MATLAB / Simulink Lab Course

Simulink Fundamentals

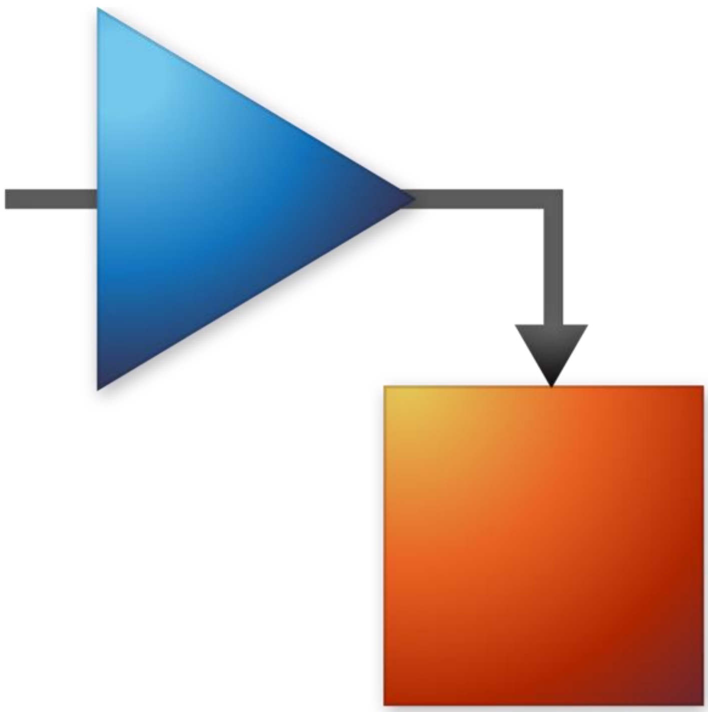

Table of Contents

Table of Figures

List of Tables

Table of Code

0 Introduction to Exercises in Simulink Fundamentals

This little booklet contains the exercises for the session about ‘Simulink Fundamentals’. The exercises are design such that the level of difficulty and the expected independent work increase from exercise to exercise.

The first example is a very easy modelling task, where almost every step is explained in detail. The second example is a little more challenging, however, a rather detailed walk-through is given. The third example is an inverse task, where a Simulink model is given, and the governing equations are to be extracted.

1 Second Order Lag

Many physical systems can be modelled by second order lag behaviour. The transfer function for these systems is

$$\frac{Y(s)}{U(s)} = G(s) = \frac{K}{1 + 2\zeta T s + T^2 s^2}$$

with damping ration ζ, gain K and time constant T. In order to be used with Simulink, the system has to be transformed to the time-domain

$$T^2\ddot{y} + 2\zeta T\dot{y} + y = Ku$$
$$\ddot{y} = -\frac{2\zeta}{T}\dot{y} - \frac{1}{T^2}y + \frac{K}{T^2}u$$

The initial conditions are given by

$$y(0) = y_0$$
$$\dot{y}(0) = \dot{y}_0$$

This system is now to be modelled in Simulink, and its response to various inputs is to be investigated.

Attention:

- **In the exercises to follow: whenever naming something (signals, blocks, etc.) make sure to NOT INCLUDE additional spaces!**
- **Also 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 this booklet! (needed for automatically checking the models)**
- **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.**

Exercise (13 points, 1 point each)

(1) In the folder "exercise_1" create an initialization file called <u>init_PT2.m</u> where the following variables are set:

```
K = 2;
T = 0.1;
zeta = 0.2;
y_0 = 1;
y_dot_0 = 0;
```

Code 1-1: init script for PT2 example

run the script.

(2) Create a new Simulink model, and save it as <u>PT2.slx</u> in the folder <u>Exercise_1</u>.

(3) Add the following blocks to the model, either via the 'Library Browser' (by clicking on the  icon) or by using the smart search function (i.e. click somewhere on the canvas and start typing the block name):

- **Simulink/Commonly Used Blocks/Integrator**
  rename to block as `Integrator_y`. Double-click on the integrator and change the initial condition to `y_0`.

- **Simulink/Commonly Used Blocks/Integrator**
  rename the block as `Integrator_y_dot`. Double-click on the integrator and change the initial condition to `y_dot_0`.

- **Simulink/Math Operations/Add**
  Make sure, the block's name is `Add`. Double-click on the add-block and change the list of signs parameter to `+--`

(4) Connect the blocks to obtain the diagram as shown in Figure 1-1. Rename the signals accordingly to `y_ddot`, `y_dot`, and `y`, by double clicking on the signal and typing the name.

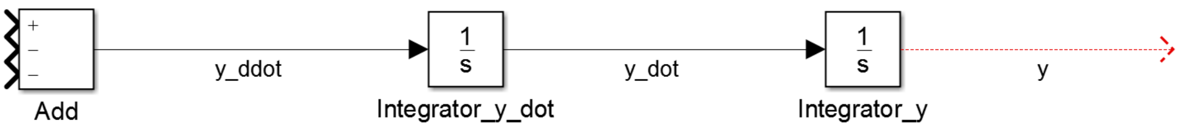

Figure 1-1: Integrator chain for PT2 excercise

(5) Add the following blocks to the diagram:

- **Simulink/Commonly Used Blocks/Gain**
  rename the block as `Gain_y`. Double-click on it to change the gain value to `1/T^2`. Flip the block horizontally by right-clicking on it, and choosing 'Rotate & Flip -> Flip Block' (or use `Ctrl+i`).

