

---

# **YAOCPTool Documentation**

***Release 0.2.0***

**Marco Aurelio S. de Aguiar**

**Oct 16, 2018**



# CONTENTS

<b>1</b>	<b>yaocptool package</b>	<b>1</b>
1.1	Subpackages	1
1.1.1	yaocptool. estimation package	1
1.1.1.1	Submodules	1
1.1.1.2	yaocptool. estimation. estimator_ abstract module	1
1.1.1.3	yaocptool. estimation. extended_ kalman_ filter module	1
1.1.1.4	yaocptool. estimation. pce_ kalman_ filter module	2
1.1.1.5	yaocptool. estimation. unscented_ kalman_ filter module	2
1.1.1.6	Module contents	3
1.1.2	yaocptool. methods package	3
1.1.2.1	Subpackages	3
1.1.2.2	Submodules	11
1.1.2.3	yaocptool. methods. augmented_ lagrangian module	11
1.1.2.4	yaocptool. methods. distributed_ solution module	12
1.1.2.5	Module contents	12
1.1.3	yaocptool. modelling package	12
1.1.3.1	Submodules	12
1.1.3.2	yaocptool. modelling. dae_ system module	12
1.1.3.3	yaocptool. modelling. dataset module	13
1.1.3.4	yaocptool. modelling. network module	15
1.1.3.5	yaocptool. modelling. node module	15
1.1.3.6	yaocptool. modelling. ocp module	15
1.1.3.7	yaocptool. modelling. simualtion_ result module	18
1.1.3.8	yaocptool. modelling. stochastic_ ocp module	18
1.1.3.9	yaocptool. modelling. system_ model module	19
1.1.3.10	Module contents	23
1.1.4	yaocptool. mpc package	23
1.1.4.1	Submodules	23
1.1.4.2	yaocptool. mpc. mpc module	23
1.1.4.3	yaocptool. mpc. plant module	25
1.1.4.4	Module contents	26
1.1.5	yaocptool. optimization package	26
1.1.5.1	Submodules	26
1.1.5.2	yaocptool. optimization. abstract_ optimization_ problem module	26
1.1.5.3	yaocptool. optimization. nonlinear_ problem module	28
1.1.5.4	yaocptool. optimization. quadratic_ problem module	28
1.1.5.5	Module contents	28
1.1.6	yaocptool. parallel package	28
1.1.6.1	Submodules	28
1.1.6.2	yaocptool. parallel. worker module	28

1.1.6.3	Module contents . . . . .	29
1.1.7	yaocptool.stochastic package . . . . .	29
1.1.7.1	Submodules . . . . .	29
1.1.7.2	yaocptool.stochastic.pce module . . . . .	29
1.1.7.3	yaocptool.stochastic.util module . . . . .	29
1.1.7.4	Module contents . . . . .	30
1.1.8	yaocptool.util package . . . . .	30
1.1.8.1	Submodules . . . . .	30
1.1.8.2	yaocptool.util.util module . . . . .	30
1.1.8.3	Module contents . . . . .	31
1.2	Submodules . . . . .	31
1.3	yaocptool.config module . . . . .	31
1.4	Module contents . . . . .	31
<b>Python Module Index</b>		<b>33</b>
<b>Index</b>		<b>35</b>

## YAOCPTOOL PACKAGE

### 1.1 Subpackages

#### 1.1.1 yaocptool.estimation package

##### 1.1.1.1 Submodules

##### 1.1.1.2 yaocptool.estimation.estimator\_abstract module

```
class yaocptool.estimation.estimator_abstract.EstimatorAbstract (**kwargs)

    __init__ (**kwargs)
    estimate (t_k, y_k, u_k)
        Estimate the state given the measurement y_k and the control u_k :param DM t_k: time of the measure-
        ments :param DM y_k: measurement :param DM u_k: control
```

##### 1.1.1.3 yaocptool.estimation.extended\_kalman\_filter module

```
class yaocptool.estimation.extended_kalman_filter.ExtendedKalmanFilter (model,
                                                                    **kwargs)
    Bases: yaocptool.estimation.estimator_abstract.EstimatorAbstract
    __init__ (model, **kwargs)
        Extended Kalman Filter for ODE and DAE systems implementation based on (1)

        (1) Mandela, R. K., Narasimhan, S., & Rengaswamy, R. (2009). Nonlinear State Estimation of Differ-
        ential Algebraic Systems. Proceedings of the 2009 ADCHEM (Vol. 42). IFAC. http://doi.org/10.3182/20090712-4-TR-2008.00129
```

##### Parameters

- **model** (*SystemModel*) – filter model
- **t\_s** (*float*) – sampling time
- **t** (*float*) – current estimator time
- **x\_mean** (*DM*) – current state estimation
- **p\_k** (*DM*) – current covariance estimation
- **r\_v** (*DM*) – process noise covariance matrix
- **r\_n** (*DM*) – measurement noise covariance matrix

- **y\_guess** (*DM*) – initial guess of the algebraic variables for the model simulation

**estimate** (*t\_k, y\_k, u\_k*)

**n\_meas**  
Number of measurements

**Return type** int

**Returns** Number of measurements

#### 1.1.1.4 yaocptool.estimated.pce\_kalman\_filter module

**class** yaocptool.estimated.pce\_kalman\_filter.**PCEKalmanFilter** (*model, \*\*kwargs*)  
Bases: *yaocptool.estimated.estimator\_abstract.EstimatorAbstract*

**\_\_init\_\_** (*model, \*\*kwargs*)

**Parameters**

- **model** (*SystemModel*) – estimator model
- **x\_mean** – a initial guess for the mean
- **p\_k** – a initial guess for the covariance of the estimator
- **h\_function** – a function that receives 3 parameters (*x, y\_algebraic, u*) and returns an measurement.
- **c\_matrix** – if **h\_function** is not give, **c\_matrix** is “C” from the measurement equation:  $y_{\text{meas}} = C[x \ y_{\text{alg}}] + D \cdot u$ . note that it has to have  $n_x + n_y$  (algebraic) columns.
- **r\_v** (*DM*) – process noise matrix
- **r\_n** (*DM*) – measurement noise matrix
- **pc\_order** (*int*) – Polynomial chaos order
- **n\_samples** (*int*) – Number of samples to be used

**estimate** (*t\_k, y\_k, u\_k*)

**n\_pol\_parameters**

#### 1.1.1.5 yaocptool.estimated.unscented\_kalman\_filter module

**class** yaocptool.estimated.unscented\_kalman\_filter.**UnscentedKalmanFilter** (*model, \*\*kwargs*)  
Bases: *yaocptool.estimated.estimator\_abstract.EstimatorAbstract*

**\_\_init\_\_** (*model, \*\*kwargs*)

Unscented Kalman Filter. Two versions are implemented standard and square-root. Implemented based on [1] and [2].

References:

[1] Wan, E. A., & Van Der Merwe, R. (2000). The unscented Kalman filter for nonlinear estimation. In Proceedings of the IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium (Cat. No.00EX373) (Vol. v, pp. 153–158). IEEE. <http://doi.org/10.1109/ASSPCC.2000.882463>

[2] Merwe, R. Van Der, & Wan, E. a. (2001). The square-root unscented Kalman filter for state and parameter-estimation. 2001 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings (Cat. No.01CH37221), 6, 1–4. <http://doi.org/10.1109/ICASSP.2001.940586>

#### Parameters

- **model** (`SystemModel`) – estimator model
- **x\_mean** – a initial guess for the mean
- **p\_k** – a initial guess for the covariance of the estimator
- **h\_function** – a function that receives 3 parameters (x, y\_algebraic, u) and returns an measurement.
- **c\_matrix** – if h\_function is not give, c\_matrix is “C” from the measurement equation:  $y_{\text{meas}} = C * [x \ y_{\text{alg}}] + D * u$ . note that it has to have  $n_x + n_y$  (algebraic) columns.
- **r\_v** ( $DM$ ) – process noise matrix
- **r\_n** ( $DM$ ) – measurement noise matrix
- **implementation** – options: ‘standard’ or ‘square-root’. (default: ‘standard’)

**static cholupdate** (*r\_matrix*, *x*, *sign*)

**estimate** (*t\_k*, *y\_k*, *u\_k*)

**n\_meas**

Number of measurements

**Return type** int

**Returns** Number of measurements

#### 1.1.1.6 Module contents

### 1.1.2 yaocptool.methods package

#### 1.1.2.1 Subpackages

**yaocptool.methods.base package**

#### Submodules

**yaocptool.methods.base.discretizationschemebase module**

**class** yaocptool.methods.base.discretizationschemebase.**DiscretizationSchemeBase** (*solution\_method*)

**\_\_init\_\_** (*solution\_method*)

Base class for discretization methods. A discretization class transforms and OCP into a NLP

**create\_initial\_guess** (*p=None*, *theta=None*)

Create an initial guess for the optimal control problem using problem.x\_0, problem.y\_guess, problem.u\_guess, and a given p and theta (for p\_opt and theta\_opt) if they are given. If y\_guess or u\_guess are None the initial guess uses a vector of zeros of appropriate size. If no p or theta is given, an vector of zeros o appropriate size is used.

**Parameters**

- **p** – Optimization parameters
- **theta** – Optimization theta

**Returns**

**create\_initial\_guess\_with\_simulation** (*u=None, p=None, theta=None*)

Create an initial guess for the optimal control problem using by simulating with a given control *u*, and a given *p* and *theta* (for *p\_opt* and *theta\_opt*) if they are given. If no *u* is given the value of *problem.u\_guess* is used, or *problem.u\_past*, then a vector of zeros of appropriate size is used. If no *p* or *theta* is given, an vector of zeros o appropriate size is used.

**Parameters**

- **u** – Control initial guess
- **p** – Optimization parameters
- **theta** – Optimization theta

**Returns**

**create\_nlp\_symbolic\_variables** (*nlp*)

Create the symbolic variables that will be used by the NLP problem :rtype: (DM, List[List[DM]], List(List(DM)), List(DM), DM, DM, DM)

**degree**

**degree\_control**

**delta\_t**

**discretize** (*x\_0=None, p=None, theta=None, last\_u=None*)

Discretize the OCP, returning a Optimization Problem

**Parameters**

- **x\_0** – initial condition
- **p** – parameters
- **theta** – theta parameters
- **last\_u** – last applied control

**Returns**

**finite\_elements**

**get\_system\_at\_given\_times** (*x, y, u, time\_dict=None, p=None, theta=None, functions=None, start\_at\_t\_0=False*)

**model**

**Return type** *SystemModel*

**problem**

**Return type** *OptimalControlProblem*

**set\_data\_to\_optimization\_result\_from\_raw\_data** (*optimization\_result, raw\_solution\_dict*)

Set the raw data received from the solver and put it in the Optimization Result object :type optimization\_result: yaocptool.methods.optimizationresult.OptimizationResult :type raw\_solution\_dict: dict

**split\_x\_and\_u** (*results\_vector, all\_subinterval=False*)



**split\_x\_y\_and\_u** (*v*, *all\_subinterval=False*)

**time\_breakpoints**

**unpack\_decision\_variables** (*decision\_variables*)  
 Return a structured data from the decision variables vector

Returns: (*x\_data*, *y\_data*, *u\_data*, *p\_opt*, *eta*)

**Parameters** *decision\_variables* – DM

**Returns** tuple

**vectorize** (*vector*)

## yaocptool.methods.base.optimizationresult module

**class** yaocptool.methods.base.optimizationresult.**OptimizationResult** (*\*\*kwargs*)

**\_\_init\_\_** (*\*\*kwargs*)

**dataset**

**first\_control** ()  
 Return the first element of the control vector

**Return type** DM

**is\_collocation**

**is\_valid**

**plot** (*plot\_list*, *figures=None*, *show=True*)  
 Plot the optimization result. It takes as input a list of dictionaries, each dictionary represents a plot. In the dictionary use keyword ‘x’ to specify which states you want to print, the value of the dictionary should be a list of state to be printed. The keywords that are accepted are: ‘x’, ‘y’, ‘u’ :param list *plot\_list*: List of dictionaries to generate the plots. :param list *figures*: OPTIONAL: list of figures to be plotted in. If not provided it will create new figures. :param bool *show*: OPTIONAL: select if matplotlib.pyplot.show should be applied after the plots.

