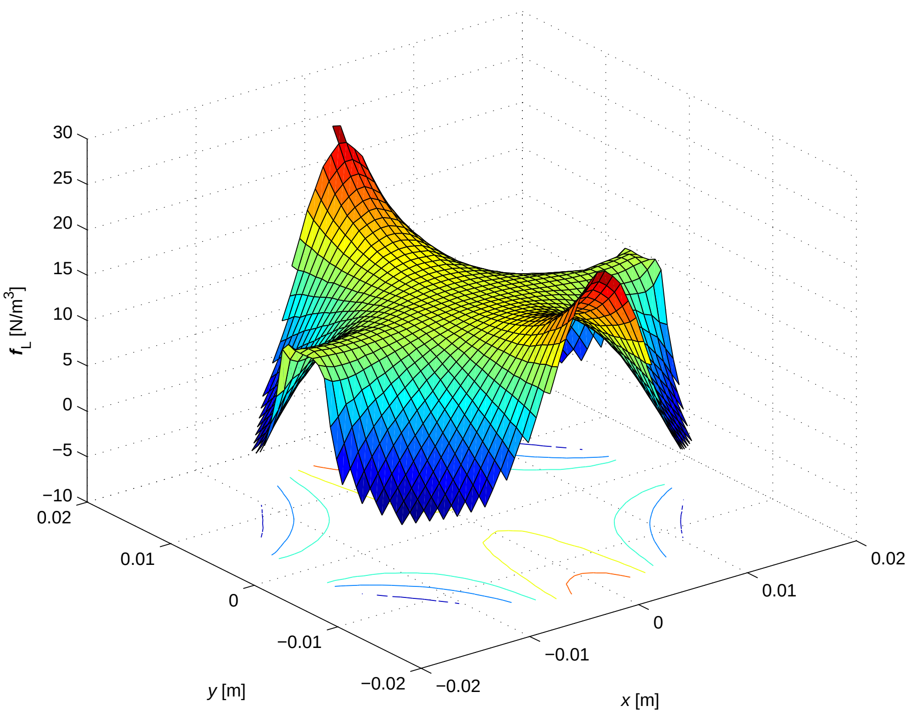
Petr Kropík
Lenka Šroubová
Pavel Štekl

Computer Support in Electrical Engineering



2012

Contents

1 Introduction to MATLAB **2**

1.1 Introduction to the technical computing . . . 2

1.2 Engineering tools – MATLAB and alternatives . . . 2

1.2.1 What is MATLAB? . . . 2

1.2.2 Alternative product . . . 2

1.3 MATLAB basics . . . 3

1.4 Syntax and operational principles . . . 7

1.5 MATLAB toolboxes . . . 8

1.6 Test of basic commands and standalone works with basic operators . . . 8

2 Matrix oriented commands, linear algebra **9**

2.1 Basic mathematical operators and types of variables . . . 9

2.1.1 Allocation . . . 9

2.1.2 Predefined constants . . . 10

2.1.3 Arithmetic operators . . . 10

2.1.4 Relational operations . . . 11

2.1.5 Logical operations . . . 12

2.2 Basic commands and basic operations instructions . . . 13

2.2.1 Formatting the numerical output (for visual form on the screen) . . . 13

2.2.2 Parentheses . . . 13

2.2.3 Separators . . . 13

2.2.4 Definition of vectors . . . 14

2.2.5 Definition of matrices . . . 17

2.3 Operations with vectors and matrices . . . 20

2.3.1 Basic matrix operations . . . 20

2.3.2 Indexing of matrix elements . . . 25

2.4 Matrix operations and tools . . . 30

2.4.1 Special types of matrices . . . 30

2.4.2 Arithmetical operations with matrices . . . 33

2.5 Vector operations and tools . . . 41

2.6 Comparison MATLAB functions with alternative products . . . 44

2.7 Example from Electrical Engineering . . . 44

2.7.1 The serial connection of resistors . . . 44

2.7.2 The parallel connection of resistors . . . 45

2.7.3 The series-parallel connection of resistors . . . 46

2.7.4 Ohm's law . . . 46

2.8 Test of matrix oriented commands . . . 47

3 Matrix operations in the technical praxis **48**

3.1 Solving the system of linear equations . . . 48

3.2 Advanced algebraic commands . . . 49

3.3 Example from Electrical Engineering . . . 51

3.3.1 The DC circuit - application of Kirchhoff's laws . . . 51

3.3.2 The DC circuit 2 - application of Kirchhoff's laws . . . 53

3.4 Test of matrix operations in the technical praxis . . . 55

4 Complex numbers and complex matrices 57
4.1 Introduction to complex mathematics in MATLAB 57
4.2 Usage of complex matrices 58
4.3 Example from Electrical Engineering 59
4.3.1 The AC circuit - application of Kirchhoff's laws 59
4.3.2 Graphical representation of the results - phasor diagram of currents . 60
4.4 Test of basic commands for complex mathematics 61

5 Graphs and graphical commands 62
5.1 Basic graphical commands and libraries 62
5.2 Further types of graphs 66
5.3 Settings the graph properties 77
5.4 Using graphical output in electrical engineering examples 79
5.4.1 Volt-ampere characteristic of linear resistor 79
5.4.2 Transfer function of electronic filter 80
5.4.3 The balanced tree-phase AC voltage source 81
5.4.4 The plotting of gradient 82
5.5 Test of graphical commands, settings and using graphs 83

6 Polynomial regressions and interpolations 85
6.1 Basic commands for solving polynomials 85
6.2 Polynomial functions and libraries 86
6.3 Graphical representation of polynomial expressions 90
6.4 Fitting functions and regression 92
6.4.1 Polynomial regression 92
6.4.2 Polynomial curve fitting 94
6.4.3 Spline technology 95
6.5 Examples of the polynomial regression applied on the electrical engineering problems 97
6.5.1 Volt-ampere characteristic of non-linear component 97
6.5.2 Curve fitting in 3D 97
6.6 Test of commands for solving polynomials and polynomial regressions 102

7 Flow control statements 103
7.1 Flow Control Statements 103
7.2 Test of flow control statements 107

8 Integration 108
8.1 Numerical integration 108
8.2 Analytical integration using Symbolic Math Toolbox 113

9 Advanced instructions for text input and output 115
9.1 Formatted printing 115
9.2 Request for user input 117
9.3 Formatted output into file with parameters 119
9.4 Formatted output into text strings 120
9.5 Predefined Dialog Boxes 120
9.6 Test of text input and output 122

10 Differential equation in the engineering praxis **124**
10.1 Basic terms and commands 124
10.2 Electrical engineering application 124
10.2.1 RC circuit in series 124
10.3 System of differential equations 126
10.4 Higher orders differential equations 127
10.5 Solver parameters settings 129
10.6 User-defined parameters 133
10.7 Non-numerical solving of ordinary differential equations 135
10.8 Test of differential equation 136

11 Differential equations – advanced part **137**
11.1 Differential equations in electrical engineering praxis 137
11.2 Test of using of the differential equations in technical praxis 141

12 Creating the comprehensive applications **142**
12.1 Parallel connection of *n* resistors 142
12.2 Quadratic equation solver with GUI 145
12.3 Test of creating the comprehensive applications 145

13 Conclusion and repetitions **147**
13.1 Final test 147
13.1.1 Final test solution 150
13.2 Conclusion 152

1 Introduction to MATLAB

1.1 Introduction to the technical computing

MATLAB is a high-level language and interactive environment for numerical computation, visualization, and programming. Using MATLAB, you can analyse data, develop algorithms, and create models and applications. The language, tools and built-in math functions give you possibilities to explore multiple approaches and reach a solution faster than with spreadsheets or traditional programming languages.

You can use MATLAB for a wide range of applications, including signal processing and communications, image and video processing, control systems, test and measurement.

There are many alternative products such as Octave, Freemat, Scilab, NCLab, Python (with package NumPy, SciPy and MatPlotLib) and other.

1.2 Engineering tools – MATLAB and alternatives

1.2.1 What is MATLAB?

MATLAB is a shortcut from Matrix Laboratory. The Mathworks Inc. [1] developed this software primarily to solve linear equation systems due to linear matrixes.

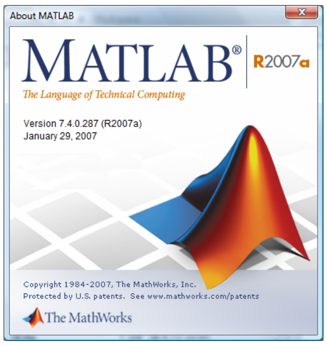
The MATLAB is environmental system and programming language too. It is mathematical oriented computer system based on solving many problems from mathematic and other research section. It is interactive system, Fig. 1.1.

In a short time this system stays standard for technician calculating and engineering. The base of this program is very similar to C language. Both programs have many similar algorithms and principles. Core of MATLAB is written in C language. Although the C language is compiler, MATLAB is designed like interpret.

MATLAB is not only programming language for technical calculation, but MATLAB is comprehensive math based tool for solving mathematics expression in branch of mathematics, engineering, economy, biology, chemistry etc. Its possible to say, that MATLAB is a philosophy [2].

1.2.2 Alternative product

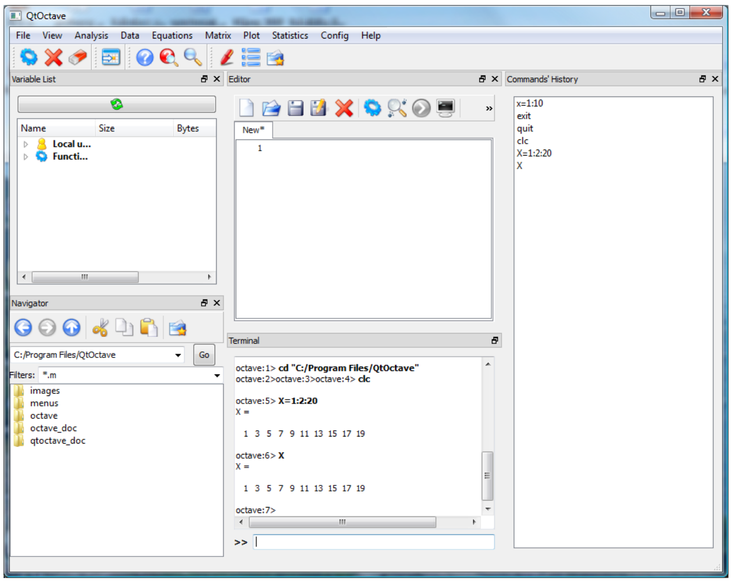
MATLAB is a standard in category of mathematics languages. But its complexity and its transcendence in this programming league defined its high price. Due this fact exist many alternatives for this product with a similar philosophy and similar efficiency. One of the best product of MATLAB style is Octave system with graphical frontend QT Octave, Fig 1.2 and FreeMat. These open source products are distributed under GNU public license.



Version 7.4.0.287 (R2007a)
January 29, 2007

Copyright 1984-2007, The MathWorks, Inc.
Protected by U.S. patents. See www.mathworks.com/patents

Figure 1.1: MATLAB logo and reprint screen from application site Mathworks Inc.



```
octave:1> cd "C:/Program Files/QtOctave"
octave:2>octave:3>octave:4> clc

octave:5> X=1:2:20
X =

  1  3  5  7  9  11  13  15  17  19

octave:6> X
X =

  1  3  5  7  9  11  13  15  17  19

octave:7>
>>
```

```
x=1:10
exit
quit
clc
X=1:2:20
X
```

Figure 1.2: Reprint of the screen from QT Octave

There exist other free to use products with similar functionality, for example SciLab, SageMath and others, but each from these groups has some special limitation in the relation to the original MATLAB.

1.3 MATLAB basics

The default MATLAB environment has three basic windows. The largest window (from these three) is for the right half of screen (default in right side of LCD screen). This window has name Command Window. This window is especially important for beginners because all commands are written just to the command line of this window. Primary function of this window is working desktop calculator style with hand input and like numeric output area. This windows rearrangement is typical for older version, in our case for version signed R2007a/b. Newest versions has other windows rearrangement, typically into five windows, but additional function has only second functionality from older one. The main window in all versions is typically Workspace and has the majority positions on the center positions on the screen. But the function is the same like in previous versions.

The left half of screen is divided into two smaller widows. Upper left is Current Directory or Workspace. All these information are valid for version R2007a/b. In newest versions are these windows in the left, respectively right columns of the screen.

Current Directory shows the actual working directory of MATLAB. It is important for saving our temporary files or scripts or function (in future). Current directory is directory for storing data, m-files etc. This directory can be set by user, it depends on the location of the user's program. Indication of the current settings of the current working directory is on the small box on the top of the menu bar on the MATLAB primary screen, Fig. 1.3.

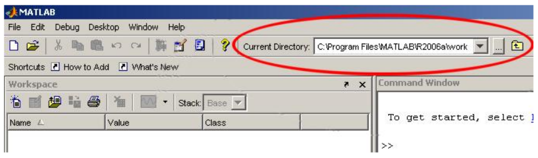

MATLAB
File Edit Debug Desktop Window Help
Current Directory: C:\Program Files\MATLAB\R2006a\work
Shortcuts How to Add What's New
Workspace
Stack: Base
Name Value Class
Command Window
To get started, select
>>

Figure 1.3: Setting up the Current directory and its setting in Matlab

The Workspace window is very useful. This window shows the currently defined variables, constants and functions. It will be detailed explained in next lessons. Each defined variable (i.e. commonly – object) has in this window small yellow symbol, the most usually small square (for usually numeric variables and constant). For the functions is this block in the small yellow cube style and other types of variables have another symbols. The last window with the name Command History shows us the flow of used commands from our lecture. It is possible to click on selected command from history for its activating. It is very useful in learning and self studying process.

These windows are possible change and close, but for beginners it is not a good idea. If you make some changes and if you want to return to the default style, you must select (from the top pull-down menu) path Desktop - Desktop layout and choose menu item Default settings. This path is possible to use in all version of MATLAB, but many of the programmers or students prefer the other windows rearrangement than default.

The next windows like output graphic window or Help window or Editor window are available from the many parts of MATLAB and in the right time we will notice about it.

The last part of screen is menu bar on top of screen. It has the similar shape like other Windows programs. In all versions of MATLAB is similar, but the newest version has another icon rearrangement, Fig. 1.5.

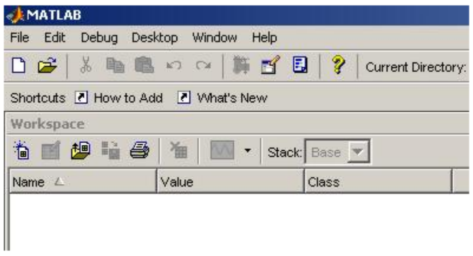

MATLAB
File Edit Debug Desktop Window Help
Current Directory:
Shortcuts How to Add What's New
Workspace
Stack: Base
Name Value Class

Figure 1.4: Detailed view for Menu items

Menu items

Fig. 1.4 depicts the detailed view for Menu items.

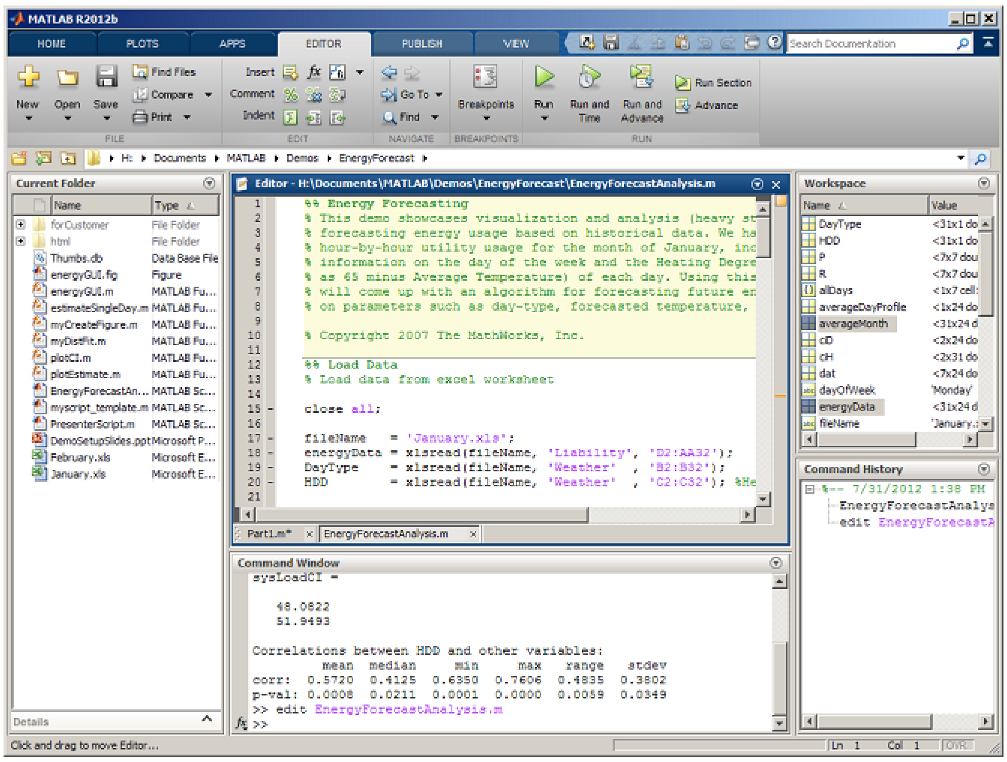
File - menu item of many Windows programs with the similar functionality. New for creating new program or script, Import for importing one, Preferences for setting the preferences of all MATLAB system including the fonts and many other).

Desktop - important function for setting the changes of windows to the standard or default style is path Desktop - Desktop Layout - Default.

Help - the most important item from menu bar. It is the best starting point of MATLAB self-study. In Demos is possible to find a lot of interesting scripts and functions. Function HELP has in MATLAB several forms. The main HELP is like library for all MATLAB environments. But in MATLAB is possible to call help with the relation with specific command for detailed information about this command (and for example detailed syntax and many more) - i.e. the context help.

For the self study is very important function *demos* which demonstrate the commands' possibilities in the illustrative example.

Demos - MATLAB - Mathematics - Basic Matrix Operations and next



Name	Type
forCustomer	File Folder
html	File Folder
Thumbs.db	Data Base File
energyGUI.fig	Figure
energyGUI.m	MATLAB Fu...
estimateSingleDay.m	MATLAB Fu...
myCreateFigure.m	MATLAB Fu...
myDistFit.m	MATLAB Fu...
plotCI.m	MATLAB Fu...
plotEstimate.m	MATLAB Fu...
EnergyForecastAn...	MATLAB Sc...
myscript_template.m	MATLAB Sc...
PresenterScript.m	MATLAB Sc...
DemoSetupSlides.ppt	Microsoft P...
February.xls	Microsoft E...
January.xls	Microsoft E...

Editor - H:\Documents\MATLAB\Demos\EnergyForecast\EnergyForecastAnalysis.m

```
%% Energy Forecasting
% This demo showcases visualization and analysis (heavy st
% forecasting energy usage based on historical data. We ha
% hour-by-hour utility usage for the month of January, inc
% information on the day of the week and the Heating Degre
% as 65 minus Average Temperature) of each day. Using this
% will come up with an algorithm for forecasting future en
% on parameters such as day-type, forecasted temperature,

% Copyright 2007 The MathWorks, Inc.

%% Load Data
% Load data from excel worksheet

close all;

fileName   = 'January.xls';
energyData = xlsread(fileName, 'Liability', 'D2:AA32');
DayType    = xlsread(fileName, 'Weather'  , 'B2:B32');
HDD        = xlsread(fileName, 'Weather'  , 'C2:C32'); %He

```

Part1.m* × EnergyForecastAnalysis.m ×

Command Window

```
sysLoadCI =

   48.0822
   51.9493

Correlations between HDD and other variables:
          mean   median      min      max    range    stdev
corr:   0.5720   0.4125   0.6350   0.7606   0.4835   0.3802
p-val:  0.0008   0.0211   0.0001   0.0000   0.0059   0.0349
>> edit EnergyForecastAnalysis.m
>>
```

Workspace

Name	Value
DayType	<31x1 do
HDD	<31x1 do
P	<7x7 dou
R	<7x7 dou
allDays	<1x7 cell:
averageDayProfile	<1x24 do
averageMonth	<31x24 d
cD	<2x24 do
cH	<2x31 do
dat	<7x24 do
dayOfWeek	'Monday'
energyData	<31x24 d
fileName	'January.:

Command History

```
%-- 7/31/2012 1:38 PM
  EnergyForecastAnalys
  edit EnergyForecastA
```

Figure 1.5: New ribbon components' arrangements in version R2012

Command Window

Command window is shown in Fig. 1.5 in the middle bottom. Work in this window has the desktop calculator mode. For programmers - it is typical scripting language (Python, Lua) style of work (sometimes called interpret(er) style). User types your code on the keyboard and after Enter key pressing, MATLAB evaluates and shows the result. Numerical input by manual typing is not very useful in case of larger tasks; we use it only for beginner's lessons or for home self study.

In this Command window, we will teach and make standalone test some simple commands for first part of beginner lecture.

Basic commands

Following commands we will test (by manual typing) in the largest window - **Command window** (type it and see what happens in Command Window).

Basic command for the first orientation in MATLAB

1. clc - most important command - remove all in this window (including error messages)
2. clear - remove all variables from memory (including their name and type)
3. who - list of used variables in memory (simple shape)
4. whos - list of used variables in memory in detailed form (size and memory)
5. what - list of M-files in current directory (described on Current Directory window)
6. exit - end of MATLAB session and escape from this program
7. quite - the same like exit
8. demo - run demo session (the same like from menu bar)
9. bench - starting program for testing power of your computer (benching)
10. help + next command - for example help elfun - detailed list elementary fc.
11. ver - returns MATLAB version with detailed summary of toolboxes.
12. !dir - provides commands of operating system shell, in this case it is Windows shell command dir - listing of files and folders in the current directory (in the Linux OS the command !ls has a similar functionality)

Very interesting command is `diary`. This command makes complete dump of used command and functions with results of mathematical operations into the text file. It is like a notepad. It is very useful for first lectures, like this one.

Write into Command Window and press ENTER

```
diary today.txt
diary on
```

Now is setting-up the file with the same content like Command Window. This file is stored in current working directory defined by Current Directory.

Please see the special using of the instruction `help`. List of the basic operation over the real variables is possible to obtain using `help elfun`. List of polynomial functions is accessible via commands `help polyfun` and the same in matrix linear algebra via command `help elmat`. Due these commands is possible access many different command with further syntax and examples. It is very valuable help in the self study and in learning MATLAB.

1.4 Syntax and operational principles

Case sensitivity is property known from common programming languages like C, C++, Java etc. Software can differentiate between upper and lower cases in names of the variables, functions and commands.

MATLAB is a ***case sensitive***. In praxis this sentence means that (for example in name of variable) variable a isn't equal to A ($a \neq A$). It is very important and many people make in this command many errors. Very useful consensus in creation names for variables is that names of vectors and matrices are from upper cases and other names for example scalar variables are created in lower case.

MATLAB syntax differs for issuing commands and for calling functions. In many cases, you can use either of the two syntaxes in commands and in function calls.

You are issuing a command when you tell MATLAB to do something. You might use a function name in the command and possibly a few switches or modifiers, usually expressed in the form of character strings. But, in most cases, there is no need to pass data such as numeric values or variables in a command. There is also no expectation that the command returns any output value.

A few examples of MATLAB commands are shown here. Note that MATLAB passes only string arguments and does not return any output:

Examples

```
whos

clear

format long e

a = 5; b = 6; c = 7;
save mydata.mat a b c
```

A function called in this syntax consists of the function name followed by one or more arguments separated by commas and enclosed in parentheses:

```
functionname(arg1, arg2, ..., argn)
```

You can assign the output of the function to one or more output values. When assigning to more than one output variable, separate the variables by commas or spaces and enclose them in square brackets `[]`:

```
out = functionname(arg1, arg2, ..., argn);

[out1,out2,...,outn] = functionname(arg1, arg2, ..., argn);
```

Examples

```
plot([0:0.1:2*pi],sin(0:0.1:2*pi))

size([1,2,3;4,5,6])

s = size([1,2,3;4,5,6])

[rows, columns] = size([1,2,3;4,5,6])
```

Notice

The semicolon can be used to construct arrays, suppress output from a MATLAB command, or to separate commands entered on the same line.

1.5 MATLAB toolboxes

Toolboxes are groups of additional special functions which are used for solving certain problems from many segments of engineering calculations. Some toolboxes affects the basic functionality of MATLAB. It is for example the toolbox for working with symbolic algebra. List of installed toolboxes we can obtain calling command `ver` (like version of MATLAB installation).

1.6 Test of basic commands and standalone works with basic operators

1. What is it case sensitivity?
2. What is toolbox?
3. Try the calculator function in command line of MATLAB Command windows or of some alternative product. Write, for example:

   ```
   1+2
   6*(120-78)
   sin(0)
   ```

   etc.

2 Matrix oriented commands, linear algebra

2.1 Basic mathematical operators and types of variables

Notice

Before starting our work in this large Command Window, it is necessary to know, that the style of work is the same like on the usual desktop calculator (that means interpreter style). It is similar to the programming language like Python, Lua or Shell scripts and different to languages like C, C++ or Java.

2.1.1 Allocation

Allocation - that means operation that join name with the value to the one expression (variable definition). For single variable is the same like in other languages. Very simple:

```
A = 5
```

And when we try to test the command `whos` now, in Command Window appear the detail information including size of variable in bytes, class and more. In the Workspace window appears new symbol in style of the yellow square with cross. This is symbol for usual numeric variable.

When we write in the next command a = 5, we have in this time two variables with names A and a, so it is necessary take care about case sensitivity!

When we type in Command Window next sentence, we obtain string variables (class char). After testing this variable using the command whos, we obtain next information.

```
B = 'text'
```

In case of string variable is shape in the Workspace window other than previous. The small yellow square is the same, but inside this yellow box is symbol for text value, small abc symbols.

For complex values it is very similar. For example definition of complex number C:

```
C = 2 + i
```

In case of complex variables apply MATLAB the same operations rules like on usual numeric variables. The differences exist only in case complex matrices and some special functions like conjugated matrices. This problem will be discussed in the next text.

The variable `ans` is the shortcut for answer and in under this variable name is stored result of the latest previous operation including inversion and similar other simple function. Try these commands and look at the list of variables:

```
1 + 1
a = 1 + 1
whos
```

Notice

Try examples. When necessary, it is good idea for detailed information about definition range and many more parameters use help and certain function.

2.1.2 Predefined constants

`ans` is variable with last result (ans like answer), the variable ans is created automatically when expressions are not assigned to anything else
`eps` is accuracy (respect computer and software possibilities - more `help eps`)
`realmin` is the smallest positive normalized double precision floating point number on this computer.
`realmax` is the largest double precision floating point number representable on this computer.
`pi` is 3.1415926535897... i.e. "π"
`i`, `j` are complex unit, i.e. sqrt(-1), it is not good idea rename it or use like index!
`Inf` is infinity, result of certain mathematical operations (for example division 1/0)
`NaN` is Not-a-Number, result of certain mathematical operation (for example 0/0)
`clock` is current date and time as date vector (in inversion order)
`date` is current date as date string (from computer memory)

2.1.3 Arithmetic operators

Arithmetic operators for matrix and array arithmetic:

A + B array sumation (element by element),

A − B array subtraction (element by element),

A ∗ B matrix multiplication,

A .∗ B array multiplication (element by element),

A / B matrix division, A is divided by B from right,

A ./ B array division, A is divided by B from right (element by element),

A \ B matrix division, B is divided by A from left,

A .\ B array division, B is divided by A from left (element by element),

A ∧ n matrix power, where n is a scalar number,

A . ∧ n array power (element by element), where n is a scalar number,

A ′ complex conjugated transpose of matrix (all complex elements has been replaced by conjugated complex numbers),

A .′ transpose of matrix (complex elements remain unchanged).

MATLAB software has two different types of arithmetic operations. Matrix arithmetic operations `* / \ ^` are defined by the rules of linear algebra. Array arithmetic operations `+ - .* ./ .\ .^` are carried out element by element, and can be used with multidimensional arrays. The dot character (.) distinguishes the array operations (element by element) from the matrix operations. However, since the matrix and array operations are the same for addition and subtraction, the character pairs .+ and .- are not used.

2.1.4 Relational operations

There are defined relational operators <, >, <=, >=, == and ∼= in MATLAB. Meanings of these operators are

A < B A is less than B,

A > B A is greater than B,

A <= B A is less than or equal to B,

A >= B A greater than or equal to B,

A == B A is equal to B,

A ∼= B A is not equal to B.

Relational operators perform element-by-element comparisons between two arrays. They return a logical array of the same size, with elements set to logical 1 (true) where the relation is true, and elements set to logical 0 (false) where it is not.

The operators <, >, <= and >= use only the real part of their operands for the comparison. The operators , == and ∼= test both real and imaginary parts.

Attention

The difference between = and ==

= means assignment of a value to a variable,

== means equal to.

Examples

```
~0
ans  =  1

~1
ans  =  0

~5
ans  =  0

a=7; b=8;

a~=b
ans =
     1

a>=b
ans =
     0

A=magic(3)
A =
     8     1     6
     3     5     7
     4     9     2
```

```
A>4
ans =
     1     0     1
     0     1     1
     0     1     0
```

Attention

- Result of relation between scalars is a scalar (value 1 – means true or value 0 – means false).
- Result of relation between matrices (with same size) is a relation matrix of 0 and 1 as the result of comparison of elements at the same coordinates.
- Result of relation between two matrices is whole true, if all elements of relation matrix are 1 (true).

Useful testing functions

any(A) at least one non-zero element at A columns,

all(A) all non-zero elements at A columns.

2.1.5 Logical operations

Relational operations can be joined by logical operators. There are two types of logical operators

- elementwise

  & means logical and (true are all together),

  | means logical or (true is at least one),

  ~ means logical not,

- short-circuit

  && means logical and (true are all together),

  || means logical or (true is at least one),

Examples

```
(3<5)&(4<6)
ans = 1

(3<5)&(4>6)
ans = 0

(3>5)&(4>6)
ans = 0

(3>5)|(4>6)
ans = 0

~(3<5)
ans =0
```

```
(~(3<5))&(4<6)
ans = 0
```

2.2 Basic commands and basic operations instructions

2.2.1 Formatting the numerical output (for visual form on the screen)

Format command
This instruction set the visible output numeric format. It is very important especially in floating point variables and numbers. This setting the format doesn't affect for internal computation and internal numeric precisions. This is only for visual expression in the Command Window.
`format short` - only 4 number after comma
`format long` - 15 digits in double precision expression and 7 digits for single precision.
`format hex` - hexadecimal format
`format bank` - fixed format for dollars and cents
`format rat` - expression values like ration of small integers
For many more possibilities, details and certain syntax see after typing `help format`.

Before starting definition of matrices, it is necessary say a lot of special marks used for these operations.

Notice

Next information are valid not only for matrices, but for vectors too. From mathematical point of view is vector only special type of matrix with size 1 x n (n is number of elements).

2.2.2 Parentheses

In MATLAB (in alternative programs often too) we have three types of parentheses. Differences between these three types of parentheses is in the method for theirs using.

`[ ]` - brackets (for vectors and matrices definition)

`( )` - classical parentheses (for mathematical operations, indexes, arguments and more)

`{ }` - braces (useful for special operation with cells, text arguments and more)

All parentheses must be paired!!!

2.2.3 Separators

Separators are used for separating of indexes, elements and for separating whole structures like vectors in the matrices. We have three types of separators named comma, semicolon and colon with different function.

`:` - colon (creator of vectors, array subscripting, separation of range borders, alternate symbol for matrix structures, and for-loop iterators)

`;` - semicolon (the end of row vectors in matrix, separation of commands, suppress output from a MATLAB command, and more)

`,` - comma (vector and matrix element separator, index separator)

Each from these separators has the secondary functionality. For example semicolon is used for separating commands in the program script and for breaking-up the program line in the editor. Colon is used for separating the elements of the range vector. And comma is used in many cases for separating indexes in the parentheses and many more.

2.2.4 Definition of vectors

Linearly spaced vectors

Row vector

```
RV = [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

or

```
[1␣2␣3␣4␣5␣6␣7␣8␣9]
```

Like vector element separator is used in this case comma. Like the second possibility is using the space, and this is due the pushing key space for separating values. This variant is not recommended due possibility of mistake with unwished push this key. In originally material from UK or USA comma is used very often.

Column vector

```
CV = [1; 2; 3; 4; 5; 6; 7; 8; 9]
```

Like vector element separator is used semicolon. This separator is used in case separation of row in matrix. But in the case of column vector it is typical using in the vector.

For transpose between column and row vector forms (mathematically named transposition) are used symbol of apostrophe. That means in you want make column vector from row vector, use apostrophe (symbol for transposition).

```
CV = RV.'
```

Using colon

The colon operator uses the following rules to create regularly spaced vectors for scalar values *m*, *n*, and *p*:
`m:n` is the same as `[m,m+1,m+2,...,n]`, or empty `[]` when `m > n`.
`m:p:n` is the same as [m,m+p,m+2p, ...,m+r*p], where r = fix((n-m)/p). This syntax returns an empty `[]` matrix when `p == 0`, `p > 0` and `m > n`, or `p < 0` and `m < n`.

Example

The following command create the vector from 0 to 0.5 with step 0.1 (it is possible use normal parentheses or without parentheses)

```
x = [0:0.1:0.5]
x =
         0    0.1000    0.2000    0.3000    0.4000    0.5000
```

To create the vector from 1 to 5 with step 1 (incremental), use command:

```
y = [1 : 5]
y =
     1     2     3     4     5
```

If the increment is one, it is not necessary to write them. Compare:

```
[1 : 1 : 5]
ans =
     1     2     3     4     5
```

In the following example, we create the vector from 0 to 7 with step 2:

```
[0 : 2 : 7]
ans =
     0     2     4     6
```

The end of this vector is 6 and this vector has four elements, because:
first element: 0
second element: $0+1*2=2$,
third element: $0+2*2=4$,
fourth element: $0+3*2=6$,
followed by: $0+4*2=8$
$6<7$ and $8>7$, 6 is the last number before 7, which satisfies the specifications (step 2).

If we create vector with a positive step, the start must be less than the end of vector, otherwise there is an empty matrix.

```
[5:1]
ans =
   Empty matrix: 1-by-0
```

Definition of the vector containing decreasing values of elements has the form:

```
z = [10 : -1 : 1]
z =
    10     9     8     7     6     5     4     3     2     1

r = [8 : -2 : 0]
r =
     8     6     4     2     0

s = [15: -3 : 4]
s =
    15    12     9     6
```

Decremental step must be written in any case. If the vector is created with a negative step, the value of starting vector element must be greater than the value of the last vector element, otherwise the command returns an empty matrix.

