C++ Programming for Financial Engineering Level 9 Group A&B Writeup

QuantNet

Minghan Li

10/27/2019

Part One: Questions and Answers

A. Exact Solutions of One-Factor Plain Options

a) and b)

Microsoft Visual Studio Debug Console

```
Batch 1:
Call Price: 2.13337
Call Price from Put/Call Parity:  2.13337
Parity Check result: The Put/Call prices satisfy parity
Put Price: 5.84628
Put Price from Put/Call Parity:  5.84628
Parity Check result: The Put/Call prices satisfy parity

Batch 2:
Call Price: 7.96557
Call Price from Put/Call Parity:  7.96557
Parity Check result: The Put/Call prices satisfy parity
Put Price: 7.96557
Put Price from Put/Call Parity:  7.96557
Parity Check result: The Put/Call prices satisfy parity

Batch 3:
Call Price: 0.204058
Call Price from Put/Call Parity:  0.204058
Parity Check result: The Put/Call prices satisfy parity
Put Price: 4.07326
Put Price from Put/Call Parity:  4.07326
Parity Check result: The Put/Call prices satisfy parity

Batch 4:
Call Price: 92.1757
Call Price from Put/Call Parity:  92.1757
Parity Check result: The Put/Call prices satisfy parity
Put Price: 1.2475
Put Price from Put/Call Parity:  1.2475
Parity Check result: The Put/Call prices satisfy parity
```

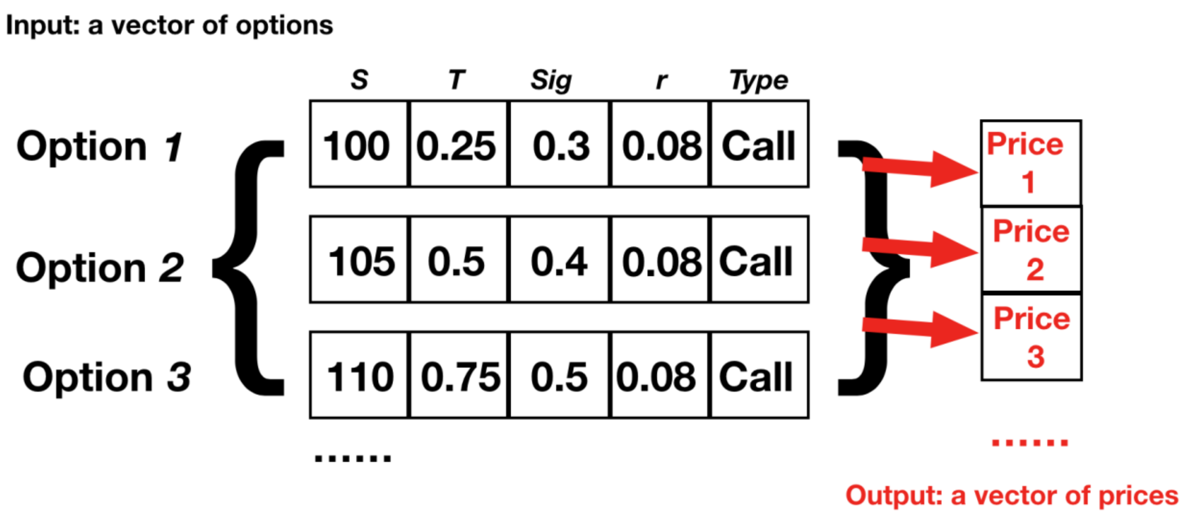
Batch 1 to 4 satisfy the put-call parity relationship under default tolerance of 0.000001. The user can modify the tolerance for parity relationship satisfaction to see different results.

c)

