Polynomial

Support for Polynomial objects in JavaScript. When used with the Complex JavaScript Library the Complex JavaScript library needs to be loaded first.

Constructor

new Polynomial(*coefficients…*) // Invoked as a Constructor
Polynomial(*coefficients…*) // Invoked as a Conversion

Arguments

coefficients Optional Polynomial Coefficients. *Coefficients* can be either JavaScript number, JavaScript Complex Object number, an array of numbers or another Polynomial objects.

Returns

Returns a normalized Polynomial object with an Array holding the coefficients. The coefficients in the Array can be normal JavaScript numbers or complex JavaScript numbers.
If *Polynomial* is invoked as a conversion the *coefficients* parameter is converted to a *Polynomial object* and returned.

If *coefficients* is undefined, an empty *Polynomial object* is returned.
Regardless if invoked as a new constructor or as a Conversion Polynomial constructor always return a Polynomial normalized object.

Example:
x=new Polynomial(1,2,3); // Return a new Polynomial object representing the Polynomial $1x^2+2x+3$
x=new Polynomial(Complex(1+1),2,Complex(3-4));
// Return a Polynomial object representing the polynomial $(1+i2)x^2+2x+(3-4i)$.
x=[1,2,3];
y=new Polynomial(x); // Return a new Polynomial object representing the Polynomial $1x^2+2x+3$

y=new Polynomial(x,25); // Return a new Polynomial object representing the Polynomial $1x^3+2x^3+3x+25$
y=new Polynomial(Complex(3-3),y);
// Returns a new Polynomial object representing the Polynomial $(3-3i)x^4$+1x3+2x3+3x+25
x=new Polynomial(1,,3); // Return a new Polynomial object representing the Polynomial $1x^2+3$

Normalized Polynomial

The Polynomial is normalized by eliminating leading zero coefficients and converting undefined coefficients to 0.

Notice that all properties or methods can work on both regular real coefficients or complex number coefficients or a mix of real and complex number coefficients. E.g.

Properties

Properties	**Description**
array()	Return a Polynomial object as an array, where each array element is the coefficients of the polynomial in descending order of power
degree()	Returns the degree of the Polynomial
getcoeff(*x^{th}*)	Return the coefficients belonging to x^{th} power of the Polynomial
isComplex()	Return true if at least one coefficient in the Polynomial is a complex number
isReal()	Return true if all the coefficients in the Polynomial are only JavaScript numbers.
join(*seperator*)	Concatenate polynomial coefficients to form a string that is returned. If a *separator* is specified, each element is separated by the *separator*
monic()	Bring the polynomial into a monic form in which the leading coefficients is 1 by dividing the leading coefficient with all the other coefficients
normalize()	The Polynomial is normalized by eliminating trailing zero coefficients and converting undefined coefficients to 0. The normalized polynomial is returned
scale()	Scale the polynomial by multiplying all the coefficients with a factor. The factor can be automatic calculated.
setcoeff(x^{th},*newcoeff*)	Set the coefficient belonging to the x^{th} power of the polynomial to the *newcoeff* value
shift(no)	Do a Polynomial Taylor shift of no
simplify()	The Polynomial is simplified by reducing complex numbers with an imaginary part of zero to a real number
toExponential(*digits*)	Return a string representation of the Polynomial using exponential notation for the coefficients and with the specified number of *digits*. Notice the *digits* is optional
toFixed(*digits*)	Return a string representation of the Polynomial where the coefficients contains a specified number of *digits* after the decimal place. Notice the *digits* is optional
toPrecision(*digits*)	Return a string representation of the Polynomial where the coefficients contains either exponential or fixed point notation depending on the size of the number and the number of significant *digits* specified. Notice the digits is optional
toString()	Return a string representation of the Polynomial object
valueof()	The primitive value number of this Polynomial object.

