

```
make sens_uncer
```

Visualise results of calculated uncertainties from the standard deviations and random latin hypercube sampling.

```
make vis_sens_uncer
```

Calculates the uncertainty from the standard deviations and then, assuming random latin hypercube sampling has been performed, displays both.

```
make sens_uncer_sep
```

1.6 How to Run the Data Fitting

Ensure that the /venv/ directory is downloaded, which contains all the necessary python libraries which the data fitting notebook, 'DataFit_int.ipynb', requires. The /venv/ directory is a virtual environment, and launching Jupyter notebook inside this virtual environment will allow Jupyter to use the required libraries without the user needing to install them. To activate the virtual environment, go to the directory where /venv/ is stored. This directory should also contain the solver used for data fitting, 'datafitPde.f90', and the notebook that runs the datafitting, 'DataFit_int.ipynb'. Once the /venv/ directory and the aforementioned programs are stored in the same location, the virtual environment needs to be launched. To launch the virtual environment, move to the location where it is stored in a terminal and type the following command:

```
source /venv/bin/activate
```

You can tell if the virtual environment is activated if you can see '(venv)' written before your current working directory in the terminal. Jupyter notebook can now be launched in this virtual environment by typing the following in the same terminal:

```
jupyter notebook
```

Once Jupyter is launched, find the notebook 'DataFit_int.ipynb' in the Jupyter file browser and open it. You can then run the cells accordingly to preform the optimisation on the diffusion coefficient. To see the result without running the notebook, open the png file, 'fitted_diff_coeffs.png'.

1.7 How to obtain the benchmarking results

The half-SPM model has been benchmarked against the open source simulation package PyBaMM by obtaining the relative absolute error and the root mean square error (RMSE) for 5 different simulations:

1. Charging at 5 A using the default input parameters
2. Discharging at 5 A starting at 95% state of charge (SOC) using the default for all other input parameters
3. A galvanostatic intermittent titration technique (GITT) experiment, where the half-cell is rested for 30 minutes (i.e. no current applied), followed by discharge at 1 A for 5 minutes and another rest period for 30 minutes
4. Charging with a diffusion coefficient of $4∗10^{-10} \ m^2 s^{-1}$ using the default for all other input parameters
5. Charging with a mean particle radius of $7 \mu m$ using the default for all other input parameters

A convergence study of the RMSE with respect to the time step is also provided.

The results can be obtained by running the following comamnd:

```
make benchmarking
```

Chapter 2

Finite Difference Formulation

This section outline the formulae used within the code in order to carry out the simulation. Whenever a variable or code snippet refers to the formulation section, this is the part of the code being referred to.

2.1 Matrix A

This is the left hand side coefficient matrix of the system of equations used within the simulation, of size $n \times n$ with elements given by

$$A_{i,j} = \begin{cases} -\frac{\Delta t D}{2\Delta r^2} + \frac{\Delta t D}{2r_i \Delta r} & j = i-1 \text{ for } 2 \leqslant i \leqslant n-1 \\ 1 + \frac{\Delta t D}{\Delta r^2} & j = i \\ -\frac{\Delta t D}{2r_i \Delta r} - \frac{\Delta t D}{2\Delta r^2} & j = i+1 \text{ for } 2 \leqslant i \leqslant n-1 \\ -\frac{\Delta t D}{\Delta r^2} & (i,j) = (1,2), (n,n-1) \end{cases}$$

2.2 Matrix B

This is the right hand side coefficient matrix of the system of equations used within the simulation, of size $n \times n$ with elements given by

$$B_{i,j} = \begin{cases} \frac{\Delta t D}{2\Delta r^2} - \frac{\Delta t D}{2r_i \Delta r} & j = i-1 \text{ for } 2 \leqslant i \leqslant n-1 \\ 1 - \frac{\Delta t D}{\Delta r^2} & j = i \\ \frac{\Delta t D}{2r_i \Delta r} + \frac{\Delta t D}{2\Delta r^2} & j = i+1 \text{ for } 2 \leqslant i \leqslant n-1 \\ \frac{\Delta t D}{\Delta r^2} & (i,j) = (1,2), (n,n-1) \end{cases}$$

Chapter 3

Miscellaneous Code

3.1 Python

3.1.1 plots.py

plots.py reads the data from the NetCDF output file generated from the main program and creates three plots: a concentration profile across particle radius over time; a contour plot of concentration across the single particle over time and if a voltage calculation was performed it also displays the voltage over time.

3.1.2 generate_inp_params.py

generate_inp_params.py reads the input file that has been generated by Input_NetCDF.ipynb and attempts to extract standard deviation values and a number of samples to perform latin hyper cube sampling, after succesfully importing standard deviations it produces a normal distribution with the initial input value as the mean. If reading standard deviation fails then the standard deviations are set to zero and the code continues. Then the code generates random numbers between 0 and 1 using random latin hyper sampling for the number of variables with standard deviation greater than 0. It then maps these onto the gaussian distributions. Then it puts all the samples into an array and saves them to a .csv file. These will act as the inputs for the random sampling code. See "make_input_file.py" and "vis_up_data.py".

3.1.3 generate_inp_params_sens.py

generate_inp_params_sens.py reads the input file that has been generated by Input_NetCDF.ipynb. It then purterbs the parameters by 10^{-4}% and saves these to a 10 by 9 array where the first row is the initial input parameters and the others are the initial input parameters with the diaganal perturbed by 10^{-4}%. The it saves the array to a .csv file. These will act as the inputs for the sensitivity analysis code. See "make_input_file.py" and "vis_uq_res.py".

3.1.4 make_input_file.py

make_input_file.py read the .csv file by generate_inp_params.py and generate_inp_params_sens.py and creates a NetCDF input file that can be read by the program.

3.1.5 vis_uq_res.py

vis_uq_res.py reads in the original parameters from the input file and regenerates the 9 x 10 array created in generate_inp_params_sens.py. Then extracts the voltage data that was created by running the main program with the perturbed parameters and calculates first derivative of the voltage with respect to the parameters. Then calculates the scaled first order sensitivities and saves this data to a .csv file. Then it plots an animation of these scaled first order sensitivities over time.

