Freeloan – finer points

# 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”.

# 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 use about three times as much time than the annuity function. It therfore 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.

When there are several successive periods with different interest rates ('rate\_thresholds' == true), the function serialLoan() computes this directly ( while it is done in a separate, external function for annuity loans).

If there are different interest rates running concurrently in different segments ('rate\_segments' == true), this is computed on the fly as well (instead of in a separate function, as with annuity loans).

# Rounding in freeLoan()

Let’s compute an annuity of a loan of 100.000 crowns over one year with a nominal interest rate of 3,5%. Annuities are payed in arrears (annuity-immediate).

We use the annuity formula

|  |
| --- |
| annuity = capital \* (1 - k) / (k – k^(termnum+1))  Where k = 1/(1+r) |

Where ‘r’ is the nominal, periodic interest rate in decimal form.

Nominal, annual interest rate = 3,5%

* r = 3,5/1200
* k = 0,997091815538014

capital = 100000

termnum = 12

* annuity = 8492,1629844068

As we cannot pay this exact sum, we have to round it. In freeLoan() we can round according to the bank’s own rounding rules (provided the parameters ‘*round\_direction’* and ‘*round\_presision’* are set).

Let us say that the bank rounds normally, to the nearest cent:

* annuity = 8492,16

By running this computation backwords, we ask ourselves: How big a loan would an annuity of 8492,16 serve? We use the inverted of the formula above:

|  |
| --- |
| capital = annuity \* (k – k^(termnum+1)) /(1 - k)  Where ‘k’ = 1/(1+r) |

By entering 8492,16 for the annuity and the same termnum and ‘k’ as above, we get that

* capital = 99999,9648569298

**Our rounded annuity is not able to service the 100.000 loan.**

What does the bank do about that? What happens to the remainder, the 0.0351430702285143 crowns?

This is signalled with the parameter ‘*remainder\_handling’.* The remainder is either levied with the last payment or ignored.

In the first case, freeLoan() makes a separate “account” for the remainder, that will increase with the interest rate of the loan from term to term. It will finally be levied with the last payment of the loan, that will either increase or decrease.

In our case, the 0.0351430702285143 crowns will have grown to 0,0363930021537088 after 12 periods/one year. This sum should be added to the last payment. To achieve as great accuracy as possible, we will add the unrounded annuity to the remainder before rounding:

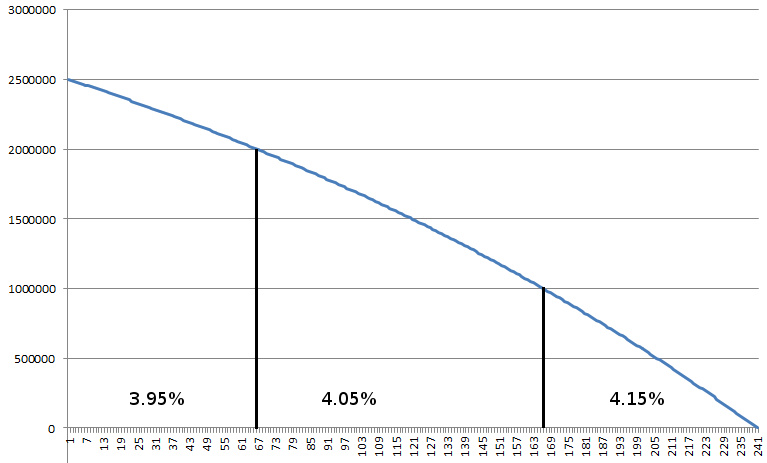
8492,1629844068 + 0,0363930021537088 = 8492,19937740896

* 8492,20

The consequence for freeLoan() is that the effective interest rate differs between the two methods. In our example, when the remainder is taken into account, the effective interest rate is 3.5567019914%. When the remainder is ignored, the effective interest rate is 3.5566277665%.

# function intervallength()

**Interest rate changing at certain thresholds**

Albeit rare, some banks offer loans where the interest rate changes with the remaining principal of the loan. Consider the house loan below, where a loan greater than 2 million has an interest rate of 3.95%, between 1 million and 2 millions 4.05% and below 1 million 4.15%. This example loan runs for 20 years with monthly payments, 240 payments in all:  
  


We can easily compute the first annuity, using the annuity formula:

|  |
| --- |
| annuity = capital \* (1 - k) / (k – k^(termnum+1))  Where k = 1/(1+r) |

It will be 15,083.72

At some point, the first threshold 2,000,000 will be passed, and the rate changes. But how many periods will this take?

We know that if we pay 15.083.72 per month in 240 months, we will have paid off the 2,5 million.

PV(240) = 2,5 millons

After an unknown number of periods – x –, we will pass the 2 million threshold. Then, the present value of the remaining loan will be 2 million.

PV(x) = 2 millons

So PV(240) - PV(x) = 500,000

We use the inverted annuity formula:

|  |
| --- |
| *capital = annuity \* (k – k^(termnum+1)) /(1 - k)*  *Where k = 1/(1+r)* |

The number or terms, ‘termnum’ in the formula, is the unknown, which we call ‘x’:

capital = annuity \* (k – k^(x+1)) /(1 - k)

We call ‘capital’ ‘PV’ and the annuity ‘a’:

I PV = a \* (k – k^(x+1)) /(1 - k)

I (PV \* (1 – k)/a )= (k – k^(x+1))

I (PV \* (1 – k)/a) –k = – k^(x+1)

I (PV \* (1 – k)/a) –k = – k^(x+1)

We multiply by -1:

I k – (PV \* (1 – k)/a) = k^(1+x)

The whole left side consists of constants. We can set

II C = k – (PV \* (1 – k)/a)

I+II C = k^(1+x)

Taking the logarithm of both sides:

log(C) = (1 + x) \* log (k)

log(C) = log (k)+ (x \* log (k))

(log(C)- log (k))/log(k) = x

Subsituting the variable names:

nomrate = 3.95%

r = nomrate/1200

k = 1/(1+r)

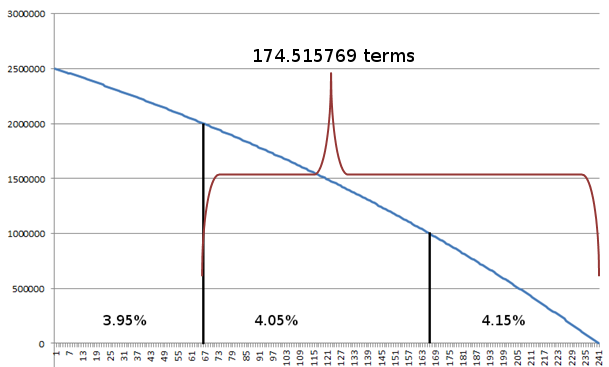
* k = 0.996719133

PV = 2,000,000

a = 15,083.72

* x = 174.515769

Thus, with an annuity of 15,083.72, the present value is 2,000,000 if the loan has 174.515769 terms/payments left to maturity.



As term numbers in the real world are always integers, we have to round this number *down* in order to be certain that the 2 million threshold is passed, and the interest rate thus can change.

174,515769 => 174

The number of periods in the first interval, where the loan runs with an interes rate of 3.95%, is thus 240 – 174 = 66.

We proceed to find the number of terms in the second interval. To that end, we must first compute the annity. The annity depends on the size of the remaining principal after 66 terms.

capital = annuity \* (k – k^(termnum+1)) /(1 - k)

capital = 15,083.72 \* (0,996719133 - 0,996719133^(174+1)) /(1 - 0,996719133)

capital = 1995619,253 = remaining principal after 66 periods.

From this remaining principal, we derive the new annuity with the new interest rate, that is now 4,05 percent annually.

The new interest rate gives us a new discounting factor, k:

nomrate = 4.05%

r = nomrate/1200

k = 1/(1+r)

* k = 0,996636352

annuity = 1995619,253 \* (1 - k) / (k – k^(termnum+1))

annuity = capital \* (1 - 0,996636352) / (0,996636352 – 0,996636352^(174+1))

annuity = 15,183.72

We now repeat what we did above. We want to find how many payments are needed for 15,183.72 to give a present value of 1 million – the next threshold.

Above we found the forumula:

(log(C)- log (k))/log(k) = x

Where C = k – (PV \* (1 – k)/a)

PV now is the next threshold – 1 million.

* x = 74,61309934

Again, we round down, and find that the last interval consists of 74 periods. Hence, we have found all the three intervals. The loan runs with

3.95% for 66 periods

4.05% for 100 periods

4.15% for 74 periods

### Annuities in advance

Here we use the formula for annuitiy-due:

|  |
| --- |
| annuity = capital \* (1 - k) / (1 – k^termnum)  Where k = 1/(1+r) |

Rearranged with respect to ‘capital’:

capital = annuity \* (k – k^(termnum+1)) / (1 - k)

We call the left side – the capital/loan/present value – for ‘PV’ and the annuity ‘a’:

I PV = a \* (1 – k^x) /(1 - k)

I (PV \* (1 – k)/a )= 1 – k^x

I (PV \* (1 – k)/a) –1 = – k^x

We multiply all terms by -1:

I 1 – (PV \* (1 – k)/a) = k^x

The whole left side consists of constants. We can set

II C = 1 – (PV \* (1 – k)/a)

I+II C = k^x

Taking the logarithm of both sides:

log(C) = x \* log (k)

x = log(C) /log(k)

