A. Exact Solutions of One-Factor Plain Options

a) We give the set of test values for option pricing. We give each set a name so that we can refer to it in later exercises (we call them batches).
Batch 1: T = 0.25, K = 65, sig = 0.30, r = 0.08, S = 60 (then C = 2.13337, P = 5.84628).
Batch 2: T = 1.0, K = 100, sig = 0.2, r = 0.0, S = 100 (then C = 7.96557, P = 7.96557).
Batch 3: T = 1.0, K = 10, sig = 0.50, r = 0.12, S = 5 (C = 0.204058, P = 4.07326).
Batch 4: T = 30.0, K = 100.0, sig = 0.30, r = 0.08, S = 100.0 (C = 92.17570, P = 1.24750).

```
@@@@@@@@@@@@@@@ Part a @@@@@@@@@@@@@@@@@@@

Batch 1 call price: C = 2.13337
Batch 1 put price: P = 5.84628
Batch 2 call price: C = 7.96557
Batch 2 put price: P = 7.96557
Batch 3 call price: C = 0.204058
Batch 3 put price: P = 4.07326
Batch 4 call price: C = 92.1757
Batch 4 put price: P = 1.2475
```

b) Implementation of the put-call parity as a mechanism to calculate the call price for a corresponding put price (vice versa).

```
Given the price of the Call for batch 1 is 2.13337, the Put is 5.84628
Given the price of the Put for batch 2 is 7.96557, the call is 7.96557
```

Implementation of the put-call parity as a mechanism to check if a given set of put/call prices satisfy parity.

```
For batch 3 C + Ke^(-rt) = 9.07326 and P + S = 9.07326
For batch 4 C + Ke^(-rt) = 101.247 and P + S = 101.247
```

c) Batch 1 prices for a monotonically increasing range of underlying values of S, for example 10, 11, 12, …, 50.

STOCK PRICE	CALL PRICE	PUT PRICE
10.00000	0.00000	53.71291
11.00000	0.00000	52.71291
48.00000	0.09441	15.80733
49.00000	0.13512	14.84803
50.00000	0.18918	13.90209

d) We wish to compute option prices as a function of i) expiry time, ii) volatility, or iii) any of the option pricing parameters.

```
std::vector<std::vector<double>> EuropeanOption::Price(const std::vector<std::vector<double>>& param,
        double S, std::string parameter_Type) {
        std::vector<std::vector<double>> ans;
        for (char& c : parameter_Type) {
        c = std::toupper(c);
        }
        if (parameter_Type == "VOLATILITY") {
        return Vol_Price(param, S);
        }
        if (parameter_Type == "EXPIRY TIME") {
        return T_Price(param, S);
        }
        if (parameter_Type == "STRIKE PRICE") {
        return K_Price(param, S);
        }
        if (parameter_Type == "INTEREST RATE") {
        return R_Price(param, S);
        }
        throw std::invalid_argument("Wrong parameter type");
}
```

This design allows me to overload the Price function to work with any matrix parameter while holding the other parameters and the underlying stock price constant. It simplifies the modification of the function to handle option pricing parameters by adding new if conditions and associated functions. The use of case-insensitive parameters and exception-based error handling avoids unexpected behavior if the user inputs an invalid parameter.

```
Batch 1
Option prices as a function of price
Call
   0.00199    0.18918    2.13337
   7.90027   16.58794   26.32824
  36.29157   46.28750   56.28712
Puts
  23.71491   13.90209    5.84628
   1.61319    0.30086    0.04115
   0.00448    0.00041    0.00003
Option prices as a function of volatility
   4.75437    5.28745    5.84628
   7.00325    7.59235    8.18535
Option prices as a function of expiry time
   6.48890    5.84628
   6.89009    5.24309
Option prices as a function of strike price
   0.05693    1.18957    5.84628
   9.58889   18.56826   28.23542
Option prices as a function of interest rate
   6.29097    6.06620    5.84628
   5.42107    5.01544    4.62943
```

Option Sensitivities

a) Implement the gamma formulae for call and put future option pricing using the data set: K = 100, S = 105, T = 0.5, r = 0.1, b = 0 and sig = 0.36. (exact delta call = 0.5946, delta put = -0.3566).

```
Exact delta call = 0.59463
Exact delta put = -0.35660
```

b) We use Batch 5 (K = 100, S = 105, T = 0.5, r = 0.1, b = 0 and sig = 0.36) to compute call delta price for a monotonically increasing range of underlying values of S, for example 10, 11, 12, …, 50

