

# *JavaScript Complex Library*

## *(Complex Numbers)*

*Version 3.0*

*By Henrik Vestermark ([hve@hvks.com](mailto:hve@hvks.com))*

## Contents

Introduction.....	4
Design Rationale and Numerical Considerations .....	4
Numeric Model and Precision .....	4
Branch Cuts and Mathematical Conventions.....	5
Error Handling and Exceptional Values .....	5
Object Semantics and Mutability.....	6
Real and Complex Return Types .....	6
Constructor and Conversion Model.....	6
Performance Considerations .....	7
Complex.....	8
Method .....	8
Functions.....	9
Constants.....	9
Chaining.....	10
Miscellaneous .....	10
API.....	10
Complex.abs().....	11
Complex.arg().....	11
Complex.add() .....	11
Complex.acos() .....	12
Complex.acosh().....	13
Complex.asin().....	13
Complex.asinh().....	14
Complex.atan().....	14
Complex.atanh().....	15
Complex.beta().....	15
Complex.cos().....	16
Complex.cosh().....	16
Complex.conj() .....	17
Complex.div().....	17
Complex.equal().....	18
Complex.erf().....	18
Complex.erfc().....	19

Complex.exp()	20
Complex.i	20
Complex.imag()	21
Complex.infinity	21
Complex.lgamma()	22
Complex.log()	22
Complex.log10()	23
Complex.minusone	23
Complex.mul()	24
Complex.negate()	25
Complex.notequal()	25
Complex.norm()	26
Complex.one	26
Complex.polar()	27
Complex.pow()	27
Complex.proj()	28
Complex.real()	28
Complex.sin()	29
Complex.sinh()	30
Complex.sub()	30
Complex.sqrt()	31
Complex.tan()	31
Complex.tanh()	32
Complex.tgamma()	32
Complex.toExponential()	33
Complex.toFixed()	33
Complex.toPrecision()	34
Complex.toString()	35
Complex.valueOf()	36
Complex.zero	36
parseComplex()	37

## Introduction

This JavaScript Complex library is used as a foundational numeric component in several web-based applications that require reliable complex arithmetic. Typical use cases include interactive mathematical tools, engineering calculators, and numerical analysis utilities executed directly in the browser.

A primary example is a web-based polynomial root finder(see <https://hvks.com/Numerical/websolver.php>), where complex arithmetic is required throughout the computation, even when the final roots are real. In such applications, intermediate values may span wide ranges in magnitude and phase, and numerical robustness is essential to avoid divergence, loss of precision, or spurious roots.

The library provides a complete set of complex operations, including arithmetic, transcendental functions, and special functions such as the gamma, beta, and error functions. All functions are designed to work consistently for real and complex inputs, allowing algorithms to be implemented without special-case branching between real and complex domains.

Particular attention has been paid to numerical stability:

- Algorithms are structured to reduce cancellation and overflow where possible.
- Argument reduction is applied to transcendental and special functions.
- Functions that are undefined or unstable at singularities follow well-defined mathematical conventions.
- For real inputs, functions often return real results directly, avoiding unnecessary complex noise.

The library is intended for numerical correctness first, rather than symbolic manipulation or graphical presentation. It integrates cleanly with standard JavaScript environments and, where appropriate, extends the Math namespace with compatible real-valued wrappers.

This documentation describes the public API, mathematical behavior, and usage patterns of the Complex library as used in production web applications.

## Design Rationale and Numerical Considerations

### Numeric Model and Precision

All computations in this library are performed using JavaScript Number values, which follow the IEEE-754 double-precision floating-point standard.

The library does not attempt to provide arbitrary-precision complex arithmetic. Instead, it focuses on numerically stable algorithms that produce reliable results within the limits of double precision. This design choice reflects the intended use of the library in browser-

based numerical applications, where performance, predictability, and integration with existing JavaScript code are critical.

In particular:

- Argument reduction is applied where appropriate to improve numerical stability.
- Algorithms are structured to reduce cancellation and overflow.
- Special functions are implemented with care around critical regions and singularities.
- The behavior of functions is consistent across real and complex domains.

Users requiring arbitrary-precision complex arithmetic should use a dedicated big-number complex library. This library is optimized for speed and stability in standard JavaScript environments.

