Project A:

Files: **EuropeanOption.hpp** (//EuropeanOption class header file.)

**EuropeanOption.cpp** (//EuropeanOption class implementation file.)

**Global\_print.h** (/\* Printing vectors and matrices \*/)

**Greeks.h** (/\* This file contains four functions to calculate Delta for Call and

Put prices, Gamma for Call and Put prices.

CallDelta() , PutDelta() , CallGamma() , PutGamma() \*/ )

**Mehser.h** (/\* mesher() creates a mesh array of double values between max\_val

and min\_val, seperated by mesh size h. \*/ )

**Normal\_Boost.h** (/\* This file contains functions to calculate probability

density values and cumulative density values of

Normal distribution (for given x) using boost libraries.

\*/)

**Pricing.h (**/\* This file contains four functions.

EuropeanCall() // Calculate Call Option price

EuropeanPut() // Calculate Put Option price

EuropeanCall(double S, EuropeanOption::OptionData\* OD) // Calculate the Call Option price using the struct.

EuropeanPut(double S, EuropeanOption::OptionData\* OD) // Calculate the Put Option price using the struct. \*/ )

**TestEuropeanOptionA1\_a.cpp // Producing output for A1\_a question.**

**TestEuropeanOptionA1\_b.cpp // Producing output for A1\_b question.**

**TestEuropeanOptionA1\_c.cpp // Producing output for A1\_c question.**

**TestEuropeanOptionA1\_d.cpp // Producing output for A1\_d question.**

**TestEuropeanOptionA1\_e.cpp // Producing output for A1\_e question.**

**TestEuropeanOptionA2\_a.cpp // Producing output for A2\_a question.**

**TestEuropeanOptionA2\_b.cpp // Producing output for A2\_b question.**

**TestEuropeanOptionA2\_c.cpp // Producing output for A2\_c question.**

**(All these TestEuropeanOptionA\_. Cpp files have main()s. When you check each exercise pelase exclude other TestEuropeanOptionA\_. Cpp files from the project built.)**

A1\_a) comments

/\* In this problem I implemented four global functions and included those inside

pricing.h , and Normal\_Boost.h header files. Those functions are

(1) double EuropeanCall(double S, double K, double T, double r, double sig, double b)

(2) double EuropeanPut(double S, double K, double T, double r, double sig, double b)

(3) double N\_BOOST(double x)

(4) double n\_BOOST(double x)

\*/

Batch 1 Output: C1 = 2.13337 and P1 = 5.84628

Batch 2 Output: C2 = 7.96557 and P2 = 7.96557

Batch 3 Output: C3 = 0.204058 and P3 = 4.07326

Batch 4 Output: C4 = 92.1757 and P4 = 1.2475

A1\_b) comments

/\*Here I implemented the following function.

double EuropeanOption::put\_call\_parity(double assetP, double val, string str) const

Usage ex : Passing asset value, call option value with string "C". It will return put option value.

Usage ex : passing asset value, put option value with stirng "P". It will return call option value.

\*/

For Batch 1

Put value using put\_call\_parity function : 5.84628

Call value using put\_call\_parity function : 2.13337

For Batch 2

Put value using put\_call\_parity function : 7.96557

Call value using put\_call\_parity function : 7.96557

For Batch 3

Put value using put\_call\_parity function : 4.07326

Call value using put\_call\_parity function : 0.204058

For Batch 4

Put value using put\_call\_parity function : 1.2475

Call value using put\_call\_parity function : 92.1757

A1\_c) comments /\*

//Here I newly implemented the following functions .

/\*

1. First I created Optiondata struct as a public member of EuropeanOption class.

struct OptionData {

double K\_s;

double T\_s;

double r\_s;

double sig\_s;

};

2. Then I created a new orverloaded constructor for the class EuropeanOption(passing four variables) to create a new instance of

struct Optiondata .

