********

ODESCA Documentation



**Contents**

[1 Introduction 3](#_Toc478544482)

[2 Requirements 3](#_Toc478544483)

[3 Basic Tool Structure 4](#_Toc478544484)

[3.1 Class structure 4](#_Toc478544485)

[3.2 Utilities 5](#_Toc478544486)

[4 Detailed Class Documentation 6](#_Toc478544487)

[4.1 BaseClass 6](#_Toc478544488)

[4.1.1 Properties 6](#_Toc478544489)

[4.2 Object 6](#_Toc478544490)

[4.2.1 Properties 7](#_Toc478544491)

[4.2.2 Methods 8](#_Toc478544492)

[4.3 Component 8](#_Toc478544493)

[4.3.1 Properties 9](#_Toc478544494)

[4.3.2 Methods 9](#_Toc478544495)

[4.4 System 10](#_Toc478544496)

[4.4.1 Properties 10](#_Toc478544497)

[4.4.2 Methods 11](#_Toc478544498)

[4.5 Steady State 11](#_Toc478544499)

[4.5.1 Properties 12](#_Toc478544500)

[4.5.2 Methods 12](#_Toc478544501)

[4.6 Approximation 13](#_Toc478544502)

[4.6.1 Properties 13](#_Toc478544503)

[4.6.2 Methods 13](#_Toc478544504)

[4.7 Linear 13](#_Toc478544505)

[4.7.1 Properties 14](#_Toc478544506)

[4.7.2 Methods 14](#_Toc478544507)

[4.8 Bilinear 14](#_Toc478544508)

# Introduction

For Model-based control algorithm development and algorithm parameterization, algebraic models of the plants have a lot of advantages over simulation models. With algebraic models, a multitude of mathematical approaches can be used for optimization and analytical purposes.

To ensure uniform modeling and a simple way of working with differential equations, the tool ODESCA was created to fill the gap of modeling and analysis of ordinary differential equation systems in MATLAB.

With ODESCA, dynamical systems of the form

xdot = f(x,u)

y = g(x,u)

can be set up by using and configuring custom components. The differential equations of these components can be connected to each other in a system which provides methods for the analysis of the dynamical behavior. This can be done by linearization at steady states and the implemented MATLAB features for linear systems. Moreover, some features for nonlinear analysis and synthesis approaches are implemented for systems.

The name ODESCA is an acronym for “**O**rdinary **D**ifferential **E**quation **S**ystems: **C**reation and **A**nalysis”.

# Requirements

The tool ODESCA was developed under MATLAB version R2016a. Therefore the correct functionality of the tool is not guaranteed for older MATLAB releases.

The differential equations are described with symbolic variables which are part of the Symbolic Math Toolbox which is required for the tool.

For the linear analysis of nonlinear systems, ODESCA uses the control system toolbox.

The base tool will work with these three licenses. However there is a collection of utility functions which provide interfaces and additional functionalities for the tool. These functions may require other licenses (E.g.: Simulink for the function which creates a nonlinear Simulink model).

# License

ODESCA is free software: you can redistribute it and/or modify

it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or

(at your option) any later version.

ODESCA is distributed in the hope that it will be useful,

but WITHOUT ANY WARRANTY; without even the implied warranty of

MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the

GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License

along with ODESCA. If not, see <http://www.gnu.org/licenses/>.

