PV-DER Simulation Utility

Release 0.0.1

Sep 15, 2021

Contents:

This is a reference for all the classes and methods available in PV-DER Simulation Utility.

CHAPTER 1

PV-DER package

1.1 DER models

1.1.1 pvder.DER_components_three_phase

Three phase PV-DER components.

class pvder.DER_components_three_phase.**SolarPVDERThreePhase**(*events*, *configFile=None*, ***kwargs*)

Bases: pvder.DER_components.PVModule, pvder.DER_components.SolarPVDER

Class for describing a Solar Photo-voltaic Distributed Energy Resource consisting of panel, converters, and control systems. Attributes: count (int): Number of instances of *SolarPVDERThreePhase*. n_ODE (int): Number of ODE's.

Creates an instance of *SolarPV_DER_ThreePhase*. :param events: An instance of *SimulationEvents*. :type events: SimulationEvents :param gridModel: An instance of 'Gridl'(only need to be suppled for stand alone simulation). :type gridModel: Grid :param powerRating: A scalar specifying the rated power (VA) of the DER. :type powerRating: float :param VrmsRating: A scalar specifying the rated RMS L-G voltage (V) of the DER. :type VrmsRating: float :param ia0,xa0,ua0: Initial value of inverter states in p.u. of the DER instance. :type ia0,xa0,ua0: complex :param xDC0,xQ0,xPLL0,wte0: Initial value of inverter states in the DER instance. :type xDC0,xQ0,xPLL0,wte0: float :param gridVoltatePhaseA,gridVoltatePhaseA,gridVoltatePhaseA: Initial voltage phasor (V) at PCC - LV side from external program (only need to be suppled if model is not stand alone). :type gridVoltatePhaseA,gridVoltatePhaseA,gridVoltatePhaseA: float :param standAlone: Specify if the DER instance is a stand alone simulation or part of a larger simulation. :type standAlone: bool :param STEADY_STATE_INITIALIZATION: Specify whether states in the DER instance will be initialized to steady state values. :type STEADY_STATE_INITIALIZATION: bool :param allow_unbalanced_m: Allow duty cycles to take on unbalanced values during initialization (default: False). :type allow_unbalanced_m: bool :param derConfig: Configuration parameters that may be supplied from an external program. :type derConfig: dict :param identifier: An identifier that can be used to name the instance (default: None). :type identifier: str

Raises

- ValueError – If parameters corresponding to *Sinverter_rated* are not available.

- `ValueError` – If rated DC link voltage is not sufficient.

Irms_calc()
Inverter current - RMS

ODE_model(*y*, *t*)
Derivatives for the equation.

S_G_calc()
Power absorbed/produced by grid voltage source.

S_PCC_calc()
Power output at PCC LV side

S_calc()
Inverter apparent power output

S_load1_calc()
Power absorbed by load at PCC LV side.

Vabrms_calc()
PCC LV side voltage - line to line RMS

Vrms_calc()
PCC LV side voltage - RMS

Vrms_min_calc()
PCC LV side voltage - RMS

Vtabrms_calc()
Inverter terminal voltage - line to line RMS

Vtrms_calc()
Inverter terminal voltage - RMS

jac_ODE_model(*y*, *t*)
Jacobian for the system of ODE's.

update_RMS()
Update RMS voltages.

update_inverter_frequency(*t*)
Update d-q quantities.

update_inverter_states(*ia*, *xa*, *ua*, *ib*, *xb*, *ub*, *ic*, *xc*, *uc*, *Vdc*, *xDC*, *xQ*, *xPLL*, *wte*)
Update inverter states

update_iref(*t*)
Update reference reference current.

update_power()
Update RMS voltages.

update_voltages()
Update voltages.

y0
List of initial states

1.1.2 pvder.DER_components_single_phase

Single phase PV-DER code.

class pvder.DER_components_single_phase.**SolarPVDERSinglePhase**(*events*, *configFile=None*, ***kwargs*)

Bases: pvder.DER_components.PVModule, pvder.DER_components.SolarPVDER

Class for describing a Solar Photo-voltaic Distributed Energy Resource consisting of panel, converters, and control systems.

count
Number of instances of *SolarPVDERSinglePhase*.

Type int

n_ODE
Number of ODE's.