- **Simulink/Commonly Used Blocks/Gain**
  rename the block as `Gain_y_dot`. Double-click on it to change the gain value to `2*zeta/T`. Flip the block horizontally by right-clicking on it, and choosing 'Rotate & Flip -> Flip Block'.

- **Simulink/Commonly Used Blocks/Gain**
  rename the block as `Gain_u`. Double-click on it to change the gain value to `K/T^2`.

(6) Connect the blocks to obtain the diagram as shown in Figure 1-2. Rename the input signal to the `Gain_u`–block as `u`.
Hint: branch a signal by right-clicking and holding down the mouse button

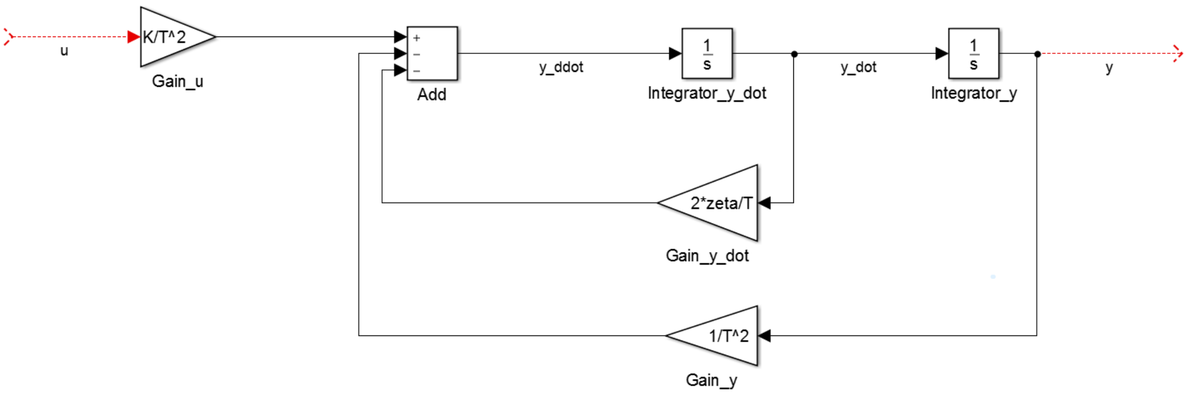

Figure 1-2: PT2 system

(7) Add the following blocks to the diagram:

- **Simulink/Sources/Step**
  Make sure, the block's name is `Step`. Double click on the block to change the step time to `5`, and keep the initial value of `0` and final value of `1`.

- **Simulink/Commonly Used Blocks/Mux**
  keep the number of inputs of 2.

- **Simulink/Commonly Used Blocks/Scope**
  Make sure, the block's name is `Scope`. Double click on the scope block to open it.
  Click on the  button in the scope window, to change its 'Configuration Properties'. Set the Number of input ports to `2`. Click on the "Layout" button and chose 2 vertically stacked fields.
  In the 'Display' tab, choose Active display 2 in the dropdown menu. Then activate the checkbox 'Show legend'

(8) Connect the blocks to obtain the diagram as shown in Figure 1-3.

