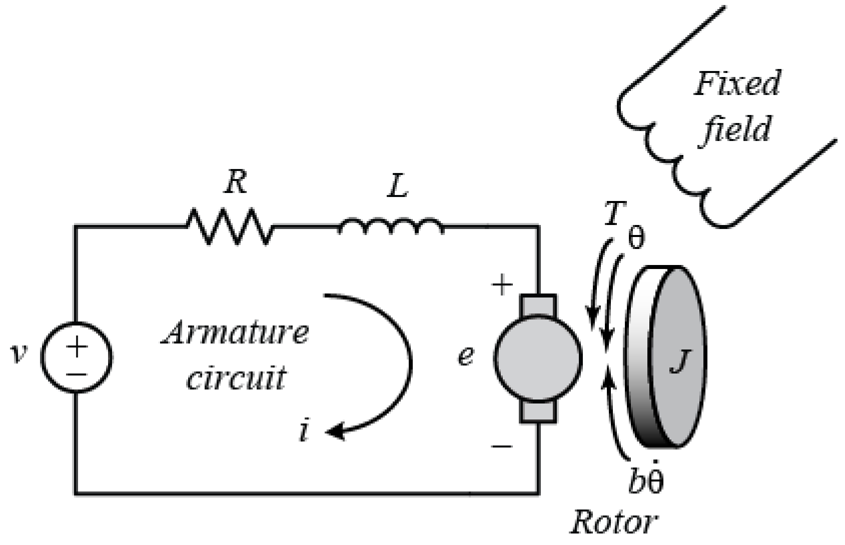
STATE-SPACE REPRESENTATION

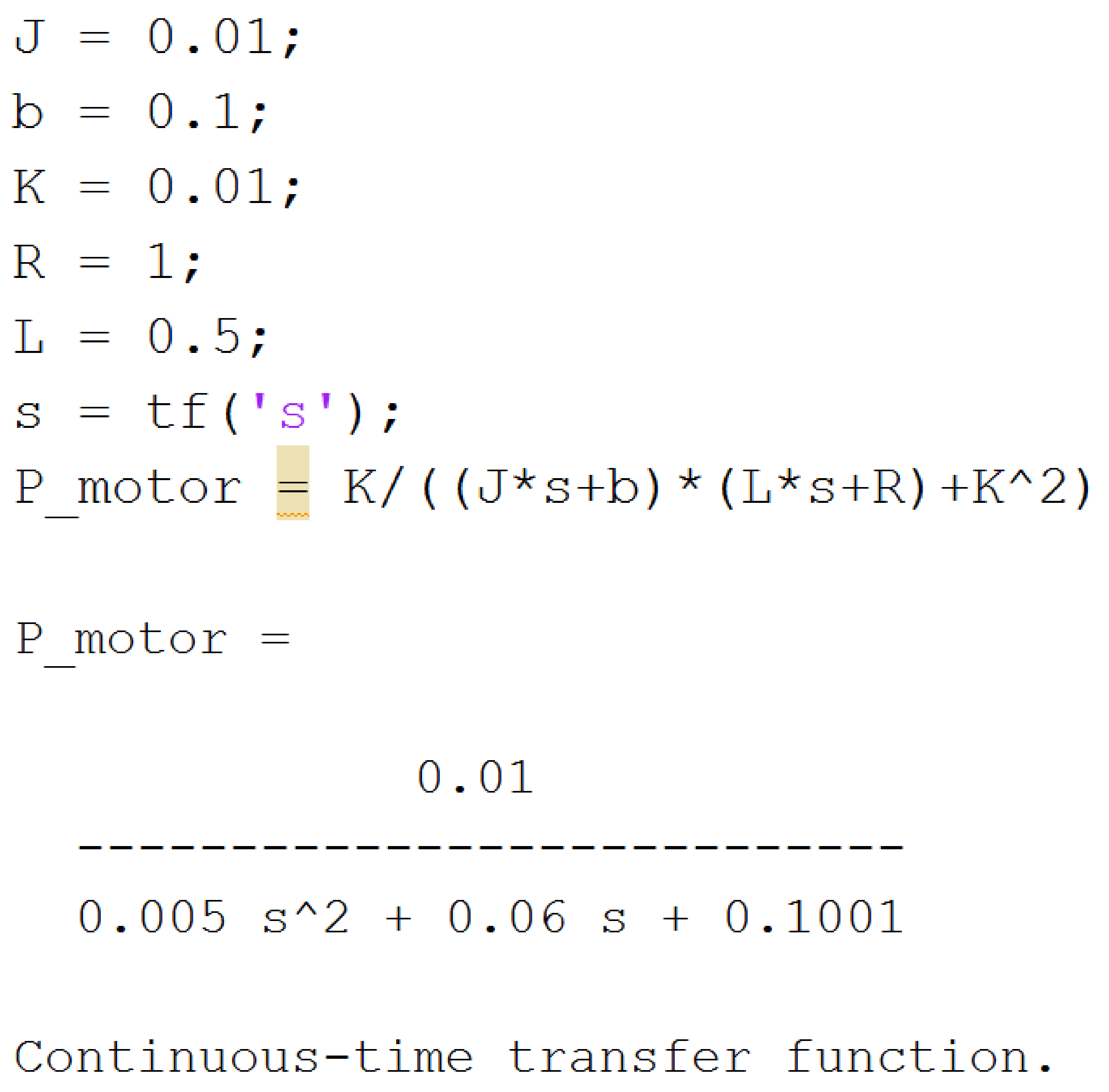
- The electric equivalent circuit of the armature and the free-body diagram of the rotor are shown in the following figure.



- We will assume that input of the system is the voltage source (V) applied to the motor's armature, while the output is the rotational speed of the shaft $\dot{\theta}$

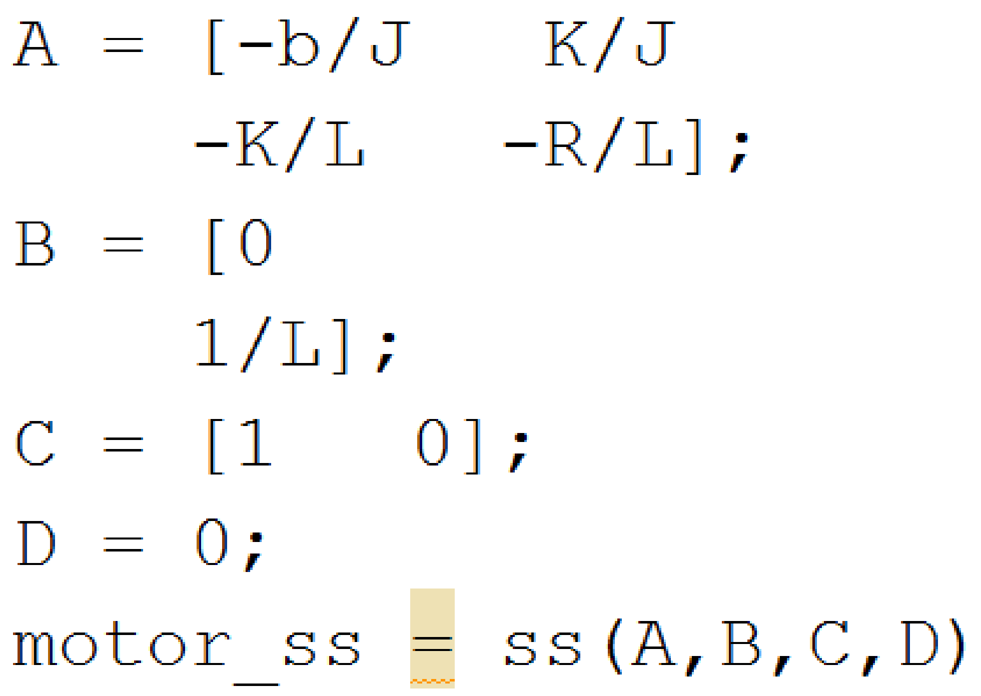
CONTROL LAB

- We can represent the open-loop transfer function of the motor in MATLAB by defining the parameters and transfer function as follows. Running this code in the command window produces the output shown below.



```
J = 0.01;
b = 0.1;
K = 0.01;
R = 1;
L = 0.5;
s = tf('s');
P_motor = K/((J*s+b)*(L*s+R)+K^2)

P_motor =

              0.01
  ---------------------------
  0.005 s^2 + 0.06 s + 0.1001

Continuous-time transfer function.
```

- We can also represent the system using the state-space equations. The following additional MATLAB commands create a state-space model of the motor when run in the MATLAB command window.

