MINISTRY OF EDUCATION AND SCIENCE OF UKRAINE

NATIONAL TECHNICAL UNIVERSITY OF UKRAINE
“IGOR SIKORSKY KYIV POLYTECHNIC INSTITUTE”

BASICS OF WOLFRAM MATHEMATICA

LECTURE NOTES

Tutorial

Recommended by Methodical council of “Igor Sikorsky Kyiv Polytechnic Institute”
as a tutorial for bachelors
according to the educational program “Electronic components and systems”
specialty 171 Electronics

Compiler: K. S. Klen

Electronic online educational publication

Kyiv
“Igor Sikorsky Kyiv Polytechnic Institute”
2022

Reviewer *Naida S.A.*, doctor of technical sciences, professor
"Igor Sikorsky Kyiv Polytechnic Institute"

Responsible editor *Yamnenko Y.S.*, doctor of technical sciences, professor

The stamp was provided by Methodical council of Igor Sikorsky Kyiv Polytechnic Institute
(protocol № 3 from 01.12.2022)
at request of the Academic council of the faculty
(protocol № 09/2022-2 from 26.09.2022)

In the preparation of bachelors in the specialty 171 Electronics, the educational program «Electronic Systems», one of the important disciplines that is a component of the training cycle is the discipline «Basics of Wolfram Mathematica». The purpose of developing a computer workshop is to give students a thorough knowledge of the basic constants, operators and functions of Mathematica, the rules of compiling programs and the purpose of Mathematica extension packages. As a result of studying the material of the computer workshop, the student should gain the ability to do engineering calculations; use visualization tools, user interface and graphing; to make calculation programs.The lecture notes contains theoretical information and tasks for up to 16 lectures and a list of recommended literature.

Register № 22/23-146. Volume 7,5 author's sheet

National technical university of Ukraine
"Igor Sikorsky Kyiv Polytechnic Institute"

Peremohy Ave., 37, Kyiv, 03056
https://kpi.ua

Certificate of inclusion in the State Register of publishers, manufacturers

and distributors of publishing products DK № 5354 dated 25.05.2017

CONTENT

Introduction

The Mathematica software package is a system of computer algebra designed to process mathematical formulas. Other systems of computer algebra include programs Maxima, Maple, MuPAD, Reduce. The main task of these programs is to automate algebraic transformations and processing of symbolic expressions. The first version of the Mathematica package was developed in 1988, version 11.0.1 of the package was released in September 2016.

The computer workshop discusses the main features of the Mathematica system for processing data arrays, plotting graphs, solving linear and transcendental equations, solving problems of mathematical analysis, data approximation, basics of programming and creating a user interface.

Lecture № 1. Introduction to Mathematica

The main features of the Mathematica system:

- integration and differentiation of functions, solution of systems of polynomial and trigonometric equations and inequalities, recurrent equations, solution of differential equations and partial differential equations, Taylor series, simplification of expressions, calculation of limits, finding finite and infinite sums and products, and a number of other problems in the symbolic form;
- polynomial interpolation of functions, calculation of elementary and special functions with a given degree of accuracy, calculation of Laplace transformations, Fourier transformations, z-transformations;
- solving problems of linear algebra, number theory and other sections of mathematics;
- presentation of data in graphic format (construction of graphs, parametric curves and surfaces, construction of geometric figures, import and export of graphic data in raster and vector formats);
- support for distributed computing (package Parallel Computing Toolkit);
- the ability to write programs in the built-in procedural-functional programming language.

In addition, the system has a number of standard extension packages (Add-Ons):

- Algebra – work with polynomials, algebraic inequalities, Hamiltonian algebra, etc.;
- Calculus – symbolic calculations of derivatives, integrals of function boundaries, direct and inverse Fourier and Laplace transforms, solution of systems of nonlinear equations, realization of invariant methods, solution of differential equations in partial derivatives, finding of complete integrals and differential invariants of nonlinear equations, Pade approximation, calculation of elliptic integrals and work with vectors;
- DiscreteMath – calculations in the field of discrete mathematics, combinatorics, computational geometry and graph theory, solving recurrent and difference equations, operations with integers, etc.;
- Geometry – functions for performing geometric calculations, creating regular rectangles and polyhedra, rotating geometric shapes on a plane and in space;
- Graphics – construction of graphs of special type, geometric figures and surfaces, graphs of parametrically and implicitly given functions, description of functions of complex variable, display of orthogonal projections of three-dimensional figures, imitation of shadows, means of graphic design;
- LinearAlgebra – solving problems of linear algebra, additional vector and matrix operations, formation of orthogonal vector bases;
- Miscellaneuos – determination of units of measurement of physical quantities, data on chemical elements, physical constants, geographical data, etc.;
- NumberTheory – functions of number theory;
- NumericalMath – implementation of numerical methods, approximation of data and analytical functions by polynomials, splines, trigonometric series, numerical integration and differentiation, solution of differential equations, calculation of roots of nonlinear equations;

- Statistics – statistical functions for continuous and discrete distribution functions, implementation of linear and nonlinear regression, calculation of parameters of distribution functions;
- Utilities – additional utilities for working with binaries and computer memory, language support, working with AutoCAD class systems, etc..

Basics of working with the system

The interface of the Mathematica system consists of the main menu and data entry area - notebook, figure 1.

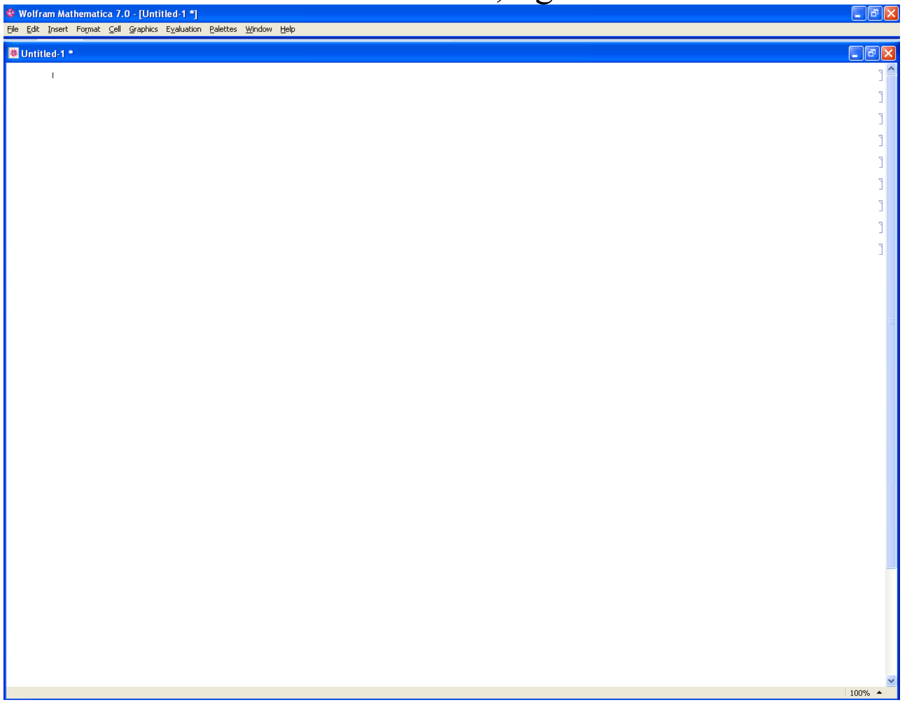

Wolfram Mathematica 7.0 - [Untitled-1 *]
File Edit Insert Format Cell Graphics Evaluation Palettes Window Help
Untitled-1 *
100%

Figure. 1. Mathematica system interface

The working document of the Mathematica system is a notebook. The notebook consists of cells. The easiest way to work with the Mathematica system is interactive. The system assigns a cell number to each input and output – In[n] and Out[n] respectively. To use the last expression, simply enter «%», to refer to the result recorded in cell number n – «%n». To execute the entered command at the end of the line, press Shift+Enter. The result of performing mathematical operations on the expressions listed in Listing 1, will be the expressions written in the output cells Out[1]-Out[6], Listing 2.

```
5^10                          Out[1]=  9765625
4+5                           Out[2]=  9
%+a                           Out[3]=  9+a
%2                            Out[4]=  9
4+5                           Out[5]=  9
9                             Out[6]=  9
```

Listing 1 *Listing 2*

The following rules must be followed when entering data:
- square brackets [] are used to indicate the arguments of functions, even if the function has no arguments, such as Random[];
- curly brackets are used to create lists, vectors and matrices;
- round brackets are used in mathematical expressions to prioritize mathematical operations;

- double square brackets are used to index an item in the list. [[i]] – returns i-th element of list, [[i,j]] – returns the j-th element in the line and i.

Operational help

The system Mathematica has a brief reference to the objects used in its environment. To display the entire list of objects, enter the command:

?*

You can also get help on all objects whose names begin with a certain letter:

?U*

To get a brief help on a specific object, you need to enter the command ?Name, for example:

?Abs

Abs[z] gives the absolute
value of the real or complex number z.

Main menu. The main menu bar has only two lines:

• with the names of the system and the downloaded file;

• main menu items.

To the right and bottom of the edit window are scroll bars with characteristic sliders that can be controlled by the mouse. At the bottom at the beginning of the scroll bar there is a so-called status bar with information about the current mode of operation (Status bar).

The main menu of the system (Figure 1) contains the following tabs:

• File – working with files: creating a new file, selecting an existing file from a directory, closing a file, saving the current file, saving a file with a renamed name, printing a document, and exiting Windows;

• Edit – performing basic editing operations (canceling the operation, copying the selected parts of the document to the clipboard with their subsequent deletion and without it, transferring the selected parts, erasing them);

• Incert – assign input elements (graphs, matrices, hyperlinks, add file elements to the working notebook, select the color of the working cell and number the cells);

• Format – setting the format for documents;

• Cell – work with functional cells (combining and disconnecting cells, setting cell status, opening and closing);

• Evaluation – system kernel management and configuration;

• Palettes – work with palettes of mathematical operators and functions, means of input of mathematical symbols and their options;

• Window – operations with windows and their location;

• Help – management of the help system.

Each menu item, when activated, detaches a drop-down submenu containing related commands. The names of the executed commands are highlighted in clear, and those that are not currently executed - in a characteristic gray blurry font.

Interface elements, such as the editing window, can be moved with the mouse and stretched in different directions. The mouse cursor usually looks like an arrow, but changes when you turn on individual parts of the interface elements. For example, when installed on the vertical border of the window, it takes the form of two-sided arrows$\leftrightarrow$, located horizontally. They indicate the possibility of moving this line horizontally. Similarly, you can stretch or compress a window by moving it vertically or diagonally.

At the beginning of the title lines of the main menu and the editing window there is a button with the system logo, which opens a submenu with the following commands:

- Restore – restore the size of the interface element;
- Move – move the interface element;
- Size – specify the size of the interface element;
- Collapse – collapse an item in a tag in the Windows taskbar;
- Expand – expand the interface element;
- Close – close the interface element.

This submenu is created by means of the Windows operating system. Also, at the end of these lines there are characteristic buttons that repeat the last three commands. They are used to control the windows of the corresponding interface elements.

Lecture № 2. Wolfram Mathematica and environmental data

Wolfram Language allows programmers to work at a much higher level than ever before, using built-in computational intelligence, which relies on a huge depth of algorithms and real knowledge, carefully integrated over three decades. Effective for creating programs from tiny to very large, which naturally allow widespread use both locally and in the cloud, Wolfram Language, based on clear principles and having an elegant unified symbolic structure, creates what becomes the most productive programming language. in the world, as well as the first true computational language of communication between humans and artificial intelligence systems.

Arithmetic operations

Operation	Arithmetic operator	Abbreviated form	Function
Addition	+	+=	Plus[x1, x2,…xn]
Subtraction	-	-=	-
Multiplication	*	*=	Times[x1, x2,…xn]
Division	/	/=	Divide[x1, x2]
Exponentiation	^	-	-

Data types

Type of numbers	Marking	Example
Integers	Integer	35, -2
Rational	Rational	35/46,
Real	Real	2.18, 3.6*10^-5
Complex	Complex	2+3*I

Named constants

E – number e;

Pi – number π;

I – imaginary unit ;

Infinity – imaginary infinity +∞, with negative infinity put the sign -;

Degree – the number of radians in degrees π/180;

EulerGamma – Euler constant 0,577216;

GoldenRatio – constant of the golden section ;

Catalan – Catalan constant 0.915966.

To obtain the numerical value of the constant, it is necessary to use the function N[].

```
N[E]
Out[1]=  2.71828
```

If necessary display the value of a number with a given number of characters, use the function N[k,n], where k – number, n – the number of characters.

```
N[5/88,10]
Out[1]=  0.05681818182
```

Built-in mathematical functions

Functions for determining divisors, the least multiple of integers

Divisors[n] – gives a list of the integers that divide n;
ExtendedGCD[n,m] – gives the extended greatest common divisor of the integers n and m;
GCD[n1,n2…] – gives the greatest common divisor of the n1, n2…;
LCM[n1,n2…] – gives the least common multiple of the n1, n2…
Mod[x,y] – gives the remainder on division of x by y.

Functions of rounding real numbers

Round[x] – gives the integer closest to x;
Floor[x] – gives the greatest integer less than or equal to x;
Ceiling[x] – gives the smallest integer greater than or equal to x;
Quotient[x,y] – returns a rounded integer x/y, less or equal to x/y.

Calculation of factorials

Factorial[n] – gives the value of n!;
Factorial2[n] – gives the value of n!!=n*(n-2)*(n-4)...

Obtaining prime numbers

Prime[n] – gives the n-th prime number;
PrimePi[n] – gives the number of primes Pi(x) less than or equal to x.

Elementary functions

Power and logarithmic functions

$\sqrt{z} \rightarrow$ Sqrt[z];
$z^a \rightarrow$ Power[z,a];
$e^z \rightarrow$ Exp[z];
$\ln(z) \rightarrow$ log[z];
$\log_a(z) \rightarrow$ log[a,z];

Trigonometric functions

$\sin(z) \rightarrow$ Sin[z];
$\cos(z) \rightarrow$ Cos[z];
$tg(z) \rightarrow$ Tan[z];
$ctg(z) \rightarrow$ Cot[z];
$\csc(z) \rightarrow$ Csc[z];
$sec(z) \rightarrow$ Sec[z].

Inverse trigonometric functions

$arc\sin(z) \rightarrow$ ArcSin[z];
$arc\cos(z) \rightarrow$ ArcCos[z];
$arcctg(z) \rightarrow$ ArcCot[z];
$arc\csc(z) \rightarrow$ ArcCsc[z];
$arcsec(z) \rightarrow$ ArcSec[z].

Hyperbolic functions

$\sinh(z) \rightarrow$ Sinh[z];
$\cosh(z) \rightarrow$ Cosh[z];

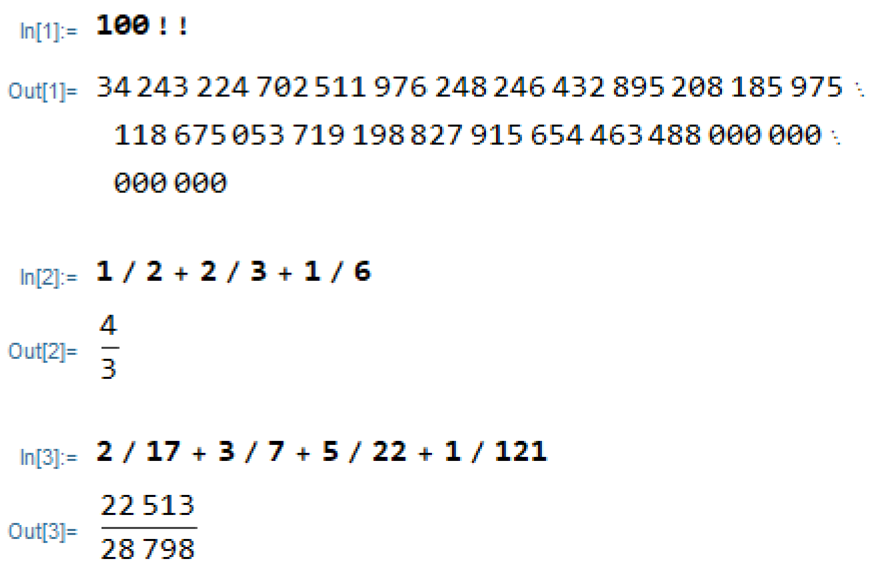
$tgh(z) \to$ Tanh[z];
$ctgh(z) \to$ Coth[z];
$\csc h(z) \to$ Csch[z];
$\sec h(z) \to$ Sech[z].

Inverse hyperbolic functions

$arc\sinh(z) \to$ ArcSinh[z];
$arc\cosh(z) \to$ ArcCosh[z];
$arcctgh(z) \to$ ArcCoth[z];
$arc\csc h(z) \to$ ArcCsch[z];
$arc\sec h(z) \to$ ArcSech[z].

Arithmetic operations with integers and rational numbers

The system Mathematica performs calculations with integers and rational numbers without errors, as illustrated in Listing 1.



```
In[1]:= 100!!

Out[1]= 34 243 224 702 511 976 248 246 432 895 208 185 975 ⋱
         118 675 053 719 198 827 915 654 463 488 000 000 ⋱
         000 000

In[2]:= 1 / 2 + 2 / 3 + 1 / 6

         4
Out[2]= —
         3

In[3]:= 2 / 17 + 3 / 7 + 5 / 22 + 1 / 121

        22 513
Out[3]= ——————
        28 798
```

Listing 1

Arithmetic operations with real numbers

Real numbers in the system Mathematica are presented in the usual or standart form. When representing a number in the usual form, the whole part of the number is separated from the fractional part by point: 1.35, 0.24. Thus 0 integers it is possible not to write and instead of 0.25 to use marking - .25. Numbers written in this form, call numbers with a fixed point.

A point at the end of a number is an indication that the number is real. For example, the number 131. – real, number 2/5 – є rational, and the number 2./5 – дійсним.

When representing a number in standart form, the number is written in the form of a mantis with whole and fractional part and order in the form of a number degree: 5.*10^-3, 3.335*10^-6. You can use a space instead of a multiplication sign. Arithmetic operations on real numbers give an approximate result. The Mathematica system operates with numbers in a range . To increase the accuracy of real numbers can be represented in rational using the functions:

Rationalize[z] – converts number z to a nearby rational;

Rationalize[z,dz] – converts number z to a nearby rational with accuracy dz.

Arithmetic operations with complex numbers

The complex number is represented as follows:

$z = \mathrm{Re}(z) + I * \mathrm{Im}(z)$.

Functions for performing operations on complex numbers:

Abs[z] – gives the absolute value of the complex number z;

Arg[z] – gives the argument of the complex number z;

Conjugate[z] – gives the complex conjugate of the complex number z;

Expressions of their transformation and calculation

Substitutions

Substitutions are a mathematical apparatus designed to calculate the functions at numerically specified values of the argument. They allow you to tabulate the values of functions. The Mathematica system uses the symbol «/.»

f(x) /. x->a;

f(x,y,…) /. {x->a,y->b,…};

{f1(x),f2(x),…} /. x->a;

{f1(x,y,..),f2(x,y,…),…} /. {x->a,y->b,…};

f(x) /. x->{x0,x1,…}.

Substitution expressions have the following meaning:

f(x) /. x->a – substitutes in the expression f(x) the value of x=a.

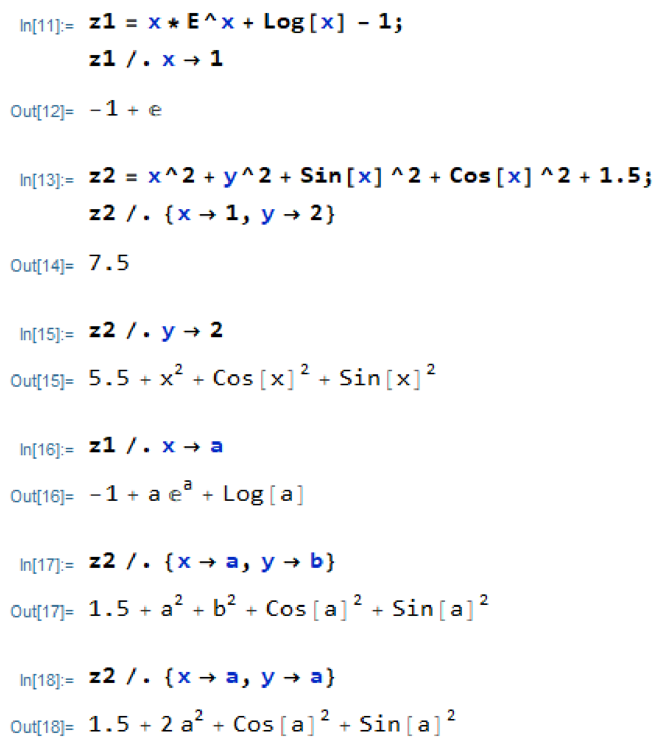
f(x,y,…) /. {x->a,y->b,…} – substitutes in the expression f(x,y,…) the values x=a, y=b,…

{f1(x),f2(x),…} /. x->a – substitutes in the expressions f1(x), f2(x),… the value x=a.

{f1(x,y,..),f2(x,y,…),…} /. {x->a,y->b,…} – substitutes in the expressions f1(x,y,…), f2(x,y,…),… the values x=a, y=b,…

f(x) /. x->{x0,x1,…} – tabulation of the function f(x).

An example of using substitutions is shown in Listing 2.



```
In[11]:= z1 = x * E^x + Log[x] - 1;
         z1 /. x → 1

Out[12]= -1 + e

In[13]:= z2 = x^2 + y^2 + Sin[x]^2 + Cos[x]^2 + 1.5;
         z2 /. {x → 1, y → 2}

Out[14]= 7.5

In[15]:= z2 /. y → 2

Out[15]= 5.5 + x^2 + Cos[x]^2 + Sin[x]^2

In[16]:= z1 /. x → a

Out[16]= -1 + a e^a + Log[a]

In[17]:= z2 /. {x → a, y → b}

Out[17]= 1.5 + a^2 + b^2 + Cos[a]^2 + Sin[a]^2

In[18]:= z2 /. {x → a, y → a}

Out[18]= 1.5 + 2 a^2 + Cos[a]^2 + Sin[a]^2
```

Listing 2

Convert expressions

Expression conversion functions:

Simplify[f] – simplifies the expression f;

FullSimplify[f] - simplifies the expression f, which contains special functions;

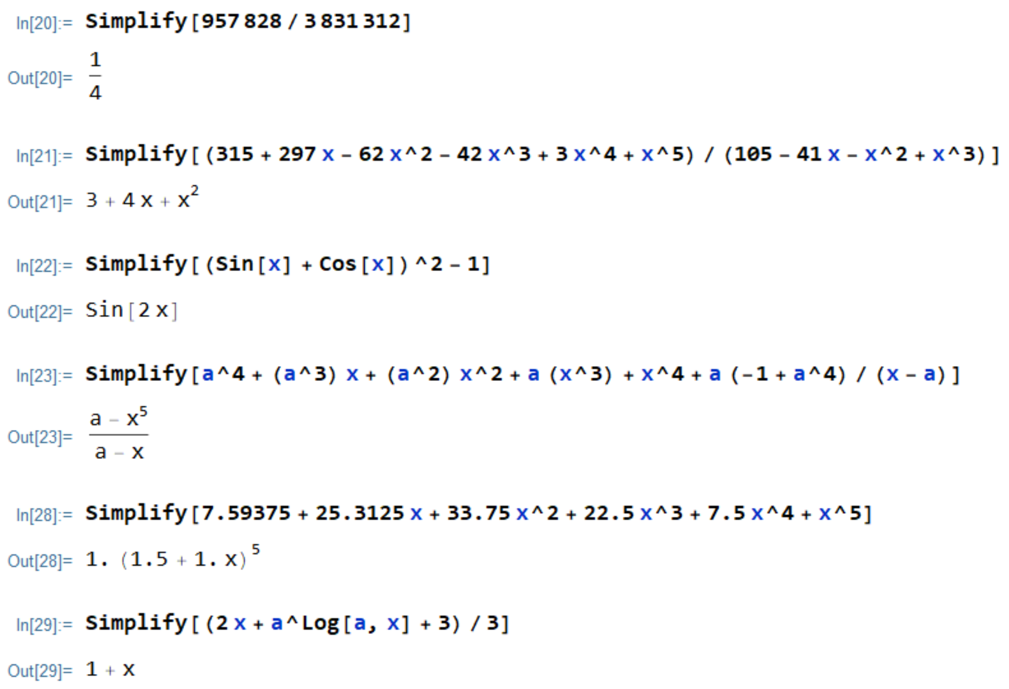
Expand[f] – expands out products and positive integer powers in f;

Collect[f,x] – collects together terms involving the same powers of expression f matching x;

TrigExpand[f] – expands out trigonometric functions;

Factor[f] – factors a polynomial over the integers.

Examples of using the Simplify function are shown in Listing 3, FullSimplify – in Listing 4.



```
In[20]:= Simplify[957828 / 3831312]

         1
Out[20]= -
         4

In[21]:= Simplify[(315 + 297 x - 62 x^2 - 42 x^3 + 3 x^4 + x^5) / (105 - 41 x - x^2 + x^3)]

Out[21]= 3 + 4 x + x^2

In[22]:= Simplify[(Sin[x] + Cos[x])^2 - 1]

Out[22]= Sin[2 x]

In[23]:= Simplify[a^4 + (a^3) x + (a^2) x^2 + a (x^3) + x^4 + a (-1 + a^4) / (x - a)]

         a - x^5
Out[23]= -------
          a - x

In[28]:= Simplify[7.59375 + 25.3125 x + 33.75 x^2 + 22.5 x^3 + 7.5 x^4 + x^5]

Out[28]= 1. (1.5 + 1. x)^5

In[29]:= Simplify[(2 x + a^Log[a, x] + 3) / 3]

Out[29]= 1 + x
```

Listing 3

```
In[30]:= Simplify[x * Sin[x] * Cos[y] + (x^2 + 1) / x * Sin[y] * Cos[x]]
```

Out[30]= $x \operatorname{Cos}[y] \operatorname{Sin}[x] + \frac{(1 + x^2) \operatorname{Cos}[x] \operatorname{Sin}[y]}{x}$

```
In[31]:= FullSimplify[x * Sin[x] * Cos[y] + (x^2 + 1) / x * Sin[y] * Cos[x]]
```

Out[31]= $\frac{\operatorname{Cos}[x] \operatorname{Sin}[y]}{x} + x \operatorname{Sin}[x + y]$

```
In[32]:= Simplify[(Sin[x] + Cos[x])^2]
```

Out[32]= $(\operatorname{Cos}[x] + \operatorname{Sin}[x])^2$

```
In[33]:= FullSimplify[(Sin[x] + Cos[x])^2]

Out[33]= 1 + Sin[2 x]

In[34]:= FullSimplify[(1 - Cos[2 x]) / 2 + Cos[x]^2 + a]

Out[34]= 1 + a
```

Listing 4

Function Expand

Modifications of the function Expand:

Expand[f] – expands out products and positive integer powers in f;

ExpandAll[f] – expands out all products and integer powers in any part of the expression f;

ExpandNumerator[f] - expands out products and powers that appear in the numerator of the expression f;

ExpandDenominator[f] - expands out products and powers that appear as denominators in the expression f;

PowerExpand[f] - expands all powers of products and powers of function f;

ComplexExpand[f] – expands the expression f assuming that all variables are real;

ComplexExpand[f,{x1,x2,…}] - expands the expression f assuming that variables matching any of the x are complex ;

FunctionExpand[f] – tries to expand out special and certain other functions in the expression f, when possible reducing compound arguments to simpler ones.

Listing 5 shows examples of using modifications of the function Expand[].

```
In[35]:=
Expand[(a + b) (a - b) (a + c) / (a + 1) ^3]
```

Out[35]= $$\frac{a^3}{(1+a)^3}-\frac{a\,b^2}{(1+a)^3}+\frac{a^2\,c}{(1+a)^3}-\frac{b^2\,c}{(1+a)^3}$$

```
In[36]:= ExpandAll[(a + b) (a - b) (a + c) / (a + 1) ^3]
```

Out[36]= $$\frac{a^3}{1+3\,a+3\,a^2+a^3}-\frac{a\,b^2}{1+3\,a+3\,a^2+a^3}+\frac{a^2\,c}{1+3\,a+3\,a^2+a^3}-\frac{b^2\,c}{1+3\,a+3\,a^2+a^3}$$

```
In[37]:= ExpandAll[Sqrt[x + 2^3] / ((a + b) (a - b)) (a - b) + (c + 2 I) ^2]
```

Out[37]= $$-4+4\,i\,c+c^2+\frac{\sqrt{8+x}}{a+b}$$

```
In[38]:= ExpandNumerator[(x^2 - 1) ^5 / (x + 1) ^3]
```

Out[38]= $$\frac{-1+5\,x^2-10\,x^4+10\,x^6-5\,x^8+x^{10}}{(1+x)^3}$$

```
In[39]:= ExpandDenominator[(x^2 - 1) ^5 / (x + 1) ^3]
```

Out[39]= $$\frac{\left(-1+x^2\right)^5}{1+3\,x+3\,x^2+x^3}$$

Listing 5

Function Collect

Modifications of the function Collect:

Collect[f,x] – collects together terms involving the same powers of expression f matching x;

Collect[f,{x1,x2,…}] – collects together terms that involve the same powers of of expression f matching x1,x2...

Listing 6 shows examples of using modifications of the function Collect[].

```
In[40]:= Collect[(x + a)^3 - (x - a)^2, x]
```

Out[40]= $-a^2 + a^3 + \left(2\,a + 3\,a^2\right) x + (-1 + 3\,a)\, x^2 + x^3$

```
In[41]:= Collect[(x + 1)^5 - EulerE[5, x] - (x + a)^3, x]
```

Out[41]= $\frac{3}{2} - a^3 + \left(5 - 3\,a^2\right) x + \left(\frac{15}{2} - 3\,a\right) x^2 + 9\,x^3 + \frac{15\,x^4}{2}$

```
In[42]:= Collect[(x - y) (x + y)^2 Sqrt[(x + y)^3], x]
```

Out[42]= $x^3 \sqrt{(x + y)^3} + x^2\, y \sqrt{(x + y)^3} - x\, y^2 \sqrt{(x + y)^3} - y^3 \sqrt{(x + y)^3}$

```
In[43]:= Collect[(x + y)^2 + (x - 1)^3 - (y - 1)^3, {x, y}]
```

Out[43]= $-2\,x^2 + x^3 - 3\,y + 4\,y^2 - y^3 + x\,(3 + 2\,y)$

```
In[44]:= Collect[1 / (x + y) + 1 / (x - y) + x - y, {x, y}]
```

Out[44]= $x + \frac{1}{x - y} - y + \frac{1}{x + y}$

Listing 6

Function Factor

Modifications of the function Factor:

Factor[f] – factors the function f over the integers;

FactorList[f] – gives a list of the factors of the function f, together with their exponents;

FactorTerms[f] – pulls out any overall numerical factor in the expression f;

FactorTermsList[f] – gives a list in which the first element is the overall numerical factor in the expression f, and the second element is the polynomial with the overall factor removed;

FactorInteger[f] – gives a list of the prime factors of the integer f, together with their exponents.

Listing 7 shows examples of using modifications of the function Factor[].

```
In[45]:= Factor[18 x^3 + 45 x^2 + x - 14]

Out[45]= (-1 + 2 x) (2 + 3 x) (7 + 3 x)

In[46]:= Factor[x^3 + x^2 (3 + a - b) + x (3 a - 3 b - a * b) - 3 a * b]

Out[46]= - (b - x) (3 + x) (a + x)

In[47]:= Factor[a^2 + a - 1]

Out[47]= -1 + a + a^2

In[48]:= Factor[6 x^3 + x^2 (13 I - 6) - 6 x (1 + 13 / 6 I) + 6]

Out[48]= (-1 + x) (3 i + 2 x) (2 i + 3 x)

In[51]:= Factor[1 / 16 (Cos[5 x] + Cos[3 x] + 10 Cos[x]), Trig → True]

         1
Out[51]= - Cos[x] (5 + Cos[4 x])
         8

In[52]:= FactorList[x^5 + x^4 - 6 x^3 - x^2 - x + 6]

Out[52]= {{1, 1}, {-2 + x, 1}, {-1 + x, 1}, {3 + x, 1}, {1 + x + x^2, 1}}

In[53]:= FactorTerms[3 x^3 - 3 x^2 - 51 x - 45]

Out[53]= 3 (-15 - 17 x - x^2 + x^3)

In[54]:= FactorTermsList[3 x^3 - 3 x^2 - 51 x - 45]

Out[54]= {3, -15 - 17 x - x^2 + x^3}

In[55]:= FactorInteger[1235]

Out[55]= {{5, 1}, {13, 1}, {19, 1}}
```

Listing 7

Functions of transformation of trigonometric expressions
TrigReduce[f] – rewrites products and powers of trigonometric functions in the expression f in terms of trigonometric functions with combined arguments.;
TrigExpand[f] – expands out trigonometric functions in the expression f;
TrigFactor[f] – factors trigonometric functions in the expression f;
TrigToExp[f] – converts trigonometric functions to exponentials.;
ExpToTrig [f] – converts exponentials in expression to trigonometric functions.
Listing 8 shows examples of using trigonometric expression transformation functions.

```
In[56]:= TrigReduce[(Sin[x] + Cos[x])^2]

Out[56]= 1 + Sin[2 x]

In[57]:= TrigReduce[2 Sin[1 / 2 (x + y)] Sin[1 / 2 (y - x)]]

Out[57]= Cos[x] - Cos[y]

In[58]:= TrigExpand[Sin[4 x]]
```

Out[58]= $4\,\mathrm{Cos}[x]^3\,\mathrm{Sin}[x] - 4\,\mathrm{Cos}[x]\,\mathrm{Sin}[x]^3$

```
In[62]:= TrigFactor[Cos[x] - Cos[y]]
```

Out[62]= $-2\,\mathrm{Sin}\left[\frac{x}{2} - \frac{y}{2}\right]\,\mathrm{Sin}\left[\frac{x}{2} + \frac{y}{2}\right]$

```
In[63]:= TrigToExp[Cos[x] + Sin[x]]
```

Out[63]= $\left(\frac{1}{2} + \frac{i}{2}\right) e^{-i x} + \left(\frac{1}{2} - \frac{i}{2}\right) e^{i x}$

```
In[64]:= TrigToExp[1 / I * Sin[I * x]]
```

Out[64]= $-\frac{e^{-x}}{2} + \frac{e^{x}}{2}$

```
In[65]:= ExpToTrig[E^x + E^(-x)]

Out[65]= 2 Cosh[x]
```

Listing 9

Lecture № 3. Tables, styles, and graphics

Representation of vectors and matrices in tabular form is possible using the functions TableForm and MatrixForm, is as follows:

TableForm[f]

Out[n] // MatrixForm

where f – name of the vector or matrix;

n - the number of the line in which the vector or matrix is located;

% - is used if the representation function % / / MatrixForm of the vector or matrix in tabular form is located after the vector (matrix).

You can also create a vector or matrix using a function List:

List [a, b, c,...] — creates vector {a,b,c,...};

List [{a, b, c,..}, {d, e, f, ..}, {g, h, k, ..}] – creates matrix {{a, b, c,..}, {d, e, f, ..}, {g, h, k, ..}}.

Listing 1 in tabular form shows the vector f3 and the matrices f4, f5, from Listing 1, Lecture 1, formed using the functions TableForm and MatrixForm.

```
TableForm[f3]
2 + 3 i
1- 2 i
5
3 +7i
7
TableForm[f4]
1       2       3
4       7        0
-5      1        8
TableForm[f5]
Sin[X]     e^-x      (-1+x)/(1+x)   Log[x]         5
1 + 2 i     3         5             Tan[1+x]        -8
4           a         B             Cos[x]          2-i
Out [29]//MatrixForm
(   rSin[x]   e^-x    (-1+x)/(1+x)  Log[x]        5    )
(   1+2 i     3       5             Tan[ 1 + x]   -8   )
(   4         a       B             Cos[x]        2- i )
```

Listing 1

Creation of vectors and matrices using the Range function

The Range function is used to create numerical lists and has the following modifications:

Range [nmax] - generates the list {1, 2, ..., nmax };

Range [nmin, nmax] - generates the list {nmin, nmax};

Range [nmin, nmax, dn] - generates the list from nmin to nmax uses step dn.

Examples of function realization are shown in Listing 2.

```
Range [7]
{1,2,3,4,5,6,7}
Range[4,10]
{4,5,6,7,8,9,10}


Range[3,8,0.5]
{3,3.5,4.,4,5,5.,5.5,6.,6.5,7.,7.5,8.}
```

Listing 2

Creation of vectors and matrices using Table functions

To create vectors and matrices, you can use the Table function, which has the form:

Table [f, { nmax }] - generates a list of nmax copies of f;

Table [f,{1, nmax}] - generates a list of the values of f from 1 to nmax;

Table [f, {n, nmin, nmax}] — generates a list of the values of f starts with n=nmin to nmax;

Table [f, {n, nmin, nmax, dn}]— generates a list of the values of f starts with n=nmin to nmax uses steps dn.

An example of using the Table function is shown in Listing 3.

```
Table [Log [x] ,{5}]
{Log[x] , Log[x] , Log[x] , Log[x] , Log[x] }
Table[Log[x] ,{x,5}]
{0,Log[2],Log[3],Log[4] ,Log[5]}
Table[Log[x] ,{x,5,10}]
{Log[5] ,Log[6] ,Log[7] ,Log[8] ,Log[9] ,Log[10] }
Table[Log[x] ,{x,l ,3 ,0 .5}]
{0,0.405465,0. 693 147,0.9162 91,1.09861}
Table [i+j ,{1,2 ,4} ,{j ,2,4}]
{{4,5, 6} ,{5, 6,7}, {6,7,8}}
```

Listing 3

Selection elements of vector and matrix

- The following methods of selecting of elements of vectors and matrices are implemented in the Mathematica system:
- use of double square brackets;
- use of the Part function;
- use the Select function.

Use of double square brackets

In this case, the expression that separates the elements of the vector or matrix is represented as:

f [[n]] f [[n_1, n_2, ...]],

where f - name of the vector or matrix;

n - the selected element;

n_i - i-th element from the set of selected elements.

Examples of element selection are shown in Listing 4

```
f1={2 ,1,4,3,5}
{2,1,4,3,5}
f2={ {1,2,3}, {3,5,7}, {2,4,6}}
{{ 1,2,3} ,{3,5,7}, {2,4,6}}
f3={a,1,b,2,3,c}
{a,1,b,2,3,c}
f4={2,a,Sin[x+y^2] ,b,1}
{2, a, Sin[x + y^2] , b, 1}
f1[[4]]
3
f2[[2,2]]
5
f1 [[{2,3}]]
{1,4}
{f2[[1,2]] ,f2[[3,1]]}
{2,2}
```

Listing 4

Output of elements of vectors and matrices is carried out by means of functions MatrixForm and TableForm.

Examples of the use of these output forms are shown in Listing 5. The use of the TableAlignments and TableSpacing options to place the vector and matrix on the screen in the desired form is shown in Listing 6.

```
F={{a,I,4},{b,2,5} ,{c,3,б}}
{{a, 1,4} ,{b,2,5}, {c,3,5}}
MatrixForm[F]
```

$$\begin{pmatrix} a & 1 & 4 \\ b & 2 & 5 \\ c & 3 & 6 \end{pmatrix}$$

```
TableForm[F]
a        1            4
b         2           5
c        3            6
```

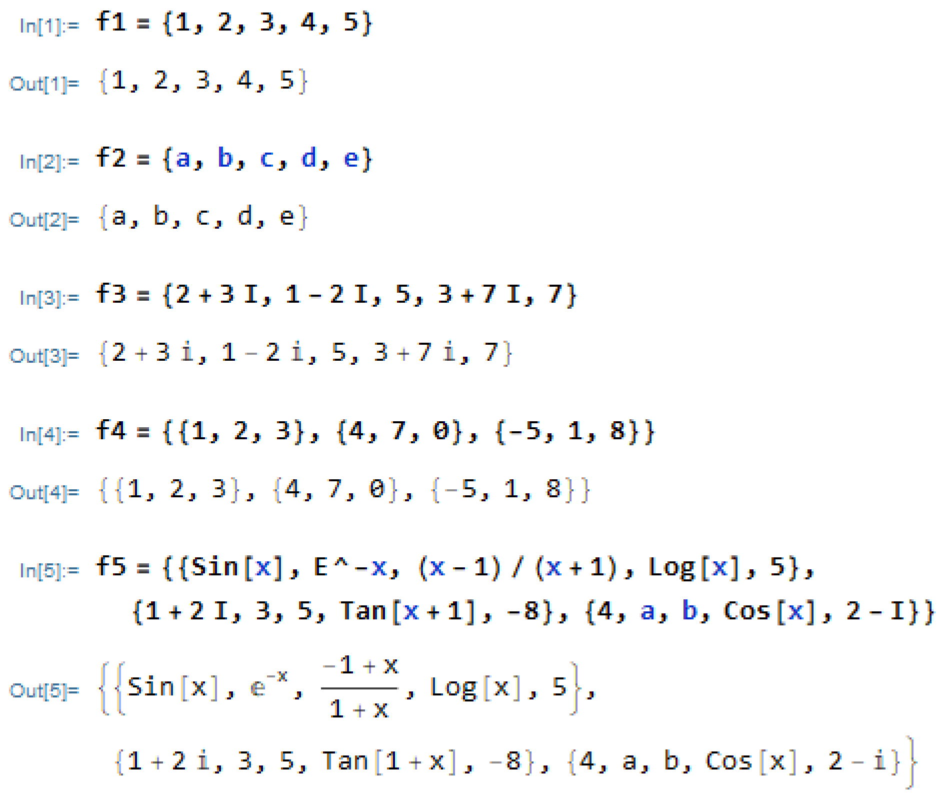
Listing 5

```
s={ 5,73426813438765,34}
{5,73426813438755,34}
TableForm[s, TableAlignments->Left]
5
73426S13438765
34
TableForm[s, TableAlignments->Center]
              5
      73426813438765
             34
s1={{1,3,4},{5,1,1} ,{3,2,1}}
{{1,3,4},{5,1,7},{3,2,1}}
TableForm[s1]
1       3       4
5       1       7
3       2       1
TableForm[s1, TableSpacing->{ 1,1}]
```

```
1   3   4
5   1   7
3   2   1
```
TableForm[s1, TableSpacing->{ 5,2}]
```
1     3     4


5     1     7


3     2     1
```
TableForm[s1, TableSpacing->{ 2,5}]
```
1           3           4
5           1           7
3           2           1
```
Listing 6

Lecture № 4. Operation with vectors and matrices

Vectors and matrices in the Mathematica system are lists. A list is a collection of data bounded by curly brackets. The vector is one-dimensional, the matrix is a two-dimensional list. Elements of vectors and matrices can be real and imaginary numbers, functions, mathematical expressions. Listing 1 shows the vectors and matrices of the various elements.



```
In[1]:= f1 = {1, 2, 3, 4, 5}

Out[1]= {1, 2, 3, 4, 5}

In[2]:= f2 = {a, b, c, d, e}

Out[2]= {a, b, c, d, e}

In[3]:= f3 = {2 + 3 I, 1 - 2 I, 5, 3 + 7 I, 7}

Out[3]= {2 + 3 i, 1 - 2 i, 5, 3 + 7 i, 7}

In[4]:= f4 = {{1, 2, 3}, {4, 7, 0}, {-5, 1, 8}}

Out[4]= {{1, 2, 3}, {4, 7, 0}, {-5, 1, 8}}

In[5]:= f5 = {{Sin[x], E^-x, (x - 1) / (x + 1), Log[x], 5},
           {1 + 2 I, 3, 5, Tan[x + 1], -8}, {4, a, b, Cos[x], 2 - I}}

                         -1 + x
Out[5]= {{Sin[x], e^-x, ------, Log[x], 5},
                         1 + x

          {1 + 2 i, 3, 5, Tan[1 + x], -8}, {4, a, b, Cos[x], 2 - i}}
```

Listing 1

Vector and matrix creation is done with the following functions.

Array [f, n]- generates a list of length n , with elements f [1], f [2],…, f [n];

Array [f, n1, n2]- generates a list of length n1, start with element f [n2], and n2 can be a number, function, expression;

Array [f, { n1, n2}] generates an n1 x n2 array of nested lists, with elementsf (n1, n2);

Array [f, n1, n2, h] - generates a list of length n1, start with element f (n2), uses head h.

Examples of creating vectors using the Array function are shown in Listing 2. Note that these functions allow you to summarize and multiply vector elements using Plus and Times functions.

```
In[21]:= Array[Tan, 5]

Out[21]= {Tan[1], Tan[2], Tan[3], Tan[4], Tan[5]}

In[22]:= Array[Sin, 5, 3]

Out[22]= {Sin[3], Sin[4], Sin[5], Sin[6], Sin[7]}

In[23]:= Array[Sin, 5, Sqrt[3]]

Out[23]= {Sin[√3], Sin[1+√3], Sin[2+√3], Sin[3+√3], Sin[4+√3]}

In[24]:= N[%]

Out[24]= {0.987027, 0.398189, -0.556742, -0.999807, -0.523654}

In[25]:= Array[Sin, 5, Sqrt[3], Plus]

Out[25]= Sin[√3] + Sin[1+√3] + Sin[2+√3] + Sin[3+√3] + Sin[4+√3]

In[26]:= N[%]

Out[26]= -0.694987

In[27]:= Array[Sin, 5, Sqrt[3], Times]

Out[27]= Sin[√3] Sin[1+√3] Sin[2+√3] Sin[3+√3] Sin[4+√3]

In[28]:= N[%]

Out[28]= -0.11456
```

Listing 2

Determining the structure of a vector or matrix

The following functions are used to determine the structure of a vector or matrix:
VectorQ [V] - gives True if expression V is a vector, and gives False otherwise;
MatrixQ [M] - gives True if expression M is a matrix, and gives False otherwise;
Length [V] - gives the number of elements in a vector V;
Length [M] - gives the number of elements in a natrix M;
MemberQ [V, n] - returns True if an element of V matches n, and False otherwise;
FreeQ [V, n] - yields True if no subexpression in V matches n, and yields False otherwise;
FreeQ [M, n] - yields True if no subexpression in M matches n, and yields False otherwise;
Dimensions [V] - gives a list of the dimensions of V;
Dimensions [M] - gives a list of the dimensions of M (the number of rows and the number of columns);
Position [V, n] - gives a list of the positions at which objects matching n appear in vector V;
Count [V, n] - gives the number of elements in V, that match n;

TensorRank [V] - gives the rank of vector V, if V is a tensor;

TensorRank [M] gives the rank of matrix M, if M is a tensor.

Transformation and creation of vectors and matrices

The Mathematica system has rich features for converting and creating new vectors and matrices (and lists of any level). The following functions can be used for this purpose:

Drop [V, n] - gives V with its first n elements dropped;

Drop [V,-n] - gives V with its last n elements dropped;

Drop [V, {n}] – gives V with its n-th element dropped.;

Drop [V, {m, n}]- gives V with elements m through n dropped;

Last [f] - gives the last element in vector (matrix) f;

Rest [V] - gives vector V with the first element removed;

Take [V, n]- gives the first n elements of vector V;

Take [V,-n]- gives the last n elements of vector V;

Take [V, {m, n}] - gives elements m through n of vector V;

Append [V, a] - gives vector V with a appended;

Prepend [V,-a] - gives vector V with a prepended ;

Insert [V, a, n] - inserts a at position n in V, the position is counted from the start;

Insert [V, a,-n] - inserts a at position n in V, the position is counted from the end;

Delete [V, n] - deletes the element at position n in vector V;

{Delete [f, n1], Delete [f, n2], Delete [f, n3], ...} - deletes the elements at positions ni in vector (matrix) and creates new vector (matrix).

Examples of using functions are shown in Listing 3.

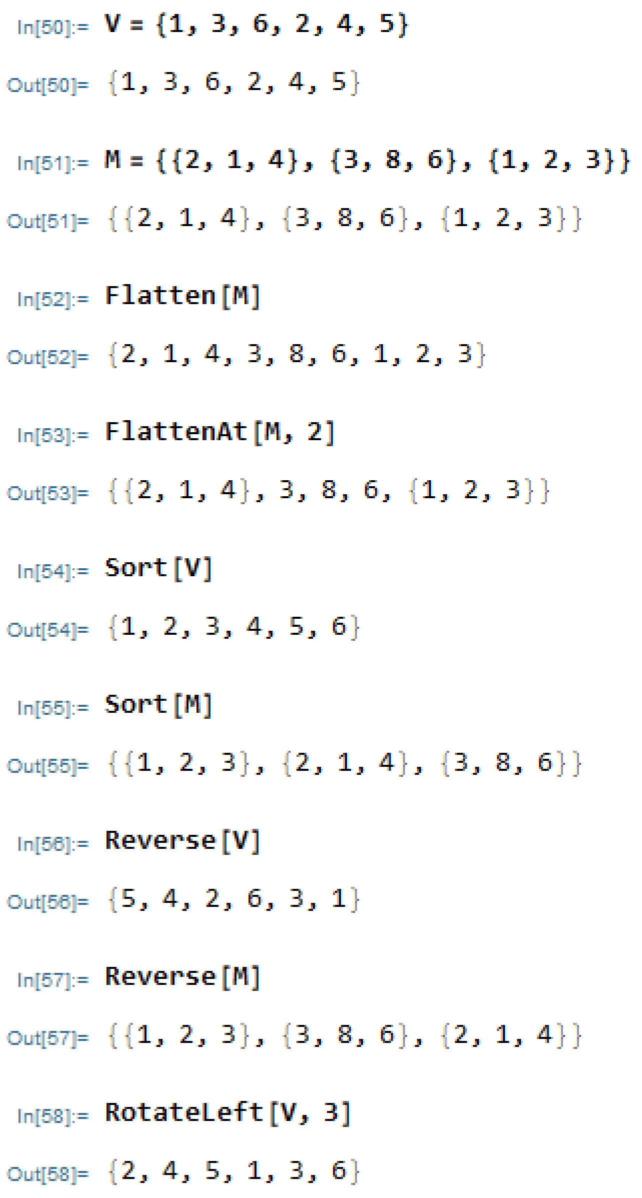
```
In[35]:= V = {1, 3, 4, 5, 7, 2, 4}

Out[35]= {1, 3, 4, 5, 7, 2, 4}

In[36]:= M = {{1, 3, 5}, {2, 4, 6}, {1, 2, 3}}

Out[36]= {{1, 3, 5}, {2, 4, 6}, {1, 2, 3}}

In[37]:= {Drop[V, 3], Drop[V, -4], Drop[V, {5}], Drop[V, {2, 5}]}

Out[37]= {{5, 7, 2, 4}, {1, 3, 4}, {1, 3, 4, 5, 2, 4}, {1, 2, 4}}

In[38]:= Delete[V, 6]

Out[38]= {1, 3, 4, 5, 7, 4}

In[39]:= {Delete[V, 1], Delete[V, -3], Delete[V, 7]}

Out[39]= {{3, 4, 5, 7, 2, 4}, {1, 3, 4, 5, 2, 4}, {1, 3, 4, 5, 7, 2}}

In[40]:= Delete[M, {2, 3}]

Out[40]= {{1, 3, 5}, {2, 4}, {1, 2, 3}}

In[41]:= Delete[M, {3, 1}]

Out[41]= {{1, 3, 5}, {2, 4, 6}, {2, 3}}

In[42]:= Last[V]

Out[42]= 4

In[43]:= Last[M]

Out[43]= {1, 2, 3}
In[44]:= Rest[V]

Out[44]= {3, 4, 5, 7, 2, 4}

In[45]:= {Take[V, 4], Take[V, -4], Take[V, {3, 6}]}

Out[45]= {{1, 3, 4, 5}, {5, 7, 2, 4}, {4, 5, 7, 2}}

In[46]:= Append[V, Sin[x]]

Out[46]= {1, 3, 4, 5, 7, 2, 4, Sin[x]}

In[47]:= Prepend[V, Sin[x]]

Out[47]= {Sin[x], 1, 3, 4, 5, 7, 2, 4}

In[48]:= Insert[V, a, 5]

Out[48]= {1, 3, 4, 5, a, 7, 2, 4}

In[49]:= Insert[V, {3, a, Sin[x]}, 5]

Out[49]= {1, 3, 4, 5, {3, a, Sin[x]}, 7, 2, 4}
```

Listing 3

The creation of new vectors and matrices is also possible by changing the location of the vector or matrix, which uses the following functions.

Flatten [M] - flattens out nested lists;
Flatten [M, n] - flattens to level n matrix M;
Sort [f] - sorts the elements of vector (matrix) f into canonical order;
Reverse [f] - reverses the order of the elements in vector (matrix) f;
RotateLeft [f] - cycles the elements in vector (matrix) f one position to the left;
RotateLeft [f, n] - cycles the elements in vector (matrix) f n positions to the left;
RotateRight [f] – cycles the elements in vector (matrix) f one position to the right;
RotateRight [f, n] - cycles the elements in vector (matrix) f n position to the right;
Transpose [M] - transposes the first two levels in matrix M.

Examples of using functions are shown in Listing 4.



```
In[50]:= V = {1, 3, 6, 2, 4, 5}

Out[50]= {1, 3, 6, 2, 4, 5}

In[51]:= M = {{2, 1, 4}, {3, 8, 6}, {1, 2, 3}}

Out[51]= {{2, 1, 4}, {3, 8, 6}, {1, 2, 3}}

In[52]:= Flatten[M]

Out[52]= {2, 1, 4, 3, 8, 6, 1, 2, 3}

In[53]:= FlattenAt[M, 2]

Out[53]= {{2, 1, 4}, 3, 8, 6, {1, 2, 3}}

In[54]:= Sort[V]

Out[54]= {1, 2, 3, 4, 5, 6}

In[55]:= Sort[M]

Out[55]= {{1, 2, 3}, {2, 1, 4}, {3, 8, 6}}

In[56]:= Reverse[V]

Out[56]= {5, 4, 2, 6, 3, 1}

In[57]:= Reverse[M]

Out[57]= {{1, 2, 3}, {3, 8, 6}, {2, 1, 4}}

In[58]:= RotateLeft[V, 3]

Out[58]= {2, 4, 5, 1, 3, 6}
```

Listing 4

Representation of vectors and matrices in tabular form is possible using the functions TableForm and MatrixForm, what look like:

TableForm[f]

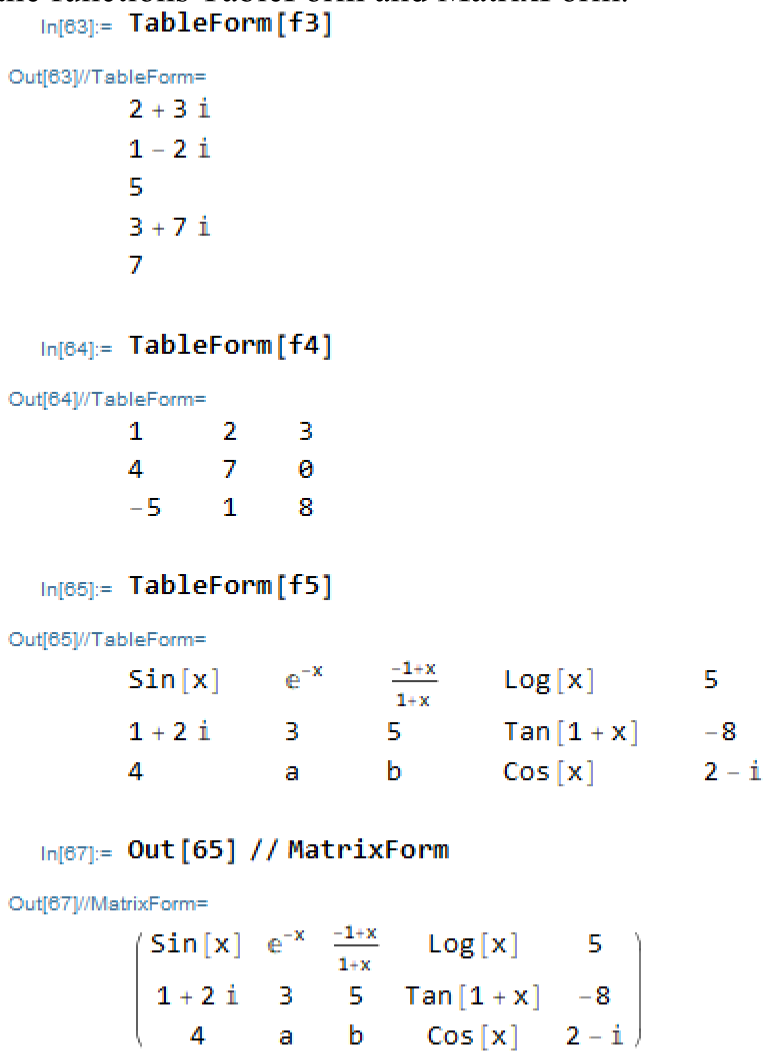
Out[n]   // MatrixForm
where f – name of the vector or matrix;
n - the number of the line in which the vector or matrix is located;
% - is used if the representation function % / / MatrixForm of the vector or matrix in tabular form is located after the vector (matrix).

You can also create a vector or matrix using a function List:

List [a, b, c,...] — creates vector {a,b,c,...};

List [{a, b, c,..}, {d, e, f, ..}, {g, h, k, ..}] – creates matrix {{a, b, c,..}, {d, e, f, ..}, {g, h, k, ..}}.

Listing 5 in tabular form shows the vector f3 and the matrices f4, f5, from Listing 1, formed using the functions TableForm and MatrixForm.



```
In[63]:= TableForm[f3]

Out[63]//TableForm=
         2 + 3 i
         1 - 2 i
         5
         3 + 7 i
         7

In[64]:= TableForm[f4]

Out[64]//TableForm=
         1      2     3
         4      7     0
         -5     1     8

In[65]:= TableForm[f5]

Out[65]//TableForm=
                         -1+x
         Sin[x]    e^-x  ----   Log[x]       5
                         1+x
         1 + 2 i   3     5      Tan[1 + x]   -8
         4         a     b      Cos[x]       2 - i

In[67]:= Out[65] // MatrixForm

Out[67]//MatrixForm=
                      -1+x
         ( Sin[x]  e^-x  ----   Log[x]       5   )
                      1+x
         ( 1 + 2 i  3     5     Tan[1 + x]   -8  )
         (   4      a     b      Cos[x]     2 - i)
```

Listing 5

Creation of vectors and matrices using the Range function

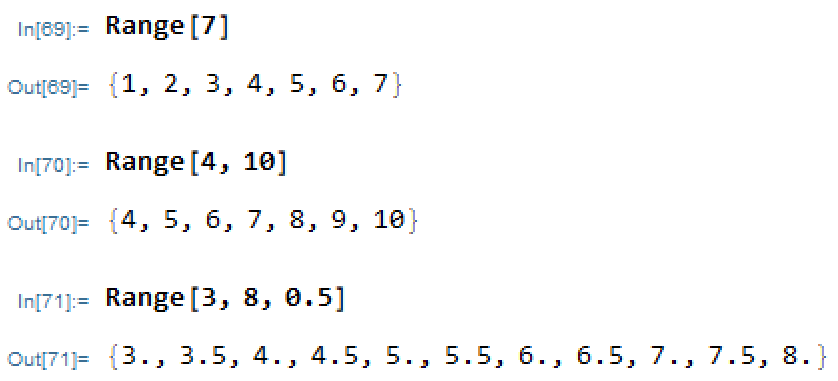
The Range function is used to create numerical lists and has the following modifications:

Range [n_{max}] - generates the list {1, 2, ..., n_{max} };

Range [n_{min}, n_{max}] - generates the list {n_{min}, n_{max}};

Range [n_{min}, n_{max}, dn] - generates the list from n_{min} to n_{max} uses step dn.

Examples of function realization are shown in Listing 6.



```
In[69]:= Range[7]

Out[69]= {1, 2, 3, 4, 5, 6, 7}

In[70]:= Range[4, 10]

Out[70]= {4, 5, 6, 7, 8, 9, 10}

In[71]:= Range[3, 8, 0.5]

Out[71]= {3., 3.5, 4., 4.5, 5., 5.5, 6., 6.5, 7., 7.5, 8.}
```

Listing 6

Creation of vectors and matrices using Table functions

To create vectors and matrices, you can use the Table function, which has the form:

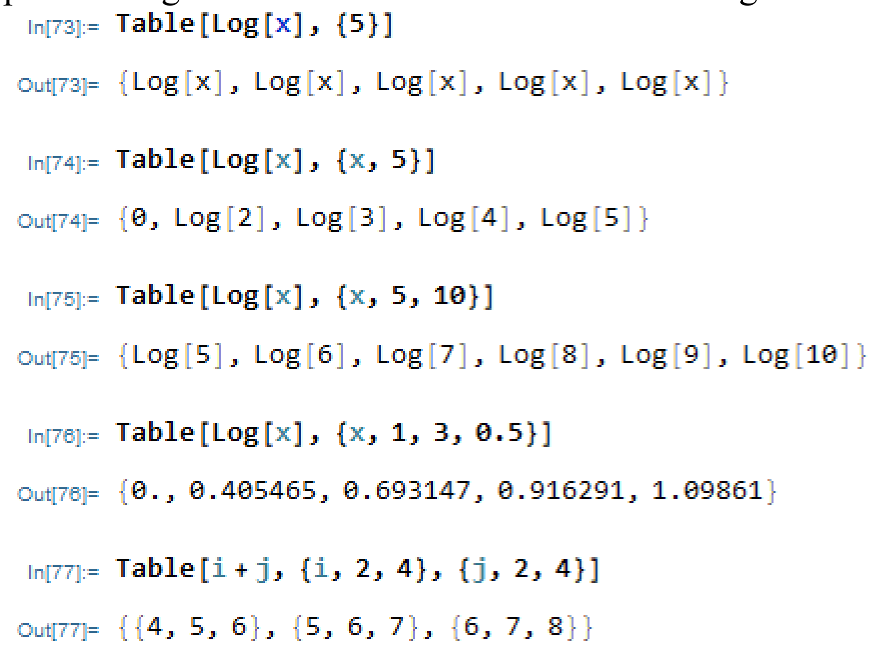
Table [f, { n_{max} }] - generates a list of n_{max} copies of f;

Table [f,{1, n_{max}}] - generates a list of the values of f from 1 to n_{max};

Table [f, {n, n_{min}, n_{max}}] — generates a list of the values of f starts with n=n_{min} to n_{max};

Table [f, {n, n_{min}, n_{max}, dn}]— generates a list of the values of f starts with n=n_{min} to n_{max} uses steps dn.

An example of using the Table function is shown in Listing 7.



```
In[73]:= Table[Log[x], {5}]

Out[73]= {Log[x], Log[x], Log[x], Log[x], Log[x]}

In[74]:= Table[Log[x], {x, 5}]

Out[74]= {0, Log[2], Log[3], Log[4], Log[5]}

In[75]:= Table[Log[x], {x, 5, 10}]

Out[75]= {Log[5], Log[6], Log[7], Log[8], Log[9], Log[10]}

In[76]:= Table[Log[x], {x, 1, 3, 0.5}]

Out[76]= {0., 0.405465, 0.693147, 0.916291, 1.09861}

In[77]:= Table[i + j, {i, 2, 4}, {j, 2, 4}]

Out[77]= {{4, 5, 6}, {5, 6, 7}, {6, 7, 8}}
```

Listing 7

Selection elements of vector and matrix

The following methods of selecting of elements of vectors and matrices are implemented in the Mathematica system:

- use of double square brackets;
- use of the Part function;
- use the Select function.

Use of double square brackets

In this case, the expression that separates the elements of the vector or matrix is represented as:

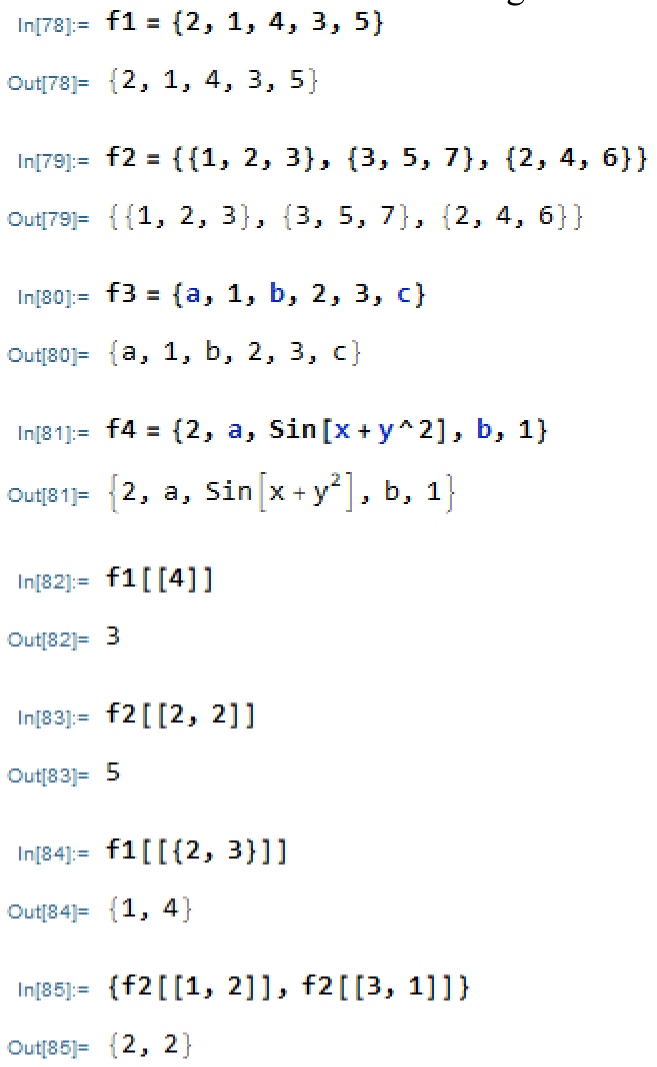
f [[n]] f [[n_1, n_2, ...]],

where f - name of the vector or matrix;

n - the selected element;

n_i - i-th element from the set of selected elements.

Examples of element selection are shown in Listing 8.



```
In[78]:= f1 = {2, 1, 4, 3, 5}

Out[78]= {2, 1, 4, 3, 5}

In[79]:= f2 = {{1, 2, 3}, {3, 5, 7}, {2, 4, 6}}

Out[79]= {{1, 2, 3}, {3, 5, 7}, {2, 4, 6}}

In[80]:= f3 = {a, 1, b, 2, 3, c}

Out[80]= {a, 1, b, 2, 3, c}

In[81]:= f4 = {2, a, Sin[x + y^2], b, 1}

Out[81]= {2, a, Sin[x + y^2], b, 1}

In[82]:= f1[[4]]

Out[82]= 3

In[83]:= f2[[2, 2]]

Out[83]= 5

In[84]:= f1[[{2, 3}]]

Out[84]= {1, 4}

In[85]:= {f2[[1, 2]], f2[[3, 1]]}

Out[85]= {2, 2}
```

Listing 8

Select elements of the vector and matrix using the Part function

The Part function is represented as follows:

{Part[f, n_1], Part[f, n_2],...},

where f – name of vector;

n_i - i-th element of vector f.

If you select elements of the matrix using the Part function, this function is represented as follow:

{Part[f, n_1, m_1], Part[f, n_2 , m_2],...},

where n_i - i-th row element of matrix;

m_i - i-th column element of matrix.

In the case of selecting elements from complex elements of a vector or matrix, the Part function is represented as follows:

Part[f, n, m, 1],

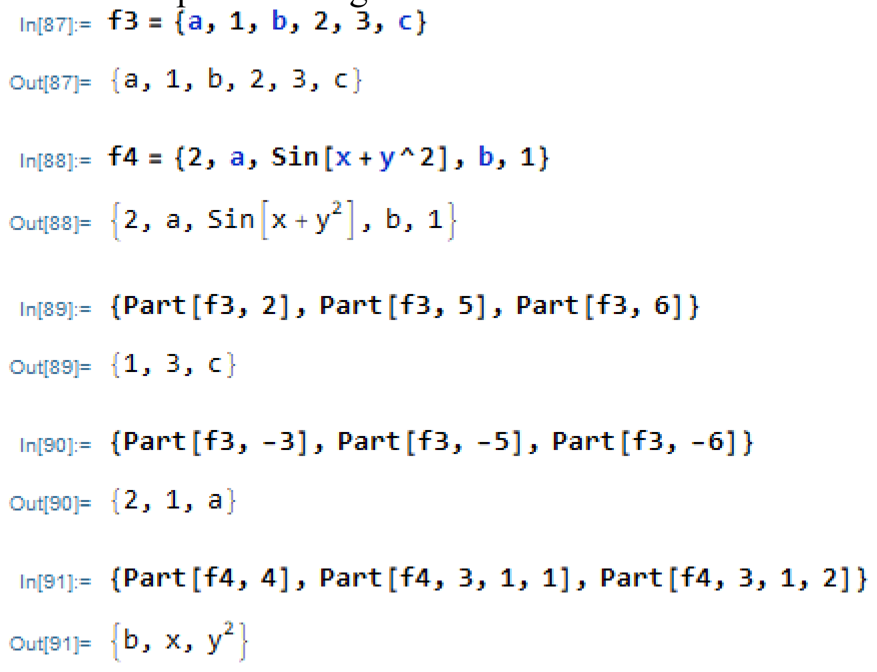
where f - name of vector or matrix;

n - number element of vector f;

m - expression level (m = l in the case of a vector, m = 2 in the case of a matrix);

1 - element number in a vector or matrix.

Examples of using the Part function are shown in Listing 9.

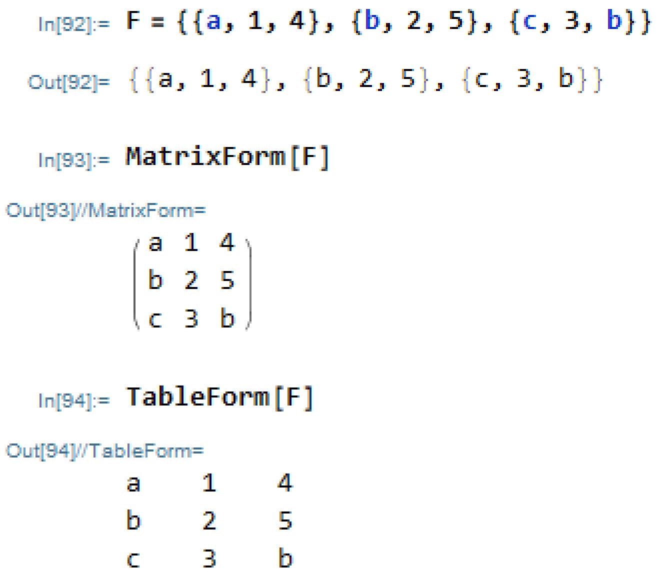


```
In[87]:= f3 = {a, 1, b, 2, 3, c}

Out[87]= {a, 1, b, 2, 3, c}

In[88]:= f4 = {2, a, Sin[x + y^2], b, 1}

Out[88]= {2, a, Sin[x + y^2], b, 1}

In[89]:= {Part[f3, 2], Part[f3, 5], Part[f3, 6]}

Out[89]= {1, 3, c}

In[90]:= {Part[f3, -3], Part[f3, -5], Part[f3, -6]}

Out[90]= {2, 1, a}

In[91]:= {Part[f4, 4], Part[f4, 3, 1, 1], Part[f4, 3, 1, 2]}

Out[91]= {b, x, y^2}
```

Listing 9

Output of elements of vectors and matrices is carried out by means of functions MatrixForm and TableForm.

Examples of the use of these output forms are shown in Listing 10. The use of the TableAlignments and TableSpacing options to place the vector and matrix on the screen in the desired form is shown in Listing 11.



```
In[92]:= F = {{a, 1, 4}, {b, 2, 5}, {c, 3, b}}

Out[92]= {{a, 1, 4}, {b, 2, 5}, {c, 3, b}}

In[93]:= MatrixForm[F]

Out[93]//MatrixForm=
  ( a  1  4 )
  ( b  2  5 )
  ( c  3  b )

In[94]:= TableForm[F]

Out[94]//TableForm=
  a    1    4
  b    2    5
  c    3    b
```

Listing 10

```
In[96]:= s = {5, 73426813438765, 34}

Out[96]= {5, 73426813438765, 34}

In[97]:= TableForm[s, TableAlignments → Left]

Out[97]//TableForm=
          5
          73426813438765
          34

In[99]:= TableForm[s, TableAlignments → Center]


Out[99]//TableForm=
                 5
          73426813438765
                34

In[101]:= s1 = {{1, 3, 4}, {5, 1, 1}, {3, 2, 1}}

Out[101]= {{1, 3, 4}, {5, 1, 1}, {3, 2, 1}}

In[102]:= TableForm[s1]

Out[102]//TableForm=
          1    3    4
          5    1    1
          3    2    1

In[103]:= TableForm[s1, TableSpacing → {1, 1}]

Out[103]//TableForm=
          1 3 4
          5 1 1
          3 2 1

In[105]:= TableForm[s1, TableSpacing → {5, 2}]

Out[105]//TableForm=
          1   3   4


          5   1   1


          3   2   1

In[106]:= TableForm[s1, TableSpacing → {2, 5}]

Out[106]//TableForm=
          1       3       4

          5       1       1

          3       2       1
```

Listing 11

Combining vectors and matrices

The combination of vectors and matrices is carried out using the following functions:

Union [F] - gives a sorted version of a list F, in which all duplicated elements have been dropped;

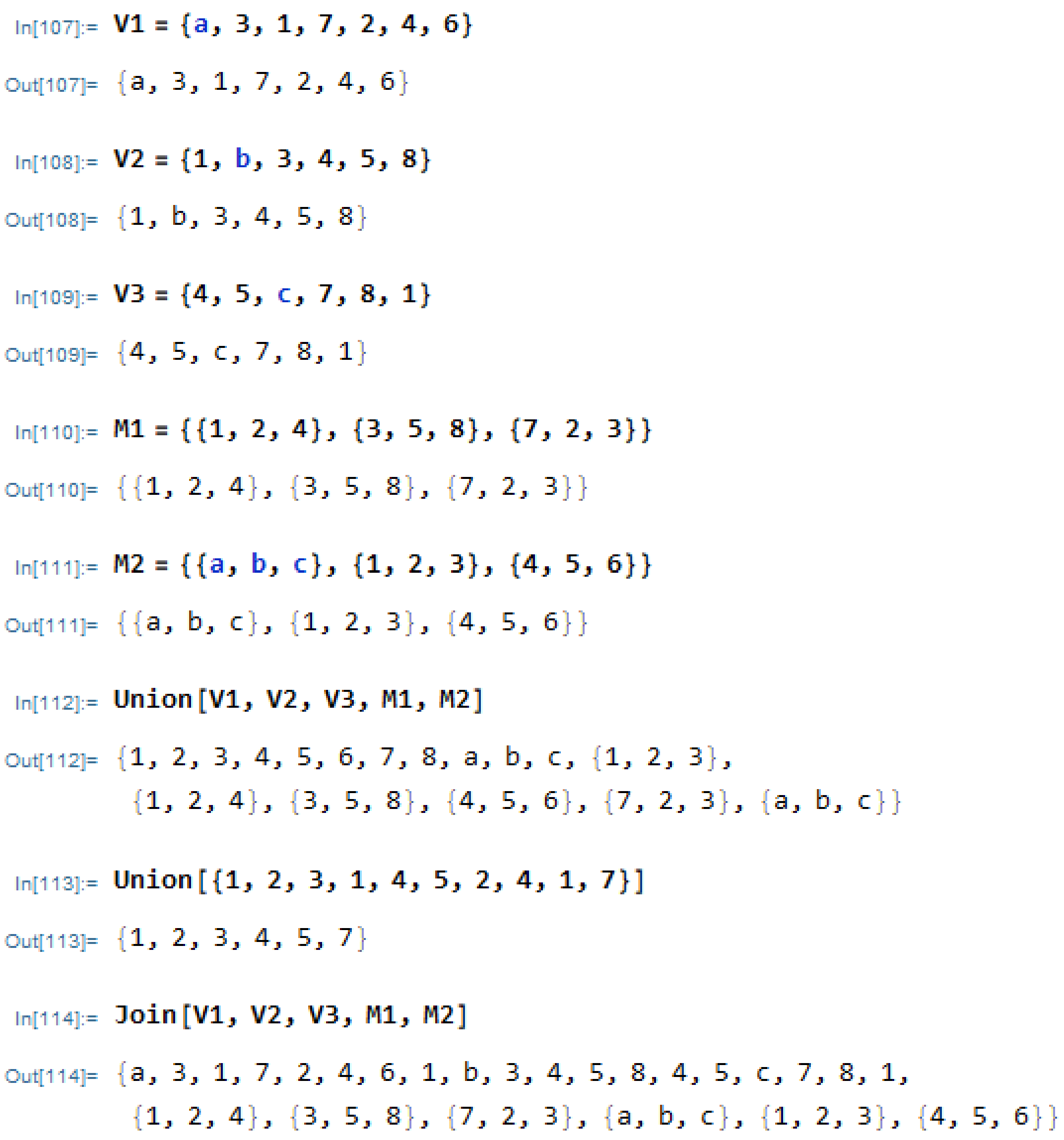
Union [f_1, f_2, ...] - combines f_1, f_2 removing repeating elements of vectors and matrices;

Join [f_1, f_2...] - combines f_1, f_2 ... in a single chain (concatenation);

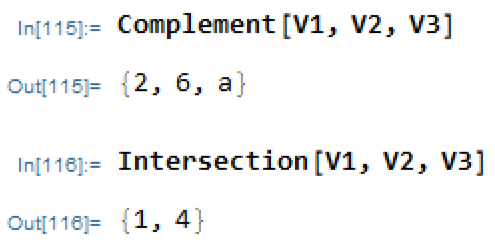
Complement [f_1, f_2, ...] - gives the elements in f that are not in any of the
f_1, f_2, ...;

Intersection [f_1, f_2, ...] - gives a sorted list of the elements common to all the lists.

Listing 12 shows examples of using functions.



```
In[107]:= V1 = {a, 3, 1, 7, 2, 4, 6}

Out[107]= {a, 3, 1, 7, 2, 4, 6}

In[108]:= V2 = {1, b, 3, 4, 5, 8}

Out[108]= {1, b, 3, 4, 5, 8}

In[109]:= V3 = {4, 5, c, 7, 8, 1}

Out[109]= {4, 5, c, 7, 8, 1}

In[110]:= M1 = {{1, 2, 4}, {3, 5, 8}, {7, 2, 3}}

Out[110]= {{1, 2, 4}, {3, 5, 8}, {7, 2, 3}}

In[111]:= M2 = {{a, b, c}, {1, 2, 3}, {4, 5, 6}}

Out[111]= {{a, b, c}, {1, 2, 3}, {4, 5, 6}}

In[112]:= Union[V1, V2, V3, M1, M2]

Out[112]= {1, 2, 3, 4, 5, 6, 7, 8, a, b, c, {1, 2, 3},
           {1, 2, 4}, {3, 5, 8}, {4, 5, 6}, {7, 2, 3}, {a, b, c}}

In[113]:= Union[{1, 2, 3, 1, 4, 5, 2, 4, 1, 7}]

Out[113]= {1, 2, 3, 4, 5, 7}

In[114]:= Join[V1, V2, V3, M1, M2]

Out[114]= {a, 3, 1, 7, 2, 4, 6, 1, b, 3, 4, 5, 8, 4, 5, c, 7, 8, 1,
           {1, 2, 4}, {3, 5, 8}, {7, 2, 3}, {a, b, c}, {1, 2, 3}, {4, 5, 6}}
```



```
In[115]:= Complement[V1, V2, V3]

Out[115]= {2, 6, a}

In[116]:= Intersection[V1, V2, V3]

Out[116]= {1, 4}
```

Listing 12

Mathematical operations on vectors and matrices

The simplest arithmetic operations on vectors and matrices will be considered in the following example.

Given vectors V1, V2 and matrices Ml, M2:

V1 = {1, 3, 5, 2, 6, 4};

V2 = {2, 7, 5, 8, 1, 3};

M1 = {{1, 2, 3}, {6, 5, 4}, {1, 3, 5}};

M2 = {{3, 2, 1}, {4, 5, 6}, {5, 3, 1}}.

Tasks:

- add, subtract, multiply and divide the vector V1 and the matrix Ml by a number 3;
- square vector V2 and matrix M2;
- calculate square root of vector V1 and matrix M1;
- calculate e^{V1}, sin(V2), ln(M1), cosh(M1).

The functions that realize mathematical operations are shown in Listing 13.

```
In[117]:= V1 = {1, 3, 5, 2, 6, 4}

Out[117]= {1, 3, 5, 2, 6, 4}

In[118]:= V2 = {2, 7, 5, 8, 1, 3}

Out[118]= {2, 7, 5, 8, 1, 3}

In[119]:= M1 = {{1, 2, 3}, {6, 5, 4}, {1, 3, 5}}

Out[119]= {{1, 2, 3}, {6, 5, 4}, {1, 3, 5}}

In[120]:= M2 = {{3, 2, 1}, {4, 5, 6}, {5, 3, 1}}

Out[120]= {{3, 2, 1}, {4, 5, 6}, {5, 3, 1}}

In[121]:= {V1 + 3, V1 - 3, V1 * 3, V1 / 3}
```

Out[121]= $\left\{\{4, 6, 8, 5, 9, 7\}, \{-2, 0, 2, -1, 3, 1\},\right.$
$$\left.\{3, 9, 15, 6, 18, 12\}, \left\{\frac{1}{3}, 1, \frac{5}{3}, \frac{2}{3}, 2, \frac{4}{3}\right\}\right\}$$

```
In[122]:= {M1 + 3, M1 - 3, M1 * 3, M1 / 3}
```

Out[122]= $\left\{\{\{4, 5, 6\}, \{9, 8, 7\}, \{4, 6, 8\}\}, \{\{-2, -1, 0\}, \{3, 2, 1\}, \{-2, 0, 2\}\},\right.$
$$\{\{3, 6, 9\}, \{18, 15, 12\}, \{3, 9, 15\}\},$$
$$\left.\left\{\left\{\frac{1}{3}, \frac{2}{3}, 1\right\}, \left\{2, \frac{5}{3}, \frac{4}{3}\right\}, \left\{\frac{1}{3}, 1, \frac{5}{3}\right\}\right\}\right\}$$

```
In[123]:= {V2^2, M2^2, Sqrt[V1], Sqrt[M1]}
```

Out[123]= $\left\{\{4, 49, 25, 64, 1, 9\},\right.$
$$\{\{9, 4, 1\}, \{16, 25, 36\}, \{25, 9, 1\}\}, \left\{1, \sqrt{3}, \sqrt{5}, \sqrt{2}, \sqrt{6}, 2\right\},$$
$$\left.\left\{\left\{1, \sqrt{2}, \sqrt{3}\right\}, \left\{\sqrt{6}, \sqrt{5}, 2\right\}, \left\{1, \sqrt{3}, \sqrt{5}\right\}\right\}\right\}$$

```
In[124]:= {E^V1, Sin[V2], Log[M1], Cosh[M2]}
```

Out[124]= $\left\{\left\{e, e^3, e^5, e^2, e^6, e^4\right\}, \{\mathrm{Sin}[2], \mathrm{Sin}[7], \mathrm{Sin}[5], \mathrm{Sin}[8], \mathrm{Sin}[1], \mathrm{Sin}[3]\},\right.$

```
  {{0, Log[2], Log[3]}, {Log[6], Log[5], Log[4]}, {0, Log[3], Log[5]}},
  {{Cosh[3], Cosh[2], Cosh[1]},
   {Cosh[4], Cosh[5], Cosh[6]}, {Cosh[5], Cosh[3], Cosh[1]}}}

In[125]:= N[%]

Out[125]= {{2.71828, 20.0855, 148.413, 7.38906, 403.429, 54.5982},
  {0.909297, 0.656987, -0.958924, 0.989358, 0.841471, 0.14112},
  {{0., 0.693147, 1.09861}, {1.79176, 1.60944, 1.38629},
   {0., 1.09861, 1.60944}}, {{10.0677, 3.7622, 1.54308},
   {27.3082, 74.2099, 201.716}, {74.2099, 10.0677, 1.54308}}}
```

Other functions designed to work with vectors and matrices are listed below:

Det [M] - gives the determinant of the square matrix;

IdentityMatrix [M] - gives the identity matrix: matrix with the diagonal elements equal 1, and others elements equals 0;

Transpose [M] - transposes the first two levels in M;

Inverse [M] - gives the inverse of a square matrix;

Tr [M] - finds the trace of the matrix M (sum of the diagonal elements);

LinearSolve [M, b] - returns the vector of unknowns of the matrix equation M * x = b, where M - matrix of coefficients of the system of equations, x - vector of unknowns, b - vector of free terms;

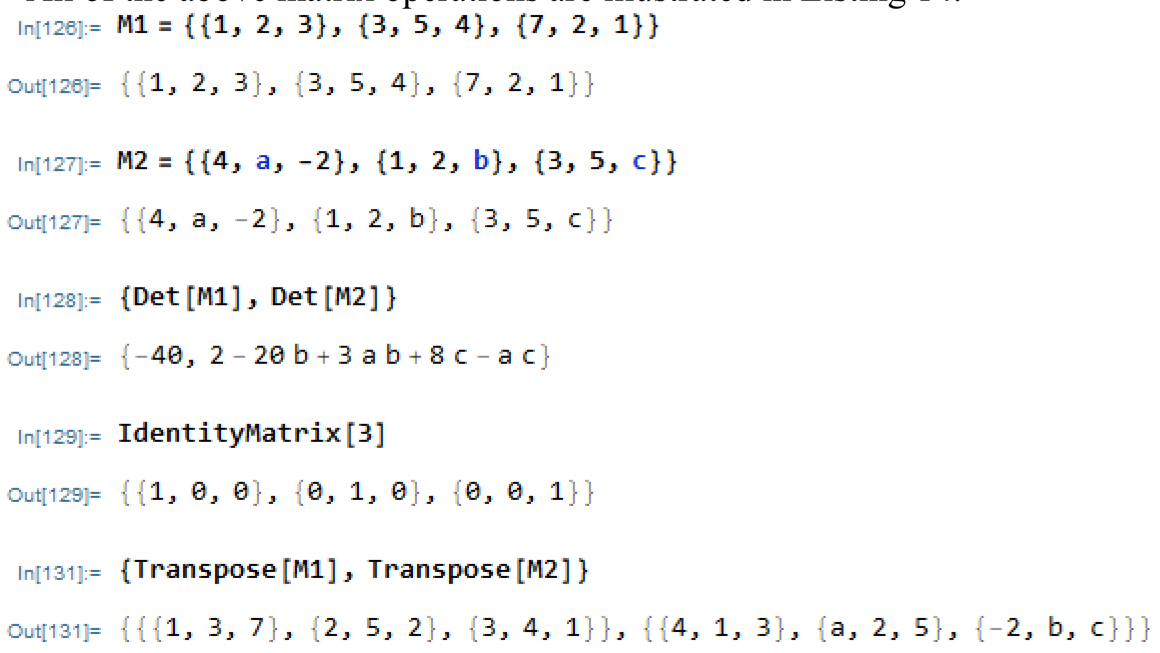
Eigensystem[M] - gives a list of the eigenvalues and eigenvectors of the square matrix M;

Eigenvalues [M] - gives a list of the eigenvalues of the square matrix M;

Eigenvectors[M] - gives a list of the eigenvectors of the square matrix M;

PseudoInverse[M] - finds the pseudoinverse of a rectangular matrix M.

All of the above matrix operations are illustrated in Listing 14.



```
In[126]:= M1 = {{1, 2, 3}, {3, 5, 4}, {7, 2, 1}}

Out[126]= {{1, 2, 3}, {3, 5, 4}, {7, 2, 1}}

In[127]:= M2 = {{4, a, -2}, {1, 2, b}, {3, 5, c}}

Out[127]= {{4, a, -2}, {1, 2, b}, {3, 5, c}}

In[128]:= {Det[M1], Det[M2]}

Out[128]= {-40, 2 - 20 b + 3 a b + 8 c - a c}

In[129]:= IdentityMatrix[3]

Out[129]= {{1, 0, 0}, {0, 1, 0}, {0, 0, 1}}

In[131]:= {Transpose[M1], Transpose[M2]}

Out[131]= {{{1, 3, 7}, {2, 5, 2}, {3, 4, 1}}, {{4, 1, 3}, {a, 2, 5}, {-2, b, c}}}
```

In[132]:= `{Inverse[M1], Inverse[M2]}`

Out[132]=
$$\left\{\left\{\left\{\frac{3}{40},-\frac{1}{10},\frac{7}{40}\right\},\left\{-\frac{5}{8},\frac{1}{2},-\frac{1}{8}\right\},\left\{\frac{29}{40},-\frac{3}{10},\frac{1}{40}\right\}\right\},\right.$$
$$\left\{\left\{\frac{-5\,b+2\,c}{2-20\,b+3\,a\,b+8\,c-a\,c},\frac{-10-a\,c}{2-20\,b+3\,a\,b+8\,c-a\,c},\frac{4+a\,b}{2-20\,b+3\,a\,b+8\,c-a\,c}\right\},\right.$$
$$\left\{\frac{3\,b-c}{2-20\,b+3\,a\,b+8\,c-a\,c},\frac{6+4\,c}{2-20\,b+3\,a\,b+8\,c-a\,c},\frac{-2-4\,b}{2-20\,b+3\,a\,b+8\,c-a\,c}\right\},\left\{-\frac{1}{2-20\,b+3\,a\,b+8\,c-a\,c},\right.$$
$$\left.\left.\left.\frac{-20+3\,a}{2-20\,b+3\,a\,b+8\,c-a\,c},\frac{8-a}{2-20\,b+3\,a\,b+8\,c-a\,c}\right\}\right\}\right\}$$

In[133]:= `{Tr[M1], Tr[M2]}`

Out[133]= $\{7,\ 6+c\}$

In[134]:= `{LinearSolve[M1, {1, 3, 7}], LinearSolve[M2, {1, 2, 3}]}`

Out[134]=
$$\left\{\{1,0,0\},\left\{\frac{-8-5\,b+3\,a\,b+2\,c-2\,a\,c}{2-20\,b+3\,a\,b+8\,c-a\,c},\frac{6-9\,b+7\,c}{2-20\,b+3\,a\,b+8\,c-a\,c},\frac{-17+3\,a}{2-20\,b+3\,a\,b+8\,c-a\,c}\right\}\right\}$$

In[135]:= `PseudoInverse[M1]`

Out[135]=
$$\left\{\left\{\frac{3}{40},-\frac{1}{10},\frac{7}{40}\right\},\left\{-\frac{5}{8},\frac{1}{2},-\frac{1}{8}\right\},\left\{\frac{29}{40},-\frac{3}{10},\frac{1}{40}\right\}\right\}$$

In[140]:= `Eigenvalues[M1]`

Out[140]= $\{\boxed{9.15\ldots},\ \boxed{-3.42\ldots},\ \boxed{1.28\ldots}\}$

In[141]:= `N[%]`

Out[141]= `{9.14592, -3.42345, 1.27752}`

In[142]:= `Eigenvectors[M1]`

Out[142]=
$$\left\{\left\{\frac{1}{71}\left(17+13\,\boxed{9.15\ldots}-\boxed{9.15\ldots}^2\right),\right.\right.$$
$$\left.\frac{1}{142}\left(-190-20\,\boxed{9.15\ldots}+7\,\boxed{9.15\ldots}^2\right),1\right\},$$
$$\left\{\frac{1}{71}\left(17+13\,\boxed{-3.42\ldots}-\boxed{-3.42\ldots}^2\right),\right.$$
$$\left.\frac{1}{142}\left(-190-20\,\boxed{-3.42\ldots}+7\,\boxed{-3.42\ldots}^2\right),1\right\},$$
$$\left\{\frac{1}{71}\left(17+13\,\boxed{1.28\ldots}-\boxed{1.28\ldots}^2\right),\right.$$
$$\left.\left.\frac{1}{142}\left(-190-20\,\boxed{1.28\ldots}+7\,\boxed{1.28\ldots}^2\right),1\right\}\right\}$$

```
In[143]:= N[%]

Out[143]= {{0.735903, 1.4973, 1.},
           {-0.552462, -0.278106, 1.}, {0.450362, -1.43751, 1.}}
```

Listing 14

The vector product is realized using the function Dot [V1, V2] or a multiplication sign in the form of a point: V1. V2 or Ml. M2, see Listing 15.

```
In[144]:= V1 = {1, 3, 5, 2, 6, 4}

Out[144]= {1, 3, 5, 2, 6, 4}

In[145]:= V2 = {2, 7, 5, 8, 1, 3}

Out[145]= {2, 7, 5, 8, 1, 3}

In[146]:= M1 = {{1, 2, 3}, {6, 5, 4}, {1, 3, 5}}

Out[146]= {{1, 2, 3}, {6, 5, 4}, {1, 3, 5}}

In[147]:= M2 = {{3, 2, 1}, {4, 5, 6}, {5, 3, 1}}

Out[147]= {{3, 2, 1}, {4, 5, 6}, {5, 3, 1}}

In[148]:= V1*V2

Out[148]= {2, 21, 25, 16, 6, 12}

In[149]:= V1.V2

Out[149]= 82

In[150]:= M1.M2

Out[150]= {{26, 21, 16}, {58, 49, 40}, {40, 32, 24}}

In[151]:= Dot[V1, V2]

Out[151]= 82

In[152]:= Dot[M1, M2]

Out[152]= {{26, 21, 16}, {58, 49, 40}, {40, 32, 24}}
```

Listing 15

Lecture № 5. Graphical functions of the Mathematica system

Two-dimensional graphics

Plot function

The Plot function allows you to build graphically defined graphs in two-dimensional space in a rectangular coordinate system. Several functions can be displayed on one graph. By default, the grid is displayed on the screen.

Plot function recording format

Plot[f,{x,xmin,xmax}];

Plot[{f1,f2,...},{x,xmin,xmax}],

where f – function, the graph of which is built,

fi – i-th function, the graph of which is built, i=1,2,…

x – function argument,

xmin, xmax – argument change interval x.

Function Plot options

The options of the Plot function are set as follows:

Option name -> option value.

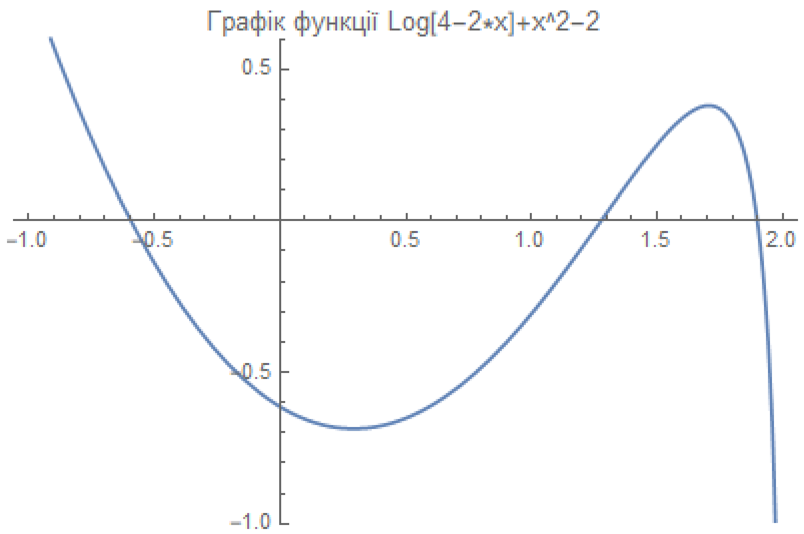
The main options of the Plot function are:

- setting the scale along the axis:
  PlotRange -> {ymin,ymax} – sets the y-axis scale from ymin to ymax with automatic step selection;
  PlotRange->{{ xmin,xmax}, {ymin,ymax}} - sets the scale on the y-axis from ymin to ymax and on the x-axis from xmin to xmax with automatic step selection;
- definition of the axis name:
  AxesLabel -> {“Tx”, “Ty”} – sets the inscriptions Tx and Ty on the x and y axes, respectively;
- determining the name of the plot:
  PlotLabel -> “T” – sets the name of the plot;
- choice of graphics style:
  Axes -> None – the schedule is built without axes.

Examples of using the Plot function.

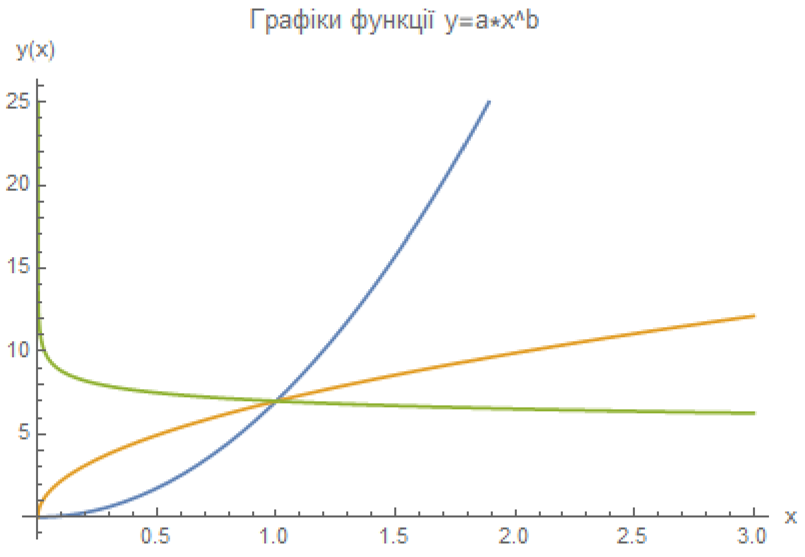
```
In[21]:= f = Log[4 - 2 * x] + x^2 - 2;
         Plot[f, {x, -1, 2}, PlotRange → {-1, 0.6},
          PlotLabel → "Графік функції Log[4-2*x]+x^2-2"]
```

Out[22]=



```
In[23]:= Plot[{7 * x^2, 7 * x^0.5, 7 * x^ (-0.1)}, {x, 0, 3},
          AxesLabel → {"x", "y(x)"}, PlotLabel → "Графіки функції y=a*x^b"]
```

Out[23]=



Listing 1

ListPlot function

Used to plot graphs given as an array of points.

List format of the ListPlot function

ListPlot[{y1,y2,…}];

ListPlot[{x1,y1},{x2,y2},…}],

where yi – i-th value of function y(x),

xi - i-th the value of the function y(x) argument.

Purpose of the ListPlot function

ListPlot[{y1,y2,…}] – plots the point values of the function y (x) with the notation of the number of points on the x-axis.

ListPlot[{x1,y1},{x2,y2},…}] – plots a list of points with specified x and y coordinates.

The ListPlot function has two options:

- definition of the axis name:

AxesLabel -> {“Tx”, “Ty”} – sets the inscriptions Tx and Ty on the x and y axes, respectively;

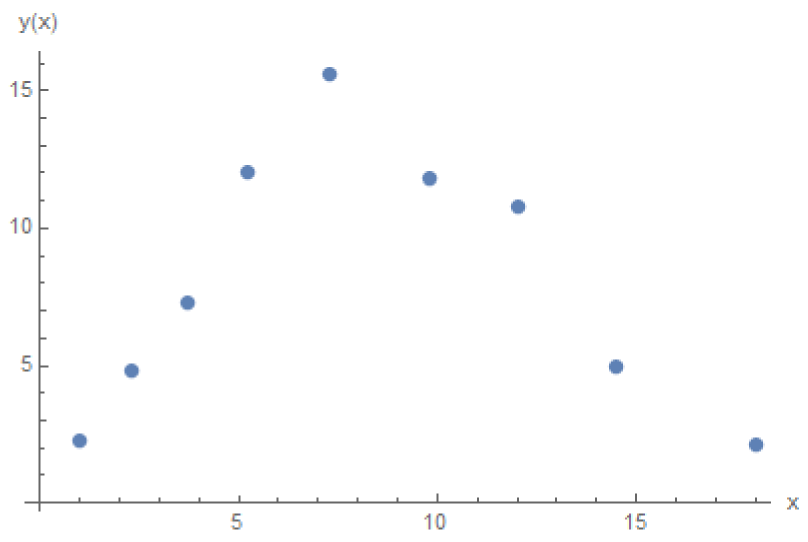
- determining the size of points:

PlotStyle -> PointSize [d] - sets the diameter of the point equal to d.

Example of using the ListPlot function.

```
In[28]:= f = {{1, 2.3}, {2.3, 4.8}, {3.7, 7.3}, {5.2, 12}, {7.3, 15.6},
            {9.8, 11.8}, {12, 10.8}, {14.5, 5}, {18, 2.1}};
         ListPlot[f, AxesLabel → {"x", "y(x)"}, PlotStyle → PointSize[0.02]]
```

Out[29]=



Listing 2

Show function

Used to plot point and analytical graphs in one plane.

Show recording format

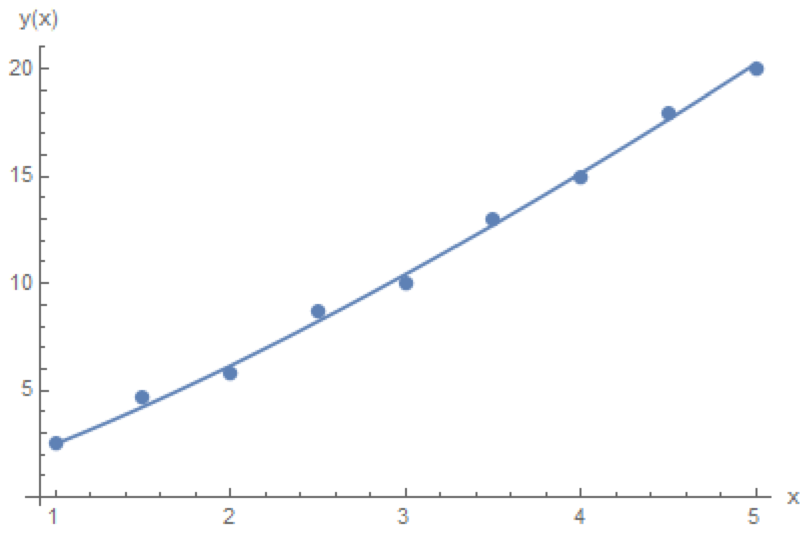
Show[r1,r2],

where r1, r2 – are variables used to denote graphs

Example of using the Show function.

```
In[30]:= f = {{1, 2.5}, {1.5, 4.7}, {2, 5.8}, {2.5, 8.7}, {3, 10}, {3.5, 13},
            {4, 15}, {4.5, 18}, {5, 20}};
         r1 := ListPlot[f, AxesLabel → {"x", "y(x)"},
           PlotStyle → PointSize[0.02]]
         r2 := Plot[2.5 x^1.3, {x, 1, 5}]
         Show[r1, r2]
```

Out[33]=



Listing 3

Choice of graphic style

The PlotStyle option allows you to select the color of the lines and their thickness.

PlotStyle option directives

- line color:

PlotStyle ->{GrayLevel[k1],GrayLevel[k2],…},

where k1,k2,… - the color codes of the lines in shades of gray of the corresponding functions are selected from the range 0..1;

PlotStyle ->{Hue[c1],Hue[c2],…},

where c1,c2,… - tabular color codes of the lines of the corresponding functions, selected from the range 0..1;

PlotStyle ->{ RGBColor[r1,g1,b1],Hue[r2,g2,b2],…},

where r1,g1,b1… - the brightness of the red, green and blue color components are selected from the range 0..1;

- line thickness:

PlotStyle ->Thickness[d] – sets the thickness of the lines of the graph as a fraction of its full width;

PlotStyle ->AbsoluteThickness[d] – sets the thickness of the graph lines in pixels;

- dashing style:

PlotStyle ->Dashing[{d1,d2,… }] – sets the stroke length of the graph lines, where di is specified as a fraction of the width of the graph line;

PlotStyle->AbsoluteDashing[d1] - sets the stroke length of the lines of the graph, where di is specified in pixels;

- point graph:

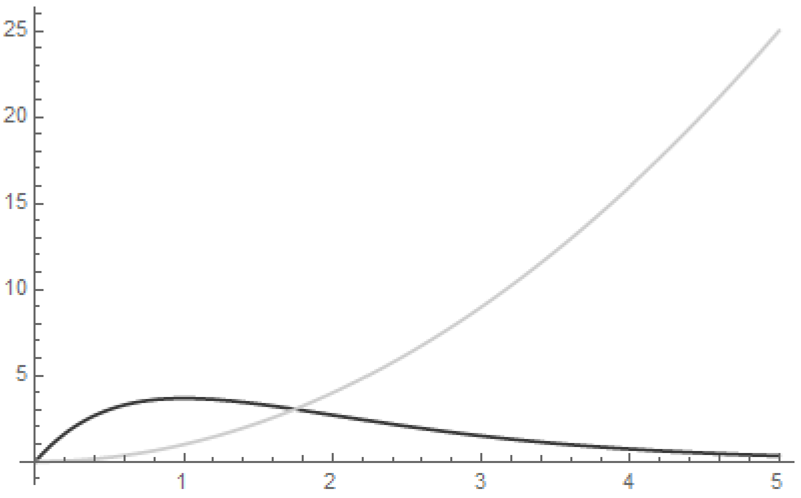
PlotStyle->PointSize[d] – graph in the form of circles with a diameter of d, which are measured in fractions of the total width of the graph;

PlotStyle->AbsolutePointSize[d] – graph in the form of circles with a diameter of d, which are measured in pixels.

Examples of using the PlotStyle function.

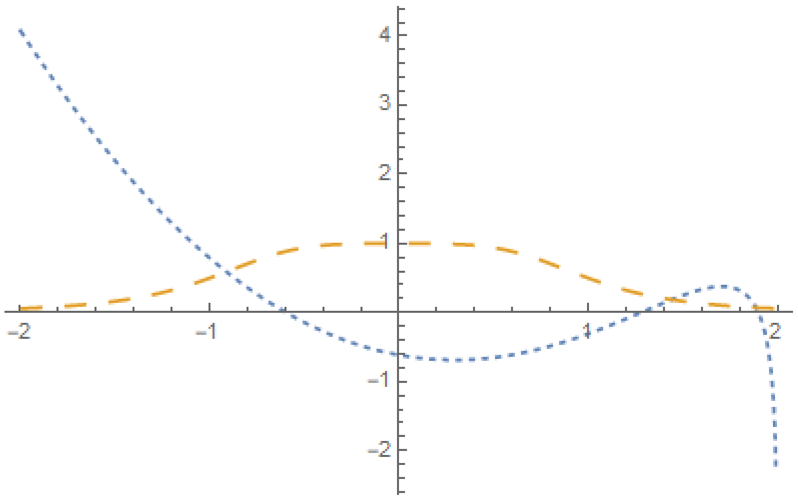
```
In[34]:= Plot[{10*x*Exp[-x], x^2}, {x, 0, 5},
          PlotStyle → {GrayLevel[0.2], GrayLevel[0.8]}]
```

Out[34]=



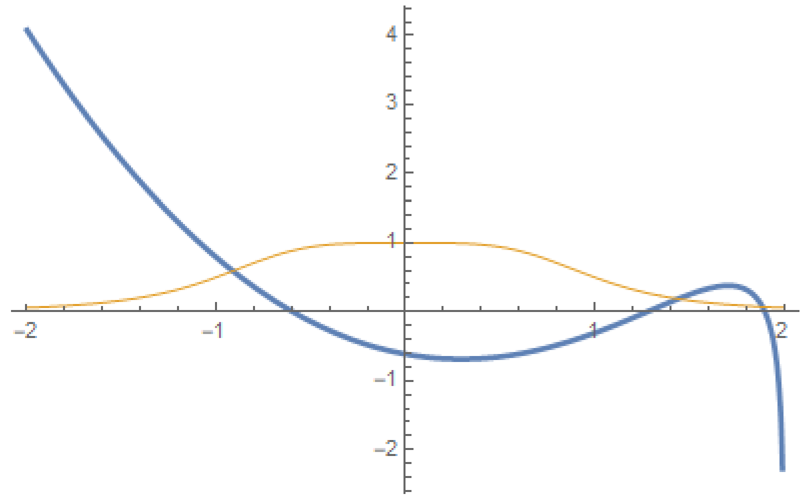
```
In[36]:= Plot[{Log[4 - 2*x] + x^2 - 2, 1/(1 + x^4)}, {x, -2, 2},
          PlotStyle → {AbsoluteDashing[3], AbsoluteDashing[10]}]
```

Out[36]=



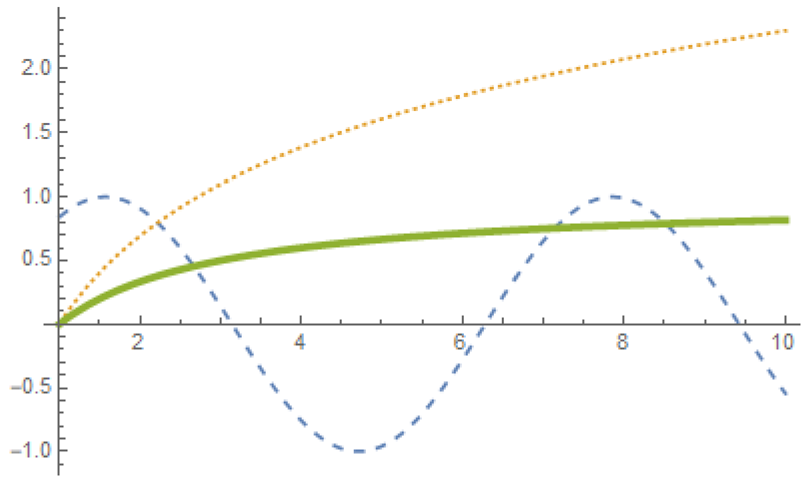
```
In[38]:= Plot[{Log[4 - 2*x] + x^2 - 2, 1 / (1 + x^4)}, {x, -2, 2},
          PlotStyle → {Thickness[0.007], Thickness[0.001]}]
```

Out[38]=



```
In[39]:= Plot[{Sin[x], Log[x], (x - 1) / (x + 1)}, {x, 1, 10},
          PlotStyle → {AbsoluteDashing[5], AbsoluteDashing[2], Thickness[0.01]}]
```

Out[39]=



Listing 4

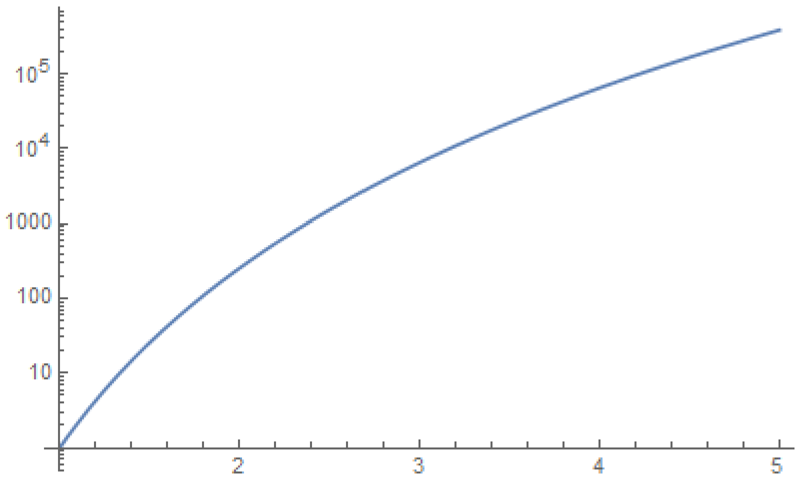
Graphs of special types

Graphing functions on a logarithmic scale

LogPlot [f, {x, xmin, xmax}] - plots a linear-logarithmic graph of the function f with a logarithmic scale on the y-axis in the range xmin..xmax.

```
In[41]:= LogPlot[x^8, {x, 1, 5}]
```

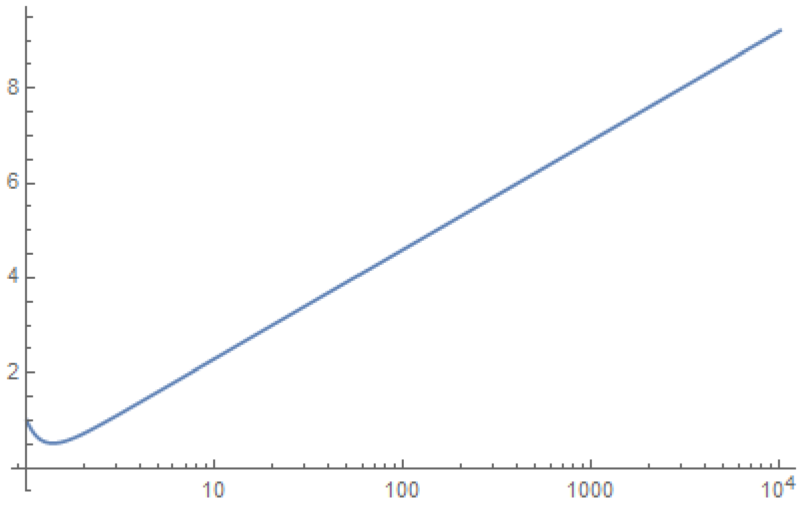
Out[41]=



LogLinearPlot[f,{x,xmin,xmax}] – plots a logarithmic-linear graph of the function f with a logarithmic scale on the x-axis in the range xmin..xmax.

```
In[42]:= LogLinearPlot[x^-5 + Log[x], {x, 1, 10000}]
```

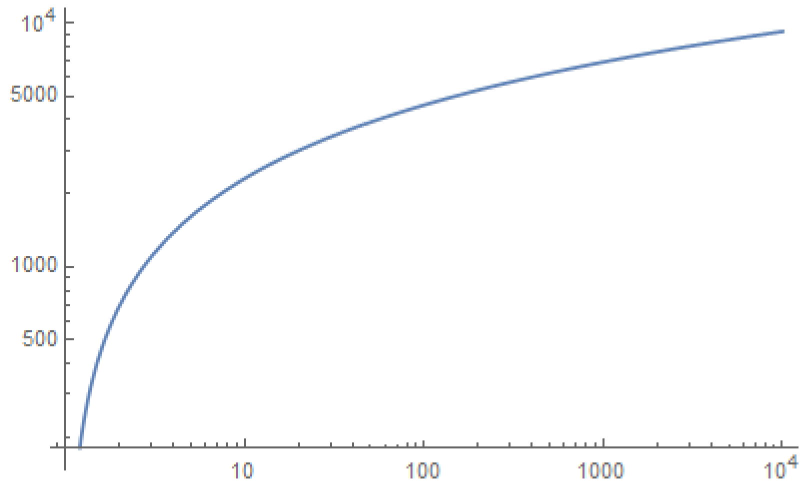
Out[42]=



LogLogPlot[f,{x,xmin,xmax}] – plots a graph of the function f with a logarithmic scale on two axes in the range xmin..xmax.

```
In[43]:= LogLogPlot[x^-5 + 1000 * Log[x], {x, 1, 10000}]
```

Out[43]=



Functions
LogListPlot[{x1,y1},{x2,y2},…]
LogLinearListPlot[{x1,y1},{x2,y2},…]
LogLogListPlot[{x1,y1},{x2,y2},…]
similar to the previous three and are used to construct scatter plots.

The function of plotting graphs in the polar coordinate system

PolarPlot[f,{t,tmin,tmax}] – plots the position of the end of the vector f when the angle t changes from tmin to tmax.

```
In[44]:= PolarPlot[(1 + 2 * Sin[x]) / (1 + 2 * Cos[x]), {x, 0, 2 * Pi}]
```

Out[44]=

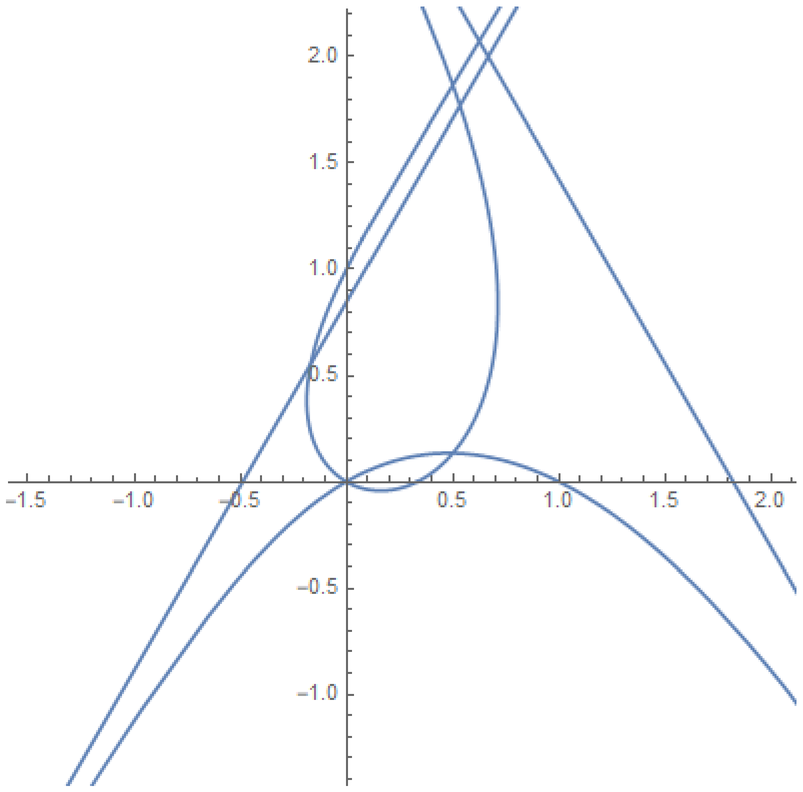
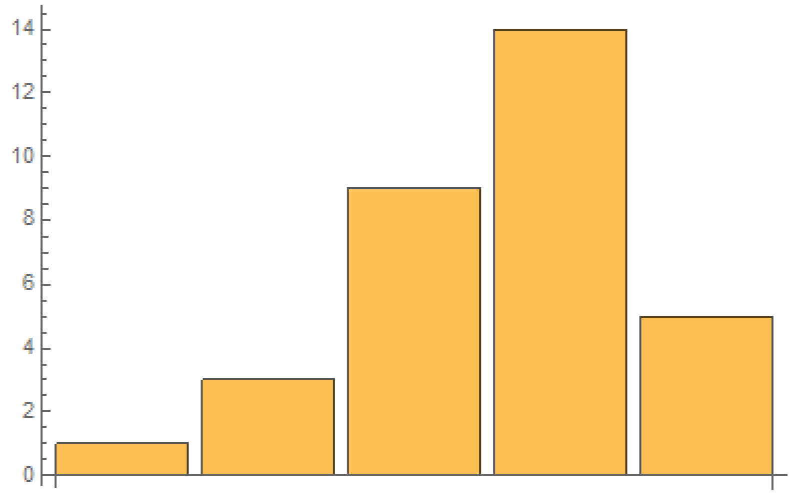


Chart construction function

BarChart[{c1,c2,…}] – makes a bar chart with bar lengths.

```
In[46]:= BarChart[{1, 3, 9, 14, 5}]
```

Out[46]=

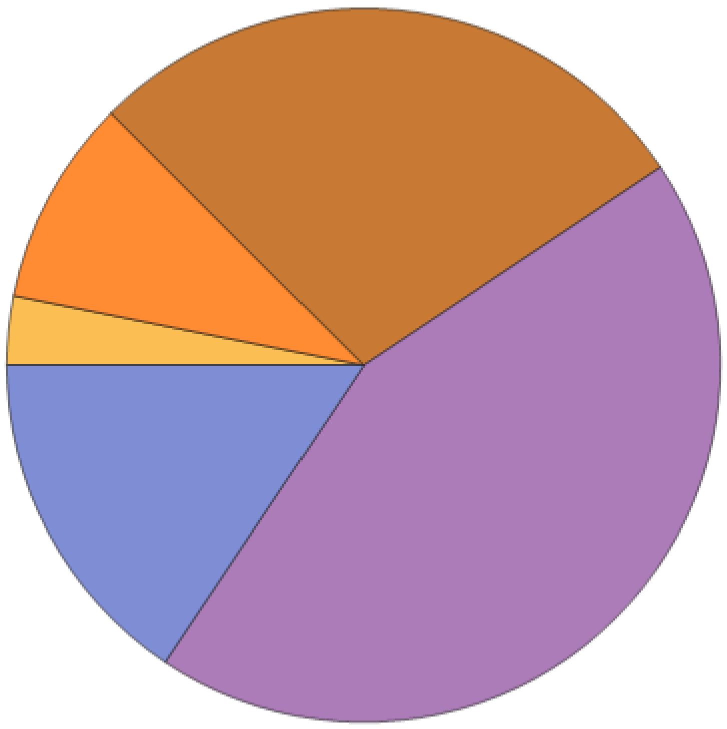


PieChart[{c1,c2,…}] – makes a pie chart with sector angle proportional to list.

The PieChart function uses a number of options, the description of which is available with the Options[PieChart] command.

```
In[47]:= PieChart[{1, 3, 9, 14, 5}]
```

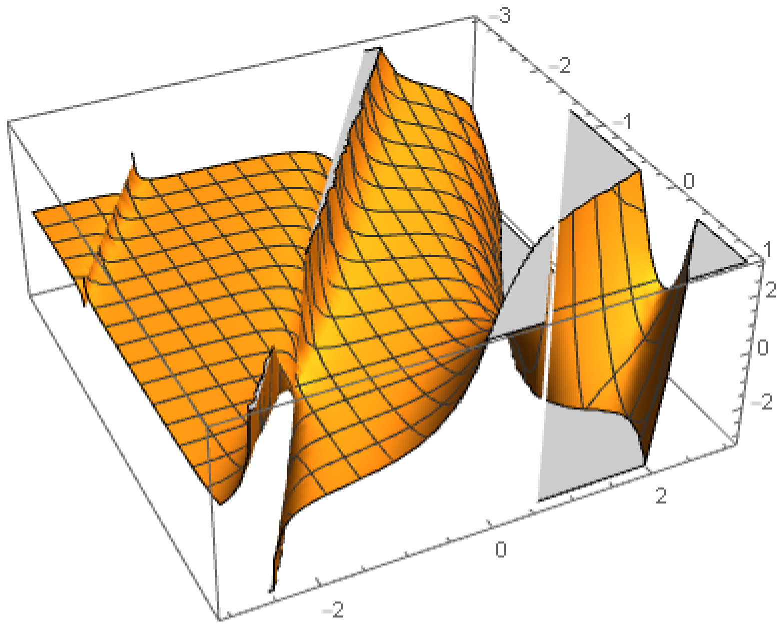
Out[47]=



Functions of three-dimensional graphics

Plot3D[f,{x,xmin,xmax},{y,ymin,ymax}] – plots the function f = f (x, y). Plot3D options are available with the Options[Plot3D] command.