Methods

Methods	Descriptions
add(a,b)	Add two Polynomial object together

compositedeflate(z)	Deflate a Polynomial with the root z using composite deflation
deflate(z)	Deflate a Polynomial with the root z
derivative()	Return the derivative polynomial
div(a,b)	Return the division of two polynomial
mul(a,b)	Return the multiplication of two polynomial
pow(p,n)	Return the power of raising the Polynomial to n
rem(a,b)	Return the remainder polynomial after dividing the polynomial a/b
sub(a,b)	Returned the subtracted polynomial a-b
value(z)	Returned the value of the Polynomial at point z
zeros()	Find all zeros of a Polynomial

Constants

zero return a new Polynomial() object
one return a new Polynomial(1) object

Miscellaneous

parsePolynomialt() Parse a Polynomial string and return a Polynomial object

Polynomial.array()

Return the Polynomial object coefficients as an Array of coefficients

Synopsis

Polynomial object.array()

Returns

The coefficients of the Polynomial object is returned in the return array

Example

```
var p = new Polynomial( 1,2,3 );        // x^2+2x+3
var coeff;

var coeff=p.array()                      // coeff=[1,2,3]
```

See Also

Polynomial.join()

Polynomial.add()

Add two Polynomials

Synopsis

Polynomial.add(a,b)

Arguments

a,b The Polynomials to be added.

Returns

The result of the Polynomial addition a+b.

Example

```
var x=new Polynomial( 1,2,3 );          // x^2+2x+3
var y=new Polynomial(5,6);               // 5x+6
var z=new Polynomial(Complex(1+i),Complex(2-2i),3); // (1+i)x^2+(2-2i)x+3
```

```
var p=Polynomial.add(x,y)        // result x^2+7x+9
var p2=Polyomial.add(z,x)        // result (1+i)x^2+(4-2i)x+6
```

See Also

Polynomial.div(), Polynomial.mul(), Polynomial.sub(), Polynomial.rem()

Polynomial.compositedeflate()

Deflate a root of the Polynomial using composite deflation

Synopsis

Polynomial object.compositedeflate(z)

Arguments

z The root by which the Polynomial is composite deflated.

Returns

No returns

Result

The *Polynomial Object* has been deflated with the root z. Notice the deflation is done using composite deflation meaning dividing the root z into Polynomial using both a forward and backward deflation and then determine for which power x^y to begin using the backward deflated coefficients to minimize the division error and using forward deflated coefficients for power higher than x^y. For the coefficients at x^y the coefficient is calculated as the average between the forward and backward deflated coefficient for x^y..

Example

```
var p = new Polynomial( 1,-6,11,-6 );// 1x^3-6x^2+11x-6
var x = 2;                           // one root is 2

p.compositedeflate(x);               // result   x^2-4x+3
```

See Also

Polynomial.deflate()

Polynomial.degree()

Return the degree of the Polynomial object

Synopsis

Polynomial object.degree()

Returns

The degree of the Polynomial object.

Example

```
var p = new Polynomial( 1,-6,11,-6 );// 1x^3-6x^2+11x-6
var n;

n=p. degree();                // n = 3
```

See Also

Polynomial.getcoeff(), Polynomial.setcoeff()

Polynomial.deflate()

Deflate a root of the Polynomial using forward deflation

Synopsis

Polynomial object.deflate(z)

Arguments

z The root by which the Polynomial is forward deflated.

Returns

No returns

Result

The *Polynomial Object* has been deflated with the root z. Notice the deflation is done using forward deflation meaning dividing the root z into Polynomial starting with the coefficients with the highest power. E.g. x^n. If the roots are deflated, using increasing magnitude of the root the forward deflation method is numerical stable. If in doubt a root cant been guarantee to be deflated in increasing order of magnitude then use the composite deflation method.

Example

var p = new Polynomial(1,-6,11,-6);// $1x^3$-$6x^2$+11x-6
var x = 2; // one root is 2

p. deflate(z); // result x^2-4x+3

See Also

Polynomial.compositedeflate()

Polynomial.derivative()

Calculate the derivative coefficients of the Polynomial object

Synopsis

Polynomial object.derivative()

Arguments

none

Returns

Return a new Polynomial, which is the derivative of the Polynomial object.

