Freeloan – short overview

# function testbench(nr)

This function is not a part of freeLoan. It is made for testing freeLoan. testbench() uses some external functions, like QuickSort.

testbench() runs on current data from Finansportalen’s house loans.

If given an integer parameter, it will compute only product number “nr” in the Finansportalen list and display a graph (at the bottom of the price list) for this product. Otherwise, it computes all the Finansportalen loans.

# function freeLoan()

function freeLoan() has these possible parameters (The white fields are user inputs. The yellow fields are retrieved from the bank. The pink field is for administrator.)

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Obligatory** | **Comment** |
| **received** | Number | Yes | The loan amount received by the borrower. (Due to fees, this might deviate from the principal in the bank's books). Any number. |
| **numberofperiods** | Integer | Yes | The number of periods the loan will run. A 20 year house loan with monthly payments has 240 periods. |
| **periods\_per\_year** | Integer | Yes | In order to compute *annual*, effective interest rate, some connection to years must be made. 2, 3, 4, 6 or 12 are recommended, but any integer should work. |
| **serial** | Boolean | No | 0, 'false' or omitted: Annuity loan. 'true'/'on'/1 = Serial loan. |
| **balloon** | Integer | No | 0, 'false' or omitted: No. Integer > 0: The planned residual value of the loan ("balloon") to be paid when the loan period is over. |
| **interestonly\_periods** | Integer | No | 0, 'false' or omitted: zero. Integer > 0: The initial number of interest-only periods requested by the customer. |
| **annuity\_due** | Boolean | No | 0, 'false' or omitted or "immediate": Annuity-immediate 1, "true" or "due": Annuity\_due \*\*) |
| **round\_direction n** | Integer | No | 0, 'false' or omitted: Annuities are rounded according to normal rules 1: Rounded up 2. Rounded down |
| **round\_presision n** | Integer | No | 0, 'false' or omitted: Payment rounded to nearest 1/100 1: Rounded to nearest integer |
| **remainder\_handling n** | Integer | No | 0, 'false' or omitted: The "global" remainder at the end of the loan period is payed / compensated with the last payment 1: The "global" remainder is ignored |
| **capitalization\_freq** | Integer | No | 0, 'false' or omitted: 12 (Capitalization 12 times a year). Integer>0: Any number of capitalizations per year (2,3,4,6 or 12 recommended). \*\*\*) |
| **interestonly\_periods\_max** | Integer | No | 0, 'false' or omitted: 0. Integer >= 0: The maximal interest only-period offered by the bank. In years. |
| **fee\_processing** | Number | No | 0, 'false' or omitted: 0. Number >= 0: Processing fee: A one-time fee of a fixed sum to be payed at the beginning of the loan period. \*\*\*\*) |
| **fee\_document** | Number | No | 0, 'false' or omitted: 0. Number >= 0: Document preparation fee: A one-time fee of a fixed sum to be payed at the beginning of the loan period. \*\*\*\*) |
| **fee\_percentage** | Number | No | 0, 'false' or omitted: 0 .Number >= 0: Percentage fee: One-time fee to be payed at the beginning of the loan period computed out of the principle (gross loan). 2 = 2%. \*\*\*\*) |
| **fee\_period\_perc** | Number | No | 0, 'false' or omitted: No percentage fee. Integer: Percentage. Loans given as a credit line have a periodical has a fee as a percentage of principal PER PERIOD |
| **rate\_thresholds n** | Boolean | No | 0, 'false' or omitted: No - initial interest rate is fixed for the whole loan for the whole loan period. 1 or 'true': Rate might change during the loan period when certain thresholds are passed. |
| **rate\_segments n** | Boolean | No | 0, 'false' or omitted: No - all segments of the loan has the same interest rate. 1 or 'true': Every segment might have separate interest rates. \*\*\*\*\*\*) |
| **price\_storage** | Array | Yes | The parameter 'price\_storage' is a two-dimensional array. It must contain at least one interest rate, in 'price\_storage[1][4]'. 'price\_storage[0]' is not expected nor used.\*\*\*\*\*\*\*) |
| **ignore\_origination** | Boolean | No | 0, 'false' or omitted: Origination fee added to the loan and included in the computation. 1 or true: Computation performed without origination fee \*) |
| **accuracy** | Integer | Yes | (FOR ADMINISTRATOR) 0: Fast, inaccurate 1: Normal 2: Extremely accurate. Only for serial loans. Does not apply to annuity loans. |

**\*)** 'ignore\_origination': Several charges could be incurred when taking a loan. Some are a percentage of the loan, some are a fixed sum. If the 'ignore\_origination' variable is TRUE, these start-up fees are ignored when computing effective interest rate. It is logically correct to include them ('ignore\_origination' = false), and they must be included in order to compute correctly according to a Norwegian state regulation. But the function is technically able to do both.

**\*\*)** 'annuity-due == false' implies Annuity-immediate:

Annuity-immediate (interest in arrears): Equal payments are made at the end of each period. This is the most commonly used loan type.

Annuity-due (interest in advance): Payments are made at the beginning of each period. For annuity loans, we presuppose that the whole of the annuity is paid in advance. For serial loans, only the interest is paid in advance.

**\*\*\*)** The annuity period could deviate from the payment period. For instance, the loan might have four annuities a year, but still be paid monthly.

**\*\*\*\*)** Origination fees: There might be several one-time fees payable at the start of the loan period: Arrangement fee, processing fee, application fee, origination fee, appraisal fee, credit report fee, tax service fee, underwriting fee, document preparation fee, wire transfer fee, office administration fee and many others. This function allows two parameters for fees of a fixed sum, called 'fee\_processing' and 'fee\_document'.