3.1.6 visual_up_data.py

visual_up_data.py extracts the voltage data calculated from the random latin hyper cube input samples and calculates the 97.5^{th} percentile and the 2.5^{th} percentile. It plots this data along side the mean voltage extracted from the original output file.

3.1.7 vis_uncer_sens.py

vis_uncer_sens.py extracts the standard deviations of the voltage at each time step created by generate_inp_↩ params.py and plots the data along with the mean voltage from the original output file.

3.2 Makefile

The Makefile compiles the code and contain various commands to run the code.

3.2.1 make clean

```
make clean
```

This PHONY command clears the directory to the same state as if just installed.

3.2.2 make

```
make
```

This command compiles the code and depending on the variable num_threads compiles with (num_threads $>$ 1) or without (num_threads = 1) OpenMP.

3.2.3 make exe

```
make exe
```

This PHONY command executes the executable.

3.2.4 make visual

```
make visual
```

This PHONY command call plots.py to visualise results.

3.2.5 make docs

```
make docs
```

This PHONY command attempts to delete previously existing doxygen files and regenerates the documentation. This command requires the doxygen packaged to be installed in order to run.

3.2.6 make sensitive

```
make sensitive
```

This PHONY command executes sens_ana.sh False to perform sensitivity analysis.

3.2.7 make vis_sens

```
make vis_sens
```

This PHONY command executes visual_uq_res.py to visualise sensitivity analysis data in a new terminal.

3.2.8 make uncertain

```
make uncertain
```

This PHONY command executes up_code.sh False to perform random latin hyper cube sampling of voltage.

3.2.9 make vis_uncer

```
make vis_uncer
```

This PHONY command executes visual_up_data.py to visualise random latin hyper cube analysis results in a new terminal.

3.2.10 make sens_uncer

```
make sens_uncer
```

This PHONY command executes sens_ana.sh False followed by up_code.sh True to perform sensitivity analysis, from this calculate an approximate uncertainty and then perform random latin hyper cube sampling to quantify uncertainty. Then plots results.

3.2.11 make vis_sens_uncer

```
make vis_sens_uncer
```

This PHONY commands visualises the approximate uncertainty calculated from sensitivity analysis.

3.2.12 make sens_uncer_sep

```
make sens_uncer_sep
```

This PHONY command calculates an approximate uncertainty using standard deviation and displays this alongside already existing data from random latin hyper cube sampling.

3.2.13 make uncer_from_sens

```
make uncer_from_sens
```

This PHONY command calculates and displays approximate uncertainty calculated from sensitivity analysis.

3.2.14 make vis_uncer_from_sens

```
make vis_uncer_from_sens
```

This PHONY command displays approximate uncertainty calculated from sensitivity analysis.

3.2.15 make benchmarking

```
make benchmarking
```

This PHONY command opens a new terminal and executes the python script that visualises the benchmarking from PyBamm.

3.2.16 make virtual

```
make virtual
```

This PHONY command adjust permissions for compile.sh, to allow it to run datafitting. Then installs the python virtual environment using sudo priviliges. Next, it sets up a directory called venv which contains the virtual environment dependencies.

3.2.17 make mods

```
make mods
```

This PHONY command installs the required python modules for running the repository in the venv directory from requirements.txt. Next, it installs Jupyter extensions to allow for collapsing of cells which makes the tutorial more user friendly.

3.3 Bash scripts

Theses scripts are used to run the main program automatically with various different input parameters.

3.3.1 sens_ana.sh

sens_ana.sh is used to create the necessary data for sensitivity analysis.

If the first input is "False": The full sensitivity analysis is performed which is as follows: The script checks that a data storage repository (data_store_sens) exists, if it doesn't it creates it. It then calls generate_inp_params_sens.py to generate a .csv file (data.csv) containing the required input parameters for the analysis. The code then loops, reading a set of input parameters, generating the input file, moving the input file to the main directory, running the code, and moving the output file to storage. It then calls the visualisation script visual_uq_res.py to visualise the results.

If the first input is "True": This is to be used when the sensitivity analysis data already exists and the user wants to calculate an approximation of the uncertainty from the standard deviations. This calls generate_inp_params.py to calculate the std_V_dat.csv data. It then visualises the data by calling vis_uncer_sens.py.

3.3.2 up_code.sh

up_code.sh is used to create the necessary data for uncertainty propagation using randon Latin hypercube sampling.

The script first checks that a data storage repository (data_store_up) and, if not, creates one. The script takes in the number of samples, generates this number of input parameters sampled using random latin hypercube sampling, and saves these to data.csv. It then runs a loop that creates an input file from this data.csv file, transfers this to the main directory, runs the code, and moves the output to a data store. Finally, it calls the visual_up_data.py script to visualise the results.

Chapter 4

Modules Index

4.1 Modules List

Here is a list of all modules with brief descriptions:

Chapter 5

Data Type Index

5.1 Data Types List

Here are the data types with brief descriptions:

Chapter 6

File Index

6.1 File List

Here is a list of all files with brief descriptions:

Chapter 7

Module Documentation

7.1 input_output_netcdf Module Reference

Functions/Subroutines

- subroutine error_check (ierr)

  Error checking subroutine for NetCDF.
- subroutine assign_int (file_id, act, vect, sing, var_name, var_id_in, var_id_out)

  I/O subroutine that reads integers from a NetCDF file or writes integers to a NetCDF file.
- subroutine assign_real (file_id, act, vect, sing, var_name, var_id_in, var_id_out)

  I/O subroutine that reads reals from a NetCDF file or writes reals to a NetCDF file.
- subroutine assign_exp_int (var, file_id, it, var_name, var_id_in)

  I/O subroutine that writes integer arrays to a variable with an infinite dimension.
- subroutine assign_exp_real (var, file_id, it, var_name, var_id_in)

  I/O subroutine that writes real arrays to a variable with an infinite dimension.
- subroutine create_sing_var (var_name, var_typ, var_len, file_id, units, act, var_id_out)

