Financial Computing (with C++)

Instructor : Dmitry Kramkov

TA : Gerard Brunick

Communication: Blackboard

Materials:

(a) cfl library + Examples

(b) Lecture Notes

(c) Optional: Books on C++ (with STL library)

(i) Stroustrup

(ii) Josuttis

2

Requirements:

1. Notebook with MSVisualStudio .Net
2. "Basic" knowledge of relevent areas:

a) C++

b) Stochastic Analysis

c) Finance (Derivatives, Arbitrage-Free Pricing)

d) Numerical Analysis

Course work:

	Home works	Exam
#	4 (best 3)	3
% of grade	50%	50%
Collab.	YES	NO

3

Results from previous years:

YEAR 2002
# of students = 33

# of solved problems	3	2	1	0
# of students	3	9	9	12

YEAR 2003
# of students = 55

# of solved problems	3	2	1	0
# of students	3	22	12	16

Goals of the course:

A: "Theoretical" — review and expand your knowledge of

1) OOP with C++
2) Mathematical Finance
3) Stochastic Calculus
4) Numerical Analysis

B: "Practical" — improve (and test!) your ability to use C++ for financial computations.

Method of study: 5

A: "Theoretical" – discuss design and implementation of a C++ library for pricing of derivatives (case study = cfl library)

B: "Practical" – use cfl library for pricing of concrete (quite complicated!) derivatives.

Classification of derivatives.

1. By the type of underlying

   - Assets
     - stocks
     - ⋮
     - fx rates
     - commodities
   - Interest rates

2. By the dependence on the history:

   a) Standard: payoff depends on the current value

   b) Path dependent: payoff depends on historical values

   c) Barrier: simple dependence on the past

Q: What item is important for practical implementations?

1. Type of underlying — NO
2. Dependence on history — YES

18

Standard option for an asset

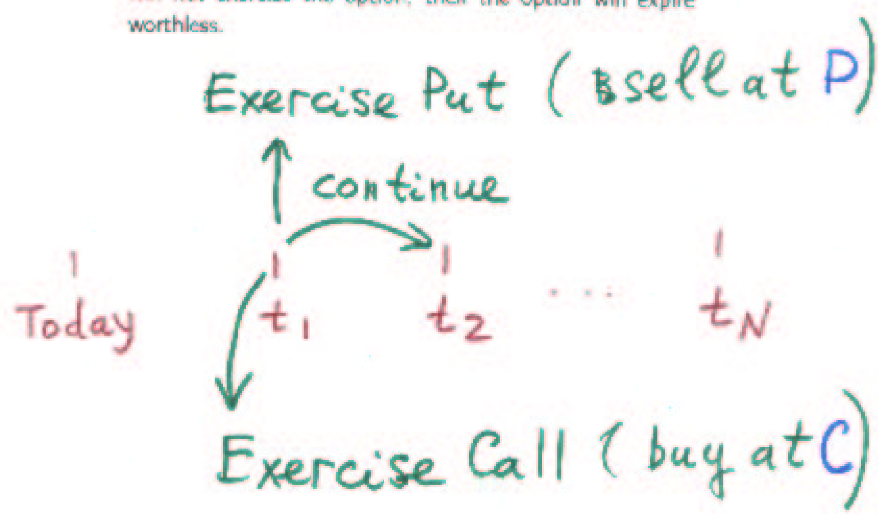
American put-call

P : strike for put option

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

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

A holder of the option can exercise it at any time t_i by selling the stock for the strike of put option P or buying the stock for the strike of call option C. If the holder will not exercise the option, then the option will expire worthless.



Standard option on interest rates

American swaption

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

Parameters of underlying swap:

N : notional.

