LP Project

Alan L. Sampaolo, Federico Crivellari, Paolo Rinaldi

24/07/2025

The LP problem

A home buyer can combine several mortgage loans to finance the purchase of a house. Given borrowing needs B and a horizon of T months to repay the loans, the home buyer would like to minimize his total cost (or equivalently, the monthly payment p made during each of the next T months). Regulations impose limits on the amount that can be borrowed from certain sources. There are n different loan opportunities available. The loan i has a fixed interest rate r_i, a length $T_i \leq T$, and a maximum amount borrowed b_i. The monthly payment on loan i is not required to be the same every month, but a minimum payment m_i is required each month. However, the total monthly payment for all loans is constant.

Requirement

Formulate a linear program to determine a combination of loans that minimizes the buyer's borrowing cost. Make reasonable assumptions regarding the data used and provide justifications for them. Test your model with different data sets and comment on the results. Perform a sensitivity analysis and discuss the most relevant findings. Optional: Formulate and solve the dual problem. Verify the complementary slackness conditions.

Assumed Data

- Total amount to finance: $B = 500{,}000$ €
- Planning horizon: $T = 240$ months

Loan	Maturity (months)	Interest rate (monthly)	Max amount (€)	Min payment (€)
A	240	0.00267	250,000	600
B	180	0.00250	150,000	500
C	120	0.00233	75,000	450
D	60	0.00217	50,000	400

Table 1: Data set used

Mathematical Formulation (Primal LP)

Parameters

- $T = 240$: total planning horizon (months)
- $B = 500{,}000$: total amount to finance
- $\mathcal{I} = \{A, B, C, D\}$: set of available loan options
- For each $i \in \mathcal{I}$:
  - r_i: monthly interest rate
  - T_i: loan duration in months
  - b_i: maximum borrowable amount (plafond of the loan)
  - m_i: minimum monthly payment

Decision Variables

- $x_{ti} \geq m_i$: monthly payment on loan i in month t
- $u_{ti} \geq 0$: outstanding principal on loan i at month t
- $p \geq 0$: constant total monthly payment

Objective Function

$$\min_{x_{ti}} \; p$$

Constraints

$$\sum_{i \in \mathcal{I}: t \leq T_i} x_{ti} = k \qquad \forall t = 1, \ldots, T \quad \text{(constant monthly installment)}$$

$$u_{0i} \leq b_i \qquad \forall i \in \mathcal{I} \quad \text{(maximum loan availability)}$$

$$u_{ti} = (1 + r_i) u_{t-1,i} - x_{ti} \qquad \forall i \in \mathcal{I},\ t = 1, \ldots, T_i \quad \text{(residual capital evolution)}$$

$$u_{T_i,i} = 0 \qquad \forall i \in \mathcal{I} \quad \text{(fully repaid by maturity)}$$

$$\sum_{i \in \mathcal{I}} u_{0i} = B \qquad \text{(total borrowed equals need)}$$

$$x_{ti} \geq m_i \qquad \text{(minimum installment per month)}$$

Variable Domains

$$\begin{cases} x_{ti} \geq m_i & \forall i \in \mathcal{I},\ t = 1, \ldots, T_i \\ u_{ti} \geq 0 & \forall i \in \mathcal{I},\ t = 0, \ldots, T_i \\ p \geq 0 \end{cases}$$

Implementation in Python with Gurobi

To solve the optimization problem numerically, we implemented the model using the `gurobipy` library in Python.
The code starts with the definition of the parameters: in our notebook we saw results for different datasets, while in this print you can see the model which uses the parameter cited above.
Subsequently, we constructed the model with indexed variables and loops, in order to consider every month for each of the four avaiable loans.
At this point we added the six constraints we saw above, set the objective function with the goal of minimize the constant monthly installment, which as a result will minimize the borrower's total cost, and solved the problem using the Gurobi optimizer.

Python Code

```
#Variables

T = 240  #Total lenght in months (20 years)
B = 500_000  #Total amount to finance

#Avaible loans

loans = ["A", "B", "C", "D"]
r = {"A": 0.00267, "B": 0.00250, "C": 0.00233, "D": 0.00217}
T_i = {"A": 240, "B": 180, "C": 120, "D": 60}
b_i = {"A": 250_000, "B": 150_000, "C": 75_000, "D": 50_000}
m_i = {"A": 600, "B": 500, "C": 450, "D": 400}

#Model construction

model = Model("primal")

x = {}                                  #Monthly payments
u = {}                                  #Outstanding principal

for i in loans:
    for t in range(T_i[i] + 1):
        u[t, i] = model.addVar(lb=0)
    for t in range(1, T_i[i] + 1):
        x[t, i] = model.addVar()

k = model.addVar(lb=0)

#Constraints

#1 Constant monthly installment
for t in range(1, T + 1):
    model.addConstr(gp.quicksum(x[t, i] for i in loans if t <=
       ↪ T_i[i]) == k)
```

```
#2 Initial residual capital <= Maximum amount avaiable
for i in loans:
    model.addConstr(u[0, i] <= b_i[i])

#3 Residual capital evolution
for i in loans:
    for t in range(1, T_i[i] + 1):
        model.addConstr(u[t, i] == u[t - 1, i] * (1 + r[i]) -
            ↪ x[t, i])

#4 Final residual capital equal to 0
for i in loans:
    model.addConstr(u[T_i[i], i] == 0)

#5 Total amount financed equal to B
model.addConstr(gp.quicksum(u[0, i] for i in loans) == B)

#6 Minimum installment constraint x >= m_i
for i in loans:
    for t in range(1, T_i[i] + 1):
        model.addConstr(x[t, i] >= m_i[i])

#Objective function: minimize the constant monthly installment

model.setObjective(k, GRB.MINIMIZE)

#Optimize

model.optimize()

#Output

if model.status == GRB.OPTIMAL:
    print(f"\n Optimal monthly installment:    {k.X:.2f}")
    for i in loans:
        print(f"\nLoan {i}:")
        for t in range(1, T_i[i] + 1):
            print(f"  Month {t:3}: payment =    {x[t, i].X:.2f}")
else:
    print("No optimal solution found.")
```