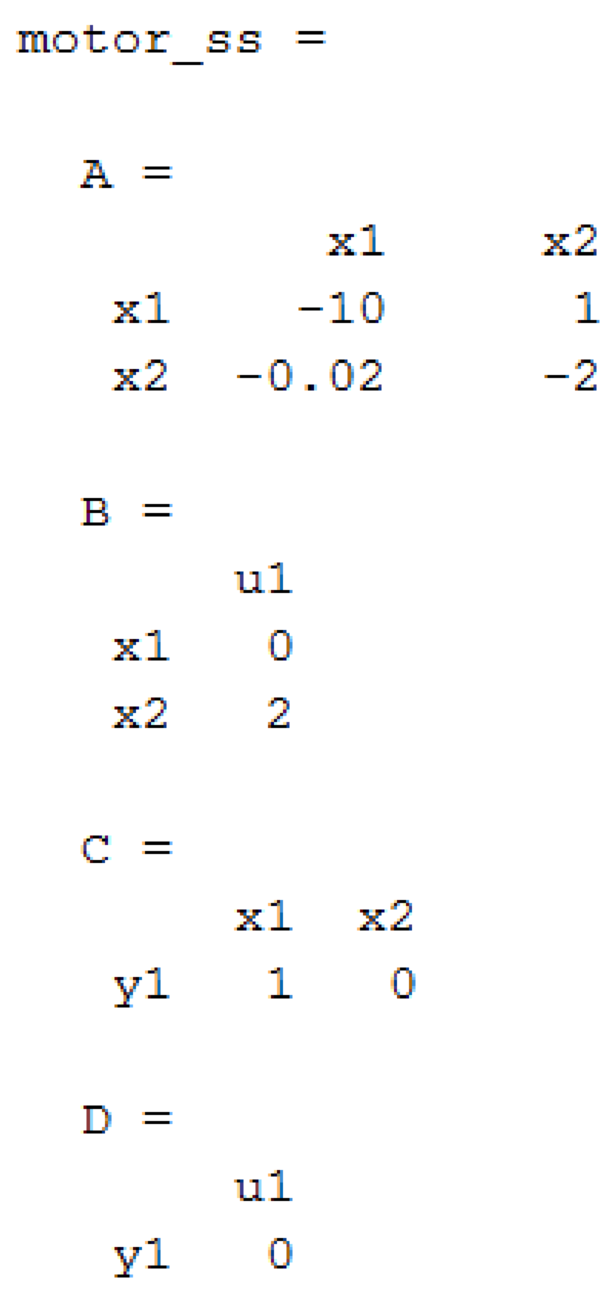


```
A = [-b/J   K/J
    -K/L   -R/L];
B = [0
    1/L];
C = [1   0];
D = 0;
motor_ss = ss(A,B,C,D)
```

- The state-space model can also be generated by converting your existing transfer functions model into state-space form. This is again accomplished with the ***ss*** command as shown below.

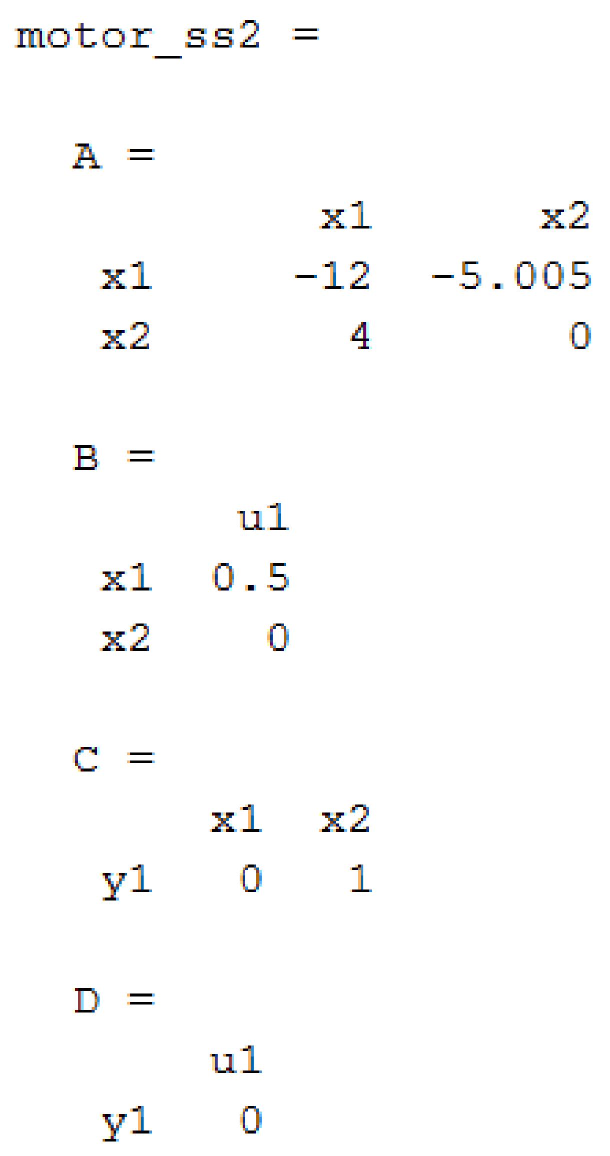
```
motor_ss2 = ss(P_motor)
```

```
motor ss = ss(A,B,C,D)
```



```
motor_ss =

  A =
            x1      x2
   x1      -10       1
   x2    -0.02      -2

  B =
        u1
   x1    0
   x2    2

  C =
        x1  x2
   y1    1   0

  D =
        u1
   y1    0
```

```
motor_ss2 = ss(P_motor)
```



```
motor_ss2 =

  A =
             x1       x2
   x1       -12   -5.005
   x2         4        0

  B =
         u1
   x1   0.5
   x2     0

  C =
        x1  x2
   y1    0   1

  D =
        u1
   y1    0
```

- System can be represented with different sate variables even though the transfer functions relating the output to the input remains the same. These systems are called similar systems. Although their state-space representations are different, similar systems have the same transfer functions and hence the same poles and eigenvalues.

```
eig1 = eig(A)
eig2 = eig(motor_ss2.A)
```

```
eig1 =

   -9.9975
   -2.0025


eig2 =

   -9.9975
   -2.0025
```

HOMEWORK 4.1

- Given the system represented in state space as follows :

$$\dot{\mathbf{x}} = \begin{bmatrix} -1 & -7 & 6 \\ -8 & 4 & 8 \\ 4 & 7 & -8 \end{bmatrix}\mathbf{x} + \begin{bmatrix} -5 \\ -7 \\ 5 \end{bmatrix} r$$
$$y = \begin{bmatrix} -9 & -9 & -8 \end{bmatrix}\mathbf{x}$$

convert the system to one where the new state vector, z, is

$$\mathbf{z} = \begin{bmatrix} -4 & 9 & -3 \\ 0 & -4 & 7 \\ -1 & -4 & -9 \end{bmatrix}\mathbf{x}$$

find and compare both of system’s eigenvalues.

