optimeed

Release 2.0.2

Aug 07, 2022

Contents

Optimeed is a free open source package that allows to perform optimization and data visualization/management.

CHAPTER 1

Requirements

- PyQt5 for visualisation -> `pip install PyQt5`
- *pyopengl* for visualisation -> `pip install PyOpenGL`
- Numpy -> `pip install numpy`
- **Optional**
  - pandas which is only used to export excel files -> `pip install pandas`
  - `nlopt` library for using other types of algorithm. -> `pip install nlopt`
  - inkscape software for exporting graphs in .png and .pdf)

CHAPTER 2

Installation

To install the latest optimeed release, run the following command:

```
pip install optimeed
```

To install the latest development version of optimeed, run the following commands:

```
git clone https://git.immc.ucl.ac.be/chdegreef/optimeed.git
cd optimeed
python setup.py install
```

CHAPTER 3

Quickstart

Examples can be found on the tutorial folder .

3.1 Quickstart Optimization

An optimization process can be presented as following:

- **Optimization algorithm:** *AlgorithmInterface*. This is the algorithm that performs the optimization, and outputs a vector of variables between [0, 1[.
- **Maths to physics:** *InterfaceMathsToPhysics*. Transforms the output vector of the optimization algorithm to the variables of a `InterfaceDevice`. The usage of this block becomes meaningful for more complex optimization problem, such as optimizing a BLDC motor while keeping the outer diameter constant. In this case, a good implementation of the M2P block automatically scales the inner dimensions of the motor to comply with this constraint.
- **Characterization:** *InterfaceCharacterization*. Based on the attributes of the device, performs some computation. This block is nearly useless for simple optimization problems (when the objective function is easily computed) but becomes interesting for more complex problems, where many things need to be precalculated before obtaining the objective functions and constraints. This for example can hold an analytical or a FEM magnetic model. A sub-optimization could also be performed there.
- **Objective and constraints:** *InterfaceObjCons*. These classes correspond to either what has to be minimized, or which constraints <=0 has to be complied with.

Quick example: $\min_{x,y\in[0,2]} f(x) = \sqrt{1+(y+3)\cdot x^2}, g(x) = 4+2\sqrt{y+3}\cdot\sqrt{1+(x-1)^2}$, under the constrained that $x \leq 0.55$. This is a bi-objective problem and will lead to a pareto front.

```
"""Now that you know everything about data and visualization, let's get started with␣
↪optimization!
Optimeed provides high-level interface to perform optimization with visualization and␣
↪data storage.
The Wiki gives more details about the optimization. To get started, you need the␣
↪following key ingredients:
```

(continues on next page)

(continued from previous page)

```
    - A device that contains the variables to be optimized ("Device") and other␣
↪parameters you would like to save
    - A list of optimization variables ("OptimizationVariable")
    - An evaluation function ("Characterization")
    - One or more objective functions ("Objectives")
    - (optional) Eventual constraints ("Constraints")
    - An optimization algorithm ("Optimization Algorithm")
    - Something that will fill the "Device" object with the optimization variables␣
↪coming from the optimization algorithm. ("MathsToPhysics")
        Don't get scared with this one, if you do not know how it can be useful, the␣
↪proposition by default works perfectly fine.
    - Something that will link all the blocks together ("Optimizer")
"""


# These are what we need for the optimization
from optimeed.optimize.optiAlgorithms import MultiObjective_GA as␣
↪OptimizationAlgorithm
from optimeed.optimize import Real_OptimizationVariable, InterfaceObjCons,␣
↪InterfaceCharacterization, OptiHistoric
from optimeed.optimize.optimizer import OptimizerSettings, run_optimization

# These are the high-level visualization tools
from optimeed.visualize.displayOptimization import OptimizationDisplayer
from optimeed.visualize import Onclick_representDevice, Represent_brut_attributes,␣
↪start_qt_mainloop
import time


class Device:
    """Define the Device to optimize."""
    x: float  # Type hinted -> will be automatically saved
    y: float  # Type hinted -> will be automatically saved

    def __init__(self):
        self.x = 1
        self.y = 1


class Characterization(InterfaceCharacterization):
    """Define the Characterization scheme. In this case nothing is performed,
     but this is typically where model code will be executed and results saved inside
↪'theDevice'."""
    def compute(self, thedevice):
        time.sleep(0.0001)


class MyObjective1(InterfaceObjCons):
    """First objective function (to be minimized)"""
    def compute(self, thedevice):
        return (1 + (thedevice.y+3)*thedevice.x**2)**0.5


class MyObjective2(InterfaceObjCons):
    """Second objective function (to be minimized)"""
    def compute(self, thedevice):
```

(continues on next page)

(continued from previous page)

```
        return 4 + 2*(thedevice.y+3)**0.5*(1+(thedevice.x-1)**2)**0.5


class MyConstraint(InterfaceObjCons):
    """Constraints, that needs to be <= 0"""
    def compute(self, thedevice):
        return thedevice.x - 0.55


if __name__ == "__main__":  # This line is necessary to spawn new processes
    """Start the main code. Instantiate previously defined classes."""
    theDevice = Device()
    theAlgo = OptimizationAlgorithm()
    theAlgo.set_option(theAlgo.OPTI_ALGORITHM, "NSGAII")  # You can change the␣
↪algorithm if you need ;)
    # theAlgo.set_option(theAlgo.NUMBER_OF_CORES, 2)  # Toggle this line to use more␣
↪cores. Default is 1 (single core).

    # Careful that it generates overhead -> only helpful when the characterization is␣
↪computationally expensive.

    theCharacterization = Characterization()

    """Variable to be optimized"""
    optimizationVariables = list()
    optimizationVariables.append(Real_OptimizationVariable('x', 0, 2))  #
    optimizationVariables.append(Real_OptimizationVariable('y', 0, 2))

    """Objective and constraints"""
    listOfObjectives = [MyObjective1(), MyObjective2()]
    listOfConstraints = [MyConstraint()]

    """Set the optimizer"""
    theOptiParameters = OptimizerSettings(theDevice, listOfObjectives,␣
↪listOfConstraints, optimizationVariables,
                                          theOptimizationAlgorithm=theAlgo,␣
↪theCharacterization=theCharacterization)

    """The logger (to automatically save the points)"""
    theOptiHistoric = OptiHistoric(optiname="opti", autosave_timer=10, autosave=True,␣
↪create_new_directory=True)

    """Start the optimization"""
    max_opti_time_sec = 10

    display_opti = True
    if display_opti:  # Display real-time graphs
        optiDisplayer = OptimizationDisplayer(theOptiParameters, theOptiHistoric,␣
↪light_background=True)
        _, theDataLink, _ = optiDisplayer.generate_optimizationGraphs()

        # Here we set the actions on click.
        theActionsOnClick = list()
        theActionsOnClick.append(Onclick_representDevice(theDataLink, [Represent_brut_
↪attributes()]))
        optiDisplayer.set_actionsOnClick(theActionsOnClick)
```

(continues on next page)

(continued from previous page)

```
        resultsOpti, convergence = optiDisplayer.launch_
↪optimization([theOptiParameters, theOptiHistoric], {"max_opti_time_sec": max_opti_
↪time_sec},
                                                                refresh_time=0.1,
↪ max_nb_points_convergence=None)  # Refresh the graphs each nth seconds

    else:  # Otherwise just focus on results ... That can be helpful if you are␣
↪confident the optimizations will converge and you need to launch several␣
↪optimizations.
        resultsOpti, convergence = run_optimization(theOptiParameters,␣
↪theOptiHistoric, max_opti_time_sec=max_opti_time_sec)

    """Gather results"""
    # Pro hint: you would probably never work with these next few lines of code,␣
↪instead you would move to the next tutorial
    # to retrieve the results from the automatically saved files.
    print("Best individuals :")
    for device in resultsOpti:
        print("x : {} \t y : {}". format(device.x, device.y))

    if display_opti:
        start_qt_mainloop()  # To keep windows alive

    """Note that the results are automatically saved if KWARGS_OPTIHISTO␣
↪autosaved=True.
    In this case, optimization folder is automatically generated in Workspace/optiX.␣
↪It contains five files:
    -> autosaved: contains all the devices evaluated during the optimization
    -> logopti: contains all the information relating to the optimization itself:␣
↪objectives, constraints, evaluation time.
    -> opticonvergence: contains all the information relative to the convergence of␣
↪the optimization (saved only at the end)
    -> results: all the best devices as decided by the optimization algorithm
    -> optimization_parameters: the class OptimizationParameters that can be reloaded␣
↪using SingleObjectSaveLoad.load
    -> summary.html: a summary of the optimization problem
    See other tutorials on how to save/load these information.
    """
```

3.2 Quickstart Visualization

Visualization implies to have a GUI, which will help to display many things: graphs, text, 3D representations, ... This software provides a clean interface to PyQt. PyQt works that way:

- A QMainWindow that includes layouts, (ex: horizontal, vertical, grid, ...)
- Layouts can include widgets.
- Widgets can be anything: buttons, menu, opengl 3D representation, graphs, ... Several high-level widgets are proposed, check *optimeed.visualize.widgets*.

3.2.1 Simple gui using OpenGL:

```
"""Optimeed also provides visualization tools using openGL. Here's how.
"""

# We already know these imports ...
from optimeed.core import ListDataStruct
from optimeed.core import LinkDataGraph, HowToPlotGraph
from optimeed.visualize import Widget_graphsVisual, MainWindow

# Now for the openGL imports:
from optimeed.visualize.widgets import Widget_openGL  # And we put all of that inside␣
↪a widget
from optimeed.visualize.openGL import DeviceDrawerInterface, Bronze_material
from optimeed.visualize.openGL.openGL_library import glPushMatrix, draw_simple_
↪rectangle, glPopMatrix, glTranslate
from optimeed.visualize.onclick import Onclick_animate, Animate_openGL  # And we put␣
↪the widget inside an action


class MyDevice:
    def __init__(self, length, height):
        self.length = length
        self.height = height
        self.surface = self.length * self.height


# The drawer class
class DeviceDrawer(DeviceDrawerInterface):
    """Drawer of the device"""
    def __init__(self):
        self.theDevice = None

    def draw(self, theDevice):
        glPushMatrix()  # Remove the previous matrices transformations
        glTranslate(-5, -5, 0)
        Bronze_material.activateMaterialProperties()  # Change colour aspect of the␣
↪material, here it will look like bronze
        draw_simple_rectangle(theDevice.length, theDevice.height)  # Thats the␣
↪interesting line
        glPopMatrix()  # Push back previous matrices transformations

    def get_init_camera(self, theDevice):
        tipAngle = 0
        viewAngle = 0
        zoomLevel = 0.1
        return tipAngle, viewAngle, zoomLevel


# From now on, that's the same as before
theDataStruct = ListDataStruct()
for xi in range(10):
    for yi in range(10):
        theDataStruct.add_data(MyDevice(xi, yi))

theDataLink = LinkDataGraph()
_ = theDataLink.add_collection(theDataStruct)
```

(continues on next page)

(continued from previous page)

```
howToPlot = HowToPlotGraph('length', 'surface', {'x_label': "length [m]", 'y_label':
↪"surface [m^2]", 'is_scattered': True})
theDataLink.add_graph(howToPlot)
theGraphs = theDataLink.get_graphs()

theActionsOnClick = list()

# That's where the fun begins
openGlDrawing = Widget_openGL()  # First we define the widget
openGlDrawing.set_deviceDrawer(DeviceDrawer())  # We set the drawer to the widget
theActionsOnClick.append(Onclick_animate(theDataLink, Animate_openGL(openGlDrawing)))␣
↪ # And we create an action out of it

# Same as before
myWidgetGraphsVisuals = Widget_graphsVisual(theGraphs,␣
↪actionsOnClick=theActionsOnClick, highlight_last=True, refresh_time=-1)  # The␣
↪widget to display the graphs
myWindow = MainWindow([myWidgetGraphsVisuals])  # A Window (that will contain the␣
↪widget)
myWindow.run(True)

# Now click on a point.
# Click on "show all"
# Watch the graph and animation together
```

3.3 Loading and saving data

You will probably have to often manipulate data, saving them and loading them.

Imagine the following structure to be saved:

```
class TopoA:
    def __init__(self):
        self.R_in = 3e-3
        self.R_out = 5e-3


class MyMotor:
    def __init__(self):
        self.rotor = TopoA()
        self.length = 5e-3
        self.dummyVariableToNotSave = 1234
```

optimeed provides a way to export that directly in JSON format. It detects the variables to save from type hints:

```
class TopoA:
    R_in: float
    R_out: float

    def __init__(self):
        self.R_in = 3e-3
        self.R_out = 5e-3
```

(continues on next page)

(continued from previous page)

```
class MyMotor:
    rotor: TopoA
    length: float

    def __init__(self):
        self.rotor = TopoA()
        self.length = 5e-3
        self.dummyVariableToNotSave = 1234
```

If type hint is not possible because some type is not known before the running time, optimeed provides an additional tool *SaveableObject*:

```
from optimeed.core import SaveableObject


class TopoA:
    R_in: float
    R_out: float

    def __init__(self):
        self.R_in = 3e-3
        self.R_out = 5e-3


class MyMotor(SaveableObject):
    length: float

    def __init__(self):
        self.rotor = TopoA()
        self.length = 5e-3
        self.dummyVariableToNotSave = 1234

    def get_additional_attributes_to_save(self):
        return ["rotor"]
```

The item can then be converted to a dictionary using *obj_to_json()*, which can then be converted to string liberal using "json.dumps" and written on a file. To recover To recover the object, read the file and interpret is as a dictionary using "json.load". Then, convert the dictionary by using *json_to_obj()*

Alternatively, it might be simpler to use the class *ListDataStruct* (or similar user-custom class), which provides high-level save and load option. This is what is done in `OptiHistoric`

CHAPTER 4

Gallery

4.1 Gallery

CHAPTER 5

License and support

5.1 License and Support

5.1.1 License

The project is distributed "has it is" under GNU General Public License v3.0 (GPL), which is a strong copyleft license. This means that the code is open-source and you are free to do anything you want with it, **as long as you apply the same license to distribute your code**. This constraining license is imposed by the use of Platypus Library as "optimization algorithm library", which is under GPL license.

It is perfectly possible to use other optimization library (which would use the same algorithms but with a different implementation) and to interface it to this project, so that the use of platypus is no longer needed. This work has already been done for NLopt, which is under MIT license (not constraining at all). In that case, **after removing all the platypus sources** (optiAlgorithms/multiObjective_GA and optiAlgorithsm/platypus/*), the license of the present work becomes less restrictive: GNU Lesser General Public License (LGPL). As for the GPL, this license makes the project open-source and free to be modified, but (nearly) no limitation is made to distribute your code.

5.1.2 Support

Github (preferably) / Send mail at christophe.degreef@uclouvain.be

CHAPTER 6

API

6.1 :py:mod:optimeed

6.1.1 Subpackages

consolidate

fit

Module Contents

Classes

Functions

class _Device(*fitFunction*, *nbArgs*)

class _Objective(*x_data*, *y_data*, *fitCriterion*)
Bases: *optimeed.optimize.InterfaceObjCons*

Interface class for objectives and constraints. The objective is to MINIMIZE and the constraint has to respect VALUE <= 0

compute(*theDevice*)
Get the value of the objective or the constraint. The objective is to MINIMIZE and the constraint has to respect VALUE <= 0

Parameters **theDevice** – Input device that has already been evaluated

Returns float.

leastSquare(*function*, *functionArgs*, *x_data*, *y_data*)
Least square calculation (sum (y-ŷ)^2)

Parameters

- **function** – Function to fit
- **functionArgs** – Arguments of the function
- **x_data** – x-axis coordinates of data to fit
- **y_data** – y-axis coordinates of data to fit

Returns least squares

r_squared(*function*, *functionArgs*, *x_data*, *y_data*)
R squared calculation

Parameters

- **function** – Function to fit
- **functionArgs** – Arguments of the function
- **x_data** – x-axis coordinates of data to fit
- **y_data** – y-axis coordinates of data to fit

Returns R squared

do_fit(*fitFunction*, *x_data*, *y_data*, **args*, *fitCriterion=leastSquare*)
Main method to fit a function

Parameters

- **fitFunction** – the function to fit (link to it)
- **x_data** – x-axis coordinates of data to fit
- **y_data** – y-axis coordinates of data to fit
- **args** – for each parameter: [min, max] admissible value
- **fitCriterion** – fit criterion to minimize. Default: least square

Returns [arg_i_optimal, . . .], y estimated, error.

parametric_analysis

Module Contents

Classes

class Parametric_Collection(***kwargs*)
Bases: *optimeed.core.collection.ListDataStruct*

class Parametric_parameter(*analyzed_attribute*, *reference_device*)
Abstract class for a parametric parameter

get_reference_device()

get_analyzed_attribute()

class Parametric_minmax(*analyzed_attribute*, *reference_device*, *minValue*, *maxValue*, *is_relative=False*, *npoints=10*)
Bases: *Parametric_parameter*

Abstract class for a parametric parameter

get_values()

class Parametric_analysis(*theParametricParameter*, *theCharacterization*, *file-name_collection=None*, *autosave=False*)

Bases: *optimeed.core.Option_class*

NUMBER_OF_CORES = 1

run()

Instantiates input arguments for analysis

evaluate(*theDevice*)

initialize_output_collection()

sensitivity_analysis

Module Contents

Classes

Functions

Attributes

_filename_sensitivityparams = sensitivity_params.json

_foldername_embarrassingly_parallel_results = _jobs_results

_filename_sensitivityresults = sensitivity.json

class SensitivityResults

Bases: *optimeed.core.SaveableObject*

Abstract class for dynamically type-hinted objects. This class is to solve the special case where the exact type of an attribute is not known before runtime, yet has to be saved.

paramsToEvaluate :List[float]

success :bool

index :int

add_data(*params*, *device*, *success*, *index*)

get_additional_attributes_to_save()

Return list of attributes corresponding to object, whose type cannot be determined statically (e.g. topology change)

class SensitivityParameters(*param_values*, *list_of_optimization_variables*, *theDevice*, *theMath-sToPhys*, *theCharacterization*)

Bases: *optimeed.core.SaveableObject*

Abstract class for dynamically type-hinted objects. This class is to solve the special case where the exact type of an attribute is not known before runtime, yet has to be saved.

list_of_optimization_variables :List[optimeed.optimize.Real_OptimizationVariable]

param_values :List[List[float]]

get_device()

get_M2P()

get_charac()

get_optivariables()

get_paramvalues()

get_additional_attributes_to_save()
Return list of attributes corresponding to object, whose type cannot be determined statically (e.g. topology change)

get_sensitivity_problem(*list_of_optimization_variables*)
This is the first method to use. Convert a list of optimization varisbles to a SALib problem

Parameters **list_of_optimization_variables** – List of optimization variables

Returns SALib problem

_get_sensitivity_result(*output*)
Convert output of "evaluate" function to SensitivityResult

_get_job_args(*theSensitivityParameters*, *index*)
Convert sensitivityparameters at index to args used in "evaluate" function

_find_missings(*theSensitivityParameters*, *studyname*)

prepare_embarrassingly_parallel_sensitivity(*theSensitivityParameters*, *studyname*)

launch_embarrassingly_parallel_sensitivity(*theSensitivityParameters*, *studyname*, *index*)

gather_embarrassingly_parallel_sensitivity(*theSensitivityParameters*, *studyname*)

evaluate_sensitivities(*theSensitivityParameters: SensitivityParameters*, *numberOfCores=2*, *studyname='sensitivity'*, *indices_to_evaluate=None*)
Evaluate the sensitivities

Parameters

- **theSensitivityParameters** – class'~SensitivityParameters'
- **numberOfCores** – number of core for multicore evaluation
- **studyname** – Name of the study, that will be the subfolder name in workspace
- **indices_to_evaluate** – if None, evaluate all param_values, otherwise if list: evaluate subset of param_values defined by indices_to_evaluate

Returns collection of class'~SensitivityResults'

analyse_sobol_create_array(*theSensitivityParameters: SensitivityParameters*, *objectives*)
Create readible result array, ordered by decreasing sobol indices.

Parameters

- **theSensitivityParameters** – class:*SensitivityParameters*
- **objectives** – array-like of objective

Returns tuples of STR, for S1 and ST

analyse_sobol_convergence(*theSensitivityParameters: SensitivityParameters*, *objectives*, *stepsize=1*)
Create dictionary for convergence plot

Parameters

- **theSensitivityParameters** – class:*SensitivityParameters*

- **objectives** – array-like of objective

Returns Dictionary

sensitivity_analysis_evaluation

Module Contents

Functions

evaluate(*inputs*)

Package Contents

Classes

Functions

class Option_class

options_bool :Dict[int, Option_bool]

options_str :Dict[int, Option_str]

options_int :Dict[int, Option_int]

options_float :Dict[int, Option_float]

options_dict :Dict[int, Option_dict]

add_option(*idOption*, *theOption*)

get_option_name(*idOption*)

get_option_value(*idOption*)

set_option(*idOption*, *value*)

_pack_options()

__str__()
Return str(self).

class Option_int(*name*, *based_value*, *choices=None*)
Bases: Base_Option

name :str

value :int

set_value(*value*)

class ListDataStruct(*compress_save=False*)
Bases: ListDataStruct_Interface

_DATA_STR = data

_COMPRESS_SAVE_STR = module_tree

__len__()

get_length()

clone(*filename*)
Clone the datastructure to a new location

save(*filename*)
Save data using json format. The data to be saved are automatically detected, see *obj_to_json()*

extract_collection_from_indices(*indices*)
Extract data from the collection at specific indices, and return it as new collection

_format_str_save()
Save data using json format. The data to be saved are automatically detected, see *obj_to_json()*

_format_data_lines()

_get_json_module_tree()

add_data(*data_in*)
Add a data to the list

get_data()
Get full list of datas

get_data_generator()
Get a generator to all the data stored

get_data_at_index(*index*)

set_data(*theData*)
Set full list of datas

set_data_at_index(*data_in*, *index*)
Replace data at specific index

reset_data()

delete_points_at_indices(*indices*)
Delete several elements from the Collection

Parameters **indices** – list of indices to delete

merge(*collection*)
Merge a collection with the current collection

Parameters **collection** – `Collection` to merge

get_nbr_elements()

Returns the number of elements contained inside the structure

class AutosaveStruct(*dataStruct*, *filename=''*, *change_filename_if_exists=True*)
Structure that provides automated save of DataStructures

__str__()
Return str(self).

get_filename()
Get set filename

set_filename(*filename*, *change_filename_if_exists*)

Parameters

- **filename** – Filename to set
- **change_filename_if_exists** – If already exists, create a new filename

stop_autosave()
Stop autosave

start_autosave(*timer_autosave*, *safe_save=True*)
Start autosave

save(*safe_save=True*)
Save

get_datastruct()
Return :class:'~DataStruct_Interface'

__getstate__()

__setstate__(*state*)

getPath_workspace()
Get workspace path (i.e., location where optimeed files will be created). Create directory if doesn't exist.

rsetattr(*obj*, *attr*, *val*)
setattr, but recursively. Works with list (i.e. theObj.myList[0].var_x)

rgetattr(*obj*, *attr*)
getattr, but recursively. Works with list.

class Parametric_Collection(**kwargs*)
Bases: *optimeed.core.collection.ListDataStruct*

class Parametric_parameter(*analyzed_attribute*, *reference_device*)
Abstract class for a parametric parameter

get_reference_device()

get_analyzed_attribute()

class Parametric_minmax(*analyzed_attribute*, *reference_device*, *minValue*, *maxValue*, *is_relative=False*, *npoints=10*)
Bases: *Parametric_parameter*

Abstract class for a parametric parameter

get_values()

class Parametric_analysis(*theParametricParameter*, *theCharacterization*, *file-name_collection=None*, *autosave=False*)
Bases: *optimeed.core.Option_class*

NUMBER_OF_CORES = 1

run()
Instantiates input arguments for analysis

evaluate(*theDevice*)

initialize_output_collection()

leastSquare(*function*, *functionArgs*, *x_data*, *y_data*)
Least square calculation (sum (y-ŷ)^2)

Parameters

- **function** – Function to fit
- **functionArgs** – Arguments of the function
- **x_data** – x-axis coordinates of data to fit

- **y_data** – y-axis coordinates of data to fit

Returns least squares

do_fit(*fitFunction*, *x_data*, *y_data*, **args*, *fitCriterion=leastSquare*)
Main method to fit a function

Parameters

- **fitFunction** – the function to fit (link to it)
- **x_data** – x-axis coordinates of data to fit
- **y_data** – y-axis coordinates of data to fit
- **args** – for each parameter: [min, max] admissible value
- **fitCriterion** – fit criterion to minimize. Default: least square

Returns [arg_i_optimal, . . .], y estimated, error.

get_sensitivity_problem(*list_of_optimization_variables*)
This is the first method to use. Convert a list of optimization varisbles to a SALib problem

Parameters **list_of_optimization_variables** – List of optimization variables

Returns SALib problem

evaluate_sensitivities(*theSensitivityParameters: SensitivityParameters*, *numberOfCores=2*, *studyname='sensitivity'*, *indices_to_evaluate=None*)
Evaluate the sensitivities

Parameters

- **theSensitivityParameters** – class‘~SensitivityParameters‘
- **numberOfCores** – number of core for multicore evaluation
- **studyname** – Name of the study, that will be the subfolder name in workspace
- **indices_to_evaluate** – if None, evaluate all param_values, otherwise if list: evaluate subset of param_values defined by indices_to_evaluate

Returns collection of class‘~SensitivityResults‘

class SensitivityParameters(*param_values*, *list_of_optimization_variables*, *theDevice*, *theMathsToPhys*, *theCharacterization*)
Bases: *optimeed.core.SaveableObject*

Abstract class for dynamically type-hinted objects. This class is to solve the special case where the exact type of an attribute is not known before runtime, yet has to be saved.

list_of_optimization_variables :List[optimeed.optimize.Real_OptimizationVariable]

param_values :List[List[float]]

get_device()

get_M2P()

get_charac()

get_optivariables()

get_paramvalues()

get_additional_attributes_to_save()
Return list of attributes corresponding to object, whose type cannot be determined statically (e.g. topology change)

analyse_sobol_convergence(*theSensitivityParameters: SensitivityParameters*, *objectives*, *step-size=1*)

Create dictionary for convergence plot

Parameters

- **theSensitivityParameters** – class:*SensitivityParameters*
- **objectives** – array-like of objective