S	Price
10	6.08558
11	6.90422
12	7.73711
13	8.58239
14	9.43856
15	10.3044
16	11.1788
17	12.0608
18	12.9499
19	13.8451
20	14.7461
21	15.6523
22	16.5633
23	17.4786
24	18.3979
25	19.321
26	20.2476
27	21.1773
28	22.11
29	23.0456
30	23.9837
31	24.9242
32	25.8671
33	26.8121
34	27.7591
35	28.708
36	29.6587
37	30.6111
38	31.5652
39	32.5207
40	33.4777
41	34.4361
42	35.3958
43	36.3567
44	37.3188
45	38.282
46	39.2463
47	40.2116
48	41.1779
49	42.1452
50	43.1133

We would like to compute option prices for a monotonically increasing range of underlying values of S. For demonstration purpose, we try to price Batch 4 as a call option with the mesh array of underlying prices from 10 to 50. We first use the global function MeshArray to create a vector of doubles separated by 1, and then pass the vector as an argument to the overloaded Price() function of European Option. Using the global PrintPriceMesh function, we get a nicely formatted column of underlying prices and the computed call option prices.

d)



S	T	sigma	K	r	b	Price
100	20	0.3	100	0.08	0.08	82.7655
100	21	0.3	100	0.08	0.08	84.0772
100	22	0.3	100	0.08	0.08	85.2882
100	23	0.3	100	0.08	0.08	86.4063
100	24	0.3	100	0.08	0.08	87.4389
100	25	0.3	100	0.08	0.08	88.3926
100	26	0.3	100	0.08	0.08	89.2735
100	27	0.3	100	0.08	0.08	90.0873
100	28	0.3	100	0.08	0.08	90.8392
100	29	0.3	100	0.08	0.08	91.5338
100	30	0.3	100	0.08	0.08	92.1757
100	31	0.3	100	0.08	0.08	92.7688
100	32	0.3	100	0.08	0.08	93.3169
100	33	0.3	100	0.08	0.08	93.8234
100	34	0.3	100	0.08	0.08	94.2914
100	35	0.3	100	0.08	0.08	94.724
100	36	0.3	100	0.08	0.08	95.1238
100	37	0.3	100	0.08	0.08	95.4933
100	38	0.3	100	0.08	0.08	95.8348
100	39	0.3	100	0.08	0.08	96.1504
100	40	0.3	100	0.08	0.08	96.4421

We create a mesh for Expiry time T from 20 to 40, and use the EuropeanMatrix function to construct a matrix of Batch 4 call options with different expiry time. The format of the matrix is illustrated by the figure above.

The matrix is then passed to the overloaded Price function for EuropeanOption class for a vector of prices for the options in the matrix.

S	T	sigma	K	r	b	Price
100	30	0.1	100	0.08	0.08	90.9282
100	30	0.2	100	0.08	0.08	91.0749
100	30	0.3	100	0.08	0.08	92.1757
100	30	0.4	100	0.08	0.08	94.0411
100	30	0.5	100	0.08	0.08	95.9422
100	30	0.6	100	0.08	0.08	97.48
100	30	0.7	100	0.08	0.08	98.5583
100	30	0.8	100	0.08	0.08	99.2363
100	30	0.9	100	0.08	0.08	99.6245
100	30	1	100	0.08	0.08	99.8284

Similarly, we create a matrix of Batch 4 call options with different volatility and price them.

S	T	sigma	K	r	b	Price
60	0.25	0.3	65	0.08	0.08	2.13337
100	1	0.2	100	0	0	7.96557
5	1	0.5	10	0.12	0.12	0.204058
100	30	0.3	100	0.08	0.08	92.1757

More generally, we can price a matrix of different options. The above result comes from pricing a matrix of call options consisting of option parameters of Batch 1 to 4.

Option Sensitivities, aka the Greeks

a)

```
Batch 5:
Call Delta: 0.594629
Put Delta: -0.356601
```

b)

S	Delta
90	0.368319
91	0.384223
92	0.400118
93	0.415977
94	0.431772
95	0.447475
96	0.463062
97	0.478508
98	0.493791
99	0.50889
100	0.523785
101	0.538459
102	0.552894
103	0.567076
104	0.580992
105	0.594629
106	0.607976
107	0.621025
108	0.633767
109	0.646196
110	0.658306

In this problem, we are asked to compute call delta price for a monotonically increasing range of underlying values of S.

