NUMERICAL MODELLING OF TECHNICAL ISSUES

1. Introduction to MATLAB system

It is quite difficult to establish a number of technical problems without numerical aid. In practice, scientists and engineers use different numerical systems designed to run given calculations and simulations. To perform these tasks, it is necessary to create a mathematical model for both the studied object and its behaviors in order to run a numeric simulation for it.

Most of the programs are based on ready-made solutions, which not always fully meet our needs and expectations. It does not mean, however, that such products are any less useful. Programs allowing to construct and write the analyzed object by oneself, give a greater freedom in creating models and running numerical simulations. An example of such a program is MATLAB – a MathWorks calculating package designed for various numerical calculations.

The heart of the package is the language interpreter allowing to implement numerical algorithms and basic libraries for matrix calculations (reversing, addition/subtraction, eigenvalues, etc.). The basic data type is a matrix, hence the name MATLAB - MATrix LABoratory. The package includes rich libraries of additional procedures allowing to solve typical calculation problems. The package is characterized by a simple, window layout, which is easy to use, and a simple, esthetic visualization of the results in the form of two- and three- dimensional plots, which adds to its functionality. An additional advantage of MATLAB is the possibility to run symbolic computation (algebra).

2. MATLAB-basic rules of working with the system

Upper and lower case:
MATLAB is case sensitive. It means that, for instance, A and a are not the same values. All commands entered after the "invitations" >> sign has appeared on the screen, have to be in lower case. You can cancel case sensitivity with a command casesen. Entering it again, restores the primary state.
Getting system help:
help <topic> command shows comments on a given topic. Under the <topic> usually a name of procedure is entered. If you enter just help, a list of topics under this command will appear. Other important rules of working in MATLAB include:
Semicolon; used after an expression or command blocks the result displaying
Two (or more) periods .. at the end of a line mean a continuation,
In command editing, there are the following possibilities:
% - marks the following part of the line as a comment,
↑ - shows the previous line
↓ - shows the next line
← - moves the cursor one character left
→ - moves the cursor one character right
Ctrl← - moves the cursor one word left
Ctrl→ - moves the cursor one word right
Home - moves the cursor to the beginning of the line
End - moves the cursor to the end of the line

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI
UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY
Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego

Esc - deletes the whole line
Ins - switches between inserting and overtyping
Del - deletes the character in front of the cursor
Backspace -deletes the character to the left of the cursor.

3. Matlab modes of use

There are four different Matlab modes of use:
Direct mode, that is **entering expressions and commands in the** IDE Matlab command line. In this mode, a single session comprises in entering lines including one or more commands (instructions), separated with commas or semicolons into the command line. ENTER at the end of the line instantly executes the commands included in it, or shows an error message. The direct mode is frequently used as a supplement mode - mainly when learning and testing how different functions or values work.
2) **Indirect ,** program **mode** - used to create programs in the editor. It is possible to run Matlab Editor from the menu File-New, or by clicking the "sheet" icon. After entering a program in the editor, save it into a file. File names follow the same rules as value names (the first character has to be a letter). Normally, the file will have ".m" extension and will be saved in the folder selected as the current folder. It is possible to run the program from the editor (RUN) or from the command line, by entering the file name without the extension (.m). You can also write programs in other simple text editors, such as "Notepad" in MS Windows.
Graphic mode. Matlab is also equipped with means allowing to work with programs having a graphic interface and using dialog boxes known from MS Windows. Although the programs are still written in the editor, they can be partly generated in a semi-automatic way, using the ***RAD*** (*Rapid Application Development*) tool type, called ***GUIDE***. You can put dialog elements (bars, editing fields, etc.) into graphic formats (*figure*) and modify their qualities.
Simulation mode. **SIMULINK** is a Matlab extension - which allows to compile "analog" models of dynamic systems from different function blocks. The models are in the form of flow charts (function charts). One can enter the input values and see the output values. SIMULINK therefore allows to study models in a way similar to the analog machines. Admittedly, it is an additional component of Matlab, but working with Simulink can be regarded as the fourth mode. In Simulink, there is a library of mechanical models, **SimMechanics** which allows to create functioning models of mechanical systems. One can find SimMechanics instructions on the page: *http://www.mathworks.com/help/ toolbox/physmod/mech/* , and movies demonstrating the use of the program can be found at: *http://www.mathworks.com/products/simmechanics/demos.html*

4. Basic elements of Matlab language

Regardless of which Matlab mode you use - you have to know its programming language – which has the same name as the whole package. MATLAB is an advanced programming language, which has, among other things, different data structures, commands controlling execution of programs and input and output data, a variety of functions, as well as object-oriented programming.
As in any other programming language - in Matlab, the content of the program is mainly comprised of **instructions,** also called **commands,** as well as **declarations** - usually omitted in simpler programs. The basic elements of commands are:
key words e.g.: for, end, if, else, while,
constants and variables (including elaborated data structures),
expressions including, among other things, calling a **function.**

5. Scalar expressions and their elements

The expressions - as in other languages - can consist of:
constants (numbers)
variables (variable names)
operators
brackets
functions
However, unlike in other languages - the expressions are related to matrices, which in special cases can be scalars (having a single number values).
Scalar constants - number formats
Similarly as in other programing languages, the number format in MATLAB can include:
plus (usually omitted) or minus sign
a tenth point (NOT A COMMA!) before the fraction part, for example: -97.6397
a so-called scientific notation can be used, in which e stands for "ten to the power of ..." for example - 1.60210e-23 means: -1.60210 times 10 to the power of -23 (-1.60210*10-23)
in complex and imaginary number notation, a symbol of imaginary unit "i" is used, and is always preceded by a number, for example: **1i; 5.7- 3.149j; 2+3e5i.**
Remember that you should not use special characters, which need a special coding. It causes complications in the operation of the interpreter.