Returns Dictionary

prepare_embarrassingly_parallel_sensitivity(*theSensitivityParameters*, *studyname*)

gather_embarrassingly_parallel_sensitivity(*theSensitivityParameters*, *studyname*)

launch_embarrassingly_parallel_sensitivity(*theSensitivityParameters*, *studyname*, *index*)

core

Subpackages

ansi2html

converter

Module Contents

Classes

Functions

Attributes

ANSI_FULL_RESET = 0

ANSI_INTENSITY_INCREASED = 1

ANSI_INTENSITY_REDUCED = 2

ANSI_INTENSITY_NORMAL = 22

ANSI_STYLE_ITALIC = 3

ANSI_STYLE_NORMAL = 23

ANSI_BLINK_SLOW = 5

ANSI_BLINK_FAST = 6

ANSI_BLINK_OFF = 25

ANSI_UNDERLINE_ON = 4

ANSI_UNDERLINE_OFF = 24

ANSI_CROSSED_OUT_ON = 9

ANSI_CROSSED_OUT_OFF = 29

ANSI_VISIBILITY_ON = 28

ANSI_VISIBILITY_OFF = 8

ANSI_FOREGROUND_CUSTOM_MIN = 30

ANSI_FOREGROUND_CUSTOM_MAX = 37

ANSI_FOREGROUND_256 = 38

ANSI_FOREGROUND_DEFAULT = 39

ANSI_BACKGROUND_CUSTOM_MIN = 40

ANSI_BACKGROUND_CUSTOM_MAX = 47

ANSI_BACKGROUND_256 = 48

ANSI_BACKGROUND_DEFAULT = 49

ANSI_NEGATIVE_ON = 7

ANSI_NEGATIVE_OFF = 27

ANSI_FOREGROUND_HIGH_INTENSITY_MIN = 90

ANSI_FOREGROUND_HIGH_INTENSITY_MAX = 97

ANSI_BACKGROUND_HIGH_INTENSITY_MIN = 100

ANSI_BACKGROUND_HIGH_INTENSITY_MAX = 107

VT100_BOX_CODES

_latex_template = Multiline-String

```
\documentclass{scrartcl}
\usepackage[utf8]{inputenc}
\usepackage{fancyvrb}
\usepackage[usenames,dvipsnames]{xcolor}
%% \definecolor{red-sd}{HTML}{7ed2d2}

\title{%(title)s}

\fvset{commandchars=\\\{\}}

\begin{document}

\begin{Verbatim}
%(content)s
\end{Verbatim}
\end{document}
```

_html_template

class _State

Bases: object

reset()

adjust(*ansi_code*, *parameter=None*)

to_css_classes()

linkify(*line*, *latex_mode*)

map_vt100_box_code(*char*)

_needs_extra_newline(*text*)

class CursorMoveUp
Bases: object

class Ansi2HTMLConverter(*latex=False*, *inline=False*, *dark_bg=True*, *line_wrap=True*, *font_size='normal'*, *linkify=False*, *escaped=True*, *markup_lines=False*, *output_encoding='utf-8'*, *scheme='ansi2html'*, *title=''*)
Bases: object

Convert Ansi color codes to CSS+HTML

Example: >>> conv = Ansi2HTMLConverter() >>> ansi = " ".join(sys.stdin.readlines()) >>> html = conv.convert(ansi)

apply_regex(*ansi*)

_apply_regex(*ansi*, *styles_used*)

_collapse_cursor(*parts*)
Act on any CursorMoveUp commands by deleting preceding tokens

prepare(*ansi=''*, *ensure_trailing_newline=False*)
Load the contents of 'ansi' into this object

attrs()
Prepare attributes for the template

convert(*ansi*, *full=True*, *ensure_trailing_newline=False*)

produce_headers()

main()
$ ls –color=always | ansi2html > directories.html $ sudo tail /var/log/messages | ccze -A | ansi2html > logs.html $ task burndown | ansi2html > burndown.html

style

Module Contents

Classes

Functions

Attributes

class Rule(*klass*, ***kw*)
Bases: object

__str__()
Return str(self).

index(*r*, *g*, *b*)

color_component(*x*)

color(*r*, *g*, *b*)

level(*grey*)

index2(*grey*)

SCHEME

intensify(*color*, *dark_bg*, *amount=64*)

get_styles(*dark_bg=True*, *line_wrap=True*, *scheme='ansi2html'*)

util

Module Contents

Functions

read_to_unicode(*obj*)

Package Contents

Classes

class Ansi2HTMLConverter(*latex=False*, *inline=False*, *dark_bg=True*, *line_wrap=True*, *font_size='normal'*, *linkify=False*, *escaped=True*, *markup_lines=False*, *output_encoding='utf-8'*, *scheme='ansi2html'*, *title=''*)

Bases: `object`

Convert Ansi color codes to CSS+HTML

Example: >>> conv = Ansi2HTMLConverter() >>> ansi = " ".join(sys.stdin.readlines()) >>> html = conv.convert(ansi)

apply_regex(*ansi*)

_apply_regex(*ansi*, *styles_used*)

_collapse_cursor(*parts*)
Act on any CursorMoveUp commands by deleting preceding tokens

prepare(*ansi=''*, *ensure_trailing_newline=False*)
Load the contents of 'ansi' into this object

attrs()
Prepare attributes for the template

convert(*ansi*, *full=True*, *ensure_trailing_newline=False*)

produce_headers()

additional_tools

Module Contents

Classes

Functions

Attributes

has_scipy = True

class fast_LUT_interpolation(*independent_variables*, *dependent_variables*)
Class designed for fast interpolation in look-up table when successive searchs are called often. Otherwise use griddata

interpolate(*point*, *fill_value=np.nan*)
Perform the interpolation :param point: coordinates to interpolate (tuple or list of tuples for multipoints) :param fill_value: value to put if extrapolated. :return: coordinates

interpolate_table(*x0*, *x_values*, *y_values*)
From sorted table (x,y) find y0 corresponding to x0 (linear interpolation)

derivate(*t*, *y*)

linspace(*start*, *stop*, *npoints*)

reconstitute_signal(*amplitudes*, *phases*, *numberOfPeriods=1*, *x_points=None*, *n_points=50*)
Reconstitute the signal from fft. Number of periods of the signal must be specified if different of 1

my_fft(*y*)
Real FFT of signal Bx, with real amplitude of harmonics. Input signal must be within a period.

cart2pol(*x*, *y*)

pol2cart(*rho*, *phi*)

partition(*array*, *begin*, *end*)

quicksort(*array*)

dist(*p*, *q*)
Return the Euclidean distance between points p and q. :param p: [x, y] :param q: [x, y] :return: distance (float)

sparse_subset(*points*, *r*)
Returns a maximal list of elements of points such that no pairs of points in the result have distance less than r. :param points: list of tuples (x,y) :param r: distance :return: corresponding subset (list), indices of the subset (list)

integrate(*x*, *y*)
Performs Integral(x[0] to x[-1]) of y dx

Parameters

- **x** – x axis coordinates (list)
- **y** – y axis coordinates (list)

Returns integral value

my_fourier(*x*, *y*, *n*, *L*)
Fourier analys

Parameters

- **x** – x axis coordinates
- **y** – y axis coordinates
- **n** – number of considered harmonic
- **L** – half-period length

Returns a and b coefficients (y = a*cos(x) + b*sin(y))

get_ellipse_axes(*a*, *b*, *dphi*)
Trouve les longueurs des axes majeurs et mineurs de l'ellipse, ainsi que l'orientation de l'ellipse. ellipse: x(t) =

A*cos(t), y(t) = B*cos(t+dphi) Etapes: longueur demi ellipse CENTRéE = sqrt(a^2 cos^2(x) + b^2 cos^2(t+phi) Minimisation de cette formule => obtention formule tg(2x) = alpha/beta

convert_color(*color*)
Convert a color to a tuple if color is a char, otherwise return the tuple.

Parameters **color** – (r,g,b) or char.

Returns

convert_color_with_alpha(*color*, *alpha=255*)
Same as meth:*convert_color* but with transparency

collection

Module Contents

Classes

Attributes

class SingleObjectSaveLoad

class DataStruct_Interface

__str__()
Return str(self).

class ListDataStruct_Interface
Bases: *DataStruct_Interface*

get_list_attributes(*attributeName*)
Get the value of attributeName of all the data in the Collection

Parameters **attributeName** – string (name of the attribute to get)

Returns list

class AutosaveStruct(*dataStruct*, *filename=''*, *change_filename_if_exists=True*)
Structure that provides automated save of DataStructures

__str__()
Return str(self).

get_filename()
Get set filename

set_filename(*filename*, *change_filename_if_exists*)

Parameters

- **filename** – Filename to set
- **change_filename_if_exists** – If already exists, create a new filename

stop_autosave()
Stop autosave

start_autosave(*timer_autosave*, *safe_save=True*)
Start autosave

save(*safe_save=True*)
Save

get_datastruct()
Return :class:'~DataStruct_Interface'

__getstate__()

__setstate__(*state*)

class ListDataStruct(*compress_save=False*)
Bases: *ListDataStruct_Interface*

_DATA_STR = data

_COMPRESS_SAVE_STR = module_tree

__len__()

get_length()

clone(*filename*)
Clone the datastructure to a new location

save(*filename*)
Save data using json format. The data to be saved are automatically detected, see *obj_to_json()*

extract_collection_from_indices(*indices*)
Extract data from the collection at specific indices, and return it as new collection

_format_str_save()
Save data using json format. The data to be saved are automatically detected, see *obj_to_json()*

_format_data_lines()

_get_json_module_tree()

add_data(*data_in*)
Add a data to the list

get_data()
Get full list of datas

get_data_generator()
Get a generator to all the data stored

get_data_at_index(*index*)

set_data(*theData*)
Set full list of datas

set_data_at_index(*data_in*, *index*)
Replace data at specific index

reset_data()

delete_points_at_indices(*indices*)
Delete several elements from the Collection

Parameters **indices** – list of indices to delete

merge(*collection*)
Merge a collection with the current collection

Parameters **collection** – `Collection` to merge

get_nbr_elements()

Returns the number of elements contained inside the structure

theLock

class Performance_ListDataStruct(*stack_size=500*)
Bases: *ListDataStruct_Interface*

_NBR_ELEMENTS = nbr_elements

_STACK_SIZE = stack_size

_COMPRESS_SAVE_STR = module_tree

_initialize(*filename*)

_get_list_from_file(*filenumber*)

extract_collection_from_indices(*indices*)
Extract data from the collection at specific indices, and return it as new collection

clone(*filename*)
Clone the datastructure to a new location

_get_str_mainfile()

get_total_nbr_elements(*count_unsaved=True*)

add_data(*theData*)
Add data to the collection

add_json_data(*theStr*)
Add already deserialized data to the collection

_save_moduletree(*theDict*)

_map_index_to_file(*index*)

_get_json_str_at_index(*index*, *refresh_cache=False*)
Internal method to return the json string at index

reorder(*permutations*)
Reorder collection accordingly to permutations. E.G, list_of_indices = [0,3,2] with collection elems [0,2,1] => collection elems = [0,2,3] :param permutations: :return: /

get_data_at_index(*index*, *ignore_attributes=None*, *none_if_error=False*)
Same as parent, with additional kwargs

Parameters

- **index** –
- **ignore_attributes** – ignore attributes to deserialize (list)
- **none_if_error** –

Returns

save(*filename*)
Save the datastructure to filename

get_data_generator(***kwargs*)

get_nbr_elements()

Returns the number of elements contained inside the structure

set_data_at_index(*data_in*, *index*)
Replace data at specific index

set_data_at_indices(*data_list*, *indices*)
Replace datas at specific indices :param data_list: list of objects to set to the collection, at specific indices :param indices: list of indices :return:

delete_points_at_indices(*indices*)
Delete several elements from the Collection

Parameters **indices** – list of indices to delete

color_palette

Module Contents

Functions

default_palette(*N*)

blackOnly(*N*)

dark2(*N*)

commonImport

Module Contents

Functions

Attributes

SHOW_WARNING = 0

SHOW_INFO = 1

SHOW_ERROR = 2

SHOW_DEBUG = 3

SHOW_LOGS = 4

SHOW_CURRENT

setCurrentShow(*show_types*)
Change text type to be displayed by PrintIfShown

getCurrentShow()
Get text type to be displayed by PrintIfShown

disableLogs()
Disable all logs

enableLogs()
Show all logs

graphs

Module Contents

Classes

class Data(*x: list*, *y: list*, *x_label=''*, *y_label=''*, *legend=''*, *is_scattered=False*, *transfo_x=lambda selfData*, *x: x*, *transfo_y=lambda selfData*, *y: y*, *xlim=None*, *ylim=None*, *permutations=None*, *sort_output=False*, *color=None*, *alpha=255*, *symbol='o'*, *symbolsize=8*, *fillsymbol=True*, *outlinesymbol=1.8*, *linestyle='-'*, *width=2*, *meta=None*)

This class is used to store informations necessary to plot a 2D graph. It has to be combined with a gui to be useful (ex. pyqtgraph)

set_kwargs(*kwargs*)
Set a kwarg after creation of the class

set_data(*x: list*, *y: list*)
Overwrites current datapoints with new set

set_meta(*meta*)
Set associated 'Z' data

get_x()
Get x coordinates of datapoints

get_symbolsize()
Get size of the symbols

symbol_isfilled()
Check if symbols has to be filled or not

get_symbolOutline()
Get color factor of outline of symbols

get_length_data()
Get number of points

get_xlim()
Get x limits of viewbox

get_ylim()
Get y limits of viewbox

get_y()
Get y coordinates of datapoints

get_meta()
Get associated 'Z' data

get_color()
Get color of the line, without transformation

get_color_alpha()
Get color of the line. Return r, g, b in 0, 255 scale

get_alpha()
Get opacity

get_width()
Get width of the line

get_number_of_points()
Get number of points

get_plot_data()
Call this method to get the x and y coordinates of the points that have to be displayed. => After transformation, and after permutations.

Returns x (list), y (list)

get_plot_meta(*x*, *y*)
Call this method to get the z coordinates of the points that been displayed. => After transformation, and after permutations.

Returns z (list)

get_permutations(*x=None*)
Return the transformation 'permutation': xplot[i] = xdata[permutation[i]]

get_invert_permutations()
Return the inverse of permutations: xdata[i] = xplot[revert[i]]

get_dataIndex_from_graphIndex(*index_graph_point*)
From an index given in graph, recovers the index of the data.

Parameters **index_graph_point** – Index in the graph

Returns index of the data

get_dataIndices_from_graphIndices(*index_graph_point_list*)
Same as get_dataIndex_from_graphIndex but with a list in entry. Can (?) improve performances for huge dataset.

Parameters **index_graph_point_list** – List of Index in the graph

Returns List of index of the data

get_graphIndex_from_dataIndex(*index_data*)
From an index given in the data, recovers the index of the graph.

Parameters **index_data** – Index in the data

Returns index of the graph

get_graphIndices_from_dataIndices(*index_data_list*)
Same as get_graphIndex_from_dataIndex but with a list in entry. Can (?) improve performances for huge dataset.

Parameters **index_data_list** – List of Index in the data

Returns List of index of the graph

set_permutations(*permutations*)
Set permutations between datapoints of the trace

Parameters **permutations** – list of indices to plot (example: [0, 2, 1] means that the first point will be plotted, then the third, then the second one)

get_x_label()
Get x label of the trace

get_y_label()
Get y label of the trace

get_legend()
Get name of the trace

get_symbol()
Get symbol

add_point(*x*, *y*)
Add point(s) to trace (inputs can be list or numeral)

delete_point(*index_point*)
Delete a point from the datapoints

isScattered()
Check if plot is scatteded

set_indices_points_to_plot(*indices*)
Set indices points to plot

get_indices_points_to_plot()
Get indices points to plot

get_linestyle()
Get linestyle

__str__()
Return str(self).

export_str()
Method to save the points constituting the trace

set_color(*theColor*)
Set trace color

set_legend(*theLegend*)
Set legend

class Graph
Simple graph container that contains several traces

add_trace(*data*)
Add a trace to the graph

Parameters **data** – *Data*

Returns id of the created trace

remove_trace(*idTrace*)
Delete a trace from the graph

Parameters **idTrace** – id of the trace to delete

get_trace(*idTrace*) → Data
Get data object of idTrace

Parameters **idTrace** – id of the trace to get

Returns *Data*

get_all_traces()
Get all the traces id of the graph

get_all_traces_ids()
Get all the traces id of the graph :return: list of id graphs

export_str()

class Graphs
Contains several *Graph*

updateChildren()

add_trace_firstGraph(*data*, *updateChildren=True*)
Same as add_trace, but only if graphs has only one id :param data: :param updateChildren: :return:

add_trace(*idGraph*, *data*, *updateChildren=True*)
Add a trace to the graph

Parameters

- **idGraph** – id of the graph
- **data** – *Data*
- **updateChildren** – Automatically calls callback functions

Returns id of the created trace

remove_trace(*idGraph*, *idTrace*, *updateChildren=True*)
Remove the trace from the graph

Parameters

- **idGraph** – id of the graph
- **idTrace** – id of the trace to remove
- **updateChildren** – Automatically calls callback functions

get_first_graph()
Get id of the first graph

Returns id of the first graph

get_graph(*idGraph*)
Get graph object at idgraph

Parameters **idGraph** – id of the graph to get

Returns *Graph*

get_all_graphs_ids()
Get all ids of the graphs

Returns list of id graphs

get_all_graphs()
Get all graphs. Return dict {id: *Graph*}

add_graph(*updateChildren=True*)
Add a new graph

Returns id of the created graph

remove_graph(*idGraph*)
Delete a graph

Parameters **idGraph** – id of the graph to delete

add_update_method(*childObject*)
Add a callback each time a graph is modified.

Parameters **childObject** – method without arguments

export_str()
Export all the graphs in text

Returns str

merge(*otherGraphs*)

reset()

is_empty()

graphs3

Module Contents

Classes

Functions

Attributes

griddata_found = True

class Plot3D_Generic(*x_label=''*, *y_label=''*, *z_label=''*, *legend=''*, *x_lim=None*, *y_lim=None*, *z_lim=None*)

get_lim(*axis*)

get_label(*axis*)

get_legend()

class GridPlot_Generic(*X*, *Y*, *Z*, ***kwargs*)
Bases: *Plot3D_Generic*

get_plot_data()

class ContourPlot(**args*, ***kwargs*)
Bases: *GridPlot_Generic*

get_levels()

get_number_of_contours()

class FilledContourPlot(**args*, ***kwargs*)
Bases: *ContourPlot*

class SurfPlot(*X*, *Y*, *Z*, ***kwargs*)
Bases: *GridPlot_Generic*

class MeshPlot(*X*, *Y*, *Z*, ***kwargs*)
Bases: *GridPlot_Generic*

class ScatterPlot3(*x*, *y*, *z*, ***kwargs*)
Bases: *Plot3D_Generic*

get_plot_data()

get_color()

convert_to_gridplot(*x*, *y*, *z*, *x_interval=None*, *y_interval=None*, *n_x=20*, *n_y=20*)
Convert set of points x, y, z to a grid

Parameters

- **x** –

- **y** –
- **z** –
- **x_interval** – [Min, max] of the grid. If none, use min and max values
- **y_interval** – [Min, max] of the grid. If none, use min and max values
- **n_x** – number of points in x direction
- **n_y** – number of points in y direction

Returns X, Y, Z as grid

inkscape_manager

Module Contents

Functions

Attributes

get_path_to_inkscape()

get_inkscape_version()

inkscape_version

inkscape_svg_to_pdf(*filename_svg*, *filename_pdf*)

inkscape_svg_to_png(*filename_svg*, *filename_png*)

linkDataGraph

Module Contents

Classes

class HowToPlotGraph(*attribute_x*, *attribute_y*, *kwargs_graph=None*, *check_if_plot_elem=None*, *meta=None*)

__str__()
Return str(self).

class LinkDataGraph

add_collection(*theCollection*, *kwargs=None*)
Add a collection (that will be a future trace)

Parameters

- **theCollection** –
- **kwargs** – kwargs associated with the collection (e.g., color, symbol style, etc.)

Returns unique id associated with the collection

remove_collection(*collectionId*)
Remove collection from the graphs

Parameters **collectionId** – ID of the collection

Returns

set_shadow_collection(*master_collectionId*, *shadow_collection*)
Link a collection to an other

Parameters

- **master_collectionId** – ID of the collection that is displayed in the graph
- **shadow_collection** – collection to link to the master.

Returns

get_graphs()

get_howToPlotGraph(*idGraph*)

add_graph(*howToPlotGraph*)
Add new graph to be plotted.

Parameters **howToPlotGraph** – *HowToPlotGraph*

Returns

get_idCollections()
Get all ids of the plotted collections

get_idGraphs()
Get all ids of the graphs

get_idTraces(*idGraph*)
Get all ids of the traces of graph $idGraph

get_idCollection_from_graph(*idGraph*, *idTrace*)
Get id of collection plotted in graph $idGraph and trace $idTrace

get_collection(*idCollection*, *getShadow=True*)

update_graphs()
Update the graphs: update graphs, traces, and X-Y data

get_collection_from_graph(*idGraph*, *idTrace*, *getShadow=True*) → optimeed.core.ListDataStruct_Interface
From indices in the graph, get corresponding collection

get_clicked_item(*idGraph*, *idTrace*, *idPoint*, *getShadow=True*)
Get the data hidden behind the clicked point

Parameters

- **idGraph** – ID of the graph
- **idTrace** – ID of the trace
- **idPoint** – ID of the point
- **getShadow** – If true, will return the data from the collection linked to the collection that is plotted

Returns Object in collection

get_clicked_items(*idGraph*, *idTrace*, *idPoint_list*, *getShadow=True*)
Same as get_clicked_item, but using a list of points

delete_clicked_item(*idGraph*, *idTrace*, *idPoint*)
Remove item from the collection

delete_clicked_items(*idGraph*, *idTrace*, *idPoints*)
Same, but for a list of points

get_graph_and_trace_from_idCollection(*idCollection*)
Reverse search: from a collection, get all associated graphs

get_idcollection_from_collection(*theCollection*)
Reverse search: from a collection, find its id

get_idPoints_from_indices_in_collection(*idGraph*, *idTrace*, *indices_in_collection*)
From indices in a collection, find the associated idPoints of the graph

myjson

Module Contents

Classes

Functions

Attributes

MODULE_TAG = __module__

CLASS_TAG = __class__

EXCLUDED_TAGS

getExecPath()

class SaveableObject
Abstract class for dynamically type-hinted objects. This class is to solve the special case where the exact type of an attribute is not known before runtime, yet has to be saved.

get_additional_attributes_to_save()
Return list of attributes corresponding to object, whose type cannot be determined statically (e.g. topology change)

get_additional_attributes_to_save_list()
Same behavior as get_additional_attributes_to_save, but where the attributes contains list of unknown items

_isclass(*theObject*)
Extends the default isclass method with typing

get_type_class(*typ*)
Get the type of the class. used to compare objects from Typing.

_get_object_class(*theObj*)

_get_object_module(*theObj*)

_object_to_FQCN(*theobj*)
Gets module path of object

_find_class(*moduleName*, *className*)

json_to_obj(*json_dict*)
Convenience class to create object from dictionary. Only works if CLASS_TAG is valid

Parameters **json_dict** – dictionary loaded from a json file.

Raises

- **TypeError** – if class can not be found
- **KeyError** – if CLASS_TAG not present in dictionary

json_to_obj_safe(*json_dict*, *cls*)
Safe class to create object from dictionary.

Parameters

- **json_dict** – dictionary loaded from a json file
- **cls** – class object to instantiate with dictionary

_instantiates_annotated_object(*_json_dict*, *_cls*)

_get_annotations(*theObj*)
Return annotated attributes (theObj being the type of the object)

obj_to_json(*theObj*)
Extract the json dictionary from the object. The data saved are automatically detected, using typehints. ex: x: int=5 will be saved, x=5 won't. Inheritance of annotation is managed by this function

_get_attributes_to_save(*theObj*)
Return list (attribute, is_first)

get_json_module_tree_from_dict(*jsonDict*)
Return dict containing {CLASS_TAG: "class_name", MODULE_TAG: "module_name", "attribute1":{"class_name": "module_name", ... }}

remove_module_tree_from_string(*theStr*)
Used to compress string by removing __module__ and __class__ entries (used with get_json_module_tree_from_dict)

apply_module_tree_to_dict(*nestedTree*, *nestedObject*, *raiseError=False*)
Restore __module__ and __class__ entries from nestedTree in nestedDict

encode_str_json(*theStr*)

decode_str_json(*theStr*)

options

Module Contents

Classes

class Base_Option(*name*, *based_value*, *choices=None*)

get_value()

get_name()

set_value(*value*)

get_choices()

class Option_bool(*name*, *based_value*, *choices=None*)
Bases: *Base_Option*

name :str

value :bool

set_value(*value*)

get_choices()

class Option_str(*name*, *based_value*, *choices=None*)
Bases: *Base_Option*

name :str

value :str

set_value(*value*)

class Option_int(*name*, *based_value*, *choices=None*)
Bases: *Base_Option*

name :str

value :int

set_value(*value*)

class Option_float(*name*, *based_value*, *choices=None*)
Bases: *Base_Option*

name :str

value :float

set_value(*value*)

class Option_dict(*name*, *based_value*, *choices=None*)
Bases: *Base_Option*

name :str

value :dict

set_value(*value*)

class Option_class

options_bool :Dict[int, Option_bool]

options_str :Dict[int, Option_str]

options_int :Dict[int, Option_int]

options_float :Dict[int, Option_float]

options_dict :Dict[int, Option_dict]

add_option(*idOption*, *theOption*)

get_option_name(*idOption*)

get_option_value(*idOption*)

set_option(*idOption*, *value*)

_pack_options()

__str__()
Return str(self).

tikzTranslator

Module Contents

Functions

Attributes

templates_tikz

format_escape_char(*theStr*)

convert_linestyle(*linestyle*)

find_all_colors(*theGraphs*)

convert_marker(*marker*)

do_preamble()

do_generate_figure()

do_specific_axis_options(*theGraph: optimeed.core.graphs.Graph*)
Get graph-specific axis options

do_specific_trace_options(*theTrace: optimeed.core.graphs.Data*, *theColor*)
Get latex trace options from Data

export_to_tikz_groupGraphs(*theGraphs: optimeed.core.graphs.Graphs*, *foldername*, *additionalPreamble=lambda: ''*, *additionalAxisOptions=lambda graphId: ''*, *additionalTraceOptions=lambda graphId*, *traceId: ''*, *debug=False*)
Export the graphs as group

Parameters

- **theGraphs** – Graphs to save
- **foldername** – Foldername to save
- **additionalPreamble** – method that returns string for custom tikz options
- **additionalAxisOptions** – method that returns string for custom tikz options
- **additionalTraceOptions** – method that returns string for custom tikz options

Returns

do_preamble3D()

format_Griddata(*X*, *Y*, *Z*)

format_scatterdata(*x*, *y*, *z*)

export_to_tikz_contour_plot(*list_of_traces3*, *foldername*, *filename_data='data'*)
Export the graphs as group

Parameters

- **list_of_traces3** – List of 3D traces
- **foldername** – Foldername to save

- **filename_data** – filename of the data

Returns

tools

Module Contents

Classes

Functions

Attributes

_workspace_path

class text_format

PURPLE = [95m

CYAN = [96m

DARKCYAN = [36m

BLUE = [94m

GREEN = [92m

YELLOW = [93m

WHITE = [30m

RED = [91m

BOLD = [1m

UNDERLINE = [4m

END = [0m

software_version()

find_and_replace(*begin_char*, *end_char*, *theStr*, *replace_function*)

create_unique_dirname(*dirname*)

Create dirname if it doesn't exists, otherwise append an integer to dirname and create it.

Parameters **dirname** – name of the directory to create

Returns name of the directory created

applyEquation(*objectIn*, *s*)

Apply literal expression based on an object

Parameters

- **objectIn** – Object
- **s** – literal expression. Float variables taken from the object are written between {}, int between []. Example: s="{x}+{y}*2" if x and y are attributes of objectIn.