**to\_dataset** ()  
 Return a dataset with the data of x, y, and u

**Return type** *DataSet*

## yaocptool.methods.base.solutionmethodsbase module

**class** yaocptool.methods.base.solutionmethodsbase.**SolutionMethodsBase** (*problem*, *\*\*kwargs*)

Bases: object

**\_\_init\_\_** (*problem*, *\*\*kwargs*)

**Parameters**

- **problem** (*OptimalControlProblem*) –
- **integrator\_type** (*str*) – str
- **solution\_method** (*str*) – str

- **degree** (*int*) – discretization polynomial degree
- **degree\_control** (*int*) –
- **discretization\_scheme** (*str*) – ('multiple-shooting' | 'collocation')
- **initial\_guess\_heuristic** (*str*) – 'simulation' or 'problem\_info'

**call\_solver** (*initial\_guess=None, p=None, theta=None, x\_0=None, last\_u=None, initial\_guess\_dict=None*)

**static collocation\_points** (*degree, cp='radau', with\_zero=False*)

**create\_control\_approximation** ()

Parametrize the control variable, accordingly to the 'degree\_control' attribute. If degree\_control == 1, then a piecewise constant control will be used (most common). If degree\_control > 1, then a piecewise polynomial approximation will be used with order 'degree\_control'.

#### Returns

**create\_optimization\_problem** ()

**create\_optimization\_result** (*raw\_solution\_dict, p=None, theta=None, x\_0=None*)

**create\_variable\_polynomial\_approximation** (*size, degree, name='var\_appr', tau=None, point\_at\_t0=False*)

**delta\_t**

**model**

**prepare** ()

**solve** (*initial\_guess=None, p=None, theta=None, x\_0=None, last\_u=None, initial\_guess\_dict=None*)

#### Parameters

- **initial\_guess** – Initial guess
- **p** – Parameters values
- **theta** – Theta values
- **x\_0** – Initial condition value
- **last\_u** – Last control value
- **initial\_guess\_dict** – Initial guess as dict

#### Returns OptimizationResult

**split\_x\_and\_u**

**split\_x\_y\_and\_u**

**time\_breakpoints**

**unvec** (*vector, degree=None*)

Unvectorize 'vector' a vectorized matrix, assuming that it was a matrix with 'degree' number of columns  
:type vector: DM a vector (flattened matrix) :type degree: int

## Module contents

### yaocptool.methods.classic package

### Submodules

### yaocptool.methods.classic.collocationscheme module

Created on Thu Jul 13 17:08:34 2017

@author: marco

**class** yaocptool.methods.classic.collocationscheme.**CollocationScheme** (*solution\_method*)

Bases: *yaocptool.methods.base.discretizationschemebase.DiscretizationSchemeBase*

**create\_initial\_guess** (*p=None, theta=None*)

Create an initial guess for the optimal control problem using `problem.x_0`, `problem.y_guess`, `problem.u_guess`, and a given `p` and `theta` (for `p_opt` and `theta_opt`) if they are given. If `y_guess` or `u_guess` are `None` the initial guess uses a vector of zeros of appropriate size.

#### Parameters

- **p** – Optimization parameters
- **theta** – Optimization theta

#### Returns

**create\_initial\_guess\_with\_simulation** (*u=None, p=None, theta=None*)

Create an initial guess for the optimal control problem using by simulating with a given control `u`, and a given `p` and `theta` (for `p_opt` and `theta_opt`) if they are given. If no `u` is given the value of `problem.u_guess` is used, or `problem.last_u`, then a vector of zeros of appropriate size is used. If no `p` or `theta` is given, an vector of zeros of appropriate size is used.

#### Parameters

- **u** –
- **p** – Optimization parameters
- **theta** – Optimization theta

#### Returns

**create\_nlp\_symbolic\_variables** (*nlp*)

Create the symbolic variables that will be used by the NLP problem :param `NonlinearOptimizationProblem` `nlp`: nonlinear optimization problem in which the variables will be created :rtype: tuple

**discretize** (*x\_0=None, p=None, theta=None, last\_u=None*)

Discretize the OCP, returning a Optimization Problem

#### Parameters

- **x\_0** – initial condition
- **p** – parameters
- **theta** – theta parameters
- **last\_u** – last applied control

**Return type** *NonlinearOptimizationProblem*

**get\_system\_at\_given\_times** (*x, y, u, time\_dict=None, p=None, theta=None, functions=None, start\_at\_t\_0=False*)

**Parameters**

- **x** (*list*) –
- **y** (*list*) –
- **u** (*list*) –
- **time\_dict** (*dict*) – Dictionary of simulations times, where the KEY is the finite\_element and the VALUE list a list of desired times example : {1:{‘t\_0’: 0.0, ‘x’:[0.0, 0.1, 0.2], y:[0.2]}}
- **p** – list
- **theta** – dict
- **start\_at\_t\_0** – bool If TRUE the simulations in each finite\_element will start at the element t\_0, Otherwise the simulation will start the end of the previous element
- **functions** (*dict*) – Dictionary of Functions to be evaluated, KEY is the function identifier, VALUE is a CasADi Function with model.all\_sym as input

**set\_data\_to\_optimization\_result\_from\_raw\_data** (*optimization\_result, raw\_solution\_dict*)

Set the raw data received from the solver and put it in the Optimization Result object :type optimization\_result: yaocptool.methods.base.optimizationresult.OptimizationResult :type raw\_solution\_dict: dict

**split\_x\_y\_and\_u** (*decision\_variables, all\_subinterval=False*)

**Parameters**

- **all\_subinterval** – Bool ‘Returns all elements of the subinterval (or only the first one)’
- **decision\_variables** – DM

**Returns** X, Y, and U -> list with a DM for each element

**time\_interpolation**

**time\_interpolation\_algebraics**

**time\_interpolation\_controls**

**time\_interpolation\_states**

**unpack\_decision\_variables** (*decision\_variables, all\_subinterval=True*)

Return a structured data from the decision variables vector

Returns: (x\_data, y\_data, u\_data, p\_opt, eta)

**Parameters**

- **decision\_variables** – DM
- **all\_subinterval** – bool

**Returns** tuple

## yaocptool.methods.classic.directmethod module

Created on Fri Oct 21 16:40:15 2016

@author: marco

```
class yaocptool.methods.classic.directmethod.DirectMethod (problem, **kwargs)
    Bases: yaocptool.methods.base.solutionmethodsbase.SolutionMethodsBase
```

```
    __init__ (problem, **kwargs)
```

#### Parameters

- **problem** – yaocptool.modelling.ocp.OptimalControlProblem
- **integrator\_type** – str
- **solution\_method** – str
- **degree** – int
- **discretization\_scheme** – str ‘multiple-shooting’ | ‘collocation’

```
    prepare ()
```

### yaocptool.methods.classic.indirectmethod module

Created on Fri Oct 21 16:39:52 2016

@author: marco

```
class yaocptool.methods.classic.indirectmethod.IndirectMethod (problem,
                                                                **kwargs)
    Bases: yaocptool.methods.base.solutionmethodsbase.SolutionMethodsBase
```

```
    __init__ (problem, **kwargs)
```

#### Parameters

- **problem** – yaocptool.modelling.ocp.OptimalControlProblem
- **integrator\_type** – str
- **solution\_method** – str
- **degree** – int
- **discretization\_scheme** – str ‘multiple-shooting’ | ‘collocation’

```
    calculate_optimal_control ()
```

```
    include_adjoint_states ()
```

```
    prepare ()
```

```
    replace_with_optimal_control (u_opt)
```

### yaocptool.methods.classic.multipleshooting module

Created on Thu Jul 13 17:08:34 2017

@author: marco

```
class yaocptool.methods.classic.multipleshooting.MultipleShootingScheme (solution_method)
    Bases: yaocptool.methods.base.discretizationschemebase.
```

```
    DiscretizationSchemeBase
```

**create\_initial\_guess** (*p=None, theta=None*)

Create an initial guess for the optimal control problem using `problem.x_0`, `problem.y_guess`, `problem.u_guess`, and a given `p` and `theta` (for `p_opt` and `theta_opt`) if they are given. If `y_guess` or `u_guess` are `None` the initial guess uses a vector of zeros of appropriate size.

**Parameters**

- **p** – Optimization parameters
- **theta** – Optimization theta

**Returns**

**create\_initial\_guess\_with\_simulation** (*u=None, p=None, theta=None*)

Create an initial guess for the optimal control problem using by simulating with a given control `u`, and a given `p` and `theta` (for `p_opt` and `theta_opt`) if they are given. If no `u` is given the value of `problem.u_guess` is used, or `problem.last_u`, then a vector of zeros of appropriate size is used. If no `p` or `theta` is given, an vector of zeros o appropriate size is used.

**Parameters**

- **u** – Control initial guess
- **p** – Optimization parameters
- **theta** – Optimization theta

**Returns**

**create\_nlp\_symbolic\_variables** (*nlp*)

Create the symbolic variables that will be used by the NLP problem :rtype: (MX, List[List[MX]], List[List[MX]], List[MX], MX, MX, List[MX])

**discretize** (*x\_0=None, p=None, theta=None, last\_u=None*)

Discretize the OCP, returning a Optimization Problem

**Parameters**

- **x\_0** – initial condition
- **p** – parameters
- **theta** – theta parameters
- **last\_u** – last applied control

**Return type** *NonlinearOptimizationProblem*

**get\_system\_at\_given\_times** (*x\_var, y\_var, u\_var, time\_dict=None, p=None, theta=None, functions=None, start\_at\_t\_0=False*)

**Parameters**

- **x\_var** – List[List[MX]]
- **y\_var** – List[List[MX]]
- **u\_var** – List[List[MX]]
- **p** – list
- **theta** – dict
- **start\_at\_t\_0** – bool If TRUE the simulations in each finite\_element will start at the element `t_0`, Otherwise the simulation will start the end of the previous element
- **functions** – Dict[str, Function|Dict[int]] dictionary of Functions to be evaluated, KEY is the function identifier, VALUE is a CasADi Function with `model.all_sym` as input

**set\_data\_to\_optimization\_result\_from\_raw\_data** (*optimization\_result*,  
*raw\_solution\_dict*)

Set the raw data received from the solver and put it in the Optimization Result object :param optimization\_result: OptimizationResult :param raw\_solution\_dict: dict

**split\_x\_y\_and\_u** (*results\_vector*, *all\_subinterval=False*)

**Parameters**

- **all\_subinterval** – Bool ‘Returns all elements of the subinterval (or only the first one)’
- **results\_vector** – DM

**Returns** X, Y, and U -> list with a DM for each element

**unpack\_decision\_variables** (*decision\_variables*)

Return a structured data from the decision variables vector

Returns: (x\_data, y\_data, u\_data, p\_opt, eta, theta\_opt)

**Parameters** **decision\_variables** – DM

**Returns** tuple

## Module contents

### 1.1.2.2 Submodules

#### 1.1.2.3 yaocptool.methods.augmented\_lagrangian module

Created on Sat Oct 22 16:53:36 2016

@author: marco

```
class yaocptool.methods.augmented_lagrangian.AugmentedLagrangian (problem,  

                                                                ocp_solver_class,  

                                                                solver_options=None,  

                                                                **kwargs)
```

Bases: *yaocptool.methods.base.solutionmethodsbases.SolutionMethodsBase*

**For a minimization problem in the form**  $\min f(x,u) = \int L(x,u) dt$  s.t.:  $\dot{x} = f(x,u)$ ,  $g_{ineq}(x,u) \leq 0$

