Documentation - Références

Math.js est une vaste bibliothèque de mathématiques pour JavaScript et Node.js. Elle dispose d'un analyseur d'expression souple et offre une solution intégrée pour travailler avec des nombres, grands nombres, nombres complexes, unités et matrices.

Puissant et facile à utiliser.

Cette documentation contient un didacticiel de mise en route, un aperçu détaillé décrivant math.js de haut niveau et une référence décrivant en détail toutes les fonctions disponibles, les constantes et les unités.

Auteur : Jos de Jong wjosdejong@gmail.com

Traduction :

Relecture / Corrections :

Licence :

Références - Constantes

Math.js contient les constantes suivantes.

Constante	Description	Valeur
`e`, `E`	Euler's number, the base of the natural logarithm.	2.718281828459045
`i`	Imaginary unit, defined as i*i=-1. A complex number is described as a + b*i, where a is the real part, and b is the imaginary part.	`sqrt(-1)`
`Infinity`	Infinity, a number which is larger than the maximum number that can be handled by a floating point number.	`Infinity`
`LN2`	Returns the natural logarithm of 2.	`0.6931471805599453`
`LN10`	Returns the natural logarithm of 10.	`2.302585092994046`
`LOG2E`	Returns the base-2 logarithm of E.	`1.4426950408889634`
`LOG10E`	Returns the base-10 logarithm of E.	`0.4342944819032518`
`NaN`	Not a number.	`NaN`
`null`	Value null.	`null`
`phi`	Phi is the golden ratio. Two quantities are in the golden ratio if their ratio is the same as the ratio of their sum to the larger of the two quantities. Phi is defined as `(1 + sqrt(5)) / 2`	`1.618033988749895`
`pi`, `PI`	The number pi is a mathematical constant that is the ratio of a circle\'s circumference to its diameter.	`3.141592653589793`
`SQRT1_2`	Returns the square root of 1/2.	`0.7071067811865476`
`SQRT2`	Returns the square root of 2.	`1.4142135623730951`
`tau`	Tau is the ratio constant of a circle\'s circumference to radius, equal to `2 * pi`.	`6.283185307179586`
`uninitialized`	Constant used as default value when resizing a matrix to leave new entries uninitialized.	
`version`	Returns the version number of math.js.	For example `0.24.1`

Exemple d'utilisation :

```
math.sin(math.pi / 4);          // 0.70711
math.multiply(math.i, math.i);  // -1
```

Références – Unites

This page lists all available units and prefixes. How to use units is explained on the page Units.

Units

Math.js comes with the following built-in units.

Base	Unit
Length	meter (m), inch (in), foot (ft), yard (yd), mile (mi), link (li), rod (rd), chain (ch), angstrom, mil
Surface	m², sqin, sqft, sqyd, sqmi, sqrd, sqch, sqmil
Volume	m³, litre (l, L, lt, liter), cc, cuin, cuft, cuyd, teaspoon, tablespoon
Liquid	volume minim (min), fluiddram (fldr), fluidounce (fldz), gill (gi), cup (cp), pint (pt), quart (qt), gallon (gal), beerbarrel (bbl), oilbarrel (obl), hogshead, drop (gtt)
Angles	rad, deg, grad, cycle
Time	second (s), seconds, minute, minutes, hour (h), hours, day, days
Mass	gram(g), tonne, ton, grain (gr), dram(dr), ounce (oz), poundmass (lbm, lb, lbs), hundredweight (cwt), stick
Electric current	ampere (A)
Temperature	kelvin (K), celsius (degC), fahrenheit (degF), rankine (degR)
Amount of substance	mole (mol)
Luminous	intensity candela (cd)
Force	newton (N), poundforce (lbf)
Binary	bit (b), byte (B)

Note that all relevant units can also be written in plural form, for example `5 meters` instead of `5 meter` or `10 seconds` instead of `10 second`.

Prefixes

The following decimal prefixes are available.

Name	Abbreviation	Value
deca	da	1e1
hecto	h	1e2
kilo	k	1e3
mega	M	1e6
giga	G	1e9
tera	T	1e12
peta	P	1e15
exa	E	1e18
zetta	Z	1e21
yotta	Y	1e24
Name	**Abbreviation**	**Value**
deci	d	1e-1
centi	c	1e-2
milli	m	1e-3
micro	u	1e-6
nano	n	1e-9
pico	p	1e-12
femto	f	1e-15
atto	a	1e-18
zepto	z	1e-21
yocto	y	1e-24

The following binary prefixes are available. They can be used with units `bit` (`b`) and `byte` (`B`).

Name	Abbreviation	Value
kilo, kibi	k, Ki	1024
mega, mebi	M, Mi	1024^2
giga, gibi	G, Gi	1024^3
tera, tebi	T, Ti	1024^4
peta, pebi	P, Pi	1024^5
exa, exi	E, Ei	1024^6
zetta, zebi	Z, Zi	1024^7
yotta, yobi	Y, Yi	1024^8

Références - Fonctions – Classement alphabétique

abs(x) , acos(x) , add(x , y) , arg(x) , asin(x) , atan(x) , atan2(y , x) ,

bignumber(x) , boolean(x) ,

ceil(x) , clone(x) , combinations(n,k) , compare(x,y) , compile(expr) ,
complex(re,im) , concat(a,b,c,... [,dim]) , conj(x) , cos(x) , cosh(x) , cot(x) , coth(x) ,
cross(x,y) , csc(x) , csch(x) , cube(x) ,

deepEqual(x,y) , det(x) , diag(X) , distribution(name) , divide(x,y) ,
dot(x,y) , dotDivide(x,y), dotMultiply(x,y) , dotPow(x,y)

equal(x, y) , eval(expr [, scope]) , exp(x) , eye(n) ,

factorial(n) , filter(x, test) , fix(x) , flatten(x) , floor(x) ,
forEach(x, callback) , format(value [, precision]) ,

gcd(a, b) ,

help(search) ,

im(x) , import(filename | object, override) , index(range1, range2, ...) , inv(x) ,

larger(x, y) , largerEq(x, y) , lcm(a, b) , log(x [, base]) , log10(x) ,

map(x, callback) , matrix(x) , max(a, b, c, ...) , mean.mean(a, b, c, ...) ,
mean.median(a, b, c, ...) , min(a, b, c, ...) , mod(x, y) , multiply(x, y) ,

norm(x [, p]) , nthRoot(a, root) , number(value) ,

ones(m, n, p, ...) ,

parse(expr [, scope]) , parser() , permutations(n) , pickRandom(array) ,
pow(x, y) , print(template, values [, precision]) , prod(a, b, c, ...) ,

random([min, max]) , randomInt([min, max]) , range(start, end [, step]) ,
re(x) , resize(x, size [, defaultValue]) , round(x [, n]) ,

sec(x) , sech(x) , select(value) , sign(x) , sin(x) , sinh(x) , size(x) , smaller(x, y) ,
smallerEq(x, y) , sort(x) , sqrt(x) , square(x) , squeeze(x) , std(a, b, c, ...) ,
string(value) , subset(x, index [, replacement]) , subtract(x, y) , sum(a, b, c, ...) ,

tan(x) , tanh(x) , to(x, unit) , transpose(x) , typeof(x) ,

unaryMinus(x) , unaryPlus(x) , unequal(x, y) , unit(x) ,

var(a, b, c, ...) ,

xgcd(a, b) ,

zeros(m, n, p, ...)

Function abs

Calculate the absolute value of a number. For matrices, the function is evaluated element wise.

Syntax : **`math.abs(x)`**

Parameters

Parameter	Type	Description
`x`	Number | BigNumber | Boolean | Complex | Array | Matrix | null	A number or matrix for which to get the absolute value

Returns

Type	Description
Number | BigNumber | Complex | Array | Matrix	Absolute value of `x`

Examples

```
math.abs(3.5);                // returns Number 3.5
math.abs(-4.2);               // returns Number 4.2

math.abs([3, -5, -1, 0, 2]);  // returns Array [3, 5, 1, 0, 2]

Voir aussi : sign
```

Function acos

Calculate the inverse cosine of a value.

For matrices, the function is evaluated element wise.

Syntax : `math.acos(x)`

Parameters

Parameter	Type	Description
x	Number | Boolean | Complex | Array | Matrix | null	Function input

Returns

Type	Description
Number | Complex | Array | Matrix	The arc cosine of x

Examples

```
math.acos(0.5);           // returns Number 1.0471975511965979
math.acos(math.cos(1.5)); // returns Number 1.5

math.acos(2);             // returns Complex 0 + 1.3169578969248166 i

Voir aussi : cos, atan, asin
```

Function add

Add two values, `x + y`. For matrices, the function is evaluated element wise.

Syntax : **`math.add(x, y)`**

Parameters

Parameter	Type	Description
`x`	Number | BigNumber | Boolean | Complex | Unit | String | Array | Matrix | null	First value to add
`y`	Number | BigNumber | Boolean | Complex | Unit | String | Array | Matrix | null	Second value to add

Returns

Type	Description
Number | BigNumber | Complex | Unit | String | Array | Matrix	Sum of `x` and `y`

Examples

```
math.add(2, 3);               // returns Number 5

var a = math.complex(2, 3);
var b = math.complex(-4, 1);
math.add(a, b);               // returns Complex -2 + 4i

math.add([1, 2, 3], 4);       // returns Array [5, 6, 7]

var c = math.unit('5 cm');
var d = math.unit('2.1 mm');
math.add(c, d);               // returns Unit 52.1 mm

Voir aussi : substract
```

Function arg

Compute the argument of a complex value. For a complex number `a + bi`, the argument is computed as `atan2(b, a)`.

For matrices, the function is evaluated element wise.

Syntax : **`math.arg(x)`**

Parameters

Parameter	Type	Description
`x`	Number | Complex | Array | Matrix | Boolean | null	A complex number or array with complex numbers

Returns

Type	Description
Number | Array | Matrix	The argument of x

Examples

```
var a = math.complex(2, 2);
math.arg(a) / math.pi;          // returns Number 0.25

var b = math.complex('2 + 3i');
math.arg(b);                    // returns Number 0.982793723247329
math.atan2(3, 2);               // returns Number 0.982793723247329

Voir aussi : re, im, cpnj, abs
```

Function asin

Calculate the inverse sine of a value.

For matrices, the function is evaluated element wise.

Syntax : `math.asin(x)`

Parameters

Parameter	Type	Description
`x`	Number | Boolean | Complex | Array | Matrix | null	Function input

Returns

Type	Description
Number | Complex | Array | Matrix	The arc sine of x

Examples

```
math.asin(0.5);           // returns Number 0.5235987755982989
math.asin(math.sin(1.5)); // returns Number ~1.5

math.asin(2);             // returns Complex 1.5707963267948966
-1.3169578969248166 i
```

Voir aussi **:** **sin**, **atan**, **acos**

`Function atan`

Calculate the inverse tangent of a value.

For matrices, the function is evaluated element wise.

Syntax : `math.atan(x)`

Parameters

Parameter	Type	Description
`x`	Number | Boolean | Complex | Array | Matrix | null	Function input

Returns

Type	Description
Number | Complex | Array | Matrix	The arc tangent of x

Examples

```
math.atan(0.5);           // returns Number 0.4636476090008061
math.atan(math.tan(1.5)); // returns Number 1.5

math.atan(2);             // returns Complex 1.5707963267948966
-1.3169578969248166 i
```

Voir aussi **:** **tan**, **asin**, **acos**

Function atan2

Calculate the inverse tangent function with two arguments, y/x. By providing two arguments, the right quadrant of the computed angle can be determined.

For matrices, the function is evaluated element wise.

Syntax : **`math.atan2(y, x)`**

Parameters

Parameter	Type	Description
`y`	Number | Boolean | Complex | Array | Matrix | null	Second dimension
`x`	Number | Boolean | Complex | Array | Matrix | null	First dimension

Returns

Type	Description
Number | Complex | Array | Matrix	Four-quadrant inverse tangent

Examples

```
math.atan2(2, 2) / math.pi;       // returns number 0.25

var angle = math.unit(60, 'deg'); // returns Unit 60 deg
var x = math.cos(angle);
var y = math.sin(angle);

math.atan(2);     // returns Complex 1.5707963267948966 -1.3169578969248166 i
```

Voir aussi : tan, atan, sin, cos

Function bignumber

Create a BigNumber, which can store numbers with arbitrary precision. When a matrix is provided, all elements will be converted to BigNumber.

Syntax : **`math.bignumber(x)`**

Parameters

Parameter	Type	Description
`value`	Number | String | Array | Matrix | Boolean | null	Value for the big number, 0 by default.

Returns

Type	Description
BigNumber	The created bignumber

Examples

```
0.1 + 0.2;                                  // returns Number 0.30000000000000004
math.bignumber(0.1) + math.bignumber(0.2);  // returns BigNumber 0.3


7.2e500;                                    // returns Number Infinity
math.bignumber('7.2e500');                  // returns BigNumber 7.2e500
```

Voir aussi : boolean, complex, index, matrix, string, unit

Function boolean

Create a boolean or convert a string or number to a boolean. In case of a number, `true` is returned for non-zero numbers, and `false` in case of zero. Strings can be `'true'` or `'false'`, or can contain a number. When value is a matrix, all elements will be converted to boolean.

Syntax : **`math.boolean(x)`**

Parameters

Parameter	Type	Description
`value`	String | Number | Boolean | Array | Matrix | null	A value of any type

Returns

Type	Description
Boolean | Array | Matrix	The boolean value

Examples

```
math.boolean(0);     // returns false
math.boolean(1);     // returns true
math.boolean(-3);     // returns true
math.boolean('true');     // returns true
math.boolean('false');     // returns false
math.boolean([1, 0, 1, 1]);     // returns [true, false, true, true]
```

Voir aussi : bignumber, complex, index, matrix, string, unit

Function ceil

Round a value towards plus infinity If `x` is complex, both real and imaginary part are rounded towards plus infinity. For matrices, the function is evaluated element wise.

Syntax : **`math.ceil(x)`**

Parameters

Parameter	Type	Description
`x`	Number | BigNumber | Boolean | Complex | Array | Matrix | null	Number to be rounded

Returns

Type	Description
Number | BigNumber | Complex | Array | Matrix	Rounded value

Examples

```
math.ceil(3.2);               // returns Number 4
math.ceil(3.8);               // returns Number 4
math.ceil(-4.2);              // returns Number -4
math.ceil(-4.7);              // returns Number -4

var c = math.complex(3.2, -2.7);
math.ceil(c);                 // returns Complex 4 - 2i

math.ceil([3.2, 3.8, -4.7]);  // returns Array [4, 4, -4]
```

Voir aussi : floor, fix, round

Function clone

Clone an object.

Syntax : `math.clone(x)`

Parameters

Parameter	Type	Description
x	*	Object to be cloned

Returns

Type	Description
*	A clone of object x

Examples

```
math.clone(3.5);              // returns number 3.5
math.clone(2 - 4i);           // returns Complex 2 - 4i
math.clone(45 deg);           // returns Unit 45 deg
math.clone([[1, 2], [3, 4]]); // returns Array [[1, 2], [3, 4]]
math.clone("hello world");    // returns string "hello world"
```

Function combinations

Compute the number of ways of picking `k` unordered outcomes from `n` possibilities.

Combinations only takes integer arguments. The following condition must be enforced: k <= n.

Syntax : **`math.combinations(n, k)`**

Parameters

Parameter	Type	Description
`n`	Number | BigNumber	Total number of objects in the set
`k`	Number | BigNumber	Number of objects in the subset

Returns

Type	Description
Number | BigNumber	Number of possible combinations.

