Matlab Laboratory 2. Mathematical operations in Matlab

Aim of the laboratory

- Understanding the Matlab computation engine
- Perform some mathematical operations in Matlab

Necessary equipment

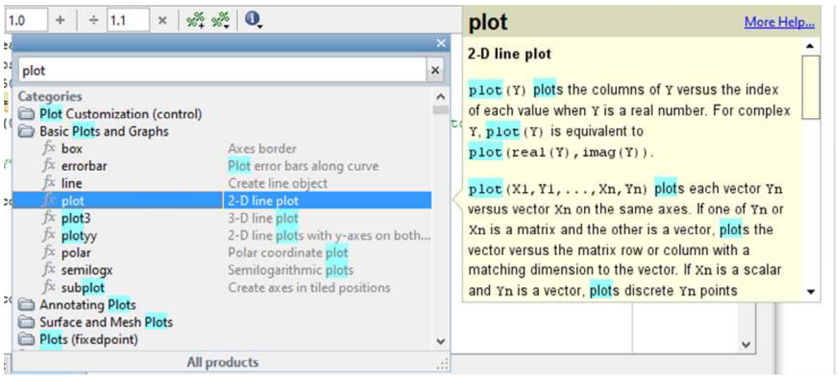
- Workstations with MATLAB

Theoretical Approach

Matlab is a very powerful computation engine that is developed around a set of mathematical operations and functions. The Matlab computation resources and graphical representation features can mainly be classified into [1]:

- Elementary mathematical operations,
- Linear algebra,
- Data analysis,
- Non-linear numerical analysis,
- Programming,
- 2D and 3D graphical representation,
- Graphical user interface,
- Support for printers,
- Data exchange.

Matlab provides a very powerful help tool. To look up specific functions in the Matlab, open the function browser by going ***Help → Function browser***. For exemplification, figure 1 illustrates looking up the key word ***plot***. Type "plot" into the top editor of the function browser. This inquiry will list all categories related to the function ***plot***. Further on, let's select "Basic Plots and Graphs" which lists all functions related to plotting signals. Holding the mouse cursor over one function, e.g ***plot***, will show the help related to the ***plot*** function.



Plot Customization (control)

Basic Plots and Graphs

box	Axes border
errorbar	Plot error bars along curve
line	Create line object
plot	2-D line plot
plot3	3-D line plot
plotyy	2-D line plots with y-axes on both...
polar	Polar coordinate plot
semilogx	Semilogarithmic plots
subplot	Create axes in tiled positions

Annotating Plots

Surface and Mesh Plots

Plots (fixedpoint)

plot

2-D line plot

`plot(Y)` plots the columns of `Y` versus the index of each value when `Y` is a real number. For complex `Y`, `plot(Y)` is equivalent to `plot(real(Y),imag(Y))`.

`plot(X1,Y1,...,Xn,Yn)` plots each vector `Yn` versus vector `Xn` on the same axes. If one of `Yn` or `Xn` is a matrix and the other is a vector, plots the vector versus the matrix row or column with a matching dimension to the vector. If `Xn` is a scalar and `Yn` is a vector, plots discrete `Yn` points

Figure 1

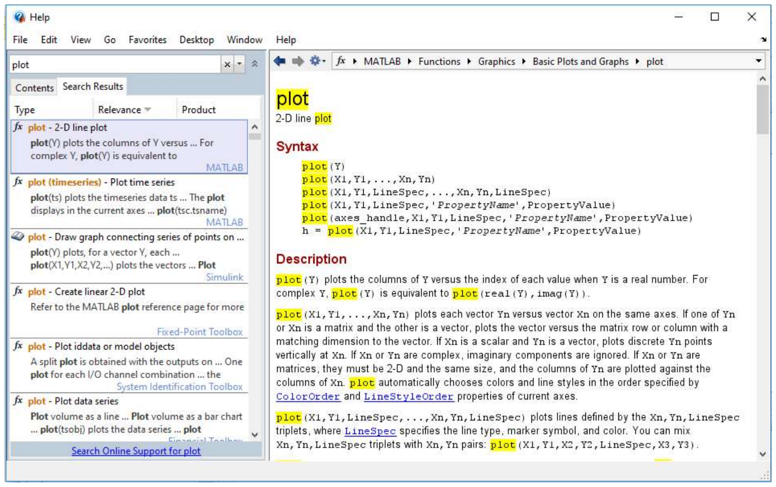
Next, double clicking the selected function form the browser will either insert the function into the command line or into the document editor respectively.