6. Operations on numbers

The simplest use of MATLAB is the calculator function, not requiring any programming knowledge. After entering constant values (e.g. 3+5) and typing ENTER, a solution will be shown

```
>>3+5
ans=
     8
```

There are the following arithmetic operations on scalars, and general operations on matrices:

Table 1. Arithmetic operators

Operator	Explanation
+	addition
-	subtraction or sign change
*	multiplication
/	division
^	exponentiation

Apart from arithmetic operations, conditional expressions (conditionals) are important elements of MATLAB language. Such expressions are created using **relational operators** and **logical operators** (table 2 and 3). The relational operators are defined in an ordered set, e.g. a number set. Using logical operators, you can create complex conditional expressions. The negating operator (not) has the highest priority, then there are the remaining logical operators (and, or). Arithmetic operators have lower priority, and relational operators have the lowest priority. That is why in conditional expressions, you should use brackets.

Table 2. Relational operators

Operator	Explanation
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
= =	equal to
~=	not equal to

Table 3. Logical operators

Name	Symbol	
negation	not	~
conjunction	and	&
disjunction	or	|

7. Matrices

The basic data organization type in MATLAB is a two-dimensional matrix. A unique type of matrix is:

scalar - a 1x1 matrix,
row vector - a one row matrix,
column vector - a one column matrix.

In MATLAB, there are a few ways of creating a matrix. The first way is to calculate the elements. Matrix rows are separated with semicolons, and elements with spaces.
Example:
» A=[2 2 2 1; 1 2 3 1]
A =
2 2 2 1
1 2 3 1
Creating a matrix by generating elements:
A=[min:step:max]
The command generates a vector starting from *min* value to the element of *max* value with a *step*. When the *step* parameter is omitted, it takes the default value *step=1*.
Example:
» B=[1:10; 2:2:20]
B =
1 2 3 4 5 6 7 8 9 10
2 4 6 8 10 12 14 16 18 20
Creating a matrix using elements of other matrices.
Example: create a D matrix from A, B and C matrices.
» A=[1 4 1; 2 0 1];
» B=[3 1; 4 1];
» C=[1 2 2 0 1; 2 4 7 1 0];
» D=[A B; C]
D =
1 4 1 3 1
2 0 1 4 1
1 2 2 0 1
2 4 7 1 0
IMPORTANT:
When creating matrices this way, you should remember about the dimension agreement.

Functions aiding matrix creation
1. creating a unit matrix with *nxn* or *mxn dimensions:*
A=eye(n)
A=eye(m,n)
A=eye([m n])
2. creating a matrix with *nxn* or *mxn* dimensions, filled with ones:
A=ones(n)
A=ones(m,n)
A=ones([m n])
3. creating a matrix with *nxn* or *mxn* dimensions, filled with zeros:
A=zeros(n)
A=zeros(m,n)
A=zeros([m n])

8. Methods of graphical data presentation and results calculation

The most common way of a graphic data presentation in MATLAB language is a two-dimensional function plot with one variable. For this purpose, a plot(x,y) of a function is used, where y=f(x). You can erase the graphic window by calling clf function. To close the graphic window, call close function. You can open additional window by calling figure function. You can open and close any window typing its number as an argument. To draw a few plots in one window, use subplot(m,n,p) function, where: m - a number of plots in a column; n - a number of plots in a line; p - a plot's consecutive number.
The scale of a plot is set automatically. To change it, call axis([xmin xmax ymin ymax]) function and type the vector determining the axis' new parameters as an argument. A plot can be described by typing variable names, titles, etc. title('text') - figure title; xlabel('text') - x axis description; ylabel('text') - y axis description; text(x,y,'text') - puts 'text' in any point with (x,y) dimensions; grid - shows or hides the grid..
For more complex variables, three-dimensional graphics are used. Most of MATLAB language functions generating 3D graphics is used to draw planes. In practice, when defining a plane, you should just give a finite set of points belonging to its area.

[x,y]=meshgrid(X,Y)	- generates **x** and **y matrices,** specifying the location of arrays by transforming them from vectors **X** and **Y**
mesh(x,y,z)	- generates mesh of the plane defined by **x**, **y** and **z matrices**
surf(x,y,z)	- generates a colored plane defined by **x**, **y** and **z matrices**
surfl(x,y,z)	- draws a colored plane defined by **x**, **y** and **z matrices, including the reflection of light.**
plot3(x,y,z)	- generates a curve in a space, defined by vectors **x**, **y** and **z**

To be able to generate common mathematical functions (especially trigonometric) in MATLAB, use the commands pre-defined in the system (table 4).