Examples

```
math.combinations(7, 5); // returns 21
```

Voir aussi : permutations, factorial

Function compare

Compare two values. Returns 1 when x > y, -1 when x < y, and 0 when x == y.

x and y are considered equal when the relative difference between x and y is smaller than the configured epsilon. The function cannot be used to compare values smaller than approximately 2.22e-16.

For matrices, the function is evaluated element wise.

Syntax : `math.compare(x, y)`

Parameters

Parameter	Type	Description
`x`	Number | BigNumber | Boolean | Unit | String | Array | Matrix | null	First value to compare
`y`	Number | BigNumber | Boolean | Unit | String | Array | Matrix | null	Second value to compare

Returns

Type	Description
Number | BigNumber | Array | Matrix	Returns the result of the comparison: 1, 0 or -1.

Examples

```
math.compare(6, 1);           // returns 1
math.compare(2, 3);           // returns -1
math.compare(7, 7);           // returns 0

var a = math.unit('5 cm');
var b = math.unit('40 mm');
math.compare(a, b);           // returns 1

math.compare(2, [1, 2, 3]);   // returns [1, 0, -1]
```

Voir aussi : equal, unequal, smaller, smallerEq, larger, largerEq

Function compile

Parse and compile an expression. Returns a an object with a function `eval([scope])` to evaluate the compiled expression.

Syntax : **`math.compile(expr)`** `// returns one node`

`math.compile([expr1, expr2, expr3, …])` `// returns an array with nodes`

Parameters

Parameter	Type	Description
`expr`	String | String[] | Matrix	The expression to be compiled

Returns

Type	Description
{eval: Function} | Array.<{eval: Function}>	code An object with the compiled expression

Examples

```
var code = math.compile('sqrt(3^2 + 4^2)');
code.eval(); // 5

var scope = {a: 3, b: 4}
var code = math.compile('a * b'); // 12
code.eval(scope); // 12
scope.a = 5;
code.eval(scope); // 20

var nodes = math.compile(['a = 3', 'b = 4', 'a * b']);
nodes[2].eval(); // 12
```

Voir aussi : parse, eval

Function complex

Create a complex value or convert a value to a complex value.

Syntax :

```
math.complex()                            // creates a complex value with zero
                                          // as real and imaginary part.
math.complex(re : number, im : string)    // creates a complex value with
provided
                                          // values for real and imaginary part.
math.complex(re : number)                 // creates a complex value with
provided
                                          // real value and zero imaginary part.
math.complex(complex : Complex)           // clones the provided complex value.
math.complex(arg : string)                // parses a string into a complex
value.
math.complex(array : Array)               // converts the elements of the array
                                          // or matrix element wise into a
                                          // complex value.
math.complex({re: number, im: number})    // creates a complex value with
provided
                                          // values for real an imaginary part.
math.complex({r: number, phi: number})    // creates a complex value with
provided
                                          // polar coordinates
```

Parameters

Parameter	Type	Description
`args`	* | Array | Matrix	Arguments specifying the real and imaginary part of the complex number

Returns

Type	Description
Complex | Array | Matrix	Returns a complex value

Examples

```
var a = math.complex(3, -4);      // a = Complex 3 - 4i
a.re = 5;                         // a = Complex 5 - 4i
var i = a.im;                     // Number -4;
var b = math.complex('2 + 6i');   // Complex 2 + 6i
var c = math.complex();           // Complex 0 + 0i
var d = math.add(a, b);           // Complex 5 + 2i
```

Voir aussi : bignumber, boolean, index, matrix, number, string, unit

Function concat

Concatenate two or more matrices.

Syntax :

```
math.concat(A, B, C, ...)
math.concat(A, B, C, ..., dim)
```

Where

- `dim: number` is a zero-based dimension over which to concatenate the matrices. By default the last dimension of the matrices.

Parameters

Parameter	Type	Description
`args`	... Array | Matrix	Two or more matrices

Returns

Type	Description
Array | Matrix	Concatenated matrix

Examples

```
var A = [[1, 2], [5, 6]];
var B = [[3, 4], [7, 8]];

math.concat(A, B);      // returns [[1, 2, 3, 4], [5, 6, 7, 8]]
math.concat(A, B, 0);   // returns [[1, 2], [5, 6], [3, 4], [7, 8]]
```

Voir aussi : ize, squeeze, subset, transpose

Function conj

Compute the complex conjugate of a complex value.
If `x = a+bi`, the complex conjugate of `x` is `a - bi`.

For matrices, the function is evaluated element wise.

Syntax : **`math.conj(x)`**

Parameters

Parameter	Type	Description
`x`	Number | BigNumber | Complex | Array | Matrix | Boolean | null	A complex number or array with complex numbers

Returns

Type	Description
Number | BigNumber | Complex | Array | Matrix	The complex conjugate of x

Examples

```
math.conj(math.complex('2 + 3i'));  // returns Complex 2 - 3i
math.conj(math.complex('2 - 3i'));  // returns Complex 2 + 3i
math.conj(math.complex('-5.2i'));  // returns Complex 5.2i
```

Voir aussi : re, im, arg, abs

Function cos

Calculate the cosine of a value.

For matrices, the function is evaluated element wise.

Syntax : `math.cos(x)`

Parameters

Parameter	Type	Description
`x`	Number | Boolean | Complex | Unit | Array | Matrix | null	Function input

Returns

Type	Description
Number | Complex | Array | Matrix	Cosine of x

Examples

```
math.cos(2);                       // returns Number -0.4161468365471422
math.cos(math.pi / 4);             // returns Number  0.7071067811865475
math.cos(math.unit(180, 'deg'));   // returns Number -1
math.cos(math.unit(60, 'deg'));    // returns Number  0.5

var angle = 0.2;
math.pow(math.sin(angle), 2) + math.pow(math.cos(angle), 2); // returns Number
~1
```

Voir aussi : cos, tan

Function cosh

Calculate the hyperbolic cosine of a value, defined as `cosh(x) = 1/2 * (exp(x) + exp(-x))`.

For matrices, the function is evaluated element wise.

Syntax : **`math.cosh(x)`**

Parameters

Parameter	Type	Description
`x`	Number | Boolean | Complex | Unit | Array | Matrix | null	Function input

Returns

Type	Description
Number | Complex | Array | Matrix	Hyperbolic cosine of x

Examples

```
math.cosh(0.5);       // returns Number 1.1276259652063807
```

Voir aussi : sinh, tanh

Function cot

Calculate the cotangent of a value. `cot(x)` is defined as `1 / tan(x)`.

For matrices, the function is evaluated element wise.

Syntax : **`math.cot(x)`**

Parameters

Parameter	Type	Description
`x`	Number | Boolean | Complex | Unit | Array | Matrix | null	Function input

Returns

Type	Description
Number | Complex | Array | Matrix	Cotangent of x

Examples

```
math.cot(2);       // returns Number -0.45765755436028577
1 / math.tan(2);   // returns Number -0.45765755436028577
```

Voir aussi : tan, sec, csc

Function coth

Calculate the hyperbolic cotangent of a value, defined as `coth(x) = 1 / tanh(x)`.

For matrices, the function is evaluated element wise.

Syntax : **`math.coth(x)`**

Parameters

Parameter	Type	Description
`x`	Number | Boolean | Complex | Unit | Array | Matrix | null	Function input

Returns

Type	Description
Number | Complex | Array | Matrix	Hyperbolic cotangent of x

Examples

```
// coth(x) = 1 / tanh(x)
math.coth(2);         // returns 1.0373147207275482
1 / math.tanh(2);     // returns 1.0373147207275482
```

Voir aussi : sinh, tanh, cosh

Function cross

Calculate the cross product for two vectors in three dimensional space. The cross product of `A = [a1, a2, a3]` and `B = [b1, b2, b3]` is defined as:

cross(A, B) = [a2 * b3 - a3 * b2, a3 * b1 - a1 * b3, a1 * b2 - a2 * b1]

Syntax

```
math.cross(x, y)
```

Parameters

Parameter	Type	Description
`x`	Array | Matrix	First vector
`y`	Array | Matrix	Second vector

Returns

Type	Description
Array | Matrix	Returns the cross product of `x` and `y`

Examples

```
math.cross([1, 1, 0],  [0, 1, 1]);  // Returns [1, -1, 1]
math.cross([3, -3, 1], [4, 9, 2]);  // Returns [-15, -2, 39]
math.cross([2, 3, 4],  [5, 6, 7]);  // Returns [-3, 6, -3]
```

Voir aussi : dot, multiply

Function csc

Calculate the cosecant of a value, defined as `csc(x) = 1/sin(x)`.

For matrices, the function is evaluated element wise.

Syntax : **`math.csc(x)`**

Parameters

Parameter	Type	Description
`x`	Number | Boolean | Complex | Unit | Array | Matrix | null	Function input

Returns

Type	Description
Number | Complex | Array | Matrix	Cosecant of x

Examples

```
math.csc(2);      // returns Number 1.099750170294617
1 / math.sin(2);  // returns Number 1.099750170294617
```

Voir aussi : sin, sec, cot

Function csch

Calculate the hyperbolic cosecant of a value, defined as `csch(x) = 1 / sinh(x)`.

For matrices, the function is evaluated element wise.

Syntax : **`math.csch(x)`**

Parameters

Parameter	Type	Description
`x`	Number | Boolean | Complex | Unit | Array | Matrix | null	Function input

Returns

Type	Description
Number | Complex | Array | Matrix	Hyperbolic cosecant of x

Examples

```
// csch(x) = 1/ sinh(x)
math.csch(0.5);       // returns 1.9190347513349437
1 / math.sinh(0.5);   // returns 1.9190347513349437
```

Voir aussi : sinh, sech, coth

Function cube

Compute the cube of a value, `x * x * x`. For matrices, the function is evaluated element wise.

Syntax : **`math.cube(x)`**

Parameters

Parameter	Type	Description
`x`	Number | BigNumber | Boolean | Complex | Array | Matrix | null	Number for which to calculate the cube

Returns

Type	Description
Number | BigNumber | Complex | Array | Matrix	Cube of x

Examples

```
math.cube(2);            // returns Number 8
math.pow(2, 3);          // returns Number 8
math.cube(4);            // returns Number 64
4 * 4 * 4;               // returns Number 64

math.cube([1, 2, 3, 4]); // returns Array [1, 8, 27, 64]
```

Voir aussi : multiply, square, pow

Function deepEqual

Test element wise whether two matrices are equal. The function accepts both matrices and scalar values.

Syntax : `math.deepEqual(x, y)`

Parameters

Parameter	Type	Description
`x`	Number | BigNumber | Boolean | Complex | Unit | Array | Matrix | null	First matrix to compare
`y`	Number | BigNumber | Boolean | Complex | Unit | Array | Matrix | null	Second matrix to compare

Returns

Type	Description
Number | BigNumber | Complex | Unit | Array | Matrix	Returns true when the input matrices have the same size and each of their elements is equal.

Examples

```
math.deepEqual(2, 4);   // returns false

a = [2, 5, 1];
b = [2, 7, 1];

math.deepEqual(a, b);   // returns false
math.equal(a, b);       // returns [true, false, true]
```

Voir aussi : equal, unequal

Function det

Calculate the determinant of a matrix.

Syntax : **`math.det(x)`**

Parameters

Parameter	Type	Description
`x`	Array | Matrix	A matrix

Returns

Type	Description
Number	The determinant of `x`

Examples

```
math.det([[1, 2], [3, 4]]); // returns -2

var A = [
  [-2, 2, 3],
  [-1, 1, 3],
  [2, 0, -1]
]
math.det(A); // returns 6
```

Voir aussi : inv

Function diag

Create a diagonal matrix or retrieve the diagonal of a matrix

When `x` is a vector, a matrix with vector `x` on the diagonal will be returned. When `x` is a two dimensional matrix, the matrixes `k`th diagonal will be returned as vector. When k is positive, the values are placed on the super diagonal. When k is negative, the values are placed on the sub diagonal.

Syntax :

```
math.diag(X)
math.diag(X, k)
```

Parameters

Parameter	Type	Description
`x`	Matrix | Array	A two dimensional matrix or a vector
`k`	Number | BigNumber	The diagonal where the vector will be filled in or retrieved. Default value: 0.

Returns

Type	Description
Matrix | Array	Diagonal matrix from input vector, or diagonal from input matrix.

Examples

```
 // create a diagonal matrix
 math.diag([1, 2, 3]);      // returns [[1, 0, 0], [0, 2, 0], [0, 0, 3]]
 math.diag([1, 2, 3], 1);   // returns [[0, 1, 0, 0], [0, 0, 2, 0], [0, 0, 0,
3]]
 math.diag([1, 2, 3], -1);  // returns [[0, 0, 0], [1, 0, 0], [0, 2, 0], [0, 0,
3]]

// retrieve the diagonal from a matrix
var a = [[1, 2, 3], [4, 5, 6], [7, 8, 9]];
math.diag(a);   // returns [1, 5, 9]
```

Voir aussi : ones, zeros, eye

Function distribution

Create a distribution object with a set of random functions for given random distribution.

Syntax : `math.distribution(name)`

Parameters

Parameter	Type	Description
`name`	String	Name of a distribution. Choose from 'uniform', 'normal'.

Returns

Type	Description
Object	Returns a distribution object containing functions: `random([size] [, min] [, max])`, `randomInt([min] [, max])`, `pickRandom(array)`

Examples

```
var normalDist = math.distribution('normal'); // create a normal distribution
normalDist.random(0, 10);                     // get a random value between 0 and 10
```

Voir aussi : random, randomInt, pickRandom

Function divide

Divide two values, `x / y`. To divide matrices, `x` is multiplied with the inverse of `y`: `x * inv(y)`.

Syntax : **`math.divide(x, y)`**

Parameters

Parameter	Type	Description
`x`	Number | BigNumber | Boolean | Complex | Unit | Array | Matrix | null	Numerator
`y`	Number | BigNumber | Boolean | Complex | Array | Matrix | null	Denominator

Returns

Type	Description
Number | BigNumber | Complex | Unit | Array | Matrix	Quotient, `x / y`

Examples

```
math.divide(2, 3);            // returns Number 0.6666666666666666

var a = math.complex(5, 14);
var b = math.complex(4, 1);
math.divide(a, b);            // returns Complex 2 + 3i

var c = [[7, -6], [13, -4]];
var d = [[1, 2], [4, 3]];
math.divide(c, d);            // returns Array [[-9, 4], [-11, 6]]

var e = math.unit('18 km');
math.divide(e, 4.5);          // returns Unit 4 km
```

Voir aussi : multiply

Function dot

Calculate the dot product of two vectors. The dot product of `A = [a1, a2, a3, ..., an]` and `B = [b1, b2, b3, ..., bn]` is defined as:

dot(A, B) = a1 * b1 + a2 * b2 + a3 * b3 + ... + an * bn

Syntax : **`math.dot(x, y)`**

Parameters

Parameter	Type	Description
`x`	Array | Matrix	First vector
`y`	Array | Matrix	Second vector

Returns

Type	Description
Number	Returns the dot product of `x` and `y`

Examples

```
math.dot([2, 4, 1], [2, 2, 3]);       // returns Number 15
math.multiply([2, 4, 1], [2, 2, 3]);  // returns Number 15
```

Voir aussi : multiply, cross

Function dotDivide

Divide two matrices element wise. The function accepts both matrices and scalar values.

