Precificação de empréstimos com matriz de transição

Walter C. Neto

29 de dezembro de 2025

Resumo

Este documento complementa minha própria dissertação de mestrado, incorporando explicitamente uma **matriz de transição de estados de rating (Markov)** ao procedimento de precificação impondo que o processo de geração do contrato deve igualar o custo do risco da operação, de forma que a taxa de juros seja ajustada ao risco, seguindo a lógica da taxa mínima que ainda justifica correr o risco de crédito daquela operação a ser precificada.

Sumário

1 Visão geral do procedimento

Esta metodologia segue o arcabouço desenvolvido na dissertação de mestrado do mesmo autor.[1]. Dessa forma, temos uma componente dada pelo fluxo financeiro comparada ao spread justo levando-se em conta a perda esperada, inserpedada e capital alocado:

- **RHS (determinístico dado o risco e premissas)**: representa o **custo econômico esperado** de originar e carregar a operação, composto por:
  1. **FTP** (custo de funding/transferência interna), calculado como juros sobre o saldo performando;
  2. **OPEX** (custo operacional), com componente fixo mensal e componente variável proporcional ao saldo;
  3. **EL** (perda esperada), como impacto ao **entrar em default**, via LGD $\times$ saldo;
  4. **(Opcional) CoC/Capital**: encargo de capital econômico via EC $\times$ CoC, aplicado ao saldo performando.
- **LHS (função objetiva da taxa)**: representa o **valor presente dos juros contratuais esperados**.

A taxa contratual anual r é encontrada ao resolver:

$$\mathrm{PV}_{\mathrm{LHS}}(r) \;=\; \mathrm{PV}_{\mathrm{RHS}}, \qquad \text{isto é,}\;\; f(r) \equiv \mathrm{PV}_{\mathrm{LHS}}(r) - \mathrm{PV}_{\mathrm{RHS}} = 0. \tag{1}$$

2 Proposta: Matriz de Transição (Markov) no pricing

2.1 Estados e dinâmica mensal

O código introduz um processo de Markov mensal sobre o conjunto de estados:

$$\mathcal{S} = \{\mathrm{AAA}, \mathrm{AA}, \mathrm{A}, \mathrm{BBB}, \mathrm{BB}, \mathrm{B}, \mathrm{CCC}, \mathrm{D}, \mathrm{PP}\},$$

onde:

- **D** = default (estado absorvente);
- **PP** = prepayment (estado absorvente).

A matriz mensal de transição é P, com $P_{ij} = \mathbb{P}(S_{t+1} = j \mid S_t = i)$. Para cada linha, vale $\sum_j P_{ij} = 1$.

[1] Correa Neto, W., *Precificação de Crédito Consignado INSS*, Dissertação de Mestrado, Fundação Getulio Vargas (FGV EAESP), São Paulo, 2025.

2.2 Propagação da distribuição de estados

Seja v_0 o vetor linha de probabilidades no tempo $t = 0$, concentrado no rating inicial:

$$v_0 = e_{s_0},$$

onde s_0 é o rating inicial e e_{s_0} é o vetor unitário.

A distribuição ao longo do tempo é obtida por:

$$v_t = v_0 P^t, \qquad t = 0, 1, \ldots, T.$$

2.3 Probabilidade de sobrevivência (performing) e eventos de default ou prepagamento

Define-se a probabilidade de estar **performing** (isto é, não estar em estados absorventes) no mês t como:

$$p_{\text{surv}}(t) = 1 - v_t(\text{D}) - v_t(\text{PP}). \tag{2}$$

Além disso, o modelo calcula a probabilidade de **entrar em default** no mês t como o incremento na massa de probabilidade do estado absorvente:

$$p_{\text{def-in}}(t) = v_t(\text{D}) - v_{t-1}(\text{D}), \qquad t \geq 1. \tag{3}$$

Análogo para prepayment:

$$p_{\text{pp-in}}(t) = v_t(\text{PP}) - v_{t-1}(\text{PP}), \qquad t \geq 1. \tag{4}$$

Interpretação econômica:

- juros contratuais e custos de carregamento (FTP, OPEX, capital) ocorrem apenas se a operação estiver **performando**, por isso são multiplicados por $p_{\text{surv}}(t)$;
- a perda esperada é modelada como um choque quando ocorre a **entrada em default** no mês t, por isso é multiplicada por $p_{\text{def-in}}(t)$.

3 Cronograma de pagamentos previstos do contrato

Dado principal N, prazo T meses e taxa anual r, o código converte a taxa para mensal efetiva:

$$i = (1 + r)^{1/12} - 1.$$

3.1 Sistema PRICE

Para PRICE, calcula-se a prestação fixa (PMT):

$$\text{PMT} = N \cdot \frac{i(1+i)^T}{(1+i)^T - 1}.$$

Em cada mês:

$$\text{Juros}_t = \text{Saldo}_{t-1} \cdot i, \qquad \text{Amort}_t = \text{PMT} - \text{Juros}_t, \qquad \text{Saldo}_t = \text{Saldo}_{t-1} - \text{Amort}_t.$$

3.2 Sistema SAC

Para SAC, a amortização é constante N/T e a prestação decresce ao longo do tempo.