Table 4. Mathematical functions

`sin(x)`	sine
`cos(x)`	cosine
`tan(x)`	tangent
`asin(x)`	arc sine
`acos(x)`	arc cosine
`atan(x)`	arc tangent
`sinh(x)`	hyperbolic sine
`cosh(x)`	hyperbolic cosine
`tanh(x)`	hyperbolic tangent
`asinh(x)`	arc hyperbolic sine
`acosh(x)`	arc hyperbolic cosine
`atanh(x)`	arc hyperbolic tangent
`sqrt(x)`	Square root
`exp(x)`	e^x
`log(x)`	Natural logarithm
`log2(x)`	Logarithm to the base 2
`log10(x)`	Logarithm to the base 10

Functions associated with operations on complex numbers

`abs(x)`	Modulus matrix of **x matrix elements**
`angle(x)`	Argument matrix of **x matrix elements**
`real(x)`	Real part matrix of **x matrix elements**
`imag(x)`	Imaginary part matrix of x matrix elements
`conj(x)`	Conjugate matrix of x matrix elements

Additional functions

round(x)	Rounds the elements of **x** matrix to the nearest integer
rem(x,y)	Returns the remainder after division of the corresponding matrix elements **x** and **y**
gcd(a,b)	Returns the greatest common divisors of numbers a and b
lcm(a,b)	Returns the least common multiples of numbers a and b

9. Algorithms

An algorithm is a finite list of instructions leading to the solution of a function. An algorithm written in a programming language is a program.

Algorithm's characteristics:

correctness - correct solutions for any given input,

finiteness - arriving at a solution after a finite number of operations,

effectiveness - the solution returned in the shortest possible time and using the least amount of memory.

As an example of an algorithm, a solution of the quadratic equation ax2+bx+c=0 will be used.

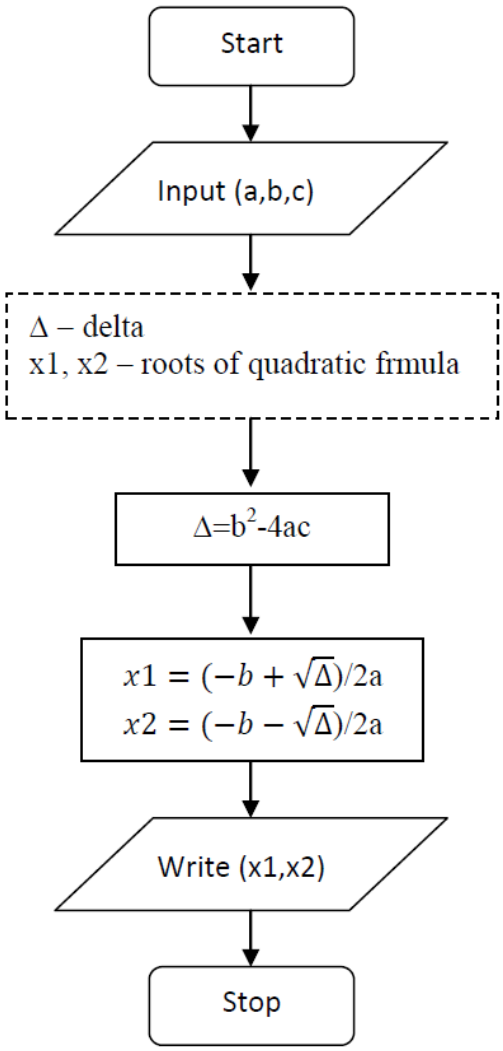


Fig. 1. An algorithm of solving a quadratic equation (general)

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI
UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY
Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego

This program is incorrect, because there occurs such a, b and c for which there is no solution:
for example a=0 - division by 0 error,
a=1, b=0, c=1 - the solution belongs to complex numbers (root of a negative number)
The correct form of the algorithm can be expressed as (modulus form):

