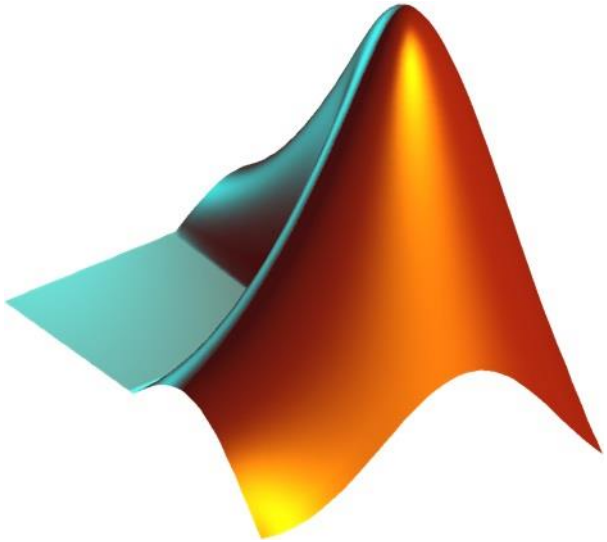
# MATLAB / Simulink Lab Course

# Simulink Fundamentals



UPDATED TO
R2019A

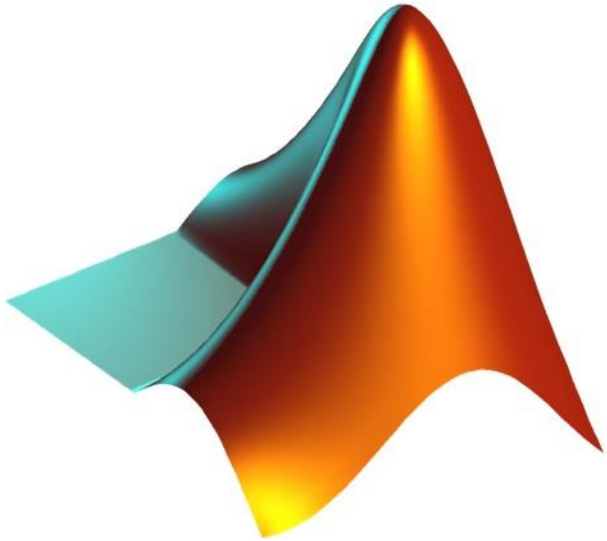
# Objectives & Preparation “Simulink Fundamentals"

- Which MathWorks products are covered?
  - ⇨ Simulink
- What skills are learnt?
  - ⇨ Basic Introduction to Simulink
  - ⇨ Modelling Simple Systems with Simulink
    (i.e. translate a set of (ordinary differential) equations into a simulation model)
  - ⇨ Derive the Governing Equations from a Given Model
  - ⇨ Basic Visualization of Simulation Results
- How to prepare for the session?

| Videos | Tutorials |
|---|---|
| **Getting Started With Simulink** https://de.mathworks.com/videos/getting-started-with-simulink-69027.html | **Create a simple model** https://de.mathworks.com/help/simulink/gs/create-a-simple-model.html |
| **Visualizing Simulation Results** https://de.mathworks.com/videos/visualizing-simulation-results-102160.html | **Model a Dynamic System** https://de.mathworks.com/help/simulink/gs/define-system.html |

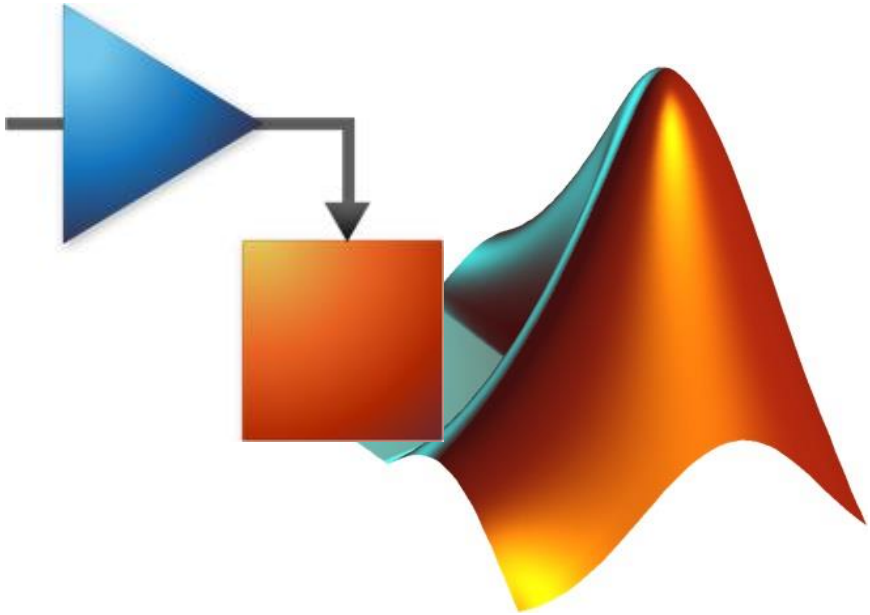
# Outline





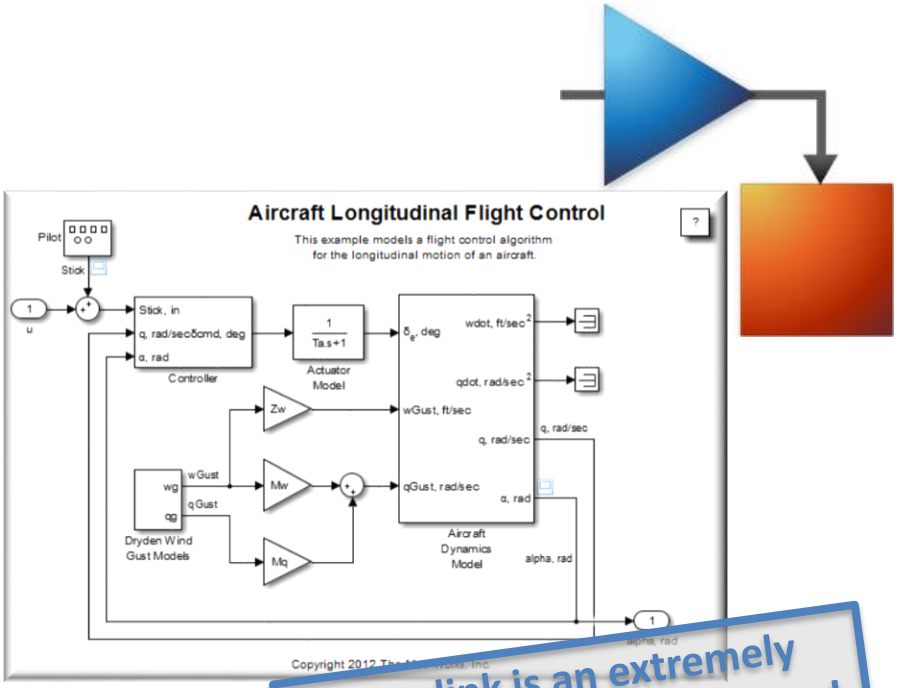
UPDATED TO R2019A

# 1. Introduction



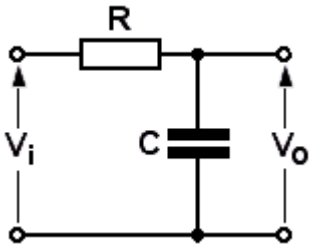
# Introduction

- Simulink 1 available in 1990
- Shipped with MATLAB starting from 1992
- Multi-domain **dynamic systems** can be
  - modelled
  - simulated
  - analyzed
- Interface: graphical **block diagramming**
- Uses compiled code + *.mex (Matlab DLL)
- Tightly integrated with MATLAB
- Add-ons expand capabilities
  - Stateflow
  - Embedded Coder
  - Simulink Real-Time
  - Simulink Verification and Validation
  - …
- **Automatic C-Code Generation**
- Most widely used in automatic control / digital signal processing




Simulink is an extremely powerful and large program! Here only the necessary things for basic modelling tasks are introduced!

# 2. A Simple Example System

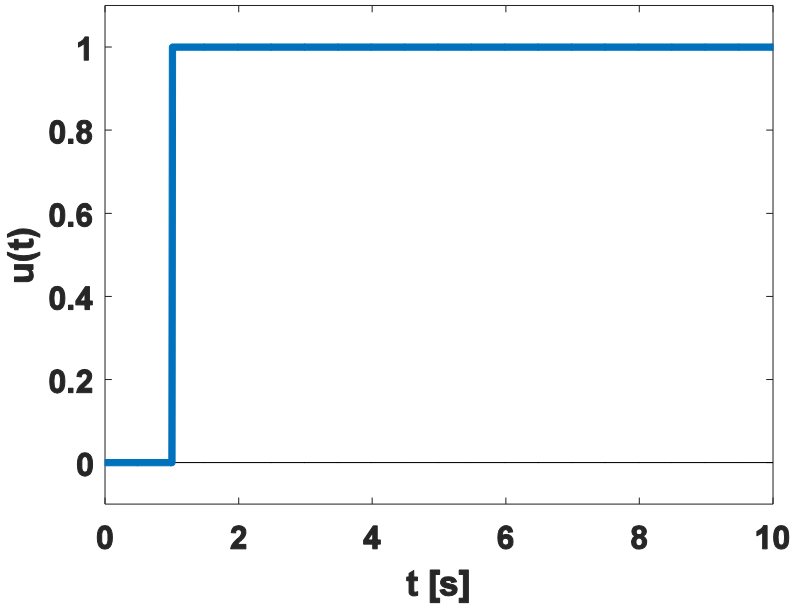


# A Simple Example System

Consider the reaction of the following first order dynamic system to a unit step input

$$\dot{x}(t) = -\frac{1}{T}x(t) + \frac{K}{T}u(t)$$
$$y(t) = Cx(t)$$

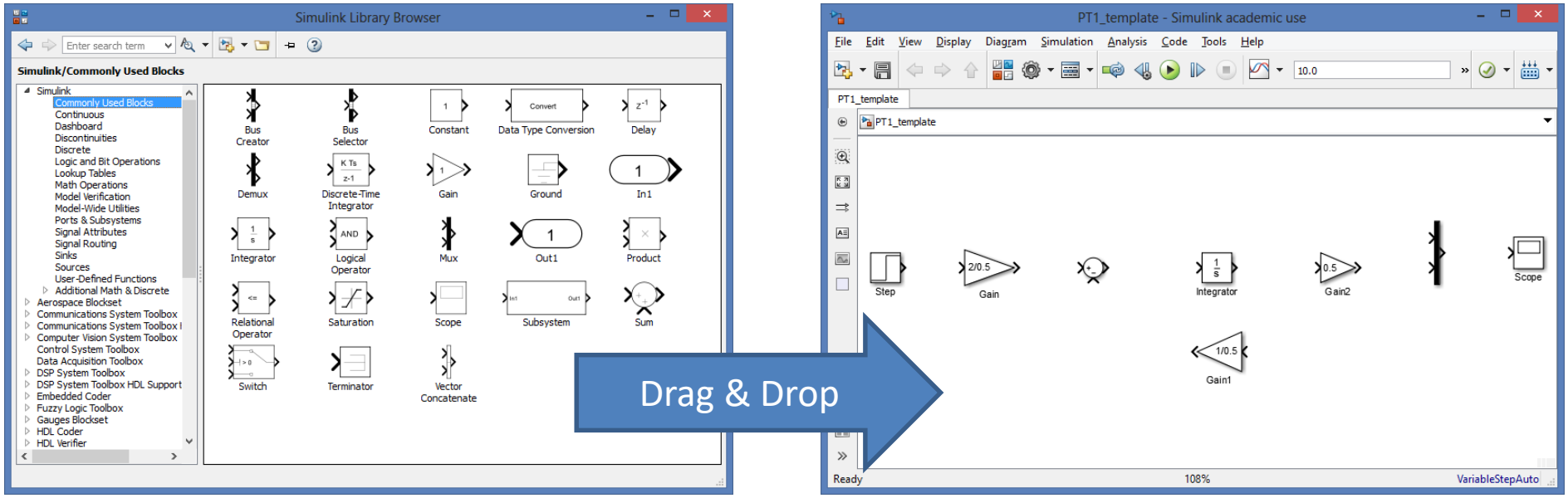
with $T = 0.5$, $K = 2$, $C = 0.5$




# A Simple Example System

## Simulink Solution

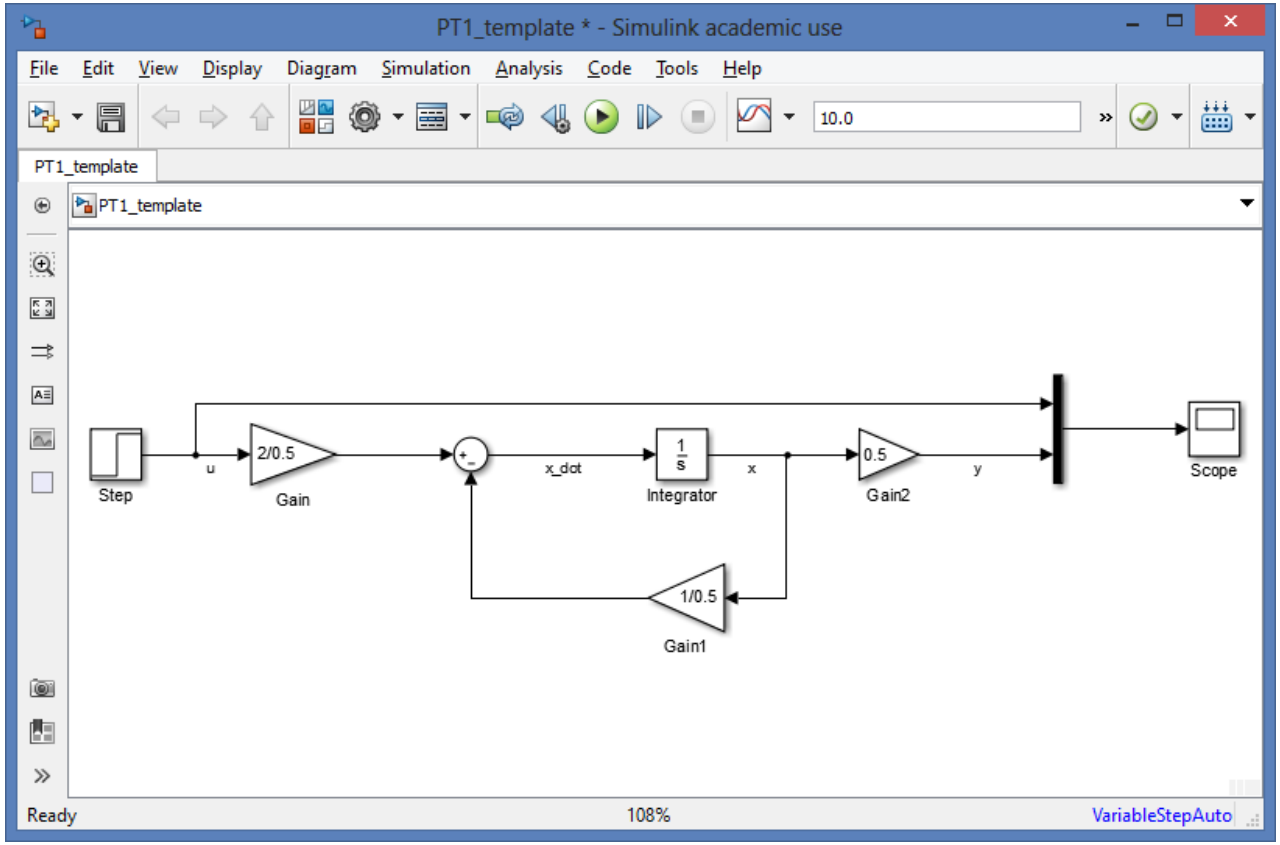
1. Assemble blocks
2. Adjust Parameters



# A Simple Example System

## Simulink Solution

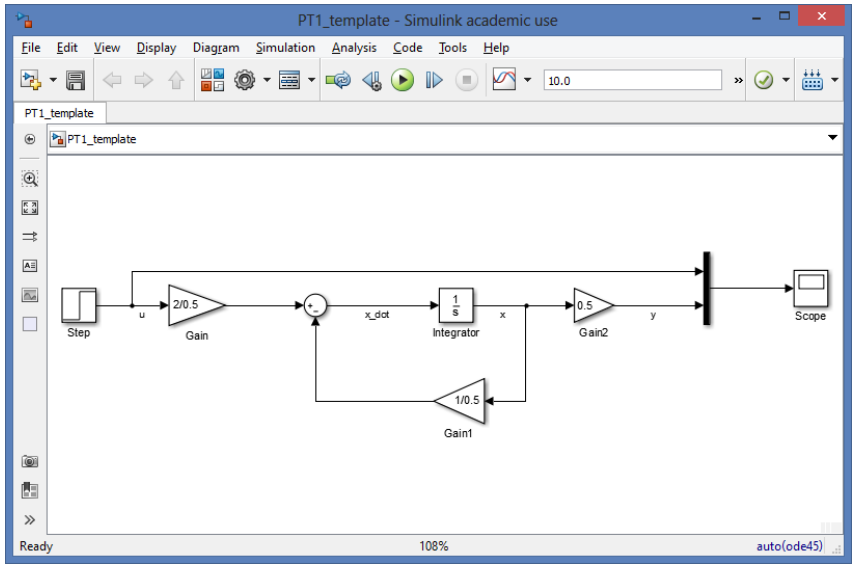
3. Connect Blocks

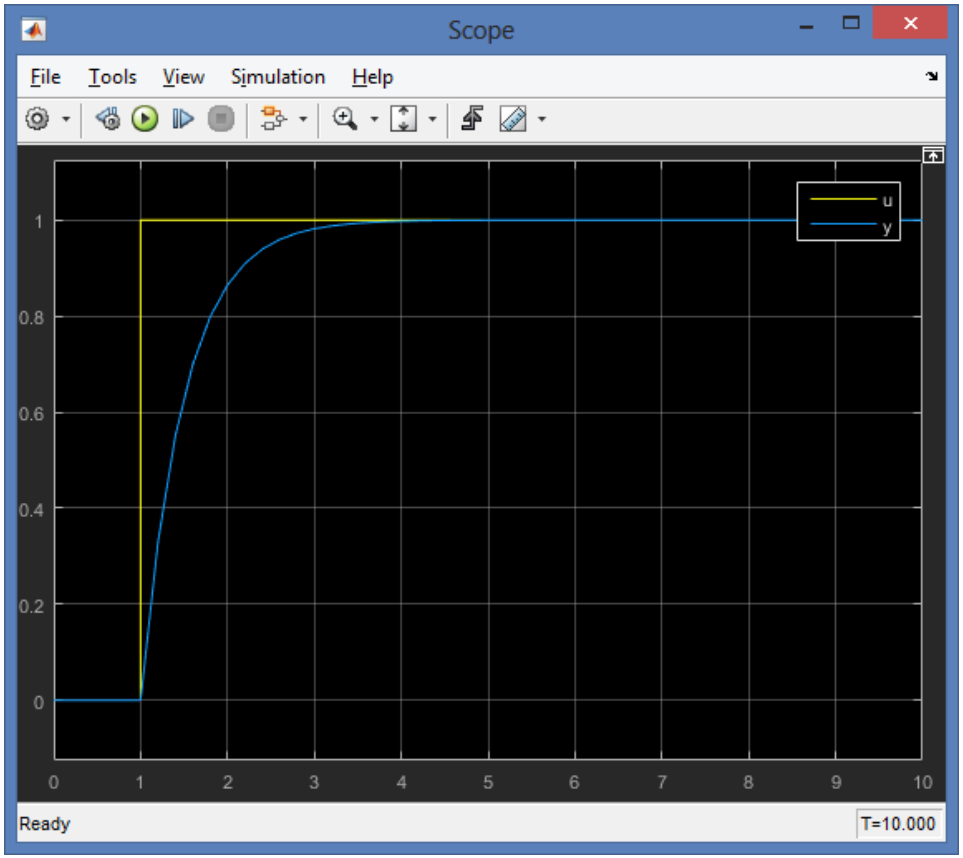


# A Simple Example System

## Simulink Solution

4. Simulate



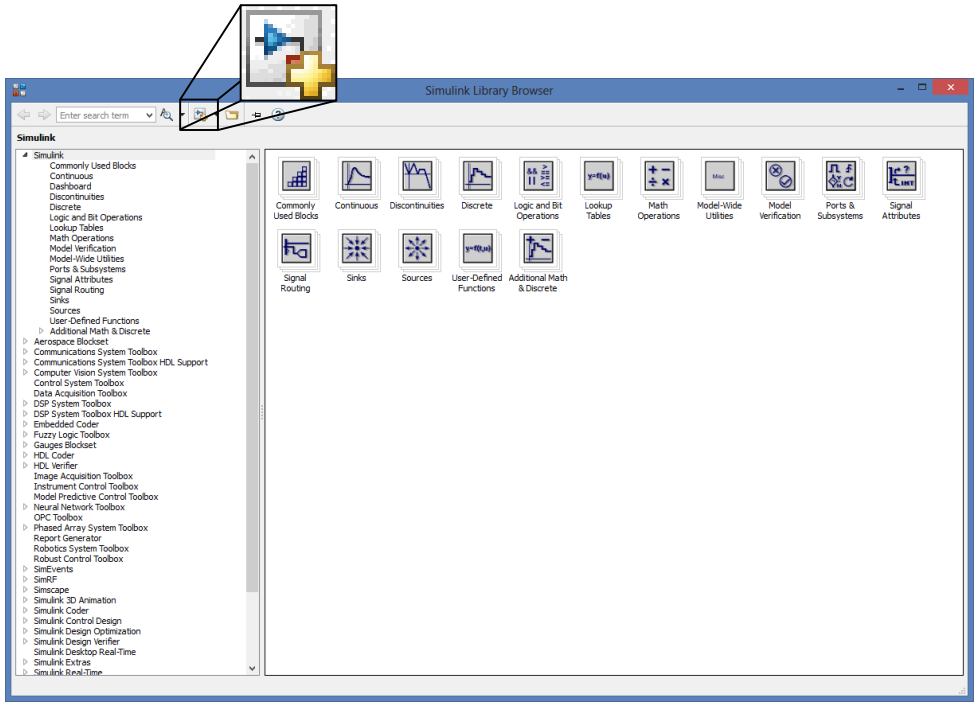
# 3. Start-Up Simulink

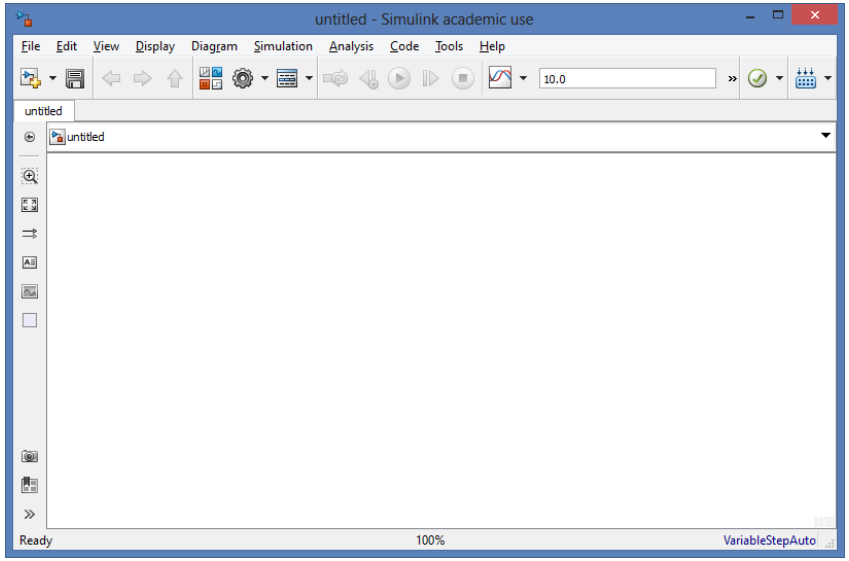
```
fx >> simulink
```

# Simulink Startup

## Command line:

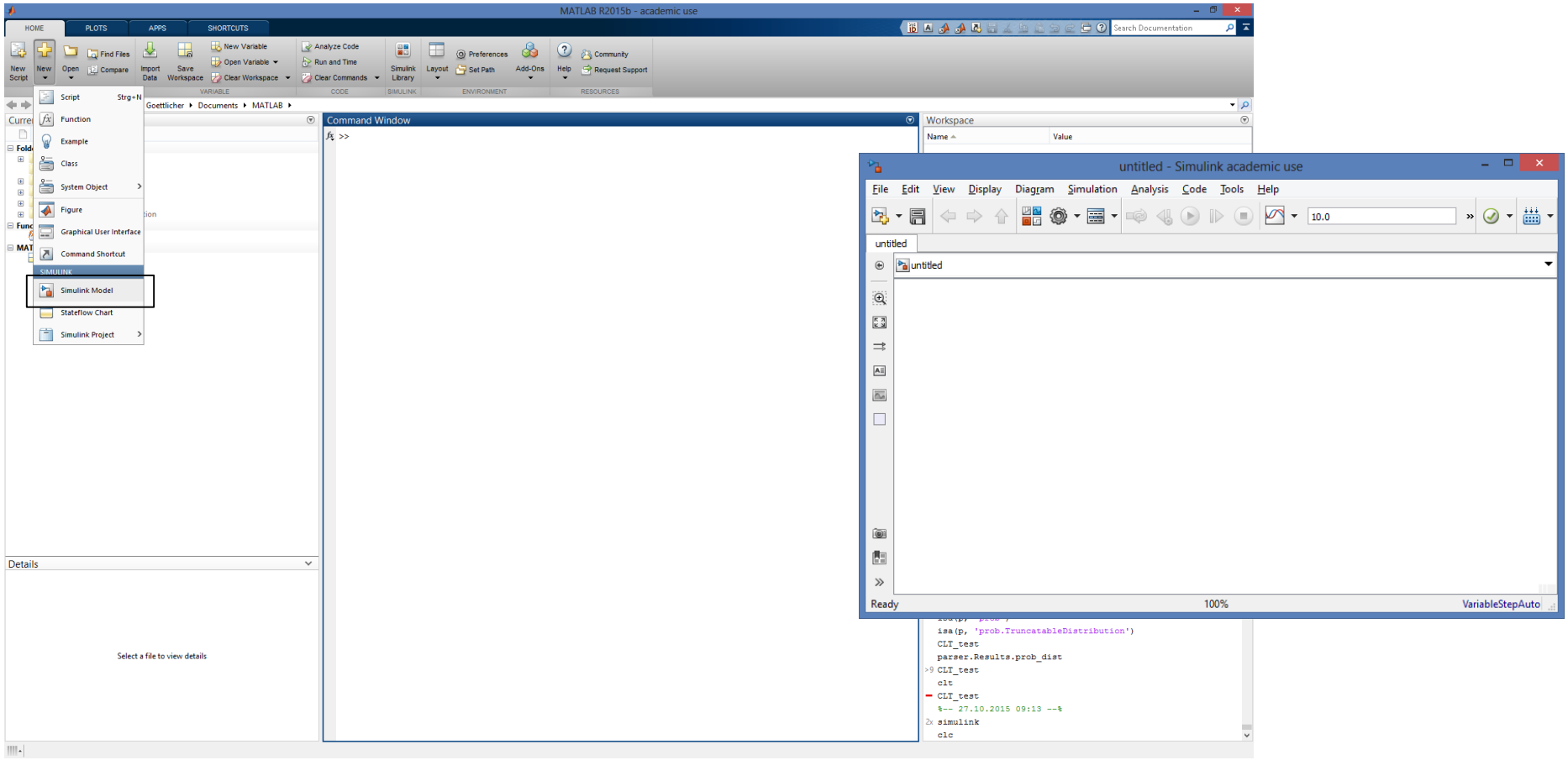
```
>>simulink
```



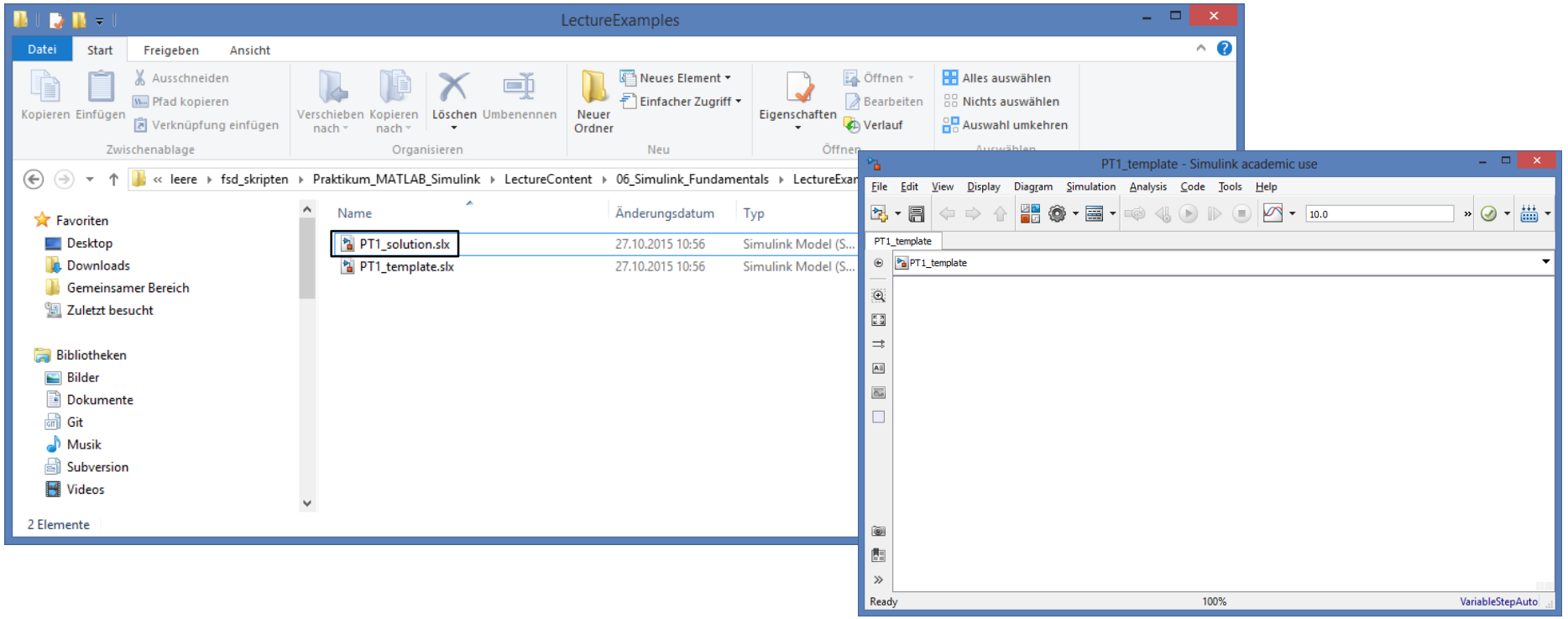
# Simulink Startup

New -> Simulink Model

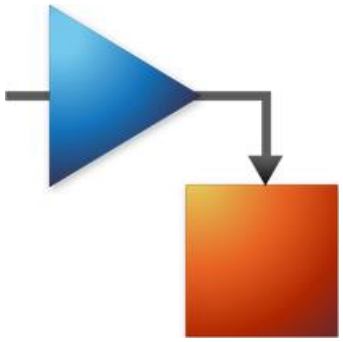


# Simulink Startup

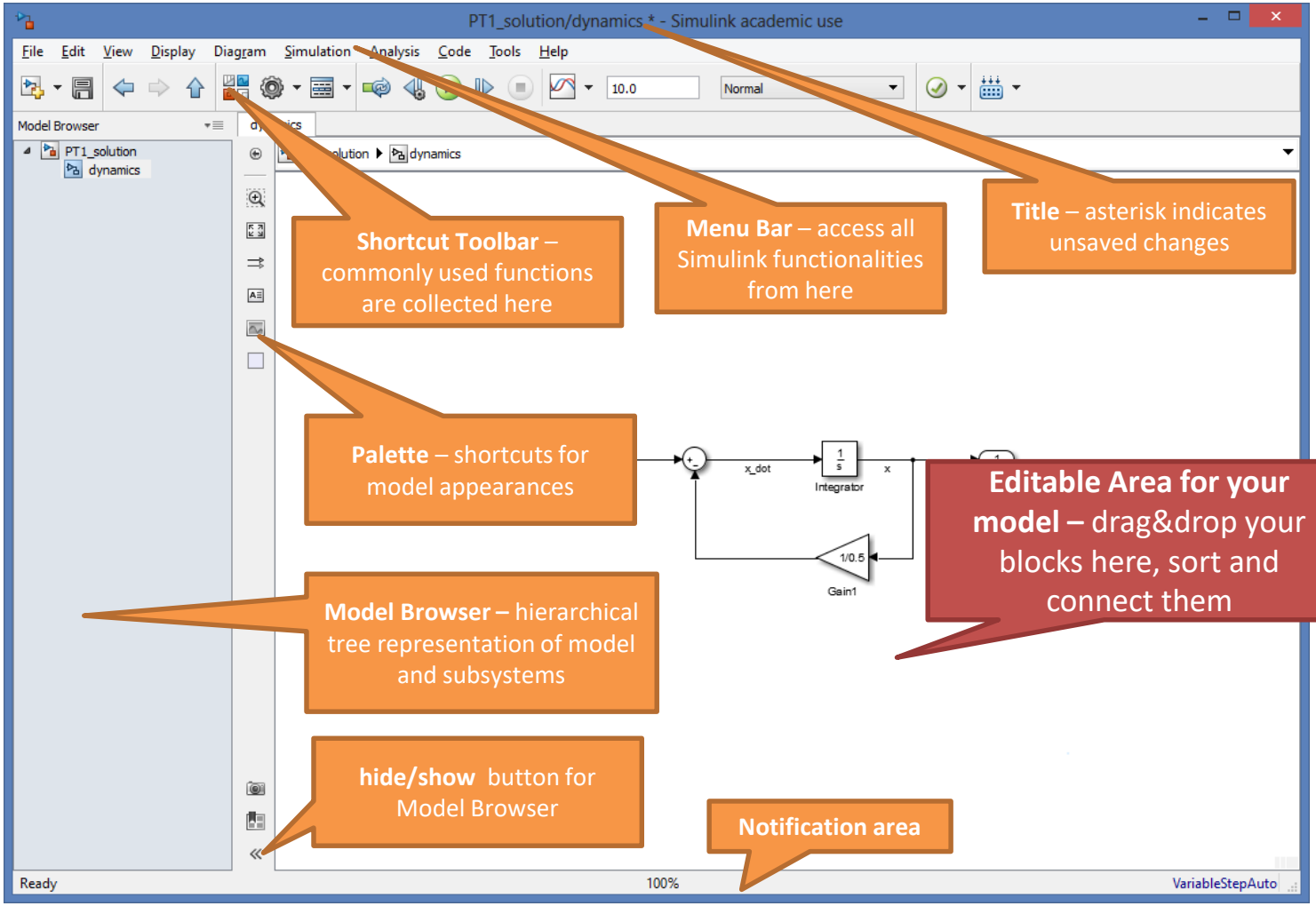
## Existing model via Windows Explorer



# 4. The Simulink Editor

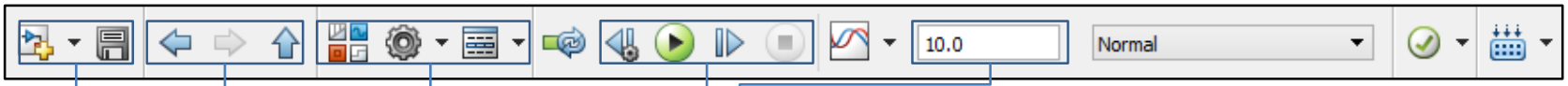


# The Simulink Editor




- Main modeling interface for Simulink!
- Use oriented at other vector graphics programs

# The Simulink Editor




**Simulation stop time:** final time of the simulation

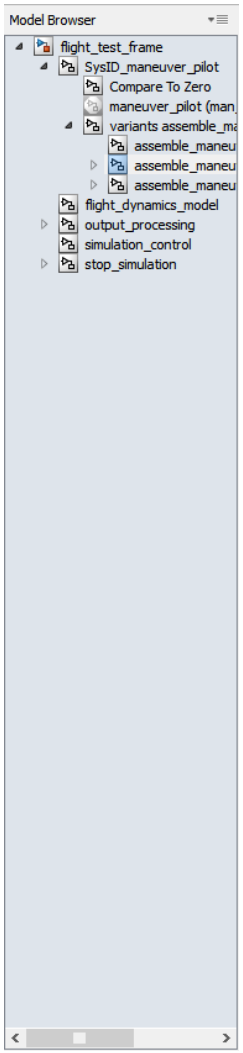
**Stepping Options**: configure single-stepping simulation
**Run / Pause / Continue**: Start / Pause / Continue Simulation
**Step Forward**: do one simulation step
**Stop:** Stop Simulation

**Library Browser:** Open Library Browser
**Model Configuration Parameters:** Open Configuration Parameters Dialog
**Model Explorer:** Open Model Explorer Window

**Back:** undo last step (`Ctrl+Z`)
**Forward:** redo last step (`Ctrl+Y`)
**Up To Parent:** Move out of current subsystem to parent model

**Save:** Save current work progress
**New Model:** create new, empty model

# The Simulink Editor

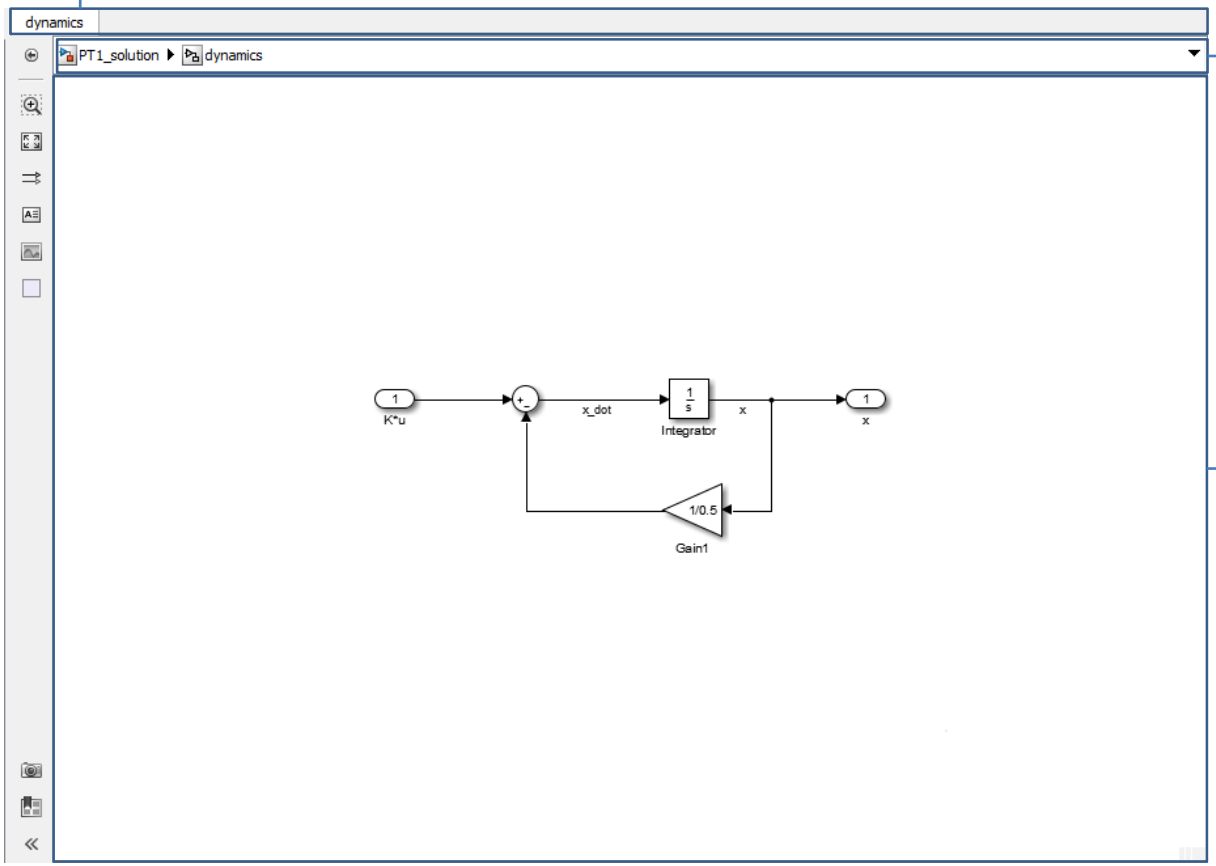



**Model Browser**
Hierarchical tree structure of the model;
Use this to quickly navigate between subsystems

# The Simulink Editor

**Tab Bar** – different model views can be opened in different tabs

**Address Bar** – address (all parent subsystems) of current subsystem



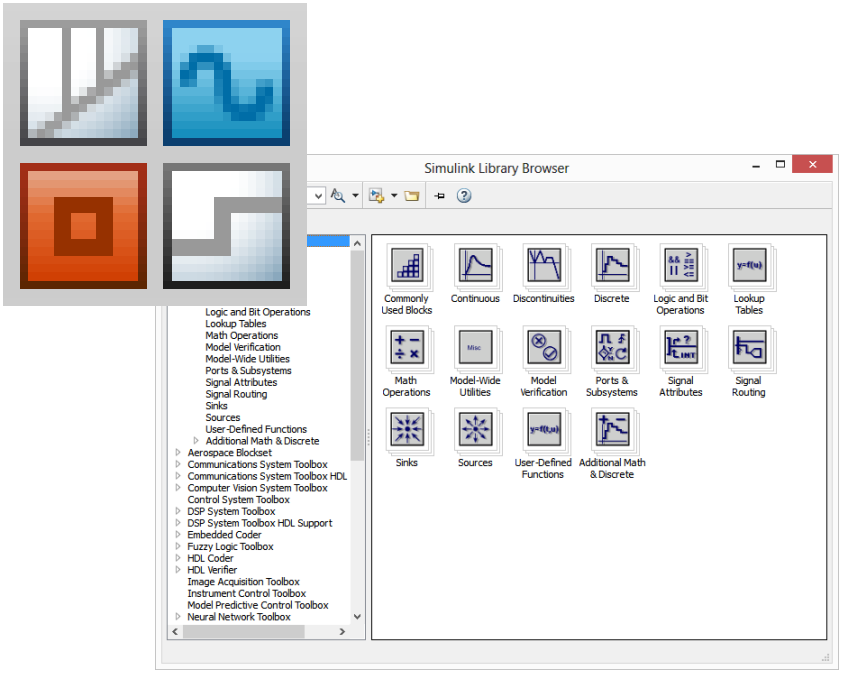

**Editable Area** –
This is where the magic happens

# The Simulink Editor

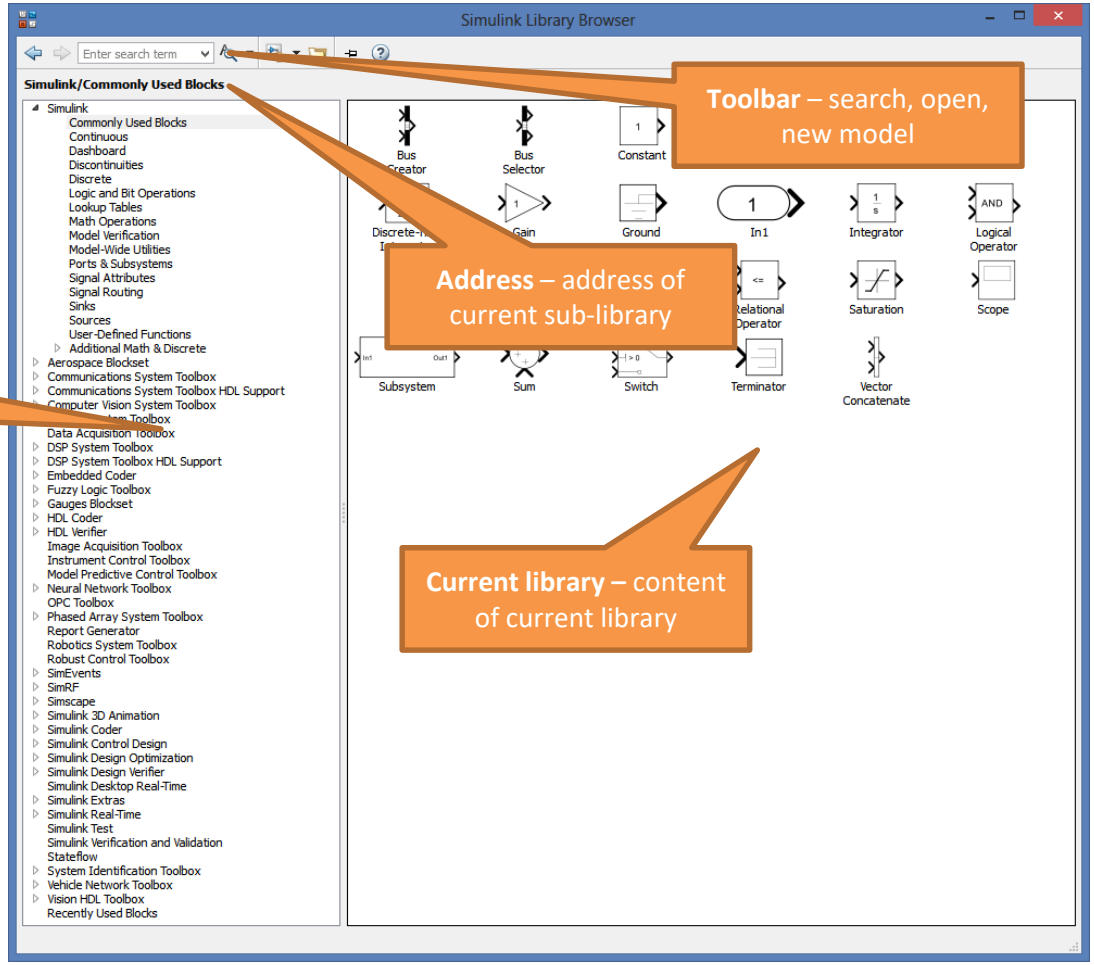
Navigating a model - Shortcuts:

| | |
|---|---|
| `double-click on subsystem` | enter subsystem |
| `mouse-wheel` | zoom in / out |
| `push & hold mouse-wheel` | pan diagram |
| `space` | fit diagram to window |
| `Alt + 1` | center view and zoom 100% |

# 5. The Library Browser



# The Library Browser




**Central source for blocks**

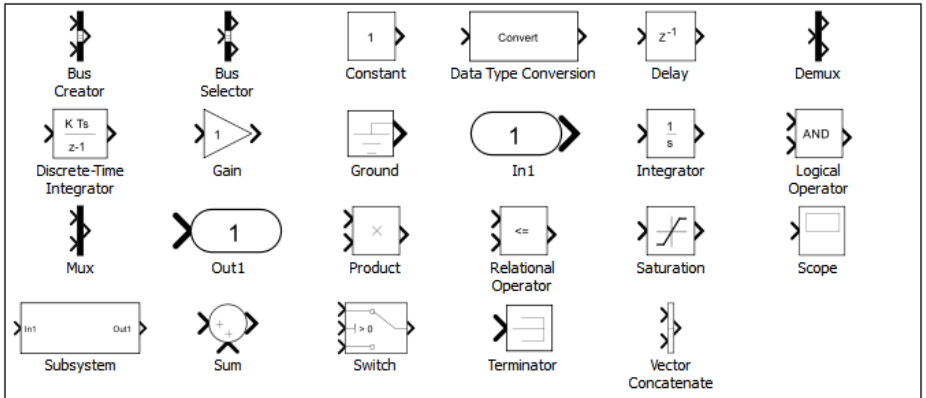
# The Library Browser


**New Model:** opens a new, empty model
**Open Model or Library:** dialog to open existing model / library

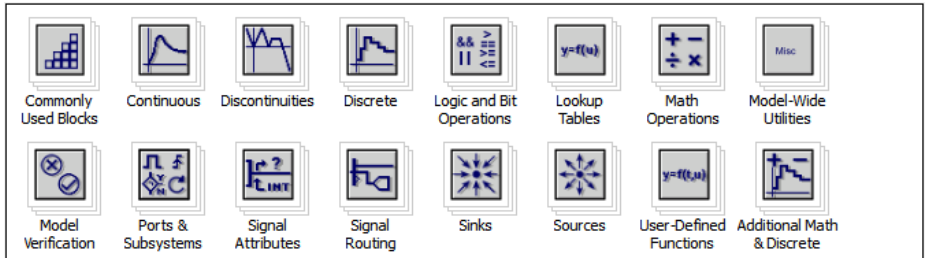
**Search:** enter search term and hit enter. Use this if you only know parts of the block name, or want to see if there is sth for your intended use.
**Customize Search:** regular expression, match case, whole word

**Back:** go back
**Forward:** go forward

# The Library Browser




**Library can contain usable blocks:**
➔ drag-& drop blocks to simulink editor

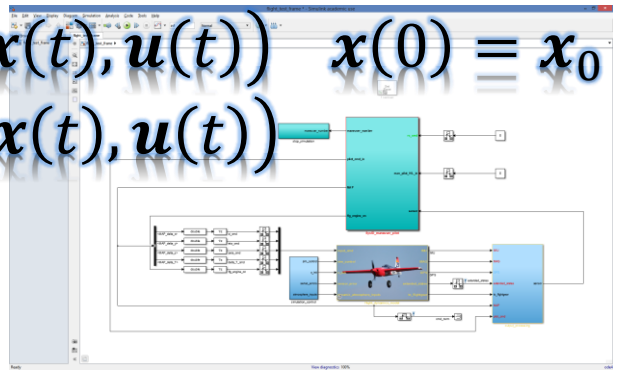



**Library can contain sub-libraries:**
➔ navigate further down tree-structure

# 6. Modeling in Simulink

- **the Working Principle of Simulink**
- Blocks
- Signals
- Subsystems
- Parameters
- Model Configuration Parameters

$$\dot{x}(t) = f\big(t, x(t), u(t)\big) \quad x(0) = x_0$$
$$y(t) = g\big(t, x(t), u(t)\big)$$



# Modeling in Simulink
## the Working Principle of Simulink

Simulink solves systems of non-linear, first order, **ordinary differential equations (ODE)**

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{f}\big(t, \boldsymbol{x}(t), \boldsymbol{u}(t)\big)$$
$$\boldsymbol{x}(t=0) = \boldsymbol{x_0}$$

using numerical integration schemes.

Consider a linear Taylor approximation of $\boldsymbol{x}(t)$ at time $t_i$

$$\boldsymbol{x}(t) = \boldsymbol{x}(t_i) + \left.\frac{d\boldsymbol{x}(t)}{dt}\right|_{t=t_i} (t - t_i) + h.o.t.$$
$$\approx \boldsymbol{x}(t_i) + \boldsymbol{f}\big(t_i, \boldsymbol{x}(t_i), \boldsymbol{u}(t_i)\big) \cdot (t - t_i)$$

Then $\boldsymbol{x}(t)$ at the next time instant $t_{i+1}$ can be approximated by (using the **integration time step** $\Delta t = t_{i+1} - t_i$):

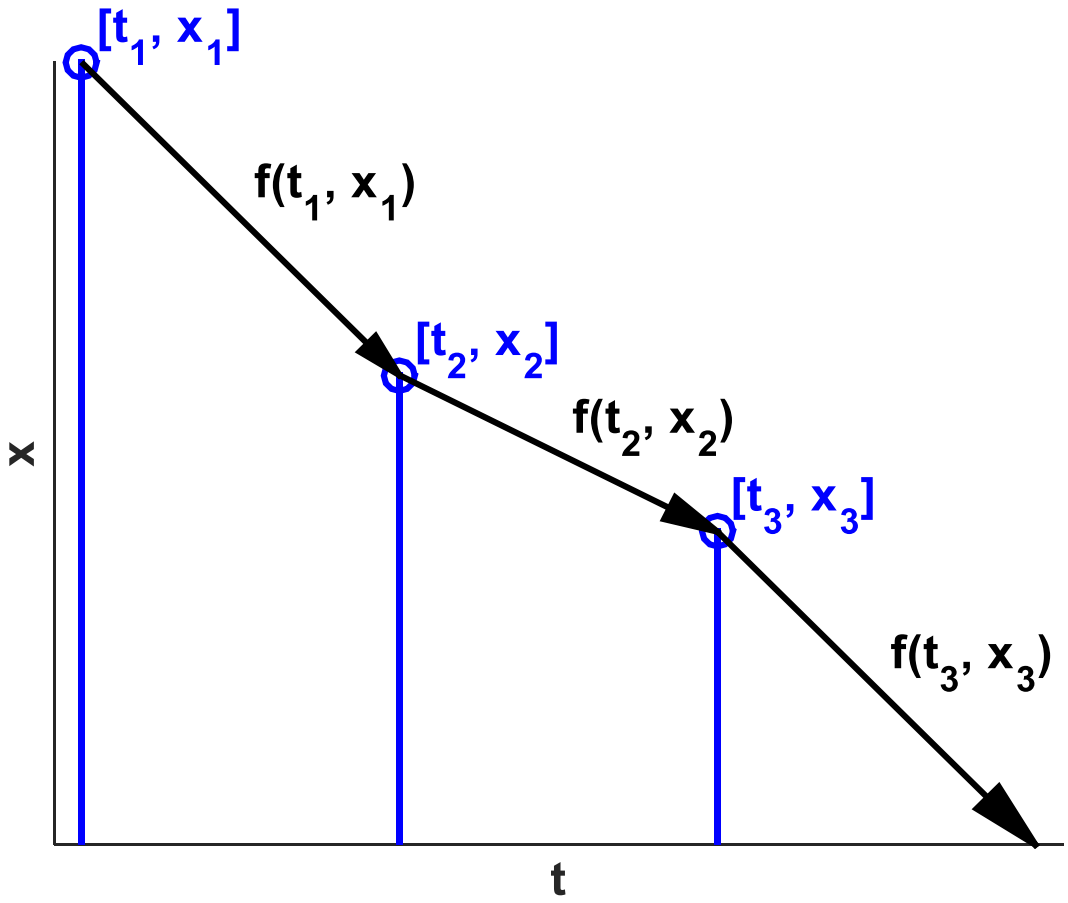
$$\boldsymbol{x}(t_{i+1}) = \boldsymbol{x}(t_i) + \boldsymbol{f}\big(t_i, \boldsymbol{x}(t_i), \boldsymbol{u}(t_i)\big) \cdot \Delta t_i$$

**This results in the so called „Euler Integration Scheme“**

sometimes referred to as first order integration, since the overall approximation error is of order $\mathcal{O}(\Delta t)$

# Modeling in Simulink
## the Working Principle of Simulink

Euler‘s Method results in a linear, one-step approximation of the ode‘s solution:



# Modeling in Simulink
## the Working Principle of Simulink

If more evaluations of the ode's right hand side at intermediate points are considered, **multi-step** methods arise:

*Instead of one evaluation of $\boldsymbol{f}(t, \boldsymbol{x}, \boldsymbol{u})$, (as in the case of the Euler method) a weighted sum of intermediate evaluations of $\boldsymbol{f}(t, \boldsymbol{x}, \boldsymbol{u})$ is used to compute the next step.*

One example is the "classic" Runge-Kutta method (4th order):

with

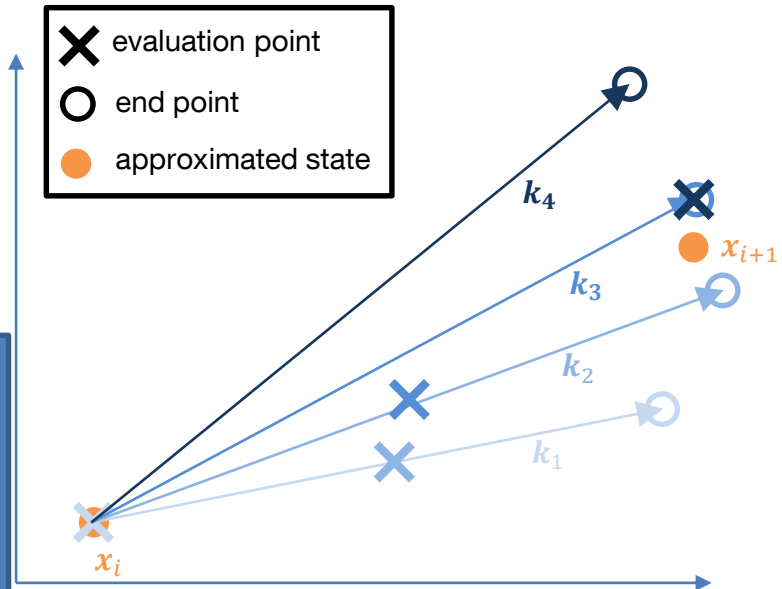
$$\boldsymbol{k}_1 = \boldsymbol{f}\left(t_i, \boldsymbol{x}_i,\ \boldsymbol{u}(t_i)\right)$$

$$\boldsymbol{k}_2 = \boldsymbol{f}\left(t_i + \frac{\Delta t}{2}, \boldsymbol{x}_i + \frac{\Delta t}{2}\boldsymbol{k}_1,\ \boldsymbol{u}\left(t_i + \frac{\Delta t}{2}\right)\right)$$

$$\boldsymbol{k}_3 = \boldsymbol{f}\left(t_i + \frac{\Delta t}{2}, \boldsymbol{x}_i + \frac{\Delta t}{2}\boldsymbol{k}_2,\ \boldsymbol{u}\left(t_i + \frac{\Delta t}{2}\right)\right)$$

$$\boldsymbol{k}_4 = \boldsymbol{f}\left(t_{i+1}, \boldsymbol{x}_i + \Delta t\boldsymbol{k}_3,\ \boldsymbol{u}(t_{i+1})\right)$$

$$\boldsymbol{x}_{i+1} = \boldsymbol{x}_i + \frac{\Delta t}{6}(\boldsymbol{k}_1 + 2\boldsymbol{k}_2 + 2\boldsymbol{k}_3 + \boldsymbol{k}_4)$$