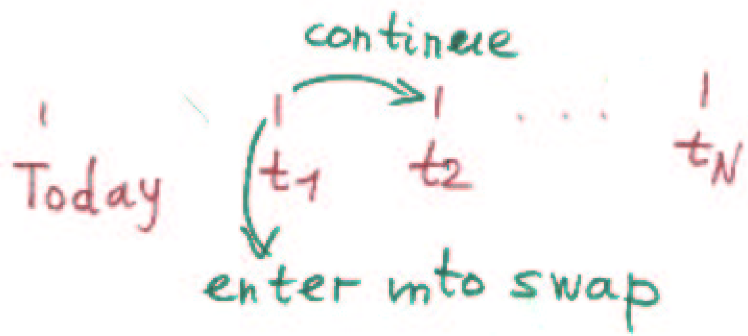
R : fixed rate

δt : interval of time between the payments given as year fraction.

m : total number of payments

side : this parameter defines the side of the swap contract, i.e. whether one pays "fixed" and receives "float" or otherwise.

A holder of the option can enter into the underlying swap agreement at any exercise time t_i. This time will become after that the issue time of the swap.



Barrier option for an asset

Down-and-out call

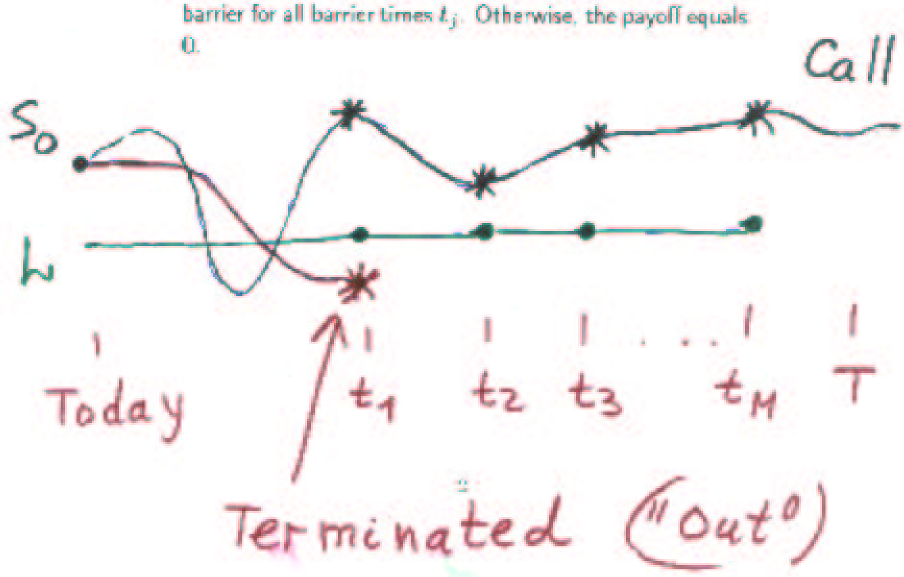
L : lower barrier

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

K : strike

T : maturity ($T > t_M$).

The payoff of the option at maturity equals the payoff of the standard call option if the spot price was above the barrier for all barrier times t_j. Otherwise, the payoff equals 0.



Path dependent option on an asset

Convertible bond

Underlying coupon bond :

N : notional

R : coupon rate

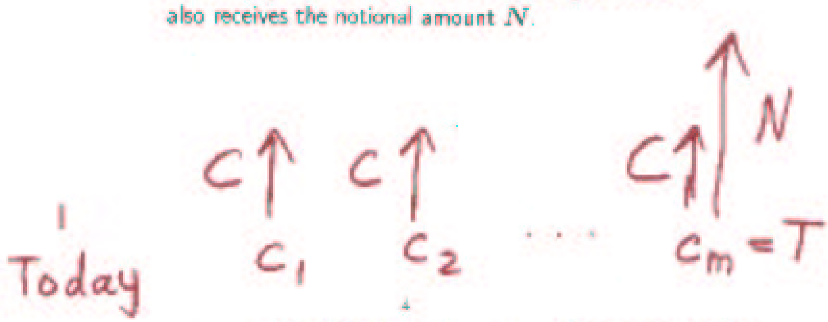
δt : interval of time between the payments given as year fraction.

m : total number of coupon payments.

The coupon times of the bond equal

$$c_i = t_0 + i\delta t, \quad 1 \leq i \leq m,$$

where t_0 is the issue time for the bond. At any of coupon times the holder of the bond receives the coupon payment $NR\delta t$. At the last coupon time he also receives the notional amount N.



$C = NR\delta t$: coupon payment

Resettable strike :

K : initial strike.

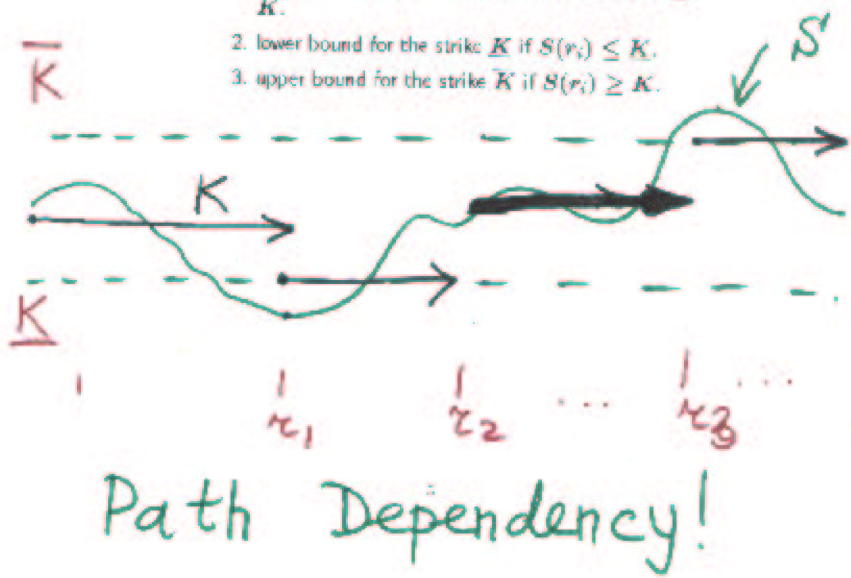
$\underline{K}$: lower bound for the strike

$\overline{K}$: upper bound for the strike

(τ_i) : reset times for the strike.

At time τ_i the value of the strike is reset to

1. spot price $S(\tau_i)$ of the stock if $\underline{K} \le S(\tau_i) \le \overline{K}$.
2. lower bound for the strike $\underline{K}$ if $S(\tau_i) \le \underline{K}$.
3. upper bound for the strike $\overline{K}$ if $S(\tau_i) \ge \overline{K}$.



13

Exercise option for bond holder :

(t_i) : exercise times for the bond.

At time t_i the holder of the bond can convert it to $N/K(t_i)$ shares of stock, where

1. $K(t_i)$ equals initial strike K if there are no reset times before t_i, i.e. $t_i < r_1$.
2. $K(t_i)$ equals the reset value at the largest reset time r_k which is less or equal t_i.

convert into $N/K(t_1)$ shares

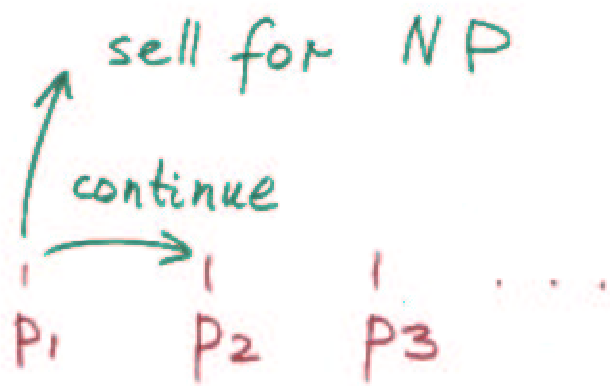
continue

$t_1 \quad t_2 \quad \cdots \quad t_N$

Put option for bond holder :

P : put redemption price, $P < 1$. At a put redemption time the holder of the bond can sell it back to issuer for put redemption amount NP.

(p_i) : put redemption times, i.e. those times when the bond holder can sell the bond back to issuer for the put redemption amount.



15

Hard call option for bond issuer :

H : hard call redemption price, $H > 1$. The issuer can buy the bond back for hard call redemption amount NH at a hard call redemption time.

(h_i) : hard call redemption times.

option for issuer

Buy back at NH.

continue

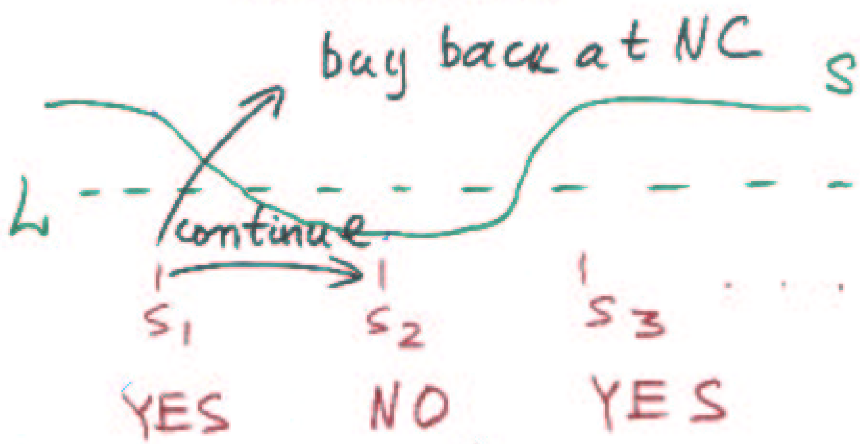
$h_1 \quad h_2 \quad h_3 \quad \dots \quad h_i \quad \dots$

Soft call option for bond issuer :

L : soft call lower barrier. The issuer of the bond can buy back the bond for the soft call redemption amount at a soft call redemption time only if the price of the stock at this time is greater than L.

C : soft call redemption price, $C > 1$. The issuer can buy the bond back for soft call redemption amount NC at a soft call redemption time only if the stock price at this time is greater that the soft call lower barrier.

(s_i) : soft call redemption times.



Option for bond issuer at maturity :

M : mandatory strike.

At maturity the issuer of the bond has the right to deliver N/M stocks to the bond holder instead of paying the notional N.

option for issuer

pay N   deliver N/M stocks

T (maturity)

$$\min\left(N, \frac{N}{M} S_T\right)$$

payoff at maturity

10

18

We consider two classical models:

1. Black model for a single asset
2. Hull-White model for interest rates.

Remark Both models will be implemented in their most general forms.

For example, our Black model will include as particular cases: 19

a) Classicar Black & Scholes model:

$r = \text{const}$

$dS_t = S_t(r\,dt + \sigma\,dW_t)$

b) Time dependent version:

Interestrate $(r_t)_{t\geq 0}$

$dS_t = S_t(r_t\,dt + \sigma_t\,dW_t)$

c) FX model

r^d : domestic rate

r^f : foreign rate

20

FX rate:

$$dS_t = S_t\left((r^d - r^f)\,dt + \sigma\,dW_t\right)$$

d) Commodity model

$$dS_t = S_t\left\{ (\theta_t - \lambda \ln S_t)\,dt + \sigma\,dW_t \right\}$$

λ: mean reversion coefficient

KEY IDEA: write the model in terms of forward prices!

$F(t,T)$: forward price

current time

maturity of forward

$$dF(t,T) = F(t,T)\, A(T)\, \sigma(t)\, dW_t$$

$A = \big(A(T)\big)_{T \geq 0}$: "shape" function

$A \equiv 1$: stocks + fx

$A(T) = e^{-\lambda T}$: commodities

constant mean-reversion

cfl::Black::Data Class Reference

[Black model for a single asset.]

The parameters of **Black** model. More...

Public Methods

Data ()
Data (const **Function** &rDiscount, const **Function** &rForward, const **Function** &rVolatility, double dInitialTime)
Data (const **Function** &rDiscount, const **Function** &rForward, double dSigma, double dInitialTime)
Data (const **Function** &rDiscount, const **Function** &rForward, const **Function** &rVolatility, const **Function** &rShape, double dInitialTime)
Data (const **Function** &rDiscount, const **Function** &rForward, double dSigma, double dLambda, double dInitialTime)
double **initialTime** () const
const **Function** & **discount** () const
const **Function** & **forward** () const
const **Function** & **volatility** () const
const **Function** & **shape** () const

Detailed Description

This class defines the parameters of the **Black** model. The set of parameters consists of discount, forward, shape and volatility curves.

See also:
cfl::Black::Model

Constructor & Destructor Documentation

cfl::Black::Data::Data( ) [inline]

Default constructor.

cfl::Black::Data::Data(const Function & *rDiscount*,
const Function & *rForward*,
const Function & *rVolatility*,
double *dInitialTime*

)

Constructs parameters of classical **Black** model.

Parameters:

rDiscount	A discount curve.
rForward	A forward curve.
rVolatility	A volatility curve.
dInitialTime	Initial time given as year fraction.

```
cfl::Black::Data::Data( const Function &  rDiscount,
                        const Function &  rForward,
                        double            dSigma,
                        double            dInitialTime
                      )
```

Constructs parameters of classical **Black** model with constant volatility.

Parameters:

rDiscount	A discount curve.
rForward	A forward curve.
dSigma	A volatility.
dInitialTime	Initial time given as year fraction.

```
cfl::Black::Data::Data( const Function &  rDiscount,
                        const Function &  rForward,
                        const Function &  rVolatility,
                        const Function &  rShape,
                        double            dInitialTime
                      )
```

Constructs parameters of general **Black** model.

Parameters:

rDiscount	A discount curve.
rForward	A forward curve.
rVolatility	A volatility curve.
rShape	A shape function. This function defines the shape of movements of the curve of forward prices.
dInitialTime	Initial time given as year fraction.

```
cfl::Black::Data::Data( const Function &  rDiscount,
                        const Function &  rForward,
                        double            dSigma,
                        double            dLambda,
                        double            dInitialTime
                      )
```

Constructs parameters of general **Black** model with stationary volatility.

Parameters:

rDiscount	A discount curve.
rForward	A forward curve.
dSigma	The volatility of spot price
dLambda	The mean-reversion coefficient for log of spot price under the risk neutral measure.
dInitialTime	Initial time given as year fraction.

Member Function Documentation

const Function& cfl::Black::Data::discount( ) const

Returns:
The discount curve.

const Function& cfl::Black::Data::forward( ) const

Returns:
The forward curve.

double cfl::Black::Data::initialTime( ) const

Returns:
The initial time as year fraction.

const Function& cfl::Black::Data::shape( ) const

Returns:
The shape curve for movements of forward prices.

const Function& cfl::Black::Data::volatility( ) const

Returns:
The volatility curve.

The documentation for this class was generated from the following file:

- **Black/Data.h**

Generated on Wed Oct 20 15:55:32 2004 for CFL by doxygen 1.3.8

Data curves for financial models.

Functions

Function **discount** (double dYield, double dInitialTime)
Function **discount** (const Function &rYield, double dInitialTime)
Function **volatility** (double dSigma, double dLambda, double dInitialTime)
Function **forward** (double dSpot, double dCostOfCarry, double dInitialTime)
Function **forward** (double dSpot, const Function &rCostOfCarry, double dInitialTime)
Function **forward** (double dSpot, double dDividendYield, const Function &rDiscount, double dInitialTime)
Function **assetShape** (double dLambda, double dInitialTime)
Function **bondShape** (double dLambda, double dInitialTime)

Detailed Description

This module contains implementations of different data curves for financial modes such as discount, forward, volatility, etc. .

Function Documentation

Function assetShape(double ***dLambda*****,**
double ***dInitialTime***
)

Stationary form for changes in forward price curves. The value of this function at time dT given as year fraction equals

```
exp(-dLambda*(dT - dInitialTime))
```

.

Parameters:

dLambda The mean-reversion rate.
dInitialTime The initial time as year fraction

Function bondShape(double ***dLambda*****,**
double ***dInitialTime***
)

Stationary form for changes in yield to maturity and discount factor curves. The value of this function at time dT given as year fraction equals

```
(1-exp(-dLambda*(dT - dInitialTime)))/dLambda
```

Parameters:

dLambda The mean-reversion rate.
dInitialTime The initial time as year fraction.

Function discount(const Function & *rYield*, double *dInitialTime*)

Constructs discount curve. The discount factor for maturity `dMaturity` given as year fraction equals

```
exp(-rYield(dMaturity)*(dMaturity - dInitialTime))
```

.