Syntax : `math.dotDivide(x, y)`

Parameters

Parameter	Type	Description
`x`	Number | BigNumber | Boolean | Complex | Unit | Array | Matrix | null	Numerator
`y`	Number | BigNumber | Boolean | Complex | Unit | Array | Matrix | null	Denominator

Returns

Type	Description
Number | BigNumber | Complex | Unit | Array | Matrix	Quotient, `x ./ y`

Examples

```
math.dotDivide(2, 4);   // returns 0.5

a = [[9, 5], [6, 1]];
b = [[3, 2], [5, 2]];

math.dotDivide(a, b);   // returns [[3, 2.5], [1.2, 0.5]]
math.divide(a, b);      // returns [[1.75, 0.75], [-1.75, 2.25]]
```

Voir aussi : divide, multiply, dotMultiply

Function dotMultiply

Multiply two matrices element wise. The function accepts both matrices and scalar values.

Syntax : `math.dotMultiply(x, y)`

Parameters

Parameter	Type	Description
`x`	Number | BigNumber | Boolean | Complex | Unit | Array | Matrix | null	Left hand value
`y`	Number | BigNumber | Boolean | Complex | Unit | Array | Matrix | null	Right hand value

Returns

Type	Description
Number | BigNumber | Complex | Unit | Array | Matrix	Multiplication of `x` and `y`

Examples

```
math.dotMultiply(2, 4); // returns 8

a = [[9, 5], [6, 1]];
b = [[3, 2], [5, 2]];

math.dotMultiply(a, b); // returns [[27, 10], [30, 2]]
math.multiply(a, b);    // returns [[52, 28], [23, 14]]
```

Voir aussi : multiply, divide, dotDivide

Function dotPow

Calculates the power of x to y element wise.

Syntax : `math.dotPow(x, y)`

Parameters

Parameter	Type	Description
`x`	Number | BigNumber | Boolean | Complex | Unit | Array | Matrix | null	The base
`y`	Number | BigNumber | Boolean | Complex | Unit | Array | Matrix | null	The exponent

Returns

Type	Description
Number | BigNumber | Complex | Unit | Array | Matrix	The value of `x` to the power `y`

Examples

```
math.dotPow(2, 3);            // returns Number 8

var a = [[1, 2], [4, 3]];
math.dotPow(a, 2);            // returns Array [[1, 4], [16, 9]]
math.pow(a, 2);               // returns Array [[9, 8], [16, 17]]
```

Voir aussi : pow, sqrt, multiply

Function equal

Test whether two values are equal.

The function tests whether the relative difference between x and y is smaller than the configured epsilon. The function cannot be used to compare values smaller than approximately 2.22e-16.

For matrices, the function is evaluated element wise. In case of complex numbers, x.re must equal y.re, and x.im must equal y.im.

Values `null` and `undefined` are compared strictly, thus `null` is only equal to `null` and nothing else, and `undefined` is only equal to `undefined` and nothing else.

Syntax : **`math.equal(x, y)`**

Parameters

Parameter	Type	Description
`x`	Number | BigNumber | Boolean | Complex | Unit | String | Array | Matrix | null | undefined	First value to compare
`y`	Number | BigNumber | Boolean | Complex | Unit | String | Array | Matrix | null | undefined	Second value to compare

Returns

Type	Description
Boolean | Array | Matrix	Returns true when the compared values are equal, else returns false

Examples

```
math.equal(2 + 2, 3);         // returns false
math.equal(2 + 2, 4);         // returns true

var a = math.unit('50 cm');
var b = math.unit('5 m');
math.equal(a, b);             // returns true

var c = [2, 5, 1];
var d = [2, 7, 1];

math.equal(c, d);             // returns [true, false, true]
math.deepEqual(c, d);         // returns false

math.equal(0, null);          // returns false
```

Voir aussi : unequal, smaller, smallerEq, larger, largerEq, compare, deepEqual

Function eval

Evaluate an expression.

Syntax

```
math.eval(expr)
math.eval(expr, scope)
math.eval([expr1, expr2, expr3, ...])
math.eval([expr1, expr2, expr3, ...], scope)
```

Parameters

Parameter	Type	Description
`expr`	String | String[] | Matrix	The expression to be evaluated
`scope`	Object	Scope to read/write variables

Returns

Type	Description
*	The result of the expression

Examples

```
math.eval('(2+3)/4');                 // 1.25
math.eval('sqrt(3^2 + 4^2)');         // 5
math.eval('sqrt(-4)');                // 2i
math.eval(['a=3', 'b=4', 'a*b']);,    // [3, 4, 12]

var scope = {a:3, b:4};
math.eval('a * b', scope);            // 12
```

Voir aussi : parse, compile

Function exp

Calculate the exponent of a value. For matrices, the function is evaluated element wise.

Syntax : `math.exp(x)`

Parameters

Parameter	Type	Description
`x`	Number | BigNumber | Boolean | Complex | Array | Matrix | null	A number or matrix to exponentiate

Returns

Type	Description
Number | BigNumber | Complex | Array | Matrix	Exponent of `x`

Examples

```
math.exp(2);                  // returns Number 7.3890560989306495
math.pow(math.e, 2);          // returns Number 7.3890560989306495
math.log(math.exp(2));        // returns Number 2

math.exp([1, 2, 3]);
// returns Array [
//   2.718281828459045,
//   7.3890560989306495,
//   20.085536923187668
// ]
```

Voir aussi : log, pow

Function eye

Create a 2-dimensional identity matrix with size m x n or n x n. The matrix has ones on the diagonal and zeros elsewhere.

Syntax

```
math.eye(n)
math.eye(m, n)
math.eye([m, n])
```

Parameters

Parameter	Type	Description
`size`	...Number | Matrix | Array	The size for the matrix

Returns

Type	Description
Matrix | Array | Number	A matrix with ones on the diagonal.

Examples

```
math.eye(3);                    // returns [[1, 0, 0], [0, 1, 0], [0, 0, 1]]
math.eye(3, 2);                 // returns [[1, 0], [0, 1], [0, 0]]

var A = [[1, 2, 3], [4, 5, 6]];
math.eye(math.size(b));         // returns [[1, 0, 0], [0, 1, 0]]
```

Voir aussi : diag, ones, zeros, size, range

Function factorial

Compute the factorial of a value

Factorial only supports an integer value as argument. For matrices, the function is evaluated element wise.

Syntax : `math.factorial(n)`

Parameters

Parameter	Type	Description
`n`	Number | BigNumber | Array | Matrix | Boolean | null	An integer number

Returns

Type	Description
Number | BigNumber | Array | Matrix	The factorial of `n`

Examples

```
math.factorial(5);    // returns 120
math.factorial(3);    // returns 6
```

Voir aussi : combinations, permutations

Function filter

Sort the items in a matrix.

Syntax : `math.filter(x, test)`

Parameters

Parameter	Type	Description
`x`	Matrix | Array	A one dimensional matrix or array to filter
`test`	Function | RegExp	A function or regular expression to test items. When `test` is a function, it must return a boolean. All entries for which `test` returns true are returned.

Returns

Type	Description
Matrix | Array	Returns the filtered matrix.

Examples

```
function isPositive (x) {
  return x > 0;
}
math.filter([6, -2, -1, 4, 3], isPositive); // returns [6, 4, 3]

math.filter(["23", "foo", "100", "55", "bar"], /[0-9]+/); // returns ["23",
"100", "55"]
```

Voir aussi : forEach, map, sort

Function fix

Round a value towards zero. For matrices, the function is evaluated element wise.

Syntax : **`math.fix(x)`**

Parameters

Parameter	Type	Description
`x`	Number | BigNumber | Boolean | Complex | Array | Matrix | null	Number to be rounded

Returns

Type	Description
Number | BigNumber | Complex | Array | Matrix	Rounded value

Examples

```
math.fix(3.2);                // returns Number 3
math.fix(3.8);                // returns Number 3
math.fix(-4.2);               // returns Number -4
math.fix(-4.7);               // returns Number -4

var c = math.complex(3.2, -2.7);
math.fix(c);                  // returns Complex 3 - 2i

math.fix([3.2, 3.8, -4.7]);   // returns Array [3, 3, -4]
```

Voir aussi : ceil, floor, round

Function flatten

Flatten a multi dimensional matrix into a single dimensional matrix.

Syntax : `math.flatten(x)`

Parameters

Parameter	Type	Description
`x`	Matrix | Array	Matrix to be flattened

Returns

Type	Description
Matrix | Array	Returns the flattened matrix

Examples

```
math.flatten([[1,2], [3,4]]);   // returns [1, 2, 3, 4]
```

Voir aussi : concat, resize, size, squeeze

Function floor

Round a value towards minus infinity. For matrices, the function is evaluated element wise.

Syntax : **`math.floor(x)`**

Parameters

Parameter	Type	Description
`x`	Number | BigNumber | Boolean | Complex | Array | Matrix | null	Number to be rounded

Returns

Type	Description
Number | BigNumber | Complex | Array | Matrix	Rounded value

Examples

```
math.floor(3.2);              // returns Number 3
math.floor(3.8);              // returns Number 3
math.floor(-4.2);             // returns Number -5
math.floor(-4.7);             // returns Number -5

var c = math.complex(3.2, -2.7);
math.floor(c);                // returns Complex 3 - 3i

math.floor([3.2, 3.8, -4.7]); // returns Array [3, 3, -5]
```

Voir aussi : ceil, fix, round

Function forEach

Iterate over all elements of a matrix/array, and executes the given callback function.

Syntax : `math.forEach(x, callback)`

Parameters

Parameter	Type	Description
`x`	Matrix | Array	The matrix to iterate on.
`callback`	Function	The callback function is invoked with three parameters: the value of the element, the index of the element, and the Matrix/array being traversed.

Examples

```
math.forEach([1, 2, 3], function(value) {
  console.log(value);
});
// outputs 1, 2, 3
```

Voir aussi : filter, map, sort

Function format

Format a value of any type into a string.

Syntax :

```
math.format(value)
math.format(value, options)
math.format(value, precision)
math.format(value, fn)
```

Where

- `value: *` The value to be formatted
- `options: Object` An object with formatting options. Available options:
  - `notation: String` Number notation. Choose from:
    - 'fixed' Always use regular number notation. For example '123.40' and '14000000'
    - 'exponential' Always use exponential notation. For example '1.234e+2' and '1.4e+7'
    - 'auto' (default) Regular number notation for numbers having an absolute value between `lower` and `upper` bounds, and uses exponential notation elsewhere. Lower bound is included, upper bound is excluded. For example '123.4' and '1.4e7'.
  - `precision: Number` A number between 0 and 16 to round the digits of the number. In case of notations 'exponential' and 'auto', `precision` defines the total number of significant digits returned and is undefined by default. In case of notation 'fixed', `precision` defines the number of significant digits after the decimal point, and is 0 by default.
  - `exponential: Object` An object containing two parameters, {Number} lower and {Number} upper, used by notation 'auto' to determine when to return exponential notation. Default values are `lower=1e-3` and `upper=1e5`. Only applicable for notation `auto`.
  - `fn: Function` A custom formatting function. Can be used to override the built-in notations. Function `fn` is called with `value` as parameter and must return a string. Is useful for example to format all values inside a matrix in a particular way.

Parameters

Parameter	Type	Description
`value`	*	Value to be stringified
`options`	Object | Function | Number	Formatting options

Returns

Type	Description
String	The formatted value

Examples

```
math.format(6.4);                                  // returns '6.4'
math.format(1240000);                              // returns '1.24e6'
math.format(1/3);                                  // returns '0.3333333333333333'
math.format(1/3, 3);                                        // returns '0.333'
math.format(21385, 2);                                      // returns '21000'
math.format(12.071, {notation: 'fixed'});                   // returns '12'
math.format(2.3,    {notation: 'fixed', precision: 2});     // returns '2.30'
math.format(52.8,   {notation: 'exponential'});             // returns '5.28e+1'
```

Voir aussi : print

Function gcd

Calculate the greatest common divisor for two or more values or arrays.

For matrices, the function is evaluated element wise.

Syntax

```
math.gcd(a, b)
math.gcd(a, b, c, ...)
```

Parameters

Parameter	Type	Description
`args`	... Number | BigNumber | Boolean | Array | Matrix | null	Two or more integer numbers

Returns

Type	Description
Number | BigNumber | Array | Matrix	The greatest common divisor

Examples

```
math.gcd(8, 12);              // returns 4
math.gcd(-4, 6);              // returns 2
math.gcd(25, 15, -10);        // returns 5

math.gcd([8, -4], [12, 6]);   // returns [4, 2]
```

Voir aussi : lcm, xgcd

Function help

Retrieve help on a function or data type. Help files are retrieved from the documentation in math.expression.docs.

Syntax : `math.help(search)`

Parameters

Parameter	Type	Description
`search`	function | string | Object	A function or function name for which to get help

Returns

Type	Description
Help	A help object

Examples

```
console.log(math.help('sin').toString());
console.log(math.help(math.add).toString());
console.log(math.help(math.add).toJSON());
```

Function im

Get the imaginary part of a complex number. For a complex number `a + bi`, the function returns `b`.

For matrices, the function is evaluated element wise.

Syntax : `math.im(x)`

Parameters

Parameter	Type	Description
`x`	Number | BigNumber | Complex | Array | Matrix | Boolean | null	A complex number or array with complex numbers

Returns

Type	Description
Number | BigNumber | Array | Matrix	The imaginary part of x

Examples

```
var a = math.complex(2, 3);
math.re(a);                     // returns Number 2
math.im(a);                     // returns Number 3

math.re(math.complex('-5.2i')); // returns Number -5.2
math.re(math.complex(2.4));     // returns Number 0
```

Voir aussi : re, conj, abs, arg

Function import

Import functions from an object or a module

Syntax

```
math.import(object)
math.import(object, options)
```

Where

- `object: Object` An object with functions to be imported.
- `options: Object` An object with import options. Available options:
  - `override: boolean` If true, existing functions will be overwritten. False by default.
  - `wrap: boolean` If true, the functions will be wrapped in a wrapper function which converts data types like Matrix to primitive data types like Array. The wrapper is needed when extending math.js with libraries which do not support these data types. False by default.

Parameters

Parameter	Type	Description
`object`	String | Object	Object with functions to be imported.
`options`	Object	Import options.

Examples

```
// define new functions and variables
math.import({
  myvalue: 42,
  hello: function (name) {
    return 'hello, ' + name + '!';
  }
});

// use the imported function and variable
math.myvalue * 2;               // 84
math.hello('user');             // 'hello, user!'

// import the npm module numbers
// (must be installed first with `npm install numbers`)
math.import('numbers', {wrap: true});

math.fibonacci(7); // returns 13
```

Function index

Create an index. An Index can store ranges having start, step, and end for multiple dimensions. Matrix.get, Matrix.set, and math.subset accept an Index as input.

Syntax : **`math.index(range1, range2, ...)`**

Where

Each range can be any of:

Parameters

Parameter	Type	Description
`ranges`	...*	Zero or more ranges or numbers.

Returns

Type	Description
Index	Returns the created index

Examples

```
var math = math.js

var b = [1, 2, 3, 4, 5];
math.subset(b, math.index([1, 3]));     // returns [2, 3]

var a = math.matrix([[1, 2], [3, 4]]);
a.subset(math.index(0, 1));             // returns 2
a.subset(math.index(1, null));          // returns [3, 4]
```

Voir aussi : bignumber, boolean, complex, matrix, number, string, unit

Function inv

Calculate the inverse of a square matrix.

