Università degli Studi di Padova

Introduction to Simulink

Riccardo Antonello
(riccardo.antonello@unipd.it)

Giulia Michieletto
(giulia.michieletto@unipd.it)

Dipartimento di Tecnica e Gestione dei Sistemi Industriali

Università degli Studi di Padova

4 Marzo 2024

Outline

- What is Simulink ?
- Starting Simulink.
- Simulink Block Libraries.
- Creation of a Simulink model.
- Simulation of a Simulink model:
  - Numerical solution of ODEs
  - ODE solvers
  - Simulation parameters
- Exporting simulation results to MATLAB.

What is Simulink ?

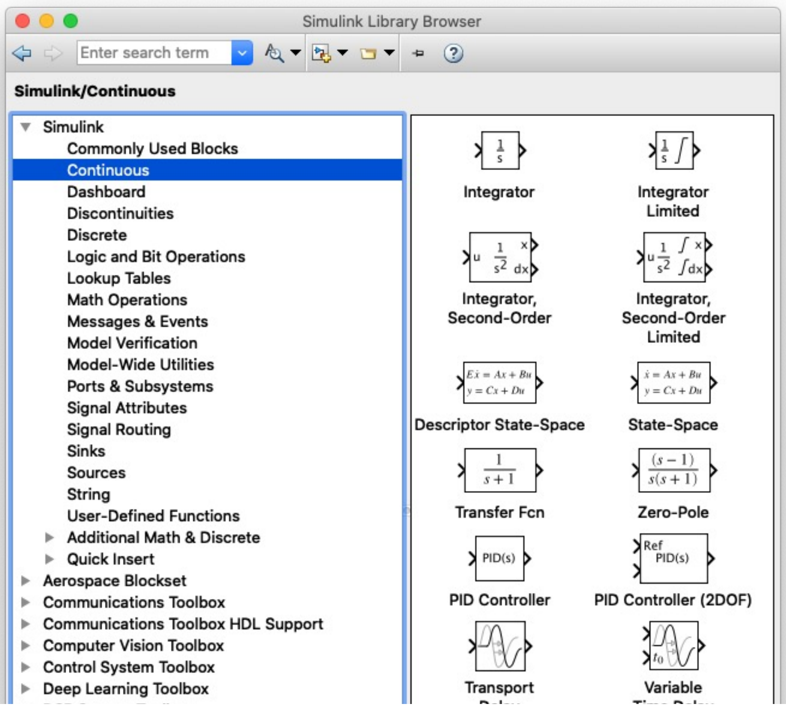
Simulink (*Simulation* + *Link*) is a *graphical environment* that allows to model, simulate and analyze dynamical systems represented in *block-diagram form*.

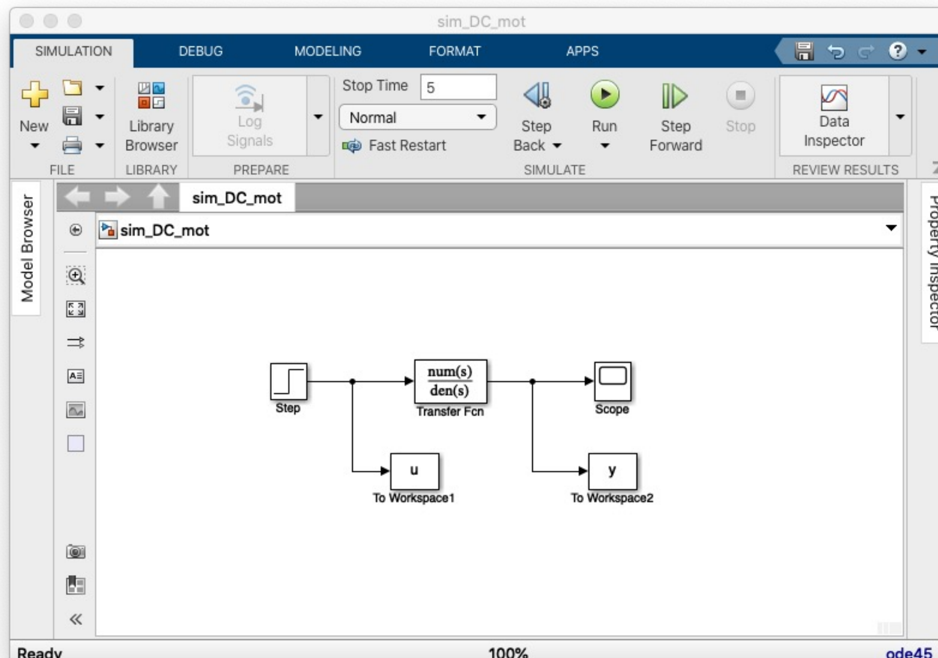
Differently from the *Control System Toolbox (CST)*, it allows to simulate generic dynamical systems, including *nonlinear*, *time-variant, multi-rate* and *hybrid-time (sampled-data)* systems.

What is Simulink ?

Simulink *models* (block diagrams) are created by using a *graphical user interface* (GUI).

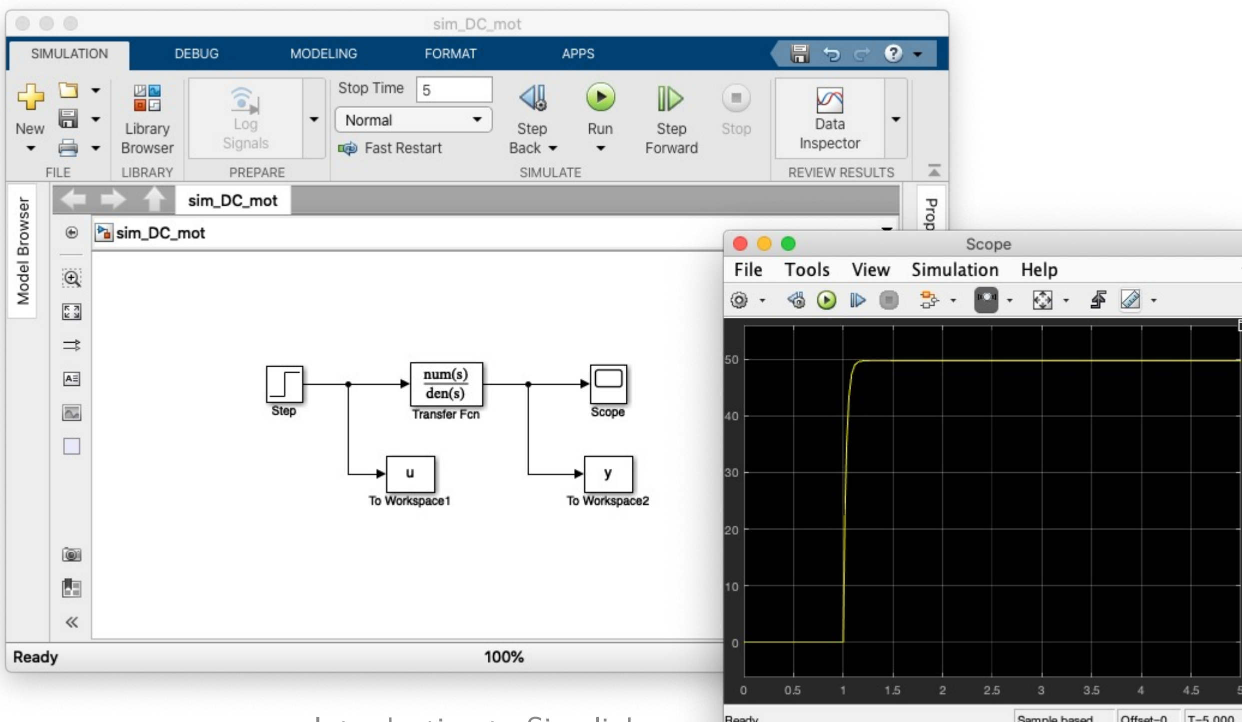
A rich library of built-in blocks is available; additional blocks can be created by the user.