# Basic Tool Structure

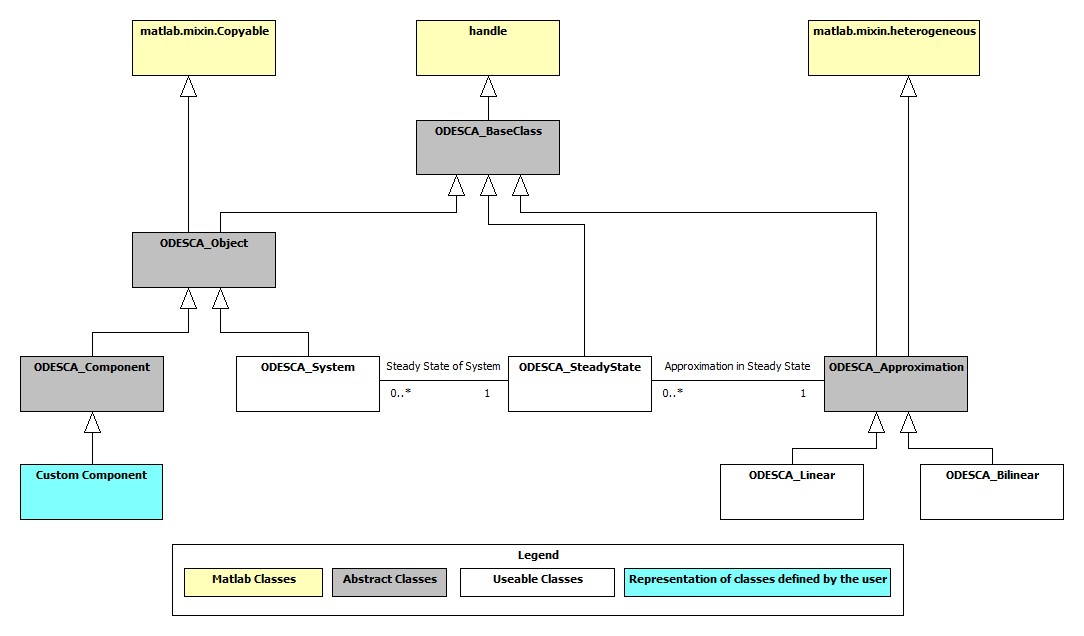
## Class structure

The tool is based on object-oriented classes. The class diagram is shown in figure 1.

Meaning of colors:

* **Yellow:**  Classes provided by MATLAB
* **Gray:** Abstract Classes, provide functionalities but cannot be instanced
* **White:** Classes which can be instanced and be used by the User
* **Blue:** Representation of classes created by the User

Figure 1: Class Diagram



* **BaseClass:**

This class is used to keep track of the ODESCA version an instance of a class was created in if it was saved and loaded again.

* **Object:**  
  The class *Object* owns the properties and methods to store differential equation systems of the form

xdot = f(x,u)

y = g(x,u)

in symbolic variables. Furthermore this class provides a system to handle the parameters in the equations and methods to organize the equation system. The classes Component and System are subclasses of Object. Therefore access to all public methods of the object class is granted.

* **Component:**   
  The class provides functionality for the creation of differential equations and their parameters.

It is used as a super class for custom components which can be added to a System.

* **CustomComponents:**

The custom components are created by the user. They are representations of the different parts of system and are used to define the differential equations which describe the dynamic behavior. After a custom component class was created it can be instanced, parameterized and added to a *System* instance. The class file for a new custom component can be create with the method “createNewComponentFile()” of the class ODESCA\_Util. For more information on the utilities, see section **3.2**.

* **System:**

The class *System*is used to combine components to a system and analyze it with mathematical approaches. Furthermore a nonlinear Simulink model can be created of the system. It is the only class in ODESCA which can be created directly by the user by calling the class constructor.

* **SteadyState:**

For the handling and analysis of a systems steady states (f(x,u) = 0!) the class called *SteadyState* is used. It is the representation of a system in a chosen steady state and provides the functionality to approximate the system around the steady state. This is useful to analyze the behavior of the *System*the steady state belongs to. A *SteadyState*cannot exist without its *System*, so if the system is deleted, the SteadyState is deleted too.

* **Approximation:**

All approximations which are attached to a SteadyState has to be subclasses of the class *Approximation*which mainly serves as an interface to ensure that all approximations provide certain methods and properties. An *Approximations* cannot exist without a *SteadyState*, so if the steadystate is deleted, the approximations is deleted too.

* **Linear:**This class is a subclass of *Approximation.* It represents the linear behavior of a system at the steady state the instance of this class belongs to. This makes use of the state space object (ss) of the control system toolbox and adds and improves functionalities. The main use of this class is to perform a linear analysis of a system in a steady state.

