derivx — Precificador de Derivativos por *Building Blocks*

MC/LSMC, PDE Crank–Nicolson e FFT (Heston) com DSL Declarativa

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

September 3, 2025

Abstract

`derivx` é uma biblioteca leve e extensível para precificação de derivativos em Python. A arquitetura separa: (i) *modelo sob risco-neutro* (gerador de trajetórias), (ii) *numerário/curva* (desconto determinístico), (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. Inclui backends MC (GBM/Heston), PDE 1D (Crank–Nicolson) para vanillas e FFT (Carr–Madan) para Heston europeu. Exemplos, testes e CI acompanham o pacote. Este documento traz teoria, API, guias de uso e exemplos.

Contents

1 Instalação e *quick start*

```
# Windows PowerShell
python -m venv .venv
.\.venv\Scripts\Activate.ps1
pip install -e ".[dev]"
pytest -q
python examples/smoke.py
```

Exemplo mínimo (via DSL).

```python
from derivx import price_from_spec, bs_call_price

spec = {
  "engine": "mc",
  "model": {"name":"gbm","r":0.05,"q":0.0,"sigma":0.2},
  "grid": {"T": 1.0, "steps": 128},
  "S0": [100.0],
  "product": {"style":"european","type":"european_call","asset":0,"K"
     :100.0},
  "n_paths": 80_000, "seed": 42,
}
p, se = price_from_spec(spec)
print("MC:", p, "  ", 1.96*se)
print("BS:", bs_call_price(100, 100, r=0.05, q=0.0, sigma=0.2, T=1.0))
```

2 Arquitetura (visão geral)

- **Curva (numerário).** `PiecewiseFlatCurve` implementa $r(t)$ *piecewise-flat* com desconto $DF(t_0, t_1) = \exp\{-\int_{t_0}^{t_1} r(u)\,du\}$ exato por trechos.
- **Modelos sob $\mathbb{Q}$.** GBM multiativo (`RiskNeutralGBM`) com $q_i(t)$, $\sigma_i(t)$ e correlação, e Heston para europeias (MC/FFT).
- **Funcionais de trajetória (PF).** Utilitários vetorizados: S_{t_k}, máximos/mínimos corridos, médias, barreiras, cestas. Payoffs são funções `paths -> np.ndarray`.
- **Motores numéricos.** *MC*: simulação e estimação $E_{\mathbb{Q}}[X]$ (antitético e CV opcionais). *PDE CN*: 1D para calls/puts; exercício americano via projeção max por passo de tempo. *FFT (Heston)*: Carr–Madan com parâmetros α, N, η.
- **Exercício (LSMC).** `ExerciseSpec` + `lsmc.price`: regressão do valor de continuação em features (por padrão $\log S$, base polinomial até grau 2 + cruzados).
- **DSL.** `price_from_spec(spec)` mapeia um dicionário JSON para engine, grade temporal e produto.

3 Base teórica

3.1 Precificação sob $\mathbb{Q}$ e numeraire

Seja $B(t) = \exp\{\int_0^t r(u)\,du\}$ o banco. Para payoff X em T:

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

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

3.2 GBM multiativo e simulação exata

Para $i = 1, \ldots, D$:

$$\frac{dS_i}{S_i} = (r(t) - q_i(t))\,dt + \sigma_i(t)\,dW_i^{\mathbb{Q}}(t), \quad \mathrm{Cov}(dW_i, dW_j) = \rho_{ij}\,dt.$$

No passo Δt:

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

com $Z \sim \mathcal{N}(0, \rho)$ via Cholesky. Antitético usa Z e $-Z$. *Bias* temporal é $\mathcal{O}(\Delta t)$ para payoffs path-dependentes; use `steps` adequados.

3.3 LSMC (Longstaff–Schwartz)

Datas de exercício $\mathcal{T} = \{t_{k_j}\}$; payoff imediato $g(S_{t_k})$. Seja V_k o valor ótimo ao tempo t_k e τ a *stopping time* ótima. Define-se o valor de continuação:

$$C_k(s) = \mathbb{E}\big[\,DF(t_k, t_{k+1})\,V_{k+1} \,\big|\, S_{t_k} = s\big].$$

O algoritmo:

1. Simule caminhos $S_{t_k}^{(n)}$.
2. No vencimento $t_{k^\star}$, $V^{(n)} = g(S_{t_{k^\star}}^{(n)})$.
3. Para k decrescendo nos tempos de exercício: estime C_k por regressão (apenas nos ITM).
4. Em cada caminho, exerça em t_k se $g(S_{t_k}) \geq \widehat{C}_k(S_{t_k})$.