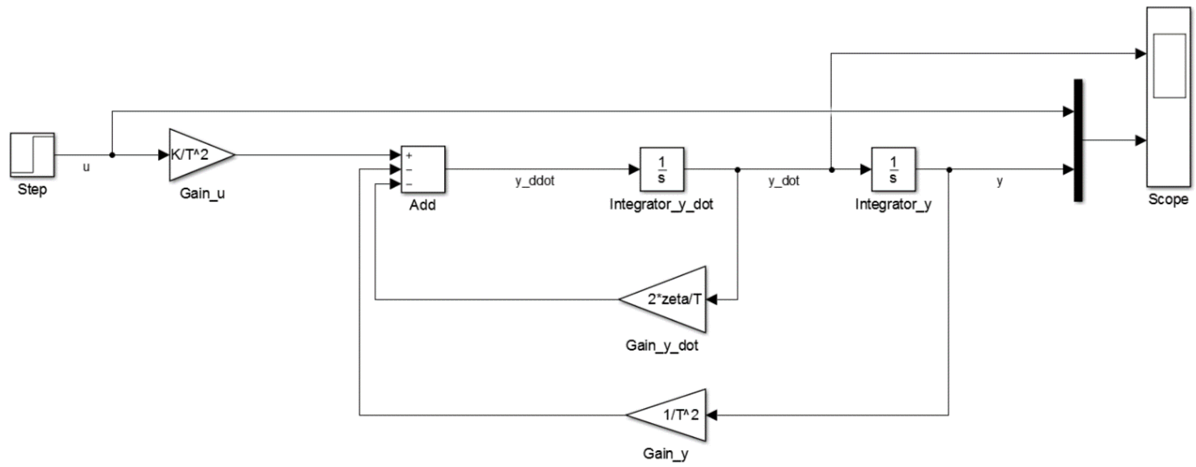

Figure 1-3: Complete PT2 system

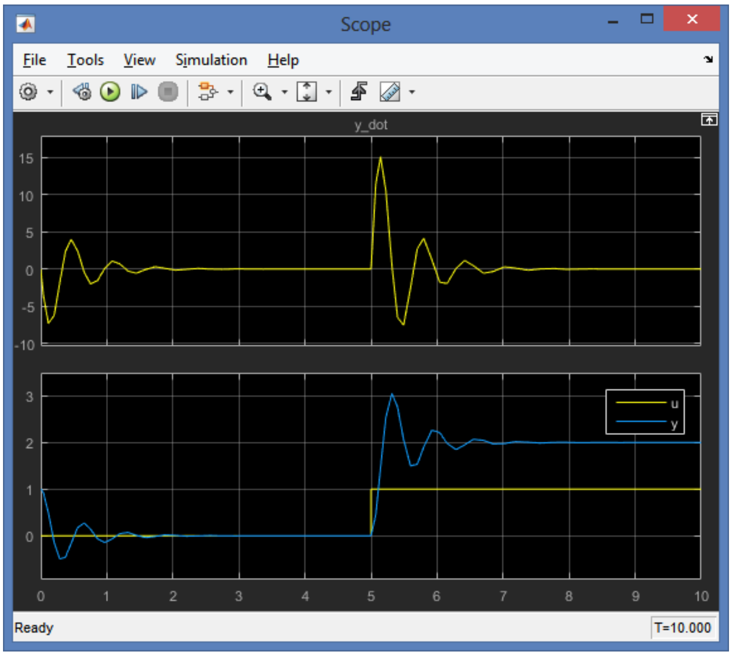

Figure 1-4: Simulation result

(9) Now the system is built up. Click the button or hit `Ctrl+T` to run the simulation. The scope should then display the simulation results as can be seen in Figure 1-4.

(10)When looking at the plots, one can see that they are not very smooth. The solver, which is automatically chosen by Simulink in the default setting, obviously takes rather large time-steps when applying the numerical integration scheme.

To improve this, click on in the Simulink main editor window to open the 'Configuration Parameters' dialog. In the 'Solver' pane, set the 'Solver' property to `ode45 (Dormand-Prince)`. Expand the 'Additional Options' area, by clicking on the arrow next to it and set the 'Max step size:' property to `0.01`. After a new simulation run, the output in the scope window should now look like Figure 1-5.

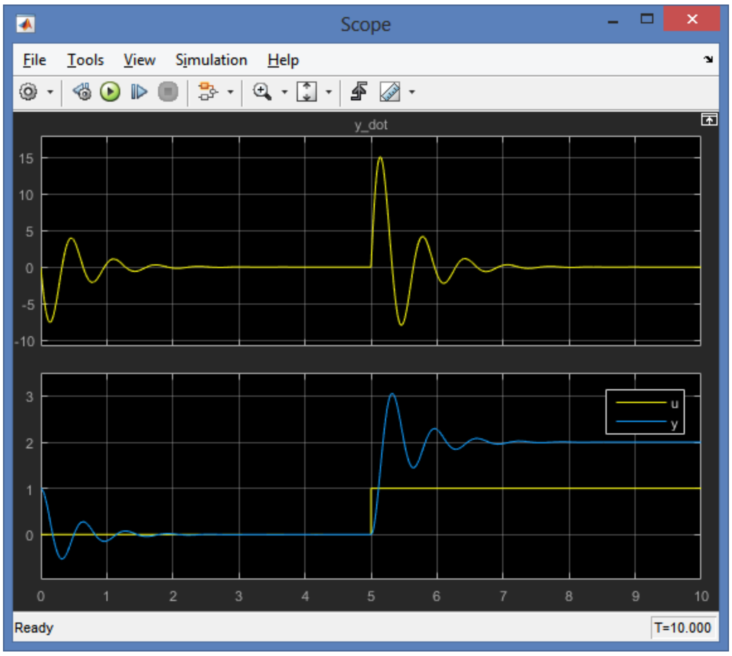

Figure 1-5: Simulation result with adapted maximum step size

(11)In order to separate the dynamics from inputs and visualization, we will now create a subsystem, that only contains the actual dynamics. To do so, mark all blocks, that are relevant to the dynamics, as can be seen in Figure 1-6. Be sure not to select the signal u going to the Mux-Block!

- Right-Click on one of the marked blocks and choose ‘Create Subsystem from Selection’ or press `Ctrl+G`. Rename the subsystem to `PT2_dynamics`.
- Enter the subsystem by double-clicking, and rename the input and output blocks as `u`, `y`, and `y_dot`. Double-click on the `y_dot` output and change its port-number to `2`. Note how the order in the top-level subsystem changes.
- Check, that the signal entering gain block `Gain_u` is still called `u`.