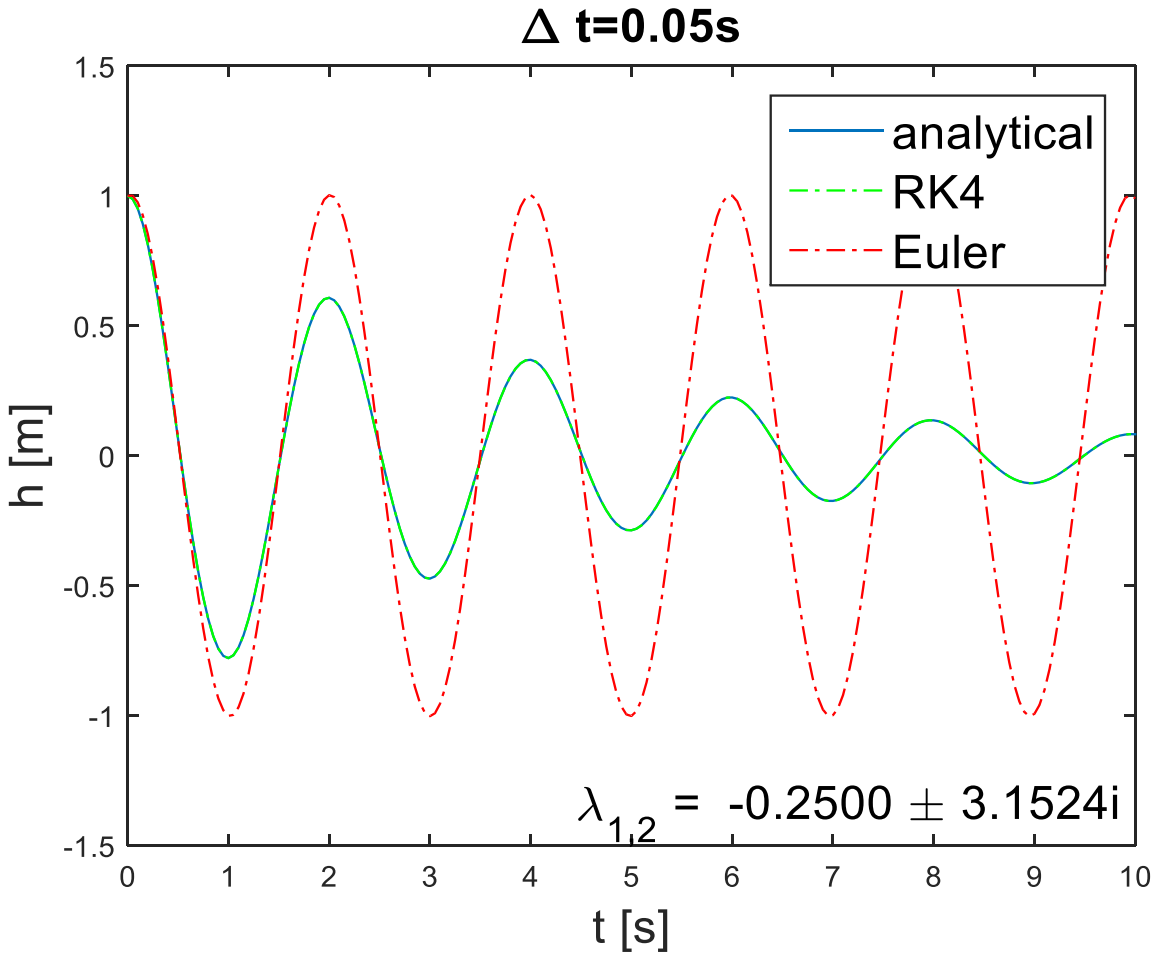



**This results in the "classic" Runge-Kutta Integration Scheme**

sometimes referred to as 4th order integration, since the overall approximation error is of order $\mathcal{O}(\Delta t^4)$

# Modeling in Simulink
## the Working Principle of Simulink

Compare integration methods (simple oscillator) :



# Modeling in Simulink
# the Working Principle of Simulink

## Variable Step-Size Algorithms

- For many multi-step methods, the result of the method with one less error order can be obtained without additional evaluations of $\boldsymbol{f}(t, \boldsymbol{x}, \boldsymbol{u})$
  ➔ i.e. 5th order integration yields result for 4th order integration as by-product
- Variable step size algorithms adjust $\Delta t$ based on the difference of two consecutive methods:

$$\left|\boldsymbol{x}_{i+1,RK5} - \boldsymbol{x}_{i+1,RK4}\right| \geq \epsilon_{upp} \quad \Rightarrow \text{decrease } \Delta t$$

$$\left|\boldsymbol{x}_{i+1,RK5} - \boldsymbol{x}_{i+1,RK4}\right| \leq \epsilon_{low} \quad \Rightarrow \text{increase } \Delta t$$

- This yields `ode45 (Dormand-Prince)`

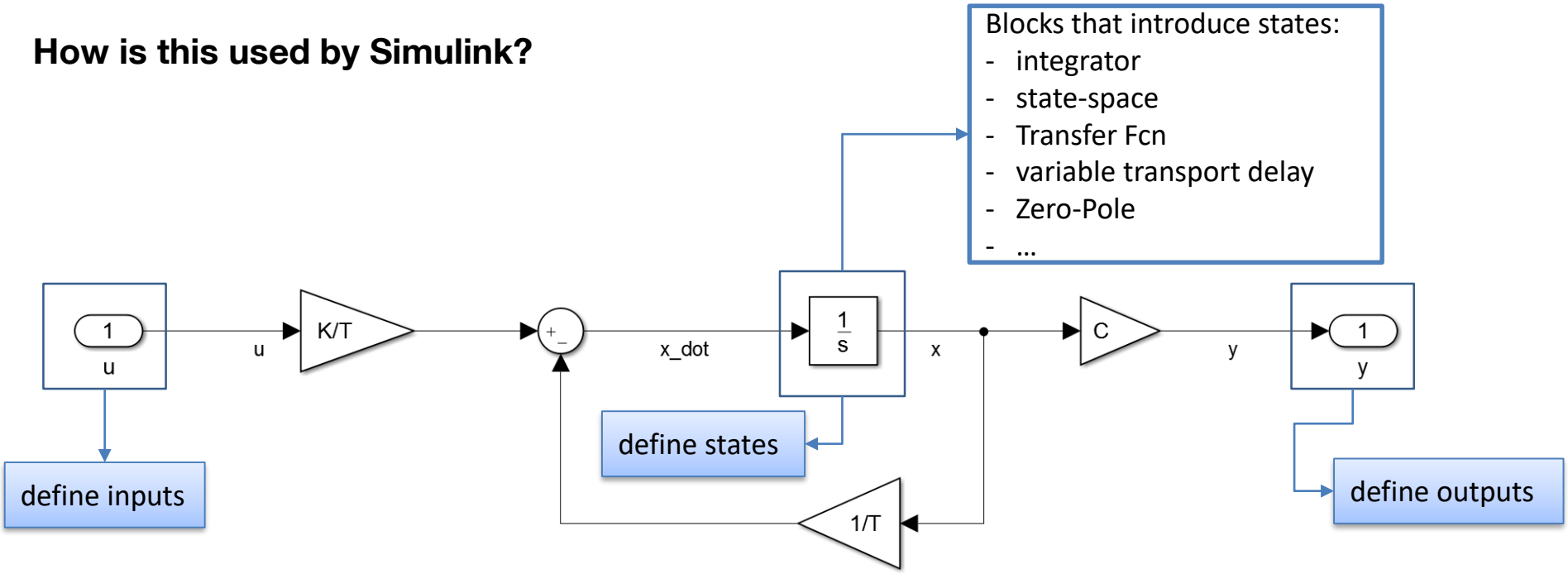
➔ Thus the step-size is adapted to the system's dynamics
(fast-changing → small step size; slow-changing → large step size)

➔ Results in comparatively fast **AND** accurate results

➔ However, **NO** a priori knowledge of evaluation times $t_i$! They are adapted online.

# Modeling in Simulink
## the Working Principle of Simulink

**How is this used by Simulink?**

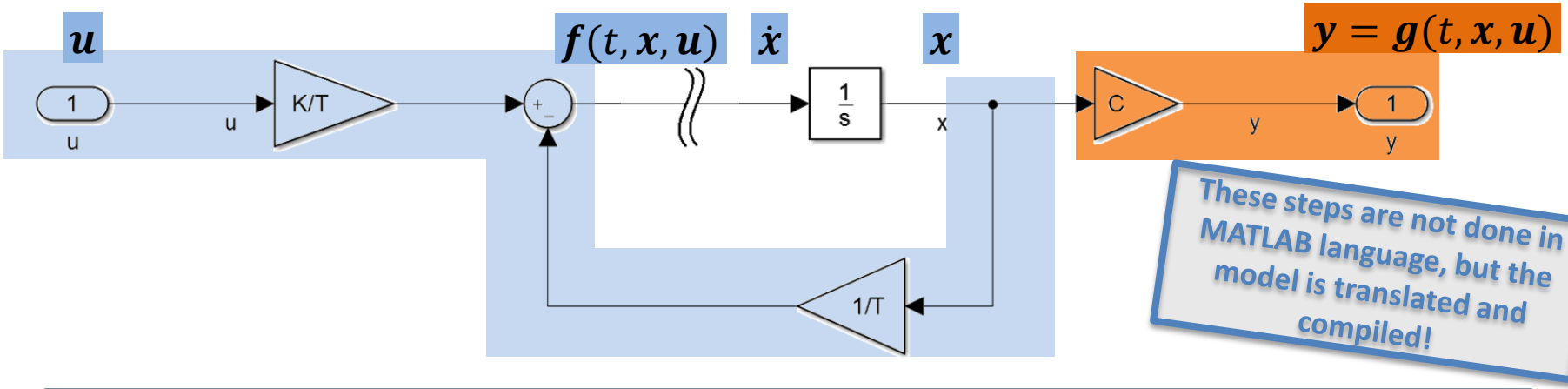


# Modeling in Simulink
## the Working Principle of Simulink

**How is this used by Simulink?**

I. translate block diagram to system of first order ordinary differential equations
   1. define states, state derivatives, outputs and inputs
   2. ‚cut' in front of integrators
   3. from integrator: go backwards through model until an input `u` or state `x` is reached ➔ $\boldsymbol{f(t,x,u)}$
   4. from output: go backwards until an input `u` or state `x` is reached ➔ $\boldsymbol{f(t,x,u)}$

II. solve system using above mentioned integration schemes




These steps are not done in MATLAB language, but the model is translated and compiled!

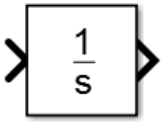
example system:

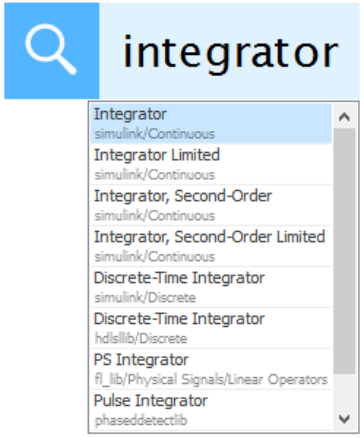
$$\dot{x} = f(t,x,u) = -\frac{1}{T}x + \frac{K}{T}u$$

$$y = g(t,x,u) = Cx$$

# 6. Modeling in Simulink

- the Working Principle of Simulink
- **Blocks**
- Signals
- Subsystems
- Parameters
- Model Configuration Parameters

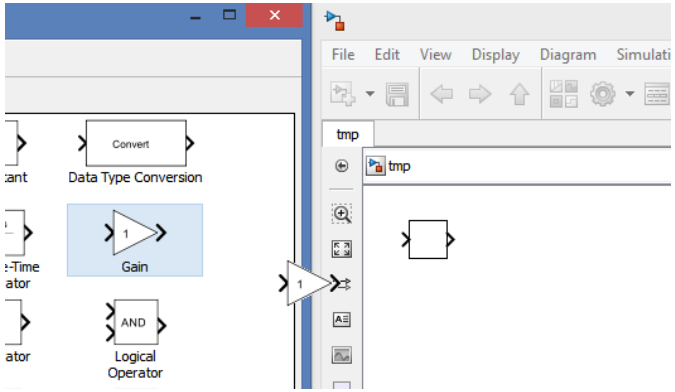


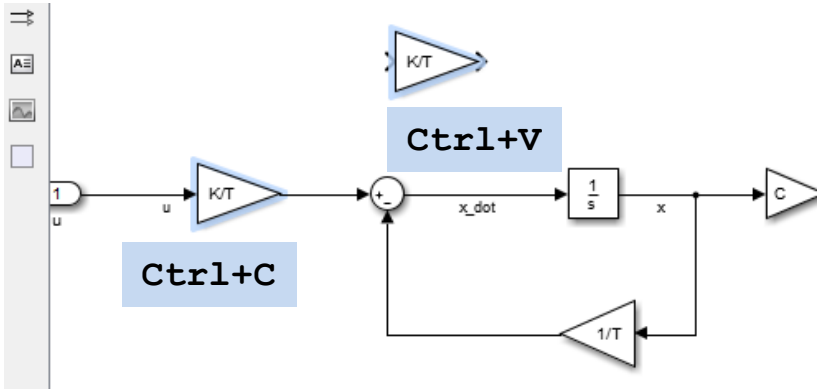
# Modeling in Simulink
## Blocks

add a block to a Simulink model:

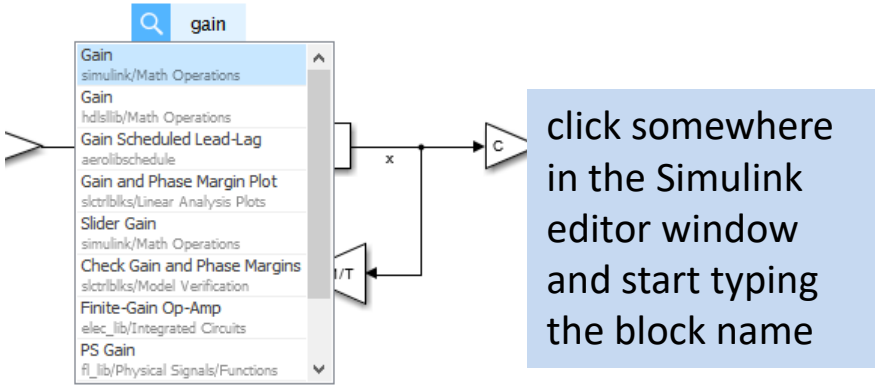
### drag & drop from Library Browser




### copy & paste another block

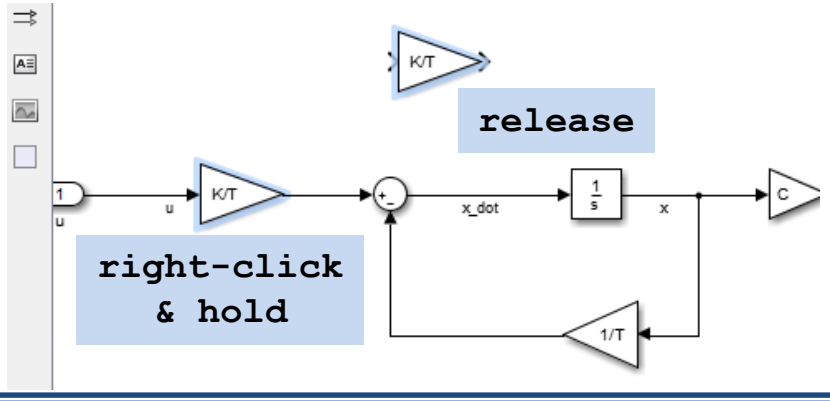



### use intelligent search



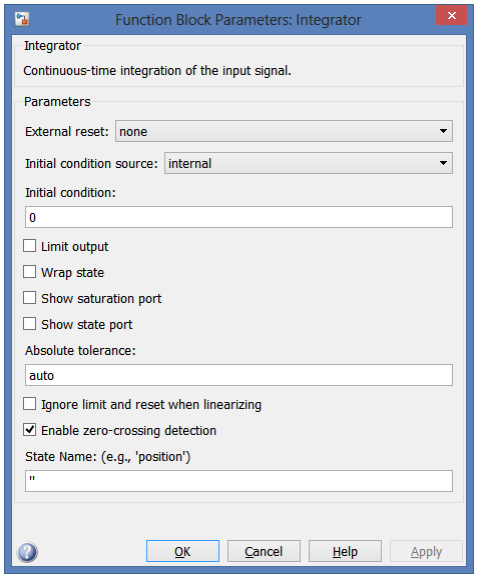

click somewhere in the Simulink editor window and start typing the block name

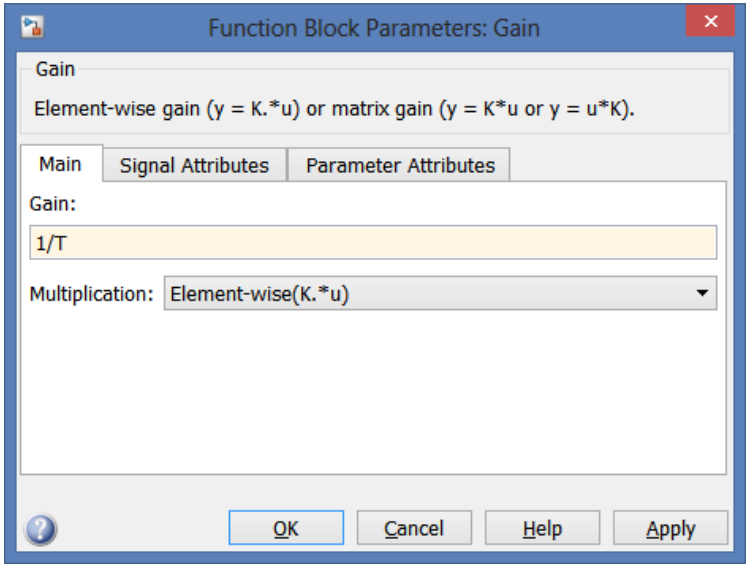
### drag & drop with right mouse button

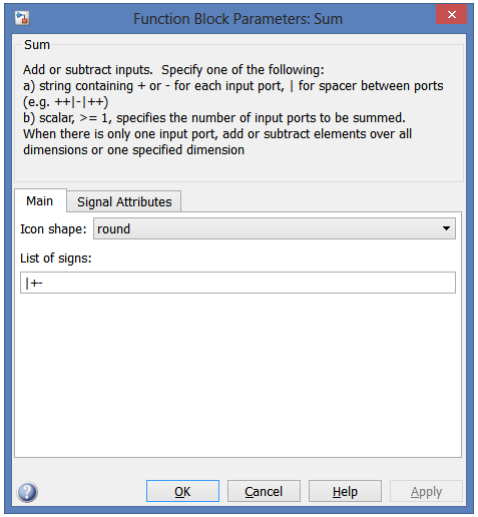


# Modeling in Simulink
## Blocks

- a double click on a block opens a specific configuration window.
- Here the block's properties can be adjusted



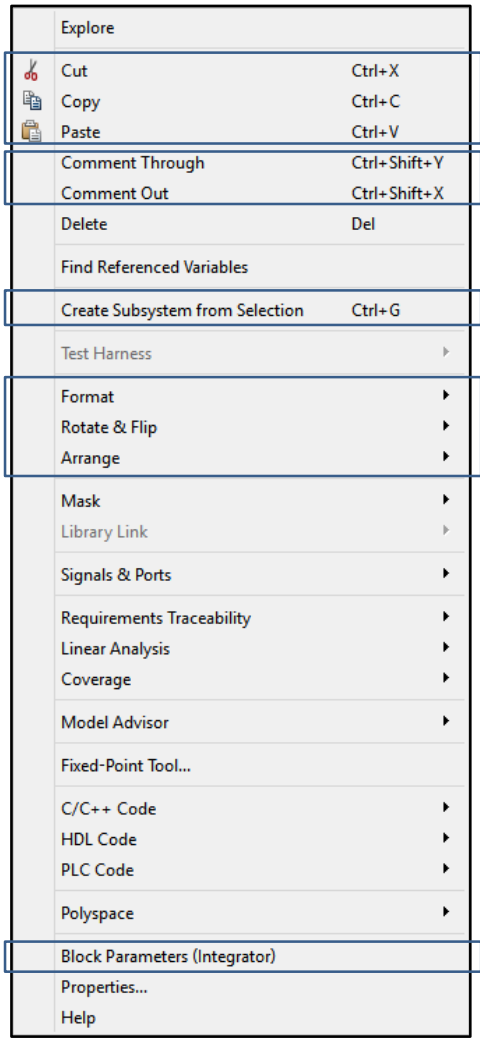




**instead of numerical values, variable names can be used as well.**

As long as those variables are defined in the workspace (or somewhere else). For example the Gain here is `1/T`, where `T` is defined in the base workspace.