Returns value (float)

arithmeticEval(*s*)

isNonePrintMessage(*theObject*, *theMessage*, *show_type=SHOW_INFO*)

getPath_workspace()
Get workspace path (i.e., location where optimeed files will be created). Create directory if doesn't exist.

setPath_workspace(*thePath*)
Set workspace path (i.e., location where optimeed files will be created)

getLineInfo(*lvl=1*)

printIfShown(*theStr*, *show_type=SHOW_DEBUG*, *isToPrint=True*, *appendTypeName=True*, *end='n'*)

universalPath(*thePath*)

add_suffix_to_path(*thePath*, *suffix*)

get_object_attrs(*obj*)

rsetattr(*obj*, *attr*, *val*)
setattr, but recursively. Works with list (i.e. theObj.myList[0].var_x)

rgetattr(*obj*, *attr*)
getattr, but recursively. Works with list.

indentParagraph(*text_in*, *indent_level=1*)
Add ' ' at beginning of strings and after each ' '.

truncate(*theStr*, *truncsize*)

get_recursive_attrs(*theObject*, *max_recursion_level=2*)

str_all_attr(*theObject*, *max_recursion_level*)

get_2D_pareto(*xList*, *yList*, *max_X=True*, *max_Y=True*)
Get 2D pareto front

Parameters

- **xList** – list of x coordinates
- **yList** – list of y coordinates
- **max_X** – True if x is to maximize
- **max_Y** – true if y is to maximize

Returns x pareto-optimal coordinates, y pareto-optimal coordinates, indices of these points in input parameters

get_ND_pareto(*objectives_list*, *are_maxobjectives_list=None*)
Return the N-D pareto front

Parameters

- **objectives_list** – list of list of objectives: example [[0,1], [1,1], [2,2]]
- **are_maxobjectives_list** – for each objective, tells if they are to be maximized or not: example [True, False]. Default: False

Returns extracted_pareto, indices: list of [x, y, ...] points forming the pareto front, and list of the indices of these points from the base list.

delete_indices_from_list(*indices*, *theList*)
Delete elements from list at indices

Parameters

- **indices** – list
- **theList** – list

merge_two_dicts(*dict1*, *dict2*)
Merge two dicts without affecting them

Returns new dictionary

deep_sizeof(*obj*)

order_lists(*ref_list*, *linked_list*)

Package Contents

Classes

Functions

Attributes

_workspace_path

class text_format

PURPLE = [95m

CYAN = [96m

DARKCYAN = [36m

BLUE = [94m

GREEN = [92m

YELLOW = [93m

WHITE = [30m

RED = [91m

BOLD = [1m

UNDERLINE = [4m

END = [0m

software_version()

find_and_replace(*begin_char*, *end_char*, *theStr*, *replace_function*)

create_unique_dirname(*dirname*)
Create dirname if it doesn't exists, otherwise append an integer to dirname and create it.

Parameters **dirname** – name of the directory to create

Returns name of the directory created

applyEquation(*objectIn*, *s*)
Apply literal expression based on an object

Parameters

- **objectIn** – Object
- **s** – literal expression. Float variables taken from the object are written between {}, int between []. Example: s="{x}+{y}*2" if x and y are attributes of objectIn.

Returns value (float)

arithmeticEval(*s*)

isNonePrintMessage(*theObject*, *theMessage*, *show_type=SHOW_INFO*)

getPath_workspace()
Get workspace path (i.e., location where optimeed files will be created). Create directory if doesn't exist.

setPath_workspace(*thePath*)
Set workspace path (i.e., location where optimeed files will be created)

getLineInfo(*lvl=1*)

printIfShown(*theStr*, *show_type=SHOW_DEBUG*, *isToPrint=True*, *appendTypeName=True*, *end='n'*)

universalPath(*thePath*)

add_suffix_to_path(*thePath*, *suffix*)

get_object_attrs(*obj*)

rsetattr(*obj*, *attr*, *val*)
setattr, but recursively. Works with list (i.e. theObj.myList[0].var_x)

rgetattr(*obj*, *attr*)
getattr, but recursively. Works with list.

indentParagraph(*text_in*, *indent_level=1*)
Add ' ' at beginning of strings and after each ' '.

truncate(*theStr*, *truncsize*)

get_recursive_attrs(*theObject*, *max_recursion_level=2*)

str_all_attr(*theObject*, *max_recursion_level*)

get_2D_pareto(*xList*, *yList*, *max_X=True*, *max_Y=True*)
Get 2D pareto front

Parameters

- **xList** – list of x coordinates
- **yList** – list of y coordinates
- **max_X** – True if x is to maximize
- **max_Y** – true if y is to maximize

Returns x pareto-optimal coordinates, y pareto-optimal coordinates, indices of these points in input parameters

get_ND_pareto(*objectives_list*, *are_maxobjectives_list=None*)
Return the N-D pareto front

Parameters

- **objectives_list** – list of list of objectives: example [[0,1], [1,1], [2,2]]
- **are_maxobjectives_list** – for each objective, tells if they are to be maximized or not: example [True, False]. Default: False

Returns extracted_pareto, indices: list of [x, y, ...] points forming the pareto front, and list of the indices of these points from the base list.

delete_indices_from_list(*indices*, *theList*)
Delete elements from list at indices

Parameters

- **indices** – list
- **theList** – list

merge_two_dicts(*dict1*, *dict2*)
Merge two dicts without affecting them

Returns new dictionary

deep_sizeof(*obj*)

order_lists(*ref_list*, *linked_list*)

SHOW_WARNING = 0

SHOW_INFO = 1

SHOW_ERROR = 2

SHOW_DEBUG = 3

SHOW_LOGS = 4

SHOW_CURRENT

setCurrentShow(*show_types*)
Change text type to be displayed by PrintIfShown

getCurrentShow()
Get text type to be displayed by PrintIfShown

disableLogs()
Disable all logs

enableLogs()
Show all logs

SHOW_WARNING = 0

SHOW_INFO = 1

SHOW_ERROR = 2

SHOW_DEBUG = 3

SHOW_LOGS = 4

SHOW_CURRENT

setCurrentShow(*show_types*)
Change text type to be displayed by PrintIfShown

getCurrentShow()
Get text type to be displayed by PrintIfShown

disableLogs()
Disable all logs

enableLogs()
Show all logs

getPath_workspace()
Get workspace path (i.e., location where optimeed files will be created). Create directory if doesn't exist.

obj_to_json(*theObj*)
Extract the json dictionary from the object. The data saved are automatically detected, using typehints. ex: x: int=5 will be saved, x=5 won't. Inheritance of annotation is managed by this function

json_to_obj(*json_dict*)
Convenience class to create object from dictionary. Only works if CLASS_TAG is valid

Parameters **json_dict** – dictionary loaded from a json file.

Raises

- **TypeError** – if class can not be found
- **KeyError** – if CLASS_TAG not present in dictionary

json_to_obj_safe(*json_dict*, *cls*)
Safe class to create object from dictionary.

Parameters

- **json_dict** – dictionary loaded from a json file
- **cls** – class object to instantiate with dictionary

encode_str_json(*theStr*)

decode_str_json(*theStr*)

get_json_module_tree_from_dict(*jsonDict*)
Return dict containing {CLASS_TAG: "class_name", MODULE_TAG: "module_name", "attribute1":{"class_name": "module_name", . . . }}

remove_module_tree_from_string(*theStr*)
Used to compress string by removing __module__ and __class__ entries (used with get_json_module_tree_from_dict)

apply_module_tree_to_dict(*nestedTree*, *nestedObject*, *raiseError=False*)
Restore __module__ and __class__ entries from nestedTree in nestedDict

indentParagraph(*text_in*, *indent_level=1*)
Add ' ' at beginning of strings and after each ' '.

rgetattr(*obj*, *attr*)
getattr, but recursively. Works with list.

printIfShown(*theStr*, *show_type=SHOW_DEBUG*, *isToPrint=True*, *appendTypeName=True*, *end='n'*)

SHOW_WARNING = 0

SHOW_DEBUG = 3

SHOW_INFO = 1

SHOW_ERROR = 2

delete_indices_from_list(*indices*, *theList*)
Delete elements from list at indices

Parameters

- **indices** – list
- **theList** – list

class SingleObjectSaveLoad

class DataStruct_Interface

__str__()
Return str(self).

class ListDataStruct_Interface
Bases: *DataStruct_Interface*

get_list_attributes(*attributeName*)
Get the value of attributeName of all the data in the Collection

Parameters **attributeName** – string (name of the attribute to get)

Returns list

class AutosaveStruct(*dataStruct*, *filename=''*, *change_filename_if_exists=True*)
Structure that provides automated save of DataStructures

__str__()
Return str(self).

get_filename()
Get set filename

set_filename(*filename*, *change_filename_if_exists*)

Parameters

- **filename** – Filename to set
- **change_filename_if_exists** – If already exists, create a new filename

stop_autosave()
Stop autosave

start_autosave(*timer_autosave*, *safe_save=True*)
Start autosave

save(*safe_save=True*)
Save

get_datastruct()
Return :class:'~DataStruct_Interface'

__getstate__()

__setstate__(*state*)

class ListDataStruct(*compress_save=False*)
Bases: *ListDataStruct_Interface*

_DATA_STR = data

_COMPRESS_SAVE_STR = module_tree

__len__()

get_length()

clone(*filename*)
Clone the datastructure to a new location

save(*filename*)
Save data using json format. The data to be saved are automatically detected, see *obj_to_json()*

extract_collection_from_indices(*indices*)
Extract data from the collection at specific indices, and return it as new collection

_format_str_save()
Save data using json format. The data to be saved are automatically detected, see *obj_to_json()*

_format_data_lines()

_get_json_module_tree()

add_data(*data_in*)
Add a data to the list

get_data()
Get full list of datas

get_data_generator()
Get a generator to all the data stored

get_data_at_index(*index*)

set_data(*theData*)
Set full list of datas

set_data_at_index(*data_in*, *index*)
Replace data at specific index

reset_data()

delete_points_at_indices(*indices*)
Delete several elements from the Collection

Parameters **indices** – list of indices to delete

merge(*collection*)
Merge a collection with the current collection

Parameters **collection** – `Collection` to merge

get_nbr_elements()

Returns the number of elements contained inside the structure

theLock

class Performance_ListDataStruct(*stack_size=500*)
Bases: *ListDataStruct_Interface*

_NBR_ELEMENTS = nbr_elements

_STACK_SIZE = stack_size

_COMPRESS_SAVE_STR = module_tree

_initialize(*filename*)

_get_list_from_file(*filenumber*)

extract_collection_from_indices(*indices*)
Extract data from the collection at specific indices, and return it as new collection

clone(*filename*)
Clone the datastructure to a new location

_get_str_mainfile()

get_total_nbr_elements(*count_unsaved=True*)

add_data(*theData*)
Add data to the collection

add_json_data(*theStr*)
Add already deserialized data to the collection

_save_moduletree(*theDict*)

_map_index_to_file(*index*)

_get_json_str_at_index(*index*, *refresh_cache=False*)
Internal method to return the json string at index

reorder(*permutations*)
Reorder collection accordingly to permutations. E.G, list_of_indices = [0,3,2] with collection elems [0,2,1] => collection elems = [0,2,3] :param permutations: :return: /

get_data_at_index(*index*, *ignore_attributes=None*, *none_if_error=False*)
Same as parent, with additional kwargs

Parameters

- **index** –
- **ignore_attributes** – ignore attributes to deserialize (list)
- **none_if_error** –

Returns

save(*filename*)
Save the datastructure to filename

get_data_generator(***kwargs*)

get_nbr_elements()

Returns the number of elements contained inside the structure

set_data_at_index(*data_in*, *index*)
Replace data at specific index

set_data_at_indices(*data_list*, *indices*)
Replace datas at specific indices :param data_list: list of objects to set to the collection, at specific indices :param indices: list of indices :return:

delete_points_at_indices(*indices*)
Delete several elements from the Collection

Parameters **indices** – list of indices to delete

default_palette(*N*)

blackOnly(*N*)

dark2(*N*)

printIfShown(*theStr*, *show_type=SHOW_DEBUG*, *isToPrint=True*, *appendTypeName=True*, *end='n'*)

SHOW_WARNING = 0

convert_color_with_alpha(*color*, *alpha=255*)
Same as meth:*convert_color* but with transparency

class Data(*x: list*, *y: list*, *x_label=''*, *y_label=''*, *legend=''*, *is_scattered=False*, *transfo_x=lambda selfData*, *x: x*, *transfo_y=lambda selfData*, *y: y*, *xlim=None*, *ylim=None*, *permutations=None*, *sort_output=False*, *color=None*, *alpha=255*, *symbol='o'*, *symbolsize=8*, *fillsymbol=True*, *outlinesymbol=1.8*, *linestyle='-'*, *width=2*, *meta=None*)

This class is used to store informations necessary to plot a 2D graph. It has to be combined with a gui to be useful (ex. pyqtgraph)

set_kwargs(*kwargs*)
Set a kwarg after creation of the class

set_data(*x: list*, *y: list*)
Overwrites current datapoints with new set

set_meta(*meta*)
Set associated 'Z' data

get_x()
Get x coordinates of datapoints

get_symbolsize()
Get size of the symbols

symbol_isfilled()
Check if symbols has to be filled or not

get_symbolOutline()
Get color factor of outline of symbols

get_length_data()
Get number of points

get_xlim()
Get x limits of viewbox

get_ylim()
Get y limits of viewbox

get_y()
Get y coordinates of datapoints

get_meta()
Get associated 'Z' data

get_color()
Get color of the line, without transformation

get_color_alpha()
Get color of the line. Return r, g, b in 0, 255 scale

get_alpha()
Get opacity

get_width()
Get width of the line

get_number_of_points()
Get number of points

get_plot_data()
Call this method to get the x and y coordinates of the points that have to be displayed. => After transformation, and after permutations.

Returns x (list), y (list)

get_plot_meta(*x*, *y*)
Call this method to get the z coordinates of the points that been displayed. => After transformation, and after permutations.

Returns z (list)

get_permutations(*x=None*)
Return the transformation 'permutation': xplot[i] = xdata[permutation[i]]

get_invert_permutations()
Return the inverse of permutations: xdata[i] = xplot[revert[i]]

get_dataIndex_from_graphIndex(*index_graph_point*)
From an index given in graph, recovers the index of the data.

Parameters **index_graph_point** – Index in the graph

Returns index of the data

get_dataIndices_from_graphIndices(*index_graph_point_list*)
Same as get_dataIndex_from_graphIndex but with a list in entry. Can (?) improve performances for huge dataset.

Parameters **index_graph_point_list** – List of Index in the graph

Returns List of index of the data

get_graphIndex_from_dataIndex(*index_data*)
From an index given in the data, recovers the index of the graph.

Parameters **index_data** – Index in the data

Returns index of the graph

get_graphIndices_from_dataIndices(*index_data_list*)
Same as get_graphIndex_from_dataIndex but with a list in entry. Can (?) improve performances for huge dataset.

Parameters **index_data_list** – List of Index in the data

Returns List of index of the graph

set_permutations(*permutations*)
Set permutations between datapoints of the trace

Parameters **permutations** – list of indices to plot (example: [0, 2, 1] means that the first point will be plotted, then the third, then the second one)

get_x_label()
Get x label of the trace

get_y_label()
Get y label of the trace

get_legend()
Get name of the trace

get_symbol()
Get symbol

add_point(*x*, *y*)
Add point(s) to trace (inputs can be list or numeral)

delete_point(*index_point*)
Delete a point from the datapoints

isScattered()
Check if plot is scatteded

set_indices_points_to_plot(*indices*)
Set indices points to plot

get_indices_points_to_plot()
Get indices points to plot

get_linestyle()
Get linestyle

__str__()
Return str(self).

export_str()
Method to save the points constituting the trace

set_color(*theColor*)
Set trace color

set_legend(*theLegend*)
Set legend

class Graph
Simple graph container that contains several traces

add_trace(*data*)
Add a trace to the graph

Parameters **data** – *Data*

Returns id of the created trace

remove_trace(*idTrace*)
Delete a trace from the graph

Parameters **idTrace** – id of the trace to delete

get_trace(*idTrace*) → Data
Get data object of idTrace

Parameters **idTrace** – id of the trace to get

Returns *Data*

get_all_traces()
Get all the traces id of the graph

get_all_traces_ids()
Get all the traces id of the graph :return: list of id graphs

export_str()

class Graphs
Contains several *Graph*

updateChildren()

add_trace_firstGraph(*data*, *updateChildren=True*)
Same as add_trace, but only if graphs has only one id :param data: :param updateChildren: :return:

add_trace(*idGraph*, *data*, *updateChildren=True*)
Add a trace to the graph

Parameters

- **idGraph** – id of the graph
- **data** – *Data*
- **updateChildren** – Automatically calls callback functions

Returns id of the created trace

remove_trace(*idGraph*, *idTrace*, *updateChildren=True*)
Remove the trace from the graph

Parameters

- **idGraph** – id of the graph
- **idTrace** – id of the trace to remove
- **updateChildren** – Automatically calls callback functions

get_first_graph()
Get id of the first graph

Returns id of the first graph

get_graph(*idGraph*)
Get graph object at idgraph

Parameters **idGraph** – id of the graph to get

Returns *Graph*

get_all_graphs_ids()
Get all ids of the graphs

Returns list of id graphs

get_all_graphs()
Get all graphs. Return dict {id: *Graph*}

add_graph(*updateChildren=True*)
Add a new graph

Returns id of the created graph

remove_graph(*idGraph*)
Delete a graph

Parameters **idGraph** – id of the graph to delete

add_update_method(*childObject*)
Add a callback each time a graph is modified.

Parameters **childObject** – method without arguments

export_str()
Export all the graphs in text

Returns str

merge(*otherGraphs*)

reset()

is_empty()

griddata_found = True

class Plot3D_Generic(*x_label=''*, *y_label=''*, *z_label=''*, *legend=''*, *x_lim=None*, *y_lim=None*, *z_lim=None*)

get_lim(*axis*)

get_label(*axis*)

get_legend()

class GridPlot_Generic(*X*, *Y*, *Z*, ***kwargs*)
Bases: *Plot3D_Generic*

get_plot_data()

class ContourPlot(**args*, ***kwargs*)
Bases: *GridPlot_Generic*

get_levels()

get_number_of_contours()

class FilledContourPlot(**args*, ***kwargs*)
Bases: *ContourPlot*

class SurfPlot(*X*, *Y*, *Z*, ***kwargs*)
Bases: *GridPlot_Generic*

class MeshPlot(*X*, *Y*, *Z*, ***kwargs*)
Bases: *GridPlot_Generic*

class ScatterPlot3(*x*, *y*, *z*, ***kwargs*)
Bases: *Plot3D_Generic*

get_plot_data()

get_color()

convert_to_gridplot(*x*, *y*, *z*, *x_interval=None*, *y_interval=None*, *n_x=20*, *n_y=20*)
Convert set of points x, y, z to a grid

Parameters

- **x** –
- **y** –
- **z** –
- **x_interval** – [Min, max] of the grid. If none, use min and max values
- **y_interval** – [Min, max] of the grid. If none, use min and max values
- **n_x** – number of points in x direction
- **n_y** – number of points in y direction

Returns X, Y, Z as grid

class HowToPlotGraph(*attribute_x*, *attribute_y*, *kwargs_graph=None*, *check_if_plot_elem=None*, *meta=None*)

__str__()
Return str(self).

class LinkDataGraph

add_collection(*theCollection*, *kwargs=None*)
Add a collection (that will be a future trace)

Parameters

- **theCollection** –
- **kwargs** – kwargs associated with the collection (e.g., color, symbol style, etc.)

Returns unique id associated with the collection

remove_collection(*collectionId*)
Remove collection from the graphs

Parameters **collectionId** – ID of the collection

Returns

set_shadow_collection(*master_collectionId*, *shadow_collection*)
Link a collection to an other

Parameters

- **master_collectionId** – ID of the collection that is displayed in the graph
- **shadow_collection** – collection to link to the master.

Returns

get_graphs()

get_howToPlotGraph(*idGraph*)

add_graph(*howToPlotGraph*)
Add new graph to be plotted.

Parameters **howToPlotGraph** – *HowToPlotGraph*

Returns

get_idCollections()
Get all ids of the plotted collections

get_idGraphs()
Get all ids of the graphs

get_idTraces(*idGraph*)
Get all ids of the traces of graph $idGraph

get_idCollection_from_graph(*idGraph*, *idTrace*)
Get id of collection plotted in graph $idGraph and trace $idTrace

get_collection(*idCollection*, *getShadow=True*)

update_graphs()
Update the graphs: update graphs, traces, and X-Y data

get_collection_from_graph(*idGraph*, *idTrace*, *getShadow=True*) → optimeed.core.ListDataStruct_Interface
From indices in the graph, get corresponding collection

get_clicked_item(*idGraph*, *idTrace*, *idPoint*, *getShadow=True*)
Get the data hidden behind the clicked point

Parameters

- **idGraph** – ID of the graph

- **idTrace** – ID of the trace
- **idPoint** – ID of the point
- **getShadow** – If true, will return the data from the collection linked to the collection that is plotted

Returns Object in collection

get_clicked_items(*idGraph*, *idTrace*, *idPoint_list*, *getShadow=True*)
Same as get_clicked_item, but using a list of points

delete_clicked_item(*idGraph*, *idTrace*, *idPoint*)
Remove item from the collection

delete_clicked_items(*idGraph*, *idTrace*, *idPoints*)
Same, but for a list of points

get_graph_and_trace_from_idCollection(*idCollection*)
Reverse search: from a collection, get all associated graphs

get_idcollection_from_collection(*theCollection*)
Reverse search: from a collection, find its id

get_idPoints_from_indices_in_collection(*idGraph*, *idTrace*, *indices_in_collection*)
From indices in a collection, find the associated idPoints of the graph

class Base_Option(*name*, *based_value*, *choices=None*)

get_value()

get_name()

set_value(*value*)

get_choices()

class Option_bool(*name*, *based_value*, *choices=None*)
Bases: *Base_Option*

name :str

value :bool

set_value(*value*)

get_choices()

class Option_str(*name*, *based_value*, *choices=None*)
Bases: *Base_Option*

name :str

value :str

set_value(*value*)

class Option_int(*name*, *based_value*, *choices=None*)
Bases: *Base_Option*

name :str

value :int

set_value(*value*)

class Option_float(*name*, *based_value*, *choices=None*)
Bases: *Base_Option*

name :str

value :float

set_value(*value*)

class Option_dict(*name*, *based_value*, *choices=None*)
Bases: *Base_Option*

name :str

value :dict

set_value(*value*)

class Option_class

options_bool :Dict[int, Option_bool]

options_str :Dict[int, Option_str]

options_int :Dict[int, Option_int]

options_float :Dict[int, Option_float]

options_dict :Dict[int, Option_dict]

add_option(*idOption*, *theOption*)

get_option_name(*idOption*)

get_option_value(*idOption*)

set_option(*idOption*, *value*)

_pack_options()

__str__()
Return str(self).

has_scipy = True

class fast_LUT_interpolation(*independent_variables*, *dependent_variables*)
Class designed for fast interpolation in look-up table when successive searchs are called often. Otherwise use griddata

interpolate(*point*, *fill_value=np.nan*)
Perform the interpolation :param point: coordinates to interpolate (tuple or list of tuples for multipoints) :param fill_value: value to put if extrapolated. :return: coordinates

interpolate_table(*x0*, *x_values*, *y_values*)
From sorted table (x,y) find y0 corresponding to x0 (linear interpolation)

derivate(*t*, *y*)

linspace(*start*, *stop*, *npoints*)

reconstitute_signal(*amplitudes*, *phases*, *numberOfPeriods=1*, *x_points=None*, *n_points=50*)
Reconstitute the signal from fft. Number of periods of the signal must be specified if different of 1

my_fft(*y*)
Real FFT of signal Bx, with real amplitude of harmonics. Input signal must be within a period.

cart2pol(*x*, *y*)

pol2cart(*rho*, *phi*)

partition(*array*, *begin*, *end*)

quicksort(*array*)

dist(*p*, *q*)
Return the Euclidean distance between points p and q. :param p: [x, y] :param q: [x, y] :return: distance (float)

sparse_subset(*points*, *r*)
Returns a maximal list of elements of points such that no pairs of points in the result have distance less than r. :param points: list of tuples (x,y) :param r: distance :return: corresponding subset (list), indices of the subset (list)

integrate(*x*, *y*)
Performs Integral(x[0] to x[-1]) of y dx

Parameters

- **x** – x axis coordinates (list)
- **y** – y axis coordinates (list)

Returns integral value

my_fourier(*x*, *y*, *n*, *L*)
Fourier analys

Parameters

- **x** – x axis coordinates
- **y** – y axis coordinates
- **n** – number of considered harmonic
- **L** – half-period length

Returns a and b coefficients (y = a*cos(x) + b*sin(y))

get_ellipse_axes(*a*, *b*, *dphi*)
Trouve les longueurs des axes majeurs et mineurs de l'ellipse, ainsi que l'orientation de l'ellipse. ellipse: x(t) = A*cos(t), y(t) = B*cos(t+dphi) Etapes: longueur demi ellipse CENTRéE = sqrt(a^2 cos^2(x) + b^2 cos^2(t+phi) Minimisation de cette formule => obtention formule tg(2x) = alpha/beta

convert_color(*color*)
Convert a color to a tuple if color is a char, otherwise return the tuple.

Parameters **color** – (r,g,b) or char.

Returns

convert_color_with_alpha(*color*, *alpha=255*)
Same as meth:*convert_color* but with transparency

rgetattr(*obj*, *attr*)
getattr, but recursively. Works with list.

rsetattr(*obj*, *attr*, *val*)
setattr, but recursively. Works with list (i.e. theObj.myList[0].var_x)

printIfShown(*theStr*, *show_type=SHOW_DEBUG*, *isToPrint=True*, *appendTypeName=True*, *end='n'*)

SHOW_ERROR = 2

SHOW_WARNING = 0

MODULE_TAG = __module__

CLASS_TAG = __class__

EXCLUDED_TAGS

getExecPath()

class SaveableObject
Abstract class for dynamically type-hinted objects. This class is to solve the special case where the exact type of an attribute is not known before runtime, yet has to be saved.

get_additional_attributes_to_save()
Return list of attributes corresponding to object, whose type cannot be determined statically (e.g. topology change)

get_additional_attributes_to_save_list()
Same behavior as get_additional_attributes_to_save, but where the attributes contains list of unknown items

_isclass(*theObject*)
Extends the default isclass method with typing

get_type_class(*typ*)
Get the type of the class. used to compare objects from Typing.

_get_object_class(*theObj*)

_get_object_module(*theObj*)

_object_to_FQCN(*theobj*)
Gets module path of object

_find_class(*moduleName*, *className*)

json_to_obj(*json_dict*)
Convenience class to create object from dictionary. Only works if CLASS_TAG is valid

Parameters **json_dict** – dictionary loaded from a json file.

Raises

- **TypeError** – if class can not be found
- **KeyError** – if CLASS_TAG not present in dictionary

json_to_obj_safe(*json_dict*, *cls*)
Safe class to create object from dictionary.