**Transforms the problem in a sequence of solution of the problem**  $\min f(x,u) = \int L(x,u) - \mu \sum \log(-g_{ineq}(x,u)) dt$  s.t.:  $\dot{x} = f(x,u)$ ,

```
__init__ (problem, ocp_solver_class, solver_options=None, **kwargs)
```

**Parameters**

- **problem** (*OptimalControlProblem*) –
- **integrator\_type** (*str*) – str
- **solution\_method** (*str*) – str
- **degree** (*int*) – discretization polynomial degree
- **degree\_control** (*int*) –
- **discretization\_scheme** (*str*) – ('multiple-shooting' | 'collocation')
- **initial\_guess\_heuristic** (*str*) – 'simulation' or 'problem\_info'

```
create_nu_initial_guess (n_r=None)
```

```
create_optimization_result (raw_solution_dict, p=None, theta=None, x_0=None)
static join_nu_to_theta (theta, nu)
model
relaxed_alg
solve_raw (initial_guess=None, p=None, theta=None, x_0=None, last_u=None)
split_x_and_u
split_x_y_and_u
time_interpolation_nu
```

#### 1.1.2.4 yaocptool.methods.distributed\_solution module

```
class yaocptool.methods.distributed_solution.DistributedSolution (subsystem_classes_list,
                                                                parameters_list, con-
                                                                nection_list,
                                                                main_to_subsystems_list=None,
                                                                **kwargs)

__init__ (subsystem_classes_list, parameters_list, connection_list, main_to_subsystems_list=None,
          **kwargs)
initialize ()
kill ()
n_subsystems
solve (x_0, initial_guess_dict=None)
start ()
```

#### 1.1.2.5 Module contents

### 1.1.3 yaocptool.modelling package

#### 1.1.3.1 Submodules

#### 1.1.3.2 yaocptool.modelling.dae\_system module

```
class yaocptool.modelling.dae_system.DAESystem (**kwargs)
```

```
__init__ (**kwargs)
```

##### Parameters

- **ode** – ODE equations
- **alg** – Algebraic equations
- **x** – State variables
- **y** – Algebraic variables
- **p** – Parameters



- **t** – Time variable
- **tau** – Tau variable

**convert\_from\_tau\_to\_time** (*t<sub>k</sub>, t<sub>kpl</sub>*)

**dae\_system\_dict**

**has\_parameters**

**is\_dae**

**is\_ode**

**join** (*dae\_sys*)

**simulate** (*x<sub>0</sub>, t<sub>f</sub>, t<sub>0</sub>=0, p=None, y<sub>0</sub>=None, integrator\_type='implicit', integrator\_options=None*)

**substitute\_variable** (*old\_var, new\_var*)

**type**

### 1.1.3.3 yaocptool.modelling.dataset module

**class** yaocptool.modelling.dataset.**DataSet** (*name='dataset', \*\*kwargs*)

**\_\_init\_\_** (*name='dataset', \*\*kwargs*)

Generic time dependent data storage. The data is stored in the self.data dictionary. self.data['entry\_name']['time'] is a row vector self.data['entry\_name']['values'] is a matrix with the same number of columns as the time vector, and rows equal to self.data['entry\_name']['size']. The data can be more easily managed using create\_entry, get\_entry, insert\_data.

#### Parameters

- **name** (*str*) – name of th dataset
- **plot\_style** (*str*) – default plot style. plot = linear interpolation, step = piecewise constant ('plot' | 'step')
- **find\_discontinuity** (*bool*) – Default: True. If True, it will try to find discontinuity on the data, and plot with gaps where data is missing/not available, instead of a line connecting all data points.
- **max\_sampling\_time** (*float*) – maximum expected distance between two time data. This is used to detect discontinuity on the data, and plot it separately.

**create\_entry** (*entry, size, names=None, plot\_style=None*)

Create an entry in the dataset

#### Parameters

- **entry** – entry name
- **size** – number of rows in the vector
- **names** (*list*) – name for each row, it should be a list with size 'size'. If 'names' is not given, then the name list [entry\_1, entry\_2, ..., entry\_size]
- **plot\_style** (*str*) – ('plot' | 'step') choose if the plot will be piecewise constant (step) or a first order interpolation (plot).

**extend** (*other\_dataset*)

Extend this DataSet with another DataSet. They don't need to be ordered, after the merging a chronological sort of the data is performed.

**Parameters** **other\_dataset** (*DataSet*) –

**Returns**

**get\_copy** ()

Return a copy of this dataset. The copy is not connected to the original data set, therefore changes in one of the dataset will not affect the other.

**Returns**

**get\_entry** (*entry*)

Return the time and values for a given entry.

**Parameters** **entry** (*str*) – entry name

**Returns** entry time, entry value

**Return type** tuple

**get\_entry\_names** (*entry*)

Get list of names of an entry

**Parameters** **entry** –

**Return type** list

**get\_entry\_size** (*entry*)

Get size of an entry

**Parameters** **entry** –

**Returns**

**insert\_data** (*entry, time, value*)

**plot** (*plot\_list, figures=None, show=True*)

Plot DataSet information. It takes as input a list of dictionaries, each dictionary represents a plot. In the dictionary use keyword 'x' to specify which states you want to print, the value of the dictionary should be a list of index of the states to be printed.

**Parameters**

- **plot\_list** (*list*) – List of dictionaries to generate the plots.
- **figures** (*list*) – list of figures to be plotted on top (optional)
- **show** (*bool*) – if the plotted figures should be shown after plotting (optional, default=True).

**sort** (*entries=None*)

Sort the dataset for given 'entries' in an chronological order, this can be used when data is not inserted in an ordered fashion.

**Parameters** **entries** (*list*) – list of entries to be sorted, if this parameter is no given all the entries will be sorted.

#### 1.1.3.4 yaocptool.modelling.network module

Created on Fri Jul 07 16:05:50 2017

@author: marco

```
class yaocptool.modelling.network.Connection (connection_id, nodes=None,  
                                              z_sym_indices_list=None)
```

```
    __init__ (connection_id, nodes=None, z_sym_indices_list=None)
```

Connection class for Network systems

```
class yaocptool.modelling.network.Network (nodes, connections_settings_dict)
```

```
    __init__ (nodes, connections_settings_dict)
```

**Example of connections\_dict:** Connections 0:  $z\_sym[0]$  in Node 0 is connected to  $z\_sym[2]$  in Node 1  
`connections_dict = {0: [{0:[0]}, {1:[2]}]}`

**it can also be a multidimensional connection:** Connection 1:  $z\_sym[0:1]$  in Node 1 is connected to  
 $z\_sym[1:2]$  in Node 4 `connections_dict = {1: [{1:[0,1]}, {4:[1,2]}]}`

```
    create_connections ()
```

```
    get_connection_defined_z ()
```

```
    get_connection_equations ()
```

```
    models
```

```
    problems
```

#### 1.1.3.5 yaocptool.modelling.node module

Created on Fri Jul 07 16:33:52 2017

@author: marco

```
class yaocptool.modelling.node.Node (node_id=-1, name="", model=None, problem=None,  
                                     **kwargs)
```

```
    __init__ (node_id=-1, name="", model=None, problem=None, **kwargs)
```

#### 1.1.3.6 yaocptool.modelling.ocp module

Created on Mon Apr 03 11:15:03 2017

@author: marco

**class** yaocptool.modelling.ocp.**OptimalControlProblem**(*model*, *\*\*kwargs*)

Optimal Control Problem class, used to define a optimal control problem based on a model (SystemModel) It has the following form:

$$\begin{aligned} \min J &= V(x(t_f), p) + \int_{t_0}^{t_f} L(x, y, u, t, p, \theta) + \sum_k S(x(t_k), y(t_k), u(t_k), t_k, p, \theta_k) \\ \text{s.t.} \quad \dot{x} &= f(x, y, u, t, p, \theta) \\ g(x, y, u, t, p, \theta) &= 0 \\ g_{eq}(x, y, u, t, p, \theta) &= 0 \\ g_{ineq}(x, y, u, t, p, \theta) &\leq 0 \\ x_{min} &\leq x \leq x_{max} \\ y_{min} &\leq y \leq y_{max} \\ u_{min} &\leq u \leq u_{max} \\ \Delta u_{min} &\leq \Delta u \leq \Delta u_{max} \\ h_{initial}(x(t_0), t_0, p, heta) &= 0 \\ h_{final}(x(t_f), t_f, p, heta) &= 0 \\ h(p) &= 0 \end{aligned}$$

where  $t_k$  is final time in each finite element.

**\_\_init\_\_**(*model*, *\*\*kwargs*)

**create\_algebraic**(*name*, *size*, *alg=None*, *y\_max=None*, *y\_min=None*, *y\_guess=None*)

**create\_cost\_state**()

Transforms the integral  $\int_{t_0}^{t_f} L(\dots) dt$  into a dynamic cost state:  $\dot{x}_c = L(\dots)$  and include  $x_c(t_f)$  into the final time cost ( $V(t_f) += x_c(t_f)$ ).

**Return type** object

**create\_optimization\_parameter**(*name*, *size=1*, *p\_opt\_min=None*, *p\_opt\_max=None*)

**create\_optimization\_theta**(*name*, *size=1*, *new\_theta\_opt\_min=None*, *new\_theta\_opt\_max=None*)

**create\_parameter**(*name*, *size=1*)

**create\_quadratic\_cost**(*par\_dict*)

**create\_state**(*name*, *size*, *ode=None*, *x\_0=None*, *var\_max=None*, *var\_min=None*, *h\_initial=None*)

**get\_p\_opt\_indices**()

**get\_theta\_opt\_indices**()

**has\_delta\_u**

**include\_algebraic**(*var*, *alg=None*, *y\_min=None*, *y\_max=None*, *y\_guess=None*)

**include\_control**(*var*, *u\_min=None*, *u\_max=None*)

**include\_equality**(*eq*)

Include time independent equality. Equality is concatenated “h”.

**Parameters** **eq** – time independent equality

**include\_final\_time\_equality**(*eq*)

Include final time equality. Equality that is evaluated at  $t=t_f$ . The equality is concatenated to “h\_final”

**Parameters** **eq** – final equality constraint

**include\_initial\_time\_equality** (*eq*)

Include initial time equality. Equality that is evaluated at  $t=t_0$ . The equality is concatenated to “h\_initial”

**Parameters** *eq* – initial equality constraint

**include\_optimization\_parameter** (*var*, *p\_opt\_min*=None, *p\_opt\_max*=None)

**include\_optimization\_theta** (*var*, *theta\_opt\_min*=None, *theta\_opt\_max*=None)

**include\_state** (*var*, *ode*=None, *x\_0*=None, *x\_min*=None, *x\_max*=None, *h\_initial*=None, *x\_0\_sym*=None, *suppress*=False)

**include\_system\_equations** (*ode*=None, *alg*=None)

**include\_time\_equality** (*eq*, *when*='default')

Include time dependent equality.  $g_{eq}(\dots, t) = 0$ , for  $t$  in  $[t_0, t_f]$

The inequality is concatenated to “g\_ineq”

**Parameters**

- **eq** – equality
- **when** (*str*) – Can be ‘default’, ‘end’, ‘start’. ‘start’ - the constraint will be evaluated at the start of every finite element ‘end’ - the constraint will be evaluated at the end of every finite element ‘default’ - will be evaluated at each collocation point of every finite element. For the multiple shooting, the constraint will be evaluated at the end of each finite element

**include\_time\_inequality** (*eq*, *when*='default')

Include time dependent inequality.  $g_{ineq}(\dots, t) \leq 0$ , for  $t$  in  $[t_0, t_f]$

The inequality is concatenated to “g\_ineq”