  I/O subroutine that creates a named variable with specific length and data type within a NetCDF file.
- subroutine create_exp_var (var_name, var_typ, var_len, file_id, units, act, var_id_out)

  I/O subroutine that creates an expanding variable with specific dimensions and data type within a NetCDF file.
- subroutine save_int (var_name, file_id, vect, sing, units, act, var_len)

  I/O subroutine that writes an integer vector or single number to existing variable within a NetCDF file.
- subroutine save_real (var_name, file_id, vect, sing, units, act, var_len)

  I/O subroutine that writes a real vector or single number to existing variable within a NetCDF file.
- subroutine fin_in_out ()

  I/O subroutine that closes NetCDF output and checkpoint files that are open.
- subroutine import_input (file_name)

  I/O subroutine that opens a NetCDF file, reads values, writes them to global variables and closes the file.
- subroutine initiate_file (file_name)

  I/O subroutine that creates a NetCDF file, initiates input variables and saves variables that are available.
- subroutine initiate_checkp (file_name)

  I/O subroutine that creates a NetCDF file, initiates checkpoint specific variables and saves available variables.
- subroutine load_checkp (check_file_name, out_file_name, conc, volt_do)

  I/O subroutine that reads from a checkpoint file and saves the concentration vector.
- subroutine update_checkp (conc, step_num)

  I/O subroutine that overwrites the concentration vector and number of time steps in the checkpoint NetCDF file.

Variables

- integer, parameter dp =kind(1.0D0)
- real(kind=real64), parameter farad = 96485.3321233100184_DP

  farad is the Faraday constant, F.
- real(kind=real64), parameter gas_con = 8.31446261815324_DP

  gas_con is the ideal gas constant, R_g.
- real(kind=real64) temp

  temp is the temperature of the simulation, T.
- real(kind=real64) rad

  rad is the radius of the particle, R.
- real(kind=real64) thick

  thick is the thickness of the electrode, L.
- real(kind=real64) rr_coef

  rr_coef is the reaction rate coefficient, K.
- real(kind=real64) dif_coef

  dif_coef is the diffusion coefficient, D.
- real(kind=real64) iapp

  iapp is the applied current density as a function of time, i_{app}.
- real(kind=real64) init_c

  init_c is the initial concentration, c_0.
- real(kind=real64) max_c

  max_c is the maximum concentration of the simulation, c_{max}.
- real(kind=real64) dt

  dt is the time step.
- real(kind=real64) vol_per

  vol_per is the active material volume fraction, ϵ_{actk}.
- real(kind=real64) final_time

  final_time is the time of the final time step.
- integer(kind=int32) sim_steps

  sim_steps is the number of simulation steps.
- integer(kind=int32) out_steps

  out_steps is the number of steps before writing to output file.
- integer(kind=int32) space_steps

  space_steps is the resolution of the radius.
- integer(kind=int32) tot_steps

  tot_steps is the total number of simulation steps.
- integer(kind=int32) output_id

  output_id is the id of the output file.
- integer(kind=int32) check_id

  check_id is the id of the checkpoint file.
- integer(kind=int32) volt_out_id

  volt_out_id is the variable id of the voltage in the output file.
- integer(kind=int32) conc_out_id

  conc_out_id is the variable id of the concentration in the output file.
- integer(kind=int32) conc_check_id

  conc_check_id is the variable id of the concentration in the checkpoint file.
- integer(kind=int32) time_check_id

  time_check_id is the id of the final time of the simulation in the checkpoint file.
- integer(kind=int32) ts_check_id

  ts_check_id is the id of the final time step of the simulation in the checkpoint file.

- logical volt_do

  volt_do is the logical value which determines whether to calculate voltage to the output file.
- logical checkpoint

  checkpoint is the logical value which determines whether to restart simulation from the checkpoint.
- integer(kind=int32) volt_do_int

  volt_do_int is the binary representation of the volt_do variable.
- integer(kind=int32) checkpoint_int

  checkpoint_int is the binary representation of the checkpoint_int.

7.1.1 Function/Subroutine Documentation

7.1.1.1 assign_exp_int()

```
subroutine input_output_netcdf::assign_exp_int (
             integer(kind=int32), dimension(:,:), intent(in) var,
             integer(kind=int32), intent(in) file_id,
             integer(kind=int32), intent(in) it,
             character(len=*), intent(in), optional var_name,
             integer(kind=int32), intent(in), optional var_id_in )
```

I/O subroutine that writes integer arrays to a variable with an infinite dimension.

This subroutine writes a 2D integer array 'var', to the variable named 'var_name' with variable id 'var_id_in' in the NetCDF file with id 'file_id' at position 'it'.

This should be used to write integer arrays 'var', to a variable 'var_name', with an infinite dimension.

Parameters

in	*var*	2D integer array
in	*var_name*	name of variable to write
in	*var_id_in*	id of variable to write
in	*file_id*	id of NetCDF file to write to
in	*it*	position within NetCDF file to write to

Definition at line 321 of file input_output_netcdf.f90.

7.1.1.2 assign_exp_real()

```
subroutine input_output_netcdf::assign_exp_real (
             real(kind=real64), dimension(:,:), intent(in) var,
             integer(kind=int32), intent(in) file_id,
             integer(kind=int32), intent(in) it,
             character(len=*), intent(in), optional var_name,
             integer(kind=int32), intent(in), optional var_id_in )
```

I/O subroutine that writes real arrays to a variable with an infinite dimension.

This subroutine writes a 2D real array 'var', to the variable named 'var_name' with variable id 'var_id_in' in the NetCDF file with id 'file_id' at position 'it'.

This should be used to write real arrays 'var', to a variable 'var_name', with an infinite dimension.

Parameters

in	*var*	2D real array
in	*var_name*	name of variable to write
in	*var_id_in*	id of variable to write
in	*file_id*	id of NetCDF file to write to
in	*it*	position within NetCDF file to write to

Definition at line 360 of file input_output_netcdf.f90.

