# LAB-STA-2. MATLAB AND SIMULINK BASICS. MODELING AND SIMULATION OF SYSTEMS. CONNECTIONS OF SYSTEMS

**A. OBJECTIVES.** 1. To familiarize with Matlab and Simulink basics.
2. To model and simulate systems under Matlab/Simulink environment.

## B. MATLAB basics [1].

Matlab is an **interactive program for scientific and engineering calculations** and also a **programming language**. There is a base program and a collection of scripts and toolboxes are also available. The functionalities of the Matlab environment make it suitable for modeling, simulation, advanced calculations and control systems design. The basic objects that you can interact with in Matlab are:

- statements and variables,
- matrices,
- graphics and
- scripts.

After starting Matlab, you shoule be able to see the command prompt marked as “>>”.

### 1. Statements and variables

Matlab uses the assignment character “=” to assign an expression to an variable such as in:

```
>>variable=expression
```

For example, a statement where we enter a 2X2 matrix has the form:

```
>>A=[1 2;3 4] <ret>

A=
   1     2
   3     4
```

Please note that Matlab is case sensiitve. After a statement is entered, the carriage return has to be pressed. The semicolon (;) can be used as in the previous expression to separate the lines in the matrix or to suppress the output.

The typical mathematical operators used in Matlab are: **addition “+”, subtraction “-”, multiplication “*”, division “/” and power “^”.** These operators can be overloaded for objects of different types (as can be seen later).

Several predefined variables are *pi*, *Inf*, *NaN*, *i* and *j*. *NaN* stands for a not-a-number and results from udefined operations. *Inf* stand for infinity $+\infty$, while *pi* stands for $\pi$. Variables $i=\sqrt{-1}$ and *j* are used to represent complex numbers. The command “**who**” shows the variables that are stored in the workspace.

```
>> who

Your variables are:

A    ans
```

The command “whos” gives additional information about the workspace variables:

```
>> whos
  Name      Size                Bytes  Class     Attributes

  A         2x2                    32  double
  ans       1x1                     8  double
```

Some common mathematical functions used in Matlab are described in Table 1.

Table 1. Common math functions used in Matlab

| | |
|---|---|
| sin(x) | Sine of the element of x |
| cos(x) | Cosine of the elements of x |
| asin(x) | Arcsine of the elements of x |
| acos(x) | Arccosine of the elements of x |
| tan(x) | Tangent of the elements of x |
| atan(x) | Arctangent of the elements of x |
| atan2(x,y) | Fourquadrant arctangent of the real elements of x and y |
| abs(x) | Absolute value of the elements of x |
| sqrt(x) | Square root of the elements of x |
| imag(x) | Imaginary part of the elements of x |
| real(x) | Real part of the elements of x |
| conj(x) | Complex conjugate of the elements of x |
| log(x) | Natural logarithm of the elements of x |
| log10(x) | Base-10 logarithm of the elements of s |
| exp(x) | Exponential of the elements of x |

The implicit display format of the numbers in Matlab is in 5 digit fixed point. Other formats are 15 digit fixed point, 5 digit floating point and 15 digit floating point as shown in the code sample below:

```
>> pi

ans =

    3.1416

>> format long; pi

ans =

   3.141592653589793

>> format short e;pi

ans =

   3.1416e+00

>> format long e;pi
```

```
ans =

     3.141592653589793e+00
```

The display format **does not** affect the computations which are carried out in double precision.

## 2. Matrix operations

- memory for matrices is allocated automatically, so it does not have to be pre-specified;
- the basic math operators are overloaded for matrices. Division is a special case of matrix multiplication in such that the right division A/B means A*inv(B), with inv denoting the matrix inverse. The left division A\B means inv(A)*B.
- **it is very important that all matrix operations are dimensionally compatible**.
- the array operations (vectors of matrices) are also preserved for element by element calculation.
- the matrix transpose operation is achieved using the apostrophe character " ' "