```
[4:-1:8]
ans =
   Empty matrix: 1-by-0
```

If we want make linear vector `t` from elements $0,1,2,3,...,n$, we may write element using this command `t = 0:`n;

If we will make some vector with the certain number of equidistant elements, we may use the function for linear row - `linspace` with the syntax:

`linspace(a, b)` generates a row vector of 100 linearly equally spaced points between `a` and `b`.

`linspace(a, b,n)` generates `n` linearly equally spaced points between `a` and `b`. For `n`< 2, `linspace(a, b,n)` returns `b`.

Example

```
VECTOR = linspace(0, 100, 10)
VECTOR =
  Columns 1 through 8
```

```
         0   11.1111   22.2222   33.3333   44.4444   55.5556   66.6667   77.7778
  Columns 9 through 10
   88.8889  100.0000

DV = linspace(20, -4, 5)
DV =
    20    14     8     2    -4
```

Notice

Command `linspace` is useful for vector construction like our previous command with colons. But if we want to obtain the same numbers, we must type it in the right shape: The vector created using `1:3` is the same as `linspace(1,3,3)` !

Example

```
a = 0 : 1 : 7
a =
     0     1     2     3     4     5     6     7

a = linspace(0,7,8)
a =
     0     1     2     3     4     5     6     7
```

Logarithmically spaced vectors

The command `linspace(1, 100, 10)` makes vector from 1 to 100 with 10 linearly equally spaced elements. If we want make logarithmically spaced vector, use command logspace(0, 2, 10) when the first parameter defined 10^0 second parameter targeting number 10^2 and the third parameter defined number of elements.

`logspace(a, b)` generates a row vector of 50 logarithmically equally spaced points between decades 10^a and 10^b. If `b` is pi, then the points are between 10^a and pi.

`logspace(a, b, n)` generates `n` logarithmically equally spaced points between 10^a and 10^b. For `n` < 2, `logspace` returns 10^b.

Example

```
linspace(1, 100, 10)
ans =
     1    12    23    34    45    56    67    78    89   100

logspace(0,2,10)
ans =
  Columns 1 through 8
    1.0000    1.6681    2.7826    4.6416    7.7426   12.9155   21.5443   35.9381
  Columns 9 through 10
   59.9484  100.0000
```

See also the following results:

```
linspace(1, 1000, 4)
ans =
           1         334         667        1000

logspace(0, 3, 4)
ans =
           1          10         100        1000
```

2.2.5 Definition of matrices

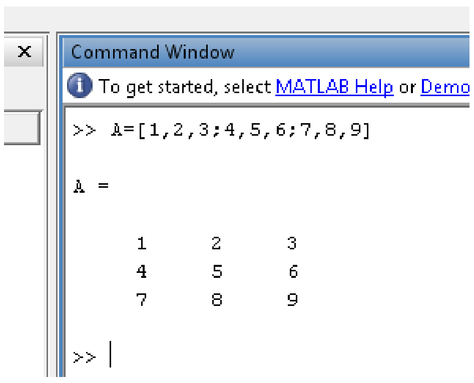
Each matrix has row and column elements sorting into rectangular or square structure. In the most usual cases of technical computations it is square structures and the name is the same - square matrices. Exist of course non-square matrices (rectangle) but its consequence is very small in the usual technical computation. In the next text we will discussed only square matrices. Some function may change the square shape of matrix for rectangular, but this is minority appearance. Entering the matrix (definition of the new matrix input)

Example

```
A = [1, 2, 3; 4, 5, 6; 7, 8, 9]
```

create this structure, Fig. 2.1:

```
A =
   1   2   3
   4   5   6
   7   8   9
```



Command Window

To get started, select MATLAB Help or Demo

```
>> A=[1,2,3;4,5,6;7,8,9]

A =

     1     2     3
     4     5     6
     7     8     9

>> |
```

Figure 2.1: Reprint of the matrix structure from Command Window

That means like border symbols are used brackets. This is important; each type of parenthesis has its own function. When we input this matrix into MATLAB, in the left upper window (Workspace) appears new definition symbol of our matrix (yellow square) with name. This is variable name of this matrix or matrix identifier.

When we click by mouse on the square (in this left upper window - Workspace) appears in the right window new structure with the name Array editor. In this editor is possible to change values of this matrix. It is very similar to Microsoft Excel editor with cells.

Structure of matrix

Matrix has own structure. Each element has your own position. It is possible to sign this position with matrix coordinates or matrix elements indexes.

The most important lines in matrices are rows (horizontal group of elements) and columns (vertical group of elements). When the number of rows is equal the number of columns, we say, that the matrix is square matrix. These types of matrix are very useful for computing (for example inversion and translation). Non square matrices are very problematic for many mathematical operations.

Important sequence of elements in matrix is diagonal line of elements. Each matrix has two diagonals. One diagonal matrix (from upper left corner to the right lower corner) names main and the secondary diagonal in the opposite direction names adjacent (from upper right corner to the left lower corner).

When the main diagonal contains only number one and the other elements in the matrix are zeros, we names this matrix like identity or unit matrix. This matrix is possible obtain like result of certain mathematical operation for example like multiply inversion matrix with the origin one. But this operation has many restrictive conditions (for example singularity of input matrix and many more).

Construction of matrix

It is possible to enter each element handy from the main window (Command window). The more efficiency way is the using M-File editor. This will be containing of next lesson. Matrices are possible download from external sources too. The next way to the certain matrix type is generating matrices. For example: matrix construction of the individual elements

```
A = [4, 2, 0; 4, 2, 6; 7, 2, 1]
```

One of vary possibilities of construction of matrices is adding two or more vectors together into matrix structures.

We have row vectors:

```
a = [1, 2, 3];
b = [4, 5, 6];
c = [7, 8, 9];
```

The command `MATRIX = [a;b;c]` construct new matrix with these elements, the vectors are separated by semicolons, Fig. 2.2:

```
MATRIX = [a;b;c]
MATRIX =
     1     2     3
     4     5     6
     7     8     9
```

The command `MATRIX_2 = [a,b,c]` construct new vector with these elements, the vectors are separated by commas:

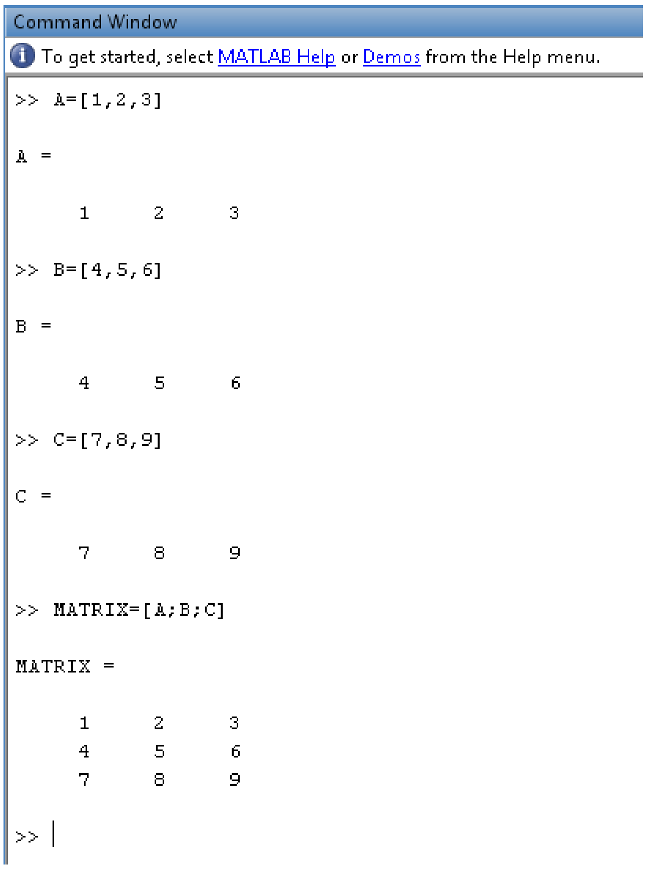
```
MATRIX_2 = [a,b,c]
MATRIX_2 =
     1     2     3     4     5     6     7     8     9
```

Notice

The same process us useful in case of column vectors too. In case of column vector is used like separator symbol comma. We have column vectors (the elements separated by semicolons):

```
d = [1; 2; 3];
e = [4; 5; 6];
f = [7; 8; 9];
```

The command `MATRIX_3 = [d,e,f]` construct new matrix with these elements, the vectors are separated by commas:


Command Window
To get started, select MATLAB Help or Demos from the Help menu.

```
>> A=[1,2,3]

A =

     1     2     3

>> B=[4,5,6]

B =

     4     5     6

>> C=[7,8,9]

C =

     7     8     9

>> MATRIX=[A;B;C]

MATRIX =

     1     2     3
     4     5     6
     7     8     9

>> |
```

Figure 2.2: Construction of the matrix from vectors

```
MATRIX_3 = [d,e,f]
MATRIX_3 =
      1      4      7
      2      5      8
      3      6      9
```

Example

We have vectors:

```
k = [1, 3, 5, 7, 9];
l = [1 : 5];
m = linspace(-4, 4, 5);
n = logspace(1, 5, 5);
```

Is it possible to create a matrix of these vectors `k`, `l`, `m`, `n`?
Solution

```
P = [k; l; m; n]

P =

             1            3            5            7            9
             1            2            3            4            5
            -4           -2            0            2            4
            10          100         1000        10000       100000
```

2.3 Operations with vectors and matrices

2.3.1 Basic matrix operations

The followed matrix *A* is given

```
A = [4, 2, 0; 4, 2, 6; 7, 2, 1]
A =
     4     2     0
     4     2     6
     7     2     1
```

Some basic matrix manipulations and operations for this matrix `A` in MATLAB are shown in following text.

- Inversion form of this matrix we obtain using function `inv(A)`, which returns the inverse matrix only of the square matrix `A`, a warning message is printed in other cases.

  ```
  ans =
     -0.2778   -0.0556    0.3333
      1.0556    0.1111   -0.6667
     -0.1667    0.1667         0
  ```

- Transpose forms obtain after using function `A'` with one exceptions in the complex elements. In case the complex elements after making transpose we obtain not the transpose matrix, but the conjugating matrix (For details, see section 2.4).

  ```
  ans =
       4     4     7
       2     2     2
       0     6     1
  ```

- After using function `size(A)` will be in the `ans` variable in this case numbers 3 and 3.

  ```
  size(A)
  ans =
       3     3
  ```

  That means this matrix has 3 rows and 3 columns.

  `[r,c] = size(A)` returns the size of matrix `A` in separate variables `r` (number of rows) and `c` (number of columns).

  ```
  [r,c] = size(A)
  r =
       3
  c =
       3
  ```

  `m = size(A,dim)` returns the size of the dimension of `A` specified by scalar `dim`.

  ```
  m = size(A,1)
  m =
       3
  n = size(A,2)
  n =
       3
  ```

Example

```
B=[1, 4, 5; -6, 9, 8]
B =
     1     4     5
    -6     9     8

[r,c] = size(B)
r =
     2
c =
     3

k = size(B,1)
k =
     2

l = size(B,2)
l =
     3
```

- Length of vector a is obtained using the command `length(a)`. If `A` is matrix, `length(A)` finds the number of elements along the largest dimension of an matrix. It is equivalent to `max(size(A))` for non-empty matrices.

```
a = [1, 6, 4, -2, 3, 3, 5, 4];
length(a)                          % number of elements of vector a
ans =
     8

B=[1, 4, 5; -6, 9, 8];
length(B)                          % size of matrix A is 2 x 3 (see previous example)
ans =
     3

max(size(B))
ans =
     3

A = [4, 2, 0; 4, 2, 6; 7, 2, 1];
length(A)                          % size of matrix A is 3 x 3
ans =
     3

max(size(A))
ans =
     3
```

The command `max` returns the largest elements in vector. Details of `max` are described in the following items.

- When you use on the same matrix `A` function `sum(A)`, you obtain summation of each column. In this case it will be `15, 6, 7` (results of the summation of each column from this matrix).

  `sum(A,dim)` sums along the dimension of `A` specified by scalar `dim`.

```
sum(A, 1)
```

```
ans =
    15     6     7
sum(A, 2)
ans =
     6
    12
    10
```

Compare the second result with the following commands.

```
 A'
ans =
     4     4     7
     2     2     2
     0     6     1
sum(A')
ans =
     6    12    10
```

- Rotation matrix 90 degrees `rot90(A)` rotates matrix `A` counterclockwise by 90 degrees.

```
rot90(A)
ans =
     0     6     1
     2     2     2
     4     4     7
```

`rot90(A,k)` rotates matrix `A` counterclockwise by $90 \cdot k$ degrees, where k is an integer.

```
rot90(A,2)
ans =
     1     2     7
     6     2     4
     0     2     4

rot90(A,3)
ans =
     7     4     4
     2     2     2
     1     6     0
```

For example: `sum(rot90(A))` - where `rot90` is matrix rotation about 90° counterclockwise.

```
sum(rot90(A))
ans =
     6    12    10
```

Attention

Using matrix transposition `A'`, different matrix is created than using rotation `rot90(A)`.

- `max` returns the largest elements in array. If `a` is a vector, `max(a)` returns the largest element in `a`. If `A` is a matrix, `max(A)` treats the columns of `A` as vectors, returning a row vector containing the maximum element from each column.

```
A = [4, 2, 0; 4, 2, 6; 7, 2, 1]
A =
     4     2     0
     4     2     6
     7     2     1
max(A)
ans =
     7     2     6
```

`[m,p] = max(A)` finds maximum values and the indices of the maximum values of `A`, and returns them in output vectors `m` and `p`. If there are several identical maximum values, the index of the first one found is returned.

```
[m,p] = max(A)
m =
     7     2     6
p =
     3     1     2
```

- `min` returns smallest elements in array. Description and possibilities of use are similar to the `max`.
- `diff(A)` calculates differences between adjacent elements of `A`.

  If `(a)` is a vector, then `diff(a)` returns a vector, one element shorter than `(a)`, of differences between adjacent elements:

```
a = [1, 6, 4, -2, 3, 3, 5, 4];
diff(a)
ans =
     5    -2    -6     5     0     2    -1
```

  If `(A)` is a matrix, then `diff(A)` returns a matrix of row differences:

```
A = [4, 2, 0; 4, 2, 6; 7, 2, 1]
A =
     4     2     0
     4     2     6
     7     2     1
diff(A)
ans =
     0     0     6
     3     0    -5
```

- `mean` returns average or mean value of array. If `A` is a matrix, `mean(A)`s treats the columns of `A` as vectors, returning a row vector of mean values.

```
mean(A)
ans =
    5.0000    2.0000    2.3333
```

- `median` returns median value of array. If `A` is a matrix, `median(A)` treats the columns of `A` as vectors, returning a row vector of median values.

```
median(A)
ans =
     4     2     1
```

- `std` returns standard deviation. If `A` is a matrix, `std(A)` returns a row vector containing the standard deviation of the elements of each column of `A`.

```
std(A)
ans =
    1.7321         0    3.2146
```

- `sort` returns sort array elements in ascending or descending order, the specified direction is depending on the value of mode (`'ascend'` – ascending order (default) or `'descend'` – descending order). If A is a matrix, `sort(A)` sorts each column of A.

```
sort(A)
ans =
     4     2     0
     4     2     1
     7     2     6

sort(A, 'descend')
ans =
     7     2     6
     4     2     1
     4     2     0
```

- Function `diag(A)` makes vector from elements from main diagonal matrix A. In our example it will be `4, 2, 1`. When use commands `diag(diag(A))` obtain matrix with the same diagonal and other elements are zeros.

```
A = [4, 2, 0; 4, 2, 6; 7, 2, 1];
diag(A)
ans =
     4
     2
     1
diag(diag(A))
ans =
     4     0     0
     0     2     0
     0     0     1
```

Notice

In opposite side, when we use command `diag(a)`, where `a` is vector (row or column - it is no matter) - we obtain matrix full of zeros with elements on the main diagonal identical with the elements of original vector `a`.

```
a = [1,3];
diag(a)
ans =
     1     0
     0     3
```

Results from this operation storing in the variable `ans` are again matrices of vectors. That means, these results are possible to allocate and use like normal next variable (or vector and matrix of course).

There are many other functions for matrix manipulations and operations with matrices, for more information see `help elmat`.

2.3.2 Indexing of matrix elements

Special instructions for selection some structures (row or column) from matrix.

When is necessary choose only one element, we must input his coordinates, for example second element in third row: `A (3, 2)` Like the first parameter is number of row and second parameter is number of column.

Example

```
A = [4, 2, 0; 4, 2, 6; 7, 2, 1]
A =
     4     2     0
     4     2     6
     7     2     1

A(3,2)
ans =
     2

A(2,3)
ans =
     6
```

Colon symbol

For example, if we want select *m*-th row from matrix `A`, we use: `A (m, :)`. This operation has only one condition - the number of rows in the matrix `A` must be equal or greater than *m*!

Example

```
A(2, :)
ans =
     4     2     6
```

If we want extract *n*-th column from matrix `A`, we must write other command `A(:, n)`. This operation has only one condition - the number of columns in the matrix `A` must be equal or greater than *n*.

Example

```
A(:, 1)
ans =
     4
     4
     7
```

That means the colon is in these cases very important symbol for selecting whole structure from matrix. If we want select row, colon is on the second position and if we have select column, we must give colon on the first position.

Example

We have matrix `M` with structure 4 x 4

```
M=[1,2,3,4;4,3,2,-1;5,6,7,8;9,0,-5,7]
M =
     1     2     3     4
     4     3     2    -1
     5     6     7     8
     9     0    -5     7

M (2, :)          % second row from M
ans =
     4     3     2    -1

M (:, 2)          % second column from M
ans =
     2
     3
     6
     0
```

The most interesting situation begins when we want to make more difficult changes, for example, if we want to extract or change rows into matrix or make changes in the columns.

This is similar as creating of vectors using colon. We obtain sub-matrices and vectors, which were rows or columns.

If we want to return second and third rows, we use the command:

```
M(2:3, : )
```

If we want to return second and third columns, we use the command:

```
M(: , 2:3)
ans =
     2     3
     3     2
     6     7
     0    -5
```

If we want to return second and third rows in reverse order, we use the command:

```
M(3:-1:2, : )
ans =
     5     6     7     8
     4     3     2    -1
```

If we want to return second and third columns in reverse order, we use the command:

```
M(: , 3:-1:2)
ans =
     3     2
     2     3
     7     6
    -5     0
```

If we want to return the first, second and fourth rows, we use the command:

```
M([1, 2, 4], :)
ans =
     1     2     3     4
     4     3     2    -1
     9     0    -5     7
```

If we want to return the first, third and fourth column, we use the command:

```
M( : ,[1, 3, 4])
ans =
     1     3     4
     4     2    -1
     5     7     8
     9    -5     7
```

The command end can serve as the last index in an indexing of vectors and matrix. If we want to return the last rows, we use the command:

```
M(end, :)
ans =
     9     0    -5     7
```

If we want to return the first and last columns, we use the command:

```
M(:, [1,end])
ans =
     1     4
     4    -1
     5     8
     9     7
```

Different parts of the matrix M are obtained using the following commands:

```
M(2:3, 2:3)
ans =
     3     2
     6     7

M(2:3, 2:end)
ans =
     3     2    -1
     6     7     8

M(2:end, 1:3)
ans =
     4     3     2
     5     6     7
     9     0    -5

M(3:-1:1, 3:-1:1) % rows and columns in reverse order
ans =
     7     6     5
     2     3     4
     3     2     1
```

And like the most beautiful command try this:

```
M(end:-1:1, end:-1:1)
```

This is totally changing the order of rows and columns of whole matrix.

```
ans =
     7    -5     0     9
     8     7     6     5
    -1     2     3     4
     4     3     2     1
```

We can refer to the elements of a MATLAB matrix with a single subscript, M(k). MATLAB stores matrices and arrays not in the shape that they appear when displayed in the MATLAB Command Window, but as a single column of elements. This single column is composed of all of the columns from the matrix, each appended to the last.

So, matrix M

```
M=[1,2,3,4;4,3,2,-1;5,6,7,8;9,0,-5,7]
M =
     1     2     3     4
     4     3     2    -1
     5     6     7     8
     9     0    -5     7
```

is actually stored in memory as the sequence

```
1, 4, 5, 9, 2, 3, 6, 0, 3, 2, 7, -5, 4, -1, 8, 7
```

The element at row 3, column 2 of matrix M (value = 6) can also be identified as element 7 in the actual storage sequence. To access this element, you have a choice of using the standard `M(3,2)` syntax, or you can use `M(6)`, which is referred to as linear indexing.

Example

```
M(3,2)        % standard indexing
ans =
     6

M(7)          % linear indexing
ans =
     6

M(2,4)        % standard indexing
ans =
    -1

M(14)         % linear indexing
ans =
    -1
```

Vectors are matrices with one row or one column. The same rules apply for indexing vectors.

Example

We have the row vector v:

```
v=[1, 5, 7, 8, -3, 6, 9, 2, 4];

v(1,3)         % the element at row 1, column 3 of vector v
ans =
     7

v(3)           % the third element in the vector v
ans =          % Compare with the previous result!
     7
```

Due to the linear indexing can be written the actual sequence of element in the vector as the index.

```
v(1)          % the first element in the vector v
ans =
     1

v(5)          % the fifth element in the vector v
ans =
    -3

v(7)          % the seventh element in the vector v
ans =
     9

v(4)          % the fourth element in the vector v
ans =
     8

v(end)        % the last element in the vector v
ans =
     4
```

We have the column vector w:

```
w = [5; 0; 7; 4]
w =
     5
     0
     7
     4

w(2,1)        % the element at row 2, column 1 of vector w
ans =
     0

w(2)          % the second element in the vector w
ans =         % Compare with the previous result!
     0

w(3)          % the third element in the vector w
ans =
     7

w(end)        % the last element in the vector w
ans =
     4
```

The replacement of element of vectors and matrices is performed by assigning of another element to the appropriate position.

The replacement of fifth element in the vector v by number 30:

```
v=[1, 5, 7, 8, -3, 6, 9, 2, 4];
v(5) = 30
v =
     1     5     7     8    30     6     9     2     4
```

The replacement of the element in the fourth line in 2nd column of matrix M by number 10

```
M = [1,2,3,4;4,3,2,-1;5,6,7,8;9,0,-5,7];
M(4,2) = 10
M =
     1     2     3     4
     4     3     2    -1
     5     6     7     8
     9    10    -5     7
```

The replacement of rows or columns of matrix is performed by assigning of another vector with the corresponding number of elements.

The replacement of fourth column of matrix M by vector w:

```
w = [5; 0; 7; 4];    % column vector
M(:,4) = w
M =
     1     2     3     5
     4     3     2     0
     5     6     7     7
     9    10    -5     4
```

Deleting of rows or column of the matrix is performed by assigning an empty matrix.
Actual size of the matrix M:

```
size(M)
ans =
     4     4
```

Deleting of the third row of matrix M:

```
M(3,:) = []
M =
     1     2     3     5
     4     3     2     0
     9    10    -5     4
```

Actual size of the matrix M:

```
size(M)
ans =
     3     4
```

Deleting of the third and fourth column of matrix M:

```
M(:,3:4) = []
M =
     1     2
     4     3
     9    10
```

Actual size of the matrix M (original size of the matrix M before deleting was: 4 rows and 4 columns):

```
size(M)
ans =
     3     2
```

2.4 Matrix operations and tools

2.4.1 Special types of matrices

- eye - identity unit (main diagonal full of ones other elements equal to zero)

  Example

  ```
  eye(3)
  ans =

         1              0              0
         0              1              0
         0              0              1
  ```

- zeros - matrix full of zeros (zeros(5) - matrix 5 x 5 full of zeros)

Example

```
zeros(2,3)
ans =

       0               0               0
       0               0               0
```

- `ones` - matrix full of ones

Example

```
ones(2,4)
ans =
     1     1     1     1
     1     1     1     1
```

- `magic` - matrix contains elements having certain unusual properties (sum of row = sum of column and sum of elements on the main diagonal).

Example

```
magic(3)
ans =
     8     1     6
     3     5     7
     4     9     2

M=magic(3)
M =
     8     1     6
     3     5     7
     4     9     2

sum(M)
ans =
    15    15    15

sum(diag(M))
ans =
    15
```

- `pascal` - matrix contains the same elements like Pascal triangle (but in square shape)

Example

```
pascal(5)
ans =

     1     1     1     1     1
     1     2     3     4     5
     1     3     6    10    15
     1     4    10    20    35
     1     5    15    35    70
```

- `hilb` - Hilbert matrix is the matrix with elements $1/(m+n-1)$, which is a famous example of a badly conditioned matrix.

Example

```
format rat

hilb(4)
ans =
       1              1/2            1/3            1/4
       1/2            1/3            1/4            1/5
       1/3            1/4            1/5            1/6
       1/4            1/5            1/6            1/7
```

- `rand` - matrix containing pseudorandom values from the standard uniform distribution on the open interval (0-1), Fig. 2.3.

Example

```
x = rand(1,1000);
y = rand(1,1000);
plot(x,y,'o')
```

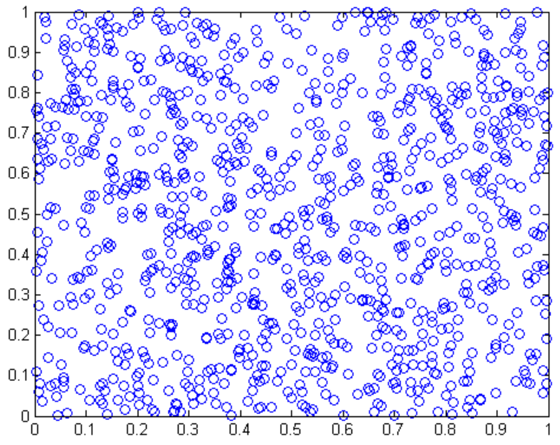


Figure 2.3: Uniformly distributed pseudorandom numbers on the open interval(0,1)

- `randn` - matrix containing pseudorandom values drawn from the standard normal distribution, Fig. 2.4.

Example

```
xn = randn(1,1000);
yn = randn(1,1000);
plot(xn,yn,'o')
```

- `diag(A)` - vector full of elements from main diagonal matrix A

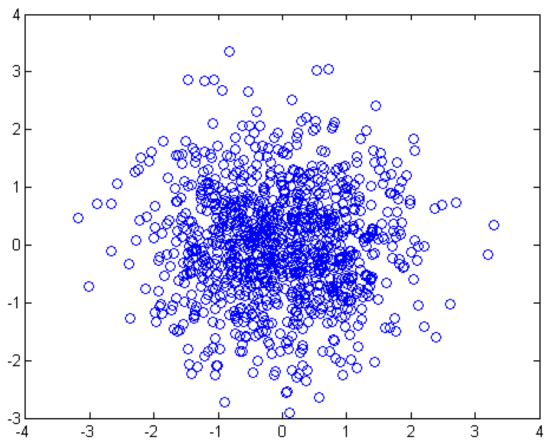


Figure 2.4: Normally distributed pseudorandom numbers

Example

```
A = [4, 2, 0; 4, 2, 6; 7, 2, 1];
diag(A)

ans =

     4
     2
     1
```

`diag(a)` - when `a` is a vector is created a square matrix with elements of `a` on the main diagonal `a`.

Example

```
a=[1,2,5];
diag(a)

ans =

     1     0     0
     0     2     0
     0     0     5
```

Other special types of matrices is possible obtain after typing `help elmat`.

2.4.2 Arithmetical operations with matrices

Matrices are the common variables like simple variables defined by one number. But the mathematical operations with matrices subjected certain mathematical laws. This is not the same like the simply scalar variables (for example division on the matrices).

```
A = [4, 2, 2; 2, 0, 1; 7, 2, 1]
B = [3, 2, 4; 1, 2, 4; 2, 0, 1]
A =
     4     2     2
     2     0     1
     7     2     1
B =
     3     2     4
     1     2     4
     2     0     1
```

Multiplication *

Multiplication of matrices is defined in a way that reflects composition of the underlying linear transformations and allows compact representation of systems of simultaneous linear equations. The matrix product `D = A * B` is defined when the column dimension of `A` is equal to the row dimension of B, or when one of them is a scalar. If A is m-by-p and B is p-by-n, their product D is m-by-n.

`A * B` $\neq$ `B * A` - basic axiom known from linear algebra

`A * B` and `B * A` are in this case matrix operations

Results obtained from MATLAB

```
D=A*B
D =
    18    12    26
     8     4     9
    25    18    37

F=B*A
F =
    44    14    12
    36    10     8
    15     6     5
```

Example

```
Z=[5,-9;3,4;-1,7]

Y= A*Z
Y =
    24   -14
     9   -11
    40   -48
```

If `A` is 3-by-3 and `Z` is 3-by-2, their product `Y` is 3-by-2.

Attention

Multiplication `Z*A` cannot be performed. If `Z` is 3-by-2 and `A` is 3-by-3, inner matrix dimensions disagree.

Notice

The result multiplication of matrix `A` with the transposed matrix `A.'` is symmetric matrix about the main diagonal:

```
A * A.'
ans =
    24    10    34
    10     5    15
    34    15    54
```

The result multiplication of matrix A with the identity matrix of the appropriate size is the same matrix A. The command for generation of identify matrix is eye. For example, `eye(5)` generates 5 x 5 matrix.

```
A * eye(3)
ans =
     4     2     2
     2     0     1
     7     2     1

eye(3) * A
ans =
     4     2     2
     2     0     1
     7     2     1

eye(3) * Z
ans =
     5    -9
     3     4
    -1     7

Z * eye(2)
ans =
     5    -9
     3     4
    -1     7
```

This is similar to the multiplication of any number by the number 1, for example:

```
2*1
ans =
     2

1*2
ans =
     2
```

Multiplication `.*`

Array multiplication `A .* B` is the element-by-element product of the arrays A and B. `A .* B` is not the matrix operation. A and B are in this case arrays not matrices. A and B must have the same size.

Array is the structure very similar to matrix. Differences exist only in the mathematical operations defined on this structure. All operations on the array are defined in style element on element. In case multiplication it is multiplication element by element.

`A * B` ≠ `A .* B`

Example

```
A .* B
```

```
ans =
    12     4     8
     2     0     4
    14     0     1

B .* A
ans =
    12     4     8
     2     0     4
    14     0     1
```

Notice

`A .* B` = `B .* A` - this multiplication is commutative, this is array multiplication, not matrices.

This operation is defined. `A`, `B` are arrays and this operation multiplies their elements. That means these elements are elements of arrays and not the ones of the matrix. This is not the matrix operation! This is very important notice.

Multiplication matrix with scalar is simply. Try to multiply `5 .* A` and discuss results.

Operations with the symbol of point we called "dot operation" or array operation. Dot operations are defined not on the matrices of normal values but on the fields of elements. The difference between field and matrix is not visible on the first look, but exist in the mathematical laws defined on these structures.

Array operations are not defined in the linear algebra. But array operations are very often usable in the case of graphical expressions namely in case 3-dimensional graphs. This problematic will be discussed later in the section dedicated the 3-dimensional graphs.

Division / and \

Division is very problematic function defined on matrices and linear algebra generally. Not the each division is possible. But the theory of this problem is very deep from theory of matrices (or linear algebra). For the similarity of this problem we do some simplifications.

`A` and `B` will be matrices with the same size and square shape.

`A / B` is the first possible operation. This is normal slash. We called this division from B the right side.

`A \ B` is the second possible operation. This is back-slash. We called this division from A the left side.

Example

```
A = [4, 2, 2; 2, 0, 1; 7, 2, 1];
B = [3, 2, 4; 1, 2, 4; 2, 0, 1];

A / B
ans =
    3.5000   -2.5000   -2.0000
    0.0000   -0.0000    1.0000
    6.0000   -5.0000   -3.0000

A \ B
ans =
         0         0    0.2000
    0.5000   -1.0000   -2.0000
    1.0000    2.0000    3.6000

B / A
ans =
    2.0000    1.0000   -1.0000
    2.4000    0.6000   -1.4000
   -0.0000    1.0000    0.0000

B \ A
ans =
    1.0000    1.0000    0.5000
   -9.5000   -0.5000    0.2500
    5.0000    0.0000    0.0000
```

`A / B` = `A * inv (B)` – in mathematical expression it is $\mathbf{A} \cdot \mathbf{B}^{-1}$. Both cases are roughly the same, except it is computed in a different way.

`A \ B` = `inv (A) * B` – in mathematical expression it is $\mathbf{A}^{-1} \cdot \mathbf{B}$. Both cases are roughly the same, except it is computed in a different way.

Notice

The slash and back slash are defined on the real number too. Please try similar operation `2 / 3` and compare with `2 \ 3` ! Answer is similar too, `2 \ 3` = `3 / 2` !

Inversion function `inv` is not defined on the each matrix. Very simply is possible say, that this is division 1/matrix, but this non mathematical axiom!

Special case is the division one matrix by oneself
`A / A` = `A * inv (A)` = **1**
`A \ A` = `inv (A) * A` = **1**
`A / A` = `A \ A`
Symbol for this identity matrix may be vary - most often thick one **1**.

Example

```
A = [4, 2, 2; 2, 0, 1; 7, 2, 1]

A / A
ans =
    1.0000         0         0
    0.0000    1.0000    0.0000
         0         0    1.0000

A * inv(A)
ans =
    1.0000    0.0000   0.0000
    0.0000    1.0000   0.0000
    0.0000    0.0000   1.0000
```

In this condition is this division equivalent operation. All previous operation respect linear algebra laws. But for praxis is necessary to test it.

Division ./ and .\

Array right division `A ./ B` and array left division `A .\ B` is the matrix with elements `A(m,n) / B(m,n)` or `A(m,n) \ B(m,n)`. `A` and `B` must have the same size. Array right division `A ./ B` is the same as array left division `B .\ A`.

```
A = [4, 2, 2; 2, 0, 1; 7, 2, 1];
B = [3, 2, 4; 1, 2, 4; 2, 0, 1];

A ./ B
ans =
    1.3333    1.0000    0.5000
    2.0000         0    0.2500
    3.5000       Inf    1.0000

A .\ B
ans =
    0.7500    1.0000    2.0000
    0.5000       Inf    4.0000
    0.2857         0    1.0000

B ./ A
ans =
    0.7500    1.0000    2.0000
    0.5000       Inf    4.0000
    0.2857         0    1.0000

B .\ A
ans =
    1.3333    1.0000    0.5000
    2.0000         0    0.2500
    3.5000       Inf    1.0000
```

Notice

```
A ./ B  =  B .\ A
B ./ A  =  A .\ B
```

This is similar to the division of any number by another number, for example:

```
3./6                                   6./3
ans =                                  ans =
    0.5000                                  2

6.\3                                   3.\6
ans =                                  ans =
    0.5000                                  2
```

Addition +

+ is addition or unary plus. `A + B` adds A and B. `A` and `B` must have the same size, unless one is a scalar. A scalar can be added to a matrix of any size.

`A + B` = `B + A`, i.e. addition is commutative. This expression shows fully equivalent function. It is valid only for the same size of matrices.

Example

```
x = +5
x =
     5

A = [4, 2, 2; 2, 0, 1; 7, 2, 1];
B = [3, 2, 4; 1, 2, 4; 2, 0, 1];
```

```
C = A + B
C =
     7     4     6
     3     2     5
     9     2     2

H = B + A
H =
     7     4     6
     3     2     5
     9     2     2

A + x
ans =
     9     7     7
     7     5     6
    12     7     6
```

Subtraction –

- is subtraction or unary minus. `A - B` subtracts `B` from `A`. `A` and `B` must have the same size, unless one is a scalar. A scalar can be subtracted from a matrix of any size.

`A - B` $\neq$ `B - A`

`A - B` $=$ `-B + A`

Example

```
A = [4, 2, 2; 2, 0, 1; 7, 2, 1];
B = [3, 2, 4; 1, 2, 4; 2, 0, 1];

A - B
ans =
     1     0    -2
     1    -2    -3
     5     2     0

B - A
ans =
    -1     0     2
    -1     2     3
    -5    -2     0

-B + A
ans =
     1     0    -2
     1    -2    -3
     5     2     0

y = -3
y =
    -3

A - y
ans =
     7     5     5
     5     3     4
    10     5     4
```

The matrix C formed by adding the matrices A and B, see previous example: Addition +

```
C-B
ans =
     4     2     2
     2     0     1
     7     2     1 % Compare with matrix A!
```

Adding A to B and then subtracting B from the result C recovers A.

Powering ∧

^ is matrix power. A^p is A to the power p, if p is a scalar.

A^2 = A * A (result in this case is equal to classical matrix operation - multiplication *)

A.^2 = A .* A (result in this case is power of each element of matrix A)

Example

```
A = [4, 2, 2; 2, 0, 1; 7, 2, 1];

A^3
ans =
   244    92    92
   107    40    41
   307   112   111

A * A * A
ans =
   244    92    92
   107    40    41
   307   112   111

A.^3
ans =
    64     8     8
     8     0     1
   343     8     1

A .* A .* A
ans =
    64     8     8
     8     0     1
   343     8     1
```

Transpose '

' is matrix transpose. A' is the linear algebraic transpose of A. For complex matrices, this is the complex conjugate transpose.

Example

```
A = [4, 2, 2; 2, 0, 1; 7, 2, 1]
A =
     4     2     2
     2     0     1
     7     2     1
A'
ans =
     4     2     7
     2     0     2
     2     1     1

K=[1+i, 2-i; 1-i, 5-2i]
K =
   1.0000 + 1.0000i   2.0000 - 1.0000i
   1.0000 - 1.0000i   5.0000 - 2.0000i

K'
ans =
   1.0000 - 1.0000i   1.0000 + 1.0000i
   2.0000 + 1.0000i   5.0000 + 2.0000i
```

Transpose `.'`

`.'` is array transpose. `A.'` is the array transpose of `A`. For complex matrices, this does not involve conjugation.

Example

```
A = [4, 2, 2; 2, 0, 1; 7, 2, 1]
A =
     4     2     2
     2     0     1
     7     2     1
A.'
ans =
     4     2     7
     2     0     2
     2     1     1                        % Compare with the result of A'!