### Branch Cuts and Mathematical Conventions

Several complex functions are mathematically multi-valued. This library follows standard principal-value conventions to ensure predictable and consistent results.

The following conventions are used throughout:

- $\arg(z)$  returns values in the range  $(-\pi, \pi]$ .
- $\log(z)$  uses the principal branch with a branch cut along the negative real axis.
- $\text{sqrt}(z)$  returns the principal square root.
- Functions derived from the log function, including  $\text{pow}$ ,  $\text{tgamma}$ , and  $\text{lgamma}$ , inherit these conventions.

These choices are applied consistently across the library and are particularly important for iterative numerical algorithms, where discontinuities or inconsistent branch selection can prevent convergence.

### Error Handling and Exceptional Values

The library does not throw exceptions for mathematical domain errors or singularities.

Instead, it follows IEEE-754 semantics:

- Invalid operations result in NaN.
- Overflow results in Infinity.
- Functions evaluated at poles or undefined points return well-defined IEEE values where possible.

This behavior is intentional and supports numerical algorithms that rely on value propagation rather than control-flow interruption. It allows invalid or divergent values to be detected and handled naturally within iterative schemes.

## Object Semantics and Mutability

Complex numbers in this library are immutable with respect to arithmetic operations.

Functions such as `add`, `sub`, `mul`, `div`, and all transcendental and special functions return new `Complex` instances and do not modify their input arguments. This avoids unintended side effects and simplifies reasoning about numerical code.

Getter and setter methods such as `real()` and `imag()` may be used explicitly to modify the components of an existing `Complex` instance when required.

## Real and Complex Return Types

When a function is applied to purely real inputs and the mathematical result is real, the library may return a real JavaScript Number value.

When invoked through the `Complex` namespace, results are returned as `Complex` objects. Additionally, when the environment does not provide certain real-valued functions (such as `Math.tgamma`, `Math.erf`, or `Math.erfc`), the library defines compatible wrappers that delegate to the `Complex` implementation and return the real part of the result.

This dual behavior avoids unnecessary complex wrapping in real-only computations while preserving correctness and consistency for complex inputs.

## Constructor and Conversion Model

The `Complex` type is implemented as a plain JavaScript function rather than using the ECMAScript class syntax. This design choice is deliberate and motivated by functional and numerical considerations, not by compatibility constraints.

In JavaScript, a function can be invoked both:

- as a constructor using `new`, and
- as a regular function performing type conversion.

This allows the `Complex` implementation to support both of the following forms:

```
var z1 = new Complex(3, 4); // explicit construction
var z2 = Complex(5);      // real-to-complex conversion
```

This dual-use pattern closely mirrors the C-style and mathematical type-conversion semantics, where a numeric value can be promoted to a complex value without explicit constructor syntax.

The ECMAScript class construct does not allow this behavior. A class constructor must be invoked with `new`, and calling it as a function is a runtime error. As a result, it is not possible to implement both construction and conversion semantics using a class without introducing additional factory functions or special method names.

By using a function-based implementation, the library provides:

- A natural and compact conversion syntax for real-to-complex promotion.
- A uniform calling convention across all numeric functions.
- Reduced syntactic noise in numerical algorithms, where frequent conversion is required.

Internally, the function detects whether it is invoked with `new` and initializes the object accordingly. When invoked without `new`, it performs a safe conversion and returns a `Complex` instance.

This approach simplifies numerical code, improves readability in mathematical expressions, and aligns with the library's emphasis on practical numerical computation rather than object-oriented formality.

### Performance Considerations

The library is designed for interactive and computational use in browser environments.

Emphasis is placed on:

- Predictable performance
- Low allocation overhead
- Suitability for iterative numerical methods

The library is not intended to replace low-level numerical libraries written in C or C++, but rather to provide a reliable, self-contained complex arithmetic layer for web-based applications.

## Complex

---

Support for complex number arithmetic in JavaScript

### Constructor

```
new Complex(real,imaginary)    // Constructor  
Complex(value)                // Conversion
```

### Arguments

*real*            The real part of a complex number  
*imaginary*      The imaginary part of a complex number

If the imaginary argument is omitted, it is treated as zero  
If there are no arguments, it is treated as a complex zero  
If *Complex* is invoked as a conversion, the value parameter is converted to a *complex number* and returned.

