Financial Derivatives with C++

Dmitry Kramkov

Carnegie Mellon University

Summer School, Vega Institute,
Pushkin, July 4-9, 2022

Plan of the course

Part I: Practical arbitrage-free pricing.

Part 2: Design of pricing library with C++.

Part 3: Standard asset options with C++.

Part 4: Quantathon (6 problems for 4 hours).

Part I

Practical arbitrage-free pricing

Outline

Financial derivatives

Rollback operator

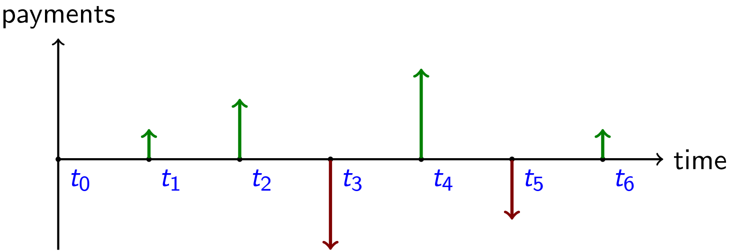
State processes

Model implementation

Financial security

$$\boxed{\textbf{Financial Security} = \textbf{Cash Flow}}$$

Example (Interest Rate Swap)



Pricing problem: compute a "fair" value of the security at t_0.

Financial derivatives

World's OTC derivatives:

- the notional volume $\approx$ \$700 trn,
- the market value $\approx$ \$20 trn.

Type of underlying: (important for design!)

1. Assets: stocks, FX rates, commodities, etc.
2. Interest rates.

Dependence on the history: (important for numerical implementation!)

1. Standard: payoff depends on the current market values
2. Path dependent: payoff depends on historical values
3. Barrier: simple dependence on the past

Example: American butterfly

This is an example of **Standard Asset** option.

P : strike for put option

C : strike for call option ($C > P$)

$(t_i)_{1 \leq i \leq N}$: exercise times

At an exercise time t_i a holder of the option can

1. **sell** the underlying stock for the strike P of put option
2. **buy** the underlying stock for the strike C of call option
3. **do nothing**, wait and exercise later.

Example: down-and-out Call

This is an example of **Barrier Asset** derivative security.

L : lower barrier

$(t_i)_{1 \leq i \leq M}$: barrier times

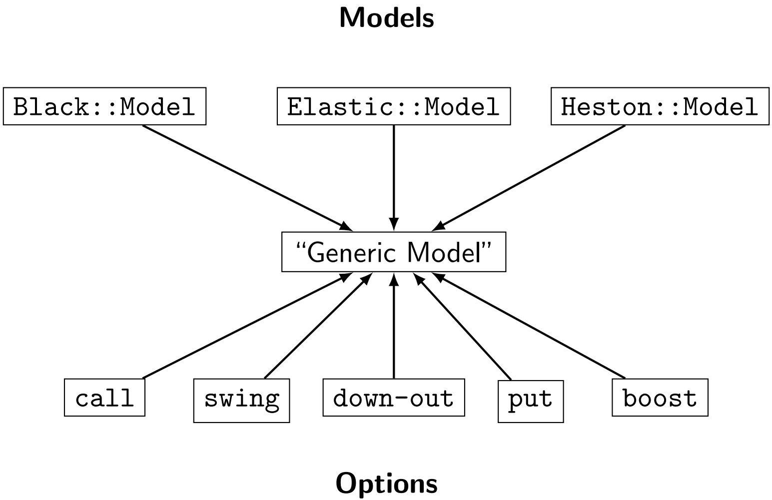
K : strike

T : maturity ($T > t_M$).

The payoff of the option at maturity equals

1. the payoff of the standard call option if the spot price was above the barrier for **all** barrier times t_j;
2. zero if the barrier was crossed at a barrier time.