Optimal Solution

The model successfully computes the optimal solution. The minimum constant monthly payment that allows the buyer to repay the required amount within the given constraints is given by:

Optimal monthly payment: **€ 2792.22**

This result is consistent with the constraints imposed on loan limits, interest rates, and

minimum payments. The model allocates the optimal mix of loans to minimize the financial burden over the repayment horizon.

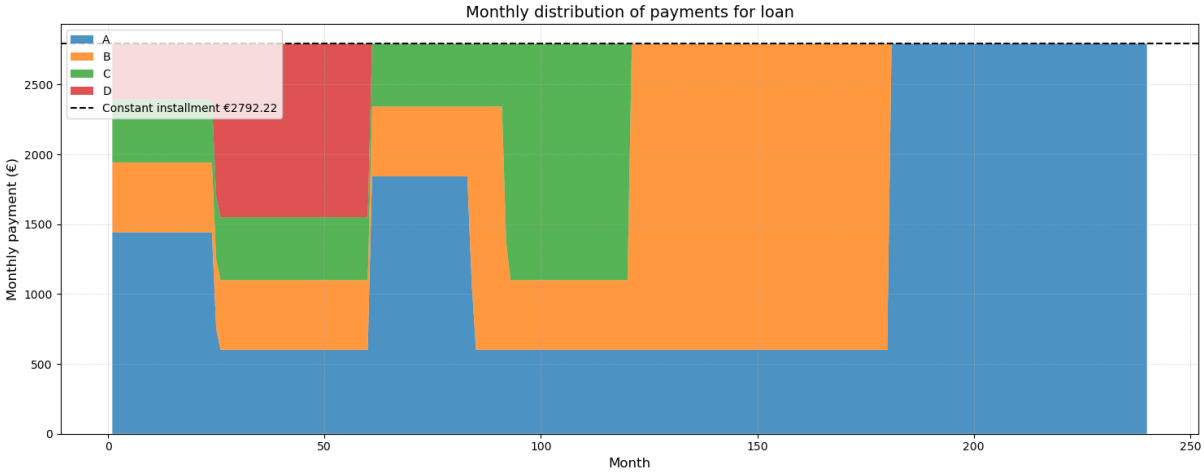

Figure 1: Monthly payments and outstanding principal over time

Standard form of the Primal problem

In order to continue our analysis, we implemented the standard form of our Primal problem. We introduced the slack variables s_b for the maximum loan availability constraint and s_m for the minimum installment per month constraint.
Then we rewrote the constraints in equality form, set the objective function with the same goal as before, and again solved the problem using the Gurobi optimizer.

Python Code

```
#Variables

T = 240  #Total lenght in months (20 years)
B = 500_000  #Total amount to finance

loans = ["A", "B", "C", "D"]
r  = {"A": 0.00267, "B": 0.00250, "C": 0.00233, "D": 0.00217}
T_i = {"A": 240, "B": 180, "C": 120, "D":  60}
b_i = {"A": 250_000, "B": 150_000, "C":  75_000, "D": 50_000}
m_i = {"A":     600, "B":     500, "C":     450, "D":    400}

#Model in standard form

model = Model("standard primal")

x = {}                                      #Monthly payments
u = {}                                      #Outstanding principal
s_b = {}                                    #Slack for constraint u0_i
   ↪      b_i
s_m = {}                                    #Slack for constraint x
   ↪      m_i
k = model.addVar()                          #Monthly installmant to
   ↪ minimize

#Creation of u_{t,i}
for i in loans:
    for t in range(T_i[i] + 1):
        u[t, i] = model.addVar()

#Creation of x_{t,i} and slack s_m[t,i]  (constraint x - s_m =
   ↪ m_i)
for i in loans:
    for t in range(1, T_i[i] + 1):
        x[t, i]   = model.addVar()
        s_m[t, i] = model.addVar()

#Slack variables for maximal loan
for i in loans:
    s_b[i] = model.addVar()

model.update()   #register all the variables
```

```
#Constraints (only in equality form)

#1 Constant monthly installment
for t in range(1, T + 1):
    model.addConstr(gp.quicksum(x[t, i] for i in loans if t <=
       ↪ T_i[i]) - k == 0)

#2 Initial residual capital + slack variable = avaiable amount
for i in loans:
    model.addConstr(u[0, i] + s_b[i] == b_i[i])

#3 Residual capital evolution
for i in loans:
    for t in range(1, T_i[i] + 1):
        model.addConstr(u[t, i] - (1 + r[i]) * u[t - 1, i] +
           ↪ x[t, i] == 0)

#4 Final residual capital = 0
for i in loans:
    model.addConstr(u[T_i[i], i] == 0)

#5 Total amount financed = B
model.addConstr(gp.quicksum(u[0, i] for i in loans) == B)

#6 Minimum installment constraint  x - s_m = m_i
for i in loans:
    for t in range(1, T_i[i] + 1):
        model.addConstr(x[t, i] - s_m[t, i] == m_i[i])

#Objective function: minimize

model.setObjective(k, GRB.MINIMIZE)

#Optimize

model.optimize()

#Output

if model.Status == GRB.OPTIMAL:
    print(f"\n Optimal monthly installment:    {k.X:.2f}")
    for i in loans:
        print(f"\nLoan {i}:")
        for t in range(1, T_i[i] + 1):
            print(f"  Month {t:3}: payment =    {x[t, i].X:.2f}")
else:
    print("No optimal solution found.")
```