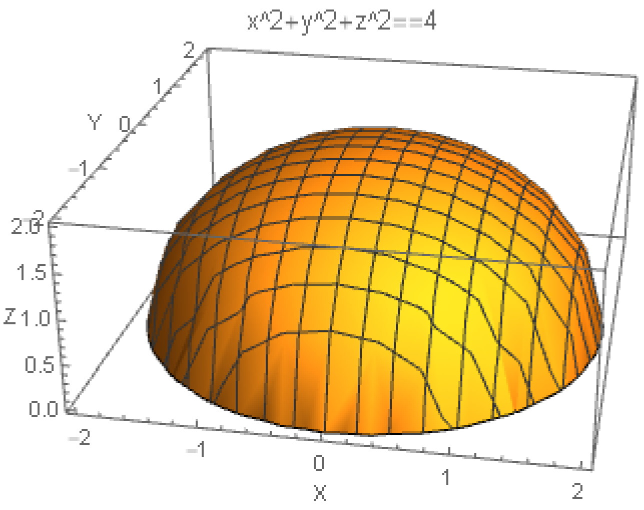
```
In[48]:= Plot3D[x*Exp[x+y]*Sin[x+y]/Cos[x+y], {x, -3, 1}, {y, -3, 3}]
```



When constructing graphs expressed in 3 arguments, one argument must be expressed in others. For example, the graph of the sphere: $x^2+y^2+z^2=R^2$, where R-radius, will be converted to $z=\sqrt{R^2-x^2-y^2}$

```
In[49]:= Plot3D[Sqrt[4 - x^2 - y^2], {x, -10, 10}, {y, -10, 10},
          AxesLabel → {"X", "Y", "Z"}, PlotLabel → "x^2+y^2+z^2==4"]
```

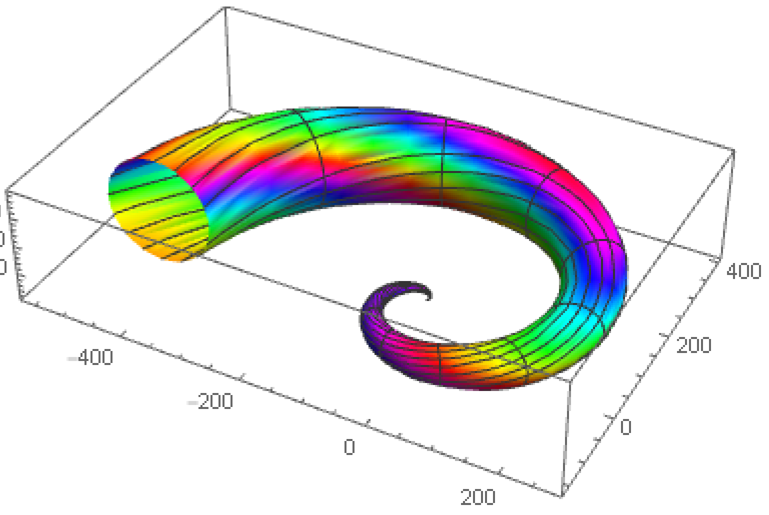
Out[49]=



ParametricPlot3D[{f1,f2,f3},{t1,t1min,t1max},{t2,t2min,t2max}] – plots a three-dimensional graph of a parametrically given function z(t1, t2) = f(x(t1, t2), y(t1, t2)). Plot3D options are available with the Options[ParametricPlot3D] command.

```
In[63]:= ParametricPlot3D[{x^2*Cos[x]*(5+Cos[x+y]),
           x^2*Sin[x]*(5+Cos[x+y]), x^2*Sin[x+y]}, {x, 0, 3*Pi},
          {y, 0, 2*Pi},
          ColorFunction → Function[{x, y, z}, Hue[x^2+y^2+z^2]]]
```

Out[63]=



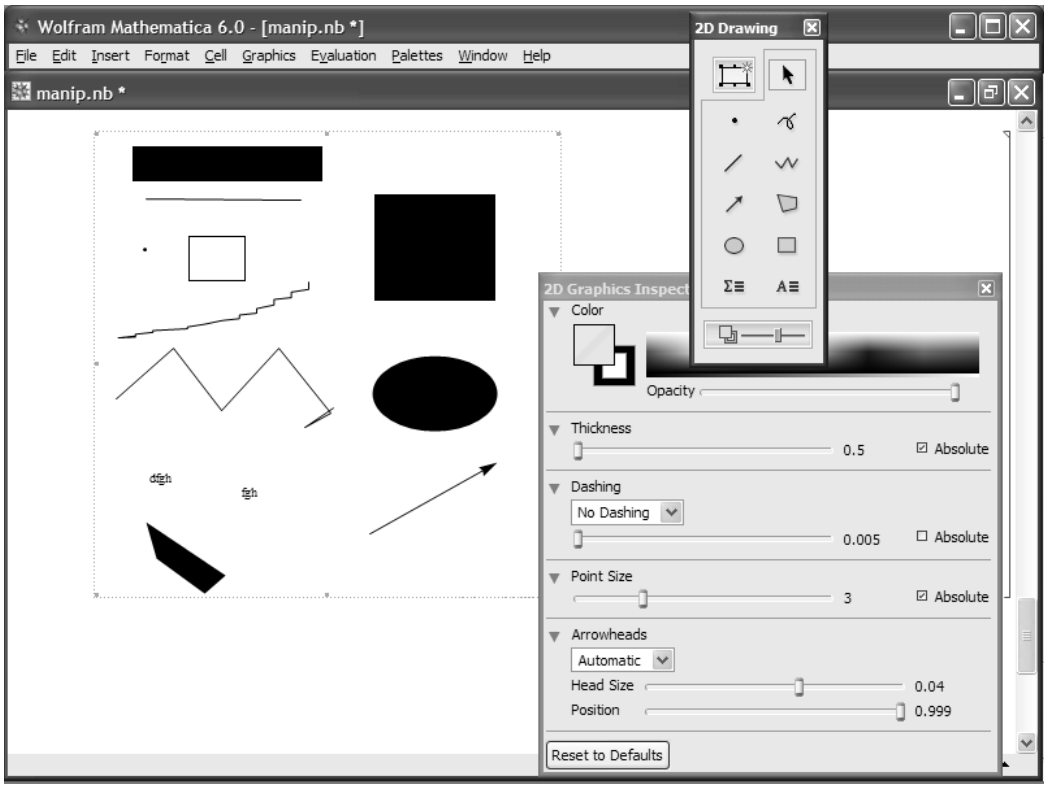
Lecture № 6. Possibilities of Mathematica system operation with heterogeneous data

Position of Graphics menu and graphic editor.

When creating complex laptops in previous versions of Mathematica clearly lacked the resources to prepare at least simple drawings and diagrams, which are often accompanied by mathematical and scientific calculations. Such drawings and diagrams, of course, can be created by programming Mathematica systems, but it requires a lot of time and the ability to program graphic tasks well. With this in mind, the developers of Mathematica introduced a tool for building simple drawings with the mouse type of the well-known graphic editor Paint. Access to it is provided from the new Graphics menu item. It contains the following commands:

- New Graphic - output window for plotting;
- Drawing Tool - output of the graphic editor window;
- Graphics Inspector - output of the graphics inspector window;
- Rendering - output of substitution of rendering operations;
- Opetations - output of the substitution of additional operations.

Working with these graphics is simple and obvious. It is illustrated by fig. 1. It shows the windows of the drawing, graphic editor and graphics inspector. Note that the graphic editor does not have the funds to build an unpainted ellipse, rectangle and polygon. However, the installation of colors, these figures are easy to obtain.


Wolfram Mathematica 6.0 - [manip.nb *]
File Edit Insert Format Cell Graphics Evaluation Palettes Window Help
manip.nb *
2D Drawing
2D Graphics Inspect
Color
Opacity
Reset to Defaults

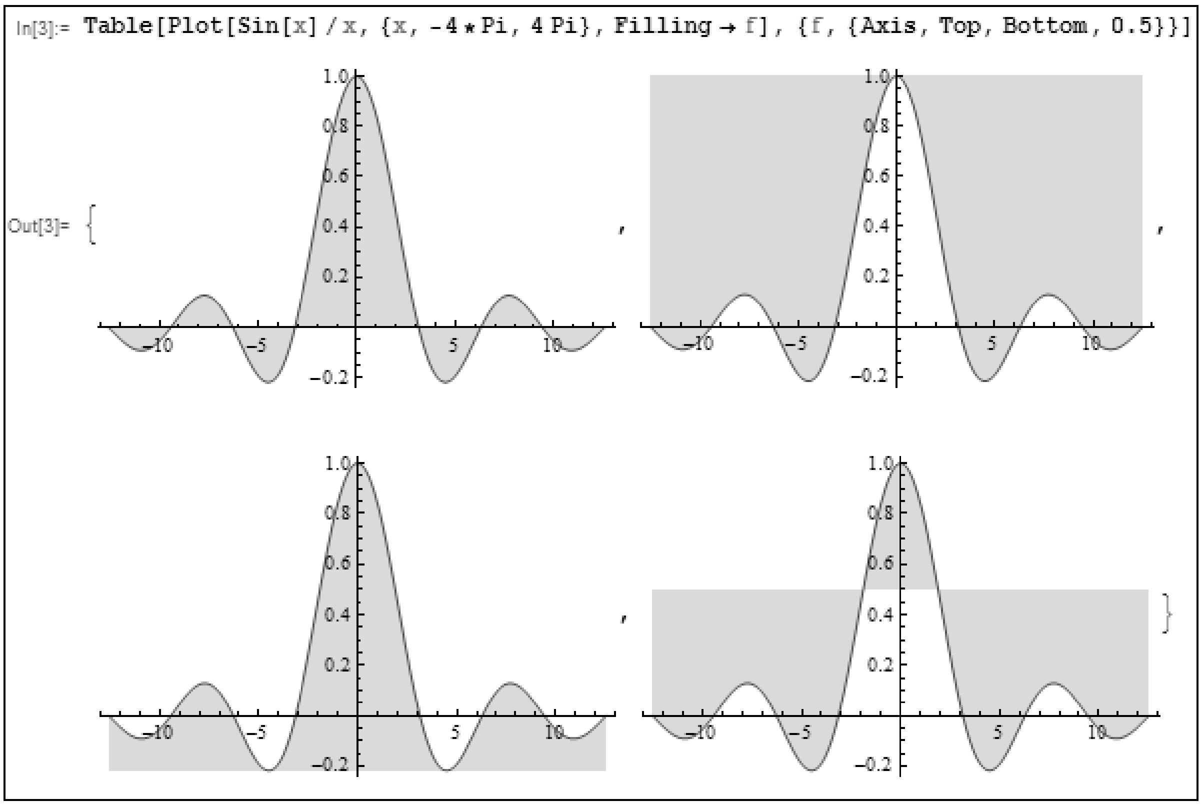
Thickness 0.5 ☑ Absolute
Dashing: No Dashing, 0.005 ☐ Absolute
Point Size 3 ☑ Absolute
Arrowheads: Automatic
Head Size 0.04
Position 0.999

Figure 1.

Use options for painting areas of two-dimensional graphs.

Of the new options for the **Plot** function in Mathematica, the most impressive are the options for painting the areas of two-dimensional graphs **Filling** (Painting) and **FillingStyle** (Painting Style). The first is disabled by default, the second is set to **Auto**.

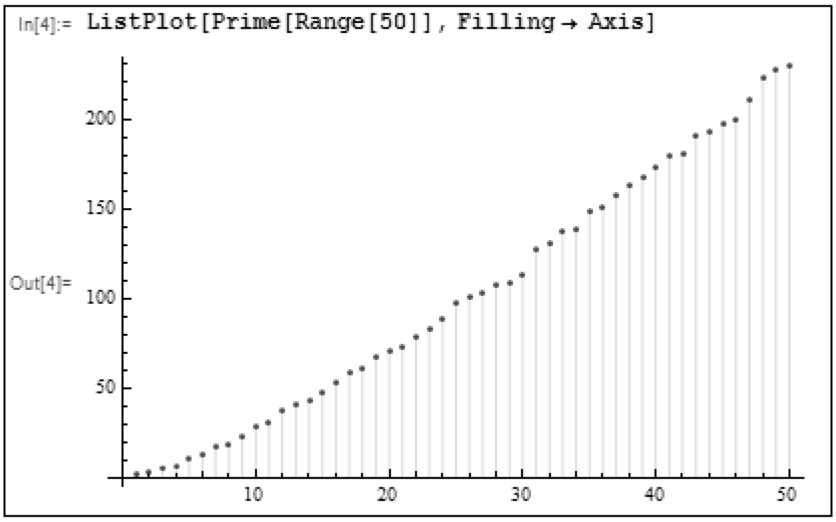
We will demonstrate the result of the Filling option (Fig. 2). On it the **Plot** function builds a graph of the **Sin [x] / x** function in the interval of change x from -4*Pi to +4*Pi with 4 types of painting. They are represented by the values of the **Filling option**: **Axis** (painting from each point of the curve to the x-axis), **Top** (painting the area from the top of the graph window to its curve), **Bottom** (painting from the curve to the bottom of the graph window) and 0.5 (painting from the graph line to horizontal with a vertical coordinate equal to 0.5).



```
In[3]:= Table[Plot[Sin[x] / x, {x, -4 * Pi, 4 Pi}, Filling → f], {f, {Axis, Top, Bottom, 0.5}}]
```

Рис. 2.

This option can also be used with the **ListPlot[list]** function, which builds points with coordinates taken from the list. It makes it possible to construct verticals connecting the points of the graph with the x axis (Fig. 3). This figure shows how the values of prime numbers change from their number. Interestingly, this dependence is close to linear.



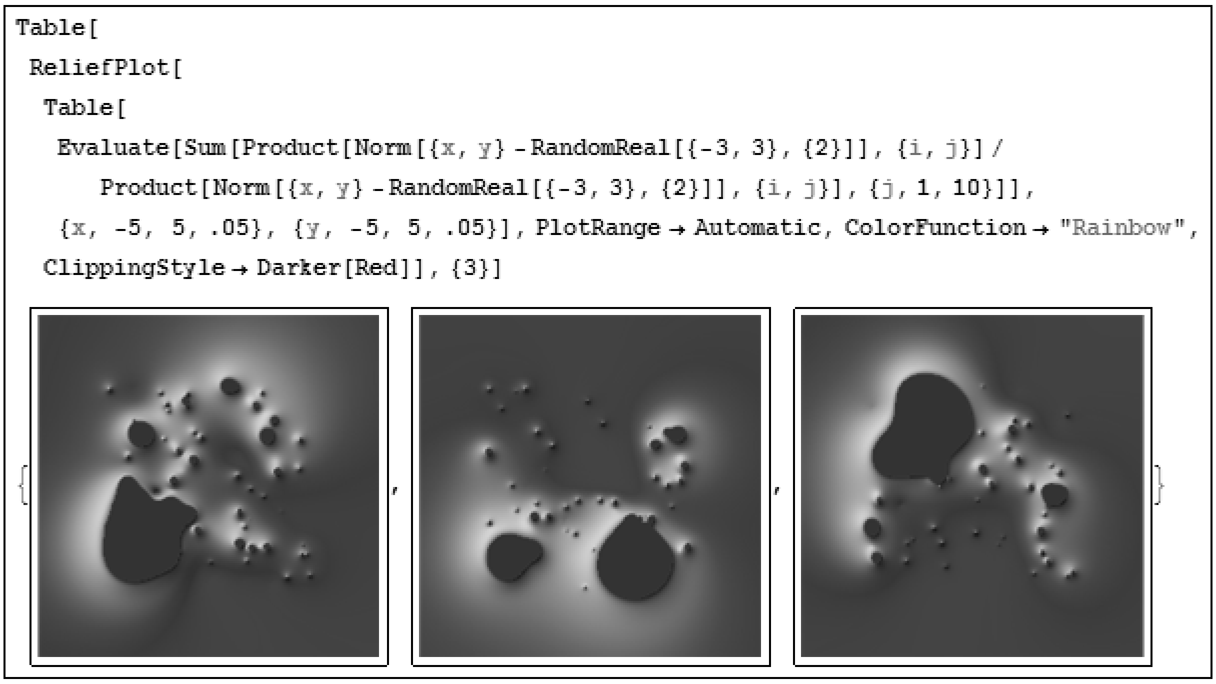
```
In[4]:= ListPlot[Prime[Range[50]], Filling→ Axis]
```

Figure 3.

Relief graphics.

The function of the Mathematica **ReliefPlot [array]** is used to construct realistic terrain graphs, which are given by the coordinates of the points of the array. An example of the application of this function is Fig. 8.61, on which it is built
the relief of the surface given by the mathematical formula i + cos (i3 + j3), where i and j change with a discreteness of 0.03 in the range from -4 to 4. The shape of the relief depends on the value of the **ColorFunction** option.

Another example of using the **ReliefPlot** function is presented in Fig. 4. Here are three arrays of random results of a number of arithmetic operations, including the norms of matrices. The obtained three reliefs are largely random and vary from start to start of the presented module.



```
Table[
 ReliefPlot[
  Table[
   Evaluate[Sum[Product[Norm[{x, y} - RandomReal[{-3, 3}, {2}]], {i, j}] /
      Product[Norm[{x, y} - RandomReal[{-3, 3}, {2}]], {i, j}], {j, 1, 10}]],
   {x, -5, 5, .05}, {y, -5, 5, .05}], PlotRange → Automatic, ColorFunction → "Rainbow",
  ClippingStyle → Darker[Red]], {3}]
```

Figure 4

Fig. 5 builds the relief of the imaginary part of the function sec (i + I * j) 2 for two values of the **PlotRange** option, equal to **All** i **Automatic**. It is easy to notice a serious change in the nature of the detection of details of the same relief. Of course, an array for this function can be created not only by mathematical expressions, but also in any other way - for example, by loading arrays of images. Many options of the **ReliefPlot** function allow you to create reliefs with different resolutions, different colors and other features.

```
In[7]:= {ReliefPlot[Table[Im[Sec[(i + I j) ^2]], {i, -3, 3, .01}, {j, -3, 3, .01}], PlotRange → All],
         ReliefPlot[
          Table[Im[Sec[(i + I j) ^2]], {i, -3, 3, .01}, {j, -3, 3, .01}], PlotRange → Automatic]}
```

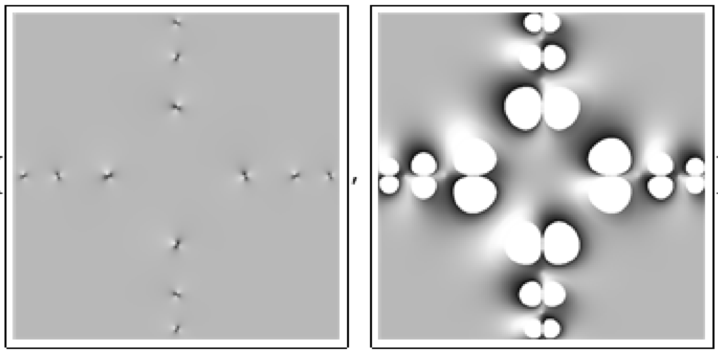
Out[7]=


Figure 5

Three-dimensional objects obtained by rotating curves.

Three-dimensional graphic objects obtained by rotating curves about an axis are quite common. For example, turning the circle at an angle π, you can get the surface of the sphere. By changing the boundaries of the angle of rotation, you can build closed or open shapes. To construct such surfaces (figures) in Mathematica is a function:

RevolutionPlot3D[fz,{t,tmin,tmax},...]

RevolutionPlot3D [{fx,fy,fz},{t,tmin,tmax},...]

In fig. 6 shows the application of this function to construct the surface of the bagel half. The figure looks quite realistic.

```
In[1]:= RevolutionPlot3D[{2 + Cos[t], Sin[t]}, {t, Pi, 2 Pi}]
```

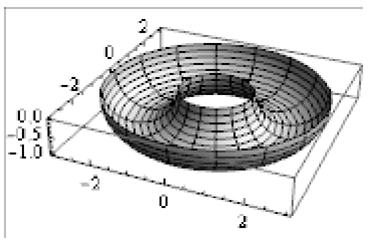
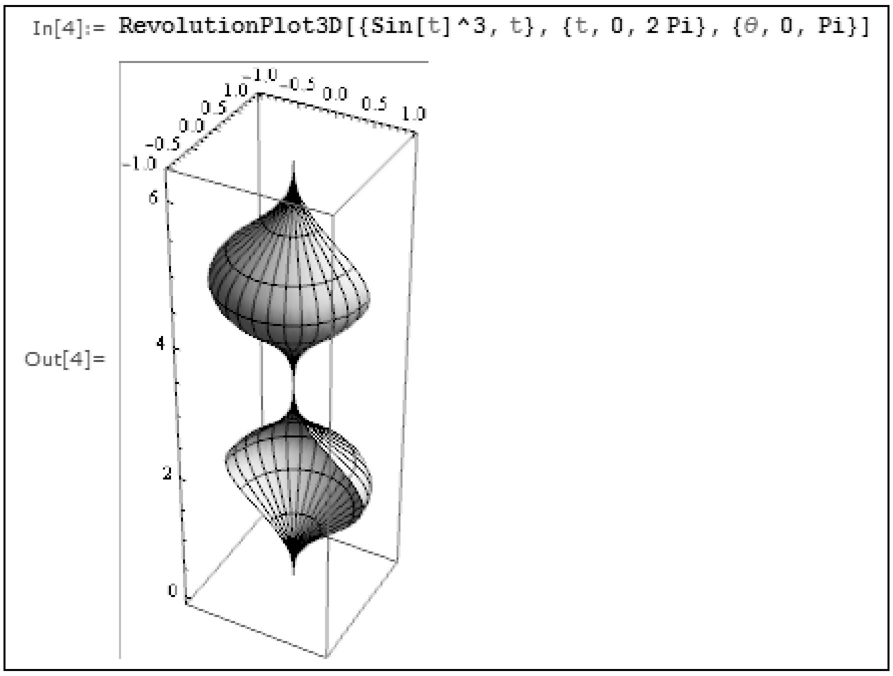
Out[1]=


Figure 6

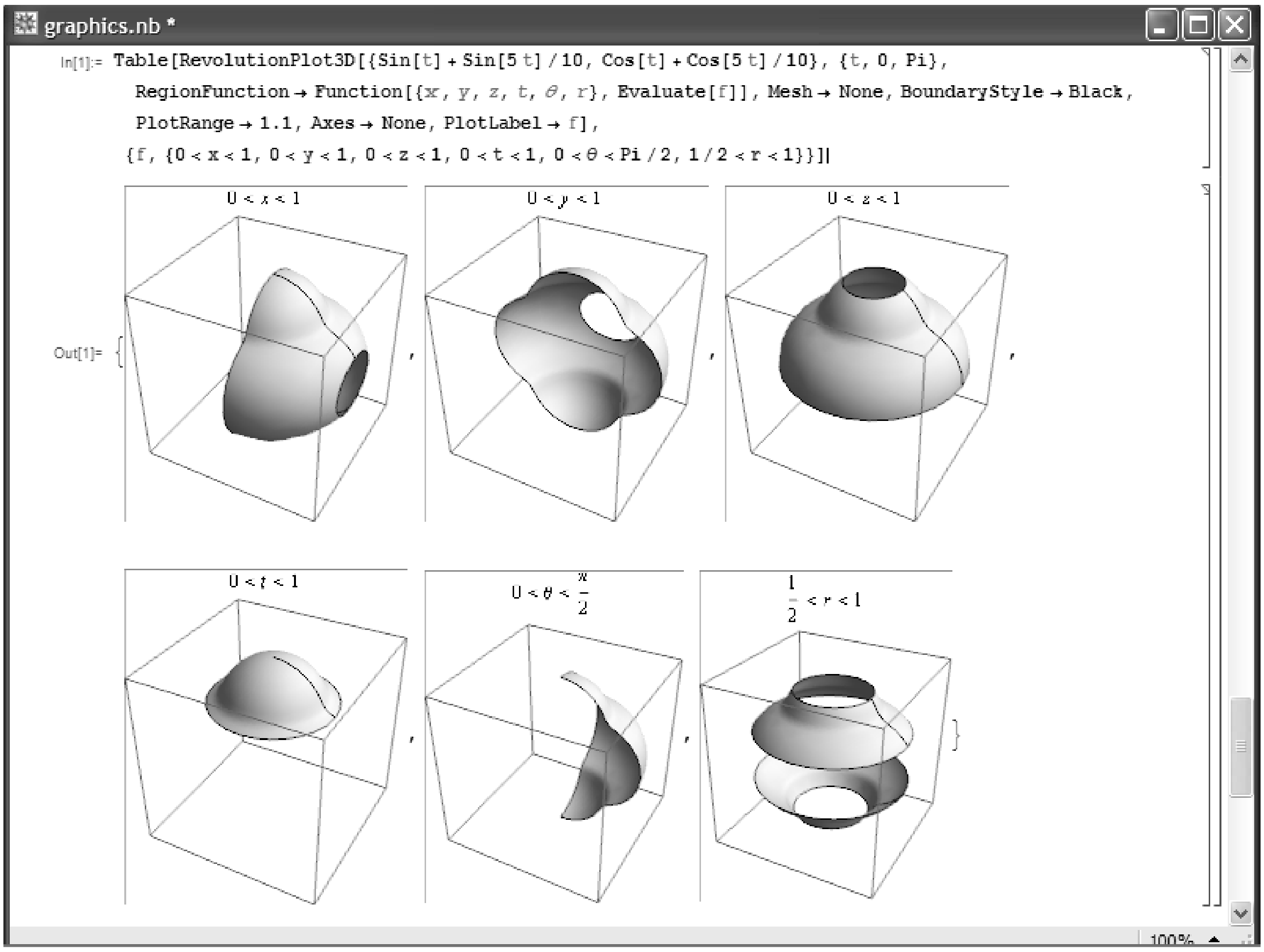
A more funny figure is built using this function, shown in Fig. 7. Here the parameters are two changing angles, and a parametric curve is used for rotation.



```
In[4]:= RevolutionPlot3D[{Sin[t]^3, t}, {t, 0, 2 Pi}, {θ, 0, Pi}]
Out[4]=
```

Figure 7

An example of using the options of the RevolutionPlot3D function is presented in Fig. 8. Here the curve of rotation is given by means of six inequalities taking which 6 figures are constructed. Unfortunately, as before, the color of the figures is reproduced only in shades of gray.



```
graphics.nb *
In[1]:= Table[RevolutionPlot3D[{Sin[t] + Sin[5 t] / 10, Cos[t] + Cos[5 t] / 10}, {t, 0, Pi},
          RegionFunction → Function[{x, y, z, t, θ, r}, Evaluate[f]], Mesh → None, BoundaryStyle → Black,
          PlotRange → 1.1, Axes → None, PlotLabel → f],
        {f, {0 < x < 1, 0 < y < 1, 0 < z < 1, 0 < t < 1, 0 < θ < Pi / 2, 1 / 2 < r < 1}}]
Out[1]= {
```

Figure .

Lecture № 7. Functions for solving algebraic equations and systems of equations in Mathematica

Analytical methods for solving algebraic and transcendental equations

Solve function

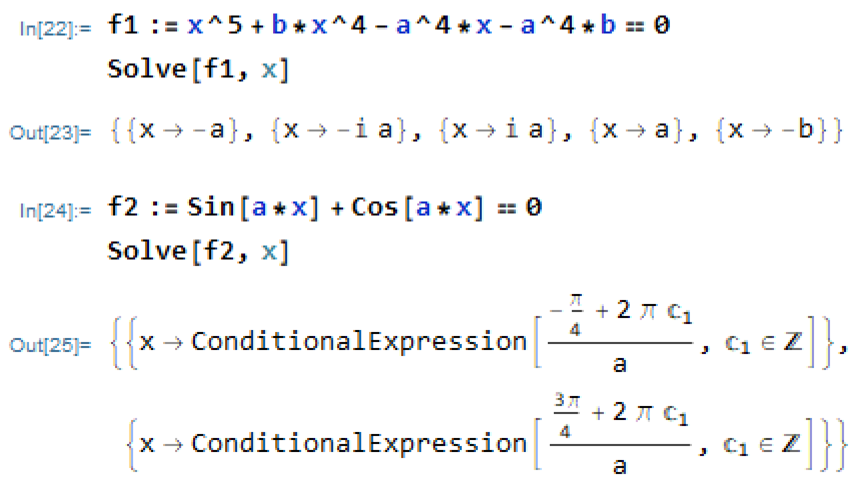
To solve the equations in analytical form, the Solve function is used, the recording format of which is as follows:

Solve[f,x],

where f – equation, which is written in any form,

x – variable name.

The equation symbol "==" is used to write the equation, for example ax^2 + bx + c == 0. Pay attention that the Solve function does not always form a solution in the most compact form, so after using this function sometimes you need to use Simplify, Expand or FullSimplify. Examples of using the Solve function are shown in Listing 1.



```
In[22]:= f1 := x^5 + b*x^4 - a^4*x - a^4*b == 0
         Solve[f1, x]

Out[23]= {{x → -a}, {x → -i a}, {x → i a}, {x → a}, {x → -b}}

In[24]:= f2 := Sin[a*x] + Cos[a*x] == 0
         Solve[f2, x]

Out[25]= {{x → ConditionalExpression[(-π/4 + 2 π C1)/a, C1 ∈ Z]},
          {x → ConditionalExpression[(3π/4 + 2 π C1)/a, C1 ∈ Z]}}
```

Listing 1

Roots function

This function is designed to determine the roots of a polynomial, it has the form:

Roots[f, x],

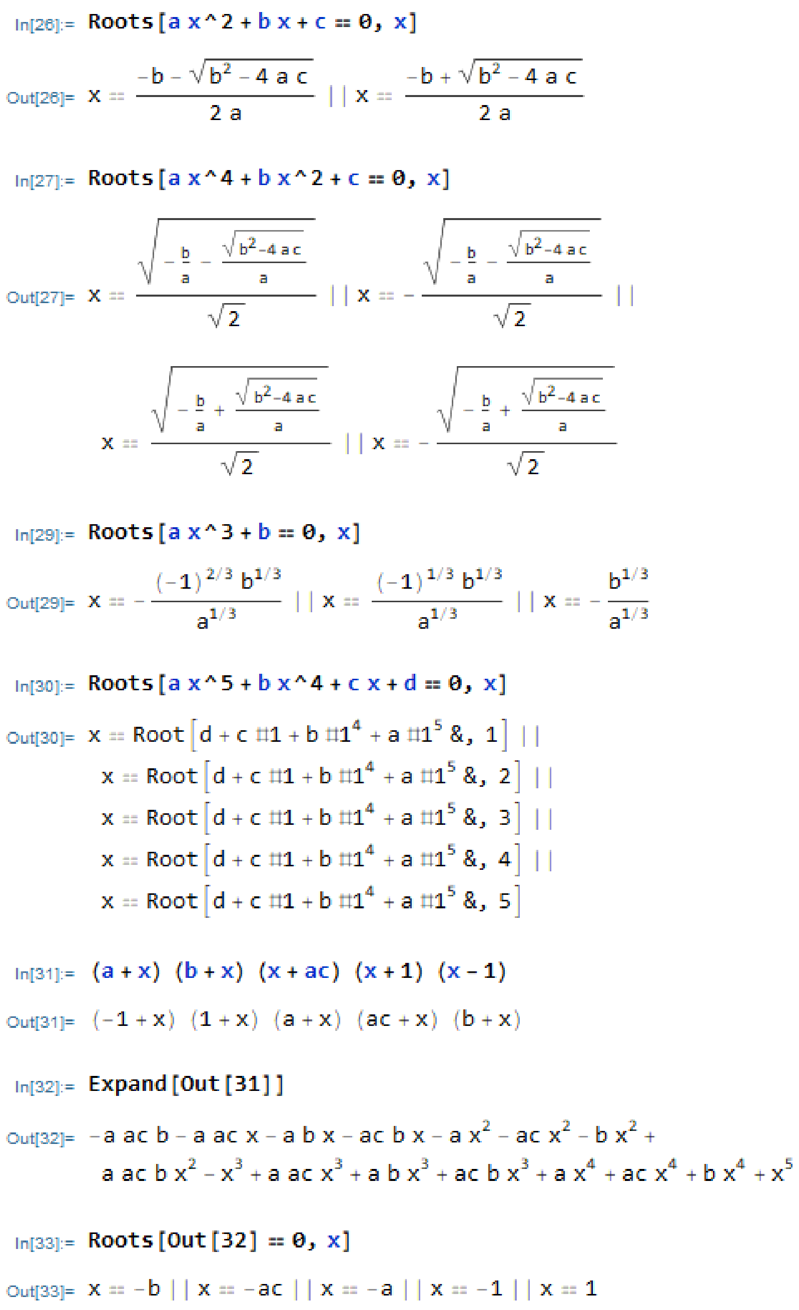
where f – polynomial, the roots of which must be found (can be represented as an equation),

x – polynomial argument.

The result of applying the Roots [f, x] function is the real and complex roots of the equation $f(x)=0$. In this case, the solution can be obtained in analytical and numerical form. The solution in the analytical form in the general case can be obtained for a polynomial not higher than the fourth degree. The solution of $f(x)=0$ does not exist if $f(x)$ – is a polynomial of the fifth and higher degree. However, if the polynomial can be factorized, then the function Roots [f, x] will find all the roots of the corresponding equation.

The solutions of the equations in these cases are shown in the examples in Listing 2 for the equations: $ax^2+bx+c=0$, $ax^4+bx^2+c=0$, $ax^3+b=0$, $ax^5+bx^4+cx+d=0$,

$(a+x)(b+x)(x+ac)(x+1)(x-1)$. Listing 2 shows that in the first three examples, the roots are found in analytical form. The fourth example shows a polynomial of the fifth degree, so its roots are not found. In the last example the polynomial of the fifth degree is given, thus the decision is received in an analytical form. This is because the polynomial $f(x)$ can be factorized. After that, the polynomial brackets were opened using the Expand function, and all its roots were found using the Roots[Out [32], x] function. Out [32] is the line number in which the polynomial is in the open form.



In[26]:= `Roots[a x^2 + b x + c == 0, x]`

Out[26]= $x == \frac{-b-\sqrt{b^2-4\,a\,c}}{2\,a} \;||\; x == \frac{-b+\sqrt{b^2-4\,a\,c}}{2\,a}$

In[27]:= `Roots[a x^4 + b x^2 + c == 0, x]`

Out[27]= $x == \frac{\sqrt{-\frac{b}{a}-\frac{\sqrt{b^2-4ac}}{a}}}{\sqrt{2}} \;||\; x == -\frac{\sqrt{-\frac{b}{a}-\frac{\sqrt{b^2-4ac}}{a}}}{\sqrt{2}} \;||$

$x == \frac{\sqrt{-\frac{b}{a}+\frac{\sqrt{b^2-4ac}}{a}}}{\sqrt{2}} \;||\; x == -\frac{\sqrt{-\frac{b}{a}+\frac{\sqrt{b^2-4ac}}{a}}}{\sqrt{2}}$

In[29]:= `Roots[a x^3 + b == 0, x]`

Out[29]= $x == -\frac{(-1)^{2/3}\,b^{1/3}}{a^{1/3}} \;||\; x == \frac{(-1)^{1/3}\,b^{1/3}}{a^{1/3}} \;||\; x == -\frac{b^{1/3}}{a^{1/3}}$

In[30]:= `Roots[a x^5 + b x^4 + c x + d == 0, x]`

```
Out[30]= x == Root[d + c #1 + b #1^4 + a #1^5 &, 1] ||
          x == Root[d + c #1 + b #1^4 + a #1^5 &, 2] ||
          x == Root[d + c #1 + b #1^4 + a #1^5 &, 3] ||
          x == Root[d + c #1 + b #1^4 + a #1^5 &, 4] ||
          x == Root[d + c #1 + b #1^4 + a #1^5 &, 5]
```

In[31]:= `(a + x) (b + x) (x + ac) (x + 1) (x - 1)`

Out[31]= `(-1 + x) (1 + x) (a + x) (ac + x) (b + x)`

In[32]:= `Expand[Out[31]]`

Out[32]= $-a\,ac\,b - a\,ac\,x - a\,b\,x - ac\,b\,x - a\,x^2 - ac\,x^2 - b\,x^2 +$
$a\,ac\,b\,x^2 - x^3 + a\,ac\,x^3 + a\,b\,x^3 + ac\,b\,x^3 + a\,x^4 + ac\,x^4 + b\,x^4 + x^5$

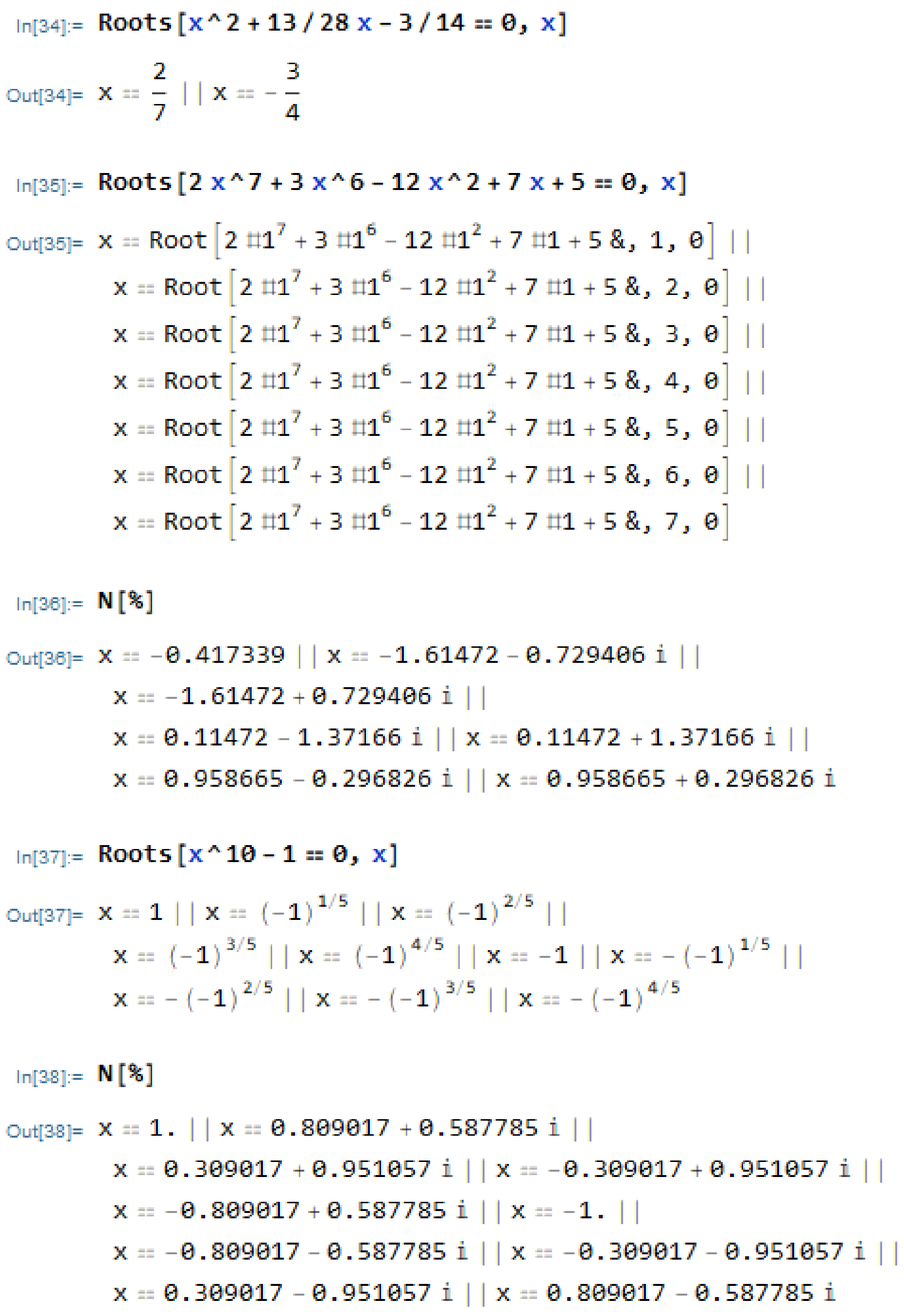
In[33]:= `Roots[Out[32] == 0, x]`

Out[33]= `x == -b || x == -ac || x == -a || x == -1 || x == 1`

Listing 2

If the coefficients of the polynomial are given in the form of numbers, then the function Roots [f, x] gives a solution in the form of an exact or approximate value of the roots. The exact value of the roots is represented by numbers in a rational form, the approximate - in the form of real numbers. Listing 3 shows examples of solving the

$x^2 + 13/28x - 3/14 = 0$, $2x^7 + 3x^6 - 12x^2 + 7x + 5 = 0$, $x^{10} - 1 = 0$. Listing 3 shows that the function determined the exact value of the roots of the first equation, could not explicitly find the roots of the second equation and found the roots of the third equation, but in an inconvenient form for the user: no roots in complex form. To obtain the solution of the second and third equations had to use the command N [%].



```
In[34]:= Roots[x^2 + 13/28 x - 3/14 == 0, x]

Out[34]= x == 2/7 || x == -3/4

In[35]:= Roots[2 x^7 + 3 x^6 - 12 x^2 + 7 x + 5 == 0, x]

Out[35]= x == Root[2 #1^7 + 3 #1^6 - 12 #1^2 + 7 #1 + 5 &, 1, 0] ||
          x == Root[2 #1^7 + 3 #1^6 - 12 #1^2 + 7 #1 + 5 &, 2, 0] ||
          x == Root[2 #1^7 + 3 #1^6 - 12 #1^2 + 7 #1 + 5 &, 3, 0] ||
          x == Root[2 #1^7 + 3 #1^6 - 12 #1^2 + 7 #1 + 5 &, 4, 0] ||
          x == Root[2 #1^7 + 3 #1^6 - 12 #1^2 + 7 #1 + 5 &, 5, 0] ||
          x == Root[2 #1^7 + 3 #1^6 - 12 #1^2 + 7 #1 + 5 &, 6, 0] ||
          x == Root[2 #1^7 + 3 #1^6 - 12 #1^2 + 7 #1 + 5 &, 7, 0]

In[36]:= N[%]

Out[36]= x == -0.417339 || x == -1.61472 - 0.729406 i ||
          x == -1.61472 + 0.729406 i ||
          x == 0.11472 - 1.37166 i || x == 0.11472 + 1.37166 i ||
          x == 0.958665 - 0.296826 i || x == 0.958665 + 0.296826 i

In[37]:= Roots[x^10 - 1 == 0, x]

Out[37]= x == 1 || x == (-1)^(1/5) || x == (-1)^(2/5) ||
          x == (-1)^(3/5) || x == (-1)^(4/5) || x == -1 || x == -(-1)^(1/5) ||
          x == -(-1)^(2/5) || x == -(-1)^(3/5) || x == -(-1)^(4/5)

In[38]:= N[%]

Out[38]= x == 1. || x == 0.809017 + 0.587785 i ||
          x == 0.309017 + 0.951057 i || x == -0.309017 + 0.951057 i ||
          x == -0.809017 + 0.587785 i || x == -1. ||
          x == -0.809017 - 0.587785 i || x == -0.309017 - 0.951057 i ||
          x == 0.309017 - 0.951057 i || x == 0.809017 - 0.587785 i
```

Лістинг 3

Applying the Roots [f, x] function to transcendental equations gives an erroneous result, so its use is irrational for this type of equation.

Numerical methods for solving algebraic and transcendental equations

There are a large number of numerical methods for solving algebraic and transcendental equations. The algorithm of any of these methods is a set of conditions for

choosing the initial approximation, the calculated ratios and signs of the end of the computational process.
The Mathematica system has many built-in functions for solving algebraic and transcendental equations in numerical form. The main ones are: NSolve, NRoots, FindRoot. Consider in detail these functions and give examples.

NSolve function

The NSolve function represends as:

NSolve[f, x],

where f – equation, x – the required unknown.

The result of this function is the roots of the equation . Roots can be real and complex numbers. can solve all equations solved by the Solve function. Its difference is only in the form of answers. The rules for using the NSolve function are shown in Listing 4 when solving the following equations: $x^3+9/4x^2-3/4x+5/16=0$, $2^x-4x+1=0$, $e^{-2x}+3/x-1=0$,
$ax^3-1=0$, $2x^5+3.2x^3-7.3x-14=0$.

NRoots function

The NRoots function represends as:

NRoots[f, x],

where f – equation, x – the required unknown.

The result of using this function is the roots of the polynomial $f(x)=0$. The rules for using the NRoots function are shown in Listing 5 when solving the following equations: $x^2+13/28x-3/14=0$, $2x^7+3x^6-12x^2+5=0$, $x^{10}-1=0$, $2^x-4x+1=0$.
Listing 5 shows that the real and complex roots are found in numerical form. An attempt to solve the transcendental equation did not succeed - no solution was obtained.

FindRoot function

The FindRoot function represends as:

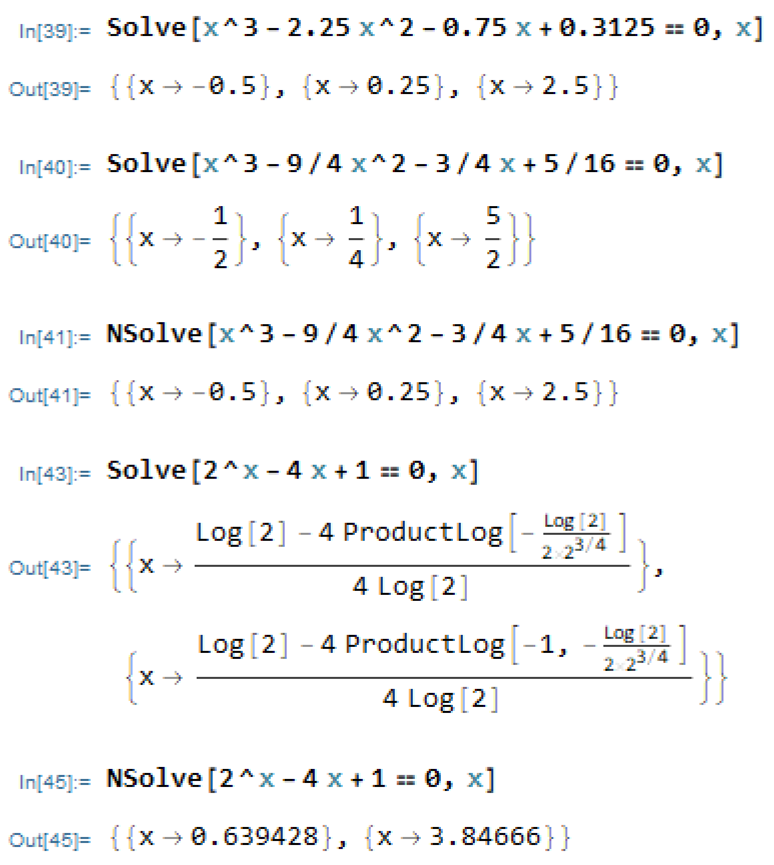
FindRoot[f, {x, x0}],

where f – equation, x – the required unknown (root (roots) of the equation), x_0 – initial approximation.

The FindRoot[f, {x, x0}] finds the root of the equation $f(x)=0$ from the range of x values close to x_0. The following method of determining the roots of algebraic and transcendental equations using a FindRoot[f, {x, x0}] function is recommended:

1. Determining the isolation region of the desired root and selecting the value of the approximationx_0.
2. Input of the equation $f(x)=0$ with assigning it a unique name.
3. Enter the FindRoot[f, {x, x0}] function with the selected value of x_0.
4. Get a solution by pressing <Shift>+<Enter>.

As an example, the sequence of actions when finding the roots of the equation
$3^x - 9x+1=0$ is considered. According to the graph of the function shown in Fig. 1, the first approximations for the roots of the function are chosen.



```
In[39]:= Solve[x^3 - 2.25 x^2 - 0.75 x + 0.3125 == 0, x]

Out[39]= {{x → -0.5}, {x → 0.25}, {x → 2.5}}

In[40]:= Solve[x^3 - 9/4 x^2 - 3/4 x + 5/16 == 0, x]
```

Out[40]= $\left\{\left\{x \to -\frac{1}{2}\right\}, \left\{x \to \frac{1}{4}\right\}, \left\{x \to \frac{5}{2}\right\}\right\}$

```
In[41]:= NSolve[x^3 - 9/4 x^2 - 3/4 x + 5/16 == 0, x]

Out[41]= {{x → -0.5}, {x → 0.25}, {x → 2.5}}

In[43]:= Solve[2^x - 4 x + 1 == 0, x]
```

Out[43]= $\left\{\left\{x \to \frac{\text{Log}[2] - 4\,\text{ProductLog}\left[-\frac{\text{Log}[2]}{2\,2^{3/4}}\right]}{4\,\text{Log}[2]}\right\},\right.$
$\left.\left\{x \to \frac{\text{Log}[2] - 4\,\text{ProductLog}\left[-1, -\frac{\text{Log}[2]}{2\,2^{3/4}}\right]}{4\,\text{Log}[2]}\right\}\right\}$

```
In[45]:= NSolve[2^x - 4 x + 1 == 0, x]

Out[45]= {{x → 0.639428}, {x → 3.84666}}
```

Listing 4 (Part 1 from 2)

```
In[48]:= Solve[E^(-2 x) + 3/x - 1, x]

    ⋯ Solve: -1 + e^-2x + 3/x is not a quantified system of equations and inequalities.

Out[48]= Solve[-1 + e^-2x + 3/x, x]

In[49]:= NSolve[E^(-2 x) + 3/x - 1, x]

    ⋯ NSolve: This system cannot be solved with the methods available to NSolve.

Out[49]= NSolve[-1 + e^-2x + 3/x, x]

In[50]:= Solve[a x^3 - 1 == 0, x]

Out[50]= {{x → 1/a^(1/3)}, {x → -(-1)^(1/3)/a^(1/3)}, {x → (-1)^(2/3)/a^(1/3)}}

In[51]:= NSolve[a x^3 - 1 == 0, x]

Out[51]= {{x → 1/a^(1/3)}, {x → -(0.5 + 0.866025 i)/a^(1/3)}, {x → -(0.5 - 0.866025 i)/a^(1/3)}}

In[52]:= Solve[2 x^5 - 3.2 x^3 + 7.3 x - 14 == 0, x]

Out[52]= {{x → -1.43009 - 0.904276 i},
          {x → -1.43009 + 0.904276 i}, {x → 0.699575 - 1.08818 i},
          {x → 0.699575 + 1.08818 i}, {x → 1.46103}}

In[53]:= NSolve[2 x^5 - 3.2 x^3 + 7.3 x - 14 == 0, x]

Out[53]= {{x → -1.43009 - 0.904276 i},
          {x → -1.43009 + 0.904276 i}, {x → 0.699575 - 1.08818 i},
          {x → 0.699575 + 1.08818 i}, {x → 1.46103}}
```

Listing 4 (Part 2 from 2)

```
In[54]:= NRoots[x^2 + 13 / 28 x - 3 / 14 == 0, x]

Out[54]= x == -0.75 || x == 0.285714

In[55]:= NRoots[2 x^7 + 3 x^6 - 12 x^2 + 7 x + 5 == 0, x]

Out[55]= x == -1.61472 - 0.729406 i ||
          x == -1.61472 + 0.729406 i || x == -0.417339 ||
          x == 0.11472 - 1.37166 i || x == 0.11472 + 1.37166 i ||
          x == 0.958665 - 0.296826 i || x == 0.958665 + 0.296826 i

In[56]:= NRoots[x^10 - 1 == 0, x]

Out[56]= x == -1. || x == -0.809017 - 0.587785 i ||
          x == -0.809017 + 0.587785 i || x == -0.309017 - 0.951057 i ||
          x == -0.309017 + 0.951057 i || x == 0.309017 - 0.951057 i ||
          x == 0.309017 + 0.951057 i || x == 0.809017 - 0.587785 i ||
          x == 0.809017 + 0.587785 i || x == 1.

In[57]:= NRoots[2^x - 4 x + 1 == 0, x]

         NRoots: 1 + 2^x - 4x == 0 is expected to be a polynomial equation in the variable x with
           numeric coefficients.

Out[57]= NRoots[1 + 2^x - 4 x == 0, x]
```

Listing 5

```
In[58]:= Plot[3^x - 9 x + 1, {x, 0, 4}]
```

Out[58]=

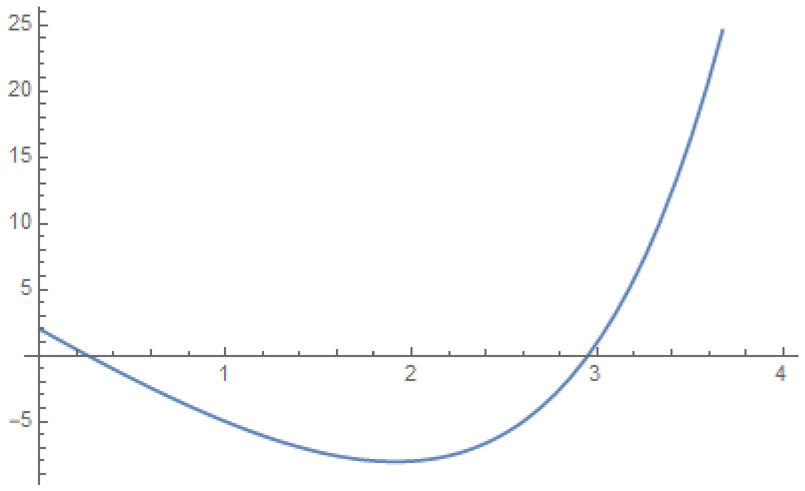
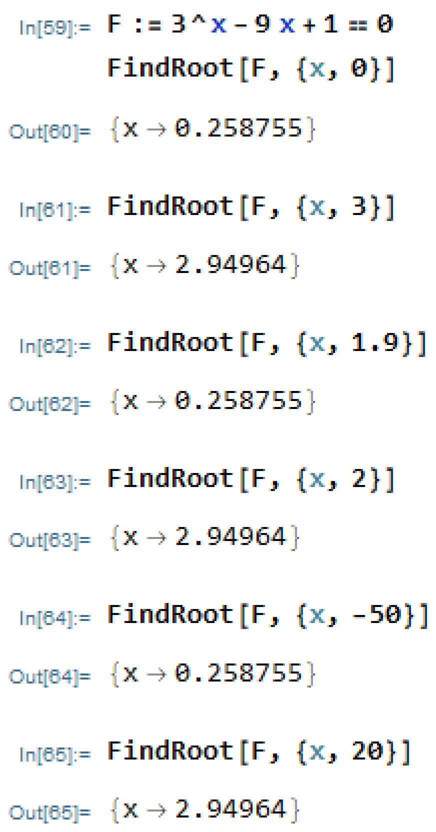


Figure1. Graph of function $f(x) = 3^x - 9x + 1$

Fig. 1 shows that the equation has two roots, the isolation regions of which can be the intervals x$_0$ – [0, 1] and x$_0$ – [2.5, 3]. he solution of the equation is shown in Listing 6. Care should be taken when choosing the initial approximation, especially in cases where the equation contains several roots. It may turn out that when the user-set approximation x$_0$ is determined, the wrong root will be determined. In our example, a small change in x$_0$ ed to a different solution: at x$_0$=1.9 FindRoot function found the root x =0.258755, , and at $x_0=2$ – x=2.94964.



```
In[59]:= F := 3^x - 9 x + 1 == 0
         FindRoot[F, {x, 0}]

Out[60]= {x → 0.258755}

In[61]:= FindRoot[F, {x, 3}]

Out[61]= {x → 2.94964}

In[62]:= FindRoot[F, {x, 1.9}]

Out[62]= {x → 0.258755}

In[63]:= FindRoot[F, {x, 2}]

Out[63]= {x → 2.94964}

In[64]:= FindRoot[F, {x, -50}]

Out[64]= {x → 0.258755}

In[65]:= FindRoot[F, {x, 20}]

Out[65]= {x → 2.94964}
```

Listing 6

Methods for solving systems of equations in the system Mathematica

The Mathematica system has rich possibilities for solving systems of algebraic equations. Built-in functions allow solving systems of linear and nonlinear equations in analytical and numerical form. Give the opportunity to check the reliability of the results quite effectively and in an original way. During operation, the system issues comments that allow the user to make decisions about the answers received. The main functions for solving systems of equations are: Solve[F,X], Solve[F,X,Y], N[Solve[F,X]], FindRoot[F,X]. Consider these functions, describe the technology of their implementation, give examples and problems for independent solution.

Solve[F,X] function

The Solve[F, X] function allows solving systems of linear and nonlinear equations in analytical form. It is represends as follows:

Solve[{f_1,f_2,...}, {x_1,x_2,...}]

where f_i – *i*-th equation presented in any form, x_i – *i*-th unknown.

Equations f_1, f_2, … an also be represented by a unifying sign &&. Examples of function Solve[F,X] representation:

Solve[{a*x^2+y==b,x+2*y==a+b},{x,y}]
Solve[a*x^2+y==b&&x+2*y==a+b,{x,y}]

When entering equations, the multiplication sign (*) can be replaced by pressing the <Space> key. When solving practical problems, it is convenient, and in some cases even advisable, to enter equations separately from the Solve function, assigning them names, which are then entered into the Solve function instead of equations. Example:

f$_1$=a*x^2+y==b; f$_2$=x+2*y==a+b
Solve[{f$_1$, f$_2$},{x,y}]
or
Solve[f$_1$&&f$_2$,{x,y}]
This form of notation simplifies the verification of the solution of the system of equations.

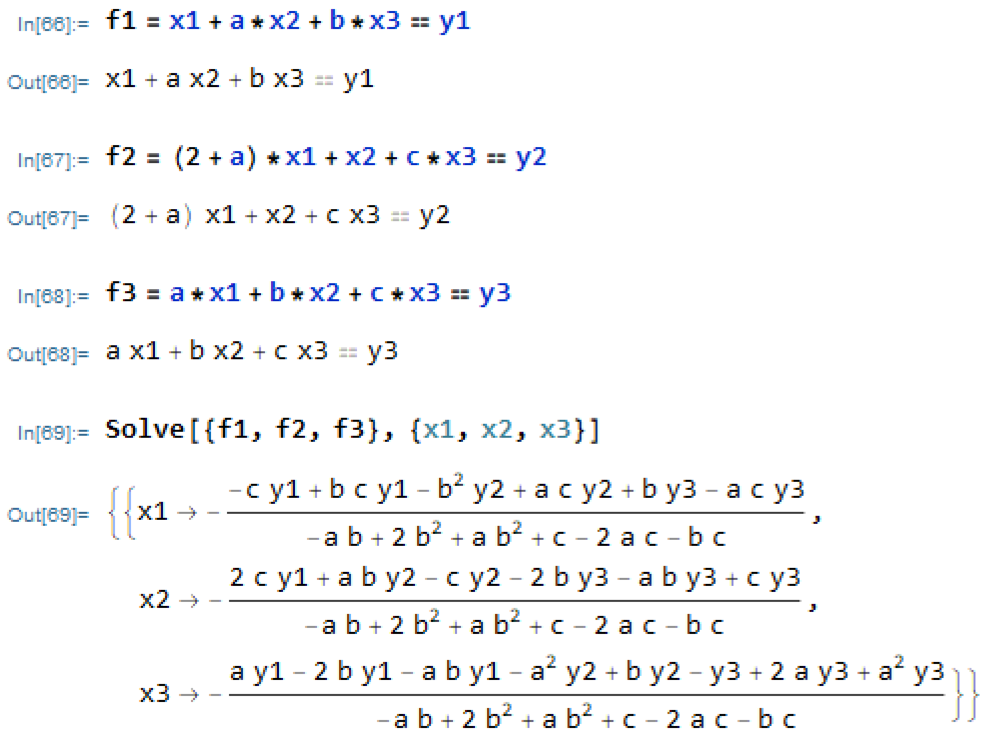
Systems of equations

Methods for solving equations:

1. Entering equations with a unique name, which is specified by the assignment sign (=).
2. Write the function Solve[{f$_1$,f$_2$,...},{x,y,...}] або Solve [f$_1$&&f$_2$&&...,{x,y,...}].
3. Checking the validity of the solution of the system of equations.

