The KAlgebra Handbook

Aleix Pol

Contents

Abstract

KAlgebra is an application that can replace your graphing calculator. It has numerical, logical, symbolic, and analysis features that let you calculate mathematical expressions on the calculator and graphically plot the results in 2D or 3D. KAlgebra is rooted in the Mathematical Markup Language (MathML); however, one does not need to know MathML to use KAlgebra.

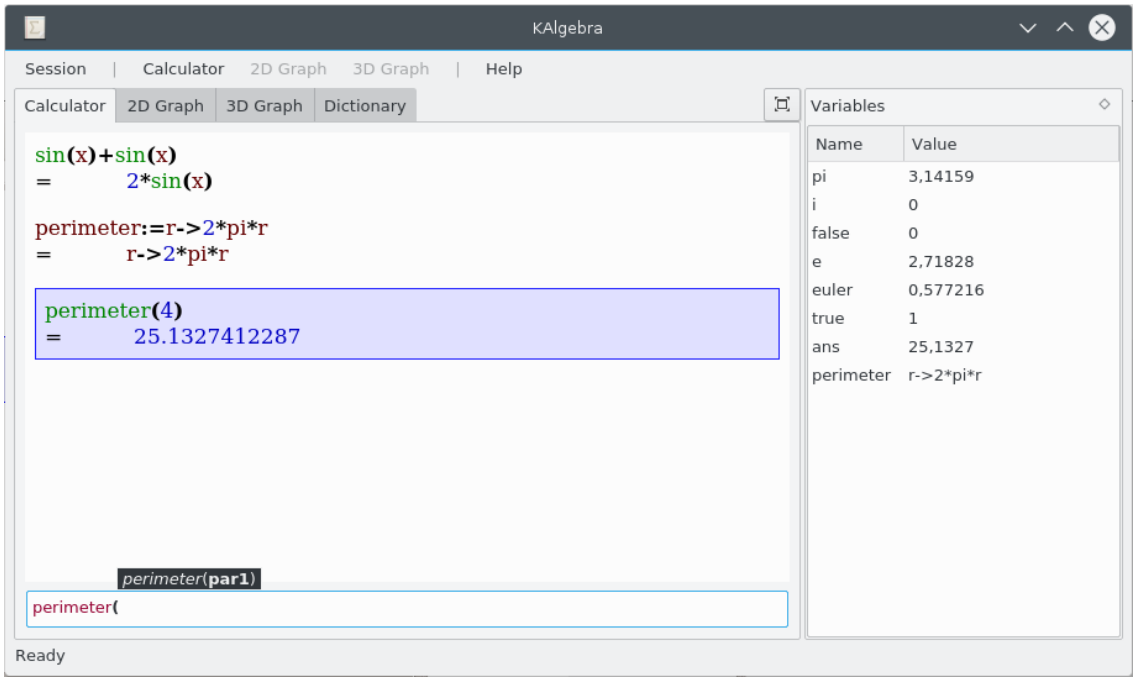
Chapter 1

Introduction

KAlgebra has numerous features that allow the user to perform all sorts of mathematical operations and show graphically. At one time, this program was MathML oriented. Now it can be used by anyone with a little mathematical knowledge to solve simple and advanced problems alike.
It includes such features as:

- A calculator for quick and easy evaluation of math functions.
- Scripting capability for advanced series of calculations
- Language capabilities including function definition and syntax autocompletion.
- Calculus functions including symbolic differentiation, vector calculus, and list manipulation.
- Function plotting with live cursor for graphical root finding and other types of analysis.
- 3D plotting for useful visualization of 3D functions.
- A built-in operator dictionary for quick reference to the many available functions.

Below is a screenshot of the KAlgebra application in action:



When the user begins a KAlgebra session, they are presented with a single window consisting of a **Calculator** tab, a **2D Graph** tab, a **3D Graph** tab and a **Dictionary** tab. Within each tab, you will find an input field to enter your functions or calculations, and a display field which shows the results.

At any time the user may manage their session with the main menu **Session** options:

Session → **New (Ctrl+N)**

Opens a new KAlgebra window.

