What is MATLAB?

MATLAB is shortcut from Matrix Laboratory. The Mathworks Inc. was developed this software system originally for solving of the linear equation systems through the use of linear matrixes.

And what is Matlab? One of the smart questions from university praxis say – Matlab is the philosophy of the solving mathematical and engineering problems. Matlab is the environmental system and programming language together. It is mathematical oriented computer system based on solving many problems from mathematic and other research section. It is interactive system based on the C language.

In present it is standard for solving vary of engineering and scientific problems from praxis. The origin in the C language cases many similar algorithms and principles of the computation.

Design of MATLAB screen and windows function

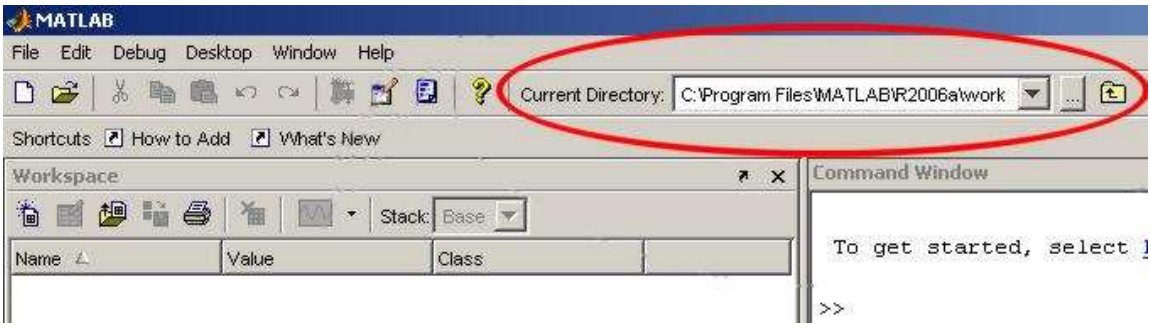
After starting Matlab from the proper directory (depend of the NET administration in the computer pool) appears three windows on the computer screen.

The basic MATLAB environment has **three basic windows**. The largest window (from these three) is for the right half of screen (default – in the right side of the LCD screen). This window has name ***Command Window*** and this is for first lecture the most important. Primary function of this window is working like desktop calculator with "fingers" input and like the numeric output area in the same time.

The left half of screen is broken down between two smaller widows. Upper left from these has name according the lower bookmark or flag and it is ***Current Directory*** or ***Workspace***. These two functions are switchable.

The menu item ***Current Directory*** shows the actual setting of the working directory of actual installed MATLAB. But many installations have in this window the default directory with the name Work and directed into disc C: with the default program installation.

This is important for saving our temporary files or scripts or function (in future exercises). Current directory is directory for storing data, m-files and many more (in future namely). This directory may be set vary, it depends on the setting the main Matlab program. Indication about the actual setting the current working directory is on the small box on the top of the menu bar on the Matlab primary screen.

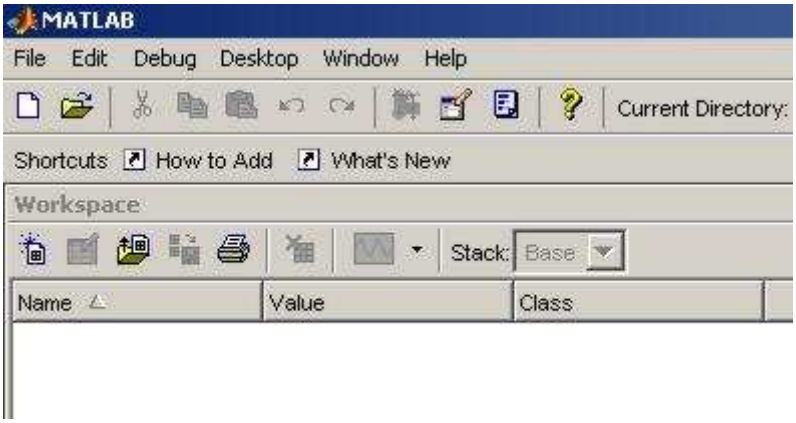


The ***Workspace*** function is more interesting. Window in this style shows the actual defined variables and constants and functions. We will see this in future. Each defined variable (in the common style object) has in this window small yellow symbol, the most usually small square.

The last window has name ***Command History*** and show the flow of used commands from our lecture. It is possible to click on selected command from history line and it is activating.

These windows are possible change and close, but for our starting lecture it is not good idea. If you make some changes and want to return to default style, you must push (from top menu) path Desktop – Desktop layout and Default settings.

The last part of screen is ***menu bar*** on top of screen. It has the similar shape like other Windows programs.



Menu items

File – classical style of Windows programs
(Open items and discuss them – New, Import, Preferences and other).

Desktop – important function for setting the changes of windows to the standard or default style is way **Desktop – Desktop Layout – Default**

Help – the most important item from menu bar. This is ideal for self-study of MATLAB. In Demos is possible finding many of interesting scripts and functions.

Demos – MATLAB – Mathematics – Basic Matrix Operations and next

Command Window

Work in this window has the desktop calculator character. For programmers – it is typical BASIC mode of work (sometimes called interpreter style). User types your code and after pushing key ***Enter*** - Matlab calculates the result. Numeric input by fingers typing is not common for large numeric values; we use it only for this exercise and for training.