### Returns

Returns a Complex object initialized with the real and imaginary values. If *Complex* is invoked as a conversion, the *value* parameter is converted to a *complex number* and returned. If *value* is another Complex number, that is returned; if *value* is undefined, *Complex.zero* is returned.

### Method

*abs()*            Return the magnitude of a complex number  
*arg()*            Return the angle of the polar representation of the Complex number.  
*conj()*            Return a new Conjugated Complex object.  
*imag()*            Return or set the Imaginary part of the Complex number.  
*negate()*          Return a new Negated Complex Object.  
*norm()*            Return the norm of the complex number.  
*real()*            Return or set the real part of the Complex number.  
*toExponential()*    Converts a Complex number to a string using exponential notation with the specified number of digits after the Decimal place.  
*toFixed()*          Converts a Complex number to a string that contains a specified number of digits after the decimal place.  
*toPrecision()*    Convert a Complex number to a string using the specified number of precision digits. Uses exponential or fixed point notation depending on the size of the number and the number of significant digits specified.  
*toString()*        Convert a Complex number to a string using a specified radix(base)  
*valueOf()*         The primitive real number value of this Complex number object.

## Functions

<code>abs()</code>	Return the magnitude of a complex number
<code>add()</code>	Return the addition of two complex numbers
<code>acos()</code>	Return arc cosine of the complex number
<code>acosh()</code>	Return the arc cosine hyperbolic of the complex number
<code>asin()</code>	Return the arcsine of the complex number
<code>asinh()</code>	Return the arc sine hyperbolic of the complex number
<code>atan()</code>	Return the arc tangent of the complex number
<code>atanh()</code>	Return the arctangent hyperbolic of the complex number
<code>beta()</code>	Return the beta function of a complex number
<code>cos()</code>	Return cosine of the complex number
<code>cosh()</code>	Return the cosine hyperbolic of the complex number
<code>div()</code>	Return the division of two complex numbers.
<code>equal()</code>	Return the Boolean value (true, false) of the equality of two complex numbers.
<code>exp()</code>	Return the complex power of e.
<code>lgamma()</code>	Return the logarithm of the gamma function of a complex number
<code>log()</code>	Return the complex natural logarithm.
<code>log10()</code>	Return the complex base 10 logarithm.
<code>mul()</code>	Return the product of two complex numbers.
<code>notequal()</code>	Return the Boolean value (true, false) of the inequality of two complex numbers.
<code>polar()</code>	Return the complex number of the polar representation.
<code>pow()</code>	Return the complex power of $x^y$ .
<code>proj()</code>	Return the projection of the complex number onto the Riemann sphere.
<code>sin()</code>	Return the sine of the complex number
<code>sinh()</code>	Return the sine hyperbolically of the complex number
<code>sub()</code>	Return the difference between two complex numbers.
<code>sqrt()</code>	Return the complex square root.
<code>tan()</code>	Return the tangent of the complex number
<code>tanh()</code>	Return the tangent hyperbolic of the complex number
<code>tgamma()</code>	Return the gamma function of a complex number

## Constants

<code>Complex.zero</code>	Return a new <code>Complex(0,0)</code> object
<code>Complex.one</code>	Return a new <code>Complex(1,0)</code> object
<code>Complex.minusone</code>	Return a new <code>Complex(-1,0)</code> object
<code>Complex.infinity</code>	Return a new <code>Complex(Number.Infinity,0)</code> object
<code>Complex.i</code>	Return a new <code>Complex(0,1)</code> object

## Chaining

`add(b)`, `sub(b)`, `mul(b)`, `div(b)` also have convenience instance methods that forward to `Complex.add/sub/mul/div` and return a new `Complex` result.

## Miscellaneous

`parseComplex()`      Parse a Complex float number string

## API

## Complex.abs()

---

Return the absolute magnitude of the Complex number.

### Synopsis

```
complex object.abs()
```

### Returns

The magnitude of the complex number is returned. Special calculations are made to prevent intermediate results from overflowing.

### Example

```
var z = new Complex(3,4);  
z.abs()           // result 5
```

### See Also

Complex.norm()

## Complex.arg()

---

Return the angle of the polar representation of the complex number.

### Synopsis

```
complex object.arg()
```

### Returns

Return the angle in radians of the polar representation of the complex number.