(12)Finally, the blocks’ sizes and positions can be adjusted by dragging&dropping them to obtain the diagram of Figure 1-7.

(13)Save your model and run the file call_test_ex1.p (right-click and choose ‘Run’), if you get the message

```
CONGRATULATIONS you passed Excercise 1!!
```

you may proceed to the next exercise.

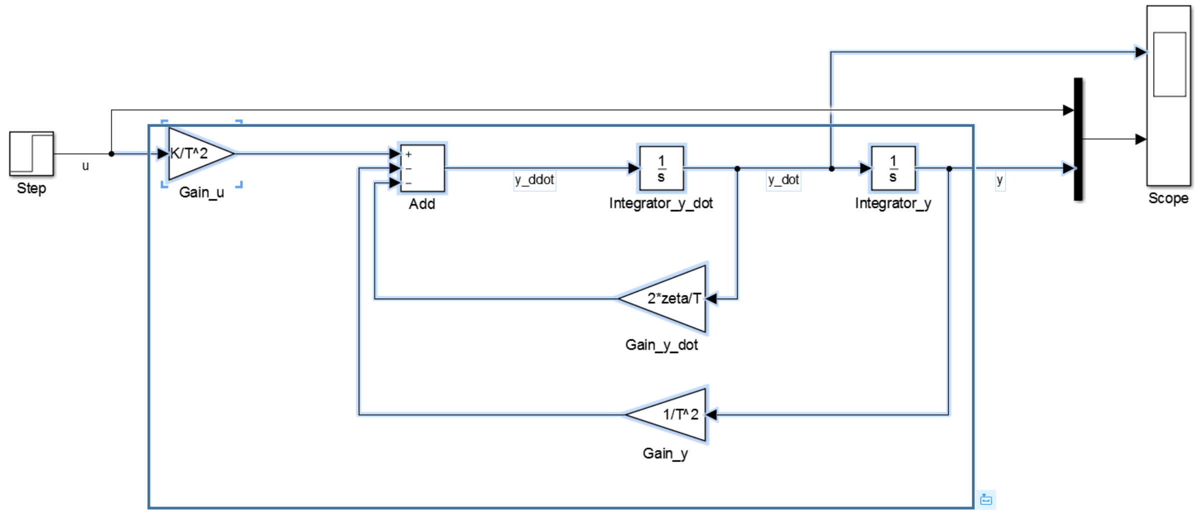

Figure 1-6: marked blocks to be converted into a subsystem

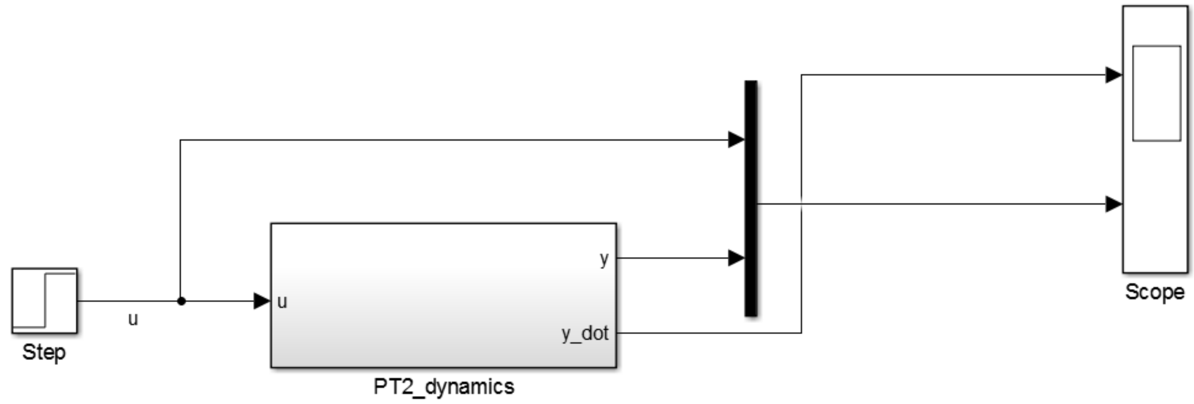

Figure 1-7: Final form of model PT2.slx

2 Non-Linear Pendulum

One of the most common examples for dynamic systems is a mathematical pendulum. An example of this can be seen in Figure 2-1.

