Level 9 Homework
Groups A&B: Exact Pricing Methods

Student Kaushik Aryan R
Date August 11th 2025

Introduction

This document presents the development and testing of a comprehensive option pricing framework focused on one-factor European options and Perpetual American options. The implementation leverages object-oriented design principles and template programming to deliver an accurate, flexible, and extensible pricing system.

Design Overview

The implementation employs a modular and hierarchical architecture built using template-based classes to support multiple numeric data types:

1. **Base Option Class**: `OptionContract<NumType>` - serves as the abstract base class with standard functionality
2. **European Options**:
   (a) Base European Option Class: `EuropeanOption<NumType>` (abstract class)
   (b) `EuropeanCall<NumType>` for European Call options
   (c) `EuropeanPut<NumType>` for European Put options
3. **Perpetual American Options**:
   (a) Base Perpetual American Option Class: `PerpetualAmericanOption<NumType>` (abstract class)
   (b) `PerpetualAmericanCall<NumType>` for American Call options
   (c) `PerpetualAmericanPut<NumType>` for American Put options
4. **Supporting Classes**:
   (a) `MatrixPricer` - For pricing options across ranges of parameters
   (b) `MeshGenerator<NumType>` - For generating parameter meshes
   (c) `NormalDistribution<NumType>` - For computing standard normal PDF and CDF using Boost

This modular design allows easy extension to support additional option types while maintaining a consistent interface. The template approach enables working with different numeric types without requiring code duplication in the future.

Matrix Pricing Implementation

The `MatrixPricer` class provides a flexible framework for computing option prices across ranges of parameters:

- `priceRange` - Computes option prices/Greeks across a range of a single parameter (typically S)
- `priceMatrix` - Computes a 2D grid of option prices/Greeks across ranges of two parameters

These methods use functors to set parameters and calculate results, providing maximum flexibility. For example:

```cpp
template<typename OptionType, typename NumType = double>
static std::vector<NumType> priceRange(
    const OptionType& baseOption,
    NumType minSpot,        // Minimum underlying price
    NumType maxSpot,        // Maximum underlying price
    NumType priceStep,      // Price increment
    std::function<NumType(const OptionType&)> evaluationFunction
)
{
    std::vector<NumType> calculatedPrices;
    calculatedPrices.reserve(static_cast<unsigned>((maxSpot - minSpot) / priceStep) + 1);

    for (NumType underlyingPrice = minSpot;
            underlyingPrice <= maxSpot;
            underlyingPrice += priceStep)
    {
        OptionType modifiedOption = baseOption;     // Create a copy
        modifiedOption.S(underlyingPrice);          // Update underlying price
        calculatedPrices.push_back(evaluationFunction(modifiedOption));
    }

    return calculatedPrices;
}

// Generate 2D matrix of option values for parameter combinations
template<typename OptionType, typename NumType = double>
static std::vector<std::vector<NumType>> priceMatrix(
    const OptionType& templateOption,       // Base option for calculations
    MeshGenerator<NumType>& firstParam,     // Grid for first dimension (e.g., spot prices)
    MeshGenerator<NumType>& secondParam,    // Grid for second dimension (e.g., time to expiry)
    std::function<void(OptionType&, NumType)> firstParamSetter,   // Function to set first parameter
    std::function<void(OptionType&, NumType)> secondParamSetter,  // Function to set second parameter
    std::function<NumType(const OptionType&)> valueCalculator     // Function to calculate result
)
{
    // Initialize result matrix
    std::vector<std::vector<NumType>> pricingMatrix;
    pricingMatrix.reserve(firstParam.size());

    // Iterate through first parameter dimension
    for (unsigned rowIndex = 0; rowIndex < firstParam.size(); ++rowIndex) {
        // Initialize current row
        std::vector<NumType> matrixRow;
        matrixRow.reserve(secondParam.size());

        // Iterate through second parameter dimension
        for (unsigned colIndex = 0; colIndex < secondParam.size(); ++colIndex) {
            OptionType workingOption = templateOption;                 // Create working copy
            firstParamSetter(workingOption, firstParam[rowIndex]);    // Set first parameter
            secondParamSetter(workingOption, secondParam[colIndex]);  // Set second parameter
            matrixRow.push_back(valueCalculator(workingOption));      // Calculate and store result
        }

        pricingMatrix.push_back(matrixRow);
    }

    return pricingMatrix;
}
```

This design provides a highly flexible framework for exploring option pricing under various parameter combinations.

The implementation uses `std::function` objects with lambda functions to create versatile functors. Factory methods within the `MatrixPricer` class provide pre-built functors for common operations like pricing, Delta, Gamma, and Vega calculations, as well as parameter setters for all option parameters (S, K, T, r, σ, b).