For array operations, the element-by-element calculation is possible using the corresponding math operators preceded by the dot character ".". For example two column vectors **x** and **y** of the same number of elements can be multiplied element by element using "**x.*y**".
The colon notation is an important tool in Matlab which allows us to generate a row/column vectors containing the numbers starting with a given value $x_i$ to a final value $x_f$ with increment dx. The next code sample illustrates the idea:

```
>>  x=[-1:0.2:1]';y=x.*sin(x);
>> [x y]

ans =

   -1.0000    0.8415
   -0.8000    0.5739
   -0.6000    0.3388
   -0.4000    0.1558
   -0.2000    0.0397
         0         0
    0.2000    0.0397
    0.4000    0.1558
    0.6000    0.3388
    0.8000    0.5739
    1.0000    0.8415
```

## 3. Graphics

Several commands to generate the plots are summarized in Table 2.

Table 2 Plot commands

| | |
|---|---|
| plot(x,y) | Plots the vector x versus the vector y |
| semilogx(x,y) | Plots the vector x versus the vector y while the x axis is $\log_{10}$ |
| semilogy(x,y) | Plots the vector x versus the vector y with the y axis in $\log_{10}$ |
| loglog(x,y) | Plots the vector x versus the vector y with $\log_{10}$ scales on both axes |

Several functions to customize the plots are illustrated in Table 3.

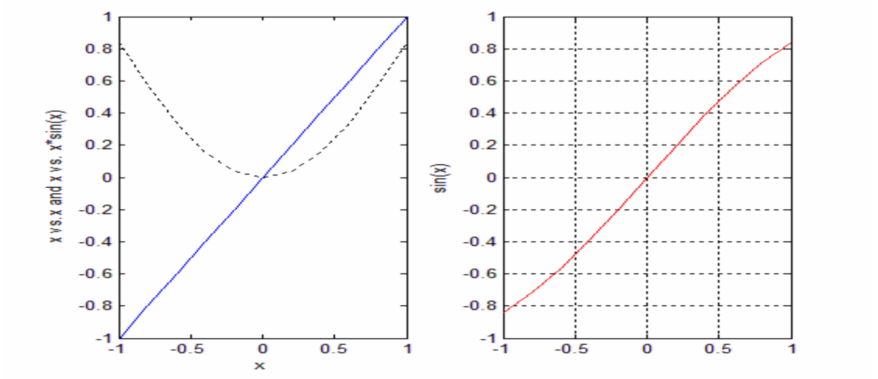
Table 3 Functions for customizing plots

| | |
|---|---|
| title('text') | Puts 'text' at the top of the plot |
| xlabel('text') | Labels the x axis with 'text' |
| ylabel('text') | Labels the y axis with 'text' |
| text(p1,p2,'text','sc') | Displays 'text' at (pi,p2) screen coordinates with (0.0,0.0) being the lower left and (1.0,1.0) the upper right of the screen |
| subplot | Divides the graphics window |
| grid | Adds grid lines to the current plot |

The most often used command is the "**plot**" command. Multiple arguments can be used for each one of the above commands. The "subplot" command is also used for creating subgraphs. The next code sample

```
>>subplot(121),plot(x,x,'b-',x,y,'k:'),xlabel('x'),ylabel('x
vs.x and x vs. x*sin(x)'),
>>subplot(122),plot(x,sin(x),'r'),ylabel('sin(x)'),grid
```

will produce the following graphics:



The command "subplot(*mnp*)" divides the plot area in *m* x *n* subplots while the positive integer "*p*" selects the subplot. Various markers can be used for graphics specified by "cm" where "c" denotes the color and "m" denotes the marker.

## 4. Scripts

Scripts (or m-files) represent a collections of commands in a single file and are usually created using the m-file editor available under Matlab. Scripts can call other scripts. The scripts that implement a function should have the file name identical with the function name. However, multiple functions can be declared inside scripts. The scripts are well documented using the symbol “**%**”. The commented text appears usually in green color.

It must be noted that since Matlab is also a programming language, complex commands can be implemented using decision statements such as “**if-then-else**”, “**for**” and “**while-do**” and so on.