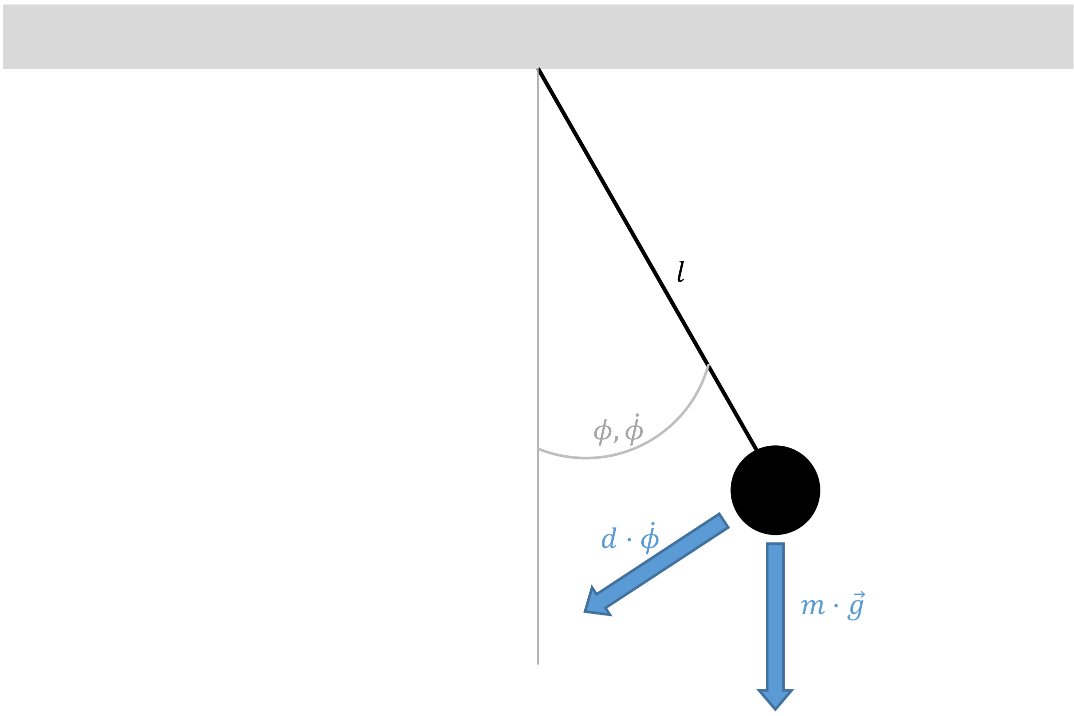

Figure 2-1: schematic of the pendulum

The non-linear dynamics are given by

$$\underbrace{\frac{d}{dt}\begin{bmatrix}\phi \\ \dot{\phi}\end{bmatrix}}_{:=\dot{x}} = \underbrace{\begin{bmatrix}\dot{\phi} \\ -\frac{g}{l}\sin\phi - \frac{d}{ml^2}\dot{\phi}\end{bmatrix}}_{:=f(x)} \tag{2-1}$$

$$\phi(0) = \phi_0$$
$$\dot{\phi}(0) = \dot{\phi}_0$$

with the Earth's gravitation g, the mass m, the length of the rod l and a damping constant d. If, in addition to the angle ϕ and the angular velocity $\dot{\phi}$, the force acting along the rod is to be investigated, the following output equation has to be considered

$$\underbrace{\begin{bmatrix}\phi \\ \dot{\phi} \\ F_N\end{bmatrix}}_{:=y} = \underbrace{\begin{bmatrix}\phi \\ \dot{\phi} \\ mg\cos\phi + ml\,\dot{\phi}^2\end{bmatrix}}_{:=g(x)} \tag{2-2}$$

Very often, this system is linearized around $(\phi, \dot{\phi}) = (0,0)$ to obtain simpler dynamics. The resulting linearized system is then

$$\frac{d}{dt}\begin{bmatrix}\phi \\ \dot{\phi}\end{bmatrix} = \begin{bmatrix}0 & 1 \\ -\frac{g}{l} & -\frac{d}{ml^2}\end{bmatrix}\begin{bmatrix}\phi \\ \dot{\phi}\end{bmatrix} \tag{2-3}$$

$$\begin{bmatrix} \phi \\ \dot{\phi} \\ F_N \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \phi \\ \dot{\phi} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix} \qquad \textbf{(2-4)}$$

Exercise (12 points, 1 point each)

Clear all previous variables from the workspace by entering **`clear`** in the command window. In the following exercise, the two systems are to be implemented and compared.

(1) In the folder 'Exercise_2' run the script init_pendulum.m to store the model's constants in the base-workspace. Open the model pendulum.slx.

(2) Add a subsystems to the model, rename it `pendulum_non_linear`. This subsystem is to have one outport, `pendulum_out` and no inport.

(3) To make the signals easily comparable, we will create a Bus Object, storing the signal information. To do so, type `buseditor` in the command window, or use 'Edit->Bus Editor' to open the bus editor window.

Create a new bus object by using the 'Add Bus' icon in the Toolbar. In the 'Properties' section on the right, call the bus object `pendulum_bus`. Click 'Apply'.

Add three bus elements by using the 'Add/Insert BusElement' icon in the Toolbar. Per default, they will be named `a`, `a1`, and `a2`. Rename them to `F_N`, `phi_dot`, and `phi` *(pay attention to the order!)*
The bus object is now stored in the base-workspace. In order to avoid having to re-create it, every time the workspace is cleared, store the bus object to a file pendulum_bus_objects.mat by using the 'Export visible Bus Objects to File' icon. Now the bus definition can be easily loaded from disk.