7.1.1.3 assign_int()

```
subroutine input_output_netcdf::assign_int (
             integer(kind=int32), intent(in) file_id,
             character(len=*), intent(in) act,
             integer(kind=int32), dimension(:), intent(inout), optional vect,
             integer(kind=int32), intent(inout), optional sing,
             character(len=*), intent(in), optional var_name,
             integer(kind=int32), intent(in), optional var_id_in,
             integer(kind=int32), intent(out), optional var_id_out )
```

I/O subroutine that reads integers from a NetCDF file or writes integers to a NetCDF file.

This subroutine reads or writes a single integer or a vector of integers, from or to a variable with a given name or variable id in a NetCDF file with a given NetCDF id.

What the subroutine does is dictated by the input arguments. One of the arguments sing and vect must be inputed into the subroutine as well as one of the arguments var_name and var_id_in.

Optionally you can save the variable id by inputting a variable to store it, var_id_out.

Parameters

in	*act*	subroutine action: 'r' for read and 'w' for write
in,out	*sing*	single integer to read/write, optional argument
in,out	*vect*	vector of integers to read/write, optional argument
in	*var_name*	name of variable to read from or write to, optional argument
in	*var_id_in*	id of variable to read from or write to, optional argument
in	*file_id*	NetCDF file id
out	*var_id_out*	id of variable to store var_id_in, optional argument

Definition at line 171 of file input_output_netcdf.f90.

7.1.1.4 assign_real()

```
subroutine input_output_netcdf::assign_real (
             integer(kind=int32), intent(in) file_id,
             character(len=*), intent(in) act,
             real(kind=real64), dimension(:), intent(inout), optional vect,
             real(kind=real64), intent(inout), optional sing,
             character(len=*), intent(in), optional var_name,
             integer(kind=int32), intent(in), optional var_id_in,
             integer(kind=int32), intent(out), optional var_id_out )
```

I/O subroutine that reads reals from a NetCDF file or writes reals to a NetCDF file.

This subroutine reads or writes a single real or a vector of reals, from or to a variable with a given name or variable id in a NetCDF file with a given NetCDF id.

What the subroutine does is dictated by the input arguments. One of the arguments sing and vect must be inputed into the subroutine as well as one of the arguments var_name and var_id_in.

Optionally you can save the variable id by inputting a variable to store it, var_id_out.

Parameters

in	*act*	subroutine action: 'r' for read and 'w' for write
in,out	*sing*	single real to read/write, optional argument
in,out	*vect*	vector of reals to read/write, optional argument
in	*var_name*	name of variable to read from or write to, optional argument
in	*var_id_in*	id of variable to read from or write to, optional argument
in	*file_id*	NetCDF file id
out	*var_id_out*	id of variable to store var_id_in, optional argument

Definition at line 250 of file input_output_netcdf.f90.

7.1.1.5 create_exp_var()

```
subroutine input_output_netcdf::create_exp_var (
             character(len=*), intent(in) var_name,
             integer(kind=int32), intent(in) var_typ,
             integer(kind=int32), intent(in) var_len,
             integer(kind=int32), intent(in) file_id,
             character(len=*), intent(in), optional units,
             character(len=*), intent(in), optional act,
             integer(kind=int32), intent(out), optional var_id_out )
```

I/O subroutine that creates an expanding variable with specific dimensions and data type within a NetCDF file.

This subroutine creates an expanding variable called 'var_name' with dimensions 'var_len x undefined' and data type 'var_typ' (in this case f90_int or f90_double) in a NetCDF file with id 'file_id'.

You can optionally prescribe units to the variable'units', and if you want to save the variable id you can input a variable to store it 'var_id_out'.

If the file is NOT in definition mode, so already exists, you can use (act='add') to add the variable to an existing netcdf file with id 'file_id'.

Parameters

in	*var_name*	name of variable to be created in NetCDF file
in	*var_len*	length of variable to be created in NetCDF file
in	*var_type*	data type of variable to be created, f90_int or f90_double
in	*file_id*	id of NetCDF file where variable is being created
in	*units*	units of variable, optional argument
out	*var_id_out*	variable to store NetCDF variable id, optional argument
in	*act*	set to 'add' to add variable to existing NetCDF file, optional argument

Definition at line 460 of file input_output_netcdf.f90.

7.1.1.6 create_sing_var()

```
subroutine input_output_netcdf::create_sing_var (
             character(len=*), intent(in) var_name,
             integer(kind=int32), intent(in) var_typ,
             integer(kind=int32), intent(in) var_len,
             integer(kind=int32), intent(in) file_id,
             character(len=*), intent(in), optional units,
             character(len=*), intent(in), optional act,
             integer(kind=int32), intent(out), optional var_id_out )
```

I/O subroutine that creates a named variable with specific length and data type within a NetCDF file.

This subroutine creates a variable called 'var_name' with length 'var_len' and data type 'var_typ' (in this case f90_int or f90_double) in a NetCDF file with id 'file_id'.

You can optionally prescribe units to the variable 'units', and if you want to save the variable id you can input a variable to store it 'var_id_out'.

If the file is NOT in definition mode, so already exists, you can use (act='add') to add the variable to an existing netcdf file with id 'file_id'.

Parameters

in	*var_name*	name of variable to be created in NetCDF file
in	*var_len*	length of variable to be created in NetCDF file
in	*var_type*	data type of variable to be created, f90_int or f90_double
in	*file_id*	id of NetCDF file where variable is being created
in	*units*	units of variable, optional argument
out	*var_id_out*	variable to store NetCDF variable id, optional argument
in	*act*	set to 'add' to add variable to existing NetCDF file, optional argument

Definition at line 405 of file input_output_netcdf.f90.

7.1.1.7 error_check()

```
subroutine input_output_netcdf::error_check (
            integer, intent(in) ierr )
```

Error checking subroutine for NetCDF.

This subroutine takes in an integer error code from NetCDF (ierr), prints out the associated error, and stops the code. If there is no error, subroutine continues.

Parameters

in	*ierr*	NetCDF error code

Definition at line 141 of file input_output_netcdf.f90.

7.1.1.8 fin_in_out()

```
subroutine input_output_netcdf::fin_in_out
```

I/O subroutine that closes NetCDF output and checkpoint files that are open.

