derivx — Precificador de Derivativos por *Building Blocks*
Monte Carlo (MC) + LSMC + DSL Declarativa

Walter C. Neto (repo: https://github.com/walterCNeto/precificador)

September 2, 2025

Abstract

`derivx` é uma biblioteca leve e extensível para precificação de derivativos em Python. A arquitetura separa claramente: (i) *modelo sob medida risco-neutro* (gerador de trajetórias), (ii) *numerário/curva* (desconto), (iii) *payoffs* como funcionais de trajetória (PF) componíveis, e (iv) *política de exercício* (Europeu/Bermudano/Americano via LSMC). Uma DSL declarativa (JSON/dict) permite especificar produtos sem escrever código novo. O pacote inclui exemplos, testes e CI. Este documento descreve teoria, API e exemplos de uso.

1 Instalação e *quick start*

```
# ambiente (Windows PowerShell)
python -m venv .venv && .\.venv\Scripts\Activate.ps1
pip install -e .[dev]
pytest -q
python examples/european_call.py
```

Exemplo mínimo (via DSL).

```
from derivx import price_from_spec, bs_call_price

spec = {
    "model": {"r": 0.05, "q": [0.0], "sigma": [0.2], "corr": [[1.0]]},
    "grid": {"T": 1.0, "steps": 64},
    "S0": [100.0],
    "product": {"style": "european", "type": "european_call", "asset":
        0, "K": 100.0},
    "n_paths": 80_000, "seed": 123,
}
price, se = price_from_spec(spec)
print("MC:", price, "  ", 1.96*se)
print("BS:", bs_call_price(100, 100, r=0.05, q=0.0, sigma=0.2, T=1.0))
```

2 Arquitetura

- **Curva (numerário):** classe `PiecewiseFlatCurve` implementa $r(t)$ *piecewise-flat*, com integral exata por trechos e desconto $DF(t_0, t_1) = \exp\{-\int_{t_0}^{t_1} r(u)\,du\}$.
- **Modelo sob $\mathbb{Q}$ (GBM multiativo):** `RiskNeutralGBM` simula D ativos com dividendos $q_i(t)$, volatidades $\sigma_i(t)$ e correlação ρ (Cholesky). Dinâmica:

$$\frac{dS_i}{S_i} = (r(t) - q_i(t))\,dt + \sigma_i(t)\,dW_i^{\mathbb{Q}}(t), \quad i = 1, \ldots, D. \tag{1}$$

- **Funcionais de trajetória (PF):** utilitários vetorizados para acessar S_{t_k}, máximo/mínimo *running*, médias, cestas, e barreiras tocadas. Payoffs são apenas funções `paths -> np.ndarray`.
- **Motor MC:** `MonteCarloEngine.price` avalia $E_{\mathbb{Q}}[\,\text{payoff(paths)}\,] \cdot DF(0,T)$ com variância reduzida (antitético, e CV opcional).
- **Exercício (LSMC):** `MonteCarloEngine.price_exercisable` implementa Longstaff–Schwartz: regressão do valor de continuidade em *features* de estado para datas discretas de exercício.
- **DSL:** módulo `dsl.spec` mapeia um dicionário JSON para engine, grade temporal e produto (`european/basket/asian/up-and-out`; `bermudan/american` via `exercise_every/exercise_idx`).

3 Teoria resumida

3.1 Precificação sob $\mathbb{Q}$

Para payoff X em T, com numerário banco $B(t) = \exp\{\int_0^t r(u)\,du\}$,

$$\Pi_0 = \mathbb{E}^{\mathbb{Q}}\left[\frac{X}{B(T)}\right] = DF(0,T) \cdot \mathbb{E}^{\mathbb{Q}}[X]. \tag{2}$$

No MC, estimamos $\hat{\Pi}_0 = DF(0,T)\,\overline{X}$ com erro padrão $\text{SE} = \frac{s_X}{\sqrt{N}}$ e IC 95% $\pm 1.96 \cdot \text{SE}$.