**Parameters**

- **when** (*str*) – Can be ‘default’, ‘end’, ‘start’. ‘start’ - the constraint will be evaluated at the start of every finite element ‘end’ - the constraint will be evaluated at the end of every finite element ‘default’ - will be evaluated at each collocation point of every finite element. For the multiple shooting, the constraint will be evaluated at the end of each finite element
- **eq** – inequality

**merge** (*problems*)

**n\_eta**

**n\_g\_eq**

**n\_g\_ineq**

**n\_h\_final**

**n\_h\_initial**

**n\_p\_opt**

**n\_theta\_opt**

**pre\_solve\_check** ()

**remove\_algebraic** (*var*, *eq*=None)

**remove\_control** (*var*)

**replace\_variable** (*original*, *replacement*, *variable\_type*='other')

**reset\_working\_model** ()

```
set_parameter_as_optimization_parameter (new_p_opt, new_p_opt_min=None,
                                          new_p_opt_max=None)

set_theta_as_optimization_theta (new_theta_opt, new_theta_opt_min=None,
                                 new_theta_opt_max=None)

class yaocptool.modelling.ocp.SuperOCP (problems, **kwargs)
    Bases: yaocptool.modelling.ocp.OptimalControlProblem
    __init__ (problems, **kwargs)
```

#### 1.1.3.7 yaocptool.modelling.simualtion\_result module

```
class yaocptool.modelling.simualtion_result.SimulationResult (n_x, n_y, n_u,
                                                             x_names=None,
                                                             y_names=None,
                                                             u_names=None,
                                                             **kwargs)

    Bases: yaocptool.modelling.dataset.DataSet

    __init__ (n_x, n_y, n_u, x_names=None, y_names=None, u_names=None, **kwargs)

    extend (other_sim_result)
        Extend this SimulationResult with other SimulationResult. It is only implemented for the case where the
        other simulation result starts at the end of this simulation. That is this.t_f == other_sim_result.t_0 .

        Parameters other_sim_result (SimulationResult) –
        Returns

    final_condition ()
        Return the simulation final condition in the form of a tuple (x_f, y_f, u_f)

        Return type DM, DM, DM

    n_u
    n_x
    n_y
    t
    u
    u_names
    x
    x_names
    y
    y_names
```

#### 1.1.3.8 yaocptool.modelling.stochastic\_ocp module

```
class yaocptool.modelling.stochastic_ocp.StochasticOCP (model, **kwargs)
    Bases: yaocptool.modelling.ocp.OptimalControlProblem
    __init__ (model, **kwargs)
    create_uncertain_parameter (name, mean, var, size=1, distribution='normal')
```

```

get_p_unc_indices ()
get_p_without_p_unc ()
get_uncertain_initial_cond_indices ()
include_time_chance_inequality (ineq, prob, rhs=None, when='default')
    Include time dependent chance inequality. Prob[ineq(..., t) <= rhs] <= prob, for t in [t_0, t_f]
    The inequality is concatenated to "g_stochastic_ineq"

```

#### Parameters

- **ineq** – inequality
- **rhs** (*list* / *DM*) – Right-hand size of the inequality
- **prob** (*list* / *DM*) – Chance/probability of the constraint being satisfied
- **when** (*str*) – Can be 'default', 'end', 'start'. 'start' - the constraint will be evaluated at the start of every finite element 'end' - the constraint will be evaluated at the end of every finite element 'default' - will be evaluated at each collocation point of every finite element. For the multiple shooting, the constraint will be evaluated at the end of each finite element

```

include_uncertain_parameter (par, mean, var, distribution='normal')
n_g_stochastic
n_p_unc
n_uncertain_initial_condition
set_initial_condition_as_uncertain (par, mean, cov, distribution='normal')
set_parameter_as_uncertain_parameter (par, mean, var, distribution='normal')

```

### 1.1.3.9 yaocptool.modelling.system\_model module

Created on Thu Jun 09 10:50:48 2016

@author: marco

```

class yaocptool.modelling.system_model.SystemModel (name='model', n_x=0, n_y=0,
                                                    n_u=0, n_p=0, n_theta=0,
                                                    **kwargs)

__init__ (name='model', n_x=0, n_y=0, n_u=0, n_p=0, n_theta=0, **kwargs)
    x - states y - (internal) algebraic z - external algebraic u - control p - constant parameters theta - parameters
    dependent of the finite_elements u_par - parametrized control parameters

    Note: when vectorizing the parameters order is [ p; theta; u_par]

all_sym

connect (u, y)
    Connect an input 'u' to a algebraic variable 'y', u = y. The function will perform the following actions: -
    include an algebraic equation u - y = 0 - remove 'u' from the input vector (since it is not a free variable
    anymore) - include 'u' into the algebraic vector, since it is an algebraic variable now.

```

#### Parameters

- **u** – input variable
- **y** – algebraic variable

**convert\_expr\_from\_tau\_to\_time** (*expr, t\_k, t\_kp1*)

**convert\_expr\_from\_time\_to\_tau** (*expr, t\_k, t\_kp1*)

**create\_algebraic\_variable** (*name='y', size=1*)

Create a new algebraic variable with the name “name” and size “size”. Size can be an int or a tuple (e.g. (2,2)). However, the new algebraic variable will be vectorized (casadi.vec) to be included in the algebraic vector (model.y\_sym).

**Parameters**

- **name** (*str*) –
- **size** (*int || tuple*) –

**Returns**

**create\_control** (*name='u', size=1*)

Create a new control variable name “name” and size “size”. Size can be an int or a tuple (e.g. (2,2)). However, the new control variable will be vectorized (casadi.vec) to be included in the control vector (model.u\_sym).

**Parameters**

- **name** – str
- **size** – int

**Returns**

**create\_parameter** (*name='p', size=1*)

Create a new parameter name “name” and size “size”

**Parameters**

- **name** – str
- **size** – int

**Returns**

**create\_state** (*name='x', size=1*)

Create a new state with the name “name” and size “size”. Size can be an int or a tuple (e.g. (2,2)). However, the new state will be vectorized (casadi.vec) to be included in the state vector (model.x\_sym).

**Parameters**

- **name** – str
- **size** – int|tuple

**Returns**

**create\_theta** (*name='theta', size=1*)

Create a new parameter name “name” and size “size”

**Parameters**

- **name** – str
- **size** – int

**Returns**

**find\_algebraic\_variable** (*x, u, guess=None, t=0.0, p=None, theta\_value=None, rootfinder\_options=None*)



**find\_equilibrium** (*additional\_eqs*, *guess=None*, *t\_0=0.0*, *rootfinder\_options=None*)

Find a equilibrium point for the model. This method solves the root finding problem:

$$f(x,y,u,t_0) = 0 \quad g(x,y,u,t_0) = 0 \quad \text{additional\_eqs}(x,y,u,t_0) = 0$$

Use *additional\_eqs* to specify the additional conditions remembering that  $\dim(\text{additional\_eqs}) = n_u$ , so the system can be well defined. If no initial guess is provided (“guess” parameter) a guess of ones will be used (not zero to avoid problems with singularities).

Returns *x\_0*, *y\_0*, *u\_0*

#### Parameters

- **rootfinder\_options** (*dict*) – options to be passed to rootfinder
- **additional\_eqs** – SX
- **guess** – DM
- **t\_0** – float

Returns (DM, DM, DM)

**get\_copy** ()

Get a copy of this model

**Return type** *SystemModel*

**get\_dae\_system** ()

Return a DAESystem object with the model equations.

**Returns** DAESystem

**get\_hardcopy** ()

Get a hard copy of this model, differently from “get\_copy”, the variables of the original copy and the hard copy will not be the same, i.e. `model.x_sym != copy.x_sym`

**Return type** *SystemModel*

**include\_algebraic** (*var*, *alg=None*)

**include\_control** (*var*)

**include\_models** (*models*)

**include\_parameter** (*p*)

**include\_state** (*var*, *ode=None*, *x\_0\_sym=None*)

**include\_system\_equations** (*ode=None*, *alg=None*)

Include model equations, (ordinary) differential equation and algebraic equation (ode and alg)

#### Parameters

- **ode** – (ordinary) differential equation
- **alg** – algebraic equation

**include\_theta** (*theta*)

**lamb\_sym**

**linearize** (*x\_bar, u\_bar*)

Returns a linearized model at a given points (X\_BAR, U\_BAR)

**merge** (*models\_list, connecting\_equations=None*)

**n\_p**

**n\_theta**

**n\_u**

**n\_x**

**n\_y**

**p**

**p\_names**

**print\_variables** ()

Print list of variable in the model (x, y, u, p, theta)

**static put\_values\_in\_all\_sym\_format** (*t=None, x=None, y=None, u=None, p=None, theta=None, u\_par=None*)

**remove\_algebraic** (*var, eq=None*)

**remove\_control** (*var*)

**remove\_parameter** (*var*)

**remove\_theta** (*var*)

**replace\_variable** (*original, replacement*)

Replace a variable or parameter by an variable or expression.

#### Parameters

- **replacement** –
- **original** – SX: and replacement, and also variable type which describes which type of variable is being remove to it from the counters. Types: 'x', 'y', 'u', 'p', 'ignore'

**simulate** (*x\_0, t\_f, t\_0=0.0, u=None, p=None, theta=None, y\_0=None, integrator\_type='implicit', integrator\_options=None*)

**Simulate model.** If *t\_f* is a float, then only one simulation will be done. If *t\_f* is a list of times, then a sequence of simulations will be done, that each *t\_f* is the end of a finite element.

#### Parameters

- **x\_0** (*list* || *DM*) – Initial condition
- **t\_f** (*float* || *list*) – Final time of the simulation, can be a list of final times for sequential simulation
- **t\_0** (*float*) – Initial time
- **u** (*list* || *DM*) – Controls of the system to be simulated
- **p** (*DM* || *SX* || *list*) – Simulation parameters
- **theta** (*dict*) – Parameters theta, which varies for each simulation for sequential simulations. If *t\_f* is a list then theta has to have one entry for each *k* in  $[0, \dots, \text{len}(t_f)]$
- **y\_0** – Initial guess for the algebraic variables
- **integrator\_type** (*str*) – 'implicit' or 'explicit'

- **integrator\_options** (*dict*) – options to be passed to the integrator

Return type *SimulationResult*

**system\_type**

**theta**

**theta\_names**

**u**

**u\_names**

**x**

**x\_names**

**x\_sys\_sym**

**y**

**y\_names**

#### 1.1.3.10 Module contents

Created on Mon Jul 10 14:27:21 2017

@author: marco

### 1.1.4 yaocptool.mpc package

#### 1.1.4.1 Submodules

#### 1.1.4.2 yaocptool.mpc.mpc module

**class** `yaocptool.mpc.mpc.MPC` (*plant, solution\_method, \*\*kwargs*)

**\_\_init\_\_** (*plant, solution\_method, \*\*kwargs*)

Model Predictive control class. Requires a plant and a solution\_method.

##### Parameters

- **plant** (*Plant/PlantSimulation*) –
- **solution\_method** (*SolutionMethodsBase*) –
- **estimator** (*EstimatorAbstract*) –
- **default\_p** (*DM*) – is a default parameter vector that will be used by the ‘solution\_method’
- **default\_theta** (*DM*) – is a default theta parameter that will be used by the ‘solution\_method’
- **include\_cost\_in\_state\_vector** (*bool*) – Typically the optimal control problem has one extra state than the plant, the dynamic cost state. By setting this variable to True, it automatically include an zero the in state vector obtained from the estimator.