K=[1+i, 2-i; 1-i, 5-2i]
K =
   1.0000 + 1.0000i   2.0000 - 1.0000i
   1.0000 - 1.0000i   5.0000 - 2.0000i

K.'
ans =
   1.0000 + 1.0000i   1.0000 - 1.0000i
   2.0000 - 1.0000i   5.0000 - 2.0000i    % Compare with the result of K'!
```

Notice

The list of all basic operations with matrices is possible obtain after typing `help ops`.

2.5 Vector operations and tools

Vectors are matrices with one row or one column. The same rules apply for vectors and for matrices.

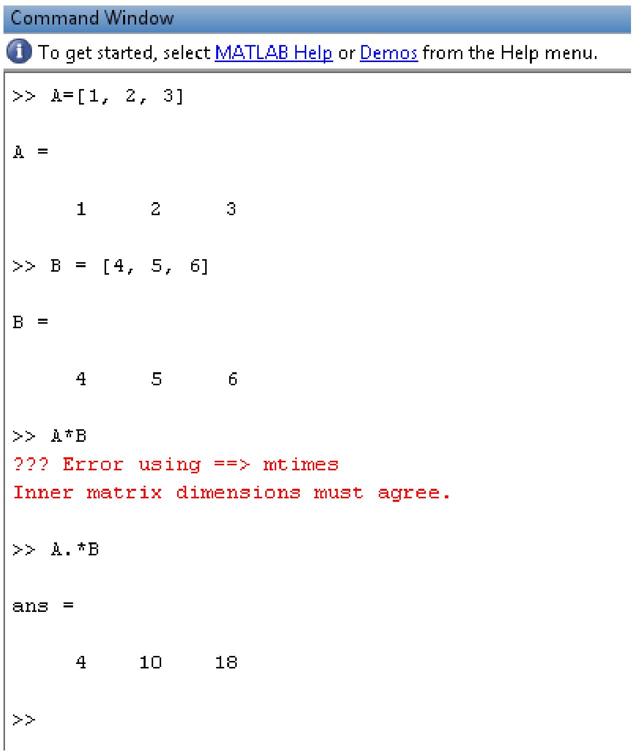
Example

We have linear row vectors a and b:

```
a = [1, 2, 3];
b = [4, 5, 6];

a.'
ans =
       1
       2
       3
b.'
ans =
       4
       5
       6
```

What will obtain like result of operation `a * b`? The right answer in this case is error, Fig. 2.5!


Command Window
To get started, select MATLAB Help or Demos from the Help menu.

```
>> A=[1, 2, 3]

A =

     1     2     3

>> B = [4, 5, 6]

B =

     4     5     6

>> A*B
??? Error using ==> mtimes
Inner matrix dimensions must agree.

>> A.*B

ans =

     4    10    18

>>
```

Figure 2.5: Differences between vector and array multiplication

If we use these structures in the dot operation `a .* b`, we obtain the right result, Fig. 2.5.

```
ans =
     4     10     18
```

Example

Some operations that can be performed with vectors a and b:

```
b .* a
ans =
     4    10    18

a * b.'
ans =
    32

b.' * a
ans =
     4     8    12
     5    10    15
     6    12    18

a.' * b
ans =
     4     5     6
     8    10    12
    12    15    18

b * a.'
ans =
    32

a + b
ans =
       5              7              9

format rat          % approximation by ratio of integers

a ./ b
ans =
       1/4            2/5            1/2

a .\ b
ans =
       4              5/2            2
```

The same problem appears in the case of multiplying two columns vectors. Non dot operation on these types of vectors is not defined. Dot operation gives us result. Dot multiplying is multiplying elements on elements. Each element multiplies with the same one on the same position in the second structure.

Example

The column vectors are transposed vectors `a` and `b` from the previous example:

```
a.' .* b.'
ans =
       4
      10
      18

b.' .* a.'
ans =
       4
      10
      18
```

2.6 Comparison MATLAB functions with alternative products

e - Euler's number

Euler's number is not defined in MATLAB. `exp(x)` is the exponential of the elements of `x`, i.e. e^x.

Example

```
exp(1)
ans =
    2.7183

exp([1,2,5,10])
ans =
  2.7183   7.3891   1.4841e+002   2.2026e+004
```

Euler's number is defined in Octave as predefined constant `e`.

2.7 Example from Electrical Engineering

2.7.1 The serial connection of resistors

a) Solve the equivalent resistance of series connected resistors $R_1 = 1\,\Omega$ and $R_2 = 3.2\,\Omega$, Fig. 2.6.

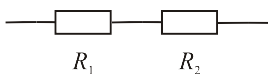


Figure 2.6: The serial connection of resistors R_1 a R_2

Solution

```
R1 = 1;
R2 = 3.2;
result = R1 + R2
```

Resistors values can be specified as a vector:

```
R =  [1,3.2];
result = R(1)+R(2)
```

where `R(1)` and `R(2)` is elements of the vector `R`,

or using built-in function `sum`

```
R =  [1,3.2];
result = sum(R)
```

where `sum(R)` is the sum of the elements of the vector `R`.

MATLAB always answers in these examples

```
result =    4.2000
```

b) Solve the equivalent resistance of series connected resistors $R_1 = 1\,\Omega$, $R_2 = 2\,\Omega$, $R_3 = 3\,\Omega$, $R_4 = 4\,\Omega$, and $R_5 = 5\,\Omega$, Fig. 2.7.

Solution

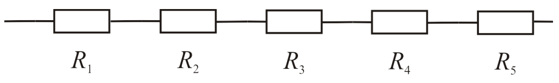


Figure 2.7: The serial connection of several resistors

```
R =  [1, 2, 3, 4, 5];
result = sum(R)
```

MATLAB answers

```
result =    15
```

2.7.2 The parallel connection of resistors

a) Solve the equivalent resistance of parallel connected resistors $R_1 = 1\,\Omega$ and $R_2 = 3.2\,\Omega$, Fig. 2.8.

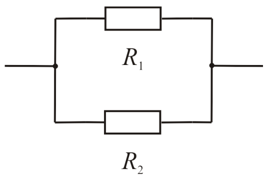


Figure 2.8: The parallel connection of resistors R_1 a R_2

Solution

```
R1 = 1;
R2 = 3.2;
result = (1/R1 + 1/R2)^(-1)
```

Resistors values can be specified as a vector (the same as in the previous example):

```
R = [1,3.2];
result = (1/R(1) + 1/R(2)).^(-1)
```

or it can be

```
result = sum(1./R).^(-1)
```

MATLAB always answers in this example

```
result =    0.7619
```

b) Solve the equivalent resistance of parallel connected resistors $R_1 = 1\,\Omega$, $R_2 = 2\,\Omega$, $R_3 = 3\,\Omega$, $R_4 = 4\,\Omega$, and $R_5 = 5\,\Omega$, Fig. 2.9 ($n = 5$).

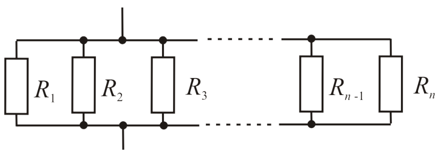


Figure 2.9: The parallel connection of several resistors

This is useful for several resistors connected in parallel, R is a vector of values of resistances.

Solution

```
R = [1, 2, 3, 4, 5];
result = sum(1./R).^(-1)
```

MATLAB answers

```
result =    0.4380
```

2.7.3 The series-parallel connection of resistors

Solve the equivalent resistance of series-parallel connected resistors $R_1 = 1\,\Omega, R_2 = 3.2\,\Omega$, $R_3 = 4\,\Omega$, Fig. 2.10

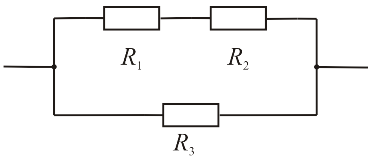


Figure 2.10: The series-parallel connection of resistors

Solution

```
R = [1, 3.2, 4];
result = (1/(R(1)+R(2)) + 1/R(3))^(-1);
```

Parentheses are used for enclosing of indexes of vector elements and for precedence in mathematical expressions.

MATLAB answers

```
result =    2.0488
```

2.7.4 Ohm's law

Find the current flowing through resistor $R_1 = 6\,\Omega$ using Ohm's law. The resistor R_1 is connected to the source $U_0 = 20\,\text{V}$, Fig. 2.11

Ohm's law states that the current through a conductor between two points is directly proportional to the potential difference across the two points.

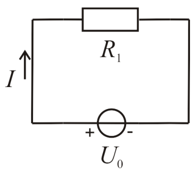


Figure 2.11: The simple circuit

Solution

```
U=20;
R1=6;
I=U/R1;
fprintf('The current through R1 is %3.2f Amperes.\n',I)
```

MATLAB answers

```
The current through R1 is 3.33 Amperes.
```

2.8 Test of matrix oriented commands

1. How is possible obtain the size of allocation variable?
2. How is possible set up the out visual format for engineering style?
3. What is variable with name ans?
4. How are two main function of colon in MATLAB?
5. How are two main function of semicolon in MATLAB?
6. How are two main function of comma in MATLAB?
7. How is the main difference between matrix and array?
8. How is the command for calculation the size of matrix?
9. What to do command `diag`?
10. How is the difference between the parentheses and brackets?
11. How is the command for summation of column elements?
12. What is identity matrix?
13. What to do command `inv`?

3 Matrix operations in the technical praxis

3.1 Solving the system of linear equations

Application of matrix in the real engineering is vary. One of the widest ranges using linear system is in analysis electrical circuit namely in the application of Kirchhoff's laws.

One way to solve the system of linear equations $\mathbf{Ax} = \mathbf{b}$ is with `x = inv(A)*b`, where matrix `A` is matrix of system coefficient and `b` is column vector of right side of system. A better way, from both an execution time and numerical accuracy standpoint, is to use the matrix division operator `x = A\b`. This produces the solution using Gaussian elimination, without forming the inverse.

Example

We have this system of linear equations:

$$3x + 4y + 2z = 10$$

$$2x - 2y - 4z = 2$$

$$10x - 8y + 2z = 8$$

Solution
We must reduce this real system of linear equations on two-matrix structure.

Main matrix `A` and the right side values into vector `b`

```
A=[3, 4, 2; 2, -2, -4; 10, -8, 2]
b=[10; 2; 8]
```

And the own solving is very simple: $\mathbf{A}^{-1} \cdot \mathbf{b}$. When you write this in MATLAB, it is:

```
x = inv(A) * b
```

It is very good to allocate to some variable name, for example `x`.

From definition of linear algebra is the same expression like in case

```
x = A \ b
```

We obtain the same results in both cases.

```
x =
    1.8261
    1.2319
   -0.2029
```

Attention

Using `A\b` instead of `inv(A)*b` for the system of 500 equations is two to three times as fast.

3.2 Advanced algebraic commands

- **Determinant**
  Last of the important matrix function named determinant. Mathematical expression of determinant is not so simple. The determinant of a matrix is a number, which can tell us something about the properties of the matrix.

  In MATLAB this function represent command `det`. It is scalar value.

  We obtain the determinant of matrix `A` using `det(A)`.

- **Rank of matrix**

  The `rank` function provides an estimate of the number of linearly independent rows or columns of a full matrix.

- **Condition number with respect to inversion**
  The condition number of a matrix measures the sensitivity of the solution of a system of linear equations to errors in the data. It gives an indication of the accuracy of the results from matrix inversion and the linear equation solution. Values of `cond(A)` near 1 indicate a well-conditioned matrix.

- **Matrix reciprocal condition number estimate**

  `c = rcond(A)` returns an estimate for the reciprocal of the condition of `A` in 1-norm using the LAPACK condition estimator. If `A` is well conditioned, `rcond(A)` is near 1.0. If `A` is badly conditioned, `rcond(A)` is near 0.0. Compared to `cond`, `rcond` is a more efficient, but less reliable, method of estimating the condition of a matrix.

Example

Solve the system of linear algebraic equations:

$$3x_1 + 4x_2 = 11$$

$$2x_1 - 5x_2 = -8$$

Solution

```
A = [3, 4; 2, -5];
b = [11; -8]
x = A \ b
```

MATLAB answers

```
x =
   1.0000
   2.0000

det([3,4;2,-5])
```

```
ans = -23
```
The determinant of matrix A is -23. $-23 \neq 0$, it is OK.

```
rank([3,4;2,-5])
```

```
ans =  2
```
This number represents 2 linearly independent rows of matrix A

```
rank([3,4,11;2,-5,-8])
```

```
ans =  2
```
This number represents 2 linearly independent rows of complete linear system with right sides of the equations.
Rank of matrix `[3,4;2,-5]` is the same as rank of matrix `[3,4,11;2,-5,-8]`, this is OK.

```
cond([3,4;2,-5])
```

```
ans =  1.7888
```
This means well-conditioned matrix.

```
rcond([3,4;2,-5])
```

```
ans =  0.36508
```
$0.36508 \neq 0$, i.e. well-conditioned matrix.

If the matrix of coefficients of the system is regular, then **the system has just one solution**.

Example

Solve the system of linear algebraic equations

$$x_1 + x_2 = 11$$

$$2x_1 + 2x_2 = -8$$

Solution

```
x = [1,1;2,2] \ [11;-8];
```

```
det([1,1;2,2])
```

```
ans = 0
```
The determinant of matrix A is 0. Pay attention to the solution of the system of equations.

```
cond([1,1;2,2])
```

```
ans =  3.8442e+16
```
This number is almost infinite, which means badly conditioned matrix.

```
rcond([1,1;2,2])
```

```
ans =  0
```
The value of zero means badly conditioned matrix.

```
rank([1,1;2,2])
```

```
ans =  1
```

This number represents only 1 linearly independent row of matrix.

```
rank([1,1,11;2,2,-8])
```

```
ans =  2
```

This number represents 2 linearly independent rows of complete linear system with right sides of the equations.
$1 \neq 2$, i.e. the rank of matrix `[1,1;2,2]` isn't the same as rank of matrix `[1,1,11;2,2,-8]`, this is badly.

If the matrix of coefficients of the system is singular and the vector of right sides is such that the equations are linearly independent, then **the system has no solution**.

Example

Solve the system of linear algebraic equations:

$$x_1 + x_2 = 10$$

$$2x_1 + 2x_2 = 20$$

Solution

```
x = [1,1;2,2] \ [10;20];
```

```
rank([1,1;2,2])
```

```
ans =  1
```

This number represents only 1 linearly independent row of matrix.

```
rank([1,1,10;2,2,20])
```

```
ans =  1
```

This number represents 1 linearly independent row of complete linear system with right sides of the equations.
$1 = 1$, i.e. rank of matrix `[1,1;2,2]` is the same as rank of matrix `[1,1,10;2,2,20]`, this is OK, but rank of matrix is 1 and we have 2 equations.

If the matrix of coefficients of the system is singular and the vector of right sides is such that the equations are linearly dependent, then **the system has infinitely many solutions**.

3.3 Example from Electrical Engineering

3.3.1 The DC circuit - application of Kirchhoff's laws

For the circuit shown in Fig. 3.1, find the current.

Given the following values:
$R_1 = 3\,\Omega, R_2 = 2\,\Omega, R_3 = 3\,\Omega, R_4 = 6\,\Omega, R_5 = 5\,\Omega, R_6 = 6\,\Omega, R_7 = 5\,\Omega, U_{01} = 48\,\mathrm{V}$, $U_{02} = 43.2\,\mathrm{V}$.

Kirchhoff's current law (Kirchhoff's first law): The principle of conservation of electric charge implies that: At any node (junction) in an electrical circuit, the sum of currents flowing into that node is equal to the sum of currents flowing out of that node, or: The algebraic sum of currents in a network of conductors meeting at a point is zero.

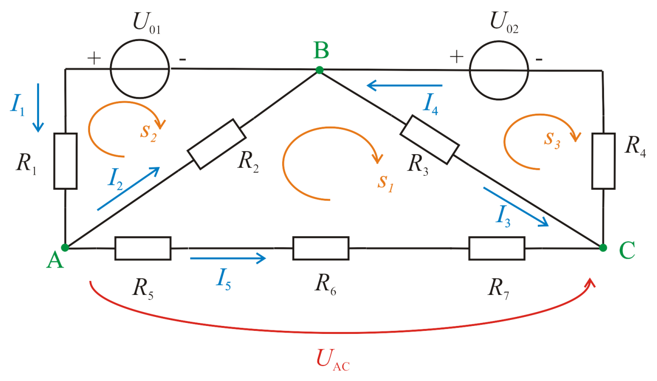


Figure 3.1: The DC circuit

Kirchhoff's voltage law (Kirchhoff's second law): The principle of conservation of energy implies that: The directed sum of the electrical potential differences (voltage) around any closed network is zero, or: More simply, the sum of the emfs in any closed loop is equivalent to the sum of the potential drops in that loop, or: The algebraic sum of the products of the resistances of the conductors and the currents in them in a closed loop is equal to the total electromotive force available in that loop.

Kirchhoff's laws for this circuit:

Kirchhoff's current law:

node A: $I_1 - I_2 - I_5 = 0$,
node C: $I_3 - I_4 + I_5 = 0$.

Kirchhoff's voltage law:

loop s_1: $R_2 I_2 + R_3 I_3 - R_5 I_5 - R_6 I_5 - R_7 I_5 = 0$,
loop s_2: $-R_1 I_1 - R_2 I_2 + U_{01} = 0$,
loop s_3: $-R_3 I_3 - R_4 I_4 + U_{02} = 0$.

Modifying of equations:

$1 \cdot I_1 - 1 \cdot I_2 + 0 \cdot I_3 + 0 \cdot I_4 - 1 \cdot I_5 = 0$
$0 \cdot I_1 + 0 \cdot I_2 + 1 \cdot I_3 - 1 \cdot I_4 + 1 \cdot I_5 = 0$
$0 \cdot I_1 + R_2 I_2 + R_3 I_3 - 0 \cdot I_4 + (-R_5 - R_6 - R_7) I_5 = 0$
$-R_1 I_1 - R_2 I_2 + 0 \cdot I_3 + 0 \cdot I_4 + 0 \cdot I_5 = -U_{01}$
$0 \cdot I_1 + 0 \cdot I_2 - R_3 I_3 - R_4 I_4 + 0 \cdot I_5 = -U_{02}$

Using matrix:

Solution

```
R1 = 3; R2 = 2; R3 = 3; R4 = 6; R5 = 5; R6 = 6; R7 = 5;
U01 = 48; U02 = 43.2;
A = [   1,   -1,    0,    0,        -1;...
        0,    0,    1,   -1,         1;...
        0,   R2,   R3,    0, -R5-R6-R7;...
      -R1,  -R2,    0,    0,         0;...
        0,    0,  -R3,  -R4,         0];
b = [0; 0; 0; -U01; -U02];
currents = A \ b
```

The results obtained from MATLAB are

```
currents =
          10.3000
           8.5500
           3.6333
           5.3833
           1.7500

det(A)
ans =  864
```

The determinant $864 \neq 0$, this is OK.

```
rank(A)
ans =  5

rank([A,b])
ans =  5
```

Rank of matrix `A` is the same as rank of matrix `[A,b]` with right sides of the equations, this is OK.

```
cond(A)
ans =  12.106
```

$\Rightarrow$ well-conditioned matrix

```
rcond(A)
ans =  0.052288
```

$0.052288 \neq 0 \Rightarrow$ well-conditioned matrix

If the matrix of coefficients of the system `A` is regular, then the system has just one solution.

These currents are the solution of systems of linear algebraic equations:

$$I_1 = 10.3000\,A$$

$$I_2 = 8.5500\,A$$

$$I_3 = 3.6333\,A$$

$$I_4 = 5.3833\,A$$

$$I_5 = 1.7500\,A$$

3.3.2 The DC circuit 2 - application of Kirchhoff's laws

For the circuit shown in Fig. 3.2, find the current.

Given the following values:
$R_1 = 15.3\ \Omega$, $R_2 = 9.9\ \Omega$, $R_3 = 16.5\ \Omega$,$R_4 = 2.6\ \Omega$, $R_5 = 7.5\ \Omega$, $R_6 = 12.1\ \Omega$, $R_7 = 1.5\ \Omega$, $R_8 = 9\ \Omega$, $R_9 = 11.6\ \Omega$, $R_{10} = 9.2\ \Omega$, $R_{11} = 14.8\ \Omega$, $R_{12} = 12.1\ \Omega$, $U_0 = 50\,\mathrm{V}$.

Kirchhoff's laws for this circuit:

Kirchhoff's current law:

node A: $I_1 - I_2 - I_3 = 0$,

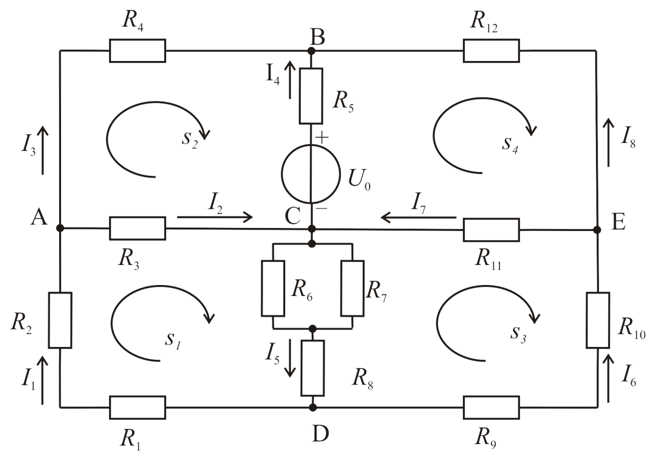


Figure 3.2: The DC circuit

node B: $I_3 + I_4 + I_8 = 0$,
node D: $-I_1 + I_5 - I_6 = 0$,
node E: $I_6 - I_7 - I_8 = 0$.

Kirchhoff's voltage law:
loop s_1: $R_1 I_1 + R_2 I_1 + R_3 I_2 + R_{67} I_5 + R_8 I_5 = 0$,
loop s_2: $-R_3 I_2 + R_4 I_3 - R_5 I_4 + U_0 = 0$,
loop s_3: $-R_9 I_6 - R_{10} I_6 - R_{11} I_7 - R_8 I_5 - R_{67} I_5 = 0$,
loop s_4: $R_5 I_4 - R_{12} I_8 + R_{11} I_7 - U_0 = 0$,
where $R_{67} = \frac{1}{\frac{1}{R_6} + \frac{1}{R_7}} = \frac{R_6 R_7}{R_6 + R_7}$.

Solution

```
U0=50;
R1=15.3;R2=9.9;R3=16.5;R4=2.6;R5=7.5;R6=12.1;R7=1.5;R8=9.0;R9=11.6;
R10=9.2;R11=14.8;R12=12.1;
R67=(1/R6+1/R7)^(-1);
A=[1,-1,-1,0,0,0,0,0;...
0,0,1,1,0,0,0,1;...
-1,0,0,0,1,-1,0,0;...
0,0,0,0,0,1,-1,-1;...
(R1+R2),R3,0,0,(R67+R8),0,0,0;...
0,-R3,R4,-R5,0,0,0,0;...
0,0,0,0,-(R67+R8),-(R9+R10),-R11,0;...
0,0,0,R5,0,0,R11,-R12];
b=[0;0;0;0;0;-U0;0;U0];
currents = A \ b
```

The results obtained from MATLAB are

```
currents =
  -0.55439
   1.34666
  -1.90105
   3.04498
  -0.79823
  -0.24384
   0.90008
  -1.14392
```

```
det(A)
ans =  1.2494e+006
```

The determinant $1.2494e + 006 \neq 0$, this is OK.

```
cond(A)
ans =  35.7236
```

$\Rightarrow$ well-conditioned matrix

```
rcond(A)
ans =  0.0159
```

$0.0159 \neq 0 \Rightarrow$ well-conditioned matrix

```
rank(A)
ans =     8

rank([A,b])
ans =     8
```

Rank of matrix `A` is the same as rank of matrix `[A,b]` with right sides of the equations, this is OK.

If the matrix of coefficients of the system `A` is regular, then the system has just one solution.

These currents are the solution of systems of linear algebraic equations:

$$I_1 = -0.55439\,A$$

$$I_2 = 1.34666\,A$$

$$I_3 = -1.90105\,A$$

$$I_4 = 3.04498\,A$$

$$I_5 = -0.79823\,A$$

$$I_6 = -0.24384\,A$$

$$I_7 = 0.90008\,A$$

$$I_8 = -1.14392\,A$$

Many examples of electrical practice is in [3].

3.4 Test of matrix operations in the technical praxis

1. Solve this system of linear equations:

$$7x_1 + 12x_2 + 3x_3 = 15$$

$$6x_1 + 5x_2 - 4x_3 = -9$$

$$3x_1 + 9x_2 - 8x_3 = 4$$

2. Solve this system of linear equations:

$$7x_1 + x_3 = 5$$

$$9x_1 - 2x_2 = -4$$

$$6x_1 + 3x_2 - 9x_3 = -7$$

3. Find the current for the circuit shown in Fig. 3.3 with the voltage source $U_0 = 60\,\mathrm{V}$ and the values of these resistors: $R_1 = 20\,\Omega$, $R_2 = 8\,\Omega$, $R_3 = 11\,\Omega$,$R_4 = 17\,\Omega$, $R_5 = 21\,\Omega$, $R_6 = 12\,\Omega$, $R_7 = 18\,\Omega$, $R_8 = 19\,\Omega$, $R_9 = 11\,\Omega$, $R_{10} = 12\,\Omega$, $R_{11} = 14\,\Omega$, $R_{12} = 12\,\Omega$.

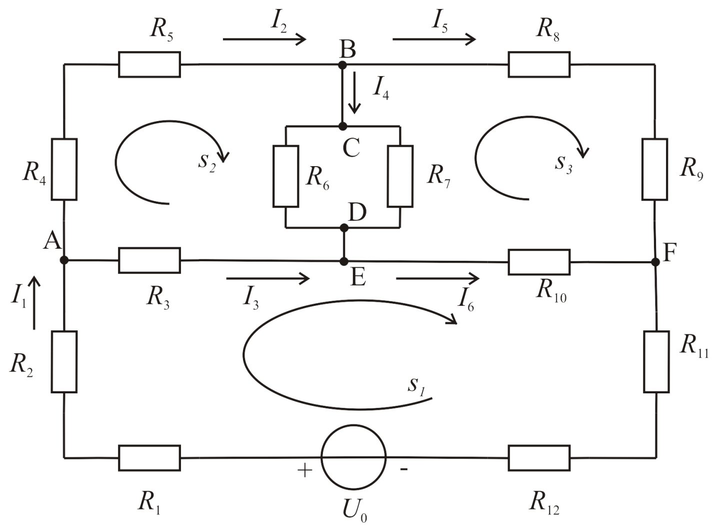


Figure 3.3: The DC circuit

4 Complex numbers and complex matrices

4.1 Introduction to complex mathematics in MATLAB

The complex number is given

```
c = -6 + 8i;
```

The command `real(c)` returns the real part of `c`.
The command `imag(c)` returns the imaginary of `c`.
The command `abs(c)` returns the complex modulus (magnitude) of `c`.
`(real(c).^2+imag(c).^2).^(1/2)` is the complex modulus (magnitude) of `c`, too.
The command `angle(c)` returns the phase angles, in radians, of `c`.

Attention

Using commands `atan` and `atan2`
The command `atan(imag(c)./real(c))` is inverse tangent, returns result in radians only in the first and fourth quadrant, from $\frac{-\pi}{2}$ to $\frac{\pi}{2}$. The results in the second and third quadrant are badly!
The command `atan2(imag(c),real(c))` is the four quadrant arctangent from $-\pi$ to π and results are OK.
Compare with results of the command `angle(c)`.

Example

```
c = -6 + 8i;

real(c)
ans =
    -6

imag(c)
ans =
     8

abs(c)
ans =
    10

(real(c).^2+imag(c).^2).^(1/2)
ans =
    10
```

```
angle(c)
ans =
    2.2143

atan(imag(c)./real(c))
ans =
   -0.9273 % attention - bad result, the result have to be in second quadrant!

atan2(imag(c),real(c)) % the result is OK. Compare with result of the command angle(c).
ans =
    2.2143
```

4.2 Usage of complex matrices

When you solve problems from range electrical engineering, you may meet the complex number like elements of matrix.

```
A = [1+i, 2-i; 1-i, 5-2i];
```

For complex matrices holds the same mathematical operations like for their normal version. Only some special kind is dot operation with apostrophe.

If we use for complex matrix apostrophe without point, obtain other matrix like in case using the dot symbol. Compare `COMPLEX '` and `COMPLEX .'`.

Example

```
A.'
ans =
   1.0000 + 1.0000i   1.0000 - 1.0000i
   2.0000 - 1.0000i   5.0000 - 2.0000i

A'
ans =
   1.0000 - 1.0000i   1.0000 + 1.0000i
   2.0000 + 1.0000i   5.0000 + 2.0000i
```

Try to explore difference on your own examples!

Function for complex numbers processing

The command `real(K)` returns the real part of elements of matrix `K`

```
K=[1+i,-5+6i;-3-4i,2-7i];
real(K)

ans =
 1    -5
-3     2
```

The command `imag(K)` returns the imaginary part of elements of matrix `K`.

```
imag(K)

ans =
     1     6
    -4    -7
```

`abs(K)` is the complex modulus (magnitude) of the elements of K,
`abs(K)` is the same as `(real(K).^2+imag(K).^2).^(1/2)`.

```
abs(K)

ans =
    1.4142    7.8102
    5.0000    7.2801
```

The command `angle(K)` returns the phase angles, in radians, of a matrix with complex elements.

```
angle(K)

ans =
    0.7854    2.2655
   -2.2143   -1.2925
```

The command `angle(K)*180/pi` is conversion of phase angles from radians to degrees.

```
angle(K)*180/pi

ans =
   45.0000  129.8056
 -126.8699  -74.0546
```

The command `conj(K)` returns the complex conjugate of K, i.e. `real(K) - i*imag(K)`

```
conj(K)

ans =
   1.0000 - 1.0000i  -5.0000 - 6.0000i
  -3.0000 + 4.0000i   2.0000 + 7.0000i
```

4.3 Example from Electrical Engineering

4.3.1 The AC circuit - application of Kirchhoff's laws

For the circuit shown in Fig. 4.1, find the currents.

This example discusses sinusoidal steady state. The circuit is analyzed by converting into the frequency domain and by using the Kirchhoff's laws. We use complex numbers to calculate the result.

$u_0(t) = U_m sin(\omega t + \varphi)$ corresponds to $\overline{U}_0 = U_m e^{\mathrm{j}\varphi}$

Given the following values:
$R_1 = 40\,\Omega, R_2 = 30\,\Omega, R_3 = 10\,\Omega$, $L = 0.1\,\mathrm{mH}$, $C = 5\,\mu\mathrm{F}$, $u_0(t) = 10 sin(\omega t + 30°)\mathrm{V}$, $\omega = 2\pi f$, $f = 50\,\mathrm{Hz}$
$u_0(t) = 10 sin(\omega t + 30°)\mathrm{V}$ corresponds to $\overline{U}_0 = 10e^{\mathrm{j}30°}\,\mathrm{V}$

Kirchhoff's laws for this circuit:
Kirchhoff's current law:
node A: $\overline{I}_1 - \overline{I}_2 - \overline{I}_3 = 0$
Kirchhoff's voltage law:
loop s_1: $R_1\overline{I}_1 + R_2\overline{I}_2 + \mathbf{j}\omega L\overline{I}_2 + R_3\overline{I}_1 - \overline{U}_0 = 0$
loop s_2: $-R_2\overline{I}_2 - \mathbf{j}\omega L\overline{I}_2 - \mathbf{j}\frac{1}{\omega C}\overline{I}_3 = 0$
Solution

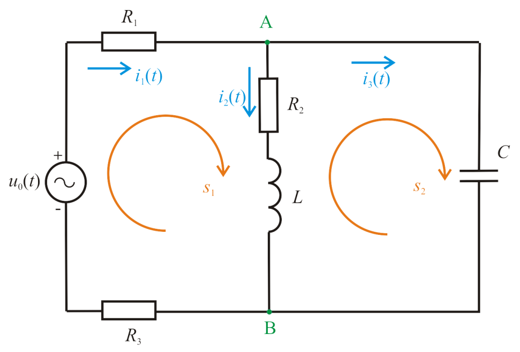


Figure 4.1: The AC circuit

```
R1=40;
R2=30;
R3=10;
L=0.1e-3;  % conversion from miliHenry to Henry
C=200e-6;  % conversion from microFarad to Farad
f = 50;
w = 2*pi*f;
Uo=10*exp(j*30*pi/180);  % conversion from degrees to radians
A=[ 1, -1, -1;...
(R1+R3),(R2+j*w*L),0; ...
0,-R2-j*w*L,(1./(j*w*C))];
b=[0;Uo;0];
x=A\b; % calculation
Im=abs(x);  % magnitudes of the currents
Irms=abs(x)/sqrt(2);  % rms value of the currents
fi=angle(x)*180/pi;  % phase constants (conversion to degrees)

for n=1:length(x)
fprintf('\n i%d = %5.2f sin(wt + (%6.2f)) A\n',n,Im(n),fi(n));

end
```

The result (waveforms of currents) obtained from MATLAB is

```
i1 =  0.17 sin(wt + ( 42.38)) A
i2 =  0.08 sin(wt + (-19.72)) A
i3 =  0.15 sin(wt + ( 70.34)) A
```

where character w corresponds to the angular frequency ω. For more information about command `fprintf`, see the section 9.1.

4.3.2 Graphical representation of the results - phasor diagram of currents

Draw the phasor diagram of currents from the circuit in Fig. 4.1.

Solution using previous results

```
compass(x)
```

The plotting command `compass(x)` draws a graph that displays the vectors with components `(real(x), imag(x))` as arrows emanating from the origin (Fig. 4.2). For more information about command `compass`, see the section 5.2.

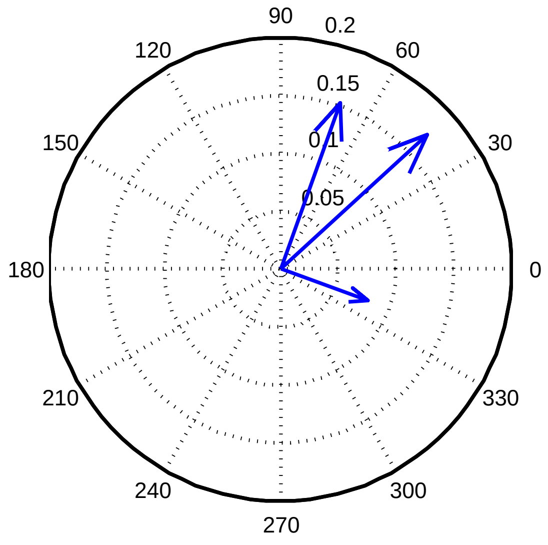


Figure 4.2: The phasor diagram of currents

4.4 Test of basic commands for complex mathematics

1. How do you get the real and imaginary part numbers of the matrix `C`, where `C = [6+7i, -5+3i; 9-5i, -4-7i]`?

2. Create a transposed matrix to the matrix `C` from the previous question.

3. The complex number is given: `c = 3+4i`. We use the command `abs(c)`. What answers MATLAB?

4. The complex number is given: `c = 2+2i`. We use the command `angle(c)`. What answers MATLAB?

5 Graphs and graphical commands

5.1 Basic graphical commands and libraries

Graphical commands and application classify on the basic and advanced scale of instructions. For usual mathematical expression of usual calculations and equations be enough for the basic scale one. This group of instruction included for example commands like `figure`, `plot`, `plot3`, `axis`, `grid`, `colors` and many more. The advanced group included more sophisticated commands like `interp1`, `interp2`, `alpha`, `shading`, `colormap` and many more. Using of some instruction on the specific application will be shown next.

The next example is necessary write due the M-file editor. In the Command window it is possible too, but this is not very comfortable and in this case of modification, this way is bad.

Example

We use basic graphical command `plot`.

In the M-file editor we entered next text.