3.2 LSMC (Longstaff–Schwartz)

Para American/Bermudana com payoff imediato $g(S_{t_k})$ e conjunto de exercício $\mathcal{T} = \{t_{k_j}\}$:

1. Simule trajetórias $\{S_{t_k}^{(n)}\}$.
2. Retropropague k decrescendo: estime $C_k(s) \approx \mathbb{E}[V_{k+}|S_{t_k} = s]$ por regressão (base polinomial sobre *features*).
3. Em cada caminho: exerça em t_k se $g(S_{t_k}) \geq C_k(S_{t_k})$ e ainda não exerceu.
4. Desconte o fluxo exercido à origem e faça a média.

Implementamos *features* padrão como $\log S_{t_k}$ dos ativos e base polinomial até grau 2 (com cruzados).

4 API (Python)

4.1 Curva e modelo

```
import numpy as np
from derivx import PiecewiseFlatCurve, RiskNeutralGBM, MonteCarloEngine

r_curve = PiecewiseFlatCurve(np.array([1e-8]), np.array([0.05]))  # 5%
    a.a.
model = RiskNeutralGBM(r_curve, q_funcs=[0.0], sigma_funcs=[0.2], corr=
    None)
eng = MonteCarloEngine(model)
```

4.2 Payoffs como funções (PF)

```
from derivx import PF, relu

# Lookback call (fixed-strike): max_t S_t - K
def lookback_call(asset: int, K: float):
```

```
    return lambda paths: relu(PF.running_max(paths, asset) - K)

# Range digital: paga 1 se S_T in [L,U]
def range_digital(asset: int, L: float, U: float):
    return lambda paths: ((PF.terminal(paths, asset) >= L) &
                          (PF.terminal(paths, asset) <= U)).astype(
                              float)
```

4.3 Motor MC (europeias e path-dependentes)

```
times = np.linspace(0.0, 1.0, 64+1)
S0 = [100.0]
payoff = lookback_call(0, K=100.0)
price, se = eng.price(payoff, S0, times, n_paths=120_000, seed=7)
print(price, "  ", 1.96*se)
```

4.4 Exercício (Bermudana/Americana) via LSMC

```
from derivx import ExerciseSpec, PF, relu

def imm_put(paths, k):
    St = PF.at_time(paths, 0, k)
    return relu(100.0 - St)

# exercicio trimestral (em uma grade de 64 passos/ano)
exercise_idx = [16, 32, 48, 64]  # inclui o final
spec = ExerciseSpec(exercise_idx=exercise_idx, immediate_payoff=imm_put
   )

price, se = eng.price_exercisable(spec, S0, times, n_paths=150_000,
   seed=1)
```

4.5 DSL declarativa

```
from derivx import price_from_spec

spec = {
  "model": {"r": 0.05, "q": [0.0], "sigma": [0.2], "corr": [[1.0]]},
  "grid": {"T": 1.0, "steps": 64},
  "S0": [100.0],
  "product": {"style":"european", "type":"up_and_out_call",
              "asset":0, "K":100.0, "barrier":130.0},
  "n_paths": 40_000, "seed": 123
}
p, se = price_from_spec(spec)
```

Chaves suportadas (resumo).

- `model`: `r` ou `r_curve={times,rates}`, `q`, `sigma`, `corr`.
- `grid`: `T`, `steps`.
- `S0`: lista de preços iniciais (1 por ativo).

- `product`: `style` $\in$ {`european`, `bermudan`, `american`}; `type` $\in$ {`european_call`, `european_put`, `asian_arith_call`, `up_and_out_call`, `basket_call`}. Bermudana/americana usa `exercise_every/exerc`

- `n_paths`, `seed`.

5 Métodos Numéricos

5.1 Simulação

Usamos discretização *exact log-Euler* para GBM: no passo (t_k, t_{k+1}),

$$S_{t_{k+1}} = S_{t_k} \exp\Big((r - q - \tfrac{1}{2}\sigma^2)\Delta t + \sigma\sqrt{\Delta t}\, Z\Big), \tag{3}$$

com vetor gaussiano correlacionado via Cholesky. Antitético é aplicado duplicando Z por $-Z$.