Session → **Full Screen Mode (Ctrl+Shift+F)**

Toogle full screen mode for KAlgebra window. The full screen mode can also be switched on and off using  button at the top right part of KAlgebra window.

Session → **Quit (Ctrl+Q)**

Shuts the program down.

Chapter 2

Syntax

KAlgebra uses an intuitive algebraic syntax for entering user functions, similar to that used on most modern graphing calculators. This section lists the fundamental built-in operators available in KAlgebra. The author of KAlgebra modeled this syntax after Maxima and maple for users that may be familiar with these programs.

For users that are interested in the inner workings of KAlgebra, user entered expressions are converted to MathML on the backend. A rudimentary understanding of the capabilities supported by MathML will go a long way toward revealing the inner capabilities of KAlgebra.

Here is a list of the available operators we have by now:

- + - * / : Addition, subtraction, multiplication and division.
- ^, **: Power, you can use them both. Also it is possible to use the unicode ² characters. Powers are one way to make roots too, you can do it like: a**(1/b)
- -> : lambda. It is the way to specify one or more free variables that will be bound in a function. For example, in the expression, `length:=(x,y)->(x*x+y*y)^0.5`, the lambda operator is used to denote that x and y will be bound when the length function is used.
- x=a..b : This is used when we need to delimit a range (bounded variable+uplimit+downlimit). This means that x goes from a to b.
- () : It is used to specify a higher priority.
- abc(params) : Functions. When the parser finds a function, it checks if abc is an operator. If it is, it will be treated as an operator, if it is not, it will be treated as a user function.
- := : Definition. It is used to define a variable value. You can do things like x:=3, x:=y being y defined or not or perimeter:=r->2*pi*r.
- ? : Piecewise condition definition. Piecewise is the way we can define conditional operations in KAlgebra. Put another way, this is a way of specifying an if, elseif, else condition. If we introduce the condition before the '?' it will use this condition only if it is true, if it finds a '?' without any condition, it will enter in the last instance. Example: piecewise { x=0 ? 0, x=1 ? x+1, ? x**2 }
- { } : MathML container. It can be used to define a container. Mainly useful for working with piecewise.
- = > >= < <= : Value comparators for equal, greater, greater or equal, less and less or equal respectively

Now you could ask me, why should the user mind about MathML? That's easy. With this, we can operate with functions like cos(), sin(), any other trigonometrical functions, sum() or product(). It does not matter what kind it is. We can use plus(), times() and everything which has its operator. Boolean functions are implemented as well, so we can do something like or(1,0,0,0,0).

Chapter 3

Using the Calculator

KAlgebra's calculator is useful as a calculator on steroids. The user may enter expressions for evaluation in **Calculate** or **Evaluate** mode, depending on the **Calculator** menu selection.

In evaluation mode KAlgebra simplifies the expression even if it sees an undefined variable. When in calculation mode KAlgebra, calculates everything and if it finds an undefined variable shows an error.