Design Decisions

Several key design decisions were made to ensure a robust and flexible implementation:

- **Template-Based Design**: Using templates allows for different numeric types and promotes code reuse.
- **Abstract Base Classes**: `OptionContract`, `EuropeanOption`, and `PerpetualAmericanOption` provide common interfaces and implementations.
- **Separation**: The pricing logic is separated from mesh generation and matrix computation.
- Usage of functors in the `MatrixPricer` class to allow for flexible parameter setting and calculation.
- Checks for valid inputs (e.g., positive S, K, non-negative T, σ) to protect against invalid parameters.

Part A: Exact Solutions of One-Factor Plain Options

1.a. Call/Put Option Pricing for given Data Sets

The solution implements a generalized Black-Scholes framework for European options with a cost-of-carry parameter `b`.
Testing with the four provided batches shows that our implementation matches the expected results:

Batch	Parameters	Expected Call	Computed Call	Expected Put	Computed Put	Parity Check
1	T=0.25, K=65, σ=0.30, r=0.08, S=60	2.13337	2.13337	5.84628	5.84628	PASS
2	T=1.0, K=100, σ=0.2, r=0.0, S=100	7.96557	7.96557	7.96557	7.96557	PASS
3	T=1.0, K=10, σ=0.5, r=0.12, S=5	0.20405	0.20405	4.07326	4.07326	PASS
4	T=30.0, K=100, σ=0.30, r=0.08, S=100	92.17570	92.17570	1.24750	1.24750	PASS

1.b. Put-call Parity Relationship

Put-call parity relationship was implemented in two ways:

- As conversion utilities (`putToCall` and `callToPut` static methods)
- As a Validation mechanism (`putCallParityCheck` static method)

All test scenarios as part of Exercise a successfully passed put-call parity verification, confirming theoretical consistency.

1.c. Implement a Mesh Generator

The `MeshGenerator<NumType>` class was implemented to help create uniform parameter grids to be used in batch option pricing calculations. This template-based utility generates equally spaced numerical sequences that serve as input ranges for the `MatrixPricer` operations. Testing with parameters from 1 to 20 with unit increments produces the following uniform distribution:

Generated Price Mesh (Range: 1-20, Step: 1)									
1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20

Table 1: Mesh generator output showing 20 uniformly distributed grid points

1.d. Compute Option prices as a function of Option Parameters (S & T)

The `MatrixPricer` class demonstrates sophisticated batch processing capabilities through comprehensive two-dimensional parameter analysis. Testing with European put options across spot price ranges (91-100) and time-to-maturity grids (1-10) validates the framework's ability to generate complete pricing surfaces:

European Put Option Price Matrix

Spot\Time	1	2	3	4	5	6	7	8	9	10
91	11.6	12.08	11.88	11.4	10.79	10.13	9.458	8.793	8.152	7.54
92	11.14	11.73	11.59	11.15	10.58	9.948	9.297	8.652	8.027	7.429
93	10.71	11.39	11.31	10.91	10.37	9.767	9.139	8.513	7.905	7.321
94	10.28	11.06	11.03	10.68	10.17	9.591	8.985	8.378	7.785	7.215
95	9.874	10.73	10.76	10.45	9.972	9.418	8.833	8.244	7.667	7.111
96	9.477	10.42	10.5	10.22	9.778	9.249	8.685	8.114	7.552	7.008
97	9.094	10.11	10.24	10	9.588	9.084	8.54	7.986	7.439	6.908
98	8.725	9.811	9.991	9.79	9.403	8.922	8.398	7.861	7.328	6.81
99	8.368	9.521	9.747	9.581	9.221	8.763	8.259	7.738	7.219	6.713
100	8.023	9.239	9.509	9.376	9.043	8.608	8.122	7.617	7.112	6.618

Table 2: European Put option pricing matrix demonstrating two-dimensional parameter analysis
(rows: T=(1-10), columns: S=(91-100), K=100, σ=0.30, r=0.08)

European Call Option Price Matrix

Spot\Time	1	2	3	4	5	6	7	8	9	10
91	10.29	17.87	24.22	29.79	34.76	39.25	43.34	47.06	50.48	53.61
92	10.83	18.52	24.93	30.54	35.55	40.07	44.18	47.92	51.35	54.5
93	11.4	19.17	25.65	31.3	36.34	40.89	45.02	48.78	52.23	55.39
94	11.97	19.84	26.37	32.06	37.14	41.71	45.86	49.65	53.11	56.28
95	12.56	20.52	27.1	32.83	37.94	42.54	46.71	50.52	53.99	57.18
96	13.17	21.2	27.84	33.61	38.75	43.37	47.56	51.38	54.88	58.08
97	13.78	21.89	28.58	34.39	39.56	44.21	48.42	52.26	55.76	58.98
98	14.41	22.6	29.33	35.18	40.37	45.04	49.28	53.13	56.65	59.88
99	15.06	23.31	30.08	35.97	41.19	45.88	50.14	54.01	57.54	60.78
100	15.71	24.02	30.85	36.76	42.01	46.73	51	54.89	58.44	61.69

