Language Guide

“ MATHPAR ”

VERSION 15.00

Gennadi Malaschonok

25.01.2025

CONTENTS

Chapter 1

Introduction

This guide for language "Mathpar" will help you to solve mathematical problems. You could use Mathpar at school and at home, at university and at work.

You can use it when you wish to do a simple numerical and algebraic operations or to plot functions.

It will help you to solve problems of different branches of mathematical analysis, algebra, geometry, problems in physics, chemistry, and more.

If you are a professional mathematician, you can get rid of the routine calculations and manipulate very large mathematical objects, using supercomputers.

You can operate with functions and functional matrices, to obtain the exact numerical and analytical solutions and solutions in which the numerical coefficients have a required accuracy.

For the first acquaintance with Mathpar and study of simple functional symbolic-numerical operations it is sufficient to study the first three chapters. The rules of data entry and running the calculations is described in the second chapter. Designations are given for elementary functions such as logarithm, sine, cosine, etc., and constants — π, e, i. There are described how to specify a vector and matrix, an arithmetic operations with vectors, the generator of random numbers, random polynomials and random matrices, how to solve an algebraic equation. You can see an example and the results for each command.

The third chapter is devoted to the construction of graphs of functions. The system allows you to plot functions defined explicitly, para-

metrically or points. Moreover, you can build multiple graphs in one coordinate system. This chapter provides commands for plotting and examples of commands.

The fourth chapter describes how to set the mathematical environment, i.e. a space of mathematical objects. At any point the user can change the environment, setting a new algebraic space. Moving from some environment to the current environment, as a rule, should be performed explicitly. In some cases, such a transformation to the current environment is automatic.

The fifth chapter describes the commands to specify the mathematical functions of one or more variables, their compositions, the calculations of the function at a point, substitution of the function, calculate the limit of a function at a point, symbolic integration compositions of elementary functions. For each example the results of calculations are given.

The sixth chapter is devoted to the series. The commands for adding, subtracting, multiplying of series and for the expansion of a function in a Taylor series with a certain number of members are given. We consider some examples.

The seventh chapter describes the commands for the solution of ordinary differential equations and systems, as well as systems of differential equations with partial derivatives.

The eighth chapter is devoted to polynomial computations. We consider the evaluation of polynomial at the point, the summation of the polynomial, computation of Gröbner bases of polynomial ideals over the rational numbers. For each command an example is given.

The ninth chapter describes the calculation of matrix functions. We can find the transposed matrix, the determinant of a matrix, the adjoint and inverse matrix, echelon form of matrix, the kernel, the characteristic polynomial of the matrix, and others.

The tenth chapter is devoted to the functions of probability theory and mathematical statistics. We can find here how to specify a discrete random variable, the command to calculate the expectation of a discrete random variable, variance, standard deviation, sum and product of two discrete random variables, the coefficient covariance of discrete random variables, the correlation coefficient, the construction of the polygon distribution and the distribution function of a discrete random variable. This chapter describes the commands to specify sampling and

computing functions for them: the sample mean, sample variance, factor covariance and correlation coefficient for two samples.

You can create your procedures and functions. The eleventh chapter describes Mathpar as a procedural programming language with statements, procedures and functions. Examples of written procedures using branching statements and loops statements are given.

In the twelfth chapter describes the commands that control the computation on a supercomputer. In order to solve computational problems that require a lot of computation time or large amounts of memory space you can use special functions, which provide the user with the resources of supercomputer. You can compute a Gröbner basis, adjoint matrix, echelon form of the matrix, inverse matrix, determinant, the kernel of a linear operator, the characteristic polynomial, etc.

In the thirteenth chapter lists the major operators is given.

Chapter 2

Acquaintance and first steps

This chapter is devoted to a first acquaintance with possibilities which Mathapar opens. The language MathPar, which is described below, may be considered as a kind of development of a TeX language. TeX serves for writing mathematical texts and preparing them for publication. It may be called "passive" in comparison with the language MathPar, which permits to execute computations, and so is a self-depended mathematical language. A problem definition and a result of computations are written in MathPar.

Just after computations you see the whole mathematical text as a pdf-image which is accustomed in scientific and technical publications.

The result may be further used in different ways.

(1) You may click the text with your mouse, and it would return to the initial form of the MathPar language. Then you may continue to edit the text or pass to a next task. There is another way to change a form of a text — using a button "⤮", placed between buttons "►" and "+".

(2) You may click **the image of the mathematical text** with the right mouse button, and the drop-down menu appears. The upper field **Show-Math-As** permits to pass to the choice of language. It is suggested to chose Tex or MathML. You may open a field you need.

For example, a matrix of the size 2×2 will be written in MathPar in the following way:

A=[[a,b],[c,d]];
in TeX as follows:
A= \left(**begin**{*array*}{*cc*}*a*&*b* *c*&*d*\\ **end**{*array*}\right).
In MathML it is much more complicated.

The text obtained in Tex or in MathML you may copy and past into TeX or HTML file and use for publication. You may also save it as an image and use in any document. It is useful, for example, when it is necessary to save a plot or a solution of a problem

2.1. Input data and run the calculations

At the center of the screen there is an entry field where it is possible to enter mathematical expressions. To start a task press the button ▶. When your cursor is disposed in the field of input you can press the combination of keys *Ctrl+Enter*.

Example is shown in figure 2.1.

On the top of the screen you can see buttons Help and Handbook. This is way to the help files. All the fields of the help pages are active, and you can run the help examples. You can copy text from the samples and transfer them into the field for user input.

When you enter mathematical expressions, they must be separated by a semicolon (;) or text comments, which are enclosed in quotation marks.

When you need to have a mathematical expression in the comment as part of the comment, it must be skirted in the dollar signs ($). For example, you can write a comment:
" Two different notations $*exp*(*x*)$ and $*e*^*x*$ are used for the exponential function."

To obtain results it is necessary to use the command **print**() and to specify the names of those expressions which are required to be printed.

If the list of statement does not contain a print statement **print**() or any other operator (**plot**(), **prints**(), etc.), it will be shown the result obtained in the last statement.

All commands or operators should begin with the symbol "back slash" (\).

The button + lets us add new entry fields. You can press the combination of keys *Crtl+Del* to remove this field or you can press the button x at the right side of this field on the screen.

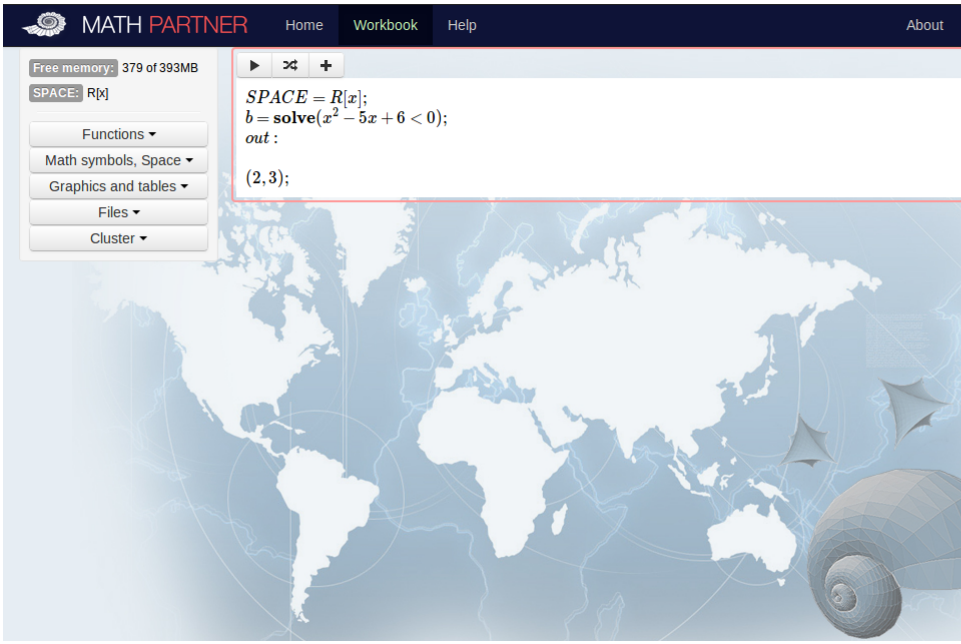


Figure 2.1: Entering data into a working page

The button C is designed to clean the values of all previously typed names. It is useful to have such button when the numerical values are entered in some sections, and the calculations are done in other sections. Clearing all names allows you to obtain a symbolic expression rather than the number.

On the left of the screen you can see fields with a current environment and a current random access memory. Under the fields differant buttons for enter of functions are placed.

2.1.1. Working with files

Functions for working with files are available at the "Files" collapsible panel from menu at the left.

Here is what you can do with files:

1) Save the result of the last run section as PDF file with "Save PDF" button. You can specify desirable paper size (dimensions are in centimeters), by default page has size A4 (21x29.7 cm)

2) Upload text files to Mathpar server with "Upload file" button. Under the button there is a list of uploaded files. Files should contain Mathpar expressions or tables in specific format.

Table contains of header – it's the first row with arbitrary strings in it – and number rows. Columns are separated with tabulation symbol. Functions for working with tables are available at the panel "Graphics and tables" (see also section 3.1 Plotting functions of help system).

3) Input Mathpar expressions from uploaded files with **fromFile**() function. E.g., to make an expression from file myfile.txt and assign this expression to variable a run: a = **fromFile**($'myfile.txt'$).

2.2. Mathematical functions

The following notations for elementary functions and constants are accepted.

2.2.1. Constants

\i — imaginary unit,
\e — the basis of natural logarithm,
\pi — the ratio of length of a circle to its diameter,
\infty — infinity sign.

2.2.2. Functions of one argument

\ln — natural logarithm,
\lg — decimal logarithm,
\sin — sine,
\cos — cosine,
\tg — tangent,
\ctg — cotangent,
\arcsin — arcsine,
\arccos — arccosine,
\arctg — arctangent,
\arcctg — arccotangent,
\sh — sine hyperbolic,
\ch — cosine hyperbolic,
\th — tangent hyperbolic,
\cth — cotangent hyperbolic,
\arcsh — arcsine hyperbolic,
\arcch — arccosine hyperbolic,
\arcth — arctangent hyperbolic,
\arccth — arccotangent hyperbolic,
\exp — exponent,
\sqrt — root square,
\abs — absolute value of real numbers (module for complex numbers),
\sign — number sign (returns 1, 0, -1 when number sign is $+$, 0, $-$, correspondingly),
\unitStep$(x, a)$ — is a function which, for $x > a$ takes the value 1, and for $x < a$ takes the value 0;
\fact — factorial. It is defined for positive integers. It is equivalent to $n!$.

2.2.3. Functions of two arguments

^ — degree,
\log — logarithm of function with given base,
\rootOf(x, n) — root of degree n of x,
\Gamma — the function Gamma,
\Gamma2 — the function Gamma 2,
\binomial — binomial coefficient.

Example.

```
SPACE = R64[x, y];
f1 = \sin(x);
f2 = \sin(\cos(x + \tg(y)));
f3 = \sin(x^2) + y;
\print(f1, f2, f3);
```

Returns:

$SPACE = R64[x, y]$;
$f1 = sin(x)$;
$f2 = sin(cos(x + tg(y)))$;
$f3 = sin(x^2) + y$.

2.3. Actions with functions

For the above functions and their compositions, you can calculate the value of the function at the point, substitute the expression into a function instead of arguments, calculate the limit of the function, calculate derivative, etc. For this purpose, the following commands are defined.

To calculate the value of a function at a point you must run $\backslash\mathbf{value}(f, [var1, var2, \ldots, varn])$, where f —function, and $var1, var2, \ldots, varn$ — values of the variables of the ring.

For the substitution of expressions to the function you must execute the $\backslash\mathbf{value}(f, [func1, func2, \ldots, funcn])$, where f — a function $func1, func2, \ldots, funcn$ — expressions that are substituted for the corresponding variables.

To calculate the limit of a function at a point you must run $\backslash\mathbf{lim}(f, var)$, where f — this function, and var — the point at which you want to find the limit.

In order to calculate the derivative of f in the variable y in the ring $\mathbb{Z}[x,\ y,\ z]$ you must run $\backslash\mathbf{D}(f, y)$. To find a mixed first-order derivative of the function f there is a command $\backslash\mathbf{D}(f, [x, y])$, to find the derivative of higher order you must use the command $\backslash\mathbf{D}(f, [x\hat{}k, z\hat{}m, y\hat{}n])$, where k, m, n indicate the order of the derivative of variables.

Examples.

```
SPACE = R[x, y];
f = \sin(x^2 + \tg(y^3 + x));
```

```
g = \value(f, [1, 2]);
\print(g);
```

Returns:

in: $SPACE = R[x, y]$;
$f = sin(x^2 + tg(y^3 + x))$;
$g = value(f,\ [1,\ 2])$;
$print(g)$;
out: $g = 0.52$;

```
SPACE = Z[x, y];
f = x + y;
g = f^2;
r = \value(f, [x^2, y^2]);
\print(g, f, r);
```

Returns:

in: $SPACE = Z[x, y]$;
$f = x + y$;
$g = f^2$;
$r = value(f, [x^2, y^2])$;
$print(g, f, r)$;
out: $g = y^2 + 2yx + x^2$;
$f = y + x$;
$r = x^2 + y^2$

```
SPACE = R64[x];
f = \sin(x) / x;
g = \lim(f, 0);
\print(g);
```

Returns:

in: $SPACE = R64[x]$;
$f = sin(x)/x$;
$g = lim(f, 0)$;
$print(g)$;
out: $g = 1.00$;

```
SPACE = Z[x, y];
f = \sin(x^2 + \tg(y^3 + x));
h = \D(f, y);
\print(h);
```

Returns:
in: $SPACE = Z[x, y]$;
$f = sin(x^2 + tg(y^3 + x))$;
$h = D(f, y)$;
$print(h)$;
out: $h = 3y^2 cos(x^2 + tg(y^3 + x))/(cos(y^3 + x))^2$;

2.4. Solution of the algebraic equation

To obtain a solution of the algebraic equation use the command **solve**.

The command $FLOATPOS = N$ is used for setting the environment. It sets the number of decimal places after the decimal point (N), which should appear in the print of the numerical results of approximate calculations. It is not connected with the process of calculation, but only with printing. By default, $FLOATPOS = 2$.

Examples.

```
SPACE = R64[x];
b = \solve(x^2 - 5x + 6 = 0);
```

Returns:
in: $SPACE = R64[x]$;
$b = solve(x^2 - 5x + 6 = 0)$;
out: $[2.00, 3.00]$;

```
SPACE = R64[x];
FLOATPOS = 6;
b = \solve(x^4 + 2x + 1 = 0);
```

Returns:
in: $SPACE = R64[x]$;
$FLOATPOS = 6$;
$b = solve(x^4 + 2x + 1 = 0)$;
out: $[-0.543689, -1.000000]$;

```
SPACE = R64[x];
FLOATPOS = 0;
b = \solve(x^3 + 3x^2 + 3x + 1 = 0);
```

Returns:
in: $SPACE = R64[x]$;
$FLOATPOS = 0$;
$b = solve(x^3 + 3x^2 + 3x + 1 = 0)$;
out: -1.

2.5. Solution of the algebraic inequalities

To obtain a solution of the algebraic inequalities use the command **solve**, which contains the inequalities. We can solve strict and not strict algebraic inequalities. Open interval is indicated in parentheses ( ), closed interval is indicated inbrackets [ ], set is denoted by braces { }.

Examples.

```
SPACE = R[x];
b = \solve(x^2-5x+6 < 0);
```

Returns:
in: $SPACE = R[x]$;
$b = solve(x^2 - 5x + 6 < 0)$;
out: $(2, 3)$.

```
SPACE = R[x];
b = \solve((x+1)^2(x-3)(x+5) \ge 0);
```

Returns:
in: $SPACE = R[x]$;
$b = solve((x + 1)^2(x - 3)(x + 5) \geq 0)$;
out: $(-\infty, -5] \cup \{-1\} \cup [3, \infty)$.

```
SPACE = R[x];
b = \solve((x^2+11x+28)/(x+5) \le 0);
```

Returns:
in: $SPACE = R[x]$;
$b = solve((x^2 + 11x + 28)/(x + 5) \leq 0)$;
out: $(-\infty, -7] \cup (-5, -4]$.

```
SPACE = Q[x];
b = \solve(x^2 + 4x - 7 = 0);
```

Returns:
in: $SPACE = Q[x]$;
$b = solve((x^2 + 4x - 7 = 0)$;
out: $[(\sqrt{11} + (2)), (2 - \sqrt{11})]$;.

2.6. Solution of the algebraic inequalities systems

To obtain a solution of the algebraic inequalities systems use the command **solve**$[In1, In2, ..., Ink]$, where $[In1, In2, ..., Ink]$ — vector, where contain inequalities. System contain strict and not strict algebraic inequalities. Open interval is indicated in parentheses (), closed interval is indicated inbrackets, set is denoted by braces { }.

Examples.

```
SPACE = R[x];
b = \solve([x^2+4x-5 > 0, x^2-2x-8 < 0]);
```

Returns:
in: $SPACE = R[x]$;
$b = solve([x^2 + 4x - 5 > 0, x^2 - 2x - 8 < 0])$;
out: $(1, 4)$.

```
SPACE = R[x];
b = \solve([x^2-x-6 \ge 0, x^2-4x-12 < 0]);
```

Returns:
in: $SPACE = R[x]$;
$b = solve([x^2 - x - 6 \geq 0, x^2 - 4x - 12 < 0])$;
out: $(-4, -2] \cup [3, 4)$.

```
SPACE = R[x];
b = \solve([x^2-4 < 0, x+1 > 0, 0.5-x > 0]);
```

Returns:
in: $SPACE = R[x]$;
$b = solve([x^2 - 4 < 0, x + 1 > 0, 0.5 - x > 0])$;
out: $(-1, 0.5)$.

2.7. Operations on subsets of the real numbers

To specify a subset use the command **set**$((a,b),(c,d])$, where a,b,c,d are numbers. Subset may consist of open intervals indicated by parentheses ( ), half-open intervals indicated by [ ) or ( ], segments indicated by brackets [ ] and points indicated by braces , or like segments.

Simple subset is denoted by the same brackets, but you need to add a backslash (\) in front of each bracket. For example \$(3,4.5)\$], \$[7,7\$] or \$8\$. The operator **set** is not required.

```
SPACE = R64[x];
a = \set((-2,1),[2,5),(5.75,6],[8,8]);
```

Returns:
in: $SPACE = R[x]$;
$a = set((-2,1),[2,5),(5.75,6],8)$;
out: $((-2.00),1.00) \cup [2.00,5.00) \cup (5.75,6.00] \cup \{8.00\}$.

With subsets we can make the following operations: union, intersection, subtraction, calculation of the symmetric difference and complement set, using the commands \$cup$, \$cap$, \$setminus$, \$triangle$ and symbol (') apostrophe.

```
SPACE = R64[x];
A=\(1,3\)\cup\[5,16\);
B=\(2,4\)\cup\[10,20\);
C=A\cup B;
D=A\cap B;
E=A\triangle B;
F=A \setminus B;
G=A';
\print(C,D,E,F,G);
```

Returns:
in: $SPACE = R64[x]$;
$A = (1,3) \cup [5,16)$;
$B = (2,4) \cup [10,20)$;
$C = A \cup B$;

$D = A \cap B$;
$E = A \triangle B$;
$F = A \setminus B$;
$G = A'$;
$print(C, D, E, F, G)$;
out: $C = (1, 4) \cup [5, 20)$
$D = (2, 3) \cup [10, 16)$
$E = (1, 2] \cup [3, 4) \cup [5, 10) \cup [16, 20)$
$F = (1, 2] \cup [5, 10)$
$G = (-\infty, 1] \cup [3, 5) \cup [16, \infty)$

2.8. Vectors and matrices

To define the row-vector you have to list its elements in square brackets.

To define the matrix you must take in square brackets a list of row vectors, for example, $A = [[1, 2], [3, 4]]$.

Element of the matrix may be obtained by specifying the row and column number in the two lower indexes of the matrix, and an element of the vector may be obtained by specifying its number in the lower index of the vector. The is an example for obtaining elements. You have to set $a = \backslash elementOf(A)$, and then obtain $a\_\{i, j\}$. If B is a vector, then you have to set $b = \backslash elementOf(B)$, and then obtain element $b\_\{i\}$.

You can get a row of the matrix as a vector-row and column of the matrix as a column vector. The row vector obtained by specifying the number of row in the first index and a sign of question (?) in the second index, for example, $a\_\{i, ?\}$. Column vector obtained by specifying the number of column in the second index and the sign of question (?) in the first index, for example, $a\_\{?, j\}$.

The names of non-commutative objects, such as matrices and vectors, must be written with the symbol ¡¡back slash¿¿ (\) and a capital letter.

To denote zero and identity matrix you can use the caps $\backslash O$ and $\backslash I$, with two indexes, indicating the number of rows and columns. With the help of the symbol $\backslash I$, you can create any size square matrix whose elements on the main diagonal are equal to 1, and the remaining elements are zero. For example, $\backslash I\_\{2, 3\}$ and $\backslash O\_\{2, 2\}$ denote the matrix $\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$ and $\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$. You can specify zero vectors, indicating the index number of elements: $\backslash O\_\{3\}$ denote the vector [0, 0, 0] and $I\_\{3\}$ denotes the vector [1, 0, 0].

Column vector can be formed by transposing the row vector, for example, $D = [7,\ 2,\ 3]^T$ — it is a column vector with three elements.

Arithmetic operations are indicated by standard signs " + ", " - ", " * ".

Example 1.

```
SPACE = Z[x];
A = [[x, 4], [y, 5]];
```

```
V = [x, y, 1, 2, x^6];
\print(A, V);
```

Returns:
in: $SPACE = Z[x]$;
$A = \begin{pmatrix} x & 4 \\ y & 5 \end{pmatrix}$;
$V = [x, y, 1, 2, x^6]$;
$print(A, V)$;
out: $A = \begin{pmatrix} x & 4 \\ y & 5 \end{pmatrix}$;
$V = [x, y, 1, 2, x^6]$;
Example 2.

```
SPACE = Z[x, y];
A = [[3, 4], [3, 1]];
B = [[2, 5], [4, 7]];
C = A + B;
G = A - B;
T = A * B;
\print(C, G, T);
```

Returns:
in: $SPACE = Z[x]$;
$A = \begin{pmatrix} 3 & 4 \\ 3 & 1 \end{pmatrix}$;
$B = \begin{pmatrix} 2 & 5 \\ 4 & 7 \end{pmatrix}$;
$C = A + B$;
$G = A - B$;
$T = A * B$;
$print(C, G, T)$;
out: $C = \begin{pmatrix} 5 & 9 \\ 7 & 8 \end{pmatrix}$;
$G = \begin{pmatrix} 1 & -1 \\ -1 & -6 \end{pmatrix}$;
$T = \begin{pmatrix} 22 & 43 \\ 10 & 22 \end{pmatrix}$;
Example 3.

```
SPACE = Z[x];
A = [[1, 4], [-4, 5]];
a = \elementOf(A);
det = a_{1, 1} * a_{2, 2} - a_{1, 2} * a_{2, 1};
\print(det);
```

Returns:

in: $SPACE = Z[x];$

$A = \begin{pmatrix} 1 & 4 \\ -4 & 5 \end{pmatrix};$

$det = a_{1,1} * a_{2,2} - a_{1,2} * a_{2,1};$

$print(det);$

out: $det = 21;$

Example 4.

```
SPACE = Z[x, y];
A = [[x^2, y], [4, x+y]];
a = \elementOf(A);
B = a_{1, ?};
C = a_{?, 2};
b = \elementOf(B);
c = \elementOf(C);
h = b_{2} * c_{1, 1};
\print(B, C, h);
```

Returns:

in: $SPACE = Z[x, y];$

$A = \begin{pmatrix} x^2 & y \\ 4 & x+y \end{pmatrix};$

$a = \backslash elementOf(A);$

$B = a_{1,?};$

$C = a_{?,2};$

$b = \backslash elementOf(B); c = \backslash elementOf(C);$

$h = b_2 \cdot c_1;$

$print(B, C, h);$

out: $B = \begin{pmatrix} x^2 \\ 4 \end{pmatrix};$

$C = \begin{pmatrix} 4 & y+x \end{pmatrix};$

$h = (y^2);$

Example 5.

```
SPACE = Z[x, y];
A = 3x * \I_{2, 2};
B = \O_{3, 3};
\print(A, B);
```

Returns:
in: $SPACE = Z[x, y]$;
$A = 3x * \mathbf{I}_{2,2}$;
$B = \mathbf{O}_{3,3}$;
$print(A, B)$;
out: $A = \begin{pmatrix} 3x & 0 \\ 0 & 3x \end{pmatrix}$;
$B = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$.

Example 6.

```
SPACE = R64[x];
A = [\pi / 2, \pi];
B = \sin(A);
C = \value(B);
\print(A, B, C);
```

Returns:
in: $SPACE = R64[x]$;
$A = [\pi/2, \pi]$;
$B = \sin(A)$;
$C = value(B)$;
$print(A, B, C)$;
out: $A = [\pi/2, \pi]$;
$B = \mathbf{s}in([\pi/2, \pi])$;
$C = [1, 0]$;

2.9. Generation of random elements

Mathpar can generate of random elements such as numbers, polynomials and matrices.

2.9.1. Generation of numbers

To create a random number you have to execute the command **randomNumber**(k), where k is the number of bits.

Example.

```
SPACE = Z[x, y, z];
a = \randomNumber(10);
b = \randomNumber(100);
\print(a, b);
```

Returns:
$SPACE = Z[x, y, z]$;
$a = 944$;
$b = 850800798881527094755736477974$.

2.9.2. Generation of random polynomial

To create a random polynomial with three variables you have to execute the command **randomPolynom**$(d1, d2, \ldots, ds, dens, bits)$, where $dens$ is a polynomial density, $bits$ is a number of bits in numerical coefficients, and $d1, d2, \ldots, ds$ denote the highest degrees of variable. If $dens = 100$, you get a polynomial that has all coefficients non-zero, all $(d1+1)(d2+1)\cdot(ds+1)$ non-zero terms. When $dens < 100$, only $dens\%$ coefficients are nonzero, and the remaining $(100 - dens)\%$ will be zero.

Example.

```
SPACE=Z[x,y,z];
f = \randomPolynom(4, 4, 10, 5);
g = \randomPolynom(4, 4, 10, 5);
h = f + g;
\print(f, g, h);
```

Returns:
$SPACE = Z[x, y, z]$;
$f = y^3x^3$;
$g = 10yx^3 + 2y$;
$h = y^3x^3 + 10yx^3 + 2y$;

2.9.3. Generation of random matrix

To create arandom numerical matrix you have to execute the command $\backslash$**randomMatrix**$(m, n, dens, bits)$, where last two arguments are the density of matrix and the number of bits in numerical elements of matrix, and first two arguments denote the sizes of a matrix.

To create a polynomial matrix you have to execute the command $\backslash$**randomMatrix**$(m, n, dens, d1, d2, \ldots, ds, pol\_dens, pol\_bits))$, where first three arguments denote the size of a matrix and its density, last two arguments are the density of polynomials and the number of bits in numerical coefficients, the numbers $d1, d2, \ldots, ds$ set the highest degrees of polynomial variables.

Example.

```
SPACE = Z[x, y, z];
matr_n = \randomMatrix(4, 4, 100, 5);
matr_p = \randomMatrix(2, 2, 100, 1, 1, 4, 25, 3);
\print(matr_n, matr_p);
```

Returns:

$SPACE = Z[x, y, z];$

$$matr_n = \begin{pmatrix} 22 & 2 & 10 & 28 \\ 23 & 28 & 1 & 19 \\ 30 & 24 & 19 & 12 \\ 27 & 22 & 22 & 17 \end{pmatrix};$$

$$matr_p = \begin{pmatrix} 6z^3x + 7z^3 + 5z^2 + 3y & 7z^4x + 2z^4 + 7zyx + 5x \\ z^4yx + 2zy + 7y + 7x + 4 & 7z^2x + 7zx + z + 6x \end{pmatrix}.$$

Chapter 3

Construction of 2D and 3D plots

3.1. Plotting functions

Mathpar table allows you to build graphics (*tablePlot*), graphs of functions, which are explicitly defined (*plot*) or parametric (*paramPlot*). You can build several different graphs in one coordinate system (*showPlots*).

Setting charting given command **set2D**(). If the command **set2D**() has no parameters, the boundaries for the graphs are calculated automatically, and for explicit functions selected interval $[0, 1]$ along the horizontal axis. The names of the coordinate axes will be X and Y, respectively. Title in the schedule will be absent.

If the command **set2D**() user not asked, it is automatically set **set2D**() with no arguments at the beginning of the session the user.

There are 7 basic options of this command with the following parameters: 1) **set2D**()
2) **set2D**$(x0, x1)$;
3) **set2D**$(x0, x1, '\, title')$;
4) **set2D**$(x0, x1, y0, y1)$;
5) **set2D**$(x0, x1, y0, y1, '\, title')$;
6) **set2D**$(x0, x1, '\, title', '\, nameOX', '\, nameOY')$;
7) **set2D**$(x0, x1, y0, y1, '\, title', '\, nameOX', '\, nameOY')$.

Numbers $x0$ and $x1$ ($x0 < x1$) sets the interval along the axis of OX. Numbers $y0$ and $y1$ ($x0 < x1$) sets the interval along the axis of OY. If these parameters are not specified, are calculated automatically. $nameOX$ — signature on the axis OX, $nameOY$ — signature on the axis OY, $title$ — header graphics.

In addition, permitted to ask one or two keys that should be the last in the list of options: BW and ES. BW refers to the construction of Cheraw and white graphics. ES indicates equality zoom scale x axis ranges from y. A total of $7 * 4 = 28$ different ways to set the parameters environment.

Character line which is depicted in the graph of each of the functions ($plot, tablePlot, paramPlot$) can be different: the solid line, dotted line, and the line that ends with an arrow. To do this, these options are: '$dash$' (dotted line), '$arrow$' (arrows) and a combination of '$dashAndArrow$', which should be at the end of the parameter list of these functions.

For example, $\backslash\mathbf{plot}(x^2 + 1,'dash')$.

If several separate graphs have names such as P=$\backslash\mathbf{plot}(x^2)$; Q=$\backslash\mathbf{tablePlot}([[1, 2], [3, 4]])$; in this case they may be represented along with the command $\backslash\mathbf{showPlots}([P, Q])$.

The resulting plot can be downloaded from the site. To do this, click on the button Download, which is located below the graph. The file is on schedule to be downloaded to your computer.

3.1.1. Plots of explicit functions

To obtain the plot of an explicit function $f = f(x)$ the command $\backslash\mathbf{plot}(f)$. Other options commands:
1) $\backslash\mathbf{plot}(f, [x0, x1])$, where $[x0, x1]$ — interval along the axis of OX;
2) $\backslash\mathbf{plot}(f, [x0, x1],'options')$, where $[x0, x1]$ — interval along the axis of OX, 'options' — takes the following values:
1)'dash' — schedule will be a dashed line;
2)'arrow' — the last point on the graph is drawn with an arrow;
3)'dashAndArrow' — schedule will be a dashed line and the last point of the graph is drawn with an arrow.
3) $\backslash\mathbf{plot}(f,'options')$. You can plot functions with parametric variables. The parametric variables are assigned when you set a environment (see ex.3).

Example 1.

```
SPACE = R64[x, y, z];
\set2D(-10, 10, -10, 10);
f = x^2 + \tg(x^2 - 1);
p = \plot(f);
```

Returns:
in: $f = x^2 + \mathbf{tg}(x^2 - 1)$;
out: fig. 3.1.

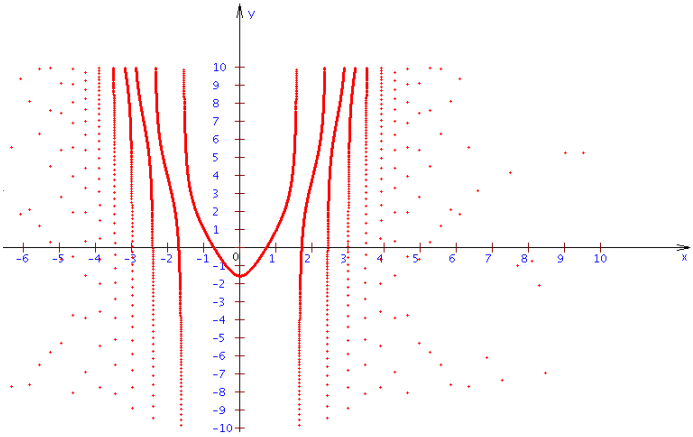


Figure 3.1: $f = x^2 + \mathbf{tg}(x^2 - 1)$

To get the graphs of several functions in one figure you must enclose the list of these functions in square brackets, as in the following example.

Example 2.

```
SPACE = R64[x, y, z];
\set2D(-10, 10, -10, 10);
f = \sin(x);
p = \plot([f, \tg(x)]);
```

Returns:
in: $f = sin(x)$;
out: fig. 3.2.

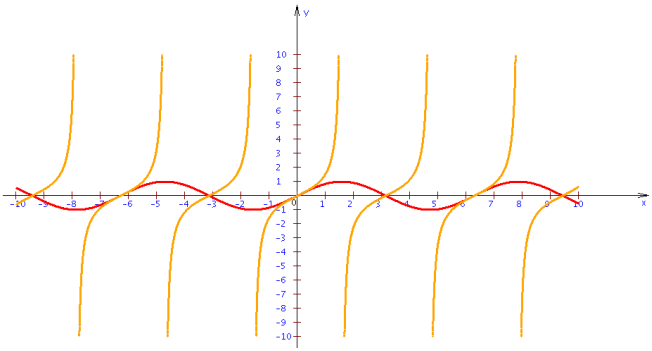


Figure 3.2: $f = \sin(x)$ and $g = \mathbf{tg}(x)$

Example 3.

```
SPACE = R64[x, y, z];
\set2D(-10, 10, 0, 2);
f = \unitBox(x,3);
p = \plot(f);
```

Example 4.

```
SPACE = R64[x, a, b, c];
\set2D(0, 2\pi, 0, 2);
\plot(a\sin(bx) + c);
```

Example 5.

```
SPACE = R64[x, y, z];
\set2D(-10, 10, -10, 10,'a','b','title');
f = x^2;
p = \plot(f);
```

Example 6.

```
SPACE = R64[x, y, z];
\set2D(-10, 10, -10, 10);
f = x;
p = \plot(f,'dash');
```

Example 7.

```
SPACE = R64[x, y, z];
f = x;
p = \plot(f,[-5,5],'arrow');
```

Example 8.

```
SPACE = R64[x, y, z];
\set2D(-10, 10, -10, 10);
\plot([x,-x],'arrow');
```

3.1.2. Plots of parametric functions

To obtain the plot of parametric function $\{f = x(t),\ g = y(t)\}$ the command $\backslash$**paramPlot**$([f, g], [t0, t1])$ is used, where $[t0, t1]$ is an interval of variation of t. Another version of the command: $\backslash$**paramPlot**$([f, g], [t0, t1], 'options')$, where $[t0, t1]$ — the range of values for the parameter change, 'options' — the following values:
1)'dash' — schedule will be a dashed line;
2)'arrow' — the last point on the graph is drawn with an arrow;
3)'dashAndArrow' — schedule will be a dashed line and the last point of the graph is drawn with an arrow.

Example 1.

```
SPACE = R64[x, y, z];
g = \sin(x);
k = \cos(x);
f = \paramPlot([g, k], [0, 2\pi]);
```

Returns:
in: $g = sin(x); k = cos(x)$;
out: fig. 3.3.

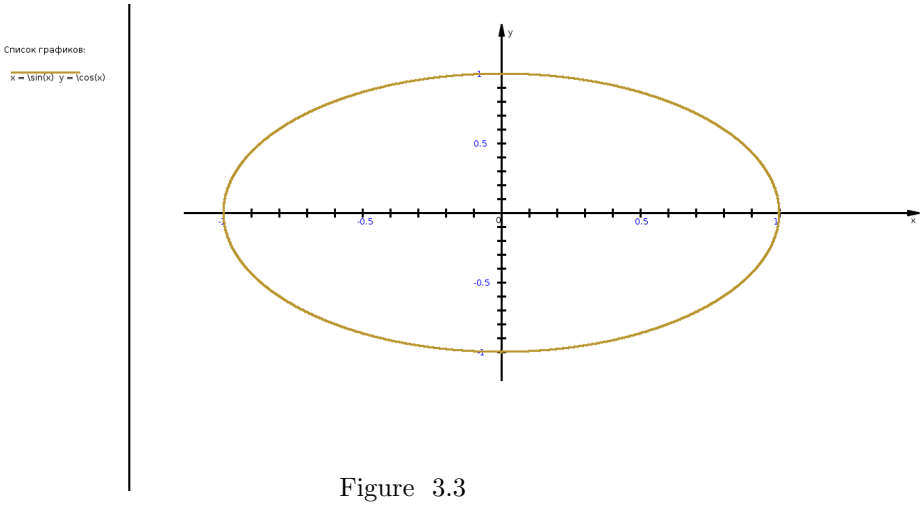
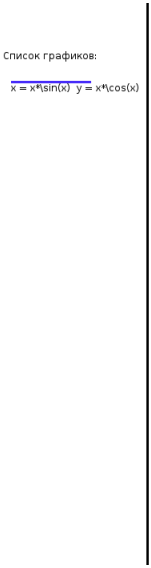


Figure 3.3

Example 2.

```
SPACE = R64[x, y, z];
g = x\sin(x);
k = x\cos(x);
f = \paramPlot([g, k], [0, 5\pi]);
```

Returns:
in: $g = x\sin(x); k = x\cos(x);$
out: fig. 3.4.



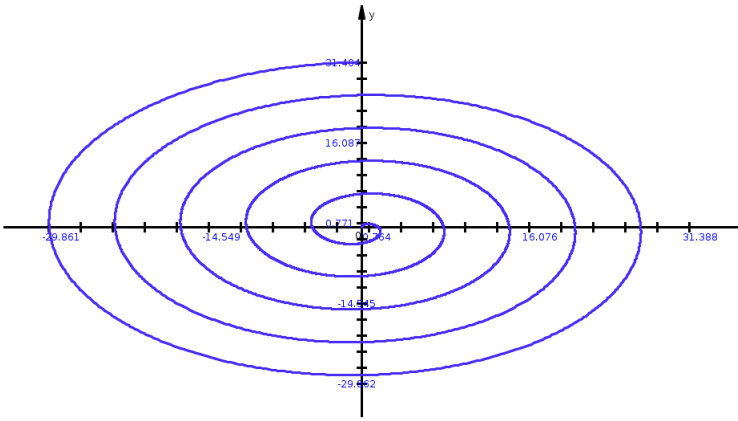


Figure 3.4

Example 3.

```
SPACE = R64[x, y, z];
g = 2\cos(x)+\cos(2x);
k = 2\sin(x)-\sin(2x);
f = \paramPlot([g, k], [0, 2\pi]);
```

Returns:
in: $g = 2\cos(x) + \cos(2x); k = 2\sin(x) - \sin(2x);$
out: fig. 3.5.

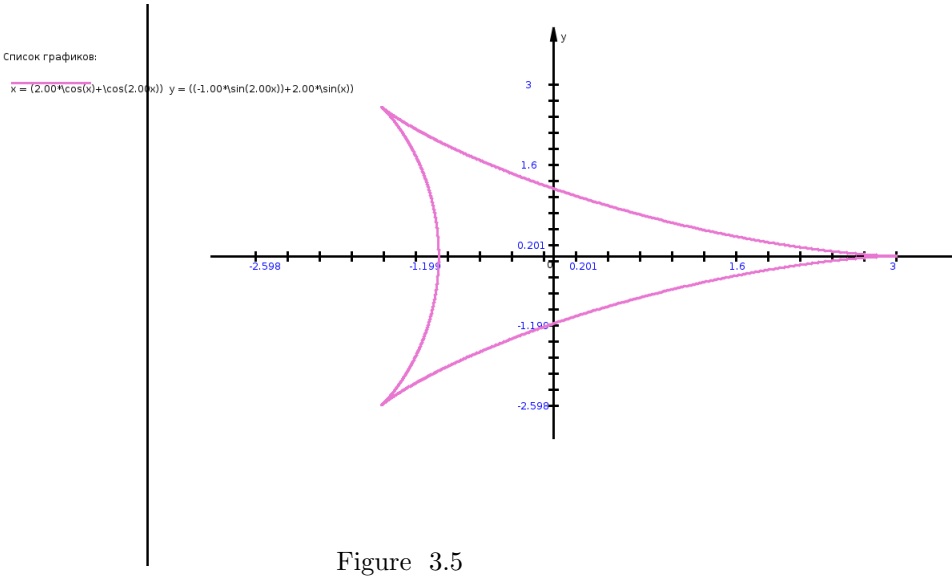


Figure 3.5

Example 4.

```
SPACE = R64[x, y, z];
g = 2\sin(x)^3;
k = 2\cos(x)^3;
f = \paramPlot([g, k], [0, 2\pi]);
```

Returns:
in: $g = 2\sin(x)^3; k = 2\cos^3(x)$
out: fig. 3.6.

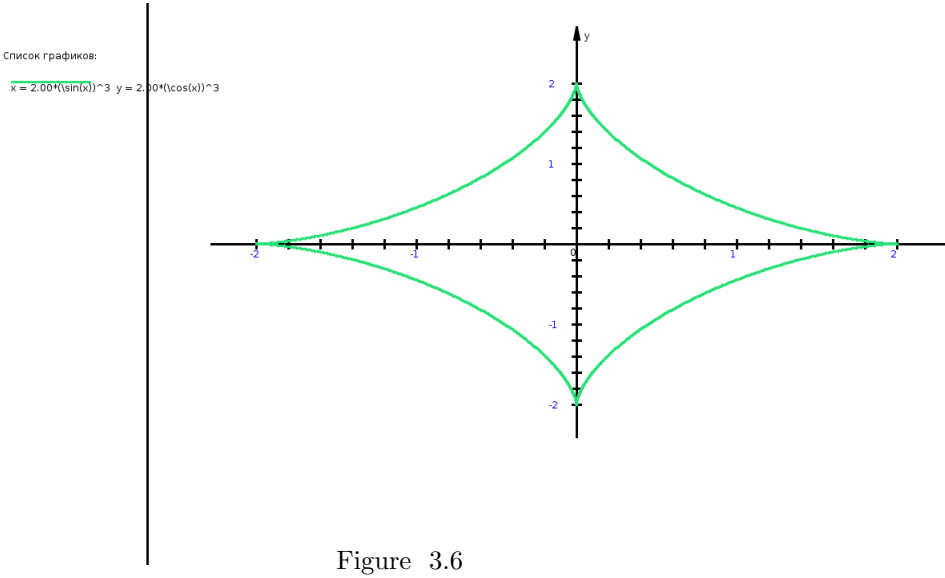


Figure 3.6

Example 5.

```
SPACE = R64[x, y, z];
g = (1+\cos(x))\cos(x);
k = (1+\cos(x))\sin(x);
f = \paramPlot([g, k], [0, 2\pi]);
```

Returns:
in: $g = (1 + \cos(x))\cos(x); k = (1 + \cos(x))\sin(x);$
out: fig. 3.7.

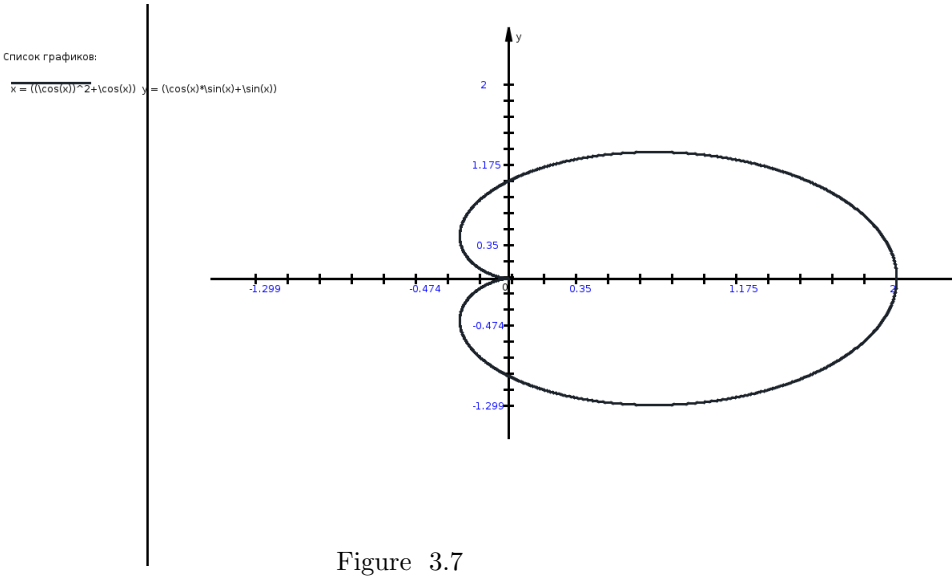


Figure 3.7

Example 6.

```
SPACE = R64[x, y, z];
g = \sin(x)(\exp(\cos(x))-2\cos(4x)+\sin(x/12)^5);
k = \cos(x)(\exp(\cos(x))-2\cos(4x)+\sin(x/12)^5);
f = \paramPlot([g, k], [0, 12\pi]);
```

Returns:
in: $g = \sin(x)(\exp(\cos(x)) - 2\cos(4x) + \sin^5(x/12))$;
$k = \cos(x)(\exp(\cos(x)) - 2\cos(4x) + \sin^5(x/12))$;
out: fig. 3.8.

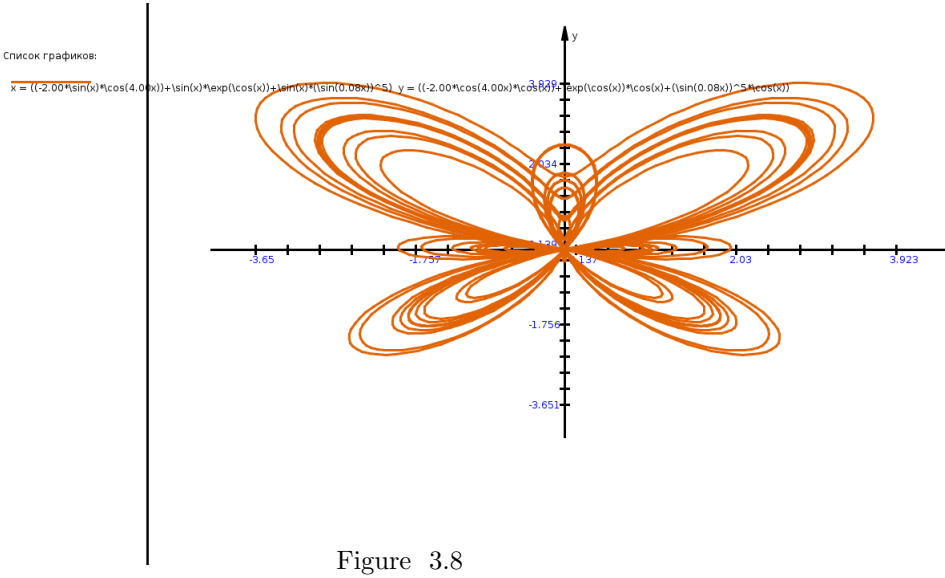


Figure 3.8

Example 7.

```
SPACE = R64[x, y, z];
\set2D('','','','','x(t)','y(t)','paramPlot');
g = \sin(x)(\exp(\cos(x))-2\cos(4x)+\sin(x/12)^5);
k = \cos(x)(\exp(\cos(x))-2\cos(4x)+\sin(x/12)^5);
f = \paramPlot([g, k], [0, 12\pi]);
```

Example 8.

```
SPACE = R64[x, y, z];
g = \sin(x);
k = \cos(x);
f = \paramPlot([g, k], [0, 2\pi],'dashAndArrow');
```

3.1.3. Plot of table function

To plot a function, which is defined by the table of points you have to execute the command: $\backslash$**tablePlot**$([[x_1,\ldots,x_n],[y_{11},\ldots,a_{1n}],\ldots,[y_{k1},\ldots,a_{kn}]])$.
Another version of the command: $\backslash$**tablePlot**$([[x_1,\ldots,x_n],[y_{11},\ldots,a_{1n}],\ldots,[y_{k1},.$

,where 'options' — the following values:
1)'dash' — schedule will be a dashed line;
2)'arrow' — the last point on the graph is drawn with an arrow;
3)'dashAndArrow' — schedule will be a dashed line and the last point of the graph is drawn with an arrow.

Example 1.

```
SPACE = R64[x, y, z];
 \tablePlot(
   [
     [0, 1, 2, 3, 4, 5],
     [0, 1, 4, 9, 16, 25],
     [0, -1, -2, -3, -4, -5],
     [0, 4, 8, 12, 16, 20]
   ]);
```

Returns:
in:
out: fig. 3.9.

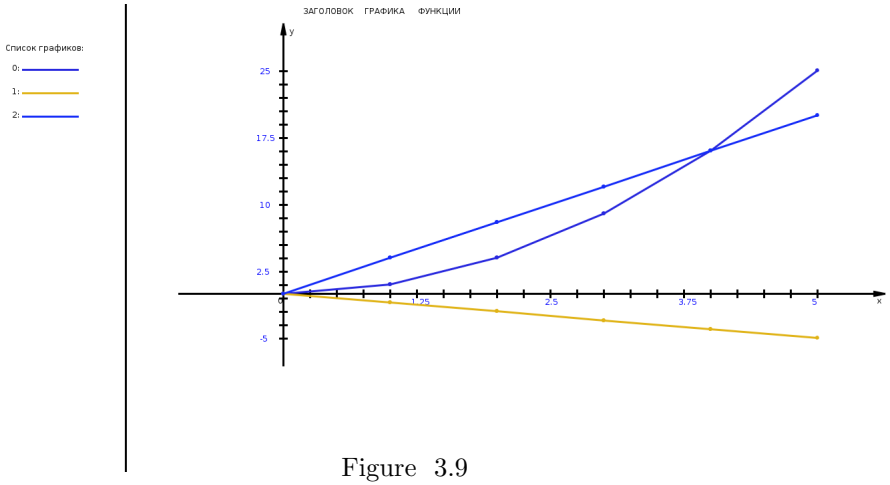


Figure 3.9

Example 2.

```
SPACE=R64[x];
```

```
\set2D(-1,5,-10,10);
"We have a table function"
A = [[0, 1, 2, 3, 4, 5], [3, 0, 4, 10, 5, 10]];
t = \table (A);
"We approximate this function by a polynomial of degree 4:"
p = \approximation(t, 4);
"Building the graph of the polynomial:" P = \plot (p, [1,5]);
"Plot a table function:" T = \tablePlot (t);
"We construct both graphs in one coordinate system:"
\showPlots ([P, T]);
```

Returns:
in:
out: $0.54x^4 - 5.64x^3 + 18.38x^2 - 17.28x + 3.17$
fig. 3.10.

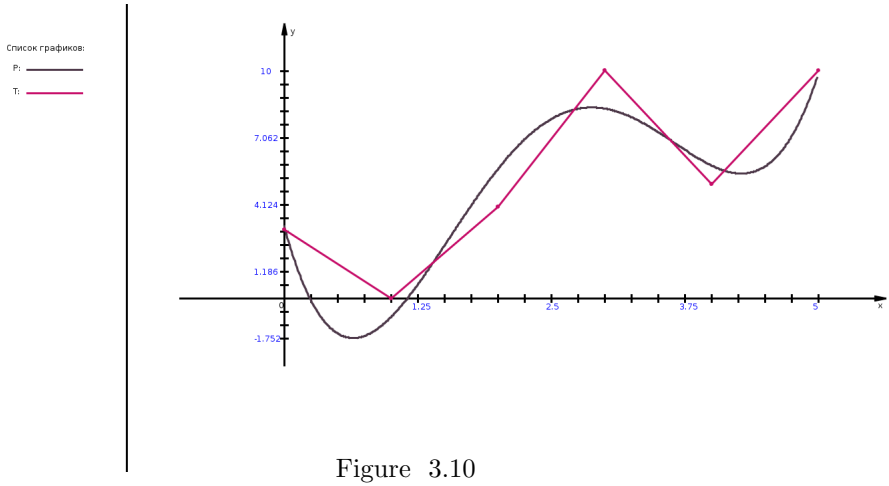


Figure 3.10

Example 3.

```
SPACE = R64[x, y, z];
\set2D(-10, 10, -10, 10, '','', 'Header of Graphics');
 \tablePlot(
   [[-3, -6, -6, -3, 3, 6, 6, 3, -3],
```

```
    [6, 3, -3, -6, -6, -3, 3, 6, 6]]);
```

Example 4.

```
SPACE = R64[x, y, z];
 \tablePlot(
   [[-3, -6, -6, -3, 3, 6, 6, 3, -3],
    [6, 3, -3, -6, -6, -3, 3, 6, 6]],'arrow');
```

Example 5.

```
SPACE = R64[x, y, z];
 \tablePlot(
   [[-3, -6, -6, -3, 3, 6, 6, 3, -3],
    [6, 3, -3, -6, -6, -3, 3, 6, 6]], 'dash');
```

Example 6.

```
SPACE = R64[x, y, z];
 \tablePlot(
   [[-3, -6, -6, -3, 3, 6, 6, 3, -3],
    [6, 3, -3, -6, -6, -3, 3, 6, 6]], 'dashAndArrow');
```

3.1.4. Functions that are defined on the points table of values

Plotting functions on the points given by the tabulated values use the command:

pointsPlot($[[x_1,\ldots,x_n],[y_1,\ldots,y_n]],[s_1,\ldots,s_n],[kv_1,\ldots,kv_n],[kg_1,\ldots,kg_n]$) where s_n — signature points, kv_n — rate of rotation about the point (ranges from 0 to 7, and signifies a shift in the (kv_n * 45) degrees), kg_n — shift factor along the axis OX (if it is negative then the displacement is to the left).

The reduced variants of this command:

pointsPlot($[[x_1,\ldots,x_n],[y_1,\ldots,y_n]],[s_1,\ldots,s_n]$)

or **pointsPlot**($[[x_1,\ldots,x_n],[y_1,\ldots,y_n]],[s_1,\ldots,s_n],[kv_1,\ldots,kv_n]$)

or **pointsPlot**($[[x_1,\ldots,x_n],[y_1,\ldots,y_n]],[s_1,\ldots,s_n],[kv_1,\ldots,kv_n],[kg_1,\ldots,kg$

Example 1.

```
\set2D(-10, 10, -10, 10);
\pointsPlot( [[0, 1, 2], [0, 1, 4]],['a','b','c']);
```

Example 2.

```
\pointsPlot([[0, 1, 2], [0, 1, 4]],['a','b','c'],[0,2,4]);
```

Example 3.

```
\pointsPlot( [[0, 1, 2], [0, 1, 4]],['a','b','c'],[0,2,4],[0,-5,5]);
```

Example 4.

```
\pointsPlot( [[0, 1, 2], [0, 1, 4]]);
```

Example 5.

```
SPACE = R64[x, y, z];
f1=\tablePlot([[1, 1], [1, 5]]);
f2=\tablePlot([[1, 5], [1, 1]]);
f3=\tablePlot([[5, 5], [1, 5]]);
f4=\tablePlot([[1, 5], [5, 5]]);
f5=\pointsPlot([[1, 1, 5, 5],[1, 5, 5, 1]],['A','B','C','D'],[6,0,0,2
\showPlots([f1, f2, f3, f4, f5]);
```

Example 6.

```
SPACE = R64[x, y, z];
f1=\tablePlot([[1, 1], [1, 5]]);
f2=\tablePlot([[1, 5], [1, 1]]);
f3=\tablePlot([[5, 5], [1, 5]]);
f4=\tablePlot([[1, 5], [5, 5]]);
f5=\pointsPlot([[1, 1, 5, 5],[1, 5, 5, 1]],['A','B','C','D'],[6,0,0,2
\showPlots([f1, f2, f3, f4, f5], 'noAxes');
```

3.1.5. Construction of various plots of functions in one coordinate system

To construct the plots of functions defined in different ways, you must first build a plot of each function and then execute the command **showPlots**($[f_1, f_2, \ldots, f_n]$).

Another version of the command:

showPlots([$f1, f2, f3, f4$],'$noAxes$'), where 'noAxes' — parameter indicating image graphics without axes, or

showPlots([$f1, f2, f3, f4$],'$lattice$'), where 'lattice' — parameter indicating the image graph with the lattice.