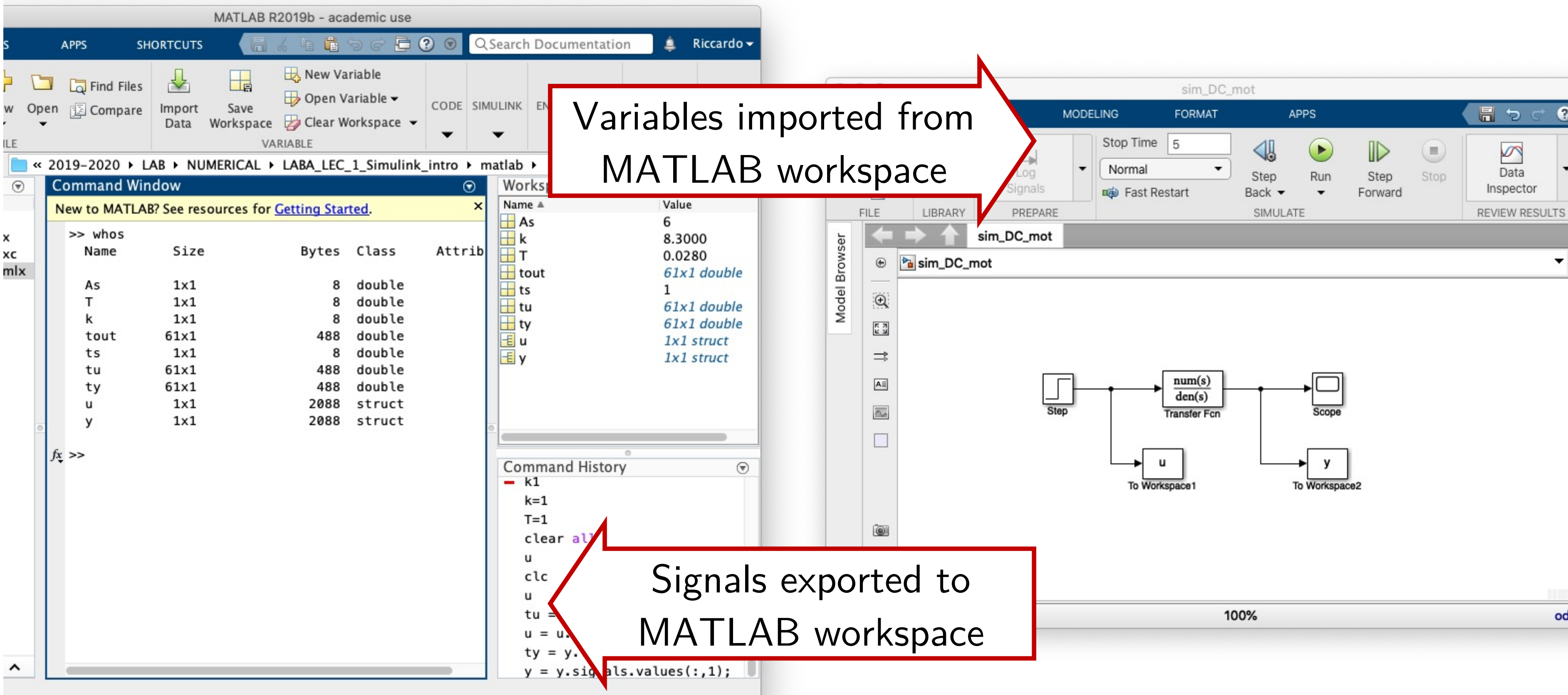
What is Simulink ?

Simulink models are simulated by numerically integrating the underlying ordinary differential equations (ODE) describing their dynamics, using the MATLAB *ode solvers suite*.



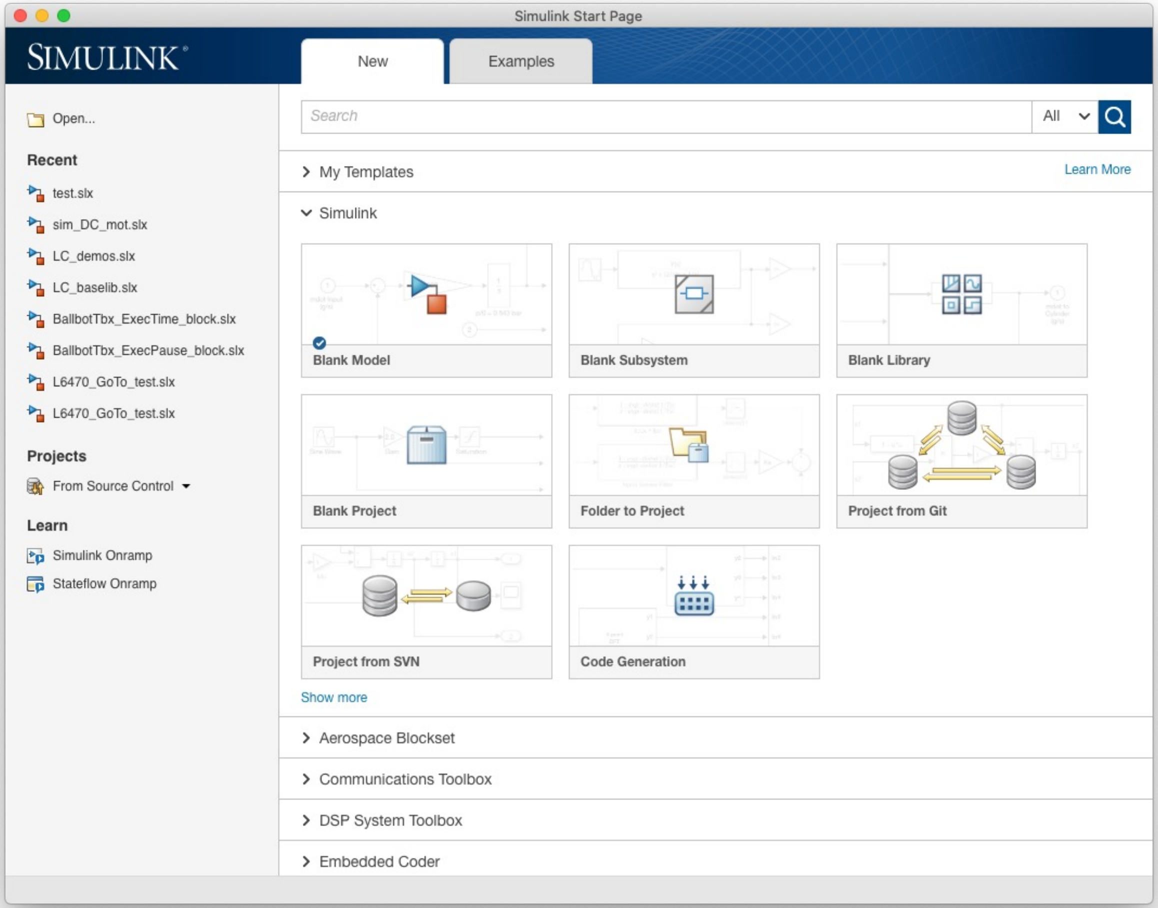
What is Simulink ?

Simulink is tightly integrated with MATLAB.



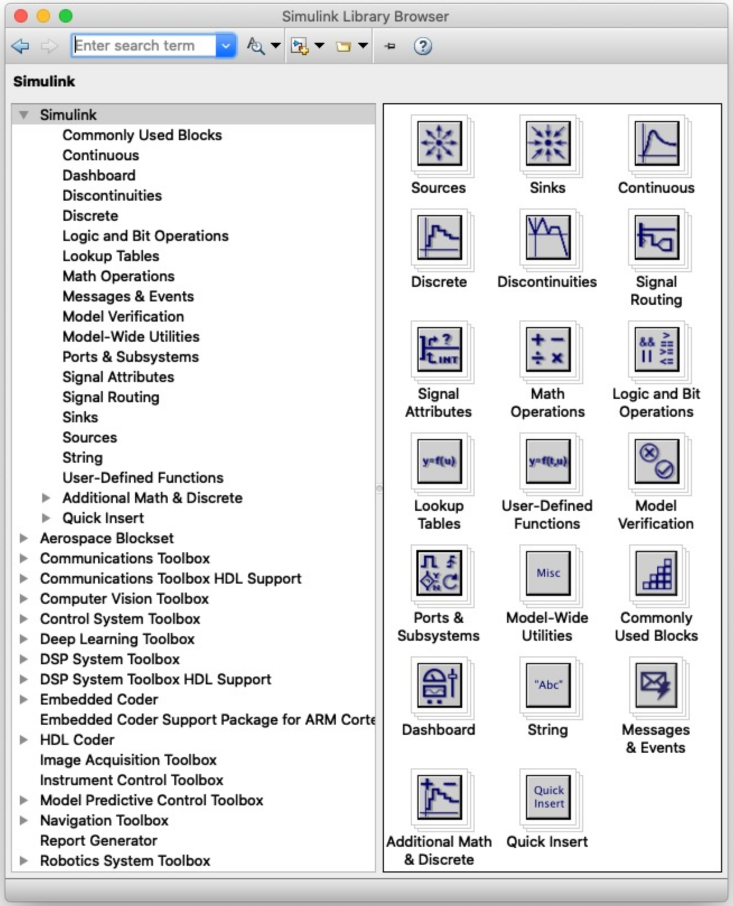
Starting Simulink

Run **`simulink`** from the MATLAB command window to open the *Simulink Start Page*.