A function example is shown below:

```
function [mean,stdev] = stat(x)
%STAT Interesting statistics.
n = length(x);
mean = avg(x,n);
stdev = sqrt(sum((x-avg(x,n)).^2)/n);

%-------------------------
function mean = avg(x,n)
%AVG subfunction
mean = sum(x)/n;
```

In a script the variables have a global scope and visibility. In a function the variable has a local visibility and only exists when the function is executed. In a function file, another function called a **subfunction** can be defined. A subfunction is not visible outside the function and cannot be invoked in Matlab.

## C. System Modeling and Simulation in Matlab.

### 1. A mass-spring-damper system

The free body diagram of the system is shown in Fig. 1 [2].

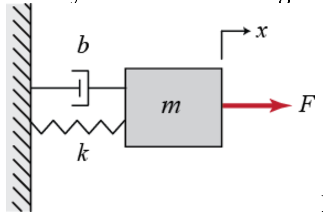


Fig. 1.

The spring force is proportional to the displacement of the mass, let it be $x$, and the viscous damping force is proportional to the velocity of the mass, namely $v = \dot{x}$. These two forces oppose the motion of the mass generated by the input force $F$. The position $x = 0$ correpsonds to unstretched spring. The forces are illustrated in Fig. 2.

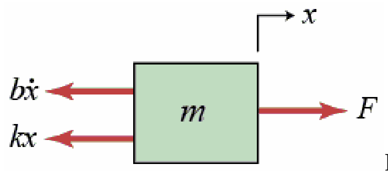


Fig. 2.

By applying the Newton’s second law and pointing out that the variables depend on time ($t$) we have (assuming zero initial displacement)

$$F(t) - b\,\dot{x}(t) - k\,x(t) = m\,\ddot{x}(t)\,, \qquad (1.1)$$

also known as the governing equation which completely describes the system’s dynamic behavior. The evolution of the dynamics are a function of the input force $F(t)$ and can be calculated in different ways. In order to write a state-space representation of the system, the second order differential equation must be transformed into a set of two first order differential equations. With position and velocity as state variables, the state-space form is

$$\dot{\mathbf{x}}(t) = \begin{pmatrix} \dot{x}(t) \\ \ddot{x}(t) \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ -k/m & -b/m \end{pmatrix} \begin{pmatrix} x(t) \\ \dot{x}(t) \end{pmatrix} + \begin{pmatrix} 0 \\ 1/m \end{pmatrix} F(t), \qquad (1.2)$$

$$y(t) = \begin{pmatrix} 1 & 0 \end{pmatrix} \begin{pmatrix} x(t) \\ \dot{x}(t) \end{pmatrix}$$

Remarks:

1) The state variables correspond to the potential energy in the spring and the kinetic energy of the mass respectively. It is usually helpful to look out for energy storage elements in the system in order to choose the state variables of the system and to deduce the order of the system.
2) In the output equation, the output is set as the mass displacement. If for the output we are interested in controlling the speed of the displaced mass, then the output matrix can change to (0  1).

**Creating the system model in Matlab**

In order to enter the equations described above, an m-file is used which looks like:

```
m = 1; %mass in kg
k = 1; %spring constant in N/m
b = 0.2; % damping constant in Ns/m
F = 1; %input force in N

A = [0 1; -k/m -b/m];
B = [0 1/m]';
C = [1 0];
D = [0];

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

Running the script will produce the following output:

```
sys =

  a =
         x1    x2
   x1     0     1
   x2    -1  -0.2

  b =
       u1
   x1   0
   x2   1

  c =
       x1  x2
   y1   1   0

  d =
       u1
   y1   0

Continuous-time state-space model.
```

The Laplace transform applied to the second order equation assuming zero initial conditions is:

$$m\,s^2\;X(s)+b\,s\;X(s)+k\;X(s)=F(s)\,, \tag{1.3}$$

redering the following transfer function

$$\frac{X(s)}{F(s)}=\frac{1}{m\,s^2+b\,s+k}\,. \tag{1.4}$$

In order to introduce the model as a transfer function there are two common ways of doing it. First, the symbolic variable “s” can be used for that as in the next m-file:

```
s=tf('s');
sys=1/(m*s^2+b*s+k);
```

Running this script will output:

```
sys =

         1
  ---------------
  s^2 + 0.2 s + 1

Continuous-time transfer function.
```

In order to interface with Simulink it may help to use another method to describe the transfer functions as in the script below:

```
num = [1];
den = [m b k];
sys = tf(num,den)
```

This script will have exactly the same output for the “*sys*” object as with the symbolic variable definition.

The input to the system (1.1), that is the force applied to the mass in this example, is also called the **forcing function**. The system dynamics subjected to zero forcing function and only to initial displacement x(0) is called the **unforced dynamic** response. The system response in all is formed from the unforced (free) dynamic response and from the forced dynamic response and this is something that we learned from solving ordinary differential equations. The differential equation in our case is (1.1).

For the transfer function (1.4), the roots of the numerator are called the **zeros** of the system while the roots of the denominator are the system’s **poles**. Setting the denominator polynomial to zero yields the **characteristic equation**. A Matlab function for finding the roots of a polynomial is “**roots**(v)”, where v is the vector containing the polynomial coefficients in descending powers of the polynomial variable. Given the roots of a polynomial, the polynomial can be recovered using the function “**poly**(r)” where r is the vector of roots. Polynomial multiplication resulting in another polynomial is performed using the **conv** function. Polynomial evaluation is done using **polyval**. For a complete reference on polynomial operations type “**help poly**” at the command prompt.

## 2. An electrical system

A simple electrical circuit is next considered for modeling. The circuit consists of three passive elements: a resistor, an inductor and a capacitor. This circuit is known as RLC circuit with the diagram presented in Fig. 3 [2].

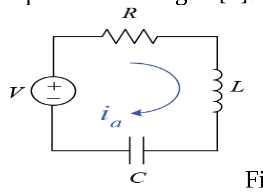


Fig. 3.

From Kirchoff’s voltage law and Kirchoff’s current law, the governing equation ca be written in the form

$$V(t)-L\,\frac{di(t)}{dt}-R\,i(t)-\frac{1}{C}\int i(t)dt=0\,. \tag{1.5}$$

An interesting analogy with the previous mechanical system can be shown. The system is of second order, the charge (integral of current) corresponds to displacement, the inductance to mass, the resistance to viscous damping and the inverse capacitance to the spring stiffness. This analogy is very powerful as it can be used in similar fashion to other systems and thus allows for a general framework of analysis. If we choose the charge and the current as state variables, the state-space representation of the system is

$$\dot{\mathbf{x}}(t) = \begin{pmatrix} i(t) \\ \dfrac{di(t)}{dt} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ -R/L & -1/(LC) \end{pmatrix} \begin{pmatrix} q(t) \\ i(t) \end{pmatrix} + \begin{pmatrix} 0 \\ 1/L \end{pmatrix} V(t), \tag{1.6}$$

$$y(t) = \begin{pmatrix} 0 & 1 \end{pmatrix} \begin{pmatrix} q(t) \\ i(t) \end{pmatrix}.$$

The transfer function can be calculated either by applying Laplace's transform to the governing equation (often assuming zero initial conditions) or by using the state-space to transfer function transformation formula:

$$\frac{I(s)}{V(s)} = \mathbf{C}(s\mathbf{I} - \mathbf{A})^{-1}\mathbf{B} + \mathbf{D} = \frac{s}{L\,s^2 + R\,s + 1/C}\,. \tag{1.7}$$

## 3. Converting between system representations

From a transfer function given, for example, as [3]:

```
num = [2 1];
den = [4 3 2];
G = tf(num,den)
```

which outputs

```
G =

      2 s + 1
  ---------------
  4 s^2 + 3 s + 2

Continuous-time transfer function.
```

a state-space model can be extracted using the *ssdata* command:

```
[A,B,C,D] = ssdata(G)
```

giving at the command prompt the following:

```
A =
   -0.7500   -0.5000
    1.0000         0
B =
     1
     0
C =
    0.5000    0.2500
D =
     0
```

This representation can be stored in an equivalent system variable H which basically offers a state-space representation:

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

which outputs:

```
H =

  a =
          x1     x2
   x1  -0.75   -0.5
   x2      1      0

  b =
       u1
   x1   1
   x2   0

  c =
         x1    x2
   y1   0.5  0.25

  d =
       u1
   y1   0

Continuous-time state-space model.
```

A zero-pole-gain model can be extracted from this system variable using the command:

```
[z,p,k] = zpkdata(H,'v')
```

```
z =
   -0.5000
p =
  -0.3750 + 0.5995i
  -0.3750 - 0.5995i
k =
    0.5000
```

The argument `v` forces a vectorized version return of the zeros and poles, useful for SISO systems. The zero-pole-gain representation can be again used for yet another representation in a new system variable `K`:

```
K = zpk(z,p,k)
K =

      0.5 (s+0.5)
  -------------------
  (s^2 + 0.75s + 0.5)

Continuous-time zero/pole/gain model.
```

The transfer function representation can be finally recovered using *tfdata* command:

```
[num,den] = tfdata(K,'v')
```

```
num =
         0    0.5000    0.2500
den =
    1.0000    0.7500    0.5000
```

### From state-space to transfer function

From a state-space representation given by the matrices **A**, **B**, **C** and **D**, a transfer function can be calculated by running the next script:

```
A = [0   1
     0   -0.05];

B = [0;
     0.001];

C = [0   1];

D = 0;

[num,den]=ss2tf(A,B,C,D)
```

which outputs

```
num =
   1.0e-03 *
         0    1.0000         0
den =
    1.0000    0.0500         0
```

Several remarks are useful at this point:

- The numerator *num* will have as many rows as there are outputs (or rows in C matrix).
- The num and den vectors return the coefficients of the corresponding numerator and denominator polynomials in the descending powers of *s*.
- Numerator and denominator must be checked **every time** since the zeroes at infinity may produce erroneous transfer functions.

A zero at infinity can be easily spotted in the next example where the numbers "-0.0000" and/or "0" in *num* are not quite zero but very small numbers:

```
num =
         0         0    1.8182   -0.0000  -44.5460
         0         0    4.5455   -7.4373   -0.0000
den =
    1.0000    0.1818  -31.1818    6.4786         0
```

These vey small numbers have to be replaced with true zeroes in order to eliminate the inconsistency.

To obtain again a state-space model from the transfer function, the *tf2ss* command is used.

Similarily, functions exist for converting from state-space to zero-pole-gain, or from transfer function to zero-pole-gain or vice-versa. The next table summarizes these functions.

| From \ To | state-space | transfer function | zero-pole-gain |
|---|---|---|---|
| state-space | | ss2tf(A,B,C,D) | ss2zp(A,B,C,D) |
| transfer function | tf2ss(num,den) | | tf2zp(num,den) |
| zero-pole-gain | zp2ss(z,p,k) | zp2tf(z,p,k) | |

***Remark:*** All the above functions and considerations are also valid for discrete-time systems.

For output arguments and help on each command type at the command prompt "*>>help function_name*".

## 4. Block Diagrams under Matlab

The basic systems connections under block diagram representation are the series connection, the parallel connection and the feedback connection depicted in Fig. 4 [2].

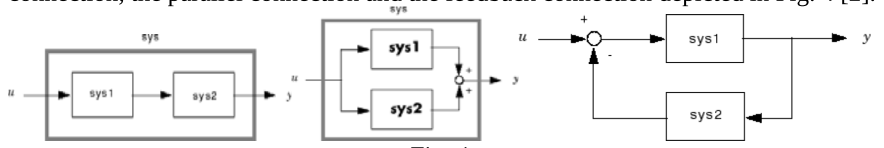


Fig. 4.

Assuming the transfer functions sys1(s)=num1(s)/den1(s) and sys2(s)=num2(s)/den2(s), the equivalent transfer function after reduction for each structure is:

sys(s)=y(s)/u(s)=sys1(s)*sys2(s) for series connection

sys(s)=y(s)/u(s)=sys1(s)+sys2(s) for parallel connection and

sys(s)=y(s)/u(s)=sys1(s) / [1 ± sys1(s)*sys2(s)], for feedback connection with "+" sign at the denominator if the feedback is negative and "–" if the feedback is positive.

If the polynomial coefficients of the numerators and denominators are given in Matlab as vectors for num1, num2, den1, den2, then in Matlab the following code can be used to calculate the connections:

```
>> sys1=tf(num1,den1);sys2=tf(num2,den2);
>> series(sys1,sys2)

ans =

          1
  -----------------
  s^3 + 3 s^2 + 2 s

Continuous-time transfer function.

>> parallel(sys1,sys2)

ans =

    s^2 + 3 s + 1
  -----------------
  s^3 + 3 s^2 + 2 s

Continuous-time transfer function.
>> feedback(sys1,sys2,-1)

ans =

        s^2 + 2 s
  ---------------------
  s^3 + 3 s^2 + 2 s + 1

Continuous-time transfer function.
```

***Remark:*** The feedback connection is extremely often encountered in control systems design since it tipically represents the connection between the controlled plant (sys2) and the controller (sys1).
As mentioned above, the three connections are the basic ones but topologies for more complicated diagrams can occur in practice. Several manipulations that are very useful are shown in a graphical manner in Fig. 5 [4], where the sign "≡" indicates equivalence.

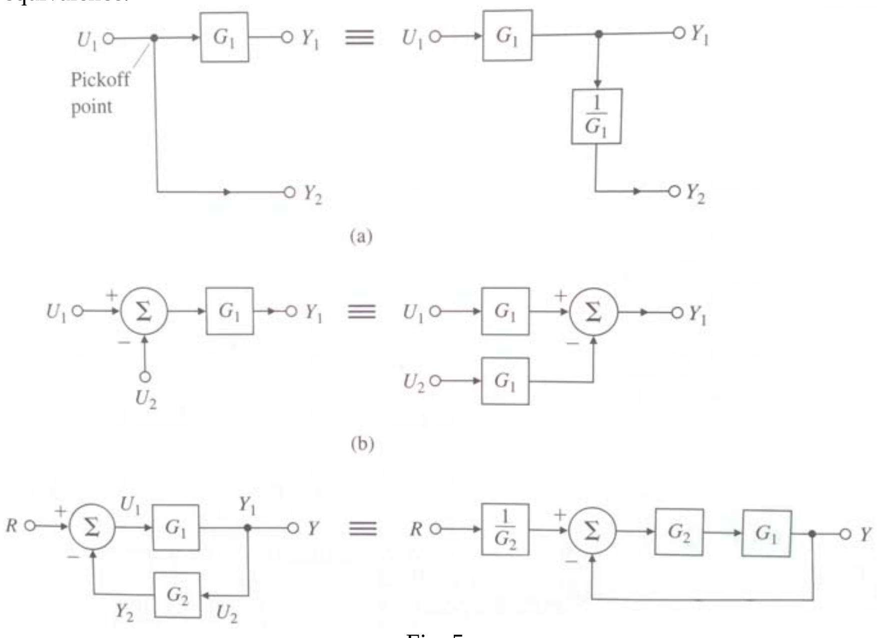


Fig. 5.

In case (a), the transfer function block $G_1$ can be moved before the pick-up point to the branches coming from $U_1$ and $Y_2$ in the form $G_1$ and $1/G_1$, respectively. In (b), the block $G_1$ can be moved before the summation point to both $U_1$ and $U_2$. In the feedback connection (c), $G_2$ can also be moved as shown in the picture without affecting the mathematical relations between the input $R$ and the output $Y$. These manipulations are very useful for block diagram reduction (simplification).

**D. System Modeling and Simulation in Simulink.**

For a short introductory material it is mandatory to first consult the appendix material **Simulink.pdf**.

**Example 1:** For the mass-spring-damper system previously modeled using Matlab, Fig. 6 shows three possible implementations of the model in Simulink.

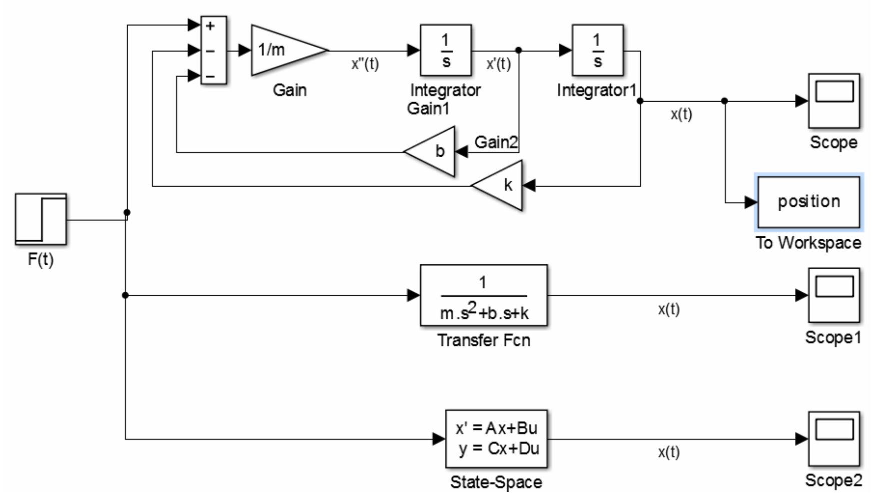


Fig. 6.

The implementations in Fig. 6 are (in top-down order): explicit mathematical equations form (EME), transfer function form (TF), state-space form (SS). The EME form implements the equation (1.1), the TF form implements equation (1.4) and the SS form implements equation (1.2). The variables, $m$, $b$, $k$ are defined in Matlab Workspace and Simulink knows where to look after them. Once again, when defining parameters, the user should pay attention to the fact that Matlab is case sensitive. The behavior of the three models in the same Simulink "*.mdl*" file is virtually identical. The position $x(t)$ of the mechanical system is regarded as the output and the mechanical force $F(t)$ is regarded as the input, both of them being labeled in Fig. 6. The "To-workspace"-type block labeled as "position" sends data back to Matlab workspace for post-processing.

The EME form is the most general since it allows for the description of nonlinear systems. The other two forms can describe only the behavior of linear systems.

Combinations of the three forms can be used without any problems. There is also another way of describing complex behavior of systems using the so-called **S-functions,** but they will not be treated in this lab.

**Remark 1**. The SS form and EME form allow for setting the initial conditions other than zero. (in vector notation for SS and individually for each integrator for EME) Zero initial conditions are assumed for the TF form.

**Remark 2**. The "Scope" blocks are very useful for visualizing the simulation results immediately after simulation. Different kinds of scope blocks are available under Simulink. Another somehow more elaborate method for displaying the results requiring an additional coding effort is to send the signals to Matlab workspace (using for example the "To-workspace" blocks) and then to plot the results using "plot" command.

**Homework:** Implement the electrical system from Section C.2 in the three forms shown for the mechanical system. Simulate the model/system and observe the current $i(t)$ response over time for various shapes of the input $v(t)$ such as impulse, step, ramp and sinewave. The parameters of the model can be chosen as R=1 kΩ, C=1 mF, L=1H.

Many models (e.g., the models presented in the first lab) can be simulated using Simulink. The following homework is proposed in this context:
**Homework 2**: Implement and simulate the student dynamic model using Matlab/Simulink. In Matlab, this can be done by implementeing an iterated function script. In Simulink the implementation is graphically, using blocks. In each case, use plots to display the time evolution of the number of freshmen and graduate students.

## References

[1] R.H. Bishop, *Modern Control Systems Analysis and Design Using Matlab*, Addison-Wesley, Boston, MA, USA, 1993.
[2] http://ctms.engin.umich.edu/CTMS/index.php?example=Introduction&section=SystemModeling
[3] http://ctms.engin.umich.edu/CTMS/index.php?aux=Extras_Conversions
[4] G.F. Franklin, J.D. Powell, A. Emami-Naeini, *Feedback Control of Dynamic Systems, 5th Edition*, Prentice Hall, Upper Saddle River, NJ, USA, 2009.