Type int

Creates an instance of *SolarPV_DER_SinglePhase*. Args: events (SimulationEvents): An instance of *SimulationEvents*. gridModel (Grid): An instance of 'Grid'(only need to be suppled for stand alone simulation). powerRating (float): A scalar specifying the rated power (VA) of the DER. VrmsRating (float): A scalar specifying the rated RMS L-G voltage (V) of the DER. ia0,xa0,ua0 (complex): Initial value of inverter states in p.u. of the DER instance. xDC0,xQ0,xPLL0,wte0 (float): Initial value of inverter states in the DER instance. gridVoltatePhaseA,gridVoltatePhaseA,gridVoltatePhaseA (float): Initial voltage phasor (V) at PCC - LV side from external program (only need to be suppled if model is not stand alone). standAlone (bool): Specify if the DER instance is a stand alone simulation or part of a larger simulation. steadyStateInitialization (bool): Specify whether states in the DER instance will be initialized to steady state values. allowUnbalancedM (bool): Allow duty cycles to take on unbalanced values during initialization (default: False). derConfig (dict): Configuration parameters that may be supplied from an external program. identifier (str): An identifier that can be used to name the instance (default: None).

Raises: ValueError: If parameters corresponding to *Sinverter_rated* are not available. ValueError: If rated DC link voltage is not sufficient.

Irms_calc()
Inverter current - RMS

ODE_model(*y*, *t*)
System of ODE's defining the dynamic DER model. :param y: Initial conditions for the states.. :type y: list of float :param t: Simulation time in seconds. :type t: float

Returns Derivates for the system of ODE's.

Return type result (list of float)

S_G_calc()
Power absorbed/produced by grid voltage source.

S_PCC_calc()
Power output at PCC LV side

S_calc()
Inverter apparent power output

S_load1_calc()
Power absorbed by load at PCC LV side.

Vabrms_calc()
PCC LV side voltage - line to lineRMS

Vrms_calc()
PCC LV side voltage - RMS

Vtrms_calc()
Inverter terminal voltage -RMS

jac_ODE_model(*y*, *t*)
Jacobian for the system of ODE's. :param y: Initial conditions for the states.. :type y: list of float :param t: Simulation time in seconds. :type t: float

Returns An array containing the elements of the Jacobian.

Return type result (array of float)

update_RMS()
Update RMS voltages.

update_inverter_frequency(*t*)
Update inverter PLL frequency. :param t: Simulation time in seconds. :type t: float

update_inverter_states(*ia*, *xa*, *ua*, *Vdc*, *xDC*, *xQ*, *xPLL*, *wte*)
Update inverter states :param ia: Inverter phase a current. :type ia: complex :param xa: Inverter controller state. :type xa: complex :param ua: Inverter controller state. :type ua: complex :param Vdc: DC link voltage. :type Vdc: float

update_iref(*t*)
Update reference reference current.

update_power()
Update RMS voltages.

update_voltages()
Update voltages.

y0
List of initial states

1.1.3 pvder.DER_check_and_initialize

Code for initializing and validating PV-DER model instances.

class pvder.DER_check_and_initialize.**PVDER_SetupUtilities**
Bases: pvder.grid_components.BaseValues

Utility class for error checking during model initialization.

attach_grid_model(*DER_arguments*)
Attach a grid model to the PV-DER instance. :param grid_model: An instance of *GridModel*.

Returns Description of return value

Return type bool

check_DER_config(*DER_config*, *DER_id*)
Check DER config.

check_circuit_parameters()
Method to check whether inverter circuit parameter's are feasible.

check_config_file(*config_file*, *config_dict*)
Check whether DER config file contains necessary fields.

check_jacobian(*t=0.0*)
Compare analytical and numerical Jacobian of the ODE model.

check_voltage()
Method to check whether inverter voltage ratings are feasible. :raises: ValueError – If any of the specificied voltage ratings is infeasible.

circuit_parameters = {}

controller_gains = {}

creation_message()
Message after PV-DER instance was created.

initialize_DER_model()
Initialize DER ratings.

Parameters **DER_arguments** (*dict*) – Key word arguments.