Note that all class names of the tool ODESCA start with the name “ODESCA\_”. E.g.: the system class is called “ODESCA\_System”

## Utilities

In addition to the classes mentioned in the section above, the tool provides a number of utility functions which provide interfaces to other programs and some additional features. These features may have additional toolboxes or programs as requirement to be used. They are grouped in the class **ODESCA\_Util** where the utility functions are static methods which can be called without creating an instance of the class.

For detailed help on all utility functions use the MATLAB help command (E.g.: for the function createNonlinearSimulinkModel type the command “help ODESAC\_Util.createNonlinearSimulinkMode()” )

List of all utility functions:

|  |  |  |
| --- | --- | --- |
| Name | Description | Requirement |
| createNonlinearSimulinkModel | Creates a Simulink model of the ODESCA\_Systen instance | MATLAB Simulink |
| createNewComponentFile | Starts a dialog to create a new custom component file from a template | Keine |
| toPDF | Creates a .pdf which documents a ODESCA\_Object class in latex style | MiKTeX (v2.9) |

# 

# Detailed Class Documentation

In this section the classes are documented with their methods and properties. Only the methods which have public access are listed because all protected and private methods can and must not be used by the user. All properties are READ ONLY and can only be modified within the methods of the classes.

## BaseClass

The BaseClass, who could guess, is the base for all classes in ODESCA. It is abstract so it cannot be instantiated. Every other class in ODESCA is derived from this class.

The main reason for the base class to exist is to keep track of the version number of ODESCA an instance of a class was create under. For this reason the class provides two hidden properties.

### Properties

|  |  |
| --- | --- |
| Name | Description |
| version | Version of ODESCA the instance of the class was first created in |
| classDefinitionVersion | Current version of ODESCA (version of the class definition) |

The two properties are hidden which means they do NOT appear in the list of properties, but trust me, they are there! The properties have a public get access.

The difference between version and classDefinitionVersion shows up after an instance which was saved to a .mat file is loaded. The property version is set at the first initialization of a class and never changes afterwards. The property classDefinitionVersion always matches the current ODESCA version. So if an instance of a class is created, the two properties have the same value. If an instance was saved and loaded in a later ODESCA version, the version is still the same as when it was saved but the classDefinitionVersion has changed. This properties can help with problems occurring after an instance is loaded from a .mat file.

## Object

The class ODESCA\_Object is the absolute base for all other classes in the tool. It provides the possibility to store everything needed to describe nonlinear differential equations, like the states, inputs, outputs, parameters and the equations themselves. It provides methods to modify the parameters and equations of the object (e.g. switch the order of inputs) and methods to get information about the object.

It is important to know that the class ODESCA\_Object is a subclass of the MATLAB class matlab.mixin.Copyable. This means that an instance of the class Object is passed by reference but a copy of the instance can be created.

This class is abstract so no instance can be created. It is meant to be the super class for the classes ODESCA\_Component and ODESCA\_System.

### Properties

|  |  |
| --- | --- |
| Name | Description |
| name | Stores the name of the instance of this class |
| param | Structure that stores the parameters used in the equations with their current value |
| p | Array that stores the symbolic counterparts for the parameters |
| paramUnits | Cell array that stores the units of the parameters |
| f | Array with the symbolic equations for the state changes |
| g | Array with the symbolic equations for the outputs |
| x | Array with the symbolic states of the system |
| u | Array with the symbolic inputs of the system |
| stateNames | Cell array that stores the names of the states |
| inputNames | Cell array that stores or the names of the inputs |
| outputNames | Cell array that stores the names of the outputs |
| stateUnits | Cell array that stores the units of the states |
| inputUnits | Cell array that stores the units of the inputs |
| outputUnits | Cell array that stores the units of the outputs |

To store the nonlinear differential equations in the form of

xdot = f(x,u)

y = g(x,u)

all components of the equations (states, inputs, parameters) have to be stored as symbolic variables.