Examples of solving linear algebraic equations are shown in Listing 7.
It is necessary to solve the following systems of linear equations:

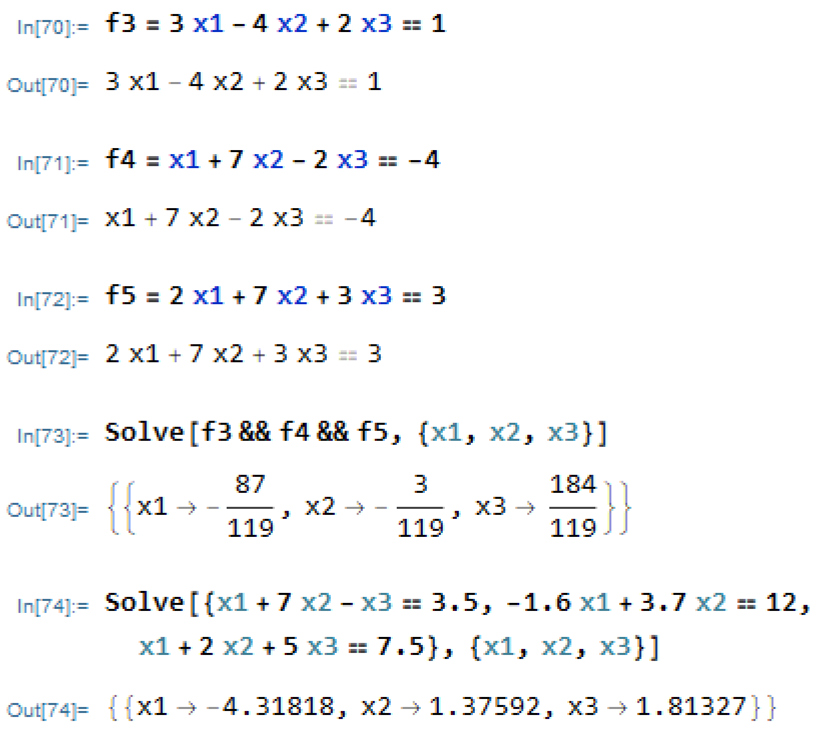
$$\begin{cases} x_1 + ax_2 - bx_3 = y_1; \\ (2+a)x_1 + x_2 + cx_3 = y_2; \\ ax_1 + bx_2 + cx_3 = y_3. \end{cases} \quad \begin{cases} 3x_1 - 4x_2 + 2x_3 = 1; \\ x_1 + 7x_2 - 2x_3 = -4; \\ 2x_1 + 7x_2 + 3x_3 = 3. \end{cases} \quad \begin{cases} x_1 + 7x_2 - x_3 = 3.5; \\ -1.6x_1 + 3.7x_2 = 12; \\ x_1 + 2x_2 + 5x_3 = 7.5. \end{cases}$$



```
In[66]:= f1 = x1 + a * x2 + b * x3 == y1

Out[66]= x1 + a x2 + b x3 == y1

In[67]:= f2 = (2 + a) * x1 + x2 + c * x3 == y2

Out[67]= (2 + a) x1 + x2 + c x3 == y2

In[68]:= f3 = a * x1 + b * x2 + c * x3 == y3

Out[68]= a x1 + b x2 + c x3 == y3

In[69]:= Solve[{f1, f2, f3}, {x1, x2, x3}]
```

Out[69]=
$$\left\{\left\{x1 \to -\frac{-c\ y1 + b\ c\ y1 - b^2\ y2 + a\ c\ y2 + b\ y3 - a\ c\ y3}{-a\ b + 2\ b^2 + a\ b^2 + c - 2\ a\ c - b\ c}, \right.\right.$$
$$x2 \to -\frac{2\ c\ y1 + a\ b\ y2 - c\ y2 - 2\ b\ y3 - a\ b\ y3 + c\ y3}{-a\ b + 2\ b^2 + a\ b^2 + c - 2\ a\ c - b\ c},$$
$$\left.\left. x3 \to -\frac{a\ y1 - 2\ b\ y1 - a\ b\ y1 - a^2\ y2 + b\ y2 - y3 + 2\ a\ y3 + a^2\ y3}{-a\ b + 2\ b^2 + a\ b^2 + c - 2\ a\ c - b\ c}\right\}\right\}$$

Listing 7 (Part 1 from 2)



```
In[70]:= f3 = 3 x1 - 4 x2 + 2 x3 == 1

Out[70]= 3 x1 - 4 x2 + 2 x3 == 1

In[71]:= f4 = x1 + 7 x2 - 2 x3 == -4

Out[71]= x1 + 7 x2 - 2 x3 == -4

In[72]:= f5 = 2 x1 + 7 x2 + 3 x3 == 3

Out[72]= 2 x1 + 7 x2 + 3 x3 == 3

In[73]:= Solve[f3 && f4 && f5, {x1, x2, x3}]

                     87            3           184
Out[73]= {{x1 -> - ---, x2 -> - ---, x3 -> ---}}
                    119           119          119

In[74]:= Solve[{x1 + 7 x2 - x3 == 3.5, -1.6 x1 + 3.7 x2 == 12,
            x1 + 2 x2 + 5 x3 == 7.5}, {x1, x2, x3}]

Out[74]= {{x1 -> -4.31818, x2 -> 1.37592, x3 -> 1.81327}}
```

Code Listing 7 (Part 2 of 2)

Listing 7 shows that the system solved the first system of equations in analytical form, the second - in the form of exact values of the unknowns, presented in rational form. The third system of equations is also solved, but the solution is represented as real numbers. This is due to the fact that the system of equations has no exact solution. Thus, the Solve function can solve systems of equations also in numerical form. The example shows that the solution is represented in the form of substitutions: x1->, x2->, x3->. This does not make it possible to verify the validity of the solution, as well as to use the values of x1, x2, x3 in further calculations.

To verify the validity of the solution, the user must present the unknown explicitly with their name: x1 = -4.31818, x2 = 1.37592, x3 = 1.81327, then use them for their intended purpose. You can get the solution explicitly using an expression of the form {x1, x2, x3} /., Which is placed before the Solve function: {x1, x2, x3} /. Solve {f1, f2, f3}, {x1, x2, x3 }]. Now x1, x2, x3 can be used for its intended purpose, including to verify the validity of the solution of the system of equations.

Systems of nonlinear algebraic equations

The method of solving systems of nonlinear equations is the same as linear ones. This is demonstrated by the example of solving the following systems of nonlinear equations:

$$\begin{cases} x + ay = b; \\ x + by^2 = a + b. \end{cases} \qquad \begin{cases} xy = a; \\ x^2 + y^2 = ab. \end{cases}$$

A record of the solution and its validation is shown in Listing 8.

```
In[75]:= f1 = x + a y == b

Out[75]= x + a y == b

In[76]:= f2 = x + b y^2 == a + b
```

Out[76]= $x + b\, y^2 == a + b$

```
In[77]:= w = Solve[{f1, f2}, {x, y}]
```

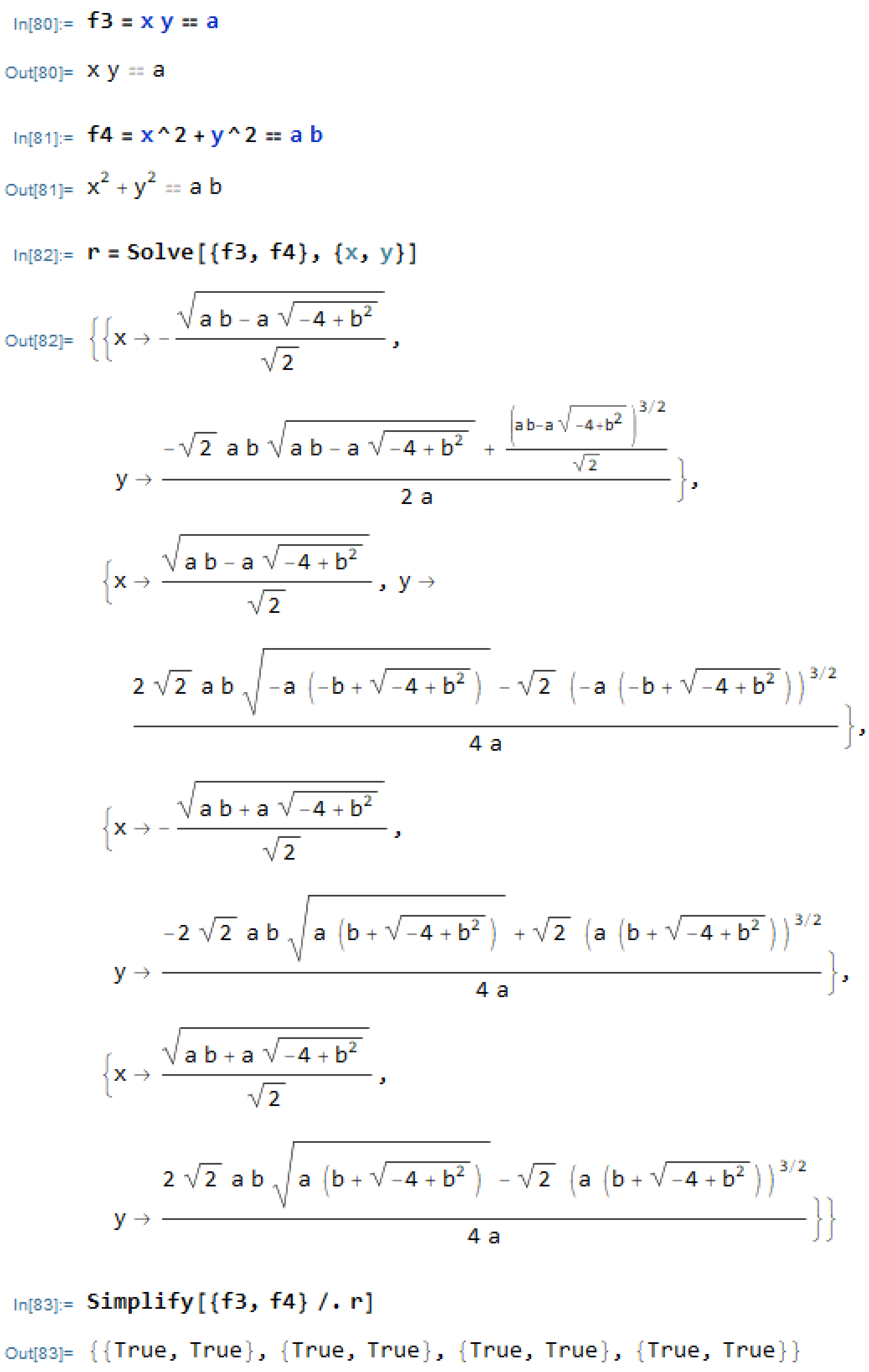
Out[77]=
$$\left\{\left\{x \to \frac{1}{2}\left(-\frac{a^2}{b} + 2\,b + \frac{a^{3/2}\sqrt{a+4\,b}}{b}\right),\; y \to \frac{a - \sqrt{a}\,\sqrt{a+4\,b}}{2\,b}\right\},\right.$$
$$\left.\left\{x \to \frac{1}{2}\left(-\frac{a^2}{b} + 2\,b - \frac{a^{3/2}\sqrt{a+4\,b}}{b}\right),\; y \to \frac{a + \sqrt{a}\,\sqrt{a+4\,b}}{2\,b}\right\}\right\}$$

```
In[78]:= {f1, f2} /. w
```

Out[78]=
$$\left\{\left\{\frac{a\left(a - \sqrt{a}\,\sqrt{a+4\,b}\right)}{2\,b} + \frac{1}{2}\left(-\frac{a^2}{b} + 2\,b + \frac{a^{3/2}\sqrt{a+4\,b}}{b}\right) == b,\right.\right.$$
$$\left.\frac{\left(a - \sqrt{a}\,\sqrt{a+4\,b}\right)^2}{4\,b} + \frac{1}{2}\left(-\frac{a^2}{b} + 2\,b + \frac{a^{3/2}\sqrt{a+4\,b}}{b}\right) == a + b\right\},$$
$$\left\{\frac{a\left(a + \sqrt{a}\,\sqrt{a+4\,b}\right)}{2\,b} + \frac{1}{2}\left(-\frac{a^2}{b} + 2\,b - \frac{a^{3/2}\sqrt{a+4\,b}}{b}\right) == b,\right.$$
$$\left.\left.\frac{\left(a + \sqrt{a}\,\sqrt{a+4\,b}\right)^2}{4\,b} + \frac{1}{2}\left(-\frac{a^2}{b} + 2\,b - \frac{a^{3/2}\sqrt{a+4\,b}}{b}\right) == a + b\right\}\right\}$$

```
In[79]:= Simplify[%]

Out[79]= {{True, True}, {True, True}}
```

Code Listing 8 (Part 1 of 2)



```
In[80]:= f3 = x y == a
Out[80]= x y == a

In[81]:= f4 = x^2 + y^2 == a b
```

Out[81]= $x^2 + y^2 == a\,b$

```
In[82]:= r = Solve[{f3, f4}, {x, y}]
```

Out[82]=

$$\left\{\left\{x \to -\frac{\sqrt{a\,b - a\sqrt{-4 + b^2}}}{\sqrt{2}},\; y \to \frac{-\sqrt{2}\;a\,b\,\sqrt{a\,b - a\sqrt{-4 + b^2}} + \frac{\left(a\,b - a\sqrt{-4 + b^2}\right)^{3/2}}{\sqrt{2}}}{2\,a}\right\},\right.$$

$$\left\{x \to \frac{\sqrt{a\,b - a\sqrt{-4 + b^2}}}{\sqrt{2}},\; y \to \frac{2\sqrt{2}\;a\,b\,\sqrt{-a\left(-b + \sqrt{-4 + b^2}\right)} - \sqrt{2}\;\left(-a\left(-b + \sqrt{-4 + b^2}\right)\right)^{3/2}}{4\,a}\right\},$$

$$\left\{x \to -\frac{\sqrt{a\,b + a\sqrt{-4 + b^2}}}{\sqrt{2}},\; y \to \frac{-2\sqrt{2}\;a\,b\,\sqrt{a\left(b + \sqrt{-4 + b^2}\right)} + \sqrt{2}\;\left(a\left(b + \sqrt{-4 + b^2}\right)\right)^{3/2}}{4\,a}\right\},$$

$$\left.\left\{x \to \frac{\sqrt{a\,b + a\sqrt{-4 + b^2}}}{\sqrt{2}},\; y \to \frac{2\sqrt{2}\;a\,b\,\sqrt{a\left(b + \sqrt{-4 + b^2}\right)} - \sqrt{2}\;\left(a\left(b + \sqrt{-4 + b^2}\right)\right)^{3/2}}{4\,a}\right\}\right\}$$

```
In[83]:= Simplify[{f3, f4} /. r]
Out[83]= {{True, True}, {True, True}, {True, True}, {True, True}}
```

Code Listing 8 (Part 2 of 2)

Solve function [F, X, Y]

The Solve function [F, X, Y], as well as the Solve function [F, X], allows to solve systems of linear and nonlinear equations in an analytical form, but only with restriction: solutions are carried out on variable X and are excluded. for variables Y. For example, the function Solve [{x + 2 * ya == 3, 2 * x + y ^ 2 + b == 7}, x, y] will determine X and exclude from the solution Y. Using the function Solve [F, X, Y] is shown in Listing 9 in solving the following systems of equations:

$$\begin{cases} ax^2 + bxy = 1; \\ xy + c = 7. \end{cases} \qquad \begin{cases} 7a + 3b + c = 1; \\ -5a + 12b = 3; \\ a + b + 2c = -7. \end{cases}$$

The first system is solved with respect to x, the second with respect to a, then with respect to a and b.

```
In[84]:= f1 = a x^2 + b x y == 1
```

Out[84]= $a\,x^2 + b\,x\,y == 1$

```
In[85]:= f2 = x y + c == 7
```

Out[85]= $c + x\,y == 7$

```
In[89]:= z = Solve[{f1, f2}, x, y]
```

Out[89]= $\left\{\left\{x \to -\frac{\sqrt{1 - 7\,b + b\,c}}{\sqrt{a}}\right\}, \left\{x \to \frac{\sqrt{1 - 7\,b + b\,c}}{\sqrt{a}}\right\}\right\}$

```
In[90]:= x /. z
```

Out[90]= $\left\{-\frac{\sqrt{1 - 7\,b + b\,c}}{\sqrt{a}}, \frac{\sqrt{1 - 7\,b + b\,c}}{\sqrt{a}}\right\}$

```
In[91]:= f3 = 7 a + 3 b + c == 1
```

Out[91]= $7\,a + 3\,b + c == 1$

```
In[92]:= f4 = -5 a + 12 b == 3
```

Out[92]= $-5\,a + 12\,b == 3$

```
In[93]:= f5 = a + b + 2 c == -7
```

Out[93]= $a + b + 2\,c == -7$

```
In[96]:= s = Solve[{f3, f4, f5}, {a, b, c}]
```

Out[96]= $\left\{\left\{a \to \frac{93}{181}, b \to \frac{84}{181}, c \to -\frac{722}{181}\right\}\right\}$

```
In[97]:= s1 = Solve[{f3, f4, f5}, a, {b, c}]
```

Out[97]= $\left\{\left\{a \to \frac{93}{181}\right\}\right\}$

```
In[98]:= a /. s1
```

Out[98]= $\left\{\frac{93}{181}\right\}$

```
In[99]:= s2 = Solve[{f3, f4, f5}, {a, b}, c]
```

Out[99]= $\left\{\left\{a \to \frac{93}{181}, b \to \frac{84}{181}\right\}\right\}$

```
In[100]:= {a, b} /. s2
```

Out[100]= $\left\{\left\{\frac{93}{181}, \frac{84}{181}\right\}\right\}$

Listing 9

NSolve function [F, X]

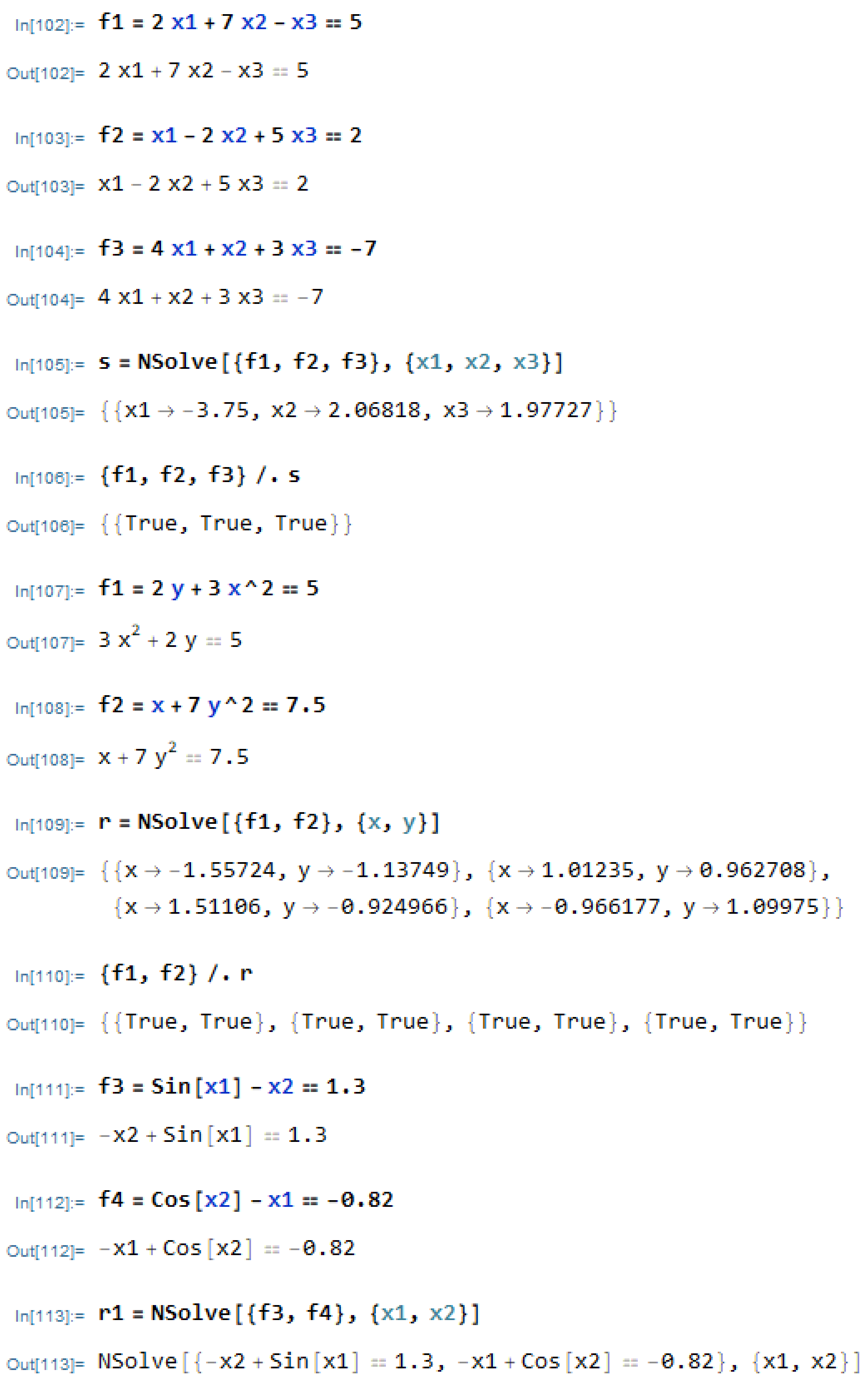
The NSolve function [F, X] allows solving systems of linear and nonlinear equations in numerical form. It is recorded as follows:

Solve [{f1, f2, ...}, {x1, x2, ...}],

where fi is the i-th equation represented in an arbitrary form, xi is the i-th unknown. Equations f1, f2,… can also be represented by the unifying sign &&. The method of solving systems of equations using the function NSolve [F, X] is almost no different from the technology of solving using the function Solve {F, X}. With its help in Listing 10 the following systems of equations are solved:

$$\begin{cases} 2x_1 + 7x_2 - x_3 = 5; \\ x_1 - 2x_2 + 5x_3 = 2; \\ 4x_1 + x_2 + 3x_3 = -7. \end{cases} \qquad \begin{cases} 2y + 3x^2 = 5; \\ x + 7y^2 = 7.5. \end{cases} \qquad \begin{cases} \sin x_1 - x_2 = 1.3 \\ \cos x_2 - x_1 = -0.82. \end{cases}$$

Listing 10 shows that the NSolve function [F, X] did not solve the last system of equations. This is because the system consists of transcendental equations. To solve it, methods are needed that are not implemented in the NSolve function [F, X].



```
In[102]:= f1 = 2 x1 + 7 x2 - x3 == 5
Out[102]= 2 x1 + 7 x2 - x3 == 5

In[103]:= f2 = x1 - 2 x2 + 5 x3 == 2
Out[103]= x1 - 2 x2 + 5 x3 == 2

In[104]:= f3 = 4 x1 + x2 + 3 x3 == -7
Out[104]= 4 x1 + x2 + 3 x3 == -7

In[105]:= s = NSolve[{f1, f2, f3}, {x1, x2, x3}]
Out[105]= {{x1 → -3.75, x2 → 2.06818, x3 → 1.97727}}

In[106]:= {f1, f2, f3} /. s
Out[106]= {{True, True, True}}

In[107]:= f1 = 2 y + 3 x^2 == 5
Out[107]= 3 x^2 + 2 y == 5

In[108]:= f2 = x + 7 y^2 == 7.5
Out[108]= x + 7 y^2 == 7.5

In[109]:= r = NSolve[{f1, f2}, {x, y}]
Out[109]= {{x → -1.55724, y → -1.13749}, {x → 1.01235, y → 0.962708},
           {x → 1.51106, y → -0.924966}, {x → -0.966177, y → 1.09975}}

In[110]:= {f1, f2} /. r
Out[110]= {{True, True}, {True, True}, {True, True}, {True, True}}

In[111]:= f3 = Sin[x1] - x2 == 1.3
Out[111]= -x2 + Sin[x1] == 1.3

In[112]:= f4 = Cos[x2] - x1 == -0.82
Out[112]= -x1 + Cos[x2] == -0.82

In[113]:= r1 = NSolve[{f3, f4}, {x1, x2}]
Out[113]= NSolve[{-x2 + Sin[x1] == 1.3, -x1 + Cos[x2] == -0.82}, {x1, x2}]
```

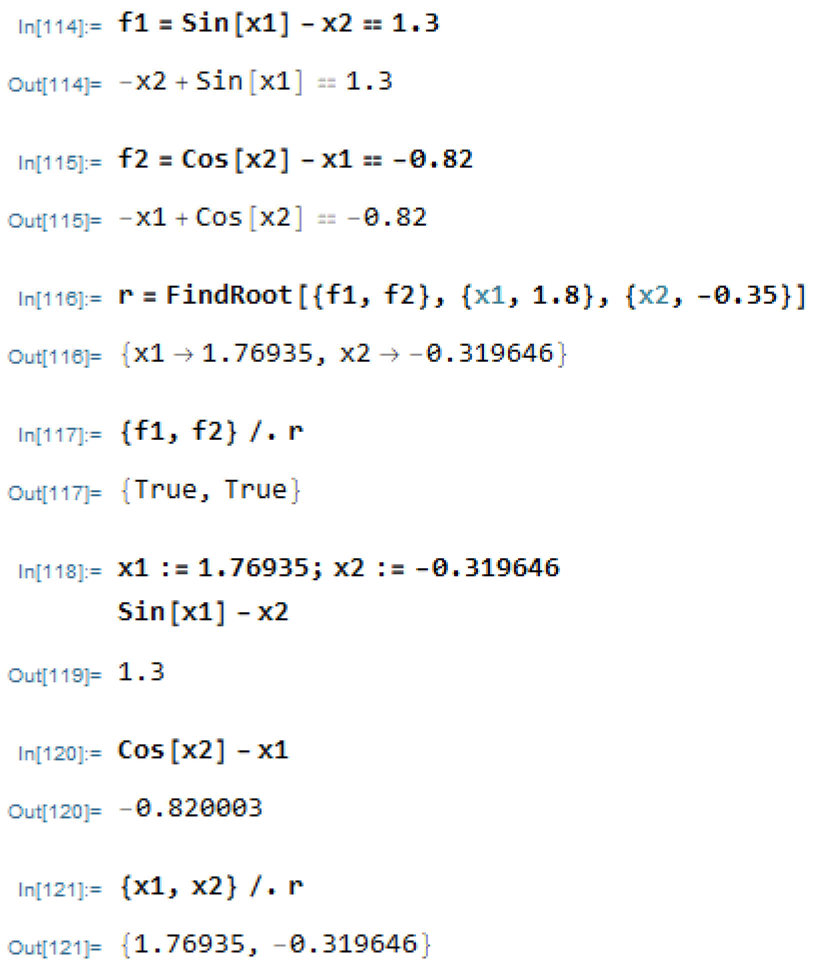
Listing 10

FindRoot function [F, {X, x0}]

The FindRoot function [F, {X, x0}] solves systems of linear and nonlinear equations by numerical iteration methods. To implement it, you need to know the initial approximations of the unknown. The function looks like:

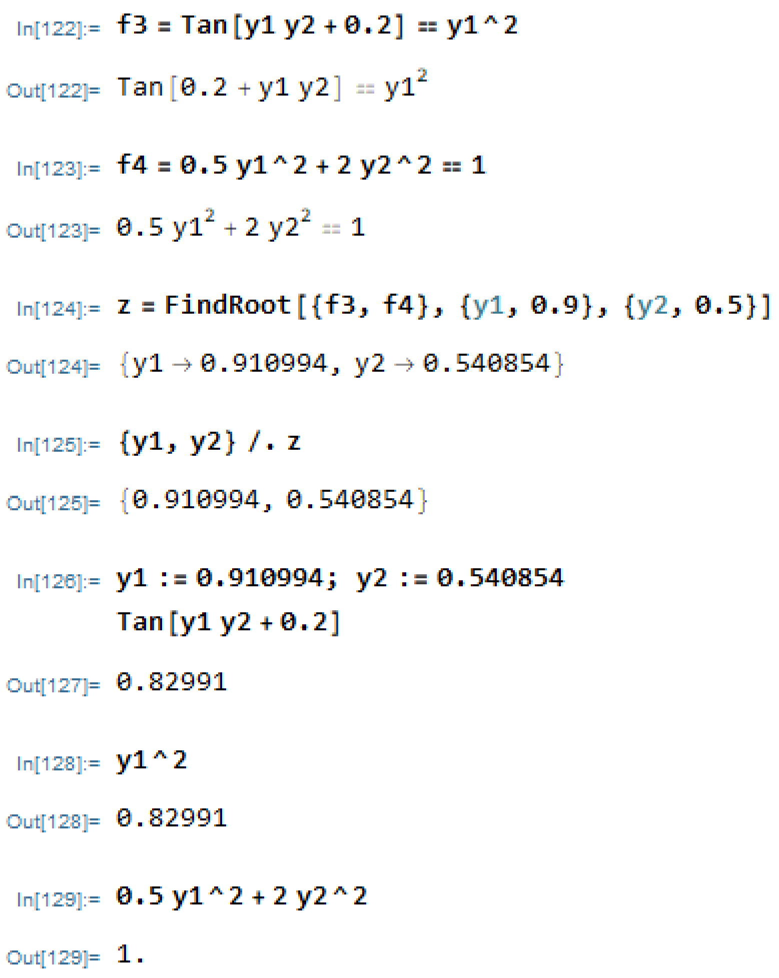
FindRoot [{f1, f2, ...}, {x1, x10}, {x2, x20}, ...],
Where fi is the i-th equation represented in an arbitrary form, xi is the i-th unknown, xi0 is the initial approximation of the i-th unknown.
Equations f1, f2,… can also be represented by the unifying sign &&.
The technique of solving equation systems using the FindRoot function [F, {X, x0}] differs significantly from the technology of solving equations using the NSolve function [F, X]. The difference is the need to determine the initial approximations. Examples of solving systems of equations by the FindRoot function [F, {X, x0}] are shown in Listing 11. Listing 11 solves the following systems of equations:

$$\begin{cases} \sin x_1 - x_2 = 1.3; \\ \cos x_2 - x_1 = -0.82; \\ x_{10} = 1.8; \\ x_{20} = -0.35. \end{cases} \qquad \begin{cases} tg(y_1 y_2 + 0.2) = y_1^2; \\ 0.5 y_1^2 + 2 y_2^2 = 1; \\ y_{10} = 0.9; \\ y_{20} = 0.5. \end{cases}$$



```
In[114]:= f1 = Sin[x1] - x2 == 1.3

Out[114]= -x2 + Sin[x1] == 1.3

In[115]:= f2 = Cos[x2] - x1 == -0.82

Out[115]= -x1 + Cos[x2] == -0.82

In[116]:= r = FindRoot[{f1, f2}, {x1, 1.8}, {x2, -0.35}]

Out[116]= {x1 → 1.76935, x2 → -0.319646}

In[117]:= {f1, f2} /. r

Out[117]= {True, True}

In[118]:= x1 := 1.76935; x2 := -0.319646
          Sin[x1] - x2

Out[119]= 1.3

In[120]:= Cos[x2] - x1

Out[120]= -0.820003

In[121]:= {x1, x2} /. r

Out[121]= {1.76935, -0.319646}
```

Code Listing 11 (Part 1 of 2)



```
In[122]:= f3 = Tan[y1 y2 + 0.2] == y1^2

Out[122]= Tan[0.2 + y1 y2] == y1^2

In[123]:= f4 = 0.5 y1^2 + 2 y2^2 == 1

Out[123]= 0.5 y1^2 + 2 y2^2 == 1

In[124]:= z = FindRoot[{f3, f4}, {y1, 0.9}, {y2, 0.5}]

Out[124]= {y1 → 0.910994, y2 → 0.540854}

In[125]:= {y1, y2} /. z

Out[125]= {0.910994, 0.540854}

In[126]:= y1 := 0.910994; y2 := 0.540854
          Tan[y1 y2 + 0.2]

Out[127]= 0.82991

In[128]:= y1^2

Out[128]= 0.82991

In[129]:= 0.5 y1^2 + 2 y2^2

Out[129]= 1.
```

Code Listing 11 (Part 2 of 2)

Eliminate function [F, x]

The Eliminate function [F, x] is designed to reduce the number of system equations by excluding the specified variables x. It looks like:
Eliminate [{f1, f2, ...}, {x1, x2,…}],
where fi is the i-th equation, presented in any form,
xi is the i-th unknown to be excluded.
Equations f1, f2, ... can also be represented by the joining sign &&.
This function transforms the original system of equations so that the number of equations and variables is reduced. The limit is one equation with one unknown. An example of using the Eliminate function is shown in Listing 12, in which a system of three equations is alternately reduced to a system of two equations and then to one equation.

$$\begin{cases} x^2 + ay^2 - z = 2; \\ x + y - bz = 7; \\ 2x + 3y + 9z = 1. \end{cases}$$

```
In[135]:= f1 = x^2 + a y^2 - z == 2

Out[135]= x^2 + a y^2 - z == 2

In[136]:= f2 = x + y - b z == 7

Out[136]= x + y - b z == 7

In[137]:= f3 = 2 x + 3 y + 9 z == 1

Out[137]= 2 x + 3 y + 9 z == 1

In[138]:= Eliminate[{f1, f2, f3}, z]

Out[138]= 9 a y^2 == 19 - 2 x - 9 x^2 - 3 y && b (-1 + 2 x + 3 y) == 63 - 9 x - 9 y

In[139]:= Eliminate[{f1, f2, f3}, {y, z}]

Out[139]= b^2 (-18 + a - 4 a x + 9 x^2 + 4 a x^2) +
            b (-48 + 126 a - 3 x - 270 a x + 54 x^2 + 36 a x^2) ==
           -18 - 3969 a + 9 x + 1134 a x - 81 x^2 - 81 a x^2

In[140]:= FullSimplify[Out[139]]

Out[140]= a (63 + b - 9 x - 2 b x)^2 + 3 (3 + b) (2 + x (-1 + 9 x) + 3 b (-2 + x^2)) == 0
```

Listing 12

Matrix methods for solving systems of linear equations

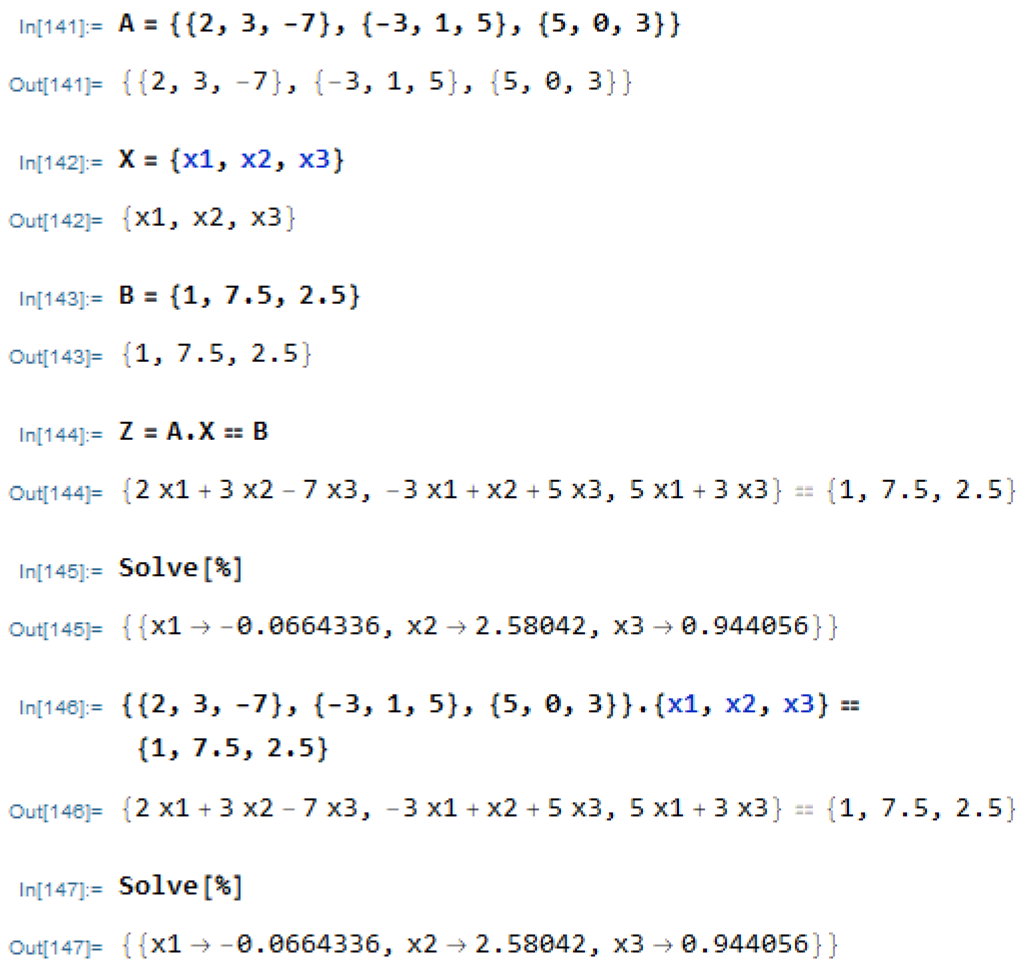
The system of linear algebraic equations can be represented as follows: A * X = B, where A is a matrix of coefficients, X is a vector of unknowns, B is a vector of free members (right parts) of the system of equations. The method of solving equations in the Mathematica system is simple and consists of the following:

1. Introduction of a matrix of coefficients with the assignment of a name, such as A.
2. Enter a vector of unknowns named X.
3. Introduction of a vector of free members named B.
4. Creating the expression z = A. X == B.
5. Introduction of the Solve function [z, X].

An example of solving a system of linear equations

$$\begin{cases} 2x_1 + 3x_2 - 7x_3 = 1; \\ -3x_1 + x_2 + 5x_3 = 7.5; \\ 5x_1 + 3x_3 = 2.5, \end{cases}$$

the matrix method is shown in Listing 13.



```
In[141]:= A = {{2, 3, -7}, {-3, 1, 5}, {5, 0, 3}}

Out[141]= {{2, 3, -7}, {-3, 1, 5}, {5, 0, 3}}

In[142]:= X = {x1, x2, x3}

Out[142]= {x1, x2, x3}

In[143]:= B = {1, 7.5, 2.5}

Out[143]= {1, 7.5, 2.5}

In[144]:= Z = A.X == B

Out[144]= {2 x1 + 3 x2 - 7 x3, -3 x1 + x2 + 5 x3, 5 x1 + 3 x3} == {1, 7.5, 2.5}

In[145]:= Solve[%]

Out[145]= {{x1 → -0.0664336, x2 → 2.58042, x3 → 0.944056}}

In[146]:= {{2, 3, -7}, {-3, 1, 5}, {5, 0, 3}}.{x1, x2, x3} ==
           {1, 7.5, 2.5}

Out[146]= {2 x1 + 3 x2 - 7 x3, -3 x1 + x2 + 5 x3, 5 x1 + 3 x3} == {1, 7.5, 2.5}

In[147]:= Solve[%]

Out[147]= {{x1 → -0.0664336, x2 → 2.58042, x3 → 0.944056}}
```

Listing 13

In addition to the above, there are the following two matrix methods for solving systems of algebraic equations in the Mathematica system.

Method 1. Determination of the vector of unknown X by the formula: X = A-1B.

The multiplication operation is written by the Dot function, and the matrix inversion operation is written by the Inverse function. Then the decision is written as follows:

X: = Dot [Inverse [A], B]

Method 2. Using the LinearSolve function.

The LinearSolve function is written as follows:

X: = LinearSolve [A, B]

Listing 14 shows an example of solving a system of linear equations

$$\begin{cases} a_1x_1 + 2x_2 - 3x_3 = b_1; \\ -7x_1 + cx_2 + x_3 = b_2; \\ x_1 + x_2 + dx_3 = b_3, \end{cases}$$

in two ways.

Special cases of solving systems of equations

A system of equations can have a number of solutions equal to the number of unknowns, such a system is called compatible. If the number of equations is infinitely large, then the system is called compatible and indefinite. If the system has no solution, it is called incompatible.

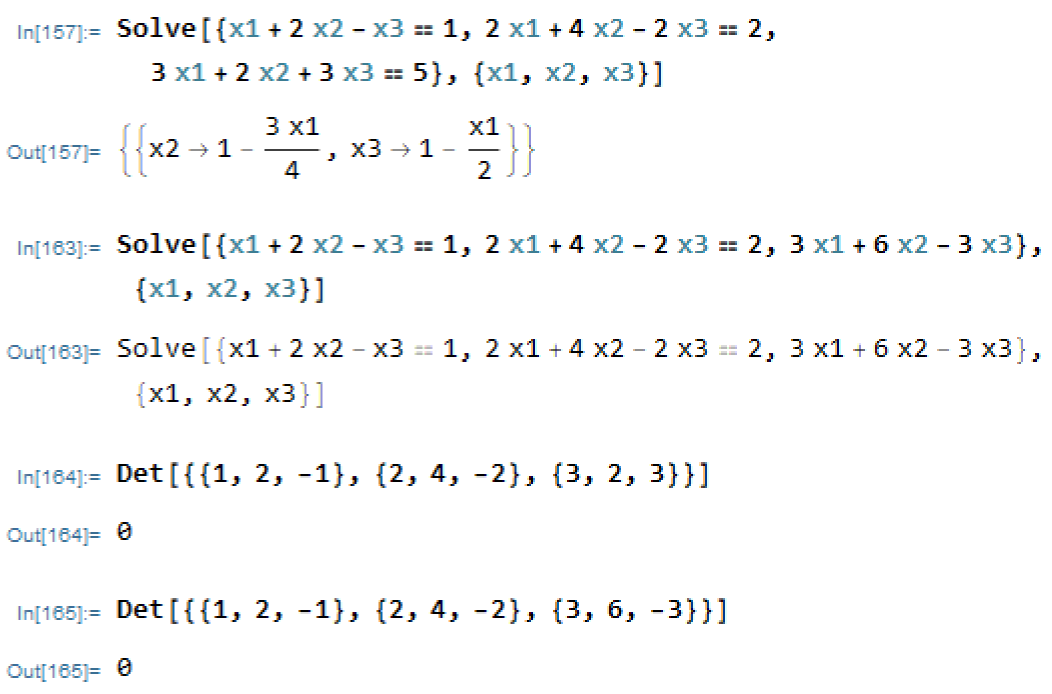
```
In[148]:= A = {{a, 2, -3}, {-7, c, 1}, {1, 1, d}}

Out[148]= {{a, 2, -3}, {-7, c, 1}, {1, 1, d}}

In[149]:= B = {b1, b2, b3}

Out[149]= {b1, b2, b3}

In[150]:= X = Dot[Inverse[A], B]

                 b3 (2 + 3 c)
Out[150]= {------------------------- +
           23 - a + 3 c + 14 d + a c d
              b2 (-3 - 2 d)                b1 (-1 + c d)
           ------------------------- + -------------------------,
           23 - a + 3 c + 14 d + a c d   23 - a + 3 c + 14 d + a c d
              (21 - a) b3                  b1 (1 + 7 d)
           ------------------------- + ------------------------- +
           23 - a + 3 c + 14 d + a c d   23 - a + 3 c + 14 d + a c d
              b2 (3 + a d)                  (2 - a) b2
           -------------------------,  ------------------------- +
           23 - a + 3 c + 14 d + a c d   23 - a + 3 c + 14 d + a c d
              b1 (-7 - c)                  b3 (14 + a c)
           ------------------------- + -------------------------}
           23 - a + 3 c + 14 d + a c d   23 - a + 3 c + 14 d + a c d

In[151]:= Simplify[%]

           b3 (2 + 3 c) - b2 (3 + 2 d) + b1 (-1 + c d)
Out[151]= {---------------------------------------------,
                 23 + 3 c + 14 d + a (-1 + c d)
           b1 + 3 b2 + 21 b3 - a b3 + 7 b1 d + a b2 d
           ------------------------------------------,
                 23 - a + 3 c + 14 d + a c d
           -(-2 + a) b2 - b1 (7 + c) + b3 (14 + a c)
           ------------------------------------------}
                 23 + 3 c + 14 d + a (-1 + c d)

In[152]:= X = LinearSolve[A, B]

           -b1 - 3 b2 + 2 b3 + 3 b3 c - 2 b2 d + b1 c d
Out[152]= {---------------------------------------------,
                    23 - a + 3 c + 14 d + a c d
           b1 + 3 b2 + 21 b3 - a b3 + 7 b1 d + a b2 d
           ------------------------------------------,
                 23 - a + 3 c + 14 d + a c d
           -7 b1 + 2 b2 - a b2 + 14 b3 - b1 c + a b3 c
           --------------------------------------------}
                  23 - a + 3 c + 14 d + a c d
```

Listing 14

The following two systems of equations demonstrate special cases in solving systems of linear equations:

$$\begin{cases} x_1 + 2x_2 - x_3 = 1; \\ 2x_1 + 4x_2 - 2x_3 = 2; \\ 3x_1 + 2x_2 + 3x_3 = 5. \end{cases} \qquad \begin{cases} x_1 + 2x_2 - x_3 = 1; \\ 2x_1 + 4x_2 - 2x_3 = 2; \\ 3x_1 + 6x_2 - 3x_3 = 3. \end{cases}$$

Their solution is shown in Listing 15.



```
In[157]:= Solve[{x1 + 2 x2 - x3 == 1, 2 x1 + 4 x2 - 2 x3 == 2,
             3 x1 + 2 x2 + 3 x3 == 5}, {x1, x2, x3}]

                        3 x1             x1
Out[157]= {{x2 → 1 - ------ , x3 → 1 - ---- }}
                         4              2

In[163]:= Solve[{x1 + 2 x2 - x3 == 1, 2 x1 + 4 x2 - 2 x3 == 2, 3 x1 + 6 x2 - 3 x3},
           {x1, x2, x3}]

Out[163]= Solve[{x1 + 2 x2 - x3 == 1, 2 x1 + 4 x2 - 2 x3 == 2, 3 x1 + 6 x2 - 3 x3},
           {x1, x2, x3}]

In[164]:= Det[{{1, 2, -1}, {2, 4, -2}, {3, 2, 3}}]

Out[164]= 0

In[165]:= Det[{{1, 2, -1}, {2, 4, -2}, {3, 6, -3}}]

Out[165]= 0
```

Listing 15

Listing 15 shows that the first system of equations has a solution: $x_2 = 1 - \frac{3x_1}{4}, x_3 = 1 - \frac{x_1}{2}$,, that is, has an infinite number of solutions (for any value x_1). The system is compatible but uncertain. The second system is also compatible and uncertain, having the solution: $x_3 = -1 + x_1 + 2x_2$, substituting values x_1, x_2. Note that in both cases the main determinant of the system is zero.

Lecture № 8. Functions of mathematical analysis

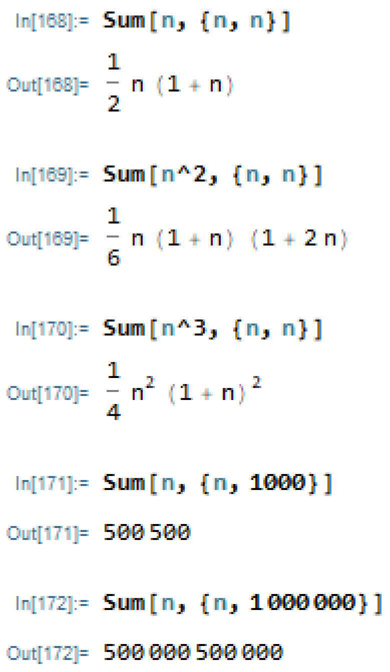
Calculation of sums and products of series

The calculation of the sums of series can be carried out in analytical or numerical form. The function is used to calculate the amounts in analytical formSum. The following Sum recording formats exist:

- Sum [fi, {i, imax}];
- Sum [fi, {i, imin, imax}];
- Sum [fi, {i, imin, imax, Δi}];
- Sum [fi, j, .. {i, imin, imax}, {j, jmin, jmax}],

where f is the summation element,
and, j - summation variables,
imin, imax - summation elements,
Δi is the step of changing the argument and.

If it is necessary to calculate the sum of the members of the series represented by the analytical function, from 1 to n, you must use the function Sum [fi, {i, imax}]. Listing 1 shows the use of the function to calculate the series 1 / n, 1 / n2, 1 / n3.

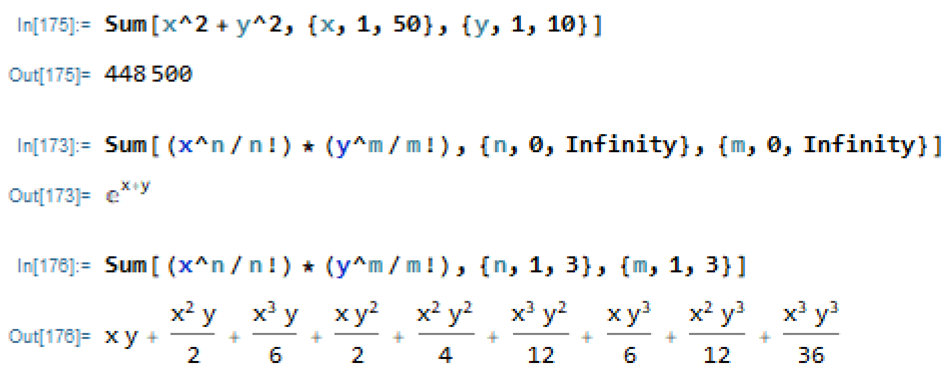


```
In[168]:= Sum[n, {n, n}]

           1
Out[168]=  - n (1 + n)
           2

In[169]:= Sum[n^2, {n, n}]

           1
Out[169]=  - n (1 + n) (1 + 2 n)
           6

In[170]:= Sum[n^3, {n, n}]

           1  2        2
Out[170]=  - n  (1 + n)
           4

In[171]:= Sum[n, {n, 1000}]

Out[171]= 500 500

In[172]:= Sum[n, {n, 1 000 000}]

Out[172]= 500 000 500 000
```

Listing 1. Using the Sum function [fi, {i, imax}]

Function Sum [fi, {i, imin, imax}] calculates the sum of the values of the function f in the range of values of the argument imin ..imax with step 1. When using the function Sum [fi, {i, imin, imax, Δi}] step of changing the argument is given by the parameter Δi.

Function Sum [fi, j, .. {i, imin, imax}, {j, jmin, jmax}] calculates the sum of several variables. An example of its use is illustrated in the calculation of the following series: $\sum_{i=1}^{50}\sum_{j=1}^{10}(x_i^2+y_j^2)$,, $\sum_{n=0}^{\infty}\sum_{m=0}^{\infty}\frac{x^n}{n!}\frac{y^m}{m!}$,, $\sum_{n=1}^{3}\sum_{m=1}^{3}\frac{x^n}{n!}\frac{y^m}{m!}$ shown in Listing 2.



```
In[175]:= Sum[x^2 + y^2, {x, 1, 50}, {y, 1, 10}]
Out[175]= 448500

In[173]:= Sum[(x^n / n!) * (y^m / m!), {n, 0, Infinity}, {m, 0, Infinity}]
```

Out[173]= e^{x+y}

```
In[176]:= Sum[(x^n / n!) * (y^m / m!), {n, 1, 3}, {m, 1, 3}]
```

Out[176]= $x\,y + \frac{x^2 y}{2} + \frac{x^3 y}{6} + \frac{x y^2}{2} + \frac{x^2 y^2}{4} + \frac{x^3 y^2}{12} + \frac{x y^3}{6} + \frac{x^2 y^3}{12} + \frac{x^3 y^3}{36}$

Listing 2

Numerical calculation of sums is performed by the NSum function, which has the same modifications as the Sum function.

The product is calculated similarly to summation. To do this, use the functions:

- Product - for calculating products in analytical and numerical form;
- NProduct - to calculate products only in numerical form.

Calculation of the function boundary

Calculating the function boundary in the system Mathematica is done using the Limit function. The syntax of its record is as follows:

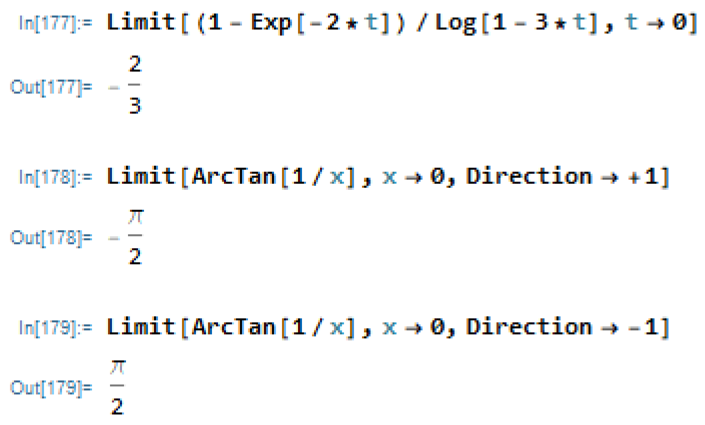
Limit [f (x), x-> x0].

The Limit function has a Direction option. The Direction option indicates the direction of approach to the border. Her record has two options:

Direction -> +1;

Direction -> -1.

A value of +1 indicates approaching the border on the left side, -1 - on the right side. An example of using the Limit function is shown in Listing 3.



```
In[177]:= Limit[(1 - Exp[-2 * t]) / Log[1 - 3 * t], t → 0]
```

Out[177]= $-\frac{2}{3}$

```
In[178]:= Limit[ArcTan[1 / x], x → 0, Direction → +1]
```

Out[178]= $-\frac{\pi}{2}$

```
In[179]:= Limit[ArcTan[1 / x], x → 0, Direction → -1]
```

Out[179]= $\frac{\pi}{2}$

Listing 3

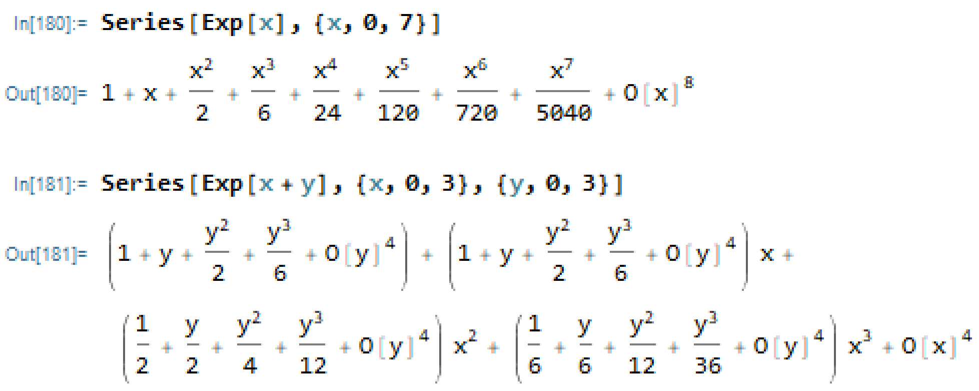
Schedule functions in power series

To decompose into a power series in the system Mathematica uses the following functions:

Series [f, {x, x0, n}] - decomposes the function f around the point x = x0 using n members of the series;

Series [f (x, y), {x, x0, nx}, {y, y0, ny}] - decomposes the function f into two variables x and y around the point (x0, y0) with the number of members nx and ny, respectively .

Examples of using functions are shown in Listing 4.



```
In[180]:= Series[Exp[x], {x, 0, 7}]
```

Out[180]= $1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \frac{x^4}{24} + \frac{x^5}{120} + \frac{x^6}{720} + \frac{x^7}{5040} + O[x]^8$

```
In[181]:= Series[Exp[x + y], {x, 0, 3}, {y, 0, 3}]
```

Out[181]= $\left(1 + y + \frac{y^2}{2} + \frac{y^3}{6} + O[y]^4\right) + \left(1 + y + \frac{y^2}{2} + \frac{y^3}{6} + O[y]^4\right) x + \left(\frac{1}{2} + \frac{y}{2} + \frac{y^2}{4} + \frac{y^3}{12} + O[y]^4\right) x^2 + \left(\frac{1}{6} + \frac{y}{6} + \frac{y^2}{12} + \frac{y^3}{36} + O[y]^4\right) x^3 + O[x]^4$

Listing 4

Calculation of derivative functions

The calculation of derivatives is carried out using the following functions:

D [f, x];

D [f, {x, n}];

D [f, x1, x2,…];

Dt [f, x];

Dt [f];

Derivative [n1, n2, ..] [f] - is a derivative of f [{x1, x2,…}] taken ni times xi. Where f is the differentiated function,
x is the variable of differentiation,
x1, x2, .. - differentiation variables,
n is the order of the derivative.

Methods for calculating integrals

Analytical methods

The integral in analytical form is calculated using the following built-in functions:

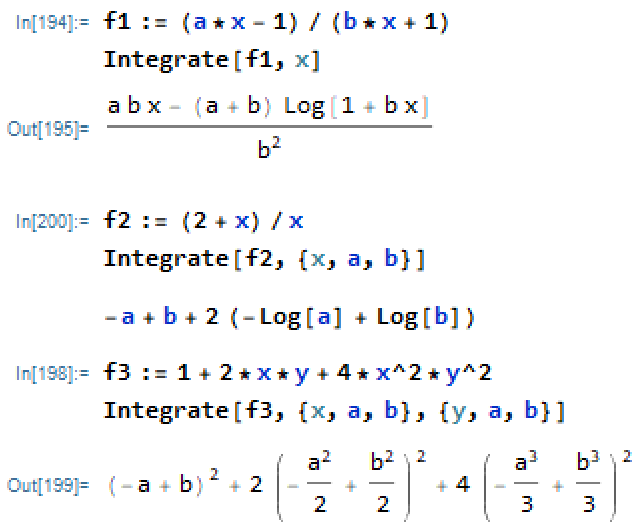
- *Integrate* $[f(x), x]$ - calculates the indefinite integral of the function by the argument x;$f(x)$

- - calculates the definite integral of the function *Integrate* $[f(x), \{x, x_н, x_к\}]$ $f(x)$ on the variable with the lower and upper limits of integration. The limits of integration can be symbolic variables, numbers and even functions;$x x_н x_к$

- - Calculates the definite integral of the function of many variables with integration limits,,,.*Integrate* $[f(x, y, \ldots), \{x, x_н, x_к\}, \{y, y_н, y_k, \ldots\}]$ $x, y, \ldots x_н x_к y_н y_к$

Examples of calculating integrals are shown in Listing 5. The following integrals are calculated:

$$\int \frac{ax-1}{bx+1} dx, \int_a^b \frac{2+x}{x} dx, \int_a^b (1 + 2xy + 4x^2y^2)\, dxdy \,.$$



```
In[194]:= f1 := (a * x - 1) / (b * x + 1)
          Integrate[f1, x]
```

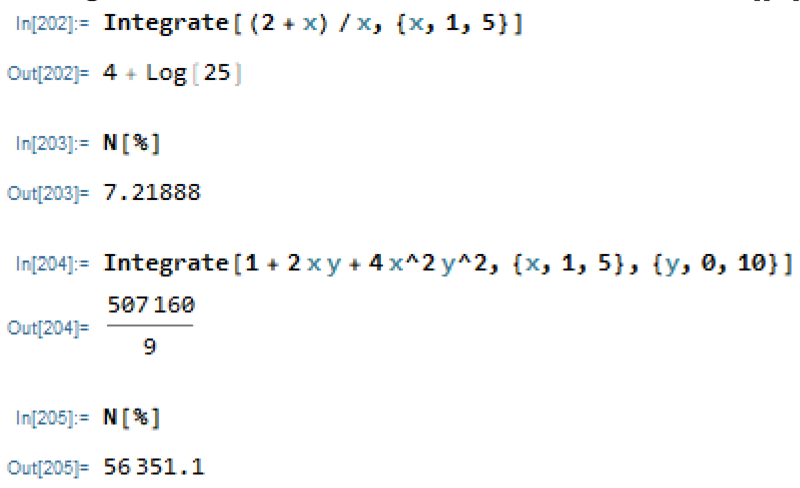
$$\text{Out[195]=}\quad \frac{a\,b\,x - (a + b)\ \text{Log}[1 + b\,x]}{b^2}$$

```
In[200]:= f2 := (2 + x) / x
          Integrate[f2, {x, a, b}]

          -a + b + 2 (-Log[a] + Log[b])

In[198]:= f3 := 1 + 2 * x * y + 4 * x^2 * y^2
          Integrate[f3, {x, a, b}, {y, a, b}]
```

$$\text{Out[199]=}\quad (-a + b)^2 + 2\left(-\frac{a^2}{2} + \frac{b^2}{2}\right)^2 + 4\left(-\frac{a^3}{3} + \frac{b^3}{3}\right)^2$$

Listing 5

In the previous example, the boundaries of integration were the symbolic variables a and b. Listing 6 shows the calculation of the integrals of the same functions in the case when the limits of integration are the numbers: = 1, = 5, = 0, = 10.$x_H x_K y_H y_K$



```
In[202]:= Integrate[(2 + x) / x, {x, 1, 5}]

Out[202]= 4 + Log[25]

In[203]:= N[%]

Out[203]= 7.21888

In[204]:= Integrate[1 + 2 x y + 4 x^2 y^2, {x, 1, 5}, {y, 0, 10}]
```

$$\text{Out[204]=}\quad \frac{507\,160}{9}$$

```
In[205]:= N[%]

Out[205]= 56 351.1
```

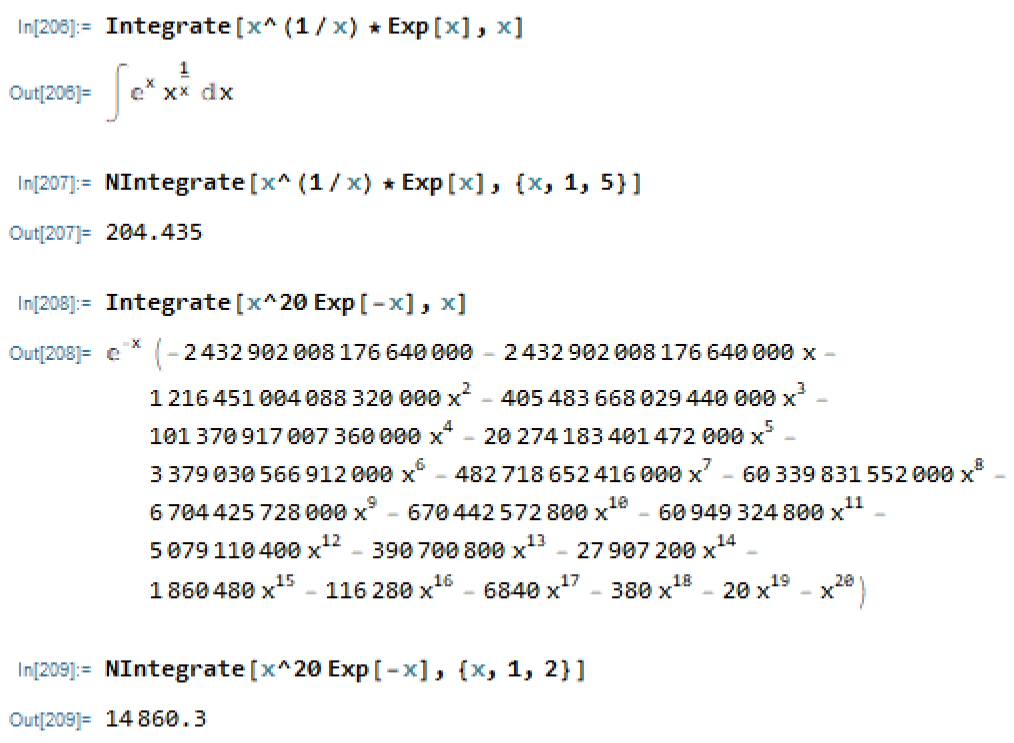
Listing 6

Listing 6 shows that in this case obtained in the form of exact solutions. Numerical values of integrals are obtained using the function N (%).

Numerical Methods

Calculation of integrals in numerical form is necessary in the following cases:
- the original is not expressed through elementary functions;
- subintegral function is given in the form of a table;
- the analytical expression of the original is too complex.