# Modeling in Simulink
## Blocks



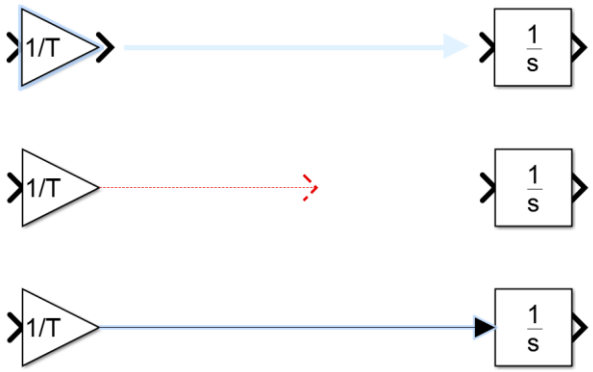

**most important block context menu entries**

*(right-click on any block to obtain this menu)*

- same as any other program…
- „comment out“ block, i.e. make it inoperative
- „take all marked blocks, and put them in one common subsystem“
- change appearance of block (color, font color, font size, shadows, block orientation, …)
- same as double-clicking on a block: change its characteristic behaviour

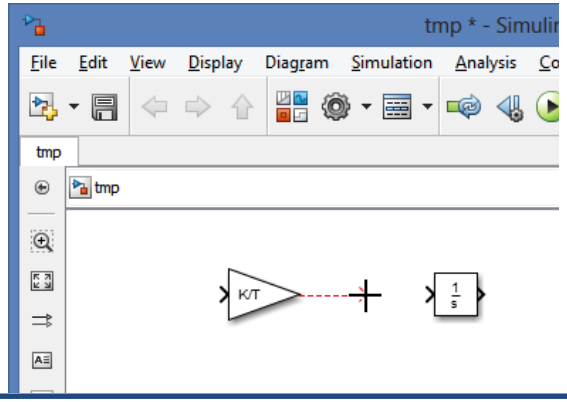
# 6. Modeling in Simulink

- the Working Principle of Simulink
- Blocks
- **Signals**
- Subsystems
- Parameters
- Model Configuration Parameters

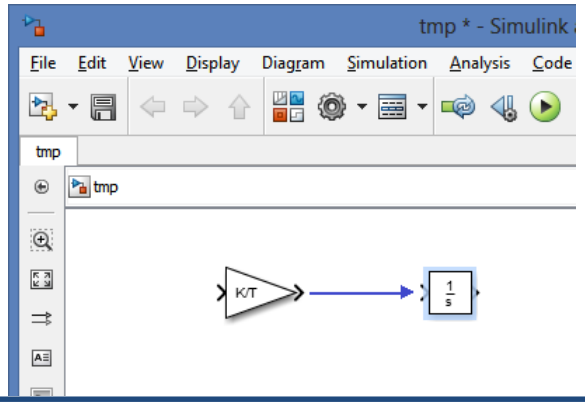


# Modeling in Simulink
## Signals

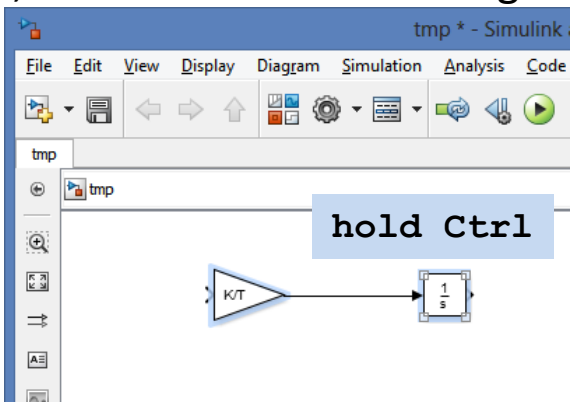
click on outport, hold to connect to inport *(or vice versa)*



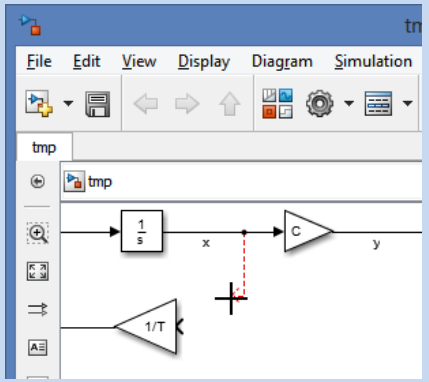

when blocks are aligned, click on blue arrow




mark first block; hold `Ctrl` while clicking on the second. *(useful for connecting multiple outputs to multiple inputs)*



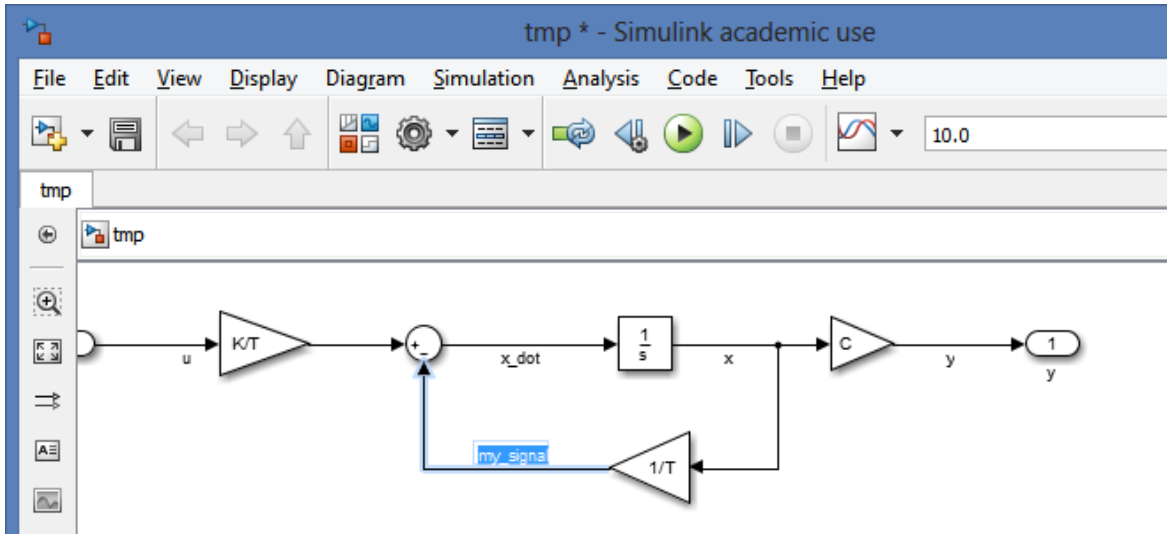

**to branch signals:** right click and hold on existing singal



# Modeling in Simulink
## Signals

**to name a signal, double click on it and type name:**

(make sure to really click **on the signal**. Double-clicking somewhere and typing adds an annotation, which is **not** associated with a signal!)




- signal names allow the unique identification of a signal
- signal names make your model more readable
- signal names can be used for logging a signal (see later slides)
- signal names allow unique mapping in a Bus signal (see later slides)

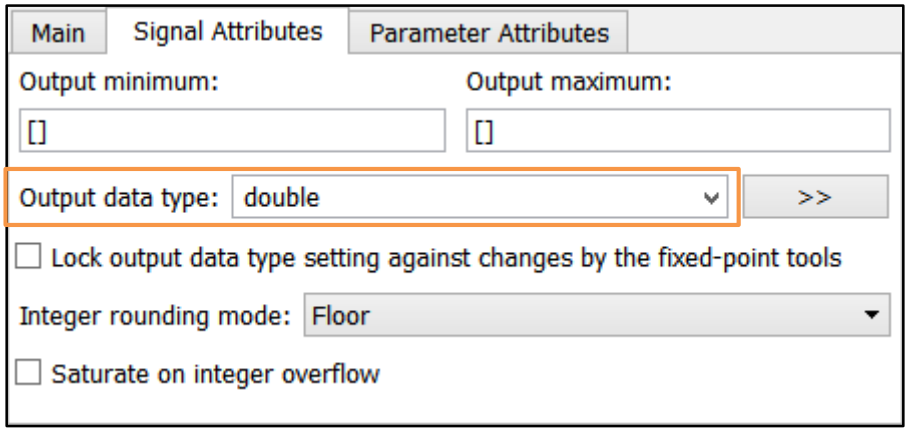
# Modeling in Simulink
## Signals

**Signals always have a designated data type. (Default is 'double')**

*(activate display of signal data type: Menubar -> Display -> Signals & Ports -> Port Data Types)*

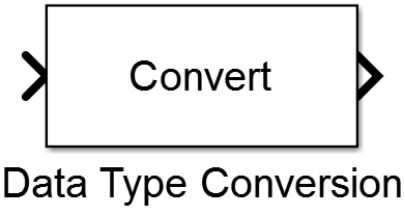
common data types are: double, single, int32, uint32, int16, uint16, int8, uint8, bool

### set via Block Properties




- many blocks have a 'Signal Attributes' tab in their block properties menu.
- here the output data type can be set

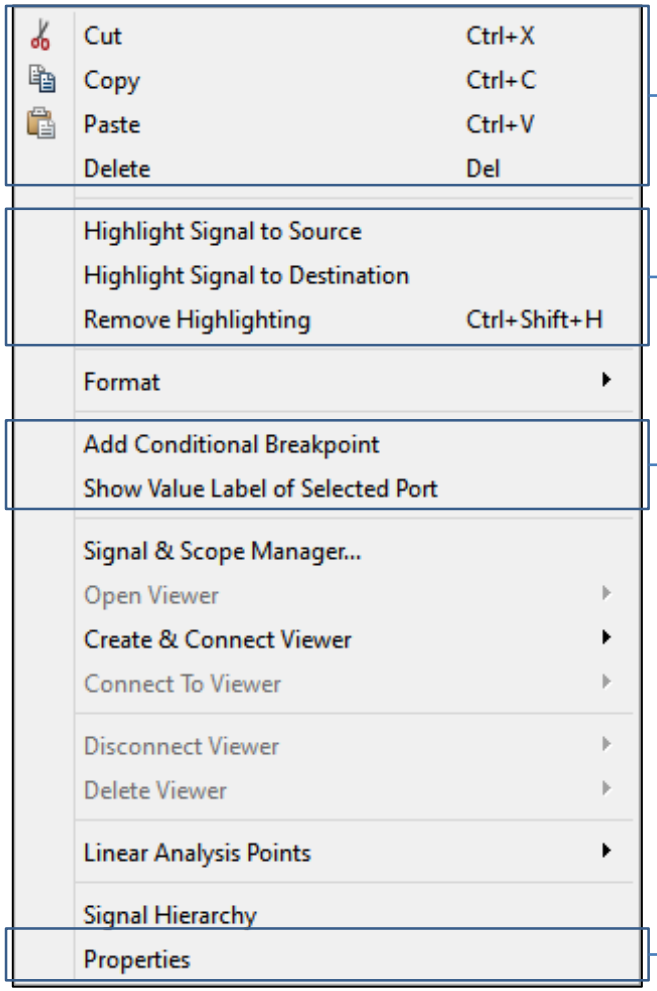
### set via 'Data Type Conversion' Block




- block to explicitly change singal data type
- equivalent to 'casting' in C / C++

the data type can be set to "Inherit via…" then Simulink applies internal rules to determine it

# Modeling in Simulink
## Signals




same as any other program…

in very complex systems, used to „understand“ signal by looking at ist source / destination
*(if you need this, you did something wrong! group blocks in subsystems to avoid overly-complex systems)*

**Add Conditional Breakpoint:**
Add Breakpoint for debugging; Simulation breaks, when signal fulfills condition
**Show Value Label of Selected Port:**
displays value of signal online during Simulation. Good for debugging / investigating the behavior of a system without having to connect a scope

configure signal properties, e.g. **signal logging**

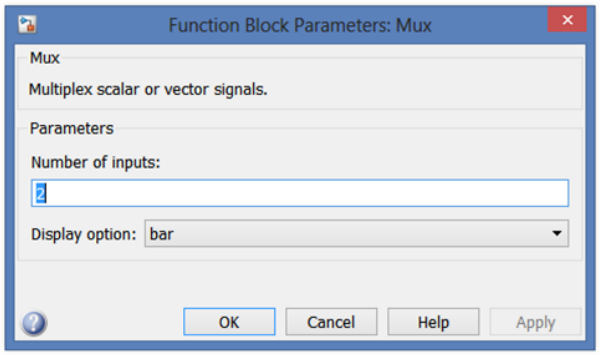
# Modeling in Simulink
## Signals

having only scalar signals can make model „messy“

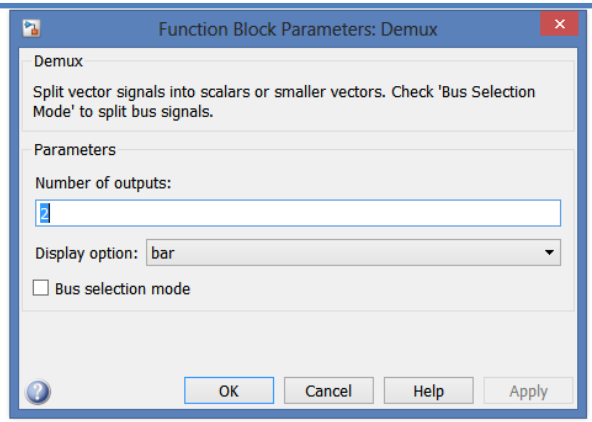
➔ use **Mux blocks** to group signals.

*This is equivalent to ‚arrays‘ in other programming languages (C/C++, Matlab)*

➔ use **Demux blocks** to ungroup signals




- number of inputs can be configured in Block parameters
- the block changes its appearance accordingly
- **only signals of same datatype / sample time can be ‚muxed‘ together!**
- **muxed signals can be inputs to a Mux block, too.**




- number of outputs can be configured in Block parameters
- the block changes its appearance accordingly
- **scalar number of outputs `x`:** input array is split into `x` equally sized signals
- **vector of output sizes `[x1, x2, …]`:** input array is split in signals of sizes $x_1$, $x_2$, …

*activate display of signal dimensions: Menubar -> Display -> Signals & Ports -> Signal Dimensions*

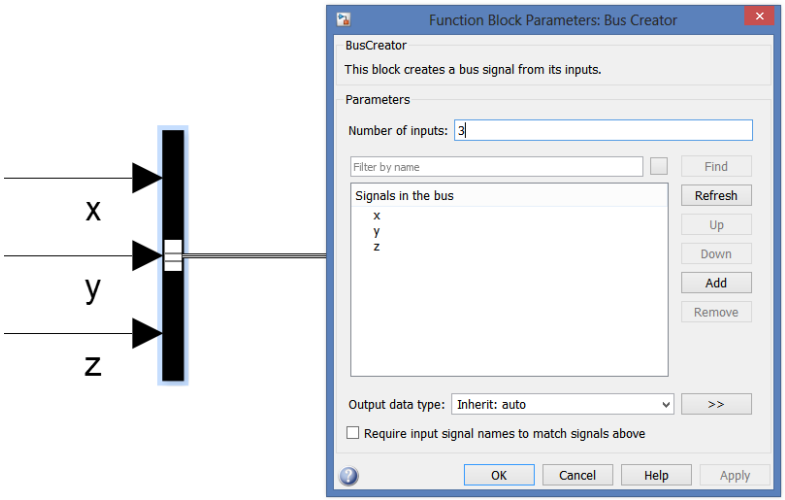
# Modeling in Simulink
## Signals

**Bus Signals** expand the functionality of muxes by storing signal name information as well

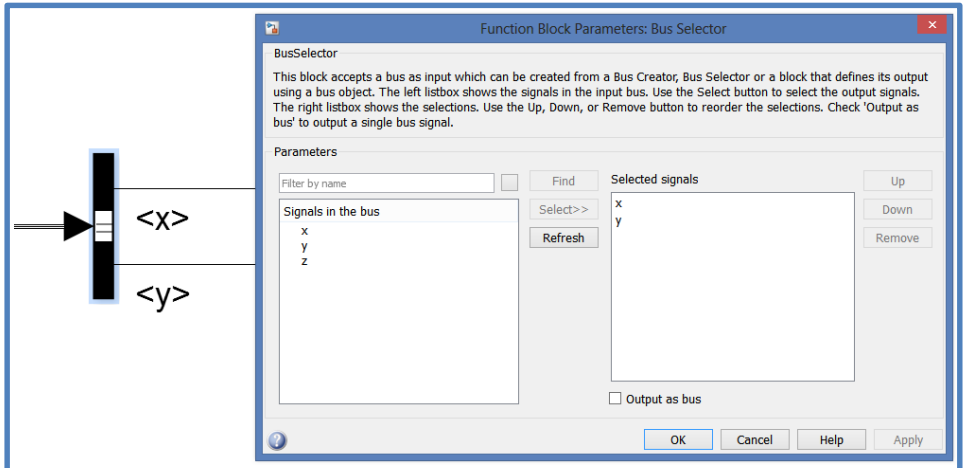
➔ **Bus Creator**: create a bus signal from other signals (scalar-, array- or other Bus Signals).