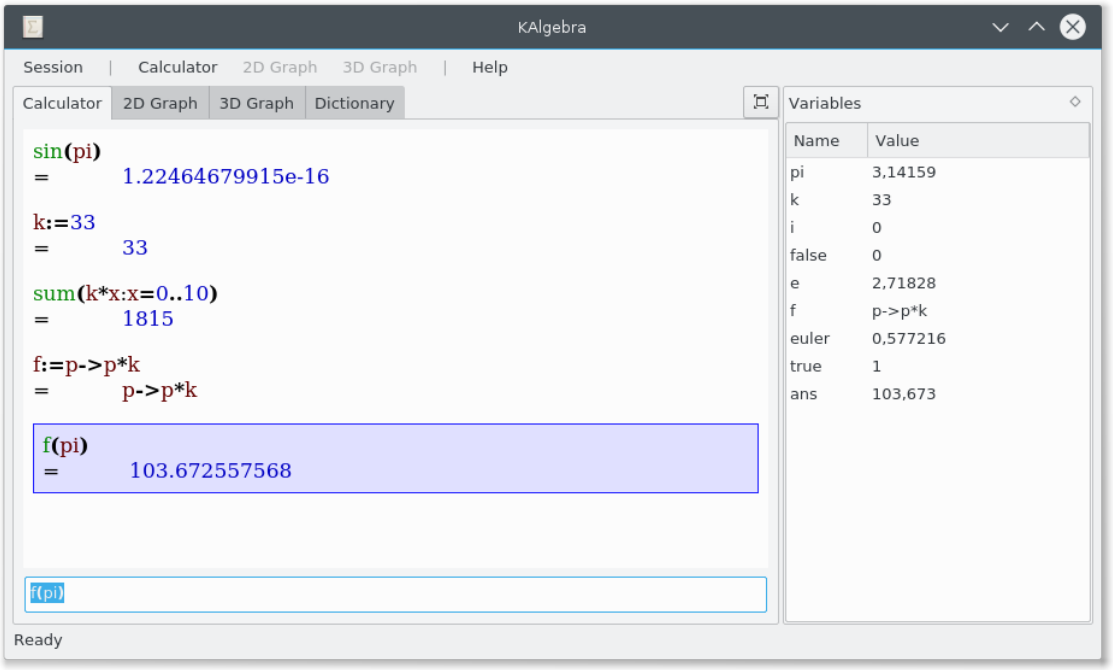
In addition to displaying the user entered equations and results in the Calculator display, all variables that are declared are displayed in a persistent frame to the right. By double clicking on a variable you will see a dialog that lets you change their values (just a way to trick the log).

The ˝ans˝ variable is special, every time you enter an expression, the ˝ans˝ variable value will be changed to the last result.

The following are example functions that can be entered in the input field of the calculator window:

- sin(pi)
- k:=33
- sum(k*x : x=0..10)
- f:=p->p*k
- f(pi)

The following shows a screenshot of the calculator window after entering the above example expressions:



A user can control the execution of a series of calculations using the **Calculator** menu options:

Calculator → Load Script (Ctrl+L)

Executes the instructions in a file sequentially. Useful if you want to define some libraries or resume some previous work.

Calculator → Save Script (Ctrl+G)

Saves the instructions you have typed since the session began to be able to reuse. Generates text files so it should be easy to fix using any text editor, like Kate.

Calculator → Export Log (Ctrl+S)

Saves the log with all results into an HTML file to be able to print or publish.

Chapter 4

2D Graphs

To add a new 2D graph on KAlgebra, select the **2D Graph** tab and click the **Add** tab to add a new function. Your focus will go to an input text box where you can type your function.

4.1 Syntax

If you want to use a typical f(x) function it is not necessary to specify it, but if you want a f(y) or a polar function, you will have to add y-> and q-> as the bounded variables.

Examples:

- sin(x)
- x^2
- y->sin(y)
- q->3*sin(7*q)
- t->vector{sin t, t**2}

If you have entered the function click on the **OK** button to display the graph in the main window.

4.2 Features

You can set several graphs on the same view. Just use the **Add** button when you are in List mode. You can set each graph its own color.

The view can be zoomed and moved with the mouse. Using the wheel you can zoom in and out. You can also select an area with the left button of the mouse and this area will be zoomed in. Move the view with the arrow keys.

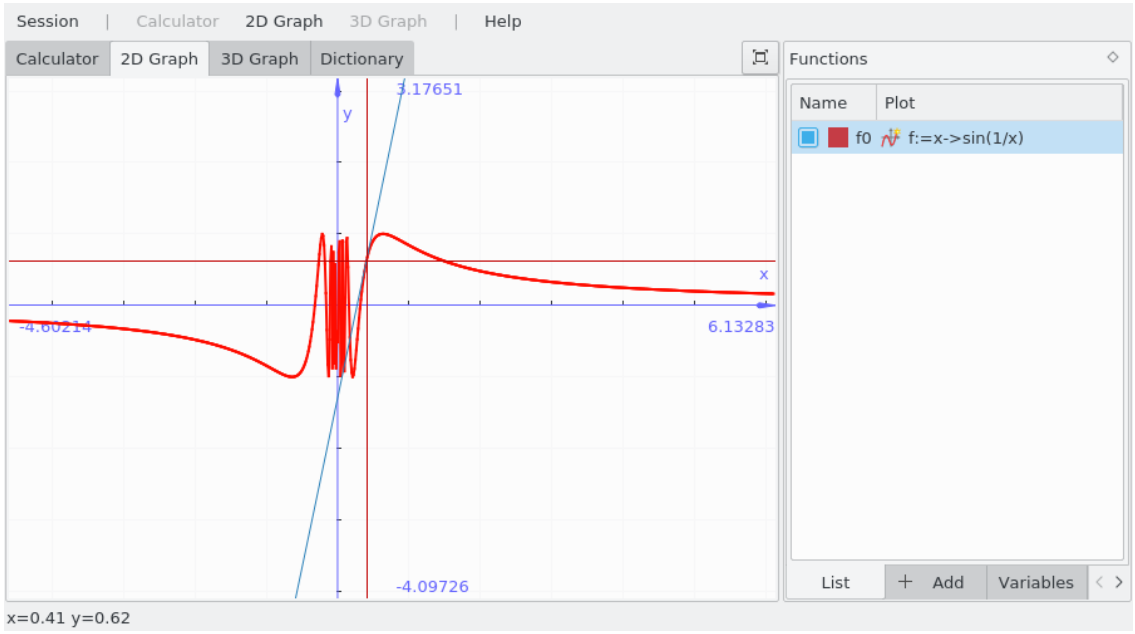
> NOTE
> The viewport of 2D graphs can be explicitly defined using the **Viewport** tab on a **2D Graph** tab.

In the **List** tab, you can open an **Editing** tab to edit or remove a function with double-click and check or uncheck the check box next to the function name to show or hide it.

In the **2D Graph** menu you find these options:

- Show or hide the grid
- Keep the aspect ratio while zooming
- Zoom in (**Ctrl**++) and zoom out (**Ctrl**+-)
- Save (**Ctrl**+**S**) the graph as image file
- Reset the view to the original zoom
- Select a resolution for the graphs

Below is a screenshot of a user who's cursor is at the rightmost root of the function, sin(1/x). The user who graphed it used very fine resolution to make this graph (as it oscillates at higher and higher frequency near the origin). There is also a live cursor feature where whenever you move your cursor over a spot, it shows you the x and y values in the bottom left corner of the screen. A live ″tangent line″ is plotted on the function at the live cursor location.



Chapter 5

3D Graphs

To draw a 3D Graph with KAlgebra select the **3D Graph** tab and you will see an input field at the bottom where you will type your function. Z cannot be defined yet. For the moment KAlgebra only supports 3D graphs explicitly dependent only on the x and y, such as (x,y)->x*y, where z=x*y.

Examples:

- (x,y)->sin(x)*sin(y)
- (x,y)->x/y

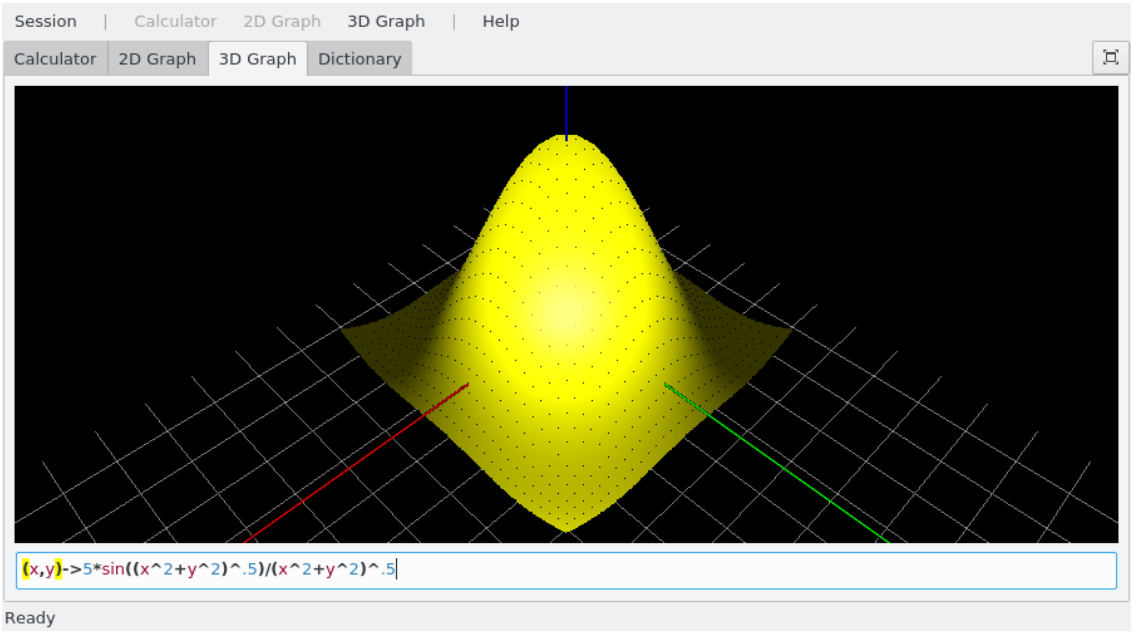
The view can be zoomed and moved with the mouse. Using the wheel you can zoom in and out. Hold the left mouse button and move the mouse to rotate the graph.

The left and right arrow keys rotate the graph around the z axis, the up and down arrow keys rotate around the horizontal axis of the view. Press **W** to zoom in the plot and **S** to zoom it out.

In the **3D Graph** menu you find these options:

- Save (**Ctrl**+**S**) the graph as image file
- Reset the view to the original zoom in the 3D graph menu
- You can draw the graphs with dots, lines or solid styles in the 3D graph menu

Below is a screenshot of the so-called ˝sombrero˝ function. This particular graph is shown using the 3D graph line-style.

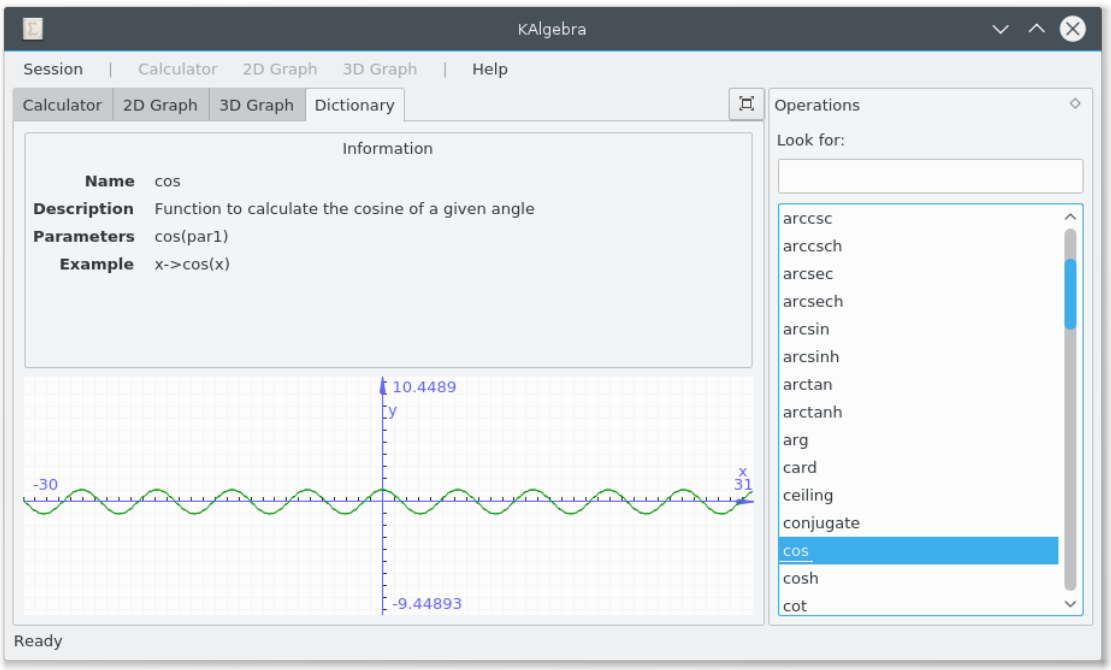

Session | Calculator 2D Graph 3D Graph | Help
Calculator 2D Graph 3D Graph Dictionary
(x,y)->5*sin((x^2+y^2)^.5)/(x^2+y^2)^.5
Ready