Design problem: models and options



Rollback operator

$$\boxed{V_s\text{: the AFP at time } s} \xleftarrow{\text{rollback}} \boxed{V_t\text{: a payoff at time } t}$$

Notation:

$$V_s = \mathcal{R}_s\left(V_t\right)$$

Main principle:

$$\boxed{\text{Arbitrage-Free Pricing} = \text{Replication}}$$

Convenient method: (for complete financial models)

$$\boxed{\text{Arbitrage-Free Pricing} = \text{Risk-Neutral Valuation}}$$

Money market measure

Denote

r_t : the short-term interest rate at t.

$B_t = \exp(\int_0^t r_u du)$: the bank account.

The **money market** (martingale) measure $\mathbb{P}^*$ is defined as such a measure that

$$\frac{X_t}{B_t} = X_t \exp\left(-\int_0^t r_u du\right)$$

is a **martingale** under $\mathbb{P}^*$ for any wealth process X:

$$X_s \exp\left(-\int_0^s r_u du\right) = \mathbb{E}_s^*\left(X_t \exp\left(-\int_0^t r_u du\right)\right)$$

$$\Updownarrow$$

$$X_s = \mathbb{E}_s^*\left(X_t \exp\left(-\int_s^t r_u du\right)\right)$$

Risk-neutral valuation

Theorem

The rollback operator has the following representation in terms of the money market measure $\mathbb{P}^*$*:*

$$V_s = \mathcal{R}_s\left(V_t\right) = \mathbb{E}_s^*\left(V_t \exp\left(-\int_s^t r_u du\right)\right)$$

Proof.

Definition of $\mathbb{P}^*$ + "Arbitrage-Free Pricing = Replication". $\square$

Remark

$$\text{Computation of } \mathcal{R}_s\left(\cdot\right) \iff \text{Computation of } \mathbb{E}_s^*\left(\cdot\right)$$

Forward measure

$B(s,t)$: price at s of the zero-coupon bond with face value $\$1$ and maturity t.
The **forward** (martingale) measure $\mathbb{P}^t$ is such a measure that

$$\frac{X_s}{B(s,t)}, \quad 0 \le s \le t,$$

is a **martingale** under $\mathbb{P}^t$ for any wealth process X:

$$\frac{X_s}{B(s,t)} = \mathbb{E}_s^t(X_t)$$

$$\Updownarrow$$

$$X_s = B(s,t)\mathbb{E}_s^t(X_t)$$

Forward measure

The term **forward martingale measure** is due to the fact, that $(F(s,t))_{0\le s\le t}$ is $\mathbb{P}^t$-martingale, where

$F(s,t)$: forward price computed at s for delivery at t.

Indeed, consider long position in the forward contract:

$X_s = 0$: value at s

$X_t = S_t - F(s,t)$: value at t

Then

$$0 = X_s = B(s,t)\mathbb{E}_s^t(X_t) = B(s,t)\mathbb{E}_s^t(S_t - F(s,t))$$

and, hence,

$$F(s,t) = \mathbb{E}_s^t(S_t).$$

Risk-neutral valuation

Theorem

The rollback operator has the following representation in terms of the forward martingale measure $\mathbb{P}^t$:

$$V_s = \mathcal{R}_s[V_t] = B(s,t)\mathbb{E}_s^t[V_t]$$

Proof.

Definition of $\mathbb{P}^t$ + "Arbitrage-Free Pricing = Replication". $\square$

Remark

To implement $\mathcal{R}_s(V_t)$ we need to implement $\mathbb{E}_s^t(V_t)$.

Problem on float rate

Problem

Notations:

$L(s,t)$: *float (LIBOR) rate computed at time* s *for maturity* t

$B(s,t)$: *discount factor computed at time* s *for maturity* t.