Raises ValueError – If specified parameters correponding to *parameter_ID* is not available.

initialize_Iac()
Initialize AC side currents.

initialize_Sinverter()
Initialize inverter power rating.

initialize_Vac()
Initialize AC side voltages.

initialize_Vdc()
Initialize DC side voltages.

initialize_basic_options()
Initialize basic options

initialize_basic_specs()
Initialize number of ODEs and phases

initialize_circuit_parameters()
Initialize C, Lf, and Rf parameters.

initialize_controller_gains()
Initialize controller settings.

initialize_derived_quantities()
Initialize quantities other than states.

initialize_grid_measurements(*DER_arguments*)
Initialize inverter states. :param gridVoltagePhaseA: Value of gridVoltagePhaseA :type gridVoltagePhaseA: complex :param gridVoltagePhaseB: Value of gridVoltagePhaseB :type gridVoltagePhaseB: complex :param gridVoltagePhaseC: Value of gridVoltagePhaseC :type gridVoltagePhaseC: complex :param gridFrequency: Value of gridFrequency :type gridFrequency: float

initialize_inverter_ratings()
Initialize inverter voltage and power ratings.

initialize_jacobian()
Create a Jacobian matrix with zero values.

initialize_states(*DER_arguments*)
Initialize inverter states.

Parameters

- **ia0** (*float*) – Initial current
- **xa0** (*float*) – Initial controller state

- **ua0** (*float*) – Initial controller state

inverter_ratings = {}

modify_DER_parameters(*parameter_ID*)
Modify the DER parameters to parameters corresponding to the given parameter ID. :param parameter_ID: User specified parameter ID (can be None). :type parameter_ID: str

module_parameters = {}

power_error_calc(*x*)
Function for power.

solver_spec = {'SLSQP': {'disp': True, 'ftol': 1e-10, 'maxiter': 10000}, 'nelder-mead'

steady_state_calc()
Find duty cycle and inverter current that minimize steady state error and return steady state values.

steadystate_solver = 'SLSQP'

steadystate_values = {}

update_DER_config(*DER_config*, *DER_parent_config*, *DER_arguments*, *DER_id*, *DER_parent_id*)
Update PV-DER design.

update_DER_parameter(*DER_config*, *DER_parent_config*, *DER_arguments*, *DER_id*, *DER_parent_id*, *DER_component*, *DER_parameter*)
Update DER config using both config file and arguments.

1.1.4 pvder.DER_utilities

Code containing utilities used by PV-DER model instances.

class pvder.DER_utilities.**PVDER_ModelUtilities**
Bases: pvder.grid_components.BaseValues, pvder.utility_classes.Utilities

Utility class for single phase and three phase PV-DER model.

DO_EXTRA_CALCULATIONS = False

MPPT_ENABLE = False

MPP_table()
Method to output Vdc reference corresponding to MPP at different insolation levels values.

PRINT_INLINE = False

Qref_EXTERNAL = False

RAMP_ENABLE = False

RAMP_FLAG = False

S_PCCph_calc(*vph*, *iph*)
Inverter apparent power output - phase a/b/c

VERBOSE = False

Vdc_actual
Actual DC link voltage. :returns: DC link voltage in Volts. :rtype: float

Vdc_ref_counter = 0

Vdc_ref_list = []

Vdc_ref_ramp(*tstart*, *Vdc_ref_target*)
Create a ramp signal for voltage reference that ramps at 1 V/s. :param tstart: A scalar specifying start time of ramp in seconds. :type tstart: float :param Vdc_ref_target: A scalar specifying target Vdc reference seconds in volts. :type Vdc_ref_target: float

Vdc_ref_total = 0

add_Vdc_ref(*t*, *Vdc_ref*)
Add new solar event.

create_parameter_dict(*parameter_ID*)
Create a DER mode.

del_Vdc_ref = 2.0

del_t_Vdc_ref = 0.5

get_Vdc_ref(*t*)
Output Vdc reference.

get_default_parameter_ID()
Return default parameter ID.

get_iref_actual(*t*, *ia_ref_command*, *iaR_ref_previous*, *iaI_ref_previous*)
Update reference reference current.

get_parameter_dictionary(*parameter_type*, *parameter_ID*, *SHOW_DICTIONARY=True*)
Return parameter dictionary for specified parameter type and parameter ID. :param parameter_type: Specify type of parameter. :type parameter_type: str :param parameter_ID: Specify parameter ID or 'all'. :type parameter_ID: str :param SHOW_DICTIONARY: Print the dictionary. :type SHOW_DICTIONARY: bool