Parameters

- **json_dict** – dictionary loaded from a json file
- **cls** – class object to instantiate with dictionary

_instantiates_annotated_object(*_json_dict*, *_cls*)

_get_annotations(*theObj*)
Return annotated attributes (theObj being the type of the object)

obj_to_json(*theObj*)
Extract the json dictionary from the object. The data saved are automatically detected, using typehints. ex: x: int=5 will be saved, x=5 won't. Inheritance of annotation is managed by this function

_get_attributes_to_save(*theObj*)
Return list (attribute, is_first)

get_json_module_tree_from_dict (*jsonDict*)
Return dict containing {CLASS_TAG: "class_name", MODULE_TAG: "module_name", "attribute1":{"class_name": "module_name", ... }}

remove_module_tree_from_string (*theStr*)
Used to compress string by removing __module__ and __class__ entries (used with get_json_module_tree_from_dict)

apply_module_tree_to_dict (*nestedTree*, *nestedObject*, *raiseError=False*)
Restore __module__ and __class__ entries from nestedTree in nestedDict

encode_str_json (*theStr*)

decode_str_json (*theStr*)

export_to_tikz_groupGraphs (*theGraphs: optimeed.core.graphs.Graphs*, *foldername*, *additionalPreamble=lambda: ''*, *additionalAxisOptions=lambda graphId: ''*, *additionalTraceOptions=lambda graphId*, *traceId: ''*, *debug=False*)
Export the graphs as group

Parameters

- **theGraphs** – Graphs to save
- **foldername** – Foldername to save
- **additionalPreamble** – method that returns string for custom tikz options
- **additionalAxisOptions** – method that returns string for custom tikz options
- **additionalTraceOptions** – method that returns string for custom tikz options

Returns

export_to_tikz_contour_plot (*list_of_traces3*, *foldername*, *filename_data='data'*)
Export the graphs as group

Parameters

- **list_of_traces3** – List of 3D traces
- **foldername** – Foldername to save
- **filename_data** – filename of the data

Returns

printIfShown (*theStr*, *show_type=SHOW_DEBUG*, *isToPrint=True*, *appendTypeName=True*, *end='n'*)

SHOW_WARNING = 0

get_path_to_inkscape ()

get_inkscape_version ()

inkscape_version

inkscape_svg_to_pdf (*filename_svg*, *filename_pdf*)

inkscape_svg_to_png (*filename_svg*, *filename_png*)

optimize

Subpackages

characterization

characterization

Module Contents

Classes

class Characterization
Bases: *optimeed.optimize.characterization.interfaceCharacterization.InterfaceCharacterization*

Interface for the evaluation of a device

compute(*theDevice*)
Action to perform to characterize (= compute the objective function) of the device.

Parameters **theDevice** – the device to characterize

interfaceCharacterization

Module Contents

Classes

class InterfaceCharacterization
Interface for the evaluation of a device

__str__()
Return str(self).

Package Contents

Classes

class InterfaceCharacterization
Interface for the evaluation of a device

__str__()
Return str(self).

class Characterization
Bases: *optimeed.optimize.characterization.interfaceCharacterization.InterfaceCharacterization*

Interface for the evaluation of a device

compute(*theDevice*)
Action to perform to characterize (= compute the objective function) of the device.

Parameters **theDevice** – the device to characterize

mathsToPhysics

interfaceMathsToPhysics

Module Contents

Classes

class InterfaceMathsToPhysics
Interface to transform output from the optimizer to meaningful variables of the device

mathsToPhysics

Module Contents

Classes

class MathsToPhysics
Bases: *optimeed.optimize.mathsToPhysics.interfaceMathsToPhysics.InterfaceMathsToPhysics*

Dummy yet powerful example of maths to physics. The optimization variables are directly injected to the device

fromMathsToPhys(*xVector*, *theDevice*, *theOptimizationVariables*)
Transforms an input vector coming from the optimization (e.g. [0.23, 4, False]) to "meaningful" variable (ex: length, number of poles, flag).

Parameters

- **xVector** – List of optimization variables from the optimizer
- **theDevice** – `InterfaceDevice`
- **opti_variables** – list of `OptimizationVariable`

fromPhysToMaths(*theDevice*, *theOptimizationVariables*)
Extracts a mathematical vector from meaningful variable of the Device

Parameters

- **theDevice** – `InterfaceDevice`
- **opti_variables** – list of `OptimizationVariable`

Returns List of optimization variables

__str__()
Return str(self).

Package Contents

Classes

class MathsToPhysics
Bases: *optimeed.optimize.mathsToPhysics.interfaceMathsToPhysics.InterfaceMathsToPhysics*

Dummy yet powerful example of maths to physics. The optimization variables are directly injected to the device

fromMathsToPhys(*xVector*, *theDevice*, *theOptimizationVariables*)
Transforms an input vector coming from the optimization (e.g. [0.23, 4, False]) to "meaningful" variable (ex: length, number of poles, flag).

Parameters

- **xVector** – List of optimization variables from the optimizer
- **theDevice** – `InterfaceDevice`
- **opti_variables** – list of `OptimizationVariable`

fromPhysToMaths(*theDevice*, *theOptimizationVariables*)
Extracts a mathematical vector from meaningful variable of the Device

Parameters

- **theDevice** – `InterfaceDevice`
- **opti_variables** – list of `OptimizationVariable`

Returns List of optimization variables

__str__()
Return str(self).

class InterfaceMathsToPhysics
Interface to transform output from the optimizer to meaningful variables of the device

objAndCons

fastObjCons

Module Contents

Classes

class FastObjCons(*constraintEquation*, *name=None*)
Bases: *optimeed.optimize.objAndCons.interfaceObjCons.InterfaceObjCons*

Convenience class to create an objective or a constraint very fast.

compute(*theDevice*)
Get the value of the objective or the constraint. The objective is to MINIMIZE and the constraint has to respect VALUE <= 0

Parameters **theDevice** – Input device that has already been evaluated

Returns float.

get_name()

interfaceObjCons

Module Contents

Classes

class InterfaceObjCons
Interface class for objectives and constraints. The objective is to MINIMIZE and the constraint has to respect VALUE <= 0

get_name()

__str__()
Return str(self).

Package Contents

Classes

class FastObjCons(*constraintEquation*, *name=None*)
Bases: *optimeed.optimize.objAndCons.interfaceObjCons.InterfaceObjCons*

Convenience class to create an objective or a constraint very fast.

compute(*theDevice*)
Get the value of the objective or the constraint. The objective is to MINIMIZE and the constraint has to respect VALUE <= 0

Parameters **theDevice** – Input device that has already been evaluated

Returns float.

get_name()

class InterfaceObjCons
Interface class for objectives and constraints. The objective is to MINIMIZE and the constraint has to respect VALUE <= 0

get_name()

__str__()
Return str(self).

optiAlgorithms

Subpackages

convergence

evolutionaryConvergence

Module Contents

Classes

class EvolutionaryConvergence

Bases: *optimeed.optimize.optiAlgorithms.convergence.interfaceConvergence.InterfaceConvergence*

convergence class for population-based algorithm

objectives_per_step :Dict[int, List[List[float]]]

constraints_per_step :Dict[int, List[List[float]]]

paretos_per_step :Dict[int, List[List[float]]]

hypervolume_per_step :Dict[int, List[float]]

set_curr_step(*theObjectives_list*, *theConstraints_list*)

_extract_N_steps(*N*)

get_pareto_convergence(*max_number_of_points=None*)

get_pareto_at_step(*step*)

get_hypervolume(*pareto*, *refPoint=None*)

get_hypervolume_convergence(*max_number_of_points*)

get_nadir_point(*pareto*)

last_step()

get_nb_objectives()

get_scalar_convergence_evolution(*max_number_of_points*)

get_graphs(*max_number_of_points=None*)

Return *Graphs*

hypervolume

Module Contents

Classes

Attributes

__author__ = Simon Wessing

class HyperVolume(*referencePoint*)

Hypervolume computation based on variant 3 of the algorithm in the paper: C. M. Fonseca, L. Paquete, and M. Lopez-Ibanez. An improved dimension-sweep algorithm for the hypervolume indicator. In IEEE Congress on Evolutionary Computation, pages 1157-1163, Vancouver, Canada, July 2006.

Minimization is implicitly assumed here!

compute(*front*)

Returns the hypervolume that is dominated by a non-dominated front.

Before the HV computation, front and reference point are translated, so that the reference point is [0, . . . , 0].

hvRecursive(*dimIndex*, *length*, *bounds*)
Recursive call to hypervolume calculation.

In contrast to the paper, the code assumes that the reference point is [0, . . . , 0]. This allows the avoidance of a few operations.

preProcess(*front*)
Sets up the list data structure needed for calculation.

sortByDimension(*nodes*, *i*)
Sorts the list of nodes by the i-th value of the contained points.

class MultiList(*numberLists*)
A special data structure needed by FonsecaHyperVolume.

It consists of several doubly linked lists that share common nodes. So, every node has multiple predecessors and successors, one in every list.

class Node(*numberLists*, *cargo=None*)

__str__()
Return str(self).

__str__()
Return str(self).

__len__()
Returns the number of lists that are included in this MultiList.

getLength(*i*)
Returns the length of the i-th list.

append(*node*, *index*)
Appends a node to the end of the list at the given index.

extend(*nodes*, *index*)
Extends the list at the given index with the nodes.

remove(*node*, *index*, *bounds*)
Removes and returns 'node' from all lists in [0, 'index'[.

reinsert(*node*, *index*, *bounds*)
Inserts 'node' at the position it had in all lists in [0, 'index'[before it was removed. This method assumes that the next and previous nodes of the node that is reinserted are in the list.

interfaceConvergence

Module Contents

Classes

class InterfaceConvergence
Simple interface to visually get the convergence of any optimization problem

Package Contents

Classes

class EvolutionaryConvergence

Bases: *optimeed.optimize.optiAlgorithms.convergence.interfaceConvergence.InterfaceConvergence*

convergence class for population-based algorithm

objectives_per_step :Dict[int, List[List[float]]]

constraints_per_step :Dict[int, List[List[float]]]

paretos_per_step :Dict[int, List[List[float]]]

hypervolume_per_step :Dict[int, List[float]]

set_curr_step(*theObjectives_list*, *theConstraints_list*)

_extract_N_steps(*N*)

get_pareto_convergence(*max_number_of_points=None*)

get_pareto_at_step(*step*)

get_hypervolume(*pareto*, *refPoint=None*)

get_hypervolume_convergence(*max_number_of_points*)

get_nadir_point(*pareto*)

last_step()

get_nb_objectives()

get_scalar_convergence_evolution(*max_number_of_points*)

get_graphs(*max_number_of_points=None*)

Return *Graphs*

class InterfaceConvergence

Simple interface to visually get the convergence of any optimization problem

pyswarm

pso

Module Contents

Classes

Functions

_is_feasible(*theList*)

_format_fx_fs(*objectives_pop*, *constraints_pop*)

class MyMapEvaluator(*evaluation_function*, *callback_on_evaluation*)

evaluate_all(*x*)

class MyMultiprocessEvaluator(*evaluation_function*, *callback_on_evaluation*, *numberOfCores*)

evaluate_all(*x*)

pso(*lb*, *ub*, *initialVectorGuess*, *theEvaluator*, *maxtime*, *callback_generation=lambda objectives*, *constraints: None*, *swarmsize=100*, *omega=0.5*, *phip=0.5*, *phig=0.5*)
Perform a particle swarm optimization (PSO)

lb: list Lower bounds of each parameter

ub: list upper bounds of each parameter

initialVectorGuess: list initial vector guess for the solution (to be included inside population)

theEvaluator : object define before maxtime : float

The maximum time (in s) before stopping the algorithm

callback_generation: function lambda (bjectives (as list), constraints (as list)) per step Useful to log convergence

swarmsize [int] The number of particles in the swarm (Default: 100)

omega [scalar] Particle velocity scaling factor (Default: 0.5)

phip [scalar] Scaling factor to search away from the particle's best known position (Default: 0.5)

phig [scalar] Scaling factor to search away from the swarm's best known position (Default: 0.5)

g [array] The swarm's best known position (optimal design)

f [scalar] The objective value at `g`

Package Contents

Classes

Functions

_is_feasible(*theList*)

_format_fx_fs(*objectives_pop*, *constraints_pop*)

class MyMapEvaluator(*evaluation_function*, *callback_on_evaluation*)

evaluate_all(*x*)

class MyMultiprocessEvaluator(*evaluation_function*, *callback_on_evaluation*, *numberOfCores*)

evaluate_all(*x*)

pso(*lb*, *ub*, *initialVectorGuess*, *theEvaluator*, *maxtime*, *callback_generation=lambda objectives*, *constraints: None*, *swarmsize=100*, *omega=0.5*, *phip=0.5*, *phig=0.5*)
Perform a particle swarm optimization (PSO)

lb: list Lower bounds of each parameter

ub: list upper bounds of each parameter

initialVectorGuess: list initial vector guess for the solution (to be included inside population)

theEvaluator : object define before maxtime : float

The maximum time (in s) before stopping the algorithm

callback_generation: function lambda (bjectives (as list), constraints (as list)) per step Useful to log convergence

swarmsize [int] The number of particles in the swarm (Default: 100)

omega [scalar] Particle velocity scaling factor (Default: 0.5)

phip [scalar] Scaling factor to search away from the particle's best known position (Default: 0.5)

phig [scalar] Scaling factor to search away from the swarm's best known position (Default: 0.5)

g [array] The swarm's best known position (optimal design)

f [scalar] The objective value at `g`

NLOpt_Algorithm

Module Contents

Classes

class ConvergenceManager

add_point(*newObj*)

set_pop_size(*popSize*)

class NLOpt_Algorithm

Bases: *optimeed.optimize.optiAlgorithms.algorithmInterface.AlgorithmInterface*, *optimeed.core.Option_class*

Interface for the optimization algorithm

ALGORITHM = 0

POPULATION_SIZE = 1

initialize(*initialVectorGuess*, *listOfOptimizationVariables*)

This function is called once parameters can't be changed anymore, before "get_convergence".

Parameters

- **initialVectorGuess** – list of variables that describe the initial individual
- **listOfOptimizationVariables** – list of *optimeed.optimize.optiVariable.OptimizationVariable*

Returns

compute()

Launch the optimization

Returns vector of optimal variables

set_evaluationFunction(*evaluationFunction*, *callback_on_evaluate*, *numberOfObjectives*, *_numberOfConstraints*)

Set the evaluation function and all the necessary callbacks

Parameters

- **evaluationFunction** – check `evaluateObjectiveAndConstraints()`
- **callback_on_evaluation** – check `callback_on_evaluation()`. Call this function after performing the evaluation of the individuals
- **numberOfObjectives** – int, number of objectives
- **numberOfConstraints** – int, number of constraints

set_maxtime(*maxTime*)
Set maximum optimization time (in seconds)

__str__()
Return str(self).

get_convergence()
Get the convergence of the optimization

Returns *InterfaceConvergence*

algorithmInterface

Module Contents

Classes

class AlgorithmInterface
Interface for the optimization algorithm

reset()

monobjective_PSO

Module Contents

Classes

class Monobjective_PSO
Bases: *optimeed.optimize.optiAlgorithms.algorithmInterface.AlgorithmInterface*, *optimeed.core.Option_class*

Interface for the optimization algorithm

NUMBER_OF_CORES = 1

initialize(*initialVectorGuess*, *listOfOptimizationVariables*)
This function is called once parameters can't be changed anymore, before "get_convergence".

Parameters

- **initialVectorGuess** – list of variables that describe the initial individual
- **listOfOptimizationVariables** – list of *optimeed.optimize.optiVariable.OptimizationVariable*

Returns

compute()
Launch the optimization

Returns vector of optimal variables

set_evaluationFunction(*evaluationFunction*, *callback_on_evaluate*, *numberOfObjectives*, *_numberOfConstraints*)
Set the evaluation function and all the necessary callbacks

Parameters

- **evaluationFunction** – check `evaluateObjectiveAndConstraints()`
- **callback_on_evaluation** – check `callback_on_evaluation()`. Call this function after performing the evaluation of the individuals
- **numberOfObjectives** – int, number of objectives
- **numberOfConstraints** – int, number of constraints

set_maxtime(*maxTime*)
Set maximum optimization time (in seconds)

__str__()
Return str(self).

get_convergence()
Get the convergence of the optimization

Returns *InterfaceConvergence*

multiObjective_GA

Module Contents

Classes

class MyProblem(*theOptimizationVariables*, *nbr_objectives*, *nbr_constraints*, *evaluationFunction*)
Bases: `optimeed.optimize.optiAlgorithms.platypus.core.Problem`

Automatically sets the optimization problem

evaluate(*solution*)
Evaluates the problem.

By default, this method calls the function passed to the constructor. Alternatively, a problem can subclass and override this method. When overriding, this method is responsible for updating the objectives and constraints stored in the solution.

solution: Solution The solution to evaluate.

class MyGenerator(*initialVectorGuess*)
Bases: `optimeed.optimize.optiAlgorithms.platypus.Generator`

Population generator to insert initial individual

generate(*problem*)

class MaxTimeTerminationCondition(*maxTime*)
Bases: `optimeed.optimize.optiAlgorithms.platypus.core.TerminationCondition`

Abstract class for defining termination conditions.

initialize(*algorithm*)
Initializes this termination condition.

This method is used to collect any initial state, such as the current NFE or current time, needed for calculating the termination criteria.

algorithm [Algorithm] The algorithm being run.

shouldTerminate(*algorithm*)
Checks if the algorithm should terminate.

Check the termination condition, returning True if the termination condition is satisfied; False otherwise. This method is called after each iteration of the algorithm.

algorithm [Algorithm] The algorithm being run.

class ConvergenceTerminationCondition(*minrelchange_percent=0.1*, *nb_generation=15*)
Bases: `optimeed.optimize.optiAlgorithms.platypus.core.TerminationCondition`

Abstract class for defining termination conditions.

initialize(*algorithm*)
Initializes this termination condition.

This method is used to collect any initial state, such as the current NFE or current time, needed for calculating the termination criteria.

algorithm [Algorithm] The algorithm being run.

shouldTerminate(*algorithm*)
Checks if the algorithm should terminate.

Check the termination condition, returning True if the termination condition is satisfied; False otherwise. This method is called after each iteration of the algorithm.

algorithm [Algorithm] The algorithm being run.

class SeveralTerminationCondition
Bases: `optimeed.optimize.optiAlgorithms.platypus.core.TerminationCondition`

Abstract class for defining termination conditions.

initialize(*algorithm*)
Initializes this termination condition.

This method is used to collect any initial state, such as the current NFE or current time, needed for calculating the termination criteria.

algorithm [Algorithm] The algorithm being run.

add(*theTerminationCondition*)

shouldTerminate(*algorithm*)
Checks if the algorithm should terminate.

Check the termination condition, returning True if the termination condition is satisfied; False otherwise. This method is called after each iteration of the algorithm.

algorithm [Algorithm] The algorithm being run.

class MyMapEvaluator(*callback_on_evaluation*)
Bases: `optimeed.optimize.optiAlgorithms.platypus.evaluator.Evaluator`

evaluate_all(*jobs*, ***kwargs*)

class MyMultiprocessEvaluator(*callback_on_evaluation*, *numberOfCores*)
Bases: optimeed.optimize.optiAlgorithms.platypus.evaluator.Evaluator

my_callback(*output*)

evaluate_all(*jobs*, ***kwargs*)

close()

class MultiObjective_GA
Bases: *optimeed.optimize.optiAlgorithms.algorithmInterface.AlgorithmInterface*, *optimeed.core.Option_class*

Based on Platypus Library. Workflow: Define what to optimize and which function to call with a Problem Define the initial population with a Generator Define the algorithm. As options, define how to evaluate the elements with a Evaluator, i.e., for multiprocessing. Define what is the termination condition of the algorithm with TerminationCondition. Here, termination condition is a maximum time.

DIVISION_OUTER = 0

OPTI_ALGORITHM = 1

NUMBER_OF_CORES = 2

KWARGS_ALGO = 3

initialize(*initialVectorGuess*, *listOfOptimizationVariables*)
This function is called just before running optimization algorithm.

compute()
Launch the optimization

Returns vector of optimal variables

set_evaluationFunction(*evaluationFunction*, *callback_on_evaluation*, *numberOfObjectives*, *numberOfConstraints*)
Set the evaluation function and all the necessary callbacks

Parameters

- **evaluationFunction** – check evaluateObjectiveAndConstraints()
- **callback_on_evaluation** – check callback_on_evaluation(). Call this function after performing the evaluation of the individuals
- **numberOfObjectives** – int, number of objectives
- **numberOfConstraints** – int, number of constraints

set_maxtime(*maxTime*)
Set maximum optimization time (in seconds)

__str__()
Return str(self).

get_convergence()
This function is called just before compute. Because the convergence is contained in opti algorithm, it must be created now.

add_terminationCondition(*theTerminationCondition*)

reset()

Package Contents

Classes

class MultiObjective_GA

Bases: *optimeed.optimize.optiAlgorithms.algorithmInterface.AlgorithmInterface*, *optimeed.core.Option_class*

Based on Platypus Library. Workflow: Define what to optimize and which function to call with a `Problem` Define the initial population with a `Generator` Define the algorithm. As options, define how to evaluate the elements with a `Evaluator`, i.e., for multiprocessing. Define what is the termination condition of the algorithm with `TerminationCondition`. Here, termination condition is a maximum time.

DIVISION_OUTER = 0

OPTI_ALGORITHM = 1

NUMBER_OF_CORES = 2

KWARGS_ALGO = 3

initialize(*initialVectorGuess*, *listOfOptimizationVariables*)

This function is called just before running optimization algorithm.

compute()

Launch the optimization

Returns vector of optimal variables

set_evaluationFunction(*evaluationFunction*, *callback_on_evaluation*, *numberOfObjectives*, *numberOfConstraints*)

Set the evaluation function and all the necessary callbacks

Parameters

- **evaluationFunction** – check `evaluateObjectiveAndConstraints()`
- **callback_on_evaluation** – check `callback_on_evaluation()`. Call this function after performing the evaluation of the individuals
- **numberOfObjectives** – int, number of objectives
- **numberOfConstraints** – int, number of constraints

set_maxtime(*maxTime*)

Set maximum optimization time (in seconds)

__str__()

Return str(self).

get_convergence()

This function is called just before compute. Because the convergence is contained in opti algorithm, it must be created now.

add_terminationCondition(*theTerminationCondition*)

reset()

class Monobjective_PSO

Bases: *optimeed.optimize.optiAlgorithms.algorithmInterface.AlgorithmInterface*, *optimeed.core.Option_class*

Interface for the optimization algorithm

NUMBER_OF_CORES = 1

initialize(*initialVectorGuess*, *listOfOptimizationVariables*)
This function is called once parameters can't be changed anymore, before "get_convergence".

Parameters

- **initialVectorGuess** – list of variables that describe the initial individual
- **listOfOptimizationVariables** – list of *optimeed.optimize.optiVariable.OptimizationVariable*

Returns

compute()
Launch the optimization

Returns vector of optimal variables

set_evaluationFunction(*evaluationFunction*, *callback_on_evaluate*, *numberOfObjectives*, *_numberOfConstraints*)
Set the evaluation function and all the necessary callbacks

Parameters

- **evaluationFunction** – check `evaluateObjectiveAndConstraints()`
- **callback_on_evaluation** – check `callback_on_evaluation()`. Call this function after performing the evaluation of the individuals
- **numberOfObjectives** – int, number of objectives
- **numberOfConstraints** – int, number of constraints

set_maxtime(*maxTime*)
Set maximum optimization time (in seconds)

__str__()
Return str(self).

get_convergence()
Get the convergence of the optimization

Returns *InterfaceConvergence*

optiHistoric

Module Contents

Classes

class OptiHistoric(*optiname='opti'*, *autosave_timer=60 * 5*, *autosave=True*, *create_new_directory=True*, *performance_datastruct=True*)
Contains all the points that have been evaluated

class _pointData(*currTime*, *objectives*, *constraints*)

time :float

objectives :List[float]

constraints :List[float]

class _LogParams

add_parameters(*params*)

get_rows_indices(*list_of_params*)

log_after_evaluation(*returned_values: dict*)
Save the output of evaluate to optiHistoric. This function should be called by the optimizer IN a process safe context.

set_results(*devicesList*)

get_best_devices_without_reevaluating(*list_of_best_params*)

set_convergence(*theConvergence*)

save()

get_convergence()

Returns convergence *InterfaceConvergence*

get_devices()

Returns List of devices (ordered by evaluation number)

get_logopti()

Returns Log optimization (to check the convergence)

start(*optimization_parameters*)
Function called upon starting the optimization. Create folders.

optiVariable

Module Contents

Classes

class OptimizationVariable(*attributeName*)
Contains information about the optimization of a variable

attributeName :str

get_attribute_name()
Return the attribute to set

add_prefix_attribute_name(*thePrefix*)
Used for nested object, lower the name by prefix. Example: R_ext becomes (thePrefix).R_ext

get_PhysToMaths(*deviceIn*)
Convert the initial value of the variable contained in the device to optimization variable value

Parameters **deviceIn** – InterfaceDevice

Returns value of the corresponding optimization variable

do_MathsToPhys(*variableValue*, *deviceIn*)
Apply the value to the device

__str__()
Return str(self).

class Real_OptimizationVariable(*attributeName*, *val_min*, *val_max*)
Bases: *OptimizationVariable*

Real (continuous) optimization variable. Most used type

val_min :float

val_max :float

get_min_value()

get_max_value()

get_PhysToMaths(*deviceIn*)
Convert the initial value of the variable contained in the device to optimization variable value

Parameters **deviceIn** – InterfaceDevice

Returns value of the corresponding optimization variable

do_MathsToPhys(*value*, *deviceIn*)
Apply the value to the device

__str__()
Return str(self).

class Binary_OptimizationVariable(*attributeName*)
Bases: *OptimizationVariable*

Boolean (True/False) optimization variable.

get_PhysToMaths(*deviceIn*)
Convert the initial value of the variable contained in the device to optimization variable value

Parameters **deviceIn** – InterfaceDevice

Returns value of the corresponding optimization variable

do_MathsToPhys(*value*, *deviceIn*)
Apply the value to the device

__str__()
Return str(self).

class Integer_OptimizationVariable(*attributeName*, *val_min*, *val_max*)
Bases: *OptimizationVariable*

Integer variable, in [min_value, max_value]

val_min :int

val_max :int

get_min_value()

get_max_value()

get_PhysToMaths(*deviceIn*)
Convert the initial value of the variable contained in the device to optimization variable value

Parameters **deviceIn** – InterfaceDevice

Returns value of the corresponding optimization variable

do_MathsToPhys(*value*, *deviceIn*)
Apply the value to the device

__str__()
Return str(self).

optimizer

Module Contents

Classes

Functions

Attributes

default

class OptimizerSettings(*theDevice*, *theObjectives*, *theConstraints*, *theOptimizationVariables*, *theOptimizationAlgorithm=None*, *theMathsToPhysics=None*, *theCharacterization=None*)
Bases: *optimeed.core.SaveableObject*

Abstract class for dynamically type-hinted objects. This class is to solve the special case where the exact type of an attribute is not known before runtime, yet has to be saved.

get_additional_attributes_to_save()
Return list of attributes corresponding to object, whose type cannot be determined statically (e.g. topology change)

get_additional_attributes_to_save_list()
Same behavior as get_additional_attributes_to_save, but where the attributes contains list of unknown items

get_device()

get_M2P()

get_charac()

get_optivariables()

get_objectives()

get_constraints()

get_optialgorithm()

class _Evaluator(*optimization_parameters: OptimizerSettings*)
This is the main class that serves as evaluator. This class is NOT process safe (i.e., copy of it might be generated upon process call)

start()

evaluate(*x*)
Evaluates the performances of device associated to entrance vector x. Outputs the objective function and the constraints, and other data used in optiHistoric.