Mathematical Formulation (Dual LP)

Dual Decision Variables

- $y1_t \in \mathbb{R}$: from constant-installment at month t, $t = 1, \ldots, T$.
- $y2_i \leq 0$: from $u_{0i} \leq b_i$, for each loan i.
- $y3_{i,t} \in \mathbb{R}$: from residual capital evolution at loan i, month t.
- $y4_i \leq 0$: from final balance constraint $u_{T_i,i} = 0$.
- $y5 \in \mathbb{R}$: from total financed constraint $\sum_i u_{0i} = B$.
- $y6_{i,t} \geq 0$: from minimum payment $x_{ti} \geq m_i$, for each i, $t = 1, \ldots, T_i$.

Dual Objective

$$\max \sum_{i \in \mathcal{I}} b_i \, y2_i \; + \; B \, y5 \; + \; \sum_{i \in \mathcal{I}} \sum_{t=1}^{T_i} m_i \, y6_{i,t}$$

Dual Constraints

$$\begin{aligned}
&(a) \quad x_{ti}\text{:} \quad y1_t + y3_{i,t} + y6_{i,t} \;\leq\; 0 \quad \forall i,\; t = 1, \ldots, T_i, \\
&(b) \quad k\text{:} \quad -\sum_{t=1}^{T} y1_t \;\leq\; 1, \\
&(c) \quad \text{from } s_m \text{ -vars:} \quad -\, y6_{i,t} \;\leq\; 0 \;\Rightarrow\; y6_{i,t} \geq 0, \\
&(d) \quad \text{from } s_b \text{ -vars:} \quad y2_i \;\leq\; 0, \\
&(e) \quad u_{0i}\text{:} \quad y2_i - (1 + r_i)\, y3_{i,1} + y5 \;\leq\; 0, \\
&(f) \quad u_{t,i},\; t = 1, \ldots, T_i - 1 : \quad y3_{i,t} - (1 + r_i)\, y3_{i,t+1} \;\leq\; 0, \\
&(g) \quad u_{T_i,i} : \quad y3_{i,T_i} + y4_i \;\leq\; 0.
\end{aligned}$$

Dual Domains

$$\begin{cases}
y1_t,\; y3_{i,t},\; y5 & \text{free (unrestricted)} \\
y2_i,\; y4_i \leq 0 & \\
y6_{i,t} \geq 0 &
\end{cases}$$

Implementation of the Dual in Python with Gurobi

Since we found the optimal solution with the Primal we continued our analysis by introducing the Dual function in order to show that our result is truly optimal, by means of the Strong Duality Theorem.
We constructed the model by declaring the Dual variables by setting the objective function we saw above: the economic interpretation in this case is to maximize the parameters' shadow values, respectively the loan caps, the total financing and the minimum payments required.
Subsequently we formulated the constraints in accordance with the rule

$$A^{\mathsf{T}}y \ \leq \ c$$

as you can see in the mathematical formulation. In particular, we started with the primal in standard form and created:

1. Each component of the y vector from the constraints of the standard primal problem (one for each constraint)
2. The constraints of the dual problem looking the number of variables that we had in the primal problem (one for each variable)
3. Each component of the y vector, generated by an equality constraint in the primal problem, has an unbounded domain
4. Each component of the vector y, generated by an inequality from the primal problem, has restriction in terms of sign

The dual objective function is created using the formulation

$$b^{\mathsf{T}}y$$

where the vector b is considered the known variables vector of the standard primal problem. You can notice, that some known variables are represented in the LHS of the constraints of the standard primal problem in order to simplify the formulation of the dual problem. In this case, the vector b presents some components equal to 0, except: b_i, B and m_i. Looking the constraints of the standard primal problem we can say that the constraints of the dual problem are created:

1. Choose the variable that generates the dual constraint
2. Since the components of y vector are associated to each constraint of the standard primal problem, it was necessary look in which constraints appear the chosen variable of the primal problem
3. Select the appropriate components of y to formulate the dual constraint
4. Look the coefficient in front of the chosen variable to create the associated dual constraint. The coefficient in front of it for each constraint in the standard primal problem became the coefficient of the corresponding y variable.