```
%Program for testing basic graphical commands
%-------------------------------------------
%Graph of sinus function
%-------------------------------------------
%Definition of range X axis (or time basis)
t=-pi:pi/100:pi;
%-------------------------------------------
y=sin(t);
%graphical instruction
plot(t,y);
grid
```

And what this script does? It is very simple example of the definition of the X axe in the range from $-\pi$ number to π number with the step $\pi/100$. This is very smooth step for graphic expression. In the next part it is calculation of y like sinus of t (like time step). And the end of this script is dedicated graphic output of final graph of whole function - `plot(t, y)`, where y is defined as `sin(t)`, Fig. 5.1.

Command `grid` of is used only for background gridlines in Figure window.

Notice

This is only notice from basic, the percentage symbol % is used for commentary in program without effect for compiling.

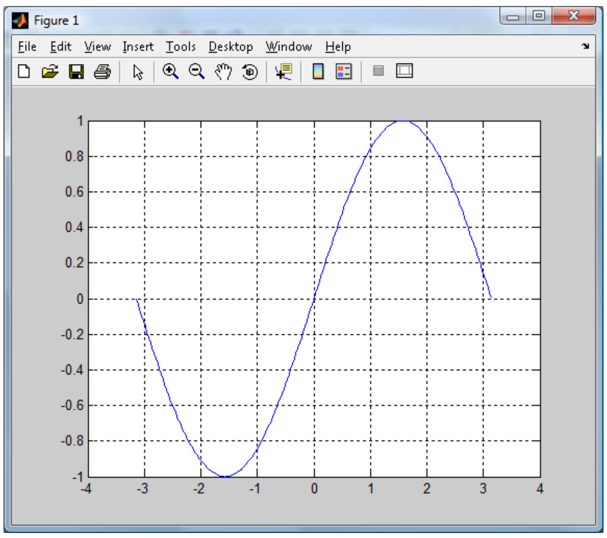


Figure 5.1: Graphical output from MATLAB

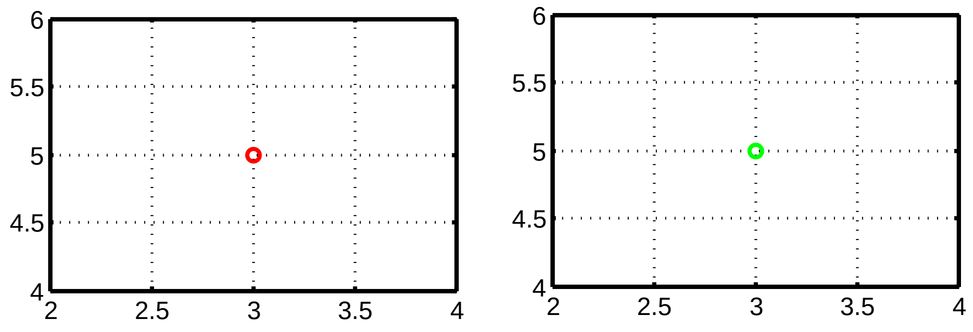


Figure 5.2: The point with coordinates $x = 3$, $y = 5$ (left figure)
and the complex number $3 + 5\mathbf{i}$ (right figure)

Example

Plotting of point with coordinates $x = 3$, $y = 5$.

```
plot(3,5,'ro')
grid
```

where `plot(x,y,LineSpec)` plots point defined by the `x`, `y`, `LineSpec` triplets, where `LineSpec` specifies the marker symbol and color. For example: `'ro'` plots the red circle. The string `LineSpec` is not mandatory.

Graphical output from MATLAB is depicted in left part of Fig 5.2.

Example

Plotting of complex number $3 + 5\mathbf{i}$.

```
plot(3+5i,'go')
grid
```

The right part of Fig 5.2 shows graphical output from MATLAB. For complex `c`, `plot(c)` is equivalent to `plot(real(c),imag(c))`. The string `'go'` provides view of complex number as the green circle. For detailed information about the complex numbers, see the section 4.1.

More graphs in one Figure window

In this case, we can use the command `plot` again. The command `plot(`x_1,y_1,x_2,y_2,...,x_n,y_n`)` plots each vector y versus vector x on the same axes.

`plot(`x_1,y_1,`LineSpec`,x_2,y_2,`LineSpec`...,x_n,y_n,`LineSpec`) plots lines defined by the x_n,y_n,`LineSpec` triplets, where `LineSpec` specifies the line type, marker symbol, and color using character string made from one element, for example: `'c+:'` plots a cyan dotted line with a plus at each data point. Details can be obtained using `help plot`.

Example

Graph of functions: $y_1 = -\sin^2(x)$ and $y_2 = \cos(x^2 - 5)$, where x is from -2π to 2π.

Solution

```
x = [-2*pi:0.05:2*pi];
y1 = -sin(x).^2;
y2 = cos(x.^2 - 5);
plot(x,y1,'k',x,y2,'r')
xlabel('x')
ylabel('y')
legend('y_1 = -sin^2(x)','y_2 = cos(x^2 - 5)')
```

The graphical output from MATLAB is depicted in Fig. 5.3. The first line is black (using string `'k'` that is included in the command `plot`), the second line is red (string `'r'`).

The commands `xlabel('...')` and `ylabel('...')` add description beside the x-axis and y-axis on the current axes.

The command `legend(string 1,string 2, ..., string n)` puts a legend on the current plot using the specified strings as labels.

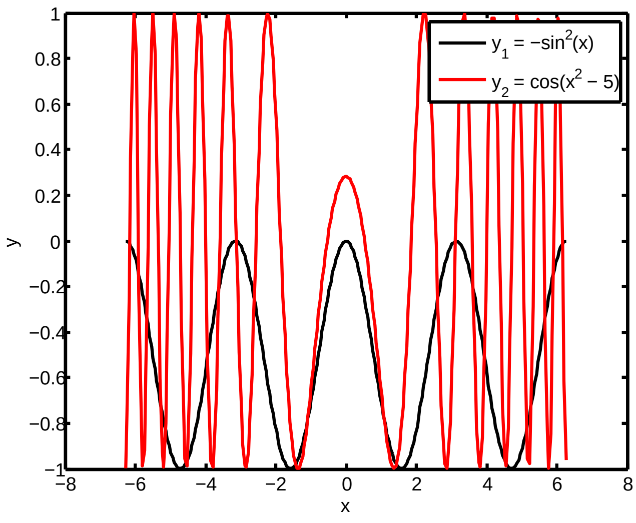


Figure 5.3: Two lines in one Figure window

Another possibility of more graphs in one Figure window is to use the command `hold`. The command `hold on` keeps the existing graph and adds the next one to it. The command `hold off` undoes the effect of `hold on`.

The same result as in the previous example can be obtained as follows:

```
x = [-2*pi:0.05:2*pi];
y1 = -sin(x).^2;
y2 = cos(x.^2 - 5);
plot(x,y1,'k')
hold on
plot(x,y2,'r')
hold off
xlabel('x')
ylabel('y')
legend('y_1 = -sin^2(x)','y_2 = cos(x^2 - 5)')
```

The displaying of each function $y_1 = -\sin^2(x)$ and $y_2 = \cos(x^2 - 5)$ independently in a separate part of the Figure window will now be shown.

The command `subplot(m,n,p)` breaks the Figure window into an m-by-n matrix of small subwindows and selects the p-th subwindow for the current plot.

Example

```
x = [-2*pi:0.05:2*pi];
y1 = -sin(x).^2;

subplot(1,2,1)
plot(x,y1,'k')
xlabel('x')
ylabel('y')
title('y_1 = -sin^2(x)')

y2 = cos(x.^2 - 5);

subplot(1,2,2)
plot(x,y2,'r')
xlabel('x')
ylabel('y')
title('y_2 = cos(x^2 - 5)')
```

The command `title('text')` adds text at the top of the current axis as graph title.

MATLAB answers with the opening of the Figure window, Fig. 5.4.

Other examples are presented in the following sections of this chapter.

If it is necessary to have two lines with different scales on the y-axis, we can use the command `plotyy`, which plots graph with y tick labels on both left and right side.

`plotyy`(x_1,y_1,x_2,y_2) plots y_1 versus x with y-axis labelling on the left and y_2 versus x with y-axis labelling on the right.

Example

Graph of functions: $y_1 = \sin(x)$ and $y_2 = 100\cos(x)$, where x is from 0 to 4π.

```
x=[0:0.01:4*pi];
y1=sin(x);
y2=100*cos(x);
plotyy(x,y1,x,y2)
```

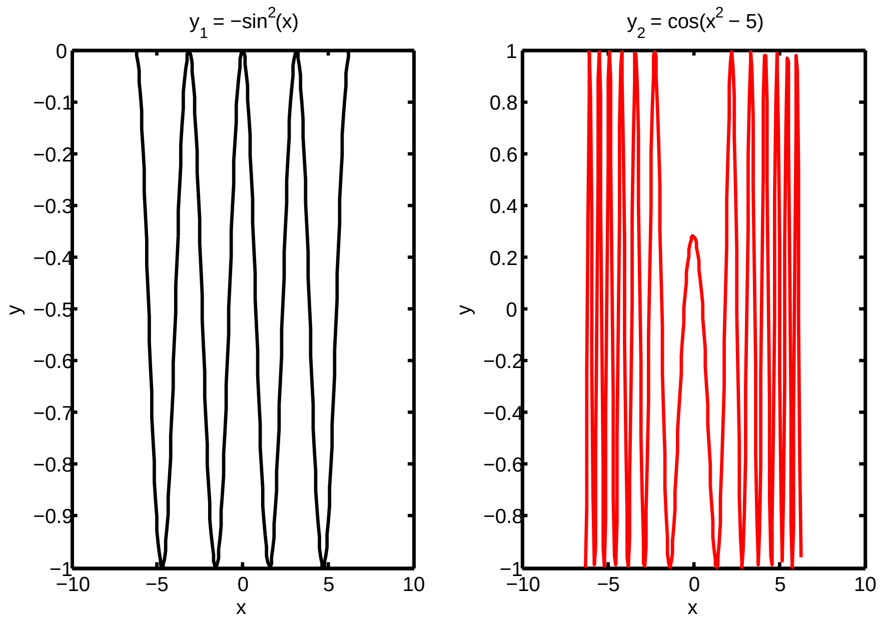


Figure 5.4: Two sub-windows in one Figure window

The graphical output from MATLAB is depicted in Fig. 5.5.

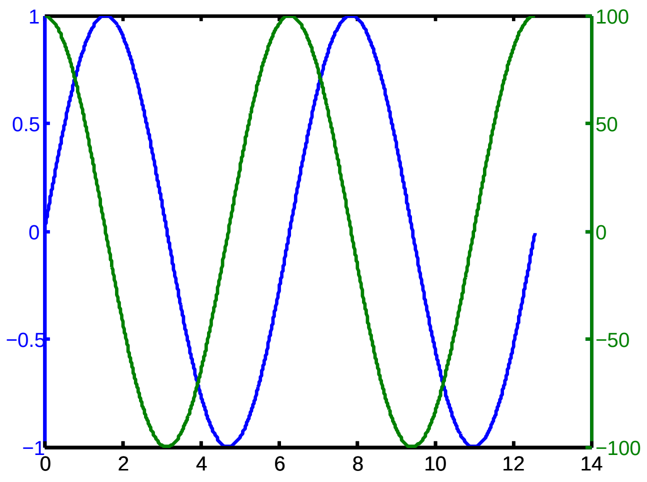


Figure 5.5: Two lines in one Figure window with y-axes on both left and right side

5.2 Further types of graphs

Semilogarithmic scale plots, log scale plot

`semilogx` and `semilogy` plot data as logarithmic scales for the x- and y-axis, respectively. `loglog` is log-log scale plot.

Example

Let's write commands to view graph of the function

$$y = 200e^{\frac{2x}{x+2}} + \sin(x)$$

where x is from 0 to 100.

Solution

We use first linearly equally spaced points between 0 and 100 (`linspace`), and logarithmically equally spaced points between 1 and 100 (`logspace`) for x.

```
x=linspace(0,100,650);
y=200.*(exp(2.*x./(x+2))+sin(x));

subplot(5,1,1)
plot(x,y,'.')

subplot(5,1,2)
semilogx(x,y,'.')

subplot(5,1,3)
semilogy(x,y,'.')

x=logspace(0,2,650);
y=200.*(exp(2.*x./(x+2))+sin(x));

subplot(5,1,4)
semilogx(x,y,'.')

subplot(5,1,5)
loglog(x,y,'.')
```

The graphical output from MATLAB is depicted in Fig. 5.6. The linearly spaced elements of x is preferable when using graphical commands `plot` and `semilogy`, which have linear scales for the x-axis. The logarithmically spaced elements of x is preferable when using graphical commands `emilogx` and `loglog`, which have logarithmic scales for the x-axis.

Polar coordinates graph

`polar` accepts polar coordinates. `polar(`ϑ, r`)` makes a plot using polar coordinates of the angle ϑ, in radians, versus the radius r.

Example

Graph of functions: $r = \sin(4\varphi)\cos(4\varphi)$, where φ is from 0 to 2π.

```
fi = 0:0.01:2*pi;
polar(fi,sin(4*fi).*cos(4*fi),'r')
```

Answer of MATLAB is shown in Fig. 5.7

`pol2cart` performs transform Polar to Cartesian coordinates. `[X,Y] = pol2cart(fi,r)` transforms corresponding elements of data stored in polar coordinates (angle φ, radius r) to Cartesian coordinates x, y.

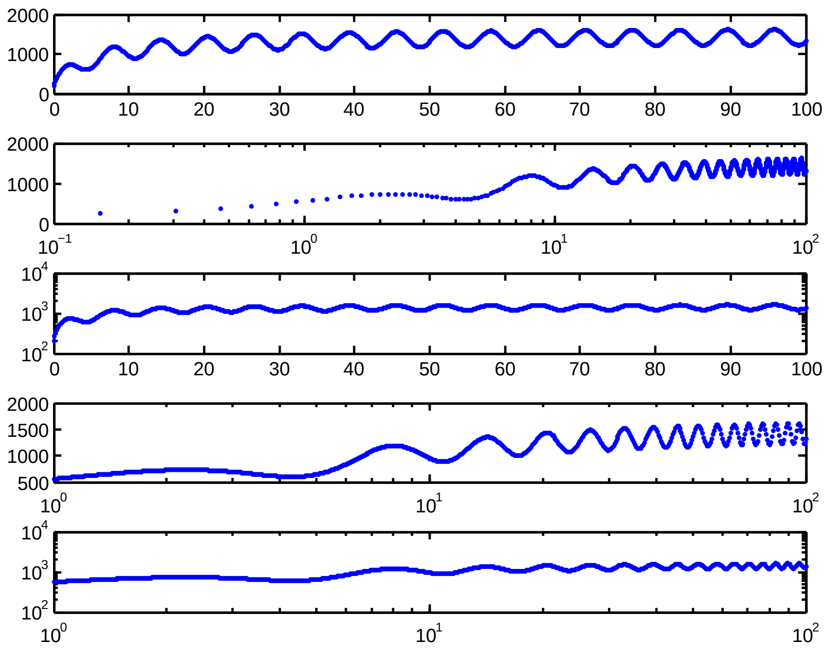


Figure 5.6: Using plot, semilogx, semilogy and loglog

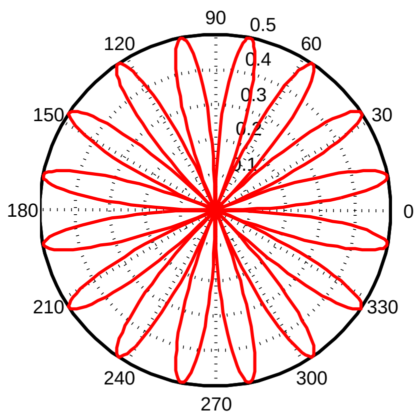


Figure 5.7: Polar coordinate plot

Example

Vectors `fi` and `r` are known from the previous example.

```
[x,y]=pol2cart(fi,r);
plot(x,y)
```

The result is depicted in Fig. 5.8

3-D graphs

3-D line plot

The `plot3` function displays a three-dimensional plot of a set of data points.

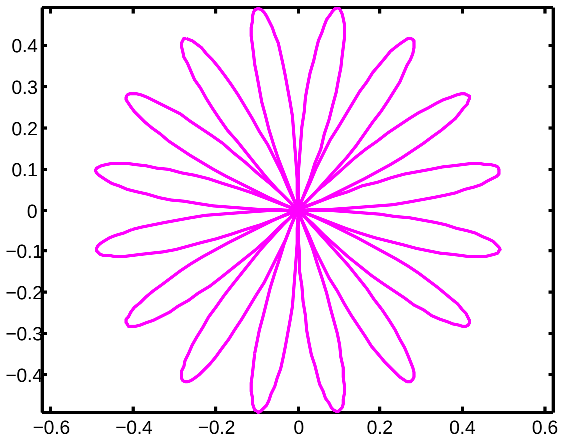


Figure 5.8: Cartesian coordinate plot (the same as in Fig. 5.7)

`plot3`(x_1,y_1,z_1,x_2,y_2,z_2,...,x_n,y_n,z_n), where x_n,y_n,z_n are vectors, plots one or more lines (according to n) in three-dimensional space through the points whose coordinates are the elements of x_n,y_n,z_n.

Example

Graph the curve described by the parametric equations: $x = t$, $y = t\sin(t)$, $z = t\cos(t)$, where t is from 0 to 20π.

```
t=[0:0.1:20*pi]
plot3(t,t.*sin(t),t.*cos(t),'g')
xlabel('x')
ylabel('y')
zlabel('z')
```

The graphical output from MATLAB is depicted in Fig. 5.9. The curve has a green color, which is caused by string `'g'` in the command `plot3`.

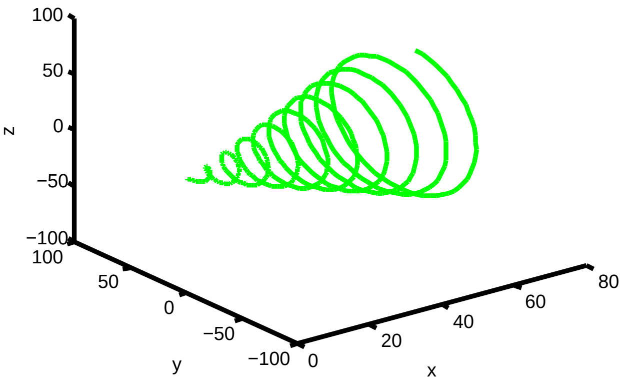


Figure 5.9: The three-dimensional plot $x = t$, $y = t\sin(t)$, $z = t\cos(t)$

Surfaces

The command mesh displays 3-D mesh surface. mesh(X,Y,Z) plots the colored parametric mesh defined by three matrix arguments.

The command meshz displays 3-D mesh surface as mesh, but with curtain.

Example

View 3-D mesh surface of the function of two variables: $z = \sin(x^2 + y^2)$, where x is from -1 to 1, y is from -1.3 to 1.3.

Solution

We need matrix arguments, so we use the command meshgrid. The command [X,Y] = meshgrid(x,y) transforms the domain specified by vectors x and y into arrays X and Y that can be used for the evaluation of functions of two variables and 3-D surface plots. The rows of the output array X are copies of the vector x and the columns of the output array Y are copies of the vector y.

Matrices X and Y are used for the calculation of matrix Z.

```
x = linspace(-1,1,20);
y = linspace(-1.3,1.3,20);
[X,Y] = meshgrid(x,y);
Z = sin(X.^2+Y.^2);

subplot(1,2,1);
mesh(X,Y,Z)
xlabel('x')
ylabel('y')
zlabel('z')

subplot(1,2,2);
meshz(X,Y,Z)
xlabel('x')
ylabel('y')
zlabel('z')
```

The graphical output of MATLAB is shown in Fig. 5.10.

The command surf displays 3-D colored surface. surf(X,Y,Z) plots the colored parametric surface defined by three matrix arguments.

The command surfl is the same as surf except that it draws the surface with highlights from a light source.

Example

This example is a continuation of the previous example. View 3-D colored surface the function of two variables $z = \sin(x^2 + y^2)$.

Solution

Two a nested loops are used instead of the command meshgrid, which would also be suitable.

```
x = linspace(-1,1,20);
y = linspace(-1.3,1.3,20);
for m = 1:length(x)
    for n = 1:length(y)
```

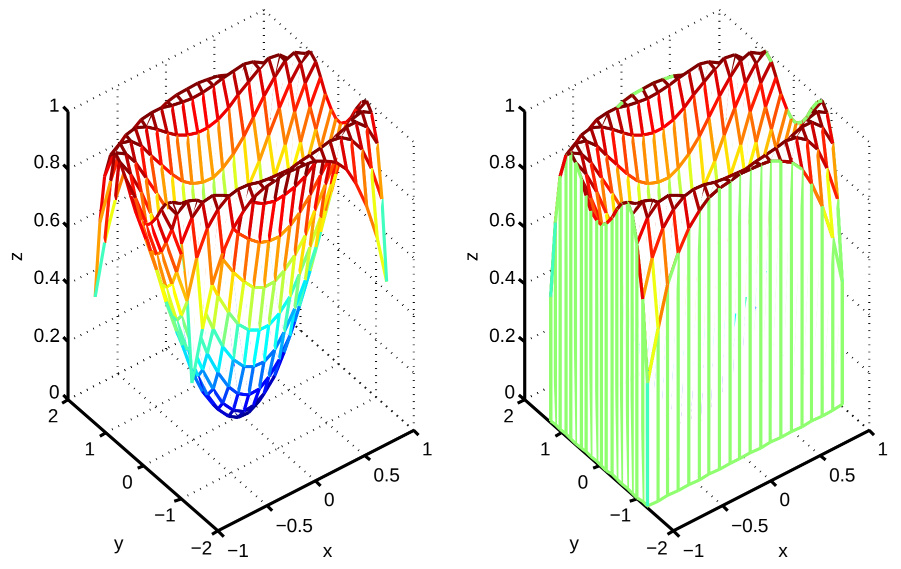


Figure 5.10: The three-dimensional mesh surface of the function of two variables: $z = \sin(x^2 + y^2)$

```
        Z(m,n) = sin(x(m)^2 + y(n)^2);
    end
end
subplot(1,2,1);
surf(X,Y,Z)
xlabel('x')
ylabel('y')
zlabel('z')

subplot(1,2,2);
surfl(X,Y,Z)
xlabel('x')
ylabel('y')
zlabel('z')
```

The result is depicted in Fig. 5.11.

Contour Plots

Contour plot of matrix can be displayed using the command `contour`.

The command `contourf` is the same as `contour` except that the areas between contours are filled with colors according to the Z-value for each level.

The command `contour3` displays 3-D contour plot.

`colorbar` appends a color bar (color scale) to the current axes of surface graphs and contour plots.

There are combination mesh/contour plot and surf/contour plot. The combination mesh/contour plot is `meshc`, which is the same as `mesh` except that a contour plot is drawn beneath the `mesh`. Similarly, `surfc` is the same as `surf` except that a contour plot is drawn beneath the `surf`.

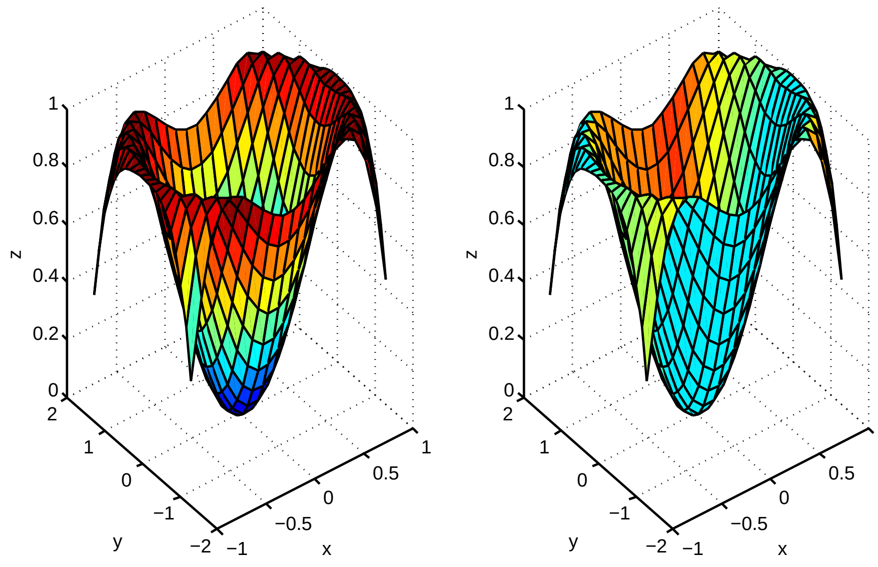


Figure 5.11: The three-dimensional colored surface of the function of two variables: $z = \sin(x^2 + y^2)$

Example

This example is a continuation of the previous example. View various contour plots the function of two variables $z = \sin(x^2 + y^2)$.

Solution

Again, we obtain matrix arguments `X` and `Y` using the command `meshgrid`.

```
x = linspace(-1,1,20);
y = linspace(-1.3,1.3,20);
[X,Y] = meshgrid(x,y);
Z = sin(X.^2+Y.^2);

subplot(3,2,1);
contour(X,Y,Z)
colorbar

subplot(3,2,2);
contourf(X,Y,Z)
colorbar

subplot(3,2,3);
contour3(X,Y,Z)

subplot(3,2,4);
contour3(X,Y,Z,'k') % contours are black
hold on
mesh(X,Y,Z) % addition the mesh surface to the contour plot
hold off

subplot(3,2,5);
meshc(X,Y,Z)

subplot(3,2,6);
surfc(X,Y,Z)
```

The graphical output from MATLAB is depicted in Fig. 5.12.

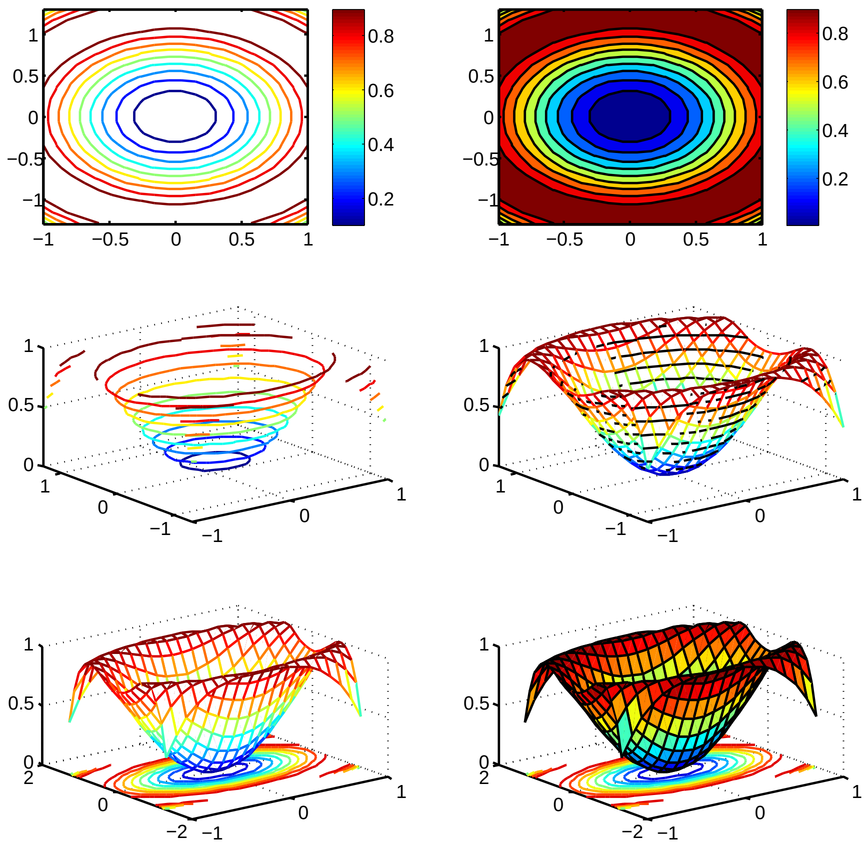


Figure 5.12: The contour plots of the function of two variables:
$z = \sin(x^2 + y^2)$

Direction plots

The command `quiver(X,Y,F,G)` plots vectors as arrows with components (f,g) at the points (x,y). The matrices X,Y,F,G must all be the same size and contain corresponding position and velocity components.

Example

View the gradient field of the function of two variables $z = \cos(x^2+y^2)$, where x, y is from -1 to 1.

Solution

We obtain matrix arguments X and Y using the command `meshgrid` and we calculate matrix Z.

We have to calculate gradient field of the function $z = \cos(x^2 + y^2)$. The command `[FX,FY] = gradient(F)`, where F is a matrix, returns the and components of the two-dimensional numerical gradient. The output of `gradient(F)` is two matrices FX,FY.

```
x = [-1:0.2:1];
y = x;
```

```
[X,Y] = meshgrid(x,y);
Z=cos(X.^2+Y.^2);

subplot(1,2,1);
contour(X,Y,Z)

[FX,FY]=gradient(Z);

hold on
quiver(X,Y,FX,FY)
hold off
```

The gradient field as arrows using `quiever` is shown in left part of Fig. 5.13.

The command `quiver3(X,Y,Z,F,G,H)` plots vectors as arrows with components (`f`,g,h) at the points (x,y,z). The matrices X,Y,Z,F,G,H must all be the same size and contain the corresponding position and velocity components.

Example

This example is a continuation of the previous example. Compute and display 3-D surface normals of the function of two variables $z = \cos(x^2 + y^2)$.

Solution

We assume that the matrices are `X`, `Y` and `Z` are known, see previous example. We have to compute 3-D surface normals of the function $z = \cos(x^2 + y^2)$. The `surfnorm` function computes surface normals for the surface defined by `X`, `Y`, and `Z`.

`[Nx,Ny,Nz] = surfnorm(X,Y,Z)` returns the components of the three-dimensional surface normals for the surface.

```
subplot(1,2,2)
mesh(X,Y,Z)

[NX,NY,NZ]=surfnorm(X,Y,Z);
hold on
quiver3(X,Y,Z,NX,NY,NZ)
hold off
```

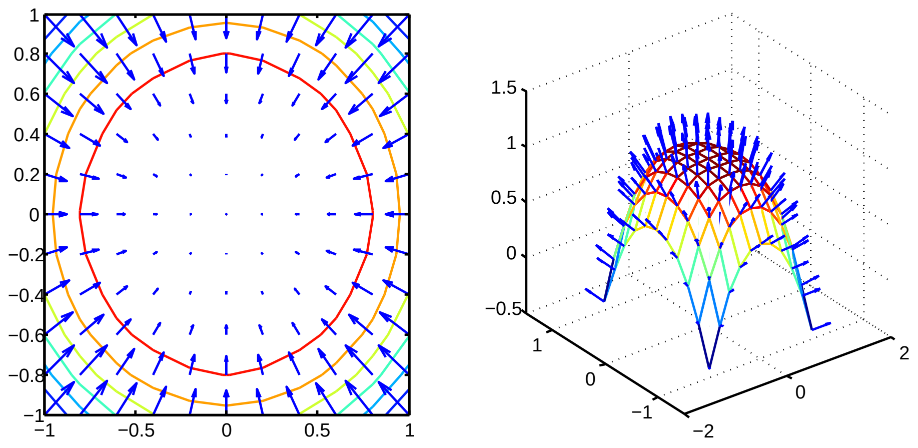


Figure 5.13: The gradient field and surface normals of the function of two variables: $z = \cos(x^2 + y^2)$

Right part of Fig. 5.13 depicts surface normals of the function of two variables $z = \cos(x^2 + y^2)$.

The `compass(x,y)` graph displays the vectors with components (x,y) as arrows emanating from the origin. x,y are in Cartesian coordinates and plotted on a circular grid.
If c is a complex number, the command `compass(c)` displays c as well as `compass(real(c), imag(c))`.

Example

```
x = round(randn(1,8).*5)
y = round(randn(1,8).*5)

subplot(1,2,1)
compass(x,y)

C = [3+4i, -2+6i, -5-1i, 3-2i]
subplot(1,2,2)
compass(C, 'r')
```

MATLAB answers

```
x =
     5     6    -4     0    -6    -6     0     8
y =
    -4     2    -1     6    -5     0     3     6
C =
   3.0000 + 4.0000i  -2.0000 + 6.0000i  -5.0000 - 1.0000i   3.0000 - 2.0000i
```

Fig. 5.14 shows the graphical output from MATLAB.

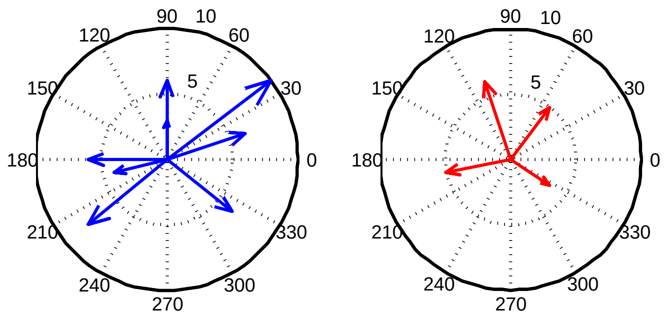


Figure 5.14: The compass graph as arrows emanating from the origin

A `feather` plot displays vectors emanating from equally spaced points along a horizontal axis. We express the vector components relative to the origin of the respective vector.

Example

Create a feather plot.