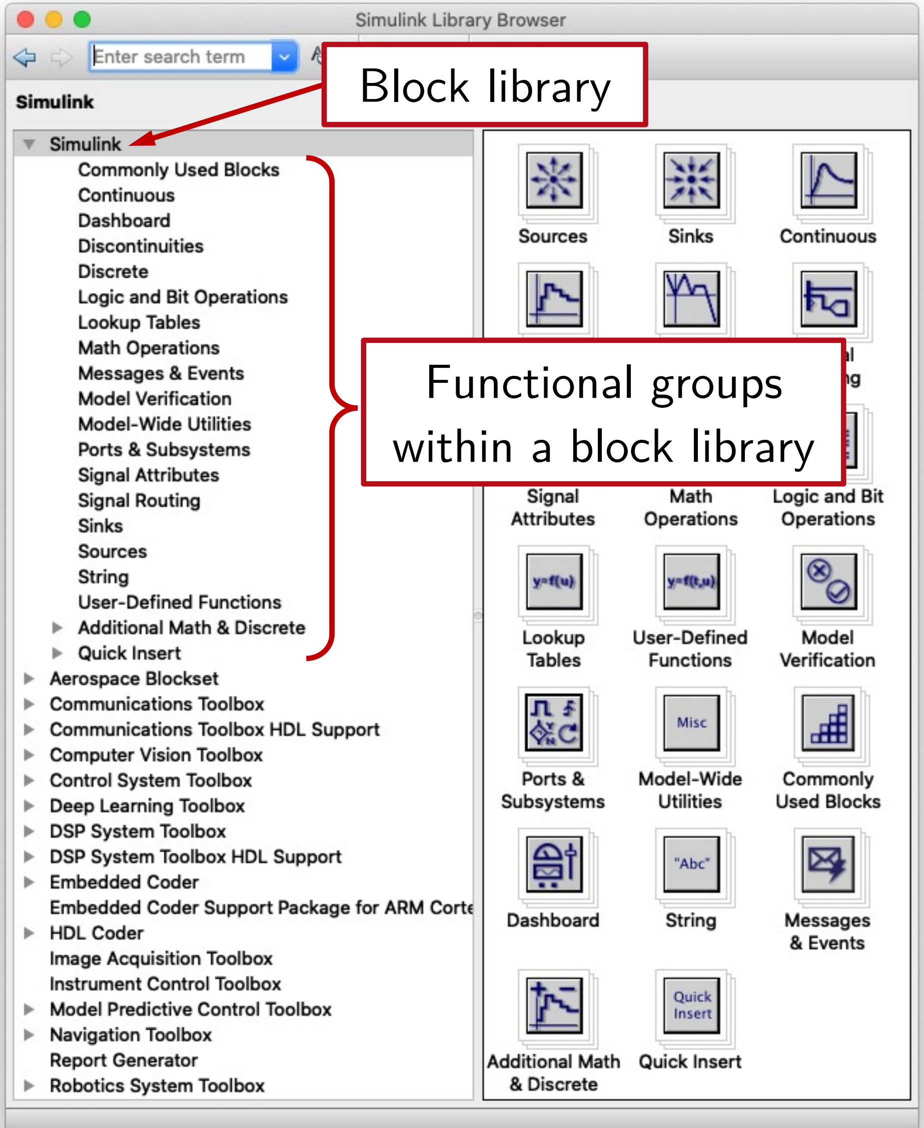


Starting Simulink

Shortcut: run **`slLibraryBrowser`** to directly access the *Simulink Library Browser*.



Simulink Block Libraries

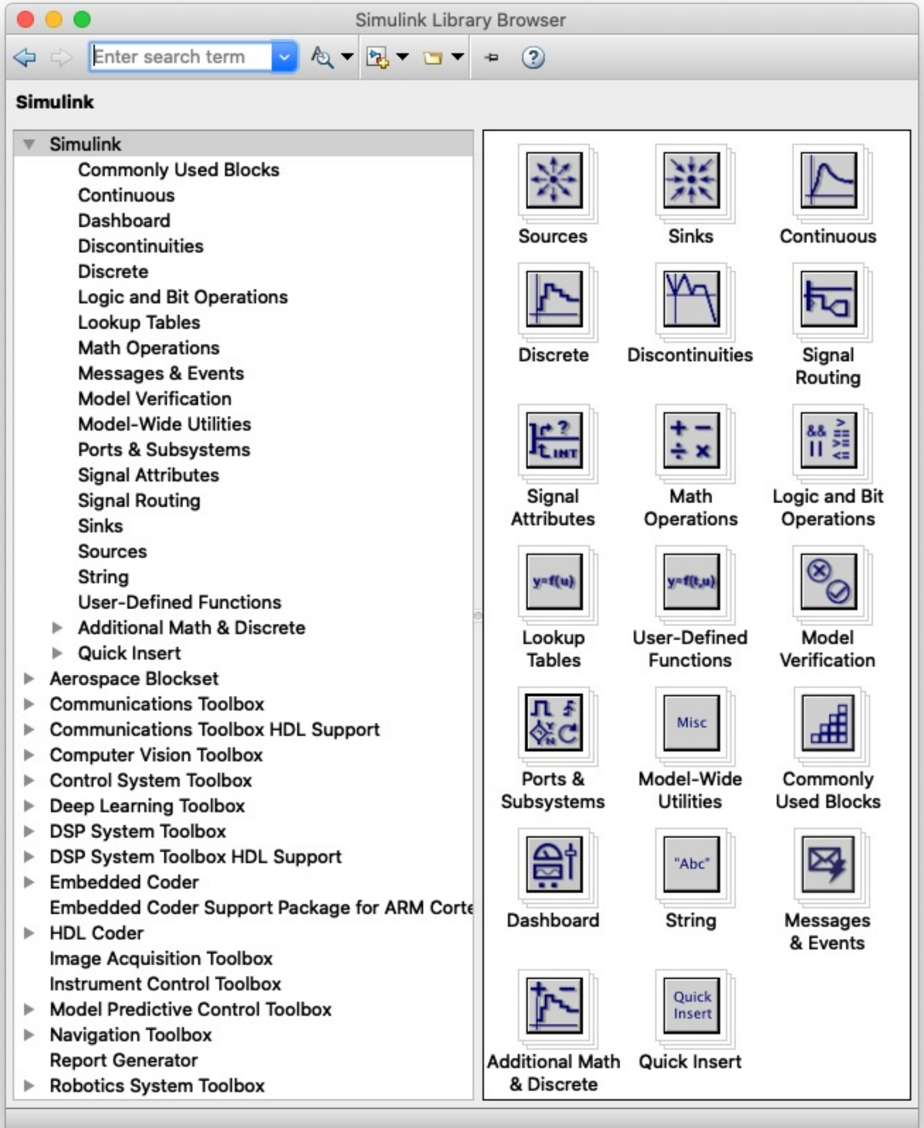


- Each toolbox provides a dedicated library.

  Simulink has its own library, with a set of base blocks.

- Blocks within a library are organized into functional groups.

Simulink Block Libraries



In the Simulink Block Library:

- ↳ **Sources**: generate input signals for other blocks.
- ↳ **Sinks**: blocks used to view signals or export data.
- ↳ **Math**: blocks implementing common math functions.
- ↳ **Continuous**: blocks for continuous-time LTI models.
- ↳ **Discrete**: blocks for discrete-time LTI models.

Creation of a Simulink model

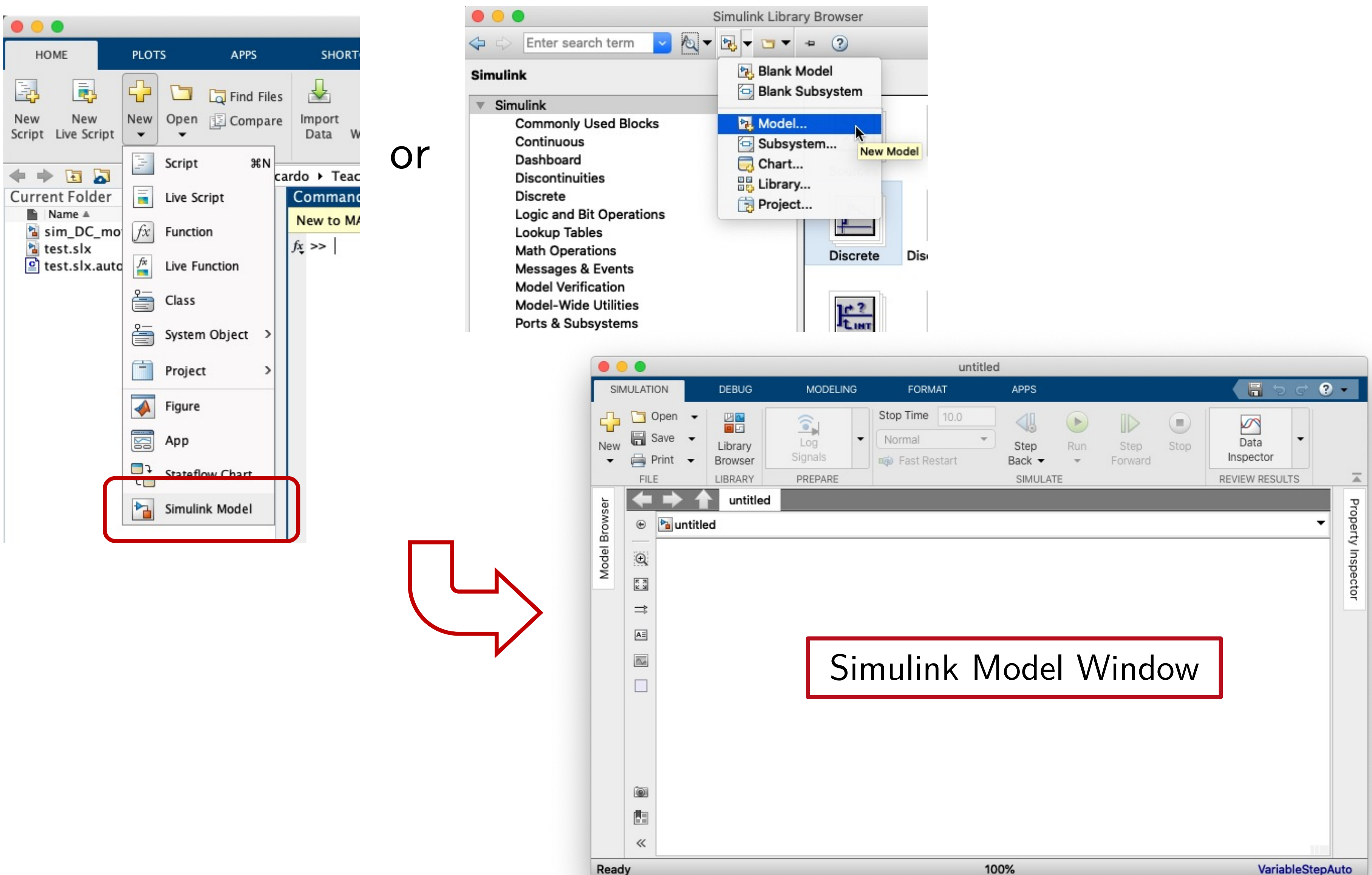
Example: consider the following simplified model of a DC motor