As an example, Listing 7 shows the calculations of indefinite and definite integrals of functions:. $y(x) = x^{\frac{1}{x}} e^{x} u\, y(x) = x^{20} e^{-x}$



```
In[206]:= Integrate[x^(1/x) * Exp[x], x]
```

$$\text{Out[206]=}\ \int e^{x}\, x^{\frac{1}{x}}\, dx$$

```
In[207]:= NIntegrate[x^(1/x) * Exp[x], {x, 1, 5}]

Out[207]= 204.435

In[208]:= Integrate[x^20 Exp[-x], x]
```

$$\text{Out[208]=}\ e^{-x}\left(-2\,432\,902\,008\,176\,640\,000 - 2\,432\,902\,008\,176\,640\,000\, x - 1\,216\,451\,004\,088\,320\,000\, x^{2} - 405\,483\,668\,029\,440\,000\, x^{3} - 101\,370\,917\,007\,360\,000\, x^{4} - 20\,274\,183\,401\,472\,000\, x^{5} - 3\,379\,030\,566\,912\,000\, x^{6} - 482\,718\,652\,416\,000\, x^{7} - 60\,339\,831\,552\,000\, x^{8} - 6\,704\,425\,728\,000\, x^{9} - 670\,442\,572\,800\, x^{10} - 60\,949\,324\,800\, x^{11} - 5\,079\,110\,400\, x^{12} - 390\,700\,800\, x^{13} - 27\,907\,200\, x^{14} - 1\,860\,480\, x^{15} - 116\,280\, x^{16} - 6840\, x^{17} - 380\, x^{18} - 20\, x^{19} - x^{20}\right)$$

```
In[209]:= NIntegrate[x^20 Exp[-x], {x, 1, 2}]

Out[209]= 14 860.3
```

Listing 7

Listing 7 shows that the first integral is not analytically solved, and the second is too cumbersome. In these cases, the result of calculating the integrals by numerical methods can be obtained using the NIntegrate function.

The format of the NIntegrate function is as follows:

$NIntegrate(f(x), \{x, x_H, x_K\})$,,

where f (x) is a subintegral function;
x - argument of subintegral function;
x_H, x_K - lower and upper limits of integration.
The method of using the NIntegrate function does not differ from the method of calculating a definite integral in analytical form.

Calculation of multiple integrals

The calculation of multiple integrals in the Mathematica system is carried out by repeatedly using the Integrate function in analytical integration or NIntegrate in numerical integration. Listing 8 shows examples of calculating multiple integrals of a function for different variants of integration limits:$f(x) = \frac{x-1}{x+1}$

- symbolic variables from a to b;
- numerical values of integration limits from 0 to 2;
- the limits of integration given by the functions: ln 2 and.$e^{1.2}$

```
          Integrate[Integrate[(x - 1) / (x + 1), x], {x, a, b}]
          (*∫_a^b ∫ (x-1) / (x+1) dx  dx*)

          1
          - (-a² + b² - 4 (a - b - (1 + a) Log[1 + a] + (1 + b) Log[1 + b]));
          2

In[212]:= (*∫_a^b ∫∫ (x-1) / (x+1) dx  dx  dx*)
          Integrate[Integrate[Integrate[(x - 1) / (x + 1), x], x],
           {x, a, b}]
            1                   1
          - - a (6 + 9 a + a²) + - b (6 + 9 b + b²) + (1 + a)² Log[1 + a] -
            6                   6
            (1 + b)² Log[1 + b];

In[213]:= (*∫_0^2 ∫∫ (x-1) / (x+1) dx  dx  dx*)
          Integrate[Integrate[Integrate[(x - 1) / (x + 1), x], x],
           {x, 0, 2}]
          28
Out[213]= -- - Log[19683]
          3

In[214]:= N[%]

Out[214]= -0.554177

In[215]:= (*∫_Log[2]^Exp[1.2] ∫∫ (x-1) / (x+1) dx  dx  dx*)
          Integrate[Integrate[Integrate[(x - 1) / (x + 1), x], x],
           {x, Log[2], Exp[1.2]}]

Out[215]= -1.31499
```

Listing 8

Calculation of improper integrals

The Mathematica system allows you to compute integrals with infinite boundaries. The same functions are used as in the case of calculating integrals with finite limits. The infinity value is denoted by either the ∞ symbol or the Infinity service constant. Examples of calculating improper integrals are given in sheet 9.

```
In[216]:= (*∫_0^∞ Sin[x]/x dx*)
          Integrate[Sin[x] / x, {x, 0, Infinity}]
```
Out[216]= $\frac{\pi}{2}$

```
In[217]:= (*∫_0^∞ (Sinh[x]+Cosh[x])/Exp[x^2] dx*)
          Integrate[(Sinh[x] + Cosh[x]) / Exp[x^2], {x, 0, Infinity}]
```
Out[217]= $\frac{1}{2} e^{1/4} \sqrt{\pi} \left(1 + \mathrm{Erf}\left[\frac{1}{2}\right]\right)$

```
In[218]:= N[%]

Out[218]= 1.73023

In[219]:= (*∫_0^∞ (x+1)/(x^3+1) dx*)
          Integrate[(x + 1) / (x^3 + 1), {x, 0, Infinity}]
```
Out[219]= $\frac{4\pi}{3\sqrt{3}}$

```
In[222]:= (*∫_-∞^∞ Exp[x]/(a+Exp[x])^2 dx*)
          Integrate[Exp[x] / (a + Exp[x])^2, {x, -Infinity, Infinity}]
```
Out[222]= ConditionalExpression$\left[\frac{1}{a}, \mathrm{Im}[a] \neq 0 \,||\, \mathrm{Re}[a] == -1 \,||\, \mathrm{Re}[a] \geq 0\right]$

```
In[223]:= (*∫_0^∞ (x+1)/(2 x+1) dx*)
          Integrate[(x + 1) / (2 x + 1), {x, 0, Infinity}]
```
Out[223]= $\int_0^{\infty} \frac{1+x}{1+2x} dx$

```
In[224]:= NIntegrate[1 / (2 x^2 + 1), {x, 0, Infinity}]

Out[224]= 1.11072
```

Listing 9

The following conclusions can be drawn from Listing 9:
- the solution of the improper integral is obtained in analytical form. To obtain the numerical value of the integral, use the command N [%];
- the expression of the original function in analytical form can be complex. To simplify it, you should use the functions Simplify, Expand, Factor or their prototypes;
- if the integral does not have an initial, then the result will be the initial expression of the integral.

Tabular integration

The subintegral function f (x) can be given in the form of a table. This is often necessary when conducting experimental research. In such cases, the calculation of the integral can be performed by the formulas of rectangles, trapezoids or parabolas. The solution can also be obtained by interpolating the function f (x) with its subsequent integration. Mathematica after version 6.0 changed the built-in ListIntegrate function to the integrated form Integrate [Interpolation []], which has the following features of use:

- *Integrate*[Interpolation;$[\{y_1, y_2, \dots, y_n\}][x], \{x, x_1, x_2\}]$
- *Integrate*[Interpolation;$[\{y_1, y_2, \dots, y_n\}$, InterpolationOrder $\to$ $k][x], \{x, x_1, x_2\}]$
- *Integrate*[Interpolation [data] [x], {x, Min [xc], Max [xc]}];

The functions use the following notation:
- y_i - the value of the function y = f (x) in the - node, 1 = 1, 2, ..., n;x_i
- x is the argument of the function y = f (x).
- x1, x2 - final values of the argument;
- k is the interpolation order.
-;data $= \{\{x_1, y_1\}, \dots, \{x_n, y_n\}\}$
-;$x_c =$ data$[[All, 1]]$

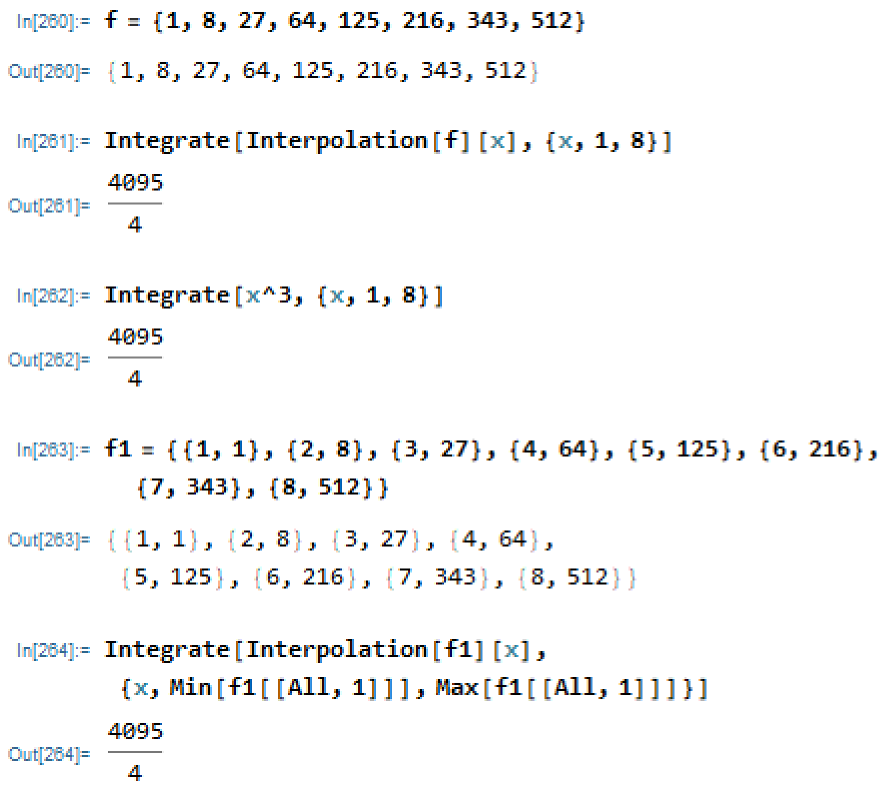
Using the function for tabular integration, you can skip writing the built-in character InterpolationOrder, then the default interpolation order will be k = 3.

As an example, calculate the value of the integral of the tabular function y = f (x), the values of which are given in table. 1.

Table 1. Tabular representation of the function y = f (x)

x	1	2	3	4	5	6	7	8
in	1	8	27	64	125	216	343	512

In this case, the integration step is constant and equal to h = 1. From table. 1 shows that the analytical expression of the function has the form y =. Listing10 shows the procedures for calculating the integral in the case of specifying a subintegral function in the form of a table and in the form of an analytical expression y =.x^3x^3



```
In[260]:= f = {1, 8, 27, 64, 125, 216, 343, 512}

Out[260]= {1, 8, 27, 64, 125, 216, 343, 512}

In[261]:= Integrate[Interpolation[f][x], {x, 1, 8}]

          4095
Out[261]= ----
           4

In[262]:= Integrate[x^3, {x, 1, 8}]

          4095
Out[262]= ----
           4

In[263]:= f1 = {{1, 1}, {2, 8}, {3, 27}, {4, 64}, {5, 125}, {6, 216},
            {7, 343}, {8, 512}}

Out[263]= {{1, 1}, {2, 8}, {3, 27}, {4, 64},
           {5, 125}, {6, 216}, {7, 343}, {8, 512}}

In[264]:= Integrate[Interpolation[f1][x],
           {x, Min[f1[[All, 1]]], Max[f1[[All, 1]]]}]

          4095
Out[264]= ----
           4
```

Listing 10

Solving differential equations in Mathematica

Mathematica system allows to solve in analytical and numerical form linear and nonlinear differential equations and systems. The solution of differential equations and systems is carried out with the help of built-in functions.

Analytical methods

Analytical methods for solving differential equations in the Mathematica system are implemented using two built-in functions:

$DSolve\left[f,\, y\left[x\right],\, x\right];$

$DSolve\left[\{f_1, f_2,\, ...\},\, \{y\left[x_1\right],\, y\left[x_2\right],\, ...\},\, \{x_1, x_2,\, ...\}\right].$

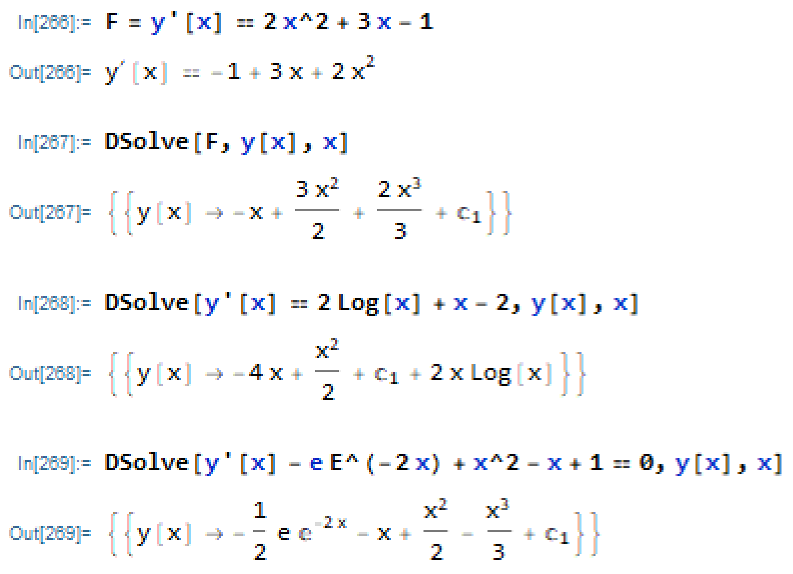
Consider in detail these functions and give examples.

Function $DSolve\left[f,\, y\left[x\right],\, x\right]$ designed to solve the differential equation f regarding the function $y(x)$ with an argument x. The function gives the general solution of the equation with the integrating constants that are denoted $c[i]$.

The differential equation is represented in an arbitrary form. Example, $y' == 2x \wedge 2 - 1,\ y' - 2x \wedge 2 == -1$ or $y' - 2\,x \wedge 2 + 1 == 0$. The function allows you to solve a differential equation of any order. Here are some examples. The following differential equations are given:

$y' = 2x^2 + 3x - 1,\ y' = 2\ln x + x - 2, y' = 3e^{-2x} - x^2 + x - 1$.

The solution of the equations is shown in Listing 11.



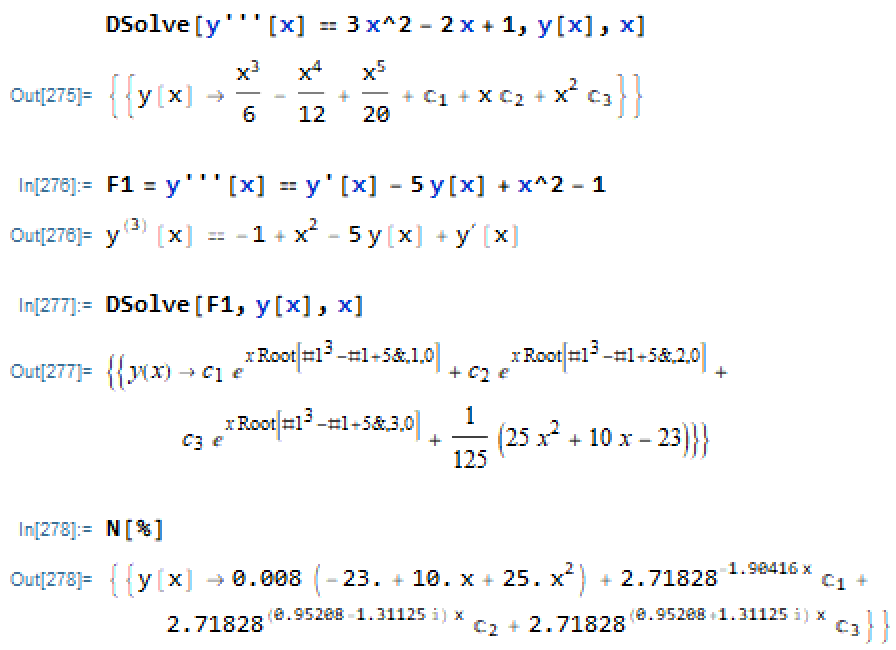
```
In[266]:= F = y'[x] == 2 x^2 + 3 x - 1

Out[266]= y'[x] == -1 + 3 x + 2 x^2

In[267]:= DSolve[F, y[x], x]

Out[267]= {{y[x] → -x + 3 x^2/2 + 2 x^3/3 + C[1]}}

In[268]:= DSolve[y'[x] == 2 Log[x] + x - 2, y[x], x]

Out[268]= {{y[x] → -4 x + x^2/2 + C[1] + 2 x Log[x]}}

In[269]:= DSolve[y'[x] - e E^(-2 x) + x^2 - x + 1 == 0, y[x], x]

Out[269]= {{y[x] → -1/2 e e^(-2 x) - x + x^2/2 - x^3/3 + C[1]}}
```

Listing 11

Listing 11 shows that when solving the first equation, it was given the name F and entered outside the function. *DSolve*. When solving the second equation, the latter is introduced directly into the function *DSolve*. When solving the third equation, it is presented in a form different from the first two. The example shows that the program found a common solution with arbitrary integrating constants, and the solution is reduced to a simple integration of the right-hand side of the equations. Function *DSolve* also allows you to solve high-order differential equations, as shown in the following examples:

$y''' = 3x^2 - 2x + 1,\ y''' = y'(x) - 5y(x) + x2 - 1$.

The solution is shown in Listing 12.



```
          DSolve[y'''[x] == 3 x^2 - 2 x + 1, y[x], x]
                             x^3   x^4   x^5
Out[275]= {{y[x] -> --- - --- + --- + C1 + x C2 + x^2 C3}}
                              6    12    20

In[276]:= F1 = y'''[x] == y'[x] - 5 y[x] + x^2 - 1
Out[276]= y^(3)[x] == -1 + x^2 - 5 y[x] + y'[x]

In[277]:= DSolve[F1, y[x], x]
Out[277]= {{y(x) -> c1 e^(x Root[#1^3 - #1 + 5&, 1, 0]) + c2 e^(x Root[#1^3 - #1 + 5&, 2, 0]) +
                 c3 e^(x Root[#1^3 - #1 + 5&, 3, 0]) + 1/125 (25 x^2 + 10 x - 23)}}

In[278]:= N[%]
Out[278]= {{y[x] -> 0.008 (-23. + 10. x + 25. x^2) + 2.71828^(-1.90416 x) C1 +
              2.71828^((0.95208 - 1.31125 i) x) C2 + 2.71828^((0.95208 + 1.31125 i) x) C3}}
```

Listing 12

Solution of differential equations under known initial conditions

The following modification of the DSolve function is used to obtain the solution of differential equations under the known initial conditions:

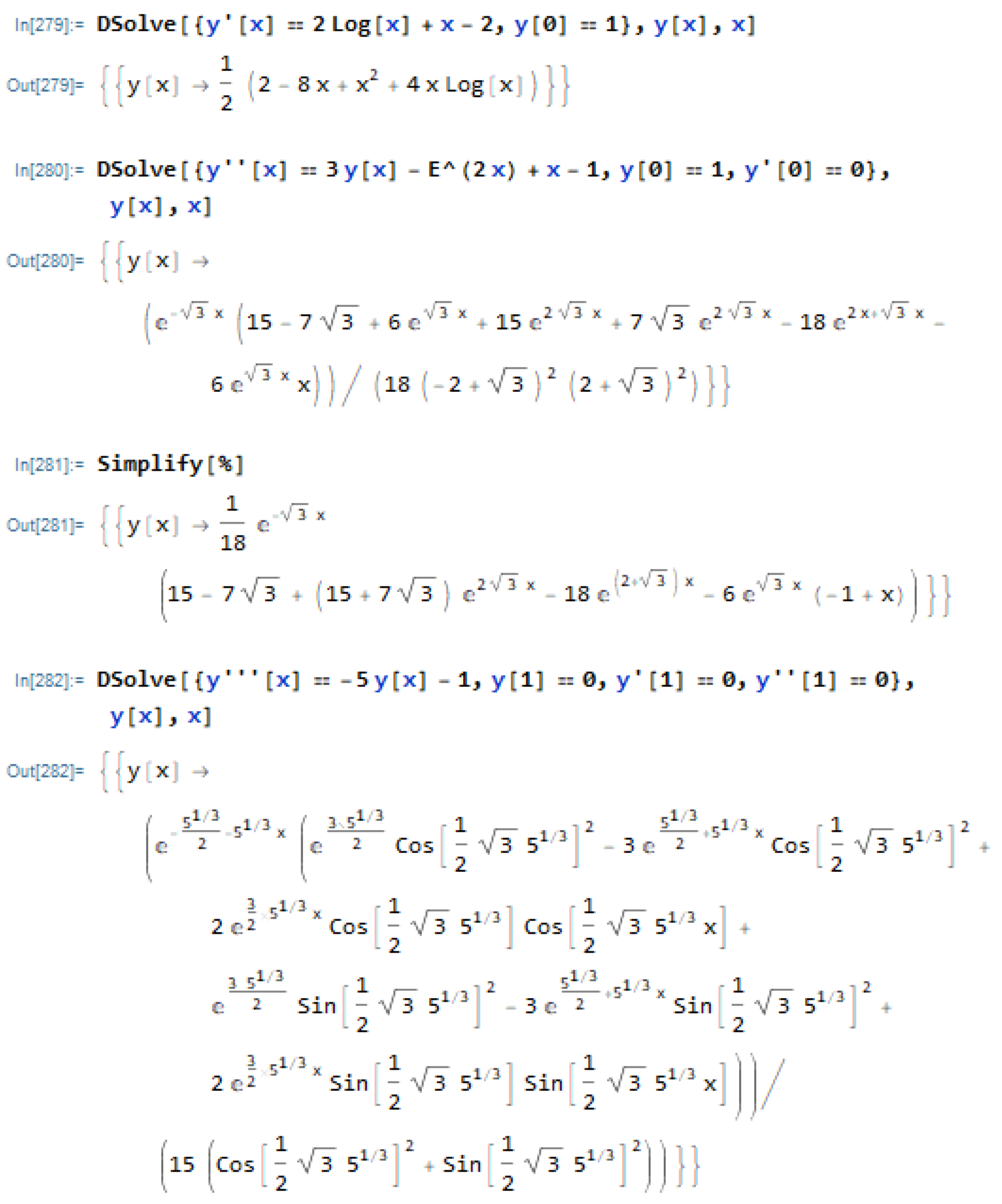
$DSolve\ [f(x,x_0),\ y\ [x],\ x]$,,

where $f(x,x_0)$ - differential equation in conjunction with the initial conditions;
$y(x)$ - desired function;
x - independent variable.

Consider examples of a separate solution of differential equations using a function $DSolve$:

$y'(x) = 2\ln x + x - 2$ under the initial condition: $y(0) = 1$;
$y''(x) = 3y(x) - e^{2x} + x - 1$ at $y(0) = 1, y'(0) = 0$;
$y''(x) = -5y(x) - 1$ at $y(1) = y'(1) = y''(1) = 0$.

The solution is shown in Listing 13.



```
In[279]:= DSolve[{y'[x] == 2 Log[x] + x - 2, y[0] == 1}, y[x], x]
```

$$\text{Out[279]= } \left\{\left\{y[x] \to \frac{1}{2}\left(2 - 8x + x^2 + 4x\,\mathrm{Log}[x]\right)\right\}\right\}$$

```
In[280]:= DSolve[{y''[x] == 3 y[x] - E^(2 x) + x - 1, y[0] == 1, y'[0] == 0},
          y[x], x]
```

$$\text{Out[280]= } \left\{\left\{y[x] \to \left(e^{-\sqrt{3}x}\left(15 - 7\sqrt{3} + 6e^{\sqrt{3}x} + 15e^{2\sqrt{3}x} + 7\sqrt{3}\,e^{2\sqrt{3}x} - 18e^{2x+\sqrt{3}x} - 6e^{\sqrt{3}x}x\right)\right) \Big/ \left(18\left(-2+\sqrt{3}\right)^2\left(2+\sqrt{3}\right)^2\right)\right\}\right\}$$

```
In[281]:= Simplify[%]
```

$$\text{Out[281]= } \left\{\left\{y[x] \to \frac{1}{18}e^{-\sqrt{3}x}\left(15 - 7\sqrt{3} + \left(15 + 7\sqrt{3}\right)e^{2\sqrt{3}x} - 18e^{\left(2+\sqrt{3}\right)x} - 6e^{\sqrt{3}x}(-1+x)\right)\right\}\right\}$$

```
In[282]:= DSolve[{y'''[x] == -5 y[x] - 1, y[1] == 0, y'[1] == 0, y''[1] == 0},
          y[x], x]
```

$$\text{Out[282]= } \Bigg\{\Bigg\{y[x] \to \Bigg(e^{-\frac{5^{1/3}}{2}-5^{1/3}x}\Bigg(e^{\frac{3\,5^{1/3}}{2}}\mathrm{Cos}\left[\frac{1}{2}\sqrt{3}\,5^{1/3}\right]^2 - 3e^{\frac{5^{1/3}}{2}+5^{1/3}x}\mathrm{Cos}\left[\frac{1}{2}\sqrt{3}\,5^{1/3}\right]^2 +$$
$$2e^{\frac{3}{2}5^{1/3}x}\mathrm{Cos}\left[\frac{1}{2}\sqrt{3}\,5^{1/3}\right]\mathrm{Cos}\left[\frac{1}{2}\sqrt{3}\,5^{1/3}x\right] +$$
$$e^{\frac{3\,5^{1/3}}{2}}\mathrm{Sin}\left[\frac{1}{2}\sqrt{3}\,5^{1/3}\right]^2 - 3e^{\frac{5^{1/3}}{2}+5^{1/3}x}\mathrm{Sin}\left[\frac{1}{2}\sqrt{3}\,5^{1/3}\right]^2 +$$
$$2e^{\frac{3}{2}5^{1/3}x}\mathrm{Sin}\left[\frac{1}{2}\sqrt{3}\,5^{1/3}\right]\mathrm{Sin}\left[\frac{1}{2}\sqrt{3}\,5^{1/3}x\right]\Bigg)\Bigg) \Big/$$
$$\left(15\left(\mathrm{Cos}\left[\frac{1}{2}\sqrt{3}\,5^{1/3}\right]^2 + \mathrm{Sin}\left[\frac{1}{2}\sqrt{3}\,5^{1/3}\right]^2\right)\right)\Bigg\}\Bigg\}$$

Listing 13

Solution of systems of differential equations in analytical form

Systems of differential equations in analytical form are also solved using the built-in function *DSolve*, which in this case has the form:

$$DSolve\,[\{f_1, f_2, ...\}, \{y_1(x), y_2(x), ...\}, x]$$

where f_i - the i-th equation of the system;

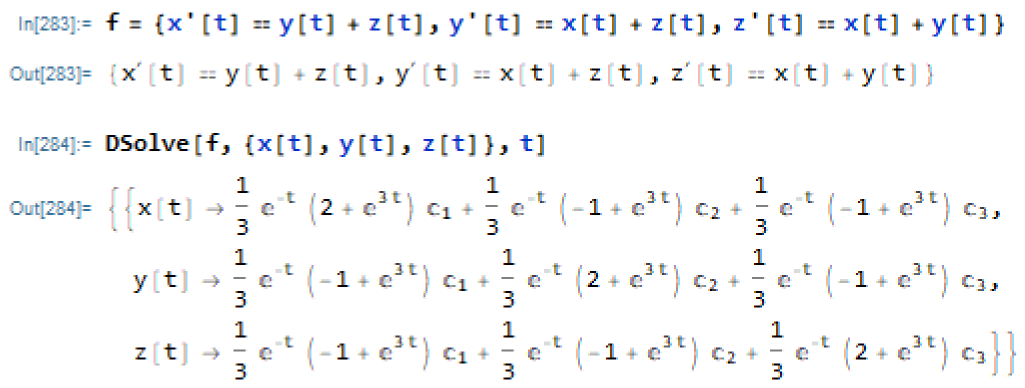
$y_i(x)$ - and the sought-after unknown;

x - independent variable.

Examples of using the function *DSolve* for the case of solving systems of differential equations are given below:

$$\begin{cases} x'(t) = y(t) + z(t) \\ y'(t) = x(t) + 3z(t) \\ z'(t) = x(t) + y(t). \end{cases}$$

The solution is shown in Listing 14.



```
In[283]:= f = {x'[t] == y[t] + z[t], y'[t] == x[t] + z[t], z'[t] == x[t] + y[t]}

Out[283]= {x′[t] == y[t] + z[t], y′[t] == x[t] + z[t], z′[t] == x[t] + y[t]}

In[284]:= DSolve[f, {x[t], y[t], z[t]}, t]
```

Out[284]=
$$\left\{\left\{x[t]\to\frac{1}{3}e^{-t}\left(2+e^{3t}\right)C_1+\frac{1}{3}e^{-t}\left(-1+e^{3t}\right)C_2+\frac{1}{3}e^{-t}\left(-1+e^{3t}\right)C_3,\right.\right.$$
$$y[t]\to\frac{1}{3}e^{-t}\left(-1+e^{3t}\right)C_1+\frac{1}{3}e^{-t}\left(2+e^{3t}\right)C_2+\frac{1}{3}e^{-t}\left(-1+e^{3t}\right)C_3,$$
$$\left.\left.z[t]\to\frac{1}{3}e^{-t}\left(-1+e^{3t}\right)C_1+\frac{1}{3}e^{-t}\left(-1+e^{3t}\right)C_2+\frac{1}{3}e^{-t}\left(2+e^{3t}\right)C_3\right\}\right\}$$

Listing 14

Listing 14 shows that the solution is obtained in General, as the initial conditions were not specified.

Subject to the substitution of initial conditions: $x(0)=1, y(0)=z(0)=0$ function record *DSolve* is as follows:

$$DSolve[\{x'[t]==y[t]+z[t], y'[t]==x[t]+z[t], z'[t]=x[t]+y[t],$$
$$x[0]=1, y[0]==z[0]==0\}, \{x[t], y[t], z[t]\}, t].$$

Numerical methods for solving differential equations

Numerical methods for solving differential equations in the Mathematica system are implemented using the following two built-in functions:

$NDSolve\ [f, y[x], \{x, x_{\min}, x_{\max}\}]$

$NDSolve\ [\{f_1, f_2, ..., y_1(x_0), y_2(x_0), ...\}, \{y_1[x], y_2[x], ...\}, \{x, x_{\min}, x_{\max}\}]$

where f - differential equation and initial conditions;

f_i - the i-th equation of the system of differential equations;

$y[x]$ - desired function;

$y_i[x]$- the i-th desired function of the system of differential equations;

$y_i(x_0)$ - i-th initial condition;

$x_{\min}, x_{\max}$ - minimum and maximum value of the independent variable;

x - the argument of the desired function.

The numerical solution functions of differential equations and systems have the StartingStepSize option, which determines the value of the initial integration step.

Numerical methods are most often used in cases where the equation in analytical form is not solved by the system or has no analytical solution. These are most nonlinear equations. Next, the built-in functions, methods of their implementation and examples are given in detail.

Function $NDSolve\ [f, y[x], \{x, x_{\min}, x_{\max}\}]$

This function solves the n-th order differential equation by calculating the required function y (x) in the range of the independent variable x from $x_{\min}$ to $x_{\max}$. The solution can be obtained in the form of a table or graph.

The method of solving the problem is shown by the example of the following differential equation:

$y'(t) - xy(x) = 1,$

under initial conditions $y(0) = 1$. The solution should be obtained in tabular and graphical form in the range from 0 to 5 in steps of 0.5.

The method of solving the differential equation using the NDSolve function consists of performing the following operations:

1. Introduction of the NDSolve function, which in our example has the form:
$NDSolve[\{y'[x] == xy[x] + 1, y[0] == 1\}, y[x], \{x, 0, 5\}]$.

2. Get the solution by pressing <Shilit> + <Enter> at the same time. The solution will be received in the form of a message without displaying the solution itself on the screen.

3. Introduction of the Table function to obtain a solution in tabular form. In our example, this function will look like:
$Table[\{x, y[x] / .Out[292]\}, \{x, 0, 5, 0.5\}]$.

Here Out [292] is the 292 line in which the solution of the equation is located. The result is a vector represented as a string.

4. Enter the TableForm function [%] to obtain a solution in the form of a table. The result is a function y (x), presented in the form of a table.

5. Introduction of the function $Plot[\{y[x] / .Out[292]\}, \{x, 0, 5\}]$. The result is a graph of the function.

The solution of the problem is shown in Listing 15. First, the solution of the equation is given in analytical form.

The following example requires solving a higher order equation:

$y'''(x) - 3x^2 y''(x) - 2xy'(x) - y(x) = 1,$

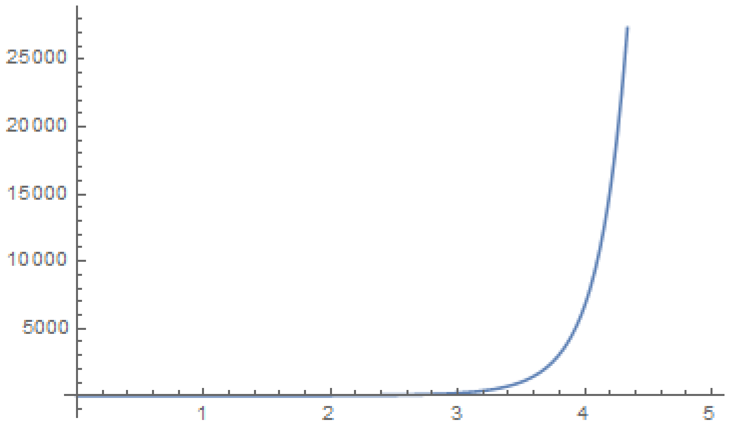
under initial conditions $y(1) = 1, y'(1) = y''(1) = 0$. The solution is obtained in the range of strokes 0 to 3 with a step of 0.5 when presenting the solution in tabular form and in the range from 0 to 2 - in graphical form. The solution is shown in Listing 16.

```
In[285]:= DSolve[{y'[x] == x y[x] + 1, y[0] == 1}, y[x], x]
```

Out[285]= $\left\{\left\{y[x] \to \frac{1}{2} e^{\frac{x^2}{2}} \left(2 + \sqrt{2\pi}\ \mathrm{Erf}\left[\frac{x}{\sqrt{2}}\right]\right)\right\}\right\}$

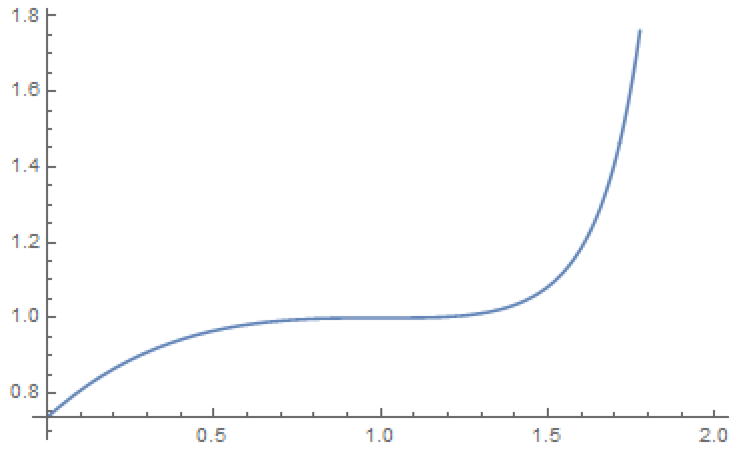
```
In[292]:= NDSolve[{y'[x] == x y[x] + 1, y[0] == 1}, y[x], {x, 0, 5}]
          {{y[x] → InterpolatingFunction[{{0., 5.}}, <>][x]}}

Out[292]= {{y[x] → InterpolatingFunction[ Domain: {{0., 5.}}
                                           Output: scalar ][x]}}

In[293]:= Table[{x, y[x] /. Out[292]}, {x, 0, 5, 0.5}]

Out[293]= {{0., {1.}}, {0.5, {1.67697}},
           {1., {3.05941}}, {1.5, {6.42488}}, {2., {16.2285}},
           {2.5, {50.9309}}, {3., {202.532}}, {3.5, {1029.82}},
           {4., {6716.79}}, {4.5, {56 240.7}}, {5., {604 648.}}}

In[294]:= TableForm[%]

Out[294]//TableForm=
          0.      1.
          0.5     1.67697
          1.      3.05941
          1.5     6.42488
          2.      16.2285
          2.5     50.9309
          3.      202.532
          3.5     1029.82
          4.      6716.79
          4.5     56 240.7
          5.      604 648.

In[295]:= Plot[y[x] /. Out[292], {x, 0, 5}]
```

Out[295]=



Listing 15

```
In[312]:= NDSolve[{y'''[x] == 3 x^(2) y''[x] + 2 x y'[x] + y[x] + 1,
             y[1] == 1, y'[1] == 0, y''[1] == 0}, y[x], {x, 0, 5}]
           {{y[x] → InterpolatingFunction[{{0., 5.}}, <>][x]}}

Out[312]= {{y[x] → InterpolatingFunction[  Domain: {{0., 5.}}
                                            Output: scalar  ][x]}}

In[314]:= Table[{x, y[x] /. Out[312]}, {x, 0, 3, 0.5}]

Out[314]= {{0., {0.736358}}, {0.5, {0.966263}}, {1., {1.}}, {1.5, {1.08349}},
            {2., {6.94346}}, {2.5, {5233.08}}, {3., {2.36809 × 10^8}}}

In[315]:= TableForm[%]

Out[315]//TableForm=
          0.      0.736358
          0.5     0.966263
          1.      1.
          1.5     1.08349
          2.      6.94346
          2.5     5233.08
          3.      2.36809 × 10^8

In[316]:= Plot[{y[x] /. Out[312]}, {x, 0, 2}]

Out[316]=
```



Listing 16

Function NDSolve $[\{f_1, f_2, ..., y_1(x_0), y_2(x_0), ...\}, \{y_1[x], y_2[x], ...\}, \{x, x_{\min}, x_{\max}\}]$

This function solves a system of n-th order differential equations by calculating the required functions $y_1[x], y_2[x], ...$ in the range of the independent variable x from $x_{\min}$ to $x_{\max}$. The solution can be obtained in the form of tables or graphs.

The method of using the function is shown in the following example:

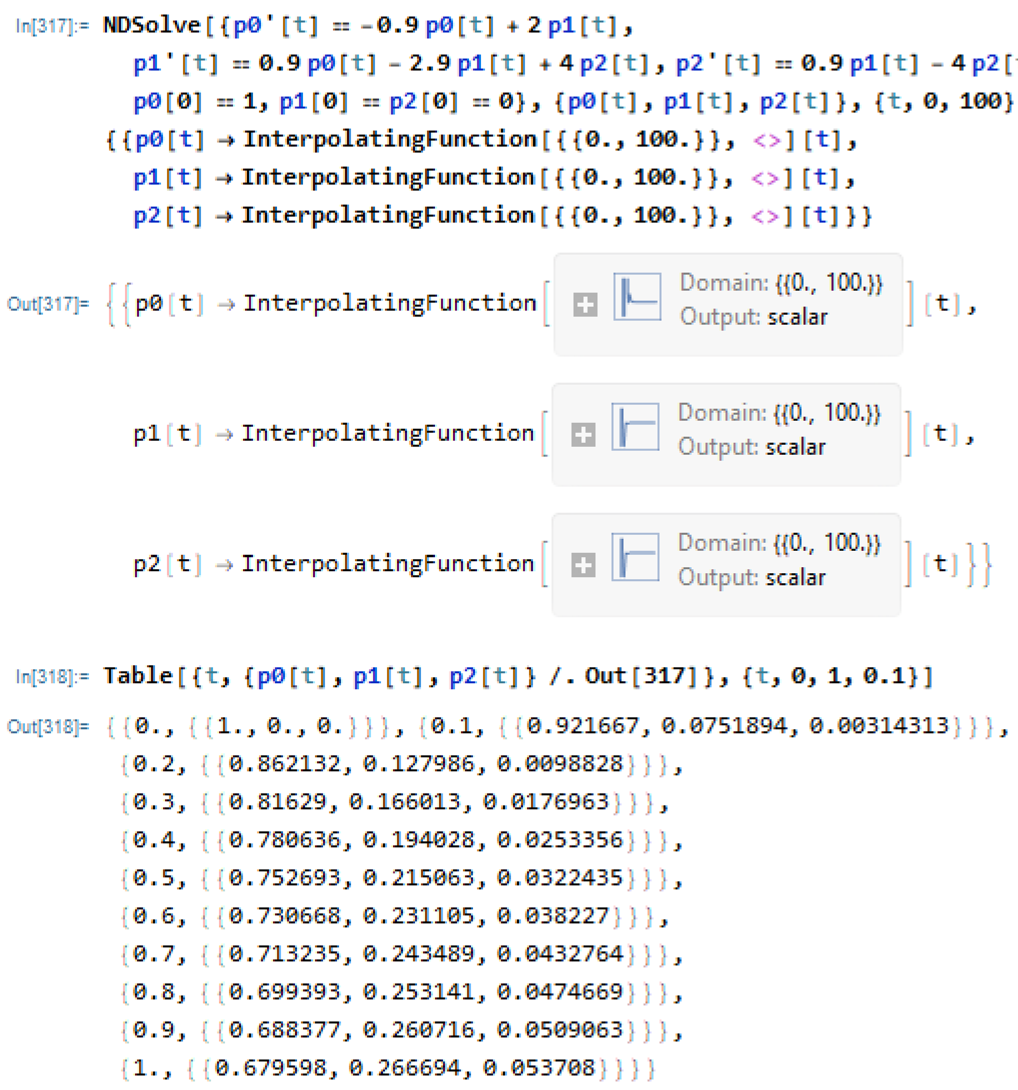
$$\begin{cases} p_0'(t) = -0.9p_0(t) + 2p_1(t) \\ p_1'(t) = 0.9p_0(t) - 2.9p_1(t) + 4p_2(t) \\ p_2'(t) = 0.9p_1(t) - 4p_2(t), \end{cases}$$

under the following initial conditions: $p_0(0) = 1, p_1(0) = p_2(0) = 0$. Get the solution in the form of tables and graphs.

In this case, the NDSolve function will look like:

NDSolve {{p0 '[t] = - 0.9 p0 [t] + 2p1 [t], p1' [t] = 0.9 p0 [t] -2.9p1 [t] + 4p2 [t], p2 '[t] = 0.9p1 [t] -4p2 [t], p0 [0] = 1, p1 [0] = p2 [0] = 0}, {p0 [t], p1 [t], p2 [t]}, {t , 0, 100}].

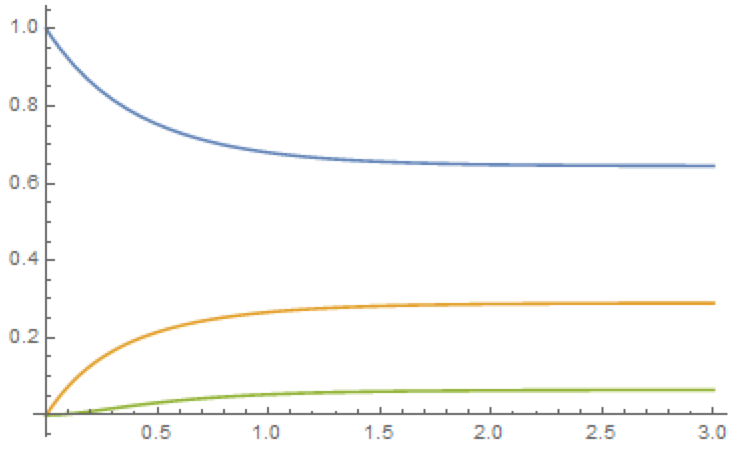
The solution is shown in Listing 17. The table is presented in the range t from 0 to 1 in steps of 0.1, and the graph is presented in the range t from 0 to 3.



```
In[317]:= NDSolve[{p0'[t] == -0.9 p0[t] + 2 p1[t],
            p1'[t] == 0.9 p0[t] - 2.9 p1[t] + 4 p2[t], p2'[t] == 0.9 p1[t] - 4 p2[t],
            p0[0] == 1, p1[0] == p2[0] == 0}, {p0[t], p1[t], p2[t]}, {t, 0, 100}]
          {{p0[t] → InterpolatingFunction[{{0., 100.}}, <>][t],
            p1[t] → InterpolatingFunction[{{0., 100.}}, <>][t],
            p2[t] → InterpolatingFunction[{{0., 100.}}, <>][t]}}

Out[317]= {{p0[t] → InterpolatingFunction[ Domain: {{0., 100.}}
                                            Output: scalar ][t],

            p1[t] → InterpolatingFunction[ Domain: {{0., 100.}}
                                            Output: scalar ][t],

            p2[t] → InterpolatingFunction[ Domain: {{0., 100.}}
                                            Output: scalar ][t]}}

In[318]:= Table[{t, {p0[t], p1[t], p2[t]} /. Out[317]}, {t, 0, 1, 0.1}]

Out[318]= {{0., {{1., 0., 0.}}}, {0.1, {{0.921667, 0.0751894, 0.00314313}}},
           {0.2, {{0.862132, 0.127986, 0.0098828}}},
           {0.3, {{0.81629, 0.166013, 0.0176963}}},
           {0.4, {{0.780636, 0.194028, 0.0253356}}},
           {0.5, {{0.752693, 0.215063, 0.0322435}}},
           {0.6, {{0.730668, 0.231105, 0.038227}}},
           {0.7, {{0.713235, 0.243489, 0.0432764}}},
           {0.8, {{0.699393, 0.253141, 0.0474669}}},
           {0.9, {{0.688377, 0.260716, 0.0509063}}},
           {1., {{0.679598, 0.266694, 0.053708}}}}
```

Listing 17 (Part 1 of 2)

```
In[319]:= TableForm[%]

Out[319]//TableForm=
          0.     1. 0. 0.
          0.1    0.921667 0.0751894 0.00314313
          0.2    0.862132 0.127986 0.0098828
          0.3    0.81629  0.166013 0.0176963
          0.4    0.780636 0.194028 0.0253356
          0.5    0.752693 0.215063 0.0322435
          0.6    0.730668 0.231105 0.038227
          0.7    0.713235 0.243489 0.0432764
          0.8    0.699393 0.253141 0.0474669
          0.9    0.688377 0.260716 0.0509063
          1.     0.679598 0.266694 0.053708

In[325]:= Plot[{{p0[t] /. Out[317]}, {p1[t] /. Out[317]},
            {p2[t] /. Out[317]}}, {t, 0, 3}]

Out[325]=
```



Listing 17 (Part 2 of 2)

Mathematica, like any other computer algebra system, is not ideal for solving differential equations. The obtained solution rarely coincides with the answer available in mathematical reference books. It is not uncommon for a built-in function to give no solution or to be erroneous, although the equation is quite simple. Here are some examples. Suppose you need to solve the following equations and systems of equations:

$y''(x) = -y(x) + tgx;$

$y''(x) = -y(x) + 4x\sin x;$

$x'(t) + y'(t) - tx(t) = t;$

$x'(t) + y'(t) + y(t) = t(t+2);$

$3z(x)z'(x) - 2x = 0.$

The system of equations is solved by analytical and numerical methods under initial conditions, $x(0)=1, y(0)=0$ in the range x from -1 to 1.

The solutions of the equations are shown in Listing 18.

```
In[326]:= DSolve[y''[x] + y[x] == Tan[x], y[x], x]

Out[326]= {{y[x] → c1 Cos[x] + Cos[x] Log[Cos[x/2] - Sin[x/2]] -
           Cos[x] Log[Cos[x/2] + Sin[x/2]] + c2 Sin[x]}}

In[327]:= DSolve[y''[x] + y[x] == 4 x Sin[x], y[x], x]

Out[327]= {{y[x] → c1 Cos[x] + c2 Sin[x] + 1/2 (-2 x^2 Cos[x] + Cos[x] Cos[2 x] -
              2 x Cos[2 x] Sin[x] + 2 x Cos[x] Sin[2 x] + Sin[x] Sin[2 x])}}

In[328]:= Simplify[%]

Out[328]= {{y[x] → (1/2 - x^2 + c1) Cos[x] + (x + c2) Sin[x]}}

In[329]:= DSolve[{x'[t] + y'[t] - t x[t] == t, x'[t] + y'[t] + y[t] == t (t + 2),
            x[0] == 1, y[0] == 0}, {x[t], y[t]}, t]

Out[329]= DSolve[{-t x[t] + x′[t] + y′[t] == t,
            y[t] + x′[t] + y′[t] == t (2 + t), x[0] == 1, y[0] == 0}, {x[t], y[t]}, t]

In[330]:= NDSolve[{x'[t] + y'[t] - t x[t] == t, x'[t] + y'[t] + y[t] == t (t + 2),
            x[0] == 1, y[0] == 0}, {x[t], y[t]}, {t, -1, 1}]

Out[330]= {{x[t] → InterpolatingFunction[ Domain: {{-1., 1.}}
                                          Output: scalar ][t],

            y[t] → InterpolatingFunction[ Domain: {{-1., 1.}}
                                          Output: scalar ][t]}}

In[331]:= DSolve[3 z[x]^2 × z'[x] - 2 x == 0, z[x], x]

Out[331]= {{z[x] → (x^2 + 3 c1)^1/3},
           {z[x] → -(-1)^1/3 (x^2 + 3 c1)^1/3}, {z[x] → (-1)^2/3 (x^2 + 3 c1)^1/3}}
```

Listing 18

Based on the listing, you can make the following comments. The solutions of the first and second equations are correct, but do not coincide with the reference data, which provide the following answers:

$$y(x) = c_1 \cos x + c_2 \sin x - \cos x \ln(\mathrm{tg}(\pi/4 + x/2)),$$

$$y(x) = c_1 \cos x + c_2 \sin x + x(\sin x - x \cos x).$$

The obtained solutions can be significantly simplified using the Simplify function [%].

The solution of the system of equations is not obtained either by analytical or numerical methods, although such a solution exists.

The answer to the solution of the third equation is interesting - three equivalent results.

When solving differential equations by numerical methods, unacceptably large errors can occur due to methodological errors and errors in choosing the integration step. It is always necessary to remember that at computer technologies of the decision of differential equations check of reliability of the received results is necessary.

Lecture № 9. Computer interpolation technologies in Mathematica environment

Interpolation, accurate in nodes

In Mathematica, interpolation, accurate in nodes, can be implemented by the following methods:

- universal;
- using the universal functions InterpolatingPolynomial and Interpolation.

Universal method

The universal method requires the solution of systems of algebraic equations, which were obtained on the basis of the data of the function, which is presented in the form of a table or matrix.$y = f(x)$

Examples of interpolation by the universal method are given below.

The function is given in the form of table. 1.$y = f(x)$

Table 1. Function in tabular form$y = f(x)$

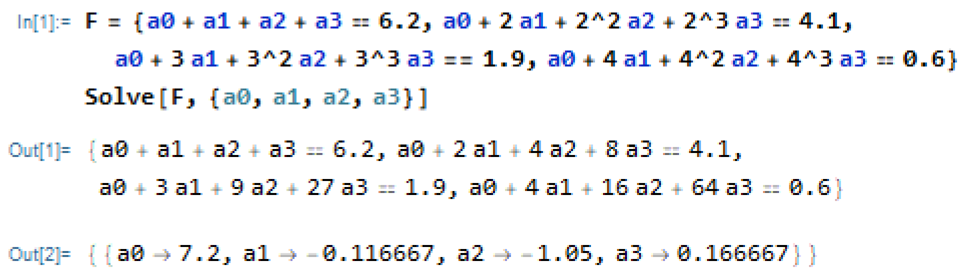
x	1	2	3	4
y	6.2	4.1	1.9	0.6

It is necessary to solve the interpolation problem, which is exact in nodes if the function is a polynomial. Since the number of nodes, the degree of the polynomial must not be higher than that is.$y = \varphi(x) n = 4 n - 1. y = a_0 + a_1 x + a_2 x^2 + a_3 x^3$

Let's make a system of equations:

$$a_0 + a_1 \cdot 1 + a_2 \cdot 1^2 + a_3 \cdot 1^3 = 6{,}2$$
$$a_0 + a_1 \cdot 2 + a_2 \cdot 2^2 + a_3 \cdot 2^3 = 4{,}1$$
$$a_0 + a_1 \cdot 3 + a_2 \cdot 3^2 + a_3 \cdot 3^3 = 1{,}9$$
$$a_0 + a_1 \cdot 4 + a_2 \cdot 4^2 + a_3 \cdot 4^3 = 0{,}6$$

The solution is obtained using the function where the initial system of equations (see Listing 1).$Solve\big(F, (a_0, a_1, a_2, a_3)\big), F -$



```
In[1]:= F = {a0 + a1 + a2 + a3 == 6.2, a0 + 2 a1 + 2^2 a2 + 2^3 a3 == 4.1,
          a0 + 3 a1 + 3^2 a2 + 3^3 a3 == 1.9, a0 + 4 a1 + 4^2 a2 + 4^3 a3 == 0.6}
        Solve[F, {a0, a1, a2, a3}]

Out[1]= {a0 + a1 + a2 + a3 == 6.2, a0 + 2 a1 + 4 a2 + 8 a3 == 4.1,
         a0 + 3 a1 + 9 a2 + 27 a3 == 1.9, a0 + 4 a1 + 16 a2 + 64 a3 == 0.6}

Out[2]= {{a0 → 7.2, a1 → -0.116667, a2 → -1.05, a3 → 0.166667}}
```

Listing 1

As a result of the received decision the interpolation formula will look:

$$y = 7{,}2 - 0.116667x - 1{,}05x^2 + 0{,}166667x^3.$$

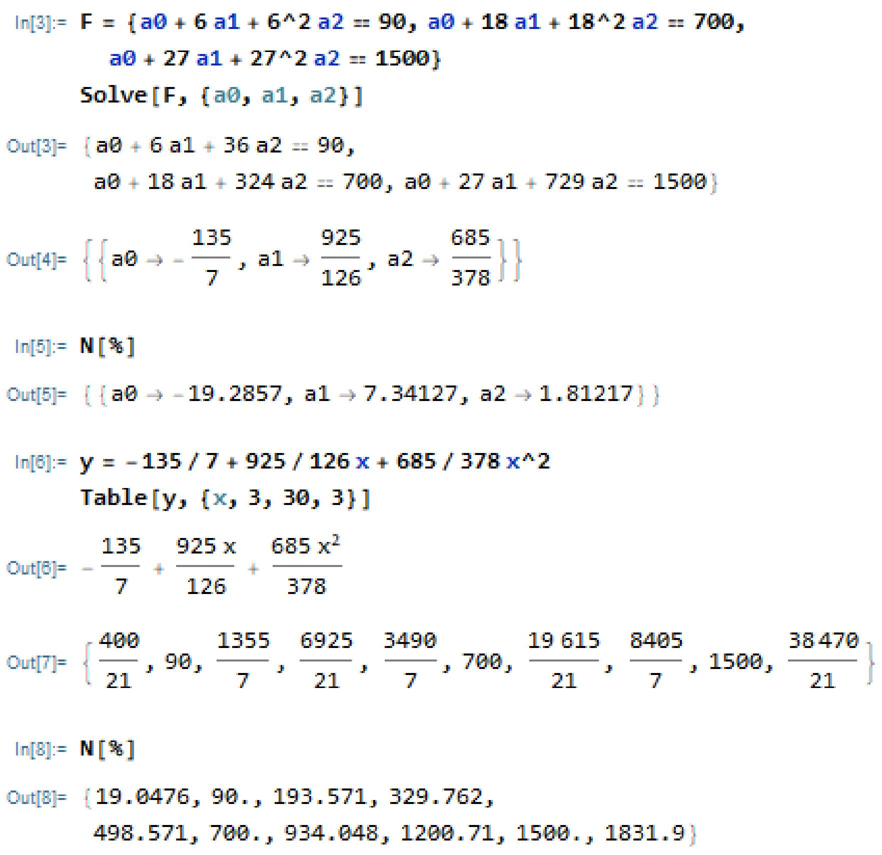
and final value of the argument, the step of the table, if, then it can be neglected.$f(x)x_H, x_K - \ h - h = 1$

In the previous example, the number of equations and the number of unknowns are the same. When solving practical problems, the number of values of the tabulated function almost always exceeds the degree of algebraic equations. In such cases, a limited number of interpolation nodes have to be selected from the entire range of source data. We illustrate this with the example of the tabulated function given in table. 2.

Table 2. Tabulated function

x	3	6	9	12	15	18	21	24	27	30
y	26	90	180	300	500	700	1000	1200	1500	2000

Let the interpolation function be a polynomial of degree. The calculation of the interpolation polynomial coefficients is shown in Listing 2. As a result of comparing the interpolation results with the original data, we can conclude that there is an interpolation error at some values of the argument. For a detailed analysis of the error in Fig. 1 constructs interpolation and data functions.$y = a_0 + a_1 x + a_2 x^2 x$



```
In[3]:= F = {a0 + 6 a1 + 6^2 a2 == 90, a0 + 18 a1 + 18^2 a2 == 700,
           a0 + 27 a1 + 27^2 a2 == 1500}
        Solve[F, {a0, a1, a2}]

Out[3]= {a0 + 6 a1 + 36 a2 == 90,
         a0 + 18 a1 + 324 a2 == 700, a0 + 27 a1 + 729 a2 == 1500}

                    135         925         685
Out[4]= {{a0 -> - ----, a1 -> -----, a2 -> -----}}
                     7          126         378

In[5]:= N[%]

Out[5]= {{a0 -> -19.2857, a1 -> 7.34127, a2 -> 1.81217}}

In[6]:= y = -135 / 7 + 925 / 126 x + 685 / 378 x^2
        Table[y, {x, 3, 30, 3}]

           135   925 x   685 x^2
Out[6]= - ---- + ----- + -------
            7     126      378

          400       1355   6925   3490        19 615   8405          38 470
Out[7]= {----, 90, -----, -----, -----, 700, ------, -----, 1500, ------}
           21         7     21      7           21       7             21

In[8]:= N[%]

Out[8]= {19.0476, 90., 193.571, 329.762,
         498.571, 700., 934.048, 1200.71, 1500., 1831.9}
```

Listing 2

```
In[13]:= f1 = {{3, 26}, {6, 90}, {9, 180}, {12, 300}, {15, 500},
           {18, 700}, {21, 1000}, {24, 1200}, {27, 1500}, {30, 2000}};
         r1 = ListPlot[f1, AxesLabel → {"x", "y"},
           PlotStyle → PointSize[0.02]];
         r2 = Plot[-135 / 7 + 925 / 126 x + 685 / 378 x^2, {x, 2, 40}];
         Show[{r1, r2}]
```

Out[16]=

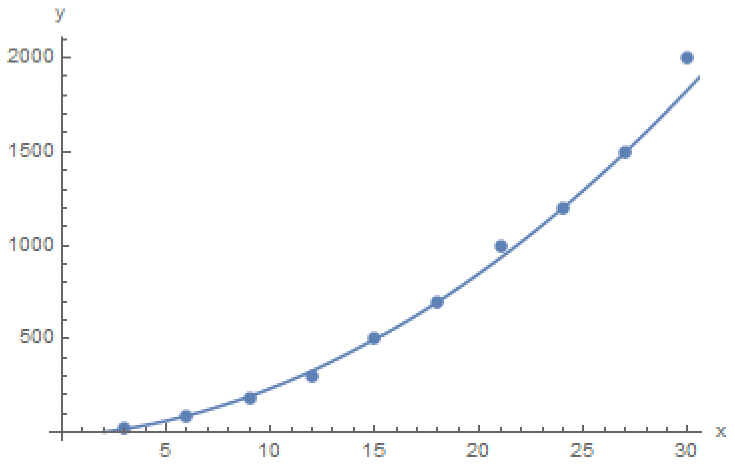
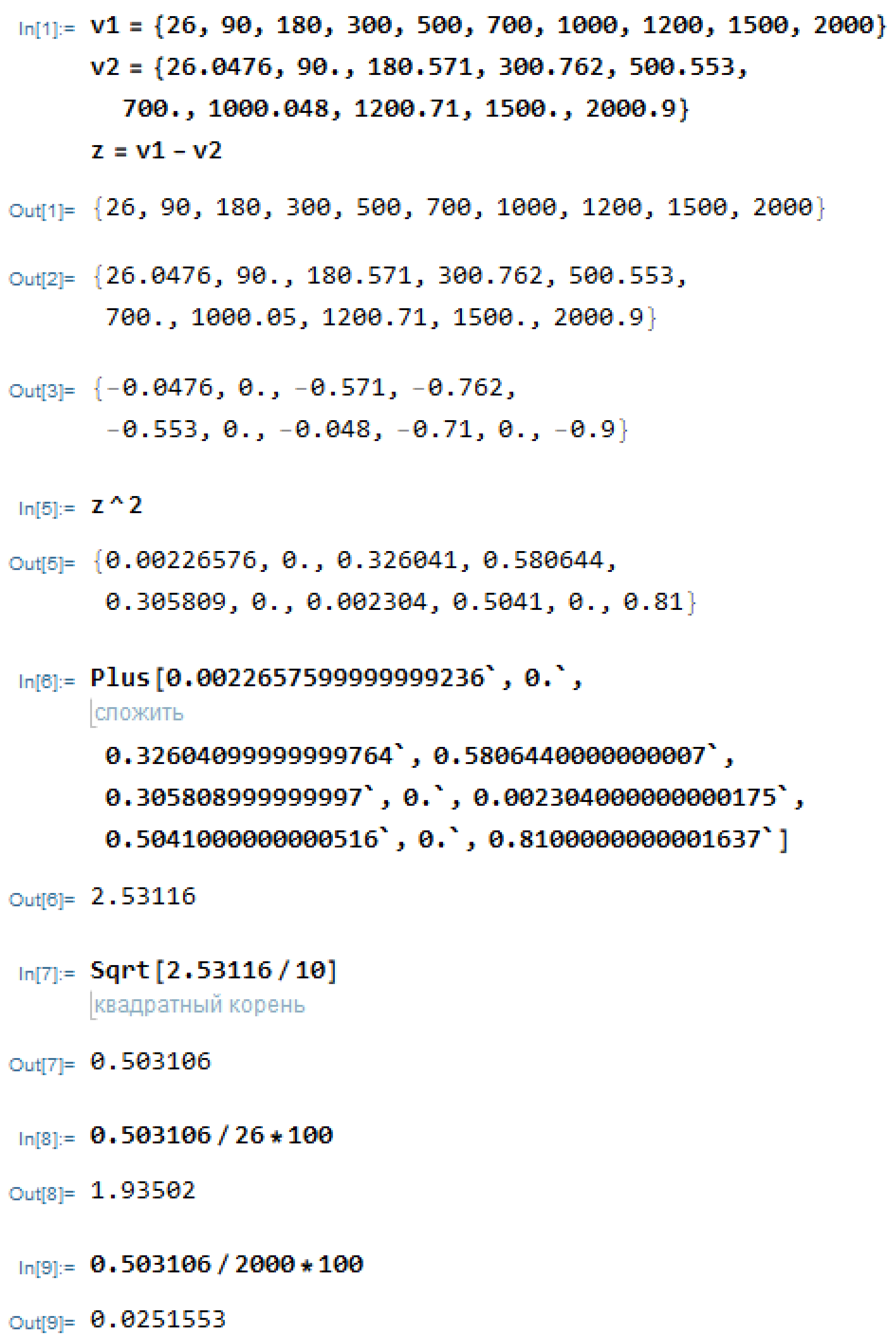


Fig. 1. Graphs of the original function and tabs

Calculate the absolute ε and relative δ RMS interpolation error by the following formulas:

$$\varepsilon = \sqrt{\frac{\sum_{i=1}^{n} \Delta_i^2}{n}}, \delta = \frac{\varepsilon}{y_{\min}} \cdot 100\% ,, \quad (1)$$

where Δi = y (xi) - φ (xi) is the difference between the values of the tabulated function y (xi) and the interpolation function φ (xi),
n is the number of values of the tabulated function,
ymin - the minimum value of the function y (x).
The calculation of absolute and relative values and errors is shown in Listing 3.



```
In[1]:= v1 = {26, 90, 180, 300, 500, 700, 1000, 1200, 1500, 2000}
        v2 = {26.0476, 90., 180.571, 300.762, 500.553,
          700., 1000.048, 1200.71, 1500., 2000.9}
        z = v1 - v2

Out[1]= {26, 90, 180, 300, 500, 700, 1000, 1200, 1500, 2000}

Out[2]= {26.0476, 90., 180.571, 300.762, 500.553,
         700., 1000.05, 1200.71, 1500., 2000.9}

Out[3]= {-0.0476, 0., -0.571, -0.762,
         -0.553, 0., -0.048, -0.71, 0., -0.9}

In[5]:= z^2

Out[5]= {0.00226576, 0., 0.326041, 0.580644,
         0.305809, 0., 0.002304, 0.5041, 0., 0.81}

In[6]:= Plus[0.0022657599999999236`, 0.`,
        сложить
         0.32604099999999764`, 0.5806440000000007`,
         0.305808999999997`, 0.`, 0.0023040000000000175`,
         0.5041000000000516`, 0.`, 0.8100000000001637`]

Out[6]= 2.53116

In[7]:= Sqrt[2.53116/10]
        квадратный корень

Out[7]= 0.503106

In[8]:= 0.503106/26*100

Out[8]= 1.93502

In[9]:= 0.503106/2000*100

Out[9]= 0.0251553
```

Listing 3

FunctionInterpolatingPolynomial

To interpolate polynomial functions use a function that has the following format:$InterpolatingPolynomial$

$InterpolatingPolynomial\ [z, x]$,,

where$z$ –source data matrix,

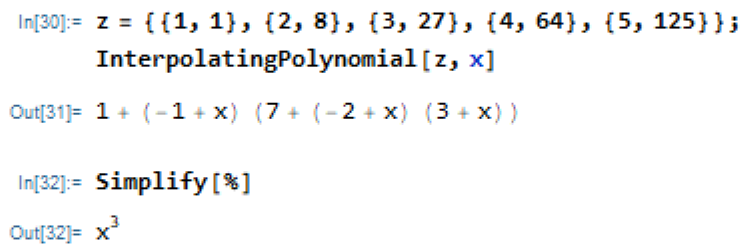
x –function argument z.

An example of using the function will be demonstrated by the example of the data given in table. 3.$InterpolatingPolynomial\ [z, x]$

Table 3. Tabular data

X	1	2	3	4	5
Y	1	8	27	64	125

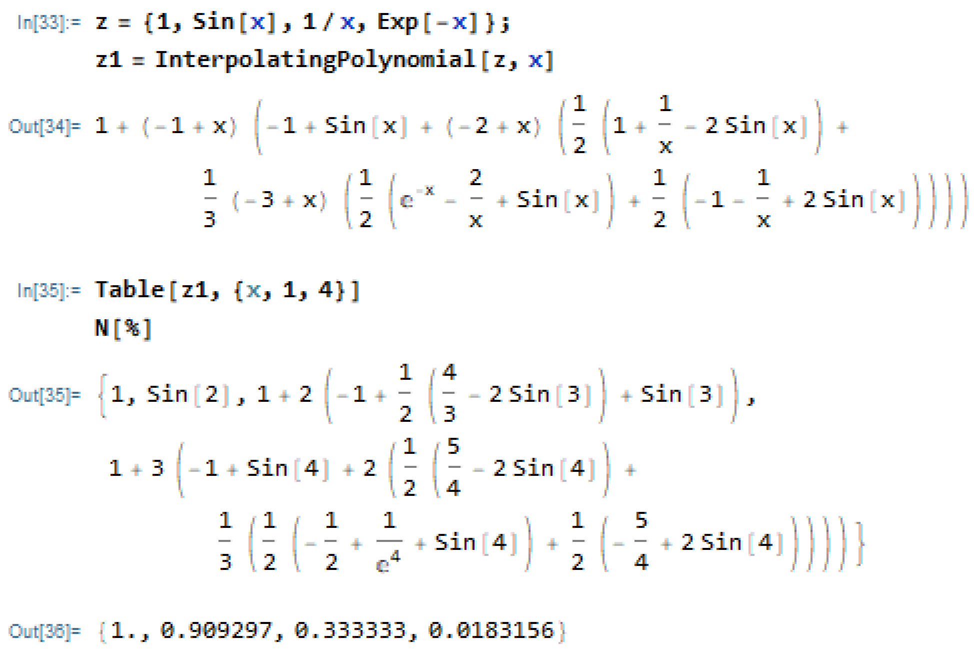
The method of using the function is shown in Listing 4. According to the listing, the interpolation function has the form $y = x^3$. The solution is accurate. The function is used to simplify the expression *Simplify* .



```
In[30]:= z = {{1, 1}, {2, 8}, {3, 27}, {4, 64}, {5, 125}};
         InterpolatingPolynomial[z, x]

Out[31]= 1 + (-1 + x) (7 + (-2 + x) (3 + x))

In[32]:= Simplify[%]

Out[32]= x^3
```

Listing 4

Function $InterpolatingPolynomial\,[z, x]$ solves interpolation problems also in the case of non-equidistant nodes. Vector z elements chan be a set of functions $f_1(x)$, $f_2(x), \ldots$. The solution will then be issued as a polynomial with the original expression stored $f_1(x), f_2(x), \ldots$, Listing 5.



```
In[33]:= z = {1, Sin[x], 1 / x, Exp[-x]};
         z1 = InterpolatingPolynomial[z, x]

Out[34]= 1 + (-1 + x) (-1 + Sin[x] + (-2 + x) (1/2 (1 + 1/x - 2 Sin[x]) +
           1/3 (-3 + x) (1/2 (e^-x - 2/x + Sin[x]) + 1/2 (-1 - 1/x + 2 Sin[x]))))

In[35]:= Table[z1, {x, 1, 4}]
         N[%]

Out[35]= {1, Sin[2], 1 + 2 (-1 + 1/2 (4/3 - 2 Sin[3]) + Sin[3]),
          1 + 3 (-1 + Sin[4] + 2 (1/2 (5/4 - 2 Sin[4]) +
            1/3 (1/2 (-1/2 + 1/e^4 + Sin[4]) + 1/2 (-5/4 + 2 Sin[4])))))}

Out[36]= {1., 0.909297, 0.333333, 0.0183156}
```

Listing 5

The values of the function z are shown in table. 4.

Table 4. The value of the function

x	1	2	3	4
y	1	*sIInx*	$1/x$	e^{-x}
$y(x)$	1	0,909297	0,333333	0,0183156

Function $Interpolating$[data]

This function allows you to solve the problem of interpolation over data (data) in the range of arguments given by this data. The approximation function is unknown to the user. The data are given in the form of a matrix of the function $y = f(x)$. When you enter this function, Mathematica does not issue an interpolation function, but a new function:

$InterpolatingFunction[\{\{x_H, x_K\}\}, < >]$,

where x_H, x_K − the range of arguments of the interpolation function.

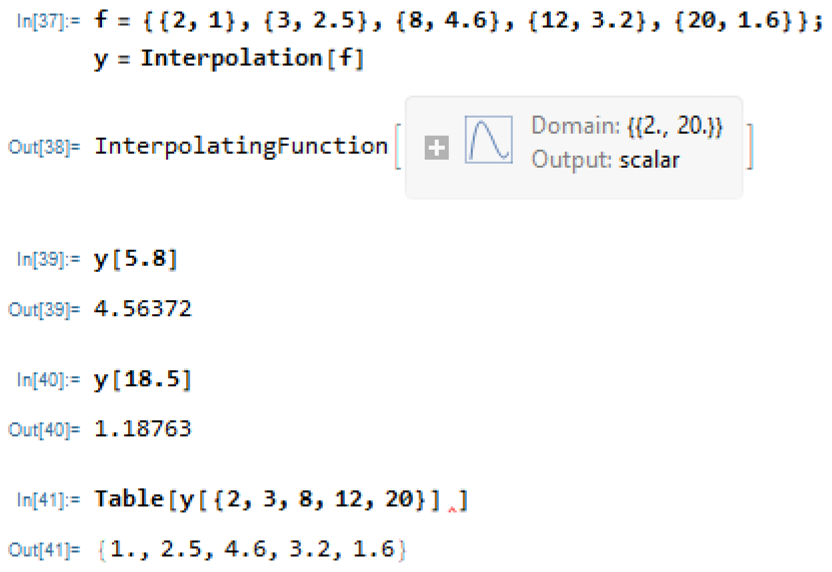
If yu nov enter the value of the argument from the range $x_H - x_K$, the answer will be the value of the function at a given value of the argument.

Consider the method of using the function by example. Tabular function $y = f(x)$ listed in table. 5. It is necessary to determine the value of the function at x= 5.8 i x = 18.5 and verify the validity of the obtained solutions.

Table 5. Function $y = ff(x)$ in tabular form

X	2	3	8	12	20
Y	1	2,5	4,6	3,2	1,6

The solution to the problem is shown in the Listing 6.



```
In[37]:= f = {{2, 1}, {3, 2.5}, {8, 4.6}, {12, 3.2}, {20, 1.6}};
         y = Interpolation[f]

Out[38]= InterpolatingFunction[ Domain: {{2., 20.}}
                                Output: scalar ]

In[39]:= y[5.8]

Out[39]= 4.56372

In[40]:= y[18.5]

Out[40]= 1.18763

In[41]:= Table[y[{2, 3, 8, 12, 20}]]

Out[41]= {1., 2.5, 4.6, 3.2, 1.6}
```

Listing 6

According to Listing 6, the function $Interpolation[data]$ interpolates data with the exact method in nodes. The results of tabulation are the exact values of the function in the interpolation nodes.

Interpolation by nonlinear functions

If the interpolation function is nonlinear, then two methods are used to determine its coefficients by the exact method in nodes:

- creation and solution of a system of nonlinear equations;

linearization of nonlinear interpolation function by coordinate transformation.

Consider both methods.

Way 1. Solving a system of nonlinear equations

The method of interpolation in this way is to solve systems of nonlinear equations. Demonstrate it on the basis of the data given in table. 6.

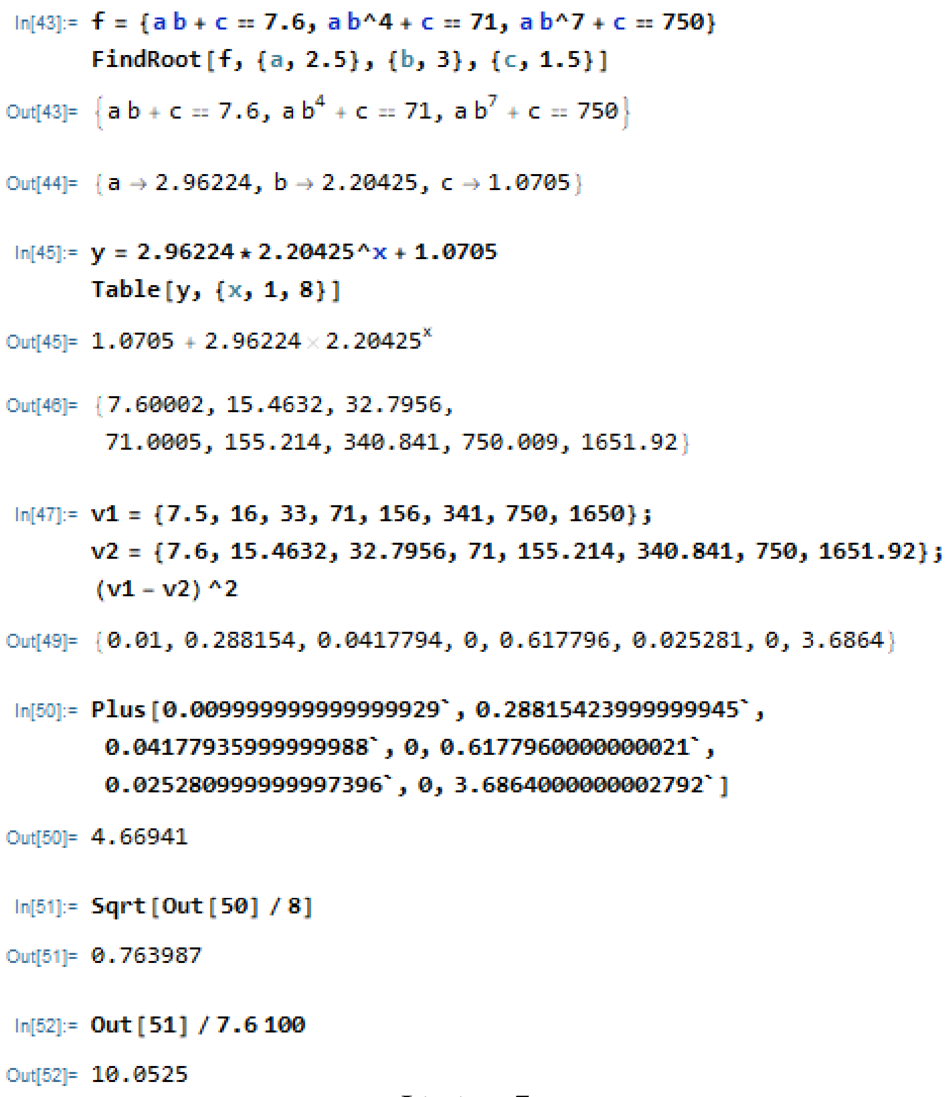
Table 6. Function $y = f(x)$ in tabular form

X	1	2	3	4	5		6	7	8
У	7,6	16	33	71	156		341	750	1650

The interpolation function has an analytical expression $y = ab^x + c$. It is necessary to determine the unknown a, b, c and the error of the interpolation function. Choose the following three coordinates of the function: (1,7.6), (4,71), (7, 750) and make a system of equations:

$$ab^1 + c = 7{,}6$$
$$ab^4 + c = 71$$
$$ab^7 + c = 750$$

We solve this system of nonlinear equations using a function FindRoot according to the following initial values of the unknown: $a = 2{,}5$, $b = 3$, $c = 1{,}5$. The solution is shown in the Listing 7.



```
In[43]:= f = {a b + c == 7.6, a b^4 + c == 71, a b^7 + c == 750}
         FindRoot[f, {a, 2.5}, {b, 3}, {c, 1.5}]

Out[43]= {a b + c == 7.6, a b^4 + c == 71, a b^7 + c == 750}

Out[44]= {a → 2.96224, b → 2.20425, c → 1.0705}

In[45]:= y = 2.96224 * 2.20425^x + 1.0705
         Table[y, {x, 1, 8}]

Out[45]= 1.0705 + 2.96224 × 2.20425^x

Out[46]= {7.60002, 15.4632, 32.7956,
          71.0005, 155.214, 340.841, 750.009, 1651.92}

In[47]:= v1 = {7.5, 16, 33, 71, 156, 341, 750, 1650};
         v2 = {7.6, 15.4632, 32.7956, 71, 155.214, 340.841, 750, 1651.92};
         (v1 - v2) ^2

Out[49]= {0.01, 0.288154, 0.0417794, 0, 0.617796, 0.025281, 0, 3.6864}

In[50]:= Plus[0.009999999999999929`, 0.28815423999999945`,
          0.04177935999999988`, 0, 0.6177960000000021`,
          0.025280999999997396`, 0, 3.6864000000002792`]

Out[50]= 4.66941

In[51]:= Sqrt[Out[50] / 8]

Out[51]= 0.763987

In[52]:= Out[51] / 7.6 100

Out[52]= 10.0525
```

Listing 7

Listing 7 shows that the mathematical model of the object is the following interpolation function:

$y = 2.96 \cdot 2.2^x + 1.07.$

The absolute and relative root mean square errors are calculated according to formula (1). The relative error is equal to 10%.

Way 2. Linearization of a nonlinear function

In Mathematica, approximation by nonlinear functions can be reduced to solving linear equations by coordinate transformation.

The alignment of functions is carried out by converting them into a linear function by replacing variables. These transformations are most simply carried out under the condition of using power, logarithmic, fractional-linear, exponential functions. The technique of interpolation by the method of linearization of nonlinear functions is carried out by performing the following actions:

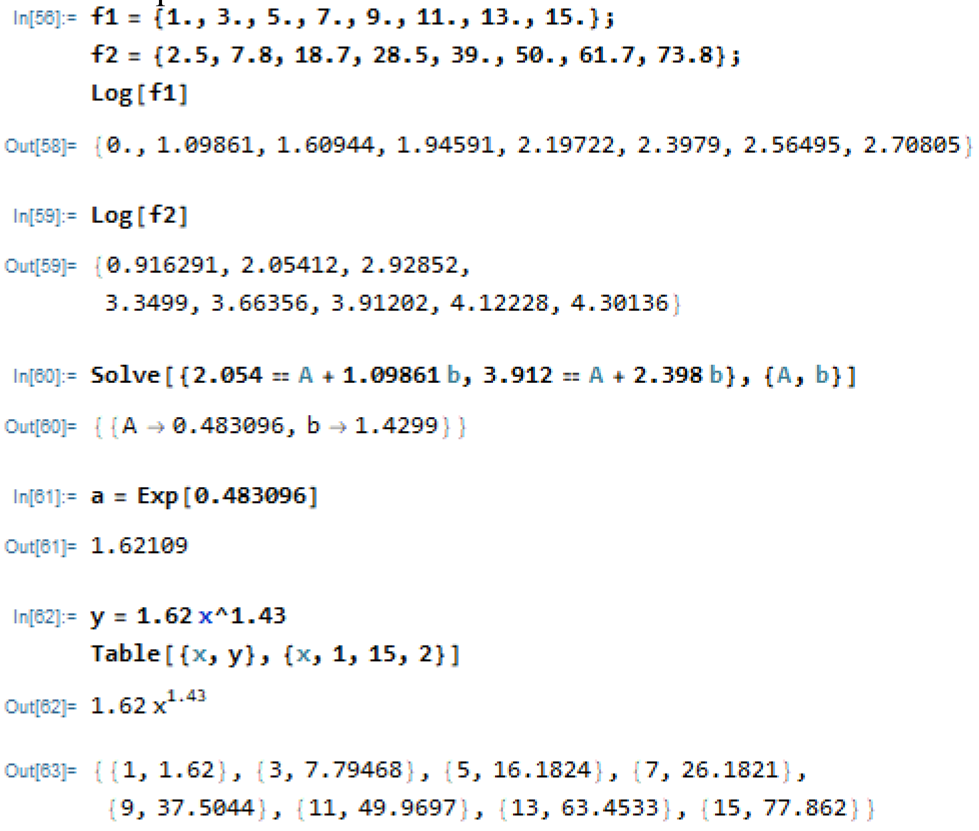
1. Transformation of the interpolation function into a linear form.
2. Transformation of the matrix of source data into a matrix of new variables.
3. Formation of a system of linear equations.
4. Solving a system of linear equations.
5. Formation of the interpolation function.
6. Checking the adequacy of the obtained model.

Consider the method on the example of the data given in table. 7.

Table 7. Function $y = f(x)$ in tabular form

X	1	3	5	7	9	11	13	15
Y	2,5	7,8	18,7	28,5	39	50	61,7	73,8

It is necessary to find a mathematical model of the object under study, if it is known that the interpolation function is an exponential function $y = axb$. The solution of the problem is shown in the Listing 8.



```
In[56]:= f1 = {1., 3., 5., 7., 9., 11., 13., 15.};
         f2 = {2.5, 7.8, 18.7, 28.5, 39., 50., 61.7, 73.8};
         Log[f1]

Out[58]= {0., 1.09861, 1.60944, 1.94591, 2.19722, 2.3979, 2.56495, 2.70805}

In[59]:= Log[f2]

Out[59]= {0.916291, 2.05412, 2.92852,
          3.3499, 3.66356, 3.91202, 4.12228, 4.30136}

In[60]:= Solve[{2.054 == A + 1.09861 b, 3.912 == A + 2.398 b}, {A, b}]

Out[60]= {{A → 0.483096, b → 1.4299}}

In[61]:= a = Exp[0.483096]

Out[61]= 1.62109

In[62]:= y = 1.62 x^1.43
         Table[{x, y}, {x, 1, 15, 2}]

Out[62]= 1.62 x^1.43

Out[63]= {{1, 1.62}, {3, 7.79468}, {5, 16.1824}, {7, 26.1821},
          {9, 37.5044}, {11, 49.9697}, {13, 63.4533}, {15, 77.862}}
```

Listing 8

The first two lines of Listing 8 contain the source data vectors with the name $f1$ and $f2$. They are followed by the same vectors on a logarithmic scale. To calculate the coefficients a and b of a linear function $lny = lna + blnx$. Coordinates are taken as initial data (3,7), (11,50), which on a logarithmic scale matter: (1.09861, 2.05412), (2.3979, 3.91202). Then the system of equations has the form:

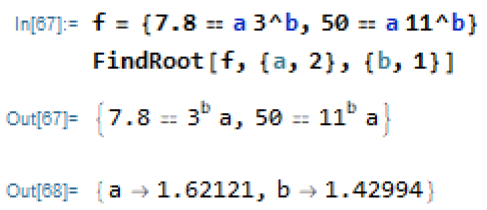
$2.05412 = A \cdot 1.098616^{b};$

$3.91202 = A \cdot 2.3979^{b},$

where $A = lna$.

The system is solved by a function *Solve*. As a result of the solution, the following values of the coefficients are obtained: $a = 0.483096$, $b = 1.4299$. Then $a = e^{AA} = 1.62109$, and the interpolation function has the form: $y = 1.62x^{1.43}$. The coefficients a and b of the interpolation function are rounded to two significant digits after the dot. The adequacy of the model is proved by tabulating the interpolation function using the function T*able*. Comparing the results of tabulation with the original data, we can conclude that the problem is solved correctly (the values of the functions are almost the same in the interpolation nodes $x = 3$ and $x = 11$), and the interpolation function is a mathematical model of the object under study. In order to compare the results of the two interpolation methods, the problem of interpolation of a nonlinear function was solved $y = ax^{b}$. The same interpolation nodes were selected $x = 3$ i $x = 11$ and a system of nonlinear equations is solved by means of a function *Findroot* on initial approximations $a_0 = 2$, $b_0 = 1$.

The solution looks like this:



```
In[67]:= f = {7.8 == a 3^b, 50 == a 11^b}
         FindRoot[f, {a, 2}, {b, 1}]

Out[67]= {7.8 == 3^b a, 50 == 11^b a}

Out[68]= {a → 1.62121, b → 1.42994}
```

According to the solution, the coefficients of the interpolation function practically coincide in the case of linearization of the equation.

Interpolation approximated in nodes

Interpolation, approximated in nodes (approximation), is carried out by the criterion of minimum standard error (least squares method). Implemented by the Mathematica system using the function Fit . Function Fit looks like:

$Fit[\{\{M\}\}, \{XX\}, x]$,

where M - is the matrix of the original data,

XX - list of basic variables,

x - is the argument of the function.

Method of interpolation using a function $Fit[\{\{M\}\}, \{XX\}, x]$ is implemented by performing the following actions:

1. 1. Introduction of a matrix of initial data with assignment to it of the unique name, for example, M.
2. 2. Introduction of basic variables X.
3. Enter the function $Fit[\{\{M\}\}, \{XX\}, x]$.

An example of solving interpolation problems will be shown in the data given in Table 8.

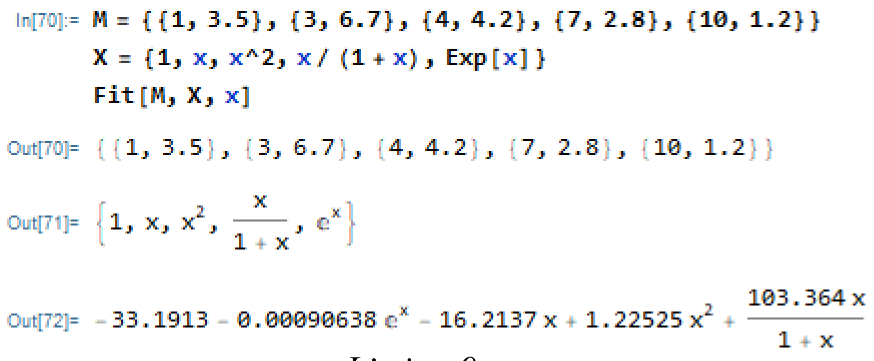
Table 8. Function in tabular form$y = f(x)$

x	1		3	4	7	10
Y	3.5		6.7	4.2	2.8	1.2

It is necessary to find the interpolation function in the basis. In this example, the function will look like:$a, x, x^2, \frac{x}{1+x}, e^x Fit$

$$Fit[\{\{1,3.5\},\{3,6.7\},\{4,4.2\},\{7,2.8\},\{10,1.2\}\},\left\{a,x,x^2,\frac{x}{1+x},Exp[x]\right\},x]$$

The solution is shown in the Listing 9.



```
In[70]:= M = {{1, 3.5}, {3, 6.7}, {4, 4.2}, {7, 2.8}, {10, 1.2}}
         X = {1, x, x^2, x / (1 + x), Exp[x]}
         Fit[M, X, x]

Out[70]= {{1, 3.5}, {3, 6.7}, {4, 4.2}, {7, 2.8}, {10, 1.2}}

Out[71]= {1, x, x^2, x/(1 + x), e^x}

Out[72]= -33.1913 - 0.00090638 e^x - 16.2137 x + 1.22525 x^2 + 103.364 x/(1 + x)
```

Listing 9

Pade approximation

The Padé approximation is used to interpolate a function given analytically by a fractional-rational function.

The Pade function has the form:

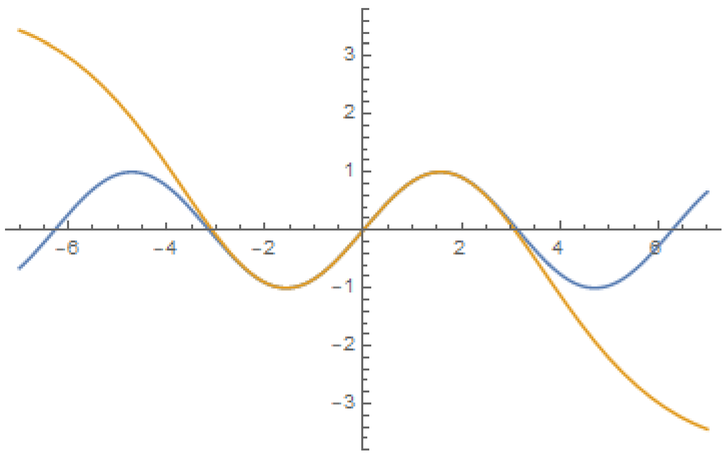
$PadeApproximant[f(x), \{x, r, \{n_1, n_2\}\}]$;

where $f\ (x)$ –function, given in analytical form,
x –argument of the function $f(x)$,
r –point near which the approximation function is valid,
$n1$ –degree of the polynomial of the numerator,
$n2$ –degree of the polynomial of the denominator.
Perform the Pade approximation of the function $y = sinx$ near $x = 0$ for $n1 = 3$ and $n_2 = 4$. The approximation procedure in Mathematica is shown in the Listing 10.

```
In[75]:= f = Sin[x]

Out[75]= Sin[x]

In[77]:= PadeApproximant[f, {x, 0, {3, 4}}]
```

Out[77]=
$$\frac{x - \frac{31 x^3}{294}}{1 + \frac{3 x^2}{49} + \frac{11 x^4}{5880}}$$

```
In[78]:= Plot[{f, Out[77]}, {x, -7, 7}]
```

Out[78]=


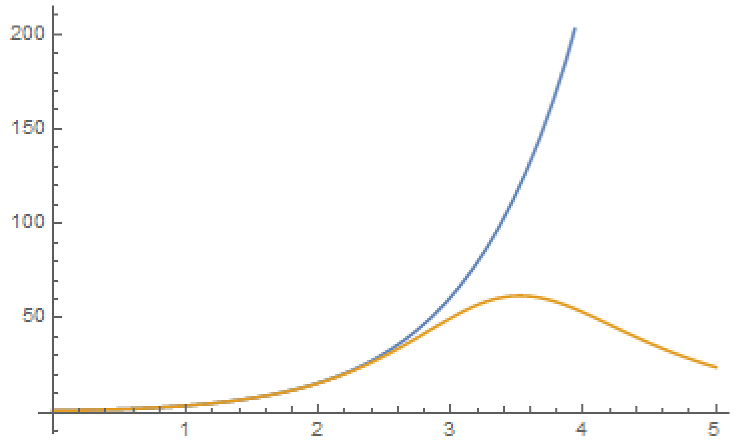
Listing 10

According to Listing 10, the approximation is performed correctly. In the range from $x = -3$ to $x = 3$ the functions coincide. The Padé approximation has no restrictions on the form of the original function, as shown in the following example. It is necessary to perform a Pade approximation of the function $y = xe^x + 1$ near $x = 0$ for $n1 = 3$ and $n2 = 4$. Make sure that the approximation is certain by plotting. The result of the approximation is shown in the Listing 11.

```
In[79]:= f1 = x Exp[x] + 1
Out[79]= 1 + e^x x
In[80]:= PadeApproximant[f1, {x, 0, {3, 4}}]
```

$$\text{Out[80]=}\ \frac{1+\frac{x}{4}+\frac{x^2}{2}-\frac{11x^3}{240}}{1-\frac{3x}{4}+\frac{x^2}{4}-\frac{11x^3}{240}+\frac{x^4}{240}}$$

```
In[81]:= Plot[{f1, Out[80]}, {x, 0, 5}]
```

Out[81]=


Listing 11

Spline interpolation

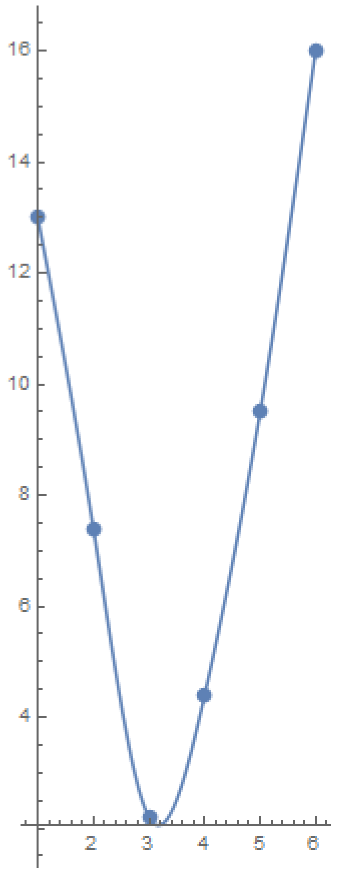
The spline interpolation implementation function is located in the SplineFit subpackage of the NumericalMath package. Cubic spline interpolation provides high accuracy of the mathematical model. Before use, you must connect this package with the following line: << Splines` або Needs["Splines`"]. The function has the following format:
SplineFit[F, type],
Where F is the data presented as a matrix,
type - type of approximation, by default - approximation by cubic splines (Cubic). Approximation with Bezier and CompositeBezier splines is also possible.

Consider the use of a function on the example of a matrix:M=((1,13), (2, 7.4), (3, 2.2), (4, 4.4), (5, 9.5), (6, 16)).

It is necessary to obtain a mathematical model of the function $y = f(x)$, using cubic spline interpolation. Check the reliability of the decision graphically.

The solution is shown in the Listing 12.

```
In[82]:= M = {{1, 13}, {2, 7.4}, {3, 2.2}, {4, 4.4}, {5, 9.5}, {6, 16}}

Out[82]= {{1, 13}, {2, 7.4}, {3, 2.2}, {4, 4.4}, {5, 9.5}, {6, 16}}

In[93]:= << Splines`
         Z1 = SplineFit[M, Cubic]

Out[94]= SplineFunction[Cubic, {0., 5.}, <>]

In[95]:= Z1[0]

Out[95]= {1, 13}

In[96]:= Z1[1]
         Z1[5]

Out[96]= {2., 7.4}

Out[97]= {6., 16.}

In[98]:= Z1[3.53]

Out[98]= {4.53, 6.92269}

In[102]:= r1 = ParametricPlot[Z1[x], {x, 0, 5}];
          r2 = ListPlot[M, PlotStyle → {PointSize[0.05]}];
          Show[r1, r2]

Out[104]=
```



Listing 12

Listing 12 shows that the solution is not explicit, ie there is no analytical expression for the interpolation function. This is a significant drawback functions SplineFit[F, type]. From the graph we can conclude that the interpolation is performed quite accurately.

Lecture № 10. Functions of spectral analysis of Mathematica system

The Mathematica system has rich capabilities of spectral analysis and signal synthesis. A large number of built-in functions allows you to perform:

- spectral analysis of signals;
- harmonic analysis of signals;
- signal filtering.

The Fourier transform is performed in the Mathematica system using the following functions:

- **FourierTransform [F (t), t, w]** - returns the result of the direct Fourier transform of the function F (t), represented by the parameter w;
- **InverseFourierTransform [F (w), w, t]** - returns the result of the inverse Fourier transform of the expression F (w), which is represented by the parameter t;
- **FourierSinTransform [F (t), t, w]** - returns the result of the sine Fourier transform of the function F (t), represented by the parameter w;
- **FourierCosTransform [F (t), t, w]** - returns the result of the cosine Fourier transform of the function F (t), represented by the parameter w;
- **FourierTransform [F, {t1, t2,...}, {w1, w2,...}]** - returns the result of the direct Fourier transform of the function F {t1, t2,…}, which is represented by the parameters {w1, w2,…};
- **InverseFourierTransform [F, {w1, w2,...}, {t1, t2,...}]** - returns the result of the inverse Fourier transform of the expression F {w1, w2,…}, which is represented by the parameters {t1, t2,…}.

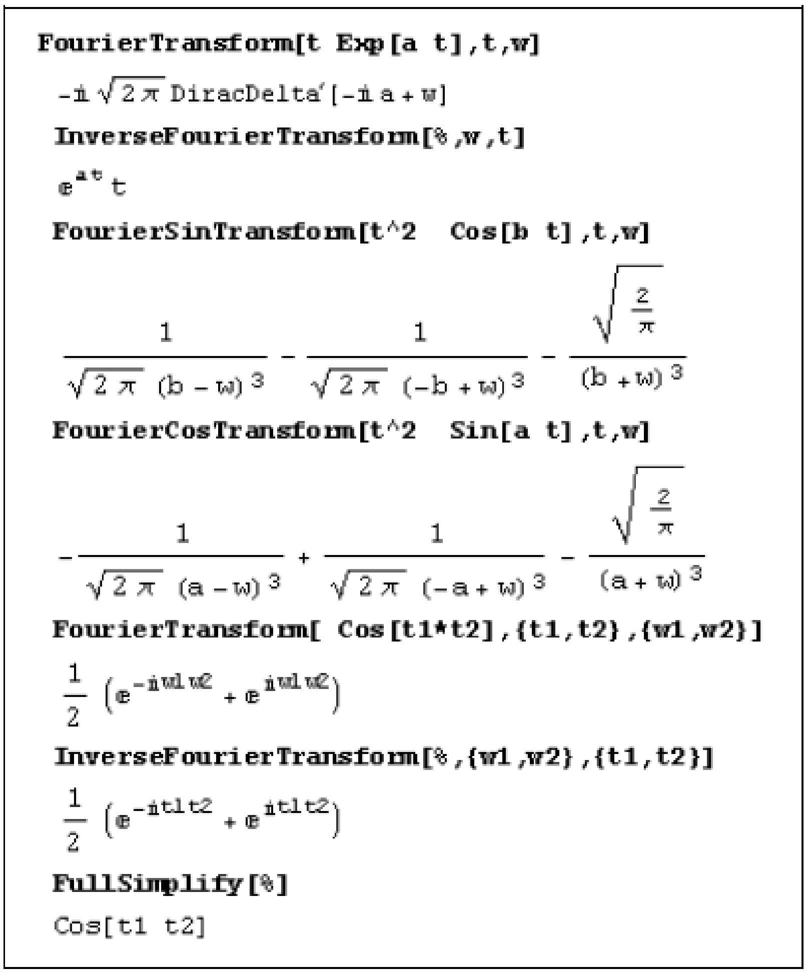
It is necessary to obtain a direct and inverse Fourier transform of the following functions:

$F(t) = te^{\alpha t}$;

$F(t) = t^2 \cos(bt)$;

$F(t) = t^2 \sin(at)$;

$F(t_1,t_2) = \cos(t_1,t_2)$.



```
FourierTransform[t Exp[a t],t,w]
```

$$-i\sqrt{2\pi}\,\text{DiracDelta}'[-i\,a+w]$$

```
InverseFourierTransform[%,w,t]
```

$$e^{a t}\,t$$

```
FourierSinTransform[t^2  Cos[b t],t,w]
```

$$\frac{1}{\sqrt{2\pi}\,(b-w)^3}-\frac{1}{\sqrt{2\pi}\,(-b+w)^3}-\frac{\sqrt{\frac{2}{\pi}}}{(b+w)^3}$$

```
FourierCosTransform[t^2  Sin[a t],t,w]
```

$$-\frac{1}{\sqrt{2\pi}\,(a-w)^3}+\frac{1}{\sqrt{2\pi}\,(-a+w)^3}-\frac{\sqrt{\frac{2}{\pi}}}{(a+w)^3}$$

```
FourierTransform[ Cos[t1*t2],{t1,t2},{w1,w2}]
```

$$\frac{1}{2}\left(e^{-i\,w1\,w2}+e^{i\,w1\,w2}\right)$$

```
InverseFourierTransform[%,{w1,w2},{t1,t2}]
```

$$\frac{1}{2}\left(e^{-i\,t1\,t2}+e^{i\,t1\,t2}\right)$$

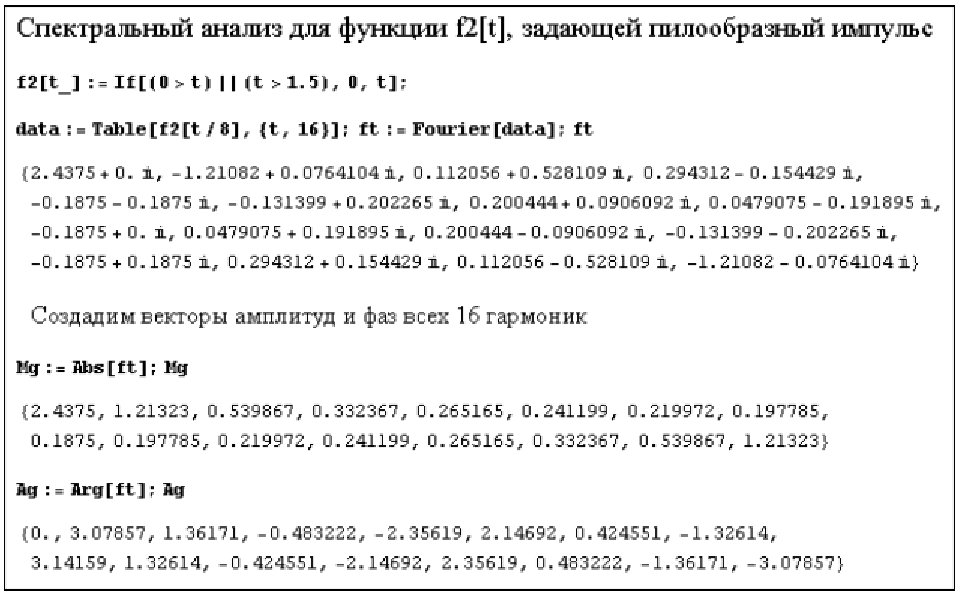
```
FullSimplify[%]
Cos[t1 t2]
```

Listing 1

Listing 1 shows that the transformations are performed correctly, because the inverse Fourier transforms coincide with the original functions.

Spectral analysis based on direct Fourier transform. Direct Fourier transform allows to obtain the frequency spectrum of the signal represented by samples of its time dependence. This is often the ultimate goal of spectral analysis.

In fig. 10.1 presents an example of spectral analysis of a simple signal - a triangular pulse given by the function If. Then, using the Fourier direct Fourier transform function, the vectors of the amplitudes Mg and the phases Ag of the harmonics of this signal are explicitly obtained.



Спектральный анализ для функции f2[t], задающей пилообразный импульс

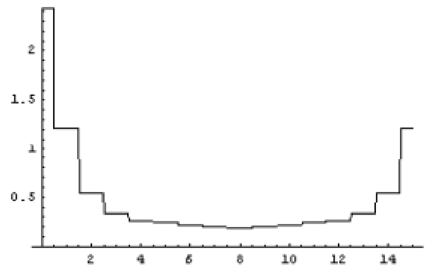
```
f2[t_] := If[(0 > t) || (t > 1.5), 0, t];

data := Table[f2[t / 8], {t, 16}]; ft := Fourier[data]; ft

{2.4375 + 0. i, -1.21082 + 0.0764104 i, 0.112056 + 0.528109 i, 0.294312 - 0.154429 i,
 -0.1875 - 0.1875 i, -0.131399 + 0.202265 i, 0.200444 + 0.0906092 i, 0.0479075 - 0.191895 i,
 -0.1875 + 0. i, 0.0479075 + 0.191895 i, 0.200444 - 0.0906092 i, -0.131399 - 0.202265 i,
 -0.1875 + 0.1875 i, 0.294312 + 0.154429 i, 0.112056 - 0.528109 i, -1.21082 - 0.0764104 i}
```

Создадим векторы амплитуд и фаз всех 16 гармоник

```
Mg := Abs[ft]; Mg

{2.4375, 1.21323, 0.539867, 0.332367, 0.265165, 0.241199, 0.219972, 0.197785,
 0.1875, 0.197785, 0.219972, 0.241199, 0.265165, 0.332367, 0.539867, 1.21323}

Ag := Arg[ft]; Ag

{0., 3.07857, 1.36171, -0.483222, -2.35619, 2.14692, 0.424551, -1.32614,
 3.14159, 1.32614, -0.424551, -2.14692, 2.35619, 0.483222, -1.36171, -3.07857}
```

Построим спектрограмму модулей гармоник, обеспечив верное отражение нулевой гармоники.

```
Plot[Mg[[Round[i] + 1]] , {i, 0, 15}, PlotPoints → 16, PlotRange → {0, Mg[[1]]}]
```



Построим спектрограмму фаз гармоник.

```
Plot[Ag[[Round[i] + 1]], {i, 0, 15}, PlotPoints → 16, PlotRange → {-π, π}]
```

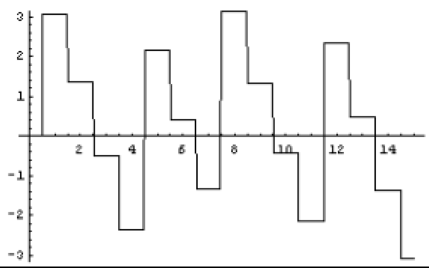


Fig. 10.1. Example of spectral analysis of a triangular pulse

Here, with the help of ladder-type graphs, the discreteness of the harmonics is emphasized, and the spectrograms of the amplitudes and phases of the harmonics of the sawtooth pulse are constructed. The symmetrical mapping of the spectral lines relative to the eighth harmonic is clearly visible - in our case there were 16 samples of the signal. This means that the amplitude and phase of the ninth harmonic are the same as in the seventh harmonic, in the tenth - the same as in the sixth, etc.

Lecture № 11. Fundamentals of programming in Mathematica

Mathematica as a programming system

The concept of input system language and implementation language

Mathematica is able to solve many problems without the use of programming. However, all system tools (alphabet, letters, numbers, operators, and special characters) are part of a problem-oriented, high-level programming language. According to its capabilities in performing mathematical and scientific and technical calculations, this language has significant advantages over conventional programming languages - Pascal, C, C ++.

Mathematica programming language capabilities

The Mathematica system contains a large number of functions, many of which implement mathematical transformations and modern computational methods, both numerical and analytical. The Mathematica programming language is a default interpreter and is not intended for creating executable files. However, individual expressions can be compiled using the Compile function, which is useful when you need to increase the speed of calculations. The language of the Mathematica system allows you to implement most types of programming: functional, structural, object-oriented, mathematical, logical, recursive, etc.

The core of the Mathematica system has a functional structure. The language of the system allows you to break programs into separate modules (blocks), procedures and functions with local variables. Object-oriented programming is based on a generalized concept of an object. In the Mathematica system, objects can be mathematical expressions, input and output data, graphs and drawings, sounds, etc. Three main properties are closely connected with the concept of object: encapsulation, inheritance and polyformism. All of them are inherent in the objects of the Mathematica system and do not require special tools for their implementation. Encapsulation means combining in one object both data and methods of their processing. Inheritance means that each object derived from other objects inherits their properties. Polyformism - a property that allows you to transfer a number of

The programming language of Mathematica is specially designed to implement any of these approaches to programming, as well as a number of others, such as recurrent programming, using which the next step of the calculation is based on the data obtained in the previous steps. Possibly recursive programming is when the function in the general case repeatedly addresses itself. Mathematica language tools allow you to implement elements of visual-oriented programming. Mathematica allows you to create palettes and panels with various buttons that allow you to control the program or enter new program objects.

The structure of the software environment of the Mathematica system

The structure of the software environment of the Mathematica system can be represented as follows, shown in Fig. 1.

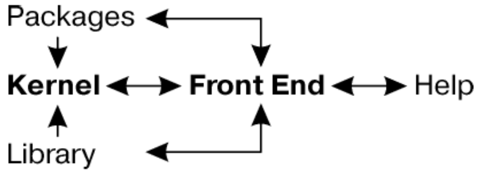
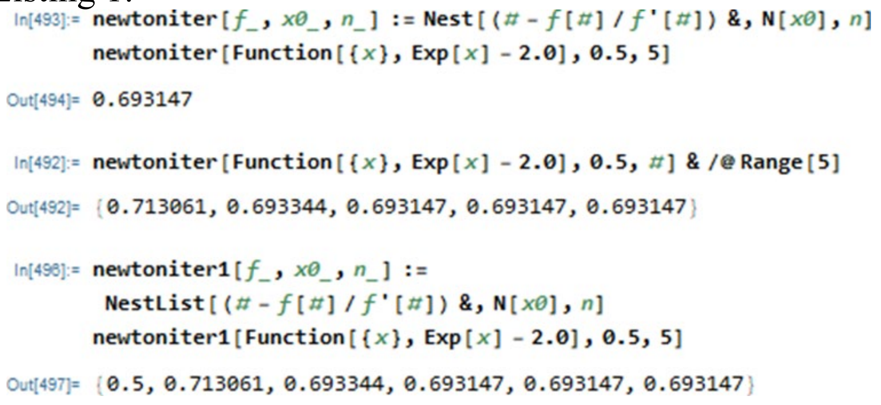


Fig. 1. The structure of the software environment of the Mathematica system

The FrontEnd interface processor is used to orient the system to a specific machine platform. It determines what the user interface of the system looks like. The central place in the systems of the Mathematica class is occupied by the machine-independent core of mathematical operations - Kernel. It contains a set of operators and functions, rules for calculations and transformations of mathematical expressions. The kernel is made compact enough for any function to be called from it quickly enough. The Library library and the Packages extension set are used to extend the feature set. Extension packages are created in Mathematica's own programming language and are the main means of expanding the system's capabilities and adapting them to specific classes of user tasks. In addition, the systems have a built-in help system - Help.

Implementation of recursive and recurrent algorithms

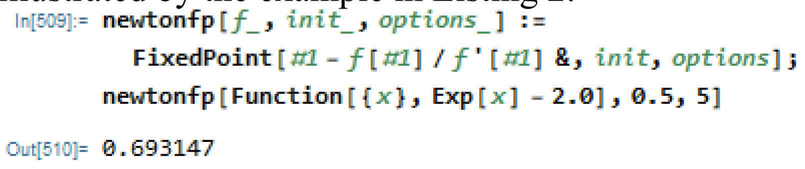
An important place in solving many mathematical problems is the implementation of recursive and recurrent algorithms. Consider a typical example of the implementation of an iterative recurrent algorithm for calculating the square root of the expression f (x) at the initial value of x0 = a, according to the following formulas of Newton's method:
x0 = a and xn = xn-1 - f (xn-1) / f '(xn-1).
This function can be written as follows:
newtoniter [f_, x0_, n_]: = Nest [(# - f [#] / f '[#]) &, N [x0], n]
Then the calculation of the root of the expression ex - 2 with the initial approximation x0 = 0.5 and the number of iterations n can be organized using the functions Nest [] and NestList [], as shown in Listing 1:



```
In[493]:= newtoniter[f_, x0_, n_] := Nest[(# - f[#] / f'[#]) &, N[x0], n]
          newtoniter[Function[{x}, Exp[x] - 2.0], 0.5, 5]

Out[494]= 0.693147

In[492]:= newtoniter[Function[{x}, Exp[x] - 2.0], 0.5, #] & /@ Range[5]

Out[492]= {0.713061, 0.693344, 0.693147, 0.693147, 0.693147}

In[496]:= newtoniter1[f_, x0_, n_] :=
           NestList[(# - f[#] / f'[#]) &, N[x0], n]
          newtoniter1[Function[{x}, Exp[x] - 2.0], 0.5, 5]

Out[497]= {0.5, 0.713061, 0.693344, 0.693147, 0.693147, 0.693147}
```

Listing 1

In the first case, the last result is returned, and in others - all intermediate. The FixedPoint function allows you to perform iterations until the result stops changing (with machine accuracy). This is illustrated by the example in Listing 2.



```
In[509]:= newtonfp[f_, init_, options_] :=
            FixedPoint[#1 - f[#1] / f'[#1] &, init, options];
          newtonfp[Function[{x}, Exp[x] - 2.0], 0.5, 5]

Out[510]= 0.693147
```

Listing 2

Recursive algorithms use calculations in which the body of the function refers to itself. Mathematica allows this possibility. A typical example of this is the calculation of the factorial by the formula N! = N * (N-1)! .

Fundamentals of procedural programming

The basis of procedural programming is the concept of procedure as a complete software module and typical controls: cycles, conditional and unconditional, etc. Although, the programming of Mathematica systems in this case remains functional, because the elements of procedural programming are given in the form of functions. However, it is possible to use traditional programming tools - conditional expressions, loops, procedures, etc.

Procedures are completely independent software modules, have their own identifier and perform some sequence of operations. They can be described in one line using the symbol ";" as a delimiter. (semicolon), for example:

```
r = (1 + x) ^ 2; r = Expand [r]; r-1
```

This procedure returns the following symbolic expression:

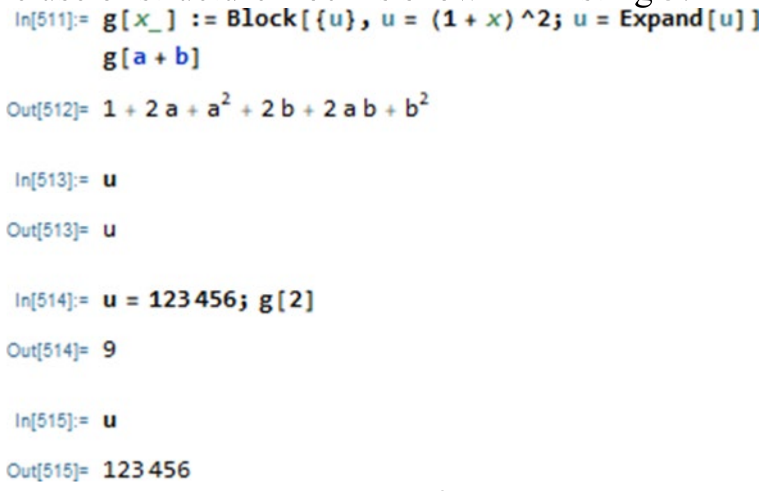
```
Expand [(1 + x) ^ 2] - 1
```

Structure is used to create full-fledged procedures and functionsBlock in two versions:

Block [{x, y, ...}, procedure] - procedure with declaration of the list of local variables x, y,…

Block [{x = x0, y = y0, ...}, procedure] is a procedure with declaration of the list of local variables x, y,… with initial values.

An example of the use of structureBlock is shown in Listing 3:



```
In[511]:= g[x_] := Block[{u}, u = (1 + x)^2; u = Expand[u]]
          g[a + b]

Out[512]= 1 + 2 a + a^2 + 2 b + 2 a b + b^2

In[513]:= u

Out[513]= u

In[514]:= u = 123456; g[2]

Out[514]= 9

In[515]:= u

Out[515]= 123456
```

Listing 3

Note that the variable u used in the body of the base structure is local, and assigning it the symbolic expression (1+ x) ^ 2 in the body of the block is ignored outside the block. If the variable u has not been defined before use in the function, it remains undefined. And if it had some value before (for example, 123456 in our case), then after leaving the procedure, it will have this value.

Organization of cycles

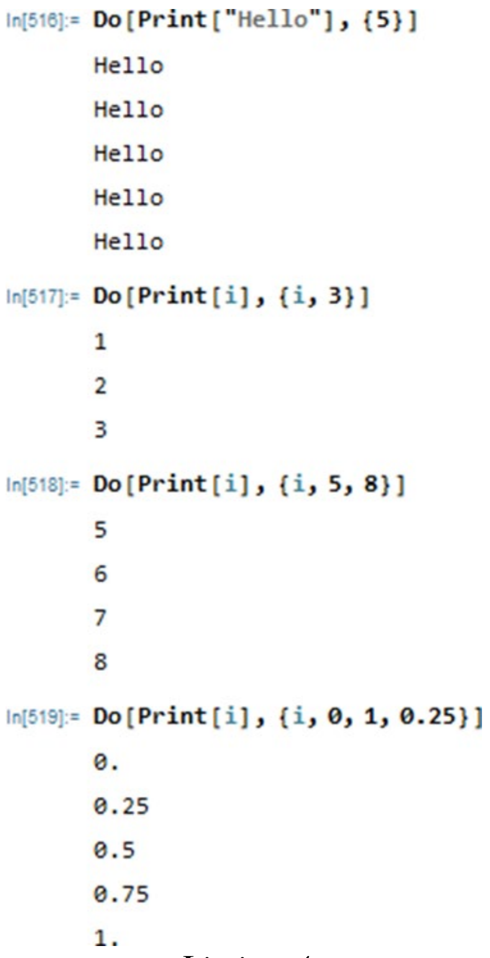
Cycles of type Do

Cycles of this type have several modifications:

- Do [expr, {imax}] - calculates the expression expr imax times.
- Do [expr, {i, imax}] - calculates the expression expr with the variable and, which alternately takes the value from 1 to imax with step 1.
- Do [expr, {i, imin, imax}] - calculates the expression expr with the variable and, which alternately takes values from imin to imax with step 1.

- Do [expr, {i, imin, imax, di}] - calculates the expression expr with the variable and, which alternately takes values from imin to imax with step di.
- Do [expr, {i, imin, imax}, {j, jmin, jmax}, ...] - calculates the expression expr using a series of nested loops with variables j, i, etc.

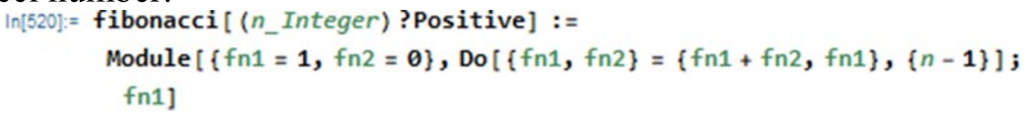
Examples of the organization of the cycle Do and its implementation are shown in Listing 4:



```
In[516]:= Do[Print["Hello"], {5}]
          Hello
          Hello
          Hello
          Hello
          Hello

In[517]:= Do[Print[i], {i, 3}]
          1
          2
          3

In[518]:= Do[Print[i], {i, 5, 8}]
          5
          6
          7
          8

In[519]:= Do[Print[i], {i, 0, 1, 0.25}]
          0.
          0.25
          0.5
          0.75
          1.
```

Listing 4

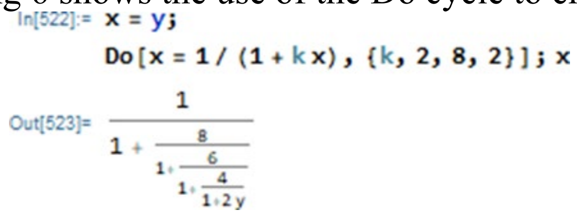
The variable i in the body of the loop - the iterator - is local. The entire program with the loop is stored in one cell. Listing 5 shows a procedure with a Do loop that calculates the nth Fibonacci number:



```
In[520]:= fibonacci[(n_Integer)?Positive] :=
           Module[{fn1 = 1, fn2 = 0}, Do[{fn1, fn2} = {fn1 + fn2, fn1}, {n - 1}];
            fn1]
```

Listing 5

Note the use of the Module function in this example. It creates a software module with local variables (in our case fn1 and fn2), which organizes the recurrent calculation of Fibonacci numbers. Listing 6 shows the use of the Do cycle to create a chain fraction:



```
In[522]:= x = y;
          Do[x = 1/ (1 + k x), {k, 2, 8, 2}]; x
```

Out[523]= $$\frac{1}{1+\frac{8}{1+\frac{6}{1+\frac{4}{1+2y}}}}$$

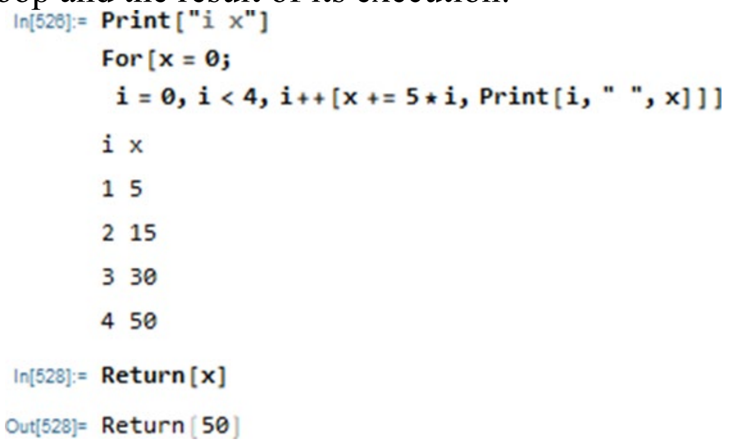
Listing 6

For loops

Another type of For loop is implemented by the function:

For [start, test, incr, body]

In it the variable of a cycle at first is appropriated value start, then cyclically changes from this value to value body with a step incr; and so on as long as the test condition is true. When the condition becomes false, the cycle ends. The example of Listing 7 shows a simple program with a For loop and the result of its execution:



```
In[526]:= Print["i x"]
          For[x = 0;
           i = 0, i < 4, i++[x += 5 * i, Print[i, " ", x]]]
          i x
          1 5
          2 15
          3 30
          4 50

In[528]:= Return[x]

Out[528]= Return[50]
```

Listing7

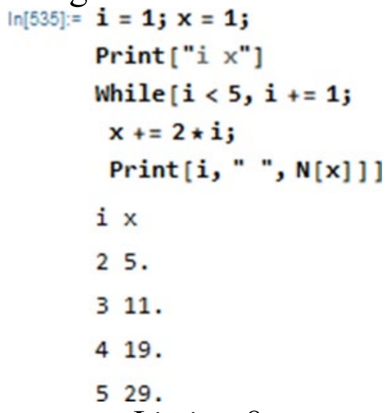
The program given above allows to observe change of values of a variable of a cycle i and a variable x which receives for each cycle of an increment equal 5 * i. The end of the document shows an example of using the Return function [x]. The For loop uses global variables, so you need to control their use.

While loops

Cycle recording form:

While [test, expr].

In this type of loop, the value of expr is calculated as long as the test condition is true.The following is an example of how to organize and use the While loop.



```
In[535]:= i = 1; x = 1;
          Print["i x"]
          While[i < 5, i += 1;
           x += 2 * i;
           Print[i, " ", N[x]]]
          i x
          2 5.
          3 11.
          4 19.
          5 29.
```

Listing 8

The device of local variables in this type of cycles is not used.

Directives for interrupting and continuing cycles

In these types of cycles and in other structures, you can use the following directives-functions:

- Abort [] - stops the calculation with the message $ Aborted.

- Break [] - exits the body of the loop or the level of nesting of the program containing the operator (loops such as Do, For and While or the body of the operator - Swith switch). The operator returns a Null value.

- Continue [] - sets the transition to the next step of the current cycle Do, For orWhile.

- Interrupt [] - interrupts calculations with the ability to resume them using a dialog box.