*This is equivalent to ‚structs' in other programming languages (C/C++, Matlab)*

➔ **Bus Selector:** select a signal from a bus.




- number of inputs can be adjusted in block properties
- signals are listed with their names (`signal_i` if they are unnamed)
- bus signals are displayed as **triple lines**



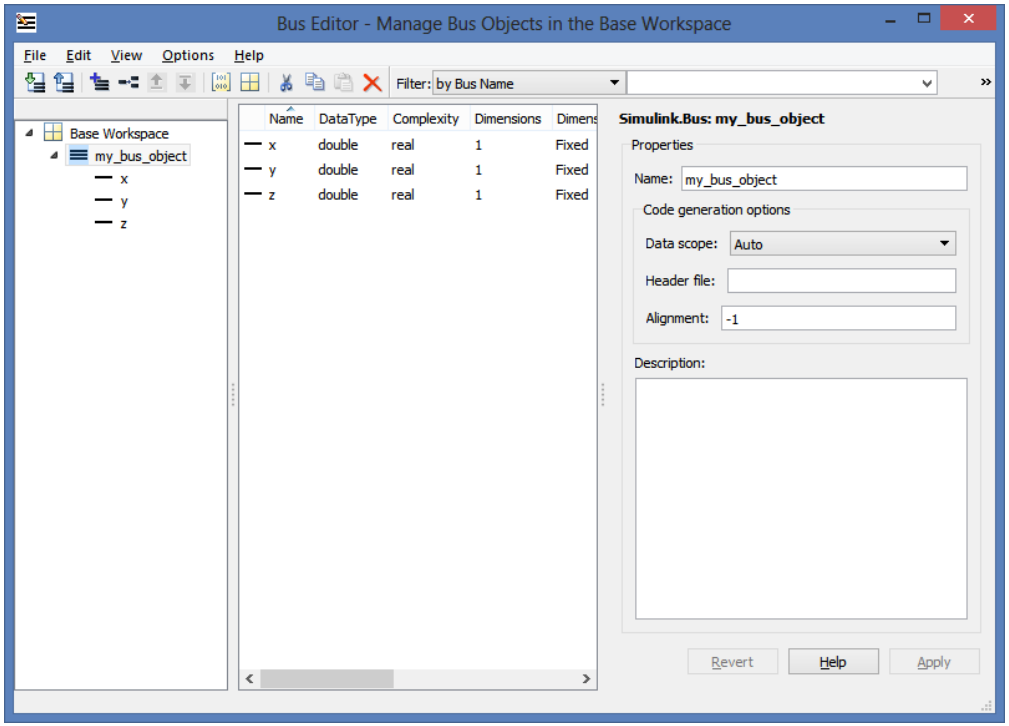

- the signals can be selected by name
- only the signals, which are used can be chosen; not all signals have to be selected

# Modeling in Simulink
## Signals

The structure of Busses can be pre-define via **Bus Objects**.

- type `>> buseditor` in the Matlab command window
- use: *Menubar -> Edit -> Bus Editor*

to open **Bus Editor** window. Here, Bus structure and element properties can be set graphically



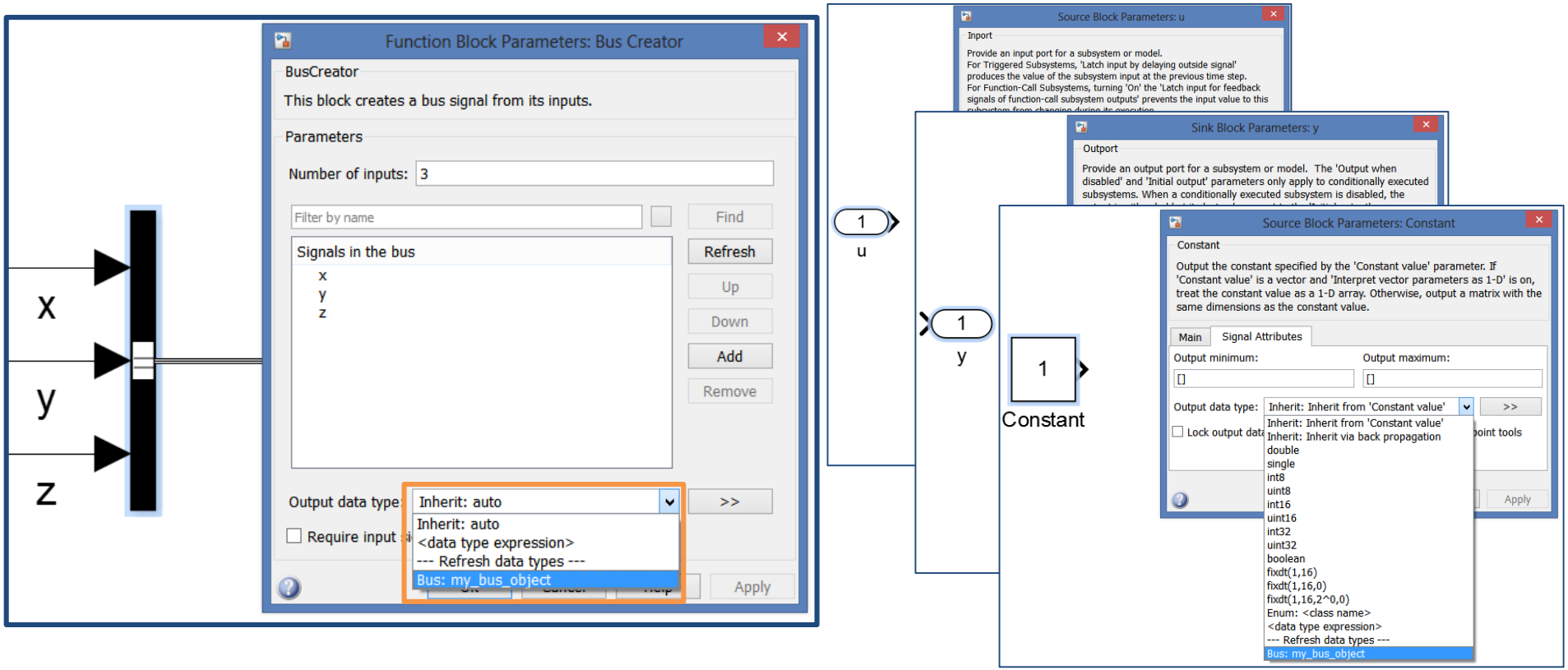

**Properties of Bus Objects**

- **use to define interfaces**
- assure that busses in use have pre-defined structure
- assure that bus-elements have defined data-types
- **necessary when using busses in root-level inports / outports**
- similar to `typedef struct {}` in C/C++ ➔ 'defines new data type'

# Modeling in Simulink
## Signals

The structure of Busses can be pre-define via **Bus Objects**.

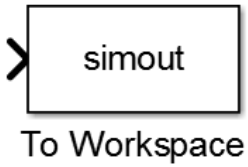
These **Bus Object** can then be chosen as **Data Type** in the **Bus Creator Block,** or any other block, where an output data type can be assigned, e.g. **inport / outport, constant, …**



# Modeling in Simulink
## Signals

To analyze simulation data with MATLAB, there are several possibilities to exchange data:

- **From / To Workspace blocks**
  - signal is stored in / read from MATLAB variable in base workspace
  - the following formats for that variable are possible

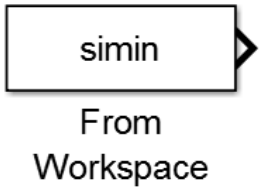



1. MATLAB timeseries object

```
% timeseries
sim_data = timeseries(t, data);
```

2. matrix: first **column** has to be time-vector, subsequent columns are data vectors

```
% matrix
sim_data = [t, data];
```




3. structure: the following fields must present

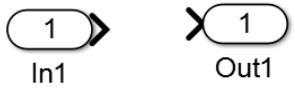
```
var.time=[TimeValues]
var.signals.values=[DataValues]
var.signals.dimensions=[DimValues]
```

```
% struct
sim_data.time = t;
sim_data.signals.values = data;
sim_data.signals.dimensions = size(data);
```

  - intermediate values are interpolated
  - out of range behavior for *from Workspace* block can be adjusted

---

- Root-Level input / output ports:




  - logging signals directlyput input / output port on root level of model
  - configure in *Data Import/Export* tab of *Model Configuration Parameters*

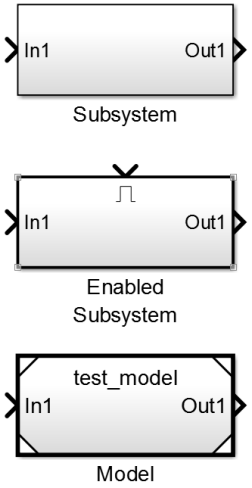
- log signals directly


  - right-click on signal, activate in *Logging and accessibility* tab in *Signal Properties*
  - configure in *Data Import/Export* tab of *Model Configuration Parameters*

additional possibilities for data import / export; Not to be used here!

# 6. Modeling in Simulink

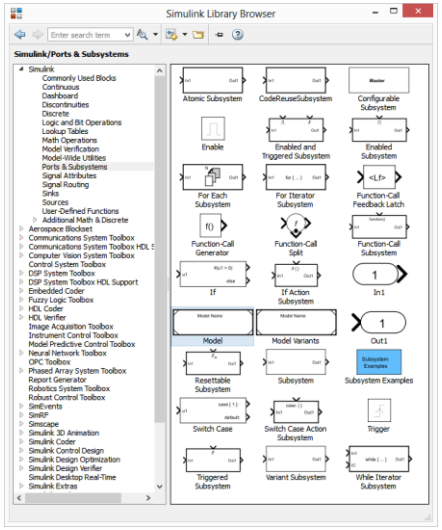
- the Working Principle of Simulink
- Blocks
- Signals
- **Subsystems**
- Parameters
- Model Configuration Parameters



# Modeling in Simulink
## Subsystems

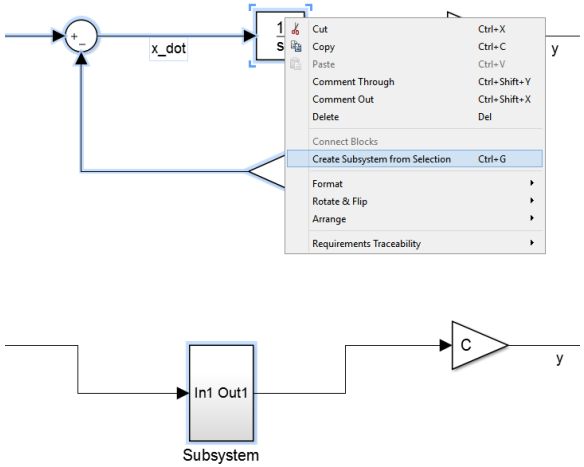
create a **subsystem** in one of the following ways:

### add from Library Browser



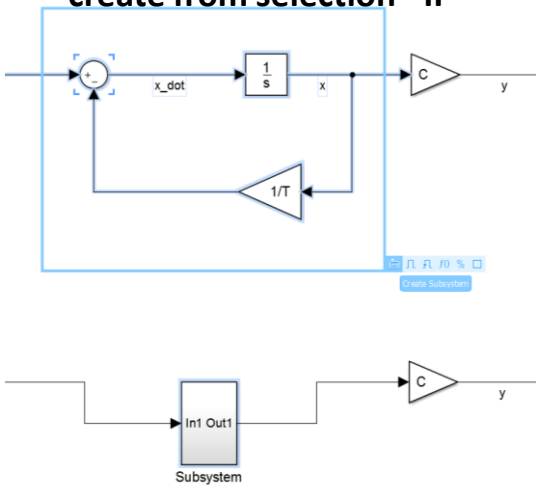

- go to *Simulink->Ports & Subsystems*
- choose the subsystem of your liking
- the subsystem will be empty

### create from selection - I




- mark some existing blocks
- right-click and choose *Create Subsystem from Selection*
- the subsystem will contain the marked blocks

### create from selection - II




- mark some existing blocks
- use menu at bottom right corner of marked area
- the subsystem will contain the marked blocks

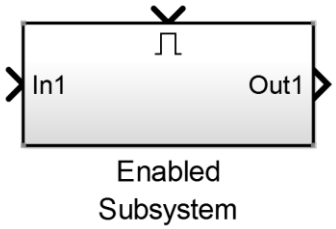
- subsystems are used to structure your model
- subsystems can be nested

- subsystems are (in general) virtual blocks, they do not perform any action on the signals they merely serve organizational purposes

# Modeling in Simulink
## Subsystems

special subsystems:

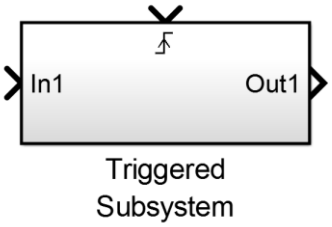
**enabled subsystem**





- create from library browser or by adding an *enable* block to an existing subsystem
- only executes when control signal at top is true
- behavior of states within subsystem (i.e. subsystem conatins a state-defining block like an integrator) can be configured via block parameters of *enable* block
  - **held:** states are held and continue at last value
  - **reset:** every time the system is enabled, the states assume their initial conditions

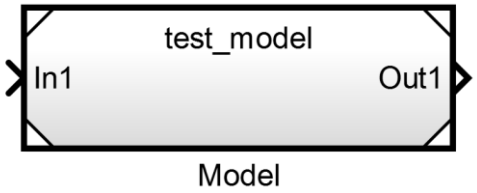
**triggered subsystem**





- create from library browser or by adding a *trigger* block to an existing subsystem
- only executes when trigger event occurs, i.e. rising and/or falling edge of the control signal
- triggering event can be configured via block parameters of *trigger* block
- always holds its outputs and states at last value between triggering events

**model reference subsystem**

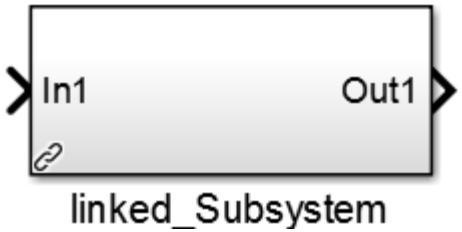



- create from library browser
- contains reference to a second Simulink model file (here `test_model.slx`)
- inputs and outputs are root-level inports and outports of the referenced system
- double-clicking opens the referenced model
- model reference can be changed by right-clicking -> Block Parameters (ModelReference)
- helps modular design, since models can be quickly interchanged
- for root level **Bus inports / Bus outports** the corresponding **Bus Objects** have to be defined

# Modeling in Simulink
## Subsystems

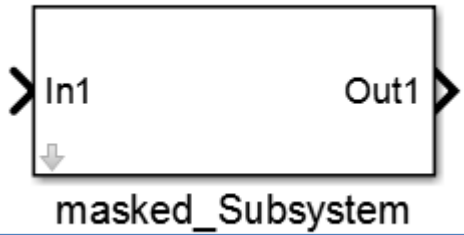
subsystems properties:

**linked subsystem**




- indicated by **chain symbol** in lower left corner
  *to active display of linked blocks use Menubar -> Display -> Library Links -> All*
- only a **reference** to a block, which is stored within a library file
- link has to be actively broken in order to make changes within the linked subsystem

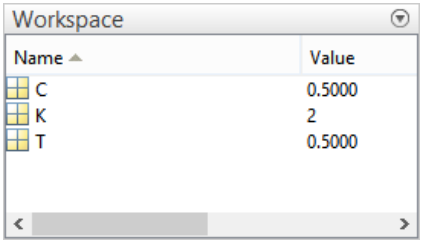
**masked subsystem**

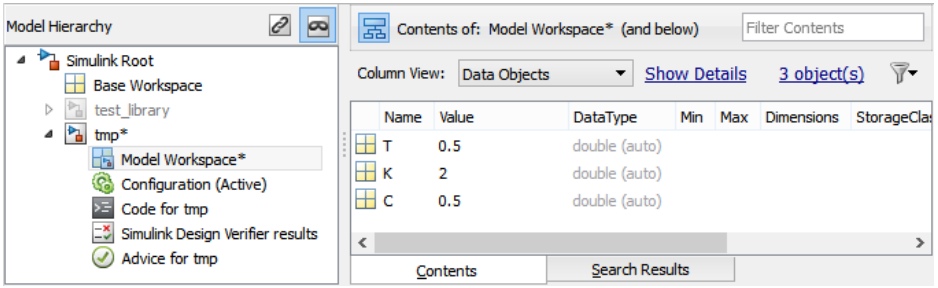



- implementation details are hidden behind a **mask**
- double-clicking on the subsystems does not open the block parameters dialog, but a user-defined **mask** dialog
- indicated by arrow in lower left corner
- to look under the mask, click on arrow or
  right click on system and choose *Mask -> Look Under Mask*

# 6. Modeling in Simulink

- the Working Principle of Simulink
- Blocks
- Signals
- Subsystems
- **Parameters**
- Model Configuration Parameters



# Modeling in Simulink
## Parameters

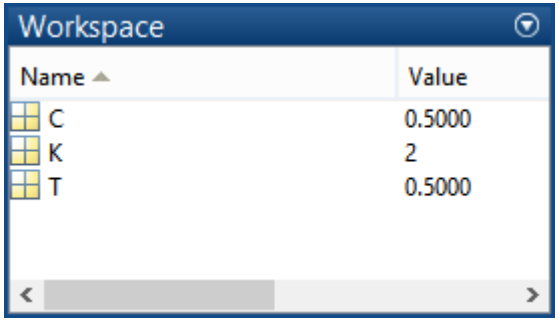
- Block Parameters don't have to be numeric values
- Instead they can be variable names; those variables have to be defined somewhere

**Possibility I: define variables in the base Workspace**

- use command line

```
>> K = 2; T = 0.5; C = 0.5;
```

store data in Matlab base Workspace




- or initialization script


```
1  % init model
2  K = 2;
3  T = 0.5;
4  C = 0.5;
```

store data in Matlab base Workspace

Pro:
- very flexible, many different Matlab objects can be used (structs, timeseries, matrices, …)