HOMEWORK 4.2

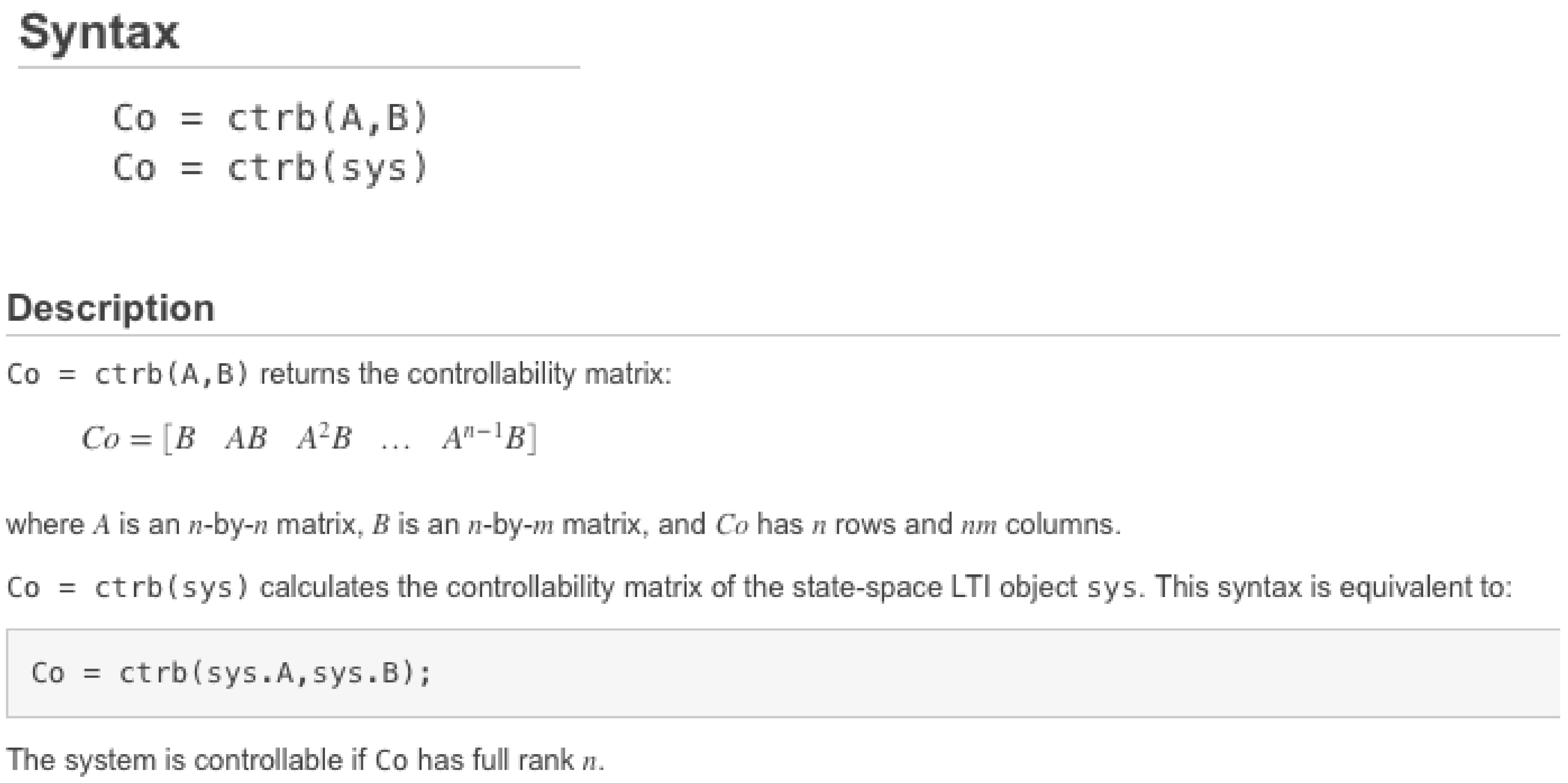
- Given the system represented in state space by Eqs, solve for y(t) using state-space and Laplace transform techniques and find the eigenvalues and the system poles.

$$\dot{\mathbf{x}} = \begin{bmatrix} 0 & 2 \\ -3 & -5 \end{bmatrix}\mathbf{x} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} e^{-t}$$

$$y = \begin{bmatrix} 1 & 3 \end{bmatrix}\mathbf{x}$$

$$\mathbf{x}(0) = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

CONTROLLABILITY



Syntax

```
Co = ctrb(A,B)
Co = ctrb(sys)
```

Description

`Co = ctrb(A,B)` returns the controllability matrix:

$$Co = [B \quad AB \quad A^2B \quad \dots \quad A^{n-1}B]$$

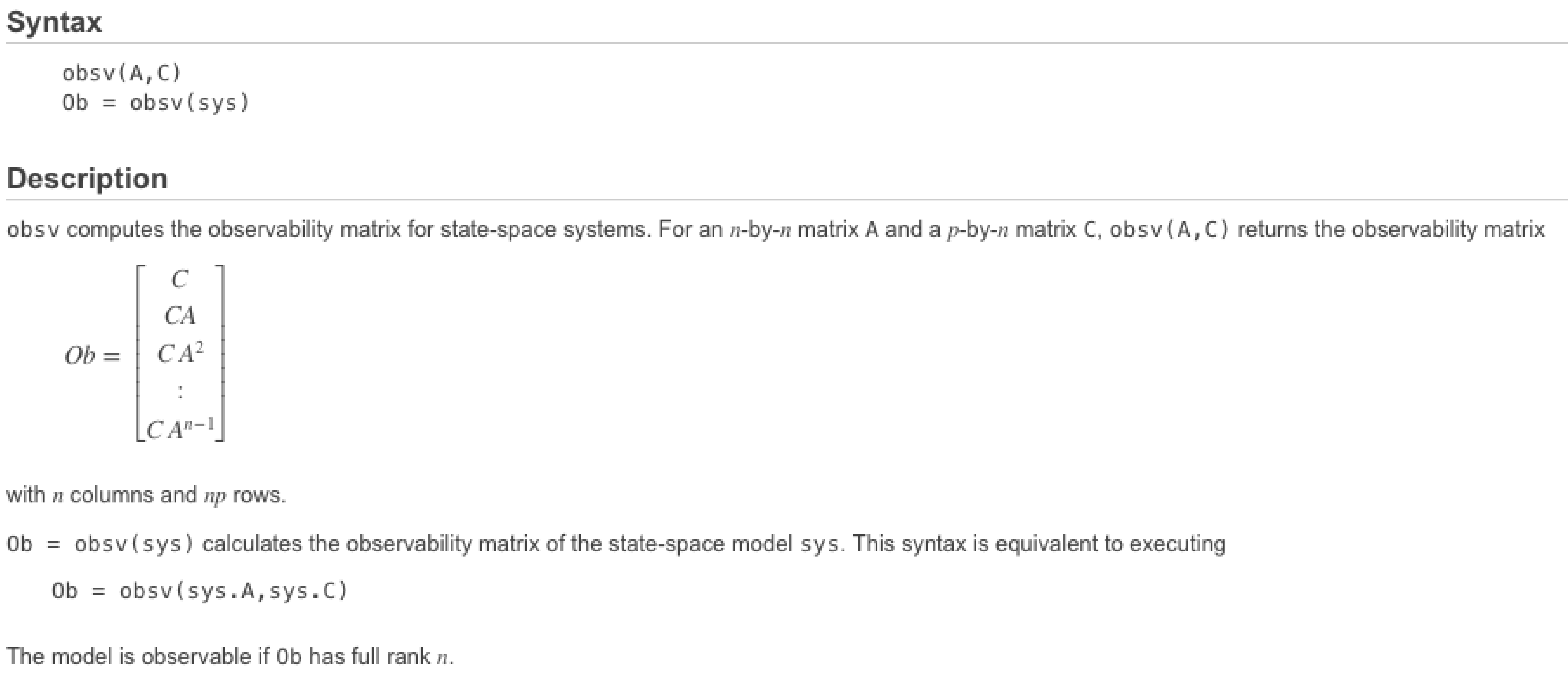
where A is an n-by-n matrix, B is an n-by-m matrix, and Co has n rows and nm columns.