Ponto central: o cronograma (saldo e juros contratuais) é conhecido dado r. O Markov entra como ponderador probabilístico para transformar esses fluxos determinísticos em **fluxos esperados**.

4 Definição do LHS e RHS (mensal) com Markov

Seja:

- Saldo_t: saldo no início do mês t (do cronograma);
- $\text{Juros}_t(r)$: juros contratuais do mês t (depende de r);
- $DF(t)$: fator de desconto no mês t, usando taxa base de desconto anual d (no código: `discount_base_aa`).

4.1 Desconto

No código, o desconto base anual d é convertido para mensal d_m e:

$$DF(t) = \frac{1}{(1+d_m)^t}.$$

4.2 LHS: juros contratuais esperados

O fluxo mensal do LHS é:

$$\text{CF}_t^{\text{LHS}}(r) = p_{\text{surv}}(t) \cdot \text{Juros}_t(r). \tag{5}$$

Logo, o valor presente do LHS:

$$\text{PV}_{\text{LHS}}(r) = \sum_{t=1}^{T} DF(t) \cdot \text{CF}_t^{\text{LHS}}(r).$$

4.3 RHS: custos esperados (FTP, OPEX, EL, e capital opcional)

Probabilidade de Default implícita via matriz de transição No modelo de precificação proposto, a probabilidade de default (PD) não é introduzida como um parâmetro exógeno e fixo por horizonte, mas emerge de forma endógena a partir de uma matriz de transição de estados do tipo Markov, com estado absorvente de default (D). A dinâmica de crédito do contrato é modelada por uma cadeia de Markov discreta, em que o vetor de probabilidades de estado no mês t, denotado por $\boldsymbol{\pi}_t$, evolui segundo

$$\boldsymbol{\pi}_{t+1} = \boldsymbol{\pi}_t \mathbf{P},$$

em que $\mathbf{P}$ representa a matriz de transição mensal entre ratings e estados absorventes.

A probabilidade acumulada de default até o mês t é dada pela massa de probabilidade no estado absorvente D,

$$\Pr(\tau \leq t) = \pi_t^{(D)},$$

enquanto a probabilidade marginal de ocorrência de default no mês t é obtida pela variação dessa massa,

$$\Pr(\tau \in (t-1, t]) = \pi_t^{(D)} - \pi_{t-1}^{(D)}.$$

Essa quantidade corresponde à probabilidade de entrada em default no período e é utilizada diretamente no cálculo da perda esperada mensal.

No processo de precificação, a perda esperada no mês t é definida como

$$EL_t = \Pr(\tau \in (t-1, t]) \cdot LGD_t \cdot EAD_t,$$

em que LGD_t e EAD_t podem variar ao longo do tempo. Dessa forma, a PD está incorporada de maneira consistente à dinâmica estocástica de migração de crédito, dispensando a imposição de uma curva de PD exógena e garantindo coerência temporal entre risco de crédito, precificação e estrutura de estados do contrato.

FTP. O modelo propõe uma taxa anual de FTP:

$$\mathrm{FTP}_{aa} = \mathrm{FTP}_{base} + \mathrm{spread}_{FTP}(T),$$

converte para mensal FTP_m e calcula:

$$\mathrm{CF}_t^{\mathrm{FTP}} = p_{\mathrm{surv}}(t) \cdot \mathrm{Saldo}_t \cdot \mathrm{FTP}_m. \tag{6}$$

OPEX. OPEX possui componente fixo mensal c_0 e componente variável anual α aplicada ao saldo (convertida para mensal):

$$\mathrm{CF}_t^{\mathrm{OPEX}} = p_{\mathrm{surv}}(t) \cdot \Big(c_0 + \mathrm{Saldo}_t \cdot \frac{\alpha}{12} \Big). \tag{7}$$

EL (perda esperada). No mês t, a perda esperada é:

$$\mathrm{CF}_t^{\mathrm{EL}} = p_{\text{def-in}}(t) \cdot \mathrm{LGD}_t \cdot \mathrm{Saldo}_t. \tag{8}$$

Capital. Se `include_capital_target=True`, há encargo:

$$\mathrm{CF}_t^{\mathrm{CAP}} = p_{\mathrm{surv}}(t) \cdot \Big(\mathrm{EC\%}_t \cdot \mathrm{Saldo}_t \Big) \cdot \mathrm{CoC}_m. \tag{9}$$

RHS total. O fluxo total do RHS:

$$\mathrm{CF}_t^{\mathrm{RHS}} = \mathrm{CF}_t^{\mathrm{FTP}} + \mathrm{CF}_t^{\mathrm{OPEX}} + \mathrm{CF}_t^{\mathrm{EL}} + \mathbf{1}_{\mathrm{cap}} \mathrm{CF}_t^{\mathrm{CAP}}.$$

E:

$$\mathrm{PV}_{\mathrm{RHS}} = \sum_{t=1}^{T} DF(t) \cdot \mathrm{CF}_t^{\mathrm{RHS}}.$$