### So what about rounding?

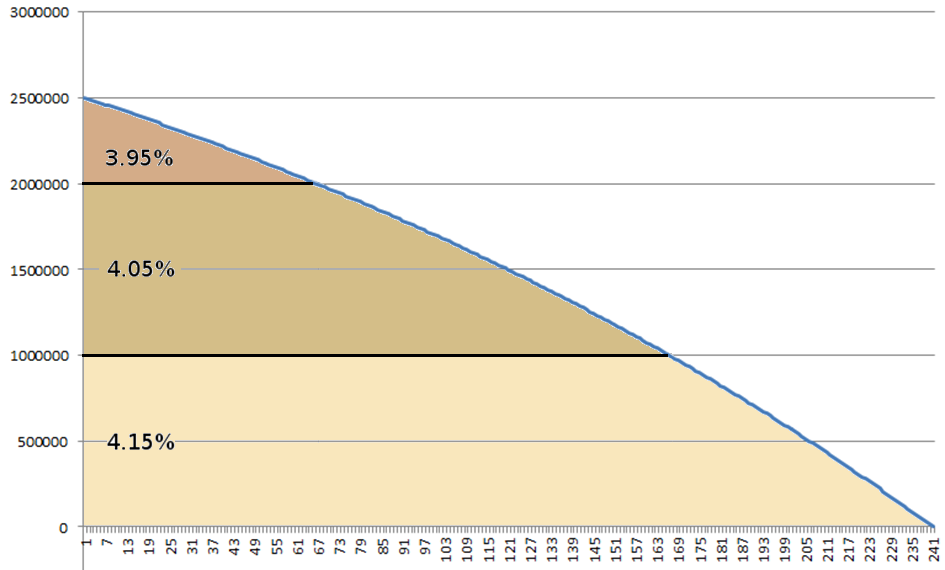
In some cases, the term numbers we find would be very close to the middle point between two integers, for instance 56,499998. Here, the rounding error, treated in the chapter “Rounding in freeLoan()” might influence the outcome.

Hence, the function intervallength() also takes the rounding error into account.

# function intervallength\_separate()

**Different nterest rates running concurrently in separate segments of the loan**

This is a complex concept, where the installments for the periodical payments are computed as annuities, but the installment part is only deducted from the uppermost segment. The uppermost segment, thus, is paid off first.

This function does not support annuitiy-due (annuities in advance).  
  


To start with the first question: What is the first payment?

It is the sum of respective annuities, all computed as if they all would run to the end of loan period:

|  |
| --- |
| capital = annuity \* (k – k^(termnum+1)) /(1 - k)  Where k = 1/(1+r) |

With respect to the annuity:

annuity = capital \* (1 – k) / (k – k^(termnum+1))

The annuity is

annuity = annuity1 +annuity2+annuity3

annuity1 = capital1 \* (1 – k1) / (k1 – k1^(termnum+1))

k1 = 1/(1+r1)

r1 = 3.95%/1200

(We wanted the interest rate in decimal form. Here, we presume 12 terms per year).

* k1 = 0.996719133

termnum = 240

capital1 = 500,000

* annuity1 = 3016.74457

Correspondlingly, we compute annuity2 and annuity3:

annuity1 = 6086.182298

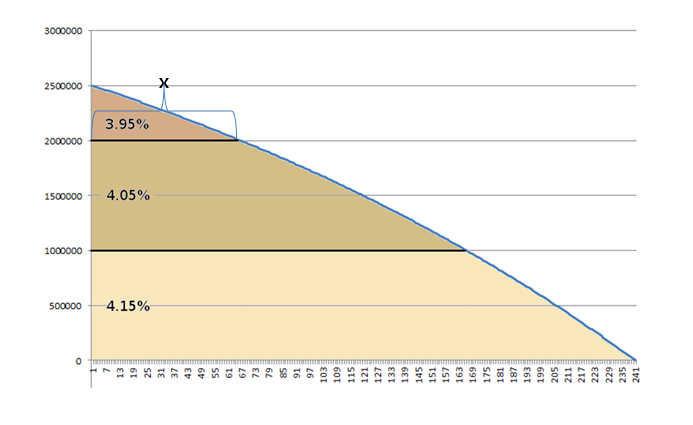
annuity2 = 6139,134436

Given the unusual loan construction, we will need the installment part of the latter two annuities.

|  |  |  |  |
| --- | --- | --- | --- |
| *Segment* | *Annuity* | *Interest* | *Installment* |
| 1 | 6139.13444 | 3458.33333 | 2680.8011 |
| 2 | 6086.1823 | 3375 | 2711.1823 |
| 3 | 3016.74457 |  |  |
| SUM | *15242.0613* |  |  |

The first segments runs as a normal annuity loan. So are the two others, but their installment part is only used to reduce the first segment.

So we are asking: Which time – x – will it take to pay down this first segment?



At the time x, we will see:

Accumulated installment part of the first segment’s annuities, PV1 (x), plus the accumulated installment part of the payments in the two other segments equals 500,000

I PV 1 (x) + 2680.8011 \* x + 2711.1823 \* x = 500,000

I PV 1 (x) + 5391.9834 \* x =500,000

I PV 1 (x) = 500,000 - 5391.9834 \* x

We used the formula:

II PV 1 (x) = annuity1 \* (k1 – k1^(termnum+1)) /(1 - k1)

The number of terms we are looking for is x:

II PV 1 (x) = annuity1 \* (k1 – k1^(x+1)) /(1 - k1)

Currently, this is solved by iterations.