acnsim Documentation

Release 0.1

Zachary Lee, Daniel Johansson

Jun 18, 2021

ACN-Data:

CHAPTER 1

ACN-Data

ACN-Data is a publicly accessible dataset for EV charging research.

1.1 Data Client

class acnportal.acndata.**DataClient**(*api_token*, *url='https://ev.caltech.edu/api/v1/'*)

API client for acndata.

Parameters

- **api_token** (*str*) – API token needed to access the acndata API.
- **url** (*str*) – Base url for all API calls. Defaults to the standard acndata API url.

token

See api_token in Args.

Type str

url

See url in Args.

Type str

count_sessions(*site*, *cond=None*)

Return the number of sessions which match the given query

Parameters

- **site** (*str*) – ACN ID from which data should be gathered.
- **cond** (*str*) – String of conditions. See API reference for the where parameter.

Returns Number of sessions which match the query.

Return type int

Raises ValueError – Raised if the site name is not valid.

get_sessions(*site*, *cond=None*, *project=None*, *sort=None*, *timeseries=False*)
Generator to return sessions from the acndata dataset one at a time.

Parameters

- **site** (*str*) – ACN ID from which data should be gathered.
- **cond** (*str*) – String of conditions. See API reference for the where parameter.

Yields *Dict* – Session as a dictionary.

Raises ValueError – Raised if the site name is not valid.

get_sessions_by_time(*site*, *start: Optional[datetime.datetime] = None*, *end: Optional[datetime.datetime] = None*, *min_energy=None*, *timeseries=False*, *count=False*)
Wrapper for get_sessions with condition based on start and end times and a minimum energy delivered.

Parameters

- **site** (*str*) – Site where data should be gathered.
- **start** (*datetime*) – Only return sessions which began after start.
- **end** (*datetime*) – Only return session which began before end.
- **min_energy** (*float*) – Only return sessions where the kWhDelivered is greater than or equal to min_energy.
- **timeseries** (*bool*) – If True return the time-series of charging rates and pilot signals. Default False.
- **count** (*bool*) – If True return the number of sessions which would be returned by the function. Default False.

Yields If count is False see get_sessions else see count_sessions.

Raises See get_sessions/count_sessions.

CHAPTER 2

ACN-Sim

ACN-Sim is a simulation environment for large-scale EV charging research.

2.1 Simulator

class acnportal.acnsim.simulator.**Simulator**(*network*, *scheduler*, *events*, *start*, *period: float = 1*, *signals=None*, *store_schedule_history=False*, *verbose=True*, *interface_type=<class 'acnportal.acnsim.interface.Interface'>*)

Central class of the acnsim package.

The Simulator class is the central place where everything about a particular simulation is stored including the network, scheduling algorithm, and events. It is also where timekeeping is done and orchestrates calling the scheduling algorithm, sending pilots to the network, and updating the energy delivered to each EV.

Parameters

- **network** (ChargingNetwork) – The charging network which the simulation will use.
- **scheduler** (*BaseAlgorithm*) – The scheduling algorithm used in the simulation. If scheduler = None, Simulator.run() cannot be called.
- **events** (EventQueue) – Queue of events which will occur in the simulation.
- **start** (*datetime*) – Date and time of the first period of the simulation.
- **period** (*float*) – Length of each time interval in the simulation in minutes. Default: 1
- **signals** (*Dict[str, ..]*) –
- **store_schedule_history** (*bool*) – If True, store the scheduler output each time it is run. Note this can use lots of memory for long simulations.
- **interface_type** (*type*) – The class of interface to register with the scheduler.

charging_rates_as_df()
Return the charging rates as a pandas DataFrame, with EVSE id as columns and iteration as index.

Returns

A DataFrame containing the charging rates of the simulation. Columns are EVSE id, and the index is the iteration.

Return type pandas.DataFrame

get_active_evs()
Return all EVs which are plugged in and not fully charged at the current time.

Wrapper for self.network.active_evs. See its documentation for more details.

Returns List of all EVs which are plugged in but not fully charged at the current time.

Return type List[*EV*]

index_of_evse(*station_id*)
Return the numerical index of the EVSE given by station_id in the (ordered) dictionary of EVSEs.

pilot_signals_as_df()
Return the pilot signals as a pandas DataFrame

Returns

A DataFrame containing the pilot signals of the simulation. Columns are EVSE id, and the index is the iteration.

Return type pandas.DataFrame

run()
If scheduler is not None, run the simulation until the event queue is empty.

The run function is the heart of the simulator. It triggers all actions and keeps the simulator moving forward. Its actions are (in order):

1. Get current events from the event queue and execute them.
2. If necessary run the scheduling algorithm.
3. Send pilot signals to the network.
4. Receive back actual charging rates from the network and store the results.

Returns None

Raises `TypeError` – If called when the scheduler attribute is None. The run() method requires a BaseAlgorithm-like scheduler to execute.

step(*new_schedule*)
Step the simulation until the next schedule recompute is required.

The step function executes a single iteration of the run() function. However, the step function updates the simulator with an input schedule rather than query the scheduler for a new schedule when one is required. Also, step will return a flag if the simulation is done.

Parameters **new_schedule** (*Dict[str, List[number]]*) – Dictionary mapping station ids to a schedule of pilot signals.

Returns True if the simulation is complete.

Return type bool

update_scheduler(*new_scheduler*)
Updates a Simulator's schedule.

exception acnportal.acnsim.simulator.**InvalidScheduleError**
Raised when the schedule passed to the simulator is invalid.

2.2 Interface

This module contains methods for directly interacting with the _simulator.

class acnportal.acnsim.interface.**Constraint**(*constraint_matrix*, *magnitudes*, *constraint_index*, *evse_index*)

constraint_index
Alias for field number 2

constraint_matrix
Alias for field number 0

evse_index
Alias for field number 3

magnitudes
Alias for field number 1

class acnportal.acnsim.interface.**InfrastructureInfo**(*constraint_matrix: numpy.ndarray, constraint_limits: numpy.ndarray, phases: numpy.ndarray, voltages: numpy.ndarray, constraint_ids: List[str], station_ids: List[str], max_pilot: numpy.ndarray, min_pilot: numpy.ndarray, allowable_pilots: Optional[List[numpy.ndarray]] = None, is_continuous: Optional[numpy.ndarray] = None*)
Class to store information about the electrical infrastructure.

Parameters

- **constraint_matrix** (*np.array[float]*) – M x N array relating the individual station currents to aggregate currents each of which is subject to a constraint. M is the number of constraints and N is the number of stations.
- **constraint_limits** (*np.array[float]*) – Limits on each constrained link. Length M.
- **phases** (*np.array[float]*) – Phase angle of the current at each station (EVSE). Length N. [deg]
- **voltages** (*np.array[float]*) – Voltage of each station. Length N. [V]
- **constraint_ids** (*List[str]*) – Unique identifier of each constraint.
- **station_ids** (*List[str]*) – Unique identifier of each station.
- **(np.array[float]** (*min_pilot*) – Maximum pilot signal supported by each station.

- **(np.array[float]** – Minimum pilot signal supported by each station. A non-zero min_pilot indicates that the station does not support any charging rates between 0 and min_pilot. It is implied that all EVSEs support a pilot signal of 0, even if min_pilot > 0.
- **allowable_pilots** (*List[np.array[float]*) – Pilot signals which each station supports. The allowable pilot signals for station i are stored in allowable_pilots[i]. If a station supports continuous pilot signals, the list is of length 2, where the first value is the lower bound on the continuous interval and the second is the upper bound. In continuous case, it is implied that all EVSEs support a pilot signal of 0, even if the continuous interval does not include 0.
- **is_continuous** (*np.array[bool]*) – True if a station supports continuous pilot signals, False otherwise.

get_station_index (*station_id: str*) → int
Get the numerical index of a given station_id in the network.

num_stations
Return total number of stations in this network.

class acnportal.acnsim.interface.**Interface** (*simulator: Simulator*)
Interface between algorithms and the ACN Simulation Environment.

active_evs
Returns a list of active EVs for use by the algorithm.

Returns List of EVs currently plugged in and not finished.

Return type List[*EV*]

active_sessions () → List[acnportal.acnsim.interface.SessionInfo]

Return a copy of the list of SessionInfo objects describing the currently charging EVs.

Returns List of currently active charging sessions.

Return type List[*SessionInfo*]

allowable_pilot_signals (*station_id: str*) → Tuple[bool, List[float]]
Returns the allowable pilot signal levels for the specified EVSE. One may assume an EVSE pilot signal of 0 is allowed regardless of this function's return values.

Parameters **station_id** (*str*) – The ID of the station for which the allowable rates should be returned.

Returns

If the range is continuous or not list[float]: The sorted set of acceptable pilot signals. If continuous this

range will have 2 values the min and the max acceptable values. [A]

Return type bool

current_datetime
Get the simulated wall time of the simulator.

Returns

The datetime corresponding to the current time step of the simulator.

Return type datetime

current_time
Get the current time (the current _iteration) of the simulator.

Returns The current _iteration of the simulator.

Return type int

evse_phase(*station_id: str*) → float
Returns the phase angle of the EVSE.

Parameters **station_id** (*str*) – The ID of the station.

Returns phase angle of the EVSE. [degrees]

Return type float

evse_voltage(*station_id: str*) → float
Returns the voltage of the EVSE.