Help regarding a function can also be interrogated by typing help in the command line:

```
help [function];
```

where [function] is the name of the function being consulted.

At this stage, the preferable approach for consulting the Matlab help is the Matlab product help available at ***Help → Product help***. For exemplification, figure 2 shows the function ***plot*** in the Matlab product help. Type "plot" in the top-left editor of the helop window, and then select the desired plot function from the left of the window. As follows, information regarding the consulted function will be displayed in the right region of the help window.


Help
File Edit View Go Favorites Desktop Window Help
plot
Contents Search Results
Type Relevance Product
MATLAB ▸ Functions ▸ Graphics ▸ Basic Plots and Graphs ▸ plot
Search Online Support for plot

plot - 2-D line plot
plot(Y) plots the columns of Y versus ... For complex Y, plot(Y) is equivalent to
MATLAB

plot (timeseries) - Plot time series
plot(ts) plots the timeseries data ts ... The plot displays in the current axes ... plot(tsc.tsname)
MATLAB

plot - Draw graph connecting series of points on ...
plot(Y) plots, for a vector Y, each ...
plot(X1,Y1,X2,Y2,...) plots the vectors ... Plot
Simulink

plot - Create linear 2-D plot
Refer to the MATLAB plot reference page for more
Fixed-Point Toolbox

plot - Plot iddata or model objects
A split plot is obtained with the outputs on ... One plot for each I/O channel combination ... the
System Identification Toolbox

plot - Plot data series
Plot volume as a line ... Plot volume as a bar chart ... plot(tsobj) plots the data series ... plot

plot

2-D line plot

Syntax

```
plot(Y)
plot(X1,Y1,...,Xn,Yn)
plot(X1,Y1,LineSpec,...,Xn,Yn,LineSpec)
plot(X1,Y1,LineSpec,'PropertyName',PropertyValue)
plot(axes_handle,X1,Y1,LineSpec,'PropertyName',PropertyValue)
h = plot(X1,Y1,LineSpec,'PropertyName',PropertyValue)
```

Description

`plot(Y)` plots the columns of `Y` versus the index of each value when `Y` is a real number. For complex `Y`, `plot(Y)` is equivalent to `plot(real(Y),imag(Y))`.

`plot(X1,Y1,...,Xn,Yn)` plots each vector `Yn` versus vector `Xn` on the same axes. If one of `Yn` or `Xn` is a matrix and the other is a vector, plots the vector versus the matrix row or column with a matching dimension to the vector. If `Xn` is a scalar and `Yn` is a vector, plots discrete `Yn` points vertically at `Xn`. If `Xn` or `Yn` are complex, imaginary components are ignored. If `Xn` or `Yn` are matrices, they must be 2-D and the same size, and the columns of `Yn` are plotted against the columns of `Xn`. `plot` automatically chooses colors and line styles in the order specified by `ColorOrder` and `LineStyleOrder` properties of current axes.

`plot(X1,Y1,LineSpec,...,Xn,Yn,LineSpec)` plots lines defined by the `Xn,Yn,LineSpec` triplets, where `LineSpec` specifies the line type, marker symbol, and color. You can mix `Xn,Yn,LineSpec` triplets with `Xn,Yn` pairs: `plot(X1,Y1,X2,Y2,LineSpec,X3,Y3)`.

Figure 2

An advantage of using the Matlab product help is that a set of related function are listed at the end of the help document, each of which can be consulted individually. These functions might turn out very useful in the development process.

Data in Matlab

The elementary data structure in Matlab is the array. An array is basically a multi-dimensional data structure. If the number of dimensions is restricted to 2, the data array is a matrix. The size of a matrix is considered *n* x *m*, where *n* is the number of rows and *m* is the number of columns of the matrix respectively. For ease of interpretation and visualization, this work will only considers matrices and the presentation can be extrapolated by the reader to multi-dimensional data.

Some classical matrices in Matlab, and their corresponding creation function, are:

- all ones, ***ones***,
- all zeros, ***zeros***,
- identity matrix, ***eye***,

- random matrix with unity distribution, ***rand***,
- random matrix with normal distribution, ***randn***.