Table 3: European Call option pricing matrix demonstrating two-dimensional parameter analysis
(rows: T=(1-10), columns: S=(91-100), K=100, σ=0.30, r=0.08)

The matrices demonstrate the framework's capability to systematically calculate option prices across multi-dimensional parameter spaces.

2. Greeks Implementation and Analysis

2.a. Implementing the Greeks for a given Data Set

The Greeks (Delta Δ, Gamma Γ, Vega v, and Theta Θ) were implemented using the provided analytical formulas. For the data set (T=0.5, K=100, S=105, σ=0.36, r=0.1, b=0), the following values for the Greeks were obtained:

Greek	Put	Call
Δ Delta	0.3566	0.5946
Γ Gamma	0.01349	0.01349
Θ Theta	8.872	8.397
Vega	26.78	26.78

These results were verified to align with theoretical expectations for Delta values provided.

2.b. Call Option Deltas and Gammas for monotonically increasing Spot values

Using the `MatrixPricer` class, the Delta and Gamma values for call options over a range of Spot Prices (S $\in$ [10, 50]) were generated. The results show how these Greeks vary with the underlying price.

Spot Price	Delta Value	Gamma Value
10	2.256e-19	7.999e-19
11	6.182e-18	1.911e-17
12	1.125e-16	3.065e-16
13	1.467e-15	3.55e-15
14	1.449e-14	3.14e-14
15	1.134e-13	2.214e-13
16	7.275e-13	1.287e-12
17	3.938e-12	6.346e-12
18	1.839e-11	2.711e-11
19	7.552e-11	1.022e-10
20	2.769e-10	3.453e-10
21	9.183e-10	1.059e-09
22	2.786e-09	2.977e-09
23	7.804e-09	7.749e-09
24	2.035e-08	1.882e-08
25	4.973e-08	4.295e-08
26	1.146e-07	9.26e-08
27	2.506e-07	1.897e-07
28	5.217e-07	3.707e-07
29	1.039e-06	6.94e-07
30	1.987e-06	1.249e-06
31	3.659e-06	2.169e-06
32	6.509e-06	3.643e-06
33	1.122e-05	5.933e-06
34	1.876e-05	9.389e-06
35	3.054e-05	1.447e-05
36	4.844e-05	2.176e-05
37	7.504e-05	3.199e-05
38	0.0001137	4.602e-05
39	0.0001687	6.489e-05
40	0.0002455	8.981e-05
41	0.0003508	0.0001221
42	0.0004927	0.0001634
43	0.000681	0.0002152
44	0.0009272	0.0002793
45	0.001244	0.0003576
46	0.001648	0.0004519
47	0.002154	0.0005642
48	0.002783	0.0006963
49	0.003554	0.0008499
50	0.00449	0.001027

Table 4: Call option Delta and Gamma values for spot prices from 10 to 50

These results demonstrate the exponential growth pattern of both Delta and Gamma values as options move closer to being in-the-money. For deep out-of-the-money options (S = 10-25), both Greeks are extremely small, indicating minimal sensitivity to underlying price changes. As the spot price increases toward 50, both sensitivities increase significantly, with Delta approaching meaningful values and Gamma showing the acceleration of this sensitivity.

2.c. Compute Option Deltas as a function of Option Parameters (S & K)

A Delta matrix was generated across Spot Prices S ∈ [90-100] and Strike Prices K ∈ [100-110], showing how Delta varies with both underlying price and strike price simultaneously.