Parameters **station_id** (*str*) – The ID of the station.

Returns voltage of the EVSE. [V]

Return type float

get_constraints() → acnportal.acnsim.interface.Constraint
Get the constraint matrix and corresponding EVSE ids for the network.

Returns

namedtuple including the following attributes:

constraint_matrix (np.ndarray): Matrix representing the constraints of the network. Each row is a constraint and each column is an index.

constraint_limits (np.ndarray): Vector of bounding limits for each constraint (1 for each constraint).

constraint_ids (List[str]): Names of each constraint. station_ids (List[str]): Names of each station.

Return type *Constraint*

get_demand_charge(*start: Optional[int] = None*) → float
Get the demand charge for the given period. ($/kW)

Parameters **start** (*int*) – Time step of the simulation where price vector should begin. If None, uses the current timestep of the simulation. Default None.

Returns Demand charge for the given period. ($/kW)

Return type float

get_prev_peak() → float
Get the highest aggregate peak demand so far in the simulation.

Returns Peak demand so far in the simulation. (A)

Return type float

get_prices(*length: int*, *start: Optional[int] = None*) → numpy.ndarray
Get a vector of prices beginning at time start and continuing for length periods. ($/kWh)

Parameters

- **length** (*int*) – Number of elements in the prices vector. One entry per period.

- **start** (*int*) – Time step of the simulation where price vector should begin. If None, uses the current timestep of the simulation. Default None.

Returns

Array of floats where each entry is the price for the corresponding period. ($/kWh)

Return type np.ndarray[float]

infrastructure_info() → acnportal.acnsim.interface.InfrastructureInfo
Returns a copy of the InfrastructureInfo object generated from interface.

Returns A description of the charging infrastructure.

Return type *InfrastructureInfo*

is_feasible(*load_currents: Dict[str, List[float]], linear: bool = False, violation_tolerance: Optional[float] = None, relative_tolerance: Optional[float] = None*) → bool
Return if a set of current magnitudes for each load are feasible.

Wraps Network's is_feasible method.

For a given constraint, the larger of the violation_tolerance and relative_tolerance is used to evaluate feasibility.

Parameters

- **load_currents** (*Dict[str, List[number]]*) – Dictionary mapping load_ids to schedules of charging rates.
- **linear** (*bool*) – If True, linearize all constraints to a more conservative but easier to compute constraint by ignoring the phase angle and taking the absolute value of all load coefficients. Default False.
- **violation_tolerance** (*float*) – Absolute amount by which schedule may violate network constraints. Default None, in which case the network's violation_tolerance attribute is used.
- **relative_tolerance** (*float*) – Relative amount by which schedule may violate network constraints. Default None, in which case the network's relative_tolerance attribute is used.

Returns

If load_currents is feasible at time t according to this set of constraints.

Return type bool

last_actual_charging_rate

Return the actual charging rates in the last period for all active sessions.

Returns

A dictionary with the session ID as key and actual charging rate as value.

Return type Dict[str, number]

last_applied_pilot_signals

Return the pilot signals that were applied in the last _iteration of the simulation for all active EVs.

Does not include EVs that arrived in the current _iteration.

Returns

A dictionary with the session ID as key and the pilot signal as value.

Return type Dict[str, float]

max_pilot_signal(*station_id: str*) → float
Returns the maximum allowable pilot signal level for the specified EVSE.

Parameters **station_id** (*str*) – The ID of the station.

Returns the maximum pilot signal supported by this EVSE. [A]

Return type float

max_recompute_time
Return the maximum recompute time of the simulator.

Returns

Maximum recompute time of the simulation in number of periods. [periods]

Return type int

min_pilot_signal(*station_id: str*) → float
Returns the minimum allowable pilot signal level for the EVSE. A zero pilot signal is always assumed to be allowed; the minimum allowable pilot signal returned here is the minimum nonzero pilot signal allowed by the EVSE if said EVSE is non-continuous.

Parameters **station_id** (*str*) – The ID of the station.

Returns the minimum pilot signal supported by this EVSE. [A]

Return type float

period
Return the length of each timestep in the simulation.

Returns Length of each time interval in the simulation. [minutes]

Return type float

remaining_amp_periods(*ev: acnportal.acnsim.interface.SessionInfo*) → float
Return the EV's remaining demand in A*periods.

Parameters **ev** (SessionInfo) – The SessionInfo object for which to get remaining demand.

Returns the EV's remaining demand in A*periods.

Return type float

exception acnportal.acnsim.interface.**InvalidScheduleError**
Raised when the schedule passed to the simulator is invalid.

class acnportal.acnsim.interface.**SessionInfo**(*station_id: str*, *session_id: str*, *requested_energy: float*, *energy_delivered: float*, *arrival: int*, *departure: int*, *estimated_departure: Optional[int] = None*, *current_time: int = 0*, *min_rates: Union[float, List[float]] = 0*, *max_rates: Union[float, List[float]] = inf*)
Class to store information relevant to a charging session.

Parameters

- **station_id** (*str*) – Unique identifier of the station (EVSE) where the session takes place.
- **session_id** (*str*) – Unique identifier of the charging session.
- **requested_energy** (*float*) – Energy requested by the user during the session. [kWh]

- **energy_delivered** (*float*) – Energy delivered already during the session. [kWh]
- **arrival** (*int*) – Time index when the session begins.
- **departure** (*int*) – Time index when the session ends.
- **estimated_departure** (*int*) – Time index when the user estimates the session will end.
- **current_time** (*int*) – Time index of the current time.
- **min_rates** (*Union[float, List[float]]*) – Lower bound for the charging rate of the session. If List (or np.array) length should be departure - arrival and each entry is a lower bound for the corresponding time period.
- **max_rates** (*Union[float, List[float]]*) – Upper bound for the charging rate of the session. If List (or np.array) length should be departure - arrival and each entry is a upper bound for the corresponding time period.

arrival_offset
Return the time (in periods) until the arrival of the EV represented by this Session, or 0 if the EV has already arrived.

remaining_demand
Return the Session's remaining demand in kWh.

remaining_time
Return the time remaining until the EV represented by this Session departs, or the time between the EV's departure and arrival if the EV has not arrived yet. If the EV has already departed, return 0.

2.3 Charging Network

class acnportal.acnsim.network.**ChargingNetwork**(*violation_tolerance: float = 1e-05*, *relative_tolerance: float = 1e-07*)
The ChargingNetwork class describes the infrastructure of the charging network with information about the types of the charging station_schedule.

Parameters

- **violation_tolerance** (*float*) – Absolute amount by which an input charging schedule may violate network constraints (A).
- **relative_tolerance** (*float*) – Relative amount by which an input charging schedule may violate network constraints (A).

active_evs
Return all EVs which are connected to an EVSE and which are not already fully charged.

Returns List of EVs which can currently be charged.

Return type List[*EV*]

active_station_ids
Return IDs for all stations which have an active EV attached.

Returns

List of the station_id of all stations which have an active EV attached.

Return type List[str]

add_constraint(*current: acnportal.acnsim.network.current.Current*, *limit: float*, *name: Optional[str] = None*) → None
Add an additional constraint to the constraint DataFrame.

Parameters

- **current** (Current) – Aggregate current which is constrained. See Current for more info.
- **limit** (*float*) – Upper limit on the aggregate current.
- **name** (*str*) – Name of this constraint.

Returns None

constraint_current(*input_schedule: numpy.ndarray*, *constraints: Optional[List[str]] = None*, *time_indices: Optional[List[int]] = None*, *linear: bool = False*)
Return the aggregate currents subject to the given constraints. If constraints=None, return all aggregate currents.

Parameters

- **input_schedule** (*np.Array*) – 2-D matrix with each row corresponding to an EVSE and each column corresponding to a time index in the schedule.
- **constraints** (*List[str]*) – List of constraint id's for which to calculate aggregate current. If None, calculates aggregate currents for all constraints.
- **time_indices** (*List[int]*) – List of time indices for which to calculate aggregate current. If None, calculates aggregate currents for all timesteps.
- **linear** (*bool*) – If True, linearize all constraints to a more conservative but easier to compute constraint by ignoring the phase angle and taking the absolute value of all load coefficients. Default False.

Returns Aggregate currents subject to the given constraints.

Return type np.ndarray

constraints_as_df() → pandas.core.frame.DataFrame
Returns the network constraints in a pandas DataFrame.

The index is the constraint IDs, and the columns are station IDs. The magnitudes (constraint limits) must be accessed separately.

Returns The network constraints as a DataFrame.

Return type pd.DataFrame

current_charging_rates
Return the current actual charging rate of all EVSEs in the network. If no EV is attached to a given EVSE, that EVSE's charging rate is 0. In the returned array, the charging rates are given in the same order as the list of EVSEs given by station_ids

Returns

numpy ndarray of actual charging rates of all EVSEs in the network.

Return type np.Array

get_ev(*station_id: str*) → acnportal.acnsim.models.ev.EV
Return the EV attached to the specified EVSE.

Parameters **station_id** (*str*) – ID of the EVSE.

Returns The EV attached to the specified station.

Return type *EV*

is_feasible(*schedule_matrix: numpy.ndarray*, *linear: bool = False*, *violation_tolerance: Optional[float] = None*, *relative_tolerance: Optional[float] = None*) → bool

Return if a set of current magnitudes for each load are feasible.

For a given constraint, the larger of the violation_tolerance and relative_tolerance is used to evaluate feasibility.

Parameters

- **schedule_matrix** (*np.Array*) – 2-D matrix with each row corresponding to an EVSE and each column corresponding to a time index in the schedule.
- **linear** (*bool*) – If True, linearize all constraints to a more conservative but easier to compute constraint by ignoring the phase angle and taking the absolute value of all load coefficients. Default False.
- **violation_tolerance** (*float*) – Absolute amount by which schedule_matrix may violate network constraints. Default None, in which case the network's violation_tolerance attribute is used.
- **relative_tolerance** (*float*) – Relative amount by which schedule_matrix may violate network constraints. Default None, in which case the network's relative_tolerance attribute is used.

Returns

If load_currents is feasible at time t according to this set of constraints.

Return type bool

phase_angles

Return dictionary of phase angles for all EVSEs in the network.

Returns

Dictionary mapping EVSE ids their input phase angle. [degrees]

Return type Dict[str, float]

plugin(*ev: acnportal.acnsim.models.ev.EV*, *station_id: str = None*) → None

Attach EV to a specific EVSE.

Parameters

- **ev** (EV) – EV object which will be attached to the EVSE.
- **[Depreciated]** –
- **station_id** (*str*) – ID of the EVSE.

Returns None

Raises KeyError – Raised when the station id has not yet been registered.

post_charging_update()

Hook to define actions to take after the charging update.

register_evse(*evse: acnportal.acnsim.models.evse.BaseEVSE*, *voltage: float*, *phase_angle: float*) → None

Register an EVSE with the network so it will be accessible to the rest of the simulation. This can only be called before any constraints have been registered in order to prevent dimensionality mismatch between the EVSE list and the

Parameters

- **evse** (EVSE) – An EVSE object.
- **voltage** (*float*) – Voltage feeding the EVSE (V).
- **phase_angle** (*float*) – Phase angle of the voltage/current feeding the EVSE (degrees).

Returns None

remove_constraint (*name: str*) → None
Remove a network constraint.

Parameters **name** (*str*) – Name of constraint to remove.

Returns None

station_ids
Return the IDs of all registered EVSEs.

Returns List of all registered EVSE IDs.

Return type List[str]

unplug (*station_id: str*, *session_id: str = None*) → None
Detach EV from a specific EVSE.

Parameters

- **station_id** (*str*) – ID of the EVSE.
- **session_id** (*str*) – ID of the session to be unplugged.

Returns None

Raises KeyError – Raised when the station id has not yet been registered.

update_constraint (*name: str*, *current: acnportal.acnsim.network.current.Current*, *limit: float*, *new_name: Optional[str] = None*) → None
Update a network constraint with a new aggregate current, limit, and name.

Parameters

- **name** (*str*) – Name of constraint to update.
- **current** (Current) – New current to update constraint with
- **limit** (*float*) – New upper limit to update constraint with
- **new_name** (*str*) – New name to give constraint

Returns None

update_pilots (*pilots: numpy.ndarray*, *i: int*, *period: float*) → None
Update the pilot signal sent to each EV. Also triggers the EVs to charge at the specified rate.

Note that if a pilot is not sent to an EVSE the associated EV WILL NOT charge during that period. If station_id is pilots or a list does not include the current time index, a 0 pilot signal is passed to the EVSE. Station IDs not registered in the network are silently ignored.

Parameters

- **pilots** (*np.Array*) – numpy array with a row for each station_id and a column for each time. Each entry in the Array corresponds to a charging rate (in A) at the station given by the row at a time given by the column.
- **i** (*int*) – Current time index of the simulation.
- **period** (*float*) – Length of the charging period. [minutes]

Returns None

voltages
Return dictionary of voltages for all EVSEs in the network.

Returns Dictionary mapping EVSE ids their input voltage. [V]

Return type Dict[str, float]

exception acnportal.acnsim.network.**StationOccupiedError**
Exception which is raised when trying to add an EV to an EVSE which is already occupied.

2.4 Current

class acnportal.acnsim.network.**Current**(*loads: Union[Dict[str, SupportsFloat], str, List[str], pandas.core.series.Series, None] = None*)
A simple representation of currents as an extension of pandas Series. Includes addition, subtraction, and multiplication (by scalar) operators.

loads
Dictionary which maps a load_id to its coefficient in

Type Dict[str, number]

the aggregate current.

Parameters **loads** (*Dict[str, number], str, or List[str], pd.Series*) – If dict, a dictionary mapping load_ids to coefficients. If str a load_id. If list, a list of load_ids. Default None. If None, loads will begin as an empty dict.

2.5 Sites

class acnportal.acnsim.network.sites.**CaltechACN**
Wrapper around caltech_acn for backward compatibility. Will be removed in future release.

2.6 Events

class acnportal.acnsim.events.**Event**(*timestamp: int*)
Base class for all events.

Parameters **timestamp** (*int*) – Timestamp when an event occurs (periods)

timestamp
See args.

Type int

event_type
Name of the event type.

Type str

precedence
Used to order occurrence for events that happen in the same timestep. Higher precedence events occur before lower precedence events.

Type float

type
Legacy accessor for event_type. This will be removed in a future release.

class acnportal.acnsim.events.**PluginEvent**(*timestamp: int*, *ev: acnportal.acnsim.models.ev.EV*)
Subclass of Event for EV plugins.

Parameters

- **timestamp** (*int*) – See Event.
- **ev** (EV) – The EV which will be plugged in.

class acnportal.acnsim.events.**UnplugEvent**(*timestamp: int*, *ev: acnportal.acnsim.models.ev.EV*)
Subclass of Event for EV unplugs.

Parameters

- **timestamp** (*int*) – See Event.
- **ev** (EV) – The EV which will be unplugged.

class acnportal.acnsim.events.**RecomputeEvent**(*timestamp: int*)
Subclass of Event for when the algorithm should be recomputed.

2.7 Event Queue

class acnportal.acnsim.events.**EventQueue**(*events=None*)
Queue which stores simulation events.

Parameters **events** (*List[Event]*) – A list of Event-like objects.

add_event(*event*)
Add an event to the queue.

Parameters **event** (*Event like*) – An Event-like object.

Returns None

add_events(*events*)
Add multiple events at a time to the queue.

Parameters **events** (*List[Event like]*) – A list of Event-like objects.

Returns None

empty()
Return if the queue is empty.

Returns True if the queue is empty.

Return type bool

get_current_events(*timestep*)
Return all events occurring before or during timestep.

Parameters **timestep** (*int*) – Time index in periods.

Returns List of all events occurring before or during timestep.

Return type List[*Event*]

get_event()
Return the next event in the queue.

Returns The next event in the queue.

Return type Event like

get_last_timestamp()
Return the timestamp of the last event (chronologically) in the event queue

Returns

Last timestamp in the event queue, or None if the event queue is empty.

Return type int

queue
Return the queue of events

2.8 Analysis

acnportal.acnsim.analysis.**aggregate_current**(*sim*)
Calculate the time series of aggregate current of all EVSEs within a simulation.

Parameters **sim** (Simulator) – A Simulator object which has been run.

Returns A numpy ndarray of the aggregate current at each time. [A]

Return type np.Array

acnportal.acnsim.analysis.**aggregate_power**(*sim*)
Calculate the time series of aggregate power of all EVSEs within a simulation.

Parameters **sim** (Simulator) – A Simulator object which has been run.

Returns A numpy ndarray of the aggregate power at each time. [kW]

Return type np.Array

acnportal.acnsim.analysis.**constraint_currents**(*sim*, *return_magnitudes=False*, *constraint_ids=None*)
Calculate the time series of current for each constraint in the ChargingNetwork for a simulation.

Parameters

- **sim** (Simulator) – A Simulator object which has been run.
- **return_magnitudes** (*bool*) – If true, return constraint currents as real magnitudes instead of complex numbers.
- **constraint_ids** (*List[str]*) – List of constraint names for which the current should be returned. If None, return all constraint currents.

Returns

A dictionary mapping the name of a constraint to a numpy array of the current subject to that constraint at each time.

Return type Dict (str, np.Array)

acnportal.acnsim.analysis.**current_unbalance**(*sim*, *phase_ids*, *unbalance_type='NEMA'*, *type=None*)
Calculate the current unbalance for each time in simulation.

Supports two definitions of unbalance. 1) The NEMA definition defined as the ratio of the maximum deviation of an RMS current from the average RMS current

over the average RMS current. (max(|I_a|, |I_b|, |I_c|) - 1/3 (|I_a| + |I_b| + |I_c|)) / (1/3 (|I_a| + |I_b| + |I_c|))

See https://www.powerstandards.com/Download/Brief%20Discussion%20of%20Unbalance%20Definitions.pdf for more info.

Parameters

- **sim** (Simulator) – A Simulator object which has been run.
- **phase_ids** (*List[str]*) – List of length 3 where each element is the identifier of phase A, B, and C respectively.
- **unbalance_type** (*str*) – Method to use for calculating phase unbalance. Acceptable values are 'NEMA'.

Returns Time series of current unbalance as a list with one value per timestep.

Return type List[float]

acnportal.acnsim.analysis.**datetimes_array**(*sim*)

Return a numpy array of datetimes over which the simulation was run.

The resolution of the datetimes list is equal to the period of the simulation, and the number of datetimes in the returned list is equal to the number of iterations of the simulation.

Note that timezone information is not included with the datetime array.

Parameters **sim** (Simulator) – A Simulator object which has been run.

Returns

List of datetimes over which the simulation was run.

Return type np.ndarray[np.datetime64]

Warns **UserWarning** – If this is called before sim is complete.

acnportal.acnsim.analysis.**demand_charge**(*sim*, *tariff=None*)

Calculate the total demand charge of the simulation.

Note this is only an accurate depiction of true demand charge if the simulation is exactly one billing period long.

Parameters

- **sim** (Simulator) – A Simulator object which has been run.
- **tariff** (*TimeOfUseTariff*) – Tariff structure to use when calculating energy costs.

Returns Total demand charge incurred by the simulation ($)

Return type float

acnportal.acnsim.analysis.**energy_cost**(*sim*, *tariff=None*)

Calculate the total energy cost of the simulation.

Parameters

- **sim** (Simulator) – A Simulator object which has been run.
- **tariff** (*TimeOfUseTariff*) – Tariff structure to use when calculating energy costs.

Returns Total energy cost of the simulation ($)

Return type float

acnportal.acnsim.analysis.**proportion_of_demands_met**(*sim*, *threshold=0.1*)
Calculate the percentage of charging sessions where the energy request was met.

Parameters

- **sim** (Simulator) – A Simulator object which has been run.
- **threshold** (*float*) – Close to finished a session should be to be considered finished. Default: 0.1. [kW]

Returns Proportion of sessions where the energy demand was fully met.

Return type float

acnportal.acnsim.analysis.**proportion_of_energy_delivered**(*sim*)
Calculate the percentage of total energy delivered over total energy requested.

Parameters **sim** (Simulator) – A Simulator object which has been run.

Returns Proportion of total energy requested which was delivered during the simulation.

Return type float

acnportal.acnsim.analysis.**total_energy_delivered**(*sim*)
Calculate total energy delivered in kWh.

Parameters **sim** (Simulator) – A Simulator object which has been run.

Returns Total energy delivered during the simulation [kWh]

Return type float

acnportal.acnsim.analysis.**total_energy_requested**(*sim*)
Calculate total energy requested in kWh.

Parameters **sim** (Simulator) – A Simulator object which has been run.

Returns Total energy requested during the simulation [kWh]

Return type float

2.9 Models

2.9.1 Battery

class acnportal.acnsim.models.battery.**Battery**(*capacity*, *init_charge*, *max_power*)
This class models the behavior of a battery and battery management system (BMS).

Parameters

- **capacity** (*float*) – Capacity of the battery [kWh]
- **init_charge** (*float*) – Initial charge of the battery [kWh]
- **max_power** (*float*) – Maximum charging rate of the battery [kW]

charge(*pilot*, *voltage*, *period*)
Method to "charge" the battery

Parameters

- **pilot** (*float*) – Pilot signal passed to the battery. [A]
- **voltage** (*float*) – AC voltage provided to the battery charger. [V]

- **period** (*float*) – Length of the charging period. [minutes]

Returns actual charging rate of the battery. [A]

Return type float

Raises ValueError – if voltage or period are <= 0.

current_charging_power
Returns the current draw of the battery on the AC side.

max_charging_power
Returns the maximum charging power of the Battery.

reset(*init_charge=None*)
Reset battery to initial state. If init_charge is not given (is None), the battery is reset to its initial charge on initialization.

Parameters **init_charge** (*float*) – charge battery should be reset to. [acnsim units]

Returns None

class acnportal.acnsim.models.battery.**Linear2StageBattery**(*capacity*, *init_charge*, *max_power*, *noise_level=0*, *transition_soc=0.8*, *charge_calculation='continuous'*)
Extends Battery with a simple piecewise linear model of battery dynamics based on SoC.

Battery model based on a piecewise linear approximation of battery behavior. The battery will charge at the minimum of max_rate and the pilot until it reaches _transition_soc. After this, the maximum charging rate of the battery will decrease linearly to 0 at 100% state of charge.

For more info on model: https://www.sciencedirect.com/science/article/pii/S0378775316317396

All public attributes are the same as Battery.

Parameters

- **noise_level** (*float*) – Standard deviation of the noise to add to the charging process. (kW)
- **transition_soc** (*float*) – State of charging when transitioning from constant current to constraint voltage.
- **charge_calculation** (*str*) – If 'stepwise', use the charging method from a previous version of acnportal, which assumes a constant maximal charging rate for the entire timestep during which the pilot signal is input. This charging method is less accurate than the _charge method, and should only be used for reproducing results from older versions of acnportal.

  If 'continuous' or not provided, use the _charge method, which assumes a continuously varying maximal charging rate.

charge(*pilot*, *voltage*, *period*)
Method to "charge" the battery based on a two-stage linear battery model.

Uses one of {_charge, _charge_stepwise} to charge the battery depending on the value of the charge_calculation attribute of this object.

acnportal.acnsim.models.battery.**batt_cap_fn**(*requested_energy*, *stay_dur*, *voltage*, *period*)
This function takes as input a requested energy, stay duration, and measurement parameters (voltage & period) and calculates the minimum capacity linear 2 stage battery such that it is feasible to deliver requested_energy in stay_dur periods.

The function returns this minimum total capacity along with an initial capacity, which is the maximum initial capacity the battery with given total capacity may have such that it is feasible (after charging at max rate) to deliver requested_energy in stay_dur periods. Thus, the returned total and initial capacities maximize the amount of time the battery behaves non-ideally during charging.

Parameters

- **requested_energy** (*float*) – Energy requested by this EV. If this fit uses data from ACN-Data, this requested_energy is the amount of energy actually delivered in real life.
- **stay_dur** (*float*) – Number of periods the EV stayed.
- **voltage** (*float*) – Voltage at which the battery is charged (V).
- **period** (*float*) – Number of minutes in a period (minutes).

2.9.2 EV

class acnportal.acnsim.models.ev.**EV**(*arrival*, *departure*, *requested_energy*, *station_id*, *session_id*, *battery*, *estimated_departure=None*)

Class to model the behavior of an Electrical Vehicle (ev).

Parameters

- **arrival** (*int*) – Arrival time of the ev. [periods]
- **departure** (*int*) – Departure time of the ev. [periods]
- **requested_energy** (*float*) – Energy requested by the ev on arrival. [kWh]
- **station_id** (*str*) – Identifier of the station used by this ev.
- **session_id** (*str*) – Identifier of the session belonging to this ev.
- **battery** (*Battery-like*) – Battery object to be used by the EV.

arrival
Return the arrival time of the EV.

charge(*pilot*, *voltage*, *period*)
Method to "charge" the ev.

Parameters

- **pilot** (*float*) – Pilot signal passed to the battery. [A]
- **voltage** (*float*) – AC voltage provided to the battery charger. [V]
- **period** (*float*) – Length of the charging period. [minutes]

Returns Actual charging rate of the ev. [A]

Return type float

current_charging_rate
Return the current charging rate of the EV. (float)

departure
Return the departure time of the EV. (int)

energy_delivered
Return the total energy delivered so far in this charging session. (float)

estimated_departure
Return the estimated departure time of the EV.

fully_charged
Return True if the EV's demand has been fully met. (bool)

maximum_charging_power
Return the maximum charging power of the battery.

percent_remaining
Return the percent of demand which still needs to be fulfilled. (float)

Defined as the ratio of remaining demand and requested energy.

remaining_demand
Return the remaining energy demand of this session. (float)

Defined as the difference between the requested energy of the session and the energy delivered so far.

requested_energy
Return the energy request of the EV for this session. (float) [acnsim units].

reset()
Reset battery back to its initialization. Also reset energy delivered.

Returns None.

session_id
Return the unique session identifier for this charging session. (str)

station_id
Return the unique identifier for the EVSE used for this charging session.

update_station_id(*station_id*)
Method to update the station where EV will charge.

2.9.3 EVSE

class acnportal.acnsim.models.evse.**BaseEVSE**(*station_id*)
Abstract base class to model Electric Vehicle Supply Equipment (charging station). This class is meant to be inherited from to implement new EVSEs.

Subclasses must implement the max_rate, allowable_pilot_signals, and _valid_rate methods.

_station_id
Unique identifier of the EVSE.

Type str

_ev
EV currently connected the the EVSE.

Type *EV*

_current_pilot
Pilot signal for the current time step. [acnsim units]

Type float

is_continuous
If True, this EVSE accepts a continuous range of pilot signals. If False, this EVSE accepts only a discrete set of pilot signals.

Type bool

allowable_pilot_signals
Returns the allowable pilot signal levels for this EVSE.

NOT IMPLEMENTED IN BaseEVSE. This method MUST be implemented in all subclasses.

Returns

List of acceptable pilot signal values or an interval of acceptable pilot signal values.

Return type List[float]

current_pilot
Return pilot signal for the current time step. (float)

ev
Return EV currently connected the the EVSE. (EV)

max_rate
Return maximum charging current allowed by the EVSE. (float)

min_rate
Return minimum charging current allowed by the EVSE. (float)

plugin(*ev*)
Method to attach an EV to the EVSE.

Parameters **ev** (EV) – EV which should be attached to the EVSE.

Returns None.

Raises *StationOccupiedError* – Exception raised when plugin is called by an EV is already attached to the EVSE.

set_pilot(*pilot*, *voltage*, *period*)
Apply a new pilot signal to the EVSE.

Before applying the new pilot, this method first checks if the pilot is allowed. If it is not, an InvalidRateError is raised. If the rate is valid, it is forwarded on to the attached EV if one is present. This method is also where EV charging is triggered. Thus it must be called in every time time period where the attached EV should receive charge.

Parameters

- **pilot** (*float*) – New pilot (control signal) to be sent to the attached EV. [A]
- **voltage** (*float*) – AC voltage provided to the battery charger. [V]
- **period** (*float*) – Length of the charging period. [minutes]

Returns None.

Raises *InvalidRateError* – Exception raised when pilot is not allowed by the EVSE.

station_id
Return unique identifier of the EVSE. (str)

unplug()
Method to remove an EV currently attached to the EVSE.

Sets ev to None and current_pilot to 0.

Returns None

class acnportal.acnsim.models.evse.**DeadbandEVSE**(*station_id*, *deadband_end=6*, *max_rate=inf*, *min_rate=None*)
Subclass of BaseEVSE which enforces the J1772 deadband between 0 - 6 A.

See BaseEVSE attributes.

_max_rate
Maximum charging current allowed by the EVSE.

Type float

_deadband_end
Upper end of the deadband. Pilot signals between 0 and this number are not allowed for this EVSE.

Type float

allowable_pilot_signals
Returns the allowable pilot signal levels for this EVSE.

It is implied that a 0 A signal is allowed.

Implements abstract method allowable_pilot_signals from BaseEVSE.

Returns

List of 2 values: the min and max acceptable values.

Return type list[float]

deadband_end
Return deadband end of the EVSE. (float)

max_rate
Return maximum charging current allowed by the EVSE. (float)

class acnportal.acnsim.models.evse.**EVSE**(*station_id*, *max_rate=inf*, *min_rate=0*)
This class of EVSE allows for charging in a continuous range from min_rate to max_rate.

See BaseEVSE attributes.

_max_rate
Maximum charging current allowed by the EVSE.

Type float

_min_rate
Minimum charging current allowed by the EVSE.

Type float

allowable_pilot_signals
Returns the allowable pilot signal levels for this EVSE.

Implements abstract method allowable_pilot_signals from BaseEVSE.

Returns

List of 2 values: the min and max acceptable values.

Return type list[float]

max_rate
Return maximum charging current allowed by the EVSE. (float)

min_rate
Return minimum charging current allowed by the EVSE. (float)

class acnportal.acnsim.models.evse.**FiniteRatesEVSE**(*station_id*, *allowable_rates*)
Subclass of EVSE which allows for finite allowed rate sets.

Most functionality remains the same except those differences noted below.

See BaseEVSE attributes.

allowable_rates
Iterable of rates which are allowed by the EVSE. On initialization, allowable_rates is converted into a list of rates in increasing order that includes 0 and contains no duplicate values.

Type iterable

allowable_pilot_signals
Returns the allowable pilot signal levels for this EVSE.

Implements abstract method allowable_pilot_signals from BaseEVSE.

Returns List of allowable pilot signals.

Return type list[float]

max_rate
Return maximum charging current allowed by the EVSE. (float)

min_rate
Return minimum charging current allowed by the EVSE. (float)

exception acnportal.acnsim.models.evse.**InvalidRateError**
Raised when an invalid pilot signal is passed to an EVSE.

exception acnportal.acnsim.models.evse.**StationOccupiedError**
Raised when a plugin event is called for an EVSE that already has an EV attached.

acnportal.acnsim.models.evse.**get_evse_by_type**(*station_id*, *evse_type*)
Factory to produce EVSEs of a given type.

Parameters

- **station_id** (*str*) – Unique identifier of the EVSE.
- **evse_type** (*str*) – Type of the EVSE. Currently supports 'BASIC', 'AeroVironment', and 'ClipperCreek'.

Returns an EVSE of the specified type and with the specified id.

Return type *EVSE*

CHAPTER 3

ACN-Sim Tutorials

ACN-Sim is a simulation environment for large-scale EV charging research.

If running in a new enviroment, such as Google Colab, run this first.

[1]:
```
# !git clone https://github.com/zach401/acnportal.git
# !pip install acnportal/.
```

3.1 ACN-Sim Tutorial: Lesson 1

3.1.1 Running an Experiment

by Zachary Lee

Last updated: 03/19/2019

In this first lesson we will learn how to setup and run a simulation using a built-in scheduling algorithm. After running the simulation we will learn how to use the analysis subpackage to analyze the results of the simulation.

[2]:
```
import pytz
from datetime import datetime

import matplotlib.pyplot as plt

from acnportal import acnsim
from acnportal import algorithms
```

Experiment Parameters

Next we need to define some parameters of the experiment. We define these at the begining of the file so they can be used consistently when setting up the simulation.

[3]:
```
# Timezone of the ACN we are using.
timezone = pytz.timezone('America/Los_Angeles')

# Start and End times are used when collecting data.
start = timezone.localize(datetime(2018, 9, 5))
end = timezone.localize(datetime(2018, 9, 6))

# How long each time discrete time interval in the simulation should be.
period = 5  # minutes

# Voltage of the network.
voltage = 220  # volts

# Default maximum charging rate for each EV battery.
default_battery_power = 32 * voltage / 1000 # kW

# Identifier of the site where data will be gathered.
site = 'caltech'
```

Network

An important part of any simulation is the ChargingNetwork on which it runs. The ChargingNetwork is a description of the physical system and contains both the set of EVSEs which make up the network as well as a constraint_matrix which represents the electrical infrastructure of the network. You can manually configure this network using the register_evse() and add_constraint() methods in ChargingNetwork or you can use a predefined network available in the sites module. In this case we use the predefined CaltechACN network.

[4]:
```
# For this experiment we use the predefined CaltechACN network.
cn = acnsim.sites.caltech_acn(basic_evse=True, voltage=voltage)
```

Events

Events are what drive action in the simulator. Events are stored in an EventQueue. This queue can be built manually by creating an Event object and using the add_event() or add_events() methods, or can be generated automatically.

In this case we will use acndata_events.generate_events() which is part of the events subpackage. acnevents provides utilities for generating events from the Caltech Charging Dataset. These events are based on real behavior of users charging actual EVs, so it is extremely valuable for running realistic simulations. In order to access the API we need a token. For now we can use the demo token, but it is highly recomended that you register for your own free token at ev.caltech.edu.

[5]:
```
API_KEY = 'DEMO_TOKEN'
events = acnsim.acndata_events.generate_events(API_KEY, site, start, end, period,␣
↪voltage, default_battery_power)
```

Scheduling Algorithm

The primary purpose of acnportal is to evaluate scheduling algorithms for large-scale EV charging. We will discuss how develop your own custom algorithm in Lesson 2, for now we will use one of the builtin scheduling algorithms, UncontrolledCharging.

[6]:
```
sch = algorithms.UncontrolledCharging()
```

Simulator

We next need to set up our simulation enviroment using the parts we have already defined. The Simulator constructor takes in a ChargingNetwork, Algorithm, and EventQueue. We also provide the start time of the simulation which all internal timestamps will be measured relative to. Finally we pass in the length of each period as well as a parameter called max_recomp. max_recomp controls how often the scheduling algorithm is called when no events occur. Here we have set max_recomp to 1, meaning the scheduling algorithm will be called every time step. If we had set it to 5, up to 5 time steps could occur before the scheduling algorithm was called. Note that the scheduling algorithm is always called when an event occurs. In this case, UncontrolledCharging only provides one charging rate, so it must be used with a max_recomp of 1.

[7]:
```
sim = acnsim.Simulator(cn, sch, events, start, period=period, verbose=False)
```

To execute the simulation we simply call the run() function.

[8]:
```
sim.run()
```

```
/home/docs/checkouts/readthedocs.org/user_builds/acnportal/envs/latest/lib/python3.7/
↪site-packages/acnportal-0.3.2-py3.7.egg/acnportal/acnsim/simulator.py:281:␣
↪UserWarning: Invalid schedule provided at iteration 84. Max violation is 15.
↪999989999999997 A on AV Pod at time index 0.
  UserWarning,
/home/docs/checkouts/readthedocs.org/user_builds/acnportal/envs/latest/lib/python3.7/
↪site-packages/acnportal-0.3.2-py3.7.egg/acnportal/acnsim/simulator.py:281:␣
↪UserWarning: Invalid schedule provided at iteration 85. Max violation is 15.
↪999989999999997 A on AV Pod at time index 0.
  UserWarning,
/home/docs/checkouts/readthedocs.org/user_builds/acnportal/envs/latest/lib/python3.7/
↪site-packages/acnportal-0.3.2-py3.7.egg/acnportal/acnsim/simulator.py:281:␣
↪UserWarning: Invalid schedule provided at iteration 86. Max violation is 15.
↪999989999999997 A on AV Pod at time index 0.
  UserWarning,
/home/docs/checkouts/readthedocs.org/user_builds/acnportal/envs/latest/lib/python3.7/
↪site-packages/acnportal-0.3.2-py3.7.egg/acnportal/acnsim/simulator.py:281:␣
↪UserWarning: Invalid schedule provided at iteration 87. Max violation is 15.
↪999989999999997 A on AV Pod at time index 0.
  UserWarning,
/home/docs/checkouts/readthedocs.org/user_builds/acnportal/envs/latest/lib/python3.7/
↪site-packages/acnportal-0.3.2-py3.7.egg/acnportal/acnsim/simulator.py:281:␣
↪UserWarning: Invalid schedule provided at iteration 88. Max violation is 15.
↪999989999999997 A on AV Pod at time index 0.
  UserWarning,
/home/docs/checkouts/readthedocs.org/user_builds/acnportal/envs/latest/lib/python3.7/
↪site-packages/acnportal-0.3.2-py3.7.egg/acnportal/acnsim/simulator.py:281:␣
↪UserWarning: Invalid schedule provided at iteration 89. Max violation is 47.99999 A␣
↪on AV Pod at time index 0.
  UserWarning,
/home/docs/checkouts/readthedocs.org/user_builds/acnportal/envs/latest/lib/python3.7/
↪site-packages/acnportal-0.3.2-py3.7.egg/acnportal/acnsim/simulator.py:281:␣
↪UserWarning: Invalid schedule provided at iteration 90. Max violation is 47.99999 A␣
↪on AV Pod at time index 0.
  UserWarning,
/home/docs/checkouts/readthedocs.org/user_builds/acnportal/envs/latest/lib/python3.7/
↪site-packages/acnportal-0.3.2-py3.7.egg/acnportal/acnsim/simulator.py:281:␣
↪UserWarning: Invalid schedule provided at iteration 91. Max violation is 47.99999 A␣
↪on AV Pod at time index 0.
  UserWarning,
/home/docs/checkouts/readthedocs.org/user_builds/acnportal/envs/latest/lib/python3.7/
↪site-packages/acnportal-0.3.2-py3.7.egg/acnportal/acnsim/simulator.py:281:␣
↪UserWarning: Invalid schedule provided at iteration 92. Max violation is 15.
↪999989999999997 A on AV Pod at time index 0.
```

(continues on next page)

(continued from previous page)

```
  UserWarning,
/home/docs/checkouts/readthedocs.org/user_builds/acnportal/envs/latest/lib/python3.7/
↪site-packages/acnportal-0.3.2-py3.7.egg/acnportal/acnsim/simulator.py:281:␣
↪UserWarning: Invalid schedule provided at iteration 93. Max violation is 47.99999 A␣
↪on AV Pod at time index 0.
  UserWarning,
/home/docs/checkouts/readthedocs.org/user_builds/acnportal/envs/latest/lib/python3.7/
↪site-packages/acnportal-0.3.2-py3.7.egg/acnportal/acnsim/simulator.py:281:␣
↪UserWarning: Invalid schedule provided at iteration 94. Max violation is 47.99999 A␣
↪on AV Pod at time index 0.
  UserWarning,
/home/docs/checkouts/readthedocs.org/user_builds/acnportal/envs/latest/lib/python3.7/
↪site-packages/acnportal-0.3.2-py3.7.egg/acnportal/acnsim/simulator.py:281:␣
↪UserWarning: Invalid schedule provided at iteration 95. Max violation is 79.99999 A␣
↪on AV Pod at time index 0.
  UserWarning,
/home/docs/checkouts/readthedocs.org/user_builds/acnportal/envs/latest/lib/python3.7/
↪site-packages/acnportal-0.3.2-py3.7.egg/acnportal/acnsim/simulator.py:281:␣
↪UserWarning: Invalid schedule provided at iteration 96. Max violation is 79.99999 A␣
↪on AV Pod at time index 0.
  UserWarning,
/home/docs/checkouts/readthedocs.org/user_builds/acnportal/envs/latest/lib/python3.7/
↪site-packages/acnportal-0.3.2-py3.7.egg/acnportal/acnsim/simulator.py:281:␣
↪UserWarning: Invalid schedule provided at iteration 97. Max violation is 79.99999 A␣
↪on AV Pod at time index 0.
  UserWarning,
/home/docs/checkouts/readthedocs.org/user_builds/acnportal/envs/latest/lib/python3.7/
↪site-packages/acnportal-0.3.2-py3.7.egg/acnportal/acnsim/simulator.py:281:␣
↪UserWarning: Invalid schedule provided at iteration 98. Max violation is 79.99999 A␣
↪on AV Pod at time index 0.
  UserWarning,
/home/docs/checkouts/readthedocs.org/user_builds/acnportal/envs/latest/lib/python3.7/
↪site-packages/acnportal-0.3.2-py3.7.egg/acnportal/acnsim/simulator.py:281:␣
↪UserWarning: Invalid schedule provided at iteration 99. Max violation is 79.99999 A␣
↪on AV Pod at time index 0.
  UserWarning,
/home/docs/checkouts/readthedocs.org/user_builds/acnportal/envs/latest/lib/python3.7/
↪site-packages/acnportal-0.3.2-py3.7.egg/acnportal/acnsim/simulator.py:281:␣
↪UserWarning: Invalid schedule provided at iteration 100. Max violation is 79.99999␣
↪A on AV Pod at time index 0.
  UserWarning,
/home/docs/checkouts/readthedocs.org/user_builds/acnportal/envs/latest/lib/python3.7/
↪site-packages/acnportal-0.3.2-py3.7.egg/acnportal/acnsim/simulator.py:281:␣
↪UserWarning: Invalid schedule provided at iteration 101. Max violation is 79.99999␣
↪A on AV Pod at time index 0.
  UserWarning,
/home/docs/checkouts/readthedocs.org/user_builds/acnportal/envs/latest/lib/python3.7/
↪site-packages/acnportal-0.3.2-py3.7.egg/acnportal/acnsim/simulator.py:281:␣
↪UserWarning: Invalid schedule provided at iteration 102. Max violation is 79.99999␣
↪A on CC Pod at time index 0.
  UserWarning,
/home/docs/checkouts/readthedocs.org/user_builds/acnportal/envs/latest/lib/python3.7/
↪site-packages/acnportal-0.3.2-py3.7.egg/acnportal/acnsim/simulator.py:281:␣
↪UserWarning: Invalid schedule provided at iteration 103. Max violation is 79.99999␣
↪A on CC Pod at time index 0.
  UserWarning,
/home/docs/checkouts/readthedocs.org/user_builds/acnportal/envs/latest/lib/python3.7/
↪site-packages/acnportal-0.3.2-py3.7.egg/acnportal/acnsim/simulator.py:281:␣
↪UserWarning: Invalid schedule provided at iteration 104. Max violation is 79.99999␣
↪A on CC Pod at time index 0.
```

(continues on next page)

(continued from previous page)

```
  UserWarning,
/home/docs/checkouts/readthedocs.org/user_builds/acnportal/envs/latest/lib/python3.7/
↪site-packages/acnportal-0.3.2-py3.7.egg/acnportal/acnsim/simulator.py:281:␣
↪UserWarning: Invalid schedule provided at iteration 105. Max violation is 111.99999␣
↪A on CC Pod at time index 0.
  UserWarning,
/home/docs/checkouts/readthedocs.org/user_builds/acnportal/envs/latest/lib/python3.7/
↪site-packages/acnportal-0.3.2-py3.7.egg/acnportal/acnsim/simulator.py:281:␣
↪UserWarning: Invalid schedule provided at iteration 106. Max violation is 112.
↪06009587279908 A on Secondary A at time index 0.
  UserWarning,
/home/docs/checkouts/readthedocs.org/user_builds/acnportal/envs/latest/lib/python3.7/
↪site-packages/acnportal-0.3.2-py3.7.egg/acnportal/acnsim/simulator.py:281:␣
↪UserWarning: Invalid schedule provided at iteration 107. Max violation is 83.
↪18930259135925 A on Secondary A at time index 0.
  UserWarning,
/home/docs/checkouts/readthedocs.org/user_builds/acnportal/envs/latest/lib/python3.7/
↪site-packages/acnportal-0.3.2-py3.7.egg/acnportal/acnsim/simulator.py:281:␣
↪UserWarning: Invalid schedule provided at iteration 108. Max violation is 112.
↪06009587279908 A on Secondary A at time index 0.
  UserWarning,
/home/docs/checkouts/readthedocs.org/user_builds/acnportal/envs/latest/lib/python3.7/
↪site-packages/acnportal-0.3.2-py3.7.egg/acnportal/acnsim/simulator.py:281:␣
↪UserWarning: Invalid schedule provided at iteration 109. Max violation is 110.
↪11980759427178 A on Secondary A at time index 0.
  UserWarning,
/home/docs/checkouts/readthedocs.org/user_builds/acnportal/envs/latest/lib/python3.7/
↪site-packages/acnportal-0.3.2-py3.7.egg/acnportal/acnsim/simulator.py:281:␣
↪UserWarning: Invalid schedule provided at iteration 110. Max violation is 138.
↪51257366605245 A on Secondary A at time index 0.
  UserWarning,
/home/docs/checkouts/readthedocs.org/user_builds/acnportal/envs/latest/lib/python3.7/
↪site-packages/acnportal-0.3.2-py3.7.egg/acnportal/acnsim/simulator.py:281:␣
↪UserWarning: Invalid schedule provided at iteration 111. Max violation is 138.
↪51257366605245 A on Secondary A at time index 0.
  UserWarning,
/home/docs/checkouts/readthedocs.org/user_builds/acnportal/envs/latest/lib/python3.7/
↪site-packages/acnportal-0.3.2-py3.7.egg/acnportal/acnsim/simulator.py:281:␣
↪UserWarning: Invalid schedule provided at iteration 112. Max violation is 27.
↪89153099572701 A on Secondary A at time index 0.
  UserWarning,
/home/docs/checkouts/readthedocs.org/user_builds/acnportal/envs/latest/lib/python3.7/
↪site-packages/acnportal-0.3.2-py3.7.egg/acnportal/acnsim/simulator.py:281:␣
↪UserWarning: Invalid schedule provided at iteration 113. Max violation is 15.
↪999989999999997 A on CC Pod at time index 0.
  UserWarning,
/home/docs/checkouts/readthedocs.org/user_builds/acnportal/envs/latest/lib/python3.7/
↪site-packages/acnportal-0.3.2-py3.7.egg/acnportal/acnsim/simulator.py:281:␣
↪UserWarning: Invalid schedule provided at iteration 114. Max violation is 15.
↪999989999999997 A on CC Pod at time index 0.
  UserWarning,
/home/docs/checkouts/readthedocs.org/user_builds/acnportal/envs/latest/lib/python3.7/
↪site-packages/acnportal-0.3.2-py3.7.egg/acnportal/acnsim/simulator.py:281:␣
↪UserWarning: Invalid schedule provided at iteration 115. Max violation is 15.
↪999989999999997 A on CC Pod at time index 0.
  UserWarning,
/home/docs/checkouts/readthedocs.org/user_builds/acnportal/envs/latest/lib/python3.7/
↪site-packages/acnportal-0.3.2-py3.7.egg/acnportal/acnsim/simulator.py:281:␣
↪UserWarning: Invalid schedule provided at iteration 116. Max violation is 15.
↪999989999999997 A on CC Pod at time index 0.
```