The names of the states are stored in the array **stateNames** and the symbolic variables representing the states are stored in the array **x** (e.g. x = [x1, x2, x3 …]) at the same position as the corresponding name.

The inputs are stored in the same way: **inputNames** stores the names and **u** stores the corresponding symbolic variables (e.g. u = [u1, u2, u3 …]).

The names of the outputs are stored in the array **outputNames**. There are no symbolic variables for the outputs because they do not appear inside the equations. The equations of the outputs are stored in the array **g** at the same position as the names in the output name array.

The equations which describe the change of the states (xdot) are stored in the array **f** where the position corresponds to the state with the same position in the state name array.

The parameters are stored in the structure **param** where the name of each field is the name of the parameter and the given value is stored in the field. The symbolic representation of each parameter is stored in the array **p** where the position corresponds to the order of the fields (first symbolic parameter corresponds to the first field of the structure). The symbolic parameters have the same name as the parameters in the structure.

The physical units of the components are stored as strings in the arrays **stateUnits**, **inputUnits**, **outputUnits** and **paramUnits** where the position corresponds to the arrays with the names. The units are mandatory even though the correctness of the strings as physical units cannot be checked.

Note that ALL properties are READ ONLY. So to modify them the methods of the object have to be used.

For detailed information on each property use the MATLAB help function in the way “**help** **ODESCA\_Object.PROPERTYNAME**” (E.g.: use “help ODESCA\_Object.param” to find out more about how the parameters are stored.)

### Methods

|  |  |
| --- | --- |
| Name | Description |
| calculateNumericEquations | Calculates the numeric equations if all parameters are set |
| checkParam | Checks if all parameters are set to a value |
| getInfo | Creates structure with information about states, inputs and outputs |
| getParam | Returns the values and names of the parameters as cell arrays |
| getSymbolicStructure | Creates a structure with the symbolic variables of the system |
| isValidSymoblic | Checks if all symbolic variables are part of the object |
| show | Shows the equations, states, inputs, outputs and parameters of the object |
| setAllParamAsInput | Sets all parameters as inputs of the object |
| setName | Sets the name of the instance |
| setParam | Sets a parameter to a scalar numeric value or to empty |
| setParamAsInput | Sets the specified parameter as an input of the object |
| switchInputs | Switches two inputs |
| switchOutputs | Switches two outputs |
| switchStates | Switches two states |

For detailed information on each method use the MATLAB help function for the class or for the single method. E.g.: “**help ODESCA\_Object**” or “**help ODESCA\_Object.METHODNAME**”

## Component

The class ODESCA\_Component is a child of the class ODESCA\_Object. It is used to create the nonlinear differential equations, inputs, outputs, states and parameters. The creation of these parts may depend on so called construction parameters defined in the component.

Instances of subclasses of the ODESCA\_Component class can be added to systems (see ODESCA\_System.addComponent()).

Note that the class ODESCA\_Component itself is ABSTRACT so it cannot be initialized directly. Use the utility function ODESCA\_Util.createNewComponentFile to create new custom components.

For detailed information on the creation of custom components see chapter 5.1 “Creation of a custom component” in this documentation.

If an instance of a subclass of the ODESCA\_Component class (e.g.: the class “Pipe” which is a subclass of “ODESCA\_Component”) is created and has construction parameters, none of the fields except of the construction parameters are filled. This is because the creation of the content depends on the construction parameters (e.g.: the number of nodes used to simulate the behavior of a pipe). So to create the equations, states, inputs, outputs and parameters the construction parameters have to be set!

After all construction parameters are set, the equations can be calculated. To do so, use the method tryCalculateEquations(). If the equations have been created, this method returns the logical value true. If the construction parameters are not set, the method does not calculate anything and returns the logical value false. After the method is called successfully, the component is fully initialized.

A component without construction parameters will be fully initialized on the creation of the instance.

Note that it is not necessary to call this method before a component is added to a system but the process of adding the component will fail if there are unset construction parameters.