S\K	100	101	102	103	104	105	106	107	108	109	110
90	0.3683	0.3542	0.3403	0.3268	0.3136	0.3008	0.2883	0.2762	0.2644	0.2530	0.2419
91	0.3842	0.3699	0.3559	0.3421	0.3287	0.3156	0.3029	0.2905	0.2784	0.2667	0.2553
92	0.4001	0.3856	0.3714	0.3575	0.3439	0.3306	0.3176	0.3050	0.2926	0.2806	0.2690
93	0.4160	0.4014	0.3870	0.3730	0.3592	0.3456	0.3324	0.3195	0.3070	0.2947	0.2828
94	0.4318	0.4171	0.4026	0.3884	0.3744	0.3608	0.3474	0.3342	0.3214	0.3090	0.2968
95	0.4475	0.4327	0.4181	0.4038	0.3897	0.3759	0.3623	0.3490	0.3360	0.3233	0.3109
96	0.4631	0.4482	0.4336	0.4192	0.4050	0.3910	0.3773	0.3639	0.3507	0.3378	0.3252
97	0.4785	0.4637	0.4490	0.4345	0.4202	0.4062	0.3923	0.3787	0.3654	0.3523	0.3395
98	0.4938	0.4790	0.4643	0.4498	0.4354	0.4213	0.4073	0.3936	0.3801	0.3669	0.3539
99	0.5089	0.4941	0.4794	0.4649	0.4505	0.4363	0.4222	0.4084	0.3948	0.3815	0.3683
100	0.5238	0.5090	0.4944	0.4799	0.4654	0.4512	0.4371	0.4232	0.4095	0.3960	0.3828

Table 5: Delta surface matrix for European Call options (Spot Prices S ∈ [90-100] and Strike Prices K ∈ [100-110])

The matrix demonstrates the expected behaviour: Delta decreases as the strike price increases (moving from left to right) and increases as the spot price increases (moving from top to bottom). The values range from approximately 0.24 for deep out-of-the-money options to 0.52 for in-the-money options.

2.d. Divided Differences Method

The divided differences methods were implemented to numerically approximate Delta and Gamma values:

$$\Delta = \frac{V(S+h) - V(S-h)}{2h} \tag{1}$$

$$\tag{2}$$

$$\Gamma = \frac{V(S+h) - 2V(S) + V(S-h)}{h^2} \tag{3}$$

For the Test Case (K=100, S=105, T=0.5, σ=0.36, r=0.1, b=0), the following values for the Greeks were obtained:

Greek Type	Put Value	Call Value
Γ Gamma	0.01349	0.01349
Δ Delta	-0.3566	0.5946
Θ Theta	-8.872	-8.397
Vega	26.78	26.78

Table 6: Greeks computed using divided differences method

The results obtained from using the divided differences methods were then validated against the analytical values of the Greeks.

To further validate the accuracy, a comprehensive comparison between analytical and divided differences Delta calculations was performed across a range of spot prices. The results demonstrate exceptional accuracy:

Table 7: Delta Comparison Results

Spot	Analytical	Divided Diff	Absolute Error
10	2.256e-19	2.256e-19	4.537e-25
11	6.182e-18	6.182e-18	9.43e-24
12	1.125e-16	1.125e-16	1.329e-22
13	1.467e-15	1.467e-15	1.365e-21
14	1.449e-14	1.449e-14	1.078e-20
15	1.134e-13	1.134e-13	6.833e-20
16	7.275e-13	7.275e-13	3.592e-19
17	3.938e-12	3.938e-12	1.608e-18
18	1.839e-11	1.839e-11	6.265e-18
19	7.552e-11	7.552e-11	2.163e-17
20	2.769e-10	2.769e-10	6.713e-17
21	9.183e-10	9.183e-10	1.898e-16
22	2.786e-09	2.786e-09	4.929e-16
23	7.804e-09	7.804e-09	1.19e-15
24	2.035e-08	2.035e-08	2.683e-15
25	4.973e-08	4.973e-08	5.698e-15
26	1.146e-07	1.146e-07	1.145e-14
27	2.506e-07	2.506e-07	2.19e-14
28	5.217e-07	5.217e-07	4.004e-14
29	1.039e-06	1.039e-06	7.022e-14
30	1.987e-06	1.987e-06	1.186e-13
31	3.659e-06	3.659e-06	1.933e-13
32	6.509e-06	6.509e-06	3.052e-13
33	1.122e-05	1.122e-05	4.676e-13
34	1.876e-05	1.876e-05	6.971e-13
35	3.054e-05	3.054e-05	1.013e-12
36	4.844e-05	4.844e-05	1.437e-12
37	7.504e-05	7.504e-05	1.999e-12
38	0.0001137	0.0001137	2.705e-12
39	0.0001687	0.0001687	3.605e-12
40	0.0002455	0.0002455	4.731e-12
41	0.0003508	0.0003508	6.102e-12
42	0.0004927	0.0004927	7.704e-12
43	0.000681	0.000681	9.608e-12
44	0.0009272	0.0009272	1.181e-11
45	0.001244	0.001244	1.439e-11
46	0.001648	0.001648	1.708e-11
47	0.002154	0.002154	2.042e-11
48	0.002783	0.002783	2.381e-11
49	0.003554	0.003554	2.748e-11
50	0.00449	0.00449	3.14e-11

The complete analysis across all 41 spot price points yields a **mean absolute error of 4.566e-12**, demonstrating the exceptