dxLibOptions Library Overview

June 2018

Information in this document is subject to change without notice and does not represent a commitment on the part of dxLibOptions.

Please visit www.dxfeed.com for latest information updates.

Contents

1. dxLibOptions 4
2. Option pricing algorithms 4
   2.1. Binomial Tree Pricing 4
   2.2. Bjerksund-Stensland Pricing 5
   2.3. Black-Scholes Pricing 5
   2.4. Black-Scholes Universal Pricing 5
   2.5. Explicit Finite Difference Pricing 6
   2.6. Merton-Reiner-Rubinstein Barrier Pricing 6
   2.7. Monte-Carlo Pricing 6
3. Other algorithms 7
   3.1. Finite difference derivative 7
   3.2. Yield curve 7
4. Examples 7
   4.1. Pricing vanilla option – Black-Scholes 7
   4.2. Pricing vanilla option – Bjerksund-Stensland 8
   4.3. Pricing double barrier option 8
   4.4. Building yield curve 8

1. dxLibOptions

dxLibOptions is a library that provides several arb-free pricing algorithms for options, as well as various helper algorithms. Below is short summary for provided algorithms as well as pointers to detailed descriptions and proofs.

2. Option pricing algorithms

2.1. Binomial Tree Pricing

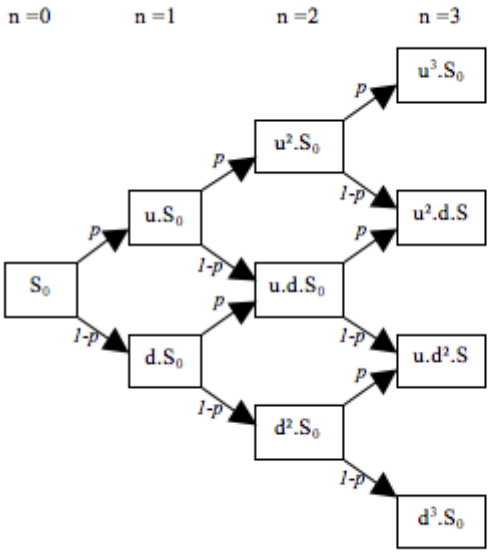
Class: com.devexperts.options.pricing.BinomialTreePricing

Directly provided results: price, delta, gamma.

Suitable for: log normal vanilla American and European options without dividend schedule.

Complexity: O(num_steps^2).

Algorithm divides time until expiration into given number of equal segments. During each segment stock goes either up or down in price. There are two methods to calculate prices and probabilities: Cox-Ross-Rubinstein and Leisen-Reimer. Cox-Ross-Rubunstein tree example:



$$p = \frac{e^{rt/n} - d}{u - d}$$

$$u = e^{\sigma\sqrt{t/n}}$$

$$d = e^{-\sigma\sqrt{t/n}}$$

Further read:

- Binominal options pricing model
- "Binominal Models for Option Valuation - Examing and Improving Convergence" by Dietmar Leisen and Matthias Reimer (original document can be found here)

2.2. Bjerksund-Stensland Pricing

Class: com.devexperts.options.pricing.BjerksundStenslandPricing

Directly provided results: price

Suitable for: log normal vanilla American options without dividend schedule.

Complexity: O(1).

Algorithm uses Black-Scholes model with one or two early exercise strategies:

- One boundary method uses precalculated boundary **I**.
- Two boundary method uses precalculated boundary I_1 until time T_1 and precalculated boundary I_2 afterwards.

Further read: "Numerical Methods versus Bjerksund and Stensland Approximations for American Options Pricing" by Marasovic Branka, Aljinovic Zdravka, Poklepovic Tea Section IV. (Original document can be found here).

2.3. Black-Scholes Pricing

Class: com.devexperts.options.pricing.BlackScholesPricing

Directly provided results: price, delta, gamma, theta, vega, rho, phi, carry_rho, speed, vanna, vomma, ultima, zomma, charm, veta, color, totto, strike_delta, strike_gamma.

Suitable for: log normal vanilla and binary European options without dividend schedule.

Complexity: O(1).

Algorithm uses Black-Scholes model, namely it has following assumptions about market:

- There is constant risk-free interest rate and there is fee free opportunity to borrow or lend any amount of cash (even fractional) at this rate.
- Stock price is geometric Brownian motion with constant drift and volatility and there is fee free opportunity to buy and sell (including short sell) the stock at this price.
- Stock pays constant continuous dividends.
- There is no arbitrage opportunity.

Further read: Black-Scholes model

2.4. Black-Scholes Universal Pricing

Class: com.devexperts.options.pricing.BlackScholesUniversalPricing

Directly provided results: price, delta, gamma, theta, vega, rho, phi, vanna, vomma.

Suitable for: log normal vanilla, binary, single/double barrier (European payout for KO rebates) and single/double touch/no-touch European options without dividend schedule.

Complexity: O(1).

Algorithm uses Black-Scholes model with border condition to solve barrier and touch options.

Further read: Barrier options (original document can be found here)

2.5. Explicit Finite Difference Pricing

Class: com.devexperts.options.pricing.ExplicitFiniteDifferencePricing

Directly provided results: price, delta, gamma, speed, theta, charm, color.

Suitable for: log normal vanilla, binary, single/double barrier and single/double touch/no-touch European and American options without dividend schedule.

Complexity: O(num_steps).

Algorithm uses finite difference method to solve Black-Scholes equation.

Further read: "Numerical Approximation of Black-Scholes equation" by Dina Dura and Ana-Maria Moşneagu (original document can be found here)

2.6. Merton-Reiner-Rubinstein Barrier Pricing

Class: com.devexperts.options.pricing.MertonReinerRubinsteinBarrierPricing

Directly provided results: price, delta, gamma, speed, theta, charm, color.

Suitable for: log normal single barrier (American payout for KO rebates) European options without dividend schedule.

Complexity: O(1).

Algorithm uses Black-Scholes model modified to account for single barriers.

Further read: E. G. Haug "The complete guide to options pricing formulas"

2.7. Monte-Carlo Pricing

Class: com.devexperts.options.pricing.MonteCarloPricing

Directly provided results: price, delta, gamma, speed, theta, charm, color.

Suitable for: log normal vanilla European options.

Complexity: O(num_steps).

Algorithm uses Euler method on geometrical Brownian motion.

Further read: "Monte Carlo and Binomial Simulations for European Option Pricing" by Robert Hon Section 3.2.1 (original document can be found here).

3. Other algorithms

3.1. Finite difference derivative

Class: com.devexperts.options.pricing.FiniteDifferenceDerivativeImpl

For all the greeks not provided by pricing algorithms calculation is done using finite difference derivative approximation. There is no need to do anything as any pricing algorithm will use this method if it do not support corresponding greek internally.

Further read: Finite difference

3.2. Yield curve

Class: com.devexperts.options.pricing.YieldCurve

Given prices of bonds with different expirations constructs interest rate curve for currency. Support bonds with and without coupons

Further read: "Methods for Constructing a Yield Curve" by Patrick S. Hagan and Graeme West (original document can be found here).

4. Examples

Note: you can use any compatible pricing algorithm like in the examples below.

4.1. Pricing vanilla option – Black-Scholes

```
VanillaParams p = new VanillaParams();
p.setUnderlying(100);
p.setStrike(110);
p.setExpiration(0.5);
p.setVolatility(0.2);
p.setInterestRate(0.01);
p.setDividendYield(0.03);
p.setStyle(OptionStyle.EUROPEAN);
p.setPayoff(OptionPayoff.CALL);
BlackScholesPricing pr = new BlackScholesPricing();
double price = pr.computePrice(p);
```

4.2. Pricing vanilla option – Bjerksund-Stensland

```
BjerksundStenslandPricing pricing = new BjerksundStenslandPricing();
pricing.setVariant(BjerksundStenslandPricing.Variant.ONE_BOUNDARY);
VanillaParams params = new VanillaParams();
params.setExpiration(3);
params.setStyle(OptionStyle.AMERICAN);
params.setInterestRate(0.08);
params.setStrike(100);
params.setVolatility(0.2);
params.setDividendYield(0.12);
params.setUnderlying(80);
params.setPayoff(OptionPayoff.PUT);
pricing.computePrice(params);
```

4.3. Pricing double barrier option

```
VanillaParams p = new VanillaParams();
p.setStrike(100);
p.setUnderlying(90);
p.setExpiration(100d / 365);
p.setVolatility(0.22);
p.setInterestRate(0.0078);
p.setDividendYield(-0.004);
p.setPayoff(OptionPayoff.CALL);
VanillaBarrierOptionParams bp1 = new VanillaBarrierOptionParams();
bp1.setBase(p);
bp1.setBarrier(84);
bp1.setBarrierType(BarrierType.DOWN_OUT);
VanillaBarrierOptionParams bp2 = new VanillaBarrierOptionParams();
bp2.setBase(bp1);
bp2.setBarrierType(BarrierType.UP_IN);
bp2.setBarrier(104);
BlackScholesUniversalPricing pricing = new BlackScholesUniversalPricing();
UniversalParams up = new UniversalParams(p, bp2);
double price = pricing.computePrice(up);
```

4.4. Building yield curve

```
double[] maturity = {1, 2, 3, 5, 7, 10, 20, 30};
double[] input = percentToFractions(0.60, 0.78, 0.91, 1.18, 1.44, 1.57, 1.90, 2.23);
YieldCurve constantResult = YieldCurve.bootstrap(maturity, input, 2,
   YieldCurve.BootstrappingMode.CONSTANT_RATE);
YieldCurve changingResult = YieldCurve.bootstrap(maturity, input, 2,
   YieldCurve.BootstrappingMode.CHANGING_RATE);
```