Sphinx format for Latex and HTML

Meher Krishna Patel

Created on : Octorber, 2017

Last updated : October, 2020

More documents are freely available at PythonDSP

Table of contents

List of figures

List of tables

1 GOCPI package

1.1 Submodules

1.2 GOCPI.CreateCases module

class GOCPI.CreateCases.CreateCases
Bases: object

A class of methods to create user-defined data cases

set_accumulated_annual_demand(*accumulated_forecast*)

Sets the accumulated annual demand for fuels per region over the forecast period.
This function relies on a similar forecasting methodology as set_specific_demand.
Fuels set in this function cannot be defined in set_specific_demand.

Parameters accumulated_forecast (*float , array*) – The forecast array of size (len(region),len(fuel),len(year))

set_accumulated_fuel(*accumulated_fuel*)
Sets the case's accumulated fuel types

Parameters specified_fuel (*list*) – list of specified fuels

set_annual_emission_limit(*annual_emission_limits*)
Sets Annual Emission Limits

Parameters annual_emission_limits (*float , array*) – Annual Emission Limits

set_annual_exogenous_emission(*annual_exogenous_emission*)
Sets Annual Exogneous Emissions

Parameters annual_exogenous_emission (*float , array*) – Annual Exogenous Emissions

set_availability_factor(*availablility_matrix*)
Sets the availability factors

Parameters availablility_matrix (*float , array*) – Matrix describing availability factors for given technologies

set_availability_technology(*availability_technology*)
Sets the cases availability_technology type

Parameters availability_technology (*list*) – List of technologies

set_capacity_factor(*factor_matrix*)
Sets capacity factors for conversion technologies.

Parameters `factor_matrix` (*float , array*) –

`set_capacity_of_one_technology_unit`(*capacity_ of_ one_ technology_ unit*)
Set the capacity of one technology units for all technologies

Parameters `capacity_of_one_technology_unit` (*float , array*) – capacities for one technology units

`set_capacity_technology`(*capacity_ technology*)
Sets the cases capacity_technology type

Parameters `capacity_technology` (*list*) – List of technologies

`set_capacity_to_activity_unit`(*region*, *technology*, *capacity_ dictionaries*, *override*)
Sets the capacity to activity parameter

Parameters
- `region` (*list*) – List of regions
- `technology` (*list*) – List of technologies
- `capacity_dictionaries` (*list*) – List of dictionaries to assign value
- `override` (*float , array*) –

`set_capital_cost`(*capital_ costs*)
Sets capital costs

Parameters `capital_costs` (*float , array*) – capital cost paramters

`set_capital_cost_storage`(*capital_ cost_ storage*)
Sets the capital costs of using storage technologies

Parameters `capital_cost_storage` (*float , array*) – capital cost of storage technologies

`set_conversion_ld`(*timeslice*, *daytype*, *link*)
Sets the Conversionld parameter

Parameters
- `timeslice` (*list*) – List of timeslices
- `daytype` (*list*) – List of daytypes
- `link` (*dict*) – Dictionary describing the connection between timeslices and daytypes

`set_conversion_lh`(*timeslice*, *dailytimebracket*, *link*, *override*)
Sets the Conversionlh parameter

Parameters
- `timeslice` (*list*) – List of timeslices
- `dailytimebracket` (*list*) – List of dailytimebracket
- `link` (*dict*) – Dictionary describing the connection between timeslices and dailytimebrackets
- `override` (*int , array*) – Override if want to manually put in the array

`set_conversion_ls`(*timeslice*, *season*, *link*)
Sets the Conversionls parameter

Parameters

- `timeslice` (*list*) – List of timeslices
- `season` (*list*) – List of seasons
- `link` (*dict*) – Dictionary describing the connection between timeslices and seasons

`set_daily_time_bracket(`*num_dailytimebrackets*`)`

Creates set of daily time brackets

Parameters `dailytimebracket` (*int*) – [description]

`set_day_split(`*daily_time_bracket*, *years*, *hour_split*, *num_days*, *num_hours*`)`

Sets the day split parameter

Parameters

- `daily_time_bracket` (*list*) – List of daily time brackets
- `years` (*list*) – List of year
- `hour_split` (*dict*) – Dictonary of hours in a daily time bracket
- `num_days` (*int*) – Number of days in a year
- `num_hours` (*int*) – Number of hours in a day