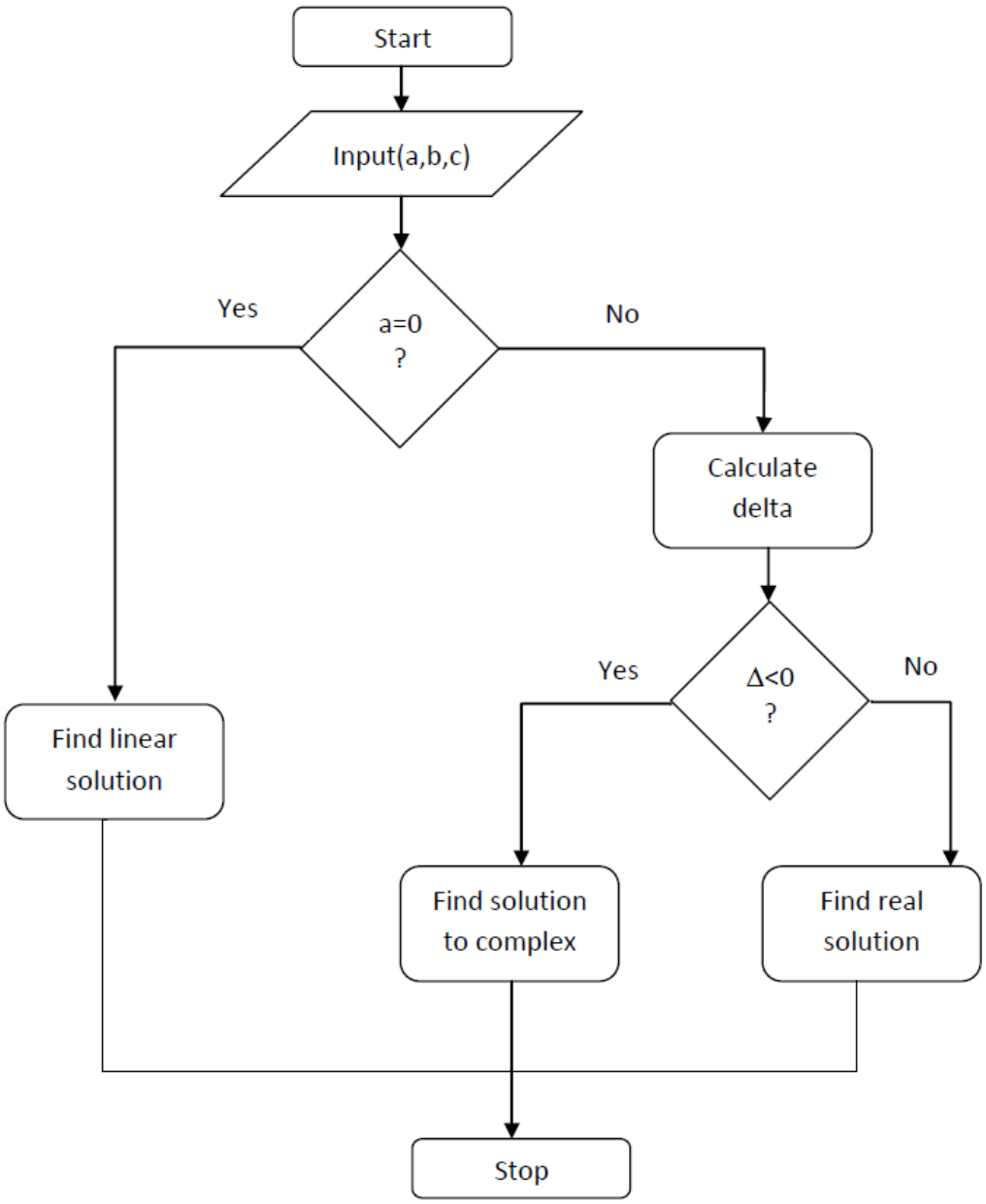


Fig.2. An algorithm of solving a quadratic equation (correct)

10. Instructions and functions in MATLAB

In MATLAB language, to perform complicated calculations, e.g. iteration, instructions and functions are commonly used. The use of these extremely useful tools will be explained on the basis of solutions to particular engineer problems.

Ex. 1. Calculate geometric characteristics of a plane figure comprised of "n" rectangular elements with dimensions b_i and h_i (fig. 1). The characteristics are expressed by the following equations:

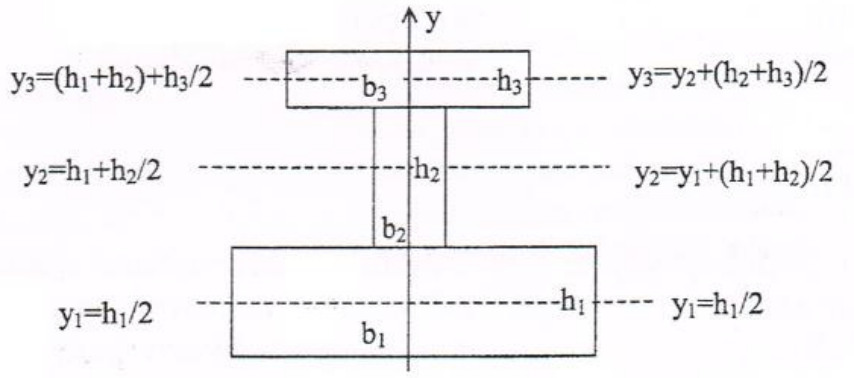
a) plane area $A = \sum_{i=1}^{n} A_i = \sum_{i=1}^{n} b_i h_i$

b) static torque $S = \sum_{i=1}^{n} y_i A_i$ where $y_1 = h_1/2$; $y_i = \sum_{k=1}^{i-1} h_k + \frac{h_i}{2} = y_{i-1} + \frac{(h_{i-1}+h_i)}{2}$;

c) center of gravity $y_c = S/A$;

d) moment of inertia $I = I_o + I_{st} = \sum_{i=1}^{n} \frac{b_i h_i^3}{12} + \sum_{i=1}^{n} d_i^2 A_i$; where $d_i = y_i - y_c$

A graphic calculation scheme has been shown in fig. 3.



The exercise can be solved in two ways:

using **for** instruction,

using matrix or vector operations and functions.