Definition at line 578 of file input_output_netcdf.f90.

7.1.1.9 import_input()

```
subroutine input_output_netcdf::import_input (
            character(len=*), intent(in), optional file_name )
```

I/O subroutine that opens a NetCDF file, reads values, writes them to global variables and closes the file.

As a test case, if no file_name is given a series of test values are prescribed instead.

Parameters

in	*file_name*	name of NetCDF file to open

Definition at line 596 of file input_output_netcdf.f90.

7.1.1.10 initiate_checkp()

```
subroutine input_output_netcdf::initiate_checkp (
            character(len=*), intent(in) file_name )
```

I/O subroutine that creates a NetCDF file, initiates checkpoint specific variables and saves available variables.

Parameters

in	*file_name*	name of NetCDF file to create

Definition at line 717 of file input_output_netcdf.f90.

7.1.1.11 initiate_file()

```
subroutine input_output_netcdf::initiate_file (
            character(len=*), intent(in) file_name )
```

I/O subroutine that creates a NetCDF file, initiates input variables and saves variables that are available.

Parameters

in	*file_name*	name of NetCDF file to create

Definition at line 662 of file input_output_netcdf.f90.

7.1.1.12 load_checkp()

```
subroutine input_output_netcdf::load_checkp (
            character(len=*), intent(in) check_file_name,
            character(len=*), intent(in) out_file_name,
            real(kind=real64), dimension(:), intent(inout) conc,
            logical, intent(in) volt_do )
```

I/O subroutine that reads from a checkpoint file and saves the concentration vector.

This subroutine reads from a checkpoint file named 'file_name' and saves the concentration vector to 'conc'. It extracts other variables to keep track of the total simulation steps and total simulation time. It also opens an old netcdf output file and gets variable ids for the variables it will write new data to.

Parameters

in	*check_file_name*	name of NetCDF checkpoint file to read
in	*out_file_name*	name of NetCDF file to write to
in,out	*conc*	concentration vector
in	*volt_do*	logic value to determine whether to write voltage

Definition at line 750 of file input_output_netcdf.f90.

7.1.1.13 save_int()

```
subroutine input_output_netcdf::save_int (
             character(len=*), intent(in) var_name,
             integer(kind=int32), intent(in) file_id,
             integer(kind=int32), dimension(:), intent(inout), optional vect,
             integer(kind=int32), intent(inout), optional sing,
             character(len=*), intent(in), optional units,
             character(len=*), intent(in), optional act,
             integer(kind=int32), intent(in), optional var_len )
```

I/O subroutine that writes an integer vector or single number to existing variable within a NetCDF file.

This subroutine writes an integer vector 'vect' or single number 'sing' to an existing variable named 'var_name' in a NetCDF file with id 'file_id' if (act='new'), this assumes the variable does not already exist and will create an integer variable called 'var_name' with length 'var_len' and units 'units' and write the integer variable ('vect' or 'sing') to this variable.

Parameters

in	*var_name*	name of variable to write to in NetCDF file
in	*file_id*	id of NetCDF file where variable is being written
in,out	*vect*	integer vector to be written, optional argument
in,out	*sing*	single integer to be written, optional argument
in	*units*	units of variable, optional argument
in	*act*	set to 'new' to create variable in NetCDF file, optional argument
in	*var_len*	length of variable to be created in NetCDF file

Definition at line 515 of file input_output_netcdf.f90.

7.1.1.14 save_real()

```
subroutine input_output_netcdf::save_real (
             character(len=*), intent(in) var_name,
             integer(kind=int32), intent(in) file_id,
             real(kind=real64), dimension(:), intent(inout), optional vect,
             real(kind=real64), intent(inout), optional sing,
             character(len=*), intent(in), optional units,
             character(len=*), intent(in), optional act,
             integer(kind=int32), intent(in), optional var_len )
```

I/O subroutine that writes a real vector or single number to existing variable within a NetCDF file.

This subroutine writes a real vector 'vect' or single number 'sing' to an existing variable named 'var_name' in a NetCDF file with id 'file_id' if (act='new'), this assumes the variable does not already exist and will create an integer variable called 'var_name' with length 'var_len' and units 'units' and write the real variable ('vect' or 'sing') to this variable.

Parameters

in	*var_name*	name of variable to write to in NetCDF file
in	*file_id*	id of NetCDF file where variable is being written

Parameters

in,out	*vect*	integer vector to be written, optional argument
in,out	*sing*	single integer to be written, optional argument
in	*units*	units of variable, optional argument
in	*act*	set to 'new' to create variable in NetCDF file, optional argument
in	*var_len*	length of variable to be created in NetCDF file

Definition at line 554 of file input_output_netcdf.f90.

7.1.1.15 update_checkp()

```
subroutine input_output_netcdf::update_checkp (
             real(kind=real64), dimension(:), intent(inout) conc,
             integer(kind=int32), intent(inout) step_num )
```

I/O subroutine that overwrites the concentration vector and number of time steps in the checkpoint NetCDF file.

Parameters

in,out	*conc*	concentration vector
in,out	*step_num*	number of time steps

Definition at line 785 of file input_output_netcdf.f90.

7.1.2 Variable Documentation

7.1.2.1 check_id

```
int32 check_id
```

check_id is the id of the checkpoint file.

Definition at line 128 of file input_output_netcdf.f90.

7.1.2.2 checkpoint

```
logical checkpoint
```

checkpoint is the logical value which determines whether to restart simulation from the checkpoint.

Definition at line 130 of file input_output_netcdf.f90.

7.1.2.3 checkpoint_int

```
int32 checkpoint_int
```

checkpoint_int is the binary representation of the checkpoint_int.

Definition at line 131 of file input_output_netcdf.f90.

7.1.2.4 conc_check_id

```
int32 conc_check_id
```

conc_check_id is the variable id of the concentration in the checkpoint file.

Definition at line 129 of file input_output_netcdf.f90.

7.1.2.5 conc_out_id

```
int32 conc_out_id
```

conc_out_id is the variable id of the concentration in the output file.

Definition at line 129 of file input_output_netcdf.f90.

7.1.2.6 dif_coef

```
real64 dif_coef
```

dif_coef is the diffusion coefficient, D.

It has units of $m^2 s^{-1}$.

Definition at line 125 of file input_output_netcdf.f90.

7.1.2.7 dp

```
integer, parameter dp =kind(1.0D0)
```

Definition at line 122 of file input_output_netcdf.f90.

7.1.2.8 dt

```
real64 dt
```

dt is the time step.

It has units of s.

Definition at line 126 of file input_output_netcdf.f90.

7.1.2.9 farad

```
real64 farad = 96485.3321233100184_DP
```

farad is the Faraday constant, F.

It has units of $Cmol^{-1}$.

Definition at line 123 of file input_output_netcdf.f90.

7.1.2.10 final_time

```
real64 final_time
```

final_time is the time of the final time step.

It has units of s .

Definition at line 126 of file input_output_netcdf.f90.

7.1.2.11 gas_con

```
real64 gas_con = 8.31446261815324_DP
```

gas_con is the ideal gas constant, R_g.

It has units of $JK^{-1}mol^{-1}$.

Definition at line 124 of file input_output_netcdf.f90.

7.1.2.12 iapp

```
real64 iapp
```

iapp is the applied current density as a function of time, i_{app}.

It has units of Am^{-2}.

Definition at line 125 of file input_output_netcdf.f90.

7.1.2.13 init_c

```
real64 init_c
```

init_c is the initial concentration, c_0.

It has units of $molm^{-3}$.

Definition at line 126 of file input_output_netcdf.f90.

7.1.2.14 max_c

```
real64 max_c
```

max_c is the maximum concentration of the simulation, c_{max}.

It has units of $molm^{-3}$.

Definition at line 126 of file input_output_netcdf.f90.

7.1.2.15 out_steps

```
int32 out_steps
```

out_steps is the number of steps before writing to output file.

Definition at line 127 of file input_output_netcdf.f90.

7.1.2.16 output_id

```
int32 output_id
```

output_id is the id of the output file.

Definition at line 128 of file input_output_netcdf.f90.

7.1.2.17 rad

```
real64 rad
```

rad is the radius of the particle, R.

It has units of m.

Definition at line 125 of file input_output_netcdf.f90.

7.1.2.18 rr_coef

```
real64 rr_coef
```

rr_coef is the reaction rate coefficient, K.

It has units of $Am^2 \left(m^3 mol^{-1}\right)^{1.5}$.

Definition at line 125 of file input_output_netcdf.f90.

7.1.2.19 sim_steps

```
int32 sim_steps
```

sim_steps is the number of simulation steps.

Definition at line 127 of file input_output_netcdf.f90.

7.1.2.20 space_steps

```
int32 space_steps
```

space_steps is the resolution of the radius.

Definition at line 127 of file input_output_netcdf.f90.

7.1.2.21 temp

```
real64 temp
```

temp is the temperature of the simulation, T.

It has units of K.

Definition at line 125 of file input_output_netcdf.f90.

7.1.2.22 thick

```
real64 thick
```

thick is the thickness of the electrode, L.

It has units of m.

Definition at line 125 of file input_output_netcdf.f90.

7.1.2.23 time_check_id

```
int32 time_check_id
```

time_check_id is the id of the final time of the simulation in the checkpoint file.

Definition at line 129 of file input_output_netcdf.f90.

7.1.2.24 tot_steps

```
int32 tot_steps
```

tot_steps is the total number of simulation steps.

Definition at line 127 of file input_output_netcdf.f90.

7.1.2.25 ts_check_id

```
int32 ts_check_id
```

ts_check_id is the id of the final time step of the simulation in the checkpoint file.

Definition at line 129 of file input_output_netcdf.f90.

7.1.2.26 vol_per

```
real64 vol_per
```

vol_per is the active material volume fraction, ϵ_{actk}.

Definition at line 126 of file input_output_netcdf.f90.

7.1.2.27 volt_do

```
logical volt_do
```

volt_do is the logical value which determines whether to calculate voltage to the output file.

Definition at line 130 of file input_output_netcdf.f90.

7.1.2.28 volt_do_int

```
int32 volt_do_int
```

volt_do_int is the binary representation of the volt_do variable.

Definition at line 131 of file input_output_netcdf.f90.

7.1.2.29 volt_out_id

```
int32 volt_out_id
```

volt_out_id is the variable id of the voltage in the output file.

Definition at line 129 of file input_output_netcdf.f90.

7.2 pde_solver Module Reference

Functions/Subroutines

- subroutine setup_crank_nicholson (A, B)

  Subroutine to setup the matrices of the system of equations.
- real(real64) function, dimension(:), allocatable crank_nicholson (A, B, c_cur)

  Crank-Nicholson function.
- real(real64) function u_scalar (x)

  Function to calculate the positive electrode OCV curve, $U(c)$.
- real(real64) function, dimension(:), allocatable u_arr (x)

  Array version for the U_scalar function.
- real(real64) function volt_scalar (cin)

  Scalar voltage calculator.
- real(real64) function, dimension(:), allocatable volt_array (arrin)

  Array voltage calculator.

Variables

- real(kind=real64) rhs_const

  Rescaled source at the flux boundary given by: $\left(dt + \frac{dt}{dr}\right) \frac{200Ri_{app}}{3\epsilon_{actk}FL}$.
- real(kind=real64) volt_con_ial

  This is a combination of parameters given by: $\frac{100i_{app}R}{3\epsilon_{actk}L}$.
- real(kind=real64) volt_con_rtf

  This is a combination of parameters given by: $\frac{2R_gT}{F}$.
- real(kind=real64) mod_dif

  The rescaled diffusion coefficient given by: $\frac{D}{R^2}$.

7.2.1 Function/Subroutine Documentation

7.2.1.1 crank_nicholson()

```
real(real64) function, dimension(:), allocatable pde_solver::crank_nicholson (
             real(real64), dimension(:,:), intent(in), allocatable A,
             real(real64), dimension(:,:), intent(in), allocatable B,
             real(real64), dimension(:), intent(in) c_cur )
```

Crank-Nicholson function.

