amortization

Return Types

AmortizationCalculation Polynomial

Method Summery

AmortizationCalculation	#newCalculation() Creates a new amortization calculation.
Polynomial	#newPolynomial([polynomial]) Creates a new polynomial which is either 0 or a copy of the specified polynomial.
Date	#nextDate(start_date, period, startday)

Method Details

newCalculation

AmortizationCalculation **newCalculation**()

Creates a new amortization calculation.

Returns

AmortizationCalculation

Sample

```
// Calculate the interest rate for an amortization schedule
// with a loan of 2000 dollars  on January 1, 2005, and 5
// monthly payments of 500 dollars starting on Febuary 28th,
// payments made on the last day of each month.

// Get a new amortization calculation.
var c = plugins.amortization.newCalculation();

// Set the rate to -1 for unknown.
c.addRateChange(-1, new Date(2005, 0, 1));

// Set the compounding period to monthly.
c.addCompoundPeriodChange(12, new Date(2005, 0, 1));

// Add the loan and the payments to the schedule.
c.addLoan(2000, new Date(2005, 0, 1));
var lastDate = null;
var period = 12;
//valid periods are:
//PERIOD_ANNUALY : 1
//PERIOD_BI_ANNUALLY : 2
//PERIOD_DAILY : 365
//PERIOD_FOUR_MONTHLY : 3
//PERIOD_FOUR_WEEKLY : 13
//PERIOD_MONTHLY : 12
//PERIOD_QUARTERLY : 4
//PERIOD_TWO_MONTHLY : 6
//PERIOD_TWO_WEEKLY : 26
//PERIOD_WEEKLY : 52
var number_count = 5;
var startday = 31;
c.addPayment(500, new Date(2005, 1, 28), lastDate, period, number_count, startday);

// Solve for the interest rate.
c.solveForUnknown();
// Get the interest rate and the error in the calculation.
// which should be small (otherwise the calculation did
// not converge for some reason.
var r = c.getUnknown();
var e = c.getError();

// When there are no unknowns you can calculate the
// actual amortization schedule.

// Same as before, use the calculated interest rate.
var c = plugins.amortization.newCalculation();
c.addRateChange(r, new Date(2005, 0, 1));
c.addCompoundPeriodChange(12, new Date(2005, 0, 1));
c.addLoan(2000, new Date(2005, 0, 1));
c.addPayment(500, new Date(2005, 1, 28), null, 12, 5, 31);

// Calculate the actual amortization schedule.
c.calculateAmortizationSchedule();

// Get the amortization schedule (which is a JSDataSet) and
// convert it to html. This way you can put it on a label.
// As a JSDataSet you can just get the values stored in
// the rows and columns to use in your script.
var s = "<html>" + c.getAmortizationSchedule().getAsHTML();

// Get the rest balance, which is the amount left over after
// the amortization schedule. In our case (since we calculated
// the rate to have nothing left, it should amount to no more
// than 1 or 2 cents due to rounding).
var rb = c.getRestBalance();
```

newPolynomial

Polynomial **newPolynomial**([polynomial])
Creates a new polynomial which is either 0 or a copy of the specified polynomial.

Parameters
[polynomial]

Returns
Polynomial

Sample

```
// create a new polynomial
var p = plugins.amortization.newPolynomial();

// make the polynomial -4*x^2 + 9
p.addTerm(-4, 2); // -4*x^2
p.addTerm(9, 0); // 9

// find the roots
var r1 = p.findRoot(1, 5E-15, 100);
var r2 = p.findRoot(-1, 5E-15, 100);

// get the derivative
q = p.getDerivative();

// show all this information in a dialog
plugins.dialogs.showInfoDialog(
        "polynomial",
        "polynomial: " + p + "\n" +
        "derivative: " + q + "\n" +
        "value in 2: " + p.getValue(2) + "\n" +
        "derivative in 2: " + p.getDerivativeValue(2) + "\n" +
        "root near 1: " + r1 + "\n" +
        "root near -1: " + r2,
        "Ok"
);

// set q to zero
q.setToZero();

// make a copy of p, then add 1 and multiply by 3*x^2
var s = plugins.amortization.newPolynomial(p);
s.addTerm(1, 0); // add 1
s.multiplyByTerm(3, 2); // mulitply by 3*x^2

// make a copy of s and add p
var t = plugins.amortization.newPolynomial(s);
t.addPolynomial(p);

// make a copy of s and multiply by p
var u = plugins.amortization.newPolynomial(s);
u.multiplyByPolynomial(p);

// show all this information in a dialog
plugins.dialogs.showInfoDialog(
        "polynomial",
        "polynomial: " + p + "\n" +
        "polynomial: " + s + "\n" +
        "sum: " + t + "\n" +
        "sum in 2: " + p.getValue(2) + " + " + s.getValue(2) + " = " + t.getValue(2) + "\n" +
        "product: " + u + "\n" +
        "product in 2: " + p.getValue(2) + " * " + s.getValue(2) + " = " + u.getValue(2) + "\n",
        "Ok"
);
```

nextDate

Date **nextDate**(start_date, period, startday)
Replace with description

Parameters
start_date
period
startday

Returns
Date
Sample

```
//Helper method to get the next date when a period is given;
//valid periods are:
//PERIOD_ANNUALY : 1
//PERIOD_BI_ANNUALLY : 2
//PERIOD_BI_MONTHLY : 365
//PERIOD_FOUR_MONTHLY : 3
//PERIOD_FOUR_WEEKLY : 13
//PERIOD_MONTHLY : 12
//PERIOD_QUARTERLY : 4
//PERIOD_TWO_MONTHLY : 6
//PERIOD_TWO_WEEKLY : 26
//PERIOD_WEEKLY : 52
var a_next_date = plugins.amortization.nextDate(new Date(2005, 1, 28),12,31);

//Sample to calculate intervals between 2 dates;
var startday = 31;
var d_start = new Date(2005, 1, 28)
var d_end = new Date(2005, 11, 20)
var allDates = new Array();
var next = d_start;
//loop over months
while (next.getTime() < d_end.getTime())
{
        next = plugins.amortization.nextDate(next,12,startday)
        allDates[allDates.length] = next;
}
if (allDates.length > 0) allDates.length = allDates.length-1 //clear was one to far
//now check how many days are left
if (allDates.length > 0)
{
        next = allDates[allDates.length-1]
}
else
{
        next = d_start;
}
var days = 0;
//loop over days
while (next.getTime() < d_end.getTime() || next.getDate() < d_end.getDate())
{
        next = plugins.amortization.nextDate(next,365,startday)
        days++;
}
//add total days
allDates[allDates.length] = '(and) '+days+' days'
//make array textual to show
application.output( allDates.join('\n') );
```