Syntax : `math.inv(x)`

Parameters

Parameter	Type	Description
`x`	Number | Complex | Array | Matrix	Matrix to be inversed

Returns

Type	Description
Number | Complex | Array | Matrix	The inverse of `x`.

Examples

```
math.inv([[1, 2], [3, 4]]);  // returns [[-2, 1], [1.5, -0.5]]
math.inv(4);                 // returns 0.25
1 / 4;                       // returns 0.25
```

Voir aussi : det, transpose

Function larger

Test whether value x is larger than y.

The function returns true when x is larger than y and the relative difference between x and y is larger than the configured epsilon. The function cannot be used to compare values smaller than approximately 2.22e-16.

For matrices, the function is evaluated element wise.

Syntax

```
math.larger(x, y)
```

Parameters

Parameter	Type	Description
x	Number | BigNumber | Boolean | Unit | String | Array | Matrix | null	First value to compare
y	Number | BigNumber | Boolean | Unit | String | Array | Matrix | null	Second value to compare

Returns

Type	Description
Boolean | Array | Matrix	Returns true when the x is larger than y, else returns false

Examples

```
math.larger(2, 3);             // returns false
math.larger(5, 2 + 2);         // returns true

var a = math.unit('5 cm');
var b = math.unit('2 inch');
math.larger(a, b);             // returns false
```

Voir aussi : equal, unequal, smaller, smallerEq, largerEq, compare

Function largerEq

Test whether value x is larger or equal to y.

The function returns true when x is larger than y or the relative difference between x and y is smaller than the configured epsilon. The function cannot be used to compare values smaller than approximately 2.22e-16.

For matrices, the function is evaluated element wise.

Syntax : **`math.largerEq(x, y)`**

Parameters

Parameter	Type	Description
`x`	Number | BigNumber | Boolean | Unit | String | Array | Matrix | null	First value to compare
`y`	Number | BigNumber | Boolean | Unit | String | Array | Matrix | null	Second value to compare

Returns

Type	Description
Boolean | Array | Matrix	Returns true when the x is larger or equal to y, else returns false

Examples

```
math.larger(2, 1 + 1);         // returns false
math.largerEq(2, 1 + 1);       // returns true
```

Voir aussi : equal, unequal, smaller, smallerEq, larger, compare

Function lcm

Calculate the least common multiple for two or more values or arrays.

lcm is defined as:

```
lcm(a, b) = abs(a * b) / gcd(a, b)
```

For matrices, the function is evaluated element wise.

Syntax

```
math.lcm(a, b)
math.lcm(a, b, c, ...)
```

Parameters

Parameter	Type	Description
`args`	... Number | BigNumber | Boolean | Array | Matrix | null	Two or more integer numbers

Returns

Type	Description
Number | BigNumber | Array | Matrix	The least common multiple

Examples

```
math.lcm(4, 6);               // returns 12
math.lcm(6, 21);              // returns 42
math.lcm(6, 21, 5);           // returns 210

math.lcm([4, 6], [6, 21]);    // returns [12, 42]
```

Voir aussi : gcd, xgcd

Function log

Calculate the logarithm of a value.

For matrices, the function is evaluated element wise.

Syntax

```
math.log(x)
math.log(x, base)
```

Parameters

Parameter	Type	Description
`x`	Number | BigNumber | Boolean | Complex | Array | Matrix | null	Value for which to calculate the logarithm.
`base`	Number | BigNumber | Boolean | Complex | null	Optional base for the logarithm. If not provided, the natural logarithm of `x` is calculated. Default value: e.

Returns

Type	Description
Number | BigNumber | Complex | Array | Matrix	Returns the logarithm of `x`

Examples

```
math.log(3.5);                  // returns 1.252762968495368
math.exp(math.log(2.4));        // returns 2.4

math.pow(10, 4);                // returns 10000
math.log(10000, 10);            // returns 4
math.log(10000) / math.log(10); // returns 4

math.log(1024, 2);              // returns 10
math.pow(2, 10);                // returns 1024
```

Voir aussi : exp, log10

Function log10

Calculate the 10-base of a value. This is the same as calculating `log(x, 10)`.

For matrices, the function is evaluated element wise.

Syntax : **`math.log10(x)`**

Parameters

Parameter	Type	Description
`x`	Number | BigNumber | Boolean | Complex | Array | Matrix | null	Value for which to calculate the logarithm.

Returns

Type	Description
Number | BigNumber | Complex | Array | Matrix	Returns the 10-base logarithm of `x`

Examples

```
math.log10(0.00001);            // returns -5
math.log10(10000);              // returns 4
math.log(10000) / math.log(10); // returns 4
math.pow(10, 4);                // returns 10000
```

Voir aussi : exp, log

Function map

Create a new matrix or array with the results of the callback function executed on each entry of the matrix/array.

Syntax : **`math.map(x, callback)`**

Parameters

Parameter	Type	Description
`x`	Matrix | Array	The matrix to iterate on.
`callback`	Function	The callback method is invoked with three parameters: the value of the element, the index of the element, and the matrix being traversed.

Returns

Type	Description
Matrix | array	Transformed map of x

Examples

```
math.map([1, 2, 3], function(value) {
  return value * value;
});  // returns [1, 4, 9]
```

Voir aussi : filter, forEach, sort

Function matrix

Create a Matrix. The function creates a new `math.type.Matrix` object from an `Array`. A Matrix has utility functions to manipulate the data in the matrix, like getting the size and getting or setting values in the matrix.

Syntax

```
math.matrix()       // creates an empty matrix
math.matrix(data)   // creates a matrix with initial data.
```

Parameters

Parameter	Type	Description
`data`	Array | Matrix	A multi dimensional array

Returns

Type	Description
Matrix	The created matrix

Examples

```
var m = math.matrix([[1, 2], [3, 4]);
m.size();                           // Array [2, 2]
m.resize([3, 2], 5);
m.valueOf();                        // Array [[1, 2], [3, 4], [5, 5]]
m.get([1, 0])                       // number 3
```

Voir aussi : bignumber, boolean, complex, index, number, string, unit

Function max

Compute the maximum value of a matrix or a list with values. In case of a multi dimensional array, the maximum of the flattened array will be calculated. When `dim` is provided, the maximum over the selected dimension will be calculated. Parameter `dim` is zero-based.

Syntax

```
math.max(a, b, c, ...)
math.max(A)
math.max(A, dim)
```

Parameters

Parameter	Type	Description
`args`	... *	A single matrix or or multiple scalar values

Returns

Type	Description
*	The maximum value

Examples

```
math.max(2, 1, 4, 3);                  // returns 4
math.max([2, 1, 4, 3]);                // returns 4

// maximum over a specified dimension (zero-based)
math.max([[2, 5], [4, 3], [1, 7]], 0); // returns [4, 7]
math.max([[2, 5], [4, 3]], [1, 7], 1); // returns [5, 4, 7]

math.max(2.7, 7.1, -4.5, 2.0, 4.1);    // returns 7.1
math.min(2.7, 7.1, -4.5, 2.0, 4.1);    // returns -4.5
```

mean, median, min, prod, std, sum, var

Function mean

Compute the mean value of matrix or a list with values. In case of a multi dimensional array, the mean of the flattened array will be calculated. When `dim` is provided, the maximum over the selected dimension will be calculated. Parameter `dim` is zero-based.

Syntax

```
mean.mean(a, b, c, ...)
mean.mean(A)
mean.mean(A, dim)
```

Parameters

Parameter	Type	Description
`args`	... *	A single matrix or or multiple scalar values

Returns

Type	Description
*	The mean of all values

Examples

```
math.mean(2, 1, 4, 3);                    // returns 2.5
math.mean([1, 2.7, 3.2, 4]);              // returns 2.725

math.mean([[2, 5], [6, 3], [1, 7]], 0);   // returns [3, 5]
math.mean([[2, 5], [6, 3], [1, 7]], 1);   // returns [3.5, 4.5, 4]
```

Voir aussi : median, min, max, sum, prod, std, var

Function median

Compute the median of a matrix or a list with values. The values are sorted and the middle value is returned. In case of an even number of values, the average of the two middle values is returned. Supported types of values are: Number, BigNumber, Unit

In case of a (multi dimensional) array or matrix, the median of all elements will be calculated.

Syntax

```
mean.median(a, b, c, ...)
mean.median(A)
```

Parameters

Parameter	Type	Description
`args`	... *	A single matrix or or multiple scalar values

Returns

Type	Description
*	The median

Examples

```
math.median(5, 2, 7);         // returns 5
math.median([3, -1, 5, 7]);   // returns 4
```

Voir aussi : mean, min, max, sum, prod, std, var

Function min

Compute the maximum value of a matrix or a list of values. In case of a multi dimensional array, the maximum of the flattened array will be calculated. When `dim` is provided, the maximum over the selected dimension will be calculated. Parameter `dim` is zero-based.

Syntax

```
math.min(a, b, c, ...)
math.min(A)
math.min(A, dim)
```

Parameters

Parameter	Type	Description
`args`	... *	A single matrix or or multiple scalar values

Returns

Type	Description
*	The minimum value

Examples

```
math.min(2, 1, 4, 3);                  // returns 1
math.min([2, 1, 4, 3]);                // returns 1

// maximum over a specified dimension (zero-based)
math.min([[2, 5], [4, 3], [1, 7]], 0); // returns [1, 3]
math.min([[2, 5], [4, 3], [1, 7]], 1); // returns [2, 3, 1]

math.max(2.7, 7.1, -4.5, 2.0, 4.1);    // returns 7.1
math.min(2.7, 7.1, -4.5, 2.0, 4.1);    // returns -4.5
```

Voir aussi : mean, median, max, prod, std, sum, var

Function mod

Calculates the modulus, the remainder of an integer division.

For matrices, the function is evaluated element wise.

The modulus is defined as : `x - y * floor(x / y)`

See http://en.wikipedia.org/wiki/Modulo_operation.

Syntax : **`math.mod(x, y)`**

Parameters

Parameter	Type	Description
`x`	Number | BigNumber | Boolean | Array | Matrix | null	Dividend
`y`	Number | BigNumber | Boolean | Array | Matrix | null	Divisor

Returns

Type	Description
Number | BigNumber | Array | Matrix	Returns the remainder of `x` divided by `y`.

Examples

```
math.mod(8, 3);                // returns 2
math.mod(11, 2);               // returns 1

function isOdd(x) {
  return math.mod(x, 2) != 0;
}

isOdd(2);                      // returns false
isOdd(3);                      // returns true
```

Voir aussi : divide

Function multiply

Multiply two values, `x * y`. The result is squeezed. For matrices, the matrix product is calculated.

Syntax : **`math.multiply(x, y)`**

Parameters

Parameter	Type	Description
`x`	Number | BigNumber | Boolean | Complex | Unit | Array | Matrix | null	First value to multiply
`y`	Number | BigNumber | Boolean | Complex | Unit | Array | Matrix | null	Second value to multiply

Returns

Type	Description
Number | BigNumber | Complex | Unit | Array | Matrix	Multiplication of `x` and `y`

Examples

```
math.multiply(4, 5.2);        // returns Number 20.8

var a = math.complex(2, 3);
var b = math.complex(4, 1);
math.multiply(a, b);          // returns Complex 5 + 14i

var c = [[1, 2], [4, 3]];
var d = [[1, 2, 3], [3, -4, 7]];
math.multiply(c, d);          // returns Array [[7, -6, 17], [13, -4, 33]]

var e = math.unit('2.1 km');
math.multiply(3, e);          // returns Unit 6.3 km
```

Voir aussi : divide

Function norm

Calculate the norm of a number, vector or matrix.

The second parameter p is optional. If not provided, it defaults to 2.

Syntax

```
math.norm(x)
math.norm(x, p)
```

Parameters

Parameter	Type	Description
x	Number | BigNumber | Complex | Boolean | Array | Matrix | null	Value for which to calculate the norm
p	Number | String	Vector space. Supported numbers include Infinity and -Infinity. Supported strings are: 'inf', '-inf', and 'fro' (The Frobenius norm) Default value: 2.

Returns

Type	Description
Number	the p-norm

Examples

```
math.abs(-3.5);                         // returns 3.5
math.norm(-3.5);                        // returns 3.5

math.norm(math.complex(3, -4));         // returns 5

math.norm([1, 2, -3], Infinity);        // returns 3
math.norm([1, 2, -3], -Infinity);       // returns 1

math.norm([3, 4], 2);                   // returns 5

math.norm([[1, 2], [3, 4]], 1)          // returns 6
math.norm([[1, 2], [3, 4]], 'inf');     // returns 7
math.norm([[1, 2], [3, 4]], 'fro');     // returns 5.477225575051661
```

Voir aussi : abs

Function nthRoot

Calculate the nth root of a value. The principal nth root of a positive real number A, is the positive real solution of the equation

```
x^root = A
```

For matrices, the function is evaluated element wise.

Syntax : **`math.nthRoot(a, root)`**

Parameters

Parameter	Type	Description
`a`	Number | BigNumber | Boolean | Array | Matrix | null	Value for which to calculate the nth root
`root`	Number | BigNumber | Boolean | null	The root. Default value: 2.

Returns

Type	Description
Number | Complex | Array | Matrix	Returns the nth root of `a`

Examples

```
math.nthRoot(9, 2);    // returns 3, as 3^2 == 9
math.sqrt(9);          // returns 3, as 3^2 == 9
math.nthRoot(64, 3);   // returns 4, as 4^3 == 64
```

Voir aussi : sqrt, pow

Function number

Create a number or convert a string, boolean, or unit to a number. When value is a matrix, all elements will be converted to number.

Syntax

```
math.number(value)
math.number(unit, valuelessUnit)
```

Parameters

Parameter	Type	Description
`value`	String | Number | Boolean | Array | Matrix | Unit | null	Value to be converted
`valuelessUnit`	Unit | string	A valueless unit, used to convert a unit to a number

Returns

Type	Description
Number | Array | Matrix	The created number

Examples

```
math.number(2);                         // returns number 2
math.number('7.2');                     // returns number 7.2
math.number(true);                      // returns number 1
math.number([true, false, true, true]); // returns [1, 0, 1, 1]
math.number(math.unit('52cm'), 'm');    // returns 0.52
```

Voir aussi : bignumber, boolean, complex, index, matrix, string, unit

Function ones

Create a matrix filled with ones. The created matrix can have one or multiple dimensions.

Syntax

```
math.ones(m)
math.ones(m, n)
math.ones([m, n])
math.ones([m, n, p, ...])
```

Parameters

Parameter	Type	Description
`size`	...Number | Array	The size of each dimension of the matrix

Returns

Type	Description
Array | Matrix | Number	A matrix filled with ones

Examples

```
math.ones(3);                    // returns [1, 1, 1]
math.ones(3, 2);                 // returns [[1, 1], [1, 1], [1, 1]]

var A = [[1, 2, 3], [4, 5, 6]];
math.zeros(math.size(A));        // returns [[1, 1, 1], [1, 1, 1]]
```

Voir aussi : zeros, eye, size, range

Function parse

Parse an expression. Returns a node tree, which can be evaluated by invoking node.eval();

Syntax

```
parse(expr)
parse(expr, options)
parse([expr1, expr2, expr3, ...])
parse([expr1, expr2, expr3, ...], options)
```

Parameters

Parameter	Type	Description
`expr`	String | String[] | Matrix	Expression to be parsed
`options`	{nodes: Object}	Available options: - `nodes` a set of custom nodes

Returns

Type	Description
Node | Node[]	node

Examples

```
var node = parse('sqrt(3^2 + 4^2)');
node.compile(math).eval(); // 5

var scope = {a:3, b:4}
var node = parse('a * b'); // 12
var code = node.compile(math);
code.eval(scope); // 12
scope.a = 5;
code.eval(scope); // 20

var nodes = math.parse(['a = 3', 'b = 4', 'a * b']);
nodes[2].compile(math).eval(); // 12
```

Function parser

Create a parser. The function creates a new `math.expression.Parser` object.