### Example

```
var z = new Complex(3,4);  
z.arg()           // result 0.927295...
```

### See Also

Complex.add()

---

Add two complex numbers.

### Synopsis

```
Complex.add(a,b)
```

### Arguments

*a,b*            The complex numbers to be added.

### Returns

The result of the complex addition.

### Example

```
var z = new Complex(3,4);  
var y=new Complex(1,2);
```

```
Complex.add(z,y)            // result (4+i6)
```

### See Also

Complex.div(), Complex.mul(), Complex.sub()

Complex.acos()

---

Return the arc cosine of the complex number.

### Synopsis

```
Complex.acos(a)
```

### Returns

Return the arc cosine of the complex number.

### Example

```
var z = new Complex(3,4);  
Complex.acos(z)            // result (0.9368124611557193-i2.3055090312434685)
```

### See Also

Complex.asin(), Complex.atan()

## Complex.acosh()

---

Return the arc cosine hyperbolic of the complex number.

### Synopsis

```
Complex.acosh(a)
```

### Returns

Return the arc cosine hyperbolic of the complex number.

### Example

```
var z = new Complex(3,4);  
Complex.acosh(z) // result (2.305509031243477+i0.9368124611557199)
```

### See Also

Complex.asinh(), Complex.atanh()

## Complex.asin()

---

Return the arcsine of the complex number.

### Synopsis

```
Complex.asin(a)
```

### Returns

Return the arc sine of the complex number.

### Example

```
var z = new Complex(3,4);  
Complex.asin(z) // result (0.6339838656391773+i2.3055090312434685)
```

### See Also

Complex.acos(), Complex.atan()

## Complex.asinh()

---

Return the arc sine hyperbolic of the complex number.

### Synopsis

```
Complex.asinh(a)
```

### Returns

Return the arc sine hyperbolic of the complex number.

### Example

```
var z = new Complex(3,4);  
Complex.asinh(z) // result (2.2999140408792695+i0.9176168533514787)
```

### See Also

Complex.acosh(), Complex.atanh()

## Complex.atan()

---

Return the arctangent of the complex number.

### Synopsis

```
Complex.atan(a)
```

### Returns

Return the arc tangent of the complex number.

### Example

```
var z = new Complex(3,4);  
Complex.atan(z) // result (1.4483069952314644+i0.15899719167999904)
```

### See Also

Complex.acos(), Complex.asin()

## Complex.atanh()

---

Return the arc tangent hyperbolic of the complex number.

### Synopsis

Complex.atanh(a)

### Returns

Return the arc tanh hyperbolic of the complex number.

### Example

```
var z = new Complex(3,4);  
Complex.atanh(z)           // result (0.11750090731143381+i1.4099210495965755)
```

### See Also

Complex.acosh(), Complex.asinh()

## Complex.beta()

---

Return the beta function of the complex numbers.

### Synopsis

Complex.beta(x,y)

### Returns

Return the beta function of the complex numbers a and b.

### Example

```
var x=new Complex(3,4), y=Complex(2,-3);  
Complex.beta(x,y)           // result (0.000594954...,0.000791355...)
```

### See Also

Complex.tgamma(), Complex.lgamma()

## Complex.cos()

---

Return the cosine of the complex number.

### Synopsis

Complex.cos(a)

### Returns

Return the cosine of the complex number.

### Example

```
var z = new Complex(3,4);  
Complex.cos(z)           // result (-27.034945603074224-i3.851153334811777)
```

### See Also

Complex.sin(), Complex.tan()

## Complex.cosh()

---

Return the hyperbolic cosine of the complex number

### Synopsis

Complex.cosh(a)

### Returns

Return the hyperbolic cosine of the complex number.

### Example

```
var z = new Complex(3,4);  
Complex.cosh(z)          // result (-6.580663040551157-i7.581552742746545)
```

### See Also

Complex.sinh(), Complex.tanh()

## Complex.conj()

---

Return the conjugate form of the complex number.

### Synopsis

*complex object*.conj()

### Returns

Return the complex conjugate form.

### Example

```
var z = new Complex(3,4);  
z.conj()           // result (3-i4)
```

### See Also

Complex.negate()

## Complex.div()

---

Divide two complex numbers.

### Synopsis

Complex.div(a,b)

### Arguments

*a,b*            The complex numbers to be divided. Special calculations are made to prevent intermediate results from overflowing.