Example 1.

```
SPACE = R64[x, y, z];
\set2D(-20, 20, -20, 20);
f1 = \plot(\tg(x));
f2 = \tablePlot([[0, 1, 4, 9, 16, 25], [0, 1, 2, 3, 4, 5]]);
f3 = \paramPlot([\sin(x), \cos(x)], [-10, 10]);
f4=\tablePlot([[0, 1, 4, 9, 16, 25], [0, -1, -2, -3, -4, -5]]);
\showPlots([f1, f2, f3, f4]);
```

Returns:
in:
out: fig. 3.11.

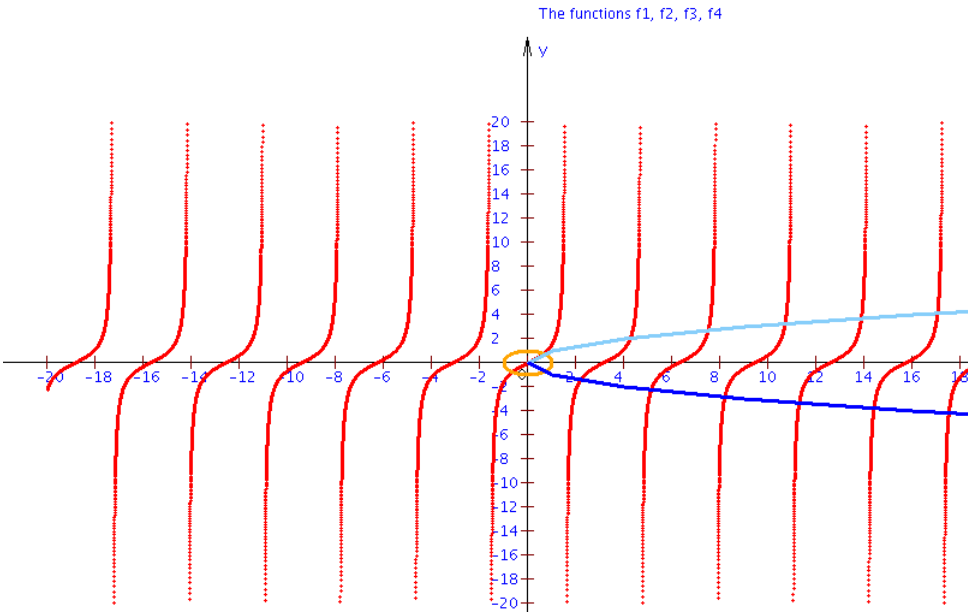


Figure 3.11: Graphs of functions defined in different ways

Example 2.

```
p1=\tablePlot([[-1, -3, 3, 3, -3, -3],[4, 3, 3, -3 , -3, 3]]);
p2=\tablePlot([[5, 5, 3, 3, 5, -1],[4, -2, -3, 3, 4, 4]]);
p3=\tablePlot([[-3, -1, -1],[-3, -2, 4]], 'dash');
p4=\tablePlot([[-1, 5],[-2, -2]], 'dash');
\showPlots([p1,p2,p3,p4], 'noAxes');
```

Example 3.

```
p1=\tablePlot([[-1, -3, 3, 3, -3, -3],[4, 3, 3, -3, -3, 3]]);
p1p=\pointsPlot([[-1, -3, 3, 3, -3],[4, 3, 3, -3 , -3]],
 ['F','B','C','D','A'],[0,0,0,4,4]);
p2=\tablePlot([[5, 5, 3, 3, 5, -1],[4, -2, -3, 3, 4, 4]]);
p3=\tablePlot([[-3, -1, -1],[-3, -2, 4]],'dash');
p4=\tablePlot([[-1, 5],[-2, -2]],'dash');
p2p=\pointsPlot([[5, 5, -1 ],[4, -2, -2]],['G','H','E'],[0,4,4]);
\showPlots([p1,p2,p3,p4,p1p,p2p], 'noAxes');
```

3.1.6. Construction of graphs

To construct the graph, use the command

plotGraph($[[a_{11},\dots,a_{1n}],\dots,[a_{n1},\dots,a_{nn}]],[[x_1,\dots,x_n],[y_1,\dots,y_n]]$),

where $[[a_{11},\dots,a_{1n}],\dots,[a_{n1},\dots,a_{nn}]]$ — adjacency matrix,

$[[x_1,\dots,x_n],[y_1,\dots,y_n]]$ — matrix of coordinates.

Example 1.

```
SPACE = R64[x, y, z];
\plotGraph([[0,1,1,0,1,0],[1,0,0,1,1,0],[1,0,0,0,1,1],[0,1,0,0,0,0],
[1,1,1,0,0,1],[0,0,1,0,1,0]],[[3,2,4,1,3,5],[3,2,2,1,1,1]]);
```

Returns:
in:
out: pict. 3.12.

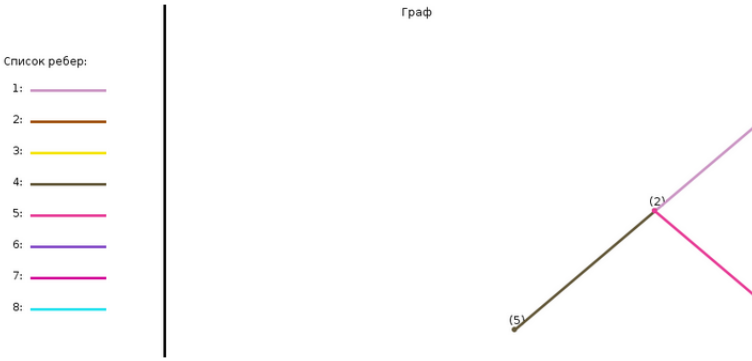


Figure 3.12: graph

In addition, you can run only with the first parameter
plotGraph($[[a_{11},\ldots,a_{1n}],\ldots,[a_{n1},\ldots,a_{nn}]]$),
where $[[a_{11},\ldots,a_{1n}],\ldots,[a_{n1},\ldots,a_{nn}]]$ — adjacency matrix.

Example 2.

```
SPACE = R64[x, y, z];
\plotGraph([[0,1,1,0,1,0],[1,0,0,1,1,0],[1,0,0,0,1,1],[0,1,0,0,0,0],
[1,1,1,0,0,1],[0,0,1,0,1,0]]);
```

Returns:
in:
out: pict. 3.13.

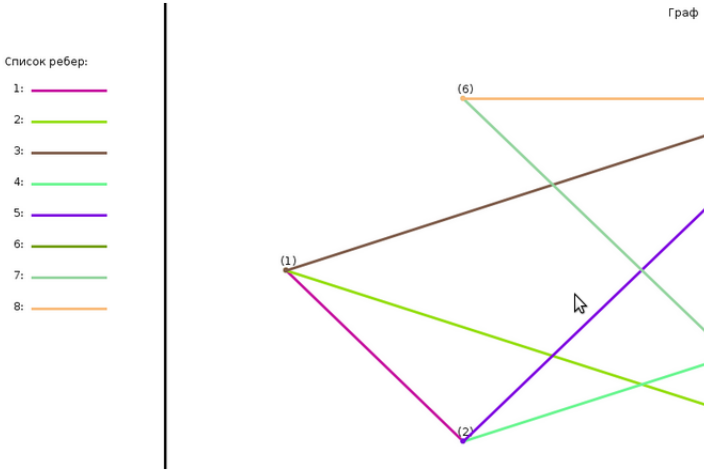


Figure 3.13: graph

You need to run a single numeric parameter **plotGraph**(N), where N — the number of vertices in a graph.

Example 3.

```
SPACE = R64[x, y, z];
\plotGraph(6);
```

Returns:
in:
out: pict. 3.14.

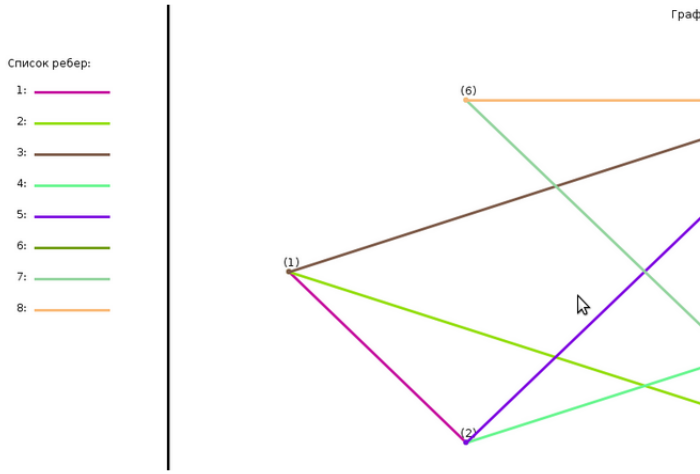


Figure 3.14: graph

3.2. 3D function graphs plotting

The environment for plotting 3D graphs is set with the command **set3D**()

There are several variants for this command:
1) **set3D**$(x0, x1, y0, y1, z0, z1)$;
2) **set3D**$(x0, x1, y0, y1, z0, z1, gridSize)$;
3) **set3D**$(x0, x1, y0, y1, z0, z1, gridSize, framesNumber)$;
4) **set3D**$(x0, x1, y0, y1, z0, z1, gridSize, framesNumber, [a1, a2, ...])$;

The numbers $x0$ and $x1$ ($x0 < x1$) define the interval on the OX axis. The numbers $y0$ and $y1$ ($y0 < y1$) define an interval on the OY axis. The numbers $z0$ and $z1$ ($z0 < z1$) specify the interval on the OZ axis. $gridSize$ is responsible for the grid size of the box in space in which the graph is drawn. $framesNumber$ is responsible for the number of frames when plotting a graph with parameters, whose change can be observed as a change of frames. $a1$ and $a2$ are responsible for the final value of the function parameters. These values will be set in the sliders that appear under the text with the user's request. When parameters are changed during frame changes, their values will change in the range from 1.0 to $a1$ for the first parameter.

3.2.1. Explicit 3D function graphs plotting. Server-side building

You can build 3D graphs of the functions that are defined explicitly.

To obtain the plot 3D of an explicit function $f = f(x, y)$ the command **plot3d**$(f, [x0, x1, y0, y1])$, is used, where $[x0, x1]$ is an interval on the axis OX, $[y0, y1]$ is an interval on the axis OY.

The obtained plot can be rotated and to increase or decrease.

Moving the mouse holding down the left ¡¡mouse¿¿ button causes the rotation of the coordinate system of schedule. After stopping the movement of the ¡¡mouse¿¿ graphics are redrawn in the new rotated coordinate system.

Moving the mouse holding down the left mouse button while pressing *Shift* button leads to a change in image scale. After stopping the movement of the ¡¡mouse¿¿ graphics are redrawn in the new scale.

Example.

```
SPACE = R64[x, y, z];
f = x^2 / 20 + y^2 / 20;
\plot3d(f, [-20, 20, -20, 20]);

SPACE = R64[x, y, z];
\plot3d([x / 20 + y^2 /20, x^2 /20 + y /20], [-20, 20, -20, 20]);

SPACE = R64[x, y, a, b];
f = ax^2 / 20 + by^2 / 20;
\plot3d(f, [-20, 20, -20, 20]);
```

3.2.2. Explicit 3D function graphs plotting. Client-side building

Also, graphs of functions that are explicitly defined can be plotted with the command **explicitPlot3d**$(f, xMin, xMax, yMin, yMax, zMin, zMax)$, where the numbers $xMin, xMax, yMin, yMax, zMin, zMax$ define a region in space having the shape of a box, in which the explicit function is represented.

In addition, the following set of arguments are allowed: $(f, xMin, xMax, yMin, yMax, zMin, zMax, gridSize)$, where $gridSize$ is responsible for the size of the grid of the box in which the graph is plotted.

You can specify only one function, as follows: **explicitPlot3d**(f), in this case, it is supposed to represent the function f in a cube $20 \times 20 \times 20$, whose center is at the origin.

Using **explicitPlot3d**() you can rotate the coordinate system by moving the mouse pointer with the left button pressed. You can also shift the origin of the coordinate system by moving the mouse pointer with the right button pressed.

To plot the graph of a function in time with changing parameters you should first specify the number of frames. Then, you should set the final value of parameters using the sliders. To build the graph, you should press the 'Plot' button. The value of the number of frames and parameters on the sliders can be specified via **set3D**().

Example.

```
SPACE = R64[x, y];
```

```
f = (x^2+y^2)/20;
ePl=\explicitPlot3d(f, -10, 10, -10, 10, -10, 10, 40);

SPACE = R64[x, y, a];
\set3D(-10, 10, -10, 10, -10, 10, 40, 25, [0.2]);
f = (ax^2+y^2)/20;
ePl=\explicitPlot3d(f);
```

3.2.3. Plotting 3D graphs of functions that are parametrically defined. Server-side building

Mathpar allows you to build 3D graphs of functions that are specified parametrically.

To plot a graph, you need to pass 3 functions $f(x,y)$, $g(x,y)$, and $h(x,y)$ using the command $\textbf{\textbackslash paramPlot3d}([[[f],[g],[h]],[x0,x1,y0,y1])$, where $[x0,x1]$ — interval on the OX-axis, $[y0,y1]$ — interval on the OY-axis.

Sphere

```
SPACE=R64[u,v];
\paramPlot3d([[\cos(u)\cos(v)],[\sin(u)\cos(v)],
[\sin(v)]], [-\pi, \pi, -\pi/2, \pi/2]);
```

Thor

```
SPACE=R64[u,v];
\paramPlot3d([[\cos(u)(\cos(v)+3)],[\sin(u)(\cos(v)+3)],
[\sin(v)]], [-\pi, \pi, -\pi, \pi]);
```

Spiral

```
SPACE=R64[u,v];
\paramPlot3d([[\cos(u)(\cos(v)+3)],[\sin(u)(\cos(v)+3)],
[\sin(v)+u]], [-2\pi, 2\pi, -\pi, \pi]);
```

Logarithmic spiral

```
SPACE=R64[u,v];
\paramPlot3d([[u\cos(u)(\cos(v)+1)],[u\sin(u)(\cos(v)+1)],
[u\sin(v)]], [0, 3\pi, -\pi, \pi]);
```

"Seashell"

```
SPACE=R64[u,v];
\paramPlot3d([[u\cos(u)(\cos(v)+1)],[u\sin(u)(\cos(v)+1)],
[u\sin(v)-(((u+3)/8)\pi)^2-20]], [0, 8\pi, -\pi, \pi]);
```

Shamrock

```
SPACE=R64[u,v];
\paramPlot3d([[\cos(u)\cos(v)+3\cos(u)(1.5+\sin(1.5u/2))],
[\sin(u)\cos(v)+3\sin(u)(1.5+\sin(1.5u/2))],
[\sin(v)+2\cos(1.5u)]], [-2\pi, 2\pi, -\pi, \pi]);
```

Dini surface

```
SPACE=R64[u,v];
\paramPlot3d([[\cos(u)\sin(v)],[\sin(u)\sin(v)],
[\cos(v)+\lg(\tg(v/2)) +0.2u-4]], [0, 4\pi, 0.0001, 2]);
```

Tape Mobius

```
SPACE=R64[u,v];
\paramPlot3d([[(1+v/2\cos(u/2))\cos(u)],[(1+v/2\cos(u/2))\sin(u)],
 [v/2\sin(u/2)]], [0, 2\pi, -1, 1]);
```

Cube

```
SPACE=R64[u,v];
\paramPlot3d([[u,v,5,u,v,-5],[v,5,u,v,-5,u],[5,u,v,-5,u,v]],
 [-5, 5, -5, 5]);
```

Cylinder

```
SPACE=R64[u,v];
\paramPlot3d([[5\cos(u)],[5\sin(u)],[v]], [-5, 5, -5, 5]);
```

Cone

```
SPACE=R64[u,v];
\paramPlot3d([[\cos(u) * (5 * (1 - v/6))],
 [\sin(u) * (5 * (1 - v/6))],[v]], [-6, 6, 0, 6]);
```

Truncated cone

```
SPACE=R64[u,v];
\paramPlot3d([[\cos(u) * (5 * (1 - v/6) + 1 * v/6)],
[\sin(u) * (5 * (1 - v/6) + 1 * v/6) ],[v]], [-5, 5, -5, 5]);
```

Hourglass

```
SPACE=R64[u,v];
\paramPlot3d([[\cos(u) * (5 * (0.5 - v/6) + 0.01*v/6)],
[\sin(u) * (5 * (0.5 - v/6) + 0.01*v/6)],[v]], [0, 2\pi, 0, 2\pi]);
```

Returns:
in: $f = x^2/20 + y^2/20$;
$plot3d(f, [-20, 20, -20, 20])$;
$plot3d([x/20 + y^2/20, x^2/20 + y/20], [-20, 20, -20, 20])$;
out: fig. 3.15.

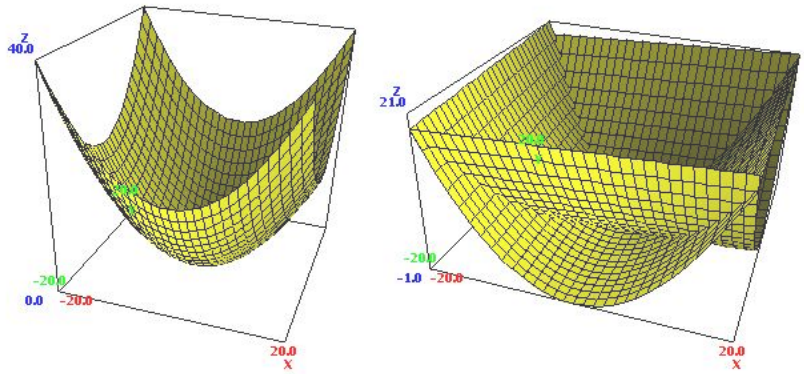


Figure 3.15: Plots 3D of functions

3.2.4. Plotting 3D graphs of functions that are parametrically defined. Client-side building

Also, graphs of functions that are defined parametrically can be plotted with the command **\parametricPlot3d**$(f, g, h, uMin, uMax, vMin, vMax, gridSize)$,

where the 3 functions $f(u,v)$, $g(u,v)$ and $h(u,v)$ for the axes OX, OY, OZ are specified. The numbers $uMin, uMax, vMin, vMax$ specify the range for the parameters of functions f, g and h. $gridSize$ is responsible for the grid size of the box in which the graph is plotted.

You can rotate the coordinate system by moving the mouse pointer with the left button pressed. You can also shift the origin of the coordinate system by moving the mouse pointer with the right button pressed.

To plot the graph of a function in time with changing parameters, you should first specify the number of frames. Then, you should set the final value of parameters using the sliders. To build the graph, you should press the 'Plot' button. The value of the number of frames and parameters on the sliders can be specified via **set3D**().

Parametric thor

```
SPACE = R64[u, v, a];
X=\cos(u)*(3+\cos(v));
Y=\sin(u)*(3+\cos(v));
Z=a*\sin(v);
\parametricPlot3d(X, Y, Z, 0, 7, 0, 7, 64);
```

Parametric spiral

```
SPACE = R64[u, v, b];
X=\cos(u)*(\cos(v)+2);
Y=\sin(u)*(\cos(v)+4b);
Z=\sin(v)+u/2+1;
\parametricPlot3d(X,Y,Z, -6.3, 6.3, -3.15, 3.15, 64);
```

3.2.5. Ploting of 3D graphs of functions that are defined implicitly

You can build 3D graphs of the functions that are defined implicitly.

To construct the graph of an implicit function $f(x,y,z) = 0$ use the command

implicitPlot3d$(f, x0, x1, y0, y1, z0, z1)$,

where the numbers $xMin, xMax, yMin, yMax, zMin, zMax$ set the box in the space, which is represented by an implicit function.

You can specify only one function, like this

implicitPlot3d(f)

in this case it is assumed that there will be shows the function f in the cube $20 \times 20 \times 20$, which is placed on a center origin.

You can rotate the coordinate system by moving the mouse pointer while holding down the left button. You can move the coordinate system by moving the mouse pointer while holding down the right button.

It is possible, optionally, to specify the coordinates of the light source, the color of the surface and the grid size. The default grid of 50 points on each edge of the box.

The color format RGB (red, green, blue) is given a number

$R * 256 * 256 + G * 256 + B$,

where each letter denotes a non-negative integer not exceeding 255. For example, $255 * 256 * 256$ — red, and $255 * 256 * 256 + 255 * 256$ — yellow (red + green).

Allowed, in addition, the following sets of arguments:

$(F, xMin, xMax, yMin, yMax, zMin, zMax, gridSize)$,

$(F, xMin, xMax, yMin, yMax, zMin, zMax, lightX, lightY, lightZ, gridSize)$,

$(F, xMin, xMax, yMin, yMax, zMin, zMax, lightX, lightY, lightZ, color, gridS$

To plot the graph of a function in time with changing parameters, you should first specify the number of frames. Then, you should set the final value of parameters using the sliders. To build the graph, you should press the 'Plot' button. The value of the number of frames and parameters on the sliders can be specified via **set3D**().

Example.

```
SPACE = R64[x, y, z];

f = -x^2+2y^2+3z^2-25;
\implicitPlot3d( f, -10, 10, -10, 10, -10, 10);
```

Hyperboloid

```
SPACE = R64[x, y, z ];
\implicitPlot3d( x^2+ y^2+ z^2-25 , -7, 7, -7, 7, -7, 7,
10, 10, 10, 255*256*256, 100);
```

Red sphere.

```
SPACE = R64[x, y, z];
f =  \sin(xyz/100) ;
\implicitPlot3d( f , -9,9,-9,9,-9,9, 10,10, 4, 255*256*256+255*256, 5
```

Yellow surface with a central symmetry.

```
SPACE = R64[x, y, z];
f = ((x+2)^2+ (y-2)^2 -1)((x-2)^2+ (y+2)^2 -1)((x+2)^2+
(y+2)^2 -1) ((x-2)^2+ (y-2)^2 -1)(x^2+ y^2 -1);
\implicitPlot3d( f, -10, 10, -10, 10, -10, 10 );
```

Organ pipes.

3.2.6. Plot different 3D graphs of functions in one coordinate system

To plot the graphs of functions defined in different ways, you need to execute the command **showPlots3D**(f, g), where f and g are commands for building other graphs of a function.

You can set the environment parameters with **set3D**().

You can rotate the coordinate system by moving the mouse pointer with the left button pressed. You can also shift the origin of the coordinate system by moving the mouse pointer with the right button pressed.

To plot the graph of a function in time with changing parameters, you should first specify the number of frames. Then, you should set the final value of parameters using the sliders. To build the graph, you should press the 'Plot' button. The value of the number of frames and parameters on the sliders can be specified via **set3D**().

```
SPACE = R64[x, y, z];
\set3D(-5,5,-5,5,-10,10,40);
f = -x^2+2y^2+3z^2-25;
g = (x^2+y^2)/20;
\showPlots3D(\implicitPlot3d(f), \explicitPlot3d(g));
```

3.3. Geometry

paintElement$'operator1; operator2; ...operatork;'$, is a drawing tool for school geometry.

Operators are defined like this:
operator ($arg\_1 : type\_1, ..arg\_n : type_n = default$): returnType

$arg\_1, ..arg\_n$ - list of operator arguments.
$type\_1, ..type\_n$ - types of arguments, strings or numbers, or other object.
= default is the value to use if no argument is given.
returnType is the type of the returned object.

Operators can have an additional label argument: string, which defines the signature of the figure, and the requirement to display in the figure:
• label not set - the shape is used as an intermediate shape and is NOT drawn.
• label = % % - the figure is drawn, but without a caption.
• label =% text% - the shape is labeled with text.

Figure signature: operator (arg_1, .. arg_n) .display (% text%);

Lines are delimited by percent signs, not quotation marks as in mfthpar.

3.3.1. Example (draw circle)

Circle (radius: r1, center: Point = Point (x1, y1)).
You can use different methods of creating a circle:
• Circle (r1, Point (x1, y1));
• Circle (r1); - the center will automatically be at x1 = 0, y1 = 0.
• R = r1; P = Point (x1, y1) .display ("% O1%); C = Circle (R, P);

3.3.2. Operators

Point (x: number1, y: number2) is the point constructor.
• x, y - point position.

```
\paintElement ('a = Point (1, 1) .display (\% A\%);');
```

Line (point1: Point, point2: Point) - line constructor.
• point1, point2 - line points. The order doesn't matter.

```
\paintElement ('l=Line(Point(1,1),Point(2,2)).display(% L%); ');
```

Polygon (point1, point2, ..., pointN: Point []) - a polygon constructor, a figure that is a list of points connected by line segments. The last point is connected to the first.
• point1, point2, ..., pointN - Polygon vertices. The length of the vertex list is not limited.

```
\paintElement (pl=Polygon(Point(1,1),Point(3,3),Point(0,2)).display()
```

Rectangle (width: number1, height: number2, bottomLeft: Point = Point (1, 1)) - the constructor of the horizontal rectangle.
• width - the width of the rectangle.
• height - the height of the rectangle.
• bottomLeft - bottom left point of the rectangle.

```
\paintElement (pl = Rectangle (1, 2, Point (4,0)). display (); ');
```

Square (size: number, bottomLeft: Point = Point (x1, y1)) - square constructor.
• size - the length of the side of the square.
• bottomLeft - bottom left point of the square.

begin verbatim paintElement (pl = Square (2, Point (4,0)). display (); ');
Triangle (point1: Point, point2: Point, point3: Point) - three-point triangle constructor. The order doesn't matter.
• point1, point2, point3 - triangle points.

```
\paintElement (pl=Triangle(Point(1,1),Point(3,3),
Point(0,2)).display (); ');
```

Circle (radius: number1, center: Point = Point (x1, y1)) is the circle constructor.
• radius - radius of the circle.
• center - the center of the circle.

```
\paintElement ('c = Circle (3) .display ();');
```

Ellipse (width: number1, height: number2, center: Point = Point (x1, y1)) - ellipse constructor.
• width - horizontal semiaxis of the ellipse.
• height - vertical semiaxis of the ellipse.
• center - the center of the ellipse.

```
\paintElement ('a = Ellipse (2, 3) .display ();');
```

normal (point: Point1, line: Line1): Point - draw a perpendicular from point to line. Returns the intersection point of a perpendicular and a line.
• point - the point from which the perpendicular is restored.
• line - a straight line to which the perpendicular is restored.

```
\paintElement ('a=Point(1, 1).display(% A%); l=Line(Point(1,3),
Point(3,1)).  display(% L%); n=normal(a,l).display(% N%); ');
```

median (point: Point, line: Line): Point - build a median from point to section. Returns the midpoint of a line segment.
• point - the point from which to build the median.
• line - the segment on which the median is based.

```
\paintElement('a =Point(1, 1).display(% A%);l=Line(Point(1,3),
Point(3,1)).display (% L%); m=median( a, l).display(% M%); ');
```

Text (text:% string1%, leftBottom: Point1, fontSize: number = 10) - write text starting at a certain point. • text - the actual text.
• leftBottom - bottom left (starting) point of the text.
• fontSize - text font size.

```
\paintElement ('a =Text(% This is text%,
 Point(1,1)).display(); ');
```

middle (line: Line): Point - find the midpoint of the line segment.
• line - the segment whose middle you want to find.

```
\paintElement (l=Line(Point(1,3),Point(3,1)).display(% L%);
 m=middle (l)  .display(% m%); ');
```

incircle (triangle: Triangle): Circle - draw a circle inscribed in the tracker.
• triangle - a triangle into which the circle should be inscribed

```
\paintElement (tr=Triangle(Point(1,3),Point(3,1),
 Point(1,1)).display();c =  incircle(tr).display (); ');
```

Circumcircle (triangle: Triangle): Circle - draw a circle around the triangle. • triangle - a triangle around which you want to describe a circle

```
\paintElement (tr=Triangle(Point(1,3),Point(3,1),
 Point(1,1)).display();c =   circumcircle(tr).display(); ');
```

lineCircleCross (line: Line, circle: Circle): Point [] - find and return points of intersection of a line and a circle.
• line - a line that can cross the circle.
• circle - a circle that can cross the line.

```
\paintElement('l= Line(Point(1,3),Point(3,1)).display(% L%);
 c=Circle(3)  .display(); p=lineCircleCross(l,c); ');
```

circlesCross (circle1: Circle, circle2: Circle): Point [] - find and return points of intersection of two circles.
• circle1, circle2 - circles that can be crossed.

```
\paintElement('c1=Circle(2).display();c2=Circle(3,Point(2,2)).display
p=circleCross(c1,c2);');
```

Chapter 4

Environment for mathematical objects

4.1. Setting of environment

The definition of any mathematical object, a number or function, a matrix or symbol, involves the definition of some environment, that is, the space which contains this object. To select the environment you have to set the *algebraic structure*. This algebraic structure is defined by numeric sets, algebraic operations in these sets and variable names.