These matrices are exemplified in table 1. The matrices are created by calling the corresponding matrix creation function. If the matrix creation function receives only one argument Matltab will create a square matrix of that size. If the matrix creation function receives two arguments, Matltab will create a matrix of the sizes indicated by the arguments. For further options regarding the creation of matrixes, refer to the Matlab help.

Table 1.

Function	echo	Function	echo

`ones(3)`	`ans =`
	
	`1  1  1`
	`1  1  1`
	`1  1  1`
		`ones(2,3)`	`ans =`
			
			`1  1  1`
			`1  1  1`

`zeros(3)`	`ans =`
	
	`0  0  0`
	`0  0  0`
	`0  0  0`
		`zeros(2,3)`	`ans =`
			
			`0  0  0`
			`0  0  0`

`eye(3)`	`ans =`
	
	`1  0  0`
	`0  1  0`
	`0  0  1`
		`n.a.`	

`rand(3)`	`ans =`
	
	`0.8147  0.9134  0.2785`
	`0.9058  0.6324  0.5469`
	`0.1270  0.0975  0.9575`
		`rand(2,3)`	`ans =`
			
			`0.7922  0.6557  0.8491`
			`0.9595  0.0357  0.9340`

`randn(3)`	`ans =`
	
	`2.7694  0.7254  -0.2050`
	`-1.3499  -0.0631  -0.1241`
	`3.0349  0.7147  1.4897`
		`randn(2,3)`	`ans =`
			
			`0.4889  0.7269  0.2939`
			`1.0347  -0.3034 -`
			`0.7873`

Assignation of a matrix to a variable is done with the operator "=". For exemplification, consider the assignation of two special matrices to variables A and B respectively:

- Pascal matrix, ***pascal***,
- Magic matrix, ***magic***.

Table 2.

Instruction	echo
`A = pascal (3)`	`A =`
	`1 1 1`
	`1 2 3`
	`1 3 6`
`B = magic (3)`	`B =`
	`8 1 6`
	`3 5 7`
	`4 9 2`

In contrast to other programming languages, Matlab doesn't have the vector defined as a specific data structure. In Matlab, a vector is regarded as a matrix which has one of the sizes equal to 1. For exemplification consider the vector assignments listed in table 3.

Table 3

Instruction	echo
`u = [3; 1; 4]`	`u =`
	`3`
	`1`
	`4`
`v = [2 0 -1]`	`v =`
	`2 0 -1`

Similarly, a scalar is also regarded as a matrix with both dimensions equal to 1. For exemplification, consider the assignment listed in table 4.

Table 4

Instruction	echo
`s = 5`	`s =`
	`5`

To be noticed in tables 1 – 4 is that, provided the functions and instructions are typed without semicolon, .i.e. the ”;” character, echo will be listed in the Matlab command window. To turn echo off, the instructions should be followed by semicolon.

Operations with matrices

The Matlab mathematical operations can be mainly classified into:

- operations with matrices and linear algebra,
- operations with polynomials and interpolations,
- operations with Fourier transforms,
- differential equations.

Mathematical operations on matrices are defined exactly as in matrix algebra. Accordingly, matrix **addition** and **subtraction** is interpreted elementwise, requiring that all operands have the same size. Consider for instance the following code sequence which illustrated the addition and subtraction of matrices with the same size.

```
A = pascal(3);
B = magic(3);
X = A + B
Y = A - B
```

The addition and subtraction results are as follows:

```
 X =

     9     2     7
     4     7    10
     5    12     8

Y =

    -7     0    -5
    -2    -3    -4
    -3    -6     4
```

Now consider the following code sequence which attempts to add two matrices of different dimensions.

```
A = pascal(3);
C = fix(10*rand(3,2));
X = A + C;
```

In this case, evaluation of the addition returns an error:

```
??? Error using ==> plus
Matrix dimensions must agree.
```

To be noticed is that the only exception to having matrices of the same dimensions is addition and subtraction with a scalar. In this case, the scalar is added to subtracted to/from every element of the matrix. Consider for exemplification

```
S = 5;
v = [2 0 -1]
z = s + v
```

which returns

```
ans =

     7     5     4
```

Multiplication of a matrix is performed as in matrix algebra. Accordingly, it requires the number of rows of the first operand to be equal to the number of columns of the second operand. For exemplification

```
A = [1 2 3;2 3 4];
B = [1 2; 3 4; 5 6];
C = A * B
```

returns

```
C =

    22    28
    31    40
```

A particular case of matrix multiplication refers to multiplying a row vector to a column vector. In this case, the following code

```
v = [2 0 -1];
u = [3; 1; 4];
w = v * u
```

returns a scalar

```
w =

    2
```

For elementwise multiplication of two matrices, which assumes same matrix dimensions as is the case for addition and subtraction respectively, one should use the point character, i.e. ".", before the multiplier. Accordingly,

```
A = pascal(3);
B = magic(3);
```

```
C = A .* B
```

returns

```
C =

     8     1     6
     3    10    21
     4    27    12
```

As was the case with addition and subtraction, multiplication of a scalar with a matrix is interpreted by having the scalar multiplied to each element of the matrix respectively.