Parameters:

rYield The continuously compounded yield curve.
dInitialTime The initial time as year fraction.

Returns:

The discount curve as function of maturity.

Function discount(double *dYield*, double *dInitialTime*)

Constructs discount curve. The discount factor for maturity `dMaturity` given as year fraction equals

```
exp(-dYield*(dMaturity - dInitialTime))
```

.

Parameters:

dYield The constant continuously compounded yield.
dInitialTime The initial time as year fraction.

Returns:

The discount curve as function of maturity.

Function forward(double *dSpot*, double *dDividendYield*, const Function & *rDiscount*, double *dInitialTime*)

Constructs forward curve for a stock. The forward price at maturity `dMaturity` given as year fraction equals

```
dSpot * exp(-dDividendYield*(dMaturity - dInitialTime))/rDiscount(dMaturity)
```

.

Parameters:

dSpot	The spot price.
dDividendYield	The dividend yield for the stock.
rDiscount	The current discount curve.
dInitialTime	The initial time as year fraction.

Function forward(double *dSpot*,
const Function & *rCostOfCarry*,
double *dInitialTime*
)

Constructs forward curve for an asset. The forward price at maturity `dMaturity` equals

```
dSpot * exp(rCostOfCarry(dMaturity)*(dMaturity - dInitialTime));
```

Parameters:

dSpot	The spot price.
dCostOfCarry	The cost-of-carry rate curve.
dInitialTime	The initial time as year fraction.

Function forward(double *dSpot*,
double *dCostOfCarry*,
double *dInitialTime*
)

Constructs forward curve for an asset. The forward price at maturity `dMaturity` equals

```
dSpot * exp(dCostOfCarry*(dMaturity - dInitialTime));
```

Parameters:

dSpot	The spot price.
dCostOfCarry	The cost-of-carry rate.
dInitialTime	The initial time as year fraction.

Function volatility(double *dSigma*,
double *dLambda*,
double *dInitialTime*
)

Constructs stationary volatility curve. The value of volatility for time `dT` given as year fraction equals

```
dSigma * sqrt((1-exp(-2*m_dLambda*(dT-dInitialTime)) / (2*m_dLambda*(dT-dInitialTime)));
```

Parameters:

dSigma	The short-term volatility
dLambda	The mean-reversion coefficient.
dInitialTime	The initial time as year fraction

Overview of cfl library

1 Any model has discrete or finite time structure.

$(t_i)_{0 \le i \le M}$: sorted vector of event times as year fractions.

t_0 : initial time

Examples :

a) exersise times

б) barrier times

Event times = All times we need to price given derivative.

Efficiency : create
the vector of event
times as small as
possible

30

2. The main class is
<u>Slice</u>: represents the value of security at event time.

K: cash at t_2

$S(t_2)$: spot price at t_2

t_1 t_2 t_3 t_4 t_5

$\max(S(t_2)-K,0)$: call's payoff at t_2

31

2 types of operations for Slice

1 At given event time t_i : all possible arithmetic, functional, ...

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

Example

t_i : event time

uSlice1 : $S(t_i)$ spot at t_i

uSlice2 : K cash at t_i

Slice uSlice3 =
max(uSlice1-uSlice2, 0.)

uSlice3 : payoff of call
at t_i
$\max(S(t_i) - K, 0.)$

Wrong code

t_i : event time
uSlice1 : spot at t_i
t_j : another event time
uSlice2 : spot at t_j

```
Slice uSlice3 =
  uSlice1 + uSlice2;
```

You want
uSlice3 : $S(t_i) + S(t_j)$
However this is not allowed!

34

Q: How we create needed Slices?

1. Basic Slices are created by a model

2. Manipulate using
   a) arithmetic & functional operations for particular time
   b) rollback between different event times.

cfl::Black::Model Class Reference

[Black model for a single asset.]

Implementation of **Black** model. More...