Example: consider the variable x_{ti}. it appears in the: first, third and sixth constraint in the standard primal problem, then the dual constraint related to this variable will be formed by $y1_t$, $y3_{i,t}$ and $y6_{i,t}$. In each constraint of the standard primal problem the variable x_{ti} has coefficient (that is an element of the matrix A) equal to 1, then also the y variables mentioned before have coefficient equal to 1. Then we obtain:

$$(a) \quad y1_t + y3_{i,t} + y6_{i,t}$$

Since the objective function in the standard primal problem has a vector c composed by zeroes, except in correspondence of k, the coefficient assigned in the objective function in the standard primal problem to x_{ti} is zero, the in the RHS of the dual constraint we have to put 0, obtaining:

$$(a) \quad y1_t + y3_{i,t} + y6_{i,t} \leq 0$$

After these passages, we solved the problem using the Gurobi optimizer.

Python Code

```
#Variables
T  = 240
B  = 500_000

loans = ["A", "B", "C", "D"]
r     = {"A": 0.00267, "B": 0.00250, "C": 0.00233, "D": 0.00217}
T_i   = {"A": 240, "B": 180, "C": 120, "D":  60}
b_i   = {"A": 250_000, "B": 150_000, "C":  75_000, "D": 50_000}
m_i   = {"A":     600, "B":     500, "C":     450, "D":    400}

#Dual model construction
dual = Model("dual")

y1 = {t: dual.addVar(lb=-GRB.INFINITY)              #From
   ↪ constant installment constraint
      for t in range(1, T+1)}

y2 = {i: dual.addVar(lb=-GRB.INFINITY)              #From u0_i +
   ↪ s_b = b_i constraint
      for i in loans}

y3 = {(i, t): dual.addVar(lb=-GRB.INFINITY)         #From
   ↪ residual evolution constraint
      for i in loans for t in range(1, T_i[i]+1)}

y4 = {i: dual.addVar(lb=-GRB.INFINITY)              #From u_Ti,i
   ↪ = 0 constraint
      for i in loans}

y5 = dual.addVar(lb=-GRB.INFINITY)                  #From initial
   ↪ residual capital constraint
```

```
y6 = {(i, t): dual.addVar(lb=-GRB.INFINITY)        #From x - s_m
   ↪ = m_i constraint
      for i in loans for t in range(1, T_i[i]+1)}
dual.update()

#Objective function: maximize
dual.setObjective(
      gp.quicksum(b_i[i] * y2[i] for i in loans)
    + B * y5
    + gp.quicksum(m_i[i] * y6[i, t] for i in loans for t in
       ↪ range(1, T_i[i]+1)),
    GRB.MAXIMIZE)

#Dual constraints

#(a) Variables x[t,i]
for i in loans:
    for t in range(1, T_i[i]+1):dual.addConstr(y1[t] + y3[i, t]
       ↪ + y6[i, t] <= 0)

#(b) Variable k
dual.addConstr(-gp.quicksum(y1[t] for t in range(1, T+1)) <= 1)

#(c) Variables s_m[t,i]
for i in loans:
    for t in range(1, T_i[i]+1):dual.addConstr(-y6[i, t] <= 0)

#(d) Variables s_b[i]
for i in loans:
    dual.addConstr(y2[i] <= 0)

#(e) Variables u[0,i]
for i in loans:
    dual.addConstr(y2[i] - (1 + r[i]) * y3[i, 1] + y5 <= 0)

#(f) Varables u[t,i]
for i in loans:
    for t in range(1, T_i[i]):dual.addConstr(y3[i, t] - (1 +
       ↪ r[i]) * y3[i, t+1] <= 0)

#(g) Variables u[T_i,i]
for i in loans:
    dual.addConstr(y3[i, T_i[i]] + y4[i] <= 0)

#Optimize
dual.optimize()

#Output
if dual.Status == GRB.OPTIMAL:
    print(f"\nOptimal value for Dual:    {dual.ObjVal:.2f}")
else:
```

```
    print("No optimal solution found for Dual.")
```

Confirmation of the Optimal Solution

As optimal result from our Dual function we had the same value as we got from the Primal:

Optimal monthly payment: **€ 2792.22**

This is the confirmation that our result is truly optimal by means of Strong Duality Theorem.

Complementary slackness

In order to verify that in the primal problem and the dual problem, we had found the optimal solution, it is necessary to verify the complementary slackness condition.
In order to verify it we deduced the slack variables of the dual problem from the its constraints and moltiplied them for the decisions and slack variables of the primal problem, obtaining in almost all cases a product very near to zero (example e-15). We obtained also products that weren't near to zero, in total 4, for them we checked if one of the factor of the moltiplication was at least near to zero with a tollerance e-7.