This function is NOT process safe: "self." is a FORK in multiprocessing algorithms. It means that the motor originally contained in self. is modified only in the fork, and only gathered by reaching the end of the fork.

Parameters **x** – Input mathematical vector from optimization algorithm

Returns dictionary, containing objective values (list of scalar), constraint values (list of scalar), and other info (motor, time)

reevaluate_solutions(*x_solutions*)

run_optimization(*optimization_parameters: OptimizerSettings*, *opti_historic*, *max_opti_time_sec=10*)
Perform the optimization.

Returns list of the best optimized devices, convergence information

Package Contents

Classes

Functions

class InterfaceCharacterization
Interface for the evaluation of a device

__str__()
Return str(self).

class Characterization
Bases: *optimeed.optimize.characterization.interfaceCharacterization.InterfaceCharacterization*

Interface for the evaluation of a device

compute(*theDevice*)
Action to perform to characterize (= compute the objective function) of the device.

Parameters **theDevice** – the device to characterize

class MathsToPhysics
Bases: *optimeed.optimize.mathsToPhysics.interfaceMathsToPhysics.InterfaceMathsToPhysics*

Dummy yet powerful example of maths to physics. The optimization variables are directly injected to the device

fromMathsToPhys(*xVector*, *theDevice*, *theOptimizationVariables*)
Transforms an input vector coming from the optimization (e.g. [0.23, 4, False]) to "meaningful" variable (ex: length, number of poles, flag).

Parameters

- **xVector** – List of optimization variables from the optimizer
- **theDevice** – `InterfaceDevice`
- **opti_variables** – list of `OptimizationVariable`

fromPhysToMaths(*theDevice*, *theOptimizationVariables*)
Extracts a mathematical vector from meaningful variable of the Device

Parameters

- **theDevice** – `InterfaceDevice`
- **opti_variables** – list of `OptimizationVariable`

Returns List of optimization variables

__str__()
Return str(self).

class InterfaceMathsToPhysics
Interface to transform output from the optimizer to meaningful variables of the device

class FastObjCons(*constraintEquation*, *name=None*)
Bases: *optimeed.optimize.objAndCons.interfaceObjCons.InterfaceObjCons*

Convenience class to create an objective or a constraint very fast.

compute(*theDevice*)
Get the value of the objective or the constraint. The objective is to MINIMIZE and the constraint has to respect VALUE <= 0

Parameters **theDevice** – Input device that has already been evaluated

Returns float.

get_name()

class InterfaceObjCons
Interface class for objectives and constraints. The objective is to MINIMIZE and the constraint has to respect VALUE <= 0

get_name()

__str__()
Return str(self).

class MultiObjective_GA
Bases: *optimeed.optimize.optiAlgorithms.algorithmInterface.AlgorithmInterface*, *optimeed.core.Option_class*

Based on Platypus Library. Workflow: Define what to optimize and which function to call with a `Problem` Define the initial population with a `Generator` Define the algorithm. As options, define how to evaluate the elements with a `Evaluator`, i.e., for multiprocessing. Define what is the termination condition of the algorithm with `TerminationCondition`. Here, termination condition is a maximum time.

DIVISION_OUTER = 0

OPTI_ALGORITHM = 1

NUMBER_OF_CORES = 2

KWARGS_ALGO = 3

initialize(*initialVectorGuess*, *listOfOptimizationVariables*)
This function is called just before running optimization algorithm.

compute()
Launch the optimization

Returns vector of optimal variables

set_evaluationFunction(*evaluationFunction*, *callback_on_evaluation*, *numberOfObjectives*, *numberOfConstraints*)
Set the evaluation function and all the necessary callbacks

Parameters

- **evaluationFunction** – check `evaluateObjectiveAndConstraints()`
- **callback_on_evaluation** – check `callback_on_evaluation()`. Call this function after performing the evaluation of the individuals

- **numberOfObjectives** – int, number of objectives
- **numberOfConstraints** – int, number of constraints

set_maxtime(*maxTime*)
Set maximum optimization time (in seconds)

__str__()
Return str(self).

get_convergence()
This function is called just before compute. Because the convergence is contained in opti algorithm, it must be created now.

add_terminationCondition(*theTerminationCondition*)

reset()

class Monobjective_PSO
Bases: *optimeed.optimize.optiAlgorithms.algorithmInterface.AlgorithmInterface*, *optimeed.core.Option_class*

Interface for the optimization algorithm

NUMBER_OF_CORES = 1

initialize(*initialVectorGuess*, *listOfOptimizationVariables*)
This function is called once parameters can't be changed anymore, before "get_convergence".

Parameters

- **initialVectorGuess** – list of variables that describe the initial individual
- **listOfOptimizationVariables** – list of *optimeed.optimize.optiVariable.OptimizationVariable*

Returns

compute()
Launch the optimization

Returns vector of optimal variables

set_evaluationFunction(*evaluationFunction*, *callback_on_evaluate*, *numberOfObjectives*, *_numberOfConstraints*)
Set the evaluation function and all the necessary callbacks

Parameters

- **evaluationFunction** – check `evaluateObjectiveAndConstraints()`
- **callback_on_evaluation** – check `callback_on_evaluation()`. Call this function after performing the evaluation of the individuals
- **numberOfObjectives** – int, number of objectives
- **numberOfConstraints** – int, number of constraints

set_maxtime(*maxTime*)
Set maximum optimization time (in seconds)

__str__()
Return str(self).

get_convergence()
Get the convergence of the optimization

Returns *InterfaceConvergence*

class Real_OptimizationVariable(*attributeName*, *val_min*, *val_max*)
Bases: OptimizationVariable

Real (continuous) optimization variable. Most used type

val_min :float

val_max :float

get_min_value()

get_max_value()

get_PhysToMaths(*deviceIn*)
Convert the initial value of the variable contained in the device to optimization variable value

Parameters **deviceIn** – InterfaceDevice

Returns value of the corresponding optimization variable

do_MathsToPhys(*value*, *deviceIn*)
Apply the value to the device

__str__()
Return str(self).

class Binary_OptimizationVariable(*attributeName*)
Bases: OptimizationVariable

Boolean (True/False) optimization variable.

get_PhysToMaths(*deviceIn*)
Convert the initial value of the variable contained in the device to optimization variable value

Parameters **deviceIn** – InterfaceDevice

Returns value of the corresponding optimization variable

do_MathsToPhys(*value*, *deviceIn*)
Apply the value to the device

__str__()
Return str(self).

class Integer_OptimizationVariable(*attributeName*, *val_min*, *val_max*)
Bases: OptimizationVariable

Integer variable, in [min_value, max_value]

val_min :int

val_max :int

get_min_value()

get_max_value()

get_PhysToMaths(*deviceIn*)
Convert the initial value of the variable contained in the device to optimization variable value

Parameters **deviceIn** – InterfaceDevice

Returns value of the corresponding optimization variable

do_MathsToPhys(*value*, *deviceIn*)
Apply the value to the device

__str__()
Return str(self).

run_optimization(*optimization_parameters: OptimizerSettings*, *opti_historic*, *max_opti_time_sec=10*)
Perform the optimization.

Returns list of the best optimized devices, convergence information

class OptimizerSettings(*theDevice*, *theObjectives*, *theConstraints*, *theOptimizationVariables*, *theOptimizationAlgorithm=None*, *theMathsToPhysics=None*, *theCharacterization=None*)
Bases: optimeed.core.SaveableObject

Abstract class for dynamically type-hinted objects. This class is to solve the special case where the exact type of an attribute is not known before runtime, yet has to be saved.

get_additional_attributes_to_save()
Return list of attributes corresponding to object, whose type cannot be determined statically (e.g. topology change)

get_additional_attributes_to_save_list()
Same behavior as get_additional_attributes_to_save, but where the attributes contains list of unknown items

get_device()

get_M2P()

get_charac()

get_optivariables()

get_objectives()

get_constraints()

get_optialgorithm()

class OptiHistoric(*optiname='opti'*, *autosave_timer=60 * 5*, *autosave=True*, *create_new_directory=True*, *performance_datastruct=True*)
Contains all the points that have been evaluated

class _pointData(*currTime*, *objectives*, *constraints*)

time :float

objectives :List[float]

constraints :List[float]

class _LogParams

add_parameters(*params*)

get_rows_indices(*list_of_params*)

log_after_evaluation(*returned_values: dict*)
Save the output of evaluate to optiHistoric. This function should be called by the optimizer IN a process safe context.

set_results(*devicesList*)

get_best_devices_without_reevaluating(*list_of_best_params*)

set_convergence(*theConvergence*)

save()

get_convergence()

Returns convergence *InterfaceConvergence*

get_devices()

Returns List of devices (ordered by evaluation number)

get_logopti()

Returns Log optimization (to check the convergence)

start(*optimization_parameters*)
Function called upon starting the optimization. Create folders.

visualize

Subpackages

graphs

colormap_pyqtgraph

Module Contents

Functions

Attributes

has_matplotlib = True

sequence

matplotlib_colormap_to_pg_colormap(*colormap_name*, *n_ticks=16*)

cmapToColormap(*cmap*, *nTicks=16*)
Converts a Matplotlib cmap to pyqtgraphs colormaps. No dependency on matplotlib. Parameters:

cmap: Cmap object. Imported from matplotlib.cm.* *nTicks*: Number of ticks to create when dict of functions is used. Otherwise unused.

author: Sebastian Hoefer

graphVisual

Module Contents

Classes

class GraphVisual(*theWidgetGraphVisual*)
Provide an interface to a graph. A graph contains traces.

set_fontTicks(*fontSize*, *fontname=None*)
Set font of the ticks

Parameters

- **fontSize** – Size of the font
- **fontname** – Name of the font

set_numberTicks(*number*, *axis*)
Set the number of ticks to be displayed

Parameters

- **number** – Number of ticks for the axis
- **axis** – Axis (string, “bottom”, “left”, “right”, “top”)

Returns

set_fontLabel(*fontSize*, *color='#000'*, *fontname=None*)
Set font of the axis labels

Parameters

- **fontSize** – font size
- **color** – color in hexadecimal (str)
- **fontname** – name of the font

get_legend() → optimeed.visualize.graphs.pyqtgraphRedefine.myLegend
Get the legend

get_axis(*axis*) → optimeed.visualize.graphs.pyqtgraphRedefine.myAxis
Get the axis

Parameters **axis** – Axis (string, “bottom”, “left”, “right”, “top”)

Returns axis object

set_fontLegend(*font_size*, *font_color*, *fontname=None*)

set_label_pos(*orientation*, *x_offset=0*, *y_offset=0*)

set_color_palette(*palette*)

apply_palette()

hide_axes()

add_feature(*theFeature*)
To add any pyqtgraph item to the graph

remove_feature(*theFeature*)
To remove any pyqtgraph item from the graph

add_data(*idGraph*, *theData*)

set_graph_properties(*theTrace*)
This function is automatically called on creation of the graph

set_lims(*xlim*, *ylim*)
Set limits of the graphs, xlim or ylim = [val_low, val_high]. Or None.

add_trace(*idTrace*, *theTrace*)
Add a `TraceVisual` to the graph, with index idTrace

set_legend()
Set default legend options (color and font)

set_title(*titleName*, ***kwargs*)
Set title of the graph

Parameters **titleName** – title to set

get_trace(*idTrace*) → optimeed.visualize.graphs.traceVisual.TraceVisual
Return the `TraceVisual` correspondong to the index idTrace

get_all_traces()
Return a dictionary {idtrace: `TraceVisual`}.

delete_trace(*idTrace*)
Delete the trace of index idTrace

delete()
Delete the graph

linkXToGraph(*graph*)
Link the axis of the current graph to an other *GraphVisual*

update()
Update the traces contained in the graph

fast_update()
Same as *update()* but faster. This is NOT thread safe (cannot be called a second time before finishing operation)

axis_equal()

log_mode(*x=False*, *y=False*)

grid_off()
Turn off grid

pyqtgraphRedefine

Module Contents

Classes

Attributes

isOnWindows
Other modified files (directly): ScatterPlotItem.py, to change point selection. Ctrl + clic: select area. Clic: only one single point:

class OnClicSelector:

def __init__(self): self.p_list = []

def add_point(self, newp): self.p_list.append(newp)

def draw(self, painter):

if len(self.p_list) > 2: pen = fn.mkPen(1) pen.setWidthF(2) painter.setPen(pen) painter.drawPolyline(QtGui.QPolygonF(self.p_list))

def reset(self): self.p_list = []

def getPath(self): return path.Path([(p.x(), p.y()) for p in self.p_list] + [(self.p_list[-1].x(), self.p_list[-1].y())])

def mouseDragEvent(self, ev):

if ev.modifiers() and QtCore.Qt.ControlModifier: ev.accept()
self.clicSelector.add_point(ev.pos()) if ev.isFinish():

path = self.clicSelector.getPath() points = self.points() contains_points = path.contains_points([(p.pos().x(), p.pos().y()) for p in points]) indices = [i for i, cond in enumerate(contains_points) if cond] points_clicked = [points[i] for i in indices] self.ptsClicked = points_clicked self.sigClicked.emit(self, self.ptsClicked) self.clicSelector.reset()

self.update()

else: ev.ignore()

class myGraphicsLayoutWidget(*parent=None*, ***_kwargs*)
Bases: `optimeed.visualize.graphs.pyqtgraph.GraphicsView`

Re-implementation of QGraphicsView that removes scrollbars and allows unambiguous control of the viewed coordinate range. Also automatically creates a GraphicsScene and a central QGraphicsWidget that is automatically scaled to the full view geometry.

This widget is the basis for `PlotWidget`, `GraphicsLayoutWidget`, and the view widget in `ImageView`.

By default, the view coordinate system matches the widget's pixel coordinates and automatically updates when the view is resized. This can be overridden by setting autoPixelRange=False. The exact visible range can be set with setRange().

The view can be panned using the middle mouse button and scaled using the right mouse button if enabled via enableMouse() (but ordinarily, we use ViewBox for this functionality).

useOpenGL(*b=True*)
Overwrited to fix bad antialiasing while using openGL

class myGraphicsLayout
Bases: `optimeed.visualize.graphs.pyqtgraph.GraphicsLayout`

Used for laying out GraphicsWidgets in a grid. This is usually created automatically as part of a `GraphicsWindow` or `GraphicsLayoutWidget`.

addItem(*item*, *row=None*, *col=None*, *rowspan=1*, *colspan=1*)
Add an item to the layout and place it in the next available cell (or in the cell specified). The item must be an instance of a QGraphicsWidget subclass.

set_graph_disposition(*item*, *row=1*, *col=1*, *rowspan=1*, *colspan=1*)
Function to modify the position of an item in the list

Parameters

- **item** – WidgetPlotItem to set
- **row** – Row
- **col** – Column
- **rowspan** –
- **colspan** –

Returns

class myItemSample(*item*)
Bases: `optimeed.visualize.graphs.pyqtgraph.graphicsItems.LegendItem.ItemSample`

Class responsible for drawing a single item in a LegendItem (sans label)

set_offset(*offset*)

set_width_cell(*width*)

paint(*p*, **args*)
Overwrites to make matlab-like samples

class myLegend(*size=None*, *offset=(30, 30)*, *is_light=False*)
Bases: optimeed.visualize.graphs.pyqtgraph.LegendItem

Legend that fixes bugs (flush left + space) from pyqtgraph's legend

set_space_sample_label(*theSpace*)
To set the gap between the sample and the label

set_offset_sample(*offset*)
To tune the offset between the sample and the text

set_width_cell_sample(*width*)
Set width of sample

updateSize()

addItem(*item*, *name*)
Overwrites to flush left

apply_width_sample()

set_font(*font_size*, *font_color*, *fontname=None*)

paint(*p*, **args*)
Overwrited to select background color

set_position(*position*, *offset*)
Set the position of the legend, in a corner.

Parameters

- **position** – String (NW, NE, SW, SE), indicates which corner the legend is close
- **offset** – Tuple (xoff, yoff), x and y offset from the edge

Returns

class myLabelItem(*text=' '*, *parent=None*, *angle=0*, ***args*)
Bases: optimeed.visualize.graphs.pyqtgraph.LabelItem

GraphicsWidget displaying text. Used mainly as axis labels, titles, etc.

Note: To display text inside a scaled view (ViewBox, PlotWidget, etc) use TextItem

setText(*text*, ***args*)
Overwrited to add font-family to options

class myAxis(*orientation*)
Bases: optimeed.visualize.graphs.pyqtgraph.AxisItem

GraphicsItem showing a single plot axis with ticks, values, and label. Can be configured to fit on any side of a plot, Can automatically synchronize its displayed scale with ViewBox items. Ticks can be extended to draw a grid. If maxTickLength is negative, ticks point into the plot.

update_label

_updateLabel()
Internal method to update the label according to the text

get_label_pos()
Overwrited to place label closer to the axis

resizeEvent(*ev=None*)
Overwrited to place label closer to the axis

set_label_pos(*orientation*, *x_offset=0*, *y_offset=0*)

set_number_ticks(*number*)

traceVisual

Module Contents

Classes

Functions

Attributes

default_colormap

_normalize_colors(*z*)

class TraceVisual(*theData*, *theWGPlot*, *highlight_last*)
Bases: `PyQt5.QtCore.QObject`

Defines a trace in a graph.

class _ModifiedPaintElem
Hidden class to manage brushes or pens

add_modified_paintElem(*index*, *newPaintElem*)

modify_paintElems(*paintElemsIn_List*)
Apply transformation to paintElemsIn_List.
Param paintElemsIn_List: list of brushes or pens to modify
Returns False if nothing has been modified, True is something has been modified

reset_paintElem(*index*)
Remove transformation of point index

reset()

signal_must_update

hide_points()
Hide all the points

get_color()
Get colour of the trace, return tuple (r,g,b)

set_color(*color*)
Set colour of the trace, argument as tuple (r,g,b)

get_base_symbol_brush()
Get symbol brush configured for this trace, return *pg.QBrush*

get_base_pen()
Get pen configured for this trace, return *pg.QPen*

get_base_symbol_pen()
Get symbol pen configured for this trace, return‘pg.QPen‘

get_base_symbol()
Get base symbol configured for this trace, return str of the symbol (e.g. ‘o’)

get_symbol(*size*)
Get actual symbols for the trace. If the symbols have been modified: return a list which maps each points to a symbol. Otherwise: return :meth:TraceVisual.get_base_symbol()

updateTrace()
Forces the trace to refresh.

get_length()
Return number of data to plot

hide()
Hides the trace

show()
Shows the trace

toggle(*boolean*)
Toggle the trace (hide/show)

get_data()
Get data to plot `Data`

get_brushes(*size*)
Get actual brushes for the trace (=symbol filling). return a list which maps each points to a symbol brush

set_brush(*indexPoint*, *newbrush*, *update=True*)
Set the symbol brush for a specific point:

Parameters

- **indexPoint** – Index of the point (in the graph) to modify
- **newbrush** – either QBrush or tuple (r, g, b) of the new brush
- **update** – if True, update the trace afterwards. This is slow operation.

set_symbol(*indexPoint*, *newSymbol*, *update=True*)
Set the symbol shape for a specific point:

Parameters

- **indexPoint** – Index of the point (in the graph) to modify
- **newSymbol** – string of the new symbol (e.g.: ‘o’)
- **update** – if True, update the trace afterwards. This is slow operation.

set_brushes(*list_indexPoint*, *list_newbrush*, *update=True*)
Same as *set_brush()* but by taking a list as input

reset_brush(*indexPoint*, *update=True*)
Reset the brush of the point indexpoint

reset_brushes(*list_indexPoint*, *update=True*)
Same as *reset_brush()* but by taking a list as input

reset_all_brushes(*update=True*)
Reset all the brushes

reset_symbol(*indexPoint*, *update=True*)
Reset the symbol shape of the point indexpoint

get_symbolPens(*size*)
Get actual symbol pens for the trace (=symbol outline). return a list which maps each points to a symbol pen

set_symbolPen(*indexPoint*, *newPen*, *update=True*)
Set the symbol shape for a specific point:

Parameters

- **indexPoint** – Index of the point (in the graph) to modify
- **newPen** – QPen item or tuple of the color (r,g,b)
- **update** – if True, update the trace afterwards. This is slow operation.

set_symbolPens(*list_indexPoint*, *list_newpens*, *update=True*)
Same as *set_symbolPen()* but by taking a list as input

reset_symbolPen(*indexPoint*, *update=True*)
Reset the symbol pen of the point indexpoint

reset_symbolPens(*list_indexPoint*, *update=True*)
Same as *reset_symbolPen()* but by taking a list as input

reset_all_symbolPens(*update=True*)
Reset all the symbol pens

get_point(*indexPoint*)
Return object pyqtgraph.SpotItem

widget_graphsVisual

Module Contents

Classes

class Widget_graphsVisualLite(*theGraphs*, ***kwargs*)
Bases: `PyQt5.QtWidgets.QWidget`

Widget element to draw a graph. The traces and graphs to draw are defined in `Graphs` taken as argument. This widget is linked to the excellent third-party library pyqtgraph, under MIT license

signal_must_update

signal_graph_changed

set_graph_disposition(*indexGraph*, *row=1*, *col=1*, *rowspan=1*, *colspan=1*)
Change the graphs disposition.

Parameters

- **indexGraph** – index of the graph to change
- **row** – row where to place the graph
- **col** – column where to place the graph
- **rowspan** – number of rows across which the graph spans
- **colspan** – number of columns across which the graph spans

Returns

__create_graph(*idGraph*)

__check_graphs()

on_click(*plotDataItem*, *clicked_points*)

update_graphs(*singleUpdate=True*)
This method is used to update the graph. This is fast but NOT safe (especially when working with threads). To limit the risks, please use self.signal_must_update.emit() instead.

Parameters **singleUpdate** – if set to False, the graph will periodically refres each self.refreshtime

fast_update()
Use this method to update the graph in a fast way. NOT THREAD SAFE.

select_folder_and_export()

exportGraphs(*filename*)
Export the graphs

export_txt(*filename_txt*)

export_svg(*filename*)

export_tikz(*foldername_tikz*)

link_axes()

get_graph(*idGraph*) → optimeed.visualize.graphs.graphVisual.GraphVisual
Get corresponding `GraphVisual` of the graph idGraph

get_trace(*idGraph*, *idTrace*) → optimeed.visualize.graphs.traceVisual.TraceVisual
Get corresponding Tracevisual

keyPressEvent(*event*)
What happens if a key is pressed. R: reset the axes to their default value

delete_graph(*idGraph*)
Delete the graph idGraph

delete()

get_all_graphsVisual()
Return a dictionary {idGraph: `GraphVisual`}.

get_layout_buttons()
Get the QGraphicsLayout where it's possible to add buttons, etc.

set_actionOnClick(*theActionOnClick*)
Action to perform when the graph is clicked

Parameters **theActionOnClick** – `on_graph_click_interface`

Returns

set_title(*idGraph*, *titleName*, ***kwargs*)
Set title of the graph

Parameters

- **idGraph** – id of the graph
- **titleName** – title to set

class Widget_graphsVisual(**args*, ***kwargs*)
Bases: *Widget_graphsVisualLite*

Create a gui for pyqtgraph with trace selection options, export and action on clic choices

refreshTraceList()
Refresh all the traces

set_actions_on_click(*actions*)

Package Contents

Classes

class Widget_graphsVisualLite(*theGraphs*, ***kwargs*)
Bases: `PyQt5.QtWidgets.QWidget`

Widget element to draw a graph. The traces and graphs to draw are defined in `Graphs` taken as argument. This widget is linked to the excellent third-party library pyqtgraph, under MIT license

signal_must_update

signal_graph_changed

set_graph_disposition(*indexGraph*, *row=1*, *col=1*, *rowspan=1*, *colspan=1*)
Change the graphs disposition.

Parameters

- **indexGraph** – index of the graph to change
- **row** – row where to place the graph
- **col** – column where to place the graph
- **rowspan** – number of rows across which the graph spans
- **colspan** – number of columns across which the graph spans

Returns

__create_graph(*idGraph*)

__check_graphs()

on_click(*plotDataItem*, *clicked_points*)

update_graphs(*singleUpdate=True*)
This method is used to update the graph. This is fast but NOT safe (especially when working with threads). To limit the risks, please use self.signal_must_update.emit() instead.

Parameters **singleUpdate** – if set to False, the graph will periodically refres each self.refreshtime

fast_update()
Use this method to update the graph in a fast way. NOT THREAD SAFE.

select_folder_and_export()

exportGraphs(*filename*)
Export the graphs

export_txt(*filename_txt*)

export_svg(*filename*)

export_tikz(*foldername_tikz*)

link_axes()

get_graph(*idGraph*) → optimeed.visualize.graphs.graphVisual.GraphVisual
Get corresponding `GraphVisual` of the graph idGraph

get_trace(*idGraph*, *idTrace*) → optimeed.visualize.graphs.traceVisual.TraceVisual
Get corresponding Tracevisual

keyPressEvent(*event*)
What happens if a key is pressed. R: reset the axes to their default value

delete_graph(*idGraph*)
Delete the graph idGraph

delete()

get_all_graphsVisual()
Return a dictionary {idGraph: `GraphVisual`}.

get_layout_buttons()
Get the QGraphicsLayout where it's possible to add buttons, etc.

set_actionOnClick(*theActionOnClick*)
Action to perform when the graph is clicked

Parameters **theActionOnClick** – `on_graph_click_interface`

Returns

set_title(*idGraph*, *titleName*, ***kwargs*)
Set title of the graph

Parameters

- **idGraph** – id of the graph
- **titleName** – title to set

class Widget_graphsVisual(**args*, ***kwargs*)
Bases: *Widget_graphsVisualLite*

Create a gui for pyqtgraph with trace selection options, export and action on clic choices

refreshTraceList()
Refresh all the traces

set_actions_on_click(*actions*)

onclick

animationGUI

Module Contents

Classes

class _AnimationTrace(*elements_list*, *theTrace*)
Contains all the element to animate for a trace

class AnimationElement(*elements*)

get()

get_element_animations(*itemNumber*, *index_in_show*)
Get the element to show :param itemNumber: item number (0 if only one think to draw) :param index_in_show: index in the list :return: The element to draw

show_all()

delete_all()

get_indices_to_show()

add_element(*indexPoint*)

add_index_to_show(*index*)

_remove_index_from_show(*index*)

set_curr_brush(*index_in_show*)

set_idle_brush(*index_in_show*)

get_number_of_elements()

map_index(*index_in_show*)

get_base_pen()

class AnimationGUI(*id=0*, *window_title='Animation'*)
Bases: `PyQt5.QtWidgets.QMainWindow`

Spawns a gui that includes button to create animations nicely when paired with `widget_graphs_visual`

SlIDER_MAXIMUM_VALUE = 500

SLIDER_MINIMUM_VALUE = 1

add_trace(*trace_id*, *element_list*, *theTrace*)
Add a trace to the animation.