This class inherits from ODESCA\_Object. Therefore it is passed by reference and can be copied.

### Properties

|  |  |
| --- | --- |
| Name | Description |
| constructionParam | Structure of parameters needed for the construction of the equations |
| FLAG\_EquationsCalculated | Flag that determines if the equations have been calculated |

The structure **constructionParam** stores all parameters which are needed in the process of creating the equations, inputs, outputs and states. E.g.: A pipe always contains the same equations but the number of states depends on the number of nodes the model uses. Therefore, if a pipe is being modeled, ‘nodes’ would be set as a construction parameter. These construction parameters have to be set before the equations are created.

The property **FLAG\_EquationsCalculated** determines if the equations have been created already. It is set false before the calculation and changed to true afterwards.

Note that ALL properties are READ ONLY. So to modify them the methods of the object have to be used

For detailed information on each property use the MATLAB help function in the way “**help** **ODESCA\_Component.PROPERTYNAME**” (E.g.: use “help ODESCA\_Component.constructionParam” to find out more about how the construction parameters are stored.)

### Methods

|  |  |
| --- | --- |
| Name | Description |
| checkConstructionParam | Checks if all construction parameters are set |
| checkEquationsCorrect | Checks if all equations were set, true if they are set |
| setConstructionParam | Set a construction parameter to a numeric value |
| tryCalculateEquations | Calculates the Equations if all construction parameters are set |

For detailed information on each method use the MATLAB help function for the class or for the single method. E.g.: “**help ODESCA\_Component**” or “**help ODESCA\_Component.METHODNAME**”

## System

The class ODESCA\_System is the most important class for the user. It is used to combine components into a system, to connect the equations of the components and most importantly to analyze the system with mathematical methods.

The system class is used directly by creating an instance of it. After initializing a new instance, one or more components can be added and connected. Furthermore, it is possible to add other systems.

By adding a component or system, all the equations, states, inputs, outputs and parameters are added to the existing arrays. This means, if there has been a change inside an instance of a component after it was added to a system, the change will not affect the system.

Note that the name of a component is added to all states, inputs, outputs and parameters when the component is added to the system. This is necessary for the tool to work correctly and the prevention of name conflicts.

It is possible to rename a component after adding it to a system. This is useful when combining two systems owning some components with the same name. Otherwise these components will be renamed.

The utility functions provide a possibility to create a nonlinear Simulink model from the system. For more information on utility functions see chapter 3.3.

A subpart of the system is the steady state. In this state the system is in an equilibrium which means with constant inputs the states are also constant:

xdot = f(x0,u0) = 0 !

The steady states are used for linear analysis of the nonlinear system. Each steady state has a reference to the system it belongs to and a system has a list of all its steady states. The steady states are represented in the class ODESCA\_SteadyState (see chapter 4.4).

Use the methods createSteadyState() to create a new steady state for a system. They can be deleted with the method removeSteadyState() or by calling the delete() method on the instance of a steady state.

The class ODESCA\_System inherits from ODESCA\_Object. Therefore it is passed by reference and can be copied. Note that on creating a copy of the system, all steady states are copied and do not depend on the original instance of ODESCA\_System.

### Properties

|  |  |
| --- | --- |
| Name | Description |
| components | Names of components which have been added to the system |
| defaultSampleTime | Default size of a time step for discrete systems |
| steadyStates | List of steady states linked to the system |

The property **defaultSampleTime** is used as the sample time if a discrete system is created without providing a sample time. The array **components** stores the names of all components which have been added to the system. In the property **steadyStates** all steady states linked to the system are stored.

Note that ALL properties are READ ONLY. So to modify them the methods of the object have to be used.

For detailed information on each property use the MATLAB help function in the way “**help** **ODESCA\_System.PROPERTYNAME**” (E.g.: use “help ODESCA\_System.steadyStates” to find out more about how the steady states are stored.)