Only matrices with real values have been considered so far. Nevertheless, the same rules stand for complex matrices as well when applying mathematical operators. One difference between real and complex matrices however stands in transposing. The transpose of a matrix *A* with real elements is realized by interchanging of the elements *a*[*i*,*j*] and *a*[*j*,*i*].

```
A = pascal(3);
A'

ans =

     1     1     1
     1     2     3
     1     3     6
```

The transpose transforms a row vector in a column vector and is performed as:

```
v = [2 0 -1];
v’

ans =

     2
     0
    -1
```

The transpose of a complex matrix on the other hand will actually act as a complex conjugated transposing. Accordongly:

```
z = [1+2i 3+4i]
z’
```

results in

```
ans =

   1.0000 - 2.0000i
   3.0000 - 4.0000i
```

If one only wants to change the row matrix into a column matrix, without having the conjugated values of the complex elements, the point character has to be employed.

```
z.'

ans =

   1.0000 + 2.0000i
   3.0000 + 4.0000i
```

Exercise 1. Determine the intermediate node voltage of the resistive divider illustrated in figure 3, considering a DC input voltage equal to 12 V and resistance values equal to R_1 = 10 kΩ and R_2 = 20 kΩ.

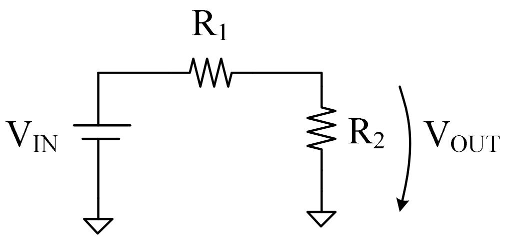


Figure 3.

The resistive divider law expresses the intermediate node voltage by

$$V_{OUT} = \frac{R_2}{R_1 + R_2} \cdot V_{IN}$$

The Matlab code to solve this task is listed as follows.

```
clear all;
close all;
clc;

R1 = 10e3;
R2 = 20e3;
Vin = 12;

Vout = (R2 / (R1 + R2)) * Vin
```

The resulting value of 8 V will be displayed in the command window.

To be noticed is that Matlab doesn't "understand" multiples and submultiples of units. In order to express kΩ, one must rather use exponentials. Accordingly, a 20 kΩ resistance is expressed by the value of 20 and the e3 multiple standing for kilo, that is 10^3.

To do…

1. Using the ***uicontrol*** function, extend the Matlab code in order to add control fields for the DC input voltage and the resistance values, and display the intermediate node voltage as a text element.
2. Repeat the exercise for a resistive current divider.

Exercise 2. Determine the intermediate node voltage of the circuit illustrated in figure 4.

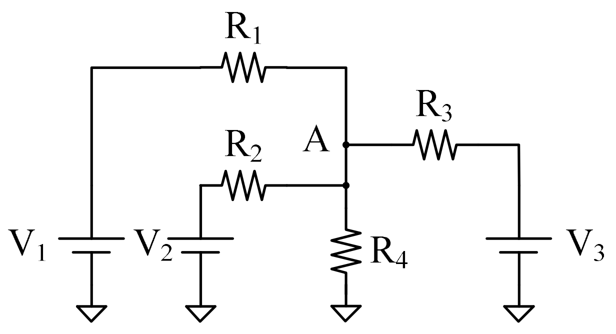


Figure 4.

The circuit is more complex than the simple resistive divider from the previous exercise. The intermediate node voltage, i.e. node A in figure 4, can be expressed using Millman's theorem. Accordingly, the node voltage in node A is equal to

$$V_A = \frac{\frac{V_1}{R_1} + \frac{V_2}{R_2} + \frac{V_3}{R_3} + \frac{0}{R_4}}{\frac{1}{R_1} + \frac{1}{R_2} + \frac{1}{R_3} + \frac{1}{R_4}}$$

To do…

1. Write a Matlab script to solve the task expressed in this exercise.
2. Using the ***uicontrol*** function, extend the Matlab code in order to add control fields for the DC input voltage and the resistance values, and display the intermediate node voltage as a text element.

