

# Polynomial

## Method Summary

void	<a href="#">addPolynomial(polynomial)</a>	Adds another polynomial to this polynomial.
void	<a href="#">addTerm(coefficient, exponent)</a>	Adds a term to this polynomial.
Number	<a href="#">findRoot(startValue, error, iterations)</a>	Finds a root of this polynomial using Newton's method, starting from an initial search value, and with a given precision.
Polynomial	<a href="#">getDerivative()</a>	Returns a polynomial that holds the derivative of this polynomial.
Number	<a href="#">getDerivativeValue(x)</a>	Returns the value of the derivative of this polynomial in a certain point.
Number	<a href="#">getValue(x)</a>	Returns the value of this polynomial in a certain point.
void	<a href="#">multiplyByPolynomial(polynomial)</a>	Multiplies this polynomial with another polynomial.
void	<a href="#">multiplyByTerm(coefficient, exponent)</a>	Multiplies this polynomial with a term.
void	<a href="#">setToZero()</a>	Sets this polynomial to zero.

## Method Details

### **addPolynomial**

void **addPolynomial** (polynomial)  
Adds another polynomial to this polynomial.

#### Parameters

{[Polynomial](#)} polynomial

#### Returns

void

#### Sample

```
// (x+1) + 2*(x+1)*x + 3*(x+1)*x^2 + 4*(x+1)*x^3
var eq = plugins.amortization.newPolynomial();
for (var i = 0; i < 4; i++)
{
    var base = plugins.amortization.newPolynomial();
    base.addTerm(1, 1);
    base.addTerm(1, 0);
    base.multiplyByTerm(1, i);
    base.multiplyByTerm(i + 1, 0);
    eq.addPolynomial(base);
}
application.output(eq.getValue(2));
```

### **addTerm**

void **addTerm** (coefficient, exponent)  
Adds a term to this polynomial.

#### Parameters

{[Number](#)} coefficient  
{[Number](#)} exponent

#### Returns

void

**Sample**

```
// (x+1) + 2*(x+1)*x + 3*(x+1)*x^2 + 4*(x+1)*x^3
var eq = plugins.amortization.newPolynomial();
for (var i = 0; i < 4; i++)
{
    var base = plugins.amortization.newPolynomial();
    base.addTerm(1, 1);
    base.addTerm(1, 0);
    base.multiplyByTerm(1, i);
    base.multiplyByTerm(i + 1, 0);
    eq.addPolynomial(base);
}
application.output(eq.getValue(2));
```

**findRoot****Number** **findRoot** (startValue, error, iterations)

Finds a root of this polynomial using Newton's method, starting from an initial search value, and with a given precision.

**Parameters**

{**Number**} startValue  
{**Number**} error  
{**Number**} iterations

**Returns****Number****Sample**

```
// Model the quadratic equation -x^2 + 4x + 0.6 = 0
var eq = plugins.amortization.newPolynomial();
eq.addTerm(-1, 2);
eq.addTerm(4, 1);
eq.addTerm(0.6, 0);
// Find the roots of the equation.
r1 = eq.findRoot(100, 1E-5, 1000);
r2 = eq.findRoot(-100, 1E-5, 1000);
application.output("eq(" + r1 + ")=" + eq.getValue(r1));
application.output("eq(" + r2 + ")=" + eq.getValue(r2));
// Find the minimum/maximum point by zeroing the first derivative.
var deriv = eq.getDerivative();
rd = deriv.findRoot(0, 1E-5, 1000);
application.output("Min/max point: " + rd);
application.output("Min/max value: " + eq.getValue(rd));
if (deriv.getDerivativeValue(rd) < 0) application.output("Max point.");
else application.output("Min point.");
```

**getDerivative****Polynomial** **getDerivative** ()

Returns a polynomial that holds the derivative of this polynomial.

**Returns****Polynomial**

**Sample**

```
// Model the quadratic equation -x^2 + 4x + 0.6 = 0
var eq = plugins.amortization.newPolynomial();
eq.addTerm(-1, 2);
eq.addTerm(4, 1);
eq.addTerm(0.6, 0);
// Find the roots of the equation.
r1 = eq.findRoot(100, 1E-5, 1000);
r2 = eq.findRoot(-100, 1E-5, 1000);
application.output("eq(" + r1 + ")=" + eq.getValue(r1));
application.output("eq(" + r2 + ")=" + eq.getValue(r2));
// Find the minimum/maximum point by zeroing the first derivative.
var deriv = eq.getDerivative();
rd = deriv.findRoot(0, 1E-5, 1000);
application.output("Min/max point: " + rd);
application.output("Min/max value: " + eq.getValue(rd));
if (deriv.getDerivativeValue(rd) < 0) application.output("Max point.");
else application.output("Min point.");
```

**getDerivativeValue****Number getDerivativeValue (x)**

Returns the value of the derivative of this polynomial in a certain point.