- **mean\_as\_parameter** (*bool*) – The mean estimated by the Estimator as parameter of the OCP. It will be put in the end of parameter vector, before the covariance if covariance\_as\_parameter=True.
- **covariance\_as\_parameter** (*bool*) – The covariance estimated by the Estimator as parameter of the OCP. It will be put in the end of parameter vector.
- **mean\_p\_indices** (*list*) – If mean\_as\_parameter=True, mean\_p\_indices is a list of tuples (pairs), where the first element of the tuple is the index in the mean vector and the second is the index in the p vector.
- **cov\_p\_indices** (*list*) – If covariance\_as\_parameter=True, cov\_p\_indices is a list of tuples (pairs), where the first element of the tuple is the index in the vectorized covariance matrix and the second is the index in the p vector.
- **state\_rearrangement\_function** (*function*) – A function that can be used to rearrange the initial condition in cases where the estimated states is not equal to initial condition vector of the OCP. For instance, when the OCP has multiple representations of the system. The provided function has the estimated states as input and has to return a initial condition vector.

**get\_measurement ()**

Get measurements from the plant. It will return a tuple with the current measurement and the current control

**Return type** tuple

**get\_new\_control** (*x\_k, u\_k, p=None*)

Use solution\_method to obtain new controls

**Parameters**

- **x\_k** – DM
- **u\_k** – DM
- **p** –

**get\_states** (*t\_k, y\_k, u\_k*)

Get states out of a measurement.

**Parameters**

- **t\_k** (*DM*) – time of the measurement
- **y\_k** (*DM*) – measurements
- **u\_k** (*DM*) – controls

**Returns** DM

**run** (*iterations=0*)

Starts computing control and sending it to the plant.

**Parameters** **iterations** (*int*) – the number of iterations that the MPC will run. To run it indefinitely use iterations = 0

**run\_fixed\_control** (*u, iterations*)

Run the plant with a fixed control, can be used for initialization purposes.

**Parameters**

- **u** (*list | DM | float | int*) – control value
- **iterations** (*int*) – the number of iterations that the MPC will run.

**run\_fixed\_control\_with\_estimator** (*u, iterations*)

Run the plant with a fixed control, can be used for initialization purposes.

**Parameters**

- **u** (*list / DM / float / int*) – control value
- **iterations** (*int*) – the number of iterations that the MPC will run.

**send\_control** (*u*)

Sent controls to the plant.

**Parameters** **u** – DM

### 1.1.4.3 yaocptool.mpc.plant module

**class** yaocptool.mpc.plant.Plant

**\_\_init\_\_** ()

**get\_measurement** ()

**n\_x**

**set\_control** (*u*)

**class** yaocptool.mpc.plant.PlantSimulation (*model, x\_0, \*\*kwargs*)

Bases: *yaocptool.mpc.plant.Plant*

Simulates a plant using a model.

**\_\_init\_\_** (*model, x\_0, \*\*kwargs*)

Plant which uses a SystemModel.simulate to obtain the measurements.

**Parameters**

- **model** (*SystemModel*) – simulation model
- **x\_0** (*DM*) – initial condition
- **t\_s** (*DM*) – (default: 1) sampling time
- **u** (*DM*) – (default: 0) initial control
- **y\_guess** (*DM*) – initial guess for algebraic variables for simulation
- **t\_0** (*DM*) – (default: 0) initial time
- **integrator\_options** (*dict*) – integrator options
- **has\_noise** (*bool*) – Turn on/off the process/measurement noise
- **r\_n** (*DM*) – Measurement noise covariance matrix
- **r\_v** (*DM*) – Process noise covariance matrix
- **noise\_seed** – Seed for the random number generator used to create noise. Use the same seed for the repeatability in the experiments.

**get\_measurement** ()

Return the plant measurement of a simulated model and advance time by 't\_s'. Return the measurement time, the measurement [x; y], and the controls.

**Return type** tuple

**Returns** (timestamp, measurement, control)

**n\_x**

**set\_control** (*u*)

set a new control for the plant

**Parameters** *u* (*DM*) – new control vector

#### 1.1.4.4 Module contents

### 1.1.5 yaocptool.optimization package

#### 1.1.5.1 Submodules

#### 1.1.5.2 yaocptool.optimization.abstract\_optimization\_problem module

**class** yaocptool.optimization.abstract\_optimization\_problem.**AbstractOptimizationProblem** (*\*\*kwargs*)

Bases: object

**\_\_init\_\_** (*\*\*kwargs*)

Abstract Optimization Problem class Optimization problem

$$\min_x f(x, p)$$

$$\text{s.t.} : g_{lb} \leq g(x, p) \leq g_{ub}$$

*Object attributes* : *x* – > *optimization variables* *g* – > *constraint*

**Parameters** *n\_x* – int

**create\_parameter** (*name*, *size=1*)

**create\_variable** (*name*, *size=1*, *lb=-inf*, *ub=inf*)

Create an optimization variable

**Parameters**

- **name** (*str*) – Name of the optimization variable.
- **size** (*int*) – Size of the variable (default = 1)
- **lb** – Lower bound of the variable. If the given ‘size’ is greater than one but a scalar is passed as lower bound, a vector of lb of size ‘size’ will be used as a lower bound. (default = [-inf]\*size)
- **ub** – Upper bound of the variable. If the given ‘size’ is greater than one but a scalar is passed as upper bound, a vector of ub of size ‘size’ will be used as an upper bound. (default = [inf]\*size)

**Returns** Return the variable

**Return type** MX

**get\_default\_call\_dict** ()

**get\_problem\_dict** ()

**get\_solver** ()

**include\_equality** (*expr*, *rhs=None*)

Include an equality with the following form *expr* = *rhs*

**Parameters**

- **expr** – expression, this is the only term that should contain symbolic variables
- **rhs** – right hand side, by default it is a vector of zeros with same size of **expr**. If the ‘**expr**’ size is greater than one but a scalar is passed as ‘**rhs**’, a vector of ‘**rhs**’ with size of ‘**expr**’ will be used as right hand side. (default = [0]\*size)

**include\_inequality** (*expr, lb=None, ub=None*)

Include inequality to the problem with the following form  $lb \leq expr \leq ub$

**Parameters**

- **expr** – expression for the inequality, this is the only term that should contain symbolic variables
- **lb** – Lower bound of the inequality. If the ‘**expr**’ size is greater than one but a scalar is passed as lower bound, a vector of **lb** with size of ‘**expr**’ will be used as a lower bound. (default = [-inf]\*size)
- **ub** – Upper bound of the inequality. If the ‘**expr**’ size is greater than one but a scalar is passed as upper bound, a vector of **ub** with size of ‘**expr**’ will be used as an upper bound. (default = [inf]\*size)

**include\_parameter** (*par*)

**include\_variable** (*variable, lb=-inf, ub=inf*)

Include a symbolic variable in the optimization problem

**Parameters**

- **variable** – variable to be included
- **lb** – Lower bound of the variable. If the given variable size is greater than one but a scalar is passed as lower bound, a vector of **lb** with size of the given variable will be used as a lower bound. (default = [-inf]\*size)
- **ub** – Upper bound of the variable. If the given variable size is greater than one but a scalar is passed as upper bound, a vector of **ub** with size of the given variable will be used as an upper bound. (default = [inf]\*size)

**set\_objective** (*expr*)

**solve** (*initial\_guess=None, call\_dict=None, p=None, lam\_x=None, lam\_g=None*)

**Parameters**

- **initial\_guess** – Initial guess
- **call\_dict** – a dictionary containing ‘**lbx**’, ‘**ubx**’, ‘**lbg**’, ‘**ubg**’. If not given, the one obtained with `self.get_default_call_dict` will be used.
- **p** – parameters
- **lam\_x** –
- **lam\_g** –

**Returns** dictionary with solution

### 1.1.5.3 yaooptool.optimization.nonlinear\_problem module

```
class yaooptool.optimization.nonlinear_problem.NonlinearOptimizationProblem (**kwargs)
    Bases: yaooptool.optimization.abstract_optimization_problem.
            AbstractOptimizationProblem
```

```
    __init__ (**kwargs)
```

Nonlinear Optimization Problem class Optimization problem

$$\begin{aligned} & \min_x f(x, p) \\ \text{s.t.: } & g_{lb} \leq g(x, p) \leq g_{ub} \end{aligned}$$

Object attributes: x -> optimization variables g -> constraint

### 1.1.5.4 yaooptool.optimization.quadratic\_problem module

```
class yaooptool.optimization.quadratic_problem.QuadraticOptimizationProblem (**kwargs)
    Bases: yaooptool.optimization.abstract_optimization_problem.
            AbstractOptimizationProblem
```

```
    __init__ (**kwargs)
```

Quadratic Optimization Problem class Optimization problem

$$\begin{aligned} & \min_x f(x, p) \\ \text{s.t.: } & g_{lb} \leq g(x, p) \leq g_{ub} \end{aligned}$$

Object attributes: x -> optimization variables g -> constraint

### 1.1.5.5 Module contents

## 1.1.6 yaooptool.parallel package

### 1.1.6.1 Submodules

#### 1.1.6.2 yaooptool.parallel.worker module

```
class yaooptool.parallel.worker.Worker (obj_class, obj_arg, function_name, queue_in,
                                         queue_out)
```

Bases: multiprocessing.process.Process

Creates new process that creates and object of class 'obj\_class' with 'obj\_arg' argument. It will consume one element from each queue\_in and call function 'function\_name' the consumed elements as argument. It will put the return of the 'function\_name' call in all Queues in queue\_out

```
    __init__ (obj_class, obj_arg, function_name, queue_in, queue_out)
        x.__init__(...) initializes x; see help(type(x)) for signature
```

```
    run ()
```

Method to be run in sub-process; can be overridden in sub-class



### 1.1.6.3 Module contents

## 1.1.7 yaocptool.stochastic package

### 1.1.7.1 Submodules

### 1.1.7.2 yaocptool.stochastic.pce module

**class** yaocptool.stochastic.pce.**PCEConverter** (*socp*, *\*\*kwargs*)

**\_\_init\_\_** (*socp*, *\*\*kwargs*)

#### Parameters

- **socp** (*StochasticOCP*) – Stochastic Optimal Control Problem
- **n\_samples** (*int*) – number of samples of the parameters. If none is provided, the minimum number of samples will be used, depending on the number of uncertain parameters and polynomial order
- **pc\_order** (*int*) – order of the polynomial, for the polynomial approximation. (default: 3)

**convert\_socp\_to\_ocp\_with\_pce** ()

**n\_pol\_parameters**

**n\_uncertain**

yaocptool.stochastic.pce.**get\_ls\_factor** (*n\_uncertain*, *n\_samples*, *pc\_order*, *lamb=0.0*)

### 1.1.7.3 yaocptool.stochastic.util module

yaocptool.stochastic.util.**sample\_parameter\_log\_normal\_distribution\_with\_sobol** (*mean*, *co-variance*, *n\_samples=1*)

Sample parameter using Sobol sampling with a log-normal distribution.

#### Parameters

- **mean** –
- **covariance** –
- **n\_samples** –

#### Returns

yaocptool.stochastic.util.**sample\_parameter\_normal\_distribution\_with\_sobol** (*mean*, *co-variance*, *n\_samples=1*)

Sample parameter using Sobol sampling with a normal distribution.

#### Parameters

- **mean** –

- **covariance** –
- **n\_samples** –

**Returns**

#### 1.1.7.4 Module contents

### 1.1.8 yaocptool.util package

#### 1.1.8.1 Submodules

#### 1.1.8.2 yaocptool.util.util module

`yaocptool.util.util.blockdiag` (*matrices\_list*)

Receives a list of matrices and return a block diagonal.

**Parameters** `matrices_list` (*list*) – list of matrices

`yaocptool.util.util.convert_expr_from_tau_to_time` (*expr, t\_sym, tau\_sym, t\_k, t\_kpl*)

`yaocptool.util.util.create_constant_theta` (*constant, dimension, finite\_elements*)

`yaocptool.util.util.expm` (*a\_matrix*)

Since casadi does not have native support for matrix exponential, this is a trick to computing it. It can be quite expensive, specially for large matrices. THIS ONLY SUPPORT NUMERIC MATRICES AND MX VARIABLES, DOES NOT SUPPORT SX SYMBOLIC VARIABLES.

