Andreas Rejbrand

# *Introduction to Algorithm Simulator*

Version 1.0.0.3 – World English Edition

# *Table of Contents*

## *About this document*

This document is meant to introduce the mathematical software Algorithm Simulator (hereinafter called "AlgoSim"). AlgoSim is a numerical calculator with features such as function plotting, one variable statistical analysis, and user-defined algorithms. The program has calculus features, represented by numerical differentiation, integration and localization of stationary points. Furthermore, AlgoSim is able to numerically solve equations, compute sums and products and create tables of functions.

## *Contact Information*

AlgoSim is created by Andreas Rejbrand, and new versions of the application will be published on the World Wide Web:

http://english.rejbrand.se/algosim

All questions and comments, positive as well as negative, are appreciated, and can be sent to Andreas Rejbrand via e-mail at andreas@rejbrand.se.

# *Numerical Computation using the Console*

The first tab in AlgoSim, called "Console," can be used for basic numerical computations. In fact, most non-graphical features can be used from this tab, except for functions that do not use *real numbers* as arguments, e.g. differentiation and integration, which require a *function* as an argument (the function that is to be differentiated or integrated).

In order to perform a computation, write a proper algebraic expression in the console and press Enter. AlgoSim will then compute the expression and write the result below the expression, at a new line. By default, expressions written by the users (i.e. inputs) are coloured black, whereas results written by AlgoSim (i.e. outputs) are coloured green.

AlgoSim recognises proper algebraic expressions. An *expression* is defined as anything returning a real number. Examples of expressions are "123" and "sin(0.6)". Expressions may contain real numbers, brackets, operators, functions, and variables. Please note, that mathematical constants technically are variables.

Real numbers can use a sign (+ or −) as prefix and may contain a decimal point (.). The special operator "E" can be used as "times 10 raised to the power of." The exponent, i.e. the real number entered directly after "E", can also use a sign but must be an integer.

Brackets are used for changing the computation priority within expressions; brackets are computed first. The brackets available are ( … ), [ … ], and { … }. All brackets must be closed, i.e. a starting bracket must be followed by an ending bracket. AlgoSim allows you to use infinitely many levels of brackets. Allowing function and variable identifiers consisting of more than one character, however, AlgoSim cannot accept the shorthand of multiplication, such in "9x" and "5sin(2)". Instead, one must always write out the multiplication signs, such in "9×x" and "5×sin(2)".

## *Table of Priority*

AlgoSim applies the following priority:

1. ( … ), [ … ], { … }
2. f(*x*)
3. !
4. ^
5. ×, /
6. +, -

Concerning operations having the same priority, expressions are computed in a left-to-right direction. For instance, 100/2/5 = (100/2)/5.

## *Operators*

Operators are symbols performing operations on preceding and following numbers, referred to as operands. An operator is either unary or binary. A unary operator requires one operand, whereas a binary operator requires two operands. The operands recognised by AlgoSim are listed below. (R is the set of real numbers, N the set of the natural numbers (the positive integers and the number 0) and Z the set of all integers.)

| *Operator* | *Type* | *Operands* | *Description* | *Example* |
|---|---|---|---|---|
| + | Binary | R | Addition | 10+30=40 |
| - | Binary | R | Subtraction | 40-30=10 |
| ×, * or · | Binary | R | Multiplication | 20×50=1000 |
| / | Binary | R[1] | Division | 1000/50=20 |
| ^ | Binary | R[2] | Raised to the power of | 2^10=1024 |
| ! | Unary | N | Factorial | 5!=120 |

***Table 1.*** Operators

[1] Consider $a/b$. $b \neq 0$.
[2] Consider $a\hat{}b$. If $a < 0$ then $b \in Z$.

## *Variables*

In AlgoSim, both constants and variables are referred to as "variables." The identifier ("name") of a variable can include the characters A…Z, a…z, 0…9 and _ (underscore). Big letters and small letters are not equivalent; "width" and "Width" are two different variables. In AlgoSim, a number of mathematical constants are pre-defined.

| ***Name*** | ***Value*** | ***Description*** |
|---|---|---|
| Pi | 3.14159265358979324 | The mathematical constant $\pi$ (pi); the ratio between the circumference and radius of a circle |
| e | 2.71828182845904524 | The mathematical constant $e$ |
| Phi | 1.61803398874989485 | The mathematical constant $\phi$ (phi); "the golden ratio" |

***Table 2.*** Pre-defined constants

## *Built-in functions*

In AlgoSim, functions can receive an arbitrary number of arguments greater than zero. The identifier of a function can include the characters A…Z, a…z, 0…9 and _ (underscore). Besides offering a large number of built-in functions, AlgoSim allows the user to define new variables. On the following pages, the built-in function will be described.