Solves the diffusion equation with a constant diffusion coefficient and a constant i_{app} using the Crank-Nicholson algorithm. The function uses the LAPACK library, calling the dgesv function for solving systems of linear equations to evolve the given state by one timestep.

Parameters

in	*A*	The left hand side coefficient matrix of the system of equations (please refer to formulation section for specific details on the matrix elements)
in	*B*	The right hand side coefficient matrix of the system of equations (please refer to formulation section for specific details on the matrix elements)
in	*c_cur*	The current time concentration array inputed into the Crank-Nicholson solver

Definition at line 115 of file pde.f90.

7.2.1.2 setup_crank_nicholson()

```
subroutine pde_solver::setup_crank_nicholson (
             real(real64), dimension(space_steps,space_steps), intent(inout) A,
             real(real64), dimension(space_steps,space_steps), intent(inout) B )
```

Subroutine to setup the matrices of the system of equations.

This subroutine takes in the corresponding matrices and assigns their elements as the appropriate coefficients in the discretised diffusion equation using the Crank-Nicholson scheme.

Parameters

in	*A*	The left hand side coefficient matrix of the system of equations (please refer to formulation section for specific details on the matrix elements)
in	*B*	The right hand side coefficient matrix of the system of equations (please refer to formulation section for specific details on the matrix elements)

Definition at line 40 of file pde.f90.

7.2.1.3 u_arr()

```
real(real64) function, dimension(:), allocatable pde_solver::u_arr (
             real(real64), dimension(:), intent(in) x )
```

Array version for the U_scalar function.

Parameters

x	Array version of the stoichiometry

Definition at line 207 of file pde.f90.

7.2.1.4 u_scalar()

```
real(real64) function pde_solver::u_scalar (
             real(real64), intent(in) x )
```

Function to calculate the positive electrode OCV curve, $U(c)$.

OCV stands for Open Circuit Voltage. Please refer to the paper, Chang-Hui Chen et. al. 2020 J. Electrochem Soc. 167 080534, https:/doi.org/10.1149/1945-7111/ab9050.

Parameters

in	*x*	The stoichiometry

Definition at line 190 of file pde.f90.

7.2.1.5 volt_array()

```
real(real64) function, dimension(:), allocatable pde_solver::volt_array (
             real(real64), dimension(:), intent(in) arrin )
```

Array voltage calculator.

Calculates array of voltages when given an array input of concentration.

Parameters

in	*arrin*	array input of concentrations

Definition at line 261 of file pde.f90.

7.2.1.6 volt_scalar()

```
real(real64) function pde_solver::volt_scalar (
            real(real64), intent(in) cin )
```

Scalar voltage calculator.

Calculates scalar voltage when given a scalar input of concentration.

Parameters

in	*cin*	input concentration

Definition at line 235 of file pde.f90.

7.2.2 Variable Documentation

7.2.2.1 mod_dif

```
real64 mod_dif
```

The rescaled diffusion coefficient given by: $\frac{D}{R^2}$.

Definition at line 25 of file pde.f90.

7.2.2.2 rhs_const

```
real64 rhs_const
```

Rescaled source at the flux boundary given by: $\left(dt + \frac{dt}{dr}\right) \frac{200 Ri_{app}}{3\epsilon_{actk} FL}$.

Definition at line 25 of file pde.f90.

7.2.2.3 volt_con_ial

`real64 volt_con_ial`

This is a combination of parameters given by: $\frac{100 i_{app} R}{3 \epsilon_{actk} L}$.

Definition at line 25 of file pde.f90.

7.2.2.4 volt_con_rtf

`real64 volt_con_rtf`

This is a combination of parameters given by: $\frac{2 R_g T}{F}$.

Definition at line 25 of file pde.f90.

Chapter 8

Data Type Documentation

8.1 volt_calc Interface Reference

Voltage calculator interface.

Public Member Functions

- volt_scalar
  Scalar voltage calculator.
- volt_array
  Array voltage calculator.

8.1.1 Detailed Description

Voltage calculator interface.

Interface that wraps both the scalar and array voltage caclulation subroutines from the module pde_solver

Definition at line 19 of file main.f90.

8.1.2 Member Function/Subroutine Documentation

8.1.2.1 volt_array()

`volt_array`

Array voltage calculator.

Calculates array of voltages when given an array input of concentration.

8.1.2.2 volt_scalar()

`volt_scalar`

Scalar voltage calculator.

Calculates scalar voltage when given a scalar input of concentration.

The documentation for this interface was generated from the following file:

- main.f90

Chapter 9

File Documentation

9.1 Formula.md File Reference

9.2 input_output_netcdf.f90 File Reference

Module file for I/O using NetCDF.

Modules

- module input_output_netcdf

Functions/Subroutines

- subroutine error_check (ierr)

  Error checking subroutine for NetCDF.
- subroutine assign_int (file_id, act, vect, sing, var_name, var_id_in, var_id_out)

  I/O subroutine that reads integers from a NetCDF file or writes integers to a NetCDF file.
- subroutine assign_real (file_id, act, vect, sing, var_name, var_id_in, var_id_out)

  I/O subroutine that reads reals from a NetCDF file or writes reals to a NetCDF file.
- subroutine assign_exp_int (var, file_id, it, var_name, var_id_in)

  I/O subroutine that writes integer arrays to a variable with an infinite dimension.
- subroutine assign_exp_real (var, file_id, it, var_name, var_id_in)

  I/O subroutine that writes real arrays to a variable with an infinite dimension.
- subroutine create_sing_var (var_name, var_typ, var_len, file_id, units, act, var_id_out)

  I/O subroutine that creates a named variable with specific length and data type within a NetCDF file.
- subroutine create_exp_var (var_name, var_typ, var_len, file_id, units, act, var_id_out)

  I/O subroutine that creates an expanding variable with specific dimensions and data type within a NetCDF file.
- subroutine save_int (var_name, file_id, vect, sing, units, act, var_len)

  I/O subroutine that writes an integer vector or single number to existing variable within a NetCDF file.
- subroutine save_real (var_name, file_id, vect, sing, units, act, var_len)

  I/O subroutine that writes a real vector or single number to existing variable within a NetCDF file.
- subroutine fin_in_out ()

  I/O subroutine that closes NetCDF output and checkpoint files that are open.