Parameters

- **trace_id** – id of the trace
- **element_list** – List of elements to save: [[OpenGL_item1, text_item1], [OpenGL_item2, text_item2], ... [OpenGL_itemN, text_itemN]]
- **theTrace** – `TraceVisual`

Returns

add_elementToTrace(*trace_id*, *indexPoint*)

delete_point(*trace_id*, *thePoint*)

reset_all()

delete_all()

pause_play()

show_all()

next_frame()

slider_handler()

frame_selector()

set_refreshTime()

is_empty()

run()

closeEvent(_)

contains_trace(*trace_id*)

export_picture()

animation_examples

Module Contents

Classes

class Animate_openGL(*theOpenGLWidget*, *theId=0*, *window_title='Animation'*)
Bases: *optimeed.visualize.onclick.animationGUI.AnimationGUI*

Implements `DataAnimationVisuals` to show opengl drawing

update_widget_w_animation(*key*, *index*, *the_data_animation*)
What to do when a new element has to be animated. Example: self.theOpenGLWidget.set_deviceToDraw(the_data_animation.get_element_animations(0, index))

Parameters

- **key** – key of the trace that has to be animated
- **index** – index that has to be animated
- **the_data_animation** – `DataAnimationTrace` that has to be animated

export_widget(*painter*)
Render scene with a painter

Parameters **painter** – PyQt painter

delete_key_widgets(*key*)
What to do when a key has to be deleted

Parameters **key** – key of the trace that has to be deleted

class Animate_openGL_and_text(**args*, *is_light=True*, ***kwargs*)
Bases: *Animate_openGL*

Implements `DataAnimationVisuals` to show opengl drawing and text

update_widget_w_animation(*key*, *index*, *the_data_animation*)
What to do when a new element has to be animated. Example: self.theOpenGLWidget.set_deviceToDraw(the_data_animation.get_element_animations(0, index))

Parameters

- **key** – key of the trace that has to be animated
- **index** – index that has to be animated
- **the_data_animation** – `DataAnimationTrace` that has to be animated

get_interesting_elements(*devices_list*)
Function called upon new trace creation. From a list, takes the interesting elements for animation :param element_list: :return: new_element_list

class Animate_lines(*get_lines_method*, *is_light=True*, *theId=0*, *window_title='Animation'*)
Bases: *optimeed.visualize.onclick.animationGUI.AnimationGUI*

Implements `DataAnimationVisuals` to show drawing made out of lines (`widget_line_drawer`)

export_widget(*painter*)
Render scene with a painter

Parameters **painter** – PyQt painter

delete_key_widgets(*key*)
What to do when a key has to be deleted

Parameters **key** – key of the trace that has to be deleted

update_widget_w_animation(*key*, *index*, *the_data_animation*)
What to do when a new element has to be animated. Example: self.theOpenGLWidget.set_deviceToDraw(the_data_animation.get_element_animations(0, index))

Parameters

- **key** – key of the trace that has to be animated
- **index** – index that has to be animated
- **the_data_animation** – `DataAnimationTrace` that has to be animated

get_interesting_elements(*devices_list*)
Function called upon new trace creation. From a list, takes the interesting elements for animation :param element_list: :return: new_element_list

class Animate_lines_and_text(**args*, ***kwargs*)
Bases: *Animate_lines*

Same as `DataAnimationLines` but also with text

update_widget_w_animation(*key*, *index*, *the_data_animation*)
What to do when a new element has to be animated. Example: self.theOpenGLWidget.set_deviceToDraw(the_data_animation.get_element_animations(0, index))

Parameters

- **key** – key of the trace that has to be animated
- **index** – index that has to be animated
- **the_data_animation** – `DataAnimationTrace` that has to be animated

collectionExporterGUI

Module Contents

Classes

class CollectionExporterGUI
Bases: `PyQt5.QtWidgets.QMainWindow`

Simple gui that allows to export data

signal_has_exported

signal_has_reset

exportCollection()
Export the collection

reset()

add_data_to_collection(*data*)
Add data to the collection to export

Parameters **data** – Whichever type you like

set_collection(*theCollection*)

onclickInterface

Module Contents

Classes

class OnclickInterface
Interface class for the action to perform when a point is clicked

onclick_animate

Module Contents

Classes

class Onclick_animate(*theLinkDataGraph*, *theAnimation*)
Bases: *optimeed.visualize.onclick.onclickInterface.OnclickInterface*

On click: add or remove an element to animate

graph_clicked(*theGraphVisual*, *index_graph*, *index_trace*, *indices_points*)
Action to perform when a graph is clicked

Parameters

- **theGraphsVisual** – class widget_graphs_visual that has called the method
- **index_graph** – Index of the graph that has been clicked
- **index_trace** – Index of the trace that has been clicked
- **indices_points** – graph Indices of the points that have been clicked

Returns

get_name()

onclick_changeSymbol

Module Contents

Classes

class Onclick_changeSymbol(*theLinkDataGraph*)
Bases: *optimeed.visualize.onclick.onclickInterface.OnclickInterface*

On Click: Change the symbol of the point that is clicked

graph_clicked(*theGraphVisual*, *index_graph*, *index_trace*, *indices_points*)
Action to perform when a graph is clicked

Parameters

- **theGraphsVisual** – class widget_graphs_visual that has called the method
- **index_graph** – Index of the graph that has been clicked
- **index_trace** – Index of the trace that has been clicked
- **indices_points** – graph Indices of the points that have been clicked

Returns

get_name()

onclick_copySomething

Module Contents

Classes

class Onclick_copySomething(*theDataLink*, *functionStrFromDevice*)
Bases: *optimeed.visualize.onclick.onclickInterface.OnclickInterface*

On Click: copy something

graph_clicked(*the_graph_visual*, *index_graph*, *index_trace*, *indices_points*)
Action to perform when a graph is clicked

Parameters

- **theGraphsVisual** – class widget_graphs_visual that has called the method
- **index_graph** – Index of the graph that has been clicked
- **index_trace** – Index of the trace that has been clicked
- **indices_points** – graph Indices of the points that have been clicked

Returns

get_name()

onclick_delete

Module Contents

Classes

class Onclick_delete(*theDataLink*)
Bases: *optimeed.visualize.onclick.onclickInterface.OnclickInterface*

On Click: Delete the points from the graph

graph_clicked(*_theGraphVisual*, *index_graph*, *index_trace*, *indices_points*)
Action to perform when a graph is clicked

Parameters

- **theGraphsVisual** – class widget_graphs_visual that has called the method
- **index_graph** – Index of the graph that has been clicked
- **index_trace** – Index of the trace that has been clicked
- **indices_points** – graph Indices of the points that have been clicked

Returns

get_name()

onclick_exportCollection

Module Contents

Classes

class Onclick_exportCollection(*theDataLink*)
Bases: *optimeed.visualize.onclick.onclickInterface.OnclickInterface*

On click: export the selected points

graph_clicked(*theGraphVisual*, *index_graph*, *index_trace*, *indices_points*)
Action to perform when a graph is clicked

Parameters

- **theGraphsVisual** – class widget_graphs_visual that has called the method
- **index_graph** – Index of the graph that has been clicked
- **index_trace** – Index of the trace that has been clicked
- **indices_points** – graph Indices of the points that have been clicked

Returns

reset_graph()

get_name()

onclick_exportToTxt

Module Contents

Classes

class Onclick_exportToTxt(*theDataLink*, *attributes_shadow=None*)
Bases: *optimeed.visualize.onclick.onclickInterface.OnclickInterface*

On click: export the data of the whole the trace selected

graph_clicked(*theGraphVisual*, *index_graph*, *index_trace*, *indices_points*)
Action to perform when a graph is clicked

Parameters

- **theGraphsVisual** – class widget_graphs_visual that has called the method
- **index_graph** – Index of the graph that has been clicked
- **index_trace** – Index of the trace that has been clicked
- **indices_points** – graph Indices of the points that have been clicked

Returns

get_name()

onclick_exportTrace

Module Contents

Classes

class Onclick_exportTrace(*theDataLink*, *getShadow=True*)
Bases: *optimeed.visualize.onclick.onclickInterface.OnclickInterface*

On click: export the data of the whole the trace selected

graph_clicked(*theGraphVisual*, *index_graph*, *index_trace*, *indices_points*)
Action to perform when a graph is clicked

Parameters

- **theGraphsVisual** – class widget_graphs_visual that has called the method
- **index_graph** – Index of the graph that has been clicked
- **index_trace** – Index of the trace that has been clicked
- **indices_points** – graph Indices of the points that have been clicked

Returns

get_name()

onclick_extractPareto

Module Contents

Classes

class Onclick_extractPareto(*theDataLink*, *max_x=False*, *max_y=False*)
Bases: *optimeed.visualize.onclick.onclickInterface.OnclickInterface*

On click: extract the pareto from the cloud of points

graph_clicked(*the_graph_visual*, *index_graph*, *index_trace*, *_*)
Action to perform when a graph is clicked

Parameters

- **theGraphsVisual** – class widget_graphs_visual that has called the method
- **index_graph** – Index of the graph that has been clicked
- **index_trace** – Index of the trace that has been clicked
- **indices_points** – graph Indices of the points that have been clicked

Returns

get_name()

onclick_measure

Module Contents

Classes

class _LineItem(*point1*, *point2*)
Bases: optimeed.visualize.graphs.pyqtgraph.GraphicsObject

Bases: GraphicsItem, QtWidgets.QGraphicsObject

Extension of QGraphicsObject with some useful methods (provided by GraphicsItem)

paint(*p*, **args*)

boundingRect()

class Onclick_measure
Bases: *optimeed.visualize.onclick.onclickInterface.OnclickInterface*

On Click: Measure distance. Click on two points to perform that action

graph_clicked(*the_graph_visual*, *index_graph*, *index_trace*, *indices_points*)
Action to perform when a graph is clicked

Parameters

- **theGraphsVisual** – class widget_graphs_visual that has called the method
- **index_graph** – Index of the graph that has been clicked
- **index_trace** – Index of the trace that has been clicked
- **indices_points** – graph Indices of the points that have been clicked

Returns

reset_distance()

display_distance()

get_name()

onclick_removeTrace

Module Contents

Classes

class Onclick_removeTrace(*theDataLink*)

Bases: *optimeed.visualize.onclick.onclickInterface.OnclickInterface*

Interface class for the action to perform when a point is clicked

graph_clicked(*theGraphVisual*, *index_graph*, *index_trace*, *_*)

Action to perform when a graph is clicked

Parameters

- **theGraphsVisual** – class widget_graphs_visual that has called the method
- **index_graph** – Index of the graph that has been clicked
- **index_trace** – Index of the trace that has been clicked
- **indices_points** – graph Indices of the points that have been clicked

Returns

get_name()

onclick_representDevice

Module Contents

Classes

class RepresentDeviceInterface

class Onclick_representDevice(*theLinkDataGraph*, *visuals*)

Bases: *optimeed.visualize.onclick.onclickInterface.OnclickInterface*

On click: show informations about the points (loop through attributes)

class DataInformationVisuals

delete_visual(*theVisual*)

add_visual(*theVisual*, *theTrace*, *indexPoint*)

get_new_index()

curr_index()

graph_clicked(*theGraphVisual*, *index_graph*, *index_trace*, *indices_points*)
Action to perform when a point in the graph has been clicked: Creates new window displaying the device and its informations

get_name()

onclick_tojson

Module Contents

Classes

class Onclick_tojson(*theDataLink*)
Bases: *optimeed.visualize.onclick.onclickInterface.OnclickInterface*

Interface class for the action to perform when a point is clicked

graph_clicked(*theGraphVisual*, *index_graph*, *index_trace*, *indices_points*)
Action to perform when a graph is clicked

Parameters

- **theGraphsVisual** – class widget_graphs_visual that has called the method
- **index_graph** – Index of the graph that has been clicked
- **index_trace** – Index of the trace that has been clicked
- **indices_points** – graph Indices of the points that have been clicked

Returns

get_name()

representDevice_examples

Module Contents

Classes

class Represent_lines(*attribute_lines*)
Bases: *optimeed.visualize.onclick.onclick_representDevice.RepresentDeviceInterface*

get_widget(*theNewDevice*)
Get Qt widget that represents the device

Parameters **theDevice** – the Device to be represented

Returns Qt widget

class Represent_txt_function(*is_light=True*, *convertToHtml=True*)
Bases: *optimeed.visualize.onclick.onclick_representDevice.RepresentDeviceInterface*

getTxt(*theNewDevice*)

get_widget(*theNewDevice*)
Get Qt widget that represents the device

Parameters **theDevice** – the Device to be represented

Returns Qt widget

class Represent_brut_attributes(*is_light=True*, *convertToHtml=True*, *recursion_level=5*)
Bases: *optimeed.visualize.onclick.onclick_representDevice.RepresentDeviceInterface*

get_widget(*theNewDevice*)
Get Qt widget that represents the device

Parameters **theDevice** – the Device to be represented

Returns Qt widget

class Represent_opengl(*DeviceDrawer*)
Bases: *optimeed.visualize.onclick.onclick_representDevice.RepresentDeviceInterface*

get_widget(*theNewDevice*)
Get Qt widget that represents the device

Parameters **theDevice** – the Device to be represented

Returns Qt widget

class Represent_image(*get_base_64_from_device*)
Bases: *optimeed.visualize.onclick.onclick_representDevice.RepresentDeviceInterface*

get_widget(*theNewDevice*)
Get Qt widget that represents the device

Parameters **theDevice** – the Device to be represented

Returns Qt widget

Package Contents

Classes

class RepresentDeviceInterface

class Onclick_animate(*theLinkDataGraph*, *theAnimation*)
Bases: *optimeed.visualize.onclick.onclickInterface.OnclickInterface*

On click: add or remove an element to animate

graph_clicked(*theGraphVisual*, *index_graph*, *index_trace*, *indices_points*)
Action to perform when a graph is clicked

Parameters

- **theGraphsVisual** – class widget_graphs_visual that has called the method
- **index_graph** – Index of the graph that has been clicked
- **index_trace** – Index of the trace that has been clicked
- **indices_points** – graph Indices of the points that have been clicked

Returns

get_name()

class Onclick_changeSymbol(*theLinkDataGraph*)
Bases: *optimeed.visualize.onclick.onclickInterface.OnclickInterface*

On Click: Change the symbol of the point that is clicked

graph_clicked(*theGraphVisual*, *index_graph*, *index_trace*, *indices_points*)
Action to perform when a graph is clicked

Parameters

- **theGraphsVisual** – class widget_graphs_visual that has called the method
- **index_graph** – Index of the graph that has been clicked
- **index_trace** – Index of the trace that has been clicked
- **indices_points** – graph Indices of the points that have been clicked

Returns

get_name()

class Onclick_copySomething(*theDataLink*, *functionStrFromDevice*)
Bases: *optimeed.visualize.onclick.onclickInterface.OnclickInterface*

On Click: copy something

graph_clicked(*the_graph_visual*, *index_graph*, *index_trace*, *indices_points*)
Action to perform when a graph is clicked

Parameters

- **theGraphsVisual** – class widget_graphs_visual that has called the method
- **index_graph** – Index of the graph that has been clicked
- **index_trace** – Index of the trace that has been clicked
- **indices_points** – graph Indices of the points that have been clicked

Returns

get_name()

class Onclick_delete(*theDataLink*)
Bases: *optimeed.visualize.onclick.onclickInterface.OnclickInterface*

On Click: Delete the points from the graph

graph_clicked(*_theGraphVisual*, *index_graph*, *index_trace*, *indices_points*)
Action to perform when a graph is clicked

Parameters

- **theGraphsVisual** – class widget_graphs_visual that has called the method
- **index_graph** – Index of the graph that has been clicked
- **index_trace** – Index of the trace that has been clicked
- **indices_points** – graph Indices of the points that have been clicked

Returns

get_name()

class Onclick_exportCollection(*theDataLink*)
Bases: *optimeed.visualize.onclick.onclickInterface.OnclickInterface*

On click: export the selected points

graph_clicked(*theGraphVisual*, *index_graph*, *index_trace*, *indices_points*)
Action to perform when a graph is clicked

Parameters

- **theGraphsVisual** – class widget_graphs_visual that has called the method
- **index_graph** – Index of the graph that has been clicked
- **index_trace** – Index of the trace that has been clicked
- **indices_points** – graph Indices of the points that have been clicked

Returns

reset_graph()

get_name()

class Onclick_exportToTxt(*theDataLink*, *attributes_shadow=None*)
Bases: *optimeed.visualize.onclick.onclickInterface.OnclickInterface*

On click: export the data of the whole the trace selected

graph_clicked(*theGraphVisual*, *index_graph*, *index_trace*, *indices_points*)
Action to perform when a graph is clicked

Parameters

- **theGraphsVisual** – class widget_graphs_visual that has called the method
- **index_graph** – Index of the graph that has been clicked
- **index_trace** – Index of the trace that has been clicked
- **indices_points** – graph Indices of the points that have been clicked

Returns

get_name()

class Onclick_exportTrace(*theDataLink*, *getShadow=True*)
Bases: *optimeed.visualize.onclick.onclickInterface.OnclickInterface*

On click: export the data of the whole the trace selected

graph_clicked(*theGraphVisual*, *index_graph*, *index_trace*, *indices_points*)
Action to perform when a graph is clicked

Parameters

- **theGraphsVisual** – class widget_graphs_visual that has called the method
- **index_graph** – Index of the graph that has been clicked
- **index_trace** – Index of the trace that has been clicked
- **indices_points** – graph Indices of the points that have been clicked

Returns

get_name()

class Onclick_extractPareto(*theDataLink*, *max_x=False*, *max_y=False*)
Bases: *optimeed.visualize.onclick.onclickInterface.OnclickInterface*

On click: extract the pareto from the cloud of points

graph_clicked(*the_graph_visual*, *index_graph*, *index_trace*, *_*)
Action to perform when a graph is clicked

Parameters

- **theGraphsVisual** – class widget_graphs_visual that has called the method
- **index_graph** – Index of the graph that has been clicked
- **index_trace** – Index of the trace that has been clicked
- **indices_points** – graph Indices of the points that have been clicked

Returns

get_name()

class Onclick_measure
Bases: *optimeed.visualize.onclick.onclickInterface.OnclickInterface*

On Click: Measure distance. Click on two points to perform that action

graph_clicked(*the_graph_visual*, *index_graph*, *index_trace*, *indices_points*)
Action to perform when a graph is clicked

Parameters

- **theGraphsVisual** – class widget_graphs_visual that has called the method
- **index_graph** – Index of the graph that has been clicked
- **index_trace** – Index of the trace that has been clicked
- **indices_points** – graph Indices of the points that have been clicked

Returns

reset_distance()

display_distance()

get_name()

class Onclick_removeTrace(*theDataLink*)
Bases: *optimeed.visualize.onclick.onclickInterface.OnclickInterface*

Interface class for the action to perform when a point is clicked

graph_clicked(*theGraphVisual*, *index_graph*, *index_trace*, *_*)
Action to perform when a graph is clicked

Parameters

- **theGraphsVisual** – class widget_graphs_visual that has called the method
- **index_graph** – Index of the graph that has been clicked
- **index_trace** – Index of the trace that has been clicked
- **indices_points** – graph Indices of the points that have been clicked

Returns

get_name()

class Onclick_representDevice(*theLinkDataGraph*, *visuals*)
Bases: *optimeed.visualize.onclick.onclickInterface.OnclickInterface*

On click: show informations about the points (loop through attributes)

class DataInformationVisuals

delete_visual(*theVisual*)

add_visual(*theVisual*, *theTrace*, *indexPoint*)

get_new_index()

curr_index()

graph_clicked(*theGraphVisual*, *index_graph*, *index_trace*, *indices_points*)
Action to perform when a point in the graph has been clicked: Creates new window displaying the device and its informations

get_name()

class Onclick_tojson(*theDataLink*)
Bases: *optimeed.visualize.onclick.onclickInterface.OnclickInterface*

Interface class for the action to perform when a point is clicked

graph_clicked(*theGraphVisual*, *index_graph*, *index_trace*, *indices_points*)
Action to perform when a graph is clicked

Parameters

- **theGraphsVisual** – class widget_graphs_visual that has called the method
- **index_graph** – Index of the graph that has been clicked
- **index_trace** – Index of the trace that has been clicked
- **indices_points** – graph Indices of the points that have been clicked

Returns

get_name()

class OnclickInterface
Interface class for the action to perform when a point is clicked

class Represent_opengl(*DeviceDrawer*)
Bases: *optimeed.visualize.onclick.onclick_representDevice.RepresentDeviceInterface*

get_widget(*theNewDevice*)
Get Qt widget that represents the device

Parameters **theDevice** – the Device to be represented

Returns Qt widget

class Represent_image(*get_base_64_from_device*)
Bases: *optimeed.visualize.onclick.onclick_representDevice.RepresentDeviceInterface*

get_widget(*theNewDevice*)
Get Qt widget that represents the device

Parameters **theDevice** – the Device to be represented

Returns Qt widget

class Represent_lines(*attribute_lines*)
Bases: *optimeed.visualize.onclick.onclick_representDevice.RepresentDeviceInterface*

get_widget(*theNewDevice*)
Get Qt widget that represents the device

Parameters **theDevice** – the Device to be represented

Returns Qt widget

class Represent_brut_attributes(*is_light=True*, *convertToHtml=True*, *recursion_level=5*)
Bases: *optimeed.visualize.onclick.onclick_representDevice.RepresentDeviceInterface*

get_widget(*theNewDevice*)
Get Qt widget that represents the device

Parameters **theDevice** – the Device to be represented

Returns Qt widget

class Represent_txt_function(*is_light=True*, *convertToHtml=True*)
Bases: *optimeed.visualize.onclick.onclick_representDevice.RepresentDeviceInterface*

getTxt(*theNewDevice*)

get_widget(*theNewDevice*)
Get Qt widget that represents the device

Parameters **theDevice** – the Device to be represented

Returns Qt widget

class Animate_lines(*get_lines_method*, *is_light=True*, *theId=0*, *window_title='Animation'*)
Bases: *optimeed.visualize.onclick.animationGUI.AnimationGUI*

Implements `DataAnimationVisuals` to show drawing made out of lines (`widget_line_drawer`)

export_widget(*painter*)
Render scene with a painter

Parameters **painter** – PyQt painter

delete_key_widgets(*key*)
What to do when a key has to be deleted

Parameters **key** – key of the trace that has to be deleted

update_widget_w_animation(*key*, *index*, *the_data_animation*)
What to do when a new element has to be animated. Example: self.theOpenGLWidget.set_deviceToDraw(the_data_animation.get_element_animations(0, index))

Parameters

- **key** – key of the trace that has to be animated
- **index** – index that has to be animated
- **the_data_animation** – `DataAnimationTrace` that has to be animated

get_interesting_elements(*devices_list*)
Function called upon new trace creation. From a list, takes the interesting elements for animation :param element_list: :return: new_element_list

class Animate_openGL(*theOpenGLWidget*, *theId=0*, *window_title='Animation'*)
Bases: *optimeed.visualize.onclick.animationGUI.AnimationGUI*

Implements `DataAnimationVisuals` to show opengl drawing

update_widget_w_animation(*key*, *index*, *the_data_animation*)
What to do when a new element has to be animated. Example: self.theOpenGLWidget.set_deviceToDraw(the_data_animation.get_element_animations(0, index))

Parameters

- **key** – key of the trace that has to be animated
- **index** – index that has to be animated
- **the_data_animation** – `DataAnimationTrace` that has to be animated

export_widget(*painter*)
Render scene with a painter

Parameters **painter** – PyQt painter

delete_key_widgets(*key*)
What to do when a key has to be deleted

Parameters **key** – key of the trace that has to be deleted

class Animate_lines_and_text(**args*, ***kwargs*)
Bases: *Animate_lines*

Same as `DataAnimationLines` but also with text

update_widget_w_animation(*key*, *index*, *the_data_animation*)
What to do when a new element has to be animated. Example: self.theOpenGLWidget.set_deviceToDraw(the_data_animation.get_element_animations(0, index))

Parameters

- **key** – key of the trace that has to be animated
- **index** – index that has to be animated
- **the_data_animation** – `DataAnimationTrace` that has to be animated

class Animate_openGL_and_text(**args*, *is_light=True*, ***kwargs*)
Bases: *Animate_openGL*

Implements `DataAnimationVisuals` to show opengl drawing and text

update_widget_w_animation(*key*, *index*, *the_data_animation*)
What to do when a new element has to be animated. Example: self.theOpenGLWidget.set_deviceToDraw(the_data_animation.get_element_animations(0, index))

Parameters

- **key** – key of the trace that has to be animated
- **index** – index that has to be animated
- **the_data_animation** – `DataAnimationTrace` that has to be animated

get_interesting_elements(*devices_list*)
Function called upon new trace creation. From a list, takes the interesting elements for animation :param element_list: :return: new_element_list

openGL

contextHandler

Module Contents

Classes

Attributes

MODE_ZOOM = 0

MODE_ROTATION = 1

MODE_LIGHT = 2

NUMBER_OF_MODES = 3

CLIC_LEFT = 0

CLIC_RIGHT = 1

class SpecialButtonsMapping

class MyText(*color*, *fontSize*, *theStr*, *windowPosition*)

class ContextHandler

set_specialButtonsMapping(*theSpecialButtonsMapping*)

set_deviceDrawer(*theDeviceDrawer*)

set_deviceToDraw(*theDeviceToDraw*)

resizeWindowAction(*new_width*, *new_height*)

mouseWheelAction(*deltaAngle*)

mouseClicAction(*button*, *my_x*, *y*)

mouseMotionAction(*my_x*, *y*)

keyboardPushAction(*key*)

keyboardReleaseAction(*key*, *my_x*, *y*)

__draw_axis__()

redraw()

get_text_to_write()

__lightingInit__()

initialize()

__reset__()

deviceDrawerInterface

Module Contents

Classes

class DeviceDrawerInterface

keyboard_push_action(*theKey*)

get_colour_scalebar()

get_colour_background()

get_opengl_options()

materials

Module Contents

Classes

Attributes

class MaterialRenderingProperties(*amb3*, *dif3*, *spec3*, *shin*)

__spec3__ = [0, 0, 0, 0]

__dif3__ = [0, 0, 0, 0]

__amb3__ = [0, 0, 0, 0]

__shin__ = 0

getSpec3()

getDif3()

getAmb3()

getShin()

activateMaterialProperties(*alpha=1*)

Emerald_material

Yellow_Emerald_material

Brass_material

Bronze_material

Silver_material

Steel_material

Copper_material

Chrome_material

Blue_material

Red_material

Green_material

Cyan_material

Pink_material

openGL_library

Module Contents

Functions

draw_closedPolygon(*xClockWise*, *yClockWise*)

draw_extrudeZ(*xList*, *yList*, *zExtrude*)

draw_triList(*theTriList*)

draw_lines(*x*, *z*)

draw_spiralSheet(*innerRadius*, *thickness*, *length*, *theAngle*, *n*, *reverseDirection=False*)

draw_spiralFront(*innerRadius*, *thicknessMaterial*, *thicknessSpiral*, *z0*, *theAngle*, *n*, *reverseDirection=False*)

draw_spiralFull(*innerRadius*, *outerRadius*, *thicknessMaterial*, *thicknessSpiral*, *length*, *n*)

draw_spiral(*innerRadius*, *outerRadius*, *thicknessMaterial*, *thicknessSpiral*, *length*, *cutAngle*, *n*)

draw_simple_rectangle(*width*, *height*)

draw_rectangle(*rIn*, *length*, *thickness*, *angle*, *reverseDirection=False*)

draw_2Dring(*innerRadius*, *outerRadius*, *z0*, *theAngle*, *n*, *reverseDirection=False*)

draw_2Dring_diff_angle(*innerRadius*, *outerRadius*, *angle_in*, *angle_out*, *n*, *reverseDirection=False*)

draw_tubeSheet(*radius*, *length*, *theAngle*, *n*, *reverseDirection=False*)

draw_cylinder(*innerRadius*, *outerRadius*, *length*, *n*, *translate=0*)

draw_part_cylinder(*innerRadius*, *outerRadius*, *length*, *angle*, *n*, *translate=0*, *drawSides=True*)

draw_disk(*innerRadius*, *outerRadius*, *n*, *translate=0*)

draw_part_disk(*innerRadius*, *outerRadius*, *thickness*, *angle*, *n*, *translate=0*)

draw_part_disk_diff_angles(*innerRadius*, *outerRadius*, *thickness*, *angle_in*, *angle_out*, *n*)

draw_carved_disk(*innerRadius*, *outerRadius*, *carvedRin*, *carvedRout*, *thickness*, *depth*, *angle*, *n*, *translate=0*)

draw_part_cylinder_throat(*rIn*, *rOut*, *rOutThroat*, *length*, *lengthThroat*, *angle*, *n*, *translate=0*)

drawWireTube(*diameter*, *xa*, *ya*, *xb*, *yb*, *n=50*, *translateZ=0*)

quaternions

Module Contents

Functions

normalize(*v*, *tolerance=0.001*)

q_mult(*q1*, *q2*)

q_conjugate(*q*)

qv_mult(*q1*, *v1*)

axisangle_to_q(*v*, *theta*)

q_to_axisangle(*q*)

q_to_mat4(*q*)

triangulate_polygon

Module Contents

Functions

IsConvex(*a*, *b*, *c*)