Exercise 3. Determine the branch currents in the circuit illustrated in figure 5 using Kirchoff's law expressed in matrix form.

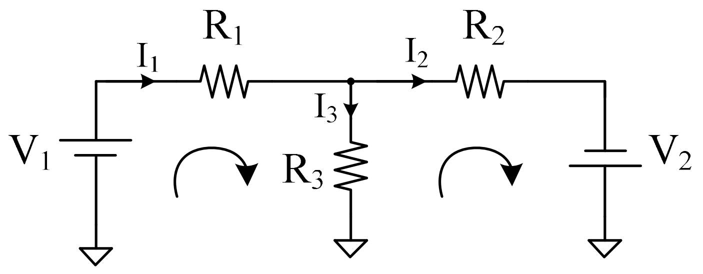


Figure 5

Kirchoff’s current law expresses that

$$I_3 = I_1 - I_2$$

Kirchoff’s voltage law expresses that

$$\begin{cases} V_1 = I_1 R_1 + (I_1 - I_2) R_3 \\ V_2 = (I_2 - I_1) R_3 + I_2 R_2 \end{cases}$$

This can be put into matrix form as

$$\begin{bmatrix} V_1 \\ V_2 \end{bmatrix} = \begin{bmatrix} R_1 + R_3 & -R_3 \\ -R_3 & R_2 + R_3 \end{bmatrix} \cdot \begin{bmatrix} I_1 \\ I_2 \end{bmatrix}$$

Let the voltage, resistance and current matrices be expressed as

$$V = \begin{bmatrix} V_1 \\ V_2 \end{bmatrix}$$

$$R = \begin{bmatrix} R_1 + R_3 & -R_3 \\ -R_3 & R_2 + R_3 \end{bmatrix}$$

$$I = \begin{bmatrix} I_1 \\ I_2 \end{bmatrix}$$

Kirchoff’s law in Matrix form is then expressed as

$$V = R \cdot I$$

The branch currents are then expressed as

$$I = R^{-1} \cdot V$$

The Matlab code to solve this task is listed as follows.

```
clear all;
close all;
clc;

V1 = 5;
V2 = 10;

V = [V1; V2];

R1 = 10e3;
R2 = 20e3;
R3 = 30e3;

R = [R1 + R3 -R3; -R3 R2 + R3];

I = R^-1 * V
```

To do…

1. Using the ***uicontrol*** function, extend the Matlab code in order to add control fields for the DC input voltage and the resistance values, and display branch currents as a text elements.

Exercise 4. Determine the branch currents in the circuit illustrated in figure 6 using Kirchoff's law expressed in matrix form.

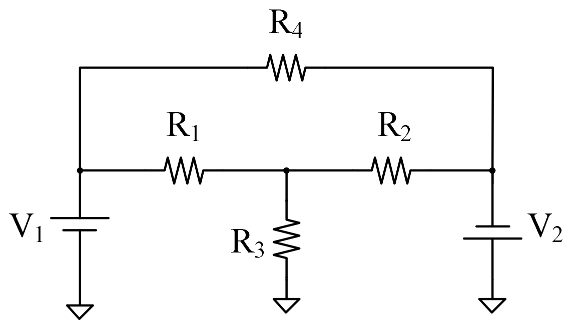


Figure 6

Exercise 5. Determine the branch currents in the circuit illustrated in figure 7 using Kirchoff's law expressed in matrix form.

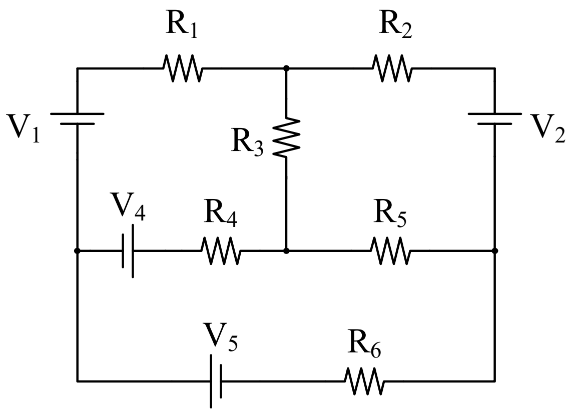


Figure 7

Exercise 6. Plot the input and the output voltage of the voltage divider illustrated in figure 6.