Program code of method 1.

```
clc;
clear;
disp('Calculating geometric characteristics of the cross-section');
%read vector data b and h
b=input('enter width [b1 b2 ... bn]:');
h=input('enter height [h1 h2 ... hn]:');
n=length(b); %number of elements n
AC=0;
I0=0;
for i=1:n,
   A(i)=b(i)*h(i);
   AC=AC+A(i);
   I0=I0+b(i)*h(i)^3/12;
end
y(1)=h(1)/2;
SC=y(1)*A(1);
for i=2:n,
   y(i)=y(i-1)+(h(i-1))/2;
   S(i)=y(i)*A(i);
   SC=SC+S(i);
end
yc=SC/AC;
Ist=0;
for i=1:n,
```

```
    d(i)=y(i)-yc;
    Ist=Ist+A(i)*d(i)^2;
end
I=I0+Ist;
%showing the results
disp('solution: A S yc I')
disp([AC SC yc I]);
```

Program code of method 1.

```
clc;
clear;
%read vector data b and h
disp('Calculating geometric characteristics of the cross-section');
b=input('Width [b1 b2 ... bn] : ');
h=input('Height [h1 h2 ... hn] : ');
n=length(b);
y=h/2;
A=b.*h; %element-wise multiplication
for i=2:n,
    y(i)=y(i)+sum(h(1:i-1));
end
S=y.*A;
yc=sum(S)/sum(A);
I0=b.*h.^3/12;
d=y-yc*ones(size(y));
Ist=A.*d.^2;
I=sum(I0+Ist);
%showing the results
disp('solution: A S yc I');
disp([A S yc I]);
```

IMPORTANT!! In both the above programs, input data should be entered the same as row vector, that is: *Width [b1 b2 ... bn] : [2 5 9 4]*

Ex. 2. Calculate the value of cutting forces and bending moments, draw their progression for a simply supported beam subjected to concentrated force.