$$P(s) = \frac{Y(s)}{U(s)} = \frac{k}{T\,s+1}, \qquad k = 8.3, \quad T = 0.028$$

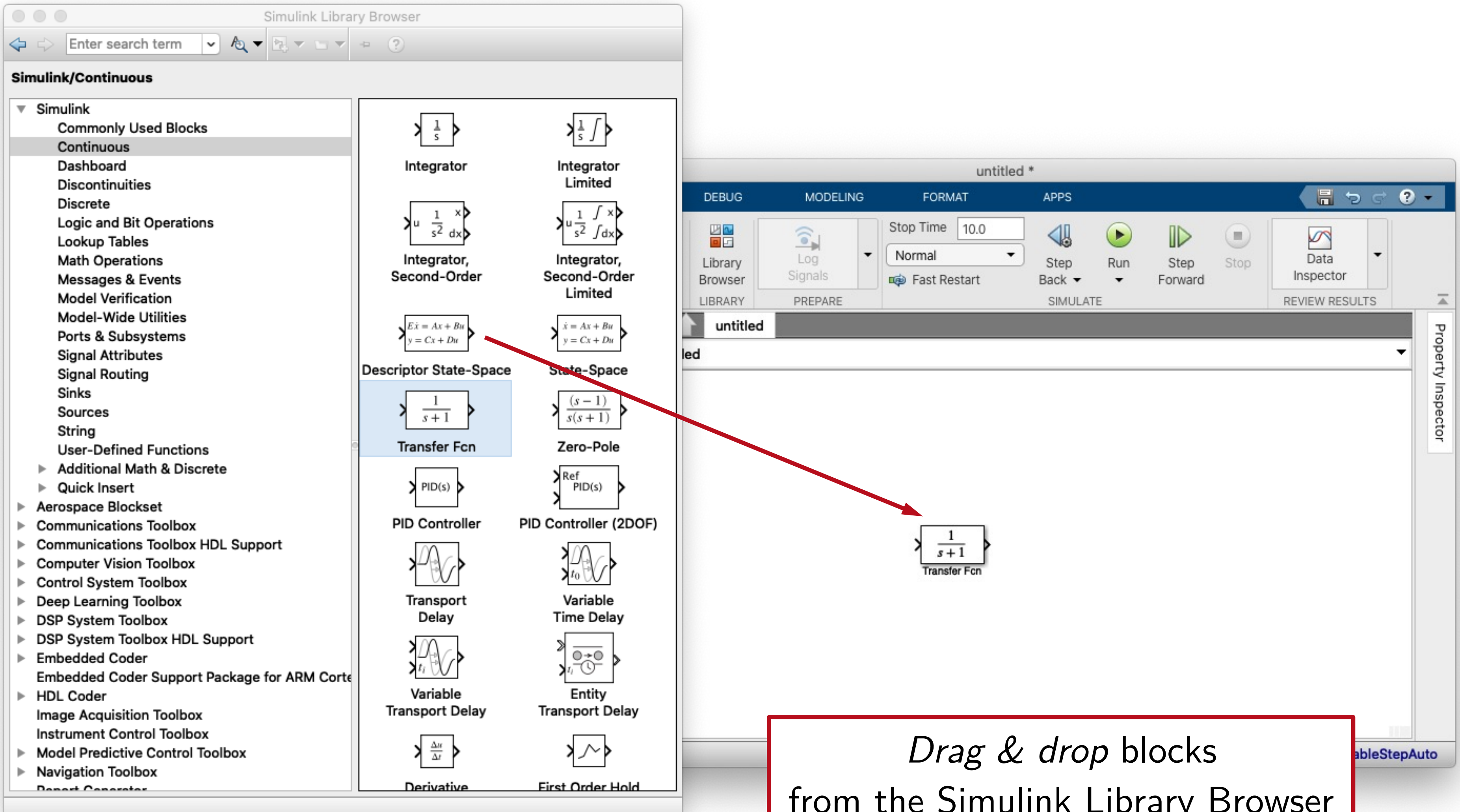
with the armature voltage u [V] as input, and the shaft speed y [rad/s] as output.

Evaluate the speed response on $[0s, 5s]$ to a 6 V step voltage input, applied at $t = 1\,s$.

Creation of a Simulink model

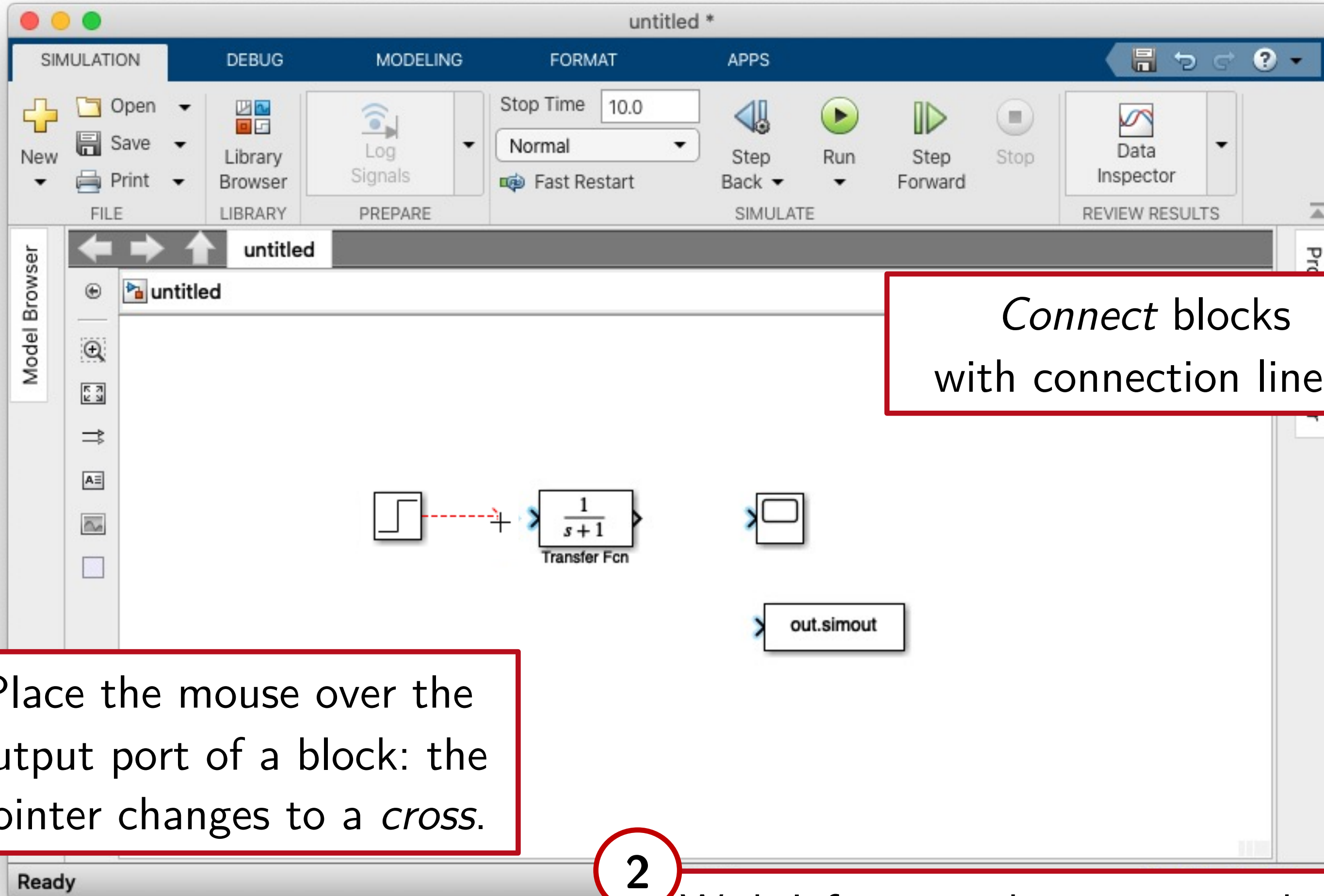


Creation of a Simulink model



Drag & drop blocks
from the Simulink Library Browser
to the Model Window.