Compute (express in terms of $B(s,t)$*) the value at time* s *of the float payment at* t. *Recall that*

$$\text{“Float payment at } t\text{”} = L(s,t)(t-s).$$

Solution

Replicating strategy

$$X_0 \longrightarrow L(s,t)(t-s)$$

is the difference $(a)-(b)$, where

$$1 \quad \xrightarrow[\text{bank}]{} \quad 1 + L(s,t)(t-s), \tag{a}$$

$$B(s,t) \quad \xrightarrow[\text{zero-coupon bond}]{} \quad 1. \tag{b}$$

The initial capital is given by

$$X_0 = 1 - B(s,t).$$

Problem on foreign exchange rates

Problem

Notations:

$F(s,t)$: *forward exchange rate (price of one unit of foreign currency) computed at time* s *for maturity* t

$B(s,t)$: *discount factor (in domestic currency) computed at time* s *for maturity* t.

Compute (express in terms of $F(s,t)$ *and* $B(s,t)$*) the value at time* s *of one unit of foreign currency paid at* t.

Solution

Replicating strategy

$$X_0 \longrightarrow €1$$

is the sum $(a) + (b)$, where

$$0 \xrightarrow[\text{long forward}]{} €1 - F(s,t), \qquad \text{(a)}$$

$$F(s,t)B(s,t) \xrightarrow[F(s,t)\text{ zero-coupon bonds}]{} F(s,t). \qquad \text{(b)}$$

The initial capital is given by

$$X_0 = F(s,t)B(s,t).$$

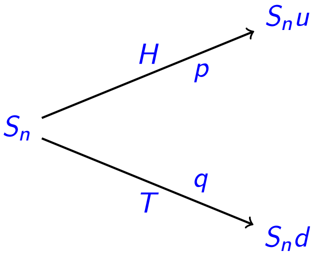
State processes

Idea: efficient storage for relevant random variables.

Consider, for example, **binomial model** with parameters:

u: relative change "up"

d: relative change "down"



Binomial model

Consider payment at $n+1$:

$$V_{n+1} = V_{n+1}(\omega_1, \dots, \omega_{n+1})$$

Rollback operator between $n+1$ and n has the form:

$$\begin{aligned} V_n &= \mathcal{R}_n\left(V_{n+1}\right)(\omega_1, \dots, \omega_n) \\ &= \frac{1}{1+r}\left(\widetilde{p}V_{n+1}(\omega_{n+1} = H) + \widetilde{q}V_{n+1}(\omega_{n+1} = T)\right) \end{aligned}$$

where $\widetilde{p}$ and $\widetilde{q}$ are the one-step risk neutral probabilities:

$$\widetilde{p} = \frac{1+r-d}{u-d}, \quad \widetilde{q} = 1 - \widetilde{p}.$$

“Naive” storage

“Naive” storage scheme: record values of

$$V_n = V_n(\omega_1, \ldots, \omega_n)$$

for **all** $(\omega_1, \ldots, \omega_n)$. Then

$$\# \text{ of records } = 2^n \text{ (too big!)}$$

The naive (universal) storage scheme is not practical for already $n \approx 100$.

Idea: the storage should be *adapted* to the *type* of derivative security we want to evaluate.

Storage for standard options

For example, to price *standard options* it is sufficient to operate with random variables in the form

$$V_n = f(S_n),$$

where $f = f(x)$ is a deterministic function. In this case,

$$\# \text{ of records } = n + 1 \text{ (fine!)}$$

Indeed, if $V_{n+1} = f_{n+1}(S_{n+1})$ then

$$V_n = \mathcal{R}_n\left(V_{n+1}\right) = f_n(S_n)$$

where $f_n(x) = \frac{1}{1+r}\left(\widetilde{p}f_{n+1}(ux) + \widetilde{q}f_{n+1}(dx)\right)$.

State processes

The spot price process $(S_n)_{0 \le n \le N}$ in binomial model is an example of a *state process*.