EuropeanOption::EuropeanOption(double K\_c, double T\_c, double r\_c, double sig\_c)

{

Optionstruct = new OptionData;

Optionstruct->K\_s = K\_c;

Optionstruct->T\_s = T\_c;

Optionstruct->r\_s = r\_c;

Optionstruct->sig\_s = sig\_c;

}

3. Then I created two overloaded global functions for EuropeanCall and EuropeanPut functions inside pricing.h

(Here I pass the asset value and the struct.)

double EuropeanCall(double S, EuropeanOption::OptionData\* OD)

double EuropeanPut(double S, EuropeanOption::OptionData\* OD)

4. Finally I created two member functions for EuropeanOption class to call above mentioned(mentioned in 3) functions.

double EuropeanOption::CallPrice\_struct(double S) const;

double EuropeanOption::PutPrice\_struct(double S ) const;

\*/

Batch1 out put using structs

Call Price : 2.13337

Put Price : 5.84628

Batch2 out put using structs

Call Price : 7.96557

Put Price : 7.96557

Batch3 out put using structs

Call Price : 0.204058

Put Price : 4.07326

Batch3 out put using structs

Call Price : 92.1757

Put Price : 1.2475

A1\_d) comments

// Here I implemented following function as a global functions.

//std::vector<double> mesher(double min\_val, double max\_val, double h) (Inside Mesher.h file)

//Function to print vector. (Inside Global\_print.h)

//void print(const vector<double>& vec)

//Function to print matrix. (Inside Global\_print.h)

//void print(const vector<vector<double>>& matrix)