In this Command window we will teach and standalone test some simple commands for first par of today lecture.

Next commands we will test (by manual typing) in the largest window – Command window (type it and see what do Matlab).

1. **clc** – most important command – remove all in this window (including error messages)
2. **clear** – remove content our working variables (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 actual 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. **date** – returns the actual system time in the specific format
13. **clock** – return actual time in the vector shape
14. **computer** – returns the type of the computer
15. **!dir** – provides commands of old DOS system, in this case it is dir of disc C:

Very interesting command is **diary**. This command makes complete list of used command and functions with results of mathematical operations to one file. This is like notice book or work pad. This is very useful for lectures like this one.

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

This file is stored in actual working directory defined by ***Current Directory***.

Case sensitivity

The most important axiom in MATLAB is **CASE SENSITIVITY.** In praxis this sentence means that a isn't equal to A **(a ≠ A).** It is very important and many peoples make in this command many errors. And our consensus will be that the vectors and matrixes will have names written thru uppercases.

Notice:

Matrix and vector objects in names use generally upper cases (capitals – f.g. MATRIX) and the real constant, text and numeric values use generally lower case (letters – f.g. scalar). Keep in mind this rule.

Basic mathematical operations

Notice:

Before starting our work in this large ***Command Window***, it is necessary to say, that the style of work is the same like on the usual desktop calculator (that means interpreter style). This is similar to computer language like Basic and different to language like C or Fortran. These are compilation types of work. Not the interpreter style.

Allocation

Allocation – that means operations that join the name of the variable 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 now the command ***whos*** – show us the detailed information)

When we write a = 5, we have two variables. A and a – take care about **case sensitivity**!

A = 'text'

(When we want allocates string variables. In this case it is normal allocation like in case real numeric value.)
For complex values it is very similar. For example definition of complex number A:

A = 2 + i

(On complex variables apply the same operations rules like on usual variables. The differences exist only in case complex. For example in the matrix operation A' and A.'). This problem will be discussed in the next text.

Variable *ans*

Ans is the shortcut for answer and in this variable is stored result of last previous operation including for example inversion and similar other simple function.

List of the basic operation on the real variables is possibly to obtain by using ***help elfun.***

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

In the Command Window is possible working like on the normal scientific calculator.

Using of the ***help*** command (for the group of command with specific consequence)

help / – returns the group of operators and specific characters

help elfun – returns the group of elementary functions
help elmat – returns the group of elementary matrix functions
help funfun – returns the group of special functions and commands
help ops – returns the group of the special characters (the same like help /)

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 this format doesn't affect for the internal computation and internal numeric precisions.

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

And many more. Details obtain after typing ***help format.***

Matrixes

Before starting in the definition of matrixes, it is necessary say a lot of special characters used for these operations – there are parentheses and punctuation marks.

Definition of row vectors:

Row vector *RV* = [1, 2, 3, 4, 5, 6, 7, 8, 9]

Like separator is used in this case comma. Exist the second possibility – and this is using key ***space*** for separating values, but it is not recommended due possibility of mistake with unwished push this key.

Column vector *CV* = [1; 2; 3; 4; 5; 6; 7; 8; 9]

Like vector elements 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 translation between column and row vector forms are used symbol of apostrophe. That means in you want make column vector from row vector, use apostrophe (symbol for transposition).

CV = RV'

Remember:

Parentheses (very often use as the border characters in the vector and matrix definition):

[ ] – brackets (for vectors and matrixes definition)
( ) – classical parentheses (for mathematical operations, indexes, arguments and more)
{ } – braces (useful for special operation, text arguments and more)

All parentheses must be always paired!!!

Separators:

: – colon (separation of indexes, alternate symbol for matrix structures)

; – semicolon (the end of row vector in matrix, separation of commands)

, – comma (vector and matrix element separator, index separator)

Definition of matrixes:

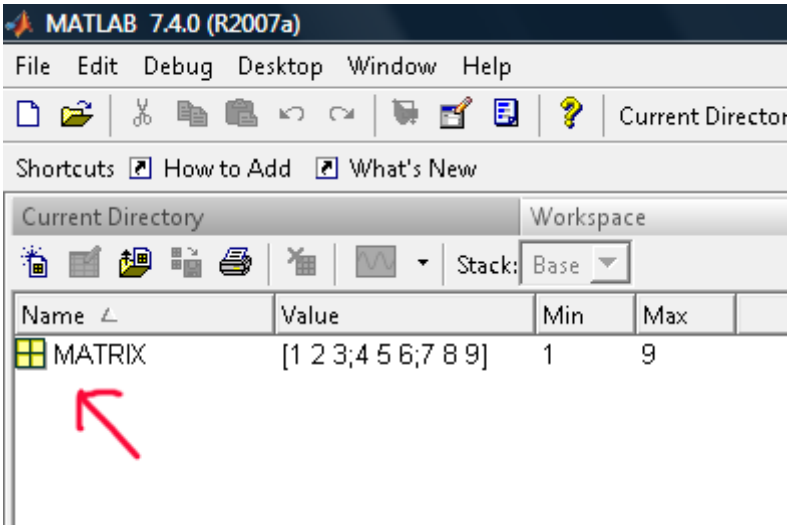
Each matrix has row and column elements sorting into square or rectangular structure. In the most usual cases of technical computations it is square structures and the name is the same – square matrixes. Exist non-square matrixes (rectangle) but its import is very small in the usual technical computation. In the next text we will discussed only square matrixes.

For example matrix: $A = [1, 2, 3; 4, 5, 6; 7, 8, 9]$

That means like border symbols are used brackets. This is important; each type of parenthesis has its own function. Separation of elements – commas, separations of rows – semicolon.

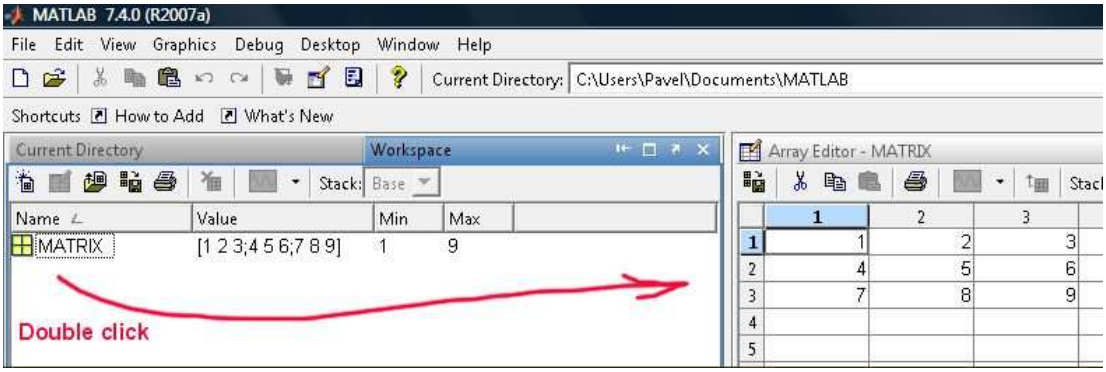
Comma and semicolon have next function, we say about this in the next text.

When we input this matrix into MATLAB, in the left upper window (***Workspace***) appears new definition symbol of our matrix (yellow square) with name.



Notice: Screen reprint was creating from the new version of Matlab R2007a

When we make double click by the mouse on the yellow square symbol (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. After closing this editor changes are valid in the new shape of the matrix.



Structure of matrix

Matrix has own structure. Each element has your own position. It is possible to sign this position with row and column coordinates.

The most important lines in matrixes 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 matrixes are very problematic for many mathematical operations and we don't use these in the next lecture.

Very interesting line in matrix is ***diagonal line of elements***. Each matrix has two diagonals. One diagonal (from upper left corner to the right lower corner) names main and the second diagonal in the opposite direction names adjacent (maybe better will by term - slave).

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. Matrixes are possible download from external sources too.

Example:

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

Inversion form obtain after using function ***inv (A)***

Transpose forms obtain after using function ***A'***

After using function ***size (A***) will be in the ***ans*** variable in this case numbers 3 and 3. That means this matrix has 3 rows and 3 columns.

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

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

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.

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.*

Constructions matrixes from vectors

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

We have vectors A = [1, 2, 3] and B = [4, 5, 6] and C = [7, 8, 9].

Command MATRIX = [A;B;C] construct new matrix with these elements.

Notice:

The same process us useful in case of column vectors too. In case of column vector is used like separator symbol comma.

Arithmetical function with matrixes

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

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

B = [3, 2, 4; 1, 2, 4; 2, 0, 1]

Multiplication

A * B ≠ B * A

A * B is in this case matrix operation
A.* B is not the matrix operation. A and B are in this case arrays not matrixes.

A * B ≠ A. * B

Array operations

A**.** * B = B**.** *A

This operation is defined. They are arrays and this operation multiplies their elements. That means these elements are elements of arrays and not the ones of matrix. This is not the matrix operation! This is very important notice. Sometimes the array operation called operation type *"element by element"*.

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 matrixes 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.

Examples – we have linear row vectors A and B

A = [1, 2, 3] , B = [4, 5, 6].

What will obtain like result of operation A * B? The right answer in this case is error!

But if we use these structures in the dot operation A**.***B obtain the right result [4 , 10, 18] !

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.

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

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

A and B will be matrixes ***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.

A / B = A * inv (B) - In mathematical expression it is $\mathbf{A.B^{-1}}$
A \ B = inv (A) * B – In mathematical expression it is $\mathbf{A^{-1} . B}$

Small notice – 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** (The symbol for this identity matrix maybe vary.)

A \ A = inv (A) * A = **1**

A / A = A \ A (In this condition is this division equivalent operation).

Command for generation of identify matrix is ***eye.*** Example ***eye(5)*** generates 5 x 5 matrix.

The list of all basic operations with matrixes is possible obtain after typing ***help ops***.

Some next examples:

Addition

A + B = B +A (Fully equivalent function. Only the same size of matrixes.)

Subtraction

A – B not the equal B – A

(In these both cases not the dot operator defined. The result will be the same.)