### Returns

The result of the complex division  $a/b$ .

### Example

```
var z = new Complex(3,4);  
var y=new Complex(1,2);  
  
Complex.div(z,y)           // result (2.2+i0.4)
```

**See Also**

Complex.add(), Complex.mul(), Complex.sub()

Complex.equal()

---

Compare two complex numbers for equality.

**Synopsis**

Complex.equal(a,b)

**Arguments**

*a,b*            The complex numbers to be compare for

**Returns**

The Boolean value of the equal comparison.

**Example**

```
var z = new Complex(3,4);
var y=new Complex(1,2);

if( Complex.equal(z,y))...           // result false
if( Complex.equal(z,z))...           // result true
if( !Complex.equal(z,y))...         // result true ! to do "notequal" comparison
```

**See Also**

Complex.notequal()

Complex.erf()

---

Return the error function of a real or complex number.

**Synopsis**

Complex.erf(z)

**Arguments**

*z* A complex or real value.

**Returns**

Returns  $\text{erf}(z)$ . For purely real inputs, the result is typically a purely real Complex number (value + i0).

**Example**

```
Complex.erf(0.5)           // (0.5204998778 + i0)
Complex.erf(new Complex(1, 1)) // (... + i ...)
```

**Notes**

If the JavaScript runtime does not provide `Math.erf`, the library defines `Math.erf(x)` as a wrapper that returns the real part of `Complex.erf(x)`.

**See Also**

`Complex.erfc()`

`Complex.erfc()`

---

Return the complementary error function of a real or complex number.

**Synopsis**

`Complex.erfc(z)`

**Arguments**

*z* A complex or real value.

**Returns**

Returns  $\text{erfc}(z) = 1 - \text{erf}(z)$ .

**Example**

```
Complex.erfc(0.5)           // (0.4795001222 + i0)
Complex.erfc(new Complex(1, 1)) // (... + i ...)
```

**Notes**

If the JavaScript runtime does not provide `Math.erfc`, the library defines `Math.erfc(x)` as a wrapper that returns the real part of `Complex.erfc(x)`.

**See Also**

Complex.erf()

Complex.exp()

---

Compute  $e^x$

**Synopsis**

Complex.exp(x)

**Arguments**

$x$             A complex number to be used as the exponent

**Returns**

$e^x$ ,  $e$  raised to the power of the specified exponent  $x$ , where  $e$  is the base of the natural logarithm, with a value of approximately 2.71828.

**Example**

```
var z = new Complex(3,4);  
Complex.exp(z)...            //  $e^z$  approximately (-13.128+i5.200)
```

**See Also**

Complex.log(), Complex.log10(), Complex.pow()

Complex.i

---

Return complex  $i$

**Synopsis**

Complex.i

**Returns**

The complex constant  $i$  ( $0+i1$ ).

**Example**

```
var z = Complex.i;           // z=(0+i1)
```

**See Also**

Complex.one, Complex.zero, Complex.minusone, Complex.infinity

Complex.imag()

---

Return or Set the imaginary part of the complex number.

**Synopsis**

*complex object*.imag(*i*)

**Arguments**

*i*      The optional imaginary value when setting the imaginary number to a new value. If omitted, the call returns the actual imaginary value of the complex number.

**Returns**

Return the imaginary part of the complex number.

**Example**

```
var z = new Complex(3,4);  
z.imag();           // result 4  
z.imag(5);         // set z to (3+i5) and return 5
```

**See Also**

Complex.real()

Complex.infinity

---

Return complex infinity

**Synopsis**

Complex.infinity

**Returns**

The complex constant infinity (Infinity+i0).

**Example**

```
var z = Complex.infinity;           // z=(Number.Infinity+i0)
```

**See Also**

Complex.zero, Complex.i, Complex.one, Complex.minusone

Complex.lgamma()

---

Return the logarithm of the gamma function for the complex number.

**Synopsis**

Complex.lgamma(x)

**Returns**

Return the logarithm gamma function of the complex number x.