4.4 LGD e EC variam com o estado corrente

Em vez de fixar LGD e EC no rating inicial, o modelo calcula, em cada mês, a média **condicional ao conjunto de ratings performando**:

$$\mathbb{P}(S_t \in \mathcal{R}) = \sum_{s \in \mathcal{R}} v_t(s), \qquad \mathcal{R} = \{\mathrm{AAA}, \ldots, \mathrm{CCC}\}.$$

Então:

$$\mathrm{LGD}_t = \frac{\sum_{s \in \mathcal{R}} v_t(s) \cdot \mathrm{LGD}(s)}{\sum_{s \in \mathcal{R}} v_t(s)}. \tag{10}$$

E:

$$\mathrm{EC\%}_t = \frac{\sum_{s \in \mathcal{R}} v_t(s) \cdot \mathrm{EC\%}(s)}{\sum_{s \in \mathcal{R}} v_t(s)}. \tag{11}$$

Interpretação: o custo de capital e a severidade de perda variam com a distribuição de probabilidade do estado corrente, permitindo que migrações de rating ao longo do tempo afetem o RHS.

5 Resolução numérica: bracketing + bisseção

A taxa anual r é encontrada resolvendo $f(r) = 0$, onde $f(r) = \mathrm{PV}_{\mathrm{LHS}}(r) - \mathrm{PV}_{\mathrm{RHS}}$.

O código utiliza:

- **Bracketing**: expande o limite superior `hi` até encontrar uma troca de sinal em $f(\cdot)$. Isso elimina o erro observado para ratings mais arriscados (ex.: B, CCC) em que $f($`lo`$)$ e $f($`hi`$)$ podiam ter o mesmo sinal.
- **Bisseção**: método robusto, convergindo para a raiz dentro do intervalo com troca de sinal.

6 Saídas do modelo

6.1 Variáveis calculadas

Para cada rating inicial e para cada configuração do RHS, são exibidos:

- taxa encontrada (a.a.);
- objetivo $f(r)$ (deve ficar próximo de zero);
- $\mathrm{PV}_{\mathrm{LHS}}$ e $\mathrm{PV}_{\mathrm{RHS}}$;
- resumo anual com PVs e fluxos (além de `balance_avg`, `lgd_avg`, `ec_pct_avg`).

6.2 Exports em CSV

O procedimento `export_outputs()` grava automaticamente, para cada cenário:

- `<tag>_detalhe_mensal.csv`: detalhe mês a mês (probabilidades, LGD/EC esperados, fluxo LHS, fluxos RHS e PVs);
- `<tag>_resumo_anual.csv`: resumo anual consolidado.

7 Resultados

A tabela resume os resultados apresentados, comparando:

- **Coluna (1)**: `include_capital_target=True` (RHS inclui CoC/capital);
- **Coluna (2)**: `include_capital_target=False` (RHS apenas FTP+OPEX+EL).

Tabela 1: Taxa anual encontrada por rating (LHS=RHS) com Markov

Rating	**(1) tx a.a.**	**(2) tx a.a.**
AAA	16,31%	15,86%
AA	16,73%	16,11%
A	17,45%	16,59%
BBB	19,05%	17,80%
BB	22,60%	20,76%
B	30,34%	27,92%
CCC	62,88%	59,36%

Interpretação Quanto pior o rating inicial, maior a taxa necessária para que o valor presente dos juros esperados (LHS) cubra o conjunto de custos esperados do RHS. A inclusão do componente de capital no RHS aumenta a taxa (coluna 1 vs. 2), com efeito mais visível à medida que o EC esperado cresce nos estados de pior rating.

8 Observações

8.1 Estados absorventes (D e PP)

Como discutido, **D** e **PP** são estados absorventes. Assim:

- não há juros contratuais em D ou PP;
- custos de carregamento (FTP/OPEX/capital) são aplicados apenas enquanto a operação está ativa;
- a perda esperada é contabilizada no instante de **entrada em default** (via $p_{\text{def-in}}(t)$).

8.2 Interpretação do LHS

O LHS foi definido como **PV dos juros contratuais**. Isso aproxima o motor de taxa mínima a uma decomposição *margem vs. custos esperados*.

8.3 Limitações

- a matriz de transição é um exemplo e deve ser calibrada com dados empíricos;
- o tratamento de prepayment no código é como saída sem payoff adicional; se desejado, pode-se incluir um payoff de liquidação (saldo) no momento de PP;
- EC considerado é uma proxy por rating;

- Para refletir corretamente a precificação no seu Banco usando esse modelo, você precisa calcular a sua matriz de Markov medindo as rolagens observadas e eventualmente adotando uma para cada carteira, produto, ou grupo homogêneo de risco.

A Apêndice A: Código `PricingMarkovRaroc.py`

Para compilar localmente, certifique-se de mudar o caminho da pasta.