`Co = ctrb(sys)` calculates the controllability matrix of the state-space LTI object `sys`. This syntax is equivalent to:

```
Co = ctrb(sys.A,sys.B);
```

The system is controllable if Co has full rank n.

OBSERVABILITY



Syntax

```
obsv(A,C)
Ob = obsv(sys)
```

Description

`obsv` computes the observability matrix for state-space systems. For an n-by-n matrix `A` and a p-by-n matrix `C`, `obsv(A,C)` returns the observability matrix

$$Ob = \begin{bmatrix} C \\ CA \\ CA^2 \\ \vdots \\ CA^{n-1} \end{bmatrix}$$

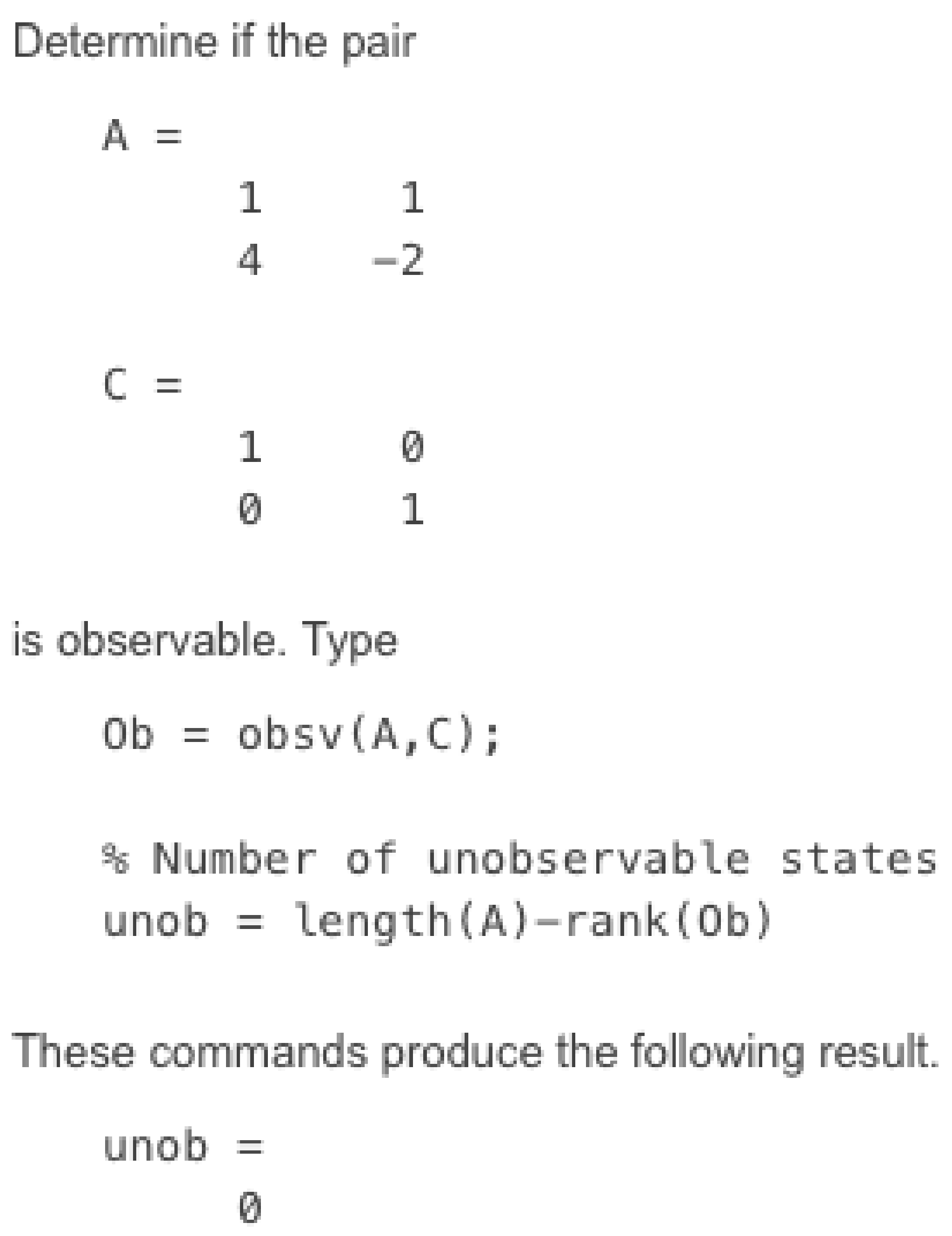
with n columns and np rows.

`Ob = obsv(sys)` calculates the observability matrix of the state-space model `sys`. This syntax is equivalent to executing

```
Ob = obsv(sys.A,sys.C)
```

The model is observable if `Ob` has full rank n.

OBSERVABILITY



Determine if the pair

```
A =
     1     1
     4    -2

C =
     1     0
     0     1
```

is observable. Type

```
Ob = obsv(A,C);

% Number of unobservable states
unob = length(A)-rank(Ob)
```

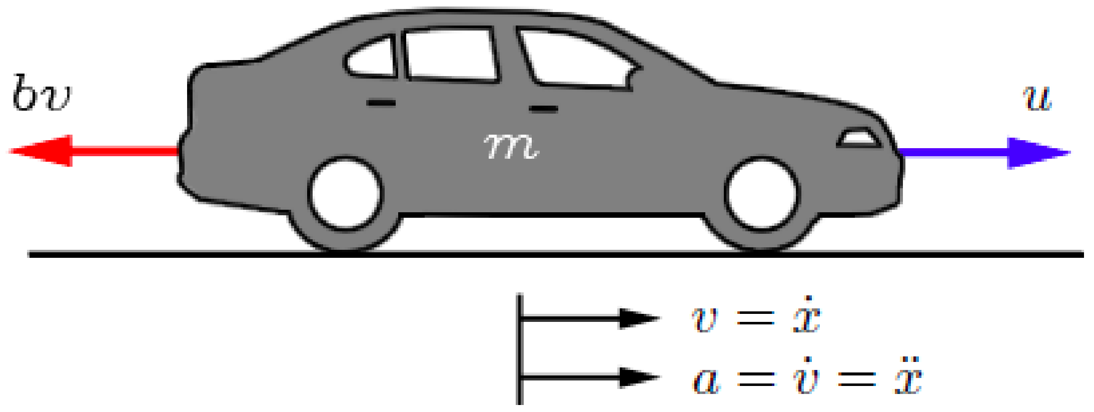
These commands produce the following result.

```
unob =
     0
```

```
J = 0.01;
b = 0.1;
K = 0.01;
R = 1;
L = 0.5;
A = [-b/J   K/J
    -K/L   -R/L];
B = [0
    1/L];
C = [1   0];
D = 0;
sys = ss(A,B,C,D);

sys_order = order(sys)
Cm = ctrb(A,B)
sys_rank = rank(Cm)


 Ob = obsv(A,C)
 Om = rank(Ob)
```

CRUISE CONTROL

- We consider here a simple model of the vehicle dynamics, shown in the free-body diagram (FBD) below.
- The vehicle, of mass m, is acted on by a control force, u. The force u represents the force generated at the road/tire interface.
- For this simplified model we will assume that we can control this force directly and will neglect the dynamics of the powertrain, tires, etc., that go into generating the force.
- The resistive forces, bv, due to rolling resistance and wind drag, are assumed to vary linearly with the vehicle velocity, v, and act in the direction opposite the vehicle's motion.