**Example**

```
var x=new Complex(3,4);  
Complex.lgamma(x)           // result (-1.756626...,4.742664...)
```

**See Also**

Complex.tgamma(), Complex.beta()

Complex.log()

---

Compute the natural logarithm of x

**Synopsis**

Complex.log(x)

**Arguments**

*x*            A complex number not equal to zero

**Returns**

Return  $\log(x)$

**Example**

```
var z = new Complex(3,4);
```

```
Complex.log(z)...           // log(x) approximately (1.609+i0.927)
```

**See Also**

`Complex.exp()`, `Complex.log10()`,

`Complex.log10()`

---

Compute the base-10 logarithm of  $x$

**Synopsis**

```
Complex.log10(x)
```

**Arguments**

$x$             A complex number not equal to zero

**Returns**

Return  $\log_{10}(x)$

**Example**

```
var z = new Complex(3,4);
```

```
Complex.log10(z)...         // log10(3+i4) approximately (0.699+i0.403)
```

**See Also**

`Complex.exp()`, `Complex.log()`,

`Complex.minusone`

---

Return complex negative one.

**Synopsis**

Complex.minusone

**Returns**

The complex constant negative one  $(-1+i0)$ .

**Example**

```
var z = Complex.minusone;      // z=(-1+i0)
```

**See Also**

Complex.zero, Complex.i, Complex.one, Complex.infinity

Complex.mul()

---

Multiply two complex numbers.

**Synopsis**

Complex.mul(a,b)

**Arguments**

*a,b*            The complex numbers to be multiplied.

**Returns**

The result of the complex multiplication.

**Example**

```
var z = new Complex(3,4);  
var y=new Complex(1,2);
```

```
Complex.mul(z,y)      // result (-5+i10)
```

**See Also**

Complex.add(), Complex.div(), Complex.sub()

## Complex.negate()

---

Return the negated complex number.

### Synopsis

*complex object*.negate()

### Returns

Return the negated complex number.

### Example

```
var z = new Complex(3,4);  
z.negate()                // result (-3-i4)
```

### See Also

Complex.conj()

Complex.notequal()

---

Compare two complex numbers for inequality.

### Synopsis

Complex.notequal(a,b)

### Arguments

*a,b*            The complex numbers to be compared for

### Returns

The Boolean value of the equal comparison.

### Example

```
var z = new Complex(3,4);  
var y=new Complex(1,2);  
  
if( Complex.notequal(z,y))... // result true  
if( Complex.notequal(z,z))... // result false  
if( !Complex.notequal(z,y))... // result false ! to do "equal" comparison
```

**See Also**

Complex.notequal()

Complex.norm()

---

Return the norm (square magnitude) of the Complex number.

**Synopsis**

*complex object*.norm()

**Returns**

The norm (squared magnitude) of the complex number is returned.

**Example**

```
var z = new Complex(3,4);  
z.norm()           // result 25
```

**See Also**

Complex.abs()

Complex.one

---

Return complex one

**Synopsis**

Complex.one

**Returns**

The complex constant one (1+i0).

**Example**

```
var z = Complex.one;           // z=(1+i0)
```

**See Also**

Complex.zero, Complex.i, Complex.minusone, Complex.infinity

Complex.polar()

---

Convert polar coordinates into a complex number.

### Synopsis

Complex.polar(mag,arg)

### Arguments

*mag*            Magnitude of a complex number  
*arg*            Angle of a complex number.

### Returns

Return the complex number.

### Example

```
var z = Complex.polar( 4, 0.5 );  
  
Complex.polar(4,0.5)        // result (3.510+i1.918)
```

### See Also

Complex.pow()

---

Compute  $x^y$

### Synopsis

Complex.pow(x,y)

### Arguments

*x*            A complex number to be raised to a power  
*y*            A complex power that x is raised to

### Returns

X to the power of y.  $x^y$

**Example**

```
var x = new Complex(3,4);  
var y = new Complex( 1 , 2 )
```

```
Complex.pow(x,y)...      //  $x^y$  approximately (-0.4198+i0.6605)
```

**See Also**

Complex.exp()

Complex.proj()

---

Return the projection of the complex number onto the Riemann sphere.

**Synopsis**

Complex.proj(z)

**Arguments**

z A complex or real value.

**Returns**

For finite values, returns z.

If z has an infinite real or imaginary component, returns (Infinity + i·±0) with the sign of zero following the sign of the imaginary part.