InTriangle(*a*, *b*, *c*, *p*)

IsClockwise(*poly*)

GetEar(*poly*)

reformatXYtoList(*xList*, *yList*)

meshPolygon(*xList*, *yList*)

Package Contents

Classes

Attributes

class DeviceDrawerInterface

> **keyboard_push_action**(*theKey*)
>
> **get_colour_scalebar**()
>
> **get_colour_background**()
>
> **get_opengl_options**()

class MaterialRenderingProperties(*amb3*, *dif3*, *spec3*, *shin*)

__spec3__ = [0, 0, 0, 0]

__dif3__ = [0, 0, 0, 0]

__amb3__ = [0, 0, 0, 0]

__shin__ = 0

getSpec3()

getDif3()

getAmb3()

getShin()

activateMaterialProperties(*alpha=1*)

Emerald_material

Yellow_Emerald_material

Brass_material

Bronze_material

Silver_material

Steel_material

Copper_material

Chrome_material

Blue_material

Red_material

Green_material

Cyan_material

Pink_material

selector

onselectInterface

Module Contents

Classes

class OnselectInterface

onselect_highlight

Module Contents

Classes

class Onselect_highlight(*theLinkDataGraphs*, *theWgPlot*)
Bases: *optimeed.visualize.selector.onselectInterface.OnselectInterface*

selector_updated(*selection_name*, *the_collection*, *selected_data*, *not_selected_data*)
Action to perform once the data have been selected

Parameters

- **selection_name** – name of the selection (deprecated ?)
- **the_collection** – the collection
- **selected_data** – indices of the data selected
- **not_selected_data** – indices of the data not selected

Returns

cancel_selector(*selection_identifier*)
Action to perform when data stopped being selected :param selection_identifier: identifier that was returned by selector_updated :return:

get_name()
Get the name of the action

Returns string

onselect_newTrace

Module Contents

Classes

class Onselect_newTrace(*theLinkDataGraphs*)
Bases: *optimeed.visualize.selector.onselectInterface.OnselectInterface*

selector_updated(*selection_name*, *the_collection*, *selected_data*, *not_selected_data*)
Action to perform once the data have been selected

Parameters

- **selection_name** – name of the selection (deprecated ?)
- **the_collection** – the collection
- **selected_data** – indices of the data selected
- **not_selected_data** – indices of the data not selected

Returns identifier that can later be used with cancel_selector

cancel_selector(*selection_identifier*)
Action to perform when data stopped being selected :param selection_identifier: identifier that was returned by selector_updated :return:

get_name()
Get the name of the action

Returns string

onselect_splitTrace

Module Contents

Classes

class Onselect_splitTrace(*theLinkDataGraphs*)
Bases: *optimeed.visualize.selector.onselectInterface.OnselectInterface*

selector_updated(*selection_name*, *the_collection*, *selected_data*, *not_selected_data*)
Action to perform once the data have been selected

Parameters

- **selection_name** – name of the selection (deprecated ?)
- **the_collection** – the collection
- **selected_data** – indices of the data selected
- **not_selected_data** – indices of the data not selected

Returns identifier that can later be used with cancel_selector

cancel_selector(*selection_identifiers*)
Action to perform when data stopped being selected :param selection_identifier: identifier that was returned by selector_updated :return:

get_name()
Get the name of the action

Returns string

Package Contents

Classes

class OnselectInterface

class Onselect_highlight(*theLinkDataGraphs*, *theWgPlot*)
Bases: *optimeed.visualize.selector.onselectInterface.OnselectInterface*

selector_updated(*selection_name*, *the_collection*, *selected_data*, *not_selected_data*)
Action to perform once the data have been selected

Parameters

- **selection_name** – name of the selection (deprecated ?)
- **the_collection** – the collection
- **selected_data** – indices of the data selected
- **not_selected_data** – indices of the data not selected

Returns

cancel_selector(*selection_identifier*)
Action to perform when data stopped being selected :param selection_identifier: identifier that was returned by selector_updated :return:

get_name()
Get the name of the action

Returns string

class Onselect_newTrace(*theLinkDataGraphs*)
Bases: *optimeed.visualize.selector.onselectInterface.OnselectInterface*

selector_updated(*selection_name*, *the_collection*, *selected_data*, *not_selected_data*)
Action to perform once the data have been selected

Parameters

- **selection_name** – name of the selection (deprecated ?)
- **the_collection** – the collection
- **selected_data** – indices of the data selected
- **not_selected_data** – indices of the data not selected

Returns identifier that can later be used with cancel_selector

cancel_selector(*selection_identifier*)
Action to perform when data stopped being selected :param selection_identifier: identifier that was returned by selector_updated :return:

get_name()
Get the name of the action

Returns string

class Onselect_splitTrace(*theLinkDataGraphs*)
Bases: *optimeed.visualize.selector.onselectInterface.OnselectInterface*

selector_updated(*selection_name*, *the_collection*, *selected_data*, *not_selected_data*)
Action to perform once the data have been selected

Parameters

- **selection_name** – name of the selection (deprecated ?)
- **the_collection** – the collection
- **selected_data** – indices of the data selected
- **not_selected_data** – indices of the data not selected

Returns identifier that can later be used with cancel_selector

cancel_selector(*selection_identifiers*)
Action to perform when data stopped being selected :param selection_identifier: identifier that was returned by selector_updated :return:

get_name()
Get the name of the action

Returns string

widgets

widget_doubleSlider

Module Contents

Classes

class widget_doubleSlider(*decimals=3*, **args*, ***kwargs*)
Bases: PyQt5.QtWidgets.QSlider

doubleValueChanged

emitDoubleValueChanged()

value()

setMinimum(*value*)

setMaximum(*value*)

setSingleStep(*value*)

singleStep()

setValue(*value*)

widget_image

Module Contents

Classes

class Widget_image(*image_b64*)
Bases: PyQt5.QtWidgets.QLabel

eventFilter(*source*, *event*)

set_image(*image_b64*)
Set new image to widget

widget_lineDrawer

Module Contents

Classes

class Widget_lineDrawer(*minWinHeight=300*, *minWinWidth=300*, *is_light=True*)
Bases: PyQt5.QtWidgets.QWidget

Widget allowing to display several lines easily

signal_must_update

on_update_signal(*listOfLines*)

delete_lines(*key_id*)
Dele the lines :param key_id: id to delete :return:

set_lines(*listOfLines*, *key_id=0*, *pen=None*)
Set the lines to display :param listOfLines: list of [x1, y1, x2, y2] corresponding to lines :param key_id: id of the trace :param pen: pen used to draw the lines :return:

paintEvent(*event*, *painter=None*)

get_extrema_lines()

widget_listWithSearch

Module Contents

Classes

class Widget_listWithSearch(**args*, ***kwargs*)
Bases: `PyQt5.QtWidgets.QWidget`

get_index_selected()

get_name_selected()

set_list(*names*)

_filter_list()

_iter_items()

widget_listWithSearchplugin

Module Contents

Classes

class Plugin_listWithSearch(*parent=None*)
Bases: `PyQt5.QtDesigner.QPyDesignerCustomWidgetPlugin`

initialize(*core*)

isInitialized()

createWidget(*parent*)

name()

group()

icon()

toolTip()

whatsThis()

isContainer()

includeFile()

widget_menuButton

Module Contents

Classes

class Widget_menuButton(*theParentButton*)
Bases: PyQt5.QtWidgets.QMenu

Same as QMenu, but integrates it behind a button more easily.

showEvent(*QShowEvent*)

mouseReleaseEvent(*QMouseEvent*)

widget_openGL

Module Contents

Classes

class Widget_openGL(*parent=None*)
Bases: PyQt5.QtWidgets.QOpenGLWidget

Interface that provides opengl capabilities. Ensures zoom, light, rotation, etc.

sizeHint()

minimumSizeHint()

set_deviceDrawer(*theDeviceDrawer*)
Set a drawer optimeed.visualize.widgets.openGL.deviceDrawerInterface.DeviceDrawerInterface

set_deviceToDraw(*theDeviceToDraw*)
Set the device to draw

initializeGL()

paintGL()

resizeGL(*w*, *h*)

mousePressEvent(*event*)

mouseMoveEvent(*event*)

keyPressEvent(*event*)

wheelEvent(*QWheelEvent*)

widget_tableWithSearch

Module Contents

Classes

class Widget_tableWithSearch(**args*, ***kwargs*)
Bases: PyQt5.QtWidgets.QWidget

cellChanged

hideRow(*row*)

showRow(*row*)

force_hide_row(*row*)

remove_forced_hide_row(*row*)

get_entries_selected()

_cellChanged()

set_entries(*names*, *numColumns=3*, *hidden=False*)

get_shown_entries()

set_item(*row*, *col*, *item*)

get_item(*row*, *col*)

_filter_list()

_iter_items()

widget_tableWithSearchplugin

Module Contents

Classes

class Plugin_tableWithSearch(*parent=None*)
Bases: PyQt5.QtDesigner.QPyDesignerCustomWidgetPlugin

initialize(*core*)

isInitialized()

createWidget(*parent*)

name()

group()

icon()

toolTip()

whatsThis()

isContainer()

includeFile()

widget_text

Module Contents

Classes

class Widget_text(*theText*, *is_light=False*, *convertToHtml=False*)
Bases: PyQt5.QtWidgets.QLabel

Widget able to display a text

set_text(*theText*, *convertToHtml=False*)
Set the text to display

class Widget_text_scrollable(*theText*, *is_light=False*, *convertToHtml=False*)
Bases: PyQt5.QtWidgets.QWidget

Same as widget_text but scrollable

set_text(*theText*, *convertToHtml=False*)

Package Contents

Classes

class Widget_image(*image_b64*)
Bases: PyQt5.QtWidgets.QLabel

eventFilter(*source*, *event*)

set_image(*image_b64*)
Set new image to widget

class Widget_lineDrawer(*minWinHeight=300*, *minWinWidth=300*, *is_light=True*)
Bases: PyQt5.QtWidgets.QWidget

Widget allowing to display several lines easily

signal_must_update

on_update_signal(*listOfLines*)

delete_lines(*key_id*)
Dele the lines :param key_id: id to delete :return:

set_lines(*listOfLines*, *key_id=0*, *pen=None*)
Set the lines to display :param listOfLines: list of [x1, y1, x2, y2] corresponding to lines :param key_id: id of the trace :param pen: pen used to draw the lines :return:

paintEvent(*event*, *painter=None*)

get_extrema_lines()

class Widget_listWithSearch(**args*, ***kwargs*)
Bases: PyQt5.QtWidgets.QWidget

get_index_selected()

get_name_selected()

set_list(*names*)

_filter_list()

_iter_items()

class Widget_menuButton(*theParentButton*)
Bases: PyQt5.QtWidgets.QMenu

Same as QMenu, but integrates it behind a button more easily.

showEvent(*QShowEvent*)

mouseReleaseEvent(*QMouseEvent*)

class Widget_openGL(*parent=None*)
Bases: PyQt5.QtWidgets.QOpenGLWidget

Interface that provides opengl capabilities. Ensures zoom, light, rotation, etc.

sizeHint()

minimumSizeHint()

set_deviceDrawer(*theDeviceDrawer*)
Set a drawer optimeed.visualize.widgets.openGL.deviceDrawerInterface.DeviceDrawerInterface

set_deviceToDraw(*theDeviceToDraw*)
Set the device to draw

initializeGL()

paintGL()

resizeGL(*w*, *h*)

mousePressEvent(*event*)

mouseMoveEvent(*event*)

keyPressEvent(*event*)

wheelEvent(*QWheelEvent*)

class Widget_tableWithSearch(**args*, ***kwargs*)
Bases: PyQt5.QtWidgets.QWidget

cellChanged

hideRow(*row*)

showRow(*row*)

force_hide_row(*row*)

remove_forced_hide_row(*row*)

get_entries_selected()

_cellChanged()

set_entries(*names*, *numColumns=3*, *hidden=False*)

get_shown_entries()

set_item(*row*, *col*, *item*)

get_item(*row*, *col*)

_filter_list()

_iter_items()

class Widget_text(*theText*, *is_light=False*, *convertToHtml=False*)
Bases: PyQt5.QtWidgets.QLabel

Widget able to display a text

set_text(*theText*, *convertToHtml=False*)
Set the text to display

class Widget_text_scrollable(*theText*, *is_light=False*, *convertToHtml=False*)
Bases: PyQt5.QtWidgets.QWidget

Same as *widget_text* but scrollable

set_text(*theText*, *convertToHtml=False*)

displayCollections

Module Contents

Classes

Functions

_is_object_selected(*object_in*, *min_max_attributes*)

_select_and_apply_action(*theCollections*, *min_max_attributes*, *theAction*, *selectionName*)

class CollectionDisplayer
Bases: PyQt5.QtWidgets.QMainWindow

GUI to display a collection.

add_collection(*theCollection*, *name=''*)
Add a collection to the GUI

set_shadow(*master_collectionId*, *shadow_collection*)
Set a shadow collection to master_collectionID (see DataLink.set_shadow_collection)

remove_collection(*theCollection*)
Remove collection from the GUI

update_graphs()

set_actions_on_click(*theActionsOnClick*)
Set actions to be performed when graph is clicked

get_datalink()

_initialize(*theCollection*)

_set_x()

_set_y()

_set_z()

set_action_selector(*theAction*)

_selector_to()

_remove_item_selector()

_cancel_selector()

_apply_selector()

_reset_colors()

displayOptimization

Module Contents

Classes

Functions

check_if_must_plot(*elem*)

run_optimization_displayer(**args*, ***kwargs*)

class OptimizationDisplayer(*theOptiParameters*, *theOptiHistoric*, *additionalWidgets=None*, *light_background=False*)

Bases: *optimeed.core.Option_class*

Class used to display optimization process in real time

signal_optimization_over

SHOW_CONSTRAINTS = 0

set_actionsOnClick(*theList*)

Set actions to perform on click, list of `on_graph_click_interface`

generate_optimizationGraphs()

Generates the optimization graphs. :return: *Graphs*, *LinkDataGraph*, :class:'~optimeed.visulaize.gui.widgets.widget_graphs_visual.widget_graphs_visual

__change_appearance_violate_constraints()

__refresh()

start_autorefresh(*timer_autosave*)

stop_autorefresh()

__set_graphs_disposition()

Set nicely the graphs disposition

launch_optimization(*args_opti*, *kwargs_opti*, *refresh_time=0.1*, *max_nb_points_convergence=100*)

Perform the optimization and spawn the convergence graphs afterwards. :param args_opti: arguments (as list) destined to launch the optimization :param kwargs_opti: keywords arguments (as dict) destined to launch the optimization :param refresh_time: float indicating the refresh time of the graphs. If it becomes laggy -> use a higher one. :param max_nb_points_convergence: maximum number of points in the graph that displays the convergence. Put None if performance is not an issue.

close_windows()

display_graphs(*theGraphs*)

create_main_window()

From the widgets and the actions on click, spawn a window and put a gui around widgetsGraphsVisual.

displaySensitivity

Module Contents

Classes

Functions

analyse_sobol_plot_convergence(*theDict*, *sobol='S1'*, *title=''*, *hold=True*)
Plot convergence of the sobol indices.

Parameters

- **theDict** – Dictionary containing sobol indices
- **sobol** – Key of the dictionary to investigate
- **title** – Title of the convergence window
- **hold** – If true, this function will be blocking (otherwise use start_qt_mainloop)

Returns window containing convergence graphs

analyse_sobol_plot_indices(*theSensitivityParameters: optimeed.consolidate.SensitivityParameters*, *objectives*, *title=''*, *hold=True*)
¨Plot first and total order sobol indices.

Parameters

- **theSensitivityParameters** – Parameters used for sensitivity study
- **objectives** – List of evaluated objectives to analyse
- **title** – Title of the window
- **hold** – If true, this function will be blocking (otherwise use plt.show())

Returns

analyse_sobol_plot_2ndOrder_indices(*theSensitivityParameters: optimeed.consolidate.SensitivityParameters*, *objectives*, *title=''*, *hold=True*)
¨Plot second order sobol indices. Args and kwargs are the same as analyse_sobol_plot_indices

class SensitivityDisplayer
Bases: `PyQt5.QtWidgets.QMainWindow`

GUI to display a sensitivity analysis.

add_study(*theCollection*, *theParameters*, *name*)
Add sensitivity study to the GUI

Parameters

- **theCollection** – Results of the sensitivity study
- **theParameters** – Parameters of the sensitivity study
- **name** – Name (for the GUI) of the sensitivity study

Returns

_set_study(*index*)

_get_sobol_indices()

_get_S1_conv()

_get_ST_conv()

fastPlot

Module Contents

Classes

Functions

Attributes

class _PlotHolders

add_plot(*x*, *y*, ***kwargs*)

get_wgGraphs()

new_plot()

set_title(*theTitle*, ***kwargs*)

reset()

axis_equal()

class WindowHolders

set_currFigure(*currFigure*)

add_plot(**args*, ***kwargs*)

set_title(**args*, ***kwargs*)

new_figure()

new_plot()

show()

get_curr_plotHolder()

get_wgGraphs(*fig=None*)

get_all_figures()

axis_equal()

add_action_on_click(*theAction*)

myWindows

plot(*x*, *y*, *hold=False*, ***kwargs*)
Plot new trace

show()
Show (start qt mainloop) graphs. Blocking

figure(*numb=None*)
Set current figure

add_action_on_click(*theAction*)

set_title(*theTitle*, ***kwargs*)
Set title of the plot

axis_equal()

get_all_figures()
Get all existing figures

get_wgGraphs(*fig=None*)
Advanced option. :return: `widget_graphs_visual`

fastPlot3

Module Contents

Functions

Attributes

hasPlotly = True

_do_scatterPlot(*theData: optimeed.core.ScatterPlot3*)

mainWindow

Module Contents

Classes

class MainWindow(*QtWidgetList*, *isLight=True*, *actionOnWindowClosed=None*, *neverCloseWindow=False*, *title_window='Awesome Visualisation Tool'*, *size=None*)
Bases: `PyQt5.QtWidgets.QMainWindow`

Main class that spawns a Qt window. Use `run()` to display it.

set_actionOnClose(*actionOnWindowClosed*)

closeEvent(*event*)

run(*hold=False*)
Display the window

keyPressEvent(*event*)

process_mainloop

Module Contents

Functions

Attributes

app

start_qt_mainloop()
Starts qt mainloop, which is necessary for qt to handle events

stop_qt_mainloop()
Stops qt mainloop and resumes to program

process_qt_events()
Process current qt events

viewOptimizationResults

Module Contents

Classes

class _OptiProjectLoader(*foldername*, *kwargsPlot=None*)
A loader for an opti project.

get_devices() → optimeed.core.ListDataStruct_Interface

get_logopti() → optimeed.core.ListDataStruct_Interface

get_convergence()

get_kwargs()

get_nbr_objectives()

class ViewOptimizationResults
Convenience class to display the results of an optimization

add_opti_project(*foldername*, *kwargsPlot=None*)
Add an opti project to visualize.

Parameters

- **foldername** – the folder containing the saved files. (as string)
- **kwargsPlot** – Check kgwargs ~*optimeed.core.graphs.Data*

get_data_link() → optimeed.core.LinkDataGraph
Return the object *LinkDataGraph*

display_graphs(*theActionsOnClick=None*, *kwargs_common=None*, *keep_alive=True*, *max_nb_points_convergence=None*, *light_background=False*)
Generates the optimization graphs.

Parameters

- **theActionsOnClick** – list of actions to perform when a graph is clicked
- **kwargs_common** – plot options (from Data class) to apply to all the graphs (ex: {"is_scattered": True}).
- **keep_alive** – if set to true, this method will be blocking. Otherwise you should manually call start_qt_mainloop().

- **max_nb_points_convergence** – maximum number of points in the graph that displays the convergence. Put None if performance is not an issue.
- **light_background** – boolean, True or False for White or Black background color in graphs

Returns widget_graphs_visual for the log opti, widget_graphs_visual for the convergence (`widget_graphs_visual`)

Package Contents

Classes

Functions

Attributes

class CollectionDisplayer

Bases: `PyQt5.QtWidgets.QMainWindow`

GUI to display a collection.

add_collection(*theCollection*, *name=''*)
Add a collection to the GUI

set_shadow(*master_collectionId*, *shadow_collection*)
Set a shadow collection to master_collectionID (see DataLink.set_shadow_collection)

remove_collection(*theCollection*)
Remove collection from the GUI

update_graphs()

set_actions_on_click(*theActionsOnClick*)
Set actions to be performed when graph is clicked

get_datalink()

_initialize(*theCollection*)

_set_x()

_set_y()

_set_z()

set_action_selector(*theAction*)

_selector_to()

_remove_item_selector()

_cancel_selector()

_apply_selector()

_reset_colors()

class SensitivityDisplayer

Bases: `PyQt5.QtWidgets.QMainWindow`

GUI to display a sensitivity analysis.

add_study(*theCollection*, *theParameters*, *name*)
Add sensitivity study to the GUI

Parameters

- **theCollection** – Results of the sensitivity study
- **theParameters** – Parameters of the sensitivity study
- **name** – Name (for the GUI) of the sensitivity study

Returns

_set_study(*index*)

_get_sobol_indices()

_get_S1_conv()

_get_ST_conv()

analyse_sobol_plot_indices(*theSensitivityParameters: optimeed.consolidate.SensitivityParameters*, *objectives*, *title=''*, *hold=True*)
¨Plot first and total order sobol indices.

Parameters

- **theSensitivityParameters** – Parameters used for sensitivity study
- **objectives** – List of evaluated objectives to analyse
- **title** – Title of the window
- **hold** – If true, this function will be blocking (otherwise use plt.show())

Returns

analyse_sobol_plot_convergence(*theDict*, *sobol='S1'*, *title=''*, *hold=True*)
Plot convergence of the sobol indices.

Parameters

- **theDict** – Dictionary containing sobol indices
- **sobol** – Key of the dictionary to investigate
- **title** – Title of the convergence window
- **hold** – If true, this function will be blocking (otherwise use start_qt_mainloop)

Returns window containing convergence graphs

analyse_sobol_plot_2ndOrder_indices(*theSensitivityParameters: optimeed.consolidate.SensitivityParameters*, *objectives*, *title=''*, *hold=True*)
¨Plot second order sobol indices. Args and kwargs are the same as analyse_sobol_plot_indices

class OptimizationDisplayer(*theOptiParameters*, *theOptiHistoric*, *additionalWidgets=None*, *light_background=False*)
Bases: *optimeed.core.Option_class*

Class used to display optimization process in real time

signal_optimization_over

SHOW_CONSTRAINTS = 0

set_actionsOnClick(*theList*)
Set actions to perform on click, list of `on_graph_click_interface`

generate_optimizationGraphs()
Generates the optimization graphs. :return: *Graphs*, *LinkDataGraph*, :class:'~optimeed.visulaize.gui.widgets.widget_graphs_visual.widget_graphs_visual

__change_appearance_violate_constraints()

__refresh()

start_autorefresh(*timer_autosave*)

stop_autorefresh()

__set_graphs_disposition()
Set nicely the graphs disposition

launch_optimization(*args_opti*, *kwargs_opti*, *refresh_time=0.1*, *max_nb_points_convergence=100*)
Perform the optimization and spawn the convergence graphs afterwards. :param args_opti: arguments (as list) destined to launch the optimization :param kwargs_opti: keywords arguments (as dict) destined to launch the optimization :param refresh_time: float indicating the refresh time of the graphs. If it becomes laggy -> use a higher one. :param max_nb_points_convergence: maximum number of points in the graph that displays the convergence. Put None if performance is not an issue.

close_windows()

display_graphs(*theGraphs*)

create_main_window()
From the widgets and the actions on click, spawn a window and put a gui around widgetsGraphsVisual.

class ViewOptimizationResults
Convenience class to display the results of an optimization

add_opti_project(*foldername*, *kwargsPlot=None*)
Add an opti project to visualize.

Parameters

- **foldername** – the folder containing the saved files. (as string)
- **kwargsPlot** – Check kgwargs ~*optimeed.core.graphs.Data*

get_data_link() → optimeed.core.LinkDataGraph
Return the object *LinkDataGraph*

display_graphs(*theActionsOnClick=None*, *kwargs_common=None*, *keep_alive=True*, *max_nb_points_convergence=None*, *light_background=False*)
Generates the optimization graphs.

Parameters

- **theActionsOnClick** – list of actions to perform when a graph is clicked
- **kwargs_common** – plot options (from Data class) to apply to all the graphs (ex: {"is_scattered": True}).
- **keep_alive** – if set to true, this method will be blocking. Otherwise you should manually call start_qt_mainloop().
- **max_nb_points_convergence** – maximum number of points in the graph that displays the convergence. Put None if performance is not an issue.
- **light_background** – boolean, True or False for White or Black background color in graphs

Returns widget_graphs_visual for the log opti, widget_graphs_visual for the convergence (`widget_graphs_visual`)

class Widget_graphsVisual(**args*, ***kwargs*)
Bases: *Widget_graphsVisualLite*

Create a gui for pyqtgraph with trace selection options, export and action on clic choices

refreshTraceList()
Refresh all the traces

set_actions_on_click(*actions*)

class MainWindow(*QtWidgetList*, *isLight=True*, *actionOnWindowClosed=None*, *neverCloseWindow=False*, *title_window='Awesome Visualisation Tool'*, *size=None*)
Bases: `PyQt5.QtWidgets.QMainWindow`

Main class that spawns a Qt window. Use `run()` to display it.

set_actionOnClose(*actionOnWindowClosed*)

closeEvent(*event*)

run(*hold=False*)
Display the window

keyPressEvent(*event*)

start_qt_mainloop()
Starts qt mainloop, which is necessary for qt to handle events

class Data(*x: list*, *y: list*, *x_label=''*, *y_label=''*, *legend=''*, *is_scattered=False*, *transfo_x=lambda selfData, x: x*, *transfo_y=lambda selfData, y: y*, *xlim=None*, *ylim=None*, *permutations=None*, *sort_output=False*, *color=None*, *alpha=255*, *symbol='o'*, *symbolsize=8*, *fillsymbol=True*, *outlinesymbol=1.8*, *linestyle='-'*, *width=2*, *meta=None*)
This class is used to store informations necessary to plot a 2D graph. It has to be combined with a gui to be useful (ex. pyqtgraph)

set_kwargs(*kwargs*)
Set a kwarg after creation of the class

set_data(*x: list*, *y: list*)
Overwrites current datapoints with new set

set_meta(*meta*)
Set associated 'Z' data

get_x()
Get x coordinates of datapoints

get_symbolsize()
Get size of the symbols

symbol_isfilled()
Check if symbols has to be filled or not

get_symbolOutline()
Get color factor of outline of symbols

get_length_data()
Get number of points

get_xlim()
Get x limits of viewbox

get_ylim()
Get y limits of viewbox

get_y()
Get y coordinates of datapoints

get_meta()
Get associated 'Z' data

get_color()
Get color of the line, without transformation

get_color_alpha()
Get color of the line. Return r, g, b in 0, 255 scale

get_alpha()
Get opacity

get_width()
Get width of the line

get_number_of_points()
Get number of points

get_plot_data()
Call this method to get the x and y coordinates of the points that have to be displayed. => After transformation, and after permutations.