Example

var p = new Polynomial(1,-6,11,-6);// $1x^3$-$6x^2$+11x-6
var dp=p. derivative(); // dp is $3x^2$-12x+11 same as Polynomial(3,-12,11);

See Also

Polynomial.div()

Divide two Polynomial numbers

Synopsis

Polynomial.div(a,b)

Arguments

a,b The Polynomials to be divided.

Returns

The result of the Polynomial division a/b.

Example

var x = new Polynomial(1,-6,11,-6); // 1x3-6x2+11x-6
var y = new Polynomial(1,-2); // x-2

Polynomial.div(x,y) // result x^2-4x+3

See Also

Polynomial.add(), Polynomial.mul(), Polynomial.sub(), Polynomial.rem()

Polynomial.getcoeff()

Get one Polynomial coefficients

Synopsis

Polynomial object.getcoeff(x^{th})

Arguments

x^{th} The coefficient to the x^{th} degree.

Returns

Return the coefficient associated with the x^{th} degree of the Polynomial object.

Example

var p = new Polynomial(1,-6,11,-6);// $1x^3-6x^2+11x-6$
var coeff;

coeff=p. getcoeff(2); // coeff=-6

See Also

Polynomial.setcoeff()(), Polynomial.degree()

Polynomial.isComplex()

Determine if the Polynomial object contains any complex numbers

Synopsis

Polynomial object.isComplex()

Returns

Return true if the Polynomial object contains any coefficients that is a complex number otherwise false.

Example

```
var p = new Polynomial( 1,2,3 );        // x^2+2x+3
var complexnumber;

var complexnumber=p.isComplex()  // return false;
```

See Also

Polynomial.isReal()

Polynomial.isReal()

Determine if the Polynomial object contains all real numbers

Synopsis

Polynomial object.isReal()

Returns

Return true if the Polynomial object coefficients contains all real number otherwise false.

Example

```
var p = new Polynomial( 1,2,3 );        // x^2+2x+3
var onlyreal;

var onlyreal=p.isReal()                 // return true;
```

See Also

Polynomial.isComplex()

Polynomial.join ()

Return the Polynomial object coefficients as a join String

Synopsis

Polynomial object.join(separator)

Arguments

separator Separator character. If omitted the default separator is “,”

Returns

The coefficients of the Polynomial object is returned as a joined string using the separator between coefficients.

Example

var p = new Polynomial(1,2,3); // x^2+2x+3
var str;

var str=p.join() // str=”1,2,3”

See Also

Polynomial.array()

Polynomial.monic()

Bring the Polynomial object into a monic form

Synopsis

Polynomial object.monic()

Returns

A monic Polynomial object where the leading coefficient to $a_n x^n$ is scaled to 1. Taking a_n and divide it up in the other coefficients $a_{n-1},\ldots a_1,a_0$. The same effect can also be archived by using the property *Polynomial*.scale($1/a_n$).

Example

```
var p = new Polynomial( 2,3,4 );      // 2x^2+3x+4

p.monic ()                            // p is now x^2+1.5x+2
```

See Also

Polynominal.scale()

Polynomial.mul()

Multiply two Polynomials

Synopsis

Polynomial.mul(a,b)

Arguments

a,b	The Polynomials to be multiplied.

Returns

The result of the Polynomial multiplication a*b.

Example

```
var x = new Polynomial( 1,-1 );       // x-1
var y = new Polynomial(1,-4,3);       // x^2-4x+3

z=Polynomial.mul(x,y)                 // x^3-6x^2+11x-6
```

See Also

Polynomial.add(), Polynomial.div(), Polynomial.sub(), Polynomial.rem()

Polynomial.normalize ()

Normalize the Polynomial object

Synopsis

Polynomial object.normalize()

Returns

A normalized Polynomial object. This mean removing leading or trailing zeros. Any undefined coefficients is converted to 0.

Example

var p = new Polynomial(0, , 1,2,3); // $0x^4+?x^3+x^2+2x+3$

p.normalize() // p is now x^2+2x+3

See Also

Polynomial.one

Return the constant one as a Polynomial object

Synopsis

Polynomial.one

Returns

The Polynomial constant one object. Same as New Polynomial(1).

Example

var x = Polynomial.one; // x=1 x is a Polynomial object

See Also

Polynomial.zero

Polynomial.parsePolynomial()

Parse and convert a Polynomial string into a Polynomial object

Synopsis

parsePolynomial(string)

Arguments

string String to be parsed into a Polynomial object

Returns

parsePolynomial() parses and return a new Polynomial object contained in s. parsePolynomial() return an empty Polynomial object if parsing fails. A Polynomial object followed the standard syntax:

$a_nx^n+a_{n-1}x^{n-1}+\ldots a_2x^2+a_1x+a_0$

where the coefficients $a_n,a_{n-1},\ldots,a_2,a_1,a_0$ can be either a regular integer, floating point number as in JavaScript or a complex JavaScript number following the complex syntax as outline in the Complex number JavaScript library packages. Notice the format for x^n need to be expressed in a string as x^n.
Furthermore you can use Polynomial arithmetic (*,+,-,/), grouping with () and power with the ^ operator, see example below. The returned Polynomial object is guarantee to not have any undefined coefficients. E.g x^5-1 is the same as new Polynomial(1,0,0,0,0,-1);
.

Example

```
var p = parsePolynomial("x^2+3x+6");        // Same as Polynomial(1,3,6) or x²+3x+6
p = parsePolynomial( "-2x^2+3x-5");         // Same as Polynomial(-2,3,-5);
p = parsePolynomial("(1x^2+2x+3)^2");       // Same as Polynomial(1,4,10,12,9);
p=parsePolynomial("(x-1)(x-2)(x-3)");       // Same as Polynomial(1,-6,11,-6);
```

Notice

The string can also contains Polynomial expression with the operator +,-,*,/,%,^ and arithmetic grouping with (). E.g.
"(x-1)(x-2)*(x^3-1)^5" or "(x-1)^15" are all valid strings that can be converted by parsePolynomial() into a Polynomial object. Complex number can also be handle e.g. "(3-i4)x^3+(-2)x^2+(-i4)x-(3)" notice you would need to group them using () as coefficient to x^n
Also notice you do not need the * operator in from of a () as it also interpret implicit multiplication correctly.

See Also

new Polynomial(), Polynomial()

Polynomial.pow()

Compute p^y where p is a Polynomial

Synopsis

Polynomial.pow(p,n)

Arguments

p	A Polynomial object to be raised to a power
n	The power that *p* is raised to. n need to be a positive integer

Returns

x to the power of y or x^y

Example

```
var p = new Polynomial(1,2,3);        // x^3+2x+3
var n = 2;

Polynomial.pow(p,2);                  //  x^4+4x^3+10x^2+12x+9
```

See Also

Polynomial.mul()

Polynomial.rem()

Divide two Polynomial numbers and return the remainder Polynomial

Synopsis

Polynomial.rem(a,b)

Arguments

a,b	The Polynomials to be divided.

Returns

The remainder Polynomial as a result of the division a/b.

Example

```
var x = new Polynomial(1,-6,11,-6 ); // 1x3-6x2+11x-6
var y = new  Polynomial(1,-2);         // x-2
```

var z=Polynomial.rem(x,y)	// result 0 because x is dividable by y with a zero remainder

See Also

Polynomial.add(), Polynomial.mul(), Polynomial.sub(), Polynomial.div()

Polynomial.scale ()

Scale the Polynomial object

Synopsis

Polynomial object.scale(scale)

Arguments

scale	Optional parameter with the scale factor. If omitted an auto scaling is performed.

Returns

A scaled Polynomial object where the all coefficients of the Polynomial are multiplied by scale parameter. If scale parameter is omitted it is auto scaled based on very large or very small coefficients. Computes a scale factor to multiply the coefficients of the polynomial. The scaling is done to avoid overflow and to avoid undetected underflow interfering with the convergence criterion.
If auto scaled, the scale factor is a power of the base (2) to avoid loss of precision.