First of all any user have to set an environment in Mathpar.

By default, a space of the four real variables $\mathbb{R}64[x, y, z, t]$ is defined. This is ring of polynomials with coefficients in the ring of real numbers, the youngest is the variable x, the eldest is the variable t.

User can change the environment, setting a new algebraic structure.

For example the spaces $\mathbb{R}64[x]$ or $\mathbb{Q}[x]$ may be suitable to solve many problems of computational mathematics. The installation command should be the follow: ¡¡SPACE=R64[x];¿¿ or ¡¡SPACE=Q[x];¿¿.

Moving a mathematical object from the previous environment to the current environment, as a rule, should be performed explicitly, using the $toNewRing()$ function. In some cases, such a transformation to the current environment is automatic.

All other names which are not listed as a variables can be chosen arbitrarily by the user for any mathematical object.

For example

$$a = x + 1, \quad f = \backslash\backslash sin(x + y) - a.$$

We follow the rule. If the object name begins with the symbol \ and a *capital letter* such object is an element of a *noncommutative* algebra, else object is an element of a *commutative* algebra.

4.2. Numerical sets with standard operations

Current version of the system supports the following numerical sets with standard operations.

Z — the set of integers $\mathbb{Z}$,

Zp — a finite field $\mathbb{Z}/p\mathbb{Z}$ where p is a prime number,

Zp32 — a finite field $\mathbb{Z}/p\mathbb{Z}$ where p is less 2^{31},

Z64 — the ring of integer numbers z such that $-2^{63} \leqslant z < 2^{63}$,

Q — the set of rational numbers,

R — the set of floating point numbers to store the approximate real numbers with arbitrary mantissa,

R64 — standard floating-point 64-bit numbers (52 digits for mantissa, 11 bits for the order and 1 bit for the sign),

R128 — standard floating-point 64-bit numbers, equipped with optional 64-bit for the order,

C — complexification of R,

C64 — complexification of R64,

C128 — complexification of R128,

CZ — complexification of of Z,

CZp — complexification of Zp,

CZp32 — complexification of Zp32,

CZ64 — complexification of Z64,

CQ — complexification of Q.

Examples of simple commutative polynomial rings:

SPACE = Z [x, y, z];

SPACE = R64 [u, v];

SPACE = C [x].

4.3. Several numerical sets

The ring $Z[x, y, z]Z[u, v, w]$, which has two subsets of variables, is the polynomial ring with variables u, v, w with coefficients in the polynomial ring $Z[x, y, z]$.

For example, the characteristic polynomial of a matrix over the ring $\mathbb{Z}[x, y, z]$ may be obtained as a polynomial with the variable u, whose coefficients are polynomials in the ring $\mathbb{Z}[x, y, z]$.

You can set algebraic space which defines several numerical sets. For example, the space $C[z]R[x,\ y]Z[n,\ m]$ allows to have the five names of variables, which defined in the sets $\mathbb{C}$, $\mathbb{R}$ and $\mathbb{Z}$, respectively. The first set is the main.

$C[z]R[x, y]Z[n, m]$ can be viewed as a polynomial ring of five variables over $\mathbb{C}$, which has the additional properties. If the polynomial does not contain the variables z, x, y, then it is a polynomial with coefficients in the set $\mathbb{Z}$. If the polynomial does not contain the variable z, then it is a polynomial with coefficients in the set $\mathbb{R}$.

Examples:

SPACE=Z[x, y]Z[u];

SPACE=R64[u, v]Z[a, b];

SPACE=C[x]R[y, z];

4.4. Idempotent algebra and tropical mathematics

User can uses the idempotent algebras. In this case the signs of "addition" and "multiplication" for the infix operations can be used for operations in tropical algebra: min, max, addition, multiplication.

Each numerical sets $\mathbb{R}$, $\mathbb{R}64$, $\mathbb{Z}$ has two additional elements ∞ and $-\infty$, and they have different elements, which is play the role of zero and unit. We denote these sets $\hat{\mathbb{R}}$, $\hat{\mathbb{R}}64$, $\hat{\mathbb{Z}}$, correspondingly. The name of tropical algebra is obtained from three words: (1) a numerical set, (2) an operation, which corresponding to the sign *plus* and (3) an operation, which corresponding to the sign *times*.

The algebras $R64MaxPlus$, $R64MinPlus$, $R64MaxMin$, $R64MinMax$, $R64MaxMult$, $R64MinMult$ are defined for the numerical set $\hat{\mathbb{R}}64$.

RMaxPlus, RMinPlus, RMaxMin, R64MinMax, RMaxMult, RMinMult are defined for the numerical set $\hat{\mathbb{R}}$.

ZMaxPlus, ZMinPlus, ZMaxMin, ZMinMax, ZMaxMult, ZMinMult are defined for the numerical set $\hat{\mathbb{Z}}$.

For example, for the algebra $ZMaxPlus$ you can do the following operations.

Example.

```
SPACE=ZMaxPlus[x, y];
a=2; b=9+x; c=a+b; d=a b+y; \print(c, d)
```

The results: $c = x + 9$; $d = y + 2x + 11$.

For each algebra we defined elements **0** and **1**, $-\infty$ and ∞. For each element a we defined the operation of closure: $a^{\times}$, i.e. the amount of $1+a+a^2+a^3+....$ For the classical algebras this operation is equivalent to $(1-a)^{-1}$.

4.5. Constants

It is possible to set or replace the following constants.

FLOATPOS — an amount of decimal positions of the real number of type R or R64, which you can see in the printed form of this number (the default value is 2).

MachineEpsilonR — machine epsilon for the number of type R and C (10^{-29} is the default value). The number whose absolute value is less than 10^{-29}, is considered to be a machine zero. To set the new value of 10^{-30}, enter the command ¡¡ MachineEpsilonR64 = 30 ¿¿.

MachineEpsilonR64 — machine epsilon for the number of type R64 and C64 (2^{-36} is the default value). The number whose absolute value is less than 2^{-36}, is considered to be a machine zero. To set the new value of 2^{-48}, enter the command ¡¡ MachineEpsilonR64 = 48 ¿¿.

Constant MachineEpsilonR (and MachineEpsilonR64) used in factoring polynomials with coefficient of type R (or R64). Each coefficient of the polynomial is divided by the number $MachineEpsilonR$ (or $MachineEpsilonR64$) and rounded to integer value.

MOD32 — — the module for a finite field of the type Zp32, its value is not greater than 2^{31}. (the default value is 268435399).

MOD — the module for a finite field of the type Z (the default value is 268 435 399).

RADIAN — (1/0) is a flag, which indicates that angles are measured in radians (default is 1: active).

STEPBYSTEP — (1/0) is a flag, which indicates that you want to display intermediate results (default is 0: turned off).

EXPAND — (1/0) is a flag, which indicates that in the input expression all brackets must be disclosed (the default is 1: active).

SUBSTITUTION — (1/0) is a flag, which indicates that the names in the input expression must be substituted of their meaning, if they have been defined before (the default is 1: active).

The constant ACCURACY is an amount of exact decimal positions in the fractional part of a real numbers of type R and C in the result of multiplication or division operation (the default value is 34).If ACCURACY = 100, then the result of arithmetic operation will be rounded to the one hundredth decimal place. Obviously, the inequality ACCURACY > MachineEpsilonR must hold.

To install the MachineEpsilonR $1/10^9$ (i.e. $1E-9$) enter the command MachineEpsilonR = 9 . After that, any number $a \in R$, will be considered as zero if $|a| < 10^{-9}$. The value ACCURACY will be set MachineEpsilonR+5; If you like another value of ACCURACY you can set the fraction ¡¡MachineEpsilonR=35/49¿¿. In this case you obtain result: MachineEpsilonR=35 and ACCURACY=49.

For the numbers $a \in R64$, $a \in R128$, $a \in C64$, $a \in C128$, there is no constant defining binary places. Arithmetic instructions are used with accuracy that equals 2.22044604925031308e-16. However, you can set the number of binary places for MachineEpsilonR64: for example, you can set MachineEpsilonR64=10.

Prime number MOD32 is a characteristic of a finite field. The constant MOD32 is used when calculations are made in a finite field Zp32 and it should be less 2^{31}.

The prime number MOD is also characteristic of the finite field, but it has no restrictions on the absolute value. The constant MOD is used when calculations are made in a finite field Zp.

The constant FLOATPOS determines the number of decimal places, are printed. In addition, it is used in the factorization of polynomials whose coefficients are an approximate numbers of type R or R64. Each coefficient of this polynomial is pre-multiplied by the number of $10^{MachineEpsilonR}$ or and rounded to an integer value. But after factoring polynomial extra factor is removed.

```
SPACE=Zp32[x, y];
MOD32=7;
f1=37x+42y+55;
f2=2f1;
\print(f1, f2 );
```

Chapter 5

Functions of one and several variables

5.1. Mathematical functions

The following notations for elementary functions and constants are accepted.

5.1.1. Constants

\i — imaginary unit,
\e — the basis of natural logarithm,
\pi — the ratio of length of a circle to its diameter,
\infty – infinity symbol.

5.1.2. Functions of one argument

\ln — natural logarithm,
\lg — decimal logarithm,
\sin — sine,
\cos — cosine,
\tg — tangent,
\ctg — cotangent,
\arcsin — arcsine,
\arccos — arccosine,

\arctg — arctangent,
\arcctg — arccotangent,
\sh — sine hyperbolic,
\ch — cosine hyperbolic,
\th — tangent hyperbolic,
\cth — cotangent hyperbolic,
\arcsh — arcsine hyperbolic,
\arcch — arccosine hyperbolic,
\arcth — arctangent hyperbolic,
\arccth — arccotangent hyperbolic,
\exp — exponent,
\sqrt — root square,
\abs — absolute value of real numbers (module for complex numbers),
\sign — number sign (returns 1, 0, -1 when number sign is $+$, 0, $-$, correspondingly),
\unitStep(x) — is a function that for $x > 0$ takes the value 1, and for $x < 0$ takes the value 0;
\fact — factorial. It is defined for positive integers and equivalent to $n!$.

5.1.3. Functions of two arguments

^ — degree,
\log — logarithm of function with given base,
\rootOf(x, n) — root of degree n of x,
\Gamma — the function Gamma,
\Gamma2 — the function Gamma 2,
\binomial — binomial coefficient.

Examples.

```
SPACE = R64[x, y];
f1 = \sin(x);
f2 = \sin(\cos(x + \tg(y)));
f3 = \sin(x^2) + y;
\print(f1, f2, f3);
```

The results:
$f1 = sin(x)$;

$f2 = sin(cos(x + tg(y)))$;
$f3 = sin(x^2) + y$.

5.2. Calculation of the value of a function in a point

To calculate the value of a function in a point execute the command $\backslash$**value**$(f, [var1, var2, \ldots, varn])$, where f is a function, and $var1$, $var2, \ldots,$ $varn$ are values of variables, which are substituted instead of corresponding variables.

You can use radians or degrees for an angular measure. For an indication of angular measure, you can set the constant RADIAN. If you do not specify the angular measure, the radians is chosen. To change the angular measure from radians to degrees, run $RADIAN = 0$. If you need to change the angular measure in radians, then run $RADIAN = 1$.

If the arguments of trigonometric functions is integer, which is equal to $15k$ or $18k$ degrees (i.e. $\pi k/12$ and $\pi k/10$ radians, $k \in \mathbb{Z}$), then values of the trigonometric functions are algebraic numbers.

Examples.

```
SPACE = R[x, y];
f = \sin(x^2 + \tg(y^3 + x));
g = \value(f, [1, 2]);
\print(g);
```

Returns:
in: $SPACE = R[x, y]$;
$f = sin(x^2 + tg(y^3 + x))$;
$g = value(f,\ [1,\ 2])$; $print(g)$;
out: $g = 0.52$;

```
SPACE = Z[x];
RADIAN = 0;
f = \sin(x);
g = \value(f, 15);
\print(g);
```

Returns:
in: $SPACE = Z[x]$;

$RADIAN = 0;$
$f = sin(x);$
$g = value(f, 15);$
$print(g);$
out: $g = (\sqrt{6} - (\sqrt{2}))/(4);$

```
SPACE = Z[x];
RADIAN = 0;
f = \sin(x);
g = \value(f, 225);
\print(g);
```

Returns:
$g = (-1 \cdot \sqrt{2})/(2);$

```
SPACE = Z[x];
RADIAN = 0;
f = \cos(x);
g = \value(f, 54);
\print(g);
```

Returns:
$g = \sqrt{(5 - \sqrt{5})/(8)};$

```
SPACE = Z[x];
RADIAN = 0;
f = \tg(x);
g = \value(f, 126);
\print(g);
```

Returns:
$g = (-1 \cdot \sqrt{(1 + 2 \cdot \sqrt{5}/(5))});$

```
SPACE = Z[x];
RADIAN = 0;
f = \sin(x);
g = \value(f, 216);
\print(g);
```

Returns:
$g = (-1 \cdot \sqrt{(5 - \sqrt{5})/(8)});$

```
SPACE = Z[x];
RADIAN = 0;
f = \cos(x);
g = \value(f, 108);
\print(g);
```

Returns:
$g = (1 - \sqrt{5})/(4)$.

5.3. Substitution of functions instead of ring variables

To calculate the composition of functions some functions must be substituted in place of the arguments. For this you must run **value**$(f, [func1, func2, \ldots, funcn])$, where f — this function, $func1, func2, \ldots, funcn$ — functions that are substituted into the corresponding places.

Example.

```
SPACE = Z[x, y];
f = x + y;
g = f^2;
r = \value(g, [x^2, y^2]);
\print(r);
```

Returns:
in: $g = y^2 + 2yx + x^2$;
$f = y + x$;
out: $r = y^4 + 2y^2x^2 + x^4$.

5.4. Calculation of the limit of a function

To calculate the limit of a function at a point you must run **lim**(f, var), where f — this function, and var — point, possibly infinite, in which you want to find the limit. The limit may not exist, may be finite or infinite.

Examples.

```
SPACE = R64[x];
f = \sin(x) / x;
g = \lim(f, 0);
\print(g);
```

Returns:

in: $SPACE = R64[x]$;
$f = sin(x)/x$;
$g = lim(f, 0)$;
$print(g)$;
out: $g = 1.00$;

```
SPACE = R64[x];
f = (x^2 - 2x + 2) / (x^2 + x - 2);
g = \lim(f, 1);
\print(g);
```

Results:

Returns:
in: $SPACE = R64[x]$;
$f = (x^2 - 2x + 2)/(x^2 + x - 2)$;
$g = lim(f, 1)$;
$print(g)$;
out: $g = \infty$;

```
SPACE = R64[x];
f = \sin(x + 3) / (x^2 + 6x + 9);
g = \lim(f, -3);
\print(g);
```

Results:

Returns:
in: $SPACE = R64[x]$;
$f = \sin(x + 3)/(x^2 + 6x + 9)$;
$g = lim(f, -3)$;
$print(g)$;
out: $g = \infty$;

```
SPACE = R64[x];
f = (1 + 1 / x)^x;
g = \lim(f, \infty);
\print(g);
```

Results:
Returns:
in: $SPACE = R64[x]$;
$f = (1 + 1/x)^x$;
$g = lim(f, \infty)$;
$print(g)$;
out: $g = 2.72$.

5.5. Differentiation of functions

To differentiate a function $f(x, y, z)$ with lowest variable x, you have to execute one of commands $\backslash\mathbf{D}(f)$, $\backslash\mathbf{D}(f, x)$ or $\backslash\mathbf{D}(f, x \hat{} 1)$. To fine the second derivative of $f(x, y, z)$ by variable y, you have to execute the command $\backslash\mathbf{D}(f, y \hat{} 2)$. And so on.

To find a mixed first-order derivative of the function f there is a command $\backslash\mathbf{D}(f, [x, y])$, to find the derivative of higher order to use the command $\backslash\mathbf{D}(f, [x \hat{} k, z \hat{} m, y \hat{} n])$, where k, m, n indicate the order of the derivative.

Examples.

```
SPACE = Z[x, y];
f = \sin(x^2 + \tg(y^3 + x));
h = \D(f, y);
\print(h);
```

Returns:
in: $SPACE = Z[x, y]$;
$f = sin(x^2 + tg(y^3 + x))$;
$h = D(f, y)$;
$print(h)$;
out: $h = 3y^2 cos(x^2 + tg(y^3 + x))/(cos(y^3 + x))^2$;

```
SPACE = Z[x, y];
f = \sin(x^2 + \tg(y^3 + x));
h = \D(f);
\print(h);
```

Returns:
in: $SPACE = Z[x, y]$;
$f = sin(x^2 + tg(y^3 + x))$;
$h = D(f)$;

$print(h)$;
out: $h = (2x\cos(x^2 + tg(y^3 + x))(\cos(y^3 + x))^2 + \cos(x^2 + tg(y^3 + x)))/(\cos(y^3 + x))^2$;

```
SPACE = Z[x, y, z];
f = x^8y^4z^9;
g = \D(f, [x^2, y^2, z^2]);
\print(g);
```

Returns:
in: $SPACE = Z[x, y, z]$;
$f = x^8y^4z^9$;
$g = D(f, [x^2, y^2, z^2])$;
$print(g)$;
out: $g = 48384z^7y^2x^6$.

5.6. Integration of the compositions of elementary functions

Symbolic integration of compositions of elementary functions is performed by using the $\backslash\mathbf{int}(f(x))dx$.

Examples.

```
SPACE = Z[x, y, z];
l1 = \int(x^6yz + 3x^2y - 2z) d x;
dl1 = \D(l1,x);
l2 = \int(x^6yz + 3x^2y - 2z) d y;
dl2 = \D(l2,y);
l3 = \int(x^6yz + 3x^2y - 2z) d z;
dl3 = \D(l3,z);
\print(l1, dl1,l2, dl2,l3, dl3);
```

Returns:
in: $SPACE = Z[x, y, z]$;
$l1 = \int(x^6yz + 3x^2y - 2z)dx$;
$dl1 = D(l1, x)$;
$l2 = \int(x^6yz + 3x^2y - 2z)dy$;
$dl2 = D(l2, y)$;
$l3 = \int(x^6yz + 3x^2y - 2z)dz$;

$dl3 = D(l3, z)$;
$print(l1, dl1, l2, dl2, l3, dl3)$;
out: $l1 = (1/7)zyx^7 - 2zx + yx^3$;
$dl1 = x^6yz + 3x^2y - 2z$. $l2 = (1/2)zy^2x^6 - 2zy + (3/2)y^2x^2$;
$dl2 = x^6yz + 3x^2y - 2z$. $l3 = (1/2)z^2yx^6 - z^2 + 3zyx^2$;
$dl3 = x^6yz + 3x^2y - 2z$.

```
SPACE = R[x];
l = \int(1/(x^2-5x+6)) d x;
dl = \D(l,x);
\print(l, dl);
```

Returns:
in: $SPACE = Q[x, y, z]$;
$l = \int(1/(x^2 - 5x + 6))dx$;
$dl = D(l, x)$;
$print(l, dl)$;
out: $l = (\ln(x - 3) - \ln(x - 2))$;
$dl = (1/(x - 3) - 1/(x - 2))$.

```
SPACE = Q[x];
l = \int(\exp(x)+\exp(-x)) d x;
dl = \D(l,x);
\print(l, dl);
```

Returns:
in: $SPACE = Q[x, y, z]$;
$l = \int(\exp(x) + \exp(-x))dx$;
$dl = D(l, x)$;
$print(l, dl)$;
out: $l = (\exp(x) - ((\exp(x))^{-1}))$;
$dl = (\exp(x) + \exp(-x))$.

```
SPACE = Q[x];
l = \int(x*\exp(x^2)) d x;
dl = \D(l,x);
\print(l, dl);
```

Returns:
in: $SPACE = Q[x, y, z]$;
$l = \int(x * \exp(x^2))dx$;
$dl = D(l, x)$;

$print(l, dl)$;
out: $l = (\exp(x^2)/2)$;
$dl = (x * \exp(x^2))$.

```
SPACE = Q[x];
l = \int((x*\ln(x)*\exp(x)+\exp(x))/x) d x;
dl = \D(l,x);
\print(l, dl);
```

Returns:
in: $SPACE = Q[x, y, z]$;
$l = \int((x * \ln(x) * \exp(x) + \exp(x))/x)dx$;
$dl = D(l, x)$;
$print(l, dl)$;
out: $l = (\ln(x) * \exp(x))$;
$dl = ((x * \ln(x) * \exp(x) + \exp(x))/x)$.

```
SPACE = R64[x];
l = \int((\ln(x+3)+\ln(x+2)+\ln(x+1))) d x;
dl = \D(l,x);
\print(l, dl);
```

Returns:
in: $SPACE = R64[x, y, z]$;
$l = \int((\ln(x + 3) + \ln(x + 2) + \ln(x + 1)))dx$;
$dl = D(l, x)$;
$print(l, dl)$;
out: $l = (((x * \ln(x + 3) + 3.00 * \ln(x + 3) + x * \ln(x + 2) + 2.00 * \ln(x + 2) + x * \ln(x + 1) + \ln(x + 1)) - 3x))$;
$dl = ((\ln(x + 3) + \ln(x + 2) + \ln(x + 1)))$.

```
SPACE = Q[x];
l = \int((2x^2+1)^3) d x;
dl = \D(l,x);
m=\factor(dl);
\print(l, m);
```

Returns:
in: $SPACE = Q[x, y, z]$;
$l = \int((2x^2 + 1)^3)dx$;
$dl = D(l, x)$;
$m = factor(dl)$;

$print(l, m)$;
out: $l = (8/7)x^7 + (12/5)x^5 + 2x^3 + x$;
$m = (2x^2 + 1)^3$.

5.7. Simplification of compositions

For transformation of a trigonometric and logarithmic function by means of identities:
$sin(x)cos(y) \pm cos(x)sin(y) = sin(x \pm y)$
$cos(x)cos(y) \pm sin(x)sin(y) = cos(x \mp y)$
$sin^2(x) + cos^2(x) = 1$
$cos^2(x) - sin^2(x) = cos(2x)$
$ln(a) + ln(b) = ln(ab)$
$ln(a) - ln(b) = ln(\frac{a}{b})$
the command **Expand**$(f(x))$ is used.

Examples.

```
SPACE=Q[x, y, z];
g=\ln(x^2*4x);
f=\Expand(g);
\print(f);
```

Returns:
in: $SPACE = Q[x, y, z]$;
$g = \ln(x^2 * 4x)$;
$f = Expand(g)$;
$print(f)$;
out: $f = \ln(x^2) + \ln(4x)$;

```
SPACE=Q[x, y, z];
g=\sin(x^2+4x+2\pi);
f=\Expand(g);
\print(f);
```

Returns:
in: $SPACE = Q[x, y, z]$;
$g = \sin(x^2 + 4x + 2\pi)$;
$f = Expand(g)$;
$print(f)$;
out: $f = (\sin(x^2) * (\cos(4x) * \cos(2) - \sin(4x) * \sin(2)) + \cos(x^2) * (\sin(4x) * \cos(2) + \cos(4x) * \sin(2)))$;

```
SPACE=Q[x, y, z];
g=\cos(\sin(x)+\cos(y));
f=\Expand(g);
\print(f);
```

Returns:

in: $SPACE = Q[x, y, z]$;
$g = \cos(\sin(x) + \cos(y))$;
$f = Expand(g)$;
$print(f)$;

out: $f = (\cos(\cos(y)) * \cos(\sin(x)) - \sin(\cos(y)) * \sin(\sin(x)))$;

For simplification of a trigonometric and logarithmic function by means of all formulas mentioned above and formulas: $ln(a)^k = k \cdot ln(a)$
$e^{iz} + e^{-iz} = 2Cos(z)$
$e^{iz} - e^{-iz} = 2iSin(z)$
$Ln(1 + iz) - Ln(1 - iz) = 2i * arctg(z)$
$Ln(1 - iz) - Ln(1 + iz) = 2i * arcctg(z)$
$e^z + e^{-z} = 2Ch(z)$
$e^z - e^{-z} = 2iSh(z)$
the command $\backslash$**Factor**$(f(x))$ is used.

Examples.

```
SPACE=Q[x, y, z];
g=\log_{2}(x)+\log_{2}(y)-\log_{2}(xz)+\lg(y)+\lg(y)-\lg(z);
f=\Factor(g);
\print(f);
```

Returns:

in: $SPACE = Q[x, y, z]$;
$g = \log_2(x) + \log_2(y) - \log_2(xz) + \lg(y) + \lg(y) - \lg(z)$;
$f = Factor(g)$;
$print(f)$;

out: $f = \lg(y^2/z) + \log_2(y/z)$;

```
SPACE=Q[x, y, z];
g=16\sin(x/48)\cos(x/48)\cos(x/24)\cos(x/12)\cos(x/6);
f=\Factor(g);
\print(f);
```

Returns:

in: $SPACE = Q[x, y, z]$;

$g = 16\sin(\frac{x}{48})\cos(\frac{x}{48})\cos(\frac{x}{24})\cos(\frac{x}{12})\cos(\frac{x}{6})$;
$f = Factor(g)$;
$print(f)$;
out: $f = \sin(0.33x)$;

```
SPACE=C64[x, y, z];
g=\ln(1-\ix) - \ln(1+\ix) + \exp(\ix) - 2\exp(-\ix) + \sin(x)^2 - \co
f=\Factor(g);
\print(f);
```

Returns:
in: $SPACE = C64[x, y, z]$;
$g = \ln(1-ix) - \ln(1+ix) + \exp(ix) - 2\exp(-ix) + \sin(x)^2 - \cos(x)^2$;
$f = Factor(g)$;
$print(f)$;
out: $f = (-1.00 * \cos(2x)) + 2.00i * (\sin(x)) + (-1.00 * \exp(-ix)) + (2.00i * (arcctg(x)))$;

Unit of commands **Factor**$(f(x))$ and **Expand**$(f(x))$ allows to solve more difficult examples:

```
SPACE=R64[x, y, z];
g=(\sin(x+y) + \sin(x-y))\cos(x) + (\sin(x+y) + \sin(x-y))\sin(y);
f=\Expand(g);
u=\Factor(f);
\print(f,u);
```

Returns:
in: $SPACE = R64[x, y, z]$;
$g = (\sin(x+y) + \sin(x-y))\cos(x) + (\sin(x+y) + \sin(x-y))\sin(y)$;
$f = Expand(g)$;
$u = Factor(g)$;
$print(f, u)$;
out: $f = 2.00 * \cos(y) * \sin(x) * \cos(x) + 2.00 * \sin(y) * \cos(y) * \sin(x)$;
$u = \sin(x) * \sin(2.00y) + \cos(y) * \sin(2.00x)$;

5.8. Arithmetic-geometric mean

Given two non-negative numbers x and y, one can define their arithmetic, geometric and harmonic means as $\frac{x+y}{2}$, $\sqrt{xy}$, and $\frac{2xy}{x+y}$, respectively. Moreover, **AGM**(x, y) denotes the arithmetic-geometric

mean of x and y. $\backslash\mathbf{GHM}(x,y)$ denotes the geometric-harmonic mean of x and y. At last, $\backslash\mathbf{MAGM}(x,y)$ denotes the modified arithmetic-geometric mean of x and y. Every mean is a symmetric homogeneous function in their two variables x and y. In contrast to well-known means, $\backslash\mathbf{AGM}(x,y)$, $\backslash\mathbf{GHM}(x,y)$, and $\backslash\mathbf{MAGM}(x,y)$ are calculated iteratively.

The arithmetic-geometric mean $\backslash\mathbf{AGM}(x,y)$ is equal to the limit of both sequences x_n and y_n, where $x_0 = x$, $y_0 = y$, $x_{n+1} = \frac{1}{2}(x_n + y_n)$, and $y_{n+1} = \sqrt{x_n y_n}$.

In the same way, the geometric-harmonic mean $\backslash\mathbf{GHM}(x,y)$ is equal to the limit of both sequences x_n and y_n, where $x_0 = x$, $y_0 = y$, $x_{n+1} = \sqrt{x_n y_n}$, and $y_{n+1} = \frac{2x_n y_n}{x_n + y_n}$.

The modified arithmetic-geometric mean $\backslash\mathbf{MAGM}(x,y)$ is equal to the limit of the sequence x_n, where $x_0 = x$, $y_0 = y$, $z_0 = 0$, $x_{n+1} = \frac{x_n + y_n}{2}$, $y_{n+1} = z_n + \sqrt{(x_n - z_n)(y_n - z_n)}$, and $z_{n+1} = z_n - \sqrt{(x_n - z_n)(y_n - z_n)}$.