Public Methods

Model (const **Data** &rData, const std::vector< double > &rEventTimes, double dInterval, double dQuality)
Model (const **Data** &rData, const std::vector< double > &rEventTimes, double dInterval, double dQuality, double dPathDependQuality)
Model (const **Data** &rData, const std::vector< double > &rEventTimes, double dInterval, const std::vector< double > &rQuality)
Model (const **Data** &rData, const std::vector< double > &rEventTimes, double dInterval, const cfl::**Brownian** &rBrownian)
const **Data** & **data** () const
unsigned **addState** (const **PathDependent** &rState)
Slice state (unsigned iTime, unsigned iState) const
Slice cash (unsigned iTime, double dAmount) const
Slice discount (unsigned iTime, double dBondMaturity) const
Slice spot (unsigned iTime) const
Slice forward (unsigned iTime, double dForwardMaturity) const

Detailed Description

This class implements **Black** model for a single asset.

See also:
cfl::Black::Data

Constructor & Destructor Documentation

cfl::Black::Model::Model(const Data & *rData*,
const std::vector< double > & *rEventTimes*,
double *dInterval*,
double *dQuality*
)

Constructs an implementation of the model. Use this constructor for pricing of standard derivatives.

Parameters:

rData	The input data of the model.
rEventTimes	The vector of event times.
dInterval	The width of the interval of initial values for the main state process.
dQuality	The quality of numerical implementation of the model. This parameter defines the trade-off between speed and accuracy.

```
cfl::Black::Model::Model( const Data &                  rData,
                          const std::vector< double > & rEventTimes,
                          double                        dInterval,
                          double                        dQuality,
                          double                        dPathDependQuality
                        )
```

Constructs an implementation of the model. Use this constructor for pricing of path dependent derivatives with one additional state process.

Parameters:

rData	The input data of the model.
rEventTimes	The vector of event times.
dInterval	The width of the interval of initial values for the main state process.
dQuality	The quality of numerical implementation of the basic state process (brownian motion).
dPathDependQuality	The quality of numerical implementation of additional state process.

```
cfl::Black::Model::Model( const Data &                  rData,
                          const std::vector< double > & rEventTimes,
                          double                        dInterval,
                          const std::vector< double > & rQuality
                        )
```

Constructs an implementation of the model. Use this constructor for pricing of path dependent derivatives with a number of additional state processes.

Parameters:

rData	The input data of the model.
rEventTimes	The vector of event times.
dInterval	The width of the interval of initial values for the main state process.
rQuality	The element with index i of the vector *rQuality* defines the quality of numerical implementation of the state process with index i.

```
cfl::Black::Model::Model( const Data &                  rData,
                          const std::vector< double > & rEventTimes,
                          double                        dInterval,
                          const cfl::Brownian &         rBrownian
                        )
```

The basic constructor of the model.

Parameters:

rData	The input data of the model
rEventTimes	The vector of event times.
dInterval	The width of the interval of initial values for the main state process.
rBrownian	A reference to an object of type **cfl::Brownian**. This object is used to implement [illegible].

Member Function Documentation

unsigned cfl::Black::Model::addState(const PathDependent & *rState*)

Adds to the model another state process.

Parameters:

rState The description of a path dependent process which becomes additional state process in the model.

Returns:

The index of the additional state process.

Slice cfl::Black::Model::cash(unsigned *iTime*,
double *dAmount*
) const

Returns the representation of the constant value *dAmount* at the event time with index *iTime*.

const Data& cfl::Black::Model::data( ) const

A reference to the input data of the model.

Slice cfl::Black::Model::discount(unsigned *iTime*,
double *dBondMaturity*
) const

Returns the discount factor at the event time with index *iTime* for maturity *dBondMaturity*.

Slice cfl::Black::Model::forward(unsigned *iTime*,
double *dForwardMaturity*
) const

Returns the forward price of the asset at the event time with index *iTime* for maturity *dForwardMaturity*.

Slice cfl::Black::Model::spot(unsigned *iTime*) const

Returns the spot price of the asset at the event time with index *iTime*.