- subroutine import_input (file_name)

  I/O subroutine that opens a NetCDF file, reads values, writes them to global variables and closes the file.
- subroutine initiate_file (file_name)

  I/O subroutine that creates a NetCDF file, initiates input variables and saves variables that are available.
- subroutine initiate_checkp (file_name)

  I/O subroutine that creates a NetCDF file, initiates checkpoint specific variables and saves available variables.
- subroutine load_checkp (check_file_name, out_file_name, conc, volt_do)

  I/O subroutine that reads from a checkpoint file and saves the concentration vector.
- subroutine update_checkp (conc, step_num)

  I/O subroutine that overwrites the concentration vector and number of time steps in the checkpoint NetCDF file.

Variables

- integer, parameter dp =kind(1.0D0)
- real(kind=real64), parameter farad = 96485.3321233100184_DP

  farad is the Faraday constant, F.
- real(kind=real64), parameter gas_con = 8.31446261815324_DP

  gas_con is the ideal gas constant, R_g.
- real(kind=real64) temp

  temp is the temperature of the simulation, T.
- real(kind=real64) rad

  rad is the radius of the particle, R.
- real(kind=real64) thick

  thick is the thickness of the electrode, L.
- real(kind=real64) rr_coef

  rr_coef is the reaction rate coefficient, K.
- real(kind=real64) dif_coef

  dif_coef is the diffusion coefficient, D.
- real(kind=real64) iapp

  iapp is the applied current density as a function of time, i_{app}.
- real(kind=real64) init_c

  init_c is the initial concentration, c_0.
- real(kind=real64) max_c

  max_c is the maximum concentration of the simulation, c_{max}.
- real(kind=real64) dt

  dt is the time step.
- real(kind=real64) vol_per

  vol_per is the active material volume fraction, ϵ_{actk}.
- real(kind=real64) final_time

  final_time is the time of the final time step.
- integer(kind=int32) sim_steps

  sim_steps is the number of simulation steps.
- integer(kind=int32) out_steps

  out_steps is the number of steps before writing to output file.
- integer(kind=int32) space_steps

  space_steps is the resolution of the radius.
- integer(kind=int32) tot_steps

  tot_steps is the total number of simulation steps.
- integer(kind=int32) output_id

  output_id is the id of the output file.

- integer(kind=int32) check_id

  check_id is the id of the checkpoint file.
- integer(kind=int32) volt_out_id

  volt_out_id is the variable id of the voltage in the output file.
- integer(kind=int32) conc_out_id

  conc_out_id is the variable id of the concentration in the output file.
- integer(kind=int32) conc_check_id

  conc_check_id is the variable id of the concentration in the checkpoint file.
- integer(kind=int32) time_check_id

  time_check_id is the id of the final time of the simulation in the checkpoint file.
- integer(kind=int32) ts_check_id

  ts_check_id is the id of the final time step of the simulation in the checkpoint file.
- logical volt_do

  volt_do is the logical value which determines whether to calculate voltage to the output file.
- logical checkpoint

  checkpoint is the logical value which determines whether to restart simulation from the checkpoint.
- integer(kind=int32) volt_do_int

  volt_do_int is the binary representation of the volt_do variable.
- integer(kind=int32) checkpoint_int

  checkpoint_int is the binary representation of the checkpoint_int.

9.2.1 Detailed Description

Module file for I/O using NetCDF.

This contains the subroutines necessary for reading and writing NetCDF files in the appropriate format.

9.3 main.f90 File Reference

Main fortran file which calls other functions from other modules.

Data Types

- interface volt_calc

  Voltage calculator interface.

Functions/Subroutines

- program main

9.3.1 Detailed Description

Main fortran file which calls other functions from other modules.

This main program calls the functions and subroutines from the pde.f90 file to solve the diffusion equation, given an input netcdf file. It contains a checkpoint system as a failsafe during simulation runtime so that the entire simulation is not lost in the event of an unforeseen error.

9.3.2 Function/Subroutine Documentation

9.3.2.1 main()

```
program main
```

Definition at line 1 of file main.f90.

9.4 Miscellaneous_Code.md File Reference

9.5 pde.f90 File Reference

Module file for solving the diffusion equation and voltage calculation.

Modules

- module pde_solver

Functions/Subroutines

- subroutine setup_crank_nicholson (A, B)

  Subroutine to setup the matrices of the system of equations.
- real(real64) function, dimension(:), allocatable crank_nicholson (A, B, c_cur)

  Crank-Nicholson function.
- real(real64) function u_scalar (x)

  Function to calculate the positive electrode OCV curve, $U(c)$.
- real(real64) function, dimension(:), allocatable u_arr (x)

  Array version for the U_scalar function.
- real(real64) function volt_scalar (cin)

  Scalar voltage calculator.
- real(real64) function, dimension(:), allocatable volt_array (arrin)

  Array voltage calculator.

Variables

- real(kind=real64) rhs_const

  Rescaled source at the flux boundary given by: $\left(dt + \frac{dt}{dr}\right) \frac{200Ri_{app}}{3\epsilon_{actk}FL}$.
- real(kind=real64) volt_con_ial

  This is a combination of parameters given by: $\frac{100i_{app}R}{3\epsilon_{actk}L}$.
- real(kind=real64) volt_con_rtf

  This is a combination of parameters given by: $\frac{2R_gT}{F}$.
- real(kind=real64) mod_dif

  The rescaled diffusion coefficient given by: $\frac{D}{R^2}$.

9.5.1 Detailed Description

Module file for solving the diffusion equation and voltage calculation.

This contains the subroutines necessary for evolving the state of the system under diffusion and calculating the voltage for each simulation timestep.

9.6 README.md File Reference

Index