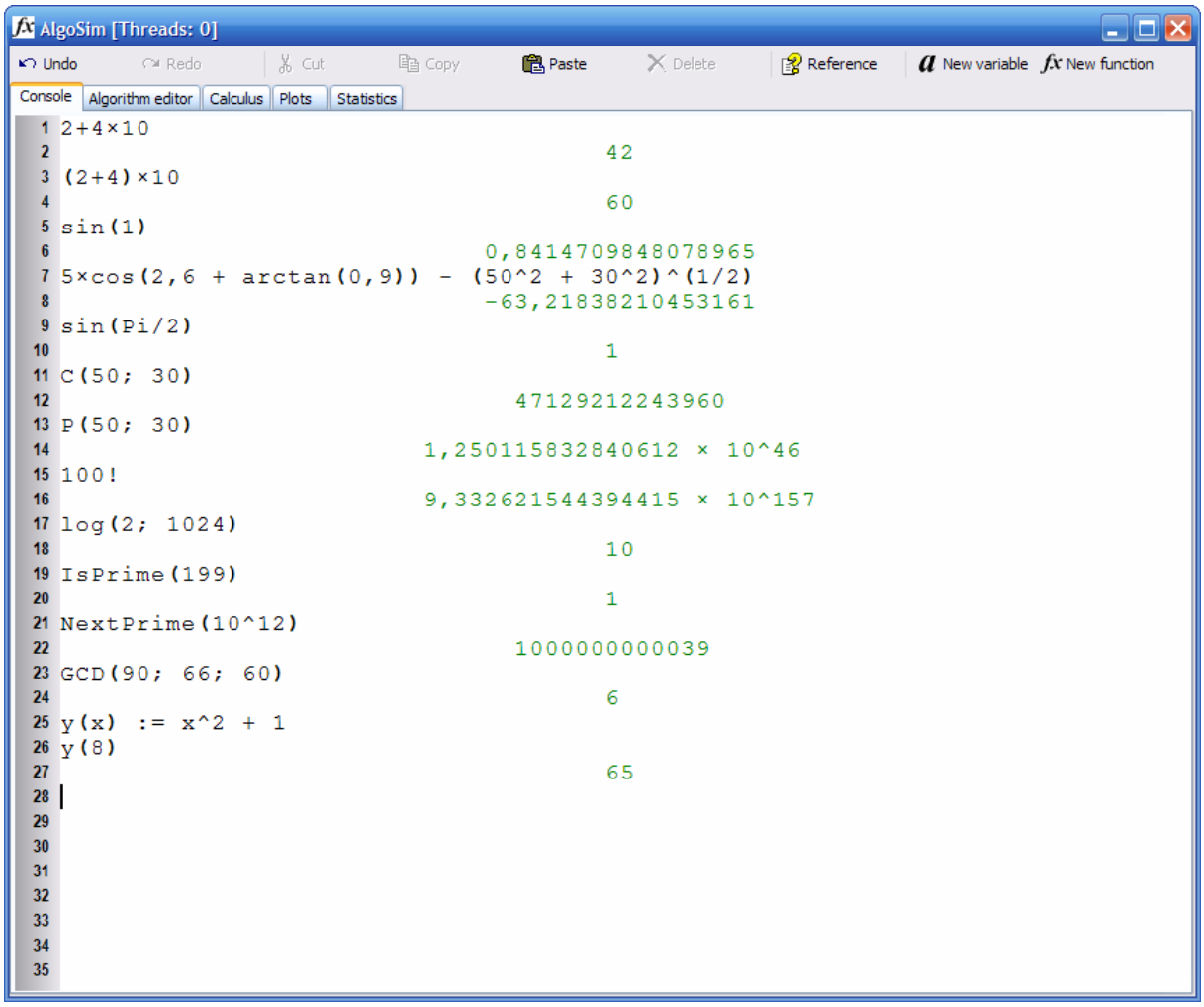



***Image 1.*** The console

### *Trigonometric and hyperbolic functions*

*Trigonometric functions*

sin(*x*), cos(*x*), tan(*x*), cot(*x*) return the sine, cosine, tangent, and cotangent of *x*, respectively.

Example:

```
sin(Pi/2)
                                    1

cos(2.9)
                        -0.9709581651495905
```

*Inverse trigonometric function*

arcsin(*x*), arccos(*x*), arctan(*x*), arccot(*x*) return the arcsine, arccosine, arctangent, and arccotangent of *x*, respectively.

Example:

```
arcsin(sin(0.85))
                                   0.85
```

*Hyperbolic functions*

sinh(*x*), cosh(*x*), tanh(*x*), coth(*x*) return the sine hyperbolicus, cosine hyperbolicus, tangent hyperbolicus, and cotangent hyperbolicus of *x*, respectively.

*Inverse hyperbolic functions*

arcsinh(*x*), arccosh(*x*), arctanh(*x*), arccoth(*x*) return the arcsine hyperbolicus, arccosine hyperbolicus, arctangent hyperbolicus, and arccotangent hyperbolicus of *x*, respectively.

### *Logarithmical functions*

*The logarithmical function*

log(*n*; *x*) returns the *n*th logarithm of *x*, i.e. the number *a* satisfying $n^a = x$.

Example:

```
log(0.5; 0.002)
                    8.96578428466208704
```

*The decadic logarithm function*

log10(*x*) returns the decadic logarithm of *x*, i.e. the number *a* satisfying $10^a = x$.

Example:

```
log10(2.5E-5)
                    -4.60205999132796239
```

*The binary logarithm function*

log2(*x*) returns the binary logarithm of *x*, i.e. the number *a* satisfying $2^a = x$.

Example:

```
log2(1024)
                    10
```

*The natural logarithm*

ln(*x*) returns the natural logarithm (the *e* logarithm) of *x*, i.e. the number *a* satisfying $e^a = x$.

Example:

```
ln(9.82)
                    2.28442112236637452
```

### *The Exponential Function*

exp(*x*) returns $e^x$.

Example:

```
exp(1)
                    2.71828182845904524
```

### *The Absolute Value Function*

$\text{abs}(x)$ returns $x$ without its sign. $\text{abs}(x) = \begin{cases} x & \text{if } x \geq 0 \\ -x & \text{if } x < 0 \end{cases}$

Example:

```
abs(50)
```

```
50
```

```
abs(-50)
```

```
50
```

### *The Sign Function*

$\text{sgn}(x)$ returns the sign of $x$. $\text{sgn}(x) = \begin{cases} -1 & \text{if } x < 0 \\ 0 & \text{if } x = 0 \\ 1 & \text{if } x > 0 \end{cases}$

Example:

```
sgn(50)
```

```
1
```

```
sgn(-50)
```

```
-1
```

```
sgn(0)
```

```
0
```

### *Rounding functions*

*The Default Rounding Function*

round($x$) rounds $x$ to the nearest integer. If $x$ precisely lies between the two surrounding integers on the number line, then $x$ will be rounded to the even number.

Example:

```
round(3.2)
```
```
3
```
```
round(3.6)
```
```
4
```
```
round(3.5)
```
```
4
```

*The Floor Function*

floor($x$) rounds $x$ to the nearest (and greatest) integer $\leq x$.

Example:

```
floor(6.8)
```
```
6
```
```
floor(-6.8)
```
```
-7
```

*The Ceiling Function*

ceil($x$) rounds $x$ to the nearest (and smallest) integer $\geq x$.

Example:

```
ceil(6.8)
```
```
7
```
```
ceil(-6.8)
```
```
-6
```

*The Integer Part Function and the Fractional Part Function*

int($x$) returns the integer part of $x$.

frac($x$) returns the fractional part of $x$.

Example:

```
int(6.7)
```

6

```
frac(6.7)
```

0.7

Please notice that $x$ = int($x$) + frac($x$).

### *Combinations and Permutations*

C($a$; $b$) returns the number of groups of $b$ elements that can be created from a set consisting of $a$ elements. The order is irrelevant; thus, the combination A, B, C equals A, C, B.

P($a$; $b$) returns the number of combinations of how $b$ elements can be arranged from a set consisting of $a$ elements. The order is relevant; thus, A, B, C and A, C, B are two different arrangements.

Example:

How many five digit numbers can be created using the digits 1 to 9? A single digit may only occur once in the number.

```
P(9; 5)
```

15120

How many different groups of five persons can be created from a class of 30 students?

```
C(30; 5)
```

142506

### *The Factorial Function*

fact($n$) = $n!$

$$n! = \prod_{k=1}^{n} k \quad \text{where} \quad n \in N$$

Example:

```
fact(5)
```

120

How many five digit numbers can be created using the digits 1 to 9? A single digit may only occur once in the number.

```
9!/4!
```

```
15120
```

### *The Square Root Function*

$\text{sqrt}(x) = x^{1/2}$, i.e. the square root of $x$.

Example:

```
sqrt(25)
                                    5

sqrt(57600)
                                   240
```

An error message is shown if sqrt is called with an argument $x$ such that $x < 0$. Non-real numbers are not supported.

### *Angle Conversion Functions*

*Degrees to radians*

DTR($x$) returns the angle $x$ (given in degrees) given in radians. $\text{DTR}(x) = \frac{\pi}{180}x$.

Example:

```
DTR(270)
                          4.71238898038468986
```

*Radians to degrees*

RTD($x$) returns the angle $x$ (given in radians) given in degrees. $\text{RTD}(x) = \frac{180}{\pi}x$.

Example:

```
RTD(0.952)
                          54.5455820964543695
```

### *Coordinate Conversion Functions*

*Rectangular to polar coordinates*

RTP(*x*; *y*) converts the rectangular coordinates (*x*, *y*) to polar coordinates (*r*, *θ*). The variables *r* and *Theta* contain the results. The value of *r* is instantaneously returned in the console.

Example:

```
RTP(-10; 20)
                    22.360679774997897

Theta
                    2.03444393579570274
```

*Polar to rectangular coordinates*

PTR(*r*, *Theta*) converts the polar coordinates (*r*, *θ*) to rectangular coordinates (*x*, *y*). The variables *x* and *y* are set. The value of *x* is instantaneously returned.

Example:

```
PTR(10; Pi/4)
                    7.07106781186547524

y
                    7.07106781186547525
```

### *Prime Number Functions*

*The Prime Verification Function*

IsPrime(*n*) returns 1 if *n* is a prime; otherwise, 0 is returned.

Example:

```
IsPrime(51)
                                    0

IsPrime(53)
                                    1

IsPrime(2311)
                                    1
```

An error message is shown if $n \notin N$.

*The Next Prime Number Function*

NextPrime(*n*) returns the prime succeeding or equal to *n*, i.e. the smallest prime $p \geq n$.

Example:

```
NextPrime(10^12)
                            1000000000039
```

An error message is shown if $n \notin N$.

*The Previous Prime Number Function*

PrevPrime(*n*) returns the prime succeeded by or equal to *n*, i.e. the greatest prime $p \leq n$.

Example:

```
PrevPrime(10^12)
                            999999999989
```

An error message is shown if $n \notin N$.

The smallest prime number is 2. If PrevPrime is called with an argument $n < 2$, an error message is shown.

*The n*th Prime Number Function

prime(*n*) returns the *n*th prime number. Please note that the function may be slow if $n \geq 100000$. The first prime, prime(1), equals 2.

Example:

```
prime(100000)
```

```
1299709
```

An error message is shown if $n \notin Z^+$.

### *The Greatest Common Divisor Function*

GCD(*a*; *b*; …) returns the greatest common divisor of the arguments *a*, *b*, …

Example:

```
GCD(50; 25; 75)
                                    25
```

If GCD(*a*; *b*; …) = 1, then the arguments are coprime.

Example:

```
GCD(7; 14; 50)
                                    1
```

An error message is shown if one or more of the arguments ∉ Z.

Moreover, negative integers and the number 0 may be used as arguments.

Example:

```
GCD(-40; -120; -480)
                                    40

GCD(0; 482; 846)
                                    2
```

If all arguments equal 0, an error message is shown.

### *The Fibonacci Number Function*

Numbers generated by the function *f* where $f(n) = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \text{ and} \\ f(n-1) + f(n-2) & \text{if } n \geq 2 \end{cases}$

$n \in N$ are called *fibonacci numbers*.

fibonacci(*n*) returns the *n*th Fibonacci number.

An error message is shown if $n \notin N$.

### *Random Number Functions*

*The Random Integer Function*

Random($n$) randomly chooses an integer $x$, satisfying $1 \leq x \leq n$. Thus, the probability that a particular number is chosen, equals $1/n$.

Example (Notice! The outputs vary!):

```
Random(5)
                                    3

Random(5)
                                    1
```

An error message is shown if Random is called with an argument $n \notin N$. Random(0) = 1.

*The Random Real Number Function*

RandomReal($x$) randomly selects a real number $a$, such that $0 < a < x$.

Example (Notice! The outputs vary!):

```
RandomReal(5)
                        3.22027781046926975

RandomReal(5)
                        4.99185574357397854
```

### *Statistical functions*

Each of the statistical functions accepts an arbitrary number of real numbers – the statistical data – as argument. Please notice, that in this section, the same statistical material – i.e. the same argument – is used in all examples.

*Number of observations*

n(*a*; *b*; …) returns the number of observations, i.e. the number of arguments.

Example:

```
n(2.9; 6.8; 1.7; 6.5; 2.9; 4.5; 6.8; 3.8)
                                  8
```

*Sum*

sum(*a*; *b*; …) returns the sum of all arguments.

Example:

```
sum(2.9; 6.8; 1.7; 6.5; 2.9; 4.5; 6.8; 3.8)
                                35.9
```

*Product*

product(*a*; *b*; …) returns the product of all arguments.

Example:

```
product(2.9; 6.8; 1.7; 6.5; 2.9; 4.5; 6.8; 3.8)
                              73480.518072
```

*Standard Deviation*

SD(*a*; *b*; …) returns the standard deviation of all arguments.

Example:

```
SD(2.9; 6.8; 1.7; 6.5; 2.9; 4.5; 6.8; 3.8)
                    1.87178891705234755
```

*Standard Deviation (n−1)*

SDNM1(*a*; *b*; …) returns the n − 1 standard deviation of all arguments.

Example:

```
SDNM1(2.9; 6.8; 1.7; 6.5; 2.9; 4.5; 6.8; 3.8)
```

```
2.00102652227728927
```

*Mean*

mean(*a*; *b*; …) returns the mean of all arguments.

Example:

```
mean(2.9; 6.8; 1.7; 6.5; 2.9; 4.5; 6.8; 3.8)
                    4.4875
```

*Smallest value*

min(*a*; *b*; …) returns the smallest number of all arguments.

Example:

```
min(2.9; 6.8; 1.7; 6.5; 2.9; 4.5; 6.8; 3.8)
                    1.7
```

*The First Quartile*

Q1(*a*; *b*; …) returns the first quartile of all arguments.

Example:

```
Q1(2.9; 6.8; 1.7; 6.5; 2.9; 4.5; 6.8; 3.8)
                    2.9
```

*Median*

Md(*a*; *b*; …) returns the median, or the second quartile, of all arguments.

Example:

```
Md(2.9; 6.8; 1.7; 6.5; 2.9; 4.5; 6.8; 3.8)
                    4.15
```

*The Third Quartile*

Q3(*a*; *b*; …) returns the third quartile of all arguments.

Example:

```
Q3(2.9; 6.8; 1.7; 6.5; 2.9; 4.5; 6.8; 3.8)
                    6.65
```

*Greatest value*

max(*a*; *b*; …) returns the greatest number of all arguments.

Example:

```
max(2.9; 6.8; 1.7; 6.5; 2.9; 4.5; 6.8; 3.8)
                                    6.8
```

*The mode frequency*

modefreq(*a*; *b*; …) returns the frequency of the mode value(s).

Example:

```
modefreq(2.9; 6.8; 1.7; 6.5; 2.9; 4.5; 6.8; 3.8)
                                    2
```

(The mode values are 6.8 and 2.9, both occurring twice.)

### *The Modulo Function*

mod(*a*; *b*) returns the remainder of the integer division *a*/*b*.