**Parameters** `a_matrix` (*DM*) – matrix

**Returns**

`yaocptool.util.util.find_variables_indices_in_vector` (*var, vector*)

`yaocptool.util.util.join_thetas` (*\*args*)

`yaocptool.util.util.remove_variables_from_vector` (*var, vector*)

Returns a vector with items removed

**Parameters**

- **var** – items to be removed
- **vector** – vector which will have items removed

**Returns**

`yaocptool.util.util.remove_variables_from_vector_by_indices` (*vector, indices*)

Returns a vector with items removed

**Parameters**

- **vector** – vector which will have items removed
- **indices** (*list*) – list of indices for which the variables need to be removed.

**Returns**

#### 1.1.8.3 Module contents

## 1.2 Submodules

## 1.3 yaocptool.config module

Created on Wed Nov 16 15:27:37 2016

@author: marco

## 1.4 Module contents



## PYTHON MODULE INDEX

### y

- yaocptool, 31
- yaocptool.config, 31
- yaocptool.estimation, 3
  - yaocptool.estimation.estimator\_abstract, 1
  - yaocptool.estimation.extended\_kalman\_filter, 1
  - yaocptool.estimation.pce\_kalman\_filter, 2
  - yaocptool.estimation.unscented\_kalman\_filter, 2
- yaocptool.methods, 12
  - yaocptool.methods.augmented\_lagrangian, 11
  - yaocptool.methods.base, 7
    - yaocptool.methods.base.discretizationschemebase, 3
    - yaocptool.methods.base.optimizationresult, 5
    - yaocptool.methods.base.solutionmethodsbase, 5
  - yaocptool.methods.classic, 11
    - yaocptool.methods.classic.collocationscheme, 7
    - yaocptool.methods.classic.directmethod, 8
    - yaocptool.methods.classic.indirectmethod, 9
    - yaocptool.methods.classic.multipleshooting, 9
  - yaocptool.methods.distributed\_solution, 12
- yaocptool.modelling, 23
  - yaocptool.modelling.dae\_system, 12
  - yaocptool.modelling.dataset, 13
  - yaocptool.modelling.network, 15
  - yaocptool.modelling.node, 15
  - yaocptool.modelling.ocp, 15
  - yaocptool.modelling.simualtion\_result, 18
  - yaocptool.modelling.stochastic\_ocp, 18
  - yaocptool.modelling.system\_model, 19
- yaocptool.mpc, 26
  - yaocptool.mpc.mpc, 23
  - yaocptool.mpc.plant, 25
- yaocptool.optimization, 28
  - yaocptool.optimization.abstract\_optimization\_problem, 26
  - yaocptool.optimization.nonlinear\_problem, 28
  - yaocptool.optimization.quadratic\_problem, 28
- yaocptool.parallel, 29
  - yaocptool.parallel.worker, 28
- yaocptool.stochastic, 30
  - yaocptool.stochastic.pce, 29
  - yaocptool.stochastic.util, 29
- yaocptool.util, 31
  - yaocptool.util.util, 30



## Symbols

[\\_\\_init\\_\\_\(\) \(yaocptool.estimation.estimator\\_abstract.EstimatorAbstract method\), 1](#)  
[\\_\\_init\\_\\_\(\) \(yaocptool.estimation.extended\\_kalman\\_filter.ExtendedKalmanFilter method\), 1](#)  
[\\_\\_init\\_\\_\(\) \(yaocptool.estimation.pce\\_kalman\\_filter.PCEKalmanFilter method\), 2](#)  
[\\_\\_init\\_\\_\(\) \(yaocptool.estimation.unscented\\_kalman\\_filter.UnscentedKalmanFilter method\), 2](#)  
[\\_\\_init\\_\\_\(\) \(yaocptool.methods.augmented\\_lagrangian.AugmentedLagrangian method\), 11](#)  
[\\_\\_init\\_\\_\(\) \(yaocptool.methods.base.discretizationschemebase.DiscretizationSchemeBase method\), 3](#)  
[\\_\\_init\\_\\_\(\) \(yaocptool.methods.base.optimizationresult.OptimizationResult method\), 5](#)  
[\\_\\_init\\_\\_\(\) \(yaocptool.methods.base.solutionmethodsbases.SolutionMethodsBase method\), 5](#)  
[\\_\\_init\\_\\_\(\) \(yaocptool.methods.classic.directmethod.DirectMethod method\), 9](#)  
[\\_\\_init\\_\\_\(\) \(yaocptool.methods.classic.indirectmethod.IndirectMethod method\), 9](#)  
[\\_\\_init\\_\\_\(\) \(yaocptool.methods.distributed\\_solution.DistributedSolution method\), 12](#)  
[\\_\\_init\\_\\_\(\) \(yaocptool.modelling.dae\\_system.DAESystem method\), 12](#)  
[\\_\\_init\\_\\_\(\) \(yaocptool.modelling.dataset.DataSet method\), 13](#)  
[\\_\\_init\\_\\_\(\) \(yaocptool.modelling.network.Connection method\), 15](#)  
[\\_\\_init\\_\\_\(\) \(yaocptool.modelling.network.Network method\), 15](#)  
[\\_\\_init\\_\\_\(\) \(yaocptool.modelling.node.Node method\), 15](#)  
[\\_\\_init\\_\\_\(\) \(yaocptool.modelling.ocp.OptimalControlProblem method\), 16](#)  
[\\_\\_init\\_\\_\(\) \(yaocptool.modelling.ocp.SuperOCP method\), 18](#)  
[\\_\\_init\\_\\_\(\) \(yaocptool.modelling.simualtion\\_result.SimulationResult method\), 18](#)  
[\\_\\_init\\_\\_\(\) \(yaocptool.modelling.stochastic\\_ocp.StochasticOCP method\), 18](#)  
[\\_\\_init\\_\\_\(\) \(yaocptool.modelling.system\\_model.SystemModel method\), 19](#)  
[\\_\\_init\\_\\_\(\) \(yaocptool.mpc.mpc.MPC method\), 23](#)  
[\\_\\_init\\_\\_\(\) \(yaocptool.mpc.plant.Plant method\), 25](#)  
[\\_\\_init\\_\\_\(\) \(yaocptool.mpc.plant.PlantSimulation method\), 25](#)  
[\\_\\_init\\_\\_\(\) \(yaocptool.optimization.abstract\\_optimization\\_problem.AbstractOptimizationProblem method\), 26](#)  
[\\_\\_init\\_\\_\(\) \(yaocptool.optimization.nonlinear\\_problem.NonlinearOptimizationProblem method\), 28](#)  
[\\_\\_init\\_\\_\(\) \(yaocptool.optimization.quadratic\\_problem.QuadraticOptimizationProblem method\), 28](#)  
[\\_\\_init\\_\\_\(\) \(yaocptool.parallel.worker.Worker method\), 28](#)  
[\\_\\_init\\_\\_\(\) \(yaocptool.stochastic.pce.PCEConverter method\), 29](#)  
**A**  
[AbstractOptimizationProblem \(class in yaocptool.optimization.abstract\\_optimization\\_problem\), 26](#)  
[all\\_sym \(yaocptool.modelling.system\\_model.SystemModel attribute\), 19](#)  
[AugmentedLagrangian \(class in yaocptool.methods.augmented\\_lagrangian\), 11](#)  
**B**  
[blockdiag\(\) \(in module yaocptool.util.util\), 30](#)  
**C**  
[calculate\\_optimal\\_control\(\) \(yaocptool.methods.classic.indirectmethod.IndirectMethod method\), 9](#)  
[call\\_solver\(\) \(yaocptool.methods.base.solutionmethodsbases.SolutionMethodsBase method\), 6](#)  
[cholupdate\(\) \(yaocptool.estimation.unscented\\_kalman\\_filter.UnscentedKalmanFilter static method\), 3](#)  
[collocation\\_points\(\) \(yaocptool.methods.base.solutionmethodsbases.SolutionMethodsBase static method\), 6](#)  
[CollocationScheme \(class in yaocptool.methods.classic.collocationscheme\), 7](#)  
[connect\(\) \(yaocptool.modelling.system\\_model.SystemModel method\), 19](#)

Connection (class in yaocptool.modelling.network), 15  
 convert\_expr\_from\_tau\_to\_time() (in module yaocptool.util.util), 30  
 convert\_expr\_from\_tau\_to\_time() (yaocptool.modelling.system\_model.SystemModel method), 19  
 convert\_expr\_from\_time\_to\_tau() (yaocptool.modelling.system\_model.SystemModel method), 20  
 convert\_from\_tau\_to\_time() (yaocptool.modelling.dae\_system.DAESystem method), 13  
 convert\_socp\_to\_ocp\_with\_pce() (yaocptool.stochastic.pce.PCEConverter method), 29  
 create\_algebraic() (yaocptool.modelling.ocp.OptimalControlProblem method), 16  
 create\_algebraic\_variable() (yaocptool.modelling.system\_model.SystemModel method), 20  
 create\_connections() (yaocptool.modelling.network.Network method), 15  
 create\_constant\_theta() (in module yaocptool.util.util), 30  
 create\_control() (yaocptool.modelling.system\_model.SystemModel method), 20  
 create\_control\_approximation() (yaocptool.methods.base.solutionmethodsbase.SolutionMethodsBase method), 6  
 create\_cost\_state() (yaocptool.modelling.ocp.OptimalControlProblem method), 16  
 create\_entry() (yaocptool.modelling.dataset.DataSet method), 13  
 create\_initial\_guess() (yaocptool.methods.base.discretizationschemebase.DiscretizationSchemeBase method), 3  
 create\_initial\_guess() (yaocptool.methods.classic.collocationscheme.CollocationScheme method), 7  
 create\_initial\_guess() (yaocptool.methods.classic.multipleshooting.MultipleShootingScheme method), 9  
 create\_initial\_guess\_with\_simulation() (yaocptool.methods.base.discretizationschemebase.DiscretizationSchemeBase method), 4  
 create\_initial\_guess\_with\_simulation() (yaocptool.methods.classic.collocationscheme.CollocationScheme method), 7  
 create\_initial\_guess\_with\_simulation() (yaocptool.methods.classic.multipleshooting.MultipleShootingScheme method), 10  
 create\_nlp\_symbolic\_variables() (yaocptool.methods.base.discretizationschemebase.DiscretizationScheme method), 4  
 create\_nlp\_symbolic\_variables() (yaocptool.methods.classic.collocationscheme.CollocationScheme method), 7  
 create\_nlp\_symbolic\_variables() (yaocptool.methods.classic.multipleshooting.MultipleShootingScheme method), 10  
 create\_nu\_initial\_guess() (yaocptool.methods.augmented\_lagrangian.AugmentedLagrangian method), 11  
 create\_optimization\_parameter() (yaocptool.modelling.ocp.OptimalControlProblem method), 16  
 create\_optimization\_problem() (yaocptool.methods.base.solutionmethodsbase.SolutionMethodsBase method), 6  
 create\_optimization\_result() (yaocptool.methods.augmented\_lagrangian.AugmentedLagrangian method), 11  
 create\_optimization\_result() (yaocptool.methods.base.solutionmethodsbase.SolutionMethodsBase method), 6  
 create\_optimization\_theta() (yaocptool.modelling.ocp.OptimalControlProblem method), 16  
 create\_parameter() (yaocptool.modelling.ocp.OptimalControlProblem method), 16  
 create\_parameter() (yaocptool.modelling.system\_model.SystemModel method), 20  
 create\_parameter() (yaocptool.optimization.abstract\_optimization\_problem.AbstractOptimizationProblem method), 26  
 create\_quadratic\_cost() (yaocptool.modelling.ocp.OptimalControlProblem method), 16  
 create\_state() (yaocptool.modelling.ocp.OptimalControlProblem method), 16  
 create\_state() (yaocptool.modelling.system\_model.SystemModel method), 20  
 create\_state() (yaocptool.modelling.system\_model.SystemModel method), 20  
 create\_uncertain\_parameter() (yaocptool.modelling.stochastic\_ocp.StochasticOCP method), 18  
 create\_variable() (yaocptool.optimization.abstract\_optimization\_problem.AbstractOptimizationProblem method), 26  
 create\_variable\_polynomial\_approximation() (yaocptool.methods.base.solutionmethodsbase.SolutionMethodsBase method), 6