Creation of a Simulink model

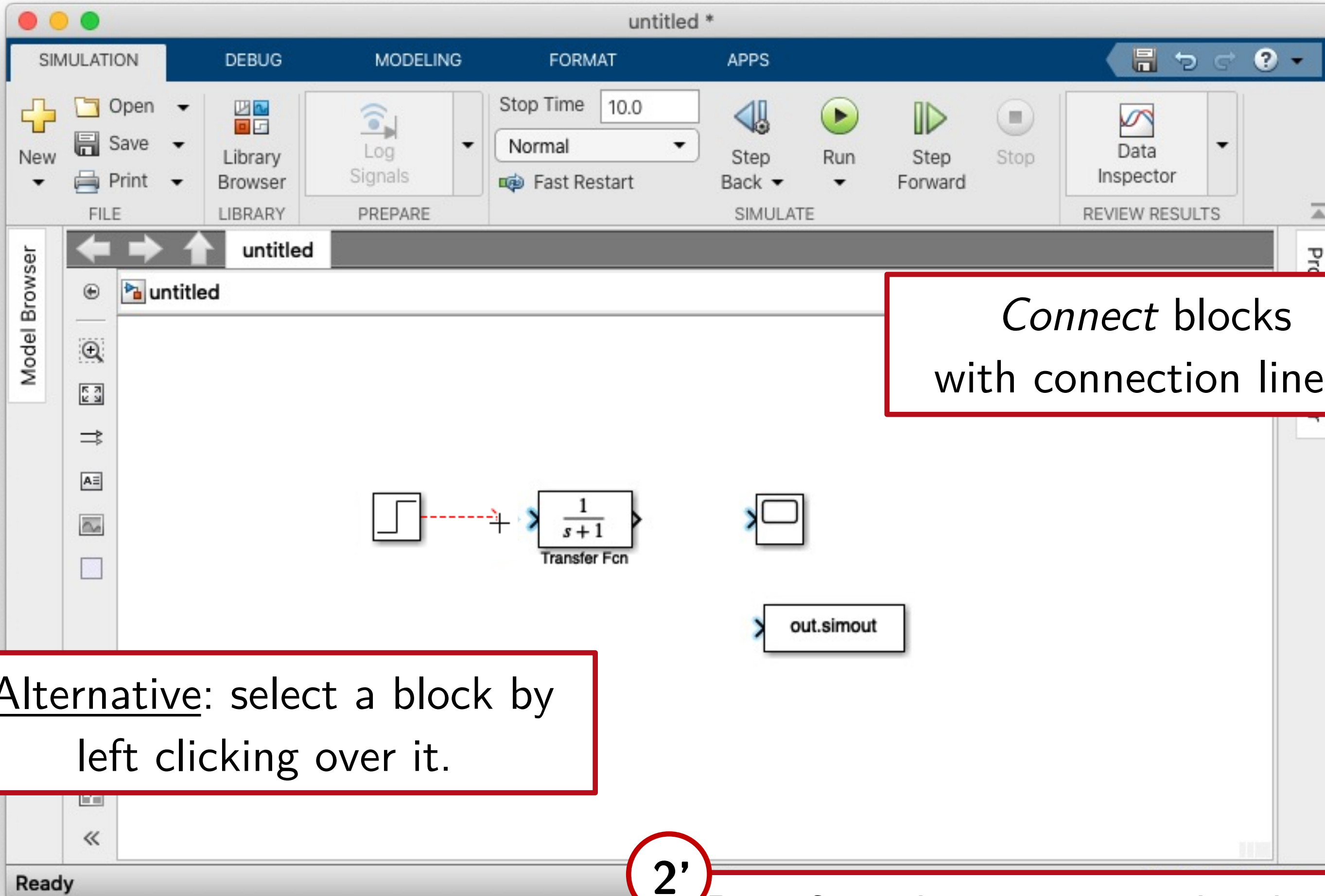


Connect blocks with connection lines.

1 Place the mouse over the output port of a block: the pointer changes to a *cross*.

2 With left mouse button pressed, move the pointer over the input terminal of another block, and release the button.

Creation of a Simulink model

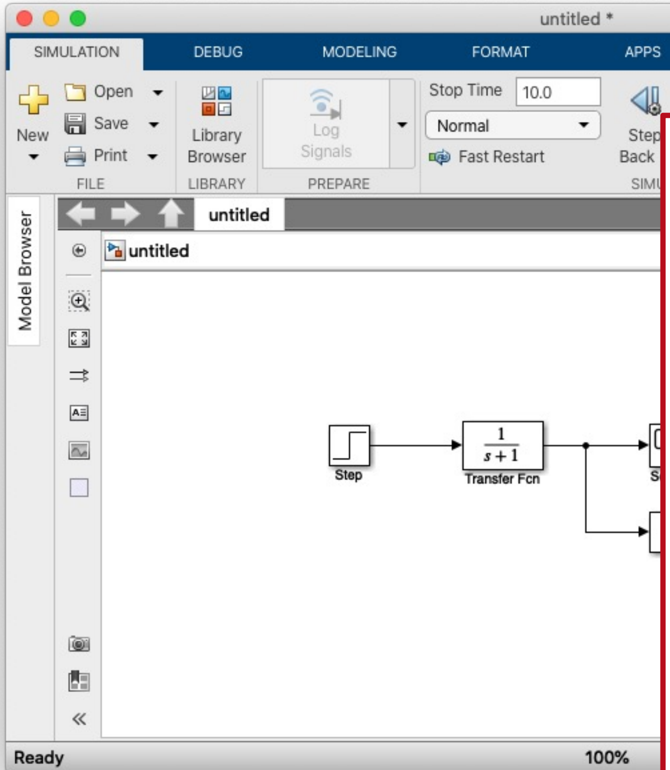


Connect blocks with connection lines.

1' Alternative: select a block by left clicking over it.

2' By left clicking over another block while pressing the *Ctrl* button, a connection between the two blocks is created.

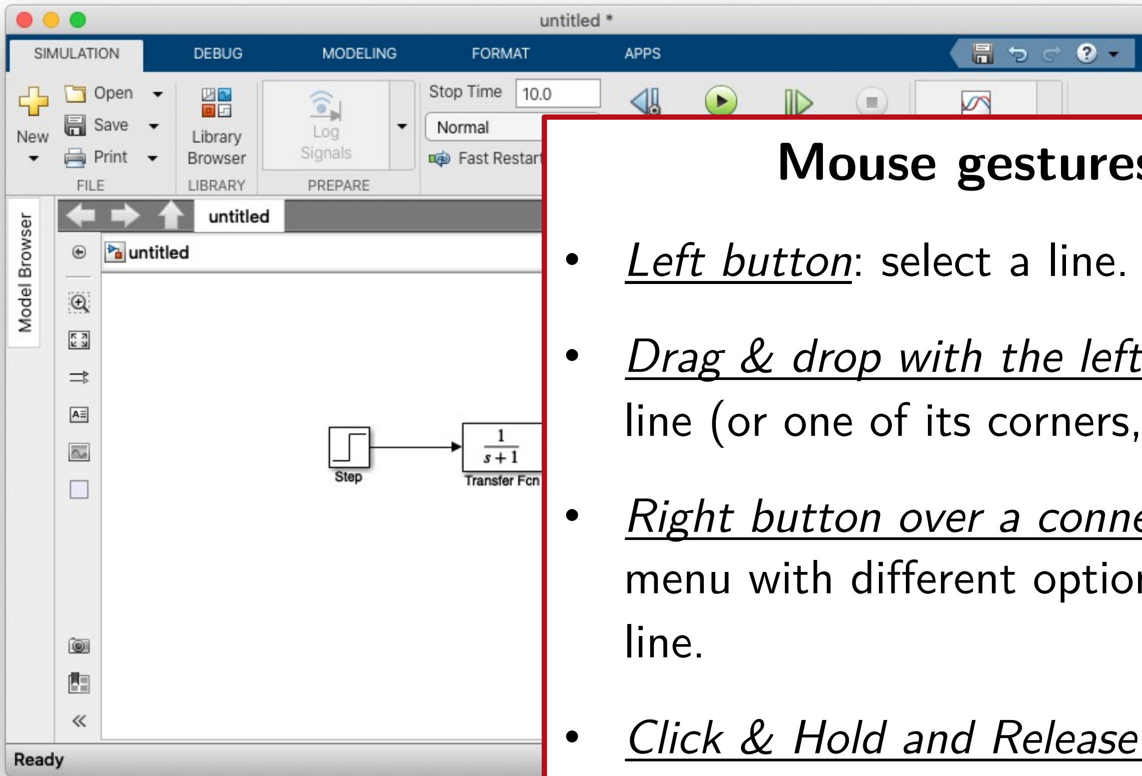
Creation of a Simulink model



Mouse gestures for blocks

- *Left button*: select a block.
- *Drag & drop with the left button*: move the block.
- *Click and Hold and Release with the left button*: select a group of blocks.
- *Right button over a block*: open a pop-up menu with different options available for the block.
- *Drag & drop with the right button*: create a copy of the selected block.
- *Double click with the left button over a block*: open the *Block Parameters window* associated with the block.