### Methods

|  |  |
| --- | --- |
| Name | Description |
| addComponent | Adds a component to the system |
| addSystem | Adds the given system to the existing |
| connectInput | Substitutes the chosen input with a symbolic expression |
| createMatlabFunction | Creates a Matlab functions out of the equations of the system |
| createSteadyState | Creates a new ODESCA\_SteadyState and links it to the system |
| removeOutput | Removes an output from the system |
| removeSteadyState | Removes a steady state and its link to the system |
| renameComponent | Renames a component within a system |
| setDefaultSampleTime | Sets the default sample time of the system |
| simulateStep | Simulates a step in the nonlinear system |
| symLinearize | Linearize the equations symbolically |

For detailed information on each method use the MATLAB help function for the class or for the single method. E.g.: “**help ODESCA\_System**” or “**help ODESCA\_System.METHODNAME**”

## Steady State

A steady state of a system is a state, where all the inputs and states (and therefore outputs) are constant over time:

xdot = f(x0,u0) = 0 !

In this state a nonlinear system can be analysed in many ways. With this class, approximation of a system in a steady state (like linearization, bilinearization, etc) can be calculated. These approximations can be used to perform analysis of a system in the steady state. A list of all approximations in the steady state is stored in the property approximations. The steady state and its approximations have references of each other.

To create a steady state, the method createSteadyState(x0, u0) of the class ODESCA\_System has to be called. It returns a new instance of the class steady state.

A steady state cannot exist without an instance of the class ODESCA\_System. The steady state is always attached to the system and the system is always attached to the steady state. If the system is deleted, the steady state will be deleted too.

Note: The steady state is completely numeric, which means all parameters of a system have to be set in order to create a steady state! To create a symbolic linearization of a system, see the method ODESCA\_System.symLinearize().

The steady state has a flag called isStructuralValid, which determines if the steady state matches the structure of the system. It is true if the system has not changed after the steady state was created, otherwise the flag is false. For an invalid steady state, no approximations can be made. Some of the methods of the approximations might not work with an invalid steady state.

The ODESCA\_SteadyState is a subclass of the MATLAB class handle so the instances is passed by reference and can be stored in variables. Note that if the instance of the class ODESCA\_System, the steady state is attached to, is deleted, the steady state is deleted as well since a steady state without a system does not make sence.

### Properties

|  |  |
| --- | --- |
| Name | Description |
| name | Name of the steady state |
| x0 | Values of the states in the steady state |
| u0 | Values of the inputs in the steady state |
| y0 | Values of the outputs in the steady state |
| approximations | Array of the system approximations at the steady state |
| system | System instance the steady state refers to |
| param | Parameter set of the system the steady state refers to |
| structuralValid | Flag to determine if the steady state is structural valid |
| numericValid | Flag to determine if the steady state is numerical valid |

The property **name** stores the name of the steady state. If it is not given while the creation, the steady state gets a default name. The properties **x0**, **u0** and **y0** store the values of the states, inputs and outputs for which the system is in a steady state.

The approximations of a system in a steady state are stored in the property **approximations**. The approximations can be created with the corresponding methods. If the steady state is deleted, all approximations are deleted as well and therefore aren’t useable anymore.

The property **system** is a reference to the ODESCA\_System instance the steady state belongs to. The property **param** holds the parameter structure of the system the steady state refers to at the point the steady state was created.

The property **structuralValid** determines if the system the steady belongs to has changed structural (e.g.: remove of an output, switch of equations). In this case some functions like the approximation are not possible anymore. The property **numericalValid** determines if the given values for x0, u0 and y0 lead to a steady state of the system. If the values are not correct in a certain rang of precision, numerical problems can occur on the analysis of the steady state. A warning is thrown in this case.

Note that ALL properties are READ ONLY. So to modify them the methods of the object have to be used.

For detailed information on each property use the MATLAB help function in the way “**help** **ODESCA\_SteadyState.PROPERTYNAME**” (E.g.: use “help ODESCA\_SteadyState.H” to find out more about how the transfer functions are stored.)