We use the same option from part (a). We create a mesh for underlying values of S from 90 to 110. We pass the mesh to overloaded Delta function for EuropeanOption class, and print the result.

c)

S	T	sigma	K	r	b	Delta
105	0.5	0.36	95	0.1	0	0.664551
105	0.5	0.36	96	0.1	0	0.650775
105	0.5	0.36	97	0.1	0	0.636871
105	0.5	0.36	98	0.1	0	0.622864
105	0.5	0.36	99	0.1	0	0.608775
105	0.5	0.36	100	0.1	0	0.594629
105	0.5	0.36	101	0.1	0	0.580446
105	0.5	0.36	102	0.1	0	0.56625
105	0.5	0.36	103	0.1	0	0.55206
105	0.5	0.36	104	0.1	0	0.537899
105	0.5	0.36	105	0.1	0	0.523785
105	0.5	0.36	106	0.1	0	0.509739
105	0.5	0.36	107	0.1	0	0.495777
105	0.5	0.36	108	0.1	0	0.481919
105	0.5	0.36	109	0.1	0	0.46818
105	0.5	0.36	110	0.1	0	0.454576
105	0.5	0.36	111	0.1	0	0.441122
105	0.5	0.36	112	0.1	0	0.427831
105	0.5	0.36	113	0.1	0	0.414716
105	0.5	0.36	114	0.1	0	0.40179
105	0.5	0.36	115	0.1	0	0.389063

For illustration, we created a mesh for Strike Price K from 95 to 115. We create a matrix based on Batch5 call option with different K. We pass the matrix to overloaded Delta function. We print the resulting Delta values and matrix.

S	T	sigma	K	r	b	Gamma
105	0.5	0.36	95	0.1	0	0.0123994
105	0.5	0.36	96	0.1	0	0.012657
105	0.5	0.36	97	0.1	0	0.0128957
105	0.5	0.36	98	0.1	0	0.013115
105	0.5	0.36	99	0.1	0	0.0133144
105	0.5	0.36	100	0.1	0	0.0134936
105	0.5	0.36	101	0.1	0	0.0136525
105	0.5	0.36	102	0.1	0	0.0137908
105	0.5	0.36	103	0.1	0	0.0139087
105	0.5	0.36	104	0.1	0	0.0140061
105	0.5	0.36	105	0.1	0	0.0140832
105	0.5	0.36	106	0.1	0	0.0141403
105	0.5	0.36	107	0.1	0	0.0141777
105	0.5	0.36	108	0.1	0	0.0141958
105	0.5	0.36	109	0.1	0	0.014195
105	0.5	0.36	110	0.1	0	0.0141759
105	0.5	0.36	111	0.1	0	0.014139
105	0.5	0.36	112	0.1	0	0.014085
105	0.5	0.36	113	0.1	0	0.0140145
105	0.5	0.36	114	0.1	0	0.0139282
105	0.5	0.36	115	0.1	0	0.0138268

Similarly, we pass the matrix to overloaded Gamma function. We print the resulting Gamma values and matrix.

S	T	sigma	K	r	b	Gamma
60	0.25	0.3	65	0.08	0.08	0.0420428
100	1	0.2	100	0	0	0.0198476
5	1	0.5	10	0.12	0.12	0.106789
100	30	0.3	100	0.08	0.08	0.000179578
105	0.5	0.36	100	0.1	0	0.0134936

More generally, the user can input a matrix of option parameters and receive a vector of either Delta or Gamma as the result. For the illustration above, we input a parameter matrix for Batch 1 to 5 and obtained their respective Gamma.

d)

```
Batch 5:
Call Delta: 0.594629
Call Delta Approximate: 0.594627
Approximate Error: 1.93016e-06
Call Gamma: 0.0134936
Call Gamma Approximate: 0.0134936
Approximate Error: 3.30592e-08

Put Delta: -0.356601
Put Delta Approximate: -0.356603
Approximate Error: 1.93016e-06
Put Gamma: 0.0134936
Put Gamma Approximate: 0.0134936
Approximate Error: 3.30599e-08
```