Listing 1: Arquivo `PricingMarkovRaroc.py` (inclusão direta do caminho local)

```
# -*- coding: utf-8 -*-
"""
Objetivo:
- LADO ESQUERDO (LHS): PV dos JUROS contratuais esperados.
- LADO DIREITO (RHS): PV dos custos esperados determinados pelo risco do cliente e premissas do Banco:
    - FTP (juros sobre saldo)
    - OPEX (fixo + vari vel sobre saldo)
    - EL (LGD * saldo quando entra em default)
    - (Opcional) CoC/Capital (EC * CoC) como componente no RHS
"""

from __future__ import annotations

import os
from dataclasses import dataclass
from typing import Callable, Dict, Literal, Tuple

import numpy as np
import pandas as pd

# ========================================
# 1) ESTADOS E MATRIZ DE TRANSI  O (mensal)
# ========================================

STATES = ["AAA", "AA", "A", "BBB", "BB", "B", "CCC", "D", "PP"]  # D=default, PP=prepayment
RATING_STATES = ["AAA", "AA", "A", "BBB", "BB", "B", "CCC"]
ABSORBING = ["D", "PP"]

TRANSITION_MATRIX_MONTHLY: pd.DataFrame = pd.DataFrame(
    data=[
        # AAA    AA      A      BBB     BB      B      CCC     D        PP
        [0.965, 0.030, 0.002, 0.000, 0.000, 0.000, 0.000, 0.0005, 0.0025],  # AAA
```

```
        [0.010, 0.960, 0.025, 0.001, 0.000, 0.000, 0.000, 0.0010, 0.0030],  # AA
        [0.000, 0.015, 0.955, 0.025, 0.002, 0.000, 0.000, 0.0015, 0.0015],  # A
        [0.000, 0.002, 0.020, 0.945, 0.025, 0.003, 0.000, 0.0030, 0.0020],  # BBB
        [0.000, 0.000, 0.003, 0.020, 0.935, 0.030, 0.005, 0.0050, 0.0020],  # BB
        [0.000, 0.000, 0.000, 0.004, 0.030, 0.920, 0.030, 0.0120, 0.0040],  # B
        [0.000, 0.000, 0.000, 0.000, 0.006, 0.040, 0.880, 0.0600, 0.0140],  # CCC
        [0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 1.0000, 0.0000],  # D
        [0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.0000, 1.0000],  # PP
    ],
    index=STATES,
    columns=STATES,
)

# checagem
for s in STATES:
    sm = float(TRANSITION_MATRIX_MONTHLY.loc[s].sum())
    if abs(sm - 1.0) > 1e-10:
        raise ValueError(f"Matriz de transi  o: linha {s} soma {sm}, deveria somar 1.")

# ====================================
# 2) PREMISSAS: LGD, OPEX, FTP, CAPITAL
# ====================================

# LGD por rating (usaremos por estado corrente)
LGD_BY_RATING: Dict[str, float] = {
    "AAA": 0.25,
    "AA": 0.30,
    "A": 0.35,
    "BBB": 0.40,
    "BB": 0.45,
    "B": 0.55,
    "CCC": 0.65,
}

# Capital econ mico como % do EAD por rating (usaremos por estado corrente)
def economic_capital_pct_ead_by_rating(rating: str) -> float:
    table = {
```

```
        "AAA": 0.02,
        "AA": 0.03,
        "A": 0.04,
        "BBB": 0.06,
        "BB": 0.09,
        "B": 0.12,
        "CCC": 0.16,
    }
    return table[rating]

# OPEX
OPERATING_COST_FIXED_MONTHLY = 30.0
OPERATING_COST_PCT_AA_ON_BALANCE = 0.008  # 0.8% a.a.

# FTP
FTP_BASE_AA = 0.12

def ftp_spread_term_aa(term_months: int) -> float:
    if term_months <= 12:
        return 0.010
    if term_months <= 24:
        return 0.015
    if term_months <= 36:
        return 0.020
    return 0.025

# Custo do capital (CoC)
COST_OF_CAPITAL_AA = 0.18

# Taxa base de desconto (proxy CDI/OIS)
DISCOUNT_BASE_AA = 0.12

# =============
# 3) ESTRUTURAS
# =============

@dataclass(frozen=True)
class LoanTerms:
    principal: float
    term_months: int
    amortization: str  # "PRICE" ou "SAC"
    start_rating: str  # rating inicial (estado inicial do Markov)

@dataclass(frozen=True)
class SolveResult:
    annual_rate: float
    objective_value: float
```

```
    pv_lhs: float
    pv_rhs: float
    details: pd.DataFrame
    annual_summary: pd.DataFrame

# =============
# 4) AUXILIARES
# =============

def aa_to_mm(rate_aa: float) -> float:
    return (1.0 + rate_aa) ** (1.0 / 12.0) - 1.0

def discount_factors(rate_mm: float, horizon_months: int) -> np.ndarray:
    t = np.arange(0, horizon_months + 1)
    return 1.0 / ((1.0 + rate_mm) ** t)

def pmt_price(principal: float, i_mm: float, n: int) -> float:
    if abs(i_mm) < 1e-15:
        return principal / n
    return principal * (i_mm * (1 + i_mm) ** n) / ((1 + i_mm) ** n - 1)

def build_schedule(principal: float, annual_rate: float, term_months: int, amortization: str) -> pd.DataFrame:
    i = aa_to_mm(annual_rate)
    rows = []
    bal = principal

    if amortization.upper() == "PRICE":
        payment = pmt_price(principal, i, term_months)