**Example**

```
Complex.proj(new Complex(Infinity, -2)) // (Infinity - i0)  
Complex.proj(new Complex(3, 4))       // (3 + i4)
```

**See Also**

Complex.abs(), Complex.arg()

Complex.real()

---

Return or Set the real part of the complex number.

**Synopsis**

*complex object*.real(*r*)

### Arguments

*r* The optional real value when setting the real number to a new value. If omitted, the call returns the actual real value of the complex number.

### Returns

Return the real part of the complex number.

### Example

```
var z = new Complex(3,4);
z.real();           // result 3
z.real(5);         //set the real part to 5 and return the result 5
```

### See Also

Complex.imag()

Complex.sin()

---

Return the sine of the complex number.

### Synopsis

Complex.sin(*a*)

### Returns

Return the sine of the complex number.

### Example

```
var z = new Complex(3,4);
Complex.sin(z)      // result (3.853738037919377-i27.016813258003932)
```

### See Also

Complex.cos(), Complex.tan()

## Complex.sinh()

---

Return the hyperbolic sine of the complex number.

### Synopsis

```
Complex.sinh(a)
```

### Returns

Return the sine hyperbolic of the complex number.

### Example

```
var z = new Complex(3,4);  
Complex.sinh(z)           // result (-6.580663040551157-i7.581552742746545)
```

### See Also

Complex.cosh(), Complex.tanh()

## Complex.sub()

---

Subtract two complex numbers.

### Synopsis

```
Complex.sub(a,b)
```

### Arguments

*a,b*            The complex numbers to be subtracted.

### Returns

The result of the complex subtraction.

### Example

```
var z = new Complex(3,4);  
var y=new Complex(1,2);  
  
Complex.sub(z,y)           // result (2+i2)
```

### See Also

`Complex.add()`, `Complex.div()`, `Complex.mul()`

`Complex.sqrt()`

---

Compute a complex square root.

### Synopsis

`Complex.sqrt(x)`

### Arguments

*x*            A complex number to be square rooted. Special calculations are performed to prevent intermediate results from overflowing.

### Returns

The square root of *x*.

### Example

```
var z = new Complex(3,4);  
Complex.sqrt(z)...        // Result (2+i1)
```

### See Also

`Complex.tan()`

---

Return the tangent of the complex number.

### Synopsis

`Complex.tan(a)`

### Returns

Return the tangent of the complex number.

### Example

```
var z = new Complex(3,4);  
Complex.tan(z) // result (-0.00018734620462947842+i0.9993559873814732)
```

**See Also**

Complex.cos(), Complex.sin()

Complex.tanh()

---

Return the hyperbolic tangent of the complex number.

**Synopsis**

Complex.tanh(a)

**Returns**

Return the tanh hyperbolic of the complex number.

**Example**

```
var z = new Complex(3,4);  
Complex.tanh(z) // result (-6.580663040551157-i7.581552742746545)
```

**See Also**

Complex.cosh(), Complex.sinh()

Complex.tgamma()

---

Return the gamma function of the complex number. The function name was renamed from gamma() to tgamma(), which is more widely recognized as the gamma function.

**Synopsis**

Complex.tgamma(x)

**Returns**

Return the gamma function of the complex number x.

**Example**

```
var x=new Complex(3,4);  
Complex.tgamma(x) // result (0.00522553...,0.1725470...)
```

**See Also**

Complex.lgamma(), Complex.beta()

Complex.toExponential()

---

Format a number using exponential notation.

**Synopsis**

*complex*.toExponential(*digits*)

**Arguments**

*Digits*        The number of digits that will appear after the decimal point. This may be a value between 0 and 20, inclusive. If this argument is omitted, as many digits as necessary will be used. A complex number is always formatted as:

*(real\_part ±i imaginary\_part)*

**Returns**

A string representation of the complex number, in exponential notation, with one digit before the decimal place and *digits* digits after the decimal place. The fractional part of the complex number is rounded or padded with zeros as necessary so that it has the specified length.

**Example**

```
var z = new Complex( 12345.6789, 12345.6789 );
z.toExponential(1);        // result (1.2e+4+i1.2e+4)
z.toExponential(5);        // result (1.23457e+4+i1.23457e+4)
z.toExponential(10);       // result (1.23456789000e+4+i1.23456789000e+4)
z.toExponential();        // result (1.23456789e+4+i1.23456789e+4)
```

**See Also**

Complex.toFixed(), Complex.toPrecision(), Complex.toString()

Complex.toFixed()

---

Format a number using fixed-point notation.

