Université Paris 1 Panthéon Sorbonne
Master 1: MATHEMATICAL METHODS, ECONOMICS AND FINANCE (MMEF)

Programming in C++

FINAL PROJECT

BY
Natalia Piątkowska, MSc
April 3, 2019

1 Loan Payment Schedule

A loan is a financial instrument which allows us to borrow some amount of money, that later we have to repay in a number of instalments. Those instalments consist of two parts: **principal** - money that we owe, and **interest** - imposed based on the value of a principal, it's the price we have to pay for a loan.

Below you can find an example of a loan payment schedule:

#	opening balance	principal	interest rate	interest	instalment	closing balance
i	PV_{i-1}	P_i	r_i	Int_i	PMT_i	PV_i
1	10 000.00	2 000.00	5%	500.00	2 500.00	8 000.00
2	8 000.00	2 000.00	5%	400.00	2 400.00	6 000.00
3	6 000.00	2 000.00	5%	300.00	2 300.00	4 000.00
4	4 000.00	2 000.00	5%	200.00	2 200.00	2 000.00
5	2 000.00	2 000.00	5%	100.00	2 100.00	0.00
	TOTAL	10 000.00				

The following dependencies hold $\forall i \in \{1, 2, ..n\}$:

$$
\begin{aligned}
&PMT_i = P_i + Int_i \\
&Int_i = PV_{i-1} \cdot r_i \\
&\sum_{i=1}^{n} P_i = PV_0 = PV \\
&PV_i = PV_{i-1} - P_i \\
&PV_n = 0
\end{aligned}
$$

In the above example, we assume that principal rates are equal. Thus, the instalments are decreasing. It doesn't have to be the case. We can play with size of principal, interest, number of periods, interest rate also may differ between periods. The important is that the above dependencies hold, otherwise the schedule is not valid.

In the examples, interest rate r_i, is periodic. In real world, the interest rate is virtually always given in yearly terms. It means that, in our example, that if the we agree to pay in a yearly frequency, no adjustments have to be made. On the other hand, if yearly rate, let us denote it R_i, and repayment occurs quarterly, we need to convert $r_i = \frac{R_i}{4}$.

Let us consider two specific cases:

1. Fixed rate, fixed instalments
2. Floating interest rate

1.1 Fixed rate, fixed instalments

Fixed rate is obvious, fixed instalments means that the sum principal + interest part is constant. Please do not confuse it with fixed principal payments. Consider an example below.

#	opening balance	principal	interest rate	interest	instalment	closing balance
i	PV_{i-1}	P_i	r_i	Int_i	PMT_i	PV_i
1	10 000.00	1 809.74	5%	500.00	2 309.75	8 190.26
2	8 190.26	1 900.24	5%	409.51	2 309.75	6 290.02
3	6 290.02	1 995.25	5%	314.50	2 309.75	4 294.77
4	4 294.77	2 095.01	5%	214.74	2 309.75	2 199.76
5	2 199.76	2 199.76	5%	109.99	2 309.75	0.00
	TOTAL	10 000.00				

Since we assume that all payments and interest rates are equal, we denote:

$$PMT = PMT_1 = PMT_2 = ... = PMT_n$$
$$r = r_1 = r_2 = ... = r_n$$

From the above, and remembering the dependencies mentioned before, and after many calculations I am going to spare you, you can end up with the equation:

$$\frac{PMT}{PV} = r \cdot \left(1 + \frac{1}{(1+r)^n - 1}\right)$$

Given 3 out of 4 variables, you should be able to compute the missing one. In case of PMT, PV and n you will end up with closed form formula. To calculate r, you will need to create an optimisation algorithm - try with binary search or Secant method.

1.2 Floating rate

When taking a loan, especially for long period (more than a year), lenders prefer to agree for floating rate in order to protect themselves from interest rate risk. Usually what happens is you can get a loan and agree to pay a fixed margin + benchmark rate. Consider an example below.

#	opening balance	principal	margin	benchmark rate	interest rate	interest	instalment	closing balance
i	PV_{i-1}	P_i	m	b_i	r_i	Int_i	PMT_i	PV_i
1	10 000.00	1 500.00	2.00%	2.38%	4.38%	437.60	1 937.60	8 500.00
2	8 500.00	1 500.00	2.00%	2.22%	4.22%	358.70	1 858.70	7 000.00
3	7 000.00	1 500.00	2.00%	0.54%	2.54%	177.54	1 677.54	5 500.00
4	5 500.00	1 500.00	2.00%	2.12%	4.12%	226.76	1 726.76	4 000.00
5	4 000.00	1 000.00	2.00%	2.30%	4.30%	171.81	1 171.81	3 000.00
6	3 000.00	1 000.00	2.00%	1.76%	3.76%	112.78	1 112.78	2 000.00
7	2 000.00	1 000.00	2.00%	2.00%	4.00%	79.98	1 079.98	1 000.00
8	1 000.00	1 000.00	2.00%	2.33%	4.33%	43.28	1 043.28	0.00
	TOTAL	10 000.00						

Each interest rate consist of a constant margin and benchmark rate, so $\forall i \in \{1, 2, ..n\}$:

$$r_i = m + b_i$$

We could use many different models to simulate benchmark rates, but for the beginning let us assume normal distribution, like I did in the example:

$$b_i \sim N(m, \sigma)$$

In C++, to simulate normally distributed random variable, you need to apply Box-Muller algorithm.

2 The task

Write a program that produces loan payment schedule, ideally in the form of a CSV output file, given the following input sets:

- Fixed principal payments, and
  - fixed interest rate
  - margin + parametrised, normally distributed floating interest rate
- Fixed interest rate and instalments, given:
  - n, PV, r
  - PMT, n, r
  - PMT, PV, r
  - PMT, n, PV

  Notice that in all 4 cases, you can calculate the missing parameter from the remaining ones. For all elements except for r, there is a closed form formula. For r, you have to set up an optimisation algorithm.

You should create a comprehensive program, that takes all the cases into account and prepares an output schedule. Your program should be intuitive and user friendly. It should be robust to user errors, such as providing nonsense input - the program ought to guide how to provide the data correctly and foresee alternative path rather than calculate nonsense output.

When asking for interest rate, user should provide it in yearly terms. The program should compute the appropriate rate based on frequency. Assume **only** the following frequencies: yearly, semi-annually, quarterly, monthly. Having this, and loan start date, you can build schedule that provides specific date - month and year will be sufficient, you don't have to provide full date - rather than an ordinal number of a period. For instance:

yearly rate	R	5.0%	

#	date	opening balance	principal	interest rate	interest	instalment	closing balance
i	t_i	PV_{i-1}	P_i	r_i	Int_i	PMT_i	PV_i
1	06/2019	4 000.00	2 000.00	2.5%	100.00	2 100.00	2 000.00
2	12/2019	2 000.00	2 000.00	2.5%	50.00	2 050.00	0.00
		TOTAL	4 000.00				

Please do pay attention to programming practices. Avoid duplicating the code. Factor out small procedures / functions instead of repeating the code. Use meaningful and self-explanatory names. Good code speaks for itself and does not need comments.

3 Rules

The task must be done in a **team of 4**. Any exceptions from that rule will be made on a case by case basis.

3.1 Submitting

Please when submitting, send me an e-mail with the following information:

- name, surname, master track, student id of each team member
- all team members should be on carbon copy
- link to **GitHub** repository containing the project, this is the only means I accept it

Note that GitHubGist is a way for quick share of pieces of code and notes, and is **not** equivalent to GitHub repository. Thus, it will **not** be accepted as means of project submission.

3.2 Good programming practices

Remember about good programming practices, in particular:

- **names** - name modules, classes, methods, functions, variables so that everybody can understand what a given object is or what it does
- **indents** - use a certain indent convention, that is clear and helps you to understand the code structure at one quick glance
- **avoid duplicating** - if you use a certain piece of instruction in many places, it is better to extract it to separate place, and then call it multiple times

3.3 Deadline

Ultimate deadline **is Sunday, 2019-05-19 at 23:59 UTC+2:00 (Paris time)**. Every day of delay decreases the score of the project by 1p.

3.4 Grading

Remember that this project, except for a few special cases, accounts of 50% of the final grade.

The grade for the project will consist of:

- [**50%**] **correctness of computations** - the program considers all the cases and provides correct output for correct input
- [**25%**] **user friendliness, robustness** - the program is user friendly, so i.a. guides user in providing correct data and deals with cases of irrational input
- [**25%**] **clean code** - application of good programming practices and uses optimal methods to provide the solution