Examples.

```
SPACE=R64[];
FLOATPOS=5;
agm=\AGM(1,5);
ghm=\GHM(1,5);
p=agm*ghm;
magm=\MAGM(1,5);
\print(agm, ghm, p, magm);
```

Results:

$$agm = 2.60401$$

$$ghm = 1.92012$$

$$p = 5$$

$$magm = 2.61051$$

5.9. The complete elliptic integrals of the first and second kind

Let us use the parameter $0 \leq k \leq 1$.

The complete elliptic integral of the first kind $K(k)$ is defined as

$$K(k) = \int_0^1 \frac{dt}{\sqrt{(1-t^2)(1-k^2t^2)}}$$

It can be computed in terms of the arithmetic-geometric mean:

$$K(k) = \frac{\pi}{2AGM(1, \sqrt{1-k^2})}$$

On the other hand, for $k < 1$, it can be computed in terms of the geometric-harmonic mean:

$$K(k) = \frac{\pi}{2} GHM(1, \frac{1}{\sqrt{1-k^2}})$$

The complete elliptic integral of the second kind $E(k)$ is defined as

$$E(k) = \int_0^1 \sqrt{\frac{1-k^2t^2}{1-t^2}} dt$$

It can be computed in terms of the modified arithmetic-geometric mean:

$$E(k) = K(k) MAGM(1, 1-k^2)$$

See also: S. Adlaj (2012) An eloquent formula for the perimeter of an ellipse. Notices of the American Mathematical Society. 59(8), 1094-1099. DOI:10.1090/noti879

5.10. The period of a simple gravity pendulum

A point mass suspended from a pivot with a massless cord. The length of the pendulum equals $L = 1$ metre. It swings under gravitational acceleration $g = 9.80665$ metres per second squared. The maximum angle that the pendulum swings away from vertical, called the amplitude, equals $\theta_0 = 2.0944$, that is, $\frac{2}{3}\pi$ radians.

Find the period T of the pendulum using the arithmetic-geometric mean

$$T = \frac{2\pi}{AGM(1, \cos(\theta_0/2))} \sqrt{\frac{L}{g}}$$

```
SPACE = R64[];
FLOATPOS = 4;
\theta_0=2.0944;
w=\cos(\theta_0/2);
Ts = 2*\pi*\sqrt{L/g}/\AGM(1,w);
 \print(Ts);
L = 1; g = 9.80665;
T=\value(Ts);   \print(T);
```

Results:
$Ts = 2.7458 * \pi * (L/g)^{(1/2)}$
$T = 2.7546$

Chapter 6

Series

A series given in the form f=**s**um _{i=k} ^{ \infty} $F(i, x, y, \ldots, z)$, where i — summation index, k — initial value of i, $F(i, x, y, \ldots, z)$ — a function of many variables, which may depend on i.

There are defined the following arithmetic operations with series: addition, subtraction, multiplication.

Let f and g are series.

To add two series to execute the **seriesAdd**(f, g).

To calculate the difference between two series should run the command **seriesSubtract**(f, g).

For multiplication of two series should run the command **seriesMultiply**(f, g).

For the expansion of a function in a Taylor series with a certain number of members, you must run **teilor**$(f, point, num)$, where f — function, *point* — point, *num* — a number of members of the series.

Examples.

```
SPACE=R[x, y];
f=\sum_{i=2}^{\infty} (2x^i y b i);
g=\sum_{i=4}^{\infty} (x^i a\sin(a i x));
h=\seriesAdd(f, g);
\print(f, g, h);
```

Results:

$$f = \sum_{i=2}^{\infty} 2(x)^i ybi;$$

$$g = \sum_{i=4}^{\infty} (x)^i a \sin(aix);$$

$$h = \sum_{i=4}^{\infty} (2(x)^i ybi + (x)^i a \sin(aix)) + \sum_{i=2}^{3} (2x^i ybi);$$

```
SPACE=R[x, y];
f=\sum_{i=1}^{\infty} (x^i y i\cos(b));
g=\sum_{i=2}^{\infty} (5x^i a\cos(a x i));
h=\seriesSubtract(f, g);
\print(f, g, h);
```

Results:

$$f = \sum_{i=1}^{\infty} (x)^i yi \cos(b);$$

$$g = \sum_{i=2}^{\infty} 5(x)^i a \cos(axi);$$

$$h = \sum_{i=2}^{\infty} ((x)^i yi \cos(b) - 5(x)^i a \cos(axi)) + xy \cos(b);$$

```
SPACE=R[x, y];
f=\sum_{i=0}^{\infty} (2x^i y b i);
g=\sum_{i=2}^{\infty} (5y^i x^2 b i\cos(a_1 x));
h=\seriesSubtract(f, g);
\print(f, g, h);
```

Results:

$$f = \sum_{i=0}^{\infty} 2(x)^i ybi;$$

$$g = \sum_{i=2}^{\infty} 5y^i x^2 bi \cos(a_1 x);$$

$$h = \sum_{i=2}^{\infty} (2x^i ybi - 5y^i x^2 bi \cos(a_1 * x)) + \sum_{i=0}^{1} (2x^i ybi);$$

```
SPACE=R[x, y]; \clean();
f=\sum_{a=6}^{\infty} (x^a a_0);
g=\sum_{a=9}^{\infty} (6x^a\cos(a_1 x));
h=\seriesMultiply(f, g);
\print(f, g, h);
```

Results:

$$f = \sum_{a=6}^{\infty} (x)^a * a_0;$$

$$g = \sum_{a=9}^{\infty} 6 * (x)^a * \cos(a_1 * x);$$

$$h = \sum_{a_2=6}^{\infty} \sum_{a=9}^{\infty} (x)^{a_2} * a_0 * 6 * (x)^a * \cos(a_1 * x);$$

```
SPACE=R[x, y];
f=\sum_{a=6}^{\infty} (y^a\sin(x a)\cos(y)a_0);
g=\sum_{a=9}^{\infty} (6y^a\sin(a x y^2));
h=\seriesMultiply(f, g);
\print(f, g, h);
```

Results:

$$f = \sum_{a=6}^{\infty} (y)^a \sin(xa) \cos(y) a_0;$$

$$g = \sum_{a=9}^{\infty} 6(y)^a \sin(axy^2);$$

$$h = \sum_{a_1=6}^{\infty} \sum_{a=9}^{\infty} (y)^{a_1} \sin(xa) \cos(y) a_0 6(y)^a \sin(axy^2);$$

```
SPACE=R[x];
FLOATPOS=15;
a=\teilor(\sin(x), 0, 7);
c=\value(a);
\print(a, c);
```

Returns:
in: SPACE=R[x];
FLOATPOS=15;
a=**teilor**$(\sin(x), 0, 7)$;
c=**value**(a); **print**(a, c);
out:
$a = ((-x^7)/(7!) + x^5/(5!) + (-x^3)/(3!) + x/(1!))$;
$c = (-0.000198412698412x^7 + 0.008333333333333x^5 - 0.166666666666666x^3 + x)$.

Chapter 7

Solution of systems of differential equations

7.1. The solution of first-order differential equations

To find the solution of the differential equation it is necessary:
1. Specify the space variables ($SPACE$).
2. Set equation and get a solution (**solveDE**).

Examples.
Equation with separating variables:

```
SPACE=Q[x,y];
\solveDE(\d(y,x) = (2/y)\sin(x)\cos(x));
```

Linear homogeneous equation:

```
SPACE=Q[x,y];
\solveDE(x\d(y,x) = y-x\exp(y/x));
```

Equation in total differentials:

```
SPACE=Q[x,y];
\solveDE((3x^2-3y^2+4x)\d(x)-(6xy+4y)\d(y) = 0);
```

7.2. Solution of differential equations

Procedure of solving a differential equation consists of four steps.
1. To set the ring ($SPACE$).
2. To set an equation(**systLDE**).
3. To set initial conditions (**initCond**).
4. Solving the equation(**solveLDE**).

Examples.

```
SPACE=R64[t];
g=\systLDE(\d(y, t, 3)+3\d(y, t, 2)+3\d(y, t)+y=1);
f=\initCond(\d(y, t, 0, 0)=0, \d(y, t, 0, 1)=0,
                  \d(y, t, 0, 2)=0);
h=\solveLDE(g, f);
\print(h);
```

Returns:
in: $SPACE = R64[t]$;
$g = y_t^{'''} + 3y_t^{''} + 3{y'}_t + y = 1$;
$f = \begin{cases} y_{t=0} &= 0, \\ y'_{t=0} &= 0, \\ y''_{t=0} &= 0, \end{cases}$
$h = solveLDE(g, f)$;
$print(h)$;
out: $h = (1.00 + (-t^2)e^{-t}/2.00) - (te^{-t} + e^{-t})$;

```
SPACE=R64[t];
g=\systLDE(\d(y, t, 2)-2\d(y, t)+y=\exp(t));
f=\initCond(\d(y, t, 0, 0)=1, \d(y, t, 0, 1)=2);
h=\solveLDE(g, f);
\print(h);
```

Returns:
in: $SPACE = R64[t]$;
$g = y_t^{''} - 2{y'}_t + y = e^t$;
$f = \begin{cases} y_{t=0} &= 1, \\ y'_{t=0} &= 2, \end{cases}$
$h = solveLDE(g, f)$;
$print(h)$;
out: $h = e^t t^2/2.00 + te^t + e^t$;

```
SPACE=R64[t];
g=\systLDE(\d(y, t, 2)+\d(y, t)-12y=3);
f=\initCond(\d(y, t, 0, 0)=1, \d(y, t, 0, 1)=0);
h=\solveLDE(g, f);
\print(h);
```

Returns:
in: $SPACE = R64[t]$;
$g = y_t^{''} + y'_t - 12y = 3$;
$f = \begin{cases} y_{t=0} & = & 1, \\ y'_{t=0} & = & 0, \end{cases}$
$h = solveLDE(g, f)$;
$print(h)$;
out: $h = 1.11e^{-1.62t} + 2.89e^{0.62t} - 3.00$;

```
SPACE=R64[t];
g=\systLDE(\d(y, t)-2y=0);
f=\initCond(\d(y, t, 0, 0)=1);
h=\solveLDE(g, f);
\print(h);
```

Returns:
in: $SPACE = R64[t]$;
$g = y'_t - 2y = 0$;
$f = y_{t=0} = 1$;
$h = solveLDE(g, f)$;
$print(h)$;
out: $h = e^{2t}$;

```
SPACE=R64[t];
g=\systLDE(\d(y, t, 2)-4y=4t);
f=\initCond(\d(y, t, 0, 0)=a, \d(y, t, 0, 1)=b);
h=\solveLDE(g, f);
\print(h);
```

Returns:
in: $SPACE = R64[t]$;
$g = y_t^{''} - 4y = 4t$;
$f = \begin{cases} y_{t=0} & = & a, \\ y'_{t=0} & = & b, \end{cases}$

$h = solveLDE(g, f)$;
$print(h)$;
out: $h = (-8.00 + (-2.00b) + 2.00a)/4.00e^{-t} + (8.00 + 2.00b + 2.00a)/4.00e^{t} - 4.00t$.

```
SPACE=R64[t];
g=\systLDE(\d(y,t,2)-4\d(y,t)+5y=0);
f=\initCond(\d(y, t, 0, 0)=0, \d(y, t, 0, 1)=1);
h=\solveLDE(g, f);
\print(h);
```

Returns:
in: $SPACE = R64[t]$;
$g = y_t^{''} - 4y_t^{'} + 5y = 0$;
$f = \begin{cases} y_{t=0} = 0, \\ y_{t=0}^{'} = 1, \end{cases}$
$h = solveLDE(g, f)$;
$print(h)$;
out: $h = 0.53i(e^{(2.05-0.95i)t}) - 0.53i(e^{(2.05+0.95i)t})$.

```
SPACE=R64[t];
g=\systLDE(\d(y,t,2)-\d(y,t)-6y=2);
f=\initCond(\d(y, t, 0, 0)=1, \d(y, t, 0, 1)=0);
h=\solveLDE(g, f);
\print(h);
```

Returns:
in: $SPACE = R64[t]$;
$g = y_t^{''} - y_t^{'} - 6y = 2$;
$f = \begin{cases} y_{t=0} = 1, \\ y_{t=0}^{'} = 0, \end{cases}$
$h = solveLDE(g, f)$;
$print(h)$;
out: $h = 0.53e^{3.00t} + 0.80e^{-2.00t} - 0.33$.

```
SPACE=R64[t];
g=\systLDE(\d(y,t,2)-9y=2-t);
f=\initCond(\d(y, t, 0, 0)=0, \d(y, t, 0, 1)=1);
h=\solveLDE(g, f);
\print(h);
```

Returns:

in: $SPACE = R64[t]$;

$g = y_t^{''} - 9y = 2 - t$;

$f = \begin{cases} y_{t=0} & = & 0, \\ y_{t=1}^{'} & = & b, \end{cases}$

$h = solveLDE(g, f)$;

$print(h)$;

out: $h = 0.26e^{3.00t} + 0.04e^{-3.00t} + 0.11t - 0.22$.

7.3. Solution of systems of differential equations

Procedure of solving a system of differential equations (SDE) consists of four parts.

1. To set the ring ($SPACE$).
2. To set a system of equations (**systLDE**).
3. To set initial conditions (**initCond**).
4. To get solution of SDE (**solveLDE**).

Examples.

```
SPACE=R64[t];
g=\systLDE(3\d(x, t)+2x+\d(y, t)=1, \d(x, t)+4\d(y, t)+3y=0);
f=\initCond(\d(x, t, 0, 0)=0, \d(x, t, 0, 1)=0,
            \d(y, t, 0, 0)=0, \d(y, t, 0, 1)=0);
h=\solveLDE(g, f);
```

Returns:

in: $SPACE = R64[t]$;

$g = \begin{cases} 3x'_t + 2x + y'_t & = & 1, \\ x'_t + 4y'_t + 3y & = & 0, \end{cases}$

$f = \begin{cases} x_{t=0} & = & 0, \\ x_{t=0}^{'} & = & 0, \\ y_{t=0} & = & 0, \\ y_{t=0}^{'} & = & 0, \end{cases}$

$h = solveLDE(g, f)$;

out: $h = [0.50 + (-0.30e^{-0.55t}) + (-0.20e^{-t}), (-0.20e^{-0.55t}) + 0.20e^{-t}]$;

In the following example, the option STEPBYSTEP = 1, gives the output of all intermediate calculations that are needed to solve this

system of differential equations. Note that it does not use the command $print()$.

```
SPACE=R64[t];
STEPBYSTEP=1;
g=\systLDE(3\d(x, t)+2x+\d(y, t)=1, \d(x, t)+4\d(y, t)+3y=0);
f=\initCond(\d(x, t, 0, 0)=0, \d(x, t, 0, 1)=0,
            \d(y, t, 0, 0)=0, \d(y, t, 0, 1)=0);
h=\solveLDE(g, f);
```

Returns:

in: $SPACE = R64[t]$;

$STEPBYSTEP = 1$;

$$g = \begin{cases} 3x'_t + 2x + y'_t &= 1, \\ x'_t + 4y'_t + 3y &= 0, \end{cases}$$

$$f = \begin{cases} x_{t=0} &= 0, \\ x'_{t=0} &= 0, \\ y_{t=0} &= 0, \\ y'_{t=0} &= 0, \end{cases}$$

$h = solveLDE(g, f)$;

out: $h = [0.50 + (-0.30e^{-0.55t}) + (-0.20e^{-t}), (-0.20e^{-0.55t}) + 0.20e^{-t}]$;

Solve this system of differential equations on the accuracy e.

```
SPACE=R[t];
e=0.00000001;
g=\systLDE(3\d(x, t)+2x+\d(y, t)=1, \d(x, t)+4\d(y, t)+3y=0);
f=\initCond(\d(x, t, 0, 0)=0, \d(x, t, 0, 1)=0,
            \d(y, t, 0, 0)=0, \d(y, t, 0, 1)=0);
h=\solveLDE(g, f, e);
```

Returns:

in: $SPACE = R[t]$;

$e = 0.00000001$;

$$g = \begin{cases} 3x'_t + 2x + y'_t &= 1, \\ x'_t + 4y'_t + 3y &= 0, \end{cases}$$

$$f = \begin{cases} x_{t=0} &= 0, \\ x'_{t=0} &= 0, \\ y_{t=0} &= 0, \\ y'_{t=0} &= 0, \end{cases}$$

$h = solveLDE(g, f, e);$
out: $h = [0.50 + (-0.30e^{-0.55t}) + (-0.20e^{-t}), (-0.20e^{-0.55t}) + 0.20e^{-t}];$

The graphics solve this system of differential equations on the accuracy e.

```
SPACE=R[t];
e=0.00000001;
g=\systLDE(3\d(x, t)+2x+\d(y, t)=1, \d(x, t)+4\d(y, t)+3y=0);
f=\initCond(\d(x, t, 0, 0)=0, \d(x, t, 0, 1)=0,
            \d(y, t, 0, 0)=0, \d(y, t, 0, 1)=0);
h=\solveLDE(g, f, e);
p=\plot(h,[-10,10,-10,10]);
```

Returns:
in: $SPACE = R[t];$
$e = 0.00000001;$

$$g = \begin{cases} 3x'_t + 2x + y'_t &= 1, \\ x'_t + 4y'_t + 3y &= 0, \end{cases}$$

$$f = \begin{cases} x_{t=0} &= 0, \\ x'_{t=0} &= 0, \\ y_{t=0} &= 0, \\ y'_{t=0} &= 0, \end{cases}$$

$h = solveLDE(g, f, e);$ $p = plot(h, [-10, 10, -10, 10]);$
out: $h = [0.50 + (-0.30e^{-0.55t}) + (-0.20e^{-t}), (-0.20e^{-0.55t}) + 0.20e^{-t}];$

The graphics solve this system of differential equations.

```
SPACE=R[t];
g=\systLDE(3\d(x, t)+2x+\d(y, t)=1, \d(x, t)+4\d(y, t)+3y=0);
f=\initCond(\d(x, t, 0, 0)=a, \d(x, t, 0, 1)=b,
            \d(y, t, 0, 0)=c, \d(y, t, 0, 1)=d);
h=\solveLDE(g, f, 1);
p=\plot(h,[-10,10,-10,10]);
```

Returns:
in: $SPACE = R[t];$

$$g = \begin{cases} 3x'_t + 2x + y'_t &= 1, \\ x'_t + 4y'_t + 3y &= 0, \end{cases}$$

$$f = \begin{cases} x_{t=0} &= a, \\ x'_{t=0} &= b, \\ y_{t=0} &= c, \\ y'_{t=0} &= d, \end{cases}$$

$h = solveLDE(g, f, 1); \qquad p = plot(h, [-10, 10, -10, 10]);$

out: $h = [0.50 + (-0.30e^{-0.55t}) + (-0.20e^{-t}), (-0.20e^{-0.55t}) + 0.20e^{-t}];$

```
SPACE=R64[t];
g=\systLDE(\d(x, t)+x-2y=0, \d(y, t)+x+4y=0);
f=\initCond(\d(x, t, 0, 0)=1, \d(y, t, 0, 0)=1);
h=\solveLDE(g, f);
```

Returns:

in: $SPACE = R64[t];$

$$g = \begin{cases} x'_t + x - 2y &= 0, \\ y'_t + x + 4y &= 0, \end{cases}$$

$$f = \begin{cases} x_{t=0} &= 1, \\ y_{t=0} &= 1, \end{cases}$$

$h = solveLDE(g, f);$

out: $h = [(4.0e^{-2.0t} + (-3.0)e^{-3.0t}, (-2.0)e^{-2.0t} + 3.0e{-}3.0t];$

```
SPACE=R64[t];
g=\systLDE(\d(x, t)+2x+2y=10\exp(2t), \d(y, t)-2x+y=7\exp(2t));
f=\initCond(\d(x, t, 0, 0)=1, \d(y, t, 0, 0)=3);
h=\solveLDE(g, f);
```

Returns:

in: $SPACE = R64[t];$

$$g = \begin{cases} x'_t + 2x + 2y &= 10e^{2t}, \\ y'_t - 2x + y &= 7e^{2t}, \end{cases}$$

$$f = \begin{cases} x_{t=0} &= 1, \\ y_{t=0} &= 1, \end{cases}$$

$h = solveLDE(g, f);$

out: $h = [e^{2.0t}, 3.0e^{2.0t}];$

```
SPACE=R64[t];
g=\systLDE(\d(x, t)-y+z=0, -x-y+\d(y, t)=0, -x-z+\d(z, t)=0);
f= \initCond(\d(x, t, 0, 0)=1, \d(y, t, 0, 0)=2,
             \d(z, t, 0, 0)=3);
```

```
h= \solveLDE(g, f);
\print(h);
```

Returns:
in: $SPACE = R64[t]$;

$$g = \begin{cases} {x'}_t - y + z &= 0, \\ -x - y + {y'}_t &= 0, \\ -x - z + {z'}_t &= 0, \end{cases}$$

$$f = \begin{cases} x_{t=0} &= 1, \\ y_{t=0} &= 2, \\ z_{t=0} &= 3, \end{cases}$$

$h = solveLDE(g, f)$;
$print(h)$;
out: $h = [(-2.00) + 5.00e^t + (-1.00e^t)t, 2.00 + (-1.00e^t), (-2.00) + 4.00e^t + (-1.00e^t)t]$;

```
SPACE=R64[t];
g=\systLDE(\d(x, t, 2)+\d(x, t)-\d(y, t)=1,
          \d(x, t)+x-\d(y, t, 2)=1+4\exp(t));
f=\initCond(\d(x, t, 0, 0)=1, \d(x, t, 0, 1)=2,
            \d(y, t, 0, 0)=0, \d(y, t, 0, 1)=1);
h= \solveLDE(g, f);
\print(h);
```

Returns:
in: $SPACE = R64[t]$;

$$g = \begin{cases} {x''}_t + {x'}_t - {y'}_t &= 1, \\ {x'}_t + x - {y''}_t &= 1 + 4e^t, \end{cases}$$

$$f = \begin{cases} x_{t=0} &= 1, \\ {x'}_{t=0} &= 2, \\ y_{t=0} &= 0, \\ {y'}_{t=0} &= 1, \end{cases}$$

$h = solveLDE(g, f)$;
$print(h)$;
out: $h = [1.00+2.00e^t+(-1.00e^t)t+(-2.00e^{-t})+(-1.00e^{-t})t, (-2.00)+(-1.00t) + 3.00e^t + (-2.00e^t)t + (-1.00e^{-t})]$;

```
SPACE=R64[t];
g=\systLDE(\d(x, t)+3x-4y=9\exp(2t),
```

```
            2x+\d(y, t)-3y=3\exp(2t));
f=\initCond(\d(x, t, 0, 0)=2, \d(y, t, 0, 0)=0);
h=\solveLDE(g, f);
\print(h);
```

Returns:
in: $SPACE = R64[t]$;

$$g = \begin{cases} x'_t + 3x - 4y &= 9e^{2t}, \\ 2x + y'_t - 3y &= 3e^{2t}, \end{cases}$$

$$f = \begin{cases} x_{t=0} &= 2, \\ y_{t=0} &= 0, \end{cases}$$

$h = solveLDE(g, f)$;
$print(h)$;
out: $h = [e^t + e^{2.00t}, e^t + (-1.00e^{2.00t})]$;

```
SPACE=R64[t];
g=\systLDE(\d(x, t)-x-2y=0, \d(y, t)-2x-y=1);
f=\initCond(\d(x, t, 0, 0)=0, \d(y, t, 0, 0)=5);
h=\solveLDE(g, f);
\print(h);
```

Returns:
in: $SPACE = R64[t]$;

$$g = \begin{cases} x'_t - x - 2y &= 0, \\ y'_t - 2x - y &= 1, \end{cases}$$

$$f = \begin{cases} x_{t=0} &= 0, \\ y_{t=0} &= 5, \end{cases}$$

$h = solveLDE(g, f)$;
$print(h)$;
out: $h = [(-0.67) + 0.17e^{3.00t} + 0.50e^{-t}, 0.33 + (-0.50e^{-t}) + 0.17e^{3.00t}]$;

```
SPACE=R64[t];
g=\systLDE(\d(x, t, 2)+\d(y, t)+y=\exp(t)-t,
           \d(x, t)-x+2\d(y, t, 2)-y=-\exp(-t));
f=\initCond(\d(x, t, 0, 0)=1, \d(x, t, 0, 1)=2,
            \d(y, t, 0, 0)=0, \d(y, t, 0, 1)=0);
h=\solveLDE(g, f);
\print(h);
```

Returns:

in: $SPACE = R64[t]$;

$$g = \begin{cases} x''_t + y'_t + y &= e^t - t, \\ x'_t - x + 2y''_t - y &= -e^{-t}, \end{cases}$$

$$f = \begin{cases} x_{t=0} &= 1, \\ x'_{t=0} &= 2, \\ y_{t=0} &= 0, \\ y'_{t=0} &= 0, \end{cases}$$

$h = solveLDE(g, f)$;

$print(h)$;

out: $h = [1.00t + e^t, 1.00 + (-1.00t) + (-1.00e^{-t})]$;

```
SPACE=R64[t];
g=\systLDE(\d(x, t, 2)+\d(y, t)=\sh(t)-\sin(t)-t,
           \d(y, t, 2)+\d(x, t)=\ch(t)-\cos(t));
f=\initCond(\d(x, t, 0, 0)=0, \d(x, t, 0, 1)=2,
            \d(y, t, 0, 0)=1, \d(y, t, 0, 1)=0);
h=\solveLDE(g, f);
\print(h);
```

Returns:

in: $SPACE = R64[t]$;

$$g = \begin{cases} x''_t + y'_t &= \mathbf{sh}(t) - sin(t) - t, \\ y''_t + x'_t &= \mathbf{ch}(t) - cos(t), \end{cases}$$

$$f = \begin{cases} x_{t=0} &= 0, \\ x'_{t=0} &= 2, \\ y_{t=0} &= 1, \\ y'_{t=0} &= 0, \end{cases}$$

$h = solveLDE(g, f)$;

$print(h)$;

out: $h = [1.00t + 0.50e^t + (-0.50e^{-t}), (-1.00tt^2)/2.00 + 0.50e^{1.00\mathbf{i}t} + (0.50e^{-1.00\mathbf{i}t})]$;

```
SPACE=R64[t];
g=\systLDE(\d(x, t, 2)-\d(x, t)+\d(y, t)=\exp(-t)+\cos(t),
           \d(x, t)-\d(y, t, 2)-\d(y, t)=2\exp(t)+\sin(t));
f=\initCond(\d(x, t, 0, 0)=2, \d(x, t, 0, 1)=1,
            \d(y, t, 0, 0)=0, \d(y, t, 0, 1)=1);
```

```
h=\solveLDE(g, f);
\print(h);
```

Returns:
in: $SPACE = R64[t]$;

$$g = \begin{cases} x''_t - x'_t + y'_t &= e^{-t} + cos(t), \\ x'_t - y''_t - y'_t &= 2e^t + sin(t), \end{cases}$$

$$f = \begin{cases} x_{t=0} &= 2, \\ x'_{t=0} &= 1, \\ y_{t=0} &= 0, \\ y'_{t=0} &= 1, \end{cases}$$

$h = solveLDE(g, f)$;
$print(h)$;
out: $h = [0.50e^{1.00\mathbf{it}} + 0.50e^{-1.00\mathbf{it}} + (-1.00e^{-t}), 0.50\mathbf{i}(e^{1.00\mathbf{it}}) + (-0.50\mathbf{i}(e^{-1.00\mathbf{it}})) + (2.00e^t)]$.

```
SPACE=R64[t];
g=\systLDE(\d(x, t)-y+z=0, -x-y+\d(y, t)=0, -x-z+\d(z, t)=0);
f= \initCond(\d(x, t, 0, 0)=a, \d(y, t, 0, 0)=b,
              \d(z, t, 0, 0)=c);
h= \solveLDE(g, f);
\print(h);
```

Returns:
in: $SPACE = R64[t]$;

$$g = \begin{cases} x'_t - y + z &= 0, \\ -x - y + y'_t &= 0, \\ -x - z + z'_t &= 0, \end{cases}$$

$$f = \begin{cases} x_{t=0} &= a, \\ y_{t=0} &= b, \\ z_{t=0} &= c, \end{cases}$$