Returns Parameters and their values

Return type dict

get_ramp_limited_iref(*t*, *ia_ref_command*)
Update reference reference current.

ia_ref_activepower_control()
Phase A current reference for constant Vdc

ia_ref_calc()
Phase A current reference

ib_ref_activepower_control()
Phase B current reference for constant Vdc

ib_ref_calc()
Phase B current reference

ic_ref_activepower_control()
Phase C current reference for constant Vdc

ic_ref_calc()
Phase C current reference

initialize_parameter_dict(*parameter_ID*, *source_parameter_ID*)
Initialize a new parameter dictinary with values from an existing parameter dictionary.

iphload1_calc(*vph*)
Current counsumed by load connected at PCC LV side - Phase A/B/C.

load_parameter_dictionary(*file_name*)
Load parameter dictionary from saved file.

m_limit = 1.0

ma
Phase A duty cycle. :returns: Duty cycle. :rtype: complex

mb
Phase B duty cycle. :returns: Duty cycle. :rtype: complex

mc
Phase C duty cycle. :returns: Duty cycle. :rtype: complex

n_steps = 0

ramp_del_t = 0.5

ramp_list = []

reset_reference_counters()
Reset counter for reference change events.

save_parameter_dictionary(*parameter_ID*, *save_format='pickle'*, *SHOW_DICTIONARY=False*)
Save parameter dictionary.

set_Vdc_ref()
Return the correct Vdc reference voltage.

show_PV_DER_parameters(*parameter_type='inverter_ratings'*)
Display rated values. :param parameter_type: A string ('inverter_ratings','controller_gains','circuit_parameters') specifying the parameter to be displayed. :type parameter_type: str

show_PV_DER_states(*quantity='voltage'*)
Display values of states in the DER model quantities. Arguments

quantity: A string ('voltage','current','power','duty cycle') specifying the electrical quantity to be displayed.

show_parameter_dictionaries()
Show all parameter dictionary types and their ID's.

show_parameter_types()
Show all parameters within all parameter dictionary types.

show_references()
Print references.

update_Ppv(*t*)
Update PV module power output based on solar events and DC link voltage.

update_Qref(*t*)
Update reactive power set-point.

update_Vdc_ref(*t*)
Update DC link voltage reference.

update_Zload1(*t*)
Update load impedance at PCC-LV side.

update_parameter_dict(*parameter_ID*, *parameter_type*, *parameter_dict*)
Update parameters.

use_frequency_estimate = False

va_calc()
PCC - LV side - Phase A

validate_model(*PRINT_ERROR=True*)
Compare error between RMS quantities and Phasor quantities.

vb_calc()
PCC - LV side - Phase B

vc_calc()
PCC - LV side - Phase C

vta_calc()
Inverter terminal voltage - Phase A

vtb_calc()
Inverter terminal voltage - Phase B

vtc_calc()
Inverter terminal voltage - Phase C

we_calc()
Calculate inverter frequency from PLL.

wgrid_calc(*t*)
Frequency of grid voltage source.

wgrid_estimate(*t*)
Estimate frequency from phasor angle.

wgrid_estimate_calc(*t*, *phi_new*, *phi_previous*)
Estimate frequency.

1.1.5 pvder.DER_features

Code for features inside PV-DER model instances.

class pvder.DER_features.**PVDER_SmartFeatures**
Bases: object

Class for describing smart inverter inverter features of PV-DER.

DER_CONNECTED = True

DER_MOMENTARY_CESSATION = False

DER_TRIP = False

DER_disconnect()
Function to disconnect PV-DER from grid.

DER_disconnect_logic(*t*)
Logic for disconnecting/deenergizing DER from grid.

DER_reconnect_logic(*t*)
Logic used to decide reconnection.

FRT(*t*)
Frequency ride through and trip logic.

FRT_INSTANTANEOUS_TRIP

FRT_initialize()
Initialize LFRT and HFRT settings.

HVRT(*t*)
Function to implement HVRT ridethrough and trip logic.

LVRT(*t*)
Function to implement LVRT ridethrough and trip logic.

Q_Volt_VAR_absorb(*Vrms_measured*)
Calculate reactive power for Volt-VAR control.

Q_Volt_VAR_inject(*Vrms_measured*)
Calculate reactive power for Volt-VAR control.

Qlimit_calc()
Calculate maximum Q reference.

RT_initialize()
Initialize VRT and FRT settings.

VOLT_VAR_ENABLE = False

VOLT_VAR_FLAG = False

VOLT_WATT_ENABLE = False

VOLT_WATT_FLAG = False

VRT_initialize()
Initialize LVRT and HVRT settings.

V_threshold_high_limit = 1.5

Volt_VAR_absorb_range(*Vrms_measured*, *del_V=0.0*)
Check if voltage in volt-VAR operating range.

Volt_VAR_inject_range(*Vrms_measured*, *del_V=0.0*)
Check if voltage in volt-VAR operating range.

Volt_VAR_logic(*t*)
Volt-VAR.

check_HFRT_settings()
Sanity check for HFRT settings.

check_LFRT_settings()
Sanity check for LFRT settings.

check_RT_config()
Check whether the config file is good.

check_VRT_settings()
Sanity check for VRT settings.

check_anomaly()
Check if voltage anomaly was detected.

disconnect_or_reconnect(*t*)
Check flags and ither disconnect or reconnect DER.

f_ref = 60.0

get_Vrms_measured()
Get Vrms measurement

initialize_Volt_VAR()
Initialize the Volt-VAR controller settings.

print_LFRT_events(*simulation_time*, *frequency*, *timer_start=0.0*, *event_name=''*, *print_inline=False*, *verbose=False*)
Print LFRT events.

print_VRT_events(*simulation_time*, *voltage*, *zone_name*, *timer_start=0.0*, *V_threshold=None*, *t_threshold=None*, *t_min_ridethrough=None*, *LVRT_mode=None*, *event_name=''*, *print_inline=True*, *verbose=False*)
Print logs for VRT events.

print_event(*text_string*, *print_inline*)
Print information about ride through events.

print_reconnect_events(*simulation_time*, *voltage*, *frequency*, *timer_start=0.0*, *event_name=''*, *print_inline=True*, *verbose=False*)
Print logs for VRT events.

show_RT_flags()
Show all RT flags.

show_RT_settings(*settings_type='LVRT'*, *PER_UNIT=True*)
Method to show LVRT settings.

t_disconnect_low_limit = 0.008333333333333333

t_reconnect_low_limit = 0.1

t_threshold_high_limit = 100.0

update_RT_config(*config_dict*)
Check whether the config file is good.

update_RT_config_old(*derConfig*)
Check whether the config file is good.

update_ridethrough_flags(*t*)
Check VRT and FRT logic.

1.2 DER Simulation

1.2.1 pvder.dynamic_simulation

Code for setting up, and running, and collecting data from PV-DER simulations.

class pvder.dynamic_simulation.**DynamicSimulation**(*PV_model*, *events*, *gridModel=None*, *tStop=0.5*, *LOOP_MODE=False*, *COLLECT_SOLUTION=True*, *jacFlag=False*, *verbosity='INFO'*, *solverType='odeint'*, *identifier=None*)
Bases: pvder.grid_components.Grid, *pvder.simulation_utilities.SimulationUtilities*, pvder.utility_classes.Utilities

Utility class for running simulations.

Creates an instance of *GridSimulation*. :param PV_model: An instance of *SolarPV_DER*. :param events: An instance of *SimulationEvents*. :param grid_model: An instance of *GridModel* (only need to be suppled in stand alone simulation). :param tStop: A scalar specifying the end time for simulation. :param tInc: A scalar specifying the time step for simulation. :param LOOP_MODE: A boolean specifying whether simulation is run in loop.

DEBUG_CONTROLLERS = False

DEBUG_CURRENTS = False

DEBUG_PLL = False

DEBUG_POWER = False

DEBUG_SIMULATION = False

DEBUG_SOLVER = False

DEBUG_VOLTAGES = False

ODE_model(*y*, *t*)
Combine all derivatives.

check_jac_availability()
Check if Jacobian matrix is available.

collect_full_trajectory(*solution*)
Collect full solution from solver.

collect_last_states()
Collect states at last time step.

collect_solution(*solution*, *t=None*)
Collect solution.

collect_states(*solution*)
Collect states from ode solution.

count = 0

debug_simulation(*t*)
Print to terminal for debugging.

get_trajectories()
Return trajectories as a dictionary.

initialize_y0_t()
Initialize y0_t.

invert_arrays()
Inverter arrays before usage by plots.