Chapter 6

Dictionary

The dictionary provides a list of all KAlgebra built in functions. It can be used to find the definition of an operation and its input parameters. It's a useful place to go to find the many capabilities of KAlgebra.

Below is a screenshot of the KAlgebra dictionary lookup of the cosine function.



Chapter 7

Commands supported by KAlgebra

7.1 plus

- Name: plus
- Description: Addition
- Parameters: plus(... parameters, ...)
- Example: x->x+2

7.2 times

- Name: times
- Description: Multiplication
- Parameters: times(... parameters, ...)
- Example: x->x*2

7.3 minus

- Name: minus
- Description: Subtraction. Will remove all values from the first one.
- Parameters: minus(... parameters, ...)
- Example: x->x-2

7.4 divide

- Name: divide
- Description: Division
- Parameters: divide(par1, par2)
- Example: x->x/2

7.5 quotient

- Name: quotient
- Description: Quotient
- Parameters: quotient(par1, par2)
- Example: x->quotient(x, 2)

7.6 power

- Name: power
- Description: Power
- Parameters: power(par1, par2)
- Example: x->x^2

7.7 root

- Name: root
- Description: Root
- Parameters: root(par1, par2)
- Example: x->root(x, 2)

7.8 factorial

- Name: factorial
- Description: Factorial. factorial(n)=n!
- Parameters: factorial(par1)
- Example: x->factorial(x)

7.9 and

- Name: and
- Description: Boolean and
- Parameters: and(... parameters, ...)
- Example: x->piecewise { and(x>-2, x<2) ? 1, ? 0 }

7.10 or

- Name: or
- Description: Boolean or
- Parameters: or(... parameters, ...)
- Example: x->piecewise { or(x>2, x>-2) ? 1, ? 0 }

7.11 xor

- Name: xor
- Description: Boolean xor
- Parameters: xor(... parameters, ...)
- Example: x->piecewise { xor(x>0, x<3) ? 1, ? 0 }

7.12 not

- Name: not
- Description: Boolean not
- Parameters: not(par1)
- Example: x->piecewise { not(x>0) ? 1, ? 0 }

7.13 gcd

- Name: gcd
- Description: Greatest common divisor
- Parameters: gcd(... parameters, ...)
- Example: x->gcd(x, 3)

7.14 lcm

- Name: lcm
- Description: Least common multiple
- Parameters: lcm(... parameters, ...)
- Example: x->lcm(x, 4)

7.15 rem

- Name: rem
- Description: Remainder
- Parameters: rem(par1, par2)
- Example: x->rem(x, 5)

7.16 factorof

- Name: factorof
- Description: The factor of
- Parameters: factorof(par1, par2)
- Example: x->factorof(x, 3)

7.17 max

- Name: max
- Description: Maximum
- Parameters: max(... parameters, ...)
- Example: x->max(x, 4)

7.18 min

- Name: min
- Description: Minimum
- Parameters: min(... parameters, ...)
- Example: x->min(x, 4)

7.19 lt

- Name: lt
- Description: Less than. lt(a,b)=a<b
- Parameters: lt(par1, par2)
- Example: x->piecewise { x<4 ? 1, ? 0 }

7.20 gt

- Name: gt
- Description: Greater than. gt(a,b)=a>b
- Parameters: gt(par1, par2)
- Example: x->piecewise { x>4 ? 1, ? 0 }

7.21 eq

- Name: eq
- Description: Equal. eq(a,b) = a=b
- Parameters: eq(par1, par2)
- Example: x->piecewise { x=4 ? 1, ? 0 }

7.22 neq

- Name: neq
- Description: Not equal. neq(a,b)=a≠b
- Parameters: neq(par1, par2)
- Example: x->piecewise { x!=4 ? 1, ? 0 }

7.23 leq

- Name: leq
- Description: Less or equal. leq(a,b)=a≤b
- Parameters: leq(par1, par2)
- Example: x->piecewise { x<=4 ? 1, ? 0 }

7.24 geq

- Name: geq
- Description: Greater or equal. geq(a,b)=a≥b
- Parameters: geq(par1, par2)
- Example: x->piecewise { x>=4 ? 1, ? 0 }

7.25 implies

- Name: implies
- Description: Boolean implication
- Parameters: implies(par1, par2)
- Example: x->piecewise { implies(x<0, x<3) ? 1, ? 0 }

7.26 approx

- Name: approx
- Description: Approximation. approx(a)=a±n
- Parameters: approx(par1, par2)
- Example: x->piecewise { approx(x, 4) ? 1, ? 0 }

7.27 abs

- Name: abs
- Description: Absolute value. abs(n)=|n|
- Parameters: abs(par1)
- Example: x->abs(x)

7.28 floor

- Name: floor
- Description: Floor value. floor(n)=⌊n⌋
- Parameters: floor(par1)
- Example: x->floor(x)

7.29 ceiling

- Name: ceiling
- Description: Ceil value. ceil(n)=⌈n⌉
- Parameters: ceiling(par1)
- Example: x->ceiling(x)

7.30 sin

- Name: sin
- Description: Function to calculate the sine of a given angle
- Parameters: sin(par1)
- Example: x->sin(x)

7.31 cos

- Name: cos
- Description: Function to calculate the cosine of a given angle
- Parameters: cos(par1)
- Example: x->cos(x)

7.32 tan

- Name: tan
- Description: Function to calculate the tangent of a given angle
- Parameters: tan(par1)
- Example: x->tan(x)

7.33 sec

- Name: sec
- Description: Secant
- Parameters: sec(par1)
- Example: x->sec(x)

7.34 csc

- Name: csc
- Description: Cosecant
- Parameters: csc(par1)
- Example: x->csc(x)

7.35 cot

- Name: cot
- Description: Cotangent
- Parameters: cot(par1)
- Example: x->cot(x)

7.36 sinh

- Name: sinh
- Description: Hyperbolic sine
- Parameters: sinh(par1)
- Example: x->sinh(x)

7.37 cosh

- Name: cosh
- Description: Hyperbolic cosine
- Parameters: cosh(par1)
- Example: x->cosh(x)

7.38 tanh

- Name: tanh
- Description: Hyperbolic tangent
- Parameters: tanh(par1)
- Example: x->tanh(x)

7.39 sech

- Name: sech
- Description: Hyperbolic secant
- Parameters: sech(par1)
- Example: x->sech(x)

7.40 csch

- Name: csch
- Description: Hyperbolic cosecant
- Parameters: csch(par1)
- Example: x->csch(x)

7.41 coth

- Name: coth
- Description: Hyperbolic cotangent
- Parameters: coth(par1)
- Example: x->coth(x)

7.42 arcsin

- Name: arcsin
- Description: Arc sine
- Parameters: arcsin(par1)
- Example: x->arcsin(x)

7.43 arccos

- Name: arccos
- Description: Arc cosine
- Parameters: arccos(par1)
- Example: x->arccos(x)

7.44 arctan

- Name: arctan
- Description: Arc tangent
- Parameters: arctan(par1)
- Example: x->arctan(x)

7.45 arccot

- Name: arccot
- Description: Arc cotangent
- Parameters: arccot(par1)
- Example: x->arccot(x)

7.46 arccosh

- Name: arccosh
- Description: Hyperbolic arc cosine
- Parameters: arccosh(par1)
- Example: x->arccosh(x)

7.47 arccsc

- Name: arccsc
- Description: Arc cosecant
- Parameters: arccsc(par1)
- Example: x->arccsc(x)

7.48 arccsch

- Name: arccsch
- Description: Hyperbolic arc cosecant
- Parameters: arccsch(par1)
- Example: x->arccsch(x)