- Return [] - aborts execution with Null return.

- Return [expr] - aborts execution with the return of the value of the expression expr.

Conditional expressions and unconditional transitions

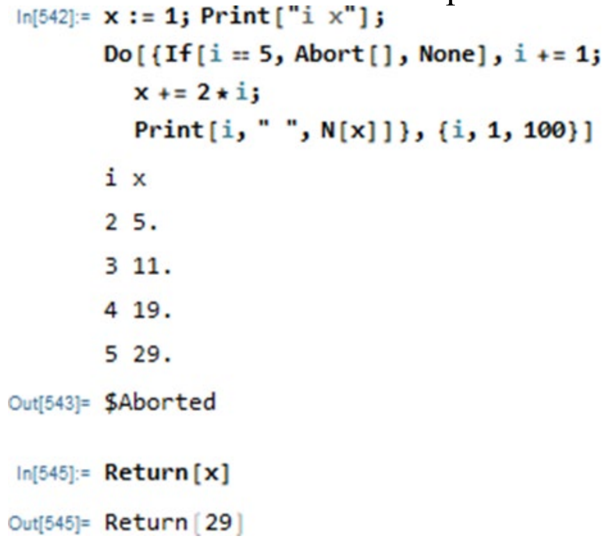
If function

As in most programming languages, conditional expressions are specified using the operator or the If function. The Mathematica system has the following modifications to the If function:

- If [condition, t, f] - returns the result t if the condition condition calculation is true, and f if the result is false.

- If [condition, t, f, u] - returns the result u, if the result of calculating the condition condition is neither true nor false.

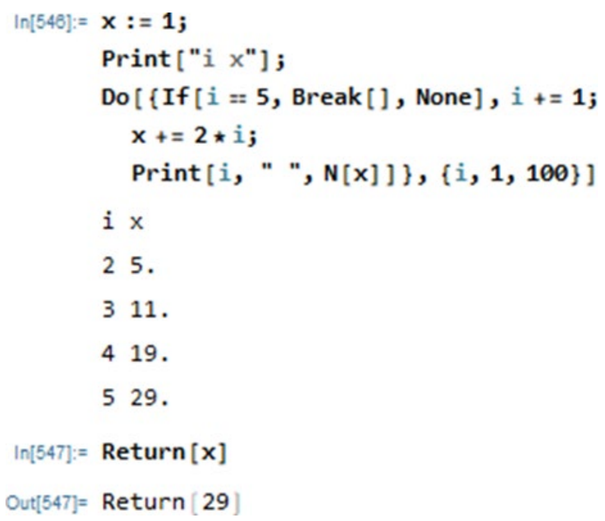
Listing 9 shows a description of a procedure with a Do loop, the output of which is organized using the If function and the Aborted interrupt directive []:



```
In[542]:= x := 1; Print["i x"];
          Do[{If[i == 5, Abort[], None], i += 1;
            x += 2 * i;
            Print[i, " ", N[x]]}, {i, 1, 100}]

          i x

          2 5.

          3 11.

          4 19.

          5 29.

Out[543]= $Aborted

In[545]:= Return[x]

Out[545]= Return[29]
```

Listing 9

A similar example using the Break function in the If function is shown in Listing 10.



```
In[546]:= x := 1;
          Print["i x"];
          Do[{If[i == 5, Break[], None], i += 1;
            x += 2 * i;
            Print[i, " ", N[x]]}, {i, 1, 100}]

          i x

          2 5.

          3 11.

          4 19.

          5 29.

In[547]:= Return[x]

Out[547]= Return[29]
```

Listing 10

In this case, no special exit messages are given.

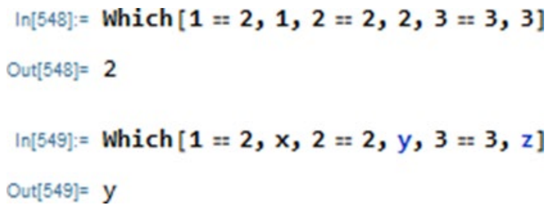
Functions - switches

To organize several branches in the system Mathematica use operators - switches Which and Switch:

- Which [test1, value1, test2, value2, ...] - calculates in order each condition testi and returns the value valuei of the first condition testi, which was true.

- Switch [expr, form1, value1, form2, value2, ...] - calculates the value of the condition expr, then compares it sequentially with each expression formi and returns the value valuei, the first match of the expression formi with the condition expr.

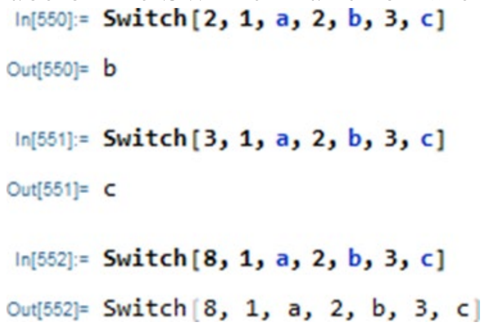
Examples of using the Which function are shown in Listing 11.



```
In[548]:= Which[1 == 2, 1, 2 == 2, 2, 3 == 3, 3]

Out[548]= 2

In[549]:= Which[1 == 2, x, 2 == 2, y, 3 == 3, z]

Out[549]= y
```

Listing 11

Examples showing the use of the Swhitch function are shown in Listing 12.



```
In[550]:= Switch[2, 1, a, 2, b, 3, c]

Out[550]= b

In[551]:= Switch[3, 1, a, 2, b, 3, c]

Out[551]= c

In[552]:= Switch[8, 1, a, 2, b, 3, c]

Out[552]= Switch[8, 1, a, 2, b, 3, c]
```

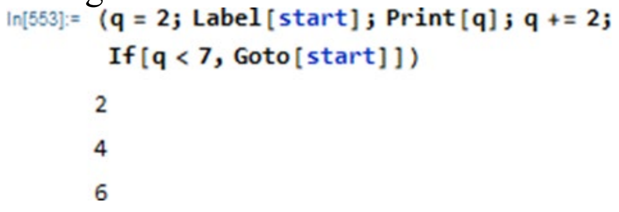
Listing 12

Note the last example: if the first parameter is not written correctly, the function is repeated.

Unconditional transitions

The unconditional transition operator Goto [tag] creates a transition to the place of the program marked with the label Label [tag]. Forms Goto [expr] and Label

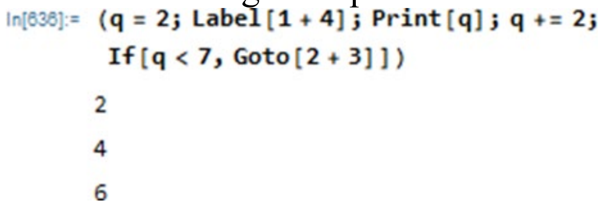
[expr] are also possible, where expr is a specific expression. Using the Goto statement is shown in Listing 13.



```
In[553]:= (q = 2; Label[start]; Print[q]; q += 2;
           If[q < 7, Goto[start]])

          2

          4

          6
```

Listing 13

Here, with the help of the Goto [start] operator, a loop is organized with the transition to the Label [start] label, which is executed while the value of q is less than 7. The q changes from the initial value 2 with step 2. An interesting feature of the value of the computed expression. For example, Goto [2 + 3] translates to Label [5] or even Label [1 + 4], as shown in the following example:



```
In[636]:= (q = 2; Label[1 + 4]; Print[q]; q += 2;
           If[q < 7, Goto[2 + 3]])

          2

          4

          6
```

Listing 14

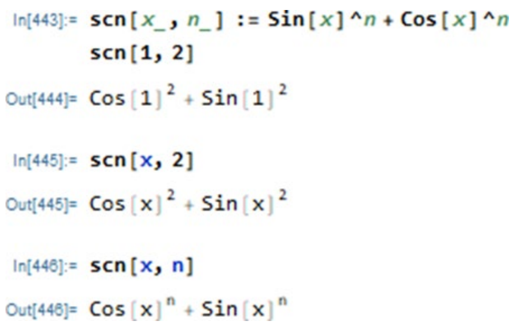
Lecture №12. Anonymous and clean features

Principlefunctional programming involves the use of only functions when writing. At the same time repeated embedding of functions in each other is possible. In some cases, especially during symbolic transformations, there is a mutual recursion of functions, accompanied by almost unlimited deepening of recursion and increasing complexity of the expressions processed by the system. The concept of a function is associated with the mandatory return of some value in response to a call to a function. Returning functions to some values allows you to use them along with operators to compose mathematical expressions. Functions are divided into internal and user-defined functions.

User functions

The process of creating a function in Mathematica is similar to other programming languages. For example, the function for reducing x to the power of n could be defined as follows:
powerxn [x, n]: = x ^ n
However, this feature is inoperable. The reason for this is that in Mathematica the symbols x and n are ordinary symbols and cannot accept formal parameters. For implementation it is necessary to use variables-samples having after their names underscores. Samples can be formal parameters of functions and perceive values of actual parameters. Thus, it would be correct to record the user function in the form:
powerxn [x_, n_]: = x ^ n

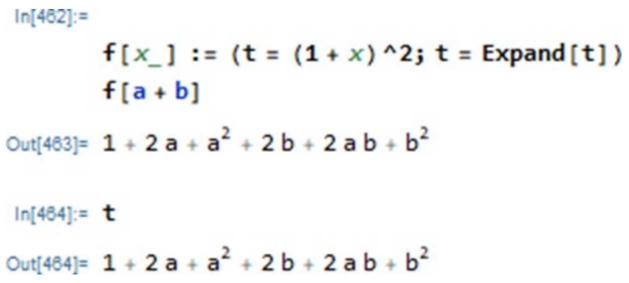
Consider another simple example, which specifies the function scn [x, n], which calculates the sum of sine in degree n and cosine in degree n, examples of which are shown in Listing 1:



```
In[443]:= scn[x_, n_] := Sin[x]^n + Cos[x]^n
          scn[1, 2]
Out[444]= Cos[1]^2 + Sin[1]^2

In[445]:= scn[x, 2]
Out[445]= Cos[x]^2 + Sin[x]^2

In[446]:= scn[x, n]
Out[446]= Cos[x]^n + Sin[x]^n
```

Listing 1

A function can consist of several expressions that are joined by parentheses:
f [x_]: = (t = (1 + x) ^ 2; t = Expand [t])
Variables of the list of parameters, after the name of which is the sign "_", are local in the body of the function or procedure with parameters. In their place the actual value of the corresponding parameter is substituted. An example illustrating the use of local variables is shown in Listing 2:

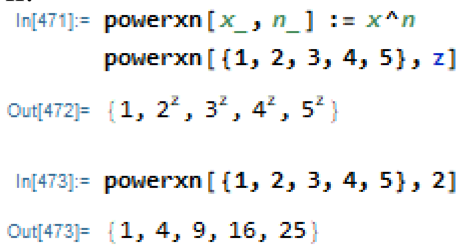


```
In[462]:=
          f[x_] := (t = (1 + x)^2; t = Expand[t])
          f[a + b]
Out[463]= 1 + 2 a + a^2 + 2 b + 2 a b + b^2

In[464]:= t
Out[464]= 1 + 2 a + a^2 + 2 b + 2 a b + b^2
```

Listing 2

Note that the variable t in the function f is global, which explains the result of the last operation. The use of global variables in the body of the function is quite possible, but creates a so-called side effect, in this case changes the value of the global variable t. To eliminate side effects, it is necessary to use samples and other special ways to create functions.

Function parameters can be lists, provided they can be combined. For example, in the powerxn function, Listing 1, suppose you use a list as an x parameter and a variable or number shown in Listing 3 as n:



```
In[471]:= powerxn[x_, n_] := x^n
          powerxn[{1, 2, 3, 4, 5}, z]

Out[472]= {1, 2^z, 3^z, 4^z, 5^z}

In[473]:= powerxn[{1, 2, 3, 4, 5}, 2]

Out[473]= {1, 4, 9, 16, 25}
```

Listing 3

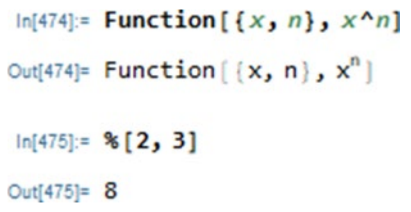
Once created, user functions can be used according to the same rules as built-in functions.

Clean functions

Sometimes it may be necessary to use a function only at the time of its creation. This function is represented only by an expression without a name, which led to its name. To create such an object is a built-in function Function, which is used in the form of:

- Function [body] - creates a pure function with the body body;
- Function [{x}, body] - creates a pure function of parameter x with body body;
- Function [{x1, x2, ...}, body] - creates a pure function of a number of parameters x1, x2, ... with body body.

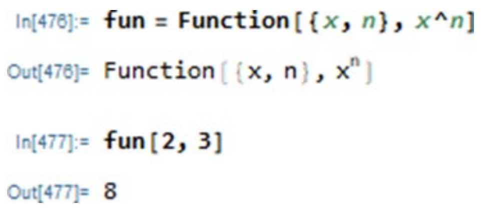
To calculate the function created in this way, a list of parameters is written in square brackets after it. As an example, Listing 4 shows the code of the net subtraction function to the degree:



```
In[474]:= Function[{x, n}, x^n]

Out[474]= Function[{x, n}, x^n]

In[475]:= %[2, 3]

Out[475]= 8
```

Listing 4

A pure function can be easily converted to a normal user function, as shown in Example Listing 5:

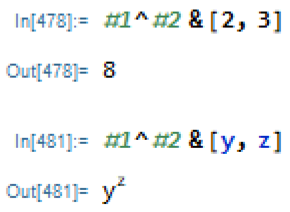


```
In[476]:= fun = Function[{x, n}, x^n]

Out[476]= Function[{x, n}, x^n]

In[477]:= fun[2, 3]

Out[477]= 8
```

Listing 5

Anonymous functions

The so-called anonymous functions have an extremely compact form of setting functions. They have neither a name nor a common definition, they are written in expressions of a special kind. In this expression instead of variables use designations # (for one variable), # 1, # 2, ... for a number of variables. End the body of the function with the symbol &. If it is necessary to calculate the value of the function, then after writing it in square brackets indicate a list of actual parameters. An example of using an anonymous function is shown in Listing 6:

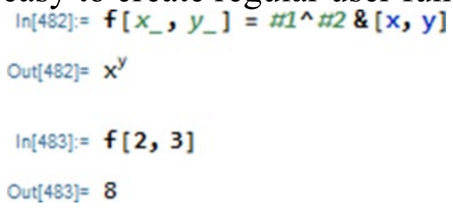


```
In[478]:= #1^#2&[2, 3]

Out[478]= 8

In[481]:= #1^#2&[y, z]
```

Out[481]= y^z

Listing 6

With anonymous features, it's easy to create regular user functions – Listing 7:



```
In[482]:= f[x_, y_] = #1^#2&[x, y]
```

Out[482]= x^y

```
In[483]:= f[2, 3]

Out[483]= 8
```

Listing 7

Superposition of functions

Functional programming involves the use of superposition of functions. To implement it, use the functions:

- Nest [expr, x, n] - applies an expression (function) to a given argument xn times.

- NestList [f, x, n] - returns the list of uses of the function f to the specified argument xn +1 times.

- Fold [f, x, list] - returns the next element in FoldList [f, x, list].

- FoldList [f, x, {a, b, ...}] - returns {x, f [x, a], f [f [x, a], b], j}.

- ComposeList [{f1, f2, ...}, x] - generates a list in the form {x, a [x], a [a [x]], ...}.

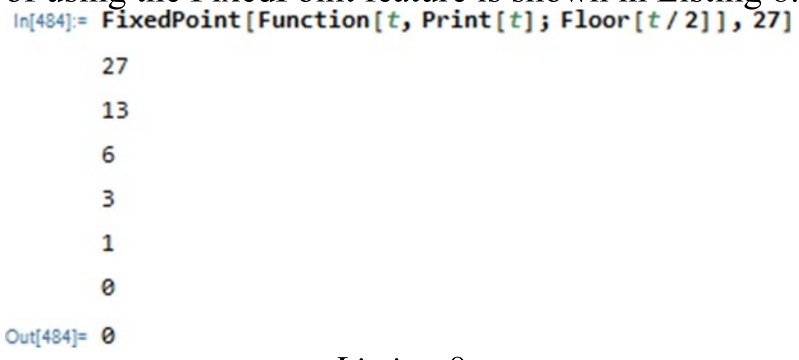
FixedPoint and Cath functions

In functional programming, instead of the cycles described below, it is possible use the following function:

- FixedPoint [f, expr] - calculates expr and applies the expression f to it until the result is repeated.

- FixedPoint [f, expr, SameTest_> comp] - calculates expr and applies the expression f to it until the next two results are true.

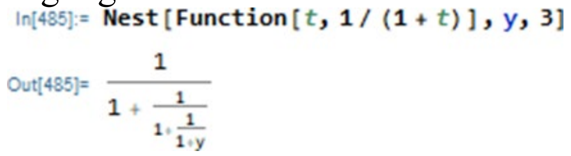
An example of using the FixedPoint feature is shown in Listing 8.



```
In[484]:= FixedPoint[Function[t, Print[t]; Floor[t / 2]], 27]

          27

          13

          6

          3

          1

          0

Out[484]= 0
```

Listing 8

The last result 0 is displayed in a separate (numbered) output cell and means the end of the process of iterations of division t by 2. A chain fraction using the function Nest can be created using the following arguments:



```
In[485]:= Nest[Function[t, 1/(1+t)], y, 3]
```

Out[485]=
$$\frac{1}{1+\frac{1}{1+\frac{1}{1+y}}}$$

Another function of this kind - Catch []:

- Catch [expr] - calculates expr, before the first execution of the function Throw [value], then returns value.

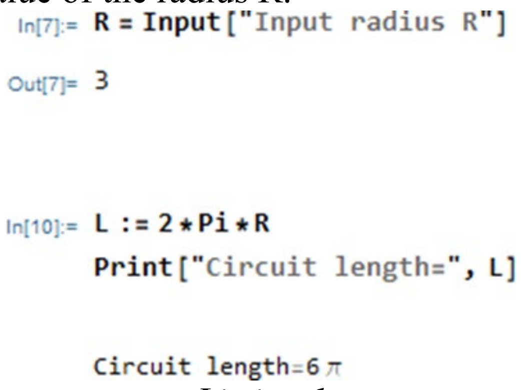
- Catch [expr, form] - calculates expr before the first execution of the Throw function [value, tag], then returns value.

- Catch [expr, form, f] - returns f [value, tag] instead of value.

Lecture № 13. Graphical objects of the user interface and means of input-output

I / O in Mathematica is organized using the FrontEnd interface processor. Additionally, the system implements a number of additional I / O functions:
- Input [] - stops the system and returns the value of the expression that will be entered in the dialog box (used to organize dialog input);
- Input [comment "] - works similarly to the previous function, additionally displays in the dialog box" comment ";
- InputString [] - reads data and saves it in the form of a character string;
- InputString ["comment"] - works similarly to the previous function, additionally displays in the dialog box "comment"
- StylePrint [expr] - creates a new cell in the current document with the default style and enters the expression expr;
- StylePrint [expr, "style"] - creates a new cell with style style in the current document and enters the expression expr in it;
- Print [expr] - displays the value of the expression expr, together with the function Input [] can be used to organize a dialogue;
- Print [prompt », expr] - displays a text comment in quotation marks on the display screen, followed by the value of the expression expr.

These functions are enough to organize the simplest dialogue with the program. Listing 1 shows the simplest example of dialogue. In this case, the length of the circle is calculated by the value of the radius R.



```
In[7]:=  R = Input["Input radius R"]

Out[7]=  3


In[10]:= L := 2*Pi*R
         Print["Circuit length=", L]

         Circuit length=6 π
```

Listing 1

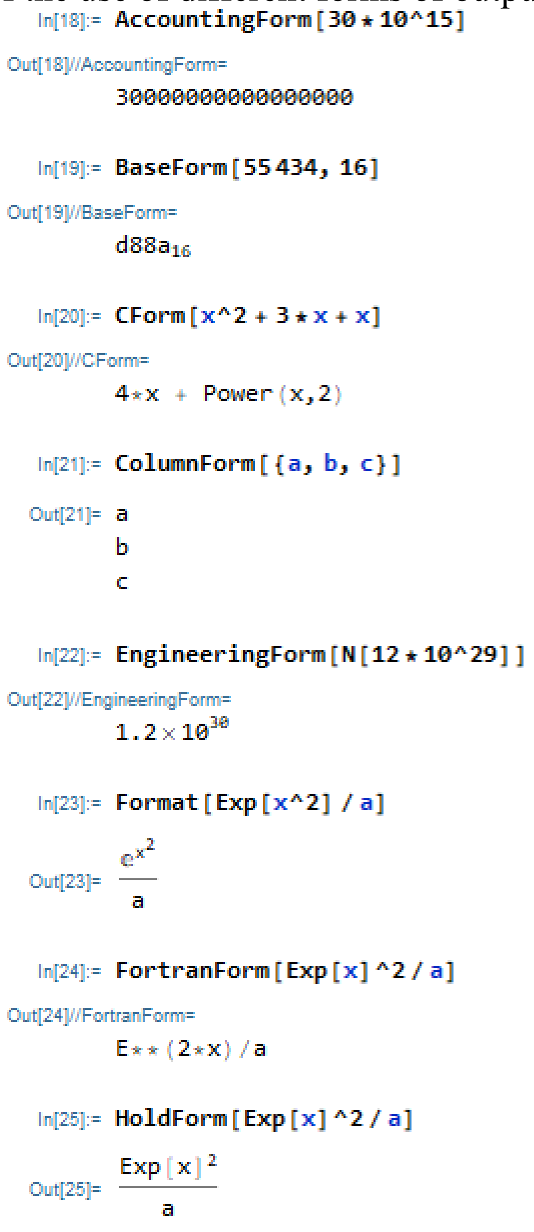
When the Input [] function is executed, a dialog box appears in the center of the screen. The window displays the query, which is specified in quotation marks as a parameter of the Input [] function. After entering the desired value (in the general example, the radius of the circle), the Input [] function returns the entered value and it is assigned to the variable R. After that, the Print [] function displays the calculated value of the circle length with a short comment.

Data output format

Mathematica implements a number of functions to customize the presentation format. The following functions are most often used:
- - AccountingForm [expr] - displays all the numbers contained in the expression expr, in the accounting form of presentation;
- CForm [expr] - outputs the expression expr in C format;

- EngineeringForm [expr] - displays the real numbers of the expression expr in engineering form;
- FortranForm [expr] - displays the expression expr in the form adopted for the language Fortran;
- FullForm [expr] - displays the full form of the expression expr without the use of special syntax;
- InputForm [expr] - displays the expression expr in input form;
- NumberForm [expr, n] - displays the expression expr in the form of a real number with an accuracy of n digits;
- - OutputForm [expr] - outputs expr in the standard output form of the Mathematica system;
- ScientificForm [expr] - displays the expression expr in scientific format;
- TeXForm [expr] - displays the expression expr in the form of the TeX language, which is focused on the layout of texts with mathematical formulas;
- TextForm [expr] - displays the expression expr in plain text format;
- TreeForm [expr] - displays the expression expr showing different levels of expression.

Listing 2 shows examples of the use of different forms of output.



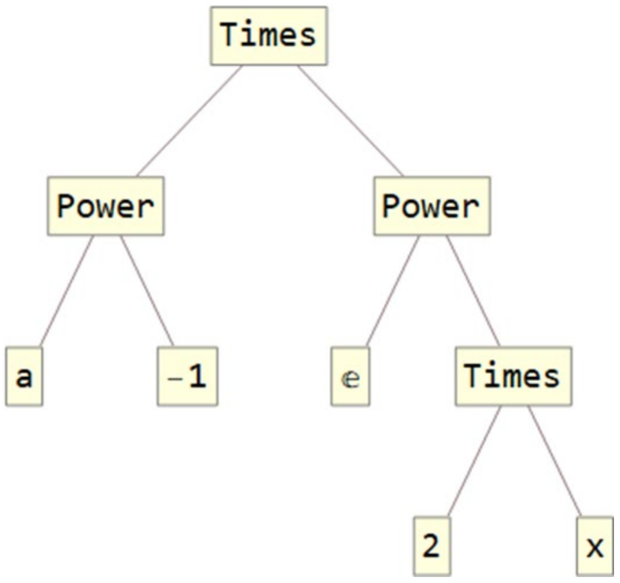
```
   In[18]:= AccountingForm[30 * 10^15]

Out[18]//AccountingForm=
          30000000000000000

   In[19]:= BaseForm[55434, 16]

Out[19]//BaseForm=
          d88a16

   In[20]:= CForm[x^2 + 3 * x + x]

Out[20]//CForm=
          4*x + Power(x,2)

   In[21]:= ColumnForm[{a, b, c}]

  Out[21]= a
           b
           c

   In[22]:= EngineeringForm[N[12 * 10^29]]

Out[22]//EngineeringForm=
          1.2 × 10^30

   In[23]:= Format[Exp[x^2] / a]

            e^(x^2)
  Out[23]= -------
              a

   In[24]:= FortranForm[Exp[x]^2 / a]

Out[24]//FortranForm=
          E**(2*x)/a

   In[25]:= HoldForm[Exp[x]^2 / a]

            Exp[x]^2
  Out[25]= ---------
               a
```

```
In[26]:= NumberForm[N[Exp[2]], 15]

Out[26]//NumberForm=
         7.38905609893065

In[27]:= OutputForm[Exp[x]^2 / a]

Out[27]//OutputForm=
          2 x
         E
         ----
          a

In[28]:= TeXForm[Exp[x]^2 / a]

Out[28]//TeXForm=
         \frac{e^{2 x}}{a}

In[29]:= ScientificForm[12 * 10^5]

Out[29]//ScientificForm=
         1200000
```

Listing 2

Listing 3 shows a few more examples of using different forms of output.

```
In[31]:= FullForm[Exp[x]^2 / a]

Out[31]//FullForm=
         Times[Power[a, -1], Power[E, Times[2, x]]]

In[32]:= TreeForm[Exp[x]^2 / a]

Out[32]//TreeForm=
```



```
In[33]:= PaddedForm[(x^3 + 2*x^2 + 3*x - 1) / (x - 1), 3]

Out[33]//PaddedForm=
          -1 +    3 x +    2 x  2 + x  3
          ─────────────────────────────
                    -1 + x

In[34]:= PrecedenceForm[12*b/c, 5]

          12 b
Out[34]=  ────
           c

In[35]:= SequenceForm[Exp[x]^2/a]

          e2 x
Out[35]=  ───
           a

In[36]:= TableForm[{{"x", "y"}, {1, 2}, {3, 4}, {5, 6}}]

Out[36]//TableForm=
          x     y
          1     2
          3     4
          5     6

In[37]:= Prefix[f[x^2]]

Out[37]= f@x2

In[38]:= Unevaluated[Exp[x]^2/a]

                     ⎡Exp[x]2⎤
Out[38]= Unevaluated ⎢───────⎥
                     ⎣   a   ⎦
```

Listing 3

GUI elements

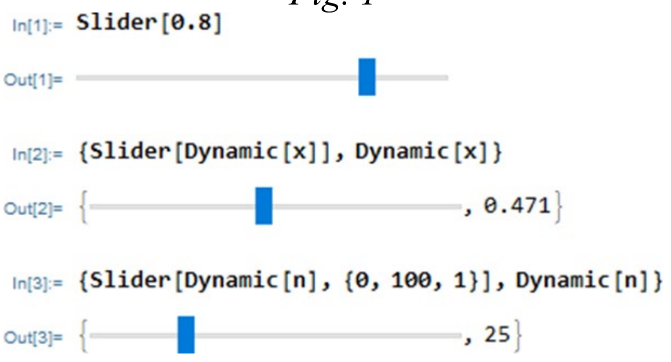
Mathematica tools allow you to create objects GUI (Graphic User Interface) laptops, which makes the latter much clearer and easier to use.

Single-coordinate sliders

Sliders allow you to smoothly or discretely change the value of a particular variable. Fig. 1 shows the repetition of three sliders by the Slider function with different variants of parameter values, Listing 4.

Slider[x] Slider[x,{x_{min},x_{max}}] Slider[x,{x_{min},x_{max},dx}]
Slider[x,{{e_1,e_2,...}}] Slider[x,{{{e_1,w_1},{e_2,w_2},...}}]

Fig. 1



```
In[1]:= Slider[0.8]

Out[1]=

In[2]:= {Slider[Dynamic[x]], Dynamic[x]}

Out[2]= {                    , 0.471}

In[3]:= {Slider[Dynamic[n], {0, 100, 1}], Dynamic[n]}

Out[3]= {                    , 25}
```

Listing 4

In the first case (upper slider), the parameter (number 0.8) determines the position of its engine. When you move the engine, the value will be returned in the range from 0 to 1.

The second slider (in the center) allows you to change the value of the variable x. To make the value of a variable dynamically available throughout the notebook, the variable is dynamic, to create it use the function Dynamic [x]. Typically, the range of values of the variable is from 0 to 1. The third slider allows you to expand the range of values of the slider from 0 to 100 in steps of 1.

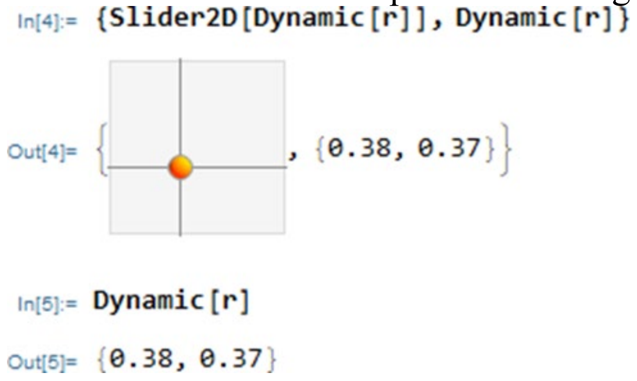
Two-coordinate sliders

To construct surfaces that describe the functions of two variables use two-coordinate sliders. They are created by the Slaider2D function, Listing 5.

Listing 5

Slider2D[{*x*,*y*}] **Slider2D[Dynamic[*pt*]]**
Slider2D[*pt*,{*min*,*max*}] **Slider2D[*pt*,{*min*,*max*,*d*}]**
Slider2D[*pt*,{{x_{min},y_{min}},{x_{max},y_{max}}}]
Slider2D[*pt*,{{x_{min},y_{min}},{x_{max},y_{max}},{*dx*,*dy*}}]

An example of creating a two-coordinate slider is presented in Fig. 2.



```
In[4]:= {Slider2D[Dynamic[r]], Dynamic[r]}

Out[4]= {     , {0.38, 0.37}}

In[5]:= Dynamic[r]

Out[5]= {0.38, 0.37}
```

Fig. 2. Two-coordinate slider

The engine of such a slider can move the mouse in any direction. The slider returns two numeric values depending on the position of the engine. These values can be used to calculate the functions of two variables and plot graphs of surfaces, three-dimensional figures, parametrically given graphs, etc.

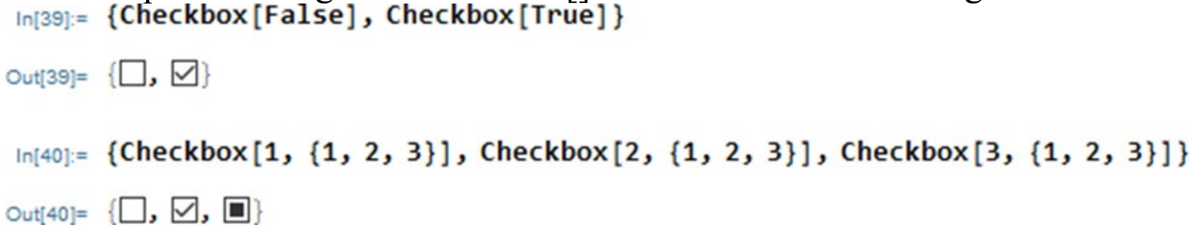
CheckBox graphical user interface element

It is often necessary to perform calculations provided that some options are set, for which it is convenient to use the CheckBox element, which is created by the CheckBox function [], Listing 6.

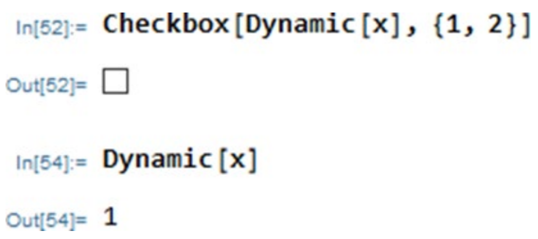
Checkbox[*x*] **Checkbox[Dynamic[*x*]]** **Checkbox[*x*,{*val*1,*val*2}]**
Checkbox[*x*,{*val*1,*val*2,*val*3,…}]

Listing 6

Examples of using the CheckBox [] function are shown in Fig. 3.



```
In[39]:= {Checkbox[False], Checkbox[True]}

Out[39]= {☐, ☑}

In[40]:= {Checkbox[1, {1, 2, 3}], Checkbox[2, {1, 2, 3}], Checkbox[3, {1, 2, 3}]}

Out[40]= {☐, ☑, ▣}
```



```
In[52]:= Checkbox[Dynamic[x], {1, 2}]

Out[52]= ☐

In[54]:= Dynamic[x]

Out[54]= 1
```

Fig. 3. CheckBox graphical user interface element

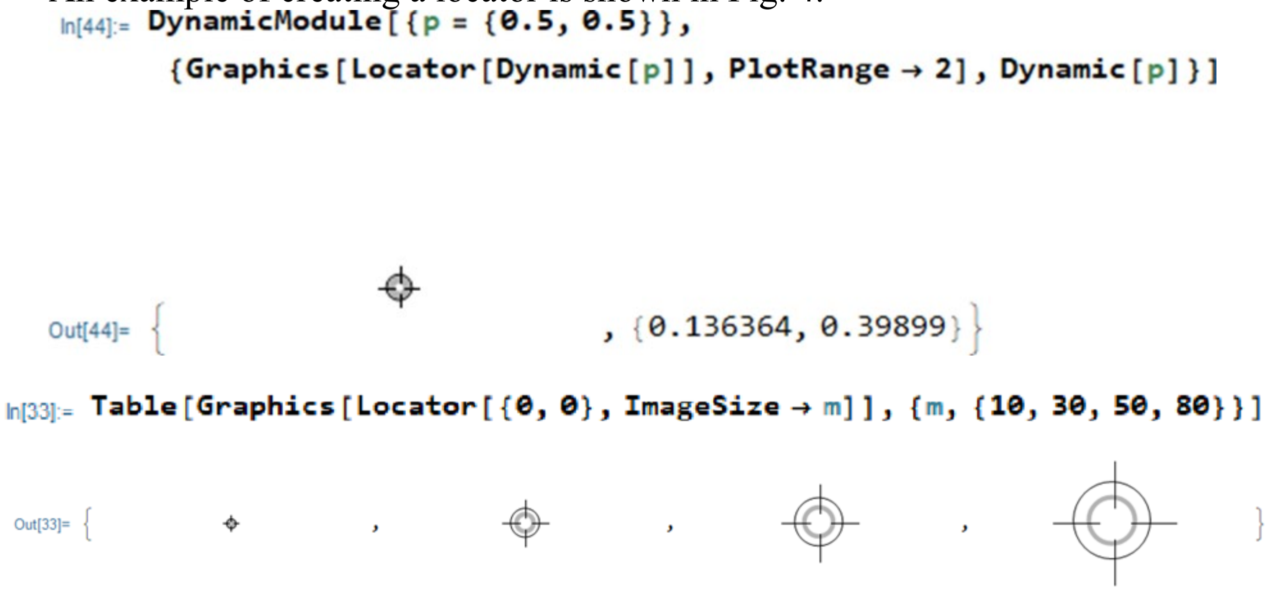
Locator

A locator is an object that has the shape of a dot in a graphics window and returns its coordinates. This object is represented by a painted circle in the middle of the intersection, which has a bright dot in the middle. The locator is moved with the mouse. To create it, use the function, Listing 7.

Locator[{*x*,*y*}]
Locator[Dynamic[*pos*]]
Locator[{*x*,*y*},*obj*]
Locator[{*x*,*y*},None]

Listing 7

An example of creating a locator is shown in Fig. 4.



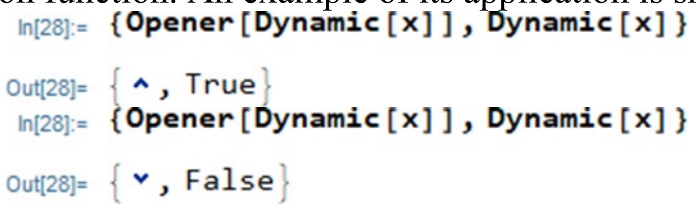
```
In[44]:= DynamicModule[{p = {0.5, 0.5}},
           {Graphics[Locator[Dynamic[p]], PlotRange → 2], Dynamic[p]}]

Out[44]= {                    , {0.136364, 0.39899}}

In[33]:= Table[Graphics[Locator[{0, 0}, ImageSize → m]], {m, {10, 30, 50, 80}}]

Out[33]= {     ,     ,     ,     }
```

Fig. 4

Locators are often used to plot points.

Mouse control functions

Sometimes it is necessary to determine the coordinates of the mouse cursor, which uses the MousePosition function. An example of its application is shown in Listing 8.



```
In[28]:= {Opener[Dynamic[x]], Dynamic[x]}

Out[28]= {^, True}
In[28]:= {Opener[Dynamic[x]], Dynamic[x]}

Out[28]= {˅, False}
```

Listing 8

The coordinates of the mouse are displayed in the format of integers. Adding the MousePosition function to the list of parameters of the Dynamic function allows you to display the coordinates of the mouse cursor as a list of current coordinates.

The Opener [x] or Opener [Dynamic [x]] function responds to each mouse click on a black triangle. For example, the command {Opener [Dynamic [x]], Dynamic [x]} displays a list in the output line:

{► ,False}

However, if you click on it, a list will appear

{▼,True}

Function **Toggler[x] Toggler[Dynamic[x]] Toggler[x,{val1,val2,...}] Toggler[x,{val1_>pict1,val2_>pict2,...}] Toggler[x,vlist,dpict]** provides the ability to respond to repeated mouse clicks.

Example,

```
In[46]:= Toggler[1, {a, b, c, d}]


Out[46]= 1
```

first displays 1. However, after pressing the mouse button 4 times, the characters a, b, c and d are displayed.

```
In[29]:= Toggler[3, {1 → "One", 2 → "Two", 3 → "Three"}]

Out[29]= One
In[30]:= Toggler[1, {1, 2, 3}, AutoAction → True]

Out[30]= 2
```

Button with the inscription

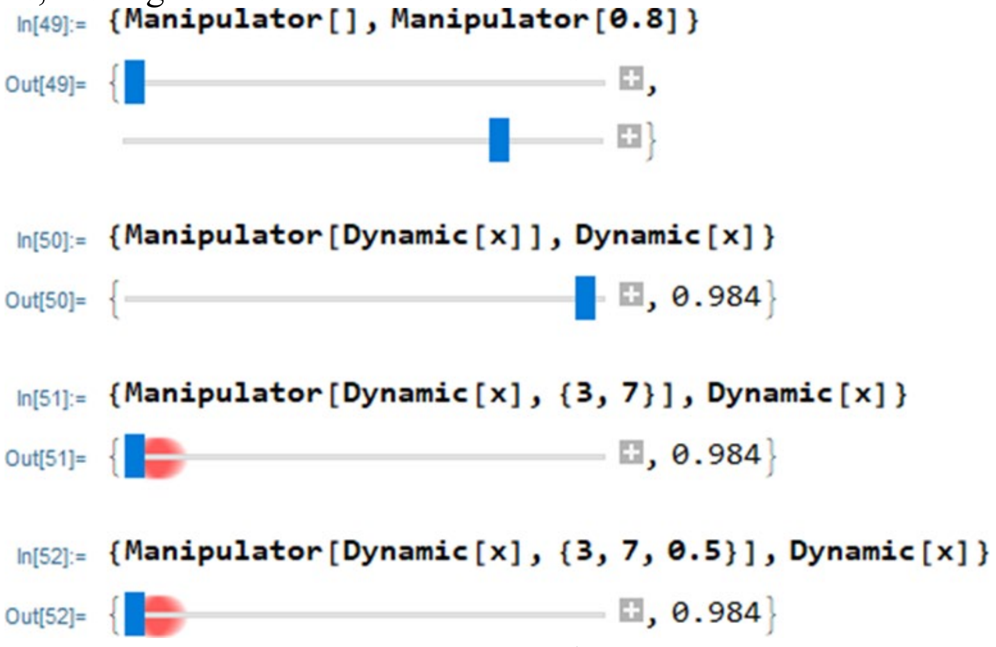
Often some actions must be performed when pressing the button with the inscription. To create such a button, use the Button [label, action] function. In the list of its parameters indicate the line with the name of the button and the expression that is performed when the mouse is activated by the mouse. An example of using the Button function to derive the factorial of number 10 is shown in Listing 9.

```
In[48]:= Button["Click Here", Print[10!]]

Out[48]= Click Here


3628800
In[31]:= Table[With[{i = i}, Button[i, Print[i!]]], {i, 10}]

Out[31]= {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

1

2

6

24

120

720
```

Listing 9

Manipulator

A manipulator is an object similar to a slider, but has greater functionality and visual capabilities. The manipulator is created using the function: Manipulator [x], Manipulator [Dynamic [x]], Manipulator [x, {xmin, xmax}], Manipulator [x, {xmin, xmax, dx}]. A distinctive feature of the manipulator is a characteristic button in the form of a gray rectangle with a "+" in it. When you activate this button with the mouse, a panel with slider controls appears under the manipulator slider (see Fig. 5, example above). By means of bodies (buttons) of management of the manipulator it is possible to start it and to provide automatic movement of the slider engine, it is possible to change the direction and speed of movement of the slider, to make its stop. As the slider approaches the start and end points, the engine has a characteristic shadow.



```
In[49]:= {Manipulator[], Manipulator[0.8]}

Out[49]= {

In[50]:= {Manipulator[Dynamic[x]], Dynamic[x]}

Out[50]= {, 0.984}

In[51]:= {Manipulator[Dynamic[x], {3, 7}], Dynamic[x]}

Out[51]= {, 0.984}

In[52]:= {Manipulator[Dynamic[x], {3, 7, 0.5}], Dynamic[x]}

Out[52]= {, 0.984}
```

Fig. 5. Manipulator

Rotator of the radius vector

In some cases, for example, when plotting functions in a polar coordinate system, you need an object that specifies the angle of rotation of the radius vector. Such an object - the angle setter - is specified by the angularSlider [] function. In fig. 6 shows an example of the application of this function.

```
In[13]:= DynamicModule[{θ = 0},
          Grid[
           {{DynamicModule[{x = RandomReal[{0, 50}]},
               {Experimental`AngularSlider[Dynamic@θ], Dynamic@θ}],
             Plot[Sin[x], {x, -10, 10},
               Epilog → {PointSize[Medium], Dynamic@Point[{θ, Sin[θ]}]}]},
            {ParametricPlot[{Cos[y], y}, {y, -10, 10}, AspectRatio → 2,
               Epilog → {PointSize[Medium], Dynamic@Point[{Cos[θ], θ}]}],
             Dynamic[θ]}}]]
```

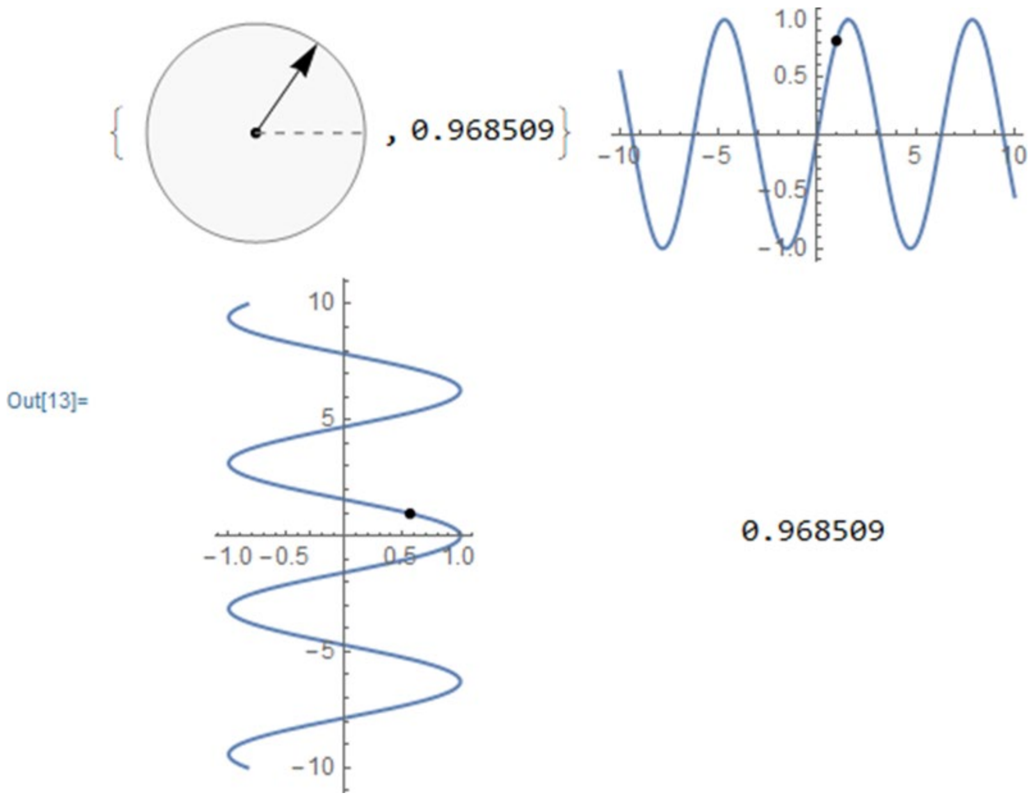
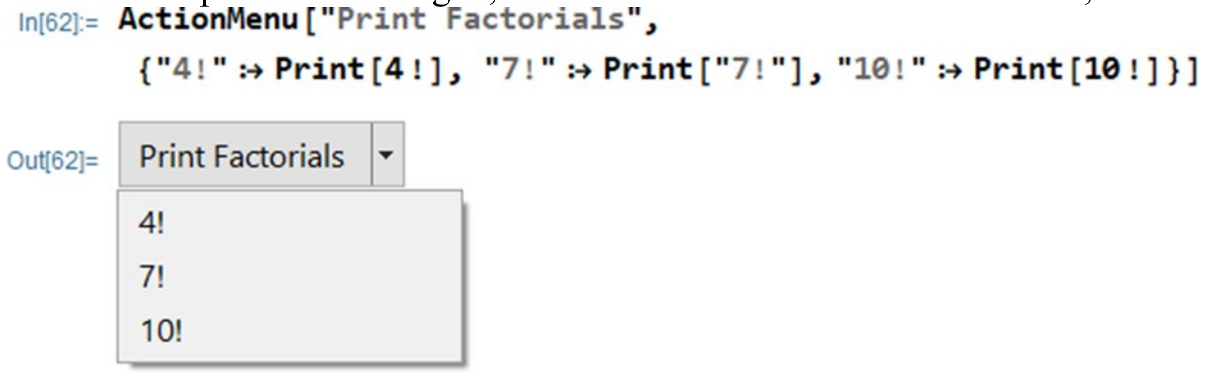


Fig. 6. Setting angle of rotation of the radius vector

The task is a graphic object in the form of a circle, inside which is placed a radius vector. It can rotate in one direction or another with the mouse, which leads to a change in the angle set by the unit. In the example shown in Fig. 6, projections of the end of the radius vector on the coordinate axis are constructed. These projections are known to have sinusoidal functions. The angle corresponding to the current position of the radius vector is indicated by dots on the graphs of sinusoidal functions.

Drop-down menu

The drop-down menu is another widely used object for building a graphical interface. It is created by the ActionMenu function [name, {lbl1:> act1, lbl2:> act2,…}]. The parameters of the function are a line with an inscription on the button and a list of names of menu items and actions performed when activating the corresponding menu items. In the example shown in Fig. 7, calculate the values of the factorials 4!, 7! i 10!.



```
In[62]:= ActionMenu["Print Factorials",
          {"4!" :→ Print[4!], "7!" :→ Print["7!"], "10!" :→ Print[10!]}]
```

Fig. 7. Example of creating a menu

Data entry panel

The expression input panel is used for interactive input of an arbitrary expression, for example, to build its graph. It is entered by the Panel function.
Panel[*expr*], Panel[*expr*,*title*], Panel[*expr*,*title*,*pos*],
Panel[*expr*,{*title*1,*title*2,…},{*pos*1,…}], Panel[].

An example of setting the panel to enter an expression and build its graph is shown in Fig. 8.

```
In[69]:= Panel[DynamicModule[{f = Sin[x]},
           Column[{InputField[Dynamic[f]],
             Dynamic[Plot[f, {x, -5, 5}]]}]]]

Out[69]=
```

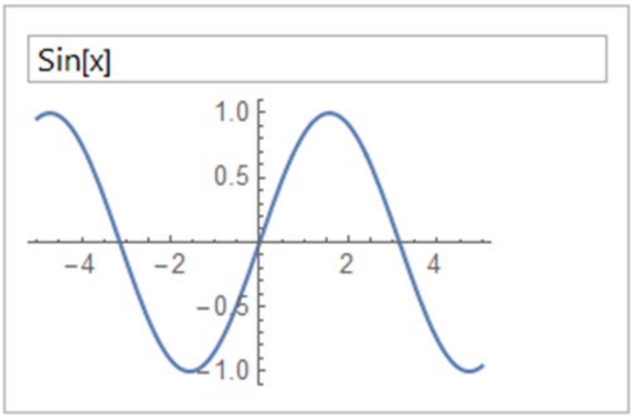
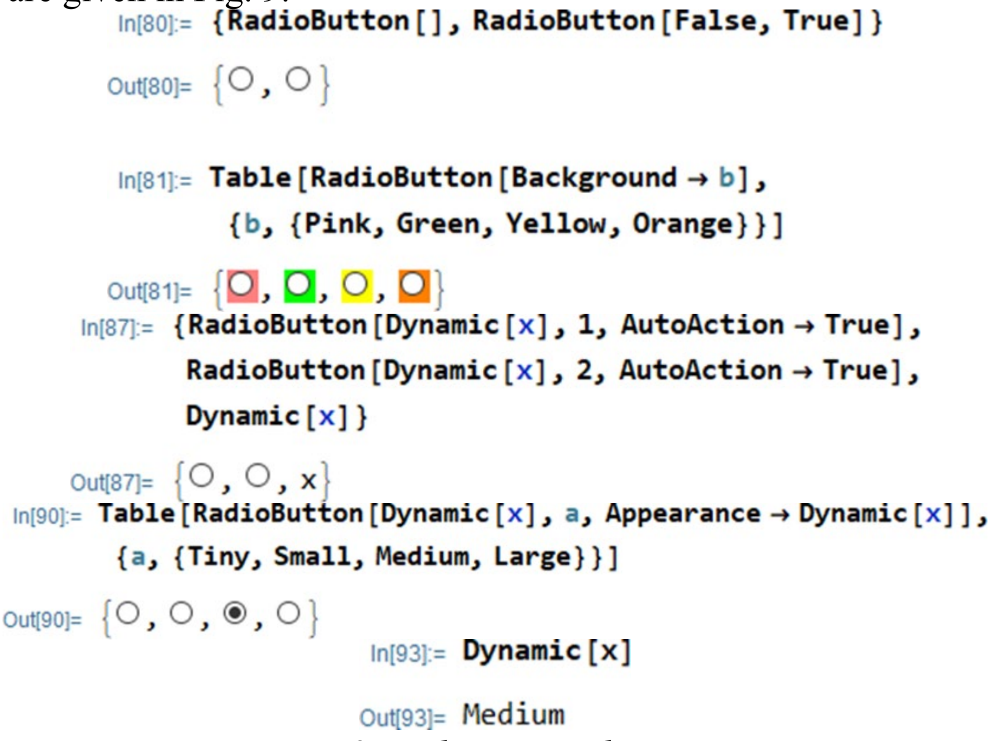


Fig. 8. Example of creating a data entry panel

Note that the default expression (in our case Sin [x]) appears in the input panel, which is a parameter of the DynamicModule function inside the list of parameters of the Panel function. This expression is used to plot the output.

Radio buttons and settings menu

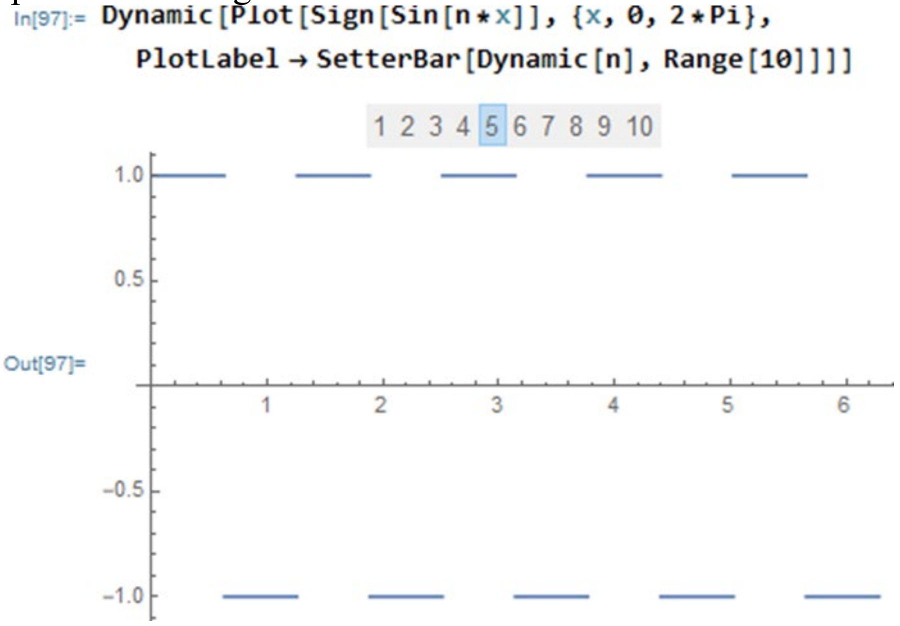
The RadioButton [x, val] function creates a so-called circle radio button. The result of the function can be used for software input of an action. Examples of application of the radio button are given in Fig. 9.



```
In[80]:= {RadioButton[], RadioButton[False, True]}

Out[80]= {○, ○}

In[81]:= Table[RadioButton[Background → b],
           {b, {Pink, Green, Yellow, Orange}}]

Out[81]= {○, ○, ○, ○}
In[87]:= {RadioButton[Dynamic[x], 1, AutoAction → True],
          RadioButton[Dynamic[x], 2, AutoAction → True],
          Dynamic[x]}

Out[87]= {○, ○, x}
In[90]:= Table[RadioButton[Dynamic[x], a, Appearance → Dynamic[x]],
          {a, {Tiny, Small, Medium, Large}}]

Out[90]= {○, ○, ●, ○}
                              In[93]:= Dynamic[x]

                              Out[93]= Medium
```

Fig. 9. RadioButton element

Another function in a number of recording formats specifies the construction of the settings menu has the following recording formats: SetterBar[*x*,{*val*1,*val*2,…}], SetterBar[Dynamic[*x*],{*val*1,*val*2,…}], SetterBar[*x*,{*val*1_>*lbl*1,*val*2_>*lbl*2,…}]. An example of it is presented in Fig. 10.

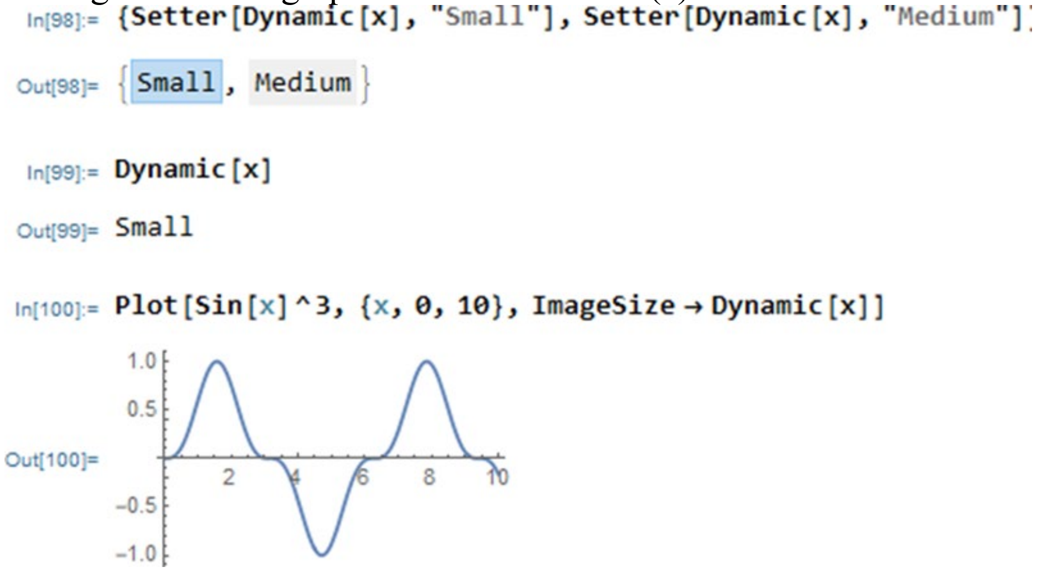


```
In[97]:= Dynamic[Plot[Sign[Sin[n*x]], {x, 0, 2*Pi},
           PlotLabel → SetterBar[Dynamic[n], Range[10]]]]
```

Fig. 10. Creating a settings menu with the SetterBar function []

In this example, it is possible to set the meander frequency by activating one or another menu item with a number..
Another option for building a settings menu is set by the function: Setter[*x*, *val*], Setter[Dynamic[*x*], *val*], Setter[*x*, *val*, *label*], Setter[*x*, {*val*1, *val*2,…}, *label*].

Fig. 11 shows an example in which the settings menu buttons provide the setting of the corresponding size of the graph of the function sin(*x*)^3.



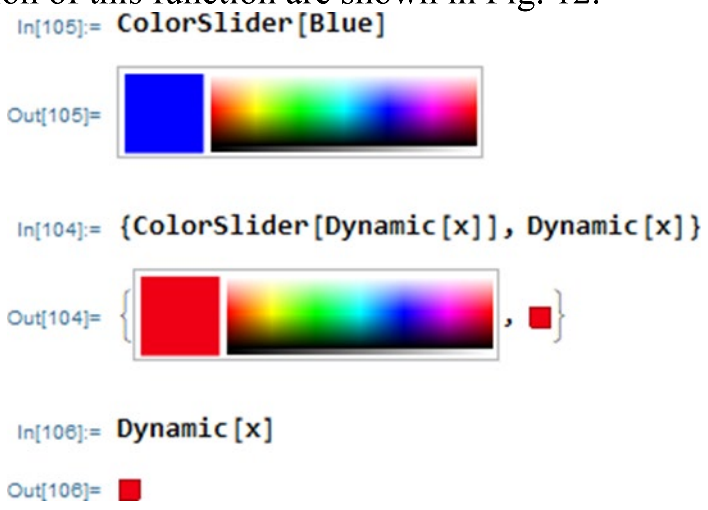
```
In[98]:= {Setter[Dynamic[x], "Small"], Setter[Dynamic[x], "Medium"]

Out[98]= {Small, Medium}

In[99]:= Dynamic[x]

Out[99]= Small

In[100]:= Plot[Sin[x]^3, {x, 0, 10}, ImageSize → Dynamic[x]]
```

Fig. 11. Example of using the Setter function []

Color slider

To change the color of GUI objects, it is convenient to use a slider, which is set by the following function: ColorSlider[*color*], ColorSlider[Dynamic[*color*]], ColorSlider[].

Examples of application of this function are shown in Fig. 12.



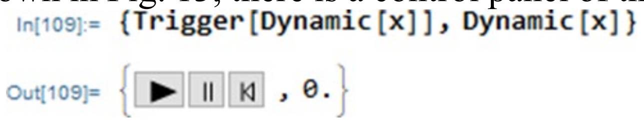
```
In[105]:= ColorSlider[Blue]

Out[105]=

In[104]:= {ColorSlider[Dynamic[x]], Dynamic[x]}

Out[104]= {  , }

In[106]:= Dynamic[x]

Out[106]=
```

Fig.12. Color slider

The color of the first square is the current color. To the left of it is the color selection panel. To select a color, simply place the mouse cursor on the desired color of the panel and press the left mouse button. The color of the control square will be similar to the selected one.

Starter mechanism

The Trigger function simulates the operation of the trigger. It has several forms of writing: Trigger[Dynamic[u]], Trigger[Dynamic[u],{umin,umax}], Trigger[Dynamic[u],{umin,umax,du}], Trigger[Dynamic[u],{umin,umax},ups]. The result of the function shown in Fig. 13, there is a control panel of the starting mechanism.



```
In[109]:= {Trigger[Dynamic[x]], Dynamic[x]}

Out[109]= {  , 0.}
```

Fig. 13. The starting mechanism

When you press the start button (large triangle), the value of the dynamic variable x begins to change from 0 to 1 within a few seconds. Then this change stops. The panel also has buttons to stop variables and return to 0 (Reset).