$h = solveLDE(g, f)$;
$print(h)$;
out: $h = [b - a - c + (-b + a + 2.00c)e^t + (b - c)e^t t, -b + c + a + (b - c)e^t, -c - a + b) + (c + a)e^t + (-c + b)e^t t]$;

```
SPACE=R64[t];
g=\systLDE(\d(y, t)+y-3x=0, -x-y+\d(x, t)=\exp(t));
f= \initCond(\d(x, t, 0, 0)=0, \d(y, t, 0, 0)=0);
```

```
h= \solveLDE(g, f);
\print(h);
```

Returns:
in: $SPACE = R64[t]$;
$g = \begin{cases} {y'}_t - y - 3x &= 0, \\ -x - y + {x'}_t &= e^t, \end{cases}$
$f = \begin{cases} x_{t=0} &= 0, \\ y_{t=0} &= 0, \end{cases}$
$h = solveLDE(g, f)$;
$print(h)$;
out: $h = [-e^t + 0.25e^{-2.00t} + 0.75e^{2.00t}, -0.08e^{-2.00t} + 0.75e^{2.00t} - 0.67e^t]$;

```
SPACE=R64[t];
g=\systLDE(\d(y, t)+\d(x, t)-x=\exp(t), 2\d(y, t)+\d(x, t)+2x=\cos(t)
f= \initCond(\d(x, t, 0, 0)=0, \d(y, t, 0, 0)=0);
h= \solveLDE(g, f);
\print(h);
```

Returns:
in: $SPACE = R64[t]$;
$g = \begin{cases} {y'}_t + {x'}_t - x &= e^t, \\ 2{y'}_t + {x'}_t + 2x &= cos(t), \end{cases}$
$f = \begin{cases} x_{t=0} &= 0, \\ y_{t=0} &= 0, \end{cases}$
$h = solveLDE(g, f)$;
$print(h)$;
out: $h = [(0.12 + 0.03i)e^{it} + (0.12 - 0.03i)e^{-it} - 0.67e^t + 0.43e^{4.00t}, -0.32e^{4.00t} - 0.50 + e^t + (-0.09 - 0.15i)e^{it} + (-0.09 + 0.15i)e^{-it}]$;

```
SPACE=R64[t];
g=\systLDE(\d(y, t)-y+x=1.5t^2, \d(x, t)+2x+4y=4t+1);
f= \initCond(\d(x, t, 0, 0)=0, \d(y, t, 0, 0)=0);
h= \solveLDE(g, f);
\print(h);
```

Returns:
in: $SPACE = R64[t]$;

$$g = \left\{ \begin{array}{rcl} {y'}_t - y + x & = & 1.5t^2, \\ {x'}_t + 2x + 4y & = & 4t + 1, \end{array} \right.$$
$$f = \left\{ \begin{array}{rcl} x_{t=0} & = & 0, \\ y_{t=0} & = & 0, \end{array} \right.$$
$h = solveLDE(g, f)$;
$print(h)$;
out: $h = [t + t^2, -0.5t^2]$;

```
SPACE=R64[t];
g=\systLDE(\d(y, t)+y-x-z=0, \d(x, t)-y+x-z=0, \d(z, t)-y-x-z=0);
f= \initCond(\d(x, t, 0, 0)=0, \d(y, t, 0, 0)=1,
             \d(z, t, 0, 0)=0);
h= \solveLDE(g, f);
\print(h);
```

Returns:
in: $SPACE = R64[t]$;
$$g = \left\{ \begin{array}{rcl} {y'}_t + y - x - z & = & 0, \\ {x'}_t - y + x - z & = & 0, \\ {z'}_t - y - x - z & = & 0, \end{array} \right.$$
$$f = \left\{ \begin{array}{rcl} x_{t=0} & = & 0, \\ y_{t=0} & = & 1, \\ z_{t=0} & = & 0, \end{array} \right.$$
$h = solveLDE(g, f)$;
$print(h)$;
out: $h = [0.33e^{-t} + 0.17e^{2.00t} + 0.5e^{-2.00t}, 0.33e^{2.00t} - 0.33e^{-t}, 0.17e^{2.00t} + 0.33e^{-t} - 0.5e^{-2.00t}]$;

```
SPACE=R64[t];
g=\systLDE(\d(y, t)-x+z=0, \d(x, t)+2y-x=0, \d(z, t)-2y+x=0);
f= \initCond(\d(x, t, 0, 0)=0, \d(y, t, 0, 0)=1,
             \d(z, t, 0, 0)=0);
h= \solveLDE(g, f);
\print(h);
```

Returns:
in: $SPACE = R64[t]$;
$$g = \left\{ \begin{array}{rcl} {y'}_t - x + z & = & 0, \\ {x'}_t + 2y - x & = & 0, \\ {z'}_t - 2y + x & = & 0, \end{array} \right.$$

$$f = \left\{ \begin{array}{rcl} x_{t=0} & = & 0, \\ y_{t=0} & = & 1, \\ z_{t=0} & = & 0, \end{array} \right.$$

$h = solveLDE(g, f)$;

$print(h)$;

out: $h = [-0.52ie^{(0.5+1.94i)t} + 0.52ie^{(0.5-1.94i)t}, (0.5+0.13i)e^{(0.5+1.94i)t} + (0.5 - 0.13i)e^{(0.5-1.94i)t}, 0.52ie^{(0.5+1.94i)t} - 0.52ie^{(0.5-1.94i)t}]$;

```
SPACE=R64[t];
g=\systLDE(\d(y, t, 2)+2x=0, \d(x, t, 2)-2y=0);
f=\initCond(\d(x, t, 0, 0)=0, \d(x, t, 0, 1)=0,
            \d(y, t, 0, 0)=0, \d(y, t, 0, 1)=1);
h=\solveLDE(g, f);
\print(h);
```

Returns:

in: $SPACE = R64[t]$;

$$g = \left\{ \begin{array}{rcl} y_t^{''} + 2x & = & 0, \\ x_t^{''} - 2y & = & 0, \end{array} \right.$$

$$f = \left\{ \begin{array}{rcl} x_{t=0} & = & 0, \\ x_{t=0}^{'} & = & 0, \\ y_{t=0} & = & 0, \\ y_{t=0}^{'} & = & 1, \end{array} \right.$$

$h = solveLDE(g, f)$;

$print(h)$;

out: $h = [(0.13 - 0.13i)e^{(1+i)t} + (0.13 + 0.12i)e^{(1-i)t} + (-0.12 - 0.13i)e^{(-1+i)t} + (-0.13+0.13i)e^{(-1-i)t}, (-0.13-0.13i)e^{(1+i)t} + (-0.13+0.13i)e^{(1-i)t} + (0.13 - 0.13i)e^{(-1+i)t} + (0.13 + 0.13i)e^{(-1-i)t}]$;

```
SPACE=R64[t];
g=\systLDE(\d(x, t)-8y+x=0, \d(y, t)-x-y=0);
f= \initCond(\d(x, t, 0, 0)=a, \d(y, t, 0, 0)=b);
h= \solveLDE(g, f);
\print(h);
```

Returns:

in: $SPACE = R64[t]$;

$$g = \left\{ \begin{array}{rcl} x'_t - 8y + x & = & 0, \\ y'_t - x - y & = & 0, \end{array} \right.$$

$f = \left\{ \begin{array}{rcl} x_{t=0} & = & a, \\ y_{t=0} & = & b, \end{array} \right.$

$h = solveLDE(g, f);$

$print(h);$

out: $h = [((4b + a)/3)e^{3.00t} + ((-4 * b + 2a)/3)e^{-3.00t}, ((-a + 2b)/6)e^{-3.00t} + ((a + 4b)/6)e^{3.00t}];$

```
SPACE=R64[t];
g=\systLDE(\d(x, t)+3x-4y=9(\exp(t))^2, \d(y, t)+2x-3y=3(\exp(t))^2);
f= \initCond(\d(x, t, 0, 0)=2, \d(y, t, 0, 0)=0);
h= \solveLDE(g, f);
\print(h);
```

Returns:

in: $SPACE = R64[t];$

$g = \left\{ \begin{array}{rcl} x'_t + 3x - 4y & = & 9(e^t)^2, \\ y'_t + 2x - 3y & = & 3(e^t)^2, \end{array} \right.$

$f = \left\{ \begin{array}{rcl} x_{t=0} & = & 2, \\ y_{t=0} & = & 0, \end{array} \right.$

$h = solveLDE(g, f);$

$print(h);$

out: $h = [e^t + e^{2.00t}, e^t - e^{2.00t}];$

```
SPACE=R64[t];
g=\systLDE(\d(x, t, 2)+\d(y, t)=\sh(t)-\sin(t)-t, \d(y, t, 2)-\d(x, t
f=\initCond(\d(x, t, 0, 0)=2, \d(x, t, 0, 1)=0,
            \d(y, t, 0, 0)=0, \d(y, t, 0, 1)=1);
h=\solveLDE(g, f);
\print(h);
```

Returns:

in: $SPACE = R64[t];$

$g = \left\{ \begin{array}{rcl} x''_t + y'_t & = & sh(t) - sin(t) - t, \\ y''_t - x'_t & = & ch(t) - cos(t), \end{array} \right.$

$f = \left\{ \begin{array}{rcl} x_{t=0} & = & 2, \\ x'_{t=0} & = & 0, \\ y_{t=0} & = & 0, \\ y'_{t=0} & = & 1, \end{array} \right.$

$h = solveLDE(g, f);$

$print(h)$;
out: $h = [1 - t + (0.5 - 0.5i)e^{it} + (0.5 + 0.5i)e^{-it}, -1 - 0.5t^2 - 0.5ie^{it} + 0.5ie^{-it} + 0.5e^{t} + 0.5e^{-t}]$;

```
SPACE=R64[t];
g=\systLDE(\d(x, t)+5y-4x=0, \d(y, t)-x=0);
f= \initCond(\d(x, t, 0, 0)=0, \d(y, t, 0, 0)=1);
h= \solveLDE(g, f);
\print(h);
```

Returns:
in: $SPACE = R64[t]$;

$$g = \begin{cases} {x'}_t + 5y - 4x &= 0, \\ {y'}_t - x &= 0, \end{cases}$$

$$f = \begin{cases} x_{t=0} &= 0, \\ y_{t=0} &= 1, \end{cases}$$

$h = solveLDE(g, f)$;
$print(h)$;
out: $h = [(0.5 + i)e^{(2+i)t} + (0.5 - i)e^{(2-i)t}, 2.5ie^{(2+i)t} - 2.5ie^{(2-i)t}]$;

7.4. LaplaceTransform and InverseLaplaceTransform

```
SPACE=R64[t];
L=\laplaceTransform(\exp(3t));
\print(L);
```

Returns:
in: $SPACE = R64[t]$;
$L = laplaceTransform(exp(3t))$.
$print(L)$;
out: $L = \frac{1.0}{t-3.0}$

```
SPACE=R64[t];
L=\inverseLaplaceTransform(1/(t-3));
\print(L);
```

7.5. Calculation of the characteristics of dynamic objects and systems

To find the transfer function of the object, you must perform the following steps:

1. Specify the space variables ($SPACE$).
2. Ask equation input - x.
3. Ask output equation - y.
4. Obtain a solution (**solveWFDS**).

Examples.

```
SPACE = R64[t];
f = \d(y,t,2)+2\d(y,t);
g = 3x;
h = \solveTransferFunction(g,f);
\print(h);
\set2D(-10, 10, -10, 10,'p','W(p)','Transfer function');
p=\plot(h);
```

Returns:

in: $SPACE = R64[t]$;
$f = y_t'' + 2y_t''$;
$g = 3x$;
$h = solveTransferFunction(g, f)$;
$print(h)$;
$set2D(-10, 10, -10, 10, 'p', 'W(p)', 'Transferfunction')$;
$p = plot(h)$;
out: $h = [3.0/(p^2 + 2.0p)]$;

To find the temporal characteristics of the object, perform the following steps:

1. Specify the space variables ($SPACE$).
2. Ask equation input - x.
3. Ask output equation - y.
4. Obtain a solution (**solveTPDS**).

```
SPACE = R64[t];
f = \d(y,t,2)+2\d(y,t);
g = 3x;
h = \solveTimeResponse(g,f);
```

```
\print(h);
\set2D(-10, 10, -10, 10,'p','k(p),h(p)','Temporal characteristics');
p=\plot(h);
```

Returns:
in: $SPACE = R64[t]$;
$f = y_t^{''} + 2y_t^{''}$;
$g = 3x$;
$h = solveTimeResponse(g, f)$;
$print(h)$;
$set2D(-10, 10, -10, 10,' p',' k(p), h(p)',' Temporalcharacteristics')$;
$p = plot(h)$;
out: $h = [(1.5exp(2.0p) * 3.0 + (-1.5) * 3.0), ((-0.75) + (-1.5p) + 0.75exp(2p))]$;

To find the frequency characteristics of the object, you must perform the following steps:

1. Specify the space variables ($SPACE$).
2. Ask equation input - x.
3. Ask output equation - y.
4. Obtain a solution (**solveCHDS**).

```
SPACE = R64[t];
f = \d(y,t,2)+2\d(y,t);
g = 3x;
h = \solveFrequenceResponse(g,f);
SPACE = R64[j,p];
\print(h);
```

Returns:
in: $SPACE = R64[t]$;
$f = y_t^{''} + 2y_t^{''}$;
$g = 3x$;
$h = solveFrequenceResponse(g, f)$;
$SPACE = R64[j, p]$;
$print(h)$;
out: $h = [3.0/(p^2j^2+2.0pj), sqrt9.0/(p^4 + 4.0p^2), arctg(sqrt2/p), 20.0lg(sqrt9.0/(p^4$

Returns:
in: $SPACE = R64[t]$;

$L = inverseLaplaceTransform(1/(t-3))$.
$print(L)$;
out: $L = exp(3t)$

Chapter 8

Polynomial computations

8.1. Calculation of the value of a polynomial at the point

To calculate the value of a function at the point you must run **value**$(f, [var1, var2, \ldots, varn])$, where f — is a polynomial in which the variables are replaced by the corresponding values of $var1, var2, \ldots, varn$.

Example.

```
SPACE=R[x, y];
f=x^2+5x(y^3+x);
g=\value(f, [1, 2]);
\print(g);
```

Returns:
in: $SPACE = R[x, y]$;
$f = x^2 + 5x(y^3 + x)$;
$g = value(f, [1, 2])$;
$print(g)$;
out: $g = 46.00$.

8.2. Factorization of polynomials. Bringing polynomials to the standard form.

Polynomials are automatically brought to the standard form, which assumes, that the first written the leading monomials, and then younger. Recall that the order of variables in the ring determines seniority of variables.

To bring to the standard form of a polynomial can also be run **expand**(f), where f — is a polynomial.

For factoring polynomials you must execute the command **factor**(f), where f — it is a polynomial.

Example.

```
SPACE=Q[x, y];
f= (y^3+x)^2(x+1)^3;
 g=\expand(f);
h=\factor(g);
\print(g,h);
```

Returns:
in: $SPACE = Q[x, y]$;
$f = (y^3 + x)^2(x + 1)^3$;
$g = expand(f)$;
$h = factor(g)$;
$print(g, h)$;
out: $g = y^6x^3 + 3y^6x^2 + 3y^6x + y^6 + 2y^3x^4 + 6y^3x^3 + 6y^3x^2 + 2y^3x + x^5 + 3x^4 + 3x^3 + x^2$;
$h = (x + 1)^3(y^3 + x)^2$;

8.3. Geometric progression. Summation of polynomial with respect to the variables

For the summation of polynomial with respect to the variables we must run **SumOfPol**$(f, [x, y], [x1, x2, y1, y2])$, where f — a polynomial, x, y — variables for summation, $x1, x2$ — range of summation over x, $y1, y2$ — range of summation over y.

If intervals of summation on all variables coincide then we can write **SumOfPol**$(f, [x, y], [x1, x2])$, where $x1, x2$ — summation interval for x and y.

Example.

```
SPACE=R[x, y, z];
f=x^2z+xy+y^3xz;
res=\SumOfPol(f, [x, y], [2, 4, -2, 3]);
\print(res);
```

Returns:

in: $SPACE = R[x, y, z]$;
$f = x^2z + xy + y^3xz$;
$res = SumOfPol(f, [x, y], [2, 4, -2, 3])$;
$print(res)$;

out: $res = 417.00z + 27.00$.

To convert a polynomial using the formula of the sum of a geometric progression, you must run **SearchOfProgression**(f). This command searches for a geometric progression with the largest number of members in this polynomial. Then do it again for the remaining members, and so on. All the detected progression be written as $S_n = b_1(q^n - 1)/(q - 1)$, where S_n — sum of the first n members, b_1 — the first term of a geometric progression, q — the geometric ratio.

Example.

```
SPACE=R[x, y, z];
f=x^3+x^4+x^5+x^6+x^7+x^8+x^9+x^10+x^11+x^12+x^13;
g=x+x^5+x^9+x^13+xyz+7x^2y^2z^2+7x^3y^3z^3+100xy+x+x^2+x^3+x^4;
f1=\SearchOfProgression(f);
g1=\SearchOfProgression(g);
\print(f1, g1);
```

Returns:

in: $SPACE = R[x, y, z]$;
$f = x^3 + x^4 + x^5 + x^6 + x^7 + x^8 + x^9 + x^{10} + x^{11} + x^{12} + x^{13}$;
$g = x+x^5+x^9+x^13+xyz+7x^2y^2z^2+7x^3y^3z^3+100xy+x+x^2+x^3+x^4$;
$f1 = SearchOfProgression(f)$;
$g1 = SearchOfProgression(g)$;
$print(f1, g1)$;

out: $f1 = (x^{14} - x^3)/(x - 1)$;

$g1 = ((100.00yx + x^{13} + x^9 + x) + (z^4y^4x^4 - zyx)/(zyx - 1) + (x^6 - x)/(x - 1) + 6.00z^3y^3x^3 + 6.00z^2y^2x^2)$.

8.4. Groebner basis of polynomial ideal

The Groebner basis of polynomial ideal may be obtained due to the Bruno Buchberger algorithm **groebnerB**(). You can obtain the same basis using matrix algorithm, which similar to F4 algorithm **groebnerB**(). We use reverse lexicographical order. The order of the variables defined in the command SPACE,

Examples.

```
SPACE=Q[x, y, z];
b=\groebnerB(x^4y^3+2xy^2+3x+1, x^3y^2+x^2, x^4y+z^2+xy^4+3);
\print(b);
```

Returns:
in: $SPACE = Q[x, y, z]$;
$b = groebnerB(x^4y^3 + 2xy^2 + 3x + 1, x^3y^2 + x^2, x^4y + z^2 + xy^4 + 3)$;
$print(b)$;
out: $b = [z^2 - x^4 + 3x^2 + (-10)x + 9, y + (-9)x^4 + (-3)x^3 - x^2 + (-81)x + 27, x^5 + 9x^2 + (-6)x + 1]$;

```
SPACE=Z[x, y, z];
b=\groebner(x^4y^3+2xy^2+3x+1, x^3y^2+x^2, x^4y+z^2+x y^4+3);
\print(b);
```

Returns:
in: $SPACE = Q[x, y, z]$;
$b = groebner(x^4y^3 + 2xy^2 + 3x + 1, x^3y^2 + x^2, x^4y + z^2 + xy^4 + 3)$;
$print(b)$;
out: $b = [z^2 - x^4 + 3x^2 + (-10)x + 9, x^5 + 9x^2 + (-6)x + 1, y + (-9)x^4 + (-3)x^3 - x^2 + (-81)x + 27]$.

8.5. Calculations in quotient ring of ideal

reduceByGB$(f, [g_1, \ldots, g_N])$ function reduces polynomial p with given set of polynomial $g_1, \ldots, g_N$.

```
SPACE = Q[x, y, z];
p = \reduceByGB(5y^2 + 3x^2 + z^2, [y + x, 5z^2 + 5z]);
```

Returns:
in: $SPACE = Q[x, y, z]$;
$p = \backslash reduceByGB(5y^2 + 3x^2 + z^2, [y + x, 5z^2 + 5z])$;
$-z + 8x^2$;
out:

In case when second argument isn't a reduced Groebner basis result depends on positions of polynomials in array: among all potential reductors the first one will be chosen.

```
SPACE = Q[x, y];
NotGB1 = [x + y, x^2 + y^2];
imForNotGBset1 = \reduceByGB(x^2 + y^2, NotGB1);
NotGB2 = [x^2 + y^2, x + y];
imForNotGBset2 = \reduceByGB(x^2 + y^2, NotGB2);
GB = \groebner(x+y, x^2+y^2);
imForGB  = \reduceByGB(x^2 + y^2, GB);
\print(GB, imForNotGBset1, imForNotGBset2, imForGB);
```

Returns:
in: $SPACE = Q[x, y]$;
$NotGB1 = [x + y, x2 + y2]$;
$imForNotGBset1 = reduceByGB(x2 + y2, NotGB1)$;
$NotGB2 = [x2 + y2, x + y]$;
$imForNotGBset2 = reduceByGB(x2 + y2, NotGB2)$;
$GB = groebner(x + y, x2 + y2)$;
$imForGB = reduceByGB(x2 + y2, GB)$;
$print(GB, imForNotGBset1, imForNotGBset2, imForGB)$;
out: $GB = [y + x, x^2]$;
$imForNotGBset1 = 2x^2$;
$imForNotGBset2 = 0$;
$imForGB = 0$;

8.6. Solution of systems of nonlinear algebraic equations

To obtain the solution of the polynomial system

$p_1 = 0$,
$p_2 = 0$,
...
$p_N = 0$,
use the command **solveNAE**$(p_1, p_2, \ldots, p_N)$.

```
SPACE = R[x, y];
\solveNAE(x^2 + y^2 - 4, y - x^2);

SPACE = R[a, b, c];
S = \solveNAE(a + b + c, a b + a c + b c, a b c - 1);
```

8.7. Other polynomial functions

For polynomials in several variables (f, g), you can calculate GCD, LCM, a resultant (as the determinant of their Sylvester matrix), a discriminant:

GCD (f, g),
LCM (f, g),
resultant (f, g),
discriminant (f).

In this case, the main variable is the highest (last) variable, which is defined in the statement SPACE.

Example.

```
PACE=Z[q,r,s,x];
p=x^4+q*x^2+r*x+s;
\discriminant(p);
```

Returns:
in: $256s^3 - 128s^2q^2 + 144sr^2q + 16sq^4 - 27r^4 - 4r^2q^3$

out: $256s^3 - 128s^2q^2 + 144sr^2q + 16sq^4 - 27r^4 - 4r^2q^3$

Chapter 9

Matrix functions

9.1. Calculation of the transposed matrix

To compute the transpose of the matrix A must run **transpose**(A) or $\mathbf{A\hat{}\{T\}}$.

Example.

```
SPACE=Z[x];
A=[[1, 2], [4, 5]];
B=A^{T};
\print(B);
```

Returns:
in: $SPACE = Z[x]$;
$A = \begin{pmatrix} 1 & 2 \\ 4 & 5 \end{pmatrix}$;
$B = A^T$;
$print(B)$;
out: $B = \begin{pmatrix} 1 & 4 \\ 2 & 5 \end{pmatrix}$.

9.2. Getting the dimensions of a matrix and a vector

You can get the number of rows, the number of columns, or both dimensions of a matrix. To do this, you need to execute one of the com-

mands **rowNumb**(A), **colNumb**(A), or **size**(A). The same commands can be used for both a vector row and a vector column.

Example.

```
SPACE=Z[x];
A=[[1, 2,1], [4, 5,9]];
r=\rowNumb(A); c=\colNumb(A); s = \size(A);
\print(r,c,s);
```

Returns:

in: $SPACE = Z[x]$;

$A = \begin{pmatrix} 1 & 2 & 1 \\ 4 & 5 & 9 \end{pmatrix}$;

$r = rowNumb(A); c = colNumb(A); s = size(A)$;

$print(r, c, s)$;

out: $r = 2 \quad c = 3 \quad s = [2, 3]$

9.3. The calculation of adjoint and inverse matrices

9.3.1. The calculation of inverse matrix

To calculate the inverse matrix for the matrix A, to execute **inverse**(A) or **A^{(−1)}**.

Examples.

```
SPACE=Z[x];
A=[[1, 4], [4, 5]];
B=\inverse(A);
\print(B);
```

Returns:

in: $SPACE = Z[x]$;

$A = \begin{pmatrix} 1 & 2 \\ 4 & 5 \end{pmatrix}$;

$B = inverse(A)$;

$print(B)$;

out: $B = \begin{pmatrix} (-5)/11 & 4/11 \\ 4/11 & (-1)/11 \end{pmatrix}$;

```
SPACE=Z[x, y];
A=[[x+y, x], [y, \cos(x)]];
B=\inverse(A);
\print(B);
```

Returns:

in: $SPACE = Z[x, y]$;

$A = \begin{pmatrix} x+y & x \\ y & cos(x) \end{pmatrix}$;

$B = inverse(A)$;

$print(B)$;

out: $B = \begin{pmatrix} \frac{\cos(x)}{y\cos(x)+x\cos(x)+(-yx)} & \frac{-x}{y\cos(x)+x\cos(x)+(-yx)} \\ \frac{-y}{(y\cos(x)+x\cos(x)+(-yx)} & y + \frac{x}{(y\cos(x)+x\cos(x)+(-yx)} \end{pmatrix}$.

9.3.2. Calculation of adjoint matrix

To calculate the adjoint matrix for a given matrix A execute **adjoint**(A) or **A**^{**star**}.

Examples.

```
SPACE=Z[x];
A=[[1, 4], [4, 5]];
B=\adjoint(A);
\print(B);
```

Returns:

in: $SPACE = Z[x]$;

$A = \begin{pmatrix} 1 & 2 \\ 4 & 5 \end{pmatrix}$;

$B = adjoint(A)$;

$print(B)$;

out: $B = \begin{pmatrix} 5 & -4 \\ -4 & 1 \end{pmatrix}$;

```
SPACE=Z[x, y];
A=[[\cos(y), \sin(x)], [\sin(y), \cos(x)]];
B=\adjoint(A);
\print(B);
```

Returns:
in: $SPACE = Z[x, y]$;
$A = \begin{pmatrix} cos(y) & sin(x) \\ sin(y) & cos(x) \end{pmatrix}$;
$B = adjoint(A)$;
$print(B)$;
out: $B = \begin{pmatrix} cos(x) & -sin(x) \\ -sin(y) & cos(y) \end{pmatrix}$.

9.4. Calculation of the matrix determinant and rank

To calculate the rank of a matrix A, you must run **rank**(A), to calculate its determinant, you must run **det**(A).

Examples.

```
SPACE=Z[x];
A=[[1, 4], [4, 5]];
B=\det(A); r=\rank(A);
\print(B,r);
```

Returns:
in: $SPACE = Z[x]$;
$A = \begin{pmatrix} 1 & 4 \\ 4 & 5 \end{pmatrix}$;
$B = det(A); r = rank(A)$;
$print(B, r)$;
out: $B = -11$; $r = 2$;

```
SPACE=R[x];
A=[[3, 4], [3, 1]];
B=\det(A);
\print(B);
```

Returns:
in: $SPACE = R[x]$;
$A = \begin{pmatrix} 3 & 4 \\ 3 & 1 \end{pmatrix}$;
$B = det(A)$;

$print(B)$;
out: $B = -9$;

```
SPACE=Z[x, y];
A=[[x^2, y], [4, x+y]];
B=\det(A);
\print(B);
```

Returns:
in: $SPACE = Z[x, y]$;
$A = \begin{pmatrix} x^2 & y \\ 4 & x+y \end{pmatrix}$;
$B = det(A)$;
$print(B)$;
out: $B = yx^2 - 4y + x^3$;

```
SPACE=Z[x, y];
A=[[x+y, \sin(x)], [y, \cos(x)]];
B=\det(A);
\print(B);
```

Returns:
in: $SPACE = Z[x, y]$;
$A = \begin{pmatrix} x+y & sin(x) \\ y & cos(x) \end{pmatrix}$;
$B = det(A)$;
$print(B)$;
out: $B = y \cdot cos(x) + x \cdot cos(x) - y \cdot sin(x)$.

9.5. Calculation of the conjugate matrix

To calculate the conjugate matrix, you must run **conjugate**(A) or **A**^{**ast**}.

Example.

```
SPACE=C[x];
A=[[1+\i, 2-\i], [-3, -2\i]];
B=A^{\ast};
\print(B);
```

Returns:
in: $SPACE = C[x]$;
$A = \begin{pmatrix} 1+i & 2-i \\ -3 & -2i \end{pmatrix}$;
$B = A^*$;
$print(B)$;
out: $B = \begin{pmatrix} 1-1.0i & -3 \\ 2+1.0i & 2.0i \end{pmatrix}$.

9.6. Computing SVD-decomposition

To calculate the SVD-decomposition of a matrix, you must execute the command **SVD**(A). As a result, three matrices $[U, D, V]$ will be calculated. The matrices U, V are unitary, the matrix D is diagonal: $A = UDV$.

Example.

```
SPACE = R64[];
A = [[2,3,4], [1,3,3], [2,4,3]];
B = \SVD(A);
\print(B);
```

9.7. Calculation of the generalized inverse matrix

To compute the generalized inverse Moore-Penrose matrix must run **genInverse**(A) or **A**^{+}.

Example.

```
SPACE=Z[x];
A=[[1, 4, 5], [2, 4, 5]];
B=A^{+};
\print(B);
```

Returns:
in: $SPACE = Z[x]$;
$A = \begin{pmatrix} 1 & 4 & 5 \\ 2 & 4 & 5 \end{pmatrix}$;
$B = A^{+}$;

$print(B)$;

out: $B = \begin{pmatrix} -1 & 1 & 0 \\ 8/41 & (-4)/41 & 0 \\ 10/41 & (-5)/41 & 0 \end{pmatrix}$.

9.8. Computation of the kernel and echelon form

9.8.1. Computation of the echelon form

To compute the echelon form of the matrix A, you should run **toEchelonForm**(A).

Examples.

```
SPACE=Z[x];
A=[[1, 4], [4, 5]];
B=\toEchelonForm(A);
\print(B);
```

Returns:

in: $SPACE = Z[x]$;

$A = \begin{pmatrix} 1 & 4 \\ 4 & 5 \end{pmatrix}$;

$B = toEchelonForm(A)$;

$print(B)$;

out: $B = \begin{pmatrix} -11 & 0 \\ 0 & -11 \end{pmatrix}$;

```
SPACE=Z[x, y];
A=[[\cos(y), \sin(x)], [\sin(y), \cos(x)]];
B=\toEchelonForm(A);
\print(B);
```

Returns:

in: $SPACE = Z[x, y]$;

$A = \begin{pmatrix} cos(y) & sin(x) \\ sin(y) & cos(x) \end{pmatrix}$;

$B = toEchelonForm(A)$;

$print(B)$;

out: $B = \begin{pmatrix} cos(y)cos(x) - sin(x)sin(y) & 0 \\ 0 & cos(y)cos(x) - sin(x)sin(y) \end{pmatrix}$.

9.8.2. Computation of the kernel

To calculate the kernel of matrix A, you should run **kernel**(A).

Examples.

```
SPACE=Z[x];
A=[[1, 4], [4, 16]];
B=\kernel(A);
\print(B);
```

Returns:
in: $SPACE = Z[x]$;
$A = \begin{pmatrix} 1 & 4 \\ 4 & 16 \end{pmatrix}$;
$B = kernel(A)$;
$print(B)$;
out: $B = \begin{pmatrix} 0 & 4 \\ 0 & -1 \end{pmatrix}$;

```
SPACE=Z[x, y];
A=[[x+y, x], [(x+y)x, x^2]];
B=\kernel(A);
\print(B);
```

Returns:
in: $SPACE = Z[x, y]$;
$A = \begin{pmatrix} x+y & x \\ (x+y)x & x^2 \end{pmatrix}$;
$B = kernel(A)$;
$print(B)$;
out: $B = \begin{pmatrix} 0 & x \\ 0 & -y-x \end{pmatrix}$.

9.9. Calculating the characteristic polynomial of matrix

To calculate the characteristic polynomial of the matrix A with entries in $R[x_1, \ldots, x_m]$, you should give the ring $R[x_1, \ldots, x_m]R[t]$ or $R[t, x_1, \ldots, x_m]$ with some new variable t and run **charPolynom**(A).

Examples.

```
SPACE=Z[x];
A=[[1, 4], [4, 5]];
B=\charPolynom(A);
\print(B);
```

Returns:
in: $SPACE = Z[x]$;
$A = \begin{pmatrix} 1 & 2 \\ 4 & 5 \end{pmatrix}$;
$B = charPolynom(A)$;
$print(B)$;
out: $B = x^2 + (-6)x + (-11)$;

```
SPACE=Z[x, y]Z[t];
A=[[\cos(y), \sin(x)], [\sin(y), \cos(x)]];
B=\charPolynom(A);
\print(B);
```

Returns:
in: $SPACE = Z[x, y]$;
$A = \begin{pmatrix} cos(y) & sin(x) \\ sin(y) & cos(x) \end{pmatrix}$;
$B = charPolynom(A)$;
$print(B)$;
out: $B = t^2 + (-\cos(x) - \cos(y))t + \cos(y)\cos(x) - \sin(x)\sin(y)$.

9.10. Calculating LSU-decomposition of the matrix

To calculate the LSU-decomposition of the matrix A, you must run **LSU**(A).

The result is a vector of three matrices $[L, D, U]$. Where L is a lower triangular matrix, U — upper triangular matrix, D — permutation matrix, multiplied by the inverse of the diagonal matrix. If the elements of the matrix A are elements of commutative domain R, then elements of matrices L, D^{-1}, U are elements of the same domain R.

Examples.

```
SPACE=Z[x];
A=[[0, 1, 0], [4, 5, 1],[1, 1, 1]];
B=\LSU(A);
\print(B);
```

Returns:

in: $SPACE = Z[x]$;

$A = \begin{pmatrix} 0 & 1 & 0 \\ 4 & 5 & 1 \\ 1 & 1 & 1 \end{pmatrix}$;

$B = LSU(A)$

$print(B)$;

out: $B = \left[\begin{pmatrix} 4 & 0 & 0 \\ 0 & 4 & 0 \\ -1 & 1 & 3 \end{pmatrix}, \quad \begin{pmatrix} 0 & 1/16 & 0 \\ 1/4 & 0 & 0 \\ 0 & 0 & 1/12 \end{pmatrix}, \quad \begin{pmatrix} 4 & 5 & 1 \\ 0 & 4 & 0 \\ 0 & 0 & 3 \end{pmatrix} \right]$.

```
SPACE=Z[x];
A=[[1, 4,0,1], [4, 5,5,3],[1,2,2,2],[3,0,0,1]];
B=\LSU(A);
\print(B);
```

Returns:

in: $SPACE = Z[x]$;

$A = \begin{pmatrix} 1 & -4 & 0 & 1 \\ 4 & 5 & 5 & 3 \\ 1 & 2 & 2 & 2 \\ 3 & 0 & 0 & 1 \end{pmatrix}$;

$B = LSU(A)$

$print(B)$;

out: $B = \left[\begin{array}{c} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 4 & -11 & 0 & 0 \\ 1 & -2 & -12 & 0 \\ 3 & -12 & 60 & -60 \end{pmatrix} \\ \\ \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1/(-11) & 0 & 0 \\ 0 & 0 & 1/132 & 0 \\ 0 & 0 & 0 & 1/720 \end{pmatrix} \\ \\ \begin{pmatrix} 1 & 4 & 0 & 1 \\ 0 & -11 & 5 & -1 \\ 0 & 0 & -12 & -13 \\ 0 & 0 & 0 & -60 \end{pmatrix} \end{array} \right]$.

```
SPACE=Z[x,y];
A=[[\cos(y),\sin(x)],[\sin(y),\cos(x)]];
B=\LSU(A);
\print(B);
```

Returns:
in: $SPACE = Z[x, y]$;
$A = \begin{pmatrix} \cos(y) & \sin(x) \\ \sin(y) & \cos(x) \end{pmatrix}$
$B = LSU(A)$
$print(B)$;
out: $B = \left[\begin{array}{c} \begin{pmatrix} \cos(y) & 0 \\ \sin(y) & \cos(y)\cos(x) + (-\sin(x)\sin(y)) \end{pmatrix} \\ \\ \begin{pmatrix} 1/\cos(y) & 0 \\ 0 & 1/((\cos(y))^2\cos(x) + (-1\cos(y)\sin(x)\sin(y))) \end{pmatrix} \\ \\ \begin{pmatrix} \cos(y) & \sin(x) \\ 0 & \cos(y)\cos(x) + (-\sin(x)\sin(y)) \end{pmatrix} \end{array} \right]$.

9.11. Choletsky Decomposition

This decomposition is done with a command where the argument is the original matrix: **cholesky**(A) or **cholesky**$(A, 0)$. In this case, the

matrix must be symmetric and positive definite, only in this case the expansion will be correctly calculated.

The result is two lower triangular matrices: $[L, S]$, with $A = l * L^T$ and $S * L = I$.

For large dense matrices, starting from a size of 100x100, you can use a fast algorithm that uses multiplication of blocks by the Winograd-Strassen algorithm: **cholesky**$(A, 1)$.

Example.

```
SPACE=R64[];
 A=[[3,2],[2,4]];
B=\cholesky(A);
\print(B);
```

Returns:

in: $SPACE = R64[]$;

$A = \begin{pmatrix} 3 & 2 \\ 2 & 4 \end{pmatrix}$;

$B = cholesky(A)$

$print(B)$;

out: $B = \left[\begin{pmatrix} 1.73 & 0 \\ 1.15 & 1.63 \end{pmatrix}, \quad \begin{pmatrix} 0.58 & 0 \\ -0.41 & 0.61 \end{pmatrix} \right]$.

9.12. LSUWMdet decomposition

To calculate the LSU-decomposition of the matrix A together with decomposition of the pseudo inverse matrix $A^{\times} = (1/det^2)WSM$, you must run **LSUWMdet**(A).

The result is a vector of five matrices and determinant of the largest non-degenerate corner block $[L, D, U, W, M, det]$. Here L and U are the lower and upper triangular matrices, S — truncated weighted permutation matrix, DM and WD — lower and upper triangular matrices. Moreover, $A = LSU$ and $A^{\times} = (1/det^2)WSM$. If the elements of the matrix A are taken from the commutative domain, then all matrices, except for S, also belong to this domain.

Example.

```
SPACE=Z[x,y];
A=[[0, 1, x],[0, 5*y, 2],[y,3x, 1]];
B=\LSUWMdet(A);
\print(B);
```

Returns:

in: $SPACE = Z[x, y];$

$$A = \begin{pmatrix} 0 & 1 & x \\ 0 & 5y & 2 \\ y & 3x & 1 \end{pmatrix};$$

$B = LSUWMdet(A)$

$print(B);$

out:
$$B = \left[\begin{pmatrix} 1 & 0 & 0 \\ 5y & -5y^2x + 2y & 0 \\ 3x & 0 & y \end{pmatrix}, \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & (-1/(5y^3x + -2y^2)) \\ 1/y & 0 & 0 \end{pmatrix}, \begin{pmatrix} y & 0 & -3x^2 + 1 \\ 0 & 1 & x \\ 0 & 0 & -5y^2x + 2y \end{pmatrix}, \begin{pmatrix} 0 & -15y^2x^3 + 5y^2x + 6yx^2 - 2y & -5y^2x + 2y \\ -5y^2x + 2y & 5y^3x^2 - 2y^2x & 0 \\ 0 & -5y^3x + 2y^2 & 0 \end{pmatrix}, \begin{pmatrix} 15y^2x^2 - 6yx & 0 & -5y^2x + 2y \\ -5y^2x + 2y & 0 & 0 \\ 25y^4x - 10y^3 & -5y^3x + 2y^2 & 0 \end{pmatrix}, \begin{bmatrix} -5y^2x + 2y \end{bmatrix} \right].$$

9.13. Calculating Bruhat decomposition of the matrix

To calculate the Bruhat decomposition of the matrix A, you must run **BruhatDecomposition**(A).

The result is a vector of three matrices $[V, D, U]$. Where V and U — upper triangular matrices, D — permutation matrix, multiplied by the inverse of the diagonal matrix. If the elements of the matrix A are elements of commutative domain R, then elements of matrices V, D^{-1},

U are elements of the same domain R.

Examples.

```
SPACE=Z[x];
A=[[1, 4,0,1], [4, 5,5,3],[1,2,2,2],[3,0,0,1]];
B=\BruhatDecomposition(A);
\print(B);
```

Returns:

in: $SPACE = Z[x]$;

$A = \begin{pmatrix} 1 & -4 & 0 & 1 \\ 4 & 5 & 5 & 3 \\ 1 & 2 & 2 & 2 \\ 3 & 0 & 0 & 1 \end{pmatrix}$;

$B = BruhatDecomposition(A)$

$print(B)$;

out: $B = \begin{bmatrix} \begin{pmatrix} -24 & 0 & 12 & 1 \\ 0 & 60 & 15 & 4 \\ 0 & 0 & 6 & 1 \\ 0 & 0 & 0 & 3 \end{pmatrix} \\ \begin{pmatrix} 0 & 0 & 1/(-144) & 0 \\ 0 & 0 & 0 & 1/(-1440) \\ 0 & 1/18 & 0 & 0 \\ 1/3 & 0 & 0 & 0 \end{pmatrix} \\ \begin{pmatrix} 3 & 0 & 0 & 1 \\ 0 & 6 & 6 & 5 \\ 0 & 0 & -24 & -16 \\ 0 & 0 & 0 & 60 \end{pmatrix} \end{bmatrix}$.

```
SPACE=Z[x,y];
A=[[\cos(y),\sin(x)],[\sin(y),\cos(x)]];
B=\BruhatDecomposition(A);
\print(B);
```

Returns:

in: $SPACE = Z[x, y]$;

$A = \begin{pmatrix} \cos(y) & \sin(x) \\ \sin(y) & \cos(x) \end{pmatrix}$

$B = BruhatDecomposition(A)$
$print(B)$;

out: $B = \begin{bmatrix} \begin{pmatrix} -\cos(y) * \cos(x) + \sin(x) * \sin(y) & \cos(y) \\ 0 & \sin(y) \end{pmatrix} \\ \begin{pmatrix} 0 & 1/(-\cos(y)\sin(y)\cos(x) + \sin(x)(\sin(y))^2) \\ 1/\sin(y) & 0 \end{pmatrix} \\ \begin{pmatrix} \sin(y) & \cos(x) \\ 0 & -\cos(y)\cos(x) + \sin(x)\sin(y) \end{pmatrix} \end{bmatrix}$.

Other functions:

LSUWMdet — Result is a vector of 6 matrices $[L, S, U, W, M, [[det]]]$. A = LSU, pseudoInverse(A) = $(1/det^2)$WSM, det is a nonzero maximum in size angular minor.

pseudoInverse — Pseudo inverse of a matrix. It, unlike the Moore-Penrose matrix, satisfies only two of the four identities. However, it is faster to compute;

SVD — SVD decomposition of a matrix over real numbers. The result is a vector of three matrices $[U, D, V^T]$. Here U, V^T — are orthogonal matrices, D is a diagonal matrix.

QR — QR decomposition of a matrix over real numbers. The result is a vector of two matrices $[Q, R]$. Here Q — is an orthogonal matrix, R — is an upper triangular matrix.

sylvester$p1, p2, kind = 0 or 1$ — the Sylvester matrix is constructed from the coefficients of the polynomials $p1, p2$. The ring Z [x, y, z, u] will be considered as a ring Z[u][x, y, z] (ring in one variable u with coefficients from Z[x, y, z].) If kind = 0, then the size of the matrix is (n1 + n2), if kind = 1, then the size of the matrix is 2*max(n1, n2).

9.14. Linear programming

Let there be given the objective function $\sum_{j=1}^{n} c_j x_j$ and conditions

$$\sum_{j=1}^{n} a_{ij} x_j \leqslant b_i, \text{ here } i = 1, 2, \ldots, m,$$

$$x_j \geqslant 0, \text{ here } j = 1, 2, \ldots, n.$$

We define $m \times n$-matrix $A = (a_{ij})$, m-dimensional vector $b = (b_i)$, n-dimensional vector $c = (c_j)$ and n-dimensional vector $x = (x_j)$.

Then the objective function can be written as $c^T x$, and and conditions can be written as

$$Ax \leqslant b,$$

$$x \geqslant 0.$$

For solving linear programming problems, you can use one of the following two commands **SimplexMax** or **SimplexMin**. The result is a vector.

Depending on the type of problem you have the following options.

1. To solve the problem

$$c^T x \to max$$

under conditions

$$Ax \leqslant b,$$

$$x \geqslant 0,$$

we use the **SimplexMax**(A, b, c).

If the objective function needs to be minimized, , i.e.

$$c^T x \to min,$$

then we use the **SimplexMin**(A, b, c).

Example.

We need to maximize the

$$3x_1 + x_2 + 2x_3$$

under the conditions

$$\begin{cases} x_1 + x_2 + 3x_3 \leqslant 30, \\ 2x_1 + 2x_2 + 5x_3 \leqslant 24, \\ 4x_1 + x_2 + 2x_3 \leqslant 36, \\ x_1, x_2, x_3 \geqslant 0. \end{cases}$$

```
SPACE = R64[];
A = [[1, 1, 3],[2, 2, 5],[4, 1, 2]];
b = [30, 24, 36];
c = [ 3,  1,  2];
x = \SimplexMax(A, b, c);
```

Returns:

in:

out: [8.0, 4.0, 0.0];

2. To solve the problem

$$c^T x \to max$$

under the conditions

$$A_1 x \leqslant b_1,$$

$$A_2 x = b_2,$$

$$x \geqslant 0,$$

we use the **SimplexMax**(A_1, A_2, b_1, b_2, c).

If the objective function needs to be minimized, i.e.

$$c^T x \to min,$$

then we use the **SimplexMin**(A_1, A_2, b_1, b_2, c).

Example.

We need to maximize the

$$7x_1 + x_3 - 4x_4$$

under the conditions

$$\begin{cases} x_1 - x_2 + 2x_3 - x_4 \leqslant 6, \\ 2x_1 + x_2 - x_3 = -1, \\ x_1, x_2, x_3, x_4 \geqslant 0. \end{cases}$$

```
SPACE = R64[];
A1 = [[1, -1,  2, -1]];
A2 = [[2,  1, -1,  0]];
b1 = [ 6];
b2 = [-1];
c  = [7, 0, 1, -4];
x = \SimplexMax(A1, A2, b1, b2, c);
```

Returns:
in:
out: [0.8, 0.0, 2.6, 0.0];

3. To solve the problem

$$c^T x \to max$$

under the conditions

$$A_1 x \leqslant b_1,$$

$$A_2 x = b_2,$$

$$A_3 x \geqslant b_3,$$

we use the **SimplexMax**$(A_1, A_2, A_3, b_1, b_2, b_3, c)$.

If the objective function needs to be minimized, i.e.

$$c^T x \to min,$$

then we use the **SimplexMin**$(A_1, A_2, A_3, b_1, b_2, b_3, c)$.

Example.

$$7x_1 + x_3 - 4x_4$$

We need to maximize the

$$x_1 + x_2$$

under the conditions

$$\begin{cases} 4x_1 - x_2 \leqslant 8, \\ 2x_1 + x_2 \leqslant 10, \\ -5x_1 + 2x_2 \geqslant -2, \\ x_1, x_2 \geqslant 0. \end{cases}$$

```
SPACE = R64[];
A1 = [[ 4, -1], [2,  1]];
A3 = [[-5,  2]];
b1 = [ 8, 10];
b3 = [-2];
c  = [1, 1];
x = \SimplexMax(A1, [[]], A3, b1, [], b3, c);
```

Returns:
in:
out: [2.0, 6.0];

4. To solve the problem

$$c^T x \to max$$

in mixed conditions desired by the matrix A and vector b, you can use the command **SimplexMax**$(A, signs, b, c)$, where an array of integers *signs* determines the signs of comparison:

-1 means "less than or equal to",

0 means "equal to",

1 means "greater than or equal to".

The array *signs* must contain the same number of elements as the vector b. If the objective function needs to be minimized, i.e.

$$c^T x \to min,$$

then we use the **SimplexMin**$(A, signs, b, c)$.

Example.

We need to minimize the

$$-2x_1 - 4x_2 - 2x_3$$

under the conditions

$$\begin{cases} -2x_1 + x_2 + x_3 \leqslant 4, \\ -x_1 + x_2 + 3x_3 \leqslant 6, \\ x_1 - 3x_2 + x_3 \leqslant 2, \\ x_1, x_2, x_3 \geqslant 0. \end{cases}$$

In:

```
SPACE = R64[];
A = [[-2,  1, 1],[-1,  1, 3],[ 1, -3, 1]];
b = [ 4,  6,  2];
c = [-2, -4, -2];
signs = [-1, -1, -1];
x = \SimplexMin(A, signs, b, c);
```

Returns:
in:
out: Simplex: LP-problem is unbounded!

Chapter 10

The functions of the probability theory and statistics

10.1. Functions of the discrete random quantity

To define a discrete random quantity, enter the matrix, in which the first line — values, and the second — the corresponding probabilities (numbers that are in the range from 0 to 1). For example: DRQ = ([1,2,3,4,5],[0.4,0.1,0.1,0.2,0.2]).

There are the following functions for working with discrete random variable:

mathExpectation(DRQ) calculates the expectation of a discrete random variable DRQ.

dispersion(DRQ) calculates the variance of a discrete random variable DRQ.

meanSquareDeviation(DRQ) calculates the standard deviation of a discrete random variable DRQ.

addQU($DRQ1, DRQ2$) adds the two discrete random variables $DRQ1$ and $DRQ2$.

multiplyQU($DRQ1, DRQ2$) multiplies two discrete random variables $DRQ1$ and $DRQ2$.

covariance$(DRQ1, DRQ2)$ calculates the covariance coefficient of two discrete random variables $DRQ1$ and $DRQ2$.

correlation$(DRQ1, DRQ2)$ calculates the correlation coefficient of two discrete random variables $DRQ1$ and $DRQ2$.

plotPolygonDistribution(DRQ, V) building polygon distributions of discrete random variable DRQ.

plotDistributionFunction(DRQ, V) constructing the distribution function of a discrete random variable DRQ, where V — is the matrix of one row, 4 elements that define the boundaries Graphics: $[x1, x2, y1, y2]$.

simplifyQU(DRQ) simplify a discrete random variable DRQ.

Examples:

```
SPACE=R64[x];
M=[[1, 2], [0. 2, 0. 8]];
g=\mathExpectation(M);
g1=\dispersion(M);
g2=\meanSquareDeviation(M);
\print(g, g1, g2);
```

Returns:
in: $SPACE = R64[x]$;
$M = \begin{pmatrix} 1 & 2 \\ 0.2 & 0.8 \end{pmatrix}$;
$g = mathExpectation(M)$;
$g1 = dispersion(M)$;
$g2 = meanSquareDeviation(M)$;
$print(g, g1, g2)$;
out: $g = 1.8$;
$g1 = 0.16$;
$g2 = 0.39$;

```
SPACE=R64[x];
M=[[7, 5, 3, 5, 1], [0. 2, 0. 1, 0. 3, 0. 1, 0. 3]];
g=\simplifyQU(M);
\print(g);
```

Returns:
in: $SPACE = R64[x]$;

$M = \begin{pmatrix} 7 & 5 & 3 & 5 & 1 \\ 0.2 & 0.1 & 0.3 & 0.1 & 0.3 \end{pmatrix}$;
$g = simplifyQU(M)$;
$print(g)$;
out: $g = \begin{pmatrix} 1 & 3 & 5 & 7 \\ 0.3 & 0.3 & 0.2 & 0.2 \end{pmatrix}$;

```
SPACE=R64[x];
M1=[[0, 1], [0. 33333, 0. 66666]];
M2=[[1, 2], [0. 25, 0. 75]];
g=\addQU(M1, M2);
g1= \multiplyQU(M1, M2);
\print(g, g1);
```

Returns:
in: $SPACE = R64[x]$;
$M1 = \begin{pmatrix} 0 & 1 \\ 0.33333 & 0.66666 \end{pmatrix}$;
$M2 = \begin{pmatrix} 1 & 2 \\ 0.25 & 0.75 \end{pmatrix}$;
$g = addQU(M1, M2)$;
$g1 = multiplyQU(M1, M2)$;
$print(g, g1)$;
out: $g = \begin{pmatrix} 1 & 2 & 3 \\ 0.08 & 0.41 & 0.49 \end{pmatrix}$;
$g1 = \begin{pmatrix} 0 & 1 & 2 \\ 0.33 & 0.16 & 0.49 \end{pmatrix}$;

```
SPACE=R64[x];
M=[[-7, -2, 0, 3, 5, 7, 9],
  [0.3, 0.05, 0.2, 0.1, 0.1, 0.2, 0.05]];
V=[-10, 10, 0, 1];
\plotPolygonDistribution(M, V);
```

Returns:
in: $SPACE = R64[x]$;
$M = \begin{pmatrix} -7 & -2 & 0 & 3 & 5 & 7 & 9 \\ 0.3 & 0.05 & 0.2 & 0.1 & 0.1 & 0.2 & 0.05 \end{pmatrix}$;
$V = [-10, 10, 0, 1]$;

plotPolygonDistribution(M, V);
out: Pic. 10.1.

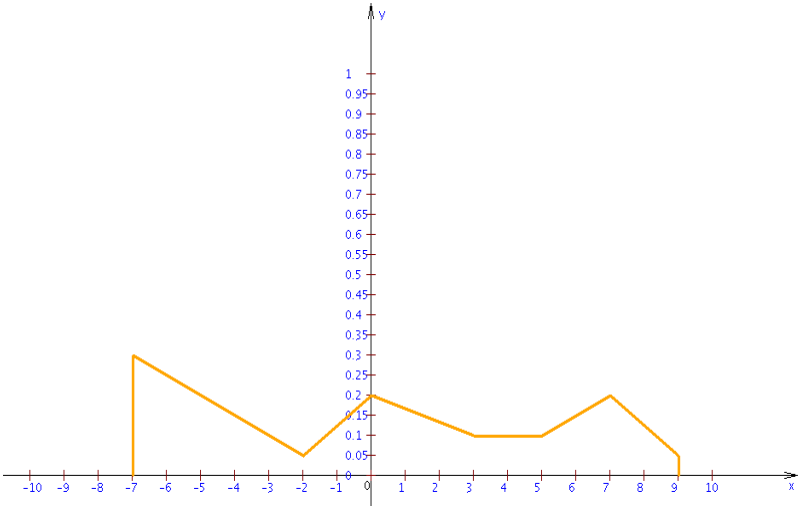


Figure 10.1: Polygon of distributions of discrete random variable from the example.

10.2. Function for sampling

Function for sampling:

W-matrix of a single line. For example, $[1, 7, 10, 15]$.

sampleMean(S) calculates the sample mean of the sample S.

sampleDispersion(S) calculates the sample variance of the sample S.

covarianceCoefficient$(S1, S2)$ calculates the coefficient of covariance for 2 sampling $S1$ and $S2$.

correlationCoefficient$(S1, S2)$ calculates the correlation coefficient for 2 sampling $S1$ and $S2$.

Example

```
SPACE=R[x, y];
S1=[0, 1];
```

```
S2=[1, 2];
g=\sampleMean(S1);
g1=\sampleDispersion(S1);
g2=\covarianceCoefficient(S1, S2);
g3=\correlationCoefficient(S1, S2);
\print(g, g1, g2, g3);
```

Returns:
in: $SPACE = R[x, y]$;
$S1 = [0, 1]$;
$S2 = [1, 2]$;
$g = sampleMean(S1)$;
$g1 = sampleDispersion(S1)$;
$g2 = covarianceCoefficient(S1, S2)$;
$g3 = correlationCoefficient(S1, S2)$;
$print(g, g1, g2, g3)$;
out: $g = 0.5$;
$g1 = 0.25$;
$g2 = 0.25$;
$g3 = 1.00$.

Chapter 11

Operators of control. Procedural programming

11.1. Procedures and functions

Mathpar system lets you create your procedures and functions. To do this, use the command **procedure**. After the command **procedure**, you must specify the name of the procedure, and then in the curly brackets describes the procedure itself.

Example.

```
\procedure myProc2() {
  d = 4;
  \print(d);
}
\procedure myProc(c, d) {
  if (c < d) {
    \return d;
  } else {
    \return d + 5;
  }
}
\myProc2();
a = 10;
c = \myProc(5 + a, a);
\print(a, c);
```

Returns:
$d = 4;\ a = 10;\ c = 15.$

11.2. Operators of branching and looping

You can use the operators of branching and looping:
if() { } **else** { } — the operator of branching;
while() { } — cycle operator with precondition;
for(; ;) { } — cycle operator with the counter.

Examples:

```
a = 5;
b = 1;
if(b < a) {
  b = b + a;
} else {
  \print(a, b);
}
if(b < a) {
  b = b + a;
} else {
  \print(a, b);
}
```

Returns:
$a = 5;\ b = 6;$

```
a = 0;
b = 10;
while(a < b) {
  a = a + 5;
  \print(a);
}
```

Returns:
$a = 5;\ a = 10;$

```
for (i = 3; i \le 11; i = i + 5) {
  \print(i);
}
```

Returns:
$i = 3$; $i = 8$.

Chapter 12

Calculations in idempotent algebras

12.1. Tropical algebras

You can work in the following tropical algebras :
SEMIFIELDS
1) On the set of integers $mathbbZ$ we define:
$ZMaxPlus$, $ZMinPlus$.
2) On the set of numbers $\mathbb{R}$ we define:
$RMaxPlus$, $RMinPlus$, $RMaxMult$, $RMinMult$.
3) On the set of numbers $\mathbb{R}64$ we define:
$R64MaxPlus$, $R64MinPlus$, $R64MaxMult$, $R64MinMult$.

SEMIRINGS
1) On the set of numbers $\mathbb{Z}$ we define:
$ZMaxMin$, $ZMinMax$, $ZMaxMult$, $ZMinMult$.
2) On the set of numbers $\mathbb{R}$ we define:
$RMaxMin$, $RMinMax$.
3) On the set of numbers $\mathbb{R}64$ we define:
$R64MaxMin$, $R64MinMax$.
Examples of tropical algebras:
SPACE = ZMaxPlus [x, y, z];
SPACE = R64MinMult [u, v];
SPACE = RMaxMin [u, v].

An example of a simple problem in a semiring $ZMaxMult$.

Example 1.

```
SPACE = ZMaxMult[x, y];
a = 2; b = 9; c = a + b; d = a*b; \print(c, d)
```

Returns:
$c = 9$;
$d = 18$.

In the remaining sections of this chapter we have given some examples of problems that are solved in the tropical algebra, which are semi-fields.

12.2. Solving systems of linear algebraic equations

The command $\backslash solveLAETropic(A, b)$ enables us to find a particular solution of the system $Ax = b$.

Example 2.

```
SPACE = R64MaxPlus[x, y];
A = [
  [1.00, 1.00, 0.00],
  [2.00, 0.00, 3.00],
  [3.00, 4.00, 2.00]
];
b = [8.00, 7.00, 11.00];
X = \solveLAETropic(A, b);
\print(X);
```

Returns:

$$X = \begin{pmatrix} 5.00 \\ 7.00 \\ 4.00 \end{pmatrix}$$

12.3. Solving systems of linear algebraic inequalities

The command $\backslash solveLAITropic(A, b)$ enables us to find a particular solution of the system of inequalities

Example 3.

```
SPACE = R64MaxPlus[x, y];
A = [
  [1.00, 1.00, 0.00],
  [2.00, 0.00, 3.00],
  [3.00, 4.00, 2.00]
];
b = [10.00, 7.00, 11.00];
X = \solveLAITropic(A, b);
\print(X);
```

Returns:
$X = [(-\infty, 5.00], (-\infty, 7.00], (-\infty, 4.00]]$

Example 4.

```
SPACE = ZMinPlus[x, y];
A = [
  [1, 1, 0],
  [2, 0, 3],
  [3, 4, 2]
];
b = [10, 7, 11];
X = \solveLAITropic(A, b);
\print(X);
```

Returns:
$X = [[9, \infty), [9, \infty), [10, \infty)]$

12.4. The solution of the Bellman equation

12.4.1. The homogeneous Bellman equation

The command $\backslash BellmanEquation(A)$ enables us to find a solution of the homogeneous Bellman equation $Ax = x$.

Example 5.

```
SPACE = R64MaxPlus[x, y]; TIMEOUT = 16;
A = [
  [0.00, -2.00, -\infty, -\infty],
  [-\infty, 0.00, 3.00, -1.00],
```

```
  [-1.00, -\infty, 0.00, -4.00],
  [2.00, -\infty, -\infty, 0.00]
];
X = \BellmanEquation(A);
\print(X);
```

Returns:

$$X = \begin{pmatrix} 0.00 & -2.00 & 1.00 & -3.00 \\ 2.00 & 0.00 & 3.00 & -1.00 \\ -1.00 & -3.00 & 0.00 & -4.00 \\ 2.00 & 0.00 & 3.00 & 0.00 \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{pmatrix}, \forall v_1, v_2, v_3, v_4.$$

12.4.2. The inhomogeneous Bellman equation

The command $\backslash BellmanEquation(A, b)$ enables us to find a solution of the inhomogeneous Bellman equation $Ax \oplus b = x$.

Example 6.

```
SPACE = R64MaxPlus[x, y];  TIMEOUT = 16;
A = [
  [0.00, -2.00, -\infty, -\infty],
  [-\infty, 0.00, 3.00, -1.00],
  [-1.00, -\infty, 0.00, -4.00],
  [2.00, -\infty, -\infty, 0.00]
];
b = [[1], [-\infty], [-\infty], [-\infty]];
X = \BellmanEquation(A, b);
\print(X);
```

Returns:

$$X = \begin{pmatrix} 0.00 & -2.00 & 1.00 & -3.00 \\ 2.00 & 0.00 & 3.00 & -1.00 \\ -1.00 & -3.00 & 0.00 & -4.00 \\ 2.00 & 0.00 & 3.00 & 0.00 \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{pmatrix} \oplus \begin{pmatrix} 1.00 \\ 3.00 \\ 0.00 \\ 3.00 \end{pmatrix},$$

$$\forall v_1, v_2, v_3, v_4.$$

12.5. The solution Bellman inequality

12.5.1. The homogeneous Bellman inequality

The command $\backslash BellmanInequality(A)$ enables us to find a solution of the homogeneous Bellman inequality $Ax \leqslant x$.

12.5.2. The inhomogeneous Bellman inequality

The command $\backslash BellmanInequality(A, b)$ enables us to find a solution of the inhomogeneous Bellman inequality $Ax \oplus b \leqslant x$.

12.6. Finding the shortest path between the vertices of the graph

12.6.1. Calculation of the table of shortest distances for all vertices of the graph

Let $A = (x_{ij})$ be matrix of distances between adjacent vertices. We put x_{ii}=0 $\forall i$ and we put $x_{ij} = \infty$, if there is no edge connecting vertices i and j. The command $\backslash searchLeastDistances(A)$ allows you to find the smallest distance between all the nodes of the graph. This results in a matrix of shortest paths between all vertices.

Example 7.

```
SPACE = R64MinPlus[x, y];  TIMEOUT = 16;
A = [
  [0.00, 7.00, 9.00, \infty, \infty, 14.00],
  [7.00, 0.00, 10.00, 15.00, \infty, \infty],
  [9.00, 10.00, 0.00, 11.00, \infty, 2.00],
  [\infty, 15.00, 11.00, 0.00, 6.00, \infty],
  [\infty, \infty, \infty, 6.00, 0.00, 9.00],
  [14.00, \infty, 2.00, \infty, 9.00, 0.00]
];
B = \searchLeastDistances(A);
\print(B);
```

Returns:

$$B = \begin{pmatrix} 0.00 & 7.00 & 9.00 & 20.00 & 20.00 & 11.00 \\ 7.00 & 0.00 & 10.00 & 15.00 & 21.00 & 12.00 \\ 9.00 & 10.00 & 0.00 & 11.00 & 11.00 & 2.00 \\ 20.00 & 15.00 & 11.00 & 0.00 & 6.00 & 13.00 \\ 20.00 & 21.00 & 11.00 & 6.00 & 0.00 & 9.00 \\ 11.00 & 12.00 & 2.00 & 13.00 & 9.00 & 0.00 \end{pmatrix}$$

12.6.2. Calculation of the shortest distances between two vertices of the graph

Let $A = (x_{ij})$ be matrix of distances between adjacent vertices. We put x_{ii}=0 $\forall i$ and we put $x_{ij} = \infty$, if there is no edge connecting vertices i and j.

The command $\backslash findTheShortestPath(A, i, j)$ allows you to find the shortest path between nodes i and j.

Example 8.

```
SPACE = R64MinPlus[x, y];  TIMEOUT = 16;
A = [
  [0.00, 7.00, 9.00, \infty, \infty, 14.00],
  [7.00, 0.00, 10.00, 15.00, \infty, \infty],
  [9.00, 10.00, 0.00, 11.00, \infty, 2.00],
  [\infty, 15.00, 11.00, 0.00, 6.00, \infty],
  [\infty, \infty, \infty, 6.00, 0.00, 9.00],
  [14.00, \infty, 2.00, \infty, 9.00, 0.00]
];
X = \findTheShortestPath(A, 0, 4);
\print(X);
```

Returns:
$X = [[0, 2, 5, 4]]$

Chapter 13

The calculations on a supercomputer

In order to solve computational problems that require large computation time or large amounts of memory, the system has special functions that provide the user with resources of supercomputer. These functions allow you to perform calculations on a dedicated set of cores. The number of kernels ordered by the user.

You have the following functions (*3ar-functions*) that apply to supercomputer:

1) **matMultPar1x8** — calculation of the matrix product;

2) **adjointDetPar** — computation of the adjoint matrix and determinant;

3) **charPolPar** — computation of the characteristic polynomial of a matrix;

4) **polMultPar** — computation of the product of two polynomials;

5) **BellmanEquationPar**A — solution of a homogeneous Bellman equation $Ax = x$;

6) **BellmanEquationPar**A, b — solution of an inhomogeneous Bellman equation $Ax + b = x$;

7) **BellmanInequalityPar**A — solution of a homogeneous Bellman inequality $Ax \leqslant x$;

8) **BellmanInequalityPar**A, b — solution of an inhomogeneous Bellman inequality $Ax + b \leqslant x$;

Before applying any of these functions, the user must specify the

parameters that define the parallel environment:

$TOTALNODES$ — total number of processors (cores), which provides for the computation

$PROCPERNODE$ — number of cores on a single node,

$CLUSTERTIME$ — maximum time (in minutes) execution of the program, after which the program is forced to end.

$MAXCLUSTERMEMORY$ — amount of memory allocated for the JVM for a one process (for -Xmx parameter).

To set the number of cores on a single node the user must know what a cluster is used and how many cores it is available on the node. By default, the $TOTALPROCNUMBER$ and $NODEPROCNUMBER$ installed so that all the cores were used per node, and $CLUSTERTIME = 1$.

The user can change the number of cores on a single node. This is an important feature, since the memory on a single node is used by all $NODEPROCNUMBER$ cores. Consequently, the user can regulate the size of of RAM that is available to one core.

13.1. Parallel polynomial computations

For parallel computation of the polynomial product you can use the $\backslash$**multiplyPar**$(p1, p2)$, where $p1$, $p2$ — given polynomials.

Example.

```
TOTALNODES=2;
PROCPERNODE=1;
CLUSTERTIME=1;
f=x^2+3y;
g=x^2+3y+3z;
\polMultPar(f,g);
```

13.2. Parallel matrix computations

For parallel computing products of matrices $m1$ and $m2$ you must use the $\backslash$**multiplyPar**$(m1, m2)$.

Example.

```
TOTALNODES = 2;
PROCPERNODE = 1;
A=[[0,1],[2,3]];
B=[[5,61],[7,8]];
\matMultPar1x8(A, B);
```

For parallel computation of the adjoint matrix for the matrix m you can use the **adjointPar**(m).

Similarly, for the matrix m you can perform the calculation of the echelon form **echelonFormPar**(m), the computation of the determinant **detPar**(m), computation of the kernel **kernelPar**(m), the calculation of the characteristic polynomial **charPolPar**(m). The command **adjointDetPar**(m) allows us to calculate the determinant and adjoint matrix simultaneously.

Example.

```
TOTALNODES = 2;
PROCPERNODE = 1;
SPACE = Z[x];
A=[[0,1],[2,3]];
\adjointDetPar(A);
```

section Running your parallel programs Mathpar allows you to download and execute your parallel programs. Your package must be in the root directory of the project. To ensure that your program is able to interact with the system management tasks, you need to add an initialization string

QueryResult queryRes = Tools.getDataFromClusterRootNode (args)

(immediately after MPI.Init) and the completion string

Tools.sendFinishMessage (args)

(before MPI.Finalize) in your main-method. You can also pass any arguments to your program from the web-interface Mathpartner. Within the program you can get them by calling queryRes.getData (). Below is an example of a parallel program that simply outputs on standard output the arguments passed to it.

```
MPI.Init(args);
QueryResult queryRes=Tools.getDataFromClusterRootNode(args);
int myRank=MPI.COMM_WORLD.getRank();
```

```
if (myRank == 0) {
    Object []ar=queryRes.getData();
    System.out.println("test...");
    for (int i=0; i<ar.length; i++){
        System.out.println(((Element)ar[i]).intValue());
    }
}
Tools.sendFinishMessage(args);
MPI.Finalize();
```

After that you need to compile the program, and the program folder packed in zip-archive.Then you need to download this file to the server, using the tab "File" and clicking "download file".

RAM is divided equally between all cores. For example, if the cluster node has 8GB of memory, then if you request 4 cores on a single processor, each will receive 2GB, and if you have requested one core - then it will get 8GB.

The command to download your zip-archive, which contains java-classes, as follows:

uploadToCluster($FileName$), where FileName - name of zip-archive.

To view a list of all your downloaded files on a cluster, use the command

showFileList().

To run your program, use the command

runUploadedClass($archieveName, classPath, param0, param1, ...$), where **archieveName** - the name of the downloaded zip-archive with the program, **classPath** - the path to the class containing main-method (full path with the packages) **paramX** - arbitrary parameters specified separated by commas, to be passed to your program.

To monitor the running programs, use the command

getStatus($taskID$)

It is also possible to get a list of all the tasks of the current user with a description of their states:

showTaskList()

To receive the content from the output stream and error stream, use the commands

getOut($taskID$)

getErr($taskID$)

Files that contain output or error stored on a cluster of two days, zip-archives containing the compiled java-classes are stored for 30 days.

Chapter 14

Operators and mathematical symbols

Naming rules for Mathematical Objects

Uppercase and lowercase letters are different everywhere. The user can give any names for mathematical objects. However, these names should not coincide with the operators and constants that are defined in the system. In addition, the names of objects, of which the multiplication is not commutative, for example, vectors and matrices, must begin with a capital Latin letters, and all other object names must start with lowercase letters. This makes it possible as soon as entering automatically get a simplified expression.

Here is a list of the main operators of the system Mathpar.

clean — clean input data (if this operator doesn't have arguments) or the date of arguments of this operator,

Infix arithmetic operators

+ — addition;

- — subtraction;

/ — division;

***** — multiplication (still a blank or absence of the operator);

times — noncommutative multiplication(still a blank or absence of the operator);

Postfix arithmetic operators
! — factorial;
$x\hat{}\{exp\}$ — exponentiation;

Comparisons
le — less than or equal to;
$>$ — greater than;
$<$ — less than;
ge — greater than or equal to;
== — it is equal;
ne — it is unequal;

Infix Boolean operators
lor — disjunction (logic OR);
\& — conjunction (logic AND);
neg — negation.
Key prefix operators
d — the symbol of derivative, wich is usualy used in the differential equations,
D — the operator of differentiation: $\mathbf{D}(f)$ and $\mathbf{D}(f,x)$ are the first derivative by x; $\mathbf{D}(f,y\hat{}3)$ is the third derivative by y;
expand — opening all brackets;
fullExpand — to expand expression containing logarithmic, exponential and trigonometric functions;
extendedGCD — extended polynomial GCD, returns a vector containing GCD and additional multipliers of arguments;
GCD — GCD of polynomials;
factor — to factor expression;
fullFactor — to factor expression containing logarithmic and exponent functions;
initCond — boundary conditions for a system of linear differential equations;
LCM — polynomial LCM;
lim — the limit of an expression;
print — the print operator of the expressions, the names of which are listed in this operator (each expression printed in the new line);
printS — the print operator, which is similar to the Pascal print operator(for printing in several lines you can use the symbol "\n";

plot — to plot explicit functions;

plot3D — to plot functions of two variables, which are given explicitly;

paramPlot — to plot parametric functions;

tablePlot — to plot of function, which are presented by the table of arguments and values;

prod — the symbol of product ($\prod$);

randomPolynom — to generate a random polynomial;

randomMatrix — to generate a random matrix;

randomNumber — to generate a random number;

sequence — the sequence;

showPlots — to display at one field of schedules of functions of different types;

solveLDE — to solve system of the linear differential equations;

systLAE — to set the system of the linear algebraic equations;

systLDE — to set the system of the linear differential equations;

sum — a summation symbol ($\sum$);

time — this operator returns the processor time in milliseconds;

value — to calculate value of expression by means of substitution of the expressions (or numbers) instead of ring variables;

Operators of the procedure, branching and loop

procedure — ad procedures;

$if(\ )\{\ \}else\{\ \}$ — operator of the branch;

$while(\ )\{\ \}$ — operator of the cycle with a precondition;

$for(\ ;\ ;\ )\{\ \}$ — cycle operator with a counter.

Matrix, matrix elements and matrix operators

[,] — setting vector (row-vector);

[[,], [,]] — the matrix may be defined as vector of vectors;

A_{i,j} — (i,j)-element of the matrix A;

A_{i,?} — row i of the matrix A;

A_{?,j} — j column of the matrix A;

\O_{n,m} — zero matrix of size $n \times m$;

\I_{n,m} — $n \times m$ matrix with ones on the diagonal;

+, -, * — addition, subtraction, multiplication;

rowNumb() — number of rows of the matrix (or vector);

colNumb() — number of columns of the matrix (or vector);

size() — both dimensions of the matrix (or number of components of the vector);

charPolynom() — calculation of a characteristic polynomial;

kernel() — calculation of a kernel (zero-space of matrix);

transpose() or **A**^T — transposing;

conjugate() or **A**^{\$ast$}$ — conjugate;

toEchelonForm() — calculation of the matrix echelon form;

det() — calculation the determinant;

rank() — calculation the rank;

inverse() or **A**^{-1} — calculation of the inverse of a matrix;

adjoint() or **A**^{\$star$}$ — calculation of the adjoint matrix;

genInverse() or **A**^{$+$} — generalized inverse of a matrix Moore-Penrose;

closure() or **A**^{\$times$}$ — closure, i.e. the amount of $I + A + A^2 + A^3 + \ldots$. For the classical algebras is equivalent to $(I - A)^{-1}$.

LSU() — LSU decomposition of a matrix. The result is a vector of three matrices [L,S,U]. Where L is a lower triangular matrix, U — upper triangular matrix, S — permutation matrix, multiplied by the inverse of the diagonal matrix.

LSUWMdet() — Result is a vector of 6 matrices $[L, S, U, W, M, [[det]]]$. A = LSU, pseudoInverse(A) = $(1/det^2)$WSM, det is a nonzero maximum in size angular minor.

BruhatDecomposition() — Bruhat decomposition of a matrix. The result is a vector of three matrices [V,D,U]. Where V and U — upper triangular matrices, D — permutation matrix, multiplied by the inverse of the diagonal matrix.

pseudoInverse() — Pseudo inverse of a matrix. It, unlike the Moore-Penrose matrix, satisfies only two of the four identities. However, it is faster to compute;

SVD() — SVD decomposition of a matrix over real numbers. The result is a vector of three matrices $[U, D, V^T]$. Here U, V^T — are orthogonal matrices, D is a diagonal matrix.

QR() — QR decomposition of a matrix over real numbers. The result is a vector of two matrices $[Q, R]$. Here Q — is an orthogonal matrix, R — is an upper triangular matrix.

sylvester($p1, p2, kind = 0 or 1$) — the Sylvester matrix is constructed from the coefficients of the polynomials $p1, p2$. The ring Z [x, y, z, u] will be considered as a ring Z[u][x, y, z] (ring in one variable u

with coefficients from Z[x, y, z].) If kind = 0, then the size of the matrix is (n1 + n2), if kind = 1, then the size of the matrix is 2*max(n1, n2).

cholesky(A) or **cholesky**$(A, 0)$ — Cholesky Decomposition of a matrix. The matrix A must be symmetric and positive definite, only in this case the expansion will be correctly calculated. **cholesky**$(A, 1)$ you can use in the case of large dense matrices, starting from a size of 100x100. Here we used block multiplication according to the Winograd–Strassen algorithm.

Chapter 15

Numerical Algorithms

15.1. Evaluation of definite and improper integrals

15.1.1. Calculation of definite integrals.

The calculation of definite integrals is performed using the Gauss method. To calculate a definite integral, you need to run the command: Nint (f, a, b, epsilon, N); Where: (a, b) - integration interval, f - integrand function, epsilon - the number of exact decimal places after the decimal point (optional), N is the number of points in the Gaussian formula (optional). The last three parameters can be omitted. The precision can be specified explicitly (using the epsilon parameter), or using the MachineEpsilon constant in the current ring.

Examples.

```
SPACE=R[x];
f = \sin(x);
B = \Nint(f, 0, \pi);
\print(B);
```

Returns:
in: $SPACE = R[x]$;
$f = sin(x)$;
$B = Nint(f, 0, \pi)$;
$print(B)$;

out: $B = 2.00$.

```
SPACE = R[x]; FLOATPOS = 18;
MachineEpsilonR = 18/22 ;
f=2*\sqrt(1-x^2);
A=\Nint(f, -1,1);
B=\value(\pi);
\print(A,B);
```

Returns:
in: $SPACE = R[x]; FLOATPOS = 18;$
$MachineEpsilonR = 18/22;$
$f = 2 * sqrt(1 - x^2);$
$A = Nint(f, -1, 1);$
$B = value(\pi);$
$print(A, B);$

out: $A = 3.141592653589793233$
$B = 3.141592653589793238$

15.1.2. Calculation of improper integrals of the first kind.

To calculate the improper integral on an infinite interval, you must run the command: \Nint (f, a, b, [...], epsilon, N); Where: (a, b) - interval of integration, where any of the boundaries of integration can be either a finite number or pm infty; f - integrand function, [...] - extremum points of the integrand in the interval (a, b) (optional), epsilon - the number of exact decimal places after the decimal point (optional), N is the number of points in the Gaussian formula (optional). The last three parameters can be omitted.

If the extremum points are not indicated, then the correctness of the result is ensured in the case when the integrand is monotonic on the interval of integration.

Improper integrals of the first kind are calculated using the following algorithm: Let, for definiteness, the interval of integration have the form: [a, ∞). We consider the integral of the function f (x) with a step of 3N. We get the segments: [a, a + 3N], [a + 3N, a + 6N], ... When the value

of the integral on the next segment becomes less than the value of the integral on the previous segment, the step is increased by a factor of 10. The calculation of the integral stops when the value of the integral in the current segment becomes less than the value of the integral in the previous segment and less than the machine zero (MachineEpsilon).

Examples

```
SPACE=R64[x];
f = \exp(-(x-5)^2);
B = \Nint(f, -\infty, \infty);
\print(B);
```

Returns:
in: $SPACE = R64[x]$;
$f = \exp(-(x-5)^2)$;
$B = Nint(f, -\infty, \infty)$;
$print(B)$;

out: $B = 1.77$;.

```
SPACE=R[x];
f = \exp(-x);
B = \Nint(f, 0, \infty);  \print(B);
```

Returns:
in: $SPACE = R[x]$;
$f = \exp(-x)$;
$B = Nint(f, 0, \infty)$; $print(B)$;

out: $B = 1.00$;.

Chapter 16

Examples of solutions of physical problems

16.1. Transferring of the heat

```
"EXERCISE 1"
"A piece of ice which has mass"
M = 10 kg;
"was put in a vessel. The ice has temperature ($\degreeC$)"
T = -10  \degreeC ;
"Find the mass of water in a vessel after transferring the"
q = 20000 kJ
"amount of heat. Specific heat of water heating is equal"
c_v = 4.2 kJ/(kg \degreeC);
"Specific heat of ice heating is equal"
c_i = 2.1 kJ/(kg \degreeC);
"The heat of fusion of ice is equal"
r = 330 kJ/kg;
"Specific heat of vaporization of water is equal"
\lambda = 2300 kJ/kg;
END

"SOLUTION OF EX. 1."
SPACE = R64[x];
"Unknown mass of water is denoted by x.
```

```
The amount of heat: to heat the ice to 0 degrees:"
q_1 = M c_i (0 - T);
"to melt the ice:"
q_2 = M r;
"to heat water to 100 degrees:"
q_3 = M c_v (100 \degreeC);
"to evaporation of water"
q_4 = (M - x) \lambda;
"we denote by x unknown value."
"By assumption, we obtain the equation"
mass = \solve(q = q_1 + q_2 + q_3 + q_4);
\print(mass);
```

16.2. Kinematics

```
"EXERCISE 2"
"Kinematic equation of motion of a point in a straight line (axis x)
has the form $x = c_1 + c_2 * t + c_3 t^3$."
"Define: (1) coordinate of a point, (2) the instantaneous velocity,
(3) the instantaneous acceleration "
END
```

```
"SOLUTION OF EX.2."
"Let us set the space with variables $t, c_1, c_2, c_3$:"
SPACE = R64[t, c_1, c_2, c_3];
"The equation of motion is"
x = c_1 + c_2 t + c_3 t^3;
"We can find the instant speed"
v = \D_t(x);
"We can find the instant acceleration"
a = \D_t(v);
\print(x, v, a);
```

```
"EXERCISE 2A"
"Solve the previous problem at a moment of time"
t_0 = 2 "seconds"
"with the following numerical values:
$c_1=4; c_2=2; c_3=-0.5$."
END
```

```
"SOLUTION OF EX.2A."
arg = [t_0, 4, 2, -0.5];
x_0 = \value (x, arg);
v_0 = \value (v, arg);
a_0 = \value (a, arg);
\print(x_0, v_0, a_0);
```

16.3. Molecular Physics

```
"EXERCISE 3"
"In the middle of the horizontal tube was placed a drop of mercury in
"The air was pumped out of the tube and the ends
of the tube was sealed. Tube length is equal l."
"When the tube was placed vertically, a drop of mercury moved down by
"The acceleration of gravity is equal g."
"The density of mercury is equal $\rho$."
"What was the initial pressure in the  tube?"
END

"SOLUTION OF EX. 3."
"Let the initial pressure $p_0$ be unknown:"
SPACE = R64[p_0];
"The pressure at the bottom of the tube increased, as added
pressure mercury drops, so new pressure is equal:"
p_1 = p_0 + \rho g h;
"Let S be the cross-sections of the tube. Then
the initial volume of air in the bottom of the tube is equal:"
v_0 = (l/2 - h/2) S;
"After turning the tube  volume of air in the bottom of the tube is e
v_1 = (l/2 - h/2 - l_d) S;
"According  to Boyle{Mariotte law we have the equation:"
initialPressure = \solve(p_0 v_0 = p_1 v_1);
\print(initialPressure);

"EXERCISE 3A"
"Solve the previous problem with the following numerical values: "
h = 0.20 m;
l = 1 m;
```

```
l_d = 0.10 m;
"The acceleration of gravity is equal"
g = 9.8 m/s^2;
\rho = 13600 kg/m^3;
END
"SOLUTION OF EX. 3A."
p_1 = p_0 + \rho g h;
v_0 = (l/2 - h/2) S;
v_1 = (l/2 - h/2 - l_d) S;
initialPressure = \solve(p_0 v_0 = p_1 v_1);
\print(initialPressure);
```

16.4. Pendulum

```
"EXERCISE 4.The period of a simple gravity pendulum."
SPACE = R64[x]; FLOATPOS = 4;\\
"A point mass suspended from a pivot with a massless cord.
The length of the pendulum equals L = 1 metre."
"It swings under gravitational acceleration g = 9.80665
metres per second squared."
"The maximum angle that the pendulum swings
away from vertical, called the amplitude, equals"
\theta_0=  (2/3) \pi;
"Find the period $T$ of the pendulum using the arithmetic-geometric m
"$$T=\frac{2\pi}{\AGM(1,\cos(\theta_0/2))}\sqrt{\frac{L}{g}}$$"
END
"SOLUTION OF EX. 4."
\theta_0=  (2/3) \pi;
w=\value(\cos(\theta_0/2));
 Ts = 2*\pi*\sqrt{L/g}/(\AGM(1,w)); \print(w,Ts);
L = 1;
g = 9.80665;
T= \value(Ts); \print(T);
```

The results:
$w = 0.5$
$Ts = 2.7458 * \pi * (L/g)^{(1/2)}$
$T = 2.7546$.