`set_days_in_day_type(`*season*, *daytype*, *year*, *link*, *override*`)`

Sets the DaysInDayType parameter

Parameters

- `season` (*list*) – List of seasons
- `daytype` (*list*) – List of daytypes
- `year` (*list*) – List of years
- `link` (*dict*) – Dictionary relating seasons to daytypes
- `override` (*int , array*) – Override if want to manually put in the array

`set_daytype(`*num_daytypes*`)`

[summary]

Parameters `num_daytypes` (*int*) – Number of daytypes

`set_depreciation_method(`*region*, *methods*, *override*`)`

Sets DepreciationMethod (1 = Sinking Fund Depreciation, 2 = Straightline Depreciation)

Parameters

- `region` (*list*) – List of regions
- `override` (*int , array*) – Manual array for setting depreciation methods
- `methods` (*dict*) – Dictionary assigning methods to regions

`set_discount_rate(`*equity*, *debt*, *market_index*, *cost_of_debt_pre_tax*, *risk_free_rate*, *effective_tax_rate*, *preference_equity*, *market_value_preference_shares*, *preference_dividends*, *market_risk_coefficient*`)`

[summary]

Parameters

- `equity` (*dict*) – Dictionary of equity totals from treasury balance sheets
- `debt` (*dict*) – Dictionary of equity totals from treasury balance sheets
- `market_index` (*int , array*) – Regional monthly index returns (Arrays)
- `cost_of_debt_pre_tax` (*dict*) – Dictionary of pre-tax cost of debts calculated from treasury balance sheets
- `risk_free_rate` (*dict*) – Dictionary of risk free rates from 10 year swap rates for each region
- `effective_tax_rate` (*dict*) – Dictionary of company tax rates for each region
- `preference_equity` (*dict*) – Dictionary of preference equity for each region
- `market_value_preference_shares` (*dict*) – Dictionary of the market value of prefence shares for each region
- `preference_dividends` (*dict*) – Dictionary of prefence dividends for each region
- `market_risk_coefficient` (*dict*) – Dictionary of markey risk coefficients

Returns Numpy array of discount rates

Return type [int, array]

`set_emission(`*emissions*`)`

Sets the cases emission types

Parameters `emissions` (*List*) – list of emission types

`set_emission_activity_ratio(`*emission_activity_ratios*`)`

Sets Emission Activity Ratios

Parameters `emission_activity_ratios` (*[float , array*) – Emission Activity Ratios

`set_emissions_penalty(`*emissions_penalties*`)`

Sets Emissions Penalties

Parameters `emissions_penalties` (*float , penalties*) – Emissions Penalties

`set_fixed_cost(`*fixed_costs*`)`

Set fixed costs

Parameters `fixed_costs` (*float , array*) – fixed cost parameters

`set_fuel(`*fuel*`)`

Sets the case's fuel types

Parameters `fuel` (*list*) – list of fuels

`set_input_activity_ratio(`*input_activity_ratios*`)`

Sets input activity ratios

Parameters `input_activity_ratios` (*float , array*) – Sets the input activity ratio

`set_min_storage_charge(`*minimum_storage_charges*`)`
Sets the minimum storage charges

Parameters `minimum_storage_charges` (*float , array*) – minimum storage parameters

`set_mode_of_operation(`*num_modes_of_operation*`)`
Create the number of modes of operation (n = 1,. . . ,num_modes_of_operation)

Parameters `num_modes_of_operation` (*int*) –

`set_model_period_emission_limit(`*model_period_emission_limits*`)`
Sets Model Period Emission Limits

Parameters `model_period_emission_limits` (*float , array*) – Model Period Emission Limits

`set_model_period_exogenous_emission(`*model_period_exogenous_emissions*`)`
Sets Model Period Exogenous Emissions

Parameters `model_period_exogenous_emissions` (*float , array*) – Model Period Exogenous Emissions

`set_operational_life(`*operational_lives*`)`
Sets operational life

Parameters `operational_lives` (*list*) –

`set_operational_life_storage(`*operational_life_storage*`)`
Sets the operational life storage

Parameters `operational_life_storage` (*float , array*) – operational life storage parameters

`set_output_activity_ratio(`*output_activity_ratios*`)`
Sets output activity ratio

Parameters `output_activity_ratios` (*float , array*) – output activity ratio parameters

`set_re_min_production_target(`*re_min_production_targets*`)`
Sets Renewable Energy Minimum Production Targets

Parameters `re_min_production_targets` (*float , array*) – Renewable Energy Minimum Production Targets