```
subplot(2,1,1)
x = [0:5,4:-1:0]
y = [-5:5]
feather(x,y)
```

```
subplot(2,1,2)
x = linspace(-2*pi,2*pi,40);
feather(x,sin(x))
grid
```

MATLAB answers

```
x =
     0      1      2      3      4      5      4      3      2      1      0
y =
    -5     -4     -3     -2     -1      0      1      2      3      4      5
```

The graphical output is shown in Fig. 5.15.

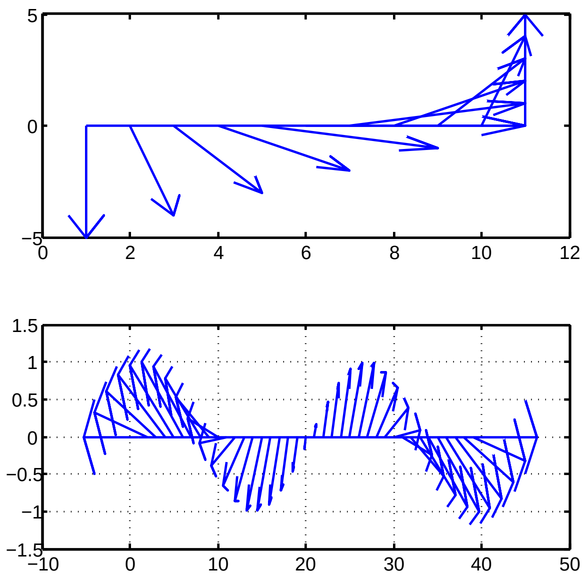


Figure 5.15: The Feather plot

Proportional representation of values expressing chart

`bar`, `barh` draws bar graph (vertical and horizontal)
`bar3`, `bar3h` plots 3-D bar chart (vertical and horizontal)
`pie` draws a pie chart
`pie3` draws a 3-D pie chart

Example

CEZ Group's electricity production in 2007 - 2011 is presented in Table:

year	2007	2008	2009	2010	2011
electricity [GWh]	73 793	67 595	65 344	68 433	69 209

Graphically display statistics on electricity production.

Solution

```
p = [73793, 67595, 65344, 68433, 69209];
subplot(4,2,1)
```

```
bar(p)
subplot(4,2,2)
barh(p,'r')
subplot(4,2,3)
pie(p)
subplot(4,2,4)
pie(p,{'2007','2008','2009','2010','2011'})
subplot(4,2,3)
pie(p,[0,0,0,0,1])
subplot(4,2,4)
pie(p,[1,1,1,1,1],{'2007','2008','2009','2010','2011'})
subplot(4,2,5)
bar3(p,'y')
subplot(4,2,6)
bar3h(p,'g')
subplot(4,2,7)
pie3(p)
subplot(4,2,8)
pie3(p,{'2007','2008','2009','2010','2011'})
```

Statistics on electricity production are depicted in Fig. 5.16.

5.3 Settings the graph properties

The command `axis` determines axis scaling and appearance.

`axis([xmin xmax ymin ymax])` sets the limits for the x- and y-axis of the current axes of 2-D plots.

`axis([xmin xmax ymin ymax zmin zmax])` sets the x-, y-, and z-axis limits of the current axes of 3-D plots.

`v = axis` returns a row vector containing scaling factors for the x-, y-, and z-axis. `v` has four or six components depending on whether the current axes is 2-D or 3-D, respectively.

`axis equal` sets the aspect ratio so that the data units are the same in every direction. The aspect ratio of the x-, y-, and z-axis is adjusted automatically according to the range of data units in the x, y, and z directions.

`axis image` is the same as axis equal except that the plot box fits tightly around the data.

`axis square` makes the current axes region square (or cubed when three-dimensional). This option adjusts the x-axis, y-axis, and z-axis so that they have equal lengths and adjusts the increments between data units accordingly.

`axis normal` automatically adjusts the aspect ratio of the axes and the relative scaling of the data units so that the plot fits the figure's shape as well as possible.

`axis off` turns off all axis lines, tick marks, and labels.

`axis on` turns on all axis lines, tick marks, and labels.

Example

```
x = [-pi/2 : 0.1 : pi/2];
plot(x, sin(x))
axis
ans =
    -2     2    -1     1
```

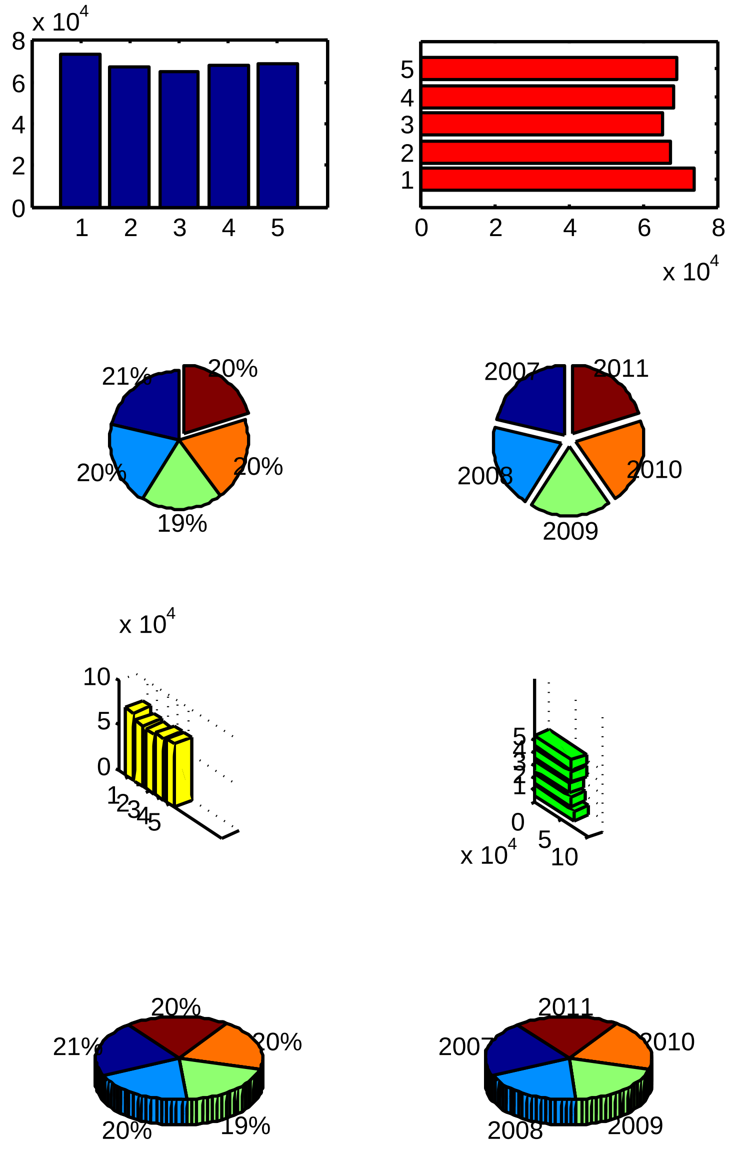


Figure 5.16: Examples of bar and pie charts

This means that range of x-axis in the plot is from -2 to 2 and range of y-axis in the plot is from -1 to 1.

The command `grid` displays grid lines for 2-D and 3-D plots.
`grid on` adds major grid lines to the current axes.
`grid off` removes major and minor grid lines from the current axes.
`grid` toggles the major grid visibility state.
`grid minor` toggles the minor grid lines of the current axes.

Example

```
x = [-2*pi : 0.1 : 2*pi];
subplot(2,1,1)
```

```
plot(x, sin(x))
grid
subplot(2,1,2)
plot(x, sin(x))
grid minor
```

The result is depicted in Fig. 5.17.

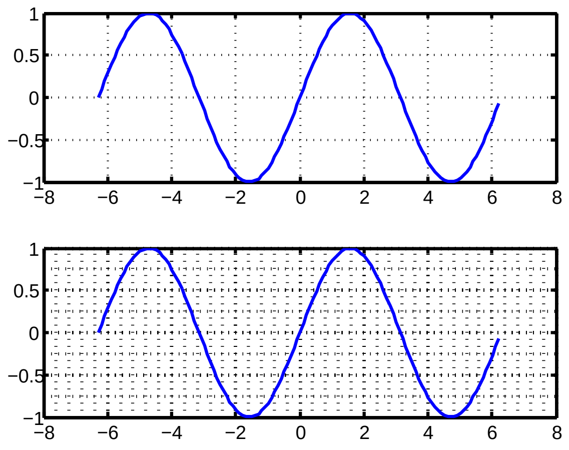


Figure 5.17: The plot of function sin(x) with grid and grid minor

5.4 Using graphical output in electrical engineering examples

5.4.1 Volt-ampere characteristic of linear resistor

Draw a Volt-ampere characteristic of linear resistor. Assume that values of current I and voltage U on a linear resistor were measured (see table):

U [V]	0	1	2	3
I [mA]	0	10	20	30

Solution

```
U=[0:3];
I=[0:10:30];
plot(U,I,'o-r')
title('Volt-ampere characteristic')
xlabel('I[mA]')
ylabel('U[V]')
```

The command `plot(U,I)` plots vector `I` versus vector `U`. Various line types, plot symbols and colors may be obtained with a character string, for example: `'o-r'` plots a red solid line with a circle at each data point. The command `title('text')` adds text at the top of the graph. The commands `xlabel('...')` and `ylabel('...')` add description beside the X-axis and Y-axis on the current axes.

The result obtained from MATLAB is graphical output, Fig. 5.18.

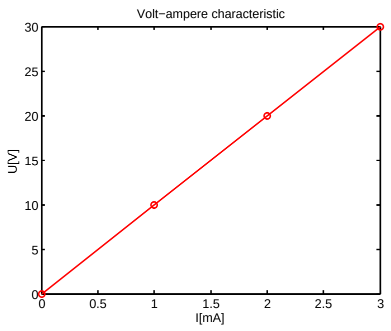


Figure 5.18: Volt-ampere resistor characteristic

5.4.2 Transfer function of electronic filter

Passive implementations of linear filters are based on combinations of resistors (R), inductors (L) and capacitors (C). These types are collectively known as passive filters, because they do not depend upon an external power supply and/or they do not contain active components such as transistors. A filter in which a capacitor provides a path to ground, presents less attenuation to low-frequency signals than high-frequency signals and is therefore a low-pass filter. Resistors on their own have no frequency-selective properties, but are added to inductors and capacitors to determine the time-constants of the circuit, and therefore the frequencies to which it responds.

For the circuit shown in Fig. 5.19 find the transfer function in the range from 1 rad/s to 10^5 rad/s and draw transfer function (dependence of magnitude and phase angle of transfer function on the angular frequency ω) into the two sub-windows in semi-logarithmic coordinates with the axis with logarithmic division. Given $R = 10\ \Omega$, $C = 1$ mF.

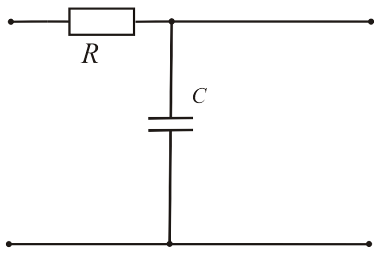


Figure 5.19: The filter

Transfer function:

$$K_U = \frac{U_2}{U_1} = \frac{-\mathrm{j}\cdot\frac{1}{\omega C}}{R-\mathrm{j}\frac{1}{\omega C}} = \frac{\frac{1}{\mathrm{j}\omega C}}{R+\frac{1}{\mathrm{j}\omega C}}$$

Solution

For characteristics of filter, we use the command `logspace` (logarithmically spaced vector) for creating `w`. During calculation, it is necessary to apply the operation "`./`" and "`.*`", because `w` is the vector. The command `abs(Ku)` calculates the complex modulus (magnitude) of the elements of `Ku`. The command `angle(Ku)` returns the phase angles, in radians.

The command `subplot(2,1,p)` breaks the Figure window into an 2-by-1 matrix of small subwindows and selects the p-th subwindow for the current plot.

The command `semilogx(...)` is the same as the command `plot(...)`, except a logarithmic (base 10) scale is used for the x-axis. The command `grid` adds major grid lines to the current axes.

```
w = logspace(0, 5, 100);
R = 10;
C = 1e-3;
Ku = 1./(j.*w.*C)./(R+1./(j.*w.*C));  % operation ./ is required
Ku_abs = abs(Ku);
Ku_angle = angle(Ku)/pi*180;  % conversion to degrees

subplot(2,1,1);
semilogx(w, Ku_abs);
grid
xlabel('\omega (rad/s)');
ylabel('|K_u|');

subplot(2,1,2);
semilogx(w, Ku_angle);
grid
xlabel('\omega (rad/s)');
ylabel('angle K_u');
```

The result obtained from MATLAB is in Fig. 5.20.

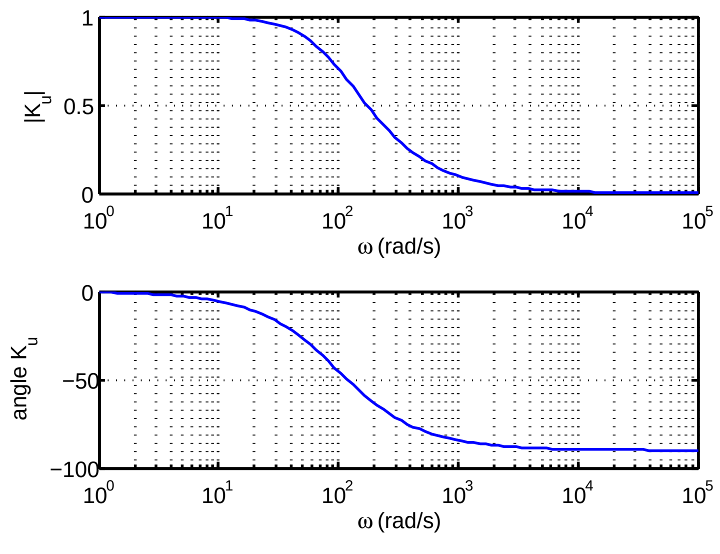


Figure 5.20: Characteristics of the filter

5.4.3 The balanced tree-phase AC voltage source

Draw a plot of balanced 3-phase system.

For a balanced 3-phase system, voltages have the same magnitude and they are out of phase by 120°.

$$u_U = U_m \sin(\omega t)$$

$$u_V = U_m \sin(\omega t - 120^\circ)$$
$$u_W = U_m \sin(\omega t - 240^\circ)$$

where U_m is magnitude and $\omega = 2\pi f$ is angular frequency.

Solution

We use command `plot`. The command `hold on` keeps the existing graph and adds the next one to it. The command `hold off` undoes the effect of `hold on`.

```
t=[0:1e-4:5e-2];
uU=10.*sin(2.*pi.*50.*t);
uV=10.*sin(2.*pi.*50.*t-2*pi/3);
uW=10.*sin(2.*pi.*50.*t-4*pi/3);
plot(t,uU,'r')
hold on     % this command keeps the existing graph and adds the next one to it.
plot(t,uV,'b')
plot(t,uW,'g')
hold off % this command undoes the effect of "hold on"
xlabel('t[s]')
ylabel('u_U[V], u_V[V], u_W[V]')
title('The balanced tree-phase system','FontSize',18)
```

The result obtained from MATLAB is depicted in Fig 5.21

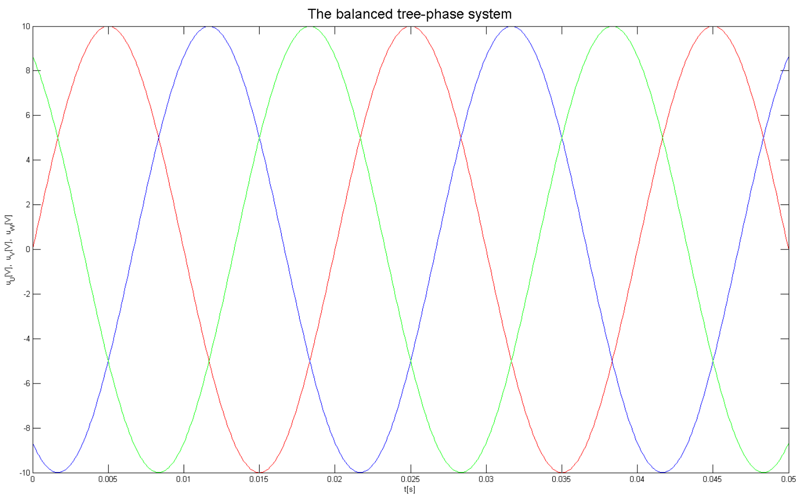


Figure 5.21: The balanced tree-phase system

5.4.4 The plotting of gradient

Given two functions $z_1 = x^2 + 0.5y^2$ and $z_2 = -x^2 - 0.5y^2$. Draw 3-D mesh surface and contour plot with gradient fields of these functions for x, y from -10 to 10.

Solution

The command `[X,Y] = meshgrid(x,y)` transforms the domain specified by vectors `x` and `y` into arrays `X` and `Y` that can be used for 3-D surface plots. The rows of the output array `X` are copies of the vector `x` and the columns of the output array `Y` are copies of the vector `y`. This is followed by calculation of `Z` using matrices `X` and `Y`. The matrix `Z` has the same dimensions as the arrays `X` and `Y`.

The command `subplot(2,2,p)` breaks the Figure window into an 2-by-2 matrix of small subwindows and selects the p-th subwindow for the current graph and the command `mesh(X,Y,Z)` plots the colored parametric mesh defined by three matrix arguments. The command `contour` is contour plot of matrix `Z` treating the values in `Z` as heights above a plane. The command `hold on` keeps the existing graph of contour and adds the next graph of gradient to it. The command `[GX,GY] = gradient(Z)` returns the numerical gradient of the matrix `Z`. The command `quiver(X,Y,GX,GY)` plots vectors as arrows with components `(GX,GY)` at the points `(X,Y)`. The command `hold off` undoes the effect of `hold on`.

```
x=-10:2:10;
y=x;
[X,Y]=meshgrid(x,y);
Z1=X.^2+0.5.*Y.^2;
Z2=-X.^2-0.5.*Y.^2;
subplot(2,2,1)
mesh(X,Y,Z1)
xlabel('x')
ylabel('y')
zlabel('z_1')

subplot(2,2,2)
mesh(X,Y,Z2)
xlabel('x')
ylabel('y')
zlabel('z_2')

subplot(2,2,3)
contour(X,Y,Z1)
hold on
[GX,GY]=gradient(Z1);
quiver(X,Y,GX,GY)
hold off

subplot(2,2,4)
contour(X,Y,Z2)
hold on
[GX,GY]=gradient(Z2);
quiver(X,Y,GX,GY)
hold off
```

The result obtained from MATLAB is depicted in Fig 5.22.

5.5 Test of graphical commands, settings and using graphs

1. Write commands to view graph of a function $y_1 = \sin^3(x)$ and $y_2 = \sin(x^3)$.
2. What is the difference between commands `plot3` and `mesh`?
3. Draw a curve by parametric equations $x = \sin(t)$, $y = t$, $z = \cos(t)$, where t is from 0 to 20π.
4. Draw graph of function $z = e^{-x^2-y^2}$, where x, y are from -3 to 3.
5. View contour plot and gradient field of function $z = e^{-x^2-y^2}$, where x, y are from -3 to 3.

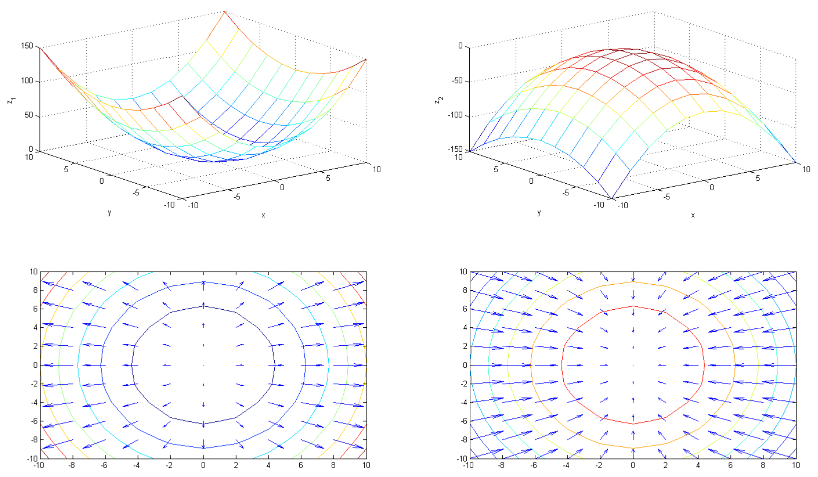


Figure 5.22: Contours and gradient of functions $z_1 = x^2 + 0.5y^2$ and $z_2 = -x^2 - 0.5y^2$

6 Polynomial regressions and interpolations

6.1 Basic commands for solving polynomials

Polynomial functions are the basic function in many scientific subjects. Very useful are for example polynomial regression. In MATLAB is strictly defined the right rules for writing polynomials. Each polynomial must be transpose into vector construct from its coefficient including zero numbers.

The polynomial 1:

$$4x^3 + 3x + 1$$

It can write:

$$4x^3 + 0x^2 + 3x^1 + 1x^0$$

The transcription into vector form is

```
V1 = [4, 0, 3, 1];
length(V1)
ans =
     4
```

The elements of vector are coefficients of polynomial. The number of elements in the vector is 4, i.e. one more than the degree of the polynomial (the polynomial degree is 3).

The polynomial 2:

$$8x^7 - 3x^3 + 12$$

It can write:

$$8x^7 + 0x^6 + 0x^5 + 0x^4 - 3x^3 + 0x^2 + 0x^1 + 12x^0$$

The transcriptions into vector forms is in this case:

```
V2 = [8,0,0,0,-3,0,0,12];
length(V2)
ans =
     8
```

The elements of vector are coefficients of polynomial. The number of elements in the vector is 8, i.e. one more than the degree of the polynomial (the polynomial degree is 7).

Zero coefficients are very important, their missing cause the bad result. It is very often error from the apprentice students in MATLAB courses.

Example

What polynomial represents the vector `V3 = [8, -3, 12]` ?

Solution

$8x^2 - 3x^1 + 12x^0$, i.e. $8x^2 - 3x + 12$.
The number of elements in the vector is 3, the polynomial degree is 2. Compare with the polynomial 2, which is represented by the vector `V2`.

6.2 Polynomial functions and libraries

Complete list of all polynomial functions obtains after typing `help polyfun`.

There are namely these five very important commands: `roots`, `poly`, `polyval`, `conv`, `deconv`

The command `roots` - find polynomial roots. `roots(V)` computes the roots of the polynomial whose coefficients are the elements of the vector `V`.

This function calculates the roots of polynomial. If we have the polynomial $2x^2+3x-3$ defined by its coefficients, we may construct vector from these coefficients

```
V = [2, 3, -3];
roots(V)
ans =
   -2.1861
    0.6861
```

`roots(V)` returns roots of equation $2x^2 + 3x - 3 = 0$. The solution of this equation is $x_1 = -2.1861$, $x_2 = 0.6861$.

Example

The classic quadratic equation: $2x^2 - 2x + 2 = 0$

Solution

Like the first step in solving this equation is construction of the vector of coefficients `KV`. The vector `KV` is in this case `KV = [2,-2,2]`.

In second step is using the function roots on the `KV` vector:

```
KV = [2,-2,2];
roots(KV)
```

And MATLAB writes on the command window results:

```
ans =

   0.5000 + 0.8660i
   0.5000 - 0.8660i
```

This means that the solving of quadratic equation $2x^2 - 2x + 2 = 0$ is:
$x_1 = 0.5 + 0.8660i$
$x_2 = 0.5 - 0.8660i$.
Roots are in the complex range.

Example

Solve the equation: $3.1x^3 + 4x^5 = 7x^2 - 11$
Solution
Equation should be modified: $4x^5 + 0x^4 + 3.1x^3 - 7x^2 + 0x^1 + 11 = 0$

```
p = [4, 0, 3.1, -7, 0, 11];
roots(p)
ans =
   0.97274 + 0.57470i
   0.97274 - 0.57470i
  -0.51238 + 1.44129i
  -0.51238 - 1.44129i
  -0.92071 + 0.00000i
```

The solving of equation is: $x_1 = 0.97274 + 0.57470i$, $x_2 = 0.97274 - 0.57470i$, $x_3 = -0.51238 + 1.44129i$, $x_4 = -0.51238 - 1.44129i$, and $x_5 = -0.92071$.

The command `poly` convert roots to polynomial. This function is inversion to function `roots`.

Example

Find a quadratic equation for the roots $x_1 = -2$, $x_2 = -3$.

```
x = [-2; -3];
poly(x)
ans =
   1   5   6
```

The quadratic equation $x^2 + 5x + 6 = 0$ has the roots $x_1 = -2$, $x_2 = -3$.

The command `polyval(V,X)` returns the value of a polynomial `V` evaluated at `X`. `V` is a vector of length $n+1$ whose elements are the coefficients of the polynomial in descending powers: $\mathrm{V}(1) \cdot X^n + \mathrm{V}(2) \cdot X^{n-1} + ... + \mathrm{V}(n) \cdot X + \mathrm{V}(n+1)$

Example

```
V = [2, 3, -3];
Y = polyval(V, 5)

Y =
    62
```

This means that `polyval(V, 5)` returns the value of a polynomial $2x^2 + 3x - 3$ evaluated at 5, i.e. $2 \cdot 5^2 + 3 \cdot 5 - 3 = 62$.

If `X` is a matrix or vector, the polynomial is evaluated at all elements in `X`.

Function `polyval` is the first function for solving the polynomial regression. This command is shortcut from polynomial values and the functionality is very similar.

We have vector defined by coefficients and we may calculate value for this polynomial in all values in user specific range. And this is the first step in polynomial regression.

`Y = polyval(V, X)` where `X = [0.1, 0.2, 0.3, 2, 3, 4]` is vector of testing number.

Example

```
V = [2, 3, -3];
X = [0.1, 0.2, 0.3, 2, 3, 4];
Y = polyval (V , X)

Y =
   -2.6800   -2.3200   -1.9200   11.0000   24.0000   41.0000
```

Testing interval is possible inputs in vector form `X = [0: 0.1 :1]`.

```
V = [2, 3, -3];
X = 0: 0.1 :1;
Y = polyval (V , X)

Y =

   Columns 1 through 7
   -3.0000   -2.6800   -2.3200   -1.9200   -1.4800   -1.0000   -0.4800

  Columns 8 through 11
    0.0800    0.6800    1.3200    2.0000
```

This is very usable namely in case we have not the certain values.

Example

Solve the equation: $2x^3 - 12x^2 + 22x = 12$
Solution:
Equation should be modified: $2x^3 - 12x^2 + 22x^1 - 12 = 0$

```
t = [2, -12, 22, -12];
x = roots(t)
x =
    3.0000
    2.0000
    1.0000
```

The solving the equation is $x_1 = 3$, $x_2 = 2$, $x_3 = 1$.
The test solving:

```
polyval(t,x)
ans =
  7.1054e-015
  5.3291e-015
            0
```

The solving the equation is OK.
Finding the equation coefficients for these roots:

```
r1 = poly(x)
r1 =
    1.0000   -6.0000   11.0000   -6.0000

roots(r1)
ans =
    3.0000
    2.0000
    1.0000
```

The roots $x_1 = 3$, $x_2 = 2$, $x_3 = 1$ are the solution to the equation $x^3 - 6x^2 + 11x^1 - 6 = 0$

```
r2 = poly(x).*2
r2 =
    2.0000  -12.0000   22.0000  -12.0000

poly(x).*3
r3 =
    3.0000  -18.0000   33.0000  -18.0000

roots(r3)
ans =
    3.0000
    2.0000
    1.0000
```

All these equations:

$$x^3 - 6x^2 + 11x^1 - 6 = 0$$

$$2x^3 - 12x^2 + 22x^1 - 12 = 0$$

$$3x^3 - 18x^2 + 33x^1 - 18 = 0$$

and other equations with coefficients n·poly(x) have the same solution $x_1 = 3$, $x_2 = 2$, and $x_3 = 1$.

The command conv performs polynomial multiplication. If we have two polynomials defined by theirs coefficients (vectors V and Q). This function calculates theirs multiplication by using command conv(V,Q). V and Q are polynomials constructed from coefficients from both origin polynomials. conv is shortcut for convolution.

Example

```
V = [2, 3, -3];
Q = [7, 6, 8, 9];
S = conv(V,Q)
S =
    14     33     13     24      3    -27
```

This means:

$$(2x^2 + 3x - 3) \cdot (7x^3 + 6x^2 + 8x + 9) = 14x^5 + 33x^4 + 13x^3 + 24x^2 + 3x - 27$$

The command deconv performs polynomial division. It is the same function, but inverse - deconv(S,Q) will provide division of these two polynomials. deconv is shortcut for deconvolution.

[P,R] = deconv(S,Q) deconvolves vector S out of vector Q. The result is returned in vector P and the remainder in vector R such that S = conv(P,Q) + R.

Example

```
[P,R] = deconv(S,Q)
P =
    2.0000    3.0000   -3.0000
R =
  1.0e-014 *
         0         0         0   -0.6217   -0.9326   -0.3109
```

The vector R can be considered as zero, 10^{-14} is very small number.

This means:

$$\frac{14x^5+33x^4+13x^3+24x^2+3x-27}{7x^3+6x^2+8x+9}=2x^2+3x-3, \text{remainder is } 0$$

Example

```
A = [1, 2, 4, 3];
S = [14, 33, 13, 24, 3, -27];
[B,R] = deconv(S,A)
B =
    14     5   -53
R =
     0     0     0    68   200   132
```

This means:

$$\frac{14x^5+33x^4+13x^3+24x^2+3x-27}{x^3+2x^2+4x+3}=14x^2+5x-53, \text{remainder is } 68x^2+200x-132$$

Test of the correct solution:

```
conv(A,B)+R
ans =
    14    33    13    24     3   -27
```

i.e. the vector S.

6.3 Graphical representation of polynomial expressions

We have two ways in presentation our data set and regression results. First of them is very easy and using the MATLAB internal possibilities. It is enough to have only data set with X and Y data groups and MATLAB calculated regression relations between theirs automatically alone.

For example use the testing data sets X and Y from courses.

In the command window is necessary inputs data sets for X and Y in next form.

```
x=[0,1,2,3,4,5,6,7,8,9,10];
y=[0.1,1.2,2.2,4.4,4.2,5.1,6.1,7.5,8.2,9.1,9.0];
```

and use command for plotting

```
plot(x,y,'o')
```

Advice

This commands and notice is very comfortable solve like M-file. Not only in the command window, but open like the new M-file (File - New - M-file or File - New - Script according to your version of MATLAB). The file must be saved in the current directory with the file extension .m.

After running this script we obtain graphic window with our data. Axis and background grids are in default style and in this time it is uninteresting. Elements are in the default blue color and default circle shape with default range for X and Y axe.

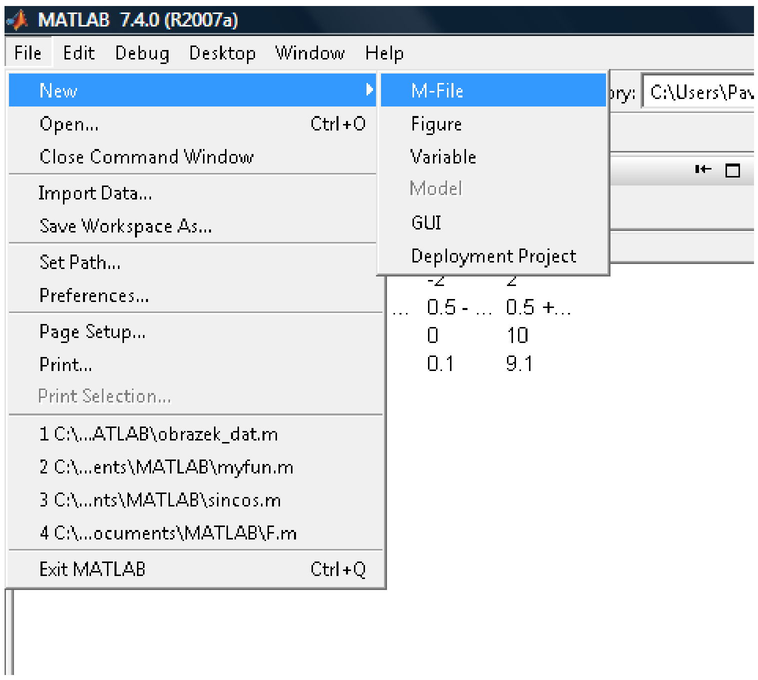
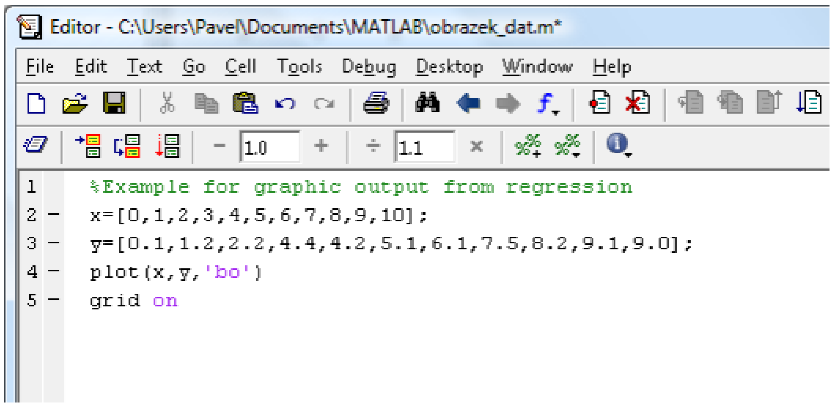


Figure 6.1: Open new M-file for the new program for regression polynomials


Editor - C:\Users\Pavel\Documents\MATLAB\obrazek_dat.m*
File Edit Text Go Cell Tools Debug Desktop Window Help

```
%Example for graphic output from regression
x=[0,1,2,3,4,5,6,7,8,9,10];
y=[0.1,1.2,2.2,4.4,4.2,5.1,6.1,7.5,8.2,9.1,9.0];
plot(x,y,'bo')
grid on
```

Figure 6.2: Example of the program for graphic output for regression

And in this moments is the rigt time for apply MATLAB default solving the polynomial and spline technology from default MATLAB Libraries. And from the top of windows menu choose the right path Tools - Basic Fitting and opens the next window with dialog parts.

In the first section named Plot fits we choose the right regression type and check the right radio button. In our graph appear the new curve and in the second part of this window appears the right shape of te equation and the right parameters like norm of residuals.

For example we choose the cubic interpolation and obtain the next graph.

The interpolation equations and coefficients are in the second small window.

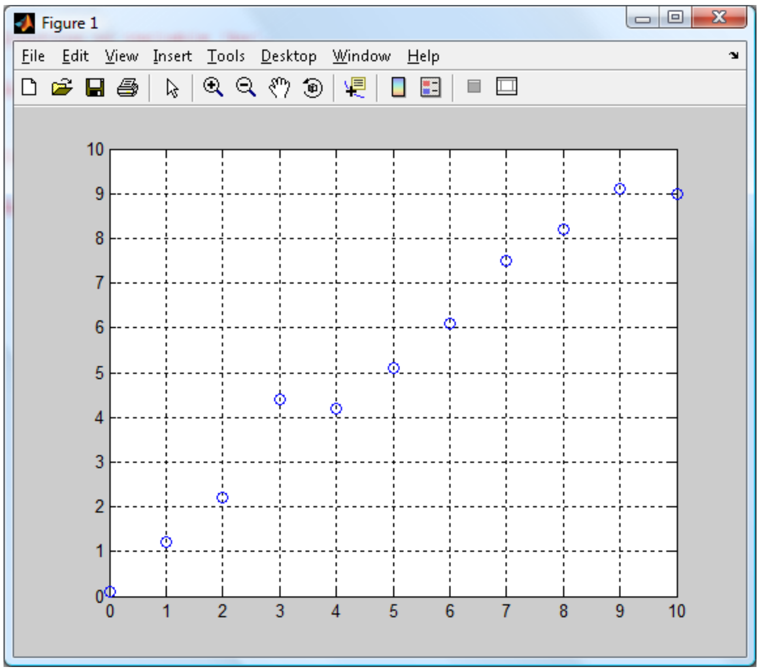
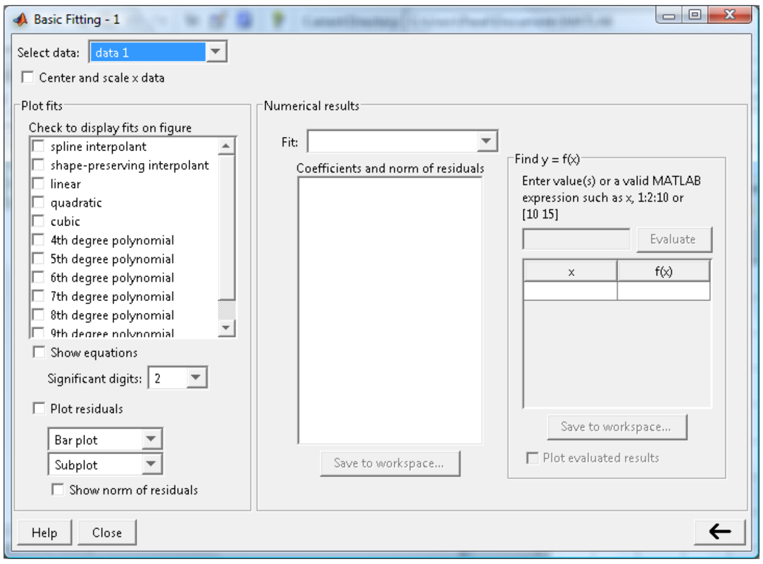


Figure 6.3: The graphic output window with new data sets



Select data: data 1

Center and scale x data

Significant digits: 2

Bar plot

Subplot

Figure 6.4: Basic fitting menu from default MATLAB libraries

6.4 Fitting functions and regression

6.4.1 Polynomial regression

The values in the vectors t a y will be interpolated with third polynomial degree.

```
%Polynomial functions
%Part of visualisation of input data
%---------------------------------
t=[0,.3,.8,1.1,1.6,2.3]';
```

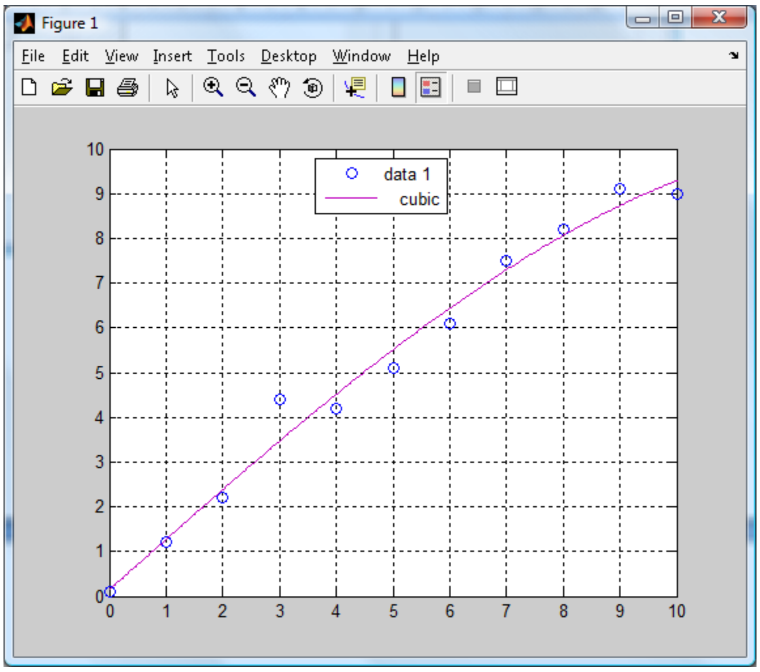
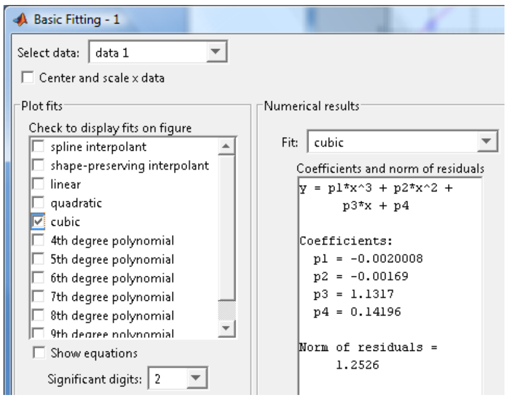


Figure 6.5: Data with the polynomials regression curve 3 degree



Select data: data 1
Center and scale x data

Plot fits
Check to display fits on figure
- spline interpolant
- shape-preserving interpolant
- linear
- quadratic
- ✓ cubic
- 4th degree polynomial
- 5th degree polynomial
- 6th degree polynomial
- 7th degree polynomial
- 8th degree polynomial
- 9th degree polynomial

Show equations
Significant digits: 2

Numerical results
Fit: cubic
Coefficients and norm of residuals

```
y = p1*x^3 + p2*x^2 +
      p3*x + p4

Coefficients:
  p1 = -0.0020008
  p2 = -0.00169
  p3 = 1.1317
  p4 = 0.14196

Norm of residuals =
     1.2526
```

Figure 6.6: Filled basic fitting table for presented data regression

```
y=[0.5,0.82,1.14,1.25,1.35,1.40]';
plot(t,y,'o')
grid on
hold on
%Part computerisation of regression data
%---------------------------------------
X=[ones(size(t)),t,t.^2,t.^3]
a=X\y
%Polynomial regression
%---------------------------------------
T=(0:0.1:2.5)';
Y=[ones(size(T)),T,T.^2,T.^3]*a;
plot(T,Y,'r-')
```

The polynomial regression is shown in Fig. 6.7.

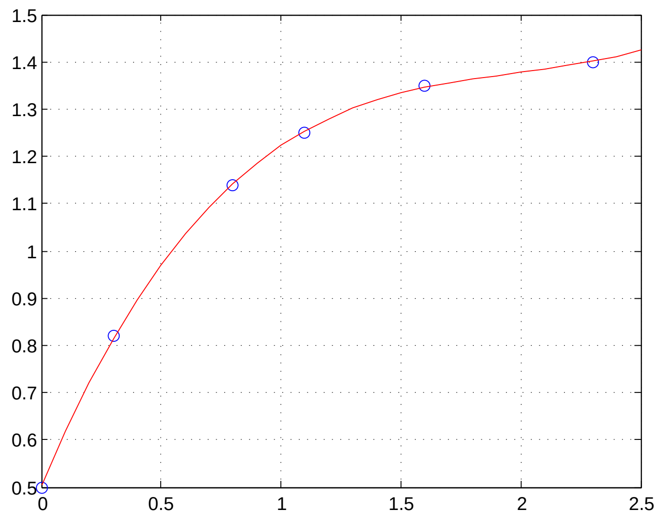


Figure 6.7: Polynomial regression

6.4.2 Polynomial curve fitting

Using the other functions like `polyfit` and `polyval` shows this M-File with the complete interpolated methodic.

Example

```
%M-File show polynomial determined of values x and y

x=[0,1,2,3,4,5,6,7,8,9,10];
y=[0.1,1.2,2.2,4.4,4.2,5.1,6.1,7.5,8.2,9.1,9.0];
plot(x,y,'bo')
grid on

%like the polynomial degree set the 3, the same as in previous case.

a=polyfit(x,y,3);
disp(a)

%polyfit calculate coefficients of the new polynomials 3 degree

k=length(x);
xx=x(1):0.1:x(k);
yy=polyval(a,xx);

%calculate values of new polynomials

hold on
plot(xx,yy,'r-')
hold off
```

MATLAB answers

```
-0.0020    -0.0017     1.1317     0.1420
```

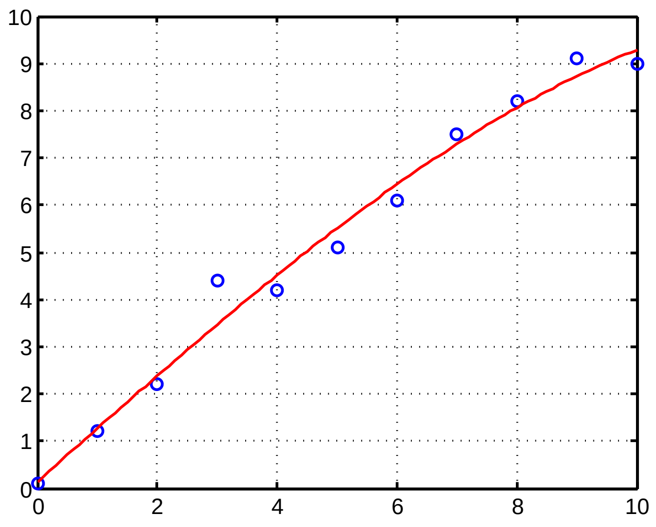


Figure 6.8: Polynomial regression

This means:
The points with coordinates x and y are interpolated with polynomial:

$$-0.002x^3 - 0.0017x^2 + 1.1317x + 0.1420$$

Fig. 6.8 depicts graphical output from MATALB.

We can calculate the y value for the point $x = 2.5$, which was not between the input values.

```
polyval(a, 2.5)
ans =
    2.9293
```

Notice

For fast easy analysis of high amount of data is useful special MATLAB feature named Basic fitting. This function is useful when we have the file full of numeric data and we want provide the basic statistical analytic on this data.

In this moment, when we have the basic graphical output only in the dot style, is necessary choose from the menu item of this window - Tools - Basic Fitting and in the last part choose the right style of numerical interpolation. Graph will be modified automatically.

6.4.3 Spline technology

`yy = spline(x,y,xx)` uses a cubic spline interpolation to find `yy`, the values of the underlying function y at the values of the interpolant `xx`.

Example

Vectors x and y have the same values as the previous example:

```
x=[0,1,2,3,4,5,6,7,8,9,10];
y=[0.1,1.2,2.2,4.4,4.2,5.1,6.1,7.5,8.2,9.1,9.0];
```

```
xx=x(1):0.1:x(end);

yy = spline(x,y,xx);

plot(x,y,'o',xx, yy, 'g')
xlabel('x')
ylabel('y')
```

Fig. 6.9 depicts cubic spline interpolation for points with with coordinates x and y.

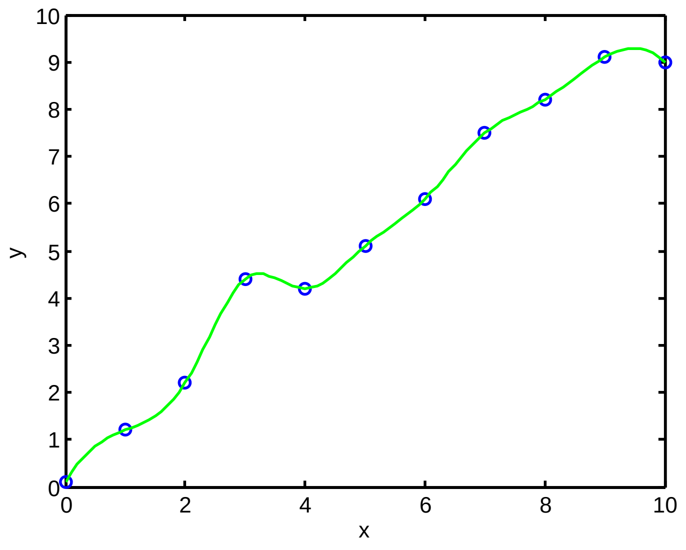


Figure 6.9: Cubic spline interpolation

We obtain the same result using function `interp1`. The function `interp1` performs one-dimensional interpolation. This function uses polynomial techniques, fitting the supplied data with polynomial functions between data points and evaluating the appropriate function at the desired interpolation points. The `spline` method is selected as the interpolation method:

```
yy = interp1(x,y,xx,'spline');
plot(x,y,'o',xx, yy , 'g')
```

Its most general form of `interp1` is `yi = interp1(x,y,xi,method)`, where `y` is a vector containing the values of a function, and `x` is a vector of the same length containing the points for which the values in `y` are given. `xi` is a vector containing the points at which to interpolate. `method` is an optional string specifying an interpolation method: `'nearest', 'linear', 'spline' or 'cubic'`.

Interpolation for two-dimensional data is also possible. The function `interp2` performs two-dimensional interpolation, an important operation for image processing and data visualization. Its most general form is `ZI = interp2(X,Y,Z,XI,YI,method)`, where `Z` is a rectangular array containing the values of a two-dimensional function, and `X` and `Y` are arrays of the same size containing the points for which the values in `Z` are given. `XI` and `YI` are matrices containing the points at which to interpolate the data. `method` is an optional string specifying an interpolation method `'nearest', 'linear' or 'cubic'`.

6.5 Examples of the polynomial regression applied on the electrical engineering problems

6.5.1 Volt-ampere characteristic of non-linear component

Draw a Volt-ampere characteristic of non-linear component. Assume that values of current and voltage on a linear component were measured (see table):

i[mA]	0	10	20	30	40	50	60	70	80
u[V]	0	1	3	6	9	12	14	15	15

Solution
`p = polyfit(x,y,n)` finds the coefficients of a polynomial p(x) of degree n that fits the data, p(x(i)) to y(i), in a least squares sense. The result p is a row vector of length n+1 containing the polynomial coefficients in descending powers. In this example, `x` is vector `I` and `y` is vector `U`.

The function `polyval(p,xx)` returns the value of a polynomial of degree n evaluated at xx. The input argument p is a vector of length n+1 whose elements are the coefficients in descending powers of the polynomial to be evaluated.

The command `plot` can plot more than one function simultaneously. In the plot command, one can specify the symbol as well as the color of the line, for example: `'o'` stands for circle and `'b'` for blue, `'g'` for green, `'k'` for black; etc.

```
U=[0,1,3,6,9,12,14,15,15];
I=[0:10:80];
xx=[0:0.05:80];
y1=polyval(polyfit(I,U,1),xx);
y2=polyval(polyfit(I,U,2),xx);
y5=polyval(polyfit(I,U,5),xx);
plot(I,U,'ok',xx,y1,'r',xx,y2,'g',xx,y5,'b')
title('Volt-ampere characteristic of non-linear component', 'Fontsize',18)
xlabel('i[mA]')
ylabel('u[V]')
legend('data points','polynomial of first order','polynomial of second order',' polynomial of fifth orde
```

The result obtained from MATLAB is in Fig

6.5.2 Curve fitting in 3D

We have measured data of humidity in a room. Dimensions of the room and humidity values are given in table. We want to create a 3D surface graph, but in some points in a room are not measured. We need some interpolation in 3D.

Measured data:

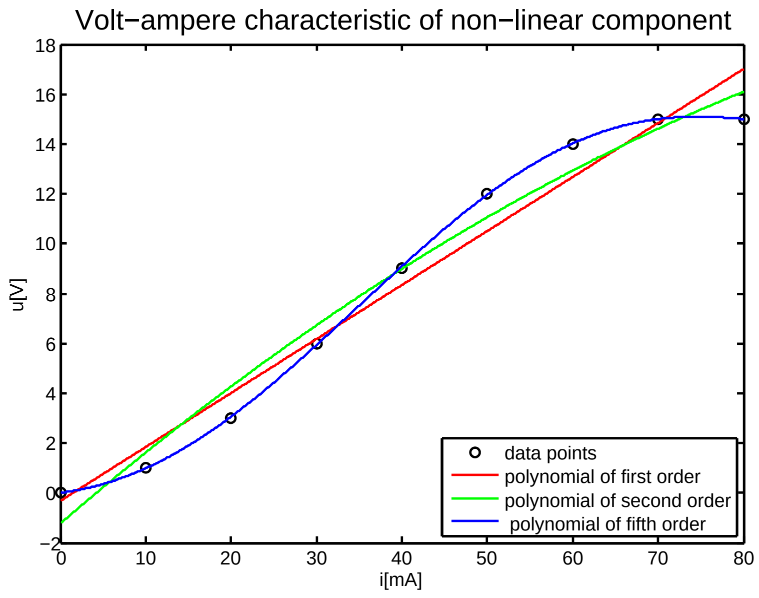


Figure 6.10: Nonlinear voltampere characteristic

6	5.5	5	4.5	4	3.5	3	2.5	2	1.5	1	0.5	
/	41.7	42.1	43.1	42.9	41.8	42.8	43.1	44.1	44.6	44.2	/	6
/	42.2	43.9	/	/	/	42.9	42.9	42.6	43.1	43.1	43.1	5.5
/	45.5	41.6	41.8	/	/	43.2	43.6	44.3	44.3	44.3	45.2	5
43.3	43.1	/	/	/	/	43.3	43.5	44	42.9	43.3	43.9	4.5
42.2	43.5	/	/	/	/	/	/	44.2	43.8	43.6	42.9	4
41.4	/	/	/	41.4	/	/	/	43.5	42.9	43.1	43.2	3.5
42	41.6	42.3	42.2	42.8	43.1	44	43.6	43.8	42.8	43	42.6	3
41.8	41.9	41.9	42.5	42.2	43.3	43.1	43.6	43.7	43.9	43.1	42.5	2.5
41.8	42.5	41.9	43.2	42.6	43.5	43.2	44.5	43.9	43.7	43	42.5	2
41.9	42.8	42	42.6	44.2	43.4	43.3	43.4	43.7	42.3	43.9	42.5	1.5
41.9	43.9	43.6	41.6	/	/	44.3	43.5	43.3	41.8	43	42.7	1
42.2	42.3	42.1	41.4	/	/	44.6	43.6	42.8	42.4	43.3	43.2	0.5

Solution

Key functions:

`interp2` - formula `ZI = interp2(X,Y,Z,XI,YI)` returns matrix `ZI` containing elements corresponding to the elements of `XI` and `YI` and determined by interpolation within the two-dimensional function specified by matrices `X`, `Y`, and `Z`. `X` and `Y` must be monotonic, and have the same format ("plaid") as if they were produced by meshgrid. Matrices `X` and `Y` specify the points at which the data `Z` is given. Out of range values are returned as `NaN`s.

`griddata` - formula `ZI = griddata(x,y,z,XI,YI)` fits a surface of the form $z = f(x,y)$ to the data in the (usually) nonuniformly spaced vectors (x,y,z). `griddata` interpolates this surface at the points specified by (`XI`,`YI`) to produce `ZI`. The surface always passes through the data points. `XI` and `YI` usually form a uniform grid (as produced by meshgrid).

```
humidity = [NaN 41.7 42.1 43.1 42.9 41.8 42.8 43.1 44.1 44.6 44.2 NaN
NaN 42.2 43.9 NaN NaN NaN 42.9 42.9 42.6 43.1 43.1 43.1
NaN 45.5 41.6 41.8 NaN NaN 43.2 43.6 44.3 44.3 44.3 45.2
43.3 43.1 NaN NaN NaN NaN 43.3 43.5 44 42.9 43.3 43.9
42.2 43.5 NaN NaN NaN NaN NaN NaN 44.2 43.8 43.6 42.9
41.4 NaN NaN NaN 41.4 NaN NaN NaN 43.5 42.9 43.1 43.2
42 41.6 42.3 42.2 42.8 43.1 44 43.6 43.8 42.8 43 42.6
41.8 41.9 41.9 42.5 42.2 43.3 43.1 43.6 43.7 43.9 43.1 42.5
41.8 42.5 41.9 43.2 42.6 43.5 43.2 44.5 43.9 43.7 43 42.5
41.9 42.8 42 42.6 44.2 43.4 43.3 43.4 43.7 42.3 43.9 42.5
41.9 43.9 43.6 41.6 NaN NaN 44.3 43.5 43.3 41.8 43 42.7
42.2 42.3 42.1 41.4 NaN NaN 44.6 43.6 42.8 42.4 43.3 43.2]

temp_x = [6 5.5 5 4.5 4 3.5 3 2.5 2 1.5 1 0.5]

temp_y = [6
5.5
5
4.5
4
3.5
3
2.5
2
1.5
1
0.5]

% we prepare axes for 3D graph
[X,Y] = meshgrid(temp_x,temp_y);

% interpolation
% we select points without value NaN and their x and y coordinates into
% variables xr, yr, zr
how_many_values = 0;
for cycle_x=1:length(temp_x)
    for cycle_y=1:length(temp_y)
        if ~(isnan(humidity(cycle_y,cycle_x)))
            how_many_values = how_many_values + 1;
            xr(how_many_values) = temp_x(cycle_x);
            yr(how_many_values) = temp_y(cycle_y);
            zr(how_many_values) = humidity(cycle_y,cycle_x);
        end;
    end;
end;

xi = temp_x(1):-0.05:temp_x(end);
yi = temp_y(1):-0.05:temp_y(end);
[XI,YI] = meshgrid(xi,yi);
ZI = griddata(xr,yr,zr,XI,YI,'cubic');
surf(XI,YI,ZI);

hold on  % to hold previous graph

plot3(xr,yr,zr,'.r');
hold off
shading interp % coloured interpolation
colorbar  % colour scheme legend
```

```
% graph including contours
figure
surfc(XI,YI,ZI);
shading interp % coloured interpolation
colorbar  % colour scheme legend

% measured data graph
figure
surf(X,Y,humidity);
shading interp % coloured interpolation
colorbar  % colour scheme legend

% three graphs in column (3x1 graph), we used the first one
subplot(3,1,1)
surf(XI,YI,ZI)
shading interp % coloured interpolation
colorbar  % colour scheme legend

% three graphs in column (3x1 graph), we used the second one
subplot(3,1,2)
surf(XI,YI,ZI)
shading interp % coloured interpolation
colorbar  % colour scheme legend

% three graphs in column (3x1 graph), we used the third one
subplot(3,1,3)
surf(XI,YI,ZI)
shading interp % coloured interpolation
colorbar  % colour scheme legend

figure % new figure creating
surf(XI,YI,ZI)
shading interp % coloured interpolation
colorbar  % colour scheme legend

% changing of colormap
% using colormapeditor - for exmple to setup threshold values
colormap(copper)

figure % new figure creating
pcolor(XI,YI,ZI)
shading interp % coloured interpolation
colorbar  % colour scheme legend

figure % new figure creating
contour(XI,YI,ZI)

figure % new figure creating
mesh(XI,YI,ZI)
hold on
contour3(XI,YI,ZI,'k') % 3D contours in black
hold off
```

Fig. 6.11 shows interpolation for the specified two-dimensional data in 3D.

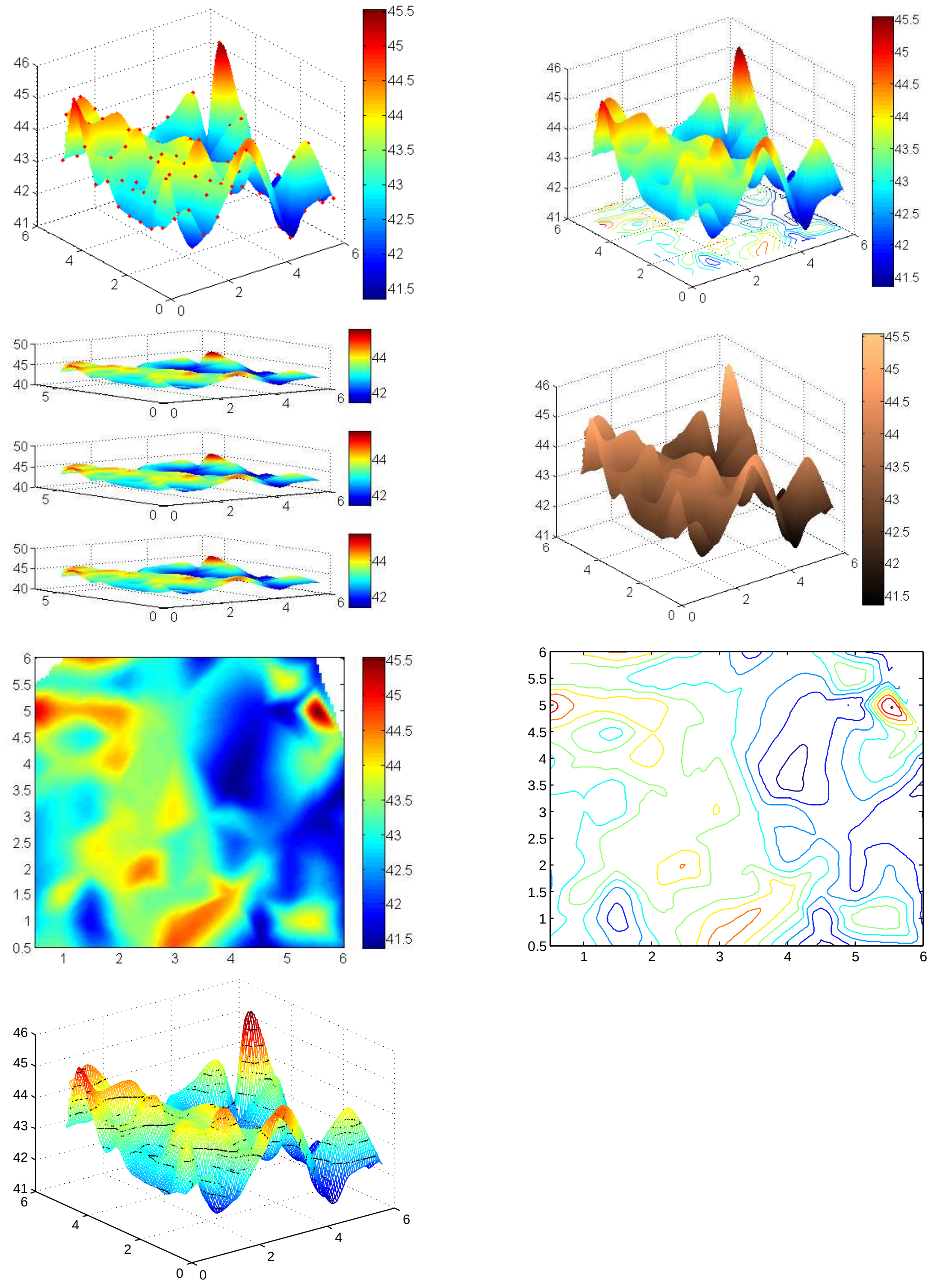


Figure 6.11: Interpolation for two-dimensional data in 3D

6.6 Test of commands for solving polynomials and polynomial regressions

1. We have polynomial $5x^3+7x^2+4$. How can you determine the value of $5\cdot2^3+7\cdot2^2+4$ and the value of $5\cdot(-2)^3+7\cdot(-2)^2+4$?

2. Solve the equation $4x^2+6x+3=0$.

3. Solve the equation $2x^7+5x^4+3x^2+x-1=0$

4. During the measurement, these values have been found:

t[s]	1	2	3	4	5	6	7
i[A]	3	5	5	6	8	7	8

   Determine the coefficients of the second degree polynomial and fit points using this polynomial of second degree.

7
Flow control statements

7.1 Flow Control Statements

The statements in m-files are generally executed from top to bottom, in the order that they appear. Flow control statements are statements which execution results in a choice being made as to which of two or more paths should be followed. These kind of commands break up the flow of execution by employing decision making, looping, and branching, enabling your program to conditionally execute particular blocks of code. This section describes the decision-making statements (if–end, if–else–end, switch–case–end), the loop statements (for–end, while–end) and auxiliary statements (break and continue).

Flow control statements in MATLAB has a syntax like in "standard" programming languages.

Loop *for*

Example

This example creates a vector size n.

```
x = [];
for k = 1:n
  x=[x, k.^2];
end
```

Example

This example creates the same vector, but in reverse order (loop step is set to -1)

```
x = [];
for k = n:-1:1
  x=[x, k.^2];
end
```

Example

This example creates a Hilbert's matrix size *a* by *b*.

```
for k = 1:a
  for m = 1:b
    H(k, m) = 1/(k+m-1);
  end
end
H
```

Example

Let us matrix:

```
A = [1:5; 10:-2:2; 0:3:12]

A =
     1     2     3     4     5
    10     8     6     4     2
     0     3     6     9    12
```

Find the square of the matrix elements and measure the time.
Solution
The commands `tic`, `toc` measures the time it takes the MATLAB software to execute the one or more lines of MATLAB code. The `tic` command starts a stopwatch timer, MATLAB executes the block of statements, `toc` stops the timer and displaying the elapsed time in seconds.

```
tic
B = A.^2;
toc
```

MATLAB answers

```
Elapsed time is 0.003426 seconds.
```

For the other method of solution we need to know the matrix dimensions:

```
[rows,columns] = size(A);

tic
for m = 1:rows
    for n = 1:columns
        B(m,n) = A(m,n)^2;
    end
end
toc
```

MATLAB answers

```
Elapsed time is 0.011099 seconds.
```

The example shows that the use of array operation is quicker than using `for` loops.
The resulting matrix B in both cases is as follows

```
B =
     1     4     9    16    25
   100    64    36    16     4
     0     9    36    81   144
```

Advice

In short, replacing loops by array operations, everywhere, where it is possible, rapidly increase execution speed of MATLAB program.

Loop *while*

Example

In this example for a given number computes and shows smallest non-negative number `n`, where `2^n>= a`.

```
n = 0;
while (2^n < a)
n = n + 1;
end
n
```

Loop's Auxiliary Statements

Auxiliary statements Break and continue are used to extend ability to control functionality of loop statements.

break breaks out of the smallest enclosing for or while loop,

continue continues with the next iteration of the loop, it passes control to the next iteration of the `for` or `while` loop in which it appears, skipping any remaining statements in the body of the loop.

Example

```
while condition
  some_commands_1;
  if(a < 5)
    break;
  if (b > 30)
    continue;
  some_commands_2;
end
```

Conditional statement if

Example

```
if n < 0
  parity = 0;
  elseif (rem(n,2) == 0)
    parity = 2;
  else
    parity = 1;
end
```

Example

Is matrix A equal to matrix B ?

```
if A == B
  command;
end
```

Is matrix A non-equal to matrix B ?

```
if any(any(A ~= B)) % because we must test
  command;
end
```

or in a simple case

```
if A == B
  ;
else
  command;
end
```

But the command

```
if A ~= B
  command;
end
```

doesn't work properly, because `command;` will be executed only in case of whole relation matrix contains zeros.

Conditional statement switch

Unlike if–end and if–else–end statements, the switch statement can have a number of possible execution paths.

Example

The code converts the value x in cm to the unit defined in variable `units`, using the switch statement.

```
x = 2.7;
units = 'm'
switch units   % convert x to centimeters
  case { 'inch', 'in' }
    y = x * 2.54;
  case { 'feet', 'ft' }
    y = x * 2.54 * 12;
  case { 'meter', 'm' }
    y = x * 100;
  case { 'millimeter', 'mm' }
    y = x / 10;
  case { 'centimeter', 'cm' }
    y = x;
  otherwise
    disp(['Unknown Units: ', units])
    y = NaN;
end
```

Exception

The general form of exception command try–catch is

```
try
  some_commands_1;
catch
  some_commands_2;
end
```

In this sequence, the statements between try and catch are executed. If an error occurs here, execution of try block has been stopped. Program execution continues with statements between catch and end. User can use `lasterr` variable to obtain the cause of the error.

7.2 Test of flow control statements

1. Create a function, which ask user for two numbers. Then your function prints, which number is greater (first or second) and prints the value of the greater number.
2. Create a function, which plots a graph of the function $\sin(t)$ or $\cos(t)$ or $\sin^2(t)$, on given interval $\langle -2\pi; 2\pi \rangle$. Ask user, which function to plot.
3. Create a function, which returns matrix given by user. In your function ask user for matrix sizes and then control, if the matrix sizes are greater then one. Then ask user, to give elements values, element by element, in row direction.

8 Integration

8.1 Numerical integration

Quadrature is a numerical method used to find the area under the graph of a function, that is, to compute a definite integral.

The quad function evaluates integral numerically, using adaptive Simpson quadrature. The quad function may be most efficient for low accuracies with nonsmooth integrands.

Let's solve the integral

$$q = \int_a^b f(x)dx$$

`q = quad(fun,a,b)` tries to approximate the integral of function fun from a to b to within an error of 1e-6 using recursive adaptive Simpson quadrature.

The `quadl` function evaluates integral numerically, using adaptive Lobatto quadrature. The `quadl` function may be more efficient than quad at higher accuracies with smooth integrands.

The `quadgk` function evaluates integral numerically, using adaptive Gauss-Kronrod quadrature. The `quadgk` function may be most efficient for high accuracies and oscillatory integrands. It supports infinite intervals and can handle moderate singularities at the endpoints. It also supports contour integration along piecewise linear paths.

The `quadv` function evaluates integral numerically, using vectorized quadrature.

Notice

The `quad2d` function evaluates double integral numerically over planar region.

The `dblquad` function evaluates double integral numerically over a rectangle.

The `triplequad` function evaluates triple integral numerically over the three dimensional rectangular region.

Example

For numeric evaluate integral MATLAB uses function quad, which used for calculation integration function from A to B recursive adaptive Simpson quadrature, as mentioned above. Using this function isn't so simple.

```
F = @(x) sin (x); % allocate the function of variable x
Q = quad (F, 0, pi) % and for this interval is result 2
Q =
    2.0000
```

More details are in the contextual help after typing `help quad`

Example

```
q = quad(@sin, 0, 2*pi)
q =
     0

q = quad(@cos, 0, pi/2)
q =
    1.0000
```

This is simplest syntax for the integral calculation. It is useful only for the simplest mathematical functions, for more difficult expression is not proper.

For case the complicated integration is better way to use Editor and make some function and from the Command Windows only this function call.

Example

We have this formula to the integration

$$y = \int_{0}^{2} \sin(x^2) \cdot \cos^2(x) dx$$

Compute this integral.

Solution

First step is preparing the new M-file with this function in the right shape. From the Menu of MATLAB choose File - New - M-File or File - New - Function (according to your version of MATLAB), Fig. 8.1 and write commands, Fig. 8.2.

```
function y = myfun(x)
y = sin(x.^2) .* cos(x).^2;
```

The file `myfun.m` must be saved in the current directory with file extension `.m` and the last action is calling this function `myfun` from Command Window, Fig. 8.3.

```
integral = quad(@myfun, 0, 2)
integral =
    0.2106
```

Example

Display course of the function $\cos(x)$ for x from 0 to 2π and course of the integral of this function

$$y = \int_{0}^{2\pi} \cos(x) dx$$

Compare course of this integral with function $\sin(x)$.

Solution

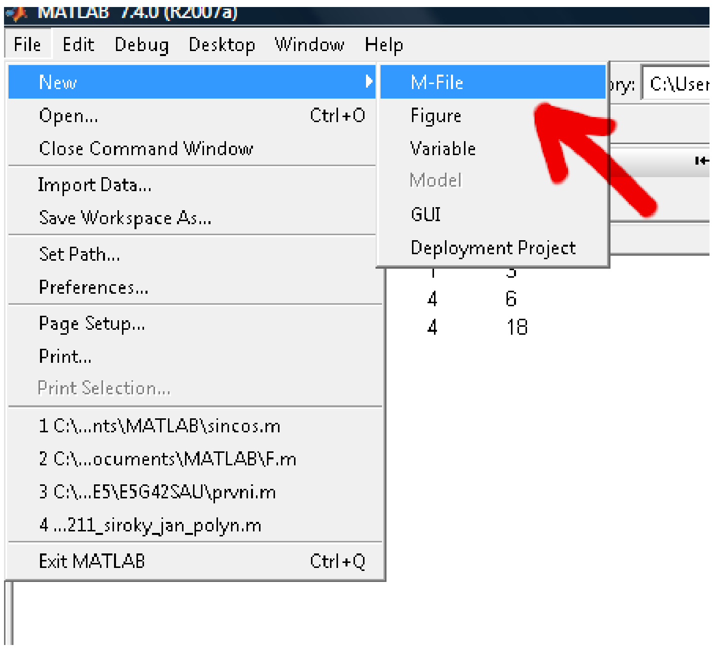
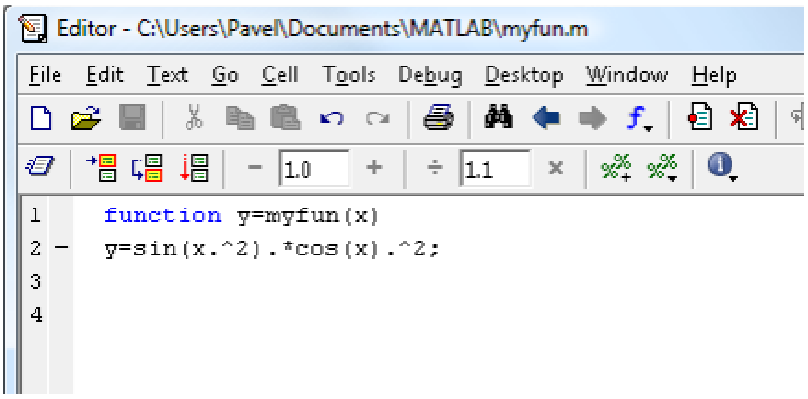


Figure 8.1: Creation the new M-file for integration function


Editor - C:\Users\Pavel\Documents\MATLAB\myfun.m
File Edit Text Go Cell Tools Debug Desktop Window Help

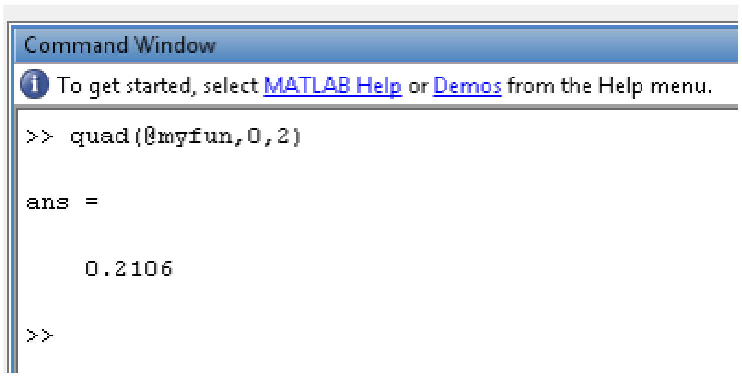
```
function y=myfun(x)
y=sin(x.^2).*cos(x).^2;
```

Figure 8.2: Creation the new M-file for integration function

```
function integration_1
% integration of function cos(x) in range from 0 to 2*pi and course of
% functions cos, integral of cos and a course of sin (to match them up)

% integration of cos
y=quad(@cos,0,2*pi);
disp('integration of function cos from 0 to 2pi')
disp(y)

x=[0 : pi/50 : 2*pi]; % vector (i.e. x axis) in range of 0 to 2*pi, step pi/50
for n=1:length(x)    % loop with fix number of iterations, loop will execute
                    % so many times how many elements vector x have
   yi(n)=quad(@cos,0,x(n));  % calculation of integral cos in range
                             % from 0 to n-th element of x vector
                             % and assignment of result to yi vector
end; % end of for loop
```


Command Window
To get started, select MATLAB Help or Demos from the Help menu.

```
>> quad(@myfun,0,2)

ans =

    0.2106

>>
```

Figure 8.3: The calling the function with function limits and result

```
subplot(2,1,1) % division of figure to 2 fields and selecting of field 1
plot(x,cos(x),'--r',x,yi,'-k')   % graph plots, selection of colors and
                                % line types
                  % cos(x) course - red, dashed line
                % integral course - black, solid line
legend('Course of cos','Course of integral') % legend
subplot(2,1,2) % division of figure to 2 fields and selecting of field 2
plot(x,sin(x)) % graph plots of sin function (to match it up)
legend('Course of function sin') % legend
```

Notice

The command disp(x) displays the variable x, without printing the variable name. For detailed information, see the section 9.1.

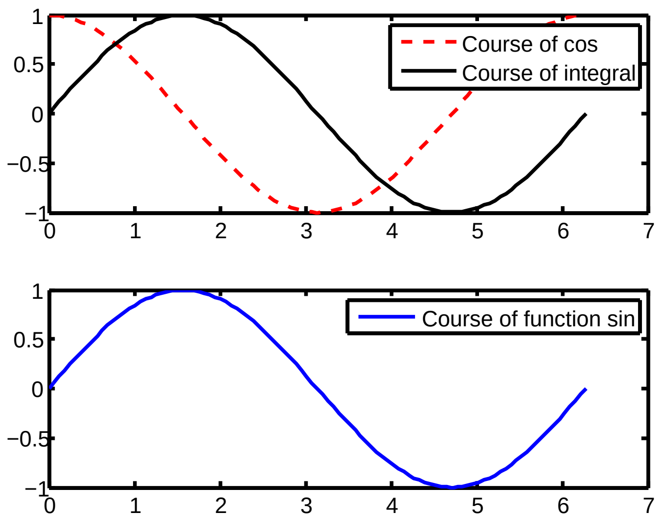


Figure 8.4: Course of the function $\cos(x)$, integral $\int_0^{2\pi} \cos(x)dx$, and function $\sin(x)$

Example

Display course of the function sin(x) for x from 0 to 2π and course of the integral of this function

$$y = \int_{0}^{2\pi} \sin(x)dx$$

Compare course of this integral with function $-\cos(x)$.
Solution

```
function integration_2
% integration of function sin(x) in range from 0 to 2*pi and course of
% functions sin(x), integral of sin(x) and a course of -cos(x) (to match them up)

% integration of sin
y=quad(@sin,0,2*pi);
disp('integration of function sin from 0 to 2pi')
disp(y)

x=[0 : pi/50 : 2*pi]; % vector (i.e. x axis) in range of 0 to 2*pi, step pi/50
for n=1:length(x) % loop with fix number of iterations, loop will execute
                  % so many times how many elements vector x have
    yi(n)=quad(@sin,0,x(n)); % calculation of integral sin in range
                               % from 0 to n-th element of x vector
                              % and assignment of result to yi vector
end;              % end of for loop
subplot(2,1,1) % division of figure to 2 fields and selecting of field 1
plot(x,sin(x),':g',x,yi,'-k') % graph plots, selection of colors and
                              % line types
                  % sin(x) course - green, dotted line
                              % integral course - black, solid line
legend('Course of sin','Course of integral') % legend
subplot(2,1,2)  % division of figure to 2 fields and selecting of field 2
plot(x,-cos(x)) % graph plots of -cos(x)(to match it up)
legend('Course of function -cos') % graph plots of -cos function (to match it up)
axis([0,7,-2,1]) % axis range redefinition
```

Example

Display course of the function $\sin(x^2) \cdot \cos^2(x)$ for x from 0 to 2 and course of the integral of this function

$$y = \int_{0}^{2} \sin(x^2) \cdot \cos^2(x)dx$$

Solution
To compute this integral, we must write the M-file function `myfun` that computes the integrand $\sin(x^2) \cdot \cos^2(x)$.

```
function y = myfun(x)
y = sin(x.^2) .* cos(x).^2;
```

This M-file is the same as M-file `myfun.m` in third example of this section 8.1, which can also be used if saved in the same folder as the following M-file `integration_3.m`.

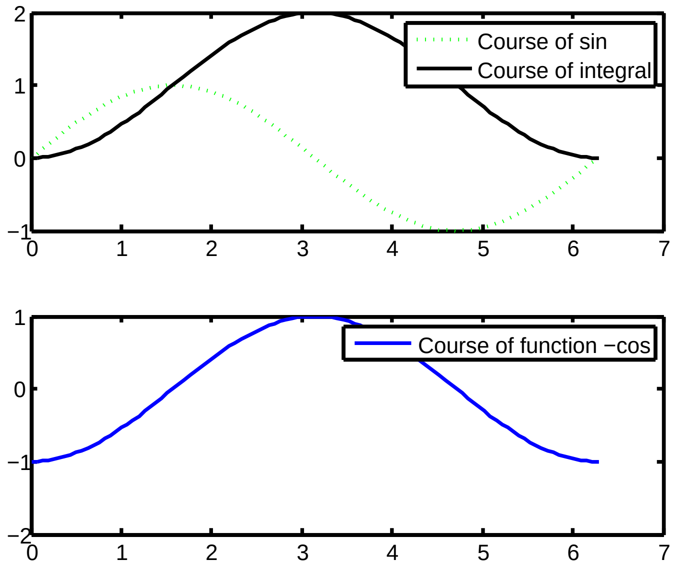


Figure 8.5: Course of the function sin(x), integral $\int_0^{2\pi} \sin(x)dx$, and $-\cos(x)$

```
function integration_3
% integration of sin(x.^2) .* cos(x).^2 in range from 0 to 2
y=quad(@myfun,0,2);
disp(y)

% course of integral sin(x.^2) .* cos(x).^2
x=[0 : 0.05 : 2]; % vector (i.e. x axis) in range of 0 to 2, step 0.05
for n=1:length(x) % loop with fix number of iterations, loop will execute
                  % so many times how many elements vector x have
    yi(n)=quad(@myfun,0,x(n)); % calculation of integral sin in range
                               % from 0 to n-th element of x vector
                              % and assignment of result to yi vector
end;              % end of for loop
plot(x,myfun(x),':m',x,yi,'-c') % graph plots, selection of colors and
                            % line types
                  % sin(x) course - green, dotted line
                            % integral course - black, solid line
legend('Course of sin(x^2)cos^2(x)','Course of integral') % legend
```

8.2 Analytical integration using Symbolic Math Toolbox

Symbolic Math Toolbox software provides a complete set of tools for symbolic computing that augments the numeric capabilities of MATLAB.

Before testing the next example, is necessary check your Matlab installation fro presence of the Symbolic Math Toolbox. This checking is possible to do with commands `ver`.

Previous example with integration represented the numerical style of problem solving. We must specify not only the main function but the low and high integration limits.

In special cases we need the analytical result of some problem. There is necessary like the first step input the symbolic variable - for example x. Definition of the symbolic variables

```
x = sym ('x')      % definition of symbolic variables x
```

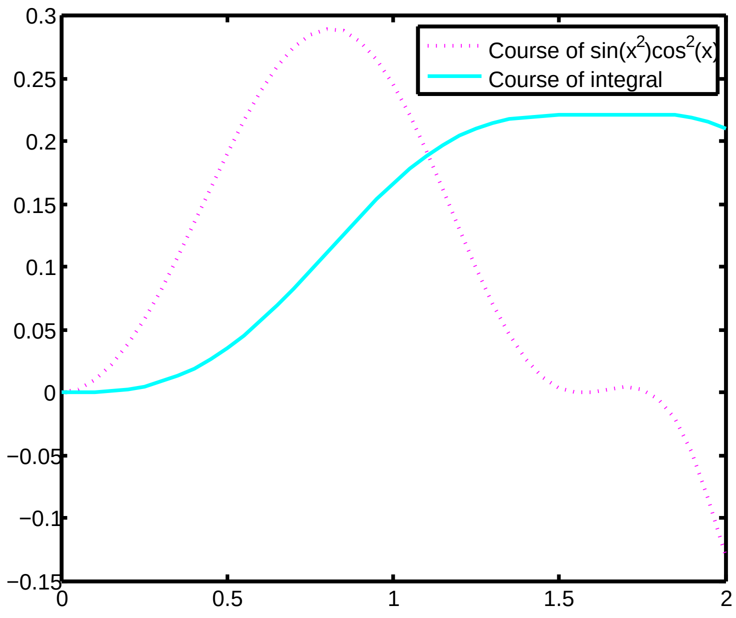


Figure 8.6: Course of the function $\sin(x^2)\cdot\cos^2(x)$ and integral $\int_0^2 \sin(x^2)\cdot\cos^2(x)dx$

```
x =
x
```

`int` - analytical integration
Examples:

```
int(x)

ans =
x^2/2

int(cos(x))

ans =
sin(x)

int(-1/x^2)

ans =
1/x
```

Notice

`diff` - analytical derivation

```
Y = diff (1/x)   % this is request for analytical solving this expression
```

MATLAB answers:

```
Y = -1/x^2
```

9 Advanced instructions for text input and output

9.1 Formatted printing

Simple text output

```
disp(3*5)
    15

variable = 5;
disp(variable)
     5

text = 'Hello';
disp(text)
    Hello

disp('text string')
   text string

disp(['text', ' ', 'string'])
   text string
```

- there is **no way to format output**
The text strings are enclosed by apostrophes (single quote marks).

Notice

`num2str` converts numbers to a string.

```
disp(['The number is ', num2str(5), '. The number increased by 1 is ', num2str(5+1),'.'])
   The number is 5. The number increased by 1 is 6.
```

Formatted text output

`fprintf(format, arguments)` formats data and displays the results on the screen.

Examples

- print values in form: 8 chars, add spaces on left handed side

```
fprintf('%8d %8d\n', a, b);
      56       34
```

- print values in form: 8 chars, add zeros on left handed side at first time and add spaces on left handed side at second time

```
fprintf('%08d %8d\n', a, b);
00000056       34
```

- typed value 8 means "at least 8 chars". **It mustn't crop the longer integer.**
- in case of decimal numbers - we want at whole 15 chars including decimal point and including 2 decimal places

```
fprintf('%15.2f\n', x);
         123.35
```

- if we want to print a sign every time (+ and -):

```
fprintf('%+d %+d\n', a, b);
+56 -34
```

Important special characters:

`\n` - new line
`\t` - tabulator
`\r` - return to the same line beginning (it's operating system depend)
`\b` - in some operating systems - beep (loudspeaker).
`\\` - print char

`%%` - print char %

`%d` - print integer decimal number (sign)
`%i` - print integer decimal number (sign)
`%o` - print octal scale number
`%u` - print decimal (decadic) number (unsigned)
`%x` or `%X` - print hexadecimal number
`%f` - print decimal number - fixed decimal point
`%e` or `%E` - print decimal number - exponential form
`%g` or `%G` - like `%f` and `%e` or `%E` - exponential form is used, if it is needed (if the number is to big). High order zeros are omitted.
`%c` - print character (stored in variable)
`%s` - print text string (stored in variable)

Examples

The answer of MATLAB is always right behind the command `fprintf`.

```
a = 234;

fprintf('Decadic scale: %d\nOctal scale: %o\n', a, a);
Decadic scale: 234
Octal scale: 352

fprintf('Hexadecimal scale: %x\n', a);
Hexadecimal scale: ea

fprintf('Hexadecimal scale: %X\n', a);
Hexadecimal scale: EA

x = 123.3456;
```

```
fprintf('Fixed decimal point: %f\n', x);
Fixed decimal point: 123.345600

fprintf('exponential: %e\n', x);
exponential: 1.233456e+002

fprintf('EXPONENTIAL: %E\n', x);
EXPONENTIAL: 1.233456E+002

fprintf('"Smart" style g: %g\n', x);
"Smart" style g: 123.346

fprintf('"Smart" style g: %g\n', x * 100000000);
"Smart" style g: 1.23346e+010

fprintf('"SMART" STYLE G: %G\n', x);
"SMART" STYLE G: 123.346

x = 1.2344e+034;

fprintf('"SMART" STYLE G: %G\n', x * 100000000);
"SMART" STYLE G: 1.2344E+042
```

Error

`error` displays message and abort function.

Example

We write in M-file editor:

```
function errorTest(x, y, z)
if nargin ~= 3
    error('Wrong number of input arguments')
end
```

We save M-file as `errorTest.m` to the current folder.

Call the function with an incorrect number of inputs. The call to `nargin`, the function that checks the number of inputs, fails and the program calls error:

```
errorTest(1, 2)
??? Error using ==> errorTest at 3
Wrong number of input arguments
```

Call the function with an correct number of inputs.

```
errorTest(1, 2, 3)
Entered correctly
 x = 1.00
 y = 2.00
 z = 3.00
```

9.2 Request for user input

Input

`r = input(prompt)` displays the `prompt` string on the screen, waits for input from the keyboard, evaluates any expressions in the input, and returns the value in `r`.

s = input(prompt, 's') returns the entered text as a MATLAB string.

Example

```
input('Enter a number: ')
```

The user enters a number 3 from the keyboard and MATLAB answer:

```
ans =
     3
```

Example

```
r = input('Enter a number: ');
```

The user enters a number 5 from the keyboard.

```
fprintf('Entered: %f\n', r);
Entered: 5.000000
```

Example

```
in_char = input('Enter a character: ', 's');
```

The user enters the character `n` from the keyboard.

```
fprintf('Entered: %c\n', in_char);
Entered: n
```

Example

```
in_text = input('Your text: ', 's')
```

The user enters the string `How are you?` from the keyboard.

```
fprintf('Was entered: %s\n', in_text);
Was entered: How are you?
```

Menu of choices for user input

The command `ch = menu('mtitle','opt1','opt2',...,'optn')` allows the user to choose from the menu, whose title is in the string variable `'mtitle'` and whose choices are string variables `'opt1'`, `'opt2'`, and so on. The menu opens in a modal dialog box. menu returns the number of the selected menu item, or 0 if the user clicks the close button on the window.

Notice

The command `menu` can always be replaced by decision-making statements(`switch`,`if`) and instructions for text input and output.

Example

Plotting of functions sin(x), cos(x) or tan(x) from $-\pi/3$ to $\pi/3$ by user choice.

```
ch = menu('Choose a function','sin','cos','tan');

x=[-pi/3 : 0.05 : pi/3];
switch(ch)
    case 1
        y = sin(x);
    case 2
        y = cos(x);
    case 3
        y = tan(x);
    otherwise
        fprintf('This option is not possible\n');
end
plot(x,y)
```

The generated menu of choices for user input is shown in Fig. 9.1. The user chooses the function selected by clicking, then the graph is plotted. The menu function is one of Predefined dialogue boxes (section 9.5).

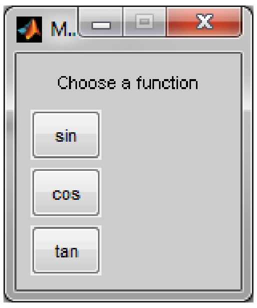


Figure 9.1: The generated menu of choices for user input

9.3 Formatted output into file with parameters

`fprintf(fileID, format, A)` applies the format to all elements of array `A` and any additional array arguments in column order, and writes the data to a text file.

```
fd = fopen('C:\\file_path\\file.txt', 'mode');
```

where `fd` - variable, so-called file descriptor and `mode` should be:
`'r'` - read
`'w'` - write, overwrite (re-write), create new file (if it is possible)
`'a'` - append at the end of the existing file
`'r+'` - read and write
`'w+'` - read and write, overwrite (re-write), create new file (if it is possible)
`'a+'` - read and write, overwrite (re-write), create new file (if it is possible) and append at the end of the existing file.

Writing to text file

```
my_file = fopen('C:\\file_path\\file.txt', 'w');
fprintf(my_file, the same parameters as on screen, arguments)
```

At the end we must close our file:

```
fclose(my_file);
```

Example

```
a = 234;
my_file = fopen('datei.txt', 'w');
fprintf(my_file,'Decadic scale: %d\n', a);
fprintf(my_file,'Octal scale: %o\n', a);
fprintf(my_file,'Hexadecimal scale: %X\n', a);
fclose(my_file);
```

In the text file `datei.txt` in the current folder, there is a written:

```
Decadic scale: 234
Octal scale: 352
Hexadecimal scale: EA
```

9.4 Formatted output into text strings

`sprintf` formats data into string.

`str = sprintf(format, A)` applies the format to all elements of array `A` and any additional array arguments in column order, and returns the results to string `str`.

Write to string variable:

```
res_string = sprintf(the same parameters as on screen, arguments);
```

Example

```
in_text = input('Your text: ', 's')
```

The user enters after being prompted `Your text:` the string `How are you?` from the keyboard.

```
mess = sprintf('You entered: %s\n', in_text);
figure     % empty figure
title(mess)   % figure's title
```

The title of empty graph in Figure window is depicted in Fig. 9.2

9.5 Predefined Dialog Boxes

Predefined dialog boxes are dialog boxes for error, user input, waiting, etc.
`dialog` creates and displays dialog box
`errordlg` creates and displays error dialog box
`helpdlg` creates and displays help dialog box
`inputdlg` creates and displays input dialog box
`listdlg` creates and displays list selection dialog box
`msgbox` creates and displays message dialog box

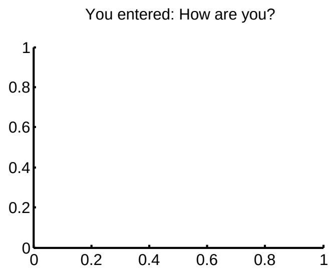


Figure 9.2: Title of empty graph

`pagesetupdlg` displays page setup dialog box
`printdlg` displays print dialog box
`questdlg`displays question dialog box
`uigetdir` displays standard dialog box for retrieving a directory
`uigetfile` displays standard dialog box for retrieving files
`uiputfile` displays standard dialog box for saving files
`uisave` displays standard dialog box for saving workspace variables
`uisetcolor` displays standard dialog box for setting an object's
`ColorSpecuisetfont` displays standard dialog box for setting an object's font characteristics
`waitbar` displays waitbar `warndlg` display warning dialog box
For more information, see the `help`.

Example

Create and open message dialog box with string `mess` from previous example.

```
msgbox(mess,'Information','help');
```

Fig. 9.3 shows the message dialog box with string `mess` "How are you?", with title "Information", and with icon.

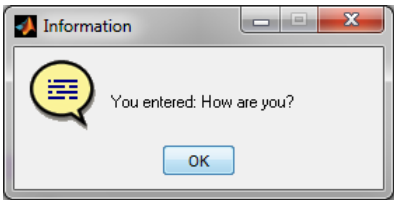


Figure 9.3: The message dialog box

Example

This program ask user, using input dialog window, for the number of apples and the number of pears. Then evaluate a sum of this fruits and the result show to user in message box window.

```
function input_dia
  message1 = sprintf('Number of apples');
  message2 = sprintf('Number of pears');
  message = {message1, message2};
  dlg_title = 'Fruits';
  num_of_lines = 1;
  preselection = {'5', '10'};
  % input dialog
  usr_inp = char(inputdlg(message, dlg_title, num_of_lines, preselection));

  % Cancel press handling
  if strcmp(usr_inp,'')
    return;
  end;

  num_of_apples = str2num(usr_inp(1,:));
  num_of_pears = str2num(usr_inp(2,:));
  num_of_fruits = num_of_apples + num_of_pears;

  resultMessage = sprintf('%d apples and %d pears is %d fruits', num_of_apples, num_of_pears, num_of_fru

  % message box
  msgbox(resultMessage);
end
```

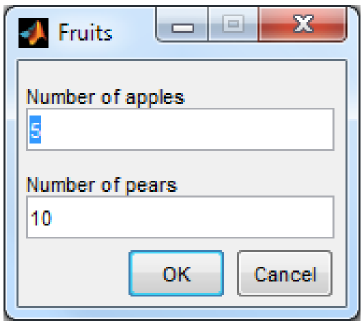
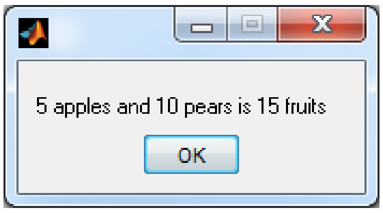


Figure 9.4: Input dialog box



5 apples and 10 pears is 15 fruits

Figure 9.5: The message dialog box

Fig. 9.4 shows the input dialog box with two input fields and Fig. 9.5 shows the message dialog box with result of evaluation.

9.6 Test of text input and output

1. Write commands to request user input. What is the difference between entering numbers and characters?

2. Write commands for displaying the entered data from previously question on the screen.
3. Give commands for writing the entered data from first question to a text file.
4. Rewrite previous task using GUI - predefined dialogue boxes.

10 Differential equation in the engineering praxis

10.1 Basic terms and commands

The solvers `ode23`, `ode45`, `ode113`, `ode15s`, `ode23s`, `ode23t`, `ode23tb` solves initial value problems for ordinary differential equations (shortcut ODE).

`[T,Y] = solver(odefun, [t0, tf], y0, options)` integrates the system of differential equations $y' = f(t, y)$ from time t0 to tf with initial conditions y0,

where solver is one of `ode23`, `ode45`, `ode113`, `ode15s`, `ode23s`, `ode23t`, `ode23tb` and `odefun` is a function handle that evaluates the right side of the differential equations.

`ode45` is based on an explicit Runge-Kutta (4,5) formula, the Dormand-Prince pair. In general, `ode45` is the best function to apply as a first try for most problems.

`ode23` is an implementation of an explicit Runge-Kutta (2,3) pair of Bogacki and Shampine.

`ode113` is a variable order Adams-Bashforth-Moulton PECE solver. It may be more efficient than ode45 at stringent tolerances and when the ODE file function is particularly expensive to evaluate.

The above algorithms are intended to solve nonstiff systems. If they appear to be unduly slow, try using one of the stiff solvers below.

`ode15s` is a variable order solver based on the numerical differentiation formulas (NDFs).

`ode23s` is based on a modified Rosenbrock formula of order 2.

`ode23t` is an implementation of the trapezoidal rule using a "free" interpolant. Use this solver if the problem is only moderately stiff and you need a solution without numerical damping.

`ode23tb` is an implementation of TR-BDF2, an implicit Runge-Kutta formula with a first stage that is a trapezoidal rule step and a second stage that is a backward differentiation formula of order two.

10.2 Electrical engineering application

10.2.1 RC circuit in series

The voltage source with constant voltage $U_0 = 100$ V is connected to in series to the resistor $R = 200\ \Omega$ and the capacitor $C = 5\ \mu$F at the moment $t = 0$. The voltage on the capacitor is $u_C(0) = 0$ at this moment $t = 0$.

According to Kirchhoff's voltage law:

$$u_R + u_C = U_0$$

$$Ri + u_C = U_0$$

$$RC\frac{\mathrm{d}u_C}{\mathrm{d}t}+u_C=U_0$$
$$\frac{\mathrm{d}u_C}{\mathrm{d}t}=\frac{U_0-u_C}{RC}$$

Solution We use first-order differential equations with constant coefficients for a solution to a series circuit R, C and constant voltage source U_0.

`[t,uC] = ode45(odefun, [t0, tf], y0)` integrates the system of differential equations $\frac{\mathrm{d}u_C}{\mathrm{d}t}=f(t,u_C)$ (in M-file `RCcircuit`) from time t0 to tf with initial conditions y0.

```
function duC=RCcircuit(t,uC)

R=200;  % R=200 Ohm
C=5e-6; % C=5 mikroF
U0=100; % U0=100V

duC=(U0-uC)/(R*C);  % first-order differential equations
end
```

The first-order differential equations is solved by calling the function `RC_solution.m`

```
function RC_solution
R=200; % R=200 Ohm
C=5e-6; % C=5 mikroF
U0=100; % U0=100V
[t,uC]=ode45(@RCcircuit,[0,0.01],0)

% ode45(@RCcircuit,[0,0.01],0) integrates the system of differential equations RCcircuit.m from time 0 t

subplot(3,1,1)
plot(t,uC,'r')
xlabel('t [s]')
ylabel('u_C [V]')
legend('u_C [V]')

current=(U0-uC)/R; % calculation current flowing through the circuit

subplot(3,1,2)
plot(t,current)
xlabel('t [s]')
ylabel('i [A]')
legend('i [A]')

uR=U0-uC; % calculation resistor voltage

subplot(3,1,3)
plot(t,uR,'g')
xlabel('t [s]')
ylabel('u_R [V]')
legend('u_R [V]')
end
```

These two user-defined functions must be saved under the name `RCcircuit.m` and `RC_solution.m` to the current folder.

Then write to the command line in Command window for solving of RC transient phenomena:

```
RC_solution
```

The graphical output of MATLAB is depicted in Fig. 10.1.

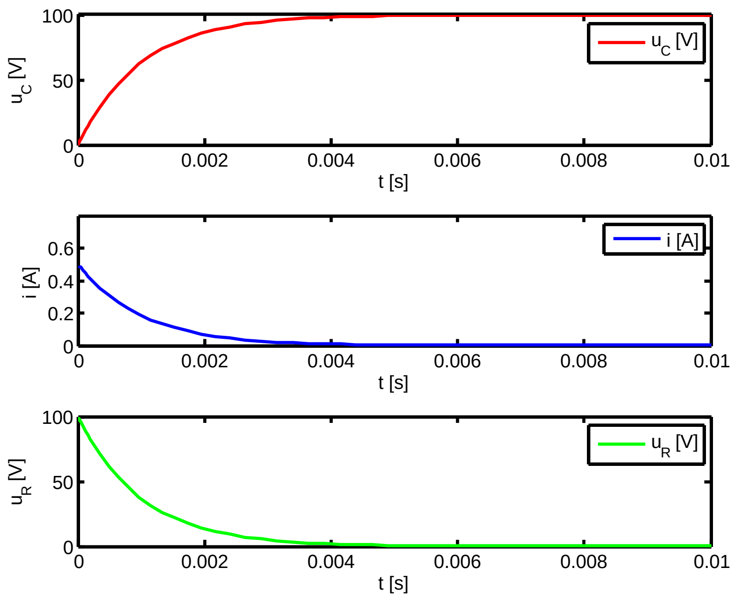


Figure 10.1: The dependence of voltages u_C, u_R and current i on time t during RC transient phenomena

10.3 System of differential equations

Solve a system of three first order differential equations:

$$\frac{\mathrm{d}y_1}{\mathrm{d}t} = -4y_1 + 10y_2y_3$$

$$\frac{\mathrm{d}y_2}{\mathrm{d}t} = 4y_1 - 10y_2y_3 - 30y_2^2$$

$$\frac{\mathrm{d}y_3}{\mathrm{d}t} = 30y_2^2$$

for the interval of integration from 0 to 3 with initial conditions $y_1(0) = 1$, $y_2(0) = 0$, $y_3(0) = 0$

```
function dy=sys_diff_eq(t,y)
% solving of the system of differential equations
% dy1/dt=-4*y1+10*y2*y3
% dy2/dt=4*y1-10*y2*y3-30*y2^2
% dy3/dt=30*y2^2

dy=zeros(3,1);
dy(1)=(-4*y(1))+(10*y(2)*y(3));
dy(2)=(4*y(1))-(10*y(2)*y(3))-(30*y(2)^2);
dy(3)=30*y(2)^2;
```

The solving of the system of differential equations will be performed in the function `solving`.

```
function solving
% solving of system of differential equations
```

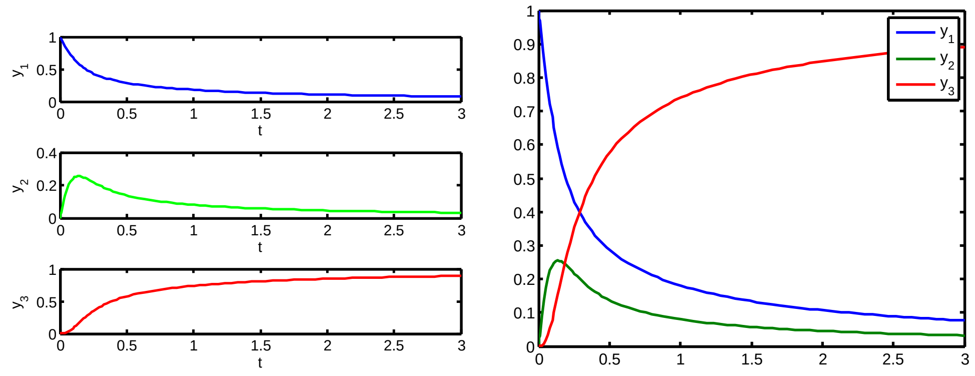


Figure 10.2: The solving of system of three differential equations

```
[t,y]=ode45(@sys_diff_eq,[0,3],[1,0,0]); % solving of system of differential equations
                                         % defined in m-file sys_diff_eq
                                         % interval of integration from 0
                                         % to 3 and vector of initial conditions
                                         % y1(0)=1,y2(0)=0,y3(0)=0
figure % graphical window creating
subplot(3,1,1) % dividing graph to three parts (subplots), active is 1st
plot(t,y(:,1),'b') % 2D plot drawing
xlabel('t')     % x axis label
ylabel('y_1')   % y axis label

subplot(3,1,2) % dividing graph to three parts (subplots), active is 2nd
plot(t,y(:,2),'g') % 2D plot drawing
xlabel('t')     % x axis label
ylabel('y_2')   % y axis label

subplot(3,1,3) % dividing graph to three parts (subplots), active is 3rd
plot(t,y(:,3),'r') % 2D plot drawing
xlabel('t')     % x axis label
ylabel('y_3')   % y axis label

figure % new graphical window creating (second)
plot(t,y) % 2D plot drawing
legend('y_1','y_2','y_3') % legend
```

These two user-defined functions must be saved under the name `sys_diff_eq.m` and `solving.m` to the current folder.

Then write to the command line in Command window for solving of system of three differential equations:

```
solving
```

The graphical output from two Figure window of MATLAB is depicted in Fig. 10.2

10.4 Higher orders differential equations

Example

Solve the Van der Pol differential equation:

$$\frac{d^2 y_1}{dt^2} - \mu(1 - y_1^2)\frac{dy_1}{dt} + y_1 = 0$$

Solution
It's a second order differential equation. We must modify it to the system of first order differential equations, using substitution:

$$\frac{\mathrm{d}y_1}{\mathrm{d}t} = y_2$$

We substitute substitution into Van der Pol differential equation:

$$\frac{\mathrm{d}y_2}{\mathrm{d}t} - \mu(1-y_1^2)y_2 + y_1 = 0$$

We modify this equation:

$$\frac{\mathrm{d}y_2}{\mathrm{d}t} = \mu(1-y_1^2)y_2 - y_1$$

We obtain the system of two differential equations:

$$\frac{\mathrm{d}y_1}{\mathrm{d}t} = y_2$$

$$\frac{\mathrm{d}y_2}{\mathrm{d}t} = \mu(1-y_1^2)y_2 - y_1$$

and we proceed as in the previous example.

There are three ways to define system of differential equations in MATLAB:
1st way:

```
function dy=fce_vdp(t,y)
mi=1;
dy=[y(2);mi*(1-y(1)^2)*y(2)-y(1)];
```

or 2nd way:

```
function dy=fce_vdp(t,y)
mi=1;
dy(1,1)= y(2);
dy(2,1)= mi*(1-y(1)^2)*y(2)-y(1);
```

or 3rd way:

```
function dy=fce_vdp(t,y)
mi=1;
dy = zeros(2,1);
dy(1)= y(2);
dy(2)= mi*(1-y(1)^2)*y(2)-y(1);
```

The solving of the system of differential equations will be performed in the function `difrov`.

```
function difrov
% solving of the system of first order differential equations
[t,y]=ode45(@fce_vdp,[0 20],[2 0]);

% solution printing
for cykl=1:length(t)
    fprintf('t = %8.4f\ty1 = %8.4f\ty2 = %8.4f\n',t(cykl),y(cykl,1),y(cykl,2));
end

% graphs
plot(t,y(:,1),'-b','LineWidth',2);
```

```
hold on
plot(t,y(:,2),'--r','LineWidth',2);

title('Solution of the VDP');
xlabel('t');
ylabel('y_{1,2}');
legend('y_1','y_2');
hold off
```

MATLAB answers

```
t =   0.0000 y1 =   2.0000 y2 =   0.0000
t =   0.0000 y1 =   2.0000 y2 =  -0.0001
t =   0.0001 y1 =   2.0000 y2 =  -0.0001
t =   0.0001 y1 =   2.0000 y2 =  -0.0002
t =   0.0001 y1 =   2.0000 y2 =  -0.0002
t =   0.0002 y1 =   2.0000 y2 =  -0.0005

...
...
...

t =  19.9559 y1 =   2.0133 y2 =   0.1413
t =  19.9780 y1 =   2.0158 y2 =   0.0892
t =  20.0000 y1 =   2.0172 y2 =   0.0404
```

Fig. 10.3 shows graphical output the solving of the Van der Pol differential equation, y_1 and $y_2 = \frac{dy_1}{dt}$.

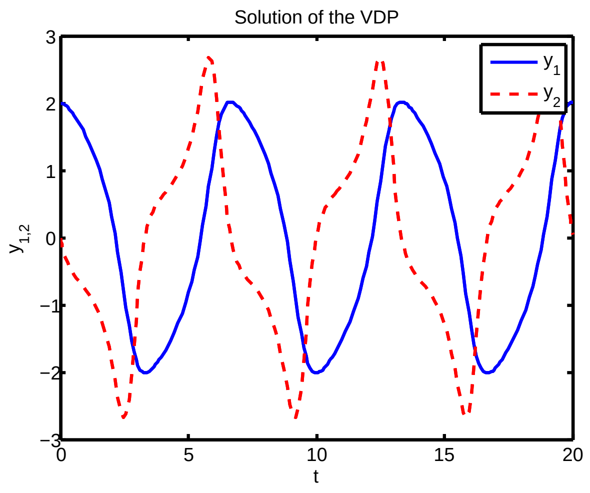


Figure 10.3: The solving of the Van der Pol differential equation, y_1 and $y_2 = \frac{dy_1}{dt}$

10.5 Solver parameters settings

Example

Solve differential equation

$$\frac{dy}{dt} - y = 0$$

for the interval of integration from 0 to 10 with initial condition $y(0) = 5$.

Solution

We modify this equation:

$$\frac{\mathrm{d}y}{\mathrm{d}t} = y$$

```
function dy=diff_eq(t,y)
dy=y;
```

The solving of the differential equation will be performed in the function `sol_diff_eq`. The input parameters of the this function are:
`ts` - the interval of integration,
`y0` - initial condition.

```
function sol_diff_eq(ts,y0)
[t,y]=ode45(@diff_eq,ts,y0);
% graph
plot(t,y)
% solution printing
for c = 1:length(t)
    fprintf('t = %8.4f\ty = %8.4f\n',t(c),y(c));
end
```

For `ts` vectors with two elements `[0, 10]`, the solver returns the solution evaluated at every integration step.

```
sol_diff_eq([0,10],5)
t =   0.0000 y =   5.0000
t =   0.0502 y =   5.2576
t =   0.1005 y =   5.5285

...
...

t =   9.5422 y = 69690.3442
t =   9.7711 y = 87612.6099
t =  10.0000 y = 110152.6760
```

For `ts` vectors with more than two elements, the solver returns solutions evaluated at the given time points. The time values must be in order, either increasing or decreasing, for example:

```
sol_diff_eq([0:2:10],5)
t =   0.0000 y =   5.0000
t =   2.0000 y =  36.9461
t =   4.0000 y = 273.0072
t =   6.0000 y = 2017.3417
t =   8.0000 y = 14906.8157
t =  10.0000 y = 110152.6760
```

or

```
sol_diff_eq([10:-1:0],5)
t =  10.0000 y =   5.0000
t =   9.0000 y =   1.8399
t =   8.0000 y =   0.6769
t =   7.0000 y =   0.2490
t =   6.0000 y =   0.0917
t =   5.0000 y =   0.0337
t =   4.0000 y =   0.0124
```

```
t =   3.0000 y =   0.0046
t =   2.0000 y =   0.0017
t =   1.0000 y =   0.0006
t =   0.0000 y =   0.0002
```

Attention

In this case, the vector `ts` decreases from 10 to 0 and initial condition applies to $y(10) = 5$.

The fourth input argument of `[t,y] = solver(odefun,[t0, tf], y0, options)` is `options`.

Using `options`, we can be set structure of optional parameters that change the default integration properties. We can create `options` using the `odeset` function. We obtain this structure:

```
options = odeset

options =
              AbsTol: []
                 BDF: []
              Events: []
         InitialStep: []
            Jacobian: []
           JConstant: []
            JPattern: []
                Mass: []
        MassConstant: []
        MassSingular: []
            MaxOrder: []
             MaxStep: []
         NonNegative: []
         NormControl: []
           OutputFcn: []
           OutputSel: []
              Refine: []
              RelTol: []
               Stats: []
          Vectorized: []
    MStateDependence: []
           MvPattern: []
        InitialSlope: []
```

Meaning of items in the `options` is as follows

AbsTol Absolute error tolerances that apply to the individual components of the solution vector.

BDF Specifies whether you want to use the BDFs instead of the default NDFs (for ode15s only).

Events Handle to a function that includes one or more event functions.

InitialStep Suggested initial step size.

Jacobian Matrix or function that evaluates the Jacobian.

JConstant Jacobian constant matrix.

JPattern Generates a sparse Jacobian matrix numerically.

Mass Mass matrix or a function that evaluates the mass matrix.

MassConstant Mass constant matrix.

MassSingular Indicates whether the mass matrix is singular.

MaxOrder Maximum order formula used to compute the solution.

MaxStep Upper bound on solver step size.

NonNegative Specifies which components of the solution vector must be non-negative.

NormControl Control error relative to norm of solution.

OutputFcn A function for the solver to call after every successful integration step.

OutputSel Specifies which components of the solution vector are to be passed to the output function.

Refine Increases the number of output points by a factor of Refine.

RelTol Relative error tolerance that applies to all components of the solution vector y.

Stats Determines whether the solver should display statistics about its computations.

Vectorized Allows the solver to reduce the number of function evaluations required.

MStateDependence Dependence of the mass matrix on y.

MvPattern Mass matrix sparsity pattern.

InitialSlope Vector representing the consistent initial slope yp_0.

If we want to set, for example, relative tolerance to 10^{-9}, we run this command

```
options = odeset('RelTol', 1e-9, 'MaxStep', 1000, 'Vectorized', 'on');
```

and we obtain structure

```
options =

             AbsTol: []
                BDF: []
             Events: []
        InitialStep: []
           Jacobian: []
          JConstant: []
           JPattern: []
               Mass: []
       MassConstant: []
       MassSingular: []
           MaxOrder: []
            MaxStep: 1000
        NonNegative: []
        NormControl: []
          OutputFcn: []
          OutputSel: []
             Refine: []
```

```
              RelTol: 1.0000e-009
               Stats: []
          Vectorized: 'on'
    MStateDependence: []
           MvPattern: []
        InitialSlope: []
```

prepared to parametrize the `ode` function calling. Values `[]` means, that we want to leave these parameters on default values.

10.6 User-defined parameters

We have again RC circuit. The voltage source with constant voltage $U_0 = 50$ V is connected to in series to the resistor $R = 10\ \Omega$ and the capacitor $C = 1\ \mu$F at the moment $t = 0$. The voltage on the capacitor is $u_C(0) = 0$ at this moment $t = 0$.

The circuit parameters U_0, R and C are defined in the function `rc_circ.m`. We defined it (for simplification) in 10.2.1 twice – in called function and in calling function. However this is not too good programmer's style. It is necessary to define user parameters one times and send them into the function `in_val.m`. The user haven't direct access to the `in_val` function parameters, but they are available via additional parameters of the `ode` function, next to the `options` variable. It is necessary to provide user parameters during the `ode` function calling at the same order as they are defined in the called function `in_val`.

Parameter values of the `ode23` function are commented in the following user-defined functions, that must be saved under the name `in_val.m` and `rc_circ.m` to the current folder.

```
function duc = in_val(t, uc, U, R, C)
% diff. equation of RCcircuit (serial), DC el. source
% this function obtain parameters U, R, C from the rc_circ function

duc = (U - uc) ./ (R .* C);
```

```
function rc_circ
% Solving of diff. equations
% Solve transient phenomena in serial RC circuit
% in case of connecting to DC voltage source in time t = 0
% using m-file in_val

% Numerical solution
% uc(0)=0 (capacitor is not charged at time T0)
% duc/dt = 1/RC*(U-uc), we compute uc
% ic = C*duc/dt
% Runge-Kutta of 2. and 3. order
% [t,uc]=ode23(@in_val,[begin_time end_time],[initial conditions],options,user_par);
% function ode23 has more parametrs, usually is possible to use implicit
% parameters can be set using function odeset

% circuit parameters setting
R = 10;
C = 10e-6;
U = 50;

% starting time [s]
T0 = 0;
```

```
% ending time [s]
Tfinal = 2e-3;

% initial condition - capacitor is not charged at time T0
Uc0 = 0;

% accuracy setting
% tol = 1e-6;
options = odeset('AbsTol', 1e-6);

% graphical tracing of computation
% options = odeset('outputFCN', @odeplot);

% calling of ODE function - diff. eq. solving
% additional parameters U, R, C are send to the in_val function
[t,uc] = ode23(@in_val,[T0 Tfinal],[Uc0],options, U, R, C);

% current ic computing
% i.e. ic = C * duc / dt, use function diff to compute differential
current_c = C .* diff(uc) ./ diff(t);

% start to create graph
% get screen size to variable scrsz
scrsz = get(0,'ScreenSize');

% create graph windows at the left corner of screen, lower edge distance
% is 1/5 of screen size, 1/2 of screen size high and wide as screen
figure('Position',[1 scrsz(4)/5 scrsz(3) scrsz(4)/2])

% create two subplots
subplot(1,2,1);

% first subplot - time dependency uc (blue color)
% solid line
plot(t, uc, '-b');

% titles and labels
title('RC circuit - u_c');
xlabel('t');
ylabel('u_c');

% second subplot
subplot(1,2,2);

% second subplot - time dependency ic (red color)
% solid line
% draw one element less - current_c is shorter - side effect of diff function
% Test it, for example, with following values: diff([5 8 9 7 10])

plot(t(1:length(t)-1),current_c,'-r');
title('RC circuit - i_c');
xlabel('t');
ylabel('i_c');
```

The graphical output of MATLAB is depicted in Fig. 10.4

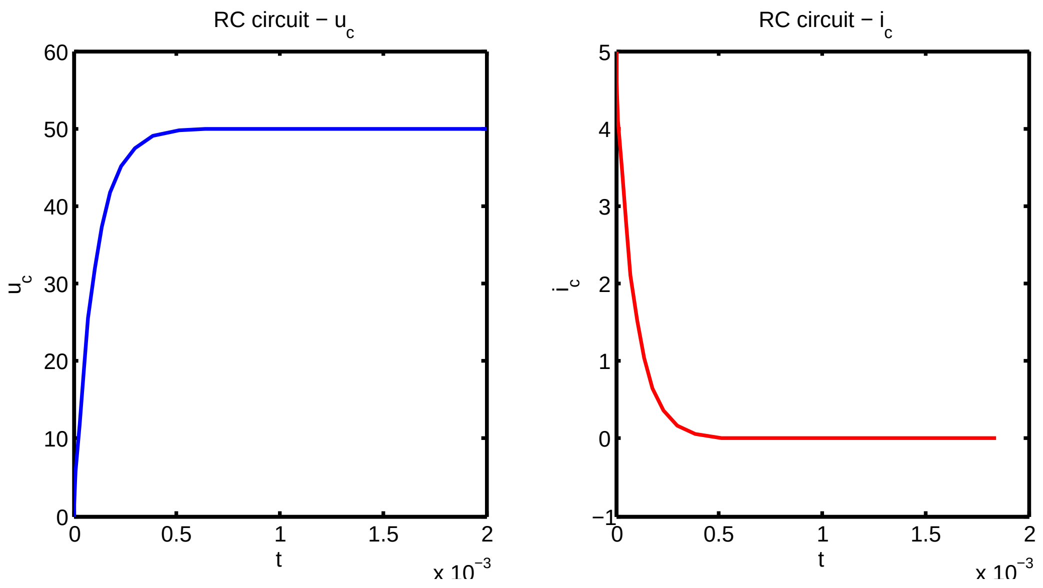


Figure 10.4: The dependence of voltage u_C and current i on time t during RC transient phenomena

10.7 Non-numerical solving of ordinary differential equations

Simpler differential equations is possible to solve non-numerically using the Symbolic Math Toolbox. In this toolbox is available function `dsolve`, which symbolically solves the ordinary differential equations.

Example

Solve differential equation

$$\frac{\mathrm{d}y}{\mathrm{d}t} = y$$

Solution

```
syms y
dsolve('Dy = y')
```

MATLAB answers

```
ans =
C2*exp(t)
```

Example

Solve differential equation

$$\frac{\mathrm{d}y}{\mathrm{d}t} = y$$

with initial condition $y(0) = 5$.
Solution

```
syms y
dsolve('Dy = y','y(0) = 5')
```

MATLAB answers

```
ans =
5*exp(t)
```

10.8 Test of differential equation

1. What solvers do you know for solving differential equations?
2. Solve the differential equation $\frac{\mathrm{d}y}{\mathrm{d}x} + 3y - 5$ for the interval of integration from 0 to 10 with initial condition $y(0) = 2$.

11 Differential equations – advanced part

11.1 Differential equations in electrical engineering praxis

RLC circuit

The voltage source with constant voltage $U_0 = 100$ V is connected to in series to the resistor $R = 200\ \Omega$, the inductor $L = 0.5$ H and the capacitor $C = 5\ \mu$F at the moment $t = 0$ Fig. 11.1 The voltage on the capacitor is $u_C(0) = 0$ at this moment $t = 0$.

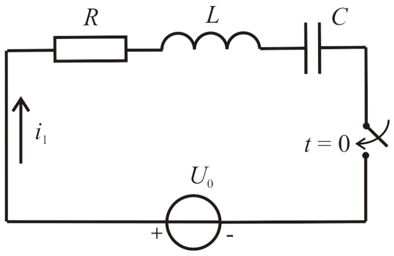


Figure 11.1: Series RLC circuit

According to Kirchhoff's voltage law:

$$u_L + u_R + u_C = U_0$$

$$L\frac{\mathrm{d}i_1}{\mathrm{d}t} + Ri_1 + \frac{1}{C}\int i\mathrm{d}t = U_0$$

$$L\frac{\mathrm{d}\frac{\mathrm{d}i_1}{\mathrm{d}t}}{\mathrm{d}t} + R\frac{\mathrm{d}i_1}{\mathrm{d}t} + \frac{1}{C}i_1 = 0$$

$$L\frac{\mathrm{d}^2 i_1}{\mathrm{d}t^2} + R\frac{\mathrm{d}i_1}{\mathrm{d}t} + \frac{1}{C}i_1 = 0$$

We solve second order differential equation. We must modify it to the system of first order differential equations, using substitution:

$$i_2 = \frac{\mathrm{d}i_1}{\mathrm{d}t}$$

We substitute substitution into differential equation:

$$L\frac{\mathrm{d}i_2}{\mathrm{d}t} + Ri_2 + \frac{1}{C}i_1 = 0$$

We modify this equation:

$$L\frac{\mathrm{d}i_2}{\mathrm{d}t} = -Ri_2 - \frac{1}{C}i_1$$

$$\frac{\mathrm{d}i_2}{\mathrm{d}t} = \frac{-Ri_2 - \frac{1}{C}i_1}{L}$$

We obtain the system of two differential equations:

$$\frac{\mathrm{d}i_1}{\mathrm{d}t} = i_2$$

$$\frac{\mathrm{d}i_2}{\mathrm{d}t} = \frac{-Ri_2 - \frac{1}{C}i_1}{L}$$

This system will be solved in MATLAB.
The inductor voltage:

$$u_L = L\frac{\mathrm{d}i_1}{\mathrm{d}t} = Li_2$$

The resistor voltage according Ohm's law:

$$u_R = Ri_1$$

The capacitor voltage according Kirchhoff's voltage law:

$$u_C = U_0 - u_R - u_L$$

The initial conditions for the solution of the differential equation must be determined. Physical initial conditions for the time $t = 0$ are known:
$i_1(0_-) = i_1(0_+) = i_1(0) = 0$ (current doesn't flow through a circuit)
$u_C(0_-) = u_C(0_+) = u_C(0) = 0$ (the capacitor isn't charged)
From Kirchhoff's voltage law:

$$u_L = U_0 - u_R - u_C$$

The following formula applies:

$$u_L = L\frac{\mathrm{d}i_1}{\mathrm{d}t}$$

Now we substitute u_L into the equation of Kirchhoff's law:

$$L\frac{\mathrm{d}i_1}{\mathrm{d}t} = U_0 - u_R - u_C \Rightarrow \frac{\mathrm{d}i_1}{\mathrm{d}t} = \frac{U_0 - u_R - u_C}{L}$$

For the time $t = 0_+$:

$$\frac{\mathrm{d}i_1}{\mathrm{d}t} = \frac{U_0 - u_C(0)}{L} - \frac{u_R}{L}$$

Now we substitute $u_R = Ri$ into the equation above

$$\frac{\mathrm{d}i_1}{\mathrm{d}t} = \frac{U_0 - u_C(0)}{L} - \frac{Ri(0)}{L}$$

After substituting of physical initial conditions:

$$\frac{\mathrm{d}i_1}{\mathrm{d}t} = \frac{100 - 0}{0.5} - \frac{200 \cdot 0}{0.5} = \frac{100}{0.5} = 200$$

The initial conditions for the our system of two the differential equation are therefore as follows:

$$i_1(0) = 0$$

$$\frac{\mathrm{d}i_1}{\mathrm{d}t}(0) = i_2(0) = 200$$

There are three ways to define the system of two differential equations in MATLAB:

1st way:

```
function dcurrent=RLC_diff_eq(t,current)
R=200; % R=200 Ohm
L=0.5; % L=0,5 H
C=5e-6; % C=5 microF
U0=100; % U0=100V
dcurrent=[current(2);(-1/C*current(1)-R*current(2))/L];
```

or 2nd way:

```
function dcurrent=RLC_diff_eq(t,current)
R=200; % R=200 Ohm
L=0.5; % L=0,5 H
C=5e-6; % C=5 microF
U0=100; % U0=100V
dcurrent(1,1)= current(2);
dcurrent(2,1)= (-1/C*current(1)-R*current(2))/L;
```

or 3rd way:

```
function dcurrent=RLC_diff_eq(t,current)
R=200; % R=200 Ohm
L=0.5; % L=0,5 H
C=5e-6; % C=5 microF
U0=100; % U0=100V
dcurrent = zeros(2,1);
dcurrent(1) = current(2);
dcurrent(2) = (-1/C*current(1)-R*current(2))/L;
```

The solving of the system of two differential equations will be performed in the function `RLC_solution`.

```
function RLC_solution
R=200; % R=200 Ohm
L=0.5; % L=0,5 H
C=5e-6; % C=5 mikroF
U0=100; % U0=100V
[t,current]=ode45(@RLC_diff_eq,[0,0.1],[0,(100/0.5)]);

subplot(2,2,1) % dividing graph to three parts (sub-plots), active is 1st
plot(t,current(:,1)) % 2-D line plot
xlabel('t [s]') % x-axis label
ylabel('i [A]') % y-axis label
legend('i [A]') % legend

% calculations
% the inductor voltage
```

```
uL = L*current(:,2);

% the resistor voltage
uR = R*current(:,1);

% the capacitor voltage
uC = U0-uR-uL;

% graphs:
subplot(2,2,2) % dividing graph to three parts (sub-plots), active is 2nd
plot(t,uC,'r') % 2-D line plot, r - color of line
xlabel('t [s]') % x-axis label
ylabel('u_C [V]') % y-axis label, character _ creates subscript
legend('u_C [V]') % legend, character _ creates subscript

subplot(2,2,3) % dividing graph to three parts (sub-plots), active is 3rd
plot(t,uR,'g') % 2-D line plot, g - color of line
xlabel('t [s]') % x-axis label
ylabel('u_R [V]') % y-axis label, character _ creates subscript
legend('u_R [V]') % legend, character _ creates subscript

subplot(2,2,4) % dividing graph to three parts (sub-plots), active is 4th
plot(t,uL,'m') % 2-D line plot, m - color of line
xlabel('t [s]') % x-axis label
ylabel('u_L [V]') % y-axis label, character _ creates subscript
legend('u_L [V]') % legend, character _ creates subscript
```

These two user-defined functions must be saved under the name `RLC_diff_eq.m` and `RLC_solution.m` to the current directory.

Then write to the command line in Command window for solving of *RLC* transient phenomena:

```
RLC_solution
```

Fig. 11.2 shows graphical output of the solving of the differential equation *RLC* transient phenomena.

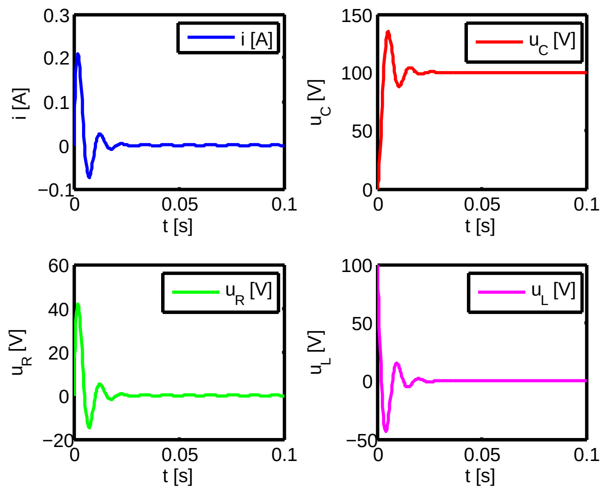


Figure 11.2: The solving of transient phenomena in the RLC circuit

11.2 Test of using of the differential equations in technical praxis

Find $i(t)$ for the series RLC circuit shown in Fig 11.3, if $R = 450\ \Omega$, $L = 15$ H and $C = 150\ \mu$F. The voltage of source is $U_0 = 200$ V. Assume that the capacitor is not charged at time $t = 0$.

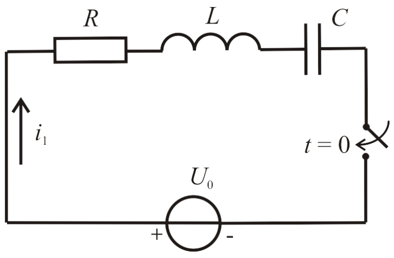


Figure 11.3: Series RLC circuit

12 Creating the comprehensive applications

12.1 Parallel connection of *n* resistors

This comprehensive program allows the user to enter the values of parallel-connected resistors (Fig. 12.1) from the keyboard, write the results on the screen and to a file.

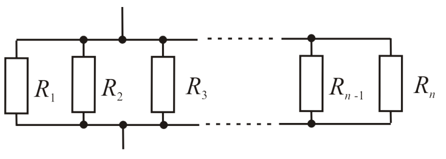


Figure 12.1: The parallel connection of several resistors

Solution
The commands and functions that comprise the new function must be put in a file whose name defines the name of the new function, with a filename extension of '.m'. At the top of the file must be a line that contains the syntax definition for the new function:

```
function [ output_args ] = filename( input_args )
```

The input and output arguments are not required.

We create functions that perform input values??, calculation, screen output, file output. These functions with names `parallel_resistors.m`, `resulting_resistance_sum.m`, `resulting_resistance_for.m`, `output_screen.m`, and `output_file.m` will be stored on the disk in the current directory. These functions will be called in the main function `n_parallel.m`, which will be stored in the same directory.

Keyboard input (output function `parallel_resistors` - vector with the values of resistors):

```
function R=parallel_resistors
n=input('Enter the number of resistor: ');
R=zeros(1,n);
if (n<=0)
    disp('Not what counts :-(')
    return
end
for c=1:n
    while(1)
        fprintf('\nEnter the resistance value R(%d)[Ohm]: ',c)
        R(c)=input('');
        if (R(c)<0)
            fprintf('\nNon-negative value is expected and R(%d)=%.3f[Ohm] have been entered!\n',c,R(c))
            continue
        else
            break;
        end
    end
end
```

The resulting value is obtained built-in functions sum and non-matrix operations (input function - vector with the values of resistors, the output resulting resistance of resistors connected in parallel):

The functions `resulting_resistance_sum` and `resulting_resistance_for` use functions `tic` and `toc`, which work together to measure elapsed time. The function `tic` starts a stopwatch timer. The function `toc` reads the stopwatch timer.

```
function Rp1=resulting_resistance_sum(R)
tic
Rp1=(sum(1./R))^(-1);
toc
```

The resulting value is obtained `for` loop (input - vector with the values of resistors):

```
function Rp2=resulting_resistance_for(R)
tic
s=0;
for n=1:length(R)
    s=s+(1./R(n));
end
Rp2=1./s;
toc
```

Screen output (input - vector with the values of resistors and the resulting resistance):

```
function output_screen(R,Rp)
fprintf('\nEntering resistance values are:\n')
for c=1:length(R)
    fprintf('R%d = %8.3f Ohm\n',c,R(c));
end
fprintf('\nThe resulting resistance value of the parallel-connected resistors: %8.3f Ohm\n',Rp)
```

Output to a file (input - vector with the values of resistors, resulting resistance of resistors connected in parallel and character. If the character is the letter `'A'` or `'a'`, the data will be stored in the file):

```
function output_file(R,Rp,soub)
soub=upper(soub);
if (soub == 'A')
    name=input('\nEnter the file name: ','s');
    fd=fopen(name,'w');
    fprintf(fd,'\nEntering resistance values are:\n');
    for c=1:length(R)
        fprintf(fd,'R%d = %8.3f Ohm\n',c,R(c));
    end;
    fprintf(fd,'\nThe resulting resistance value of the parallel-connected resistors: %8.3f Ohm\n',Rp);
    fclose(fd);
end
```

All previous function call in one function (main program) `n_parallel`:

```
function n_parallel
R=parallel_resistors;
if (min(size(R))==0)
    return
end

how=input('\nUsing the sum or for? Enter a character S or F: ','s');
how=upper(how);
switch (how)
    case 'S'
        Rp=resulting_resistance_sum(R);
    case 'F'
        Rp=resulting_resistance_for(R);
    otherwise
        disp('Bad choice');
        return
end

output_screen(R,Rp);

s=input('\nWould you like to write the results to a file? If so, enter A: ','s');
output_file(R,Rp,s);
```

After the call of function `n_parallel`, the user is prompted to enter data. Diary of Command window:

```
Enter the number of resistor: 5

Enter the resistance value R(1)[Ohm]: 1

Enter the resistance value R(2)[Ohm]: -3

 Non-negative value is expected and R(2)=-3.000[Ohm] have been entered!

Enter the resistance value R(2)[Ohm]: 3

Enter the resistance value R(3)[Ohm]: 4

Enter the resistance value R(4)[Ohm]: 2

Enter the resistance value R(5)[Ohm]: 5

Using the sum or for? Enter a character S or F: S

Elapsed time is 0.000082 seconds.

Entering resistance values are:
R1 =    1.000 Ohm
R2 =    3.000 Ohm
R3 =    4.000 Ohm
R4 =    2.000 Ohm
R5 =    5.000 Ohm

The resulting resistance value of the parallel-connected resistors:    0.438 Ohm

Would you like to write the results to a file? If so, enter A: A
```

```
Enter the file name: data.txt
```

Users enter twice the value of R_2 due to his an error (negative value). Data were stored in a file `data.txt`. The file name chosen by the user. The file appears in the current directory.

12.2 Quadratic equation solver with GUI

Create a solver with GUI, which solve quadratic equation. Try created program – solve quadratic equation $10x^2 + 36x - 65 = 0$.

Solution

We use inputdlg and msgbox included in predefined dialog boxes set in MATLAB standard libraries.

```
function diakvadr
    message1 = sprintf('Coeficient a');
    message2 = sprintf('Coeficient b');
    message3 = sprintf('Coeficient c');

    message = {message1, message2, message3};
    dlgTitle = 'Quadratic equation';
    numOfLines = 1;
    presel = {'1','1','1'};
    usrInp = inputdlg(message, dlgTitle, numOfLines, presel);
    usrInp = char(usrInp);
    if strcmp(usrInp, '')
        return;
    end;
    x = str2num(usrInp(1,:));
    y = str2num(usrInp(2,:));
    z = str2num(usrInp(3,:));
    if ((x == []) | (y == []) | (z == []))
        return;
    end;
    C=[x,y,z];
    result = roots(C);
    resMessage = sprintf('For input a = %.2f, b = %.2f, c = %.2f\n', x, y, z);
    for loop=1:length(result)
        resMessage = [resMessage, sprintf('Result [%d] is %.2f + %.2fi\n', loop, real(result(loop)), ima
    end;
    msgbox(resMessage, 'Finished evaluation', 'help');
end
```

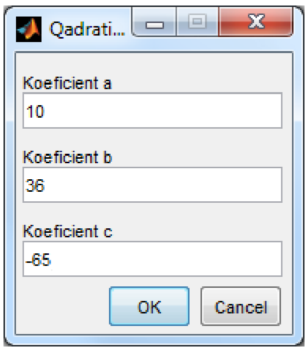
Running this code produced dialogs depicted in 12.2 and 12.3.

12.3 Test of creating the comprehensive applications

Create comprehensive program, which allows the user to enter the values of serially connected resistors (Fig. 12.4) from the keyboard. Calculate the resulting value of the serial connection of resistors.

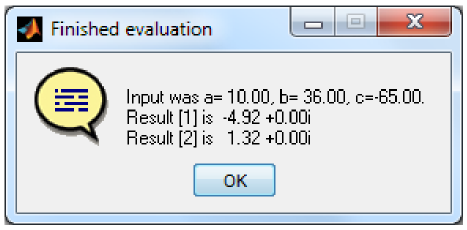
1. Write the results on the screen.

2. Write the results to a file.



Koeficient a: 10
Koeficient b: 36
Koeficient c: -65

Figure 12.2: Input of parameters of the quadratic equation



```
Input was a= 10.00, b= 36.00, c=-65.00.
Result [1] is  -4.92 +0.00i
Result [2] is   1.32 +0.00i
```

Figure 12.3: Results of the quadratic equation

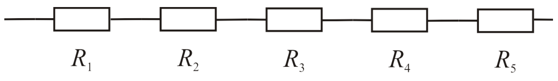


Figure 12.4: The serial connection of several resistors

13 Conclusion and repetitions

13.1 Final test

This section presents the final test, including the correct answers.

1. Which instruction is used for acquiring the number of elements of the 1-D defined vector?

   (a) vector
   (b) length
   (c) diag
   (d) getvect

2. For which type of matrix is impossible to use the transposition mark (`A'`) without change value of some elements?

   (a) for all matrixes
   (b) for matrix with complex elements
   (c) for matrixes with real elements
   (d) for empty matrixes

3. Solve this system of linear equations:

$$5x_1 - x_3 = 2$$

$$6x_1 + 3x_2 = 12$$

$$9x_1 + 4x_2 + x_3 = 20$$

   (a) A = [5,0,-1;6,3,0;9,4,1]; b = [2;12;20]; x = A \ b
   (b) A = [5,0,-1;6,3,0;9,4,1]; b = [2;12;20]; x = A \ b.'
   (c) A = [5,0,-1;6,3,0;9,4,1]; b = [2;12;20]; x = A.' \ b
   (d) A = [5,0,-1;6,3,0;9,4,1]; b = [2;12;20]; x = A / b

4. Which instruction is useful for broken down of the one graphic window into more areas?

   (a) picture on
   (b) subplot

   (c) plot area
   (d) grid on

5. Which value contains the variable x after execution of the next script:

```
x = 100;
while(x > 50)
x = x - 5;
end;
```

   (a) 100
   (b) 50
   (c) 55
   (d) 45

6. Which command is possible to replace the functionality of several commands if-else used for several values of one variable?

   (a) conditional command of loop (cycle) - while
   (b) command switch
   (c) command case
   (d) loop (cycle) for

7. Which instruction is useful for interruption of the actual iteration of loop and proceed go on the rest of program?

   (a) stop
   (b) continue
   (c) break
   (d) end

8. Solve the equation $6x^6 + x^4 - x^3 + 5x^2 + x = 1$.

   (a) `p = [6,1,-1,5,1;1]; x = roots(p)`
   (b) `p = [6,1,-1,5,1,1,1]; x = polyval(p,1)`
   (c) `p = [6,0,1,-1,5,1,0]; x = polyval(p,6)`
   (d) `p = [6,0,1,-1,5,1,-1]; x = roots(p)`

9. Evaluate integral of function $f(t)$ where $t \in \langle -5\pi, 10\pi \rangle$. Function is defined using formula:

$$f(t) = \sin(t^2)\cos^2(t)$$

Create necessary m-file for evaluating integral of this function

$$y = \int_{-5\pi}^{10\pi} f(t)dt$$

Create program in MATLAB (all necessary m-files and command-window commands), use MATLAB integrated function for evaluating of this integral for interval $\langle -5\pi, 10\pi \rangle$.

Print this result to the screen (to the command window) and create a graph of the origin function $f(t)$ for interval of t given by user from command window (ask user the lower and upper limit of interval from your program). Choosing of suitable step for graph evaluation is on your own opinion.

10. Create program in MATLAB (all necessary m-files and command-window commands), which solve given differential equation numerically:

$$\frac{\mathrm{d}y}{\mathrm{d}t} = 10y - 8y^6 + 15$$

Solve this differential equation for given interval $t \in \langle -1, -0.9 \rangle$ and given initial condition $y(0) = 1.2$.

Create a graph of the result (evaluated function). Save solution (vector) y into file.

13.1.1 Final test solution

Questions' solution :

1.(b),
2.(b),
3.(a),
4.(b),
5.(b),
6.(b),
7.(c),
8.(d).

9. Listing of program :

```
function y = myFun(t)
  y = sin(t .^ 2) .* cos(t) .^ 2;
end

function integration
  intg = quad(@myFun, -5*pi, 10*pi);
  fprintf('Integral is %f\n', intg);

  lowerLimit = input('Lower limit of function: ');
  upperLimit = input('Upper limit of function: ');

  t = linspace(lowerLimit, upperLimit, 1000);
  y = myFun(t);
  plot(t,y);
  xlabel('\it{t}');
  ylabel('\it{y}');
end
```

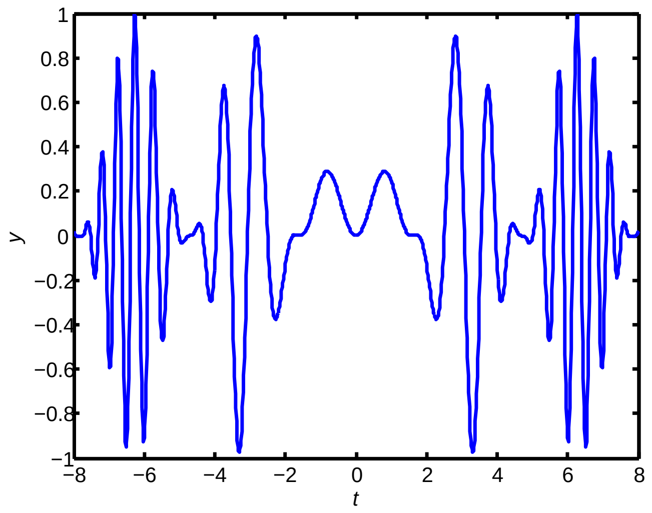


Figure 13.1: The graphical output of function $f(t) = \sin(t^2)\cos^2(t)$ within the limits chosen by user

The function call and answer of MATLAB (listing of Command Window)

```
integration
Integral is 0.427862
Lower limit of function: -8
Upper limit of function: 8
```

The graphical output is depicted in Fig 13.1.

10. Listing of program :

```
function dy=diffEq(t,y)
   dy = 10.*y - 8.*y.^6 + 15;
end

function sol_diffEq
   [t,y]=ode45(@diffEq,[-1,-0.9],1.2);

   plot(t,y)    % graph
   xlabel('t')
   ylabel('y')

   f = fopen('data_y.txt','w');
   for c = 1:length(y)
       fprintf(f,'%8.4f\n',y(c));  % saving of solution into file
   end
   fclose(f);
end
```

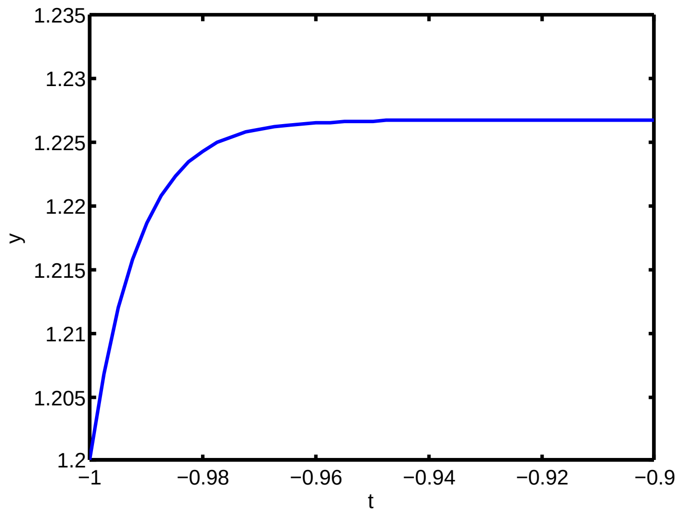


Figure 13.2: The graphical output of solution y of differential equation $\frac{dy}{dt} = 10y - 8y^6 + 15$

The data saved in the file `data_y.txt`

```
1.2000
1.2068
1.2120
...
...
1.2268
```

Fig 13.2 shows the graphical output from MATLAB.

13.2 Conclusion

This MATLAB book and his using in the electro technical praxis appear from the knowledge of the university teachers' team from the West Bohemian University of Pilsen. The book content was chosen with the respect to teaching knowledge from the separate passage and his application in the teaching process with the students.

MATLAB has the most important standard rule in the field of technical computing. Exist cheaper GNU alternative, but in theirs application is necessary calculate with some compromises.

MATLAB has important rule not only in the teaching process, but in the research and scientific work in international scale and his rule in the technical and engineering praxis is impossible compensate with the other alternative programs.

Bibliography

[1] http://mathworks.com.

[2] D. C. Hanselman and B. L. Littlefield. *Mastering MATLAB*. Prentice Hall, 2011.

[3] John Okyere Attia. *Electronics and Circuit Analysis Using MATLAB*. Boca Raton London New York Washington: CRC Press LLC, 1999.