Example:

```
mod(10; 4)
                                    2

mod(268; 42)
                                   16

mod(10; 5)
                                    0
```

## *Declaring new variables*

In AlgoSim, the user is able to declare new variables[3]. This is most easily done in the console using the special assignment operator, :=. To declare a variable and assign it a value, use the following syntax:

```
Indentifier := Expression
```

Example:

```
a := 50
b := a/10
Width := 100/(2×Pi)
```

Declared variables can be used everywhere in the application.

Example:

```
Width × sin(b)
                    -15,2617538363448816
```

The value of a user-defined variable can be altered afterwards. Example:

```
a := 50
a := 100
a + 1
                                101
```

### *Remove a declared variable*

In order to remove a declared variable, the special command #delete may be called, using the following syntax:

```
#delete Identifier
```

Example:

```
#delete a
#delete b
#delete Width
```

### *Remove all declared variables*

To remove all declared variables, simply call the command #deleteAllVars.

Example:

```
#deleteAllVars
```

[3] The built-in constants Pi, Phi and e, however, are protected and can neither be overridden nor deleted.

Variables can also be declared using the dialogue box "New variable" via the "New variable" toolbar button or the Ctrl+Alt+V shortcut.

## *Defining new functions*

AlgoSim allows the user to define new functions with arbitrary number of real arguments, using the syntax

```
Identifier(Arg1; Arg2; …; ArgN) := Expression
```

where the expression may contain Arg1, Arg2, …, and ArgN. User-defined functions can be used everywhere in the application and they may be redefined.

Example:

```
y(x) := x^2
y(6)
                                36

y(9)
                                81

f(a; b) := (a + 2×b) / 3
f(5; 7)
                    6.33333333333333333

10 + f(9; y(5))
                    29.6666666666666667
```

If a function – irrespective of it being a built-in or a user-defined one – is called with an unintended number of arguments, an error message is shown.

Functions may also be defined using the dialogue box "New function" via the "New function" toolbar button or the Ctrl+Alt+F shortcut.

> ***Note!***
>
> In AlgoSim, the user is allowed to create a function with the same identifier as an already existing variable and vice versa – the function is anyway recognized by being preceded by "(".

Example:

```
y := 50
y(x) := x^2
y
                                50

y(4)
                                16

y(y)
                              2500
```

## *Writing decimal numbers as fractions*

AlgoSim can convert the most recent output to the form $p/q$ where $p$ and $q \in \mathbb{Z}$ and are co-prime. This process, however, may malfunction due to approximation errors. To convert the latest output, use the shortcut Ctrl+Space. Then, the fraction $p/q$ will be inserted on the current input line.

Example:

```
                                0.123
123/1000 ← Press Ctrl+Space to insert this fraction
                                ⋮
                             -0.123456
-1929/15625
                                ⋮
                               0.496
62/125
                                ⋮
                  0.577777777777777778
26/45
```

## *The Ans variable*

After every successful computation of an expression, the special variable Ans will be set to the computed value. Then, if an operator (+, −, ×, /, ^, !) is inserted at the beginning of the next input, "Ans" will be inserted just before it. This is a means of "continuing" the last computation.

Example:

```
50/8
                                6.25
Ans-40/9 ← Ans will automaticaly be inserted
                   1.80555555555555556
Ans-1
                   0.80555555555555556
```

## *Preferences Variables*

Preference variables are variables, all of which beginning with _ (an underscore), which state settings in AlgoSim.

### *_digits*

The _digits variable states the number of significant digits used in computational outputs. The default value is 12, but this may be changed by the user. Please notice, that the number of significant digits must be an integer in the interval [2, 18].

Example:

```
_digits := 18
Pi
                    3.14159265358979324

_digits := 12
Pi
                       3,14159265359

_digits := 3
Pi
                             3,14
```

Preferences variables cannot be deleted with the #delete command; however, they will be reset to their original values if the command #deleteAllVars is executed.

## *Other special commands*

The special command #quit will (without any preceding warning) quite AlgoSim.

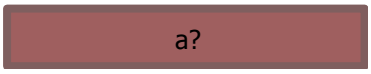
Example:

```
#quit
```

# *The Algorithm Editor*

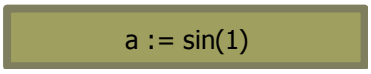
AlgoSim has an editor able to create flowcharts on the screen, which the program later can follow. In other words, the user is able to create algorithms solving mathematical problems. The flowcharts consists of elements (nodes) performing partial tasks. When AlgoSim has performed the task associated with one node, the program will go on to the node *linked* to the completed node. Links between nodes are rendered as lines between them. Below, the types of nodes available in this version of AlgoSim are shown.

## *Input*



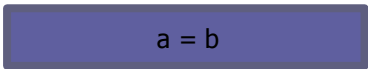

The variable entered (here *a*) is given the value that the user enters in a dialogue box during execution of the algorithm.

## *Assignment*



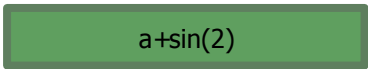

The variable entered (here *a*) is given the value entered to the right of the assignment operator (here sin(1)).

## *Condition*




If the condition stated (here $a = b$) is true, then AlgoSim will proceed to the node connected with a solid line; otherwise, AlgoSim will proceed to the node connected with a dashed line.

## *Output*




The value entered (here $a$+sin(2)) will be shown in a message box.

## *Creating Flowcharts*

Creating flowcharts, nodes are inserted via the "Insert" button in the "Algorithm Editor" tab. To change the text displayed on a node, simply double click it. In order to draw a solid connection between node A and node B, drag with the right mouse button between A and B. In order to draw a dashed (*if false*) connection between node A and node B, hold down the Shift key while dragging. In order to remove a solid connection between node A and node B, simply right click node A; in order to remove a dashed connection, hold down Shift while clicking. To select which node AlgoSim should start with during execution, Ctrl click the proper node.

Nodes can freely be moved and the "Adjust to grid" option will (while moving) position nodes using an invisible grid; this makes it easier to position the nodes side by side.

In order to save an algorithm, choose the "Save Algorithm" button or the Ctrl+S (Save) shortcut. To open a previously saved algorithm, choose the "Open Algorithm" button or the Ctrl+O (Open) shortcut. To clear the editor and create a new algorithm, choose the "New Algorithm" button or the Ctrl+N (New) shortcut.

Please notice that algorithm files by default use the ".algorithm" file suffix.

An image file containing the flow chart may be created and saved by using the "Save diagram as image" button. The image will be saved in the EMF (Win32 Enhanced Metafile) format; thus, it can be magnified and rescaled without any quality loss at all.

## *Executing Algorithms*

In order to execute an algorithm, press the "Execute" button. Some algorithms may take long time to execute; an analogue clock shows how much time that has passed. In order to abort the execution of an algorithm, press the "Abort" button. If the algorithm does not respond, try to hold down the Ctrl key and press the "Terminate thread" button instead.