Python Code: Complementary slackness

```
#(a) Variables x[t,i]
for i in loans:
    for t in range(1, T_i[i]+1):
      if (y1[t].X + y3[i, t].X + y6[i, t].X - 0)*x[t,i].X != 0:
        print('Violation of slackness linked to constraint (a),
           ↪ loan ', i, ' time ', t, ' quantity ', (y1[t].X +
           ↪ y3[i, t].X + y6[i, t].X)*x[t,i].X)

#(b) Variables k
for i in loans:
    for t in range(1, T_i[i]+1):
        if (1 + y1[t].X)*k.X != 0:
            print('Violation of slackness linked to constraint
               ↪ (b), loan ', i, ' time ', t, ' quantity ',
               ↪ (y1[t].X + y3[i, t].X + y6[i, t].X)*x[t,i].X)

#(c) Variables s_m[t,i]
for i in loans:
    for t in range(1, T_i[i]+1):
        if (0+y6[i, t].X)*s_m[t,i].X != 0:
            print('Violation of slackness linked to constraint
               ↪ (c), loan ', i, ' time ', t, ' quantity ',
               ↪ (0+y6[i, t].X)*s_m[t,i].X)

#(d) Variables s_b[i]
for i in loans:
      if (0-y2[i].X)*s_b[i].X != 0:
        print('Violation of slackness linked to constraint (d),
           ↪ loan ', i,' quantity ', (0-y2[i].X)*s_b[i].X)

#(e) Variables u[0,i]
for i in loans:
    if (0-y2[i].X+(1 + r[i]) * y3[i, 1].X-y5.X)*u[0,i].X != 0:
        print('Violation of slackness linked to constraint (c),
           ↪ loan ', i,' quantity ', (0-y2[i].X+(1 + r[i]) *
           ↪ y3[i, 1].X-y5.X)*u[0,i].X)

#(f) Variables u[t,i]
```

```
for i in loans:
    for t in range(1, T_i[i]):
        if (0 - y3[i, t].X + (1 + r[i]) * y3[i, t + 1].X) * u[t,
           ↪ i].X != 0:
            print('Violation of slackness linked to constraint
               ↪ (f), loan ', i, ' time ', t,
                   ' quantity ', (0 - y3[i, t].X + (1 + r[i]) *
                      ↪ y3[i, t + 1].X) * u[t, i].X)

#(g) Variables u[T_i,i]
for i in loans:
    if (0 - y3[i, T_i[i]].X + y4[i].X) * u[T_i[i], i].X != 0:
        print('Violation of slackness linked to constraint (g),
           ↪ loan ', i,' quantity ', (0 - y3[i, T_i[i]].X +
           ↪ y4[i].X) * u[T_i[i], i].X)
```

Python Code: Check slackness violation in constrain (c)

```
print(f"s_m[84, B] = {s_m[84, 'B'].X}")
print(f"dual slack[84, B] = {y6['B', 84].X}")
```

Python Code: Check slackness violation in constrain (f)

```
print(f"u[2, 'A'] = {u[2, 'A'].X}")
print(f"dual slack[2, 'A'] = {-y3['A', 2].X + (1 + r['A']) *
   ↪ y3['A', 3].X}")

print(f"u[4, 'A'] = {u[4, 'A'].X}")
print(f"dual slack[4, 'A'] = {-y3['A', 4].X + (1 + r['A']) *
   ↪ y3['A', 5].X}")

print(f"u[18, 'A'] = {u[18, 'A'].X}")
print(f"dual slack[18, 'A'] = {-y3['A', 18].X + (1 + r['A']) *
   ↪ y3['A', 19].X}")
```

Sensitivity analysis

In order to continue our analysis, it is interesting see how the optimal solution changes if we change the parameters of the model. We performed two kind of analysis:

1. Sensitivity analysis: we created three different scenarios assuming the change of only one parameter, maintaining the others as they are presented in the initial primal problem (Page 1);
2. Sensitivity report.

Scenario 1: growth of all interest rates by 0.5%

The model successfully computes the optimal solution. The minimum constant monthly payment that allows the buyer to repay the required amount within the given constraints is given by:

Optimal monthly payment: **€ 2918.85**

This result show that a very little shock of a macroeconomic condition, as an increase of interest rates, produce a very different optimal solution from the previous one.
This shows that the model reacts a lot if the interest rates change by a very little quantity.

Scenario 2: the financial plafond of the cheapest loan (D) is increased by 50k

You can notice that in the primal problem the loan D represents the cheapest one because it has small maturity and low interest rate (compared with the others). So, it is interesting understand what is the optimal solution of the model if the plafond of loan D is increased by 50k. The model successfully computes the optimal solution. The minimum constant monthly payment that allows the buyer to repay the required amount within the given constraints is given by:

Optimal monthly payment: **€ 2791.53**

This result shows that even if the plafond of the cheapest loan is increased the optimal solution doesn't change a lot.
This shows that the model is only minimally sensitive to changes in the loan plafond.

Scenario 3: disapper loan C and loans A and B offer larger plafonds

For this scenario we tried to change a bit more the loans. We assumed that the the loan C is not more avaiable while the plafonds of loans A and B are increased (this to show how the model reacts to a little more strong perturbations in the loans parameters). The model successfully computes the optimal solution. The minimum constant monthly payment that allows the buyer to repay the required amount within the given constraints is given by:

Optimal monthly payment: **€ 2799.72**

This result shows that if some conditions on the loans changes the optimal solution doesn't change a lot.
This shows that the model is only minimally sensitive (more than in the previous case) to changes in the loan conditions.