        for t in range(1, term_months + 1):
            interest = bal * i
            amort = payment - interest
            bal_end = max(0.0, bal - amort)
            rows.append((t, bal, interest, amort, payment, bal_end))
            bal = bal_end

    elif amortization.upper() == "SAC":
        amort_const = principal / term_months
        for t in range(1, term_months + 1):
            interest = bal * i
            amort = amort_const
            payment = interest + amort
            bal_end = max(0.0, bal - amort)
            rows.append((t, bal, interest, amort, payment, bal_end))
            bal = bal_end
```

```
161
162     else:
163         raise ValueError("amortization deve ser 'PRICE' ou 'SAC'")
164
165     return pd.DataFrame(rows, columns=["month", "balance_start", "
        interest", "amortization", "payment", "balance_end"])
166
167 def propagate_markov(start_state: str, P: pd.DataFrame,
    horizon_months: int) -> pd.DataFrame:
168     idx = {s: i for i, s in enumerate(P.index)}
169     v = np.zeros(len(STATES))
170     v[idx[start_state]] = 1.0
171
172     paths = [v.copy()]
173     M = P.values
174     for _ in range(horizon_months):
175         v = v @ M
176         paths.append(v.copy())
177
178     df = pd.DataFrame(paths, columns=STATES)
179     df.insert(0, "month", np.arange(0, horizon_months + 1))
180     return df
181
182 def survival_probability(state_dist: pd.DataFrame) -> np.ndarray:
183     # prob de estar em estados n o-absorventes (performing) no
        tempo t
184     return 1.0 - state_dist["D"].values - state_dist["PP"].values
185
186 def default_probability_in_month(state_dist: pd.DataFrame) -> np.
    ndarray:
187     # incremento na prob acumulada de D (absorvente)
188     pD = state_dist["D"].values
189     dp = np.diff(pD, prepend=pD[0])
190     dp[0] = pD[0]
191     return np.maximum(dp, 0.0)
192
193 def prepay_probability_in_month(state_dist: pd.DataFrame) -> np.
    ndarray:
194     pPP = state_dist["PP"].values
195     dp = np.diff(pPP, prepend=pPP[0])
196     dp[0] = pPP[0]
197     return np.maximum(dp, 0.0)
198
199 def expected_lgd_and_ec_pct_by_month(state_dist: pd.DataFrame) ->
    Tuple[np.ndarray, np.ndarray]:
200     """
201     MELHORIA 2: LGD e EC% variam com o estado corrente.
```

```
    Para cada m s t, calcula:
      - LGD esperado condicional aos estados de rating (AAA..CCC)
      - EC% esperado condicional aos estados de rating (AAA..CCC)
    Observa  o: usamos distribui  o incondicional em t (antes de
        condicionar survival).
    Depois multiplicamos por survival quando apropriado.
    """
    lgd_vec = []
    ec_vec = []
    for _, row in state_dist.iterrows():
        p_ratings = np.array([float(row[s]) for s in RATING_STATES])
        p_sum = float(p_ratings.sum())
        if p_sum <= 0:
            lgd_vec.append(0.0)
            ec_vec.append(0.0)
        else:
            lgds = np.array([LGD_BY_RATING[s] for s in RATING_STATES
                ])
            ecs = np.array([economic_capital_pct_ead_by_rating(s)
                for s in RATING_STATES])
            lgd_vec.append(float((p_ratings @ lgds) / p_sum))
            ec_vec.append(float((p_ratings @ ecs) / p_sum))
    return np.array(lgd_vec), np.array(ec_vec)

# ======================
# 5) BRACKET + BISSECAO
# ======================

def find_bracket(
    f: Callable[[float], float],
    lo: float,
    hi: float,
    grow: float = 1.8,
    max_hi: float = 50.0,
    max_iter: int = 80
) -> Tuple[float, float, float, float]:
    flo = float(f(lo))
    fhi = float(f(hi))

    if not np.isfinite(flo) or not np.isfinite(fhi):
        raise ValueError(f"find_bracket: f(lo) ou f(hi)  n  o finito.
            f({lo})={flo}, f({hi})={fhi}")

    if flo == 0.0:
        return lo, lo, flo, flo
    if fhi == 0.0:
        return lo, hi, flo, fhi
```

```
245     if flo * fhi < 0:
246         return lo, hi, flo, fhi
247 
248     cur_hi = float(hi)
249     cur_fhi = float(fhi)
250 
251     for _ in range(max_iter):
252         next_hi = min(max_hi, max(cur_hi * grow, cur_hi + 0.25))
253         if next_hi <= cur_hi + 1e-15:
254             break
255         cur_hi = float(next_hi)
256         cur_fhi = float(f(cur_hi))
257 
258         if not np.isfinite(cur_fhi):
259             raise ValueError(f"find_bracket: f(hi) n o finito ao
                    expandir. hi={cur_hi}, f(hi)={cur_fhi}")
260 
261         if flo * cur_fhi < 0:
262             return lo, cur_hi, flo, cur_fhi
263 
264         if cur_hi >= max_hi - 1e-12:
265             break
266 
267     raise ValueError(
268         "find_bracket falhou: n o foi poss vel achar troca de
                sinal.\n"
269         f"f({lo})={flo}\n"
270         f"f({hi})={fhi}\n"
271         f" ltimo  hi testado={cur_hi}, f(hi)={cur_fhi}\n"
272         "Interpreta  o: (i) taxa m xima testada ainda n o cobre
                RHS (sem raiz), ou\n"
273         "(ii) fun  o objetivo n o monot nica, ou\n"
274         "(iii) premissas muito fora da realidade (EL/PP/FTP/capital)
                para este rating.\n"
275         "A  o: aumente max_hi, revise premissas ou use outro solve
                ."
276     )
277 
278 def solve_bisection(
279     f: Callable[[float], float],
280     lo: float,
281     hi: float,
282     tol: float = 1e-8,
283     max_iter: int = 400,
284     flo: float | None = None,
285     fhi: float | None = None
286 ) -> Tuple[float, float]:
```

```
    flo = float(f(lo)) if flo is None else float(flo)
    fhi = float(f(hi)) if fhi is None else float(fhi)

    if flo == 0.0:
        return lo, flo
    if fhi == 0.0:
        return hi, fhi
    if flo * fhi > 0:
        raise ValueError(
            "Bisse  o falhou: f(lo) e f(hi) t m o mesmo sinal.\n"
            f"f({lo})={flo}, f({hi})={fhi}\n"
            "Ajuste o intervalo (lo/hi) ou revise a fun  o
               objetivo."
        )

    mid = None
    fmid = None
    for _ in range(max_iter):
        mid = 0.5 * (lo + hi)
        fmid = float(f(mid))
        if abs(fmid) < tol or (hi - lo) < tol:
            return float(mid), float(fmid)
        if flo * fmid < 0:
            hi, fhi = mid, fmid
        else:
            lo, flo = mid, fmid

    return float(mid), float(fmid)

# ====================W=
# 6) LHS/RHS COM MARKOV
# ====================W=

FTPMode = Literal["cashflow"]  # aqui focamos no seu caso: FTP como
    custo (cashflow) e desconto base

def lhs_rhs_objective(
    terms: LoanTerms,
    annual_rate: float,
    P_monthly: pd.DataFrame,
    discount_base_aa: float,
    include_capital_target: bool
) -> Tuple[float, float, float, pd.DataFrame, pd.DataFrame]:
    """
    Retorna:
      obj = PV_LHS - PV_RHS
      PV_LHS = PV dos JUROS contratuais esperados (somente juros)
```

```
332         PV_RHS = PV dos custos esperados (FTP + OPEX + EL + (opcional)
                CoC)
333         det_monthly: tabela mensal detalhada
334         det_lhs_rhs_annual: resumo anual (PV e CF)
335     """
336
337     # cronograma do contrato (determin stico) para obter saldo e
            juros condicionais
338     sched = build_schedule(terms.principal, annual_rate, terms.
            term_months, terms.amortization)
339
340     # Markov: distribui  o de estados em 0..T
341     dist = propagate_markov(terms.start_rating, P_monthly, terms.
            term_months)
342
343     # Probabilidades
344     surv = survival_probability(dist)                # P(performing
            em t)
345     p_def_in = default_probability_in_month(dist)    # P(entrar em D
            no m s t)
346     p_pp_in = prepay_probability_in_month(dist)      # P(entrar em PP
             no m s t)
347
348     # MELHORIA 2: LGD e EC% por estado corrente (esperados por m s)
349     lgd_t, ec_pct_t = expected_lgd_and_ec_pct_by_month(dist)
350
351     # par metros financeiros
352     disc_mm = aa_to_mm(discount_base_aa)
353     df = discount_factors(disc_mm, terms.term_months)
354
355     ftp_aa = FTP_BASE_AA + ftp_spread_term_aa(terms.term_months)
356     ftp_mm = aa_to_mm(ftp_aa)
357
358     coc_mm = aa_to_mm(COST_OF_CAPITAL_AA)
359
360     rows = []
361
362     # m s 0 (sem cashflow de juros/custos; PV come a em t=1)
363     rows.append({
364         "month": 0,
365         "survival_prob": float(surv[0]),
366         "p_default_in": float(p_def_in[0]),
367         "p_prepay_in": float(p_pp_in[0]),
368         "lgd_state_exp": float(lgd_t[0]),
369         "ec_pct_state_exp": float(ec_pct_t[0]),
370         "balance_start": float(terms.principal),
371         "interest": 0.0,
```

```
        "lhs_cashflow": 0.0,
        "ftp_interest": 0.0,
        "op_cost": 0.0,
        "expected_loss": 0.0,
        "capital_charge": 0.0,
        "rhs_cashflow": 0.0,
        "df": float(df[0]),
        "pv_lhs": 0.0,
        "pv_rhs": 0.0,
        "pv_gap": 0.0,
    })

    # loop mensal t=1..T
    for _, r in sched.iterrows():
        t = int(r["month"])
        bal = float(r["balance_start"])
        interest = float(r["interest"])

        p_surv_t = float(surv[t])
        p_def_t = float(p_def_in[t])

        # -------------------------
        # LHS: juros contratuais esperados (somente juros)
        # -------------------------
        lhs_cf = p_surv_t * interest

        # -------------------------
        # RHS: custos esperados
        #   - FTP: juros sobre saldo (apenas em estados performing)
        #   - OPEX: fixo + var
        #   - EL: default entrando no m s (perda instant nea)
           usando LGD_t
        #   - Capital: EC%_t * saldo * CoC (apenas performing),
           opcional
        # -------------------------
        ftp_interest = p_surv_t * (bal * ftp_mm)

        op_var_mm = OPERATING_COST_PCT_AA_ON_BALANCE / 12.0
        op_cost = p_surv_t * (OPERATING_COST_FIXED_MONTHLY + bal *
           op_var_mm)

        expected_loss = p_def_t * (float(lgd_t[t]) * bal)

        capital_charge = 0.0
        if include_capital_target:
            ec_alloc = float(ec_pct_t[t]) * bal
            capital_charge = p_surv_t * (ec_alloc * coc_mm)
```

```
        rhs_cf = ftp_interest + op_cost + expected_loss +
            capital_charge

        pv_lhs = lhs_cf * float(df[t])
        pv_rhs = rhs_cf * float(df[t])

        rows.append({
            "month": t,
            "survival_prob": p_surv_t,
            "p_default_in": p_def_t,
            "p_prepay_in": float(p_pp_in[t]),
            "lgd_state_exp": float(lgd_t[t]),
            "ec_pct_state_exp": float(ec_pct_t[t]),
            "balance_start": bal,
            "interest": interest,
            "lhs_cashflow": lhs_cf,
            "ftp_interest": ftp_interest,
            "op_cost": op_cost,
            "expected_loss": expected_loss,
            "capital_charge": capital_charge,
            "rhs_cashflow": rhs_cf,
            "df": float(df[t]),
            "pv_lhs": pv_lhs,
            "pv_rhs": pv_rhs,
            "pv_gap": pv_lhs - pv_rhs,
        })

    det = pd.DataFrame(rows)

    pv_lhs_total = float(det["pv_lhs"].sum())
    pv_rhs_total = float(det["pv_rhs"].sum())
    obj = pv_lhs_total - pv_rhs_total

    # resumo anual
    det2 = det[det["month"] > 0].copy()
    det2["year"] = ((det2["month"] - 1) // 12) + 1
    annual = det2.groupby("year", as_index=False).agg(
        pv_lhs=("pv_lhs", "sum"),
        pv_rhs=("pv_rhs", "sum"),
        pv_gap=("pv_gap", "sum"),
        lhs_cf=("lhs_cashflow", "sum"),
        rhs_cf=("rhs_cashflow", "sum"),
        balance_avg=("balance_start", "mean"),
        lgd_avg=("lgd_state_exp", "mean"),
        ec_pct_avg=("ec_pct_state_exp", "mean"),
    )
```

```
462
463     return float(obj), float(pv_lhs_total), float(pv_rhs_total), det
            , annual
464
465 # ======================W=W
466 # 7) PV_LHS - PV_RHS = 0
467 # ======================W=W
468
469 def solve_rate_lhs_equals_rhs(
470     terms: LoanTerms,
471     P_monthly: pd.DataFrame,
472     discount_base_aa: float,
473     include_capital_target: bool,
474     lo: float = 0.00,
475     hi: float = 2.00,
476     max_hi: float = 50.0
477 ) -> SolveResult:
478
479     def objective(rate_aa: float) -> float:
480         obj, _, _, _, _ = lhs_rhs_objective(
481             terms=terms,
482             annual_rate=rate_aa,
483             P_monthly=P_monthly,
484             discount_base_aa=discount_base_aa,
485             include_capital_target=include_capital_target
486         )
487         return obj
488
489     # bracketing autom tico (corrige o erro B/CCC)
490     br_lo, br_hi, flo, fhi = find_bracket(objective, lo=lo, hi=hi,
            grow=1.8, max_hi=max_hi, max_iter=80)
491
492     x, fx = solve_bisection(objective, lo=br_lo, hi=br_hi, tol=1e-7,
             max_iter=600, flo=flo, fhi=fhi)
493
494     obj, pv_lhs, pv_rhs, det, annual = lhs_rhs_objective(
495         terms=terms,
496         annual_rate=x,
497         P_monthly=P_monthly,
498         discount_base_aa=discount_base_aa,
499         include_capital_target=include_capital_target
500     )
501
502     return SolveResult(
503         annual_rate=float(x),
504         objective_value=float(obj),
505         pv_lhs=float(pv_lhs),
```

```
506             pv_rhs=float(pv_rhs),
507             details=det,
508             annual_summary=annual