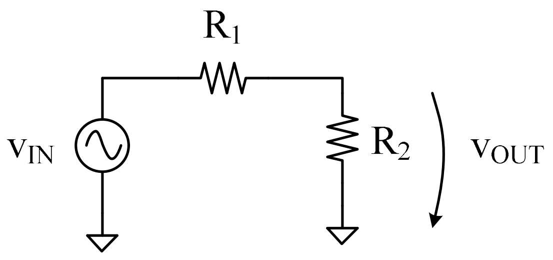


Figure 6.

Let the input voltage be generically expressed by

$$v_{IN} = A \cdot \sin(2\pi f t)$$

According to the resistive divider law, the output voltage is expressed by

$$v_{OUT} = \frac{R_2}{R_1 + R_2} \cdot v_{IN} = \frac{A \cdot R_2}{R_1 + R_2} \cdot \sin(2\pi f t)$$

For illustration, let the circuit parameter set be:

- A = 3 V,
- f = 50 Hz,
- R_1 = 10kΩ
- R_2 = 20kΩ

The Matlab code which plots the resistive divider input and output voltage is listed as follows.

```
clear all;
close all;
clc;

A = 3;              %amplitude
f = 50;             %frequency

per = 1/f;
t = 0:per/100:2*per;  %time domain

R1 = 10e3;
R2 = 20e3;

vin = A * sin(2 * pi * f * t);    %input singnal
vout = (R2 / (R1 + R2)) * vin;    %output signal

figure();                 %open a new plot window

subplot(2,1,1);           %top plot
plot(t,vin);
grid on;

subplot(2,1,2);           %bottom plot
plot(t,vout);
grid on;
```

The resistive divider input and output voltage are plotted in figure 7.

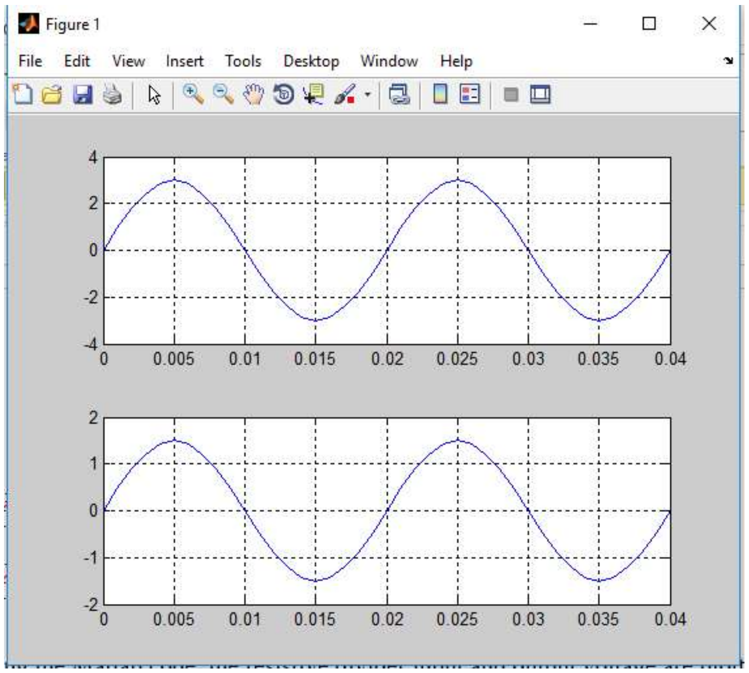


Figure 7.

To do…

1. Study the ***subplot*** function using the MATLAB help.
2. Using the ***uicontrol*** function, extend the Matlab code in order to add control fields for the input signal amplitude, frequency and phase respectively, as well as for the resistance values.

Exercise 7. Replace the DC voltage source V_1 from figure 4 with a sine voltage source. Plot the input and the node A voltage respectively. Add control fields for the input signal amplitude, frequency and phase respectively, as well as for the resistance values.

Exercise 8. Replace the DC voltage source V_1 from figures 5, 6 and 7 with a sine voltage source. Plot the input voltage and the branch currents respectively. Add control fields for the input signal amplitude, frequency and phase, as well as for the resistance values.