Syntax : **`math.parser()`**

Parameters

Parameter	Type	Description

Returns

Type	Description
Parser	Parser

Examples

```
var parser = new math.parser();

// evaluate expressions
var a = parser.eval('sqrt(3^2 + 4^2)'); // 5
var b = parser.eval('sqrt(-4)');        // 2i
var c = parser.eval('2 inch in cm');    // 5.08 cm
var d = parser.eval('cos(45 deg)');     // 0.7071067811865476

// define variables and functions
parser.eval('x = 7 / 2');               // 3.5
parser.eval('x + 3');                   // 6.5
parser.eval('function f(x, y) = x^y');  // f(x, y)
parser.eval('f(2, 3)');                 // 8

// get and set variables and functions
var x = parser.get('x');                // 7
var f = parser.get('f');                // function
var g = f(3, 2);                        // 9
parser.set('h', 500);
var i = parser.eval('h / 2');           // 250
parser.set('hello', function (name) {
  return 'hello, ' + name + '!';
});
parser.eval('hello("user")');           // "hello, user!"

// clear defined functions and variables
parser.clear();
```

Voir aussi : eval, compile, parse

Function permutations

Compute the number of ways of obtaining an ordered subset of `k` elements from a set of `n` elements.

Permutations only takes integer arguments. The following condition must be enforced: k <= n.

Syntax

```
math.permutations(n)
math.permutations(n, k)
```

Parameters

Parameter	Type	Description
`n`	Number | BigNumber	The number of objects in total
`k`	Number | BigNumber	The number of objects in the subset

Returns

Type	Description
Number | BigNumber	The number of permutations

Examples

```
math.permutations(5);     // 120
math.permutations(5, 3);  // 60
```

Voir aussi : combinations, factorial

Function pickRandom

Random pick a value from a one dimensional array. Array element is picked using a random function with uniform distribution.

Syntax : `math.pickRandom(array)`

Parameters

Parameter	Type	Description
`array`	Array	A one dimensional array

Returns

Type	Description
Number	One of the elements of the provided input array

Examples

```
math.pickRandom([3, 6, 12, 2]);       // returns one of the values in the array
```

Voir aussi : random, randomInt

Function pow

Calculates the power of x to y, `x ^ y`. Matrix exponentiation is supported for square matrices `x`, and positive integer exponents `y`.

Syntax : **`math.pow(x, y)`**

Parameters

Parameter	Type	Description
`x`	Number | BigNumber | Boolean | Complex | Array | Matrix | null	The base
`y`	Number | BigNumber | Boolean | Complex | null	The exponent

Returns

Type	Description
Number | BigNumber | Complex | Array | Matrix	The value of `x` to the power `y`

Examples

```
math.pow(2, 3);               // returns Number 8

var a = math.complex(2, 3);
math.pow(a, 2)                // returns Complex -5 + 12i

var b = [[1, 2], [4, 3]];
math.pow(b, 2);               // returns Array [[9, 8], [16, 17]]
```

Voir aussi : multiply, sqrt

Function print

Interpolate values into a string template.

Syntax

```
math.print(template, values)
math.print(template, values, precision)
```

Parameters

Parameter	Type	Description
`template`	String	A string containing variable placeholders.
`values`	Object	An object containing variables which will be filled in in the template.
`precision`	Number	Number of digits to format numbers. If not provided, the value will not be rounded.

Returns

Type	Description
String	Interpolated string

Examples

```
// the following outputs: 'Lucy is 5 years old'
math.print('Lucy is $age years old', {age: 5});

// the following outputs: 'The value of pi is 3.141592654'
math.print('The value of pi is $pi', {pi: math.pi}, 10);

// the following outputs: 'hello Mary! The date is 2013-03-23'
math.print('Hello $user.name! The date is $date', {
  user: {
    name: 'Mary',
  },
  date: new Date(2013, 2, 23).toISOString().substring(0, 10)
});
```

Voir aussi : format

Function prod

Compute the product of a matrix or a list with values. In case of a (multi dimensional) array or matrix, the sum of all elements will be calculated.

Syntax

```
math.prod(a, b, c, ...)
math.prod(A)
```

Parameters

Parameter	Type	Description
`args`	... *	A single matrix or or multiple scalar values

Returns

Type	Description
*	The product of all values

Examples

```
math.multiply(2, 3);           // returns 6
math.prod(2, 3);               // returns 6
math.prod(2, 3, 4);            // returns 24
math.prod([2, 3, 4]);          // returns 24
math.prod([[2, 5], [4, 3]]);   // returns 120
```

Voir aussi : mean, median, min, max, sum, std, var

Function random

Return a random number larger or equal to `min` and smaller than `max` using a uniform distribution.

Syntax

```
math.random()                // generate a random number between 0 and 1
math.random(max)             // generate a random number between 0 and max
math.random(min, max)        // generate a random number between min and max
math.random(size)            // generate a matrix with random numbers between 0
and 1
math.random(size, max)       // generate a matrix with random numbers between 0
and max
math.random(size, min, max)  // generate a matrix with random numbers between
min and max
```

Parameters

Parameter	Type	Description
`size`	Array | Matrix	If provided, an array or matrix with given size and filled with random values is returned
`min`	Number	Minimum boundary for the random value, included
`max`	Number	Maximum boundary for the random value, excluded

Returns

Type	Description
Number | Array | Matrix	A random number

Examples

```
math.random();       // returns a random number between 0 and 1
math.random(100);    // returns a random number between 0 and 100
math.random(30, 40); // returns a random number between 30 and 40
math.random([2, 3]); // returns a 2x3 matrix with random numbers between 0 and 1
```

Voir aussi : randomInt, pickRandom

Function randomInt

Return a random integer number larger or equal to `min` and smaller than `max` using a uniform distribution.

Syntax

```
math.randomInt()              // generate a random integer between 0 and 1
math.randomInt(max)           // generate a random integer between 0 and max
math.randomInt(min, max)      // generate a random integer between min and max
math.randomInt(size)          // generate a matrix with random integer
                                 between 0 and 1
math.randomInt(size, max)     // generate a matrix with random integer
                                 between 0 and max
math.randomInt(size, min, max) // generate a matrix with random integer
                                 between min and max
```

Parameters

Parameter	Type	Description
`size`	Array | Matrix	If provided, an array or matrix with given size and filled with random values is returned
`min`	Number	Minimum boundary for the random value, included
`max`	Number	Maximum boundary for the random value, excluded

Returns

Type	Description
Number | Array | Matrix	A random integer value

Examples

```
math.randomInt();       // returns a random integer between 0 and 1
math.randomInt(100);    // returns a random integer between 0 and 100
math.randomInt(30, 40); // returns a random integer between 30 and 40
math.randomInt([2, 3]); // returns a 2x3 matrix with random integers between 0
and 1
```

Voir aussi : random, pickRandom

Function range

Create an array from a range. By default, the range end is excluded. This can be customized by providing an extra parameter `includeEnd`.

Syntax :

```
math.range(str [, includeEnd])              // Create a range from a string,
                                            // where the string contains the
                                            // start, optional step, and end,
                                            // separated by a colon.
math.range(start, end [, includeEnd])       // Create a range with start and
                                            // end and a step size of 1.
math.range(start, end, step [, includeEnd]) // Create a range with start, step,
                                            // and end.
```

Where

- `str: String` A string 'start:end' or 'start:step:end'
- `start: {Number | BigNumber}` Start of the range
- `end: Number | BigNumber` End of the range, excluded by default, included when parameter includeEnd=true
- `step: Number | BigNumber` Step size. Default value is 1.
- `includeEnd: boolean` Option to specify whether to include the end or not. False by default.

Parameters

Parameter	Type	Description
`args`	*	Parameters describing the ranges `start`, `end`, and optional `step`.

Returns

Type	Description
Array | Matrix	range

Examples

```
math.range(2, 6);         // [2, 3, 4, 5]
math.range(2, -3, -1);    // [2, 1, 0, -1, -2]
math.range('2:1:6');      // [2, 3, 4, 5]
math.range(2, 6, true);   // [2, 3, 4, 5, 6]
```

Voir aussi : ones, zeros, size, subset

Function re

Get the real part of a complex number. For a complex number `a + bi`, the function returns `a`.

For matrices, the function is evaluated element wise.

Syntax : **`math.re(x)`**

Parameters

Parameter	Type	Description
`x`	Number | BigNumber | Complex | Array | Matrix | Boolean | null	A complex number or array with complex numbers

Returns

Type	Description
Number | BigNumber | Array | Matrix	The real part of x

Examples

```
var a = math.complex(2, 3);
math.re(a);                     // returns Number 2
math.im(a);                     // returns Number 3

math.re(math.complex('-5.2i')); // returns Number 0
math.re(math.complex(2.4));     // returns Number 2.4
```

Voir aussi : im, conj, abs, arg

Function resize

Resize a matrix

Syntax

```
math.resize(x, size)
math.resize(x, size, defaultValue)
```

Parameters

Parameter	Type	Description
`x`	* | Array | Matrix	Matrix to be resized
`size`	Array | Matrix	One dimensional array with numbers
`defaultValue`	Number | String	Zero by default, except in case of a string, in that case defaultValue = '' Default value: 0.

Returns

Type	Description
* | Array | Matrix	A resized clone of matrix `x`

Examples

```
math.resize([1, 2, 3, 4, 5], [3]); // returns Array  [1, 2, 3]
math.resize([1, 2, 3], [5], 0);    // returns Array  [1, 2, 3, 0, 0]
math.resize(2, [2, 3], 0);         // returns Matrix [[2, 0, 0], [0, 0, 0]]
math.resize("hello", [8], "!");    // returns String 'hello!!!'
```

Voir aussi : size, squeeze, subset

Function round

Round a value towards the nearest integer. For matrices, the function is evaluated element wise.

Syntax

```
math.round(x)
math.round(x, n)
```

Parameters

Parameter	Type	Description
`x`	Number | BigNumber | Boolean | Complex | Array | Matrix | null	Number to be rounded
`n`	Number | BigNumber | Boolean | Array | null	Number of decimals Default value: 0.

Returns

Type	Description
Number | BigNumber | Complex | Array | Matrix	Rounded value

Examples

```
math.round(3.2);              // returns Number 3
math.round(3.8);              // returns Number 4
math.round(-4.2);             // returns Number -4
math.round(-4.7);             // returns Number -5
math.round(math.pi, 3);       // returns Number 3.142
math.round(123.45678, 2);     // returns Number 123.46

var c = math.complex(3.2, -2.7);
math.round(c);                // returns Complex 3 - 3i

math.round([3.2, 3.8, -4.7]); // returns Array [3, 4, -5]
```

Voir aussi : ceil, fix, floor

Function sec

Calculate the secant of a value, defined as `sec(x) = 1/cos(x)`.

For matrices, the function is evaluated element wise.

Syntax : **`math.sec(x)`**

Parameters

Parameter	Type	Description
`x`	Number | Boolean | Complex | Unit | Array | Matrix | null	Function input

Returns

Type	Description
Number | Complex | Array | Matrix	Secant of x

Examples

```
math.sec(2);      // returns Number -2.4029979617223822
1 / math.cos(2);  // returns Number -2.4029979617223822
```

Voir aussi : cos, csc, cot

Function sech

Calculate the hyperbolic secant of a value, defined as `sech(x) = 1 / cosh(x)`.

For matrices, the function is evaluated element wise.

Syntax : **`math.sech(x)`**

Parameters

Parameter	Type	Description
`x`	Number | Boolean | Complex | Unit | Array | Matrix | null	Function input

Returns

Type	Description
Number | Complex | Array | Matrix	Hyperbolic secant of x

Examples

```
// sech(x) = 1/ cosh(x)
math.sech(0.5);       // returns 0.886818883970074
1 / math.cosh(0.5);   // returns 1.9190347513349437
```

Voir aussi : cosh, csch, coth

Function select

Wrap any value in a Selector, allowing to perform chained operations on the value.

All methods available in the math.js library can be called upon the selector, and then will be evaluated with the value itself as first argument. The selector can be closed by executing `selector.done()`, which returns the final value.

The Selector has a number of special functions:

- `done()` Finalize the chained operation and return the selectors value.
- `valueOf()` The same as `done()`
- `toString()` Executes `math.format()` onto the selectors value, returning a string representation of the value.

Syntax : **`math.select(value)`**

Parameters

Parameter	Type	Description
`value`	*	A value of any type on which to start a chained operation.

Returns

Type	Description
math.chaining.Selector	The created selector

Examples

```
math.select(3)
    .add(4)
    .subtract(2)
    .done();     // 5

math.select( [[1, 2], [3, 4]] )
    .set([1, 1], 8)
    .multiply(3)
    .done();     // [[24, 6], [9, 12]]
```

Function sign

Compute the sign of a value. The sign of a value x is:

- 1 when x > 1
- -1 when x < 0
- 0 when x == 0

For matrices, the function is evaluated element wise.

Syntax : `math.sign(x)`

Parameters

Parameter	Type	Description
`x`	Number | BigNumber | Boolean | Complex | Array | Matrix | null	The number for which to determine the sign

Returns

Type	Description
Number | BigNumber | Complex | Array | Matrix	The sign of `x`

Examples

```
math.sign(3.5);               // returns 1
math.sign(-4.2);              // returns -1
math.sign(0);                 // returns 0

math.sign([3, 5, -2, 0, 2]);  // returns [1, 1, -1, 0, 1]
```

Voir aussi : abs

Function sin

Calculate the sine of a value.

For matrices, the function is evaluated element wise.

Syntax : **`math.sin(x)`**

Parameters

Parameter	Type	Description
`x`	Number | Boolean | Complex | Unit | Array | Matrix | null	Function input

Returns

Type	Description
Number | Complex | Array | Matrix	Sine of x

Examples

```
math.sin(2);                      // returns Number 0.9092974268256813
math.sin(math.pi / 4);            // returns Number 0.7071067811865475
math.sin(math.unit(90, 'deg'));   // returns Number 1
math.sin(math.unit(30, 'deg'));   // returns Number 0.5

var angle = 0.2;
math.pow(math.sin(angle), 2) + math.pow(math.cos(angle), 2); // returns Number
~1
```

Voir aussi : cos, tan

Function sinh

Calculate the hyperbolic sine of a value, defined as `sinh(x) = 1/2 * (exp(x) - exp(-x))`.

For matrices, the function is evaluated element wise.

Syntax : **`math.sinh(x)`**

Parameters

Parameter	Type	Description
`x`	Number | Boolean | Complex | Unit | Array | Matrix | null	Function input

Returns

Type	Description
Number | Complex | Array | Matrix	Hyperbolic sine of x

Examples

```
math.sinh(0.5);       // returns Number 0.5210953054937474
```

Voir aussi : cosh, tanh

Function size

Calculate the size of a matrix or scalar.

Syntax : `math.size(x)`

Parameters

Parameter	Type	Description
`x`	Boolean | Number | Complex | Unit | String | Array | Matrix	A matrix

Returns

Type	Description
Array | Matrix	A vector with size of `x`.

Examples

```
math.size(2.3);                 // returns []
math.size('hello world');       // returns [11]

var A = [[1, 2, 3], [4, 5, 6]];
math.size(A);                   // returns [2, 3]
math.size(math.range(1,6));     // returns [5]
```

Voir aussi : resize, squeeze, subset

Function smaller

Test whether value x is smaller than y.

The function returns true when x is smaller than y and the relative difference between x and y is larger than the configured epsilon. The function cannot be used to compare values smaller than approximately 2.22e-16.

For matrices, the function is evaluated element wise.

Syntax : **`math.smaller(x, y)`**

Parameters

Parameter	Type	Description
`x`	Number | BigNumber | Boolean | Unit | String | Array | Matrix | null	First value to compare
`y`	Number | BigNumber | Boolean | Unit | String | Array | Matrix | null	Second value to compare

Returns

Type	Description
Boolean | Array | Matrix	Returns true when the x is smaller than y, else returns false

Examples

```
math.smaller(2, 3);            // returns true
math.smaller(5, 2 * 2);        // returns false

var a = math.unit('5 cm');
var b = math.unit('2 inch');
math.smaller(a, b);            // returns true
```

Voir aussi : equal, unequal, smallerEq, larger, largerEq, compare

Function smallerEq

Test whether value x is smaller or equal to y.

The function returns true when x is smaller than y or the relative difference between x and y is smaller than the configured epsilon. The function cannot be used to compare values smaller than approximately 2.22e-16. For matrices, the function is evaluated element wise.

Syntax : **`math.smallerEq(x, y)`**

Parameters

Parameter	Type	Description
`x`	Number | BigNumber | Boolean | Unit | String | Array | Matrix | null	First value to compare
`y`	Number | BigNumber | Boolean | Unit | String | Array | Matrix | null	Second value to compare

Returns

Type	Description
Boolean | Array | Matrix	Returns true when the x is smaller than y, else returns false

Examples

```
math.smaller(1 + 2, 3);        // returns false
math.smallerEq(1 + 2, 3);      // returns true
```

Voir aussi : equal, unequal, smaller, larger, largerEq, compare

Function sort

Sort the items in a matrix.

Syntax

```
math.sort(x)
math.sort(x, compare)
```

Parameters

Parameter	Type	Description
`x`	Matrix | Array	A one dimensional matrix or array to sort
`compare`	Function | 'asc' | 'desc'	An optional comparator function. The function is called as `compare(a, b)`, and must return 1 when a > b, -1 when a < b, and 0 when a == b. Default value: 'asc'.

Returns

Type	Description
Matrix | Array	Returns the sorted matrix.

Examples

```
math.sort([5, 10, 1]); // returns [1, 5, 10]
math.sort(['C', 'B', 'A', 'D']); // returns ['A', 'B', 'C', 'D']

function sortByLength (a, b) {
  return a.length - b.length;
}
math.sort(['Langdon', 'Tom', 'Sara'], sortByLength); // returns ['Tom', 'Sara',
'Langdon']
```

Voir aussi : filter, forEach, map

Function sqrt

Calculate the square root of a value.

For matrices, the function is evaluated element wise.

Syntax : **`math.sqrt(x)`**

Parameters

Parameter	Type	Description
`x`	Number | BigNumber | Boolean | Complex | Array | Matrix | null	Value for which to calculate the square root.

Returns

Type	Description
Number | BigNumber | Complex | Array | Matrix	Returns the square root of `x`

Examples

```
math.sqrt(25);                // returns 5
math.square(5);               // returns 25
math.sqrt(-4);                // returns Complex -2i
```

Voir aussi : square, multiply

Function square

Compute the square of a value, `x * x`. For matrices, the function is evaluated element wise.

Syntax : `math.square(x)`

Parameters

Parameter	Type	Description
`x`	Number | BigNumber | Boolean | Complex | Array | Matrix | null	Number for which to calculate the square

Returns

Type	Description
Number | BigNumber | Complex | Array | Matrix	Squared value

Examples

```
math.square(2);           // returns Number 4
math.square(3);           // returns Number 9
math.pow(3, 2);           // returns Number 9
math.multiply(3, 3);      // returns Number 9

math.square([1, 2, 3, 4]);  // returns Array [1, 4, 9, 16]
```

Voir aussi : multiply, cube, sqrt, pow

Function squeeze

Squeeze a matrix, remove inner and outer singleton dimensions from a matrix.

Syntax : **`math.squeeze(x)`**

Parameters

Parameter	Type	Description
`x`	Matrix | Array	Matrix to be squeezed

Returns

Type	Description
Matrix | Array	Squeezed matrix

Examples

```
math.squeeze([3]);           // returns 3
math.squeeze([[3]]);         // returns 3

var A = math.zeros(3, 1);    // returns [[0], [0], [0]] (size 3x1)
math.squeeze(A);             // returns [0, 0, 0] (size 3)

var B = math.zeros(1, 3);    // returns [[0, 0, 0]] (size 1x3)
math.squeeze(B);             // returns [0, 0, 0] (size 3)

// only inner and outer dimensions are removed
var C = math.zeros(2, 1, 3); // returns [[[0, 0, 0]], [[0, 0, 0]]] (size 2x1x3)
math.squeeze(C);             // returns [[[0, 0, 0]], [[0, 0, 0]]] (size 2x1x3)
```

Voir aussi : subset

Function std

Compute the standard deviation of a matrix or a list with values. The standard deviation is defined as the square root of the variance: `std(A) = sqrt(var(A))`. In case of a (multi dimensional) array or matrix, the standard deviation over all elements will be calculated.

Optionally, the type of normalization can be specified as second parameter. The parameter `normalization` can be one of the following values:

- 'unbiased' (default) The sum of squared errors is divided by (n - 1)
- 'uncorrected' The sum of squared errors is divided by n
- 'biased' The sum of squared errors is divided by (n + 1)

Syntax

```
math.std(a, b, c, ...)
math.std(A)
math.std(A, normalization)
```

Parameters

Parameter	Type	Description
`array`	Array | Matrix	A single matrix or or multiple scalar values
`normalization`	String	Determines how to normalize the variance. Choose 'unbiased' (default), 'uncorrected', or 'biased'. Default value: 'unbiased'.

Returns

Type	Description
*	The standard deviation

Examples

```
math.std(2, 4, 6);                    // returns 2
math.std([2, 4, 6, 8]);               // returns 2.581988897471611
math.std([2, 4, 6, 8], 'uncorrected'); // returns 2.23606797749979
math.std([2, 4, 6, 8], 'biased');     // returns 2

math.std([[1, 2, 3], [4, 5, 6]]);     // returns 1.8708286933869707
```

Voir aussi : mean, median, max, min, prod, sum, var

Function string

Create a string or convert any object into a string. Elements of Arrays and Matrices are processed element wise.

Syntax : `math.string(value)`

Parameters

Parameter	Type	Description
`value`	* | Array | Matrix | null	A value to convert to a string

Returns

Type	Description
String | Array | Matrix	The created string

Examples

```
math.string(4.2);               // returns string '4.2'
math.string(math.complex(3, 2); // returns string '3 + 2i'

var u = math.unit(5, 'km');
math.string(u.to('m'));         // returns string '5000 m'

math.string([true, false]);     // returns ['true', 'false']
```

Voir aussi : bignumber, boolean, complex, index, matrix, number, unit

Function subset

Get or set a subset of a matrix or string.

Syntax

```
math.subset(value, index)                                  // retrieve a subset
math.subset(value, index, replacement [, defaultValue])  // replace a subset
```

Parameters

Parameter	Type	Description
`matrix`	Array | Matrix | String	An array, matrix, or string
`index`	Index	An index containing ranges for each dimension
`replacement`	*	An array, matrix, or scalar. If provided, the subset is replaced with replacement. If not provided, the subset is returned
`defaultValue`	*	Default value, filled in on new entries when the matrix is resized. If not provided, new matrix elements will be left undefined. Default value: undefined.

Returns

Type	Description
Array | Matrix | String	Either the retrieved subset or the updated matrix.

Examples

```
// get a subset
var d = [[1, 2], [3, 4]];
math.subset(d, math.index(1, 0));        // returns 3
math.subset(d, math.index([0, 2], 1));   // returns [[2], [4]]

// replace a subset
var e = [];
var f = math.subset(e, math.index(0, [0, 2]), [5, 6]);  // f = [[5, 6]]
var g = math.subset(f, math.index(1, 1), 7, 0);         // g = [[5, 6], [0, 7]]
```

Voir aussi : size, resize, squeeze, index

Function subtract

Subtract two values, `x - y`. For matrices, the function is evaluated element wise.

Syntax : **`math.subtract(x, y)`**

Parameters

Parameter	Type	Description
`x`	Number | BigNumber | Boolean | Complex | Unit | Array | Matrix | null	Initial value
`y`	Number | BigNumber | Boolean | Complex | Unit | Array | Matrix | null	Value to subtract from `x`

Returns

Type	Description
Number | BigNumber | Complex | Unit | Array | Matrix	Subtraction of `x` and `y`

Examples

```
math.subtract(5.3, 2);         // returns Number 3.3

var a = math.complex(2, 3);
var b = math.complex(4, 1);
math.subtract(a, b);           // returns Complex -2 + 2i

math.subtract([5, 7, 4], 4);   // returns Array [1, 3, 0]

var c = math.unit('2.1 km');
var d = math.unit('500m');
math.subtract(c, d);           // returns Unit 1.6 km
```

Voir aussi : add

Function sum

Compute the sum of a matrix or a list with values. In case of a (multi dimensional) array or matrix, the sum of all elements will be calculated.

Syntax

```
math.sum(a, b, c, ...)
math.sum(A)
```

Parameters

Parameter	Type	Description
`args`	... *	A single matrix or or multiple scalar values

Returns

Type	Description
*	The sum of all values

Examples

```
math.sum(2, 1, 4, 3);               // returns 10
math.sum([2, 1, 4, 3]);             // returns 10
math.sum([[2, 5], [4, 3], [1, 7]]); // returns 22
```

Voir aussi : mean, median, min, max, prod, std, var

Function tan

Calculate the tangent of a value. `tan(x)` is equal to `sin(x) / cos(x)`.

For matrices, the function is evaluated element wise.

Syntax : **`math.tan(x)`**

Parameters

Parameter	Type	Description
`x`	Number | Boolean | Complex | Unit | Array | Matrix | null	Function input

Returns

Type	Description
Number | Complex | Array | Matrix	Tangent of x

Examples

```
math.tan(0.5);                    // returns Number 0.5463024898437905
math.sin(0.5) / math.cos(0.5);    // returns Number 0.5463024898437905
math.tan(math.pi / 4);            // returns Number 1
math.tan(math.unit(45, 'deg'));   // returns Number 1
```

Voir aussi : atan, sin, cos

Function tanh

Calculate the hyperbolic tangent of a value, defined as `tanh(x) = (exp(2 * x) - 1) / (exp(2 * x) + 1)`.

For matrices, the function is evaluated element wise.

Syntax : **`math.tanh(x)`**

Parameters

Parameter	Type	Description
`x`	Number | Boolean | Complex | Unit | Array | Matrix | null	Function input

Returns

Type	Description
Number | Complex | Array | Matrix	Hyperbolic tangent of x

Examples

```
// tanh(x) = sinh(x) / cosh(x) = 1 / coth(x)
math.tanh(0.5);                   // returns 0.46211715726000974
math.sinh(0.5) / math.cosh(0.5);  // returns 0.46211715726000974
1 / math.coth(0.5);               // returns 0.46211715726000974
```

Voir aussi : sinh, cosh, coth

Function to

Change the unit of a value.

For matrices, the function is evaluated element wise.

Syntax : `math.to(x, unit)`

Parameters

Parameter	Type	Description
`x`	Unit | Array | Matrix	The unit to be converted.
`unit`	Unit | Array | Matrix	New unit. Can be a string like "cm" or a unit without value.

Returns

Type	Description
Unit | Array | Matrix	value with changed, fixed unit.

Examples

```
math.to(math.unit('2 inch'), 'cm');                    // returns Unit 5.08 cm
math.to(math.unit('2 inch'), math.unit(null, 'cm'));   // returns Unit 5.08 cm
math.to(math.unit(16, 'bytes'), 'bits');               // returns Unit 128 bits
```

Voir aussi : unit

Function transpose

Transpose a matrix. All values of the matrix are reflected over its main diagonal. Only two dimensional matrices are supported.

Syntax : `math.transpose(x)`

Parameters

Parameter	Type	Description
`x`	Array | Matrix	Matrix to be transposed

Returns

Type	Description
Array | Matrix	The transposed matrix

Examples

```
var A = [[1, 2, 3], [4, 5, 6]];
math.transpose(A);               // returns [[1, 4], [2, 5], [3, 6]]
```

Voir aussi : diag, inv, subset, squeeze

Function typeof

Determine the type of a variable.

Function `typeof` recognizes the following types of objects:

Object	Returns	Example
Array	`'array'`	`math.typeof ([1, 2, 3])`
boolean	`'boolean'`	`math.typeof (true)`
Date	`'date'`	`math.typeof (new Date())`
null	`'null'`	`math.typeof(null)`
number	`'number'`	`math.typeof(3.5)`
Object	`'object'`	`math.typeof ({a: 2, b: 3})`
RegExp	`'regexp'`	`math.typeof (/a regexp/)`
string	`'string'`	`math.typeof ('hello world')`
undefined	`'undefined'`	`math.typeof(undefined)`
math.chaining.Selector	`'selector'`	`math.typeof (math.select(2))`
math.type.BigNumber	`'bignumber'`	`math.typeof (math.bignumber('2.3e500'))`
math.type.Complex	`'complex'`	`math.typeof (math.complex(2, 3))`
math.type.Help	`'help'`	`math.typeof (math.help('sqrt'))`
math.type.Index	`'index'`	`math.typeof (math.index(1, 3))`
math.type.Matrix	`'matrix'`	`math.typeof (math.matrix([[1,2], [3, 4]]))`
math.type.Range	`'range'`	`math.typeof (math.range(0, 10))`
math.type.Unit	`'unit'`	`math.typeof (math.unit('45 deg'))`

Syntax : **`math.typeof(x)`**

Parameters

Parameter	Type	Description
`x`	*	The variable for which to test the type.

Returns

Type	Description
String	Lower case type, for example 'number', 'string', 'array'.

Examples

```
math.typeof(3.5);                     // returns 'number'
math.typeof(math.complex('2 - 4i'));  // returns 'complex'
math.typeof(math.unit('45 deg'));     // returns 'unit'
math.typeof('hello world');           // returns 'string'
```

Function unaryMinus

Inverse the sign of a value, apply a unary minus operation.

For matrices, the function is evaluated element wise. Boolean values and strings will be converted to a number. For complex numbers, both real and complex value are inverted.

Syntax : `math.unaryMinus(x)`

Parameters

Parameter	Type	Description
x	Number | BigNumber | Boolean | String | Complex | Unit | Array | Matrix | null	Number to be inverted.

Returns

Type	Description
Number | BigNumber | Complex | Unit | Array | Matrix	Returns the value with inverted sign.

Examples

```
math.unaryMinus(3.5);      // returns -3.5
math.unaryMinus(-4.2);     // returns 4.2
```

Voir aussi : add, subtract, unaryPlus

Function unaryPlus

Unary plus operation. Boolean values and strings will be converted to a number, numeric values will be returned as is.