**Parameters**

{Number} x

**Returns**

Number

**Sample**

```
// Model the quadratic equation -x^2 + 4x + 0.6 = 0
var eq = plugins.amortization.newPolynomial();
eq.addTerm(-1, 2);
eq.addTerm(4, 1);
eq.addTerm(0.6, 0);
// Find the roots of the equation.
r1 = eq.findRoot(100, 1E-5, 1000);
r2 = eq.findRoot(-100, 1E-5, 1000);
application.output("eq(" + r1 + ")=" + eq.getValue(r1));
application.output("eq(" + r2 + ")=" + eq.getValue(r2));
// Find the minimum/maximum point by zeroing the first derivative.
var deriv = eq.getDerivative();
rd = deriv.findRoot(0, 1E-5, 1000);
application.output("Min/max point: " + rd);
application.output("Min/max value: " + eq.getValue(rd));
if (deriv.getDerivativeValue(rd) < 0) application.output("Max point.");
else application.output("Min point.");
```

**getValue****Number getValue (x)**

Returns the value of this polynomial in a certain point.

**Parameters**

{Number} x

**Returns**

Number

**Sample**

```
// Model the quadratic equation -x^2 + 4x + 0.6 = 0
var eq = plugins.amortization.newPolynomial();
eq.addTerm(-1, 2);
eq.addTerm(4, 1);
eq.addTerm(0.6, 0);
// Find the roots of the equation.
r1 = eq.findRoot(100, 1E-5, 1000);
r2 = eq.findRoot(-100, 1E-5, 1000);
application.output("eq(" + r1 + ")=" + eq.getValue(r1));
application.output("eq(" + r2 + ")=" + eq.getValue(r2));
// Find the minimum/maximum point by zeroing the first derivative.
var deriv = eq.getDerivative();
rd = deriv.findRoot(0, 1E-5, 1000);
application.output("Min/max point: " + rd);
application.output("Min/max value: " + eq.getValue(rd));
if (deriv.getDerivativeValue(rd) < 0) application.output("Max point.");
else application.output("Min point.");
```

**multiplyByPolynomial**

**void multiplyByPolynomial (polynomial)**  
Multiplies this polynomial with another polynomial.

**Parameters**

{[Polynomial](#)} polynomial

**Returns**

void

**Sample**

```
// Model the quadratic equation (x+1)*(x+2) = 0
var eq = plugins.amortization.newPolynomial();
eq.addTerm(1, 1);
eq.addTerm(1, 0);
var eq2 = plugins.amortization.newPolynomial();
eq2.addTerm(1, 1);
eq2.addTerm(2, 0);
eq.multiplyByPolynomial(eq2);
// Find the roots of the equation.
r1 = eq.findRoot(100, 1E-5, 1000);
r2 = eq.findRoot(-100, 1E-5, 1000);
application.output("eq(" + r1 + ")=" + eq.getValue(r1));
application.output("eq(" + r2 + ")=" + eq.getValue(r2));
```

**multiplyByTerm**

**void multiplyByTerm (coefficient, exponent)**  
Multiplies this polynomial with a term.

**Parameters**

{[Number](#)} coefficient  
{[Number](#)} exponent

**Returns**

void

**Sample**

```
// (x+1) + 2*(x+1)*x + 3*(x+1)*x^2 + 4*(x+1)*x^3
var eq = plugins.amortization.newPolynomial();
for (var i = 0; i < 4; i++)
{
    var base = plugins.amortization.newPolynomial();
    base.addTerm(1, 1);
    base.addTerm(1, 0);
    base.multiplyByTerm(1, i);
    base.multiplyByTerm(i + 1, 0);
    eq.addPolynomial(base);
}
application.output(eq.getValue(2));
```

**setToZero**

```
void setToZero()
```

Sets this polynomial to zero.

**Returns**

```
void
```

**Sample**

```
var eq = plugins.amortization.newPolynomial();
eq.addTerm(2, 3);
application.output(eq.getValue(1.1));
eq.setToZero();
application.output(eq.getValue(1.1));
```