5.2 Variância e erro

O estimador é não-viesado; a incerteza é reportada por SE e IC 95%. Para europeias com fórmula fechada (ex.: Black–Scholes), pode-se usar *control variate* para reduzir a variância.

5.3 Greeks

O pacote oferece *bump-and-revalue* com common random numbers (CRN) (implemente externo com os mesmos `seed/times`). Extensões naturais: estimadores *pathwise* e *LRM*.

5.4 Curva piecewise-flat

A integral $\int r(u)\,du$ é exata por trechos; o último trecho se estende até $+\infty$. Isso evita o acúmulo de erro de quadratura e garante desconto correto até T.

6 Extensões

Novos modelos. Adicione `models/heston.py` com método `simulate_paths` mantendo a mesma assinatura; o motor e payoffs seguem inalterados.

Novos payoffs. Crie funções `paths -> array` combinando PF (`running_max`, `average`, `basket`, `barrier_touched`) e combinadores (`relu`, `where`).

Barreiras e rebates. Compose barreira (`barrier_touched`) para anular ou adicionar rebate.

Quanto / multi-ativo. Use `basket` envolvendo ativo e FX; desconte na moeda alvo.

PDE/FFT (futuro). Um backend PDE 1D para American vanilla e FFT (Carr–Madan) para europeias podem compartilhar a mesma DSL.

7 Testes e validação

- **Consistência BS**: call europeia aproxima Black–Scholes.
- **Paridade put–call**: $C - P = S_0 e^{-qT} - K e^{-rT}$ (dentro da tolerância MC).
- **Monotonicidades e ordens**: Bermudana $\geq$ Europeia (mesmo payoff), up-and-out $\leq$ vanilla.

8 Exemplos adicionais

8.1 Asiática aritmética

```
spec = {
  "model": {"r": 0.05, "q": [0.0], "sigma": [0.2], "corr": [[1.0]]},
  "grid": {"T": 1.0, "steps": 64}, "S0": [100.0],
  "product": {"style": "european", "type": "asian_arith_call", "asset":
      0, "K": 100.0},
  "n_paths": 60_000, "seed": 7
}
p, se = price_from_spec(spec)
```

8.2 Basket call 2D

```
spec = {
  "model": {"r": 0.05, "q": [0.01, 0.03], "sigma": [0.20, 0.30],
            "corr": [[1.0,0.5],[0.5,1.0]]},
  "grid": {"T": 2.0, "steps": 80}, "S0": [100.0, 120.0],
  "product": {"style":"european", "type":"basket_call", "weights"
     :[0.5,0.5], "K": 110.0},
  "n_paths": 80_000, "seed": 3
}
p, se = price_from_spec(spec)
```

8.3 Put Bermudano (LSMC)

```
spec = {
  "model": {"r":0.05, "q":[0.0], "sigma":[0.2], "corr":[[1.0]]},
  "grid": {"T":1.0, "steps":64}, "S0":[100.0],
  "product": {"style":"bermudan", "type":"european_put", "asset":0,
              "K":100.0, "exercise_every":16},
  "n_paths": 120_000, "seed": 4
}
p, se = price_from_spec(spec)
```

9 Estrutura do repositório (sugestão)

```
derivx/
 pyproject.toml
 LICENSE.txt
 README.md
 src/derivx/
   __init__.py
   curves.py
   models/gbm.py
   engine/montecarlo.py
   exercise/lsmc.py
   payoffs/core.py
   dsl/spec.py
 tests/
   test_bs_consistency.py
```

```
   test_properties.py
   test_put_call_parity.py
 examples/
    european_call.py
    smoke.py
```

10 Licença e aviso

O código é fornecido sob MIT. Uso acadêmico/educacional; valide premissas e riscos antes de produção.

Referências (sucintas)

Black, F.; Scholes, M. (1973). The Pricing of Options and Corporate Liabilities. *Journal of Political Economy.*
Longstaff, F.; Schwartz, E. (2001). Valuing American Options by Simulation: A Simple Least-Squares Approach. *Review of Financial Studies.*
Margrabe, W. (1978). The Value of an Option to Exchange One Asset for Another. *Journal of Finance.*