For matrices, the function is evaluated element wise.

Syntax : `math.unaryPlus(x)`

Parameters

Parameter	Type	Description
`x`	Number | BigNumber | Boolean | String | Complex | Unit | Array | Matrix | null	Input value

Returns

Type	Description
Number | BigNumber | Complex | Unit | Array | Matrix	Returns the input value when numeric, converts to a number when input is non-numeric.

Examples

```
math.unaryPlus(3.5);      // returns 3.5
math.unaryPlus(1);     // returns 1
```

Voir aussi : unaryMinus, add, subtract

Function unequal

Test whether two values are unequal.

The function tests whether the relative difference between x and y is larger than the configured epsilon. The function cannot be used to compare values smaller than approximately 2.22e-16.

For matrices, the function is evaluated element wise. In case of complex numbers, x.re must unequal y.re, or x.im must unequal y.im.

Values `null` and `undefined` are compared strictly, thus `null` is unequal with everything except `null`, and `undefined` is unequal with everying except. `undefined`.

Syntax : **`math.unequal(x, y)`**

Parameters

Parameter	Type	Description
`x`	Number | BigNumber | Boolean | Complex | Unit | String | Array | Matrix | null | undefined	First value to compare
`y`	Number | BigNumber | Boolean | Complex | Unit | String | Array | Matrix | null | undefined	Second value to compare

Returns

Type	Description
Boolean | Array | Matrix	Returns true when the compared values are unequal, else returns false

Examples

```
math.unequal(2 + 2, 3);       // returns true
math.unequal(2 + 2, 4);       // returns false

var a = math.unit('50 cm');
var b = math.unit('5 m');
math.unequal(a, b);           // returns false

var c = [2, 5, 1];
var d = [2, 7, 1];

math.unequal(c, d);           // returns [false, true, false]
math.deepEqual(c, d);         // returns false

math.unequal(0, null);        // returns true
```

Voir aussi : equal, deepEqual, smaller, smallerEq, larger, largerEq, compare

Function unit

Create a unit. Depending on the passed arguments, the function will create and return a new math.type.Unit object. When a matrix is provided, all elements will be converted to units.

Syntax :

```
math.unit(unit : string)
math.unit(value : number, unit : string)
```

Parameters

Parameter	Type	Description
`args`	* | Array | Matrix	A number and unit.

Returns

Type	Description
Unit | Array | Matrix	The created unit

Examples

```
var a = math.unit(5, 'cm');    // returns Unit 50 mm
var b = math.unit('23 kg');    // returns Unit 23 kg
a.to('m');                     // returns Unit 0.05 m
```

Voir aussi : bignumber, boolean, complex, index, matrix, number, string

Function var

Compute the variance of a matrix or a list with values. In case of a (multi dimensional) array or matrix, the variance over all elements will be calculated.

Optionally, the type of normalization can be specified as second parameter. The parameter `normalization` can be one of the following values:

- 'unbiased' (default) The sum of squared errors is divided by (n - 1)
- 'uncorrected' The sum of squared errors is divided by n
- 'biased' The sum of squared errors is divided by (n + 1) Note that older browser may not like the variable name `var`. In that case, the function can be called as `math['var'](...)` instead of `math.var(...)`.

Syntax

```
math.var(a, b, c, ...)
math.var(A)
math.var(A, normalization)
```

Parameters

Parameter	Type	Description
`array`	Array | Matrix	A single matrix or or multiple scalar values
`normalization`	String	Determines how to normalize the variance. Choose 'unbiased' (default), 'uncorrected', or 'biased'. Default value: 'unbiased'.

Returns

Type	Description
*	The variance

Examples

```
math.var(2, 4, 6);                    // returns 4
math.var([2, 4, 6, 8]);               // returns 6.666666666666667
math.var([2, 4, 6, 8], 'uncorrected'); // returns 5
math.var([2, 4, 6, 8], 'biased');      // returns 4

math.var([[1, 2, 3], [4, 5, 6]]);     // returns 3.5
```

Voir aussi : mean, median, max, min, prod, std, sum

Function xgcd

Calculate the extended greatest common divisor for two values. See
http://en.wikipedia.org/wiki/Extended_Euclidean_algorithm.
http://fr.wikipedia.org/wiki/Algorithme_d%27Euclide_%C3%A9tendu

Syntax : `math.xgcd(a, b)`

Parameters

Parameter	Type	Description
`a`	Number | BigNumber | Boolean	An integer number
`b`	Number | BigNumber | Boolean	An integer number

Returns

Type	Description
Array	Returns an array containing 3 integers `[div, m, n]` where `div = gcd(a, b)` and `a*m + b*n = div`

Examples

```
math.xgcd(8, 12);             // returns [4, -1, 1]
math.gcd(8, 12);              // returns 4
math.xgcd(36163, 21199);      // returns [1247, -7, 12]
```

Voir aussi : gcd, lcm

Function zeros

Create a matrix filled with zeros. The created matrix can have one or multiple dimensions.

Syntax

```
math.zeros(m)
math.zeros(m, n)
math.zeros([m, n])
math.zeros([m, n, p, ...])
```

Parameters

Parameter	Type	Description
`size`	...Number | Array	The size of each dimension of the matrix

Returns

Type	Description
Array | Matrix | Number	A matrix filled with zeros

Examples

```
math.zeros(3);                  // returns [0, 0, 0]
math.zeros(3, 2);               // returns [[0, 0], [0, 0], [0, 0]]

var A = [[1, 2, 3], [4, 5, 6]];
math.zeros(math.size(A));       // returns [[0, 0, 0], [0, 0, 0]]
```

Voir aussi : ones, eye, size, range

History

2014-11-22, version 1.1.1

- Fixed Unit divided by Number returning zero.
- Fixed BigNumber downgrading to Number for a negative base in `pow`.
- Fixed some typos in error messaging (thanks @andy0130tw) and docs.

2014-11-15, version 1.1.0

- Implemented functions `dot` (dot product), `cross` (cross product), and `nthRoot`.
- Officially opened up the API of expression trees:
    - Documented the API.
    - Implemented recursive functions `clone`, `map`, `forEach`, `traverse`, `transform`, and `filter` for expression trees.
    - Parameter `index` in the callbacks of `map` and `forEach` are now cloned for every callback.
    - Some internal refactoring inside nodes to make the API consistent:
    - Renamed `params` to `args` and vice versa to make things consistent.
    - Renamed `Block.nodes` to `Block.blocks`.
    - `FunctionNode` now has a `name: string` instead of a `symbol: SymbolNode`.
    - Changed constructor of `RangeNode` to `new RangeNode(start: Node, end: Node [, step: Node])`.
    - Nodes for a `BlockNode` must now be passed via the constructor instead of via a function `add`.
- Fixed `2e` giving a syntax error instead of being parsed as `2 * e`.

2014-09-12, version 1.0.1

- Disabled array notation for ranges in a matrix index in the expression parser (it is confusing and redundant there).
- Fixed a regression in the build of function subset not being able to return a scalar.
- Fixed some missing docs and broken links in the docs.

2014-09-04, version 1.0.0

- Implemented a function `filter(x, test)`.
- Removed `math.distribution` for now, needs some rethinking.
- `math.number` can convert units to numbers (requires a second argument)
- Fixed some precedence issues with the range and conversion operators.
- Fixed an zero-based issue when getting a matrix subset using an index containing a matrix.

2014-08-21, version 0.27.0

- Implemented functions `sort(x [, compare])` and `flatten(x)`.
- Implemented support for `null` in all functions.
- Implemented support for "rawArgs" functions in the expression parser. Raw functions are

invoked with unevaluated parameters (nodes).
- Expressions in the expression parser can now be spread over multiple lines, like '2 +\n3'.
- Changed default value of the option `wrap` of function `math.import` to false.
- Changed the default value for new entries in a resized matrix when to zero. To leave new entries uninitialized, use the new constant `math.uninitialized` as default value.
- Renamed transform property from `__transform__` to `transform`, and documented the transform feature.
- Fixed a bug in `math.import` not applying options when passing a module name.
- A returned matrix subset is now only squeezed when the `index` consists of scalar values, and no longer for ranges resolving into a single value.

2014-08-03, version 0.26.0

- A new instance of math.js can no longer be created like `math([options])`, to prevent side effects from math being a function instead of an object. Instead, use the function `math.create([options])` to create a new instance.
- Implemented `BigNumber` support for all constants: `pi`, `tau`, `e`, `phi`, `E`, `LN2`, `LN10`, `LOG2E`, `LOG10E`, `PI`, `SQRT1_2`, and `SQRT2`.
- Implemented `BigNumber` support for functions `gcd`, `xgcd`, and `lcm`.
- Fixed function `gxcd` returning an Array when math.js was configured as `{matrix: 'matrix'}`.
- Multi-line expressions now return a `ResultSet` instead of an `Array`.
- Implemented transforms (used right now to transform one-based indices to zero-based for expressions).
- When used inside the expression parser, functions `concat`, `min`, `max`, and `mean` expect an one-based dimension number.
- Functions `map` and `forEach` invoke the callback with one-based indices when used from within the expression parser.
- When adding or removing dimensions when resizing a matrix, the dimensions are added/removed from the inner side (right) instead of outer side (left).
- Improved index out of range errors.
- Fixed function `concat` not accepting a `BigNumber` for parameter `dim`.
- Function `squeeze` now squeezes both inner and outer singleton dimensions.
- Output of getting a matrix subset is not automatically squeezed anymore except for scalar output.
- Renamed `FunctionNode` to `FunctionAssignmentNode`, and renamed `ParamsNode` to `FunctionNode` for more clarity.
- Fixed broken auto completion in CLI.
- Some minor fixes.

2014-07-01, version 0.25.0

- The library now immediately returns a default instance of mathjs, there is no need to instantiate math.js in a separate step unless one ones to set configuration options:

```
// instead of:
var mathjs = require('mathjs'),  // load math.js
    math = mathjs();             // create an instance

// just do:
var math = require('mathjs');
```

- Implemented support for implicit multiplication, like `math.eval('2a', {a:3})` and `math.eval('(2+3)(1-3)')`. This changes behavior of matrix indexes as well: an expression like `[...][...]` is not evaluated as taking a subset of the first matrix, but as an implicit multiplication of two matrices.
- Removed utility function `ifElse`. This function is redundant now the expression parser has a conditional operator `a ? b : c`.
- Fixed a bug with multiplying a number with a temperature, like `math.eval('10 * celsius')`.
- Fixed a bug with symbols having value `undefined` not being evaluated.

2014-06-20, version 0.24.1

- Something went wrong with publishing on npm.

2014-06-20, version 0.24.0

- Added constant `null`.
- Functions `equal` and `unequal` support `null` and `undefined` now.
- Function `typeof` now recognizes regular expressions as well.
- Objects `Complex`, `Unit`, and `Help` now return their string representation when calling `.valueOf()`.
- Changed the default number of significant digits for BigNumbers from 20 to 64.
- Changed the behavior of the conditional operator (a ? b : c) to lazy evaluating.
- Fixed imported, wrapped functions not accepting `null` and `undefined` as function arguments.

2014-06-10, version 0.23.0

- Renamed some functions (everything now has a logical, camel case name):
  - Renamed functions `edivide`, `emultiply`, and `epow` to `dotDivide`, `dotMultiply`, and `dotPow` respectively.
  - Renamed functions `smallereq` and `largereq` to `smallerEq` and `largerEq`.
  - Renamed function `unary` to `unaryMinus` and added support for strings.
- `end` is now a reserved keyword which cannot be used as function or symbol name in the expression parser, and is not allowed in the scope against which an expression is evaluated.
- Implemented function `unaryPlus` and unary plus operator.
- Implemented function `deepEqual` for matrix comparisons.
- Added constant `phi`, the golden ratio (`phi = 1.618...`).
- Added constant `version`, returning the version number of math.js as string.
- Added unit `drop` (`gtt`).
- Fixed not being able to load math.js using AMD/require.js.
- Changed signature of `math.parse(expr, nodes)` to `math.parse(expr, options)` where `options: {nodes: Object.<String, Node>}`
- Removed matrix support from conditional function `ifElse`.
- Removed automatic assignment of expression results to variable `ans`. This functionality can be restored by pre- or postprocessing every evaluation, something like:

```
function evalWithAns (expr, scope) {
```

```
  var ans = math.eval(expr, scope);
  if (scope) {
    scope.ans = ans;
  }
  return ans;
}
```

2014-05-22, version 0.22.0

- Implemented support to export expressions to LaTeX. Thanks Niels Heisterkamp (@nheisterkamp).
- Output of matrix multiplication is now consistently squeezed.
- Added reference documentation in the section /docs/reference.
- Fixed a bug in multiplying units without value with a number (like `5 * cm`).
- Fixed a bug in multiplying two matrices containing vectors (worked fine for arrays).
- Fixed random functions not accepting Matrix as input, and always returning a Matrix as output.

2014-05-13, version 0.21.1

- Removed `crypto` library from the bundle.
- Deprecated functions `Parser.parse` and `Parser.compile`. Use `math.parse` and `math.compile` instead.
- Fixed function `add` not adding strings and matrices element wise.
- Fixed parser not being able to evaluate an exponent followed by a unary minus like `2^-3`, and a transpose followed by an index like `[3]'[1]`.

2014-04-24, version 0.21.0

- Implemented trigonometric hyperbolic functions `cosh`, `coth`, `csch`, `sech`, `sinh`, `tanh`. Thanks Rogelio J. Baucells (@rjbaucells).
- Added property `type` to all expression nodes in an expression tree.
- Fixed functions `log`, `log10`, `pow`, and `sqrt` not supporting complex results from BigNumber input (like `sqrt(bignumber(-4))`).

2014-04-16, version 0.20.0

- Switched to module `decimal.js` for BigNumber support, instead of `bignumber.js`.
- Implemented support for polar coordinates to the `Complex` datatype. Thanks Finn Pauls (@finnp).
- Implemented BigNumber support for functions `exp`, `log`, and `log10`.
- Implemented conditional operator `a ? b : c` in expression parser.
- Improved floating point comparison: the functions now check whether values are nearly equal, against a configured maximum relative difference `epsilon`. Thanks Rogelio J. Baucells (@rjbaucells).
- Implemented function `norm`. Thanks Rogelio J. Baucells (@rjbaucells).
- Improved function `ifElse`, is now specified for special data types too.
- Improved function `det`. Thanks Bryan Cuccioli (@bcuccioli).
- Implemented `BigNumber` support for functions `det` and `diag`.
- Added unit alias `lbs` (pound mass).

- Changed configuration option `decimals` to `precision` (applies to BigNumbers only).
- Fixed support for element-wise comparisons between a string and a matrix.
- Fixed: expression parser now trows IndexErrors with one-based indices instead of zero-based.
- Minor bug fixes.

2014-03-30, version 0.19.0

- Implemented functions `compare`, `sum`, `prod`, `var`, `std`, `median`.
- Implemented function `ifElse` Thanks @mtraynham.
- Minor bug fixes.

2014-02-15, version 0.18.1

- Added unit `feet`.
- Implemented function `compile` (shortcut for parsing and then compiling).
- Improved performance of function `pow` for matrices. Thanks @hamadu.
- Fixed broken auto completion in the command line interface.
- Fixed an error in function `combinations` for large numbers, and improved performance of both functions `combinations` and `permutations`.