## D

dae\_system\_dict (yaocptool.modelling.dae\_system.DAESystem attribute), 13

DAESystem (class in yaocptool.modelling.dae\_system), 12

DataSet (class in yaocptool.modelling.dataset), 13

dataset (yaocptool.methods.base.optimizationresult.OptimizationResult attribute), 5

degree (yaocptool.methods.base.discretizationschemebase.DiscretizationSchemeBase attribute), 4

degree\_control (yaocptool.methods.base.discretizationschemebase.DiscretizationSchemeBase attribute), 4

delta\_t (yaocptool.methods.base.discretizationschemebase.DiscretizationSchemeBase attribute), 4

delta\_t (yaocptool.methods.base.solutionmethodsbase.SolutionMethodsBase attribute), 6

DirectMethod (class in yaocptool.methods.classic.directmethod), 9

DiscretizationSchemeBase (class in yaocptool.methods.base.discretizationschemebase), 3

discretize() (yaocptool.methods.base.discretizationschemebase.DiscretizationSchemeBase method), 4

discretize() (yaocptool.methods.classic.collocationscheme.CollocationScheme method), 7

discretize() (yaocptool.methods.classic.multipleshooting.MultipleShootingScheme method), 10

DistributedSolution (class in yaocptool.methods.distributed\_solution), 12

## E

estimate() (yaocptool.estation.estimator\_abstract.EstimatorAbstract method), 1

estimate() (yaocptool.estation.extended\_kalman\_filter.ExtendedKalmanFilter method), 2

estimate() (yaocptool.estation.pce\_kalman\_filter.PCEKalmanFilter method), 2

estimate() (yaocptool.estation.unscented\_kalman\_filter.UnscentedKalmanFilter method), 3

EstimatorAbstract (class in yaocptool.estation.estimator\_abstract), 1

expm() (in module yaocptool.util.util), 30

extend() (yaocptool.modelling.dataset.DataSet method), 13

extend() (yaocptool.modelling.simualtion\_result.SimulationResult method), 18

ExtendedKalmanFilter (class in yaocptool.estation.extended\_kalman\_filter), 1

## F

final\_condition() (yaocptool.modelling.simualtion\_result.SimulationResult

method), 18

find\_algebraic\_variable() (yaocptool.modelling.system\_model.SystemModel method), 20

find\_equilibrium() (yaocptool.modelling.system\_model.SystemModel method), 20

find\_variables\_indices\_in\_vector() (in module yaocptool.util.util), 30

finite\_elements (yaocptool.methods.base.discretizationschemebase.DiscretizationSchemeBase attribute), 4

first\_control() (yaocptool.methods.base.optimizationresult.OptimizationResult method), 5

**G**

get\_connection\_defined\_z() (yaocptool.modelling.network.Network method), 15

get\_connection\_equations() (yaocptool.modelling.network.Network method), 15

get\_discretization\_scheme() (yaocptool.modelling.dataset.DataSet method), 14

get\_location() (yaocptool.modelling.system\_model.SystemModel method), 21

get\_model\_system() (yaocptool.modelling.system\_model.SystemModel method), 21

get\_default\_call\_dict() (yaocptool.optimization.abstract\_optimization\_problem.AbstractOptimizationProblem method), 26

get\_entry() (yaocptool.modelling.dataset.DataSet method), 14

get\_entry\_names() (yaocptool.modelling.dataset.DataSet method), 14

get\_entry\_size() (yaocptool.modelling.dataset.DataSet method), 14

get\_hudkay() (yaocptool.modelling.system\_model.SystemModel method), 21

get\_is\_factor() (in module yaocptool.stochastic.pce), 29

get\_measurement() (yaocptool.mpc.mpc.MPC method), 24

get\_measurement() (yaocptool.mpc.plant.Plant method), 25

get\_measurement() (yaocptool.mpc.plant.PlantSimulation method), 25

get\_new\_control() (yaocptool.mpc.mpc.MPC method), 24

get\_p\_opt\_indices() (yaocptool.modelling.ocp.OptimalControlProblem method), 16

get_p_unc_indices() tool.modelling.stochastic_occup.StochasticOCP method), 18	(yaocp-	include_equality() tool.optimization.abstract_optimization_problem.AbstractOptimizationProblem method), 26	(yaocp-
get_p_without_p_unc() tool.modelling.stochastic_occup.StochasticOCP method), 19	(yaocp-	include_final_time_equality() tool.modelling.occup.OptimalControlProblem method), 16	(yaocp-
get_problem_dict() tool.optimization.abstract_optimization_problem.AbstractOptimizationProblem method), 26	(yaocp-	include_inequality() tool.optimization.abstract_optimization_problem.AbstractOptimizationProblem method), 27	(yaocp-
get_solver() (yaocptool.optimization.abstract_optimization_problem.AbstractOptimizationProblem method), 26	(yaocp-	include_optimization_parameter() tool.modelling.occup.OptimalControlProblem method), 16	(yaocp-
get_states() (yaocptool.mpc.mpc.MPC method), 24			
get_system_at_given_times() tool.methods.base.discretizationschemebase.DiscretizationSchemeBase method), 4	(yaocp-	include_models() tool.modelling.system_model.SystemModel method), 21	(yaocp-
get_system_at_given_times() tool.methods.classic.collocationscheme.CollocationScheme method), 7	(yaocp-	include_optimization_parameter() tool.modelling.occup.OptimalControlProblem method), 17	(yaocp-
get_system_at_given_times() tool.methods.classic.multipleshooting.MultipleShootingScheme method), 10	(yaocp-	include_optimization_theta() tool.modelling.occup.OptimalControlProblem method), 17	(yaocp-
get_theta_opt_indices() tool.modelling.occup.OptimalControlProblem method), 16	(yaocp-	include_parameter() tool.modelling.system_model.SystemModel method), 21	(yaocp-
get_uncertain_initial_cond_indices() tool.modelling.stochastic_occup.StochasticOCP method), 19	(yaocp-	include_parameter() tool.optimization.abstract_optimization_problem.AbstractOptimizationProblem method), 27	(yaocp-
<b>H</b>		include_state() (yaocptool.modelling.occup.OptimalControlProblem method), 17	
has_delta_u (yaocptool.modelling.occup.OptimalControlProblem attribute), 16		include_state() (yaocptool.modelling.system_model.SystemModel method), 21	
has_parameters tool.modelling.dae_system.DAESystem attribute), 13	(yaocp-	include_system_equations() tool.modelling.occup.OptimalControlProblem method), 17	(yaocp-
<b>I</b>		include_system_equations() tool.modelling.system_model.SystemModel method), 21	
include_adjoint_states() tool.methods.classic.indirectmethod.IndirectMethod method), 9	(yaocp-	include_theta() tool.modelling.system_model.SystemModel method), 21	(yaocp-
include_algebraic() tool.modelling.occup.OptimalControlProblem method), 16	(yaocp-	include_time_chance_inequality() tool.modelling.stochastic_occup.StochasticOCP method), 19	(yaocp-
include_algebraic() tool.modelling.system_model.SystemModel method), 21	(yaocp-	include_time_equality() tool.modelling.occup.OptimalControlProblem method), 17	(yaocp-
include_control() tool.modelling.occup.OptimalControlProblem method), 16	(yaocp-	include_time_inequality() tool.modelling.occup.OptimalControlProblem method), 17	(yaocp-
include_control() tool.modelling.system_model.SystemModel method), 21	(yaocp-	include_uncertain_parameter() tool.modelling.stochastic_occup.StochasticOCP method), 19	(yaocp-
include_equality() tool.modelling.occup.OptimalControlProblem method), 16	(yaocp-	include_variable() tool.optimization.abstract_optimization_problem.AbstractOptimizationProblem method), 27	(yaocp-

method), 27

IndirectMethod (class in yaocptool.methods.classic.indirectmethod), 9

initialize() (yaocptool.methods.distributed\_solution.DistributedSolution method), 12

insert\_data() (yaocptool.modelling.dataset.DataSet method), 14

is\_collocation (yaocptool.methods.base.optimizationresult.OptimizationResult attribute), 5

is\_dae (yaocptool.modelling.dae\_system.DAESystem attribute), 13

is\_ode (yaocptool.modelling.dae\_system.DAESystem attribute), 13

is\_valid (yaocptool.methods.base.optimizationresult.OptimizationResult attribute), 5

## J

join() (yaocptool.modelling.dae\_system.DAESystem method), 13

join\_nu\_to\_theta() (yaocptool.methods.augmented\_lagrangian.AugmentedLagrangian static method), 12

join\_thetas() (in module yaocptool.util.util), 30

## K

kill() (yaocptool.methods.distributed\_solution.DistributedSolution method), 12

## L

lamb\_sym (yaocptool.modelling.system\_model.SystemModel attribute), 21

linearize() (yaocptool.modelling.system\_model.SystemModel method), 21

## M

merge() (yaocptool.modelling.ocp.OptimalControlProblem method), 17

merge() (yaocptool.modelling.system\_model.SystemModel method), 22

model (yaocptool.methods.augmented\_lagrangian.AugmentedLagrangian attribute), 12

model (yaocptool.methods.base.discretizationschemebase.DiscretizationSchemeBase attribute), 4

model (yaocptool.methods.base.solutionmethodbase.SolutionMethodBase attribute), 6

models (yaocptool.modelling.network.Network attribute), 15

MPC (class in yaocptool.mpc.mpc), 23

MultipleShootingScheme (class in yaocptool.methods.classic.multipleshooting), 9

## N

n\_eta (yaocptool.modelling.ocp.OptimalControlProblem attribute), 17

n\_g\_eq (yaocptool.modelling.ocp.OptimalControlProblem attribute), 17

n\_g\_ineq (yaocptool.modelling.ocp.OptimalControlProblem attribute), 17

n\_g\_stochastic (yaocptool.modelling.stochastic\_ocp.StochasticOCP attribute), 19

n\_h\_final (yaocptool.modelling.ocp.OptimalControlProblem attribute), 17

n\_h\_initial (yaocptool.modelling.ocp.OptimalControlProblem attribute), 17

n\_meas (yaocptool.estimation.extended\_kalman\_filter.ExtendedKalmanFilter attribute), 2

n\_meas (yaocptool.estimation.unscented\_kalman\_filter.UnscentedKalmanFilter attribute), 3

n\_p (yaocptool.modelling.system\_model.SystemModel attribute), 22

n\_p\_opt (yaocptool.modelling.ocp.OptimalControlProblem attribute), 17

n\_p\_unc (yaocptool.modelling.stochastic\_ocp.StochasticOCP attribute), 19

n\_pol\_parameters (yaocptool.estimation.pce\_kalman\_filter.PCEKalmanFilter attribute), 2

n\_pol\_parameters (yaocptool.stochastic.pce.PCEConverter attribute), 29

n\_subsystems (yaocptool.methods.distributed\_solution.DistributedSolution attribute), 12

n\_theta (yaocptool.modelling.system\_model.SystemModel attribute), 22

n\_theta\_opt (yaocptool.modelling.ocp.OptimalControlProblem attribute), 17

n\_u (yaocptool.modelling.simualtion\_result.SimulationResult attribute), 18

n\_u (yaocptool.modelling.system\_model.SystemModel attribute), 22

n\_uncertain (yaocptool.stochastic.pce.PCEConverter attribute), 29

n\_uncertain\_initial\_condition (yaocptool.modelling.stochastic\_ocp.StochasticOCP attribute), 19