jac_ODE_model(*y*, *t*)
Combine all derivatives.

jac_list = ['SolarPVDERThreePhase', 'SolarPVDERSinglePhase', 'SolarPVDERThreePhaseBala

reset_stored_trajectories()
Reset for plotting.

run_simulation(*gridVoltagePhaseA=None*, *gridVoltagePhaseB=None*, *gridVoltagePhaseC=None*, *y0=None*, *t=None*)
Call the ODE solver and collect states.

show_simulation_time()
Show simulation time.

tInc = 0.001

tStart = 0.0

t_calc()
Vector of time steps for simulation

time_series_PCC_HV_side_voltage()
Calculate time series PCC voltage.

time_series_PCC_LV_side_voltage()
Calculate time series PCC voltage.

time_series_PLL()
Calculate time series PLL and d-q quantities.

time_series_Ppv()
Calculate time series Solar PV power output.

time_series_RMS()
Calculate time series RMS quantities.

time_series_S()
Calculate time series apparent power.

time_series_Zload1(*tOverride=None*)
Calculate time series load impedance.

time_series_duty_cycle()
Calculate time series PCC voltage.

time_series_inv_terminal_voltage()
Calculate time series inverter terminal voltage.

time_series_phase_angle()
Calculate time series phase angle between grid and inverter voltage.

time_series_power_transfer()
Calculate time series power transfer between grid and PCC.

time_series_standalone_grid()
Time series grid voltage and frequency for standalone model.

update_grid_measurements(*gridVoltagePhaseA*, *gridVoltagePhaseB*, *gridVoltagePhaseC*)
Update grid voltage and frequency in non-standalone model. :param gridVoltagePhaseA: Value of gridVoltagePhaseA :type gridVoltagePhaseA: complex :param gridVoltagePhaseB: Value of gridVoltagePhaseB :type gridVoltagePhaseB: complex :param gridVoltagePhaseC: Value of gridVoltagePhaseC :type gridVoltagePhaseC: complex

y0
Combine all initial conditions from solution.

1.2.2 pvder.simulation_events

Manage simulation events.

class pvder.simulation_events.**SimulationEvents**(*events_spec=None*, *SOLAR_EVENT_ENABLE=True*, *GRID_EVENT_ENABLE=True*, *LOAD_EVENT_ENABLE=True*, *verbosity='INFO'*, *identifier=''*)

Bases: pvder.utility_classes.Utilities

Utility class for events.

Creates an instance of *SimulationEvents*. :param SOLAR_EVENT_ENABLE: A boolean to enable solar insolation events. :param GRID_EVENT_ENABLE: A boolean to enable grid voltage or frequency events. :param LOAD_EVENT_ENABLE: A boolean to enable load change events at PCC-LV side.

Tactual_default = 298.15

Zload1_actual_default = (10000000+0j)

add_grid_event(*T*, *Vgrid=1.0*, *Vgrid_angle=0.0*, *fgrid=60.0*)
Add new grid event. :param T: A scalar specifying start time of grid event in seconds. :type T: float :param Vgrid: A scalar specifying grid voltage magnitude in fraction. :type Vgrid: float :param Vgrid_angle: A scalar specifying grid voltage angle in radians. :type Vgrid_angle: float :param fgrid: A scalar specifying grid frequency in Hz. :type fgrid: float

add_load_event(*T*, *Zload1_actual=(10000000+0j)*)
Add new load event. :param T: A scalar specifying start time of load event in seconds. :param Zload1_actual: A complex scalar specifying load in ohm.

add_solar_event(*T*, *Sinsol=100.0*, *Tactual=298.15*)
Add new solar event. :param T: A scalar specifying start time of solar event in seconds. :type T: float :param Sinsol: A scalar specifying solar insolation in percentage. :type Sinsol: float :param Tactual: A scalar specifying module temperature in Kelvin. :type Tactual: float

count = 0

create_random_events(*t_event_start, t_event_end, t_event_step, events_type=['insolation', 'voltage']*)
Create random events of specified types.

create_random_insolation_events(*t_event*)
Create random voltage event at specified time.

create_random_voltage_events(*t_event*)
Create random voltage event at specified time.

del_t_event = 0.001

grid_events(*t*)
Generate grid event during simulation at specified time. :param t: A scalar specifying the time (s). :type t: float

insolation_ramp(*tstart*, *tstop*, *Sinsol_target*, *tstep=0.25*)
Create a ramp signal. :param tstart: A scalar specifying start time of solar insolation event in seconds. :param tstop: A scalar specifying stop time of solar insolation event in seconds. :param Sinsol_target: A scalar specifying target solar insolation at 'tstop' in percentage. :param tstep: A scalar specifying time step size.

load_events(*t*)
Generate load event at PCC LV side during simulation. :param t: A scalar specifying the time (s). :type t: float

override_angle = True

remove_grid_event(*T*)
Remove grid event at 'T'.

remove_load_event(*T=None*, *REMOVE_ALL=False*)
Remove solar event at 'T'

remove_solar_event(*T=None*, *REMOVE_ALL=False*)
Remove solar event at 'T'.

reset_event_counters()
Reset event counts.

show_events()
Print all the simulation events.

simulation_events_list
List of all simulation events.

solar_events(*t*)
Generate solar event during simulation at specified time. :param t: A scalar specifying the time (s). :type t: float

update_event_totals()
Update event counts.

update_events_spec(*events_spec*)
Update the _events_spec dictionary.

voltage_ramp(*tstart*, *tstop*, *vg_target*, *tstep=0.5*)
Create a ramp signal for grid voltage. :param tstart: A scalar specifying start time of grid voltage event in seconds. :param tstop: A scalar specifying stop time of grid voltage event in seconds. :param vg_target: A scalar specifying target grid voltage at 'tstop' in fraction. :param tstep: A scalar specifying time step size (optional).

1.2.3 pvder.simulation_utilities

Manage simulation results and ODE solver.

class pvder.simulation_utilities.**SimulationResults**(*simulation*, *figure_index=1*, *PER_UNIT=True*, *font_size=18*, *PLOT_TITLE=True*, *SOLUTION_TIME=True*, *verbosity='INFO'*, *identifier=None*)

Bases: pvder.utility_classes.Utilities

Utility class for simulation results.

Creates an instance of *SimulationResults*. Args: simulation: An instance of *GridSimulation*. figure_index: An integer specifying the figure index. font_size: An integer spcifying the font size to be used withing plots. PLOT_TITLE: A boolean specifying whether the title will be displayed in plots. verbosity: A string specifying the verbosity level (DEBUG,INFO,WARNING,ERROR).

SAVE_PLOT_JPEG = False

SAVE_PLOT_SVG = False

available_plot_types = ['power', 'active_power', 'active_power_Ppv_Pac_PCC', 'active_p

change_units()
Change units from per unit to S.I. or vice versa.

compare_with_external(*external_time_values*, *external_plot_values*, *external_plot_legends*, *plot_type='power'*)
Compare with external simulation.

count = 0

figure_DPI = 1200

group_quantities_for_plotting(*plot_type*)
Plot power from simulation. Args: plot_type (string): The quantities to be plotted.

Raises ValueError – If *plot_type* is not available.

parameters = {'figure': {'height': 8, 'width': 8}}

plot_DER_simulation(*plot_type='power'*)
Plot desired electrical quantity from simulation. Args: plot_type (str): Specify type of plot (to see available plot types use SimulationResults.available_plot_types).

plot_multiple(*time_values*, *plot_values*, *legends*, *plot_title*, *y_labels*)
Function to plot multiple time series on same plot.

save_plot(*plot_object*, *plot_name='results'*)
Save the plots.

class pvder.simulation_utilities.**SimulationUtilities**
Bases: object

Utility class for dynamic simulations.

ODE_model_ode(*t*, *y*)
" Combine all derivatives when using ode method

call_ODE_solver(*derivatives*, *jacobian*, *y*, *t*)
Call the SciPy ODE solver.

call_ode_solver()
Use the SciPy ode solver.

call_odeint_solver(*derivatives*, *jacobian*, *y*, *t*)
Use the SciPy odeint solver.

check_simulation(*infodict*, *t*)
Check whether the ODE solver failed at any time step.

initialize_solver(*solver_type*, *t0=0.0*)
Initialize an integrator.

jac_ODE_model_ode(*t*, *y*)
Combine all derivatives.

max_steps = 1000

solver_list = ['odeint', 'ode-vode-bdf']

CHAPTER 2

License

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index

D

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

Y

Z