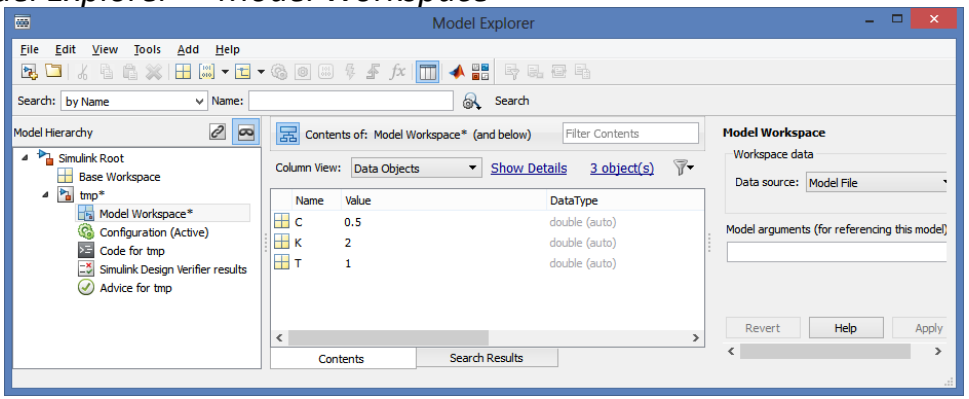
Con:
- every time the model is opened, the variables have to be initialized

# Modeling in Simulink
## Parameters

- Block Parameters don't have to be numeric values
- Instead they can be variable names; those variables have to be defined somewhere

**Possibility II: define variables in the Model Workspace**

- open base Workspace in Model Explorer via *View -> Model Explorer -> Model Workspace*
- create new variable via *Add -> Matlab Variable*
  Configure name and value
- Alternatively drag&drop existing variables from Base Workspace (higher up in the tree on the left)




Pro:
- data is stored directly with model, no external scripts for initialization is necessary

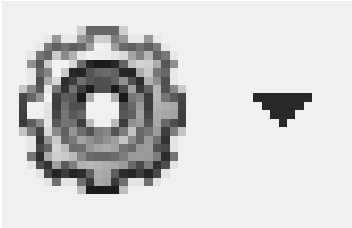
Con:
- creating variables in Model Explorer can be tedious
- Advanced data-types (structs, timeseries etc.) are hard to create

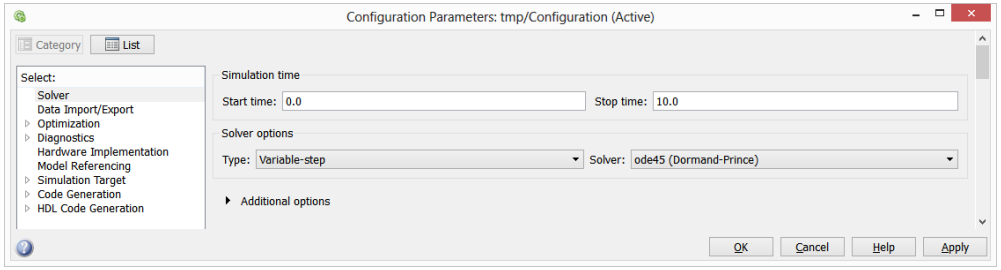
**Variables in the Model Workspace always are always preferred over Variables in the Base Workspace!**

*If two variables with the same name exist in Model and Base Workspace, the value of the variable in the Model Workspace will be used during simulation!*

# 6. Modeling in Simulink

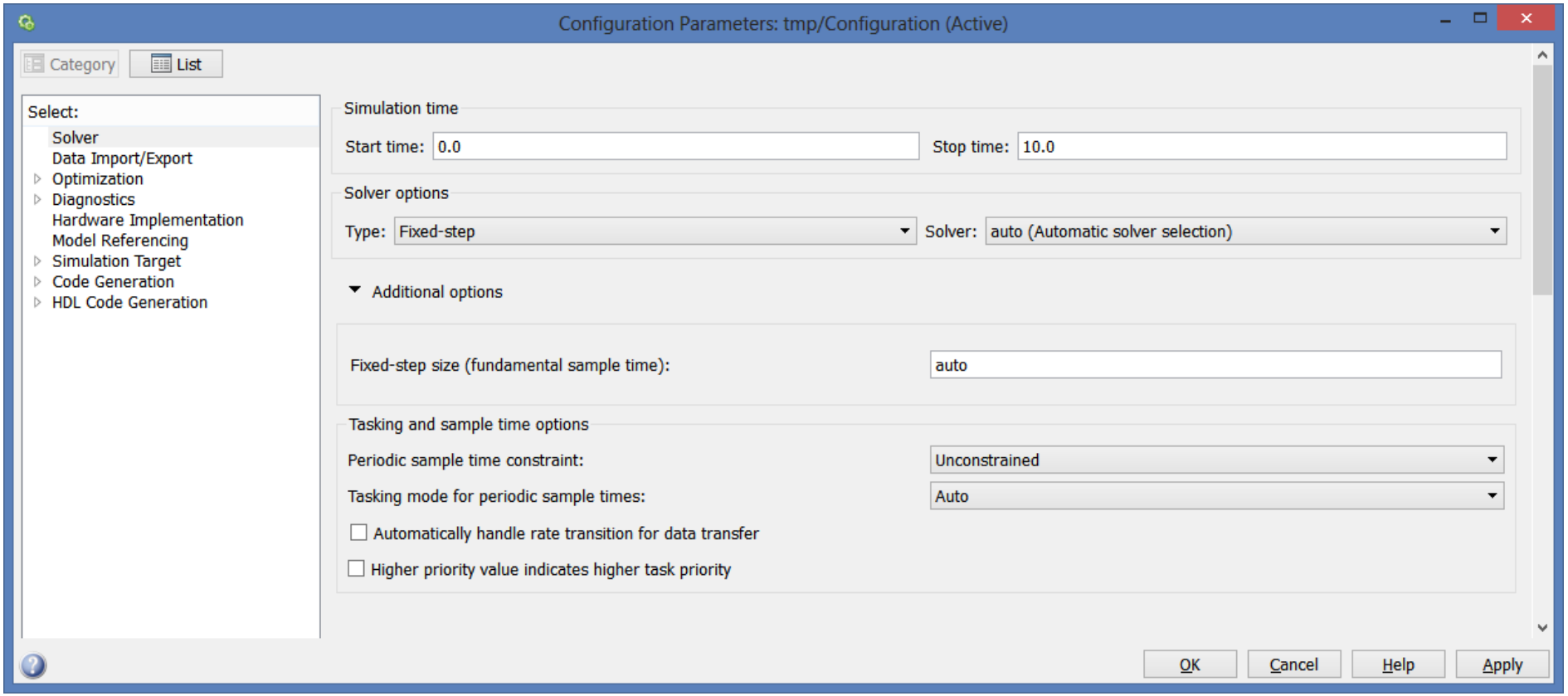
- the Working Principle of Simulink
- Blocks
- Signals
- Subsystems
- Parameters
- **Model Configuration Parameters**





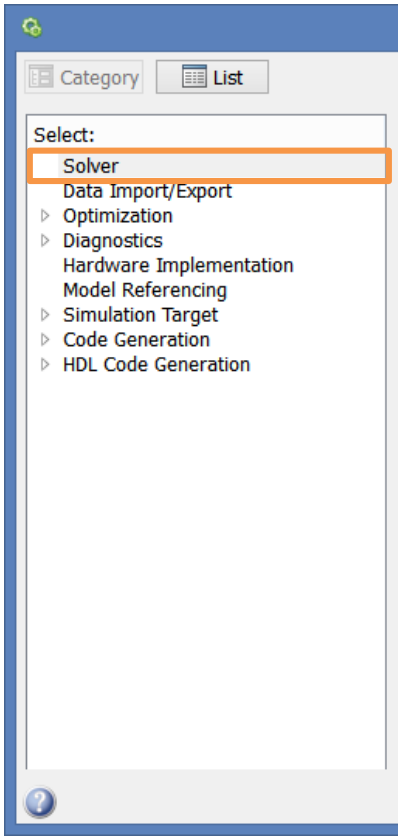
# Modeling in Simulink
## Model Configuration Parameters

clicking the *gear*  symbol or choosing *Menu Bar -> Simulation -> Model Configuration Parameters* opens the Configuration Parameters window



# Modeling in Simulink
# Model Configuration Parameters

The **Solver** Pane:

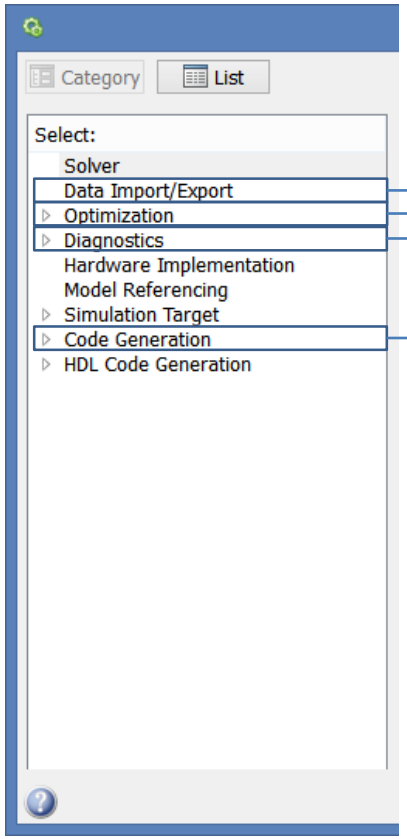



Most important pane in this course:

- choose solver type (variable-step, fixed-step)
- choose solver (ode3, ode4, ode5, ode45, ode23, …)
- for variable-step solvers
  - set Min / Max step sizes
  - set initial step size
- for fixed-step solvers
  - set fixed step size
- set simulation start / stop time

# Modeling in Simulink
# Model Configuration Parameters

## The other Panes




configure logging of signals (define logging data-type, behavior of root level inputs / outputs, …

optimization options for simulation and code generation (focus on faster buils / faster runs,…)

Diagnostic settings; define which event is to be treated as warning, error or ignored;
*For example: are unconnected signals ignored, is only a warning issued, or is it treated as error*

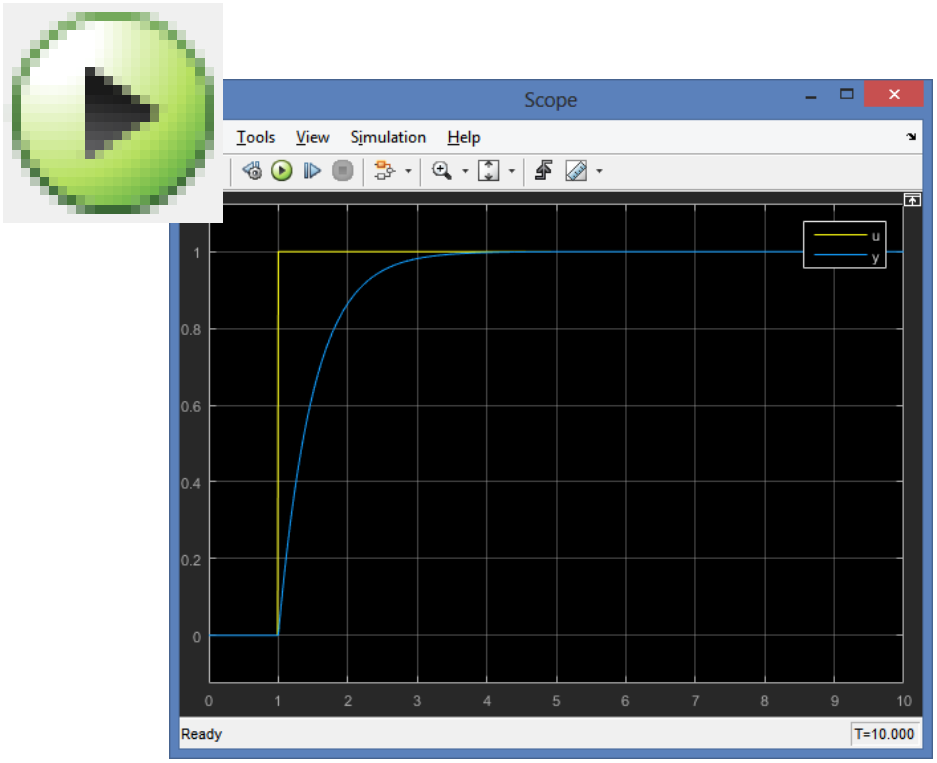
If the model is used for C-Code Generation, this can be configured here (target processor, compiler toolchain, interfaces, custom code inclusion, …)

# Modeling in Simulink

useful shortcuts:

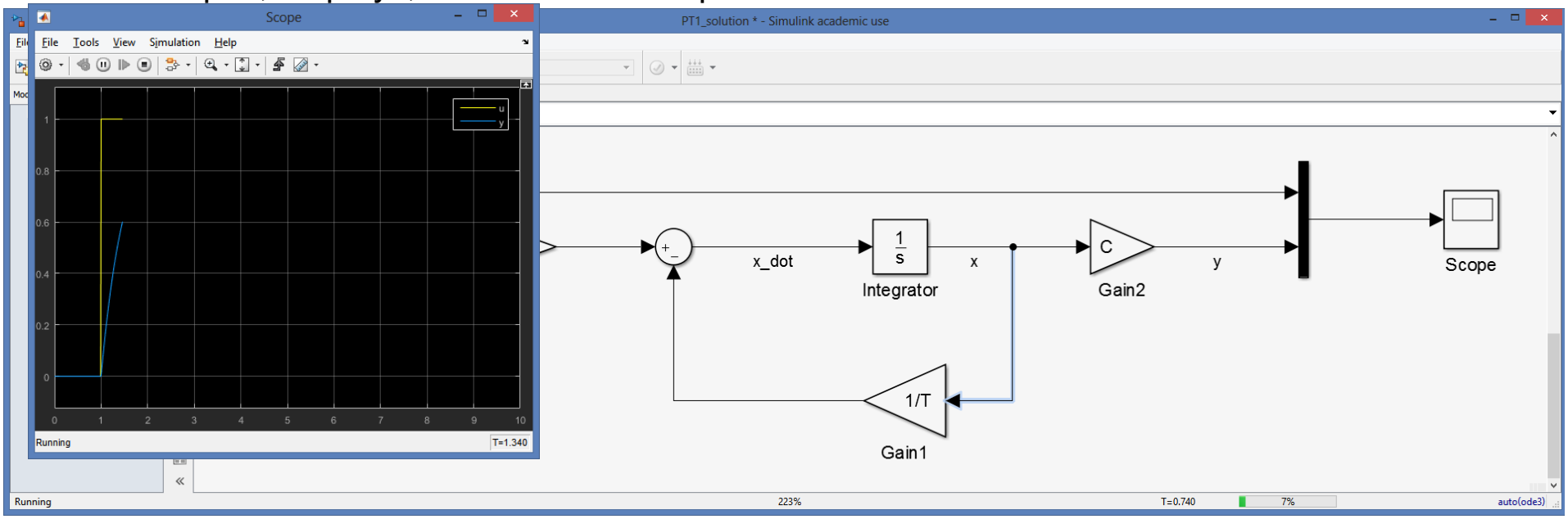
| | |
|---|---|
| **Single click on block / signal** | mark block / signal |
| **Click and hold around several elements** | mark multiple elements |
| **Shift + single click on block / signal** | Add to / remove from selection |
| **Shift + drag marked block** | disconnect block from signals; move block only |
| **Drag & drop marked block / signal** | move elements around |
| **double-click on subsystem** | enter subsystem |
| **Double-click on signal / block name** | rename signal / block name |
| **Ctrl + mark second block** | quick connect blocks |
| **click and hold right mouse button on signal** | branch signal |

# 7. Simulation



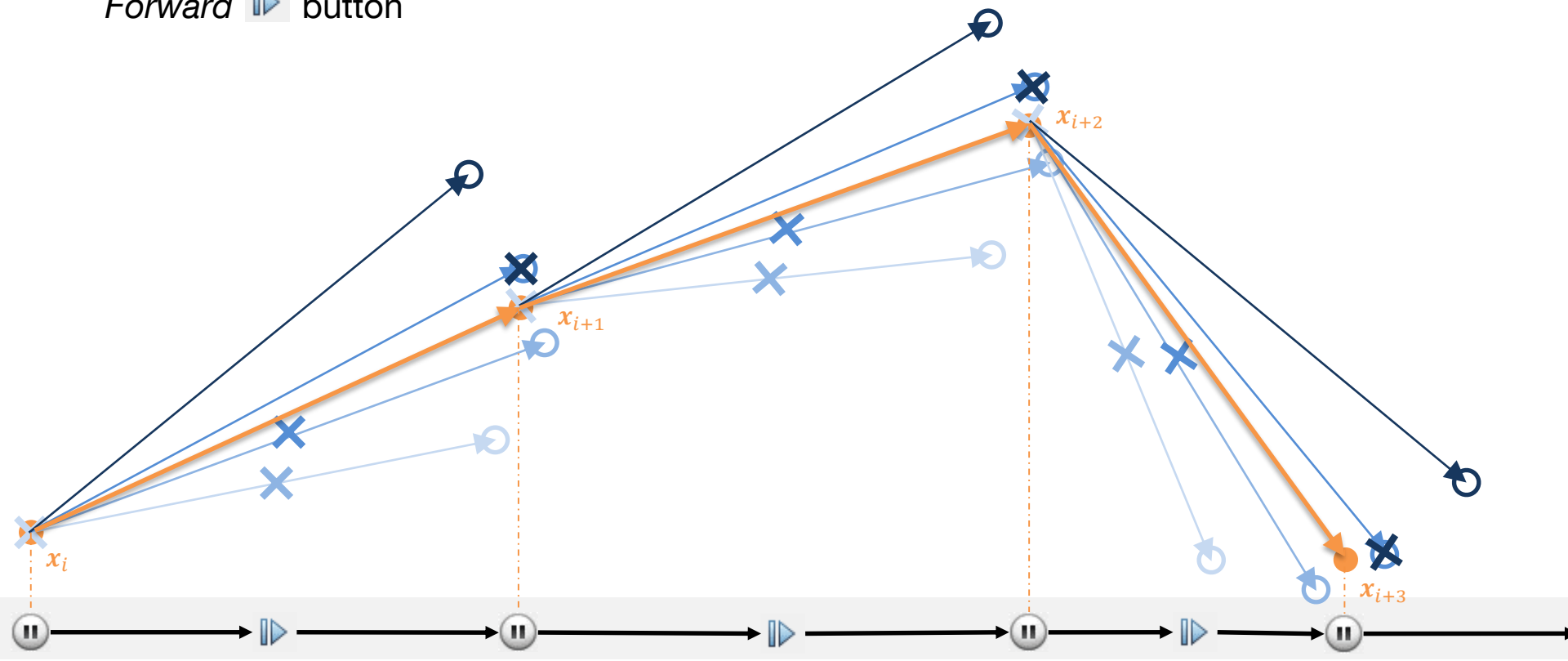
# Simulation

- Once the modeling process is finished, the model can be **simulated.** To start the simulation press the *play* button, or choose *Menu Bar -> Simulation -> Run*
- The editor will turn gray (no structural model changes are possible during simulation)
  ➔ some blocks have *tunable* parameters, which can be changed during Simulatin (e.g. *Gain*)
- In the notification bar, the message *'Running'* is displayed, along with the current simulation time
- all scopes, displays, value labels are updated online



# Simulation

- Pause the simulation with the *Pause* button, or by setting a *conditional break point* (right-click on signal, set break point and define condition)
- While the simulation is paused, single integration steps can be performed with the *Step Forward* button



# Simulation

When hitting the *Run* button, the Simulink engine goes through the following phases

1. **Model Compilation**: translate the model to an executable form
   - evaluate all block parameters to determine their values (i.e. check if the expressions are valid / the variables exist)
   - determine signal attributes (data type, name, dimensionality etc)
   - attribute propagation (determine attributes, which are not specified)
   - block operations (block reduction, model flattening, …)
   - determine sample times (specified and unspecified)
2. **Link Phase:** link model
   - allocate memory (for signals, states, run-time parameters, block data structures)
   - create method execution list (block methods to create output)
3. **Simulation Loop Phase:** successively compute states and outputs
   - Loop initialization (determine initial states / outputs)
   - Loop iteration:
     1. compute outputs
     2. compute states (depending on solver)
     3. check for discontinuities
     4. compute time for next time-step
     5. repeat

Model Compilation can also be initiated by *Simulation -> Update Diagram* or using Ctrl+D. USE THIS FREQUENTLY TO CHECK FOR SYNTHACTICAL ERRORS IN YOUR MODEL!!!!