CRUISE CONTROL

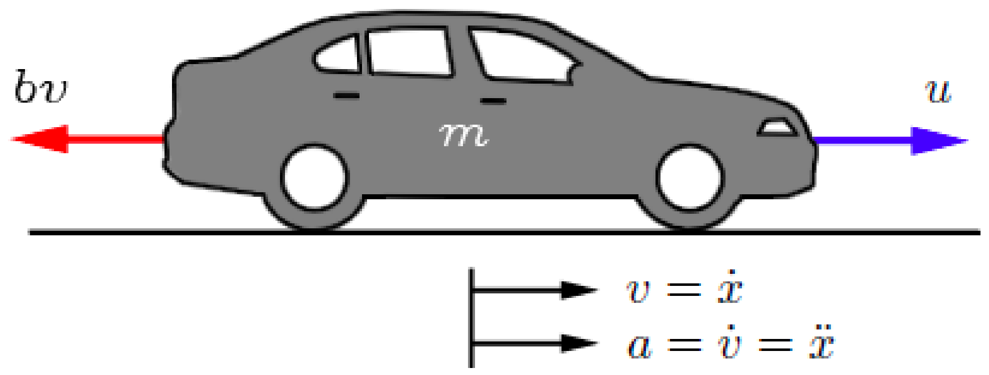
- Summing forces in the x-direction and applying Newton's 2nd law, we arrive at the following system equation:

$$m\dot{v} + bv = u$$

- Since we are interested in controlling the speed of the vehicle, the output equation is chosen as follows :

$$y = v$$

SIMULINK



- Using Newton's 2nd law, the governing equation for this system becomes:

$$m\dot{v} = u - bv$$

- where u is the force generated between the road/tire interface and can be controlled directly. For this example, let's assume that

```
m = 1000 kg

b = 50 N.sec/m

u = 500 N
```

SIMULINK

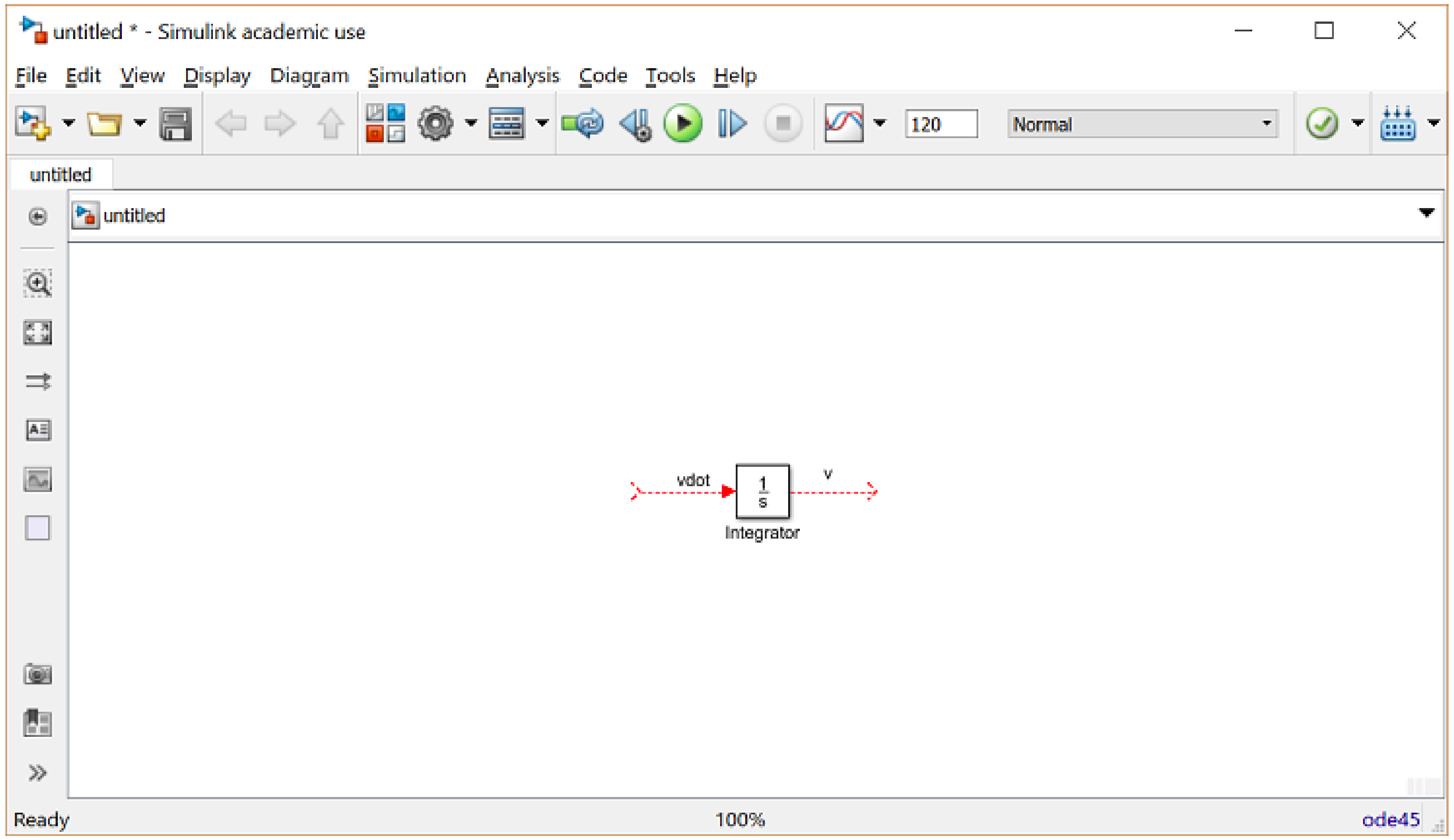
- This system will be modeled by summing the forces acting on the mass and integrating the acceleration to give the velocity. Open Simulink and open a new model window. First, we will model the integral of acceleration.

$$\int \frac{dv}{dt}\, dt = v$$

- Insert an Integrator block (from the Continuous library) and draw lines to and from its input and output terminals.

SIMULINK

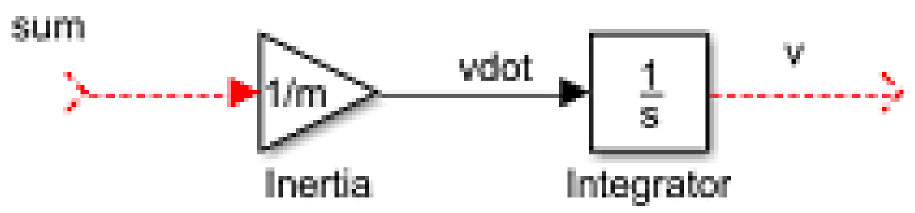
- Label the input line "vdot" and the output line "v" as shown below. To add such a label, double click in the empty space just above the line.


untitled * - Simulink academic use
File Edit View Display Diagram Simulation Analysis Code Tools Help
120
Normal
untitled
vdot
1/s
v
Integrator
Ready
100%
ode45

- Since the acceleration (dv/dt) is equal to the sum of the forces divided by mass, we will divide the incoming signal by the mass.

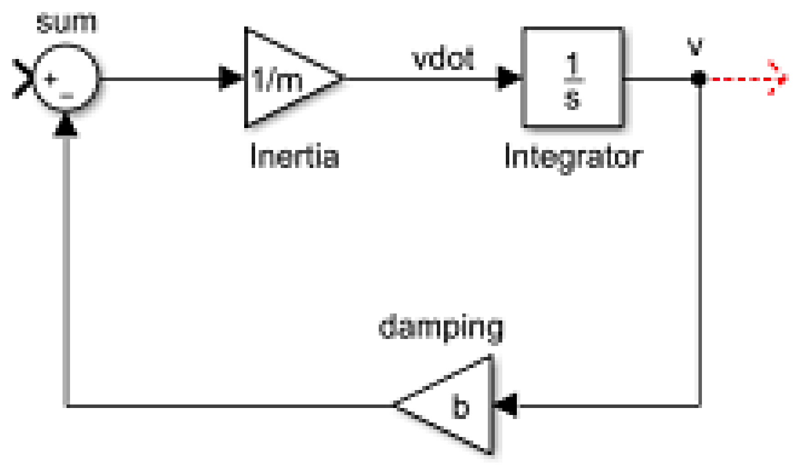
SIMULINK

- Insert a Gain block (from the Math Operations library) connected to the Integrator block input line and draw a line leading to the input of the Gain block.
- Edit the Gain block by double-clicking on it and change its value to "1/m".
- Change the label of the Gain block to "inertia" by clicking on the word "Gain" underneath the block.

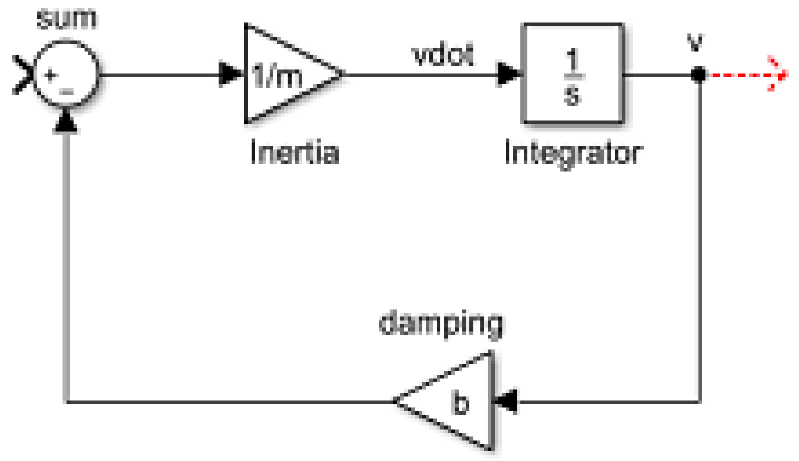


SIMULINK

- Attach a Sum block (from the Math Operations library) to the line leading to the inertia Gain block.
- Change the signs of the Sum block to “+-".
- Insert a Gain block below the Inertia block, select it by single-clicking on it, and select **Flip Block** from the **Rotate & Flip** menu (or type **Ctrl-I**) to flip it left-to-right.
- Set the block's value to "b" and rename this block to “damping”.
- Tap a line (hold **Ctrl** while drawing) off the Integrator block's output and connect it to the input of the damping Gain block.
- Draw a line from the damping Gain block output to the negative input of the Sum Block.

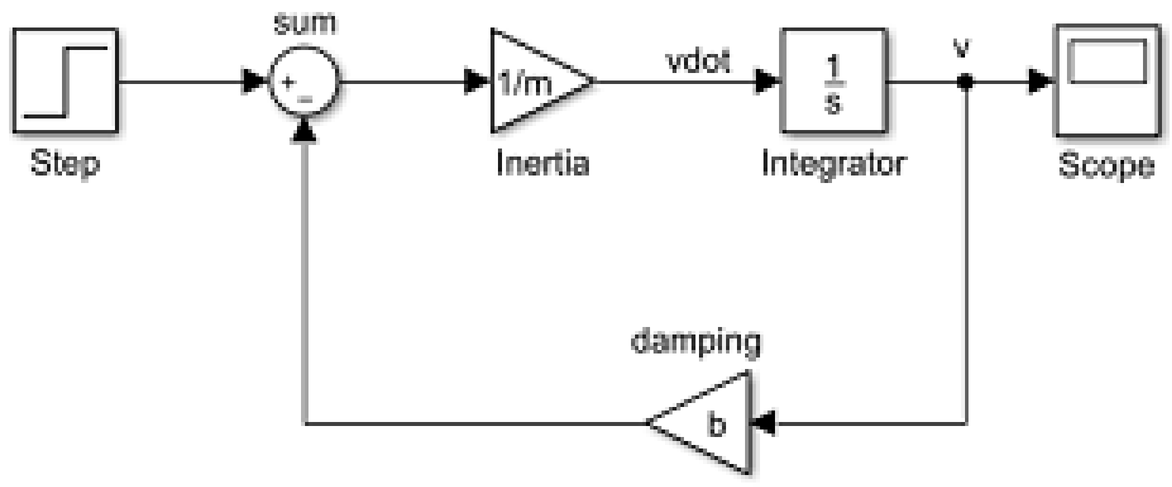


- Now, we will add in the forces which are represented in Equation . First, we will add in the damping force.Attach a Sum block (from the Math Operations library) to the line leading to the inertia Gain block.
- Change the signs of the Sum block to “+-".
- Insert a Gain block below the Inertia block, select it by single-clicking on it, and select **Flip Block** from the **Rotate & Flip** menu (or type **Ctrl-I**) to flip it left-to-right.
- Set the block's value to "b" and rename this block to “damping”.
- Tap a line (hold **Ctrl** while drawing) off the Integrator block's output and connect it to the input of the damping Gain block.
- Draw a line from the damping Gain block output to the negative input of the Sum Block.