Functions for marking additional information on graphs

Function ClickPane[image,func], ClickPane[image,{{xmin,ymin},{xmax,ymax}},func] used to indicate a point in the image image using the expression func. Most often, the ClickPane function is used to select with the mouse cursor a certain place in the image, in which you need to click to place a graphic object, such as a point, arrowhead, circle, etc. In fig. 14 shows an example of the use of a function to denote the extremum of a sinusoidal function by means of an arrow emanating from the inscription "Extremum".

```
In[111]:= DynamicModule[{pt = {Pi / 2, 3}},
            ClickPane[Plot[3 Sin[x], {x, -18, 6},
              Epilog → {Dynamic@Arrow[{{2, 5}, pt}], Text["Екстремум", {4, 5}]},
              PlotRange → 6],
             (With[{x = (2 Round[-1 / 2 + 2 #[[1]] / Pi] + 1) Pi / 2},
                pt = {x, 3 Sin[x]}]) &]]
```

Out[111]=

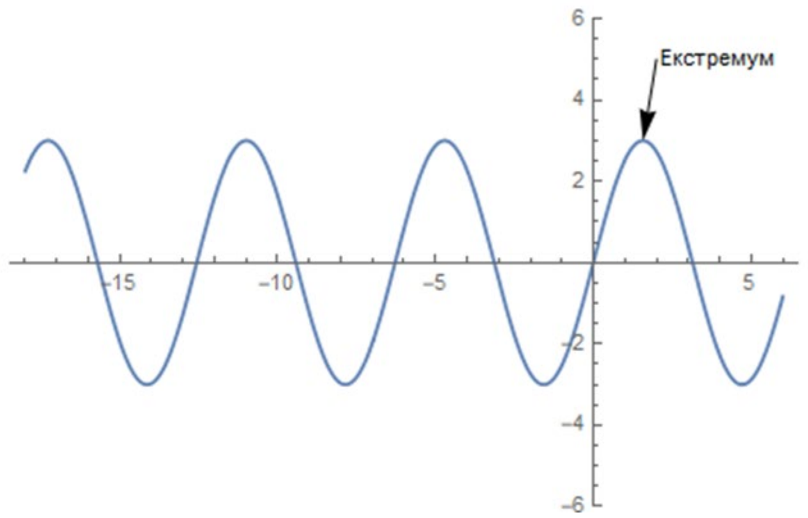


Fig. 14. Example of using the ClickPane function []

Function Tooltip[*expr*, *label*] displays the expression expr and replaces it with the label if the mouse cursor is set to the expression expr. In fig. 15 shows the simplest implementation of this function.

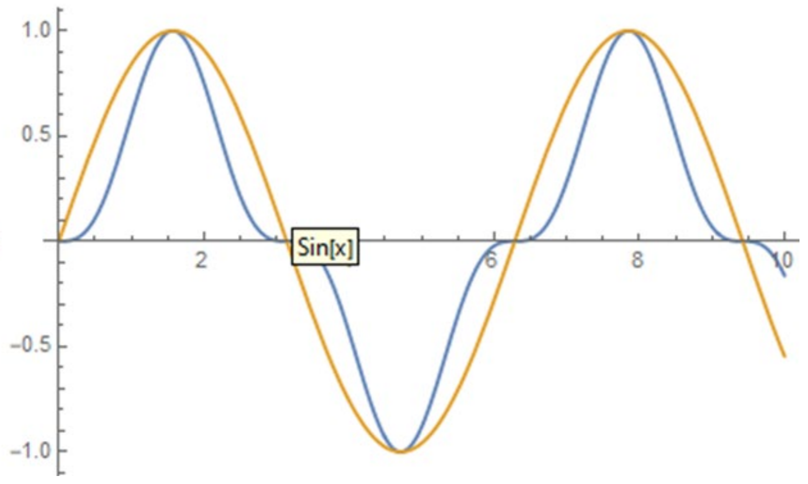
```
In[112]:= Tooltip[2 + 3, "Це сума 2+3"]

Out[112]= 5

          Це сума 2+3
```

Fig. 15. Example of using the Tooltip function []

If you hover the mouse cursor over the result displayed in the line, the text text "This is the sum of 2 +3!", Which was specified as a label parameter, will be displayed..
In another example, Fig. 16, shows the use of this function to recognize one of two curves constructed in one color.

```
In[113]:= Plot[Tooltip[{Sin[x]^3, Sin[x]}], {x, 0, 10}]
```

Out[113]=



```
In[113]:= Plot[Tooltip[{Sin[x]^3, Sin[x]}], {x, 0, 10}]
```

Out[113]=

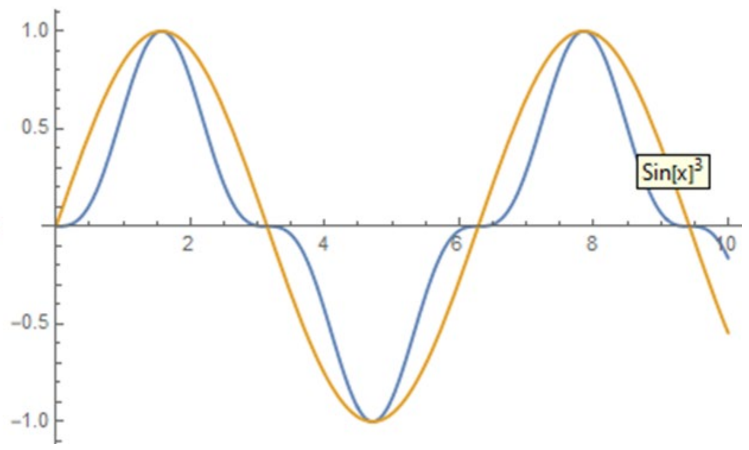


Fig. 16. Graphic recognition by Tooltip function []

To recognize a graph with the Tooltip [] function, you must hover the mouse cursor over one of the curves. Fig. 17 shows an example of using the Tooltip [] function to determine the exact value of the ordinate of a point graph of a sinusoid when hovering over the desired point of the mouse cursor.

```
In[114]:= ListPlot[Table[Tooltip[N[Sin[i/2]]], {i, 20}]]
```

Out[114]=

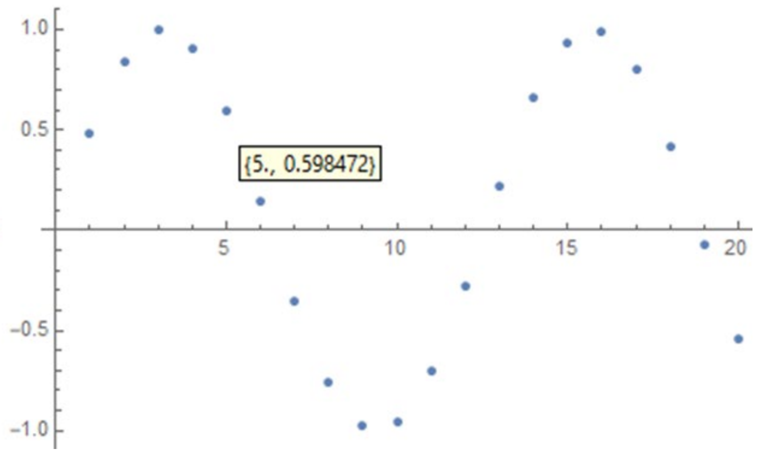


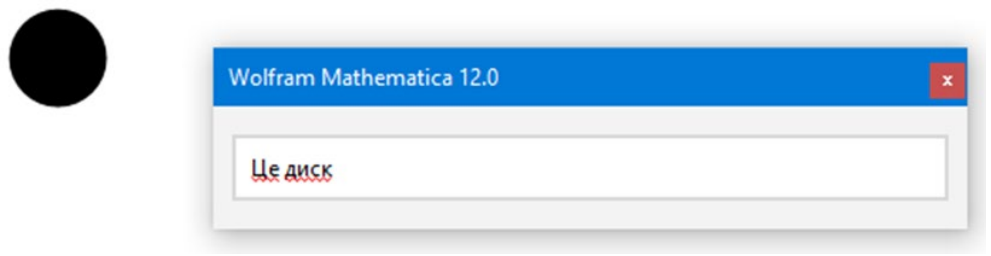
Fig. 17. Determining the value of the ordinate of a scatter plot by the Tooltip function []

Cursor-activated messages

The function PopupWindow [] provides control over the mouse cursor placed in it object. If the cursor is placed on the object, it is modified and a panel with the specified message appears. For example, in Fig. 18 object is a circle, which at the time of hovering the mouse cursor changes color. Then, after the panel labeled "This is a disk" appears, the circle restores color.

```
In[115]:= PopupWindow[Graphics[Disk[], ImageSize → 50], "Це диск"]
```

Out[115]=



Wolfram Mathematica 12.0

Це диск

Fig. 18. Messages activated by the cursor

Display the menu and select its position

The MenuView function [] provides the creation of menus with drop-down items under numbers that correspond to the value of the variable n. In the example of Fig. 19 calculate the function tan (nx) for different n.

```
In[116]:= MenuView[Table[Plot[Tan[n*x], {x, -4, 4}], {n, 5}]]
```

Out[116]=

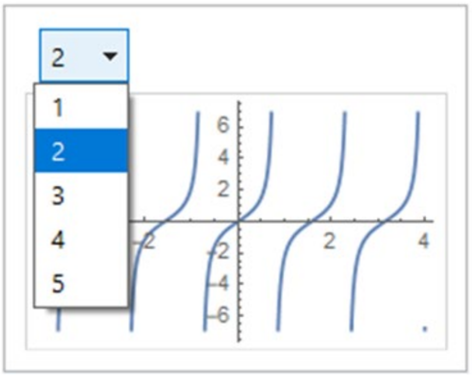


Fig. 19. Example of using the MenuView function []

Display of menus with attachments and navigation between them

The TabView function [] displays a menu with attachments that you can switch between by left-clicking. An example of the application of this function is shown in Fig. 20. In this case, an image is created in the object window, which is created by a cellular automaton using the CellularAutomaticc [] function with different values of the r (rule) parameter. Possible values of r are selected from the list by activating the corresponding tab with the mouse.

```
In[117]:= TabView[
  Table[Row[{"rule", r}] →
    ArrayPlot[CellularAutomaton[r, {{1}, 0}, 30]],
   {r, {30, 50, 90, 150}}]]
```

Out[117]=

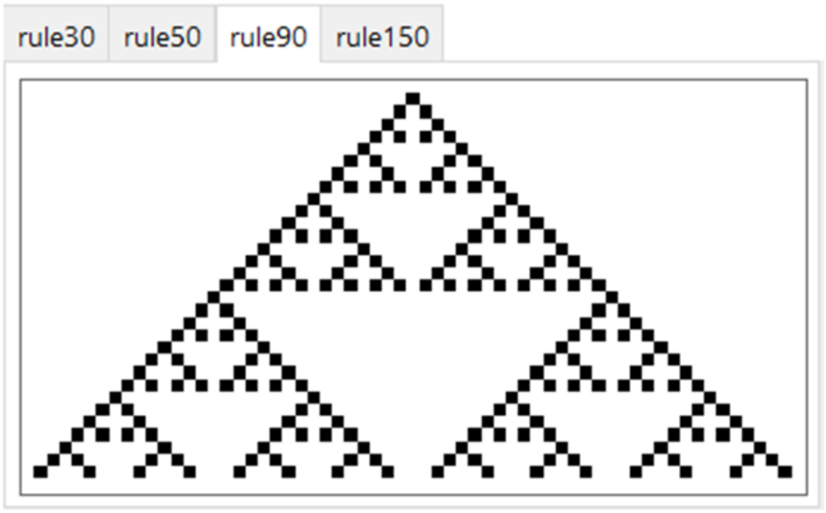
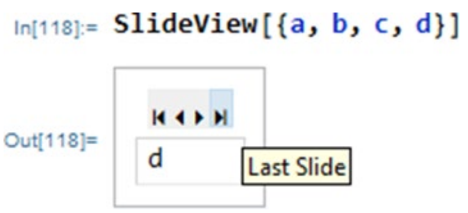


Fig. 20. Example of using the TabView function []

Slide menu

Slide menus are used to create presentations. To create a slide menu, use the SlaidMenu [] function. Working with the function is shown in Fig. 21. When activating the buttons with triangle images, you can select a symbol from the list. Switching can go in any direction, as well as immediately at the beginning or end of the list.



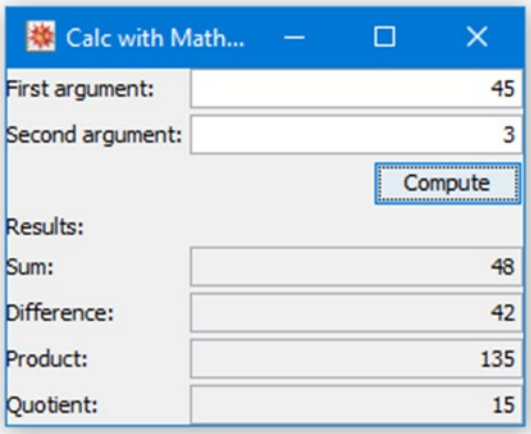
```
In[118]:= SlideView[{a, b, c, d}]
Out[118]= d
```

Fig. 21. Example of using the SlaidMenu function []

Creating windows of interactive dialogue with the user

Mathematica provides the ability to create dialog boxes based on the GUIKit extension package. Fig. 22 shows a call to this packet and a window to demonstrate arithmetic operations with two input numbers. To create even such a simple example requires a program containing about sixty lines. It is represented by the file Calculator.m.

```
In[119]:= Needs["GUIKit`"];

In[120]:= GUIRunModal["Wolfram/Example/Calculator"]
```



First argument:	45
Second argument:	3
Results:	
Sum:	48
Difference:	42
Product:	135
Quotient:	15

Fig. 22. Example of using GUIKit

Lecture № 14. Rules for writing effective code and debugging programs

Like any software system, Mathematica is designed to automatically process information - in our case mathematical. Most often, this processing is reduced to calculations and their visualization, such as graphics.

Information can be represented in a variety of ways: numbers, mathematical formulas, text symbols and other elements of information.

At the level of its input programming language and communication with the user, the Mathematica system operates with three main data classes:

- numerical, representing numbers of different types;
- symbolic, representing symbols, texts and mathematical expressions (formulas);
- lists - data in the form of many similar or different types of data.

Each of these data classes, in turn, has a number of special, more private data types.

Working with integers

Mathematica uses integers with different bases and decimal numbers with a floating point (they are often called floating point numbers), presented in different notations. Of the integers widely used binary numbers with base 2, octal with base 8, decimal with base 10 and hexadecimal numbers with base 16. The most common are decimal numbers (DECIMAL). Each digit of such numbers has a representation given by one of the Arabic numerals: 0, 1, 2, 9. Weighting factor of the highest digit relative to the previous one is equal to 10.

The construction is used to calculate numbers with an arbitrary base **Basis ^^ number**.

The number must be written according to the rules of writing numbers with the appropriate basis. For bases greater than 10, the letters a to z are used to denote the values of numbers. The most well-known of the numbers with a bit rate of more than 10 are hexadecimal numbers (HEX - from the word hexagonal). The digit of such numbers can matter:

HEX 0 1 2 3 4 5 6 7 8 9 a b c d e f

DECIMAL 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

The senior category has a weighting factor relative to a given category, equal to16.

Examples of problems of hexadecimal and life numbers:

16 ^^ 123abcde

305839326

2 ^^ 1010111

87

A function is used to represent numbers with an arbitrary base n (up to 32) **BaseForm [expr, n]** – returns the expression expr in the form

BaseForm [87,2]

1010111_2

BaseForm [305839326,16]

123abcde$_{16}$

To obtain lists of numbers of different integers is a function **IntegerDigits [n, b, len]**, where n is a number, b is the base, and len is the length of the numerical sequence, supplemented by zeros on the left. Parameters b and len may be missing. Examples of application of this function are presented below:

IntegerDigits [1234]

{1,2,3,4}

IntegerDigits [1234,2]

{1,0,0,1,1,0,1,0,0,1,0}

IntegerDigits [10!, 8]

{1,5,6,5,7,4,0,0}

IntegerDigits [10!, 10,12]

{0,0,0,0,0,3,6,2,8,8,0,0}

In fact, the methods used in symbolic mathematics to represent numbers are more compact than the simplest, but this does not change the essence of the main thing: the greater the number of digits of the number, the more memory is allocated for its storage. Particular attention is paid to the compact storage of numbers in the Mathematica system, which has reduced several times the memory consumption for storing large numerical arrays and reduce the time spent working with them.

A typical example of working with large-digit integers is the calculation of the factorial n! = 1 * 2 * 3 * ... * n (under 0! = 1):

1000! / 950!

```
287731343120013000702468807306664495322316807860141258 \
460384342116480177434914877476604012266378453263443734 \
5798335770376809481108035993600000000000000
```

(20! +5!) / 22!

$$\frac{20274183401472001}{93666727314800640000}$$

Note that the "\" sign at the end of the output line of the first example means the translation of the following characters to a new line.

Integer data are integers, such as 1, 2, or 123, that are provided by the system without error or bit limit.

Moreover, arithmetic operations on integers the system also performs without errors and without limiting the number of digits:

+123456789123456789123456789 ^ 2

```
15241578780673678546105778281054720515622620750190521
```

% / +123456789123456789123456789

```
123456789123456789123456789
```

100000000000000000000000000000000000000000000000000 + 123

```
100000000000000000000000000000000000000000000000123
```

10000000000000000000000000000000000000000000000000-1

999999999999999999999999999999999999999999999999

Rational data are given by the ratio of integers, such as 123/567, and also represent the result accurately:

1000000/3000000

$\frac{1}{3}$

(124-1) / (455 + 1)

$\frac{41}{152}$

Working with real type numbers

Real numbers can have different forms, for example 123.456, 1.23456 10 ^ 2, 12345.6 10 ^ 2 і т.д. In the general case, they contain a mantissa with integer and fractional part and an order that is entered as the power of 10. As a rule, real numbers in symbolic mathematics systems can have a mantissa with any but finite number of characters..
The space between the mantissa and the order is equivalent to the multiplication sign *:

23.456 * 10 ^ 100

2.3456×10^{101}

10 ^ -100

1/100000000000000000000000000000000000000000000000000000000 \
00000000000000000000000000000000000000000000

10. ^ - 100

1×10^{-100}

As is customary in most programming languages, the whole part of the mantissa is separated from the fractional part by a dot, not a comma. The sign "\" means the translation of a line (part of a number). It, by the way, can be used in the input cells.

Mathematica performs operations on numbers, initially as integers. However, setting the punctuation mark means that the number is considered valid. For example, the number 1 is an integer, but 1. is already a real number. The function is used to represent the expression expr in the form of a real number **N [expr]** or **N [expr, number_digits_result]**

1/3

$\frac{1}{3}$

1./3

0.333333

N [1/3]

0.333333

N [2 * Pi, 50]

6.283185307179586476925286766559005768394338

N [2 * Pi, 500]

6.28318530717958647692528676655900576839433879875021164194988918461563
\632812572417997256069650684234135964296173026564613294187689219101l1
6\446345071881625696223490056820540387704221111928924589790986076392

```
88\57621951331866892256951296467573566330542403818291297133846920697
220\9086532964267872145204982825474491740132126311763497630418419256
5850\818343072873578518072002266106109764093304276829390388302321886
61145\40731519183906184372234763865223586210237096148924759925499134
703771\5054497824558763660238983
```

Real numbers always have some error in representing the results due to their inevitable rounding and the existence of the so-called "machine zero" - the smallest number that is perceived as zero.

Mathematica has two system variables that allow you to display the values of the maximum and minimum possible values of the numbers with which the system operates: **$MaxMachineNumber та $ MinMachineNumber.**

Character data and strings

Symbolic data in the General case can be individual characters (for example, a, b, ..., z), strings (strings) and mathematical expressions expr (from expression - expression), represented in symbolic form. Character strings are specified by a string of characters in quotation marks, such as "Sssss". They can use the following control characters for small objects:

\ **n** new line (line feed),

\ **t** tabulation.

It is shown in following examples:

«Hello my friend!»

Hello my friend!

«Hello \ nmy \ friend!»

Hello

my

friend!

«Hello \ tmy \ tfriend!»

Hello my friend!

Keep in mind that control characters are not displayed by the printer and are not displayed, but only force them to perform their assigned actions. Mathematica has many functions for working with strings, which will be described below.

Both operators and functions are used to write mathematical expressions. Their features will be discussed later. In the meantime, let's note some subtleties of the syntax of the system used when writing arithmetic operations:

- the multiplication sign can be replaced by a space;
- built-in functions start with a capital letter and usually repeat their common mathematical notation (except for those whose names have Greek letters - are they reproduced in Latin?
- we letters by the sound of the corresponding Greek letters);
- parentheses (...) are used to highlight parts of expressions and prioritize their execution;
- parameters of functions are set in square brackets [...];

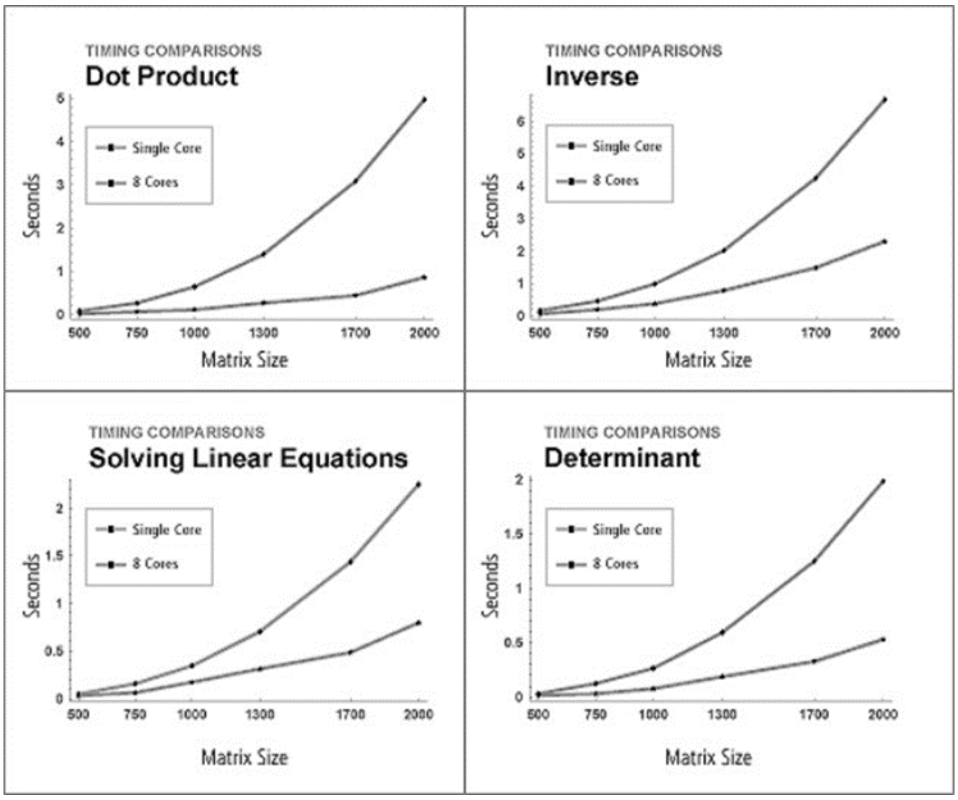
• curly braces are used when specifying lists {...}.

Support for multi-core microprocessors

Currently, the leading manufacturers of microprocessors (Intel, AMD, IBM, etc.) are moving to the production of new multicore microprocessors (multicore processors), which implement methods of parallel computing, are previously used only in the super? Computers (including clusters). Mathematica became the first SCM, in some versions of which for the first time provided both Hyper Threading technology and support for the capabilities of the latest multi-core processors. These advantages are realized primarily in solving problems in linear algebra. In this area, the easiest way to break down computational processes into separate parts performed by individual processor cores.

The interface part of Mathematica is implemented as a separate process, separate from the processes of its computing core. This creates a dialog interface even when the multi-core processor is under maximum load. Even when using a dual-core Mathematica processor provides work on various kernels of the interface module and a computing kernel (kernel). As a result, the "reflection" time, traditionally significant for previous versions of Mathematica, has been reduced by about 1000 times.

The figure shows a comparison of computation times in solving the four most characteristic problems of linear algebra in SCM Mathematica on a computer. Comparative results of matrix calculations in Mathematica for PCs with single-core and 8-core microprocessors.



Lecture № 15. Mathematica data storage commands

Although Mathematica is focused on mathematical applications, it is quite complete with functions for working with strings. They may be needed both for organizing the output of text messages (such as labels on graphs) and for organizing text dialogue when developing extension packages and system applications. In addition, we must always remember that Mathematica is a system of symbolic mathematics, so that the symbolic transformation, both purely mathematical and conventional, it, of course, is given a lot of attention.

Many functions for working with strings perform common transformations, available in most high-level programming languages. String is an arbitrary string of characters enclosed in quotation marks, such as "String". Below are some functions for working with strings:

- StringByteCount ["string"] - returns the full number of bytes used to store characters in the string "string";
- StringDrop ["string", {m, n}] - returns the string "string", deleting characters from m to n;
- StringJoin ["sl", "s2", ...] or StringJoin [{"s1", "s2", ...}] - forms a string containing the concatenation (merging) of the specified strings "s1";
- Stringlnsert ["string1", "string2", M] - inserts the string "string2" into the string "string1", starting from the position M from the beginning of this string (with a negative M the position is subtracted from the end of the specified string);
- StringLength ["string"] - returns the number of characters in a string;
- StringReplace ["string", "s1 ->" spl "] or StringReplace [" string ", {" s1 "-> "spl", "s2" -> "sp2", ...}] - replaces s1 "with" spi "whenever they appear as a substring" string ";
- StringReverse ["string"] - changes the order of the characters in the string "string" to the opposite;
- StringPosition ["string", "sub"] - returns a list of line items "sub" in the line "String";
- StringTake ["string", n] - returns a string consisting of the first n characters of the string "String";
- StringTake ["string", -n] - returns the last n characters of the string "string";
- StringTake ["string", {n}] - returns the nth character in the string "string";
- StringTake ["string", {m, n}] - returns a string of characters located in positions m to n of the string "string".

These features are well known to programmers working with modern programming languages. A large number of additional functions for working with strings can be found in the application. The large number of such functions in the programming language of the Mathematica system indicates its universal nature and great opportunities in solving even seemingly far from mathematics problems. The following are examples of a number of string operations.

In	**Out**
StringByteCount ["Hello!"]	6

StringDrop ["Hello my friend!", 6]	my friend!
StringDrop ["Hello my friend!", -10]	Hello
StringDrop ["Hello my friend!", {7}]	Hello y friend!
StringDrop ["Hello my friend!", {6, 8}]	Hello friend!
Stringlnsert ["Hello friend!", "My", 6]	Hello my friend!
StringJoin ["Hello", "my"] <> "friend!"	Hello my friend!
StringLength ["Hello"]	5
StringPosition ["Hello my friend!", "E"]	{{2, 2}, {13, 13}}
StringReplace ["Hilo", "i" -> "el"]	Hello
StringReverse ["Hello!"]!	OlleH
StringTakef "Hello my friend!", 6]	Hello
StringTake ["Hello my friend!", -8]	friend!
StringTake ["Hello my friend!", {7, 9}]	my

Note a few more features related to working with characters and strings:

- FromCharacterCode [n] - returns a string consisting of one character with code n;
- FromCharacterCode [{n1, n2, ...}] - returns a string consisting of a sequence of characters with codes ni;
- Characters ["string"] - returns a list of integer codes corresponding to the string characters "string";
- ToLowerCase ["string"] - produces a string in which all letters are converted to lowercase;
- ToString [expr] - returns a string corresponding to the output form of the expression exrr. Options set line width, format type, etc .;
- ToUpperCase ["string"] - produces a string in which all letters are converted to uppercase;
- Unique [] - creates a new character with a name in the form $ nnn (nnn - a unique sequence number);
- Unique [x] - creates a new character with a name in the form x $ nnn (nnn - a unique sequence number);
- Unique [{x, y, ...}] - creates a list of new characters with unique names;
- Unique ["xxx"] - creates a new character with a name in the form xxxnnn (nnn - a unique sequence number);
- Unique [name, {attrl, attr2, ...}] - creates a character with the specified attri attributes;
- UpperCaseQ [string] - returns True if all string characters are uppercase (uppercase), otherwise returns False.

The examples below show how to work with these functions.

In	**Out**
ToCharacterCode ["Hello!"]	
{72,101,108,108,111,33}	
FromCharacterCode [{72,101, 108, 108, 111, 33}]	Hello!
ToExpression ["2 + 3 * 4"]	14

ToLowerCase ["HeLLo!"]	Hello!
ToUpperCase ["Hello"]	HELLO

Введення (In)	**Виведення (Out)**
x: = ToString [2 + 3 * 4]	
X	14
Unique []	$ 1
Unique [xyz]	xyz $ 2
Unique [xyz]	xyz $ 3
UpperCaseQ ["Hello"]	False
UpperCaseQ ["HELLO"]	True

Streams and files

The Mathematica system has advanced tools for working with streams and files. A stream is a continuous sequence of data circulating inside a computer. The exchange of threads occurs almost continuously, for example, when entering the input stream comes from the keyboard to the computer, when printing the data stream comes from the computer to the printer through the printer port, etc.

A file is an ordered data structure that has a name and is stored on any medium, most often on a magnetic disk. Files can have different formats and different types of access to the information stored on them. The most common document files in the Mathematica system are sequential access files and have a text format.

Serial access means that information from an open file can be read strictly sequentially from its beginning to the end indicated by a special label. it resembles reading from a tape. The text format means that all data is written as ASCII code. Therefore, you can read such a file using any text editor that works with texts in the form of ASCII codes.

Streams and files have a lot in common: names, a certain structure, the need to open before use and close after use. However, if the user encounters the files at the beginning of the system (you need to call a file with a demo document or save it, and then call another file), then the concept of flow at work the system is almost non-existent, although beyond our control, data streams are constantly flowing between the computer and its peripherals.

Simplified work with files

Before considering the rather large capabilities of the system to work with files in general, we note the simplified method of calling a file using the double character "<<":

<< filename

This command reads the file with the specified filename and stores in the computer's memory the definitions contained in it. The file must be specified in full, ie together with the extension. The exception is when the file is in the main directory of the system. This command is equivalent to a function **Get ["filename", key]**.

To write an object (variable, array, list, etc.) in the file are simplified commands:

• expr >> filename - passes the value of expr to a file with a given name;

• exp >>> filename - adds expr to the end of the file with the specified name.

These commands are essentially abbreviated (and therefore more convenient) forms of the following functions:

• Get ["filename", "key"] - reads a file that is encoded by the Encode function using the key "key";

• GetContext ["context '"] - loads a file with the specified context;

• Put [exprl, expr2, ..., "filename"] - writes a sequence of expri expressions to a file named filename;

• PutAppend [expr1, expr2, ..., "filename"] - appends a sequence of expri expressions to a file named filename.

Another simplified feature -! !! filename - displays the contents of the file with the specified name.

The following examples show writing a list to a C: \ ma.vat file, reading it, then adding another list to the file, and controlling the context of the file:

{{1, 2,3}, {4,5,6}, {a, b, c}} >> C: \ ma.val

<< C: \ ma. val

{{1, 2, 3}, {4, 5, б), {a, b, c}} {d, e, f} >>> C: \ ma.val

<< C: \ ma. val

{D, e, f}

!! З: \ ma.val
1, 2, 3, 4, 5, б, a, b, з d, e, f

This form of call is especially convenient for calling extension files and system applications. The file is specified according to the rules accepted in MS-DOS. Application package files have the extension .t. We have already given examples of using the definitions contained in the system extension package files.

There are a number of features for working with files:

• ReadList ["filename"] - reads all remaining expressions in the file "filename" and returns them as a list;

• ReadList ["filename", type] - reads from the file "filename" objects of the specified type type to the end of the file. Returns a list of counted objects;

• ReadList "" filename ", {typel, type2, ...}] - reads objects of the specified types typei to the end of file filename;

• ReadList ["filename", types, n] - reads only the first n objects of the specified types types from the file filename;

• Save ["filename", x1, x2, ...] - creates a file with the specified filename name, which contains the values of variables x1, x2, ...;

•! command - executes the specified command of the operating system.

Suppose that a text editor creates a file with the full name C: \ datas.txt in ASCII format, containing just six numbers with punctuation, placed in two lines and representing an array of 2x3 elements:

```
1 11.2 34.5
2. 3.4 56
```

Then the structure of the file can be judged using the command **!! З: \ datas.txt**

```
1 1.2 34.5 2. 3.4 56.
```

It is easy to see that the file structure corresponds to the structure of the array. However, reading the file with the << name command gives the following result:

<< C: \ datas. txt

```
380.8
```

The result represents the calculation of the expressions of another line of the file. Reading with the ReadList function without an additional argument also gives erroneous results:

ReadList ["З: \ datas.txt"]

```
{41.4, 380.8}
```

It is easy to see that the function perceived EVERY line of the file contents as the result of multiplying three numbers (a space in Mathematica means multiplication). With the additional parameter Number all numbers are read correctly:

ReadList ["З: \ datas.txt", Number]

```
{1, 1.2, 34.5, 2., 3.4, 56.}
```

However, we received a one-dimensional list - the data is simply read line by line.
Applying an additional parameter in the form {Number, Number} gives the following result:

ReadList ["З: .txt", {Number, Number}]

```
{{1, 1.2), {34.5, 2.}, {3.4, 56.}}
```

The correct result can be obtained using the option RecordList-> True.

ReadList ["C: .txt", Number, RecordLists-> True]

{{1, 1.2, 34.5), {2, 3.4, 56.}}

Share Add-On package files use features that allow you to specify the context of the files:

- Needs ["context '", "filename"] - downloads a file if the specified context is not in the download list;
- Needs ["context s"] - downloads a file whose name is determined by the ContextToFilename ["context h"] function if the specified context is not in the download list.

Downloading files with their specified contexts avoids conflicts between different extension packages used at the same time.

Unified approach

In the general case, the following functions are used to read data from an arbitrary file:

- OpenRead ["file"] - opens the specified file for reading and returns the corresponding object of type InputStream.
- Read [stream] - reads expressions with the specified input stream and returns it.
- Read [stream, type] - reads one object of the specified type and returns it.
- Read [stream, {type1, type2, ...}] - reads and returns a sequence of objects of these types.

As the type of read object can be specified:
Byte - one byte of data, represented as a whole code,
Character - a single character,
Expression - a complete expression of Mathematics,
Number - integer or real number in exponential format, Real - real number in exponential format,
Record - a sequence of characters separated by a string "\ n", String - a string ending with a newline,
Word is a sequence of characters separated by either the string "\ t" or "".
The Read function Returns EndOfFile if the end of the file is reached.
Close [stream] - closes the specified stream.

The example below reads the first line of a text file and displays it in a laptop cell window if the file is not empty:

```
fileStream = OpenRead [ "3: \ sampleFile.txt"];

textLine = Read [fileStream, String];

If [textLine ≠ "EndOfFile",

Print [textLine];

];
```

Close [fileStream];

Using files of other programming languages

Of the functions for working with files, the following function-directive should be especially noted:

- Splice ["file .mx"] - inserts into files in other programming languages calculated expressions of the Mathematica system, which must be written in parentheses of the form <* and *>;
- Splice ["infile", "outfile"] - reads the infile file, interprets the fragments between the parentheses <* and *>, and writes the result to the outfile file.

This feature is especially important when using programs in the programming languages C (.me extension), Fortran (.mf extension) and TECH (.mtex extension), for formats for which Mathematica has the means to convert expressions (CForm, FortranForm and TexForm, respectively). Thus, it is possible to export Mathematica system expressions to programs written in these languages.

Let's explain the application of the Splice directive function. Suppose there is an exported program in C, which must calculate the numerical value of some integral, and we want to get a formula for this integral by means of the Mathematica system. Suppose it is represented by the file demo.me. It can be viewed as follows:

!! demo.me

```
#include "mdefs.h"

double f (x)

double x;

{

double y;

y = <* Integrate [Sin [x] ^ 5, x] *>;

return (2 * y- 1);

}
```

After performing the function Splice ["demo.me"], the program will be written to the file demo.c, in which the expression in parentheses <* ... *> is replaced by the calculated value of the integral (in the form CForm). The file will look like this:

!! demo.c

```
#include "mdefs.h" double f (x) double x;
```

```
{

double y;

y = 5 * Cos (x) / 8 + 5 * Cos (3 * x) / 48- Cos (5 * x) / 80;

return (2 * y- 1);

}
```

Record definitions

Of the simple functions that provide files with defined definitions, it is also worth noting the Save function:

```
Save [ "filename", symb1, symb2, ...]
```

It adds symbi character definitions to the filename file (simplified Save forms are possible).
Here is an example of its use:

```
f [x_] = Sin [x] + y

y + Sin [x]

y = a

a
Save [ "demol", f]

!! demol

f [x_] = y + Sin [x]

y = a
```

Other features for working with files

In general, the tools of the Mathematica system provide the ability to work with various files, inherent in MS-DOS, without leaving the system environment. The functions related to this group are given in the appendix. These functions are characterized by the fact that at the time of execution they do not give a visible effect. Such functions include functions for copying directories and files,
change their names, delete them, etc. They are well known to MS-DOS users and can be run from the Mathematica environment.

Considering the large list of file and current operations, you can involuntarily conclude that they are redundant. But here's a simple rule: if you don't want to use these features, don't! They are designed for the user, seriously engaged in docking Mathematica systems with other software systems.

An important place is occupied by functions that provide information about directories, files and streams. These include the following functions:

- Directory [] - returns the current working directory;
- DirectoryStack [] - returns the contents of the directory stack, which represents the sequence of directories used in the current session;
- $ Display- returns a list of files and channels (pipes- channel or abstract file) used by the output function

$ DisplayFunction by default;

- FileByteCount ["filename"] - returns the number of bytes in the file;
- FileDate ["filename"] - returns the date and time of the last modification of the file as a list;
- FileInformation ["filename"] - returns information about the photo;
- FileNames [] - lists all files in the current working directory;
- FileNames ["form"] - lists all files in the current working directory whose names match the form template;
- FileNames [{"forml", "form2", ...}] - lists all files whose names correspond to any of the formi templates;
- FileNames [forms, {"dirl", "dir2", ...}] - lists files with names corresponding to forms templates in any of the specified diri directories;
- FileType ["filename"] - returns the file type: File, Directory or None (if the specified file does not exist);
- $ HomeDirectory - gives the name of the "home" directory of the user;
- $ Output - gives a list of files and channels in which the standard output of the Mathematica system is sent;
- ParentDirectory [] - returns the name of the parent directory for the current working directory;
- ParentDirectory ["dir"] - returns the name of the parent directory for the dir directory;
- $ Path - gives a list of directories to view when trying to find an external file;
- StreamPosition [stream] - returns an integer that indicates the position of the current point in the open stream;
- Streams [] - returns a list of all streams currently open;
- Streams ["name"] - lists only streams with the specified name.

The following examples illustrate the use of most of these fairly simple functions:

Directory []

C: \ PROGRAM FILES \ WOLFRAM RESEARCH \ MATHEMATICA \ 4.0

DirectoryStack []

{} **/ $ Display**

stdout

FileByteCount ["C: .val"]

46

FileDatef'C: .val "]

{1999 року, 8, 3, 16, 4, 44}

FileInformation ["C: .val"]

{File-> C: \ ma.val, FileType-> File, Date -> 3142685084, ByteCount -> 46}

Filenames []

{Examples, FILES, MATHEMATICA.EXE, MATH.EXE, MATHINSTALLER.EXE, MATHKERNEL.EXE}

FileType ["C: .val"]

File HomeDirectory []

c: \ $ 0utput

{OutputStream [stdout, 1]}

ParentDirectory []

3: \ m3 Streams []

{OutputStream [stdout, 1],

OutputStream [stderr, 2]}

The above reasoning about the redundancy of the set of operations can be applied to these functions.

Lecture № 16. Interaction of Mathematica system with MatLab program using MatLink package

There is a multi-platform package called MATLink for calling the MATLAB function directly from Mathematics and organizing the exchange of data and variables between the two systems. It makes it easy to call MATLAB functions directly from Mathematic and transfer different data from one system to another.

Installation. First, go to the MATLink home page and follow the instructions. The easiest way is to download the archive and unzip it to this folder:

In [1]: =

```
SystemOpen @ FileNameJoin [{$ UserBaseDirectory,"Applications"}]
```

Next, follow the instructions for the specific operating system in the section "Link with MATLAB" on the main page.

Using MATLink. MATLink can be loaded by calculating the cell with the code:

In [2]: =

```
Needs ["MATLink`"]
```

And then run the MATLAB command:

In [3]: =

```
OpenMATLAB []
```

This will launch a new process in MATLAB with which Mathematica will be able to interact. MEvaluate should be used to use arbitrary MATLAB commands. The data will be transmitted as a string.

In [4]: =

```
MEvaluate ["Magic (4)"]
```

Out [4] =

```
(* ==>
ans =
   16 2 3 13
    5 11 10 8
    9 7 6 12
    4 14 15 1
  *)
```

To transfer data to MATLAB, you must use MSet:

In [5]: =

```
MSet [ "x", Range [10]]; MEvaluate [ "x"]
```

Out [5] =

```
(* ==>

x =

  1 2 3 4 5 6 7 8 9 10

*)
```

Use MGet to restore data:

In [6]: =

```
MGet ["X"]
```

Out [6] =

```
(* ==> {1., 2., 3., 4., 5., 6., 7., 8., 9., 10.} *)
```

A large number of data types are supported, including liquefied arrays, structures, and cells.

MATLAB functions can be called directly from Mathematica using the MFunction function:

In [7]: =

```
eig = MFunction ["Eig"];

eig [{{1, 2}, {3, 1}}]
```

Out [7] =

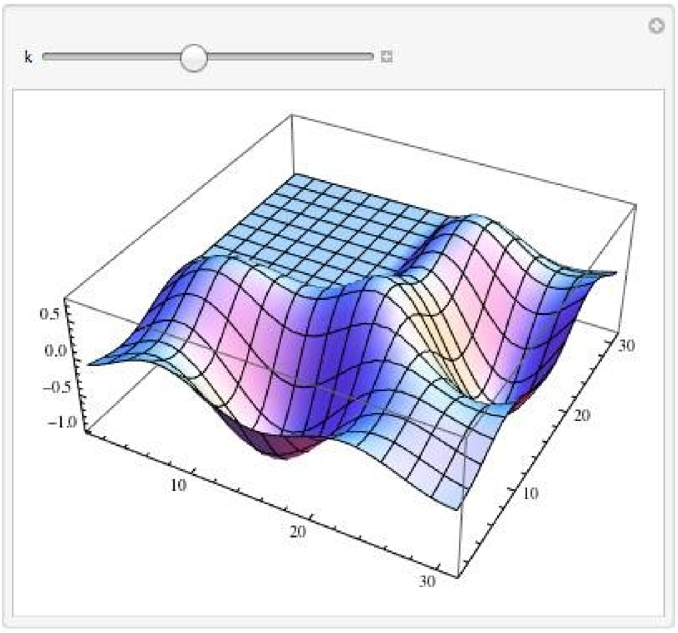
```
(* ==> {{3.44949}, {-1.44949}} *)
```

Construct in Mathematica the surface of the MATLAB logo and add manipulator that will adjust the period of oscillation:

In [8]: =

```
Manipulate [
ListPlot3D @ MFunction ["Membrane"] [k],
{K, 1, 12, 1}
  ]
```

Out [8] =

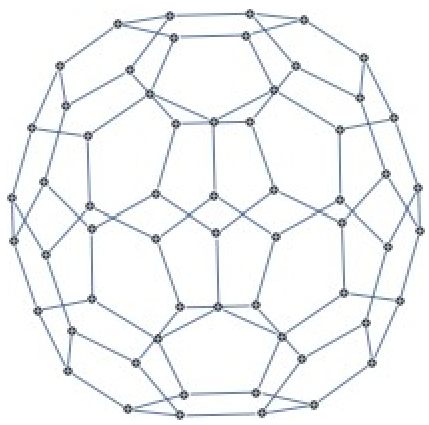


Fullerene structure (bucky ball) straight from MATLAB:

In [9]: =

```
AdjacencyGraph @ Round @ MFunction ["Bucky"] []
```

Out [9] =

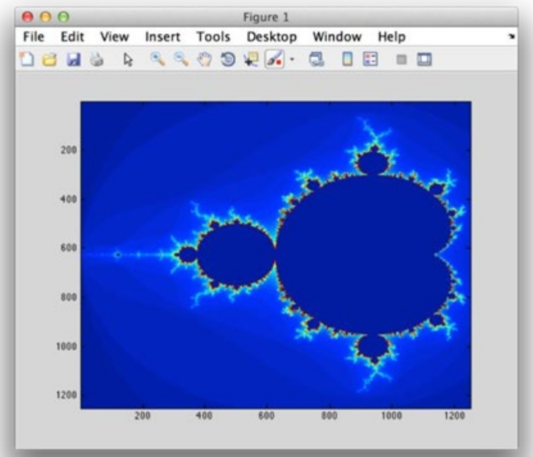


It is also easy to display Mathematica data in a separate scalable window for images used in MATLAB:

In [9]: =

```
mlf = LibraryFunctionLoad [ "demo_numerical", "mandelbrot", {Complex}, Int
eger];
mandel = Table [mlf [x + I y], {y, -1.25, 1.25, .002}, {x, -2., 0.5, .002}];
MFunction ["Image", "Output" -> False] [mandel]
```

Out [9] =



The following examples illustrate the solution of real problems using MATLink, allowing you to use the best qualities of MATLAB and Mathematica.

Rapid Delaunay triangulation. Mathematica includes the DelaunayTriangulation function inside the ComputationalGeometry package (In version 10, this package became built into the kernel and is now called DelaunayMesh. It is optimized and now its performance is not inferior to MATLAB - ed.), But it works very slowly (although it also has its strengths, such as the use of exact arithmetic and working with collinear points). This, in turn, leads to the fact that ListDensityPlot works very inefficiently (which becomes noticeable when building several thousand points or more). Using MATLink, we can use the Delaunay function from MATLAB to calculate the Delaunay triangulation of some set of points as follows:

In [10]: =

```
delaunay = Composition [Round, MFunction ["Delaunay"]];
```

Since the Mathematica function returns a list of contiguous vertices, we need to post-process the result in order to compare with the result from MATLAB'a:

In [11]: =

```
Needs ["ComputationalGeometry`"];
delaunayMma [points_]: =
        Module [{tr, triples},
                tr = DelaunayTriangulation [points];
                triples = Flatten [
                        Function [{v, list},
                                Switch [Length [list],
                                (* Account for nodes with connectivity 2 or less *
)
                                1, {},
                                2, {Flatten [{v, list}]}, _, {v, ##} & @@@ Partiti
on [list, 2, 1, {1, 1}]]
                        ] @@@ tr, 1];
                Cases [GatherBy [triples, Sort], a_ /; Length [a] ==3 :> A[[1]]]
        ]
```

A random set of points usually has a unique Delaunay triangulation, so we will need to verify that the systems give the same result.

In [12]: =

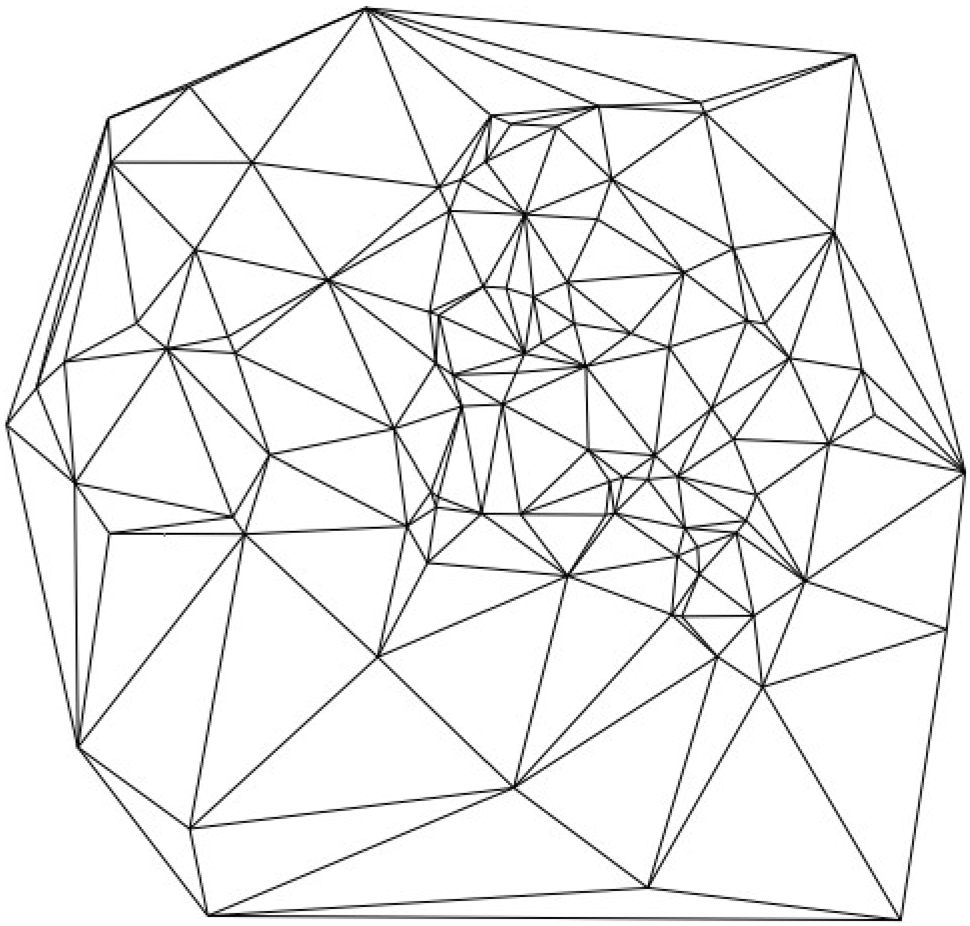
```
pts = RandomVariate [NormalDistribution [], {100, 2}];
Sort [Sort / @ Delaunay [pts]] === Sort [Sort /@ DelaunayMma [pts]]
```

And build a triangulation with:

In [13]: =

```
trianglesToLines [t_]: =
Union @ Flatten [{{# 1, # 2}, {# 2, # 3}, {# 1, # 3}} & @@
            Transpose [Sort / @ t], {{1, 3}, {2}}];
Graphics @ GraphicsComplex [pts, Line @ trianglesToLines @ delaunay [pts]]
```

Out [13]: =



However, in addition to the fact that delaunay runs much faster than

DelaunayTriangulation (especially for large datasets), it is also a faster triangulator that is used inside ListDensityPlot. So we can use delaunay from MATLAB to develop our own version of listDensityPlot, which runs faster than the built-in function and can also handle large data sets as follows:

In [14]: =

```
Options[ListDensityPlot] = Options[Graphics] ~Join~ {ColorFunction -> Autom
atic, MeshStyle -> None, Frame -> True};
```

```
listDensityPlot [data_? MatrixQ, opt: OptionsPattern []]: =
          Module [{in, out, Tri, colfun},
                    tri = delaunay [data [[All, 1;;2]]];
                    colfun = OptionValue [ColorFunction];
                    If[Not@MatchQ [colfun, _Symbol | _Function],Check[Colfun =
ColorData [colfun], colfun = Automatic]];
                    If[Colfun === Automatic, colfun = ColorData [ "LakeColors"]];
                    Graphics [
                              GraphicsComplex [data [[All, 1;;2]],
                                        GraphicsGroup [{EdgeForm [OptionValue [Mesh
Style]], Polygon[Tri]}],
                                        VertexColors -> colfun / @ Rescale [data [[All, 3]
]]
                              ],
                              Sequence @@ FilterRules [{opt}, Options[Graphics]], Me
thod -> { "GridLinesInFront" -> True}
                    ]
          ]
```

Let's compare the received function with built-in, using thus an array from 30 000 points:

In [15]: =

```
pts = RandomReal [{-1, 1}, {30000, 2}];
```

```
values = Sin[3 Sqrt[# 1 ^ 2 + # 2 ^ 2]] & @@@ pts;
```

In [16]: =

```
listDensityPlot [ArrayFlatten [{{pts, List / @ Values}}], Frame -> True] // Absol
uteTiming
```

Out [16] =

```
{0.409001, --Graphics--}
```

In [17]: =

```
ListDensityPlot [ArrayFlatten [{{pts, List / @ Values}}]] // AbsoluteTiming
```

Out [17] =

```
{12.416587, --Graphics--}
```

The difference in execution speed turned out to be quite significant (~ 30 times). To work with hundreds of thousands of points ListDensityPlot is almost unsuitable, while listDensityPlot takes only a few seconds.

It is also important to note that to measure the speed of MATLink, you must use the AbsoluteTiming function, which calculates all the time spent, while Timing measures only the time when the CPU was used by the Mathematica core, without measuring the time spent by MATLAB.

Audio filtering with signal processing tools (signal processing toolbox). As you know, signal processing functionality was missing in Mathematica until the ninth version, and is still inferior to MATLAB tools in terms of functionality and ease of use. Suppose we have 8 versions of Mathematica, new features are missing and we want to perform frequency analysis of some audio file and implement filtering. Here's how to do it:

In [18]: =

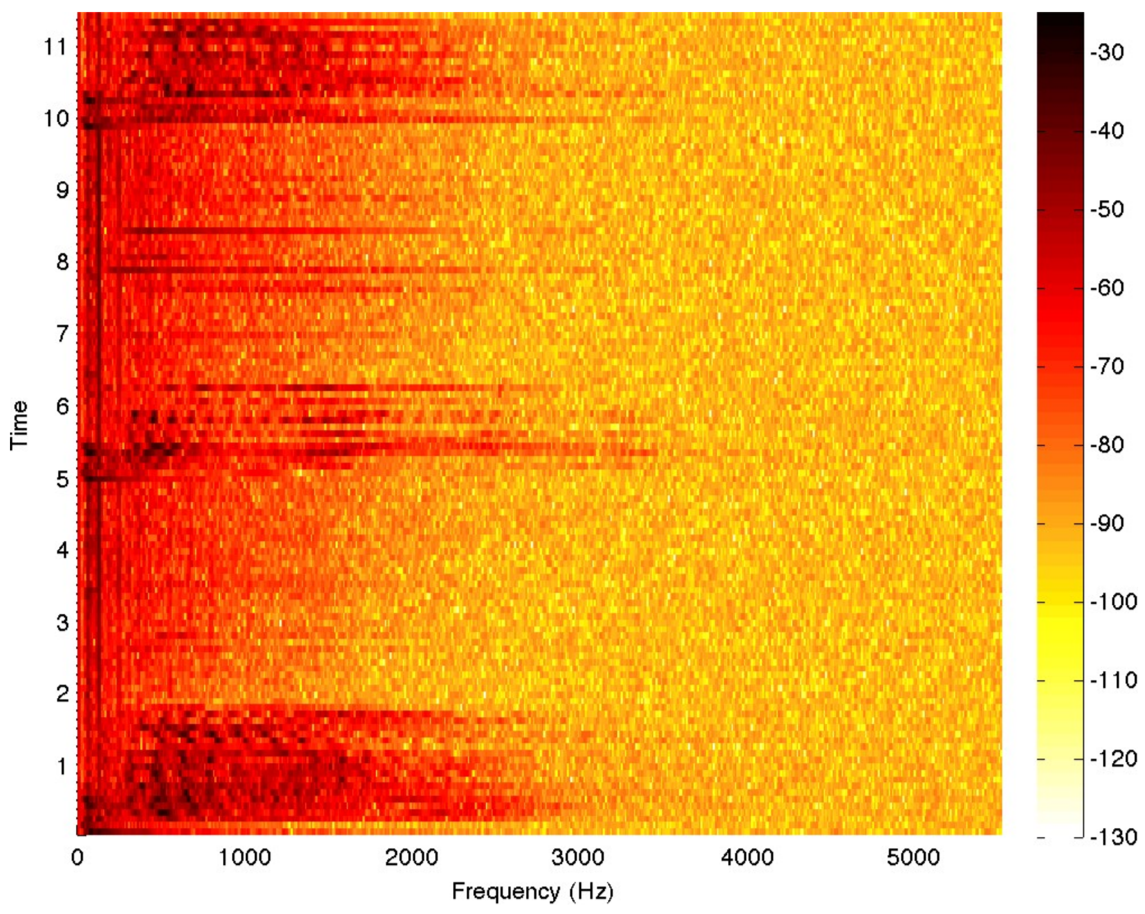
```
{data, Fs} = {# [[1, 1, 1]], # [[1, 2]]} & @ExampleData[{ "Sound","Apollo13Pr
oblem"}];
spectrogram = MFunction["Spectrogram", "Output" -> False]; (*Use MATLAB'
s spectrogram *)
spectrogram[data, 1000, 0, 1024, fs]
```

Obviously, the frequencies mainly fall within the range below 2.5 kHz, so we can develop a MATLAB low-pass filter, as well as make an auxiliary function that will return the filtered data:

In [19]: =

```
MSet ["Fs", Fs];
MEvaluate ["
        [Z, p, k] = butter (6, 2.5e3 / fs, 'low');
        [Sos, g] = zp2sos (z, p, k);
        Hd = dfilt.df2tsos (sos, g);
  "]
filter = MFunction ["Myfilt", "@ (X) filter (Hd, x)"];
```

Out [19] =



Now we have prepared everything in order to apply the filtering function to the data directly from Mathematica. This example shows how we can fill in the gaps in functionality. This way, we can save a lot of time on filter design in Mathematica (which is not the easiest task) and many hours on debugging it. The code for the Butterworth filter can be taken from anywhere - from a file share or Stack Overflow, from fragments of previously written code, or, as in this case, from the example in the documentation. Small changes in the parameters according to your needs, and we can now work with this material in Mathematica.

We will process some data by means of our filter and we will construct the spectrogram:

In [19]: =

```
filteredData = filter @data;
```

```
spectrogram [filteredData, 1000, 0, 1024, Fs]
```

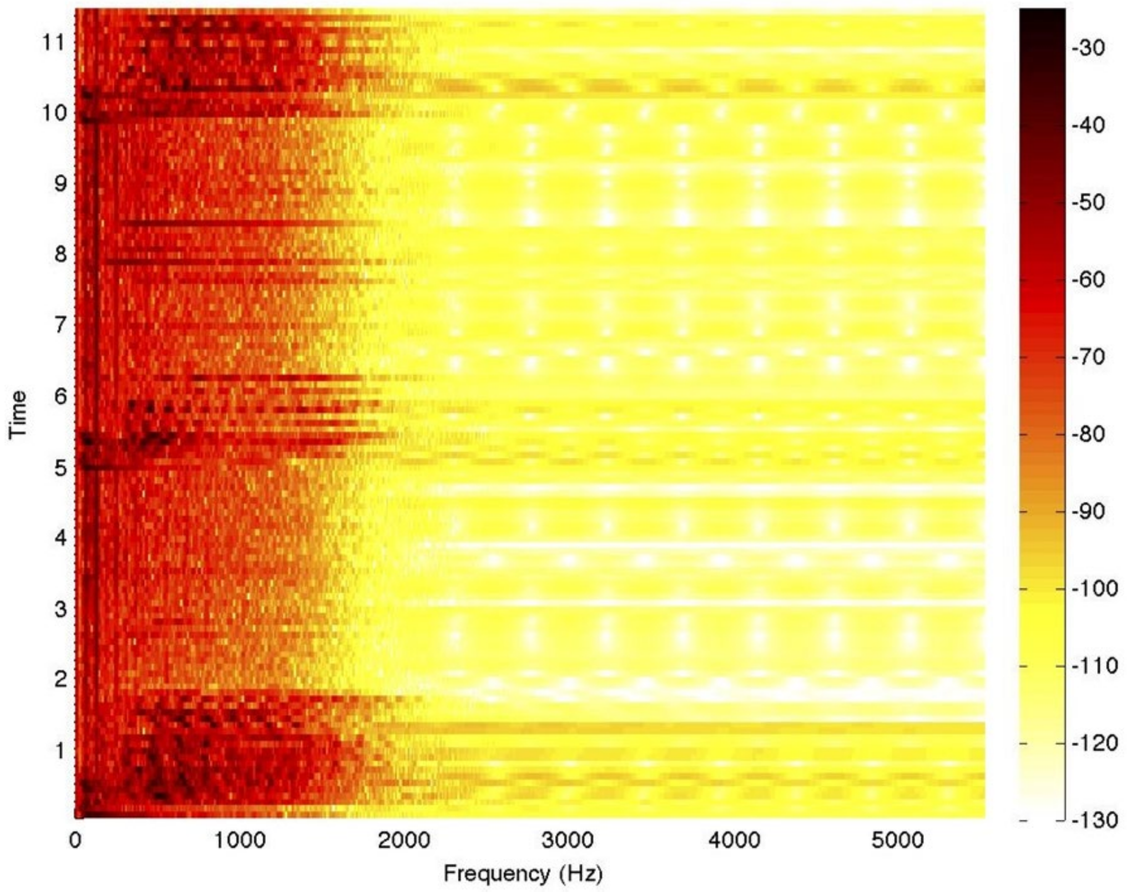
We can play both audio files - filtered and original - and compare the difference in their sound:

In [20]: =

```
ListPlay[data, SampleRate -> fs]
```

```
ListPlay[FilteredData, SampleRate -> fs]
```

Out [20] =



Literature

1. Mathematica 5. Tutorial. System of symbolic, graphic and numerical calculations / Shmidsky J.K. - Dialectics 2004 - 592 p.

2. Chen K., Jiblin P., Irving A. Matlab in mathematical research: Per. from English - M .: Mir, 2001.346 p.

3. Differential Equations with Mathematica, Third Edition / Brian R. Hunt, Ronald L. Lipsman, John E. Osborn, Donald A. Outing, Jonathan Rosenberg - 2009 JohnWiley & Sons, 271 pp.

4. A Physicist's Guide to Mathematica, Second Edition / Patrick T. Tam – 2008Academic Pres, 728 pp.

5. Computer Solutions in Physics: With Applications in Astrophysics, Biophysics, Differential Equations, and Engineering / Steve VanWyk - World Scientific 2008 - 282 pp.

6. Mathematica by Example, Fourth Edition / Martha L. Abell, James P. BraseltonPublisher: Academic Press 2008 - 576 pp.

7. Mathematica DeMYSTiFied / Jim Hoste - McGraw-Hill Professional 2008 – 320 pp.

8. Mathematica Navigator: Mathematics, Statistics and Graphics, Third Edition /Heikki Ruskeepaa Academic Press 2009 - 1136 pp.