# Simulation

## Algebraic Loops

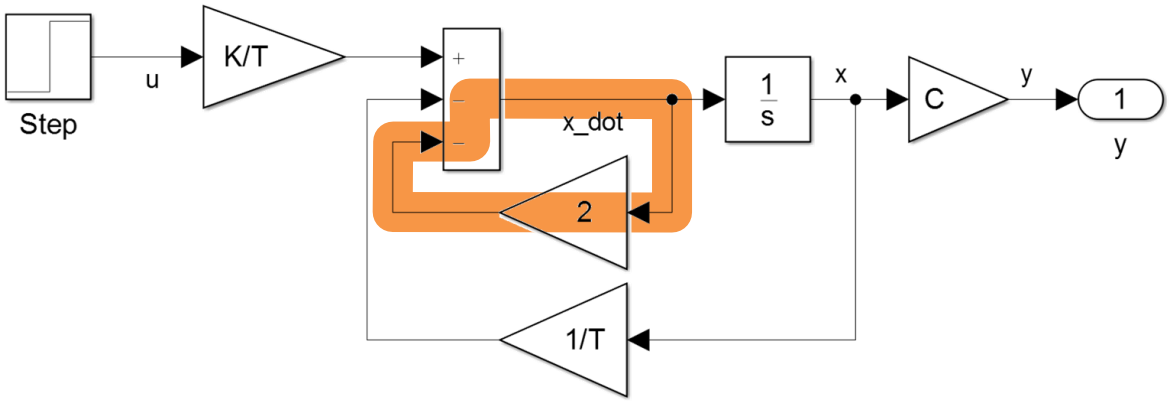
- results if current value of derivative is dependent on derivative itself

$$\dot{x}_i = \tilde{f}_i(\boldsymbol{x}, \dot{x}_i, \boldsymbol{u})$$
$$\Leftrightarrow 0 = f_i(\boldsymbol{x}, \dot{x}_i, \boldsymbol{u}) \quad \text{for some } i$$

  - ➔ differential algebraic equation (DAE)
  - ➔ instead of evaluating $\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u})$ directly, a system of non-linear equations has to be solved to obtain $\dot{x}_i$ (e.g. for `ode4` this has to be done 4 times!)
- very slow compared to standard ODE
- can be numerically unstable, i.e. no solution is possible
- code generation fails for systems with algebraic loops

Example:

$$\dot{x} = -2\dot{x} - \frac{1}{T}x + \frac{K}{T}u$$
$$y = Cx$$



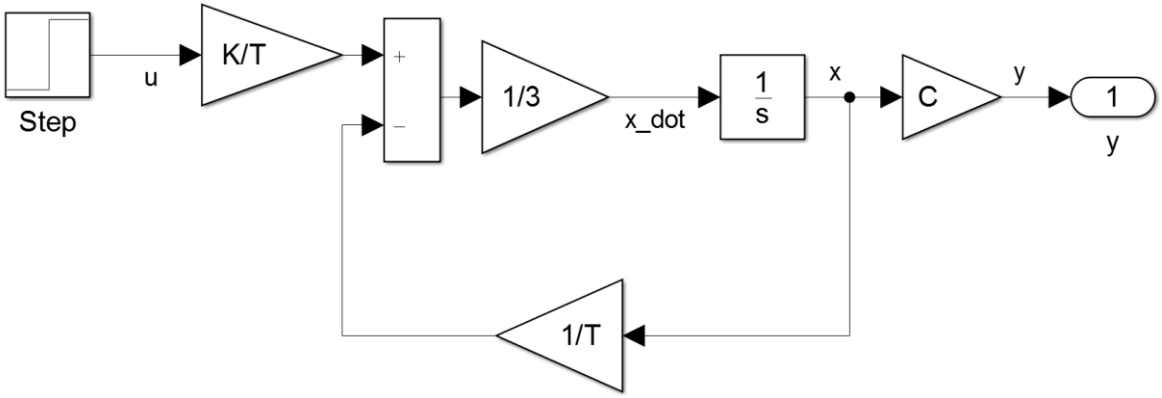
# Simulation

## Algebraic Loops - Remedies

### 1. reformulate system

Example:

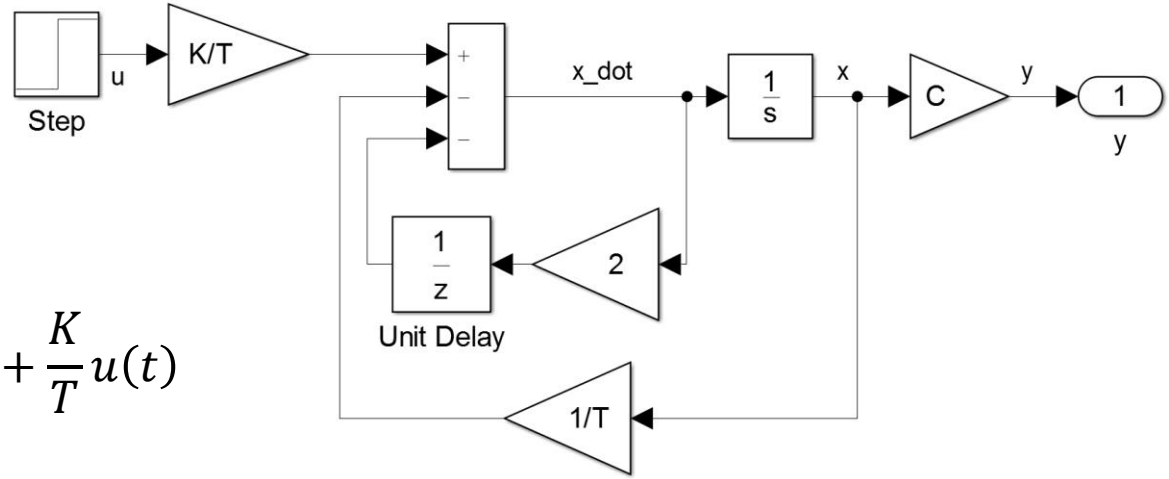
$$\dot{x} = \frac{1}{3}\left(-\frac{1}{T}x + \frac{K}{T}u\right)$$
$$y = Cx$$




### 2. introduce delay

Example:

$$\dot{x}(t) = -2\dot{x}(t-\Delta t) - \frac{1}{T}x(t) + \frac{K}{T}u(t)$$
$$y(t) = Cx(t)$$

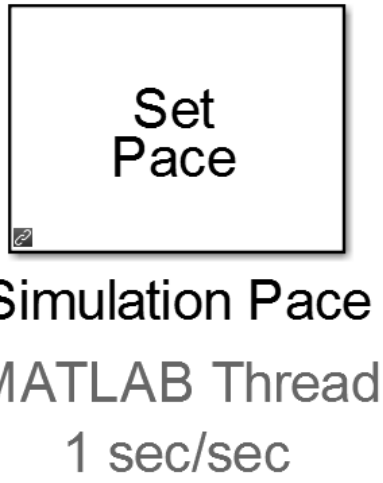


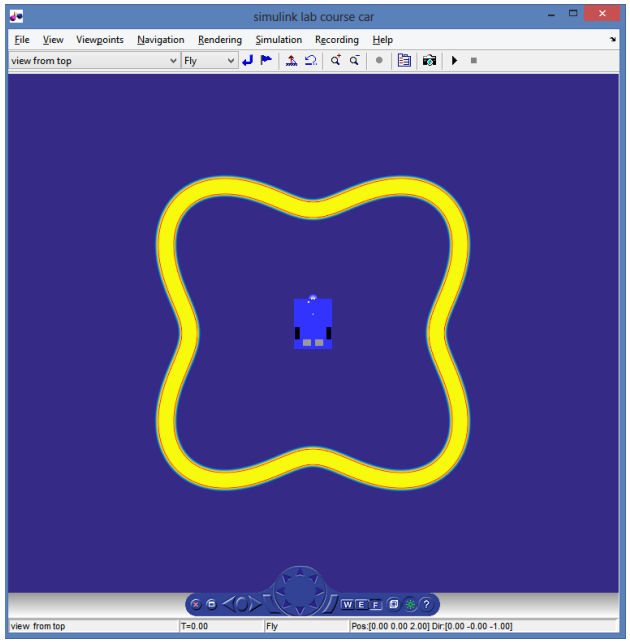
# Simulation

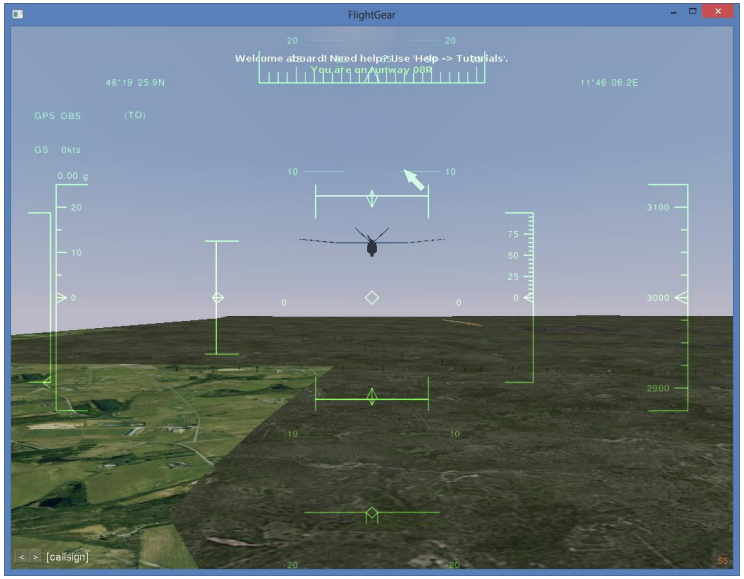
**(pseudo) Real-Time Simulation**

Instead of performing a simulation as fast as possible, sometimes it is desirable to see the result in real-time (simulation of real dynamic systems, e.g. line-following robot, flight simulator).

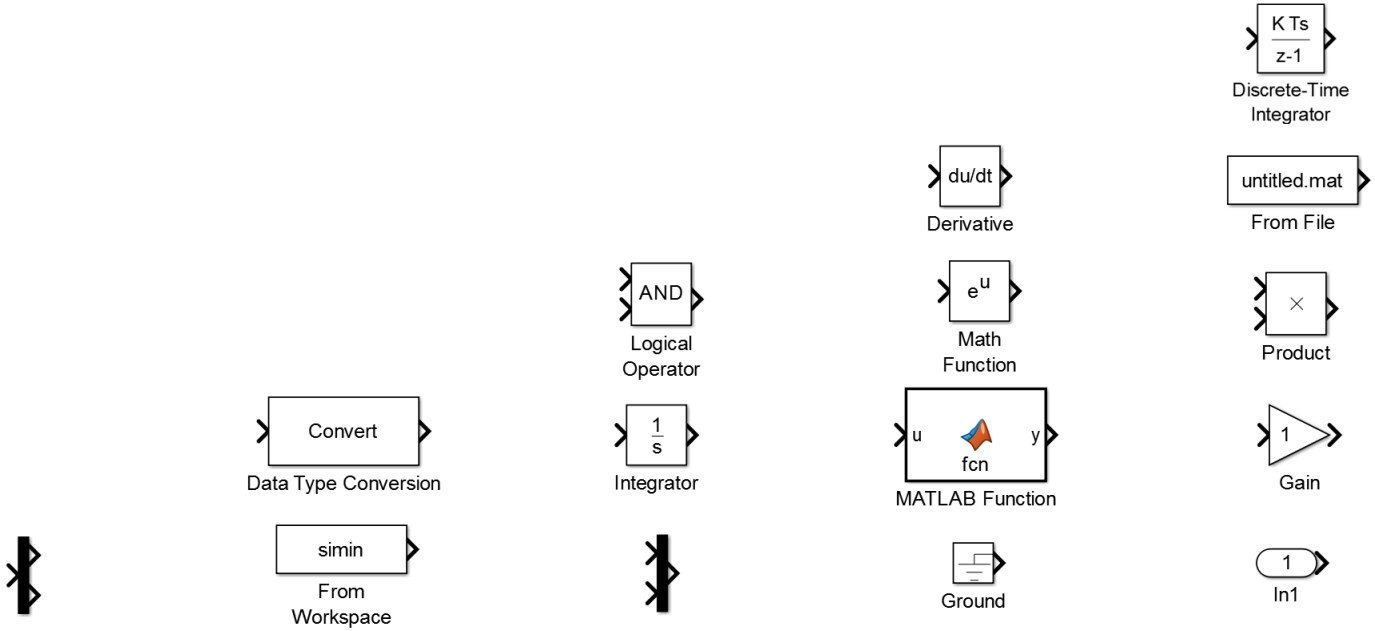
To achieve this, the *Simulation Pace* block can be added anywhere in the model. It slows down the simulation to (pseudo) real-time.



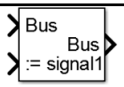
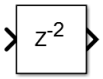



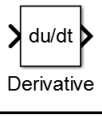
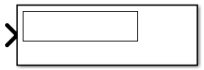
# 8. List of Useful Blocks and Descriptions



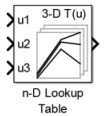
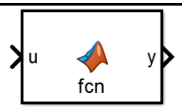
# List of Useful Blocks and Descriptions

| | |
|---|---|
|   | **Band limited white noise:** random number; used to model sensor noise |
|  | **Bus Creator:** creates signal busses |
|  | **Bus Selector:** selects signals from busses |
|   | **Bus Assignment:** assigns a new value to one signal in a bus |
|   | **Constant:** outputs constant value |
|   | **Data Type Conversion:** changes data-type of signal |
|   | **Delay:** delays signal by one time-step (for discrete time signals only) |
|  | **Demux:** splits array of signals |

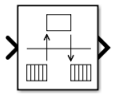
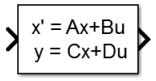
# List of Useful Blocks and Descriptions

| | |
|---|---|
|  Derivative | **Derivative:** approximate derivative of singal; should be avoided, since it can introduce numerical problem! |
|  Discrete-Time Integrator | **Discrete Time Integrator:** integrator for controller implementation in discrete time systems |
|  Display | **Display**: display value of signal during simulation (scalar, vector, matrix signals) |
|  From File | **From File:** load signal from file |
|  From Workspace | **From Workspace:** load signal from workspace |
|  Gain | **Gain:** scales signal with gain value |
|  Ground | **Ground:** avoid “unconnected input port” warning by using ground block; outputs signal with same data-type as inport to which it is connected |
|  In1 | **In:** input to subsystem / model (on root level) |

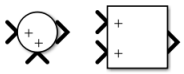
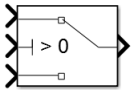
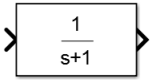
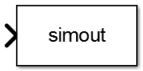
# List of Useful Blocks and Descriptions

| | |
|---|---|
|  | **Integrator:** continuous time integrator for modelling dynamics |
|  | **Logical Operator:** perform logical operations on signals |
|  | **n-D Lookup Table:** use tabulated data and interpolation algorithms |
|  | **Math Function:** exp(), log, square, pow(), reciprocal, transpose, … |
|  | **MATLAB Function:** implement custom behavior in MATLAB syntax |
|  | **Mux:** make array of signals |
|  | **Out:** output of subsystem / model (on root level) |
|  | **Product:** product of signals; can be configured to divide, too. |

# List of Useful Blocks and Descriptions

| | |
|---|---|
|   | **Rate Transition:** change sample time of signal (up-sampling: zero-order hold; down-sampling: use latest value) |
|   | **Relational Operator:** perform relational operations between signals |
|   | **Saturation:** limit signal to defined range |
|   | **Scope:** display time-series of data for simulation output |
|   | **Signal Builder:** build your own signal patterns via GUI |
|   | **Sine Wave:** sine wave input |
|   | **Sqrt:** compute square root of a signal |
|   | **State-Space:** define linear state space model |

# List of Useful Blocks and Descriptions

| | |
|---|---|
|  | **Step:** step input |
|  | **Sum / Add:** add signals; can be configured to subtract, too. |
|  | **Switch:** switch between two signals (of same data type) using a control signal |
|  | **Terminator:** counterpart of *Ground*. Use this to terminate unused signals and avoid “unconnected outport” warning |
|  | **Transfer Fcn:** implement transfer function for signal |
|  | **To File:** write signal data directly to file on harddisc |
|  | **To Workspace:** write signal data to variable in workspace |
|  | **Trigonometric Function:** sin, cos, tan, asin, acos, … |

# Further Reading

The following topics might be interesting for further reading (**non**-comprehensive overview):

| Topic | Description | Documentation article | Simulink User's Guide |
|---|---|---|---|
| **Data-Dictionary** | an additional source for simulation parameters, which extends the functionality of base / model workspaces | **Simulink Data Dictionary** | Chapter 70 |
| **Simulation Data Inspector** | a tool for quick examination / comparison of logged signals, without having to write plot routines in MATLAB | **Inspecting and Comparing Simulation Data** | Chapter 30 |
| **Model Explorer** | manage complete model (configurations, workspaces, signals, etc.) | **Exploring, Searching, and Browsing Models** | Chapter 13 |
| **Masks** | creating / working with 'masked' systems; designing your own masks | **Working with Block Masks** | Chapter 40 |
| **Importing / Exporting Data** | import / export data for use with MATLAB or other programs | **Inspecting and Comparing Simulation Data** | Chapter 30 |
| **Simulink Debugger** | extended Debugging functionalities | **Simulink Debugger** | Chapter 35 |

# Excercises

- **Make sure to start Matlab 2019a 64bit!**
- Download the exercise material from moodle
  - Unzip „Templates.zip“ to obtain the templates for the exercises
  - Unzip „Testing_Students.zip“ to obtain the automatic testing routines for the exercises
  - **Add all folders and subfolders to the Matlab path!**

In order for the automatic checks to work properly:

- whenever naming something (signals, blocks, etc.) make sure to **NOT INCLUDE additional spaces**!
- make sure, that you **DO NOT hit ENTER after entering the name**, this will include a line-break, not confirm your entry!
- When entering variables (Gains, Integrators etc), **do NOT INCLUDE additional spaces and stick to the order** in the booklet!
- Make sure that when naming a signal, you really name the signal, and don’t add an annotation by double-clicking next to the signal-line.
- The automatic model checks only work on PCs, i.e. if you work with your own Mac, you will most likely encounter errors.