Change of Dataset

Finally we performed the initial analysis but using a different dataset, in order to understand how our model behaves when changing the borrowing need B.
In this case we considered:

Assumed Data

- Total amount to finance: $B = 300{,}000$ €
- Planning horizon: $T = 240$ months

Loan	Maturity (months)	Interest rate (monthly)	Max amount (€)	Min payment (€)
A	240	0.00417	200,000	700
B	180	0.00375	100,000	600
C	120	0.00333	75,000	500

Table 2: Change of Dataset

As you can see from the notebook, in this case our result was given by:

Optimal monthly payment: **€ 1935.87**

Comparison monthly payments

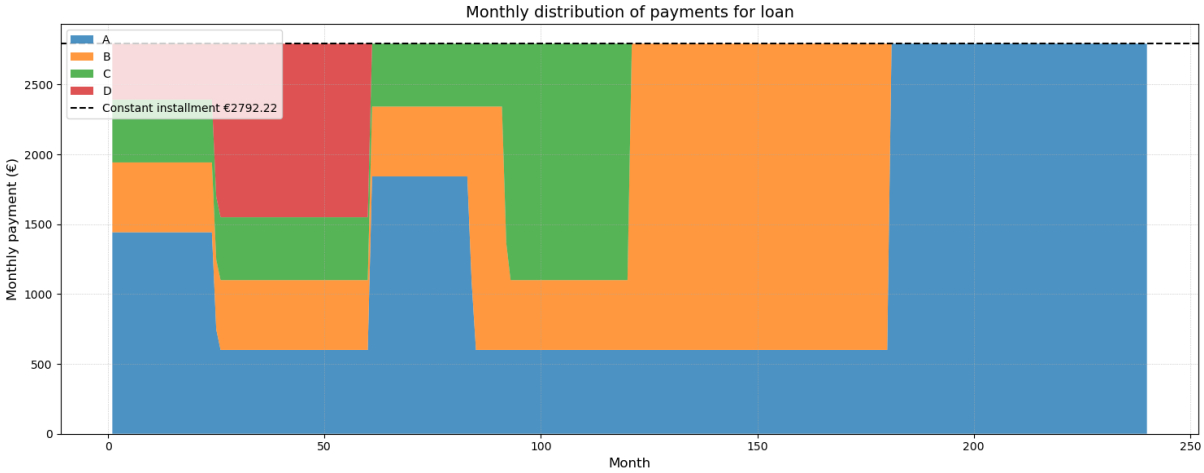

Figure 2: Monthly payments over time – initial problem

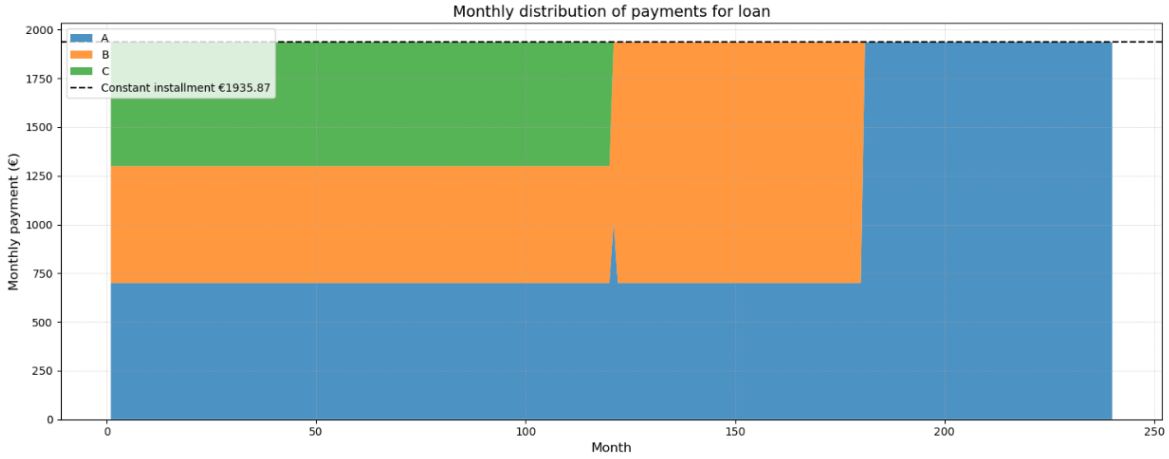

Figure 3: Monthly payments over time – change of dataset

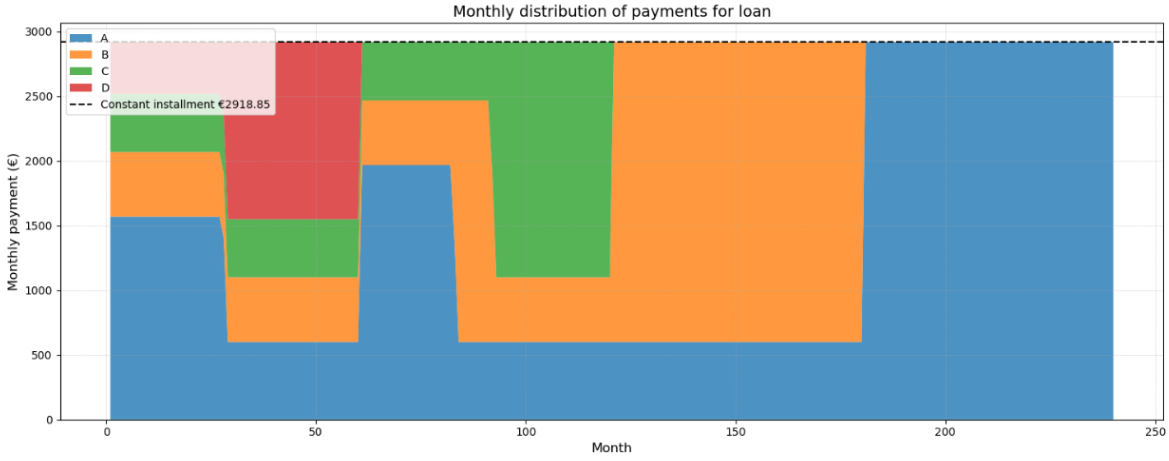

Figure 4: Monthly payments over time – scenario 1: growth of all interest rates by 0.5%