S	DeltaDD
90	0.368319
91	0.384223
92	0.400118
93	0.415977
94	0.431771
95	0.447474
96	0.463061
97	0.478507
98	0.49379
99	0.508889
100	0.523784
101	0.538457
102	0.552893
103	0.567075
104	0.58099
105	0.594627
106	0.607974
107	0.621023
108	0.633765
109	0.646194
110	0.658304

In this section, we first perform similar task to a) and b). We calculate Delta and Gamma for Batch5, but instead of using exact solution, we use the new DeltaDD and GammaDD functions with parameter h = 0.2 for divided difference method. The approximated solution is then subtracted from the exact solution to obtain the error for the approximation. We also use the overloaded DeltaDD function to perform the task of b) for divided difference approximation. The user can also do the same thing for GammaDD.

h	Call Delta (e)	Call Gamma (e)	Put Delta (e)	Put Gamma (e)
0.1	4.82543e-07	8.26086e-09	4.82543e-07	8.26441e-09
0.2	1.93016e-06	3.30592e-08	1.93016e-06	3.30599e-08
0.3	4.3428e-06	7.43851e-08	4.3428e-06	7.43852e-08
0.4	7.72041e-06	1.32242e-07	7.72041e-06	1.32242e-07
0.5	1.20629e-05	2.06631e-07	1.20629e-05	2.06631e-07
0.6	1.73701e-05	2.97553e-07	1.73701e-05	2.97553e-07
0.7	2.36419e-05	4.0501e-07	2.36419e-05	4.0501e-07
0.8	3.08781e-05	5.29004e-07	3.08781e-05	5.29004e-07
0.9	3.90785e-05	6.69538e-07	3.90785e-05	6.69538e-07
1	4.82428e-05	8.26612e-07	4.82428e-05	8.26612e-07

Next, we perform an error analysis for approximation of Delta and Gamma for Batch5. As shown in the result, the error for all approximations increases as h increases.

B. Perpetual American Options

a) and b)

```
Batch 6:
Call Price: 18.5035
Put Price: 3.03106
```

Calculate the call and put prices for Batch6 by calling the Price function for AmericanPerpetual Class.

c)

S	Price
100	13.6174
101	14.0603
102	14.5131
103	14.9758
104	15.4486
105	15.9316
106	16.4249
107	16.9286
108	17.4429
109	17.9678
110	18.5035
111	19.0501
112	19.6078
113	20.1765
114	20.7566
115	21.3481
116	21.951
117	22.5656
118	23.192
119	23.8302
120	24.4804

Create a S mesh from 100 to 120. Price Batch6 call option using overloaded Price function for a vector of S inputs.