`set_re_tag_fuel(`*re_tag_fuels*`)`
Sets RE Tag Fuels

Parameters `re_tag_fuels` (*float , array*) – RE Tag Fuels

`set_re_tag_technology(`*re_tag_technologies*`)`
Sets RE Tag Technology

Parameters `re_tag_technologies` (*float , array*) – RE Tag Technologies

set_region(*regions*)
Sets the datacase's regions analysis

Parameters regions (*list*) – list of regions

set_reserve_margin(*reserve_margins*)
Sets reserve margins

Parameters reserve_margins (*float, array*) – Reserve Margins

set_reserve_margin_tag_fuel(*reserve_margin_fuel_tags*)
Sets the reserve margin tag fuels

Parameters reserve_margin_fuel_tags (*float, array*) – Sets the reserve margin tag fuel parameters

set_reserve_margin_tag_technology(*reserve_margin_tag_technologies*)
Sets Reserve Margin Tag Technology

Parameters reserve_margin_tag_technologies (*float, array*) – Reserve Margin Tag Technologies

set_residual_capacity(*residential_capacities*)
Set residual capacity

Parameters residential_capacities (*float, array*) – residual capacities parameter

set_residual_storage_capacity(*residual_storage_capacities*)
Sets residual storage capacities

Parameters residual_storage_capacities (*float, array*) – residual storage capacities

set_season(*num_seasons*)
Creates set of seasons

Parameters num_seasons (*int*) – Number of seasons

set_specified_annual_demand(*specified_forecast*)
Sets the annual demand for fuels per region over the forecast period (Must be accurate)

Parameters forecast (*float, array*) – The forecast array of size (len(region),len(fuel),len(year))

set_specified_demand_profile(*specified_annual_demand*, *region*, *fuel*, *year*, *timeslice*, *profile*, *override*)
Sets the specified annual demand profiles using the specified annual demand.

Parameters

- specified_annual_demand (*float, array*) – Specified annual demand profiles
- region (*list*) – List of regions
- fuel (*list*) – List of fuels
- year (*list*) – List of years
- timeslice (*list*) – List of timeslices
- profile (*Dict*) – Dictionary of fuel allocations to timeslices

- `override` (*float , array*) – Manual override for the specified annual demand profiles.

`set_specified_fuel(`*specified_fuel*`)`
Sets the case's specified fuel types

Parameters `specified_fuel` (*list*) – list of specified fuels

`set_storage(`*storage*`)`
Sets storage set of the datacase

Parameters `storage` (*list*) – list of storage types

`set_storage_level_start(`*storage_level_start*`)`
Sets the storage level starting point

Parameters `storage_level_start` (*float , array*) – storage starting level

`set_storage_max_charge_rate(`*storage_max_level_charge_rates*`)`
Sets the storgae max charge rate

Parameters `storage_max_level_charge_rates` (*float , array*) – Storage max level charge rates

`set_storage_max_discharge_rate(`*storage_max_level_discharge_rates*`)`
Sets storage technologies maximum discharge rates

Parameters `storage_max_level_discharge_rates` (*float , array*) – Discharge rates for storage paramters

`set_technology(`*technology*`)`
Sets the cases technology type

Parameters `technology` (*list*) – List of technologies

`set_technology_from_storage(`*technology_from_storage*`)`
Sets technology from storage binary paramter

Parameters `technology_from_storage` (*float , array*) – technology from storage parameter

`set_technology_to_storage(`*technology_to_storage*`)`
Sets the technology to storage parameter

Parameters `technology_to_storage` (*float , array*) – technology to storage parameter

`set_timeslice(`*timeslice*`)`
Set of timeslices

Parameters `timeslice` (*list*) – list of timeslices

`set_total_annual_max_capacity(`*total_annual_max_capacities*`)`
Sets the total annual maximum capacities

Parameters `total_annual_max_capacities` (*float , array*) – Total Annual Max Capacities

set_total_annual_min_capacity(*total_annual_min_capacities*)
Sets the totoal annual minimum capacities

Parameters total_annual_min_capacities (*float , array*) – Total Annual Min Capacities

set_total_technology_annual_activity_lower_limit(*total_technology_activity_lower_limits*)
Sets the Total Technology Activity Lower Limits

Parameters total_technology_activity_lower_limits (*float , array*) – Technology Activity Lower Limits

set_total_technology_annual_activity_upper_limit(*total_technology_annual_activity_upper_*
Sets the Total Technology Activity Upper Limits