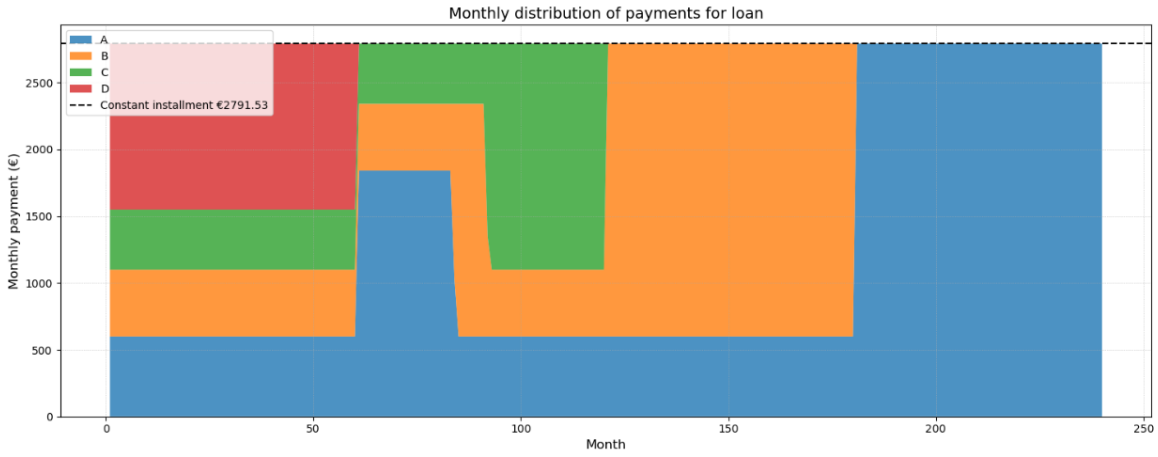

Figure 5: Monthly payments over time – scenario 2: the financial plafond of the cheapest loan (D) is increased by 50k

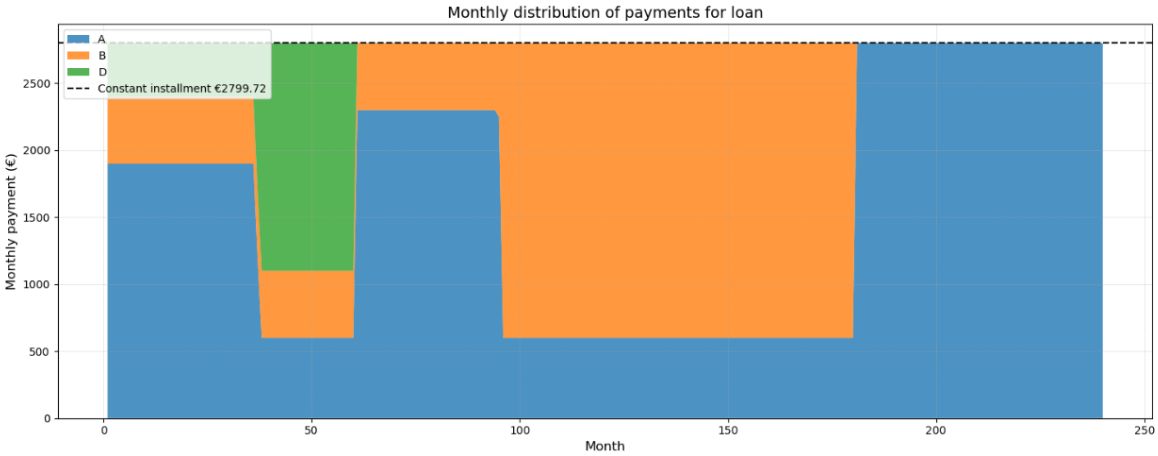

Figure 6: Monthly payments over time – scenario 3: disapper loan C and loans A and B offer larger plafonds

Sensitivity Report

The **Reduced Cost** quantifies how much the contribution of a currently non-basic variable must change to become worthwhile in the solution.

The **Shadow Price** of a constraint represents the rate of change in the optimal objective function value with respect to a one-unit increase in the RHS of the constraint.

```
Variables with reduced cost != 0
Empty DataFrame
Columns: [Variable, Reduced Cost]
Index: []
```

Figure 7: Output reduced cost

```
Constraints with shadow price != 0
                          Constraint  Shadow Price
847                    total_financed      0.005621
0    constant_monthly_installment[1]     -0.005606
243           capital_evolution[1,A]      0.005606
244           capital_evolution[2,A]      0.005591
```

Figure 8: Output shadow prices

- **All variables** have a reduced cost equal to zero, which means that all variables in the model are currently active in the optimal solution.
- **Constraint** `total_financed` (defined as `gp.quicksum(u[0, i] for i in loans) == B`): increasing the RHS B by one unit leads to a decrease of approximately **0.005621** in the optimal monthly installment.
- **Constraint** `constant_monthly_installment[1]` (defined as `gp.quicksum(x[t, i] for i in loans if t <= T_i[i]) == k` for $t = 1$):
  increasing the RHS by one unit leads to an increase of approximately **0.005606** in the optimal monthly installment.
- **Constraint** `capital_evolution[1,A]` (defined as `u[t, i] == u[t - 1, i] * (1 + r[i]) - x[t, i]` for $t = 1$, $i = A$): increasing the RHS by one unit leads to a decrease of approximately **0.005606** in the optimal monthly installment.
- **Constraint** `capital_evolution[2,A]` (defined as `u[t, i] == u[t - 1, i] * (1 + r[i]) - x[t, i]` for $t = 2$, $i = A$): increasing the RHS by one unit leads to a decrease of approximately **0.005591** in the optimal monthly installment.

Python Code: Implementation of the sensitivity report

```
#Variables
T = 240  #Total length in months (20 years)
B = 500_000  #Total amount to finance

#Available loans
loans = ["A", "B", "C", "D"]
r = {"A": 0.00267, "B": 0.00250, "C": 0.00233, "D": 0.00217}
T_i = {"A": 240, "B": 180, "C": 120, "D": 60}
b_i = {"A": 250_000, "B": 150_000, "C": 75_000, "D": 50_000}
m_i = {"A": 600, "B": 500, "C": 450, "D": 400}

#Model construction
model = Model("primal")

x = {}                                  #Monthly payments
u = {}                                  #Outstanding principal

for i in loans:
```

```
    for t in range(T_i[i] + 1):
        u[t, i] = model.addVar(lb=0, name=f"u[{t},{i}]")
    for t in range(1, T_i[i] + 1):
        x[t, i] = model.addVar(name=f"x[{t},{i}]")

k = model.addVar(lb=0, name="k")

#Constraints

#1 Constant monthly installment
for t in range(1, T + 1):
    model.addConstr(
        gp.quicksum(x[t, i] for i in loans if t <= T_i[i]) == k,
           ↪ name=f"constant_monthly_installment[{t}]")

#2 Initial residual capital <= Maximum available amount
for i in loans:
    model.addConstr(u[0, i] <= b_i[i],
       ↪ name=f"initial_capital_limit[{i}]")

#3 Residual capital evolution
for i in loans:
    for t in range(1, T_i[i] + 1):
        model.addConstr(
            u[t, i] == u[t - 1, i] * (1 + r[i]) - x[t, i],
               ↪ name=f"capital_evolution[{t},{i}]")

#4 Final residual capital equal to 0
for i in loans:
    model.addConstr(u[T_i[i], i] == 0,
       ↪ name=f"final_capital_zero[{i}]")

#5 Total amount financed equal to B
model.addConstr(
    gp.quicksum(u[0, i] for i in loans) == B,
       ↪ name="total_financed")

#6 Minimum installment constraint x >= m_i
for i in loans:
    for t in range(1, T_i[i] + 1):
        model.addConstr(
            x[t, i] >= m_i[i], name=f"MinInstallment[{t},{i}]")

#Objective function: minimize the constant monthly installment
model.setObjective(k, GRB.MINIMIZE)

#Optimize
model.optimize()

#Output
if model.status == GRB.OPTIMAL:
```

```
    print(f"\nOptimal monthly installment:    {k.X:.2f}")
    for i in loans:
        print(f"\nLoan {i}:")
        for t in range(1, T_i[i] + 1):
            print(f"  Month {t:3}: payment =    {x[t, i].X:.2f}")
else:
    print("No optimal solution found.")
```

Python Code: Implementation of the sensitivity report

```
if model.status == GRB.OPTIMAL:

    var_data = []
    for var in model.getVars():
        if var.RC != 0:
            allow_inc = var.getAttr("SAObjUp")
            allow_dec = var.getAttr("SAObjLow")
            var_data.append([var.VarName, var.RC, allow_inc,
               ↪ allow_dec])

    var_df = pd.DataFrame(var_data, columns=["Variable",
       ↪ "Reduced Cost", "Allowable Increase", "Allowable
       ↪ Decrease"])
    var_df = var_df.reindex(var_df["Reduced
       ↪ Cost"].abs().sort_values(ascending=False).index)

    constr_data = []
    for constr in model.getConstrs():
        pi = constr.getAttr("Pi")
        if pi != 0:
            allow_inc = constr.getAttr("SARHSLow")
            allow_dec = constr.getAttr("SARHSUp")
            constr_data.append([constr.ConstrName, pi,
               ↪ allow_inc, allow_dec])

    constr_df = pd.DataFrame(constr_data, columns=["Constraint",
       ↪ "Shadow Price", "Allowable Increase", "Allowable
       ↪ Decrease"])
    constr_df = constr_df.reindex(constr_df["Shadow
       ↪ Price"].abs().sort_values(ascending=False).index)

    print("\nVariables with reduced cost != 0")
    print(var_df.head(4).round(6))

    print("\nConstraints with shadow price != 0")
    print(constr_df.head(4).round(6))
```