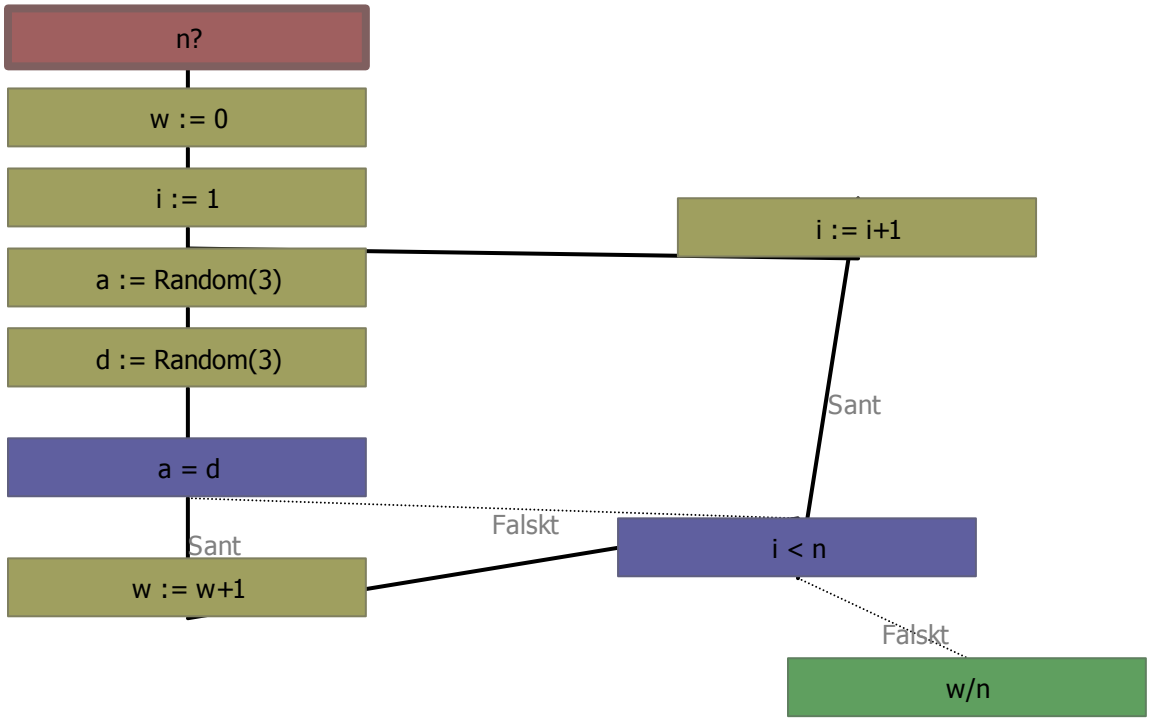
## *Example of an Algorithm*

The following is a classical probability problem:

*Individual X has won a competition and is therefore placed in front of three doors, named A, B, and C, respectively. "Behind two of the doors there is a cactus," the competition leader says, "but behind the third door, there is a grand new car. You may position yourself in front of one of the three doors, and then you will win the object behind it. Which door will you choose?" When X has placed herself in front of door A, the leader opens one of the other doors, B, hiding a cactus. "Would you like to stay in front of door A, or would you like to change to door C?" the leader asks. In order to obtain maximum probability of winning the new car, should X stick to door A, or change to door C?*

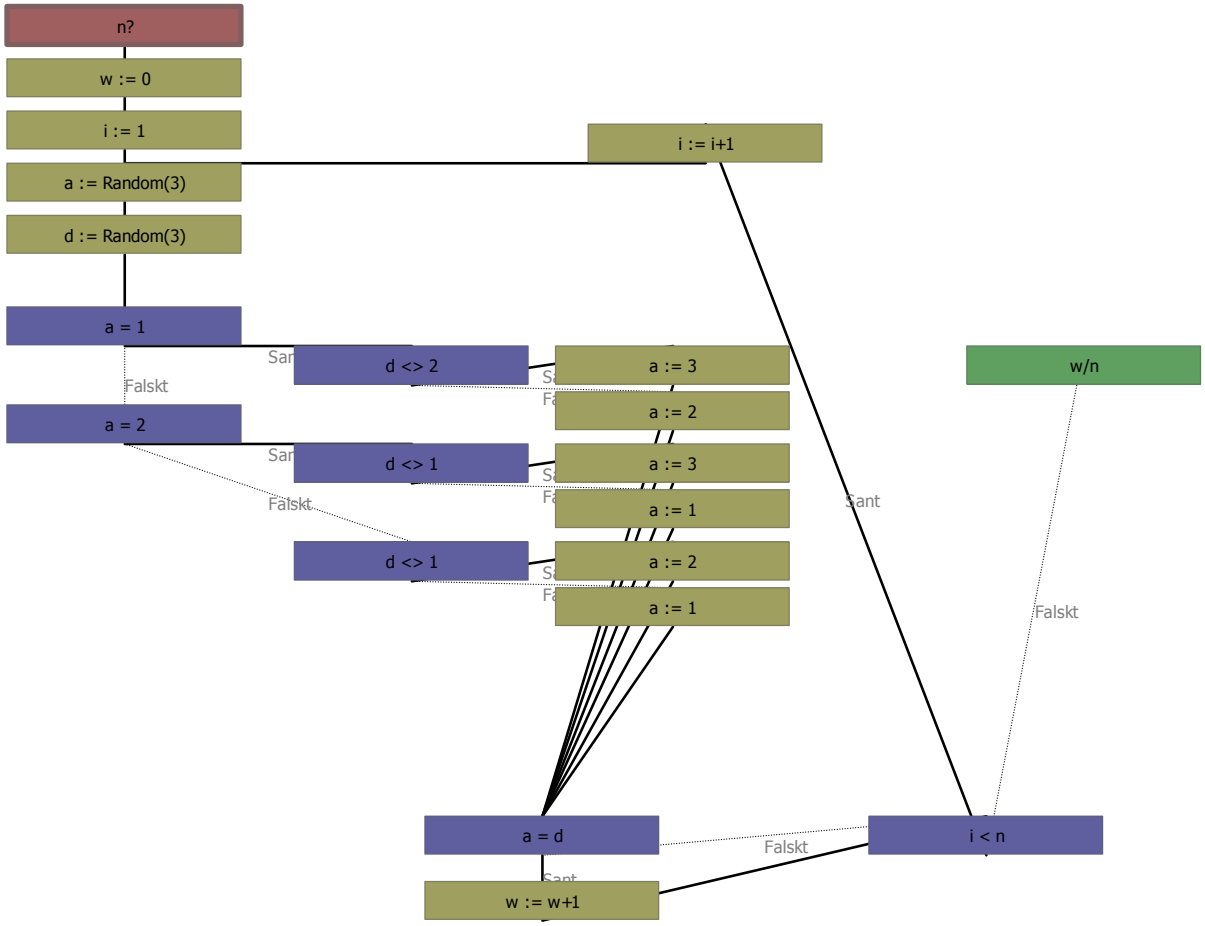
Let us test the both scenarios by means of two algorithms!