Powering

A^2 = A * A (result in this case is equal to classical matrix operation – multiply)
A.^2 = A. * A (result in this case is power of each element of matrix A)

Predefined constants

ans	variable with last result (ans like answer)
eps	accuracy (respect computer and software possibilities)
pi	3.14159... i.e. " π "
i, j	complex unit , sqrt(-1), it is not good idea rename it or use like index !
inf	infinity, result of certain mathematical operations (for example division "1/0")
NaN	Not-a-Number, result of certain mathematical operation (for example "0/0")
clock	actual date and time (in inversion order)
date	actual date (from computer memory)

Special types of matrixes

eye – identity unit (main diagonal full of ones other elements equal to zero)
zeros – matrix full of zeros (zeros (5) – matrix 5 x 5 full of zeros)
ones – matrix full of ones
diag(A) – vector full of elements from main diagonal matrix A
magic – matrix contains elements having certain unusual properties (sum of row = sum of column and sum of elements on the main diagonal).
pascal – matrix contains the same elements like Pascal triangle (but in square shape)
rand – matrix from randomise numbers from interval (0-1)

Other special types of matrixes is possible obtain after typing ***help elmat***.

Indexing in matrix elements

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

Colon symbol

For example, if we want select second raw from matrix A, type: ***A (2, :)*** This operation has only one condition – the size of matrix A must be equal or higher to 2!

If we want extract first column from matrix A, we must write other command ***A(:, 1)***

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.

M (matrix M has structure 4 x 4)

Second row from M – M (2, :)
Second column from M – M (: , 2)

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.
MATRIX = [1, 2, 3; 4, 5, 6; 7, 8, 9]

And if we want write second and third rows: M (2:3, :)

And if we want write second and third column: M (: , 2:3)

And if we have change rows order (the same): M (3:-1:2, :)
And if we have change columns order (the same) M (: , 3:-1:2)

And like the most beautiful command try this: M (3:-1:1, 3:-1:1)
(This is totally reordered of whole matrix.)

When is necessary choose only one element, we must input his coordinates, for example second element in third raw: ***A (3, 2)***

Like the first parameter is number of raw and second parameter is number of column.

Using these functions allow make vectors or similar indexed structure.

Special forms of matrix are sub matrixes and colon functions (vectors)

For example:

X = [0:0.1:1] - create vector from 0 to 1 with step 0.1 (it is possible use normal parentheses)

X = [1:5] – create vector from 1 to 5 (incremental)
(if the increment is one it is not necessary to write them)

X = [10:-1:0] – decrement style of vector definition (decrement is necessary write in all cases)

If we want make linear vector X from elements 0, 1, 2, 3, …. end element we may write this command X = (0: end)

And 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:

VECTOR = linspace(0, 100, 10)

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:

A = 0: 1 : 10 *is the same like* ***A = linspace(0,10,11) !***

This command makes vector from 0 to 100 with 10 elements. If we want make logarithmic 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.

Linear equation and matrixes principles

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 Kirhoffs laws.

Whole system reduces after using certain rules to the system of linear equations. This linear system is very efficiency solves due to matrixes. The human acceptable style of solving of this linear system based on the Gaussian eliminates method with the step-by-step elimination of separate variables. But Matlab prefer the other method:

We have for example this linear system:

$3x + 4y + 2z = 10$
$2x - 2y - 4z = 2$
$10x - 8y + 2z = 8$

This real linear system we must reduce 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: A^{-1} **.** B
When you write this in MATLAB, it is..

C = inv(A) * B

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

From definition of linear algebra is the same expression like in case ***C = A \ B***. We obtain the same results in both equations.

Complex numbers

In Matlab we have two defined complex units – i and j. It is advices to don't use these characters to other variables like is for example index in the cycle or similar.

When you solve problems from range electrical engineering, you may meet the complex number like elements of matrix. For this complex matrix hold the other computation rules than in the case the usual real elements. Operation named transposition exist in the linear algebra only with the symbol **.** In other side we obtain conjugate matrix, not the transposition! We must use the dot operation mark for the transposition.

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

For complex matrixes 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. COMPLEX ‘ ≠ COMPLEX **.’** Try to explore difference!

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.

Determinant of matrix A we obtain using ***det(A)***

Some special function in technical praxis

Integration

For numeric evaluate integral MATLAB uses function ***quad***, which used for calculation integration function from A to B recursive adaptive Simpson quadrature.

Using this function isn’t so simple.

Q = quad (fun, A, B, tol)

F = @(x) sin (x); (allocate the function of variable x)

Q = quad (F, 0, pi) (and for this interval is result 2)

More details are in the contextual help after typing ***help quad***

Example:

Integral = quad(‘sin(x)’, 0, 2*pi,0.1)

Symbolic variables for analytic calculus

Previous 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.