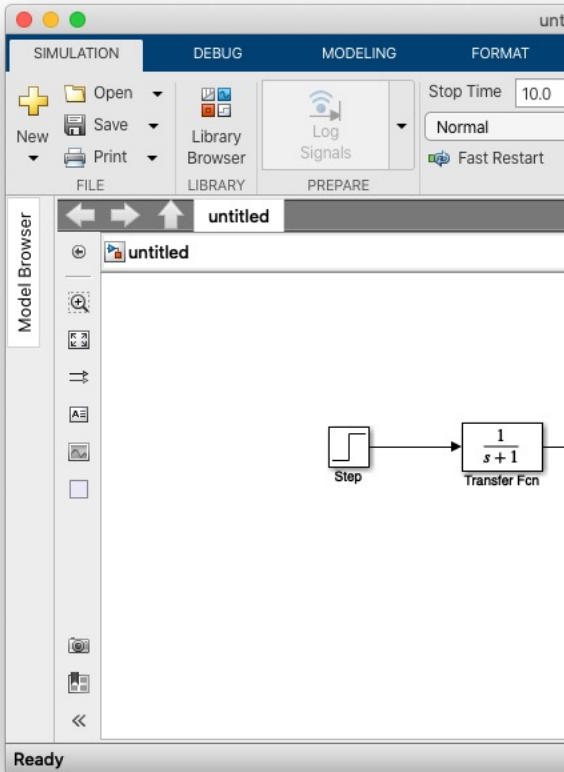
Creation of a Simulink model



Mouse gestures for connections

- *Left button*: select a line.
- *Drag & drop with the left button*: drag the connection line (or one of its corners, if selected).
- *Right button over a connection line*: open a pop-up menu with different options available for the connection line.
- *Click & Hold and Release with the right button*: create a derivation point in a connection line.
- *Double click with the left button over a connection line*: assign a label to the connection.

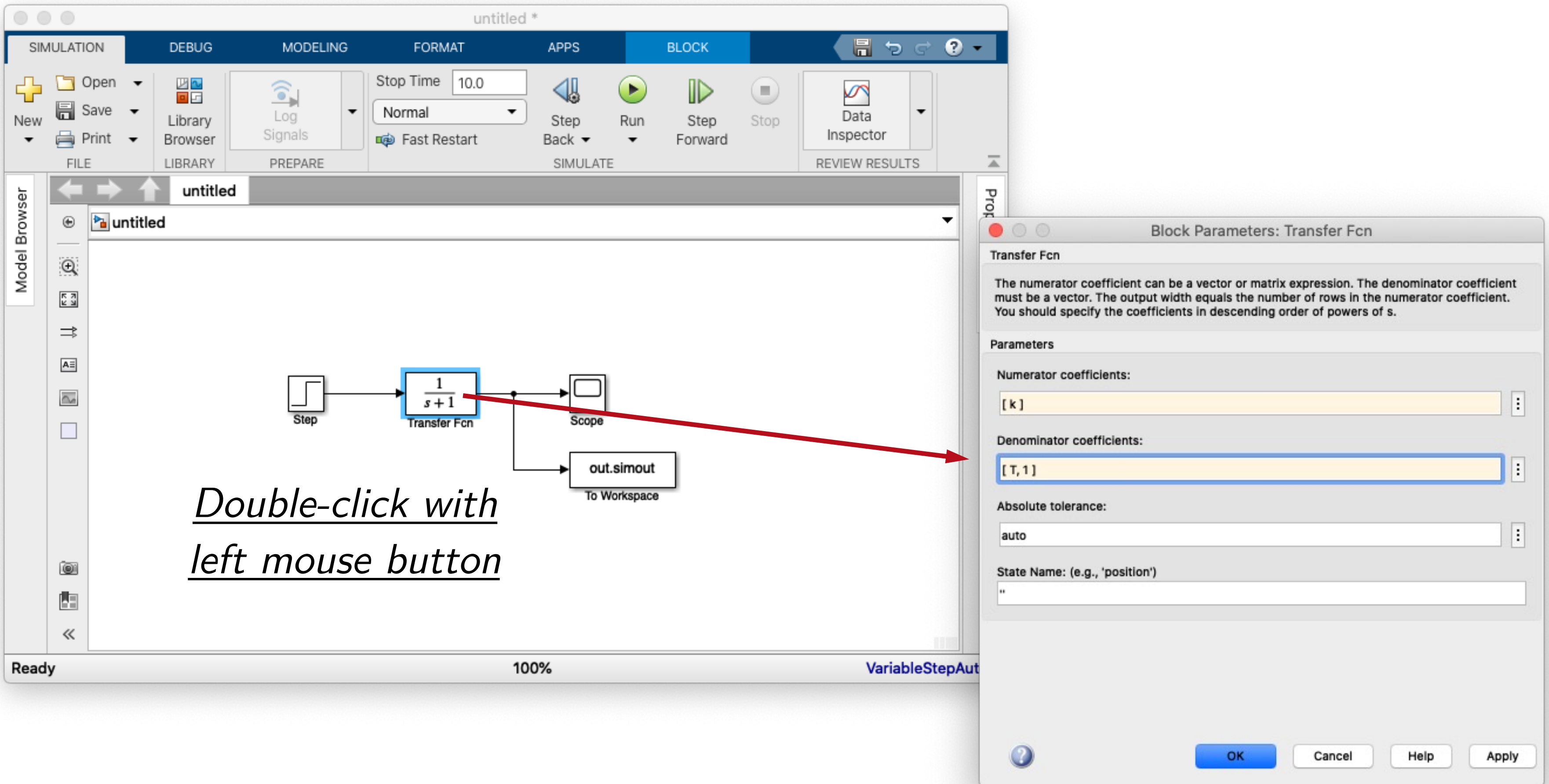
Creation of a Simulink model



Keyboard shortcuts

- To delete a block, select it and press the `Canc` button; alternatively, use the *Delete option* in the context menu of the block (right click to access it).
- To duplicate blocks, use *cut & paste* (i.e. `Ctrl+C/V`), or *drag it while holding* `Ctrl` button pressed.
- To insert a block along a connection, *drag the block over the connection* and then release it.
- To disconnect a block from a connection, *drag it while holding* `Shift` button pressed.
- To flip a block, use `Ctrl+I`.
- To rotate a block (clockwise of 90°), use `Ctrl+R`.

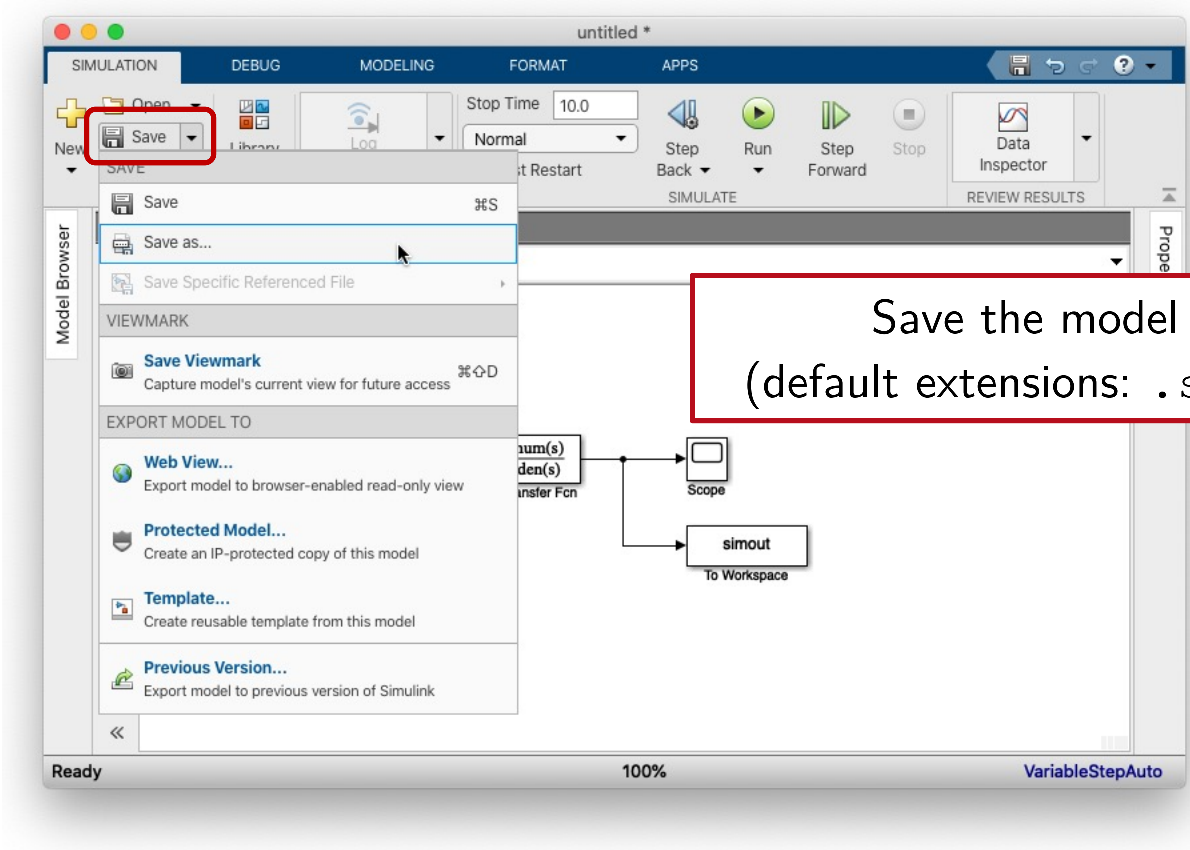
Creation of a Simulink model



Double-click with left mouse button

Set the relevant parameters of each block by accessing the *Block Parameter Window.*

Creation of a Simulink model



Save the model to file
(default extensions: `.slx`, `.mdl`).