Example

var p = new Polynomial(2,3,4);	// $2x^2+3x+4$
p.scale (0.5)	// p is now $x^2+1.5x+2$

See Also

Polynomial.setcoeff()

Set one Polynomial coefficient

Synopsis

Polynomial object.setcoeff(x^t,*newcoeff*^{h})

Arguments

x^{th}	The coefficient to the x^{th} degree.
newcoeff	The replacement value for the x^{th} coefficient

Returns

Return the new coefficient associated with the x^{th} degree of the Polynomial object.

Example

var p = new Polynomial(1,-6,11,-6);// $1x^3-6x^2+11x-6$
var coeff;

coeff=p. setcoeff(2,8); // coeff=8 and Polynomial is $1x^3+8x^2+11x-6$

See Also

Polynomial.getcoeff()(), Polynomial.degree()

Polynomial.shift()

Do a Polynomial Taylor shift of the Polynomial resulting in new coefficients

Synopsis

Polynomial object.shift(*n*)

Returns

Do a Polynomial Taylor shift of offset *n*. The new Polynomial has the same roots as the original Polynomial shift *n* to the left. E.g. if a root was 2 prior to shift then *Polynomial object*.shift(1) result in the new polynomial has a root of 1. The original Polynomial root can be recreated by adding the shift *n*, to all the root and these roots are the roots of the original Polynomial.

Example

var p = new Polynomial(1,0,0,0,-1); // $1x^{5-1}$

p. shift(1); // Polynomial is now $1x^5+5x^4+10x^3+10x^2+5x$

See Also

Polynomial.scale()

Polynomial.simplify()

Simplify the Polynomial coefficients

Synopsis

Polynomial object.simplify()

Returns

Simplify Polynomial where all complex number coefficients with an imaginary part of 0 has been converted into a real number.

Example

var p = new Polynomial(1,New Complex(-6,0),11,-6); // $1x^3(-6+i0)x^2+11x-6$

p. simplify(); // Polynomial is now $1x^3-6x^2+11x-6$

See Also

Polynomial.getcoeff()(), Polynomial.degree()

Polynomial.sub()

Subtract two Polynomials

Synopsis

Polynomial.sub(a,b)

Arguments

a,b The Polynomials to be subtracted.

Returns

The result of the Polynomial subtraction.

Example

```
var p1=new Polynomial( 1,2,3 );                    //x^2+2x+3
var p2=new Polynomial (4,5);                       //4x+5
var p3=new Polynomial(Complex(1+i),Complex(2-2i),3); // (1+i)x^2+(2-2i)x+3

Polynomial.sub(p1,p2);                             // x^2-2x-2
Polynomial.sub(p3,p1);                             // (0+i)x^2+(0-2i)x+0
```

See Also

Polynomial.add(), Polynomial.div(), Polynomial.mul(), Polynomial.rem()

Polynomial.toExponential()

Format a Polynomial using exponential notation for the coefficients

Synopsis

Polynomial.toExponential(digits)

Arguments

Digits The number of digits that will appear after the decimal point. This may be a value between 0 and up. If this argument is omitted, as many digits as necessary will be used.

Returns

A string representations of the Polynomial, where all coefficients are in exponential notation, with one digit before the decimal place and *digits* digits after the decimal place. The fractional part of the Polynomial coefficients number is rounded, or padded with zeros, as necessary, so that is has the specified length.

Example

```
var p=new Polynomal(10.5567,1.66,-200)    //10.5567x^2+1.66x-200
p.toExponential(1);                       // 1.1e+2x^2+1.7e+0-2e+2
p.toExponential(3);                       // 1.056e+1x^2+1.66x-2e+2
p.toExponential();                        // 1.05567e+1x^2+1.66x-2e+2
```

See Also

Polynomial.toFixed(), Polynomial.toPrecision(), Polynomial.toString()

Polynomial.toFixed()

Format a Polynomial using fixed-point notation for the coefficients

Synopsis

Pylonomial.toFixed(digits)

Arguments

Digits	The number of digits that will appear after the decimal point for the Polynomial coefficients. This may be a value between 0 and 20, inclusive. If this argument is omitted, it is treated as zero.