2014-01-18, version 0.18.0

- Changed matrix index notation of expression parser from round brackets to square brackets, for example `A[1, 1:3]` instead of `A(1, 1:3)`.
- Removed need to use the `function` keyword for function assignments in the expression parser, you can define a function now like `f(x) = x^2`.
- Implemented a compilation step in the expression parser: expressions are compiled into JavaScript, giving much better performance (easily 10x as fast).
- Renamed unit conversion function and operator `in` to `to`. Operator `in` is still available in the expression parser as an alias for `to`. Added unit `in`, an abbreviation for `inch`. Thanks Elijah Insua (@tmpvar).
- Added plurals and aliases for units.
- Implemented an argument `includeEnd` for function `range` (false by default).
- Ranges in the expression parser now support big numbers.
- Implemented functions `permutations` and `combinations`. Thanks Daniel Levin (@daniel-levin).
- Added lower case abbreviation `l` for unit litre.

2013-12-19, version 0.17.1

- Fixed a bug with negative temperatures.
- Fixed a bug with prefixes of units squared meter `m2` and cubic meter `m3`.

2013-12-12, version 0.17.0

- Renamed and flattened configuration settings:
    - `number.defaultType` is now `number`.
    - `number.precision` is now `decimals`.

    - `matrix.defaultType` is now `matrix`.
- Function `multiply` now consistently outputs a complex number on complex input.
- Fixed `mod` and `in` not working as function (only as operator).
- Fixed support for old browsers (IE8 and older), compatible when using es5-shim.
- Fixed support for Java's ScriptEngine.

2013-11-28, version 0.16.0

- Implemented BigNumber support for arbitrary precision calculations. Added settings `number.defaultType` and `number.precision` to configure big numbers.
- Documentation is extended.
- Removed utility functions `isScalar`, `toScalar`, `isVector`, `toVector` from `Matrix` and `Range`. Use `math.squeeze` and `math.size` instead.
- Implemented functions `get` and `set` on `Matrix`, for easier and faster retrieval/replacement of elements in a matrix.
- Implemented function `resize`, handling matrices, scalars, and strings.
- Functions `ones` and `zeros` now return an empty matrix instead of a number 1 or 0 when no arguments are provided.
- Implemented functions `min` and `max` for `Range` and `Index`.
- Resizing matrices now leaves new elements undefined by default instead of filling them with zeros. Function `resize` now has an extra optional parameter `defaultValue`.
- Range operator `:` in expression parser has been given a higher precedence.
- Functions don't allow arguments of unknown type anymore.
- Options be set when constructing a math.js instance or using the new function `config(options`. Options are no longer accessible via `math.options`.
- Renamed `scientific` notation to `exponential` in function `format`.
- Function `format` outputs exponential notation with positive exponents now always with + sign, so outputs `2.1e+3` instead of `2.1e3`.
- Fixed function `squeeze` not being able squeeze into a scalar.
- Some fixes and performance improvements in the `resize` and `subset` functions.
- Function `size` now adheres to the option `matrix.defaultType` for scalar input.
- Minor bug fixes.

2013-10-26, version 0.15.0

- Math.js must be instantiated now, static calls are no longer supported. Usage:
    - node.js: `var math = require('mathjs')();`
    - browser: `var math = mathjs();`
- Implemented support for multiplying vectors with matrices.
- Improved number formatting:
    - Function `format` now support various options: precision, different notations (`fixed`, `scientific`, `auto`), and more.
    - Numbers are no longer rounded to 5 digits by default when formatted.
    - Implemented a function `format` for `Matrix`, `Complex`, `Unit`, `Range`, and `Selector` to format using options.
    - Function `format` does only stringify values now, and has a new parameter `precision` to round to a specific number of digits.
    - Removed option `math.options.precision`, use `math.format(value [,`

`precision])` instead.
    - Fixed formatting numbers as scientific notation in some cases returning a zero digit left from the decimal point. (like "0.33333e8" rather than "3.3333e7"). Thanks @husayt.
- Implemented a function `print` to interpolate values in a template string, this functionality was moved from the function `format`.
- Implemented statistics function `mean`. Thanks Guillermo Indalecio Fernandez (@guillermobox).
- Extended and changed `max` and `min` for multi dimensional matrices: they now return the maximum and minimum of the flattened array. An optional second argument `dim` allows to calculate the `max` or `min` for specified dimension.
- Renamed option `math.options.matrix.default` to `math.options.matrix.defaultType`.
- Removed support for comparing complex numbers in functions `smaller`, `smallereq`, `larger`, `largereq`. Complex numbers cannot be ordered.

2013-10-08, version 0.14.0

- Introduced an option `math.options.matrix.default` which can have values `matrix` (default) or `array`. This option is used by the functions `eye`, `ones`, `range`, and `zeros`, to determine the type of matrix output.
- Getting a subset of a matrix will automatically squeeze the resulting subset, setting a subset of a matrix will automatically unsqueeze the given subset.
- Removed concatenation of nested arrays in the expression parser. You can now input nested arrays like in JavaScript. Matrices can be concatenated using the function `concat`.
- The matrix syntax `[...]` in the expression parser now creates 1 dimensional matrices by default. `math.eval('[1,2,3,4]')` returns a matrix with size `[4]`, `math.eval('[1,2;3,4]')` returns a matrix with size `[2,2]`.
- Documentation is restructured and extended.
- Fixed non working operator `mod` (modulus operator).

2013-09-03, version 0.13.0

- Implemented support for booleans in all relevant functions.
- Implemented functions `map` and `forEach`. Thanks Sebastien Piquemal (@sebpic).
- All construction functions can be used to convert the type of variables, also element-wise for all elements in an Array or Matrix.
- Changed matrix indexes of the expression parser to one-based with the upper-bound included, similar to most math applications. Note that on a JavaScript level, math.js uses zero-based indexes with excluded upper-bound.
- Removed support for scalars in the function `subset`, it now only supports Array, Matrix, and String.
- Removed the functions `get` and `set` from a selector, they are a duplicate of the function `subset`.
- Replaced functions `get` and `set` of `Matrix` with a single function `subset`.
- Some moving around with code and namespaces:
    - Renamed namespace `math.expr` to `math.expression` (contains Scope, Parser, node objects).
    - Renamed namespace `math.docs` to `math.expression.docs`.

    - Moved `math.expr.Selector` to `math.chaining.Selector`.
- Fixed some edge cases in functions `lcm` and `xgcd`.

2013-08-22, version 0.12.1

- Fixed outdated version of README.md.
- Fixed a broken unit test.

2013-08-22, version 0.12.0

- Implemented functions `random([min, max])`, `randomInt([min, max])`, `pickRandom(array)`. Thanks Sebastien Piquemal (@sebpic).
- Implemented function `distribution(name)`, generating a distribution object with functions `random`, `randomInt`, `pickRandom` for different distributions. Currently supporting `uniform` and `normal`.
- Changed the behavior of `range` to exclude the upper bound, so `range(1, 4)` now returns `[1, 2, 3]` instead of `[1, 2, 3, 4]`.
- Changed the syntax of `range`, which is now `range(start, end [, step])` instead of `range(start, [step, ] end)`.
- Changed the behavior of `ones` and `zeros` to geometric dimensions, for example `ones(3)` returns a vector with length 3, filled with ones, and `ones(3,3)` returns a 2D array with size [3, 3].
- Changed the return type of `ones` and `zeros`: they now return an Array when arguments are Numbers or an Array, and returns a Matrix when the argument is a Matrix.
- Change matrix index notation in parser from round brackets to square brackets, for example `A[0, 0:3]`.
- Removed the feature introduced in v0.10.0 to automatically convert a complex value with an imaginary part equal to zero to a number.
- Fixed zeros being formatted as null. Thanks @TimKraft.

2013-07-23, version 0.11.1

- Fixed missing development dependency

2013-07-23, version 0.11.0

- Changed math.js from one-based to zero-based indexes.
    - Getting and setting matrix subset is now zero-based.
    - The dimension argument in function `concat` is now zero-based.
- Improvements in the string output of function help.
- Added constants `true` and `false`.
- Added constructor function `boolean`.
- Fixed function `select` not accepting `0` as input. Thanks Elijah Manor (@elijahmanor).
- Parser now supports multiple unary minus operators after each other.
- Fixed not accepting empty matrices like `[[], []]`.
- Some fixes in the end user documentation.

2013-07-08, version 0.10.0

- For complex calculations, all functions now automatically replace results having an imaginary part of zero with a Number. (`2i * 2i` now returns a Number `-4` instead of a Complex `-4 + 0i`).
- Implemented support for injecting custom node handlers in the parser. Can be used for example to implement a node handler for plotting a graph.
- Implemented end user documentation and a new `help` function.
- Functions `size` and `squeeze` now return a Matrix instead of an Array as output on Matrix input.
- Added a constant tau (2 * pi). Thanks Zak Zibrat (@palimpsests).
- Renamed function `unaryminus` to `unary`.
- Fixed a bug in determining node dependencies in function assignments.

2013-06-14, version 0.9.1

- Implemented element-wise functions and operators: `emultiply` (`x .* y`), `edivide` (`x ./ y`), `epow` (`x .^ y`).
- Added constants `Infinity` and `NaN`.
- Removed support for Workspace to keep the library focused on its core task.
- Fixed a bug in the Complex constructor, not accepting NaN values.
- Fixed division by zero in case of pure complex values.
- Fixed a bug in function multiply multiplying a pure complex value with Infinity.

2013-05-29, version 0.9.0

- Implemented function `math.parse(expr [,scope])`. Optional parameter scope can be a plain JavaScript Object containing variables.
- Extended function `math.expr(expr [, scope])` with an additional parameter `scope`, similar to `parse`. Example: `math.eval('x^a', {x:3, a:2});`.
- Implemented function `subset`, to get or set a subset from a matrix, string, or other data types.
- Implemented construction functions number and string (mainly useful inside the parser).
- Improved function `det`. Thanks Bryan Cuccioli (@bcuccioli).
- Moved the parse code from prototype math.expr.Parser to function math.parse, simplified Parser a little bit.
- Strongly simplified the code of Scope and Workspace.
- Fixed function mod for negative numerators, and added error messages in case of wrong input.

2013-05-18, version 0.8.2

- Extended the import function and some other minor improvements.
- Fixed a bug in merging one dimensional vectors into a matrix.
- Fixed a bug in function subtract, when subtracting a complex number from a real number.

2013-05-10, version 0.8.1

- Fixed an npm warning when installing mathjs globally.

2013-05-10, version 0.8.0

- Implemented a command line interface. When math.js is installed globally via npm, the application is available on your system as 'mathjs'.
- Implemented `end` keyword for index operator, and added support for implicit start and end (expressions like `a(2,:)` and `b(2:end,3:end-1)` are supported now).
- Function math.eval is more flexible now: it supports variables and multi-line expressions.
- Removed the read-only option from Parser and Scope.
- Fixed non-working unequal operator != in the parser.
- Fixed a bug in resizing matrices when replacing a subset.
- Fixed a bug in updating a subset of a non-existing variable.
- Minor bug fixes.

2013-05-04, version 0.7.2

- Fixed method unequal, which was checking for equality instead of inequality. Thanks @FJS2.

2013-04-27, version 0.7.1

- Improvements in the parser:
    - Added support for chained arguments.
    - Added support for chained variable assignments.
    - Added a function remove(name) to remove a variable from the parsers scope.
    - Renamed nodes for more consistency and to resolve naming conflicts.
    - Improved stringification of an expression tree.
    - Some simplifications in the code.
    - Minor bug fixes.
- Fixed a bug in the parser, returning NaN instead of throwing an error for a number with multiple decimal separators like `2.3.4`.
- Fixed a bug in Workspace.insertAfter.
- Fixed: math.js now works on IE 6-8 too.

2013-04-20, version 0.7.0

- Implemented method `math.eval`, which uses a readonly parser to evaluate expressions.
- Implemented method `xgcd` (extended eucledian algorithm). Thanks Bart Kiers (@bkiers).
- Improved math.format, which now rounds values to a maximum number of digits instead of decimals (default is 5 digits, for example `math.format(math.pi)` returns `3.1416`).
- Added examples.
- Changed methods square and cube to evaluate matrices element wise (consistent with all other methods).
- Changed second parameter of method import to an object with options.
- Fixed method math.typeof on IE.
- Minor bug fixes and improvements.

2013-04-13, version 0.6.0

- Implemented chained operations via method math.select(). For example `math.select(3).add(4).subtract(2).done()` will return `5`.

- Implemented methods gcd and lcm.
- Implemented method `Unit.in(unit)`, which creates a clone of the unit with a fixed representation. For example `math.unit('5.08 cm').in('inch')` will return a unit which string representation always is in inch, thus `2 inch`. `Unit.in(unit)` is the same as method `math.in(x, unit)`.
- Implemented `Unit.toNumber(unit)`, which returns the value of the unit when represented with given unit. For example `math.unit('5.08 cm').toNumber('inch')` returns the number `2`, as the representation of the unit in inches has 2 as value.
- Improved: method `math.in(x, unit)` now supports a string as second parameter, for example `math.in(math.unit('5.08 cm'), 'inch')`.
- Split the end user documentation of the parser functions from the source files.
- Removed function help and the built-in documentation from the core library.
- Fixed constant i being defined as -1i instead of 1i.
- Minor bug fixes.

2013-04-06, version 0.5.0

- Implemented data types Matrix and Range.
- Implemented matrix methods clone, concat, det, diag, eye, inv, ones, size, squeeze, transpose, zeros.
- Implemented range operator `:`, and transpose operator `'` in parser.
- Changed: created construction methods for easy object creation for all data types and for the parser. For example, a complex value is now created with `math.complex(2, 3)` instead of `new math.Complex(2, 3)`, and a parser is now created with `math.parser()` instead of `new math.parser.Parser()`.
- Changed: moved all data types under the namespace math.type, and moved the Parser, Workspace, etc. under the namespace math.expr.
- Changed: changed operator precedence of the power operator:
    - it is now right associative instead of left associative like most scripting languages. So `2^3^4` is now calculated as `2^(3^4)`.
    - it has now higher precedence than unary minus most languages, thus `-3^2` is now calculated as `-(3^2)`.
- Changed: renamed the parsers method 'put' into 'set'.
- Fixed: method 'in' did not check for units to have the same base.

2013-03-16, version 0.4.0

- Implemented Array support for all methods.
- Implemented Array support in the Parser.
- Implemented method format.
- Implemented parser for units, math.Unit.parse(str).
- Improved parser for complex values math.Complex.parse(str);
- Improved method help: it now evaluates the examples.
- Fixed: a scoping issue with the Parser when defining functions.
- Fixed: method 'typeof' was not working well with minified and mangled code.
- Fixed: errors in determining the best prefix for a unit.

2013-03-09, version 0.3.0

- Implemented Workspace
- Implemented methods cot, csc, sec.
- Implemented Array support for methods with one parameter.

2013-02-25, version 0.2.0

- Parser, Scope, and expression tree with Nodes implemented.
- Implemented method import which makes it easy to extend math.js.
- Implemented methods arg, conj, cube, equal, factorial, im, largereq, log(x, base), log10, mod, re, sign, smallereq, square, unequal.

2013-02-18, version 0.1.0

- Reached full compatibility with Javascripts built-in Math library.
- More functions implemented.
- Some bugfixes.

2013-02-16, version 0.0.2

- All constants of Math implemented, plus the imaginary unit i.
- Data types Complex and Unit implemented.
- First set of functions implemented.

2013-02-15, version 0.0.1

- First publish of the mathjs package. (package is still empty)