Output for Batch 1

7.792e-036, 53.7129,

1.88964e-032, 52.7129,

1.64204e-029, 51.7129,

6.19303e-027, 50.7129,

1.17207e-024, 49.7129,

1.24529e-022, 48.7129,

8.11387e-021, 47.7129,

3.47944e-019, 46.7129,

1.03991e-017, 45.7129,

2.27042e-016, 44.7129,

3.76503e-015, 43.7129,

4.89964e-014, 42.7129,

5.1436e-013, 41.7129,

4.45934e-012, 40.7129,

3.25772e-011, 39.7129,

2.0405e-010, 38.7129,

1.11246e-009, 37.7129,

5.34891e-009, 36.7129,

2.29454e-008, 35.7129,

8.8713e-008, 34.7129,

3.11926e-007, 33.7129,

1.00545e-006, 32.7129,

2.99241e-006, 31.7129,

8.27565e-006, 30.7129,

2.13896e-005, 29.7129,

5.19359e-005, 28.713,

0.000119024, 27.713,

0.000258548, 26.7132,

0.000534403, 25.7134,

0.00105474, 24.714,

0.00199417, 23.7149,

0.00362244, 22.7165,

0.00633924, 21.7193,

0.010714, 20.7236,

0.0175282, 19.7304,

0.0278174, 18.7407,

0.0429083, 17.7558,

0.0644467, 16.7774,

0.0944125, 15.8073,

0.135117, 14.848,

0.189181, 13.9021,

Output for Batch 2

3.05867e-031, 90,

7.50875e-029, 89,

9.39062e-027, 88,

6.76817e-025, 87,

3.08735e-023, 86,

9.58525e-022, 85,

2.14514e-020, 84,

3.62362e-019, 83,

4.79653e-018, 82,

5.13087e-017, 81,

4.55058e-016, 80,

3.41899e-015, 79,

2.21616e-014, 78,

1.25872e-013, 77,

6.34869e-013, 76,

2.87668e-012, 75,

1.18282e-011, 74,

4.45231e-011, 73,

1.54615e-010, 72,

4.98755e-010, 71,

1.50356e-009, 70,

4.25893e-009, 69,

1.13899e-008, 68,

2.8884e-008, 67,

6.97279e-008, 66,

1.60803e-007, 65,

3.5539e-007, 64,

7.54907e-007, 63,

1.54526e-006, 62,

3.05545e-006, 61,

5.84877e-006, 60,

1.08604e-005, 59,

1.95985e-005, 58,

3.44297e-005, 57,

5.89738e-005, 56.000

9.8635e-005, 55.0001

0.000161299, 54.0002

0.000258226, 53.0003

0.000405172, 52.0004

0.000623757, 51.0006

0.000943109, 50.0009

Output for Batch 3

2.48011, 1.34932,

3.20115, 1.07036,

3.97989, 0.849091,

4.80493, 0.674134,

5.6668, 0.536003,

6.55778, 0.426982,

7.47169, 0.340889,

8.40362, 0.272821,

9.34971, 0.218913,

10.3069, 0.176132,

11.2729, 0.142104,

12.2458, 0.114972,

13.2241, 0.0932814,

14.2067, 0.0758952,

15.1927, 0.0619213,

16.1815, 0.050659,

17.1724, 0.0415573,

18.165, 0.0341815,

19.159, 0.0281882,

20.1541, 0.0233052,

21.1501, 0.0193162,

22.1468, 0.0160492,

23.1442, 0.0133667,

24.142, 0.0111585,

25.1401, 0.00933635,

26.1386, 0.00782914,

27.1374, 0.0065795,

28.1363, 0.00554104,

29.1355, 0.00467613,

30.1347, 0.00395419,

31.1341, 0.00335029,

32.1336, 0.00284408,

33.1332, 0.00241887,

34.1329, 0.00206101,

35.1326, 0.00175922,

36.1323, 0.00150425,

37.1321, 0.00128842,

38.1319, 0.00110539,

39.1317, 0.000949906

40.1316, 0.000817588

41.1315, 0.000704793

Output for Batch 4

6.08558, 5.15738,

6.90422, 4.97602,

7.73711, 4.8089,

8.58239, 4.65419,

9.43856, 4.51036,

10.3044, 4.37617,

11.1788, 4.25056,

12.0608, 4.13264,

12.9499, 4.02165,

13.8451, 3.91694,

14.7461, 3.81792,

15.6523, 3.72411,

16.5633, 3.63506,

17.4786, 3.55038,

18.3979, 3.46974,

19.321, 3.39282,

20.2476, 3.31936,

21.1773, 3.24911,

22.11, 3.18184,

23.0456, 3.11735,

23.9837, 3.05547,

24.9242, 2.99603,

25.8671, 2.93888,

26.8121, 2.88387,

27.7591, 2.83088,

28.708, 2.7798,

29.6587, 2.73052,

30.6111, 2.68294,

31.5652, 2.63697,

32.5207, 2.59252,

33.4777, 2.54952,

34.4361, 2.50789,

35.3958, 2.46756,

36.3567, 2.42848,

37.3188, 2.39057,

38.282, 2.3538,

39.2463, 2.3181,

40.2116, 2.28342,

41.1779, 2.24973,

42.1452, 2.21698,

43.1133, 2.18512,

A1\_e) comments

//Here I am claculating Call option prices and Put option prices as function of expiry time T and function of volatility. Output Matrix : First colum-Call price, Second colum-Put price.

Range for maturity 0.25 to 1. Step size 0.25

Output for Batch 1 (Function of Maturity)

2.13337, 5.84628,

4.03758, 6.4889,

5.6754, 6.89009,

7.15299, 7.15556,

Output for Batch 2 (Function of Maturity)

3.98776, 3.98776,

5.6372, 5.6372,

6.90126, 6.90126,

7.96557, 7.96557,

Output for Batch 3 (Function of Maturity)

0.00213952, 4.70659,

0.0350238, 4.45267,

0.10723, 4.24654,

0.204058, 4.07326,

Output for Batch 4 (Function of Maturity)

6.96184, 4.98171,

10.3881, 6.46708,

13.2114, 7.38784,

15.7113, 8.02295,

Volatility varies from 0.1 to 0.5 with stepsize 0.1

Output for Batch 1 (Function of volatility)

0.173097, 3.88601,

1.04145, 4.75437,

2.13337, 5.84628,

3.29033, 7.00325,

4.47244, 8.18535,

Output for Batch 2 (Function of volatility)

3.98776, 3.98776,

7.96557, 7.96557,

11.9235, 11.9235,

15.8519, 15.8519,

19.7413, 19.7413,

Output for Batch 3 (Function of volatility)

5.46951e-010, 3.8692,

0.000807918, 3.87001,

0.0213027, 3.89051,

0.0895302, 3.95873,

0.204058, 4.07326,

Output for Batch 4 (Function of volatility)

90.9282, 1.96045e-005,

91.0749, 0.146674,

92.1757, 1.2475,

94.0411, 3.11291,

95.9422, 5.01397,

A2\_a) comments

// In this problem I included greeks.h header here and implemented following four global fucntions there.

//double CallDelta(double S, double K, double T, double r, double sig, double b)

//double PutDelta(double S, double K, double T, double r, double sig, double b)

//double CallGamma(double S, double K, double T, double r, double sig, double b)

// double PutGamma(double S, double K, double T, double r, double sig, double b)

// Then I created 4 public EuropeanOption member fucntions to access above 4.

/\* To call global sensitivities in the greeks.h file

double Call\_CallDelta (double U) const ;

double Call\_PutDelta (double U) const ;

double Call\_CallGamma(double U) const ;

double Call\_PutGamma(double U) const ; \*/

For values K =100, S=105, T=0.5, r=0.1, b=0 and sig=0.36

Gamma Call : 0.0134936

Gamma Put : 0.0134936

Delta Call : 0.594629

Delta Put : -0.356601

A2\_b)

// In this problem I calculated call delta and put delta according to S = [10,50];

// Out put matrix : first column will display call delta and second column will display put delta price.

Delta call Delta put

2.25551e-019, -0.951229,

6.18174e-018, -0.951229,

1.12536e-016, -0.951229,

1.46673e-015, -0.951229,

1.44882e-014, -0.951229,

1.1336e-013, -0.951229,

7.27491e-013, -0.951229,

3.93787e-012, -0.951229,

1.83921e-011, -0.951229,

7.55191e-011, -0.951229,

2.76878e-010, -0.951229,

9.18317e-010, -0.951229,

2.78591e-009, -0.951229,

7.80377e-009, -0.951229,

2.0348e-008, -0.951229,

4.97346e-008, -0.951229,

1.14647e-007, -0.951229,

2.50577e-007, -0.951229,

5.21714e-007, -0.951229,

1.03903e-006, -0.951228,

1.98667e-006, -0.951227,

3.65877e-006, -0.951226,

6.5092e-006, -0.951223,

1.1216e-005, -0.951218,

1.87624e-005, -0.951211,

3.05351e-005, -0.951199,

4.84406e-005, -0.951181,

7.50367e-005, -0.951154,

0.00011368, -0.951116,

0.000168682, -0.951061,

0.000245471, -0.950984,

0.000350761, -0.950879,

0.000492699, -0.950737,

0.000681019, -0.950548,

0.000927157, -0.950302,

0.00124435, -0.949985,

0.00164771, -0.949582,

0.00215423, -0.949075,

0.00278276, -0.948447,

0.00355398, -0.947675,

0.00449025, -0.946739,

A2\_c)

// Calculating call delta and call gamma values using numerical differentiation methods.

For values K =100, S=105, T=0.5, r=0.1, b=0 and sig=0.36

Delta Call using sensitivity equation : 0.594629

Calldelta\_numerical values for h = 0.1 to 1.0

0.594628

0.594627

0.594624

0.594621

0.594617

0.594611

0.594605

0.594598

0.59459

0.59458

Output shows that the smaller values of h gives better approximations

Gamma Call using sensitivity equation : 0.0134936

Callgamma\_numerical values for h = 0.1 to 1.0

0.0134936

0.0134936

0.0134936

0.0134935

0.0134934

0.0134933

0.0134932

0.0134931

0.013493

0.0134928

Output shows that smaller values of h gives better approximations