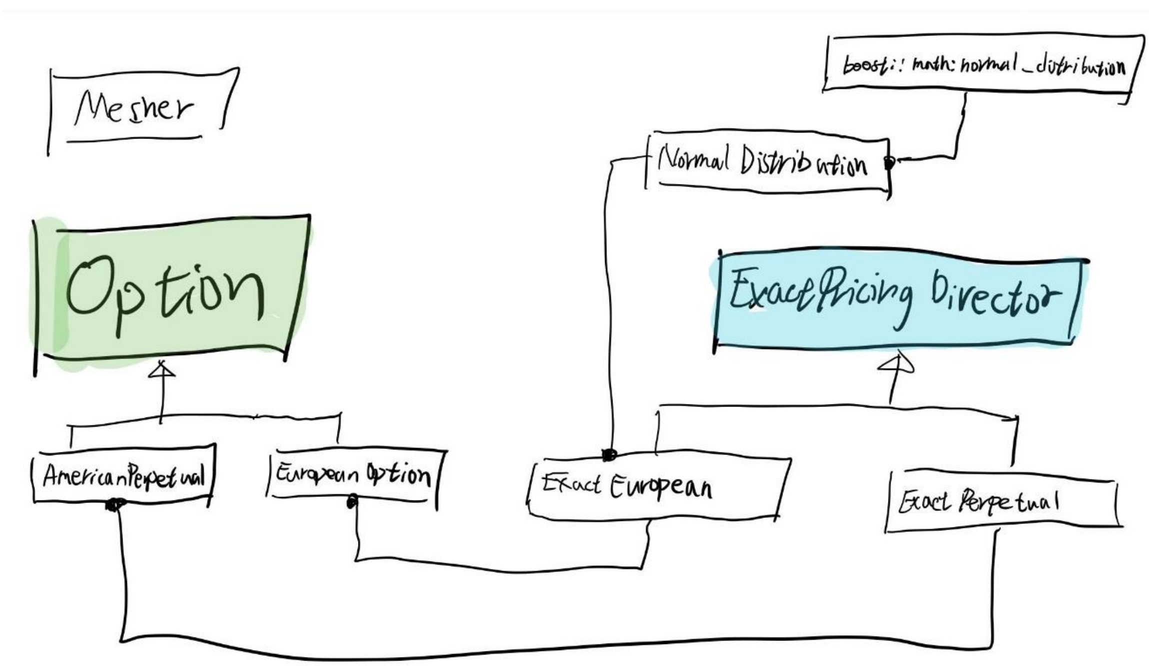
d)

S	sigma	K	r	b	Price
110	0.1	100	0.1	0.02	18.5035
110	0.1	101	0.1	0.02	18.0998
110	0.1	102	0.1	0.02	17.7087
110	0.1	103	0.1	0.02	17.3298
110	0.1	104	0.1	0.02	16.9625
110	0.1	105	0.1	0.02	16.6065
110	0.1	106	0.1	0.02	16.2611
110	0.1	107	0.1	0.02	15.9261
110	0.1	108	0.1	0.02	15.601
110	0.1	109	0.1	0.02	15.2855
110	0.1	110	0.1	0.02	14.9791
110	0.1	111	0.1	0.02	14.6816
110	0.1	112	0.1	0.02	14.3926
110	0.1	113	0.1	0.02	14.1117
110	0.1	114	0.1	0.02	13.8387
110	0.1	115	0.1	0.02	13.5734
110	0.1	116	0.1	0.02	13.3153
110	0.1	117	0.1	0.02	13.0643
110	0.1	118	0.1	0.02	12.8201
110	0.1	119	0.1	0.02	12.5825
110	0.1	120	0.1	0.02	12.3512

We test the matrix pricer for Perpetual American option with a matrix of Batch6 call option with different strike prices. The user can pass a more generic matrix of parameters for distinct Perpetual American options to the price function of AmericanPerpertual class and receive a vector of option prices using the exact solution method.

Part Two: Justification for Design

Overview:



This option pricing program is designed to follow an object-oriented approach and the single responsibility principle, while outsourcing the container and algorithms to Standard Template Library and Boost Library. The option data are stored in collection of option classes derived from a single parent class called Option. Member functions are implemented for each type of option classes to calculate various pricing components for options, but the actual calculations are delegated to another collection of classes derived from a parent class called ExactPricingDirector.

The original goal was to implement polymorphic pricing functions for a single option class with a data member to indicate its option type. However, during the implementation, this approach results in the failure of calculation delegation to the pricing classes, violating the single responsibility principle. As we prefer not to hard code the calculation into the option classes, this approach is abandoned.

The most obvious flaw in the final approach is that many member functions are declared as static, which is far from an ideal implementation. The reason for doing this is to enable flexibility in CalculateArray and CalculateMatrix functions. The last argument in these two functions take in a function pointer, which the two functions then use to calculate components for an array of input prices or a matrix of input parameters. Declaring the member functions as static is only implementation possible in my knowledge to accomplish this task. In the end, the decision was made because the benefit of flexibility in these two functions outweighs the drawbacks of declaring member functions as static. To minimize the dis advantage, the static functions, CalculateArray, and CalculateMatrix are all declared as private functions. Since our task is to price options, and we don’t make any modifications to these member functions and member data during the process, declaring them as static functions won’t cause too much harm.