Cuidados práticos: (i) restringir regressão ao ITM; (ii) usar desconto correto entre janelas (Δt possivelmente múltiplas); (iii) evitar overfit (cross-fit A/B); (iv) checar monotonia: Bermudan densificando $\Rightarrow$ aproxima Americano.

3.4 PDE 1D Crank–Nicolson (vanillas)

Para um ativo sob GBM, a PDE de Black–Scholes para $V(S,t)$:

$$\partial_t V + (r - q)S\,\partial_S V + \tfrac{1}{2}\sigma^2 S^2 \partial_{SS}^2 V - rV = 0.$$

CN discretiza implicitamente em t e centralmente em S, levando a um sistema tridiagonal por passo. **Condições de contorno** usuais:

- *Call*: $V(0,t) = 0$; $V(S_{\max}, t) \approx S_{\max}e^{-q(T-t)} - Ke^{-r(T-t)}$.
- *Put*: $V(0,t) \approx Ke^{-r(T-t)}$; $V(S_{\max}, t) \to 0$.

Americano: após resolver o passo, projeta-se $V \leftarrow \max(V, \text{payoff imediato})$. **Convergência**: $\mathcal{O}(\Delta t + \Delta S^2)$. Aumentar `NS`, `NT` e `Smax_mult` melhora acurácia.

3.5 FFT (Heston europeu)

Em Heston, precificação por Carr–Madan calcula preços via a transformada de Fourier do payoff amortecido por $\alpha > 0$:

$$C(K) = \frac{e^{-\alpha k}}{\pi} \int_0^\infty \Re\big(e^{-iuk}\,\psi(u)\big)\,du, \quad k = \ln K.$$

Com discretização uniforme em frequência (η) e tamanho de malha N, uma FFT retorna uma grade de preços para diversos *strikes*. Parâmetros:

- α (damping, típico 1.0–2.0),
- N (tamanho da FFT, potência de 2),
- η (passo em frequência; governa espaçamento em strikes).

4 Como usar a biblioteca

4.1 Camada 1 — API direta (curvas, modelos, motores)

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

# Curva "piecewise-flat": 5% a.a.
r_curve = PiecewiseFlatCurve(np.array([1e-8]), np.array([0.05]))

# Modelo GBM monofator
model = RiskNeutralGBM(r_curve, q_funcs=[0.0], sigma_funcs=[0.2], corr=
    None)
eng = MonteCarloEngine(model)

# Grade temporal e S0
times = np.linspace(0.0, 1.0, 128+1)
S0 = [100.0]

# Payoff europeu: call
from derivx import PF, relu
payoff = lambda paths: relu(PF.terminal(paths, asset=0) - 100.0)

price, se = eng.price(payoff, S0, times, n_paths=80_000, seed=42)
print(price, "  ", 1.96*se)
```

Exercício via LSMC.

```
from derivx import ExerciseSpec

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

exercise_idx = [16, 32, 48, 64]  # inclui vencimento
spec = ExerciseSpec(exercise_idx=exercise_idx, immediate_payoff=imm_put
    )

price, se = eng.price_exercisable(spec, S0, times, n_paths=150_000,
    seed=7)
```

4.2 Camada 2 — DSL declarativa (price_from_spec)