STOCK PRICE	CALL PRICE	PUT PRICE
10.00000	0.00000	-0.95123
11.00000	0.00000	-0.95123
49.00000	0.00355	-0.94768
50.00000	0.00449	-0.94674

c) Now you can input a matrix of option parameters and receive a matrix of either Delta or Gamma as the result.

```
Delta Matrix
   0.00025    0.00449    0.02863
   0.09641    0.21577    0.36832
   0.59463    0.65831    0.76149
Gamma Matrix
   0.03713    0.02978    0.02483
   0.02129    0.01863    0.01656
   0.01420    0.01355    0.01242
Delta as a function of volatility
  -0.32229   -0.34034   -0.35044
  -0.35660   -0.35895   -0.35811
Gamma as a function of volatility
   0.02556    0.02044    0.01704
   0.01420    0.01136    0.01022
Delta as a function of expiry time
  -0.35660   -0.35016
  -0.35011   -0.34038
Gamma as a function of expiry time
   0.01420    0.02059
   0.01131    0.00955
Delta as a function of strike price
  -0.35660   -0.00363   -0.02107
  -0.04062   -0.11028   -0.22052
Gamma as a function of strike price
   0.01420    0.01419    0.01420
   0.01420    0.01420    0.01420
```

```
Delta as a function of interest rate
  -0.36746   -0.36380   -0.36018
  -0.35660   -0.34606   -0.33921
Gamma as a function of interest rate
   0.01463    0.01448    0.01434
   0.01420    0.01378    0.01351
```

d) We now use divided differences to approximate option sensitivities and compare the accuracy with various values of the parameter h.

```
h = 0.10000
Exact delta call = 0.5946286597 numerical approximation = 0.6968499038
Exact delta Put = -0.3566007648 numerical approximation = -0.3031500962

h = 0.0100000000
Exact delta call = 0.5946286597 numerical approximation = 0.6968505250
Exact delta Put = -0.3566007648 numerical approximation = -0.3031494750

h = 0.0010000000
Exact delta call = 0.5946286597 numerical approximation = 0.6968505312
Exact delta Put = -0.3566007648 numerical approximation = -0.3031494688

h = 0.0001000000
Exact delta call = 0.5946286597 numerical approximation = 0.6968505313
Exact delta Put = -0.3566007648 numerical approximation = -0.3031494687

h = 0.0000100000
Exact delta call = 0.5946286597 numerical approximation = 0.6968505311
Exact delta Put = -0.3566007648 numerical approximation = -0.3031494686

h = 0.0000010000
Exact delta call = 0.5946286597 numerical approximation = 0.6968505275
Exact delta Put = -0.3566007648 numerical approximation = -0.3031494664
```

```
h = 0.1000000000
Exact Gamma call = 0.0141976523 numerical approximation = 0.0130694984
Exact Gamma Put = 0.0141976523 numerical approximation = 0.0130694984

h = 0.0100000000
Exact Gamma call = 0.0141976523 numerical approximation = 0.0130695017
Exact Gamma Put = 0.0141976523 numerical approximation = 0.0130695017

h = 0.0010000000
Exact Gamma call = 0.0141976523 numerical approximation = 0.0130695170
Exact Gamma Put = 0.0141976523 numerical approximation = 0.0130695064
```

```
h = 0.0001000000
Exact Gamma call = 0.0141976523 numerical approximation = 0.0130711442
Exact Gamma Put = 0.0141976523 numerical approximation = 0.0130700784

h = 0.0000100000
Exact Gamma call = 0.0141976523 numerical approximation = 0.0132160949
Exact Gamma Put = 0.0141976523 numerical approximation = 0.0131450406

h = 0.0000010000
Exact Gamma call = 0.0141976523 numerical approximation = 0.0284217094
Exact Gamma Put = 0.0141976523 numerical approximation = 0.0213162821
```

B. Perpetual American Options

b) We compute the price of a perpetual American option with K = 100, sig = 0.1, r = 0.1, b = 0.02, S = 110.

```
American perpetual call 18.50350
American perpetual put 3.03106
```

c) We compute the call and put option price for a monotonically increasing range of underlying values of S.

```
STOCK PRICE      CALL PRICE      PUT PRICE
  10.00000         0.00826     9034894.53586
  11.00000         0.01123     4995571.21262
```

```
  49.00000         1.37233      462.36033
  50.00000         1.46448      407.78680
```

d) We compute perpetual American option prices as a function of price, interest rate and strike price.

```
Perpetual American option price as a function of price
   0.71437    1.46448    2.63274
   4.32291    6.64256    9.70270
  15.93161   18.50350   24.48045

Perpetual American option price as a function of volatility
Call
  18.50350   30.65634   34.61878
  38.45117   45.64845   48.99293
```

```
Put
   3.03106   15.02926   19.07128
  23.00547   30.44919   33.93077

Perpetual American option price as a function of strike price
Call
  18.50350   69.64138
  40.80093   30.34607
Put
   3.03106    0.04053
   0.23103    0.60561

Perpetual American option price as a function of interest rate
Call
  37.12241   26.09628   21.26156
  18.50350   14.49976   13.19330
Put
   4.52514    3.85799    3.38721
   3.03106    2.32630    2.02021
```

Groups C&D: Monte Carlo Pricing Methods

C. Monte Carlo 101

a) The code implements the Euler-Maruyama method which is a numerical approximation used to solve stochastic differential equations. It uses the SDE definition namespace to implement the drift and diffusion of the SDE.

$$\begin{cases} X_{n+1} = X_n + aX_n\Delta t_n + bX_n\Delta W_n \\ X_0 \quad = A. \end{cases}$$

```
// The FDM (in this case explicit Euler)
VNew = VOld  + (k * drift(x[index-1], VOld))
             + (sqrk * diffusion(x[index-1], VOld) * dW);
```

VNew represents X_n+1 and VOld X_n.
K is the time increment.

dW is the Wiener increments of each iteration.
The outer loop generates a path for each iteration, with each path representing a possible future trajectory of the stock price overtime.
The inner loop updates the price at each time increment and gives the final price of the stock at the expiration time T. The final price is then used to compute the payoff of the option of each path.
After all the paths are simulated, the average of the payoffs is discounted to compute the present value of the options.

b) As the number of time steps (NT) increase from 300 to 500, the results tend to get closer to the true exact value and the values also seem to converge.
Increasing the number of simulations (NSIM) appears to improve accuracy. For a fixed NT = 300, increasing NSIM from 100,000 to 200,000 gives results closer to the exact price.

```
Batch 1 put price: P = 5.84628
Batch 2 call price: C = 7.96557
Batch 2 put price: P = 7.96557
```

Batch1								
NSIM (simulations)	**NT = 300**	**NT = 400**	**NT = 450**	**NT = 500**	**Deviation (NT = 300)**	**Deviation (NT = 400)**	**Deviation (NT = 450)**	**Deviation (NT = 500)**
100000	5.85221	5.83558	5.84311	5.83818	0.00593	-0.0107	-0.00317	-0.0081
150000	5.85465	5.8553	5.85471	5.84978	0.00837	0.00902	0.00843	0.0035
200000	5.86071	5.85361	5.85744	5.85533	0.01443	0.00733	0.01116	0.00905
300000	5.86033	5.86176	5.86153	5.85286	0.01405	0.01548	0.01525	0.00658
Batch2								
NSIM (simulations)	**NT = 300**	**NT = 400**	**NT = 450**	**NT = 500**	**Deviation (NT = 300)**	**Deviation (NT = 400)**	**Deviation (NT = 450)**	**Deviation (NT = 500)**
100000	7.99066	7.94815	7.96426	7.95795	0.02509	-0.01742	-0.00131	-0.00762
150000	7.99258	7.97871	7.98057	7.97493	0.02701	0.01314	0.015	0.00936
200000	8.00076	7.97697	7.98365	7.98477	0.03519	0.0114	0.01808	0.0192
300000	7.99765	7.98908	7.98933	7.97273	-0.03208	-0.02351	-0.02376	-0.00716

```
Batch 4 call price: C = 92.1757
Batch 4 put price: P = 1.2475
```

c) From the previous table we can see that higher NT and higher NSIM contribute to reducing the deviation. Consequently, for 2 decimal accuracy we can choose 500 < NT<1001 and 500000 < NSIM < 1000000 and monitor the deviation for each combination. Note that when NT = 1000 and NSIM = 1000000 we obtained an estimate of $1.24861, which is accurate to 2 decimal places.

```
1 factor MC with explicit Euler
Number of subintervals in time: 1000
Number of simulations: 1000000
10000
```

```
990000
1000000
Price, after discounting: 1.24861,
Number of times origin is hit: 0
```

D. Advanced Monte Carlo

Batch 1						
NSIM (simulations)	**Standar Deviation NT = 300**	**Standar Deviation NT = 400**	**Standar Deviation NT = 500**	**Error (NT = 300)**	**Error (NT = 400)**	**Error (NT = 500)**
100000	6.06286	6.04473	6.05203	0.0191724	0.0191151	0.0191382
150000	6.06086	6.05123	6.05376	0.0156491	0.0156242	0.0156307
200000	6.06392	6.04835	6.05692	0.0135593	0.0135245	0.0135437
300000	6.06075	6.04962	6.04919	0.01405	0.011045	0.0110443
Batch 2						
NSIM (simulations)	**Standar Deviation NT = 300**	**Standar Deviation NT = 400**	**Standar Deviation NT = 500**	**Error (NT = 300)**	**Error (NT = 400)**	**Error (NT = 500)**
100000	10.4292	10.3994	10.4107	0.0329801	0.0328859	0.0329215
150000	10.4254	10.4165	10.4165	0.0269183	0.0268953	0.0268953
200000	8.00076	10.4084	10.4241	0.0233319	0.0232739	0.0233089
300000	10.4291	10.4123	10.4126	0.0190408	0.0190102	0.0190107

While the number of time steps (NT) appears to have little effect on the accuracy of the result, the standard error consistently decreases as the number of simulations (NSIM)increases. We can conclude that the MC simulation in this case is more sensitive to the number of simulations than to the number of time steps and that its result converges to the exact value as we increase the NSIM.

E. Excel Visualization

Option price for a monotonically increasing range of underlying values of S