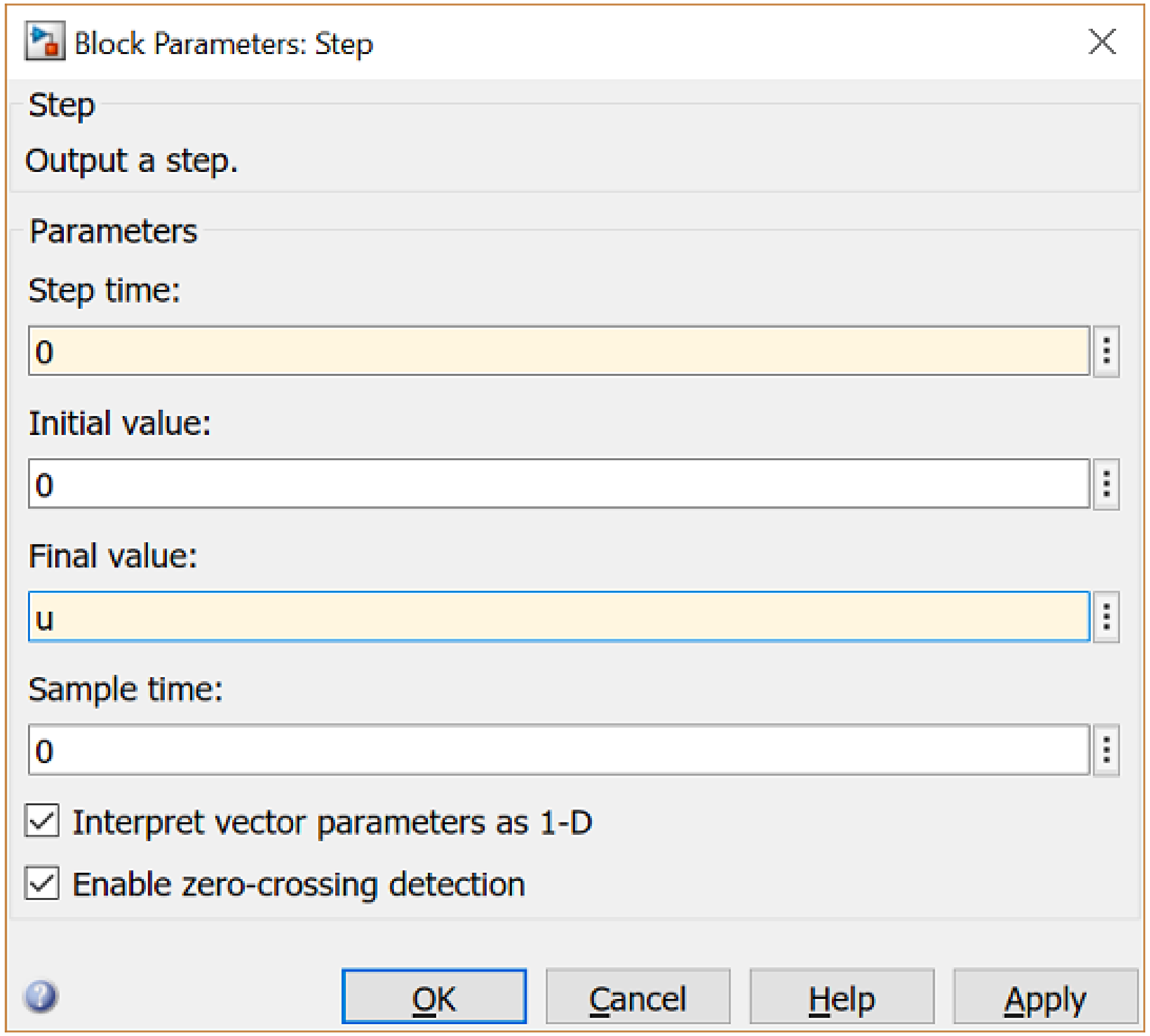
SIMULINK

- The second force acting on the mass is the control input, u. We will apply a step input.
- Insert a Step block (from the Sources library) and connect it with a line to the positive input of the Sum Block.
- To view the output velocity, insert a Scope block (from the Sinks library) connected to the output of the Integrator.



SIMULINK

- To provide an appropriate step input of 500 at time equals zero, double-click the Step block and set the Step Time to "0" and the Final Value to "u".



Block Parameters: Step

Step

Output a step.

Parameters

Step time:

0

Initial value:

0

Final value:

u

Sample time:

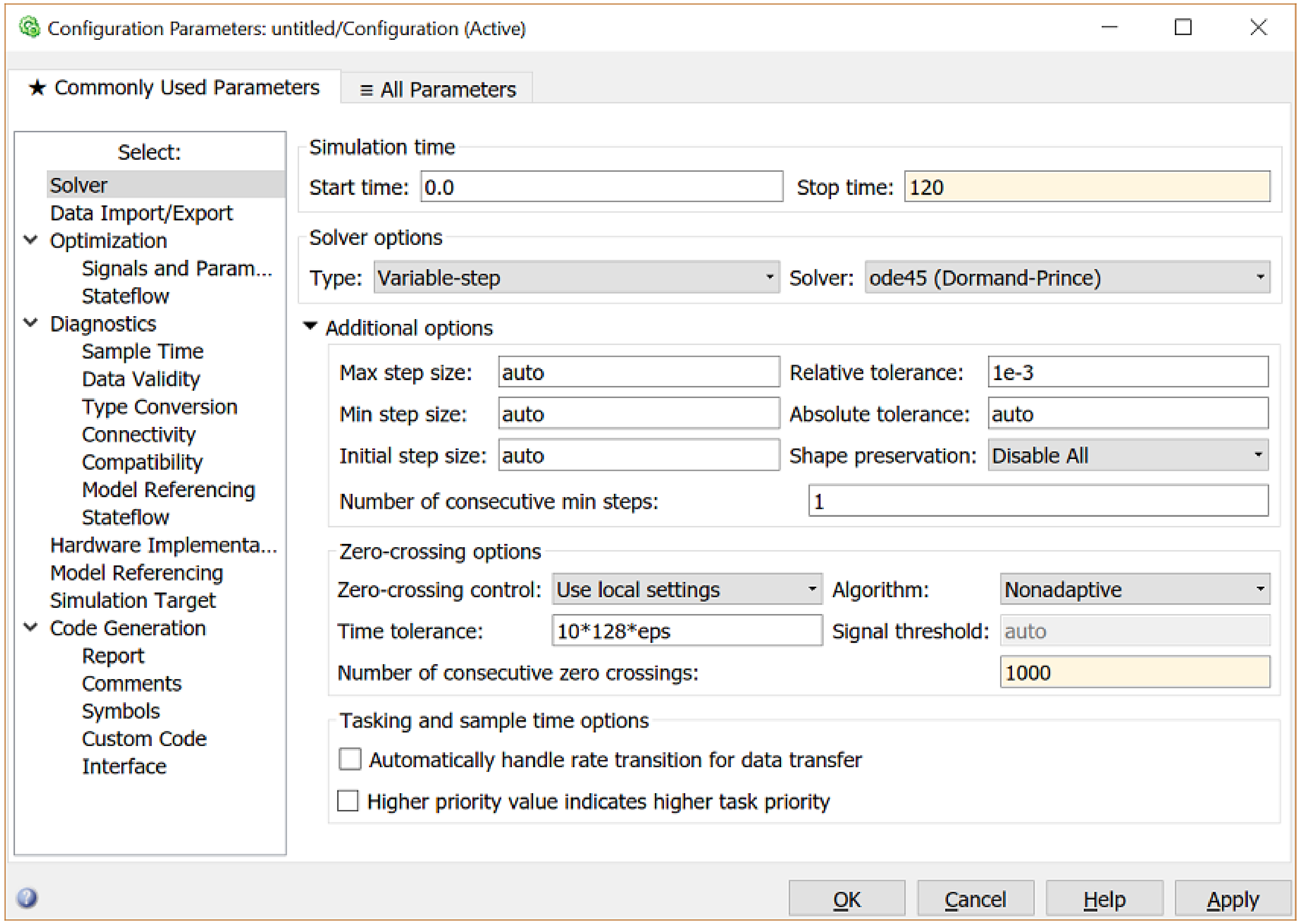
0

☑ Interpret vector parameters as 1-D

☑ Enable zero-crossing detection

OPEN-LOOP RESPONSE

- To simulate this system, first, an appropriate simulation time must be set.
- Select Parameters from the Simulation menu and enter "120" in the Stop Time field. 120 seconds is long enough to view the open-loop response.