First, we will create a file named *beam.m*

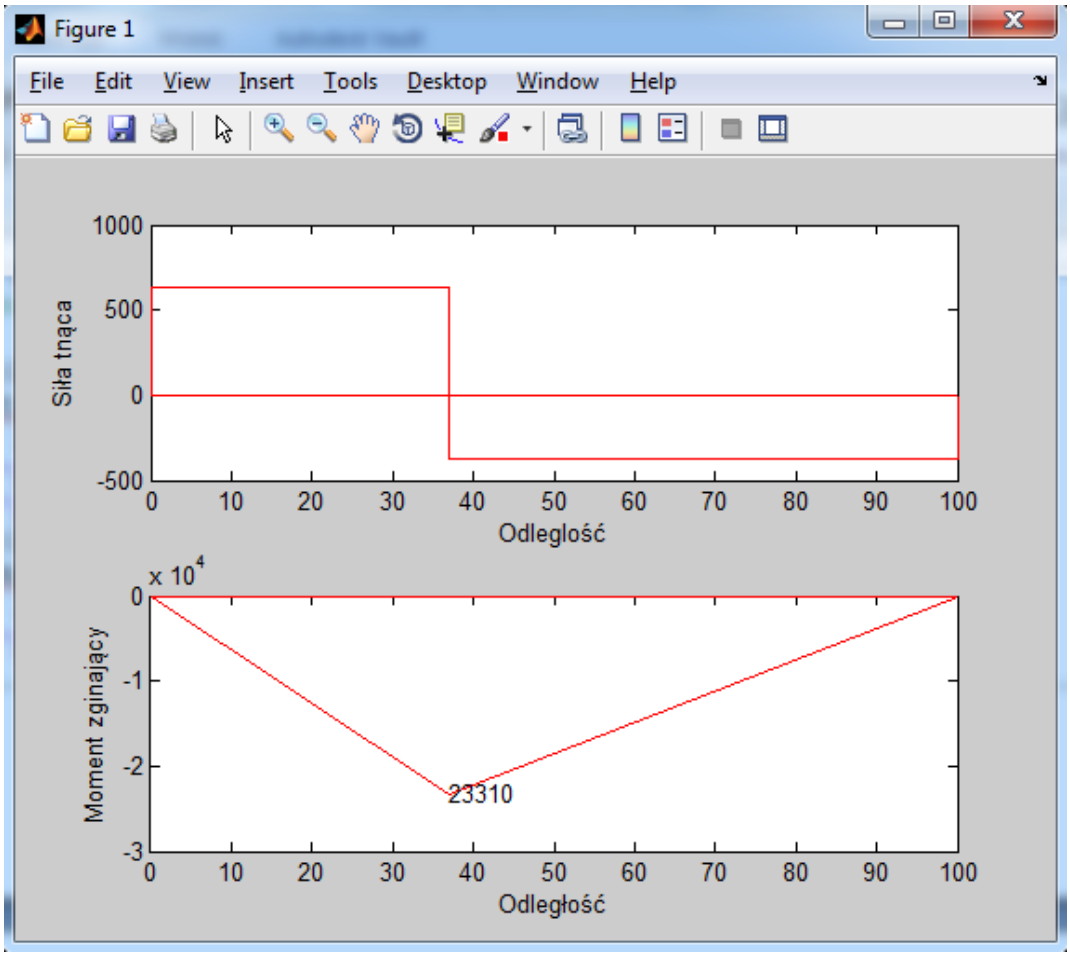
```
>> save beam.m
>> edit beam.m
```

In the dialog box of the editor, enter the following program.

```
%The program draws plots of cutting force and bending moments
%of a simply supported beam subjected to concentrated force.
%Program data:
% l - beam length
% P - concentrated force value
% x - distance between the force's point of application and the left support
clear
clc
disp('The program draws plots of cutting force and bending moments of a simply supported beam')
disp(' subjected to concentrated force applied in a given point of the beam')
disp(' ')
%entering data
l=input('Enter the length of the simply supported beam l= ');
```

```
while l<=0
disp(' !!! Length must be positive value!!!')
l=input('Enter the length of the simply supported beam l= ');
end
P=input('Enter the value of concentrated force P= ');
x=input('enter the distance between the force's point of application and the beam's left support x= ');
while x<0 | x>l
disp(' !!! The force's application point must be within the beam's length !!!')
x=input('enter the distance between the force's point of application and the beam's left support x= ');
end
%calculating the reaction
disp(' ');
disp('Reaction in the left support:');
Ra=P*(l-x)/l
disp('Reaction in the right support:');
Rb=P*x/l
%values of internal forces in the chosen points of the beam
disp('Maximum bending moment:')
Mmax=P*x*(l-x)/l
i=[0 0 x x l l];
T=[0 Ra Ra -Rb -Rb 0];
M=[0 0 -Mmax -Mmax 0 0];
%drawing plots
clfclf
subplot(2,1,1)
plot(i,T,'Color','red')
line([0 l],[0 0],'Color','red')
xlabel('Distance')
ylabel('Cutting force')
subplot(2,1,2)
plot(i,M,'Color','red')
line([0 l],[0 0],'Color','red')
xlabel('Distance')
ylabel('Bending moment')
text(x,-Mmax,num2str(Mmax))
save results.mat
```

The program results:
Below the plots of cutting forces and bending moments for a simply supported beam with length l=100 subjected to force P=1000 applied on the distance of x=37 from the left support have been presented. The plots are examples of the program results:



Odległość-distance
Moment zgninający-bending moment
Siła tnąca-cutting force

Important!! When entering the data, you should remember that with regards to this script they are respectively - *l and x* in [mm], whereas *P* in [N].

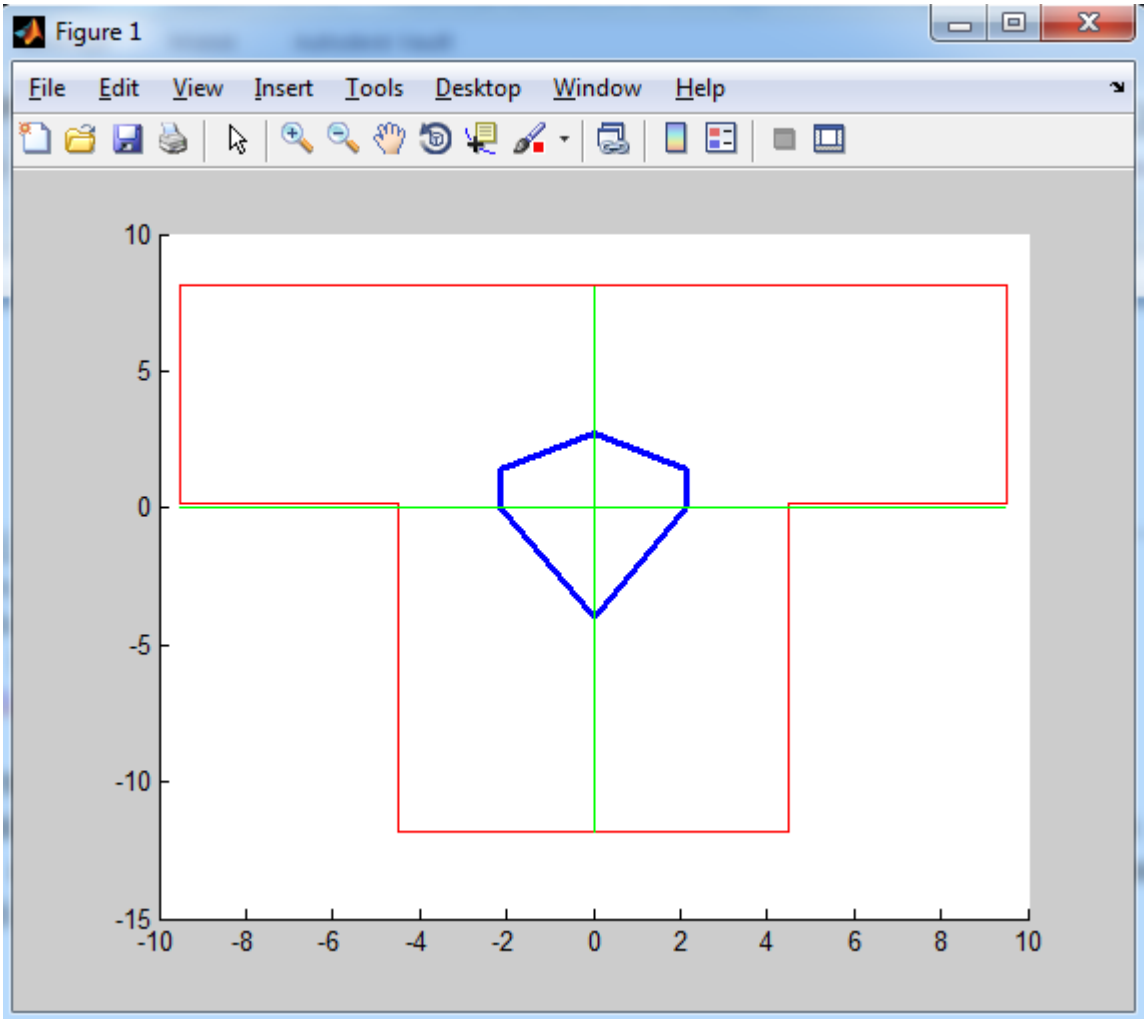
Ex. 3. Calculate geometric characteristics and draw the core of T-cross section/
First, we will create a file *tprofile.m*.
>>save tprofile.m
>> edit tprofile.m
In the dialog box of the editor, enter the following program.

```
%The program calculates geometric characteristics and draws the core of T-cross section
%Program data:
% h - cross section height
% b - flange width
% t - web thickness
% d - flange thickness
clear
clc
disp('The program draws the core of T-cross section')
disp(' ')
```

```
%entering data
h=input('Enter the cross section total height h= ');
while h<=0
disp(' The height must be positive value!')
h=input('Enter the cross section total height h= ');
end
b=input('Enter the flange width b= ');
while b<=0
disp(' The width must be positive value!')
b=input('Enter the flange width b= ');
end
t=input('Enter the web thickness t= ');
while t<=0 | t>=b
disp(' The web thickness must be positive value, less than the flange width!')
t=input('Enter the web thickness t= ');
end
d=input('Enter the flange thickness d= ');
while d<=0 | d>=h
disp(' The flange thickness must be positive value, less than the cross section height!')
d=input('Enter the flange thickness d= ');
end
%geometrical characteristics of the cross section
disp(' ')
disp('Area:')
A=b*d + (h-d)*t
Sx=b*d*d/2 + (h-d)*t*(d+(h-d)/2);
disp('The distance between the center of gravity and the top of the cross section')
yc=Sx/A
disp('Moments of inertia:')
Ix=b*d^3/12 + b*d*(yc-d/2)*(yc-d/2) + t*(h-d)^3/12 + t*(h-d)*(d+(h-d)/2-yc)*(d+(h-d)/2-yc)
Iy=d*b^3/12 + (h-d)*t^3/12
disp('Squares of radius of gyration:')
ix2=Ix/A
iy2=Iy/A
%calculating the peak of the core
u(1)=0;
v(1)=-ix2/yc;
u(2)=-iy2/(b/2);
v(2)=0;
e=(h-d)/(t-b);
x0=(yc+b*e-d)/(2*e);
u(3)=-iy2/x0;
y0=yc+b*e-d;
v(3)=-ix2/y0;
u(4)=0;
v(4)=-ix2/-(h-yc);
u(5)=-u(3);
v(5)=v(3);
u(6)=-u(2);
v(6)=0;
disp('Coordinates of the core peaks in a system going through the cross section's center of gravity :');
```

```
[u' v']
%drawing the cross section and core
clfclf
x=[-b/2 b/2 b/2 t/2 t/2 -t/2 -t/2 -b/2 -b/2];
y=[yc yc yc-d yc-d yc-h yc-h yc-d yc-d yc];
line(x,y,'Color','red');
u(7)=u(1);
v(7)=v(1);
line(u,v,'LineWidth',2.5)
line([-b/2 b/2],[0 0],'Color','green');
line([0 0],[yc-h yc],'Color','green');
save results.mat
```