7.49 arcsec

- Name: arcsec
- Description: Arc secant
- Parameters: arcsec(par1)
- Example: x->arcsec(x)

7.50 arcsech

- Name: arcsech
- Description: Hyperbolic arc secant
- Parameters: arcsech(par1)
- Example: x->arcsech(x)

7.51 arcsinh

- Name: arcsinh
- Description: Hyperbolic arc sine
- Parameters: arcsinh(par1)
- Example: x->arcsinh(x)

7.52 arctanh

- Name: arctanh
- Description: Hyperbolic arc tangent
- Parameters: arctanh(par1)
- Example: x->arctanh(x)

7.53 exp

- Name: exp
- Description: Exponent (eˆx)
- Parameters: exp(par1)
- Example: x->exp(x)

7.54 ln

- Name: ln
- Description: Base-e logarithm
- Parameters: ln(par1)
- Example: x->ln(x)

7.55 log

- Name: log
- Description: Base-10 logarithm
- Parameters: log(par1)
- Example: x->log(x)

7.56 conjugate

- Name: conjugate
- Description: Conjugate
- Parameters: conjugate(par1)
- Example: x->conjugate(x*i)

7.57 arg

- Name: arg
- Description: Arg
- Parameters: arg(par1)
- Example: x->arg(x*i)

7.58 real

- Name: real
- Description: Real
- Parameters: real(par1)
- Example: x->real(x*i)

7.59 imaginary

- Name: imaginary
- Description: Imaginary
- Parameters: imaginary(par1)
- Example: x->imaginary(x*i)

7.60 sum

- Name: sum
- Description: Summatory
- Parameters: sum(par1 : var=from..to)
- Example: x->x*sum(t*t:t=0..3)

7.61 product

- Name: product
- Description: Productory
- Parameters: product(par1 : var=from..to)
- Example: x->product(t+t:t=1..3)

7.62 diff

- Name: diff
- Description: Differentiation
- Parameters: diff(par1 : var)
- Example: x->(diff(x^2:x))(x)

7.63 card

- Name: card
- Description: Cardinal
- Parameters: card(par1)
- Example: x->card(vector { x, 1, 2 })

7.64 scalarproduct

- Name: scalarproduct
- Description: Scalar product
- Parameters: scalarproduct(... parameters, ...)
- Example: x->scalarproduct(vector { 0, x }, vector { x, 0 })[1]

7.65 selector

- Name: selector
- Description: Select the par1-th element of par2 list or vector
- Parameters: selector(par1, par2)
- Example: x->scalarproduct(vector { 0, x }, vector { x, 0 })[1]

7.66 union

- Name: union
- Description: Joins several items of the same type
- Parameters: union(... parameters, ...)
- Example: x->union(list { 1, 2, 3 }, list { 4, 5, 6 })[rem(floor(x), 5)+3]

7.67 forall

- Name: forall
- Description: For all
- Parameters: forall(par1 : var)
- Example: x->piecewise { forall(t:t@list { true, false, false }) ? 1, ? 0 }

7.68 exists

- Name: exists
- Description: Exists
- Parameters: exists(par1 : var)
- Example: x->piecewise { exists(t:t@list { true, false, false }) ? 1, ? 0 }

7.69 map

- Name: map
- Description: Applies a function to every element in a list
- Parameters: map(par1, par2)
- Example: x->map(x->x+x, list { 1, 2, 3, 4, 5, 6 })[rem(floor(x), 5)+3]

7.70 filter

- Name: filter
- Description: Removes all elements that don't fit a condition
- Parameters: filter(par1, par2)
- Example: x->filter(u->rem(u, 2)=0, list { 2, 4, 3, 4, 8, 6 })[rem(floor(x), 5)+3]

7.71 transpose

- Name: transpose
- Description: Transpose
- Parameters: transpose(par1)
- Example: x->transpose(matrix { matrixrow { 1, 2, 3, 4, 5, 6 } })[rem(floor(x), 5)+3][1]

Chapter 8

Credits and License