X=sym (‘x’) – definition of symbolic variables

Y = diff (1/x) – this is request for analytical solving this expression

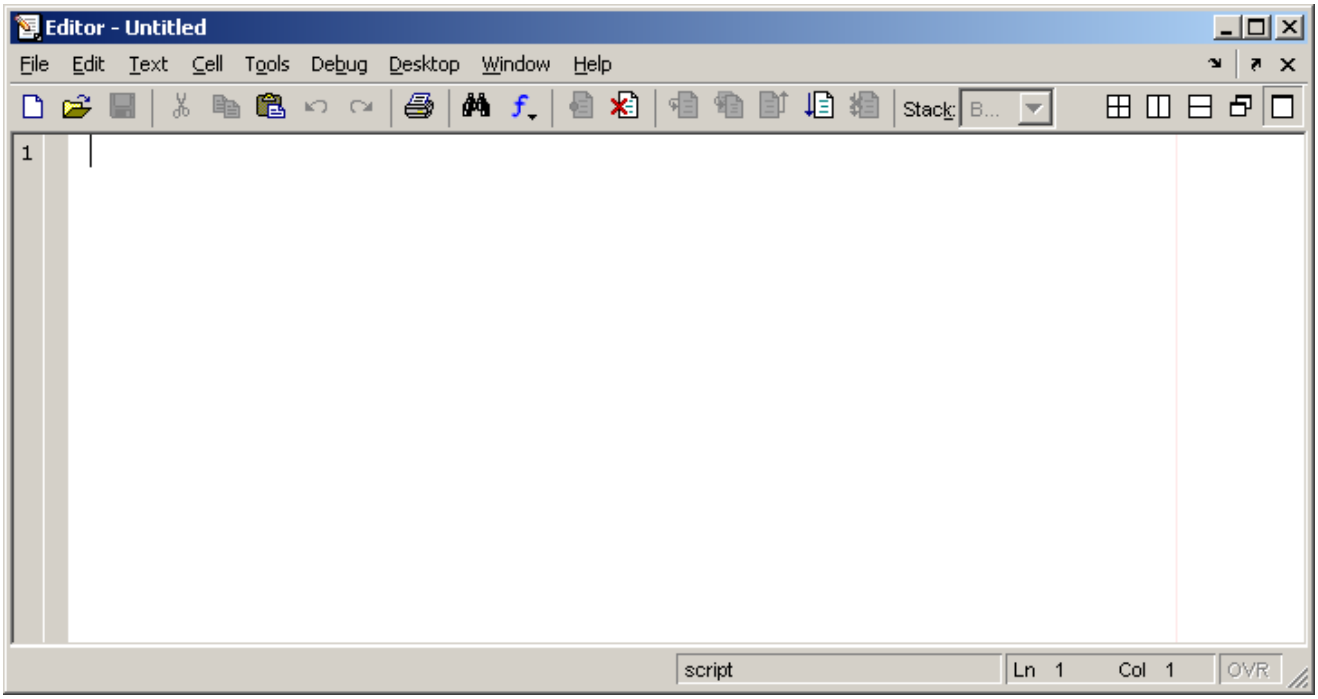
MATLAB answers:

Y = -1/x^2

diff – analytical derivation
int – analytical integration

Second lecture

M-File editor



Working in the interpretation style in the main window (Command window) is not suitable for long group of commands and sentences.

For writing the longer scripts are very useful including M-File editor. M-Files are named scripts and functions which build the skelet of MATLAB function group. Many of commands used in MATLAB are functions and it is possible to edit them.

M-File editor is the next window, which is possible open from menu bar after click sequence.

File – New – M-File

After this sequence is possible open new window named Editor. The basic window has style usual window, but it is small code editor for editing and inputting scripts and functions. This editor has on the left side column from numbers, but these numbers are not useful like row label, only for debugging and error searching.

In this editor is possible trace code, make commentary and many useful things with our scripts.

The main window is possible broken into variable number of sub windows by the clicking on the symbol in menu bar (small windows in the right position in menu bar).

Graphics

The very high power of MATLAB is graphical system. Many graphical instructions including usual graphic, functional oriented graphic, parametric, 2D and 3D graphic a many others. We will try to show some of the most useful from them.

2D graphs

```
%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 on
axis([-pi,pi,-1,1]);

xlabel('-\pi \leq {\itt} \leq \pi');
ylabel('\it sin(t)');

title('\bf \it Sinus function');

text(1,-1/3,'\it {Notice into graph}');
```

Notices to program listing:

Very interesting is the definition of the time basis. It is vector defined due range with colons.

Symbol percent % is used for commentary and this line has green colour.

Command ***plot (t,y)*** by using to draw mark on requested coordinates.

After using ***help plot*** it is clear, that in this case missing the third parameter for definition the marker style, colour and other parameters.

Command ***axis*** defined the borders of plotting area and scaling this graph for these numbers.

Commands ***xlabel*** and ***ylabel*** adds the specific text on the current axes. Very interesting in this case is the parentheses content. The specific text is formatting by text command compatible with the text editor TEX. (For example shortcut ***it*** means italic style).

Command ***title*** makes possible insert the name of current graph (into apostrophes).

Command ***text*** makes possible insert text notice into graph area on the demanding coordinates.

Command legend adds into the picture the legend with text.

legend('sin(x)',4)

The parameter 4 determines the position of the legend into graphic window. Details about this command we obtain – help legend.