Parameters total_technology_annual_activity_upper_limits (*float , array*) – Technology Activity Upper Limits

set_total_technology_period_activity_lower_limit(*total_technology_period_activity_lower_*
Sets Total Technology Period Activity Lower Limits

Parameters total_technology_period_activity_lower_limits (*[type]*) – Total Technology Period Activity Lower Limit

set_total_technology_period_activity_upper_limit(*total_technology_period_activity_upper_*
Sets Total Technology Period Activity Upper Limits

Parameters total_technology_period_activity_upper_limits (*float , array*) – Total Technology Period Activity Upper Limit

set_trade_route(*trade*)

Sets the TradeRoute parameter between regions (Assume it is the same across fuels and years)

Parameters trade (*int , array*) – 4D array representing trade relationships between regions, fuels and years. You must model this manually.

set_variable_cost(*variable_costs*)
Sets variable costs

Parameters variable_costs (*float , array*) – variable costs parameters

set_year(*start_year*, *end_year*, *interval*)
Sets a list of forecast years

Parameters

- start_year (*int*) – Starting year for forecasting (Less than end_year)
- end_year (*int*) – Ending year for forecasting (Greater than start_year)
- interval (*int*) – Gap for forecasting period

set_year_split(*timeslices*, *years*, *splits*)

Creates 2D Numpy Array Parameter Splits. (Note: The index positions of timelices and splits must match)

Parameters

- `timeslices` (*list*) – List of timeslices
- `years` (*list*) – List of years
- `splits` (*dict*) – A dictionary linking yearsplits to timeslices

1.3 GOCPI.Energysystems module

`class GOCPI.Energysystems.Energy_Systems(`*year*, *region*, *emission*, *technology*, *capacity_technology*, *availability_technology*, *fuel*, *specified_fuel*, *accumulated_fuel*, *timeslice*, *mode_of_operation*, *storage*, *daytype*, *season*, *dailytimebracket*`)`

Bases: `object`

A class of methods to initialise energy sytems and create the data/model files needed for optimisation.

`create_data_file(`*file_location*, *defaults_dictionary*, *toggle_defaults*`)`

Creates the osemosys datafile

Parameters

- `file_location` (*str*) – String of directory to save data file
- `defaults_dictionary` (*dict*) – Dictionary setting the default values for parameters
- `toggle_defaults` (*Bool*) – Boolean (True/False to only print the default functions

`create_model_file(`*root*, *file*`)`

Creates the model file necessary for the project to run

Parameters `for the basic problem` (*Parameters*) –
Returns The loaded in parameters and sets

`load_datacase(`*case*, *system*`)`

Loads the data case to a correct configured and intialised energy system

(The load status dictionary must be compatible with the data_case and system_case)

Parameters

- `case` (*object*) – Energy system datacase
- `system` (*object*) – Initialised energy system
- `load_status` (*dict*) – Dictionary setting the required sets and parameters to load

Returns Returns the updated dictionary
Return type system_case (dict)

1.4 GOCPI.Forecasting module

class GOCPI.Forecasting.Forecasting

Bases: object

calculate_cagr_forecasts(*cagr_dictionary*, *base_year_dictionary*, *fuel*, *year*)

Forecasts base year fuels by a constant average growth rate for a forecast period

Parameters

- cagr_dictionary (*Dict*) – Dictionary of constant average growth rates per fuel
- base_year_dictionary (*[type]*) – Dictionary of base year fuel consumption in energy types
- fuel (*list*) – List of Fuels
- year (*list*) – List of forecast years

Returns 2D Array of demand forecasts per fuel

Return type [float, array]

calculate_constant_average_growth_rate(*start_year*, *end_year*, *start_value*, *end_value*)

Calculates the constant average growth rate (CAGR)

Parameters

- start_year (*int*) – Starting year
- end_year (*int*) – Ending year
- start_value (*int*) – Initial value
- end_value (*int*) – Final value

Returns Constant average growth rate (1+ decimal)

Return type cagr

energy_balance_base(*root*, *IEA_World_Energy_Balances_1*, *IEA_World_Energy_Balances_2*, *create_excel_spreadsheet*, *output_file*)

Creates the baseline energy balance for forecasting

Parameters

- root (*path*) – Path to provide access to all the files
- IEA_World_Energy_Balances_1 (*str*) – File name for Energy Balance A to K
- IEA_World_Energy_Balances_2 (*[type]*) – File name for Energy Balance L to Z
- create_excel_spreadsheet (*boolean*) – True/false on whether to create a spreadsheet
- output_file (*str*) – Name of output energy balance spreadsheet

Returns Dictionary of energy balances and unique lists (Use these key words to access: Energy Balances, Fuel, Geography, Technology)

Return type (dict)

1.5 GOCPI.Navigation module

class GOCPI.Navigation.Navigation(*target_root*, *target_file*)

Bases: object

Navigation is a class for navigating, manipulating and editing data in the GOCPI model.