Definition

A process $(X_t)_{0 \le t \le T}$ is called a **state process** if $\forall s < t$ and any deterministic function $f = f(x)$ there is a deterministic function $g = g(x)$ such that

$$g(X_s) = \mathcal{R}_s \left(f(X_t) \right).$$

State processes

Remark

For a stochastic process $X = (X_t)_{0 \leq t \leq T}$ and time t denote by

$$\mathcal{X}_t = \{f(X_t) : \quad f \text{ is deterministic function } \}$$

the family of random variables determined by (measurable with respect to) X_t. We have that

For arbitrary X: for any time t the family $\mathcal{X}_t$ is *closed* under any arithmetic or functional operation

For state process X: for two times $s < t$ the families $\mathcal{X}_s$ and $\mathcal{X}_t$ are *closed* under the rollback operator $s \leftarrow t$: for any $V_t \in \mathcal{X}_t$

$$V_s = \mathcal{R}_s\left(V_t\right) \in \mathcal{X}_s$$

Markov processes

Definition

A stochastic process $X = (X_t)_{0 \leq t \leq T}$ is called a **Markov process** if for any $s < t$ and any deterministic $f = f(x)$ there is a deterministic $g = g(x)$ such that

$$g(X_s) = \mathbb{E}_s\left(f(X_t)\right)$$

Remark (Intuitive definition)

At time t the future behavior of X (the distribution of $(X_s)_{t \leq s}$) is completely determined by the *current value* X_t (does not depend on the particular trajectory between times 0 and t).

State and Markov processes

Theorem

The following conditions are equivalent:

1. X *is a state process*
2. *for any maturity* T
   2.1 $(X_t)_{0\leq t\leq T}$ *is a Markov process under the forward measure* $\mathbb{P}^T$
   2.2 *the discount factor computed at* t *for maturity* T *is determined by* X_t*, that is,*

   $$B(t,T) = f(X_t)$$

   for some deterministic $f = f(x)$

Proof.

Formula for rollback operator:

$$\mathcal{R}_t(V_T) = B(t,T)\mathbb{E}_t^T(V_T).$$

□

"Implementation" of a financial model

An "implementation" of a financial model consists of

1. Specification of a state process X (the choice of state process is determined by the type of derivative security).
2. Implementation of all necessary operations for random variables from the classes

$$\mathcal{X}_t = \{f(X_t) : \; f \text{ is a deterministic function } \}, \; t > t_0.$$

   2.1 For given time t: all arithmetic and functional operations
   2.2 Between two times $s < t$: rollback operator.

Part II

Design of pricing library with C++

Outline

Basic componenets: state process and event times

Single asset models

Black model in `cfl`

Basic components

State process:

$$X = (\underbrace{X^0, \dots, X^{d-1}}_{d-\text{dimensional}})$$

Vector of event times: (initial time $=$) $t_0 < t_1 < \dots < t_N$

At an event time t_i we operate with random variables:

$$\mathcal{X}_{t_i} = \left\{ f(X_{t_i}) = f(X^0_{t_i}, \dots, X^{d-1}_{t_i}) : \quad f = f(x^0, \dots, x^{d-1}) \right\}.$$

A random variable $f(X_{t_i})$ is represented by the class `cfl::Slice`.

Event times

Any model in `cfl` has *discrete* time structure:

$(t_i)_{0 \leq i \leq M}$: **sorted** vector of **event times** given as year fractions;

t_0: *initial time*.

Event times: all times needed to price a *particular* derivative security. Examples:

1. Exercise times
2. Maturity
3. Barrier times
4. Reset times

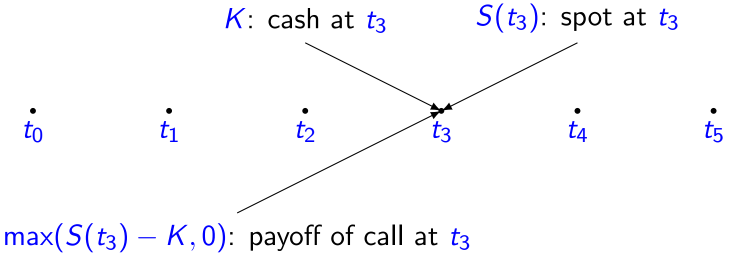
Numerical efficiency: create the vector of event times with a *smallest* size.

cfl::Slice

The main class in the library is `cfl::Slice`.

Basic idea: `cfl::Slice` represents the value of a (derivative) security at a **particular** event time.