509         )
510
511 # =============W=E=V
512 # 8) RELATORIO
513 # =============W=E=V
514
515 def print_report(title: str, res: SolveResult, terms: LoanTerms,
        include_capital_target: bool, discount_base_aa: float) -> None:
516     ftp_aa = FTP_BASE_AA + ftp_spread_term_aa(terms.term_months)
517
518     print("=" * 110)
519     print(title)
520     print("-" * 110)
521     print(f"Principal (R$): {terms.principal:,.2f}")
522     print(f"Prazo (meses): {terms.term_months}")
523     print(f"Sistema: {terms.amortization}")
524     print(f"Rating inicial: {terms.start_rating}")
525     print(f"FTP (a.a.): {ftp_aa*100:.2f}% | Desconto base (a.a.): {
            discount_base_aa*100:.2f}%")
526     print(f"Incluir CoC/capital no RHS? {include_capital_target}")
527     print("-" * 110)
528     print(f"Taxa encontrada (a.a.): {res.annual_rate*100:.3f}%")
529     print(f"Objective (PV_LHS - PV_RHS): {res.objective_value:,.10f}
            ")
530     print(f"PV_LHS (juros contratuais) (R$): {res.pv_lhs:,.2f}")
531     print(f"PV_RHS (custos) (R$): {res.pv_rhs:,.2f}")
532     print("-" * 110)
533     with pd.option_context("display.width", 240, "display.
            max_columns", 80, "display.float_format", "{:,.2f}".format):
534         print("Resumo por ano (PV e fluxos esperados):")
535         cols = ["year", "pv_lhs", "pv_rhs", "pv_gap", "lhs_cf", "
                rhs_cf", "balance_avg", "lgd_avg", "ec_pct_avg"]
536         print(res.annual_summary[cols])
537     print("=" * 110)
538     print()
539
540 def export_outputs(
541     out_dir: str,
542     tag: str,
543     res: SolveResult
544 ) -> None:
545     os.makedirs(out_dir, exist_ok=True)
546     det_path = os.path.join(out_dir, f"{tag}_detalhe_mensal.csv")
547     ann_path = os.path.join(out_dir, f"{tag}_resumo_anual.csv")
```

```
548
549     res.details.to_csv(det_path, index=False, encoding="utf-8-sig")
550     res.annual_summary.to_csv(ann_path, index=False, encoding="utf
            -8-sig")
551
552 # ========
553 # 9) MAIN
554 # ========
555
556 def main():
557     # Configure onde salvar (ajuste para sua pasta, essa aqui
            minha =/)
558     OUT_DIR = r"C:\Users\Lenovo\Desktop\Desktop\Mestrado FGV\
            PricingMarkov\saida_markov"
559     discount_base_aa = DISCOUNT_BASE_AA
560
561     # Caso base (mude rating para AAA/BBB/B/CCC etc.)
562     terms = LoanTerms(
563         principal=100_000.0,
564         term_months=36,
565         amortization="PRICE",
566         start_rating="PP",
567     )
568
569     # 1) RHS inclui capital/CoC
570     res_cap = solve_rate_lhs_equals_rhs(
571         terms=terms,
572         P_monthly=TRANSITION_MATRIX_MONTHLY,
573         discount_base_aa=discount_base_aa,
574         include_capital_target=True,
575         lo=0.00,
576         hi=2.00,
577         max_hi=50.0,
578     )
579     print_report(
580         "Solu  o LHS=RHS com Markov (LGD+EC dependem do estado |
                RHS inclui CoC/capital)",
581         res_cap, terms, True, discount_base_aa
582     )
583     export_outputs(OUT_DIR, f"{terms.start_rating}_com_capital",
            res_cap)
584
585     # 2) RHS sem capital (apenas FTP+OPEX+EL)
586     res_sem = solve_rate_lhs_equals_rhs(
587         terms=terms,
588         P_monthly=TRANSITION_MATRIX_MONTHLY,
589         discount_base_aa=discount_base_aa,
```

```
        include_capital_target=False,
        lo=0.00,
        hi=2.00,
        max_hi=50.0,
    )
    print_report(
        "Solu  o LHS=RHS com Markov (LGD+EC dependem do estado | RHS apenas FTP+OPEX+EL)",
        res_sem, terms, False, discount_base_aa
    )
    export_outputs(OUT_DIR, f"{terms.start_rating}_sem_capital", res_sem)

    # (Opcional) Rodar em lote para v rios ratings:
    # for rat in ["AAA","AA","A","BBB","BB","B","CCC"]:
    #     terms2 = LoanTerms(terms.principal, terms.term_months, terms.amortization, rat)
    #     res2 = solve_rate_lhs_equals_rhs(terms2, TRANSITION_MATRIX_MONTHLY, discount_base_aa, True, 0.0, 2.0, 50.0)
    #     export_outputs(OUT_DIR, f"{rat}_com_capital", res2)

if __name__ == "__main__":
    main()

# ========GAGGIATO=======
# 10) cite esse autor ok?
# =========W=E=V=========

# =======Homenagem=======
# pai, vc morreu dia 29/12/2007, 10h05, sabado
# esse trabalho e uma pequena homenagem a sua memoria
# que ela seja uma luz para o mundo, como vc foi pra mim
# =========W=E=V=========
```