```
from derivx import price_from_spec

spec = {
  "engine":"mc",  # "mc" | "pde" | "fft"
  "model":{"name":"gbm","r":0.05,"q":0.0,"sigma":0.2},
  "grid":{"T":1.0,"steps":128},   # PDE usa s   T
  "S0":[100.0],
  "product":{"style":"european","type":"european_call","asset":0,"K"
     :100.0},
  "n_paths":80_000,"seed":7
}
price, se = price_from_spec(spec)
```

Chaves suportadas.

- `engine`: `"mc"`, `"pde"`, `"fft"`.
- `model`:
  - GBM: `{"name":"gbm","r":..., "q":..., "sigma":...}`.
  - Heston: `{"name":"heston","r":...,"q":...,"kappa":...,"theta":...,"xi":...,"rho":...,"v`
- `grid`: `{"T":..., "steps":...}` (MC); PDE usa `{"T":...}`.
- `S0`: lista de preços iniciais.
- `product`:
  - Vanilla: `european_call`, `european_put`.
  - Path-dep.: `asian_arith_call`, `up_and_out_call`.
  - Exercício: `{"style":"bermudan","type":"european_put","exercise_every":16}`.
- Parâmetros do motor: MC (`n_paths`, `seed`), PDE (`NS`, `NT`, `Smax_mult`), FFT (`alpha`, `N`, `eta`).

4.3 Precisão & Performance: *knobs* práticos

Backend	Parâmetro	Efeito prático
MC	`n_paths`	SE $\propto 1/\sqrt{N}$; mais paths $\Rightarrow$ erro menor (custo linear).
MC	`steps`	Menor bias temporal em path-dep./LSMC; custo $\propto$ paths $\times$ steps.
MC	`seed`	Reprodutibilidade (use CRN p/ Greeks).
PDE	`NS, NT`	Convergência $\mathcal{O}(\Delta t + \Delta S^2)$; aumente malha até estabilizar.
PDE	`Smax_mult`	Domínio $[0, S_{\max}]$; use 5–7 vezes o strike como regra inicial.
FFT	α	Damping (1–2 típico). Muito baixo/alto pode instabilizar.
FFT	N, η	Resolução em frequência/strike; aumente N para malha mais densa.

5 Validação, testes e reprodutibilidade

- **Consistência BS**: MC (GBM) $\approx$ Black–Scholes (dentro de $k \cdot$ SE).
- **PDE vs BS**: put europeu CN $\approx$ BS; americano PDE $\geq$ europeu.
- **Propriedades**: up&out $\leq$ vanilla; Bermudan com janelas mais densas $\nearrow$ American.
- **Heston**: MC $\approx$ FFT (tolerância baseada em SE).

```
# Windows
$env:PYTEST_DISABLE_PLUGIN_AUTOLOAD="1"
python -m pytest -q
```

6 Exemplos práticos (fim-a-fim)

6.1 (E1) Europeu: MC vs Black–Scholes

```
from math import exp
from derivx import price_from_spec, bs_call_price

S0=K=100.0; r=0.05; q=0.0; sigma=0.2; T=1.0
spec = {"engine":"mc","model":{"name":"gbm","r":r,"q":q,"sigma":sigma},
        "grid":{"T":T,"steps":128},"S0":[S0],
        "product":{"style":"european","type":"european_call","asset":0,
           "K":K},
        "n_paths":80_000,"seed":42}
pmc,se = price_from_spec(spec)
pbs = bs_call_price(S0,K,r,q,sigma,T)
print(f"MC={pmc:.4f}    {1.96*se:.4f} | BS={pbs:.4f}")
```

6.2 (E2) PDE: put europeu vs americano

```
from derivx import price_from_spec
euro={"engine":"pde","model":{"name":"gbm","r":0.05,"q":0.0,"sigma"
   :0.2},
      "grid":{"T":1.0},"S0":[100.0],
      "product":{"style":"european","type":"european_put","asset":0,"K"
         :100.0},
      "NS":800,"NT":800,"Smax_mult":5.0}
amer={**euro,"product":{"style":"american","type":"european_put","asset
   ":0,"K":100.0}}
pe,_ = price_from_spec(euro); pa,_ = price_from_spec(amer)
print(f"PDE euro={pe:.4f}  PDE amer={pa:.4f}  (amer >= euro)")
```

6.3 (E3) Barreira up&out e monotonia

```
base={"engine":"mc","model":{"name":"gbm","r":0.05,"q":0.0,"sigma"
   :0.2},
      "grid":{"T":1.0,"steps":128},"S0":[100.0],"n_paths":100_000,"seed
         ":7}
van,_  = price_from_spec({**base,"product":{"style":"european","type":"
   european_call","asset":0,"K":100.0}})
uo130,_= price_from_spec({**base,"product":{"style":"european","type":"
   up_and_out_call","asset":0,"K":100.0,"barrier":130.0}})
uo140,_= price_from_spec({**base,"product":{"style":"european","type":"
   up_and_out_call","asset":0,"K":100.0,"barrier":140.0}})
print(f"UO130={uo130:.4f} <= UO140={uo140:.4f} <= Vanilla={van:.4f}")
```

6.4 (E4) Asiática aritmética $\leq$ vanilla