Precise defintion: `cfl::Slice` describes random variables in the form $f(X(t_i))$, where $f = f(x)$ is a deterministic function, X is a state process and t_i is an event time.



`cfl::Slice`

There are 2 types of operations for `cfl::Slice`:

1. At given event time t_i: all possible arithmetic, functional, etc.. For example, if

   `uSpot`: `cfl::Slice` for the spot price $S(t_i)$ at t_i

   `dK`: `double` for the cash amount K at t_i

   then

   ```
   Slice uCall = max(uSpot - dK, 0.);
   ```

   creates `cfl::Slice` for the payoff

   $$\max(S(t_i) - K, 0)$$

   of the call option with strike K and maturity t_i.

`cfl::Slice`

However, if

$t_1 < t_2$: event times

`uSpot1`: `cfl::Slice` for the spot price $S(t_1)$ at t_1

`uSpot2`: `cfl::Slice` for the spot price $S(t_2)$ at t_2

then the following code

```
Slice uSum = uSpot1 + uSpot2
```

is **wrong** (should not compile)!

Remark

You might want to create a `cfl::Slice` for $S(t_1) + S(t_2)$, (say, to price an Asian option), but this operation is not allowed!

cfl::Slice

There are 2 types of operations for `cfl::Slice`:

2. Between two event times $t_i < t_j$: only **rollback** operator.

$$\boxed{\text{AFP at } t_i} \xleftarrow{\text{rollback}} \boxed{\text{Payoff at time } t_j}$$

Algorithm for pricing of standard call:

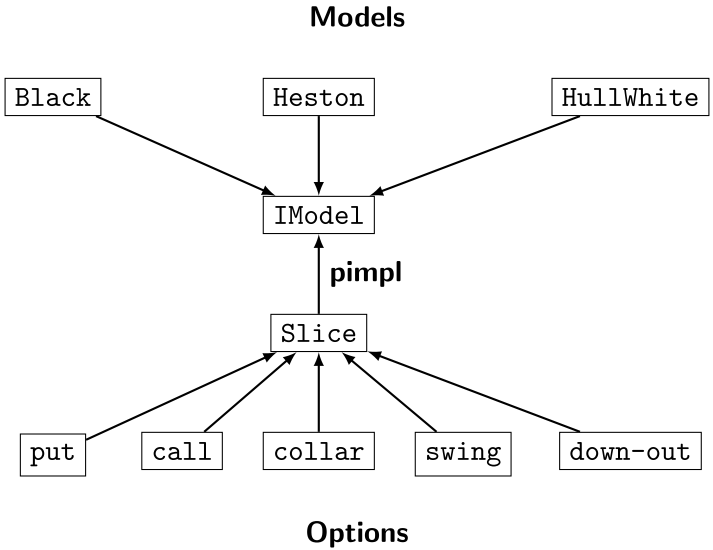
```
...
//two event times:  0 (initial) and 1 (maturity)
Slice uCall = max(uModel.spot(1) - dK, 0);
uCall.rollback(0);
```

Typical program flow

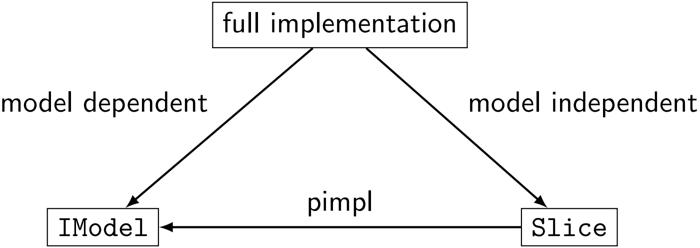
1. Basic objects of the type `cfl::Slice` such as
   1.1 spot prices
   1.2 discount factors, etc.

   are created by an implementation of a particular financial model
2. We then manipulate these basic objects using the provided operators and functions:
   2.1 for given event time: all arithmetic and functional operations;
   2.2 between two event times: rollback operator.

Design of `cfl`: `IModel` and `Slice`



Architecture of `cfl`



`IModel`: abstract (interface) class that defines model-specific behavior of `Slice`.