(4) To use the newly created bus object, navigate to the outport in the `pendulum_non_linear` subsystems. Double-click on it and adjust the 'Data type' property in the 'Signal Attributes' pane to read `Bus: pendulum_bus`. Note how the appearance of the Block changes. If `Bus: pendulum_bus` is not available, click on `--- Refresh data types ---` in the same dropdown menu. After this, all available bus-datatypes should be present.

(5) Add the following Blocks to the subsystem `pendulum_non_linear`:

- **Simulink/Commonly Used Blocks/Integrator**
  rename block as `Integrator_phi`. Set the initial condition to `phi_0`.
- **Simulink/Commonly Used Blocks/Integrator**
  rename block as `Integrator_phi_dot`. Set the initial condition to `phi_dot_0`.
- **Simulink/Math Operations/Add**
  Make sure, the block's name is `Add`. Set the 'list of signs' parameter to `--`
- **Simulink/Commonly Used Blocks/Gain**
  rename block as `Gain_phi_dot`. Set the gain value to `d/(m*l^2)`
- **Simulink/Math Operations/Trigonometric Function**
  rename the block `sin`. Make sure, the 'Function' parameter is set to `sin`
- **Simulink/Commonly Used Blocks/Gain**
  rename block as `Gain_sin_phi`. Set the gain value to g`/l`

For the Output Equation, add the following Blocks

- **Simulink/Math Operations/Trigonometric Function**
  rename the block `cos`. Make sure, the 'Function' parameter is set to `cos`
- **Simulink/Math Operations/Math Function**
  rename the block `square`. Make sure, the 'Function' parameter is set to `square`
- **Simulink/Commonly Used Blocks/Gain**
  rename block as `Gain_cos_phi`. Set the gain value to `m*g`
- **Simulink/Commonly Used Blocks/Gain**
  rename block as `Gain_square_phi_dot`. Set the gain value to `m*l`
- **Simulink/Math Operations/Add**
  Make sure, the block's name is `Add_F_N`. Set the 'list of signs' parameter to `++`

For the signal subsystem output, add the following block:

- **Simulink/Commonly Used Blocks/Bus Creator**
  set 'Number of inputs' property to `3` and 'Output data type' to `Bus: pendulum_bus`.

(6) Connect the blocks to model the non-linear system equation. An example for this is given in Figure 2-2. Make sure to name the signals `F_N, phi_ddot, phi_dot,` and `phi.`

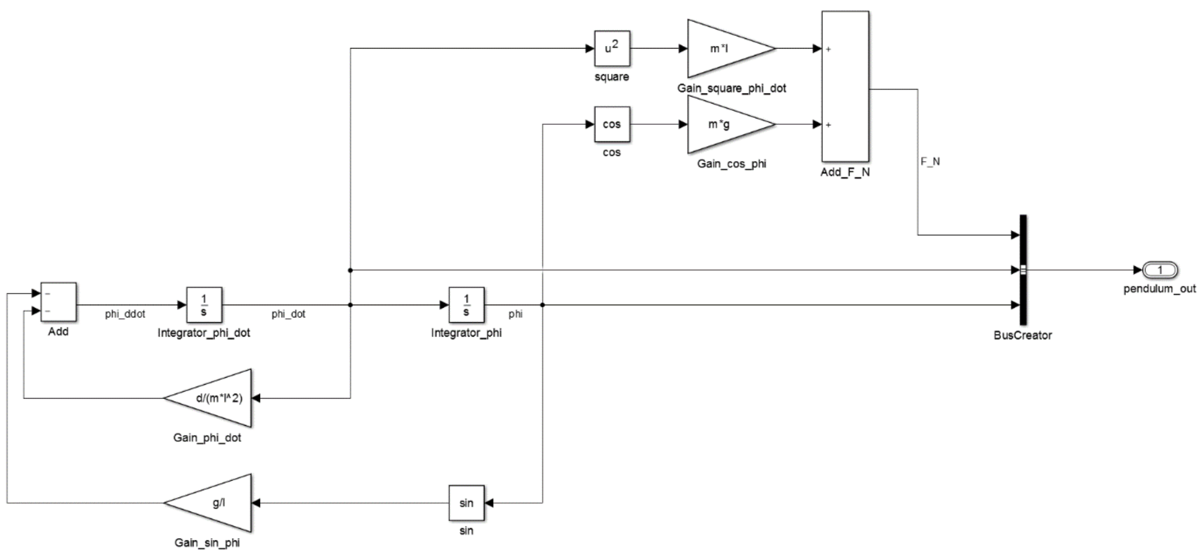

Figure 2-2: non-linear pendulum model

(7) Add a second subsystem on the root level, rename it to `visualization`.

- It shall have two inputs (`pendulum_non_linear` and `pendulum_linear`)
- Both inputs shall be of Bus type, with the `pendulum_bus` bus object definition (block properties of inports, set data-type to `pendulum_bus`)
- It shall have no outputs
- One Scope shall be used in the subsystem

i) It shall have three inputs (open scope -> Configuration Properties-> Main-> Number of Input ports)

ii) It shall have three vertically stacked subplots (Configuration Properties -> Main -> Layout-> [3 1])

iii) In all Displays, a legend is to be displayed (Configuration Properties ->Display: choose ‘Active display’, activate checkbox ‘Show legend’, click Apply after every active display was configured)

iv) In all Displays, the line-style of the second plot is to be dashed (View -> Style -> choose ‘Active display’, chose second element in ‘Properties for Line’, set ‘Line’ property to )

- In the three subplots, `phi`, `phi_dot` and `F_N` of the non-linear simulation shall be compared to the linear simulation. To do so, use two ‘Bus Selector’ blocks to split the two incoming bus signals (`pendulum_non_linear` and `pendulum_linear`). Then, use three ‘Mux’ blocks with two inputs each to mux the respective linear / non-linear signals, before the Mux-outputs are connected to the scope.

(8) Now the simulation can be run. Ignoring the warnings for now, the Scope should look like Figure 2-3.

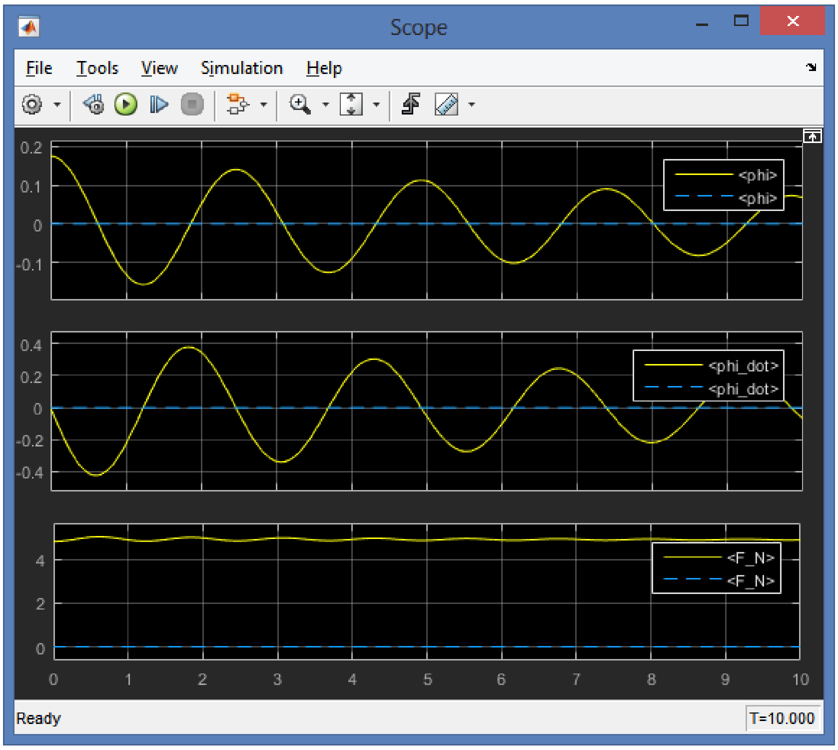

Figure 2-3: Simulation output for non-linear simulation

(9) Save your model and run the file <u>call_test_ex2_non_linear.p</u> (right-click and choose ‘Run’), in order to verify your model so far. Take care of any arising errors, until you get the message

```
CONGRATULATIONS you passed the first part of Exercise 2!!
```

(10) The second part of the exercise consists of modelling the linearized version of the system. To get the structure right, copy the subsystem `pendulum_non_linear` and rename it to `pendulum_linear`. Then adapt the model to reflect the linearized dynamics. This is done by changing the implementations of equations (2-1) and (2-2) into implementations of equations (2-3) and (2-4).
Connect the output of the linearized dynamics, `pendulum_linear`, with the second `inport` of the `visualization` subsystem. The scope output should look like Figure 2-4.
Is the linearization a good approximation? Why?