***Let us stick to door A***




Execution of this algorithm, with $n = 1000$, results in, $w/n = 0{,}329 \approx 1/3$.

### *Let us change to door C*




Execution of this algorithm, with $n = 1000$, results in $w/n = 0{,}678 \approx 2/3$.

Hence, X should change to door C[4].

[4] For a more theoretical discussion about this problem, please see Rejbrand, Andreas. *Ett klassiskt sannolikhetsproblem*. Katrineholm 2004.

# *Calculus*

Behind the "Calculus" tab, several functions related to calculus, such as numerical differentiation, integration and equation solving exist.

## *Numerical differentiation*

Enter the identifier of the function and at which point $x$ the function is to be differentiated. Also, enter $h$, half the horizontal distance between the two graph-intersecting points of the secant approximating the tangent. One might believe that smaller values of $h$ would result in higher precision, but that is not always the case. Often, the default value $h = 10^{-6}$ is appropriate, whereas smaller values might result in approximation errors.

AlgoSim uses a central difference quotient to approximate the derivative.

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h}$$

Choose "Differentiate" to compute the derivative. The approximated value of it will be written in the console.

Choose "Calculate equation of tangent" to compute the equation of the tangent to the function at the selected point $x$. Press "Draw tangent" to draw the tangent in the graph window. Please note, however, that the graph window settings must be appropriate for this to look good. See the section "The Graph Window" on page 36.

## *Numerical Integration*

Enter the identifier of the function to be integrated as well as the lower and the upper limits of integration. Also, enter the step length $h$. The default value of $h$ makes 500 steps necessary. For instance, if the limits are 0 and 5, respectively, then $h = 0.01$. After having changed the limits, press "Auto" in order to compute the appropriate step length (so that the number of step will equal 500). A small value of $h$ result in a precise approximation, but the computation will take longer time.

Click "Integrate" to compute the integral; the approximation will be written in the console. AlgoSim uses Simpson's formula to approximate integrals.

## *Numerical localization of stationary points*

The "Stationary Points" panel can be used for localization stationary points, i.e. local maxima, minima and saddle points of a function graph. Choose the lower and upper limits $a$ and $b$ within which the search will be performed. A low value of the step length $h$ results in a precise approximation, but the computation will take longer time. "Auto" will adjust the step length so that the total number of steps will equal 500.

Choose one of the buttons "Maximum," "Minimum" and "Saddle point" to locate the next local minimum, maximum or saddle point after $a$, respectively.

## *Function Sums and Products*

The "Function sum and product" panel is used to compute the sum and the product of a function between two limits (inclusively). Enter the function identifier, the limits *a* and *b* and choose to compute either the sum (Σ) or the product (Π).

## *Numerical Equation Solver*

Via the "Solve" panel, equations with one unknown can be numerically solved. Enter the equation and the unknown variable. Furthermore, enter an initial approximation in the "Origin" field. There is, however, no need that this approximation must be approximately equal to the actual root. Also, enter the number of iterations to be made; theoretically, a greater number of iterations results in higher accuracy, but most often, many decimals will be correctly approximated even with only five approximations. The default value is 50, though.

Click "Solve" to solve the equation. If an error occurs, it might be a good idea to change the value of the initial approximation. The computed root is written in the console. Information about the LHS (left hand side), RHS (light hand side) and the absolute error |RHS − LHS| is presented. Thus, AlgoSim evaluates whether the approximated root satisfies the equation or not.

AlgoSim uses Newton-Raphsons algorithm.

## *Table*

The "Table" functions allows the user to obtain a table with two columns, the first of which containing arguments from *a* to *b* (inclusively) (the step between each argument equals *h*), and the second of which containing the value of the function for that argument. Choose "Create table" to compute the values and render the table. The statistical lists *x* and *y* are used for arguments and function values, respectively. The fact that the table function uses the statistical lists implies that the computed values can be analysed and plotted in the graph window. (See the section "Statistical Computations" on page 39.)

Example: The following lists show the 10 first prime numbers. (Function: prime(*n*); *a* = 1; *b* = 10; *h* = 1)

| n | prime(n) |
|---|---|
| 1 | 2 |
| 2 | 3 |
| 3 | 5 |
| 4 | 7 |
| 5 | 11 |
| 6 | 13 |
| 7 | 17 |
| 8 | 19 |
| 9 | 23 |
| 10 | 29 |

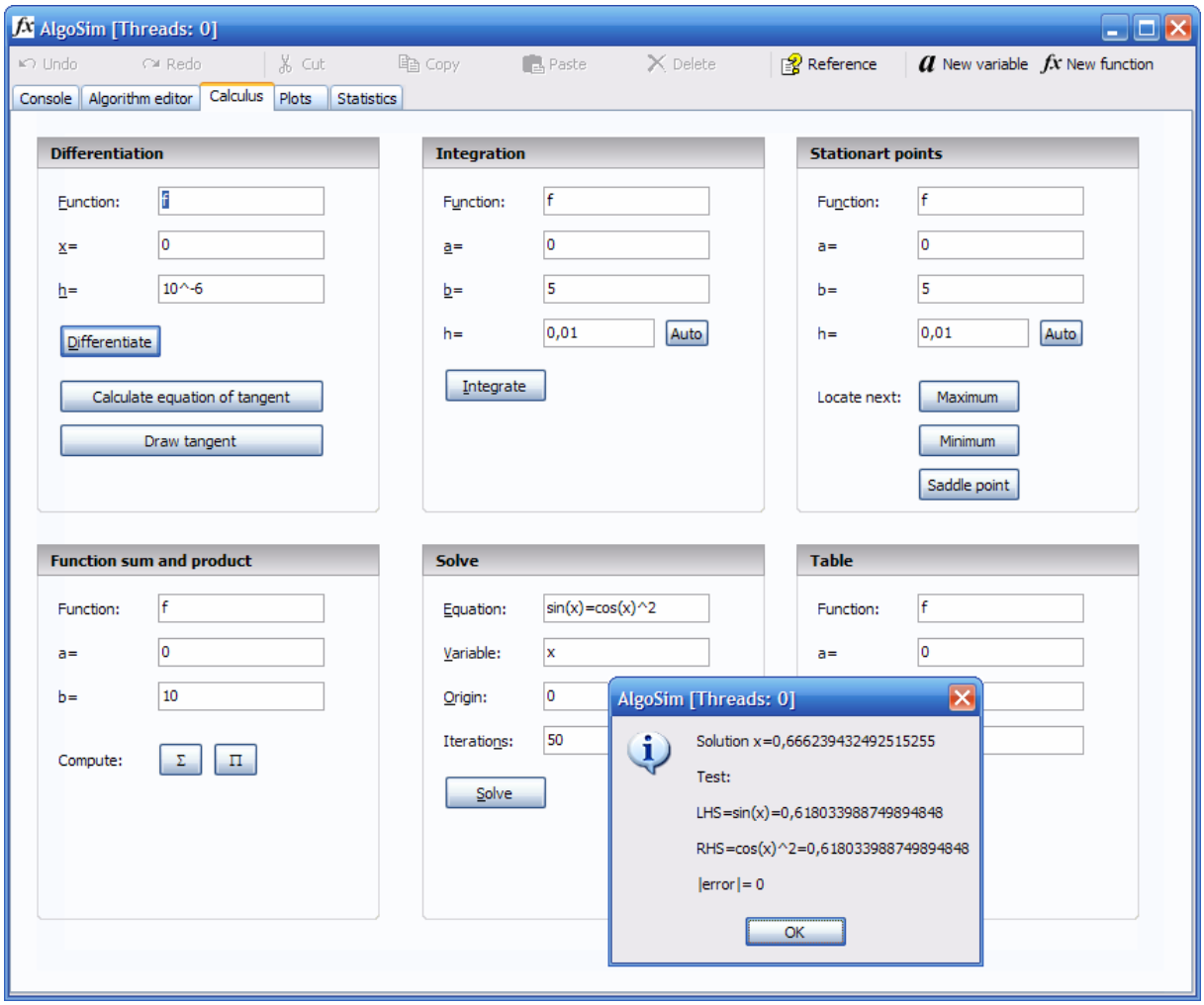



***Image 2.*** Calculus

## *The Graph Window*

Behind the "Plots" tab, a coordinate system exists, where function graphs as well as statistical diagrams can be drawn. On the left side of the page, the user is able to specify the visible region. The "scl" values specify the distance between the grid lines. The user is also able to determine whether to show coordinate axes, grid and labels along the axes. The list box below these controls specifies the functions to be drawn and their plot colours. The following predefined colours can be used:

- clBlack (black)
- clRed (red)
- clGreen (green)
- clBlue (blue)
- clYellow (yellow)
- clNavy (navy blue; darker than blue)
- clMaroon (maroon; deep purplish-red)

Other colours can be chosen as well, by double clicking in the colour box. This will open the Rejbrand Colour Selector, which may not have been translated to your language.

The current image may be saved as a file either in the EMF format or in the AlgoSim ASG format by clicking the "Save image" button.

The ASG format saves only function plots – not statistical diagrams. The graphs are saved with all mathematical information; thus, they can be resized and rescaled without any quality loss at all. Moreover, the files became very small in bytes. However, ASG files can only be viewed in AlgoSim Image Viewer. Users of computers not having AlgoSim installed are unable to view the graphs. An ASG file can later be converted into an EMF file, though.

The EMF format saves the whole image – both function graphs and statistical diagrams. Most information is saved, except for function plots, which are saved (approximated) as polygons ("polylines" to be more correct); thus, images can be magnified, rescaled and printed without any quality loss, except for function graphs, which will become less "smooth." EMF images occupy somewhat more disc space than ASG files. EMF files, however, can be viewed on all Windows computers, printed and inserted in applications such as Microsoft Word.

| | ***ASG*** | ***EMF*** |
|---|---|---|
| Saves function graphs | Yes | Yes |
| Saves statistical diagrams | No | Yes |
| Save function graphs … | Precisely | Approximately (polygons) |
| File Size | Very small | Small |
| Can be resized and rescaled | Yes | Yes (Graphs "unsmooth") |
| Can be opened in AlgoSim Image Viewer | Yes | Yes |
| Can be opened in Windows | No | Yes |

***Table 3.*** Comparison between ASG files and EMF files.

Magnification of a rectangular area within the graph window may be performed by dragging a rectangle with the left mouse button.

> ***Note!***
>
> It is not possible to draw a function requiring more than one argument, as *y* represents the function value and *x* the only argument. However, it is possible to create a *new* function *referring* to the old function and having one of the arguments variable and all other fixed (constant).

```
y(a; b) := a + 2×b
y2(b) := y(a; b)
```

Now, the function y2 of *b* can be drawn. Instead of being an argument, *a* will be a fixed constant.

