Dandeliion Client

Release 0.2.0

Dandeliion Team

Jul 28, 2024

CONTENTS

The Dandeliion client provides a pybamm-like python interface to run and access simulations on a remote Dandeliion server instance.

Example for such a python script:

```
import dandeliion.client as dandeliion
import numpy as np
import matplotlib.pyplot as plt
from pybamm import Experiment

credential = ("juser", "dandeliion")
simulator = dandeliion.Simulator(credential)

params = 'example_bpx.json'
experiment = Experiment(
  [
      (
          "Discharge at 10 A for 100 seconds",
          "Rest for 10 seconds",
          "Charge at 6 A for 100 seconds",
      )
  ]
  * 2,
  termination="250 s",
  period="1 second",
)

var_pts = {"x_n": 16, "x_s": 8, "x_p": 16, "r_n": 16, "r_p": 16}

initial_condition = (
  {
      "Initial concentration in electrolyte [mol.m-3]": 1000.0,
      "Initial state of charge": 1.0,
      "Initial temperature [K]": 298.15,
  }
)

solution = dandeliion.solve(
      simulator=simulator,
      params=params,
      model='DFN',
      experiment=experiment,
      initial_condition=initial_condition,
      var_pts=var_pts,
))

plt.plot(
    solution["Time [s]"],
    solution["Current [A]"],
    label="DandeLiion",
)  # Current vs Time
```

where the following classes & methods have been used:

client.Simulator	Data class containing informations about simulation server.
client.Experiment	Base class for experimental conditions under which to run the model.
client.Solution	Dictionary-style class for the solutions of a simulation run returned by `solve()`.
client.solve	Method for submitting/running a Dandeliion simulation.

CHAPTER

ONE

CLIENT.SIMULATOR

class client.**Simulator**(*credential: tuple[str, str]*)

Data class containing informations about simulation server.

credential

tuple consisting of the username and password

Type
tuple[str,str]

__init__(*credential: tuple[str, str]*) → None

Methods

__init__(credential)

Attributes

credential

CHAPTER

TWO

CLIENT.EXPERIMENT

class client.**Experiment**(*operating_conditions: list[str | tuple[str]]*, *period: str = '1 minute'*, *temperature: float | None = None*, *termination: list[str] | None = None*)

Base class for experimental conditions under which to run the model. In general, a list of operating conditions should be passed in. Each operating condition should be either a *pybamm.step.BaseStep* class, which can be created using one of the methods *pybamm.step.current*, *pybamm.step.c_rate*, *pybamm.step.voltage* , *pybamm.step.power*, *pybamm.step.resistance*, or *pybamm.step.string*, or a string, in which case the string is passed to *pybamm.step.string*.

Parameters

- **operating_conditions** (*list[str]*) – List of strings representing the operating conditions.
- **period** (*str, optional*) – Period (1/frequency) at which to record outputs. Default is 1 minute. Can be overwritten by individual operating conditions.
- **temperature** (*float, optional*) – The ambient air temperature in degrees Celsius at which to run the experiment. Default is None whereby the ambient temperature is taken from the parameter set. This value is overwritten if the temperature is specified in a step.
- **termination** (*list[str], optional*) – List of strings representing the conditions to terminate the experiment. Default is None. This is different from the termination for individual steps. Termination for individual steps is specified in the step itself, and the simulation moves to the next step when the termination condition is met (e.g. 2.5V discharge cut-off). Termination for the experiment as a whole is specified here, and the simulation stops when the termination condition is met (e.g. 80% capacity).

__init__(*operating_conditions: list[str | tuple[str]]*, *period: str = '1 minute'*, *temperature: float | None = None*, *termination: list[str] | None = None*)

Methods

__init__(operating_conditions[, period, ...])	
copy()	
process_steps(unprocessed_steps, period, temp)	
read_termination(termination)	Read the termination reason.
search_tag(tag)	Search for a tag in the experiment and return the cycles in which it appears.

CHAPTER

THREE

CLIENT.SOLUTION

class client.**Solution**(*sim: Simulation*)

Dictionary-style class for the solutions of a simulation run returned by *solve()*. Currently contains:

- 'Time [s]'
- 'Voltage [V]'
- 'Current [A]'

__init__(*sim: Simulation*)

Parameters
sim (*Simulation*) – Dandeliion simulation run (has to have finished successfully)

Methods

__init__(sim)

CHAPTER

FOUR

CLIENT.SOLVE

client.**solve**(*simulator:* Simulator, *params: str*, *experiment:* Experiment, *var_pts: dict*, *model: str*, *initial_condition: dict = None*, *t_output: list = None*, *dt_eval: float = 0.1*) → *Solution*

Method for submitting/running a Dandeliion simulation.

Parameters

- **simulator** (Simulator) – instance of simulator class providing information to connect to simulation server
- **params** (*str*) – path to BPX parameter file
- **experiment** (Experiment) – instance of pybamm Experiment; currently only those supported with
  - only `pybamm.experiment.step.steps.Current` steps (or their equivalent in str representation) as steps
  - time and/or voltage termination criteria
- **var_pts** (*dict*) – simulation mesh specified by the following parameters in dictionary:
  - x_n: Number of nodes in the electrolyte (negative electrode)
  - x_s: Number of nodes in the electrolyte (separator)
  - x_p: Number of nodes in the electrolyte (positive electrode)
  - r_n: Number of nodes in particles (negative electrode)
  - r_p: Number of nodes in particles (positive electrode)
- **model** (*str*) – name of model to be simulated
- **initial_condition** (*dict, optional*) – dictionary of additional initial conditions (overwrites parameters provided in parameter file if they exist). Currently supported initial conditions are:
  - 'Initial temperature [K]'
  - 'Initial concentration in electrolyte [mol.m-3]'
  - 'Initial state of charge'
- **t_output** (*list, optional*) – list of times to create outputs for. If not provided, then output times derived from experiment will be used (requires time stop criterion to be provided then)
- **dt_eval** (*float, optional*) – time step used for resolving discontinuities in experiment. Default is 0.1 seconds.

Returns
solution for this simulation run

Return type
Solution

INDEX

Symbols

C

E

S