Together, these one-time start-up fees are referred to as "origination fees".

(There are normally recurring / periodical fees as well. One is in the variable 'fee\_period\_perc'. Others are in the array 'price\_storage').

There is also a parameter for a one-time percentage fee due at the beginning of the loan period: 'fee\_percentage'. This if often the case for open credit facilities. We make the presumption that the loan is drawn to the credit limit at the start of the loan period, so that this percentage is simply computed from the initial principal.

**\*\*\*\*\*)** Normally, you are offered one interest rate for the whole loan sum for the whole loan period. But in some loan contracts the interest rate changes when the loan is payed down beyond certain thresholds. This is signaled with 'rate\_thresholds' == true.

\*\*\*\*\*\*) Even if the interest rate might change when the principal passes certain thresholds, you normally have only one interest rate at the time. But there are loan contracts where the loan might run with separate interest rates in each segment. For instance, the rate for the segment between 0 and 500,000 could be 4%, between 500,000 and 1,000,000 3.5% and above 1,000,000 it could be 3%. All applying at the same time. This is the case when 'rate\_segments' == true.

**\*\*\*\*\*\*\*)** The parameter 'price\_storage' is a two-dimensional array - a matrix. Each row in the matrix consists of four elements:

|  |  |
| --- | --- |
| price\_storage[step][1] | Lower limit for the rate segment #step. Omitted/empty Interpreted as zero. |
| price\_storage[step][2] | Upper limit for the rate segment #step. Omitted/empty interpreted as "Unlimited". |
| price\_storage[step][3] | The periodical fee for each payment in segment #step. |
| price\_storage[step][4] | OBLIGATORY: The annual interest rate in the segment (as % per anno) |

Each segment/step is defined by a lower and upper limit. The lowest segment comes first, at 'step== 1' . We presuppose that the segments don't overlap.

n ) These data are currently not obtained from the banks by Finansportalen. It would require new fields in “Datafanger”.

## Changing rates

freeLoan() handles three different types of loan segments:

i) Normal: The interest rate for the whole loan is determined by the initial size of the principal when the loan is given. The rate does not change during the loan period. This is the most common and the simplest model.

ii) Thresholds (the parameter 'rate\_thresholds' == true): The interest rate for the whole loan changes when the principal is paid off below certain thresholds. For instance, for a loan of 1,5 million, the rate might be 3.5% until it is paid off below 1 million, when the interest rate might rise to 3.75%. But there is only one rate at the time for the whole loan.

iii) Concurrent rates (the parameter 'rate\_segments' == true): Separate segments of the loan have different interest rates at the same time. All installments are deducted from the uppermost segment of the loan. This segment, thus, is payed off first. For instance, the rate for the segment between 0 and 500,000 could be 4%, between 500,000 and 1,000,000 i could be 3.5% and above 1,000,000 it could be 3%. All applying at the same time. freeLoan() cannot compute annuity loans with payment in advance running with concurrent rates.

# function annuityLoan()

function annuityLoan() has two daughter functions, intervallength() and intervallength\_separate(). It uses almost the same parameters as freeLoan. This is explained in comments in the code.

The main purpose is to find the effective interest rate of the loan. In order to compute this, we first have to know how big every payment is and at which time i occurs. Fees must be included.

When this ihas been computed and stored in an array, we perform iterations to find the rate.

With annuity loans, we don’t store every single payment, as they are the same every period during the whole down payment period (or, if the rate can change – for the whole interval where the rate is the same).

When there are several successive periods with different interest rates ('rate\_thresholds' == true), we must first find out how many periods each interval consists of. This is done with the external function intervallength().

If there are different interest rates running concurrently in different segments ('rate\_segments' == true), the sum of the annuities are still the same during each time interval. We compute the length of this interval with the external function intervallength\_separate().

Finally, we compute the interest rate with iterations, according to Newton’s method. This is quite efficient, and sufficient accuracy (10 decimals) are normally achieved in 3-4 iterations.

As the annuity and other potentially useful results were computed “on the fly” in order to find the effective interest rate, we return these too in the function result. The function, thus, returns an array.

# function serialLoan()

function annuityLoan() has almost the same parameters as freeLoan. It works in much the same fashion as annuityLoan(), but as the payments in a serial loans are different from payment to payment, we store every payment before doing the iterations. This is quite simple, but slow.

In the preface to freeLoan(), the derivation of a formula for the present value of a serial loan is shown. But it is currently not used, because it does not work with rounding.

Typically, a periodic payment comes with an infinite number of decimals. Javascript supports 18, and a monthly payment can be for instance 18,310.385827589115777561 crowns. With normal rounding rules, we pay 18,310.39 crowns. In an annuity formula, this will be the same amount every time. The remainder will also be the same every time, and can be handled as a separate stream of annuities.

But in a serial loan, all payments are different. Hence, the remainder is not known until we have computed the payment. The formula for a serial loan gives us the present value of the unrounded payments, but it we don’t know the effect of the rounding. We don’t even know how big our rounding error is.

So every payment is computed. This makes the function about three times slower than the annuity function. It thus is given an administrator option: In periods with heavy traffic, it is possible, via the parameter ‘accuracy’ to decrease the computing time by about 30%, while still maintaining a normally good enough accuracy.

An alternative venue, not implemented, would be to have two serialLoan() functions, where one utilizes the formula, potentially cutting computation time by 60% or more, but with an uncertain accuracy - albeit probably better than having queues during high traffic.