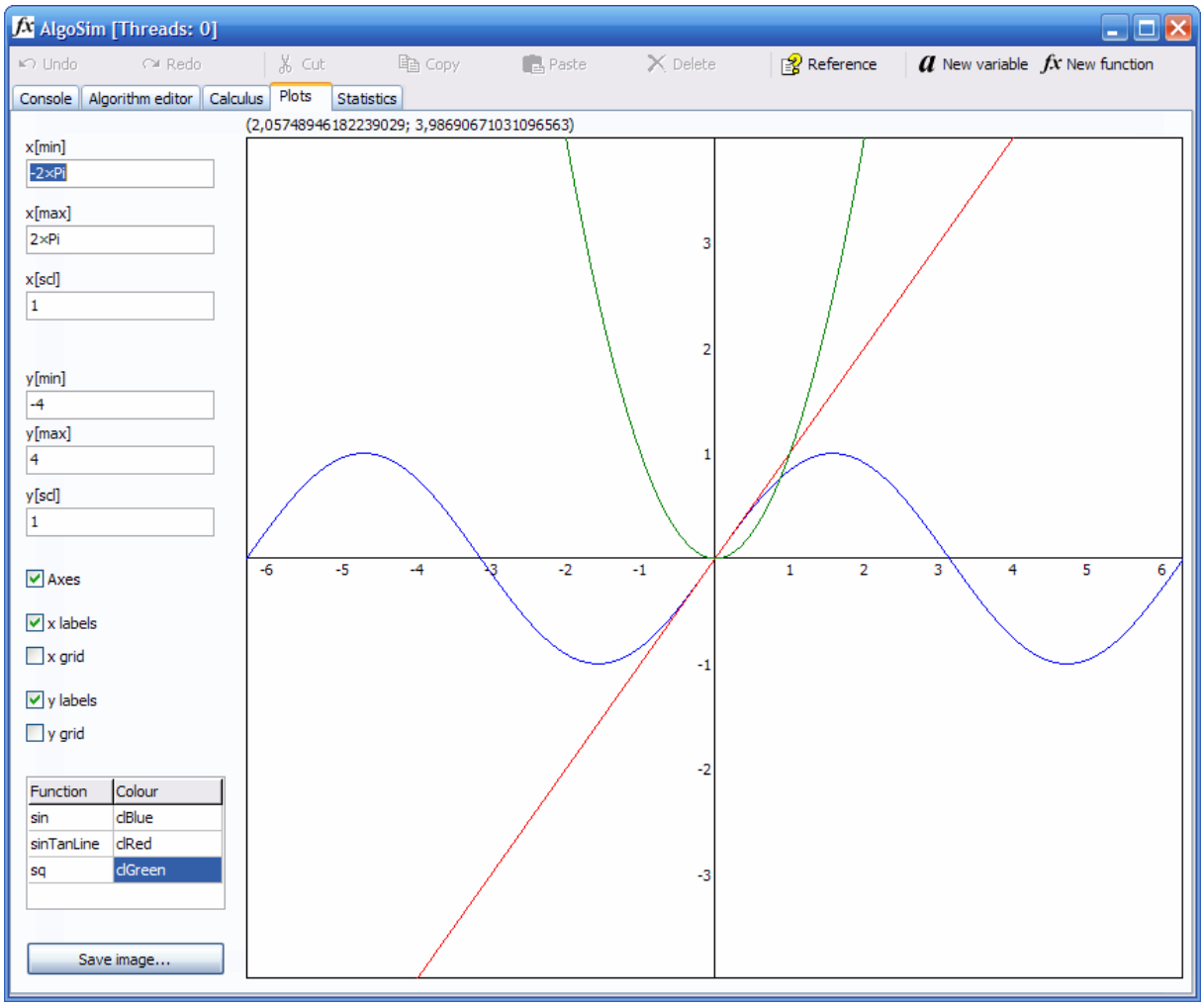



***Image 3.*** The Graph Window. A tangent to the sine function has been drawn using the differentiation function.

## *Statistical Computations*

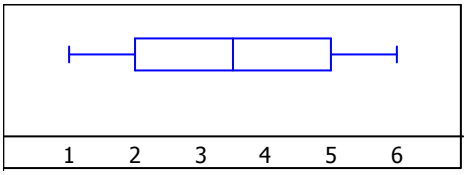
The "Statistics" tab allows the user to perform statistical computations easier than from the console. Statistical data is entered in the List $x$ list and, if the material uses paired variables, List $y$ as well. The "Save list" and "Open list" buttons may be used to save and open list files.

In order to analyse a list of data, select the list in the "Show statistics based on list" field and press "Compute." The parameters that will be computed are listed below.

- $n$ Number of observations
- $\Sigma$ Sum
- $\Pi$ Product
- $\mu$ (Arithmetical) Mean
- $\sigma$ Standard deviation (n)
- $\sigma(n\text{-}1)$ Standard deviation (n − 1)
- min Smallest number
- Q1 First quartile
- Md Median (Second quartile)
- Q3 Third quartile
- max Greatest number
- f list List of the frequencies of each value in the list

If the "Show diagram" checkbox is checked, a statistical diagram will be drawn in the graph window. The type of the diagram is selected in the "Type" field. Currently, there are three types available.

- ***Plot***
  Plots the $(x, y)$ values, such that$(x_i, y_i)$ = (List $x_i$, List $y_i$)
- ***Frequency bars***
  This diagram only uses List $x$. For every unique value in the list, a bar will be drawn at the corresponding $x$ value in the graph window; the height of each bar equals the absolute frequency of that value.
- ***Box***
  This diagram only uses List $x$. A "box" indicating min, Q1, Md, Q3, and max is drawn with a vertical centre at $y = 5$.



The diagram will be drawn in the colour chosen in the "Colour" field.

## *Settings*

The settings of AlgoSim can be altered via the system menu (right click the blue list at top of AlgoSim's window) and the "Settings" menu item.

The user can enter a command file (script file) to be executed every time AlgoSim starts. A script file is a plain text file containing ordinary (console acceptable) commands, and may be used to define new variables and functions at program start-up.

Below is an example of a typical script file.

```
c := 299792458
g := 9.82
h := 6.6260687E-34
N := 6.0221419E23
_digits := 16
```

The console background colour, text colour, font identifier and text size can be changed as well. All settings are saved between sessions. It is also possible to specify another, external, application to be run instead of AlgoSim when AlgoSim is started having the Ctrl key hold down. (AlgoSim will then be closed and the external program started.) For instance, this may be suitable for use together with some Microsoft keyboards having a "Calculator" hotkey. One might want to start a CAS instead of AlgoSim if the Ctrl key is hold down.

In the system menu, the "Show/hide toolbar" item is available as well, and the "About AlgoSim" item opens a dialogue box showing version number, copyright information and support options.

## *The Reference Function*

The "Reference" toolbar button and the keyboard shortcut F1 (Help) can be used to show a dialogue box presenting all built-in and user-defined variables, functions, operators and brackets. Brief instructions are included as well.

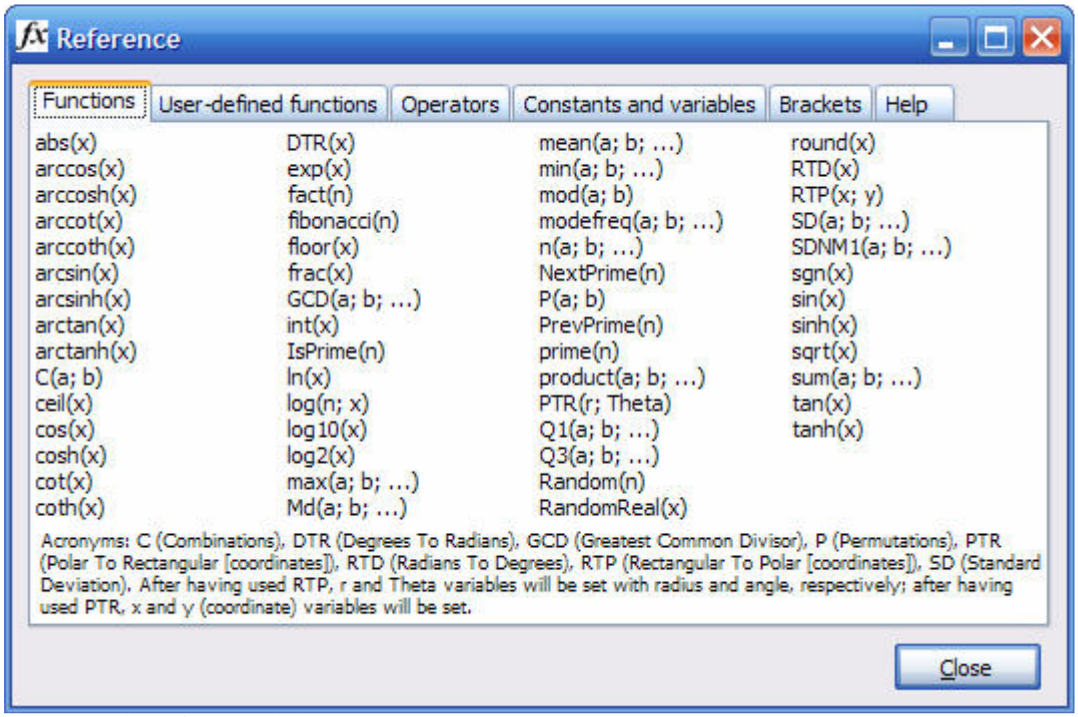



***Image 4.*** The reference dialogue

## *System Requirements*

AlgoSim is tested and compatible with the following system requirements:

| | ***Minimum requirement*** | ***Recommended setting*** |
|---|---|---|
| Operative system | Windows 98, ME, 2000 | Windows XP |
| Processor | Pentium 200 MHz | Pentium 4 2,0 GHz |
| Random Access Memory | 32 MB | 256 MB |
| Screen Font Technology | (no requirement) | Microsoft ClearType |

***Table 4.*** System requirements

# *Legal information*

Algorithm Simulator is created by Andreas Rejbrand (http://english.rejbrand.se).




http://english.rejbrand.se