Once the model is ready, a *simulation* can be run.

Numerical solution of ODEs

Transfer function

$$P(s) = \frac{Y(s)}{U(s)} = \frac{k}{Ts+1}$$

Model simulation $=$
numerically solve the underlying ODEs describing its dynamics.

$$T\frac{dy(t)}{dt} + y(t) = k\,u(t)$$

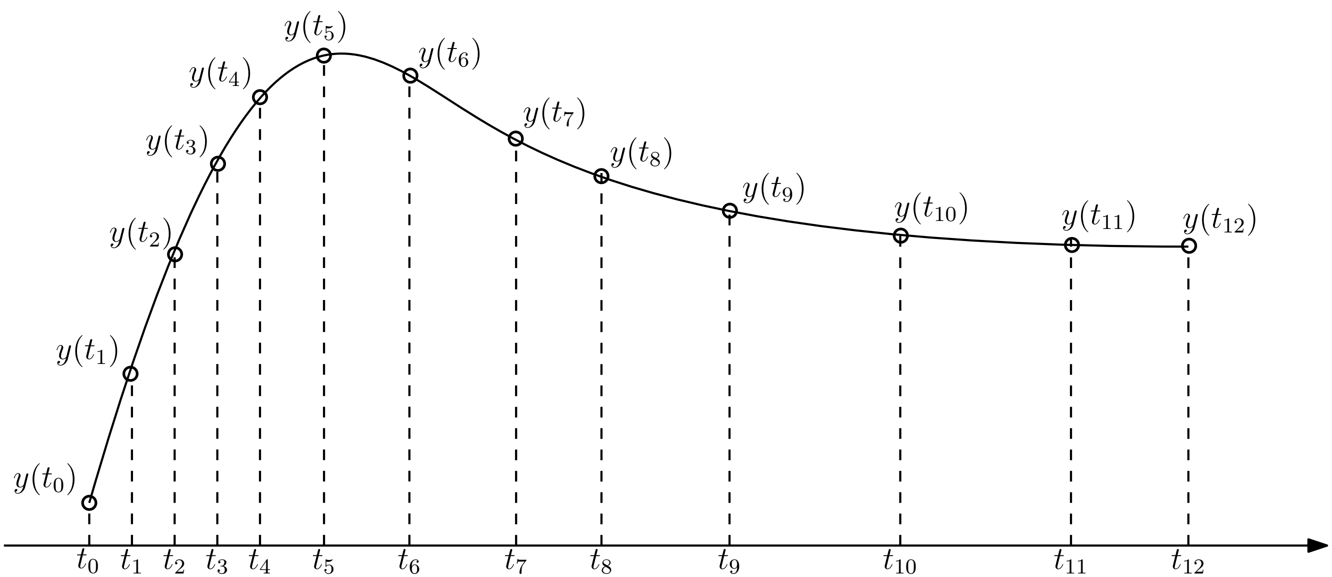
Ordinary Differential Equation (ODE)

$$\frac{dy(t)}{dt} = f\left(y(t),\, u(t)\right)$$

$$f\left(y(t),\, u(t)\right) \triangleq -\frac{1}{T}\,y(t) + \frac{k}{T}\,u(t)$$

Numerical solution of ODEs

Numerical solution (**integration**) of an ODE: find the values $y(t_i)$ of the solution $y(t)$ in specific *integration instants* t_i within the *integration interval* $[t_0, t_f]$, given the input $u(t)$ and the *initial condition* $y(t_0)$.



Numerical solution of ODEs

The numerical solution at t_{i+1} can be computed with the following iterative scheme:

$$\frac{dy(t)}{dt} = f\left(y(t), u(t)\right)$$

$$\Downarrow$$

$$y(t_{i+1}) = y(t_i) + \int_{t_i}^{t_{i+1}} f\left(y(\tau), u(\tau)\right)\, d\tau$$

provided that the integral can be numerically approximated with a certain *integration method*.

Numerical solution of ODEs

Original ODE (*integral form*)

$$y(t_{i+1}) = y(t_i) + \int_{t_i}^{t_{i+1}} f\left(y(\tau), u(\tau)\right)\, d\tau$$

Numerical approximation

$$y(t_{i+1}) = y(t_i) + T_i\, g\left(y, u, T, f\right)$$

- $T_i = t_{i+1} - t_i$: i^{th} *integration step* size.
- $g(\cdots)$: depends on the *integration method.*

ODE solvers

- **Fixed-step methods**: the integration step size is a fixed value, selected "a priori".

- **Variable-step methods**: at every integration instant t_i, the next integration step size T_{i+1} is adjusted in order to keep the integration error below a specified threshold.

  The integration error is generally estimated by comparing solutions obtained with different integration steps and/or methods (orders).

ODE solvers

↳ **Single-step methods**: the solution $y(t_{i+1})$ at t_{i+1} is obtained by using *only* the solution $y(t_i)$ and the input $u(t_i)$ of last time step t_i.

– e.g. *Euler*, *Runge-Kutta*, *Adams* methods.

↳ **Multi-step methods**: the solution $y(t_{i+1})$ at t_{i+1} is obtained by using the solutions $y(t_j)$ and inputs $u(t_j)$ from multiple previous time steps t_j with $j \leq i$.

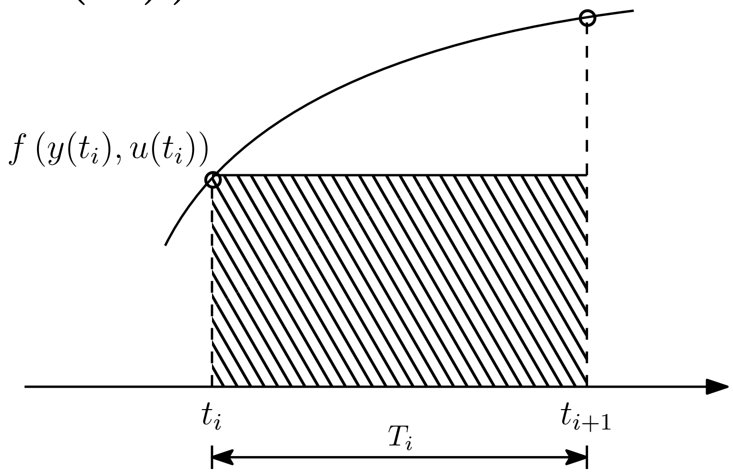
– e.g. *Predictor-corrector* methods.

ODE solvers

Euler integration method:

$$y(t_{i+1}) = y(t_i) + \int_{t_i}^{t_{i+1}} f\left(y(\tau), u(\tau)\right)\, d\tau$$

$$\approx y(t_i) + T_i\, f\left(y(t_i), u(t_i)\right)$$



⇒ Simple single-step method, but not very accurate.

ODE solvers

Runge-Kutta (R-K) methods: like Euler method, use only the solution $y(t_i)$ from last step as initial condition to determine $y(t_{i+1})$ (*single-step methods*).

However, they employ multiple evaluations (*stages*) of the function $f(\cdots)$ within the integration interval, to improve accuracy.

Most MATLAB solvers for *non-stiff* problems are based on these methods.

ODE solvers

General purpose *variable-step* solvers:

- ↳ **ode45**: compares 4th and 5th orders R-K methods to get the step size. Works well for most of the models. To be preferred as first trial. Not appropriate for stiff models.

- ↳ **ode23**: compares 3rd and 4th orders R-K methods to get the step size. Faster than ode45, but less accurate.

- ↳ **ode113**: *multi-step* method for computational demanding models. Very accurate, but not suited for *hybrid-time* systems (continuous + discrete-time).

- ↳ **discrete**: to be used for *purely* discrete-time models.

ODE solvers

Variable-step solvers for "*stiff models*"(1):

↳ **ode15s**: very effective for stiff models. First method to consider when ode45 does not work, or is very slow. Not suited for hybrid-time systems.

↳ **ode23s**: faster than ode15s, but less accurate.

↳ **ode23t**, **ode23tb**: alternative methods based on the trapezoidal integration rule.

---

(1) *Stiff models* are those described by set of equations that have large differences in their time constants.

General purpose methods would take into account only the shortest time constant, therefore they would advance very slowly (or they would not converge at all).

ODE solvers

Fixed-step solvers:

↳ **ode5**: 5th order R-K method (*Dormand-Prince* formula).

↳ **ode4**: 4th order R-K method.