### Methods

|  |  |
| --- | --- |
| Name | Description |
| isNumericValid | Checks if the steady state is numerically valid for the system |
| setName | Sets the name of the steady state |
| delete | Custom delete() method which overwrites the default delete() |

The methods for the approximations are listed below:

|  |  |
| --- | --- |
| Name | Description |
| linearize | Calculate the linear approximation of the system in the steady state |
| linear | Returns the instances of the ODESCA\_Linear class |

For detailed information on each method use the MATLAB help function for the class or for the single method. E.g.: “**help ODESCA\_SteadyState**” or “**help ODESCA\_SteadyState.METHODNAME**”

## Approximation

The approximation class serves as an interface for all approximations which can be created of a system in a steady state. It specifies the behavior for the deletion and the reference to the steady state it belongs to.

An approximation cannot exist without a steady state.

The class is abstract and cannot be instanziated. It is a subclass of matlab.mixin.heterogeneous which means all subclasses which derive from approximation can be stored in the same array even so they may have different properties and methods.

### Properties

|  |  |
| --- | --- |
| Name | Description |
| steadyState | Steady State the approximation belongs to |

The property **steadyState** is a reference to the steady state the approximation belongs to. Since an approximation cannot exist without a steady state, this property is always filled. If the steady state it refers to is deleted, this approximation is deleted too.

### Methods

|  |  |
| --- | --- |
| Name | Description |
| delete | Custom delete() method which overwrites the default delete() |

The class approximation overwrites the default delete() behavior to make sure the delete method is called correctly. If the method is called at an approximation, the reference to the approximation is automatically removed from the steady state it belongs to.

## Linear

The class Linear represents the linear approximation of a system in a steady state. It has the form

where the Matrices A, B, C and D are numeric.

The class provides a number of method for the linear analysis of the system like the plotting of bode- and nyquist plots or the analysis of stability, controllability and observability.

The linear class contains a state space object (ss) and a transfer function object (tf) of the control system toolbox which represent the same linearization. The methods of the class linear make use of the functionalities of the control system toolbox and extend them.

The methods of this class are designed to work with arrays of the class to make analysis and comparison as easy as possible.

The instances of the class have to belong to a steady state. If the steady state the instance belongs to is deleted, the instance is deleted too.

### Properties

|  |  |
| --- | --- |
| Name | Description |
| A | Time continuous system matrix |
| B | Time continuous input matrix |
| C | Output matrix |
| D | Feedthrough matrix |
| Ad | Time discrete system matrix |
| Bd | Time discrete input matrix |
| discreteSampleTime | Default sample time of the system the steady state refers to |
| ss | State space model of the linearization |
| tf | transfer functions of the linearization |

The properties **A**, **B**, **C** and **D** store the system matrices of a LTI system. **Ad** and **Bd** store the time discrete counterparts of the A and B. The property **discreteSampleTime** stores the sample time with which the discrete matrices Ad and Bd where calculated.

The property **ss** stores a state space object (ss) of the control system toolbox. It contains all information available in the system, the linearization’s steady state belongs to and is set up for the time continuous linear system. The property **tf** stores a transfer function object (tf) of the control system toolbox.

### Methods

|  |  |
| --- | --- |
| Name | Description |
| bodeplot | Plots the bode diagram for the linearization |
| nyquistplot | Plots the nyquist diagram for the linearization |
| stepplot | Plots the step response for the linearization |
| isAsymptoticStable | Checks if the linearizations are asymptotically stable |
| isObservable | Checks if the linearizations are controllable |
| isControllable | Checks if the linearizations are observable |
| discretize | Calculates the discrete linear matrices |

NOTE: The methods of the class steady state are designed to work for arrays of the class! This is useful to compare linear approximations with little effort.

For detailed information on each method use the MATLAB help function for the class or for the single method. E.g.: “**help ODESCA\_Linear**” or “**help ODESCA\_Linear.METHODNAME**”

## Bilinear

! This class is not yet implemented but planned in future releases. !