Returns x (list), y (list)

get_plot_meta(*x*, *y*)
Call this method to get the z coordinates of the points that been displayed. => After transformation, and after permutations.

Returns z (list)

get_permutations(*x=None*)
Return the transformation 'permutation': xplot[i] = xdata[permutation[i]]

get_invert_permutations()
Return the inverse of permutations: xdata[i] = xplot[revert[i]]

get_dataIndex_from_graphIndex(*index_graph_point*)
From an index given in graph, recovers the index of the data.

Parameters **index_graph_point** – Index in the graph

Returns index of the data

get_dataIndices_from_graphIndices(*index_graph_point_list*)
Same as get_dataIndex_from_graphIndex but with a list in entry. Can (?) improve performances for huge dataset.

Parameters **index_graph_point_list** – List of Index in the graph

Returns List of index of the data

get_graphIndex_from_dataIndex(*index_data*)
From an index given in the data, recovers the index of the graph.

Parameters **index_data** – Index in the data

Returns index of the graph

get_graphIndices_from_dataIndices(*index_data_list*)
Same as get_graphIndex_from_dataIndex but with a list in entry. Can (?) improve performances for huge dataset.

Parameters **index_data_list** – List of Index in the data

Returns List of index of the graph

set_permutations(*permutations*)
Set permutations between datapoints of the trace

Parameters **permutations** – list of indices to plot (example: [0, 2, 1] means that the first point will be plotted, then the third, then the second one)

get_x_label()
Get x label of the trace

get_y_label()
Get y label of the trace

get_legend()
Get name of the trace

get_symbol()
Get symbol

add_point(*x*, *y*)
Add point(s) to trace (inputs can be list or numeral)

delete_point(*index_point*)
Delete a point from the datapoints

isScattered()
Check if plot is scatteded

set_indices_points_to_plot(*indices*)
Set indices points to plot

get_indices_points_to_plot()
Get indices points to plot

get_linestyle()
Get linestyle

__str__()
Return str(self).

export_str()
Method to save the points constituting the trace

set_color(*theColor*)
Set trace color

set_legend(*theLegend*)
Set legend

class Graphs
Contains several `Graph`

updateChildren()

add_trace_firstGraph(*data*, *updateChildren=True*)
Same as add_trace, but only if graphs has only one id :param data: :param updateChildren: :return:

add_trace(*idGraph*, *data*, *updateChildren=True*)
Add a trace to the graph

Parameters

- **idGraph** – id of the graph
- **data** – *Data*
- **updateChildren** – Automatically calls callback functions

Returns id of the created trace

remove_trace(*idGraph*, *idTrace*, *updateChildren=True*)
Remove the trace from the graph

Parameters

- **idGraph** – id of the graph
- **idTrace** – id of the trace to remove
- **updateChildren** – Automatically calls callback functions

get_first_graph()
Get id of the first graph

Returns id of the first graph

get_graph(*idGraph*)
Get graph object at idgraph

Parameters **idGraph** – id of the graph to get

Returns Graph

get_all_graphs_ids()
Get all ids of the graphs

Returns list of id graphs

get_all_graphs()
Get all graphs. Return dict {id: Graph}

add_graph(*updateChildren=True*)
Add a new graph

Returns id of the created graph

remove_graph(*idGraph*)
Delete a graph

Parameters **idGraph** – id of the graph to delete

add_update_method(*childObject*)
Add a callback each time a graph is modified.

Parameters **childObject** – method without arguments

export_str()
Export all the graphs in text

Returns str

merge(*otherGraphs*)

reset()

is_empty()

class Onclick_measure

Bases: *optimeed.visualize.onclick.onclickInterface.OnclickInterface*

On Click: Measure distance. Click on two points to perform that action

graph_clicked(*the_graph_visual*, *index_graph*, *index_trace*, *indices_points*)

Action to perform when a graph is clicked

Parameters

- **theGraphsVisual** – class widget_graphs_visual that has called the method
- **index_graph** – Index of the graph that has been clicked
- **index_trace** – Index of the trace that has been clicked
- **indices_points** – graph Indices of the points that have been clicked

Returns

reset_distance()

display_distance()

get_name()

class _PlotHolders

add_plot(*x*, *y*, ***kwargs*)

get_wgGraphs()

new_plot()

set_title(*theTitle*, ***kwargs*)

reset()

axis_equal()

class WindowHolders

set_currFigure(*currFigure*)

add_plot(**args*, ***kwargs*)

set_title(**args*, ***kwargs*)

new_figure()

new_plot()

show()

get_curr_plotHolder()

get_wgGraphs(*fig=None*)

get_all_figures()

axis_equal()

add_action_on_click(*theAction*)

myWindows

plot(*x*, *y*, *hold=False*, ***kwargs*)
Plot new trace

show()
Show (start qt mainloop) graphs. Blocking

figure(*numb=None*)
Set current figure

add_action_on_click(*theAction*)

set_title(*theTitle*, ***kwargs*)
Set title of the plot

axis_equal()

get_all_figures()
Get all existing figures

get_wgGraphs(*fig=None*)
Advanced option. :return: `widget_graphs_visual`

class FilledContourPlot(**args*, ***kwargs*)
Bases: *ContourPlot*

class ContourPlot(**args*, ***kwargs*)
Bases: `GridPlot_Generic`

get_levels()

get_number_of_contours()

class SurfPlot(*X*, *Y*, *Z*, ***kwargs*)
Bases: `GridPlot_Generic`

class MeshPlot(*X*, *Y*, *Z*, ***kwargs*)
Bases: `GridPlot_Generic`

class ScatterPlot3(*x*, *y*, *z*, ***kwargs*)
Bases: `Plot3D_Generic`

get_plot_data()

get_color()

printIfShown(*theStr*, *show_type=SHOW_DEBUG*, *isToPrint=True*, *appendTypeName=True*, *end='n'*)

SHOW_WARNING = 0

hasPlotly = True

_do_scatterPlot(*theData: optimeed.core.ScatterPlot3*)

class Widget_graphsVisualLite(*theGraphs*, ***kwargs*)
Bases: `PyQt5.QtWidgets.QWidget`

Widget element to draw a graph. The traces and graphs to draw are defined in `Graphs` taken as argument. This widget is linked to the excellent third-party library pyqtgraph, under MIT license

signal_must_update

signal_graph_changed

set_graph_disposition(*indexGraph*, *row=1*, *col=1*, *rowspan=1*, *colspan=1*)
Change the graphs disposition.

Parameters

- **indexGraph** – index of the graph to change
- **row** – row where to place the graph
- **col** – column where to place the graph
- **rowspan** – number of rows across which the graph spans
- **colspan** – number of columns across which the graph spans

Returns

__create_graph(*idGraph*)

__check_graphs()

on_click(*plotDataItem*, *clicked_points*)

update_graphs(*singleUpdate=True*)
This method is used to update the graph. This is fast but NOT safe (especially when working with threads). To limit the risks, please use self.signal_must_update.emit() instead.

Parameters singleUpdate – if set to False, the graph will periodically refres each self.refreshtime

fast_update()
Use this method to update the graph in a fast way. NOT THREAD SAFE.

select_folder_and_export()

exportGraphs(*filename*)
Export the graphs

export_txt(*filename_txt*)

export_svg(*filename*)

export_tikz(*foldername_tikz*)

link_axes()

get_graph(*idGraph*) → optimeed.visualize.graphs.graphVisual.GraphVisual
Get corresponding `GraphVisual` of the graph idGraph

get_trace(*idGraph*, *idTrace*) → optimeed.visualize.graphs.traceVisual.TraceVisual
Get corresponding Tracevisual

keyPressEvent(*event*)
What happens if a key is pressed. R: reset the axes to their default value

delete_graph(*idGraph*)
Delete the graph idGraph

delete()

get_all_graphsVisual()
Return a dictionary {idGraph: `GraphVisual`}.

get_layout_buttons()
Get the QGraphicsLayout where it's possible to add buttons, etc.

set_actionOnClick(*theActionOnClick*)
Action to perform when the graph is clicked

Parameters theActionOnClick – `on_graph_click_interface`

Returns

set_title(*idGraph*, *titleName*, ***kwargs*)
Set title of the graph

Parameters

- **idGraph** – id of the graph
- **titleName** – title to set

class Widget_graphsVisual(**args*, ***kwargs*)
Bases: *Widget_graphsVisualLite*

Create a gui for pyqtgraph with trace selection options, export and action on clic choices

refreshTraceList()
Refresh all the traces

set_actions_on_click(*actions*)

class Onclick_representDevice(*theLinkDataGraph*, *visuals*)
Bases: *optimeed.visualize.onclick.onclickInterface.OnclickInterface*

On click: show informations about the points (loop through attributes)

class DataInformationVisuals

delete_visual(*theVisual*)

add_visual(*theVisual*, *theTrace*, *indexPoint*)

get_new_index()

curr_index()

graph_clicked(*theGraphVisual*, *index_graph*, *index_trace*, *indices_points*)
Action to perform when a point in the graph has been clicked: Creates new window displaying the device and its informations

get_name()

class RepresentDeviceInterface

class Onclick_animate(*theLinkDataGraph*, *theAnimation*)
Bases: *optimeed.visualize.onclick.onclickInterface.OnclickInterface*

On click: add or remove an element to animate

graph_clicked(*theGraphVisual*, *index_graph*, *index_trace*, *indices_points*)
Action to perform when a graph is clicked

Parameters

- **theGraphsVisual** – class widget_graphs_visual that has called the method
- **index_graph** – Index of the graph that has been clicked
- **index_trace** – Index of the trace that has been clicked
- **indices_points** – graph Indices of the points that have been clicked

Returns

get_name()

class Onclick_changeSymbol(*theLinkDataGraph*)
Bases: *optimeed.visualize.onclick.onclickInterface.OnclickInterface*

On Click: Change the symbol of the point that is clicked

graph_clicked(*theGraphVisual*, *index_graph*, *index_trace*, *indices_points*)
Action to perform when a graph is clicked

Parameters

- **theGraphsVisual** – class widget_graphs_visual that has called the method
- **index_graph** – Index of the graph that has been clicked
- **index_trace** – Index of the trace that has been clicked
- **indices_points** – graph Indices of the points that have been clicked

Returns

get_name()

class Onclick_copySomething(*theDataLink*, *functionStrFromDevice*)
Bases: *optimeed.visualize.onclick.onclickInterface.OnclickInterface*

On Click: copy something

graph_clicked(*the_graph_visual*, *index_graph*, *index_trace*, *indices_points*)
Action to perform when a graph is clicked

Parameters

- **theGraphsVisual** – class widget_graphs_visual that has called the method
- **index_graph** – Index of the graph that has been clicked
- **index_trace** – Index of the trace that has been clicked
- **indices_points** – graph Indices of the points that have been clicked

Returns

get_name()

class Onclick_delete(*theDataLink*)
Bases: *optimeed.visualize.onclick.onclickInterface.OnclickInterface*

On Click: Delete the points from the graph

graph_clicked(*_theGraphVisual*, *index_graph*, *index_trace*, *indices_points*)
Action to perform when a graph is clicked

Parameters

- **theGraphsVisual** – class widget_graphs_visual that has called the method
- **index_graph** – Index of the graph that has been clicked
- **index_trace** – Index of the trace that has been clicked
- **indices_points** – graph Indices of the points that have been clicked

Returns

get_name()

class Onclick_exportCollection(*theDataLink*)
Bases: *optimeed.visualize.onclick.onclickInterface.OnclickInterface*

On click: export the selected points

graph_clicked(*theGraphVisual*, *index_graph*, *index_trace*, *indices_points*)
Action to perform when a graph is clicked

Parameters

- **theGraphsVisual** – class widget_graphs_visual that has called the method
- **index_graph** – Index of the graph that has been clicked
- **index_trace** – Index of the trace that has been clicked
- **indices_points** – graph Indices of the points that have been clicked

Returns

reset_graph()

get_name()

class Onclick_exportToTxt(*theDataLink*, *attributes_shadow=None*)
Bases: *optimeed.visualize.onclick.onclickInterface.OnclickInterface*

On click: export the data of the whole the trace selected

graph_clicked(*theGraphVisual*, *index_graph*, *index_trace*, *indices_points*)
Action to perform when a graph is clicked

Parameters

- **theGraphsVisual** – class widget_graphs_visual that has called the method
- **index_graph** – Index of the graph that has been clicked
- **index_trace** – Index of the trace that has been clicked
- **indices_points** – graph Indices of the points that have been clicked

Returns

get_name()

class Onclick_exportTrace(*theDataLink*, *getShadow=True*)
Bases: *optimeed.visualize.onclick.onclickInterface.OnclickInterface*

On click: export the data of the whole the trace selected

graph_clicked(*theGraphVisual*, *index_graph*, *index_trace*, *indices_points*)
Action to perform when a graph is clicked

Parameters

- **theGraphsVisual** – class widget_graphs_visual that has called the method
- **index_graph** – Index of the graph that has been clicked
- **index_trace** – Index of the trace that has been clicked
- **indices_points** – graph Indices of the points that have been clicked

Returns

get_name()

class Onclick_extractPareto(*theDataLink*, *max_x=False*, *max_y=False*)
Bases: *optimeed.visualize.onclick.onclickInterface.OnclickInterface*

On click: extract the pareto from the cloud of points

graph_clicked(*the_graph_visual*, *index_graph*, *index_trace*, *_*)
Action to perform when a graph is clicked

Parameters

- **theGraphsVisual** – class widget_graphs_visual that has called the method
- **index_graph** – Index of the graph that has been clicked
- **index_trace** – Index of the trace that has been clicked
- **indices_points** – graph Indices of the points that have been clicked

Returns

get_name()

class Onclick_measure
Bases: *optimeed.visualize.onclick.onclickInterface.OnclickInterface*

On Click: Measure distance. Click on two points to perform that action

graph_clicked(*the_graph_visual*, *index_graph*, *index_trace*, *indices_points*)
Action to perform when a graph is clicked

Parameters

- **theGraphsVisual** – class widget_graphs_visual that has called the method
- **index_graph** – Index of the graph that has been clicked
- **index_trace** – Index of the trace that has been clicked
- **indices_points** – graph Indices of the points that have been clicked

Returns

reset_distance()

display_distance()

get_name()

class Onclick_removeTrace(*theDataLink*)
Bases: *optimeed.visualize.onclick.onclickInterface.OnclickInterface*

Interface class for the action to perform when a point is clicked

graph_clicked(*theGraphVisual*, *index_graph*, *index_trace*, *_*)
Action to perform when a graph is clicked

Parameters

- **theGraphsVisual** – class widget_graphs_visual that has called the method
- **index_graph** – Index of the graph that has been clicked
- **index_trace** – Index of the trace that has been clicked
- **indices_points** – graph Indices of the points that have been clicked

Returns

get_name()

class Onclick_tojson(*theDataLink*)
Bases: *optimeed.visualize.onclick.onclickInterface.OnclickInterface*

Interface class for the action to perform when a point is clicked

graph_clicked(*theGraphVisual*, *index_graph*, *index_trace*, *indices_points*)
Action to perform when a graph is clicked

Parameters

- **theGraphsVisual** – class widget_graphs_visual that has called the method
- **index_graph** – Index of the graph that has been clicked
- **index_trace** – Index of the trace that has been clicked
- **indices_points** – graph Indices of the points that have been clicked

Returns

get_name()

class OnclickInterface
Interface class for the action to perform when a point is clicked

class Represent_opengl(*DeviceDrawer*)
Bases: *optimeed.visualize.onclick.onclick_representDevice.RepresentDeviceInterface*

get_widget(*theNewDevice*)
Get Qt widget that represents the device

Parameters **theDevice** – the Device to be represented

Returns Qt widget

class Represent_image(*get_base_64_from_device*)
Bases: *optimeed.visualize.onclick.onclick_representDevice.RepresentDeviceInterface*

get_widget(*theNewDevice*)
Get Qt widget that represents the device

Parameters **theDevice** – the Device to be represented

Returns Qt widget

class Represent_lines(*attribute_lines*)
Bases: *optimeed.visualize.onclick.onclick_representDevice.RepresentDeviceInterface*

get_widget(*theNewDevice*)
Get Qt widget that represents the device

Parameters **theDevice** – the Device to be represented

Returns Qt widget

class Represent_brut_attributes(*is_light=True*, *convertToHtml=True*, *recursion_level=5*)
Bases: *optimeed.visualize.onclick.onclick_representDevice.RepresentDeviceInterface*

get_widget(*theNewDevice*)
Get Qt widget that represents the device

Parameters **theDevice** – the Device to be represented

Returns Qt widget

class Represent_txt_function(*is_light=True*, *convertToHtml=True*)
Bases: *optimeed.visualize.onclick.onclick_representDevice.RepresentDeviceInterface*

getTxt(*theNewDevice*)

get_widget(*theNewDevice*)
Get Qt widget that represents the device

Parameters **theDevice** – the Device to be represented

Returns Qt widget

class Animate_lines(*get_lines_method*, *is_light=True*, *theId=0*, *window_title='Animation'*)
Bases: *optimeed.visualize.onclick.animationGUI.AnimationGUI*

Implements `DataAnimationVisuals` to show drawing made out of lines (`widget_line_drawer`)

export_widget(*painter*)
Render scene with a painter

Parameters **painter** – PyQt painter

delete_key_widgets(*key*)
What to do when a key has to be deleted

Parameters **key** – key of the trace that has to be deleted

update_widget_w_animation(*key*, *index*, *the_data_animation*)
What to do when a new element has to be animated. Example: self.theOpenGLWidget.set_deviceToDraw(the_data_animation.get_element_animations(0, index))

Parameters

- **key** – key of the trace that has to be animated
- **index** – index that has to be animated
- **the_data_animation** – `DataAnimationTrace` that has to be animated

get_interesting_elements(*devices_list*)
Function called upon new trace creation. From a list, takes the interesting elements for animation :param element_list: :return: new_element_list

class Animate_openGL(*theOpenGLWidget*, *theId=0*, *window_title='Animation'*)
Bases: *optimeed.visualize.onclick.animationGUI.AnimationGUI*

Implements `DataAnimationVisuals` to show opengl drawing

update_widget_w_animation(*key*, *index*, *the_data_animation*)
What to do when a new element has to be animated. Example: self.theOpenGLWidget.set_deviceToDraw(the_data_animation.get_element_animations(0, index))

Parameters

- **key** – key of the trace that has to be animated
- **index** – index that has to be animated
- **the_data_animation** – `DataAnimationTrace` that has to be animated

export_widget(*painter*)
Render scene with a painter

Parameters **painter** – PyQt painter

delete_key_widgets(*key*)
What to do when a key has to be deleted

Parameters **key** – key of the trace that has to be deleted

class Animate_lines_and_text(**args*, ***kwargs*)
Bases: *Animate_lines*

Same as DataAnimationLines but also with text

update_widget_w_animation(*key*, *index*, *the_data_animation*)
What to do when a new element has to be animated. Example: self.theOpenGLWidget.set_deviceToDraw(the_data_animation.get_element_animations(0, index))

Parameters

- **key** – key of the trace that has to be animated
- **index** – index that has to be animated
- **the_data_animation** – DataAnimationTrace that has to be animated

class Animate_openGL_and_text(**args*, *is_light=True*, ***kwargs*)
Bases: *Animate_openGL*

Implements DataAnimationVisuals to show opengl drawing and text

update_widget_w_animation(*key*, *index*, *the_data_animation*)
What to do when a new element has to be animated. Example: self.theOpenGLWidget.set_deviceToDraw(the_data_animation.get_element_animations(0, index))

Parameters

- **key** – key of the trace that has to be animated
- **index** – index that has to be animated
- **the_data_animation** – DataAnimationTrace that has to be animated

get_interesting_elements(*devices_list*)
Function called upon new trace creation. From a list, takes the interesting elements for animation :param element_list: :return: new_element_list

class DeviceDrawerInterface

keyboard_push_action(*theKey*)

get_colour_scalebar()

get_colour_background()

get_opengl_options()

class MaterialRenderingProperties(*amb3*, *dif3*, *spec3*, *shin*)

__spec3__ = [0, 0, 0, 0]

__dif3__ = [0, 0, 0, 0]

__amb3__ = [0, 0, 0, 0]

__shin__ = 0

getSpec3()

getDif3()

getAmb3()

getShin()

activateMaterialProperties(*alpha=1*)

Emerald_material

Yellow_Emerald_material

Brass_material

Bronze_material

Silver_material

Steel_material

Copper_material

Chrome_material

Blue_material

Red_material

Green_material

Cyan_material

Pink_material

class OnselectInterface

class Onselect_highlight(*theLinkDataGraphs*, *theWgPlot*)

Bases: *optimeed.visualize.selector.onselectInterface.OnselectInterface*

selector_updated(*selection_name*, *the_collection*, *selected_data*, *not_selected_data*)

Action to perform once the data have been selected

Parameters

- **selection_name** – name of the selection (deprecated ?)
- **the_collection** – the collection
- **selected_data** – indices of the data selected
- **not_selected_data** – indices of the data not selected

Returns

cancel_selector(*selection_identifier*)

Action to perform when data stopped being selected :param selection_identifier: identifier that was returned by selector_updated :return:

get_name()

Get the name of the action

Returns string

class Onselect_newTrace(*theLinkDataGraphs*)

Bases: *optimeed.visualize.selector.onselectInterface.OnselectInterface*

selector_updated(*selection_name*, *the_collection*, *selected_data*, *not_selected_data*)

Action to perform once the data have been selected

Parameters

- **selection_name** – name of the selection (deprecated ?)
- **the_collection** – the collection
- **selected_data** – indices of the data selected
- **not_selected_data** – indices of the data not selected

Returns identifier that can later be used with cancel_selector

cancel_selector(*selection_identifier*)
Action to perform when data stopped being selected :param selection_identifier: identifier that was returned by selector_updated :return:

get_name()
Get the name of the action

Returns string

class Onselect_splitTrace(*theLinkDataGraphs*)
Bases: *optimeed.visualize.selector.onselectInterface.OnselectInterface*

selector_updated(*selection_name*, *the_collection*, *selected_data*, *not_selected_data*)
Action to perform once the data have been selected

Parameters

- **selection_name** – name of the selection (deprecated ?)
- **the_collection** – the collection
- **selected_data** – indices of the data selected
- **not_selected_data** – indices of the data not selected

Returns identifier that can later be used with cancel_selector

cancel_selector(*selection_identifiers*)
Action to perform when data stopped being selected :param selection_identifier: identifier that was returned by selector_updated :return:

get_name()
Get the name of the action

Returns string

class Widget_listWithSearch(**args*, ***kwargs*)
Bases: PyQt5.QtWidgets.QWidget

get_index_selected()

get_name_selected()

set_list(*names*)

_filter_list()

_iter_items()

class Widget_image(*image_b64*)
Bases: PyQt5.QtWidgets.QLabel

eventFilter(*source*, *event*)

set_image(*image_b64*)
Set new image to widget

class Widget_lineDrawer(*minWinHeight=300*, *minWinWidth=300*, *is_light=True*)
Bases: PyQt5.QtWidgets.QWidget

Widget allowing to display several lines easily

signal_must_update

on_update_signal(*listOfLines*)

delete_lines(*key_id*)
Dele the lines :param key_id: id to delete :return:

set_lines(*listOfLines*, *key_id=0*, *pen=None*)
Set the lines to display :param listOfLines: list of [x1, y1, x2, y2] corresponding to lines :param key_id: id of the trace :param pen: pen used to draw the lines :return:

paintEvent(*event*, *painter=None*)

get_extrema_lines()

class Widget_menuButton(*theParentButton*)
Bases: PyQt5.QtWidgets.QMenu

Same as QMenu, but integrates it behind a button more easily.

showEvent(*QShowEvent*)

mouseReleaseEvent(*QMouseEvent*)

class Widget_openGL(*parent=None*)
Bases: PyQt5.QtWidgets.QOpenGLWidget

Interface that provides opengl capabilities. Ensures zoom, light, rotation, etc.

sizeHint()

minimumSizeHint()

set_deviceDrawer(*theDeviceDrawer*)
Set a drawer optimeed.visualize.widgets.openGL.deviceDrawerInterface.DeviceDrawerInterface

set_deviceToDraw(*theDeviceToDraw*)
Set the device to draw

initializeGL()

paintGL()

resizeGL(*w*, *h*)

mousePressEvent(*event*)

mouseMoveEvent(*event*)

keyPressEvent(*event*)

wheelEvent(*QWheelEvent*)

class Widget_tableWithSearch(**args*, ***kwargs*)
Bases: PyQt5.QtWidgets.QWidget

cellChanged

hideRow(*row*)

showRow(*row*)

force_hide_row(*row*)

remove_forced_hide_row(*row*)

get_entries_selected()

_cellChanged()

set_entries(*names*, *numColumns=3*, *hidden=False*)

get_shown_entries()

set_item(*row*, *col*, *item*)

get_item(*row*, *col*)

_filter_list()

_iter_items()

class Widget_text(*theText*, *is_light=False*, *convertToHtml=False*)

Bases: `PyQt5.QtWidgets.QLabel`

Widget able to display a text

set_text(*theText*, *convertToHtml=False*)

Set the text to display

class Widget_text_scrollable(*theText*, *is_light=False*, *convertToHtml=False*)

Bases: `PyQt5.QtWidgets.QWidget`

Same as `widget_text` but scrollable

set_text(*theText*, *convertToHtml=False*)

6.1.2 Package Contents

VERSION = 2.1.1

CHAPTER 7

Developer guide

7.1 Developer documentation

7.1.1 Packages for doc:

- *pip install sphinx*
- *pip install sphinx-autoapi*
- *pip install sphinx_rtd_theme*

7.1.2 To regenerate API:

- uncomment line # 'autoapi.extension' in conf.py.
- run make html
- run hack.py script
- recomment line # 'autoapi.extension'
- run make html
- Eventually update project on https://readthedocs.org/projects/optimeed/

7.1.3 To updata packages on PyPi:

- Change version in setup.py and in optimeed/__init__.py
- Create new wheel file code::*python setup.py sdist bdist_wheel*
- Upload it on pypi code::*twine upload dist/**

Python Module Index

o

Index

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

Y