(continues on next page)

(continued from previous page)

```
  UserWarning,
/home/docs/checkouts/readthedocs.org/user_builds/acnportal/envs/latest/lib/python3.7/
↪site-packages/acnportal-0.3.2-py3.7.egg/acnportal/acnsim/simulator.py:281:␣
↪UserWarning: Invalid schedule provided at iteration 117. Max violation is 15.
↪999989999999997 A on CC Pod at time index 0.
  UserWarning,
/home/docs/checkouts/readthedocs.org/user_builds/acnportal/envs/latest/lib/python3.7/
↪site-packages/acnportal-0.3.2-py3.7.egg/acnportal/acnsim/simulator.py:281:␣
↪UserWarning: Invalid schedule provided at iteration 162. Max violation is 15.
↪999989999999997 A on CC Pod at time index 0.
  UserWarning,
/home/docs/checkouts/readthedocs.org/user_builds/acnportal/envs/latest/lib/python3.7/
↪site-packages/acnportal-0.3.2-py3.7.egg/acnportal/acnsim/simulator.py:281:␣
↪UserWarning: Invalid schedule provided at iteration 245. Max violation is 47.99999␣
↪A on AV Pod at time index 0.
  UserWarning,
/home/docs/checkouts/readthedocs.org/user_builds/acnportal/envs/latest/lib/python3.7/
↪site-packages/acnportal-0.3.2-py3.7.egg/acnportal/acnsim/simulator.py:281:␣
↪UserWarning: Invalid schedule provided at iteration 246. Max violation is 47.99999␣
↪A on AV Pod at time index 0.
  UserWarning,
/home/docs/checkouts/readthedocs.org/user_builds/acnportal/envs/latest/lib/python3.7/
↪site-packages/acnportal-0.3.2-py3.7.egg/acnportal/acnsim/simulator.py:281:␣
↪UserWarning: Invalid schedule provided at iteration 247. Max violation is 47.99999␣
↪A on AV Pod at time index 0.
  UserWarning,
/home/docs/checkouts/readthedocs.org/user_builds/acnportal/envs/latest/lib/python3.7/
↪site-packages/acnportal-0.3.2-py3.7.egg/acnportal/acnsim/simulator.py:281:␣
↪UserWarning: Invalid schedule provided at iteration 248. Max violation is 47.99999␣
↪A on AV Pod at time index 0.
  UserWarning,
/home/docs/checkouts/readthedocs.org/user_builds/acnportal/envs/latest/lib/python3.7/
↪site-packages/acnportal-0.3.2-py3.7.egg/acnportal/acnsim/simulator.py:281:␣
↪UserWarning: Invalid schedule provided at iteration 249. Max violation is 15.
↪999989999999997 A on AV Pod at time index 0.
  UserWarning,
/home/docs/checkouts/readthedocs.org/user_builds/acnportal/envs/latest/lib/python3.7/
↪site-packages/acnportal-0.3.2-py3.7.egg/acnportal/acnsim/simulator.py:281:␣
↪UserWarning: Invalid schedule provided at iteration 250. Max violation is 15.
↪999989999999997 A on AV Pod at time index 0.
  UserWarning,
/home/docs/checkouts/readthedocs.org/user_builds/acnportal/envs/latest/lib/python3.7/
↪site-packages/acnportal-0.3.2-py3.7.egg/acnportal/acnsim/simulator.py:281:␣
↪UserWarning: Invalid schedule provided at iteration 251. Max violation is 15.
↪999989999999997 A on AV Pod at time index 0.
  UserWarning,
/home/docs/checkouts/readthedocs.org/user_builds/acnportal/envs/latest/lib/python3.7/
↪site-packages/acnportal-0.3.2-py3.7.egg/acnportal/acnsim/simulator.py:281:␣
↪UserWarning: Invalid schedule provided at iteration 252. Max violation is 15.
↪999989999999997 A on AV Pod at time index 0.
  UserWarning,
/home/docs/checkouts/readthedocs.org/user_builds/acnportal/envs/latest/lib/python3.7/
↪site-packages/acnportal-0.3.2-py3.7.egg/acnportal/acnsim/simulator.py:281:␣
↪UserWarning: Invalid schedule provided at iteration 253. Max violation is 15.
↪999989999999997 A on AV Pod at time index 0.
  UserWarning,
/home/docs/checkouts/readthedocs.org/user_builds/acnportal/envs/latest/lib/python3.7/
↪site-packages/acnportal-0.3.2-py3.7.egg/acnportal/acnsim/simulator.py:281:␣
↪UserWarning: Invalid schedule provided at iteration 254. Max violation is 15.
↪999989999999997 A on AV Pod at time index 0.
```

(continues on next page)

(continued from previous page)

```
  UserWarning,
/home/docs/checkouts/readthedocs.org/user_builds/acnportal/envs/latest/lib/python3.7/
↪site-packages/acnportal-0.3.2-py3.7.egg/acnportal/acnsim/simulator.py:281:␣
↪UserWarning: Invalid schedule provided at iteration 255. Max violation is 15.
↪999989999999997 A on AV Pod at time index 0.
  UserWarning,
```

Analysis

Once the simulator has been run, we can analyze the results. For this purpose acnsim offers a package called analysis. One thing we may be interested in is the proportion of total users' energy demand that we were able to meet. To find this we can use the proportion_of_energy_delivered() method from the analysis subpackage. The only argument to this function is the Simulator object itself.

[9]:
```
total_energy_prop = acnsim.proportion_of_energy_delivered(sim)
print('Proportion of requested energy delivered: {0}'.format(total_energy_prop))
```

```
Proportion of requested energy delivered: 1.0
```

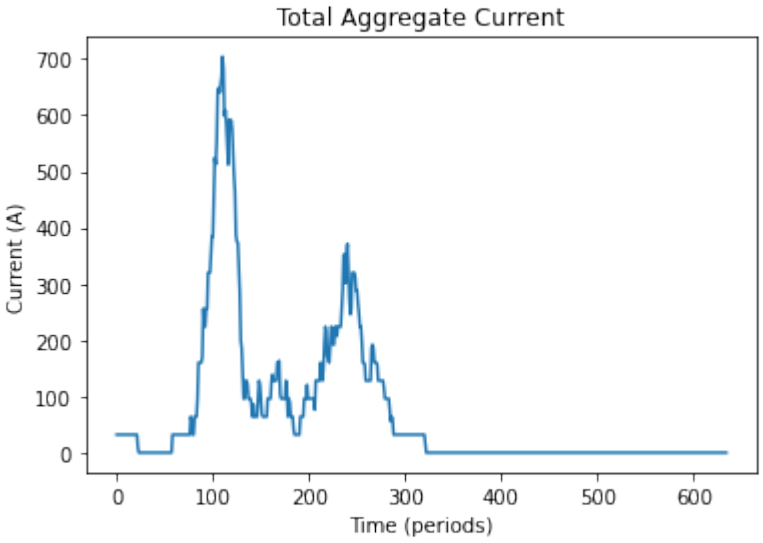
We may also be interested in the peak demand of the system as this determines our big the root transformers and cables in our system must be as well as the demand charge we may have to pay. The Simulator has a built in property which keeps track of this peak usage called peak.