n\_x (yaocptool.modelling.simualtion\_result.SimulationResult attribute), 18

n\_x (yaocptool.modelling.system\_model.SystemModel attribute), 22

n\_x (yaocptool.mpc.plant.Plant attribute), 25

n\_x (yaocptool.mpc.plant.PlantSimulation attribute), 26

n\_y (yaocptool.modelling.simualtion\_result.SimulationResult attribute), 18

n\_y (yaocptool.modelling.system\_model.SystemModel attribute), 22

Network (class in yaocptool.modelling.network), 15

Node (class in yaocptool.modelling.node), 15

NonlinearOptimizationProblem (class in yaocptool.methods.classic.nonlinearoptimization), 10

tool.optimization.nonlinear\_problem), 28

## O

OptimalControlProblem (class in yaocp-tool.modelling.ocp), 15

OptimizationResult (class in yaocp-tool.methods.base.optimizationresult), 5

## P

p (yaocptool.modelling.system\_model.SystemModel attribute), 22

p\_names (yaocptool.modelling.system\_model.SystemModel attribute), 22

PCEConverter (class in yaocptool.stochastic.pce), 29

PCEKalmanFilter (class in yaocp-tool.estimation.pce\_kalman\_filter), 2

Plant (class in yaocptool.mpc.plant), 25

PlantSimulation (class in yaocptool.mpc.plant), 25

plot() (yaocptool.methods.base.optimizationresult.OptimizationResult method), 5

plot() (yaocptool.modelling.dataset.DataSet method), 14

pre\_solve\_check() (yaocp-tool.modelling.ocp.OptimalControlProblem method), 17

prepare() (yaocptool.methods.base.solutionmethodsbases.SolutionMethodsBase method), 6

prepare() (yaocptool.methods.classic.directmethod.DirectMethod method), 9

prepare() (yaocptool.methods.classic.indirectmethod.IndirectMethod method), 9

print\_variables() (yaocp-tool.modelling.system\_model.SystemModel method), 22

problem (yaocptool.methods.base.discretizationschemebase.DiscretizationSchemeBase attribute), 4

problems (yaocptool.modelling.network.Network attribute), 15

put\_values\_in\_all\_sym\_format() (yaocp-tool.modelling.system\_model.SystemModel static method), 22

## Q

QuadraticOptimizationProblem (class in yaocp-tool.optimization.quadratic\_problem), 28

## R

relaxed\_alg (yaocptool.methods.augmented\_lagrangian.AugmentedLagrangian attribute), 12

remove\_algebraic() (yaocp-tool.modelling.ocp.OptimalControlProblem method), 17

remove\_algebraic() (yaocp-tool.modelling.system\_model.SystemModel method), 22

remove\_control() (yaocp-tool.modelling.ocp.OptimalControlProblem method), 17

remove\_control() (yaocp-tool.modelling.system\_model.SystemModel method), 22

remove\_parameter() (yaocp-tool.modelling.system\_model.SystemModel method), 22

remove\_theta() (yaocp-tool.modelling.system\_model.SystemModel method), 22

remove\_variables\_from\_vector() (in module yaocp-tool.util.util), 30

remove\_variables\_from\_vector\_by\_indices() (in module yaocptool.util.util), 30

replace\_variable() (yaocp-tool.modelling.ocp.OptimalControlProblem method), 17

replace\_variable() (yaocp-tool.modelling.system\_model.SystemModel method), 22

replace\_with\_optimal\_control() (yaocp-tool.methods.classic.indirectmethod.IndirectMethod method), 9

reset\_working\_model() (yaocp-tool.modelling.ocp.OptimalControlProblem method), 17

run() (yaocptool.mpc.mpc.MPC method), 24

run() (yaocptool.parallel.worker.Worker method), 28

run\_fixed\_control() (yaocptool.mpc.mpc.MPC method), 24

run\_fixed\_control\_with\_estimator() (yaocp-tool.mpc.mpc.MPC method), 24

## S

sample\_parameter\_log\_normal\_distribution\_with\_sobol() (in module yaocptool.stochastic.util), 29

sample\_parameter\_normal\_distribution\_with\_sobol() (in module yaocptool.stochastic.util), 29

send\_control() (yaocptool.mpc.mpc.MPC method), 25

set\_control() (yaocptool.mpc.plant.Plant method), 25

set\_control() (yaocptool.mpc.plant.PlantSimulation method), 26

set\_data\_to\_optimization\_result\_from\_raw\_data() (yaocptool.methods.base.discretizationschemebase.DiscretizationSchemeBase method), 4

set\_data\_to\_optimization\_result\_from\_raw\_data() (yaocptool.methods.classic.collocationscheme.CollocationScheme method), 8

set\_data\_to\_optimization\_result\_from\_raw\_data() (yaocptool.methods.classic.multipleshooting.MultipleShootingScheme method), 11

set\_initial\_condition\_as\_uncertain() (yaocp-tool.modelling.stochastic\_occup.StochasticOCP method), 19  
 set\_objective() (yaocptool.optimization.abstract\_optimization\_problem.AbstractOptimizationProblem method), 27  
 set\_parameter\_as\_optimization\_parameter() (yaocp-tool.modelling.occup.OptimalControlProblem method), 17  
 set\_parameter\_as\_uncertain\_parameter() (yaocp-tool.modelling.stochastic\_occup.StochasticOCP method), 19  
 set\_theta\_as\_optimization\_theta() (yaocp-tool.modelling.occup.OptimalControlProblem method), 18  
 simulate() (yaocptool.modelling.dae\_system.DAESystem method), 13  
 simulate() (yaocptool.modelling.system\_model.SystemModel method), 22  
 SimulationResult (class in yaocp-tool.modelling.simualtion\_result), 18  
 SolutionMethodsBase (class in yaocp-tool.methods.base.solutionmethodsbase), 5  
 solve() (yaocptool.methods.base.solutionmethodsbase.SolutionMethodsBase method), 6  
 solve() (yaocptool.methods.distributed\_solution.DistributedSolution method), 12  
 solve() (yaocptool.optimization.abstract\_optimization\_problem.AbstractOptimizationProblem method), 27  
 solve\_raw() (yaocptool.methods.augmented\_lagrangian.AugmentedLagrangian method), 12  
 sort() (yaocptool.modelling.dataset.DataSet method), 14  
 split\_x\_and\_u (yaocptool.methods.augmented\_lagrangian.AugmentedLagrangian attribute), 12  
 split\_x\_and\_u (yaocptool.methods.base.solutionmethodsbase.SolutionMethodsBase attribute), 6  
 split\_x\_and\_u() (yaocp-tool.methods.base.discretizationschemebase.DiscretizationSchemeBase method), 4  
 split\_x\_y\_and\_u (yaocp-tool.methods.augmented\_lagrangian.AugmentedLagrangian attribute), 12  
 split\_x\_y\_and\_u (yaocp-tool.methods.base.solutionmethodsbase.SolutionMethodsBase attribute), 6  
 split\_x\_y\_and\_u() (yaocp-tool.methods.base.discretizationschemebase.DiscretizationSchemeBase method), 4  
 split\_x\_y\_and\_u() (yaocp-tool.methods.classic.collocationscheme.CollocationScheme method), 8  
 split\_x\_y\_and\_u() (yaocp-tool.methods.classic.multipleshooting.MultipleShootingScheme method), 11  
 start() (yaocptool.methods.distributed\_solution.DistributedSolution method), 12  
 StochasticOCP (class in yaocp-tool.modelling.stochastic\_occup), 18  
 substitute\_variable() (yaocp-tool.modelling.dae\_system.DAESystem method), 13  
 SuperOCP (class in yaocptool.modelling.occup), 18  
 system\_type (yaocptool.modelling.system\_model.SystemModel attribute), 23  
 SystemModel (class in yaocp-tool.modelling.system\_model), 19

## T

t (yaocptool.modelling.simualtion\_result.SimulationResult attribute), 18  
 theta (yaocptool.modelling.system\_model.SystemModel attribute), 23  
 theta\_names (yaocptool.modelling.system\_model.SystemModel attribute), 23  
 time\_breakpoints (yaocp-tool.methods.base.discretizationschemebase.DiscretizationScheme attribute), 5  
 time\_breakpoints (yaocp-tool.methods.base.solutionmethodsbase.SolutionMethodsBase attribute), 6  
 time\_interpolation (yaocp-tool.methods.classic.collocationscheme.CollocationScheme attribute), 8  
 time\_interpolation\_algebraics (yaocp-tool.methods.classic.collocationscheme.CollocationScheme attribute), 8  
 time\_interpolation\_controls (yaocp-tool.methods.classic.collocationscheme.CollocationScheme attribute), 8  
 time\_interpolation\_nu (yaocp-tool.methods.augmented\_lagrangian.AugmentedLagrangian attribute), 12  
 time\_interpolation\_states (yaocp-tool.methods.classic.collocationscheme.CollocationScheme attribute), 8  
 to\_dataset() (yaocptool.methods.base.optimizationresult.OptimizationResult method), 5  
 type (yaocptool.modelling.dae\_system.DAESystem attribute), 13

## U

u (yaocptool.modelling.simualtion\_result.SimulationResult attribute), 18  
 u (yaocptool.modelling.system\_model.SystemModel attribute), 23  
 u\_names (yaocptool.modelling.simualtion\_result.SimulationResult attribute), 18



[u\\_names \(yaocptool.modelling.system\\_model.SystemModel attribute\), 23](#)  
[unpack\\_decision\\_variables\(\) \(yaocptool.methods.base.discretizationschemebase.DiscretizationSchemeBase method\), 5](#)  
[unpack\\_decision\\_variables\(\) \(yaocptool.methods.classic.collocationscheme.CollocationScheme method\), 8](#)  
[unpack\\_decision\\_variables\(\) \(yaocptool.methods.classic.multipleshooting.MultipleShootingScheme method\), 11](#)  
[UnscentedKalmanFilter \(class in yaocptool.estimation.unscented\\_kalman\\_filter\), 2](#)  
[unvec\(\) \(yaocptool.methods.base.solutionmethodsbasesolutionmethodsbasesolving method\), 6](#)  
**V**  
[vectorize\(\) \(yaocptool.methods.base.discretizationschemebase.DiscretizationSchemeBase method\), 5](#)  
**W**  
[Worker \(class in yaocptool.parallel.worker\), 28](#)  
**X**  
[x \(yaocptool.modelling.simualtion\\_result.SimulationResult attribute\), 18](#)  
[x \(yaocptool.modelling.system\\_model.SystemModel attribute\), 23](#)  
[x\\_names \(yaocptool.modelling.simualtion\\_result.SimulationResult attribute\), 18](#)  
[x\\_names \(yaocptool.modelling.system\\_model.SystemModel attribute\), 23](#)  
[x\\_sys\\_sym \(yaocptool.modelling.system\\_model.SystemModel attribute\), 23](#)  
**Y**  
[y \(yaocptool.modelling.simualtion\\_result.SimulationResult attribute\), 18](#)  
[y \(yaocptool.modelling.system\\_model.SystemModel attribute\), 23](#)  
[y\\_names \(yaocptool.modelling.simualtion\\_result.SimulationResult attribute\), 18](#)  
[y\\_names \(yaocptool.modelling.system\\_model.SystemModel attribute\), 23](#)  
[yaocptool \(module\), 31](#)  
[yaocptool.config \(module\), 31](#)  
[yaocptool.estimation \(module\), 3](#)  
[yaocptool.estimation.estimator\\_abstract \(module\), 1](#)  
[yaocptool.estimation.extended\\_kalman\\_filter \(module\), 1](#)  
[yaocptool.estimation.pce\\_kalman\\_filter \(module\), 2](#)  
[yaocptool.estimation.unscented\\_kalman\\_filter \(module\), 2](#)  
[yaocptool.methods \(module\), 12](#)