Classes:

<u>NormalDistribution.hpp:</u>

NormalDistribution class is wrapper class for boost::math::normal_distribution. The purpose of having NormalDistribution class is primarily to perform the CDF and PDF calculations for Normal (0, 1) distribution in option pricing formulas accurately. For flexibility in the future, NormalDistribution is implemented as a template class, and NormalDistribution objects can be declared for different mean and standard deviations other than 0 and 1.

<u>Mesher.hpp:</u>

This file is modified version of Mesher.hpp provided by Professor Duffy. In this version, we added a MeshArray global function, which generate a vector of doubles separated by equal distance. We also added the EuropeanMatrix and PerpetualMatrix global functions, which generate a matrix of option parameters for the same option, by altering one of the parameters for input option in each row of parameters.

Option.hpp:

Option class is a base class for AmericanPerpetual and EuropeanOption classes. Option class contains 7 essential elements to determine an option's characteristics.

Parameters initialized:

• T (expiry time/maturity). This is a number, e.g. T = 1 means one year. K (strike price).

• sig (volatility).

• r (risk-free interest rate).

• S (current stock price where we wish to price the option).

• C = call option price, P = put option price.

• b = cost of carry

For the default constructor, the order of parameters inputted is according to an instruction thread on QuantNet. The parameters function returns a vector of option parameters according to this order.

EuropeanOption.hpp

This file is modified from EuropeanOption.hpp from DataSim.

In this version, we added the following:

1. Public Price(), Delta(), and Gamma() functions, each of which is implemented to take no

argument, a vector of doubles (underlying prices), and a matrix of option parameters.

2. Private functions CallPrice(double U), PutPrice(double U), CallDelta(double U), PutDelta(double U), CallPutGamma(), and CallPutGamma(double U) are added to perform the different functions as well as to perform specific calculations in Price(), Delta(), and Gamma() functions.

3. A pointer to ExactEuropean Pricing class, to which the actual calculations are delegated.

4. DeltaDD and GammaDD functions to approximate Gamma and Delta of the option using the divided difference method.

5. PriceParity and its helper functions to check Put/Call price parity

Modifications: The EuropeanOption class is now a derived class of Option class

AmericanPerpetual.hpp:

This file contains the function declaration for class AmericanPerpetual, a derived class from Option. The class structure is similar to the EuropeanOption class, except the T variable in the parent Option class is always initialized to -100. The reason is that the T parameter technically does not exist for Perpetual American Options, and this initialization helps us to detect complicated problems in the future. Accordingly, the T parameter is made completely isolated from the public use. No getter or setter functions are implemented, and the parameter function does not return T. The Gamma and Delta functions were also not implemented for this class. The pricing component is delegated to a dynamically allocated ExactPerpetual pricing class.

ExacPricingDirector.hpp:

ExactPricingDirector class is a parent class for all classes that perform exact pricing calculations for options. This class reflects an orginal attempt to imbed a pointer to a pricing director in Option class and store the corresponding different derived classes of pricing directors in the derived classes of Option class. This attempt failed but is still worth exploring.

ExactEuropean.hpp:

ExactEuropean is a derived class from ExactPricingDirector. ExactEuropean class performs the calculations of Price, Gamma, and Delta for European Options using exact pricing formulas. Many of the member functions and member data are declared as static, which is not an optimal implementation; however, this is the only implementation possible so far to pass them into CalculateArray and CalculateMatrix functions as function pointers. The benefit of this level of flexibility is far more significant for the design. The static functions are mostly made private to avoid interference from the user.

ExactPerpetual.hpp:

ExactPerpetual is another derived class from ExactPricingDirector. The general structure is similar to ExactEuropean class. ExactPerpetual class performs the calculations of price of Perpetual American Options using exact pricing formulas. CalculateArray function takes only 5 input arguments, different from CalculateArray function in ExactEuropean class, as Perpetual American Options don't have expiry time T.