Modification – standalone exercises

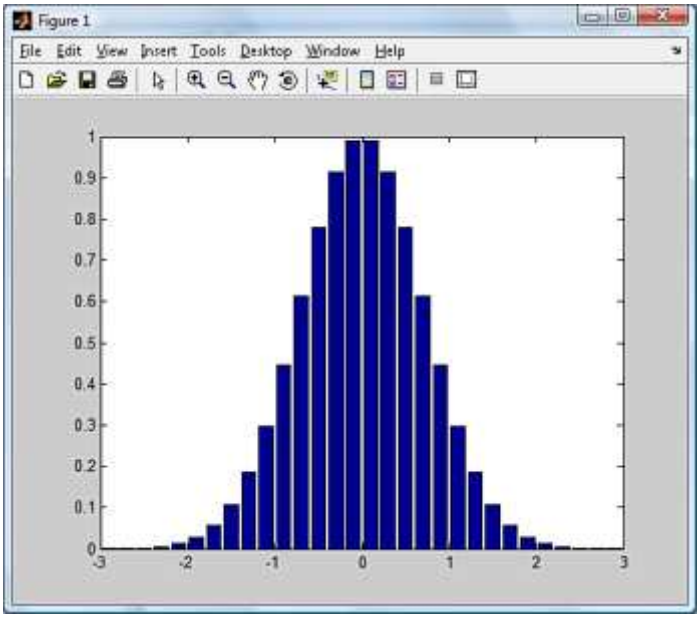
Change step size into time base definition – smother graph
Change colour of graph curve
Change thickness of the graph line
Change the text notice and change their position (fg. inflex point)
Change the setting the legend position.

Special types of 2D graphs

Bar graph

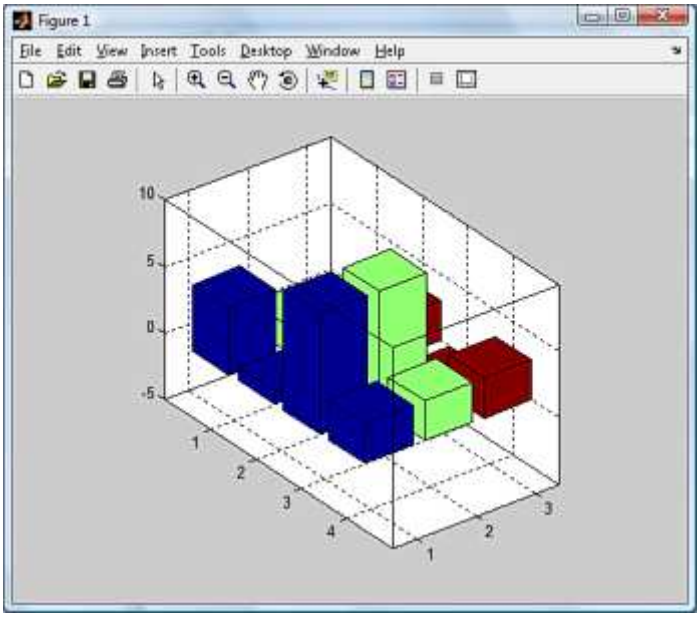
x = -2.9:0.2:2.9;
bar(x, exp(-x.*x))

Result from this script at the next picture



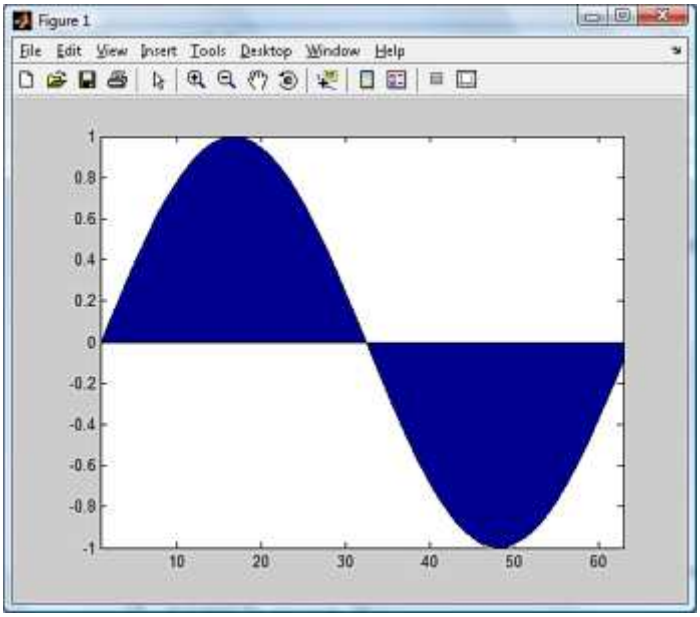
3D bar graph

A=[5 2 1;1 2 3;9 9 -1;3 3 3]
bar3(A)
box on



Area graph

X=0:0.1:2*pi;
Y= sin(X)
area(Y)