```
asian,_= price_from_spec({**base,"product":{"style":"european","type":"
   asian_arith_call","asset":0,"K":100.0}})
print(f"Asian={asian:.4f} <= Vanilla={van:.4f}")
```

6.5 (E5) Bermudana (LSMC): frequência de exercício

```
def berm(ex_every):
  spec={"engine":"mc","model":{"name":"gbm","r":0.05,"q":0.0,"sigma"
      :0.2},
         "grid":{"T":1.0,"steps":256},"S0":[100.0],
         "product":{"style":"bermudan","type":"european_put","asset":0,"
            K":100.0,"exercise_every":ex_every},
         "n_paths":120_000,"seed":7}
  return price_from_spec(spec)
for ex in (8,16,32):
  p,se = berm(ex)
  print(f"ex_every={ex:>2}: {p:.4f}    {1.96*se:.4f}")
```

6.6 (E6) Heston: FFT vs MC

```
common={"name":"heston","r":0.05,"q":0.0,"kappa":1.5,"theta":0.04,"xi"
    :0.5,"rho":-0.7,"v0":0.04}
fft={"engine":"fft","model":common,"grid":{"T":1.0},"S0":[100.0],
      "product":{"style":"european","type":"european_call","asset":0,"K"
         :100.0},
      "alpha":1.5,"N":4096,"eta":0.25}
mc={"engine":"mc","model":common,"grid":{"T":1.0,"steps":512},"S0"
    :[100.0],
     "product":{"style":"european","type":"european_call","asset":0,"K"
        :100.0},
     "n_paths":200_000,"seed":7}
p_fft,_=price_from_spec(fft); p_mc,se=price_from_spec(mc)
print(f"FFT={p_fft:.4f} | MC={p_mc:.4f}    {1.96*se:.4f}")
```

6.7 (E7) Basket 2D (GBM correlacionado)

```
spec={"engine":"mc",
      "model":{"name":"gbm","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":100_000,"seed":3}
p,se=price_from_spec(spec)
print(p, "  ", 1.96*se)
```

6.8 (E8) Greeks por *bump & revalue* (CRN)

```
import numpy as np
base={"engine":"mc","model":{"name":"gbm","r":0.05,"q":0.0,"sigma"
    :0.2},
      "grid":{"T":1.0,"steps":256},"S0":[100.0],
      "product":{"style":"european","type":"european_call","asset":0,"K
         ":100.0},
      "n_paths":120_000,"seed":11}
def price(spec): from derivx import price_from_spec; return
    price_from_spec(spec)[0]
```

```
h=1.0
p0=price(base)
pU=price({**base,"S0":[100.0+h]})
pD=price({**base,"S0":[100.0-h]})
delta=(pU-pD)/(2*h); gamma=(pU-2*p0+pD)/(h*h)
print("Delta~",delta,"Gamma~",gamma)
```

7 Estrutura do repositório (resumo)

```
src/derivx/
  curves.py           # PiecewiseFlatCurve (df, ints)
  models/gbm.py       # RiskNeutralGBM (paths; df via curva)
  engine/montecarlo.py
  engine/pde.py       # CN 1D vanillas (euro/amer)
  engine/fft.py       # Heston FFT (Carr-Madan)
  exercise/lsmc.py    # LSMC genérico
  payoffs/core.py     # PF utilitários (terminal, média, etc.)
  dsl/spec.py         # mapeia dict -> engine/produto
tests/                # sanity e propriedades
examples/             # scripts reprodutíveis
```

8 Licença e aviso

MIT. Uso acadêmico/educacional; valide premissas, calibração e risco de modelo antes de produção.

Referências

Black, F.; Scholes, M. (1973) *The Pricing of Options and Corporate Liabilities.*
Longstaff, F.; Schwartz, E. (2001) *Valuing American Options by Simulation.*
Carr, P.; Madan, D. (1999) *Option Valuation Using the FFT.*
Heston, S. (1993) *A Closed-Form Solution for Options with Stochastic Volatility.*
Margrabe, W. (1978) *The Value of an Option to Exchange One Asset for Another.*