Slice cfl::Black::Model::state(unsigned *iTime*,

unsigned [illegible] State
const

Returns the representation of the state process with index [illegible] at the event time with index [illegible]

The documentation for this class was generated from the following file:

- BlackModel.h

Generated on [illegible] by doxygen [illegible]

Output for Black model

39

We compute the value of the option as the function of relative change in the spot price:

perturbed price

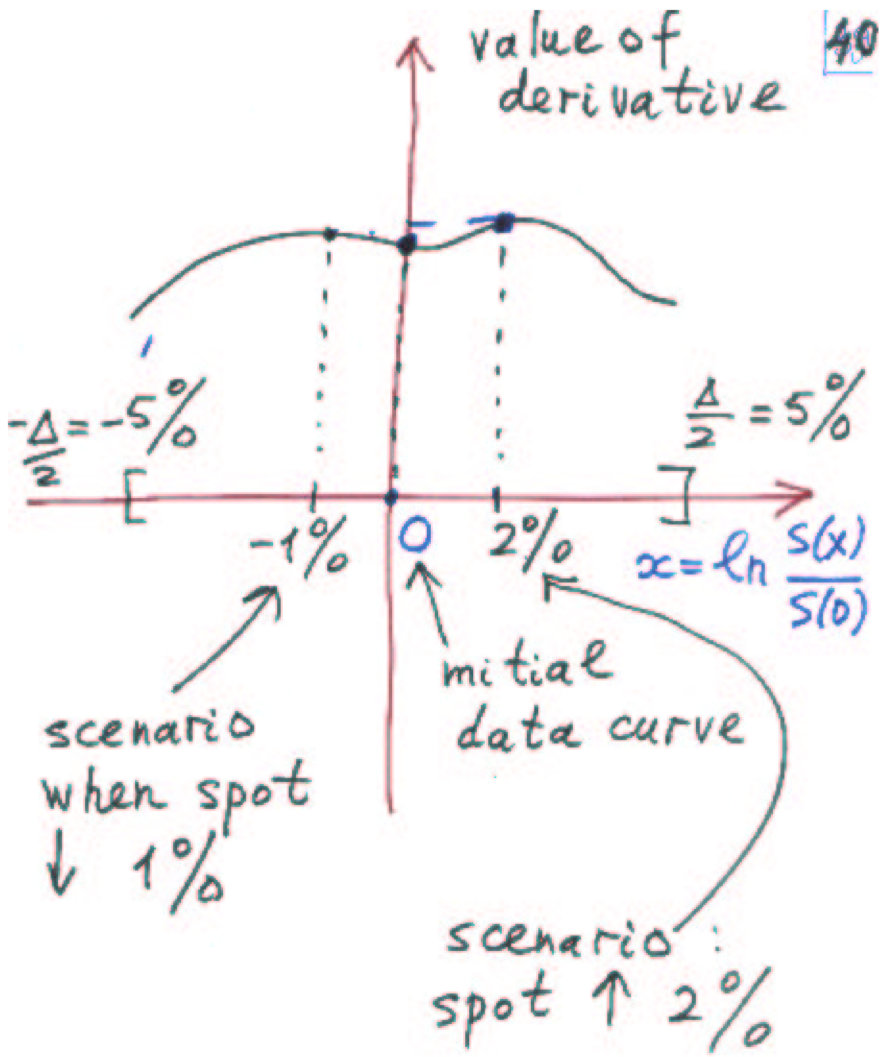
$$x = \ln \frac{S(x)}{S(0)}$$

initial price

Output:

$$V_0 = \left(V_0(x)\right)_{-\frac{\Delta}{2} \le x \le \frac{\Delta}{2}}$$

Δ: interval for relative changes



40
value of derivative
$-\frac{\Delta}{2} = -5\%$
$\frac{\Delta}{2} = 5\%$
-1%
0
2%
$x = \ln \frac{S(x)}{S(0)}$
initial data curve
scenario when spot ↓ 1%
scenario: spot ↑ 2%

Risk parameters:

delta

$$\Delta = \frac{\partial}{\partial x} V_0(x) \Big|_{x=0}$$

$$\Gamma = 10^{-2} \frac{\partial^2 V(x)}{\partial x^2} \Big|_{x=0}$$

1% gamma