## Synopsis

*complex*.toFixed(digits)

## Arguments

*Digits*        The number of digits that will appear after the decimal point. This may be a value between 0 and 20, inclusive. If this argument is omitted, it is treated as zero. A complex number is always formatted as:  
(*real\_part* ±i *imaginary\_part*)

## Returns

A string representation of the complex number that does not use exponential notation and has exactly *digits* digits after the decimal point. The *complex number* is rounded as necessary, and the fraction part is padded with zeros if required, so that it has the specified length. If the *complex number* is greater than 1e+21, this method calls *number.toString()* and returns a string in exponential notation.

## Example

```
var z = new Complex( 12345.6789, 12345.6789 );
z.toFixed(5);            // result (12345.7+i12345.7)
z.toFixed(6);            // result (12345.678900+i12345.678900)
z.toFixed();            // result (12346+i1234.6)
```

## See Also

Complex.toExponential(), Complex.toPrecision(), Complex.toString()

Complex.toPrecision()

---

Format the significant digits of a complex number.

## Synopsis

*complex*.toPrecision(digits)

## Arguments

*Digits*        The number of significant digits to appear in the returned string. This may be a value between 1 and 21, inclusive. If this argument is omitted, the *toString()* method is used instead to convert the complex number to a base-10 value. A complex number is always formatted as:

*(real\_part ±i imaginary\_part)*

### Returns

A string representation of the *complex number* that contains *precision* significant digits. If *precision* is large enough to include all the digits of the integer part of the number, the returned string uses fixed-point notation. Otherwise, exponential notation is used with one digit before the decimal place and *precision* – 1 digits after the decimal place. The number is rounded or padded with zeros as necessary.

### Example

```
var z = new Complex( 12345.6789, 12345.6789 );
z.toPrecision(1);           // result (1e+4+i1e+4)
z.toPrecision(3);          // result (1.23e+4+i1.2e+4)
z.toPrecision(5);          // result (12346+i12346)
```

### See Also

Complex.toExponential(), Complex.toFixed(), Complex.toString()

Complex.toString()

---

Format the significant digits of a complex number.

### Synopsis

*complex*.toString(radix)

### Arguments

*Radix*            If omitted, the base 10 will be used to convert the complex number to a string. Otherwise, the radix will be used (2..36). A complex number is always formatted as:

*(real\_part ±i imaginary\_part)*

### Returns

A string representation of the *complex number*, in the indicated radix.

### Example

```
var z = new Complex( 12345.6789, 12345.6789 );
z.toString();           // result (1234.6789+i1234.6789)
```

### See Also

`Complex.toExponential()`, `Complex.toFixed()`, `Complex.toPrecision()`

`Complex.valueOf()`

---

Return the primitive number value.

### Synopsis

*complex object*.`valueOf()`

### Returns

The primitive value of the *complex number* is returned, which is the same as the *complex number*.`real()`.

### Example

```
var z = new Complex(3,4);  
z.valueOf()           // return 3
```

### See Also

`Complex.zero`

---

Return complex zero.

### Synopsis

`Complex.zero`

### Returns

The complex constant zero (0+i0).

### Example

```
var z = Complex.zero;
```

### See Also

`Complex.one`, `Complex.i`, `Complex.minusone`, `Complex.infinity`

## parseComplex()

---

Convert a string to a complex number.

### Synopsis

*parseComplex(s)*

### Arguments

*s*            The string to be parsed and converted to a *complex number*.

### Returns

parseComplex() parses and returns a new Complex number contained in *s*.  
parseComplex() returns a Complex NaN number if parsing fails. A complex number can be in the format:

*(real\_part ±i imaginary\_part)*

Where either the *real\_part* or the *imaginary\_part* can be missing but not both at the same time. parseComplex() can also parse a string, omitting the leading and trailing parentheses.

### Example

```
var z = parseComplex(“(1.2 -i 3.4E-5)”); // result (1.2-i3.4E-5)
z = parseComplex(“(1.2)” );           // result (1.2 +i0)
z = parseComplex(“(-i1.2)” );         // result (0 -i1.2)
```

### See Also