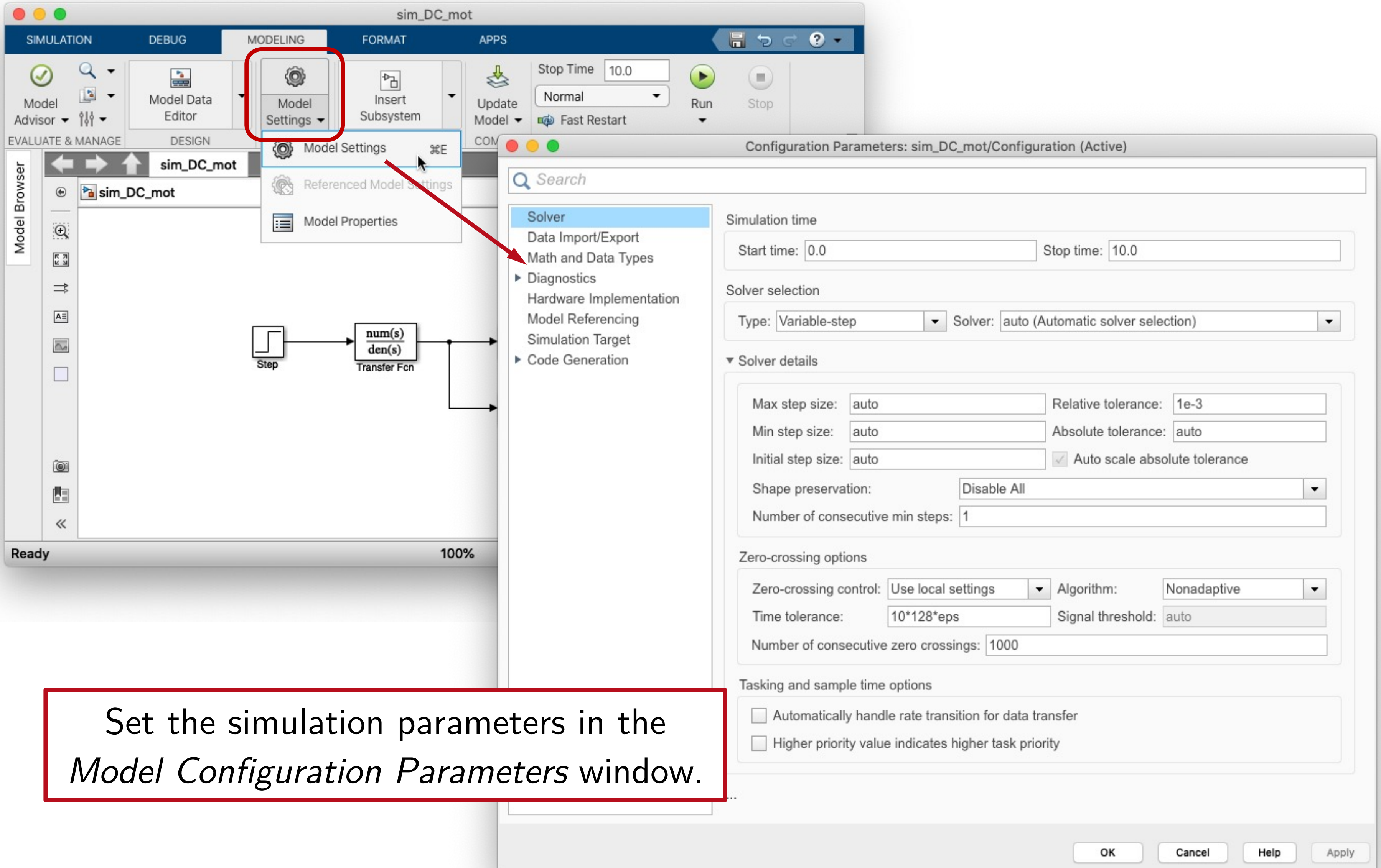
↳ **ode3**: 3rd order R-K method (*Bogacki-Shampine* formula).

↳ **ode2**: Heun method.

↳ **ode1**: Euler method.

↳ **discrete**: to be used for *purely* discrete-time models.

Simulation Parameters



Set the simulation parameters in the *Model Configuration Parameters* window.

Simulation Parameters

Main *variable-step* solvers parameters:

- **Max step size**: the integration step is adjusted up to this value. Too large values could hide some solution details. For periodic solutions, it must be set to a fraction of the period.

- **Min step size**: the integration step is never reduced below this value. Must be smaller than the smallest time constant of the model.

Simulation Parameters

Main *variable-step* solvers parameters:

↳ **Relative and absolute tolerances**: define the max *relative* and *absolute* integration errors.

At each step, if the integration error estimate exceeds the threshold:

$$\boxed{\max\left(\mathrm{RelTol} \times |y(t_i)|,\ \mathrm{AbsTol}\right)}$$

the solution is dropped, and the integration is repeated with a shorter step size.

Simulation Parameters

$$\boxed{\max\left(\mathrm{RelTol} \times |y(t_i)|,\ \mathrm{AbsTol}\right)}$$

`RelTol`: defines the number of significant digits of the solution. The default value `1e-3` corresponds to a 0.1% precision.

`AbsTol`: defines a threshold value below which the solution can be considered negligible (default: `1e-6`).

Simulation Parameters

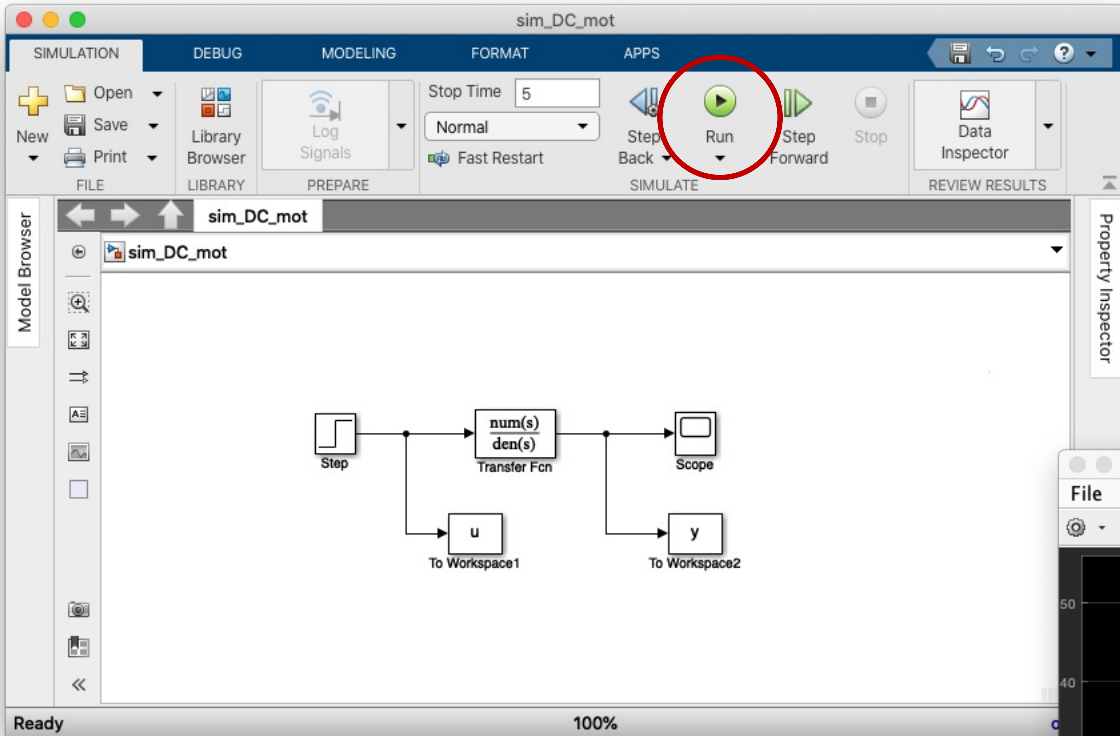
Main *variable-step* solvers parameters (cont'd):

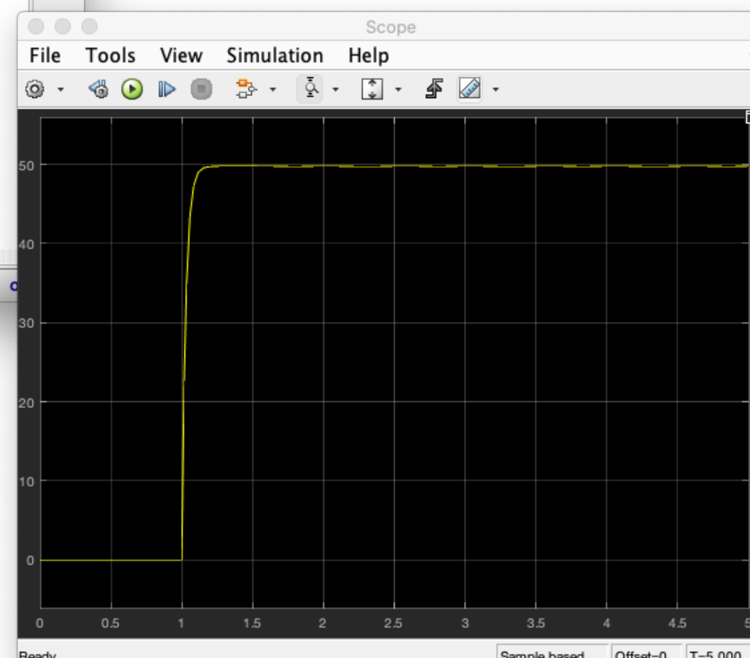
↳ **Fixed-step size**: set the integration step size.

For discrete-time models, it corresponds to the fundamental sample time T_s of the model (or a sub-multiple of it).

The `auto` value corresponds to $1/50$ of the integration interval.

Running Simulation





Run the simulation by pressing the *Run button* on the toolbar of the Model Window.

Analyze simulation results

At the end of simulation, results exported with either *To Workspace* blocks, or the *Data Logging feature* of *Scope* blocks, are available in the MATLAB workspace for further processing.