Configuration Parameters: untitled/Configuration (Active)

★ Commonly Used Parameters | ≡ All Parameters

Select:
- Solver
- Data Import/Export
- Optimization
  - Signals and Param...
  - Stateflow
- Diagnostics
  - Sample Time
  - Data Validity
  - Type Conversion
  - Connectivity
  - Compatibility
  - Model Referencing
  - Stateflow
- Hardware Implementa...
- Model Referencing
- Simulation Target
- Code Generation
  - Report
  - Comments
  - Symbols
  - Custom Code
  - Interface

Simulation time

Start time: 0.0 Stop time: 120

Solver options

Type: Variable-step Solver: ode45 (Dormand-Prince)

Additional options

Max step size: auto Relative tolerance: 1e-3

Min step size: auto Absolute tolerance: auto

Initial step size: auto Shape preservation: Disable All

Number of consecutive min steps: 1

Zero-crossing options

Zero-crossing control: Use local settings Algorithm: Nonadaptive

Time tolerance: 10*128*eps Signal threshold: auto

Number of consecutive zero crossings: 1000

Tasking and sample time options

☐ Automatically handle rate transition for data transfer

☐ Higher priority value indicates higher task priority

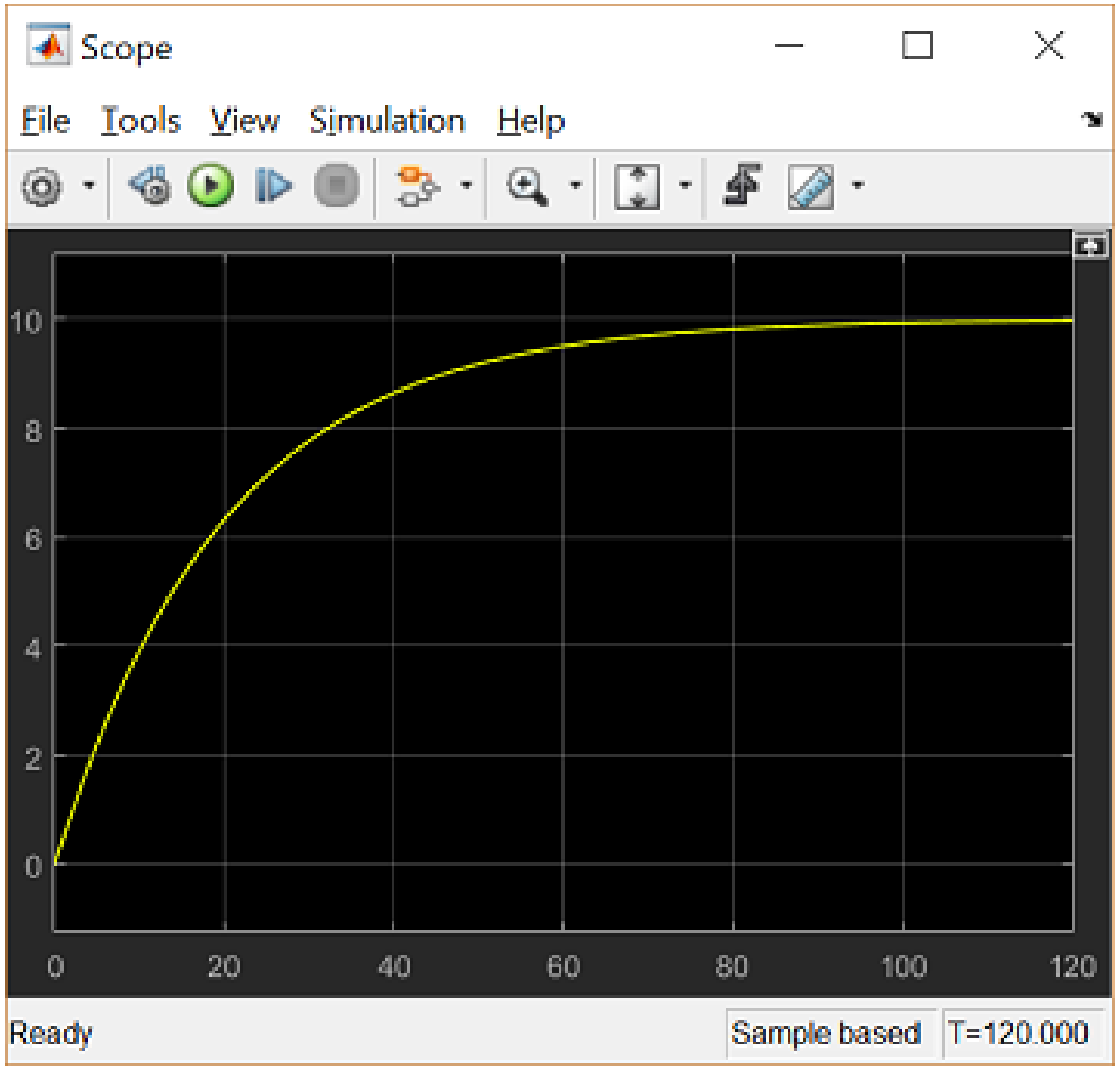
OPEN-LOOP RESPONSE

- The physical parameters must now be set. Run the following commands at the MATLAB prompt:

```
m = 1000;

b = 50;

u = 500;
```

- Run the simulation (hit **Ctrl-T** or select **Run** from the **Simulation** menu). When the simulation is finished you should see the following output.

OPEN-LOOP RESPONSE



Ready

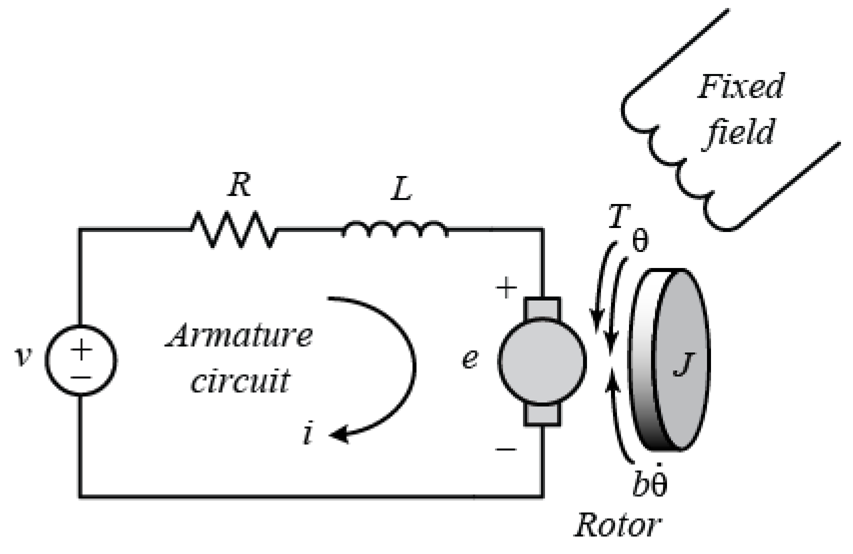
Sample based

T=120.000

HOMEWORK 4.3

- Build the given DC Motor with Simulink using the physical parameters given below :

(J)	moment of inertia of the rotor	0.01 kg.m^2
(b)	motor viscous friction constant	0.1 N.m.s
(Ke)	electromotive force constant	0.01 V/rad/sec
(Kt)	motor torque constant	0.01 N.m/Amp
(R)	electric resistance	1 Ohm
(L)	electric inductance	0.5 H


Fixed field
R
L
T
θ
+
v
Armature circuit
e
J
i
−
$b\dot{\theta}$
Rotor