Returns

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

Example

var p=new Polynomal(10.5567,1.66,-200)	//10.5567x^2+1.66x-200
p.toFixed(1);	// 10.1x^2+1.7-200
p.toFixed(3);	// 10.557x^2+1.66x-200
p.toFixed();	// 11x^2+2x-200

See Also

Pylonomial.toExponential(), Polynomial.toPrecision(), Pylonomial.toString()

Polynomial.toPrecision()

Format the significant digits of a Polynomial coefficients

Synopsis

Polynomial.toPrecision(digits)

Arguments

Digits The number of significant digits to appear as the coefficients 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.

Returns

A string representations of the *Polynomial*, that contains *precisions* significant digits in the cofficients. If *precision* is large enough to include all the digits of the integer part of 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 p=new Polynomal(10.5567,1.66,-200) //10.5567x^2+1.66x-200
p.toFixed(1); // 1.1e+1x^2+1.7-2e+2
p.toFixed(3); // 1.557e+1x^2+1.66x-200

See Also

Polynomial.toExponential(), Polynomial.toFixed(), Polynomial.toString()

Polynomial.toString()

Format the significant digits of a Polynomial coefficients

Synopsis

Polynomial.toString(radix)

Arguments

Radix If omitted the base 10 will be used to convert the Polynomial coefficients to a string. Otherwise the radix will be used (2..36).

Returns

A string representations of the *Polynomial*, in the indicated radix, returned as a normalized number.

Example

var p=new Polynomal(10.5567,1.66,-200) //10.5567x^2+1.66x-200

p.toString(); // //10.5567x^2+1.66x-200

See Also

Polynomial.toExponential(), Polynomial.toFixed(), Polynomial.toPrecision()

Polynomial.valueof()

Return the primitive value of the coefficients as an array

Synopsis

Polynomial object.valueof()

Returns

The primitive value of the *Polynomial* is returned in normalized form.

Example

```
var p = new Polynomial(1,2,3);          //x^2+2x+3
p.valueOf()               // return 1
```

See Also

Polynomial.value()

Return the value of the Polynomial at point x.

Synopsis

Polynomial object.value(z)

Arguments

z	z is the point at which to calculate the value of the Polynomial object. z can be an integer, floating point or complex number.

Returns

Return the value of the Polynomial at point z.

Example

```
var p = new Polynomial(1,-5,6);        //x^2-5x+6
p.value(1)                             // return 2
```

See Also

Polynomial.zero

Return an Empty Polynomial object

Synopsis

Polynomial.zero

Returns

The Polynomial object where the coefficients is undefined

Example

```
var p = Polynomial.zero;
```

See Also

Polynomial.one

Polynomial.zeros()

Find all the roots of the Polynomial.

Synopsis

Polynomial object.zeros(method,verbose,composite)

Arguments (Notice all arguments is optional)

method	Select the method to use for finding the zeros. Currently the following methods are supported: *"Newton", "Ostrowski", "Halley", Householder"*. If method is undefined it will default to Newton's method.
verbose	If verbose is true verbose information of how the root finding progress is generated and return as Array[0] in the return Array. If *verbose* is undefined it defaults to false;
composite	If composite is true the deflation is done using the composite deflation method. Since we find the root in increasing magnitude the default

forward deflation method is just as an accurate as using the composite deflation method. If composite is undefined it defaults to false;

Returns

Return all the zeros of the Polynomial object as an Array. For a polynomial with degree n the roots in the Array is from 1..n for a total of n roots. Array [0] contains the verbose information generated as a textual string.

Example

```
var p = new Polynomial(1,-5,6);         //x^2-5x+6
var x=p.zeros("Newton",false,false); // return
                                     // x[2]=1.9999999999999997,
                                     // x[1]=3.0000000000000042
                                     // x[0]=""; verbose is false;
```

See Also