3D graphs

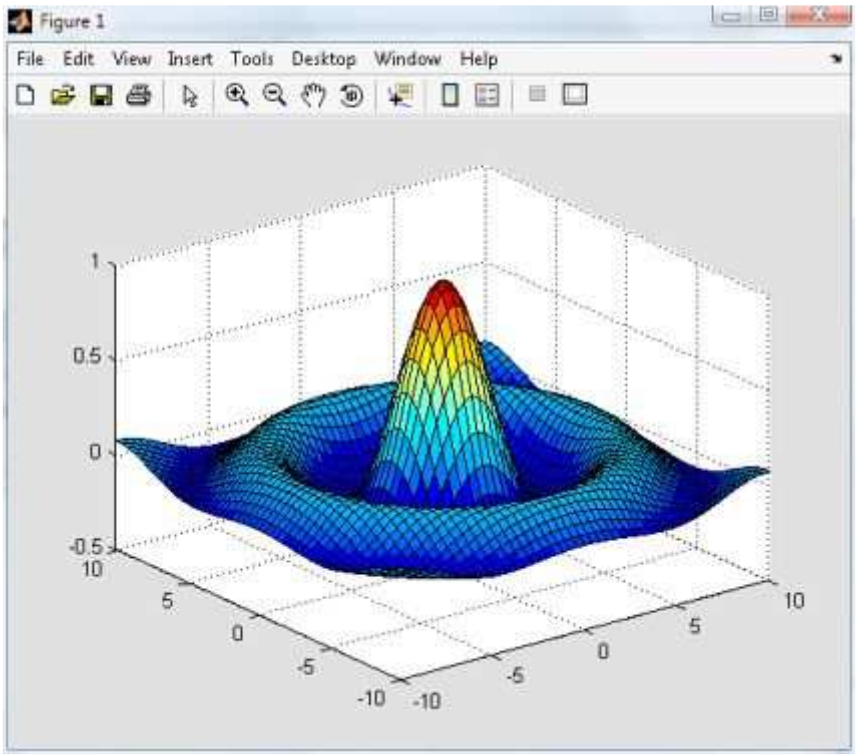
Advanced graphics commands for 3D graphic

```
%Advanced graphic command for 3D functionality
%-----------------------------------------------------------------
%Definition of basic plate mesh

[X,Y]=meshgrid(-10:.5:10);

%Definition of own 3D function
R=sqrt(X.^2+Y.^2)+eps;
Z=sin(R)./(R);

%Graphical output
%----------------------------------
surf(X,Y,Z);
%shading function
shading interp;
```



3D graph creating by function surf without shading

Notices to program listing:

Very important is the definition of the basic plate. It is like “floor” of the graphics system. The verbal translation of the command ***meshgrid*** is the grid with the mesh structure. And the values represent the size of these windows in this mesh.

The most interesting part of this script is the graphic output.

Command ***surf*** (shortcut from surface) makes the graph with the colour surface.

Command ***mesh*** – graph will be in the mesh style.
Command ***surfl*** – graph will be in the surface style with lightning effects.
Command ***surfc*** – graph will be in the surface style with the contour effect

When we want to smooth the surface, we must use the shading instruction. For shading with the interpolation colour scheme it is: ***shading interp***

It is possible to change colour scheme using the ***colormap*** command.

Cold colour scheme in the blue style - ***colormap (cool)***
Hot colour scheme in the red style – ***colormap(hot)***
Copper colour scheme – ***colormap(copper)***

And many more – complete listing the 3D graph possibilities due ***help graph3d***

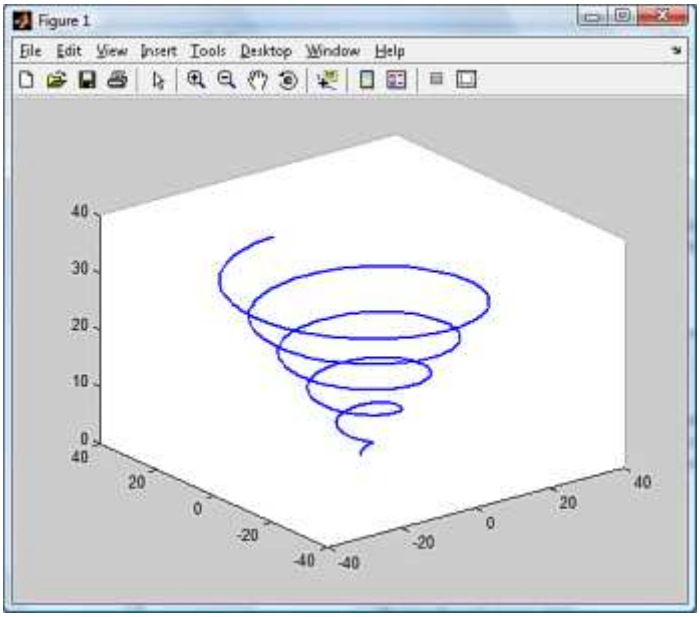
Modification

Using the command subplot (x,y,z) divide the area of the graph window into 4 section. Into each section draw different graph.
1 – mesh
2 – surf
3 – surfl
4 – contour

Other types of graphical interpretation

plot3 command

```
t = 0: pi/50 : 10*pi;
plot3(t.*sin(t), t.*cos(t), t)
```



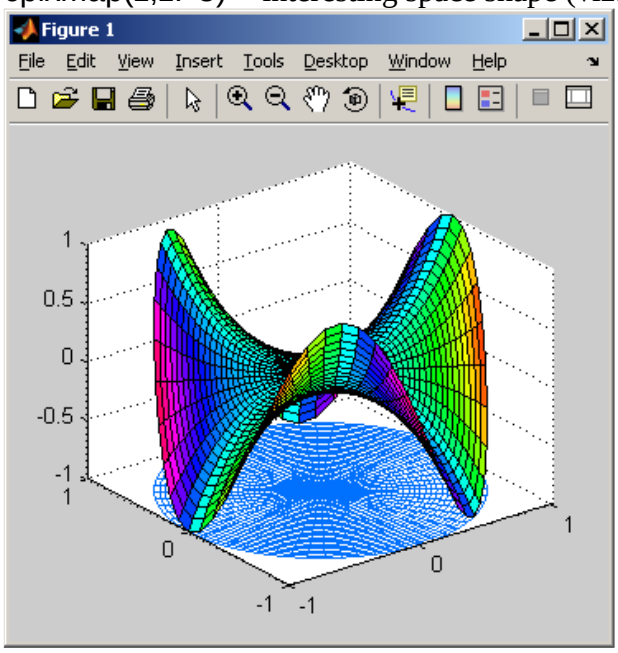
ezplot command

ezplot ('sin(x)')
(creating graph of entered function).

Complex graphs

cplxroot(3) – definition of Reimann space of complex plane $z^{1/3}$

z = cplxgrid(30)
cplxmap(z,z.^3) – interesting space shape (viz. picture)



Polynomial function

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

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

And transcription into vector form is V = [4, 0, 3, 1]

Complete list of all polynomial functions obtains after typing ***help polyfun.***

Command ***roots*** – find polynomial roots. (roots(V) – V is vector polynomial coefficients)
Command ***poly*** – convert roots to polynomial. This function is inversion to function roots.
Command ***conv*** – multiply polynomials
Command ***deconv*** – divide polynomials

Roots

This function calculates the roots of polynomial. If we have the polynomial defined by its coefficients, we may construct vector from these coefficients V = (2, 3, -3)

And roots(V) returns roots for this function

Conv

If we have two polynomials defined by theirs coefficients (vectors V and Q) we may calculate theirs multiply by using command conv(V,Q)

Deconv

It is the same function, but inverse - deconv(V,Q) will provide division of these two polynomials.

Polyval

If we have vector with coefficients we may calculate value for this polynomial in all values describing by other vectors.

Y = polyval (V , X) where X = vector of testing number [0.1, 0.2, 0.3, 2, 3, 4]

Testing interval is possible insert in form vector *X = (0: 0.1 :1)* (if we have not the certain values)

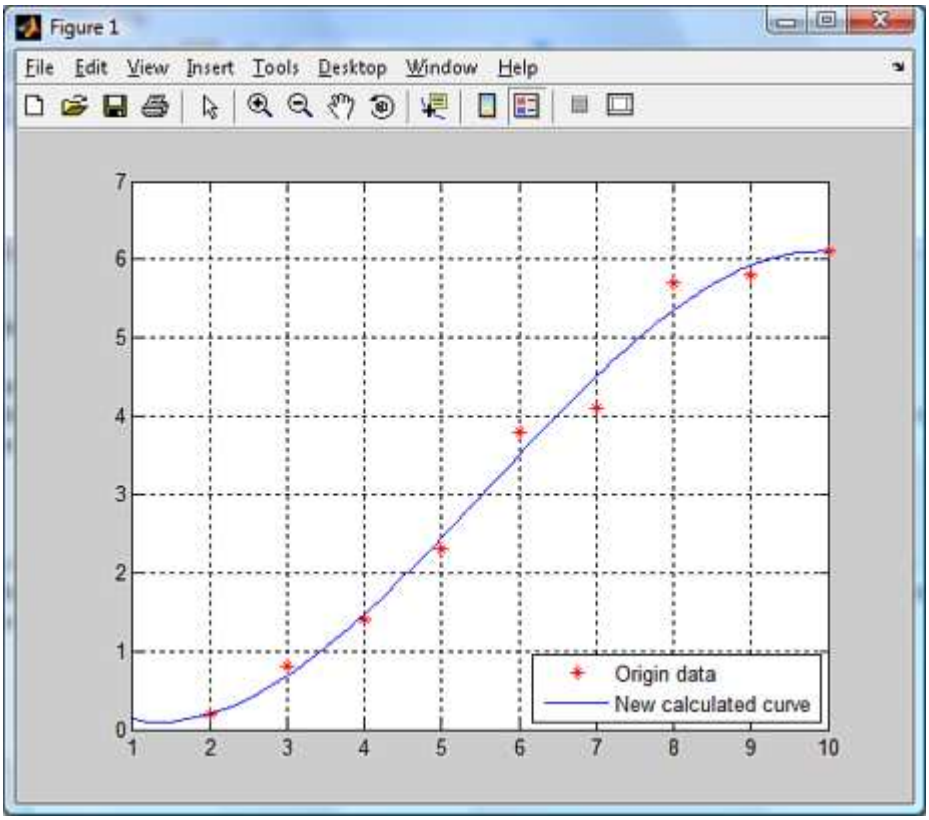
The simplest example of the using polynomial regression

Other function like ***polyfit*** and using ***polyval*** shows this M-File…

```
% Testing the polynomial regression
% ---------------------------------------------------
% Input data sets X and Y
X=1:1:10
Y= [0.1,0.2,0.8,1.4,2.3,3.8,4.1,5.7,5.8,6.1]
% Graphic output
plot(X,Y,'r*')
grid on
% Polynomial regression
POL=polyfit(X,Y,3);
% The new x-axis values
XX=1:0.1:10;
% The new polynomial calculation y-axis values
YY=polyval(POL,XX)
hold on
plot(XX,YY)
legend('Origin data','New calculated curve',4)
```

Notice:

For the fast 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.