The program results:
Presented below are the geometric characteristics and a sketch of the T-cross section core, with total height h=20 flange width b=19, web thickness t=9 and flange thickness d=8. They are examples of the program results:



The results of mathematical calculations will be saved in the results.mat file.

Ex. 4. Solve Lame's problem. A thick wall tube with inner radius *a* and outer radius *b* is subjected to internal pressure p_a and external pressure p_b (fig. 18.4). Calculate the radial and circumferential stress in radius function, according to the equations:

$$\sigma_r = \frac{a^2 p_a - b^2 p_b}{b^2 - a^2} - \frac{a^2 b^2 (p_a - p_b)}{(b^2 - a^2) r^2}$$

$$\sigma_t = \frac{a^2 p_a - b^2 p_b}{b^2 - a^2} + \frac{a^2 b^2 (p_a - p_b)}{(b^2 - a^2) r^2}.$$

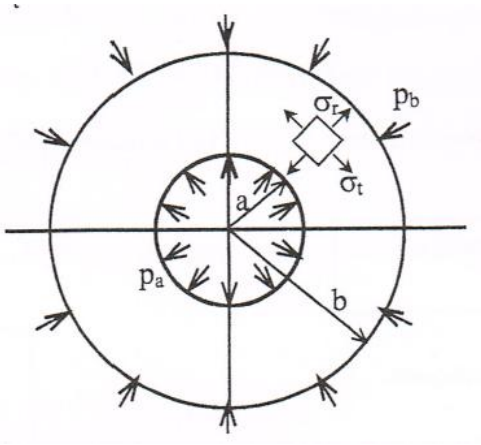


Fig. 18.4. Lame's problem

Create a file lame.m In the code editor, enter the above code.

```
clc;
clear;
%reading data
a=input('Enter the inner radius a= ');
b=input('Enter the outer radius b= ');
pa=input('Enter the internal pressure pa= ');
pb=input('Enter the external pressure pb= ');
if a>=b
   disp('Error!! should be a<b!!')
else
   r=linspace(a,b,20);
   sr=(a^2*pa-b^2*pb)/(b^2-a^2)-a^2*b^2*(pa-pb)/(b^2-a^2)./r.^2;
   st=(a^2*pa-b^2*pb)/(b^2-a^2)+a^2*b^2*(pa-pb)/(b^2-a^2)./r.^2;
   disp('Radial stress : r, sr, st ');
   disp([r', sr', st']);
   subplot(1,2,1);
   plot(r,sr);
   title('Radial stress');
   subplot(1,2,2);
   plot(r,st);
   title('Circumferential stress');
end
```