[10]:
```
print('Peak aggregate current: {0} A'.format(sim.peak))
```

```
Peak aggregate current: 704.0 A
```

Finally, we can plot the output of our simulation. For now we will just plot total aggregate current draw:

[11]:
```
# Plotting aggregate current
agg_current = acnsim.aggregate_current(sim)
plt.plot(agg_current)
plt.xlabel('Time (periods)')
plt.ylabel('Current (A)')
plt.title('Total Aggregate Current')
plt.show()
```



If running in a new environment, such as Google Colab, run this first.

[1]:
```
# !git clone https://github.com/zach401/acnportal.git
# !pip install acnportal/.
```

3.2 ACN-Sim Tutorial: Lesson 2

3.2.1 Implementing a Custom Algorithm

by Zachary Lee

Last updated: 03/19/2019

In this lesson we will learn how to develop a custom algorithm and run it using ACN-Sim. For this example we will be writing an Earliest Deadline First Algorithm. This algorithm is already available as part of the SortingAlgorithm in the algorithms package, so we will compare the results of our implementation with the included one.

3.2.2 Custom Algorithm

All custom algorithms should inherit from the abstract class BaseAlgorithm. It is the responsibility of all derived classes to implement the schedule method. This method takes as an input a list of EVs which are currently connected to the system but have not yet finished charging. Its output is a dictionary which maps a station_id to a list of charging rates. Each charging rate is valid for one period measured relative to the current period.

For Example: * schedule['abc'][0] is the charging rate for station 'abc' during the current period * schedule['abc'][1] is the charging rate for the next period * and so on.

If an algorithm only produces charging rates for the current time period, the length of each list should be 1. If this is the case, make sure to also set the maximum resolve period to be 1 period so that the algorithm will be called each period. An alternative is to repeat the charging rate a number of times equal to the max recompute period.

As mentioned previously our new algorithm should inherit from BaseAlgorithm or a subclass of it.

We can override the **init**() method if we need to pass additional configuration information to the algorithm. In this case we pass in the increment which will be used when searching for a feasible rate.

We next need to override the schedule() method. The signature of this method should remain the same, as it is called internally in Simulator. If an algorithm needs additional parameters consider passing them through the constructor.

[2]:
```
from acnportal.algorithms import BaseAlgorithm

class EarliestDeadlineFirstAlgo(BaseAlgorithm):
    """ Algorithm which assigns charging rates to each EV in order of
    estimated departure time.

    Implements abstract class BaseAlgorithm.

    For this algorithm EVs will first be sorted by estimated departure time. We will
    then allocate as much
    current as possible to each EV in order until the EV is finished charging or an␣
↪infrastructure
    limit is met.

    Args:
```

(continues on next page)

(continued from previous page)

```
        increment (number): Minimum increment of charging rate. Default: 1.
    """
    def __init__(self, increment=1):
        super().__init__()
        self._increment = increment
        self.max_recompute = 1

    def schedule(self, active_evs):
        schedule = {ev.station_id: [0] for ev in active_evs}

        # Next, we sort the active_evs by their estimated departure time.
        sorted_evs = sorted(active_evs, key=lambda x: x.estimated_departure)

        # We now iterate over the sorted list of EVs.
        for ev in sorted_evs:
            # First try to charge the EV at its maximum rate. Remember that each␣
↪schedule value
            #   must be a list, even if it only has one element.
            schedule[ev.station_id] = [self.interface.max_pilot_signal(ev.station_id)]

            # If this is not feasible, we will reduce the rate.
            #   interface.is_feasible() is one way to interact with the constraint set
            #   of the network. We will explore another more direct method in lesson␣
↪3.
            while not self.interface.is_feasible(schedule, 0):

                # Since the maximum rate was not feasible, we should try a lower rate.
                schedule[ev.station_id][0] -= self._increment

                # EVs should never charge below 0 (i.e. discharge) so we will clip␣
↪the value at 0.
                if schedule[ev.station_id][0] < 0:
                    schedule[ev.station_id] = [0]
                    break
        return schedule
```

Note the structure of the schedule dict which is returned should be something like:

```
{
    'CA-301': [32, 32, 32, 16, 16, ..., 8],
    'CA-302': [8, 13, 13, 15, 6, ..., 0],
    ...,
    'CA-408': [24, 24, 24, 24, 0, ..., 0]
}
```

For the special case when an algorithm only calculates a target rate for the next time interval instead of an entire schedule of rates, the structure should be:

```
{
    'CA-301': [32],
    'CA-302': [8],
    ...,
    'CA-408': [24]
}
```

Note that these are single element lists and NOT floats or integers.

3.2.3 Running the Algorithm

Now that we have implemented our algorithm, we can try it out using the same experiment setup as in lesson 1. The only difference will be which scheduling algorithm we use. For fun, lets compare our algorithm against to included implementation of the earliest deadline first algorithm.

[3]:
```
from datetime import datetime
import pytz
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
from copy import deepcopy

from acnportal import algorithms
from acnportal import acnsim


# -- Experiment Parameters ------------------------------------------------------
↪---------------------------------
timezone = pytz.timezone('America/Los_Angeles')
start = timezone.localize(datetime(2018, 9, 5))
end = timezone.localize(datetime(2018, 9, 6))
period = 5  # minute
voltage = 220  # volts
default_battery_power = 32 * voltage / 1000 # kW
site = 'caltech'

# -- Network --------------------------------------------------------------------
↪---------------------------------
cn = acnsim.sites.caltech_acn(basic_evse=True, voltage=voltage)

# -- Events ---------------------------------------------------------------------
↪---------------------------------
API_KEY = 'DEMO_TOKEN'
events = acnsim.acndata_events.generate_events(API_KEY, site, start, end, period,␣
↪voltage, default_battery_power)


# -- Scheduling Algorithm -------------------------------------------------------
↪---------------------------------
sch = EarliestDeadlineFirstAlgo(increment=1)
sch2 = algorithms.SortedSchedulingAlgo(algorithms.least_laxity_first)
```

[4]:
```
# -- Simulator ------------------------------------------------------------------
↪---------------------------------
sim = acnsim.Simulator(deepcopy(cn), sch, deepcopy(events), start, period=period,␣
↪verbose=False)
sim.run()
```

[5]:
```
# For comparison we will also run the builtin earliest deadline first algorithm
sim2 = acnsim.Simulator(deepcopy(cn), sch2, deepcopy(events), start, period=period,␣
↪verbose=False)
sim2.run()
```

3.2.4 Results

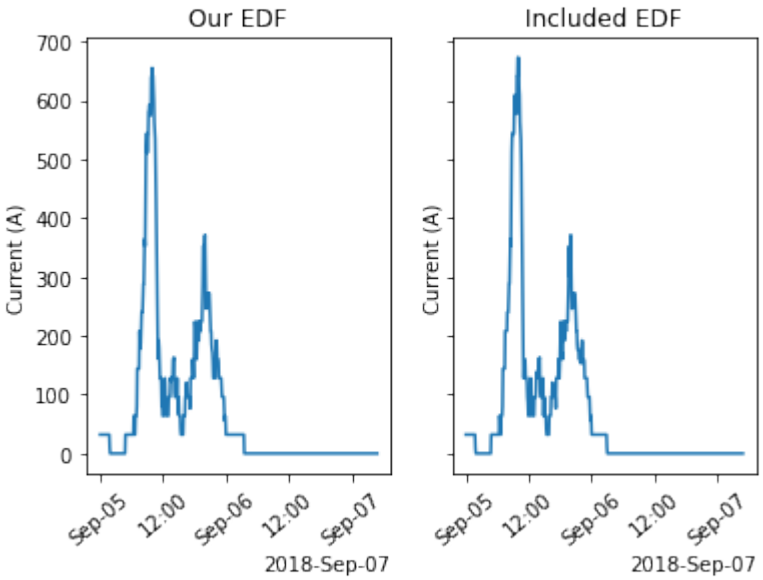
We can now compare the two algorithms side by side by looking that the plots of aggregated current. We see from these plots that our implementation matches th included one quite well. If we look closely however, we might see a small difference. This is because the included algorithm uses a more efficient bisection based method instead of our simpler linear search to find a feasible rate.

[6]:
```
# Get list of datetimes over which the simulations were run.
sim_dates = mdates.date2num(acnsim.datetimes_array(sim))
sim2_dates = mdates.date2num(acnsim.datetimes_array(sim2))

# Set locator and formatter for datetimes on x-axis.
locator = mdates.AutoDateLocator(maxticks=6)
formatter = mdates.ConciseDateFormatter(locator)

fig, axs = plt.subplots(1, 2, sharey=True, sharex=True)
axs[0].plot(sim_dates, acnsim.aggregate_current(sim), label='Our EDF')
axs[1].plot(sim2_dates, acnsim.aggregate_current(sim2), label='Included EDF')
axs[0].set_title('Our EDF')
axs[1].set_title('Included EDF')
for ax in axs:
    ax.set_ylabel('Current (A)')
    for label in ax.get_xticklabels():
        label.set_rotation(40)
    ax.xaxis.set_major_locator(locator)
    ax.xaxis.set_major_formatter(formatter)

plt.show()
```



CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

a

Index

Symbols

A

B

C

D

E

F

G

I

L

M

N

P

Q

R

S

T

U

V