____

____

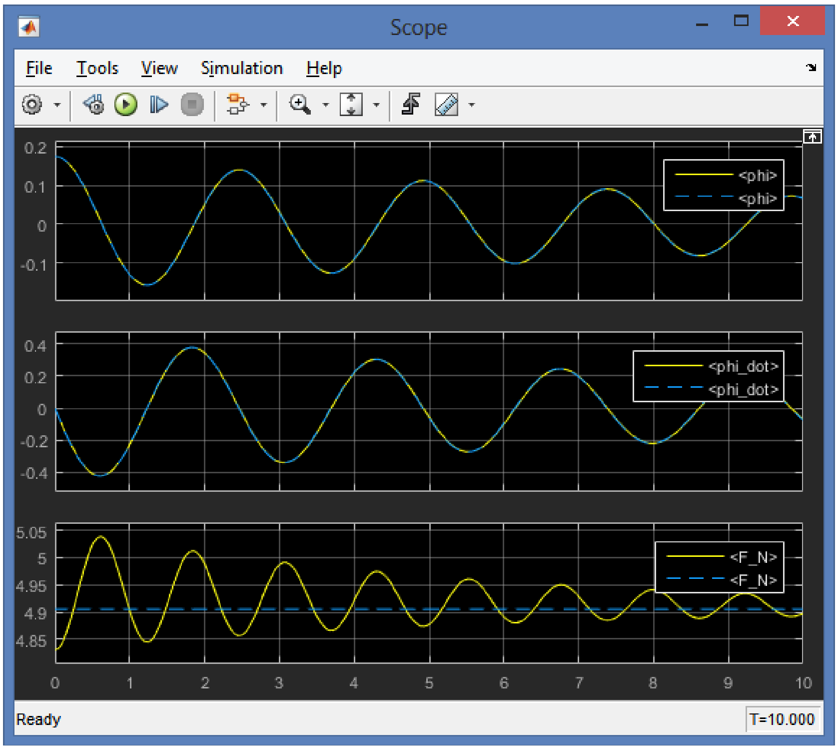

Figure 2-4: comparison of non-linear and linearized pendulum dynamics

(11) Run the file call_compare_pendulum.p to compare your implementation against a reference implementation.

(12) Change the initial values in the script init_pendulum.m to

```
phi_0 = 178 / 180 * pi;
phi_dot_0 = 0;
```

Code 2-1: changed initial values for pendulum simulation

and run the simulation again. How is the linear approximation to be considered now? Why?

____

____

____

____

3 Aircraft Point Mass Equations of Motion

One very important skill to have, when it comes to Simulink models, is to understand what other people implemented, i.e. to extract the governing equations from a given model.

This will be done, using an example implementation of the point mass equations of motion of an aircraft. The signals in Table 3-1 are present in the model

signal name	symbol	interpretation
States		
`h`	h	Current altitude of the aircraft
`V`	V	Current velocity of the aircraft
`chi`	χ	Current course of the aircraft
`gamma`	γ	Current climb angle of the aircraft
State derivatives		
`h_dot`	$\dot{h}$	Altitude derivative
`V_dot`	$\dot{V}$	Velocity derivative
`chi_dot`	$\dot{\chi}$	Course derivative
`gamma_dot`	$\dot{\gamma}$	Climb angel derivative
Inputs		
`alpha`	α	Angle of attack
`delta_T`	δ_T	Normalized thrust command
`mu`	μ	Bank angle
Other signals		
`rho`	ρ	Air density
`CL_0`	C_{L0}	Zero lift coefficient
`CL_alpha`	$C_{L\alpha}$	Lift curve slope
`CL`	C_L	Lift coefficient
`L`	L	Lift
`CD_0`	C_{D0}	Base-drag
`k`	k	Induced drag coefficient
`CD`	C_D	Drag coefficient
`D`	D	Drag
`T_0`	T_0	Idle thrust
`T_deltaT`	$T_{\delta T}$	Thrust curve slope
`alpha_T`	α_T	Engine installation angle
`X_P`	X_P	x-component of propulsion vector
`Z_P`	Z_P	z-component of propulsion vector
`m`	m	mass
`g`	g	Gravity constant
`S_ref`	S_{ref}	Wing reference area

Table 3-1: signals in point mass equations of motion model

Exercise (12 points)

Clear all previous variables from the workspace by entering `clear` in the command window.

(1) Run the script <u>init_point_mass_EOM.m</u> to initialize the model, and open the file <u>point_mass_EOM.slx</u>. **(0 points)**

(2) The aircraft aerodynamics are implemented in <u>point_mass_EOM/aerodynamics</u>. Extract the equations for the lift and drag coefficients C_L and C_D as well as the expressions for lift and drag L and D. Write them down as functions of α, V, ρ, S_{ref} and the constant terms. **(4 points)**

$$C_L =$$

$$C_D =$$

$$L =$$

$$D =$$

(3) The propulsion system is modelled in <u>point_mass_EOM/propulsion</u>. Extract the equations for the thrust T, as well as for the propulsion components in x and z direction X_P and Z_P. Write them down as functions of δ_T and constants **(3 points)**

$$T =$$

$$X_P =$$

$$Z_P =$$

(4) Extract the equation for the change in velocity $\dot{V}$ from the subsystem <u>point_mass_EOM/V_dot</u> and write it down as function of X_P, D, γ and constant terms. **(1 point)**

$$\dot{V} =$$

(5) Extract the equation for the change in course $\dot{\chi}$ from the subsystem <u>point_mass_EOM/chi_dot</u> and write it down as function of L, Z_P, μ, V, γ and constant terms. **(1 point)**

$$\dot{\chi} =$$

(6) Extract the equation for the change in climb angle $\dot{\gamma}$ from the subsystem point_mass_EOM/gamma_dot and write it down as function of L, Z_P, μ, V, γ and constant terms. **(1 point)**

$$\dot{\gamma} =$$

(7) Extract the equation for the change in altitude $\dot{h}$ from the subsystem point_mass_EOM/h_dot and write it down as function of V and γ. **(1 point)**

$$\dot{h} =$$

(8) Describe in your own words, what happens in the subsystem point_mass_EOM/Integration? **(1 point)**