Find_File

Type string

TODO: Fill out all functions below

Find_File()

Find_File searches for a target file, from a base directory, to construct a target directory.

Inputs: target_root = The base directory to search from (string). target_file = The name of the target file (string).

Outputs: f = Combinated target file location (string).

create_linear_programme_file(*directory*, *data_file*, *model_file*, *output_file*)

Creates the model file through executing model system commands

Parameters

- directory (*str*) – Name of directory to put data into
- data_file (*str*) – Name of energy system data file
- model_file (*str*) – Name of energy system model file
- output_file (*str*) – Name of output linear programme

1.6 GOCPI.Optimisation module

class GOCPI.Optimisation.Optimisation

Bases: object

Prepare and runs optimisation with IBM ILOG CPLEX Optimisation Studio

create_linear_programme_file(*directory*, *data_file*, *model_file*, *output_file*)

Creates the model file through executing model system commands

Parameters

- directory (*str*) – Name of directory to put data into
- data_file (*str*) – Name of energy system data file
- model_file (*str*) – Name of energy system model file
- output_file (*str*) – Name of output linear programme

reset(*tarinfo*)

Resets the tarfile information when creating tar files This is to input into the filter when using tar.add()

Parameters tarinfo (*Object*) – Tar Object containing an ID of 0 and the root as the name

Returns Tar Object containing an ID of 0 and the root as the name
Return type tarinfo (Object)

`run_cplex_local(`*model_file*`)`
This function runs cplex on the local device if the energy system is of a small enough complexity

`run_ibm_wml_do(`*apikey*, *url*, *deployment_space_name*, *cloud_object_storage_credential*, *service_instance_id*, *deployment_space_exists*, *data_assets_exist*, *data_asset_dictionary*, *model_name*, *model_type*, *model_runtime_uid*, *model_tar_file*, *num_nodes*, *deployment_exists*, *payload_input_data_id*, *payload_input_data_file*, *payload_output_data_id*`)`

This function enables the user to solve python-based optimisation models. The legacy offering to solve optimisation models on IBM cloud was using the docplex python api to run Cplex on DOcloud. As of September 2020, the DOcloud was discontinued with Decision Optimisation functionalities imported to IBM's Watson Machine Learning Service. The new process requires the energy system model to be written in python. This project saw the implementation of the osemosys modelling methodology in GNU Mathprog written into LP Files. IBM Decision Optimisation in cannot deploy models in LP File formats to get jobs. Therefore, this function is for future work in converting the entire energy system modelling tool to python-based only. This is well-documented the report in the Future Work Section. Note: You must have access to IBM Watson Studio and Cloud Products through the IBM Academic Initiative or Similar.

Parameters

- `apikey` (*str*) – API key from user's IBM Cloud Account
- `url` (*[type]*) – URL for the server the user is using for the IBM services
- `deployment_space_name` (*str*) – Name of the deployment space
- `cloud_object_storage_credential` (*str*) – Credential for the cloud object storage asset
- `service_instance_id` (*str*) – Service instance id for the service being used (IBM WML)
- `deployment_space_exists` (*boolean*) – True/False if the deployment space already exists
- `data_assets_exist` (*boolean*) – True/False if the data assets (e.g. input data stored on cloud)
- `data_asset_dictionary` (*dict*) – A dictionary of data assets to stored on IBM cloud
- `model_name` (*str*) – Name of the model
- `model_type` (*str*) – Name of the model
- `model_runtime_uid` (*str*) – Runtime ID for the model
- `model_tar_file` (*tar*) – Tar file containing the python model
- `num_nodes` (*int*) – Number of nodes the model is run off.
- `deployment_exists` (*boolean*) – True/False if the deployment already exists
- `payload_input_data_id` (*str*) – Name of input data

- `payload_input_data_file` (*dataframe*) – Input data file in the form of a dataframe
- `payload_output_data_id` (*str*) – Name of output data file

`use_bash_shell(`*command*`)`

Execute bash commands in python scripts

Parameters `command` (*str*) – Command to execute

1.7 Module contents

Indices and tables

- genindex
- modindex
- search

Python Module Index

g

Index

U