`Slice`: concrete class for random variables $f(X_{t_k})$, where

$f = f(x)$: deterministic function,

t_k: event time,

X: state process.

`cfl::Slice` and `cfl::IModel`

The class `Slice` represents the random variable in the form: $f(X^{i_1}_{t_k}, \dots, X^{i_m}_{t_k})$.

Components (private members) of `Slice`:

1. array of values (`std::valarray<double>`): discretization of $f = f(x^{i_1}, \dots, x^{i_m})$
2. vector of dependences: $i_1, \dots, i_m$
3. (index of) current event time t_k
4. **pimpl** of `IModel`

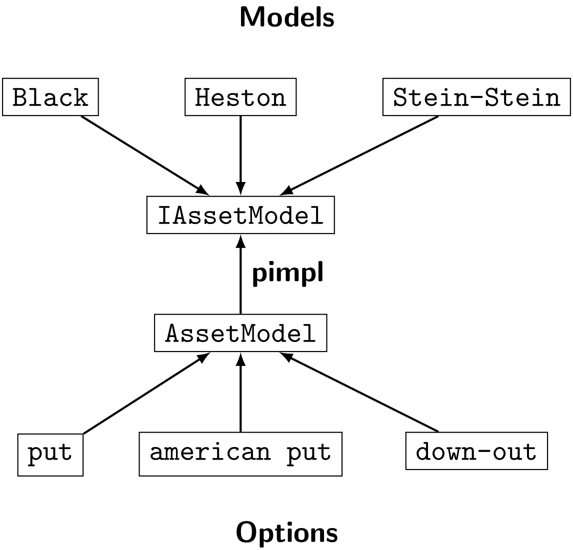
The interface class `IModel` contains declarations of **model-specific** functions needed to define the behavior of the class `Slice`.

Single asset model in `cfl`

A single asset model in `cfl` is represented by the "universal" class `cfl::AssetModel`.

- The main idea is to "separate" the evaluation of derivatives from the implementations of financial models.
- Creation of basic payoffs (basic `Slice` objects):
  - cash payments,
  - spot prices,
  - forward prices,
  - discount factors.
- It is constructed from the implementation on a free store of the interface class `cfl::IAssetModel`.
- The interface class `cfl::IAssetModel` and the concrete class `cfl::AssetModel` are related by **pimpl** idiom.

Design with `pimpl`: models and options on a stock



Work with `cfl::AssetModel`

Step 1: construct the vector of event times

$$t_0, t_1, \dots, t_M,$$

where t_0 is the initial time and $t_1, \dots, t_M$ depend on the option.

Goal: make the vector of the event times as small as possible.

Step 2: write the pricing algorithm using

1. Basic payoffs (instances of `cfl::Slice`) produced by `cfl::AssetModel`.
   1.1 cash amounts at event times.
   1.2 spot prices at event times.
   1.3 discount factors at event times for any maturity.
   1.4 forward prices at event times for any maturity.
2. Operations on instances of `cfl::Slice`.
   2.1 At given event time: everything.
   2.2 Between event times: rollback.

Black model in `cfl`

Generalized (and standard) Black model is implemented in the namespace `cfl::Black`:

1. The class `cfl::Black::Data` defines the parameters of the Black model. Recall, that the set of parameters consists of
   - initial time (as year fraction),
   - discount curve,
   - forward curve,
   - volatility curve,
   - "shape" curve.
2. The functions `cfl::Black::model` implement the *"standard"* single asset model in the library, namely, the class `cfl::AssetModel`.

Output for Black model

It is important (for example, for risk-management) to compute the value of an option in Black model as the function of **relative change** in the price of the stock:

$$V = (V(x))_{\frac{\Delta}{2} \leq x \leq \frac{\Delta}{2}}$$

where

$V(x)$: the price of the option corresponding to the scenario that the spot price changes by x percents

x: relative change in the spot price

Δ: width of the interval for relative changes.

Output for Black model