The code results for exemplary data

```
Enter the inner radius a=44
Enter the outer radius b=55
Enter the internal pressure pa= 1200
Enter the external pressure pb= 344
Radial stress : r, sr, st
  1.0e+003 *

    0.0440   -1.2000    3.5556
    0.0446   -1.1386    3.4942
    0.0452   -1.0796    3.4352
    0.0457   -1.0228    3.3784
    0.0463   -0.9682    3.3237
    0.0469   -0.9155    3.2711
    0.0475   -0.8648    3.2203
    0.0481   -0.8158    3.1714
    0.0486   -0.7687    3.1242
    0.0492   -0.7231    3.0787
    0.0498   -0.6792    3.0347
    0.0504   -0.6367    2.9923
    0.0509   -0.5957    2.9513
    0.0515   -0.5561    2.9117
    0.0521   -0.5178    2.8733
    0.0527   -0.4807    2.8363
    0.0533   -0.4449    2.8004
    0.0538   -0.4102    2.7657
    0.0544   -0.3766    2.7321
    0.0550   -0.3440    2.6996
```

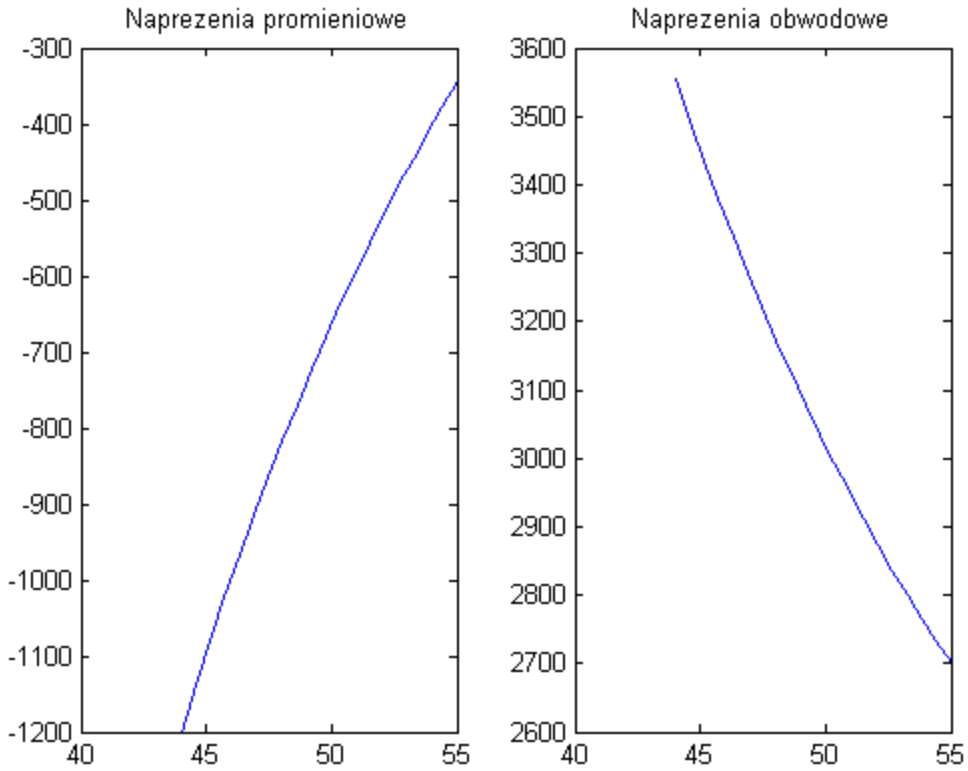


Fig. 5. The results of Lame's problem
Naprężenia promieniowe-radial stress
Naprężenia obwodowe-tangencial stress

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI
UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY
Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego

11. SYMBOLIC Library

Basic symbolic functions	
Name	Symbol
Defining symbolic variable 'a'	sym(‘a’)
Defining symbolic variable with names a,b,c in forms ‘a’ ‘b’ ‘c’	syms a b c
Simplifying the expression of symbolic variable a	sipmplify(a)
Changing the symbolic variable a into a character	char(a)
Substituting the w value for 't' symbol in symbolic variable f	subs(f, ‘t’,w)
Calculating the function inversed to x(t), that is, t(x)	finverse(x,t)
Calculating symbolic expression of a complex function f(g(x))	compose(f,g)
Solving algebraic equations	solve
Solving differential equations	dsolve
Calculating the symbolic expression of the function derivative f	diff(f)
Calculating the symbolic expression of the original function f	int(f)
Drawing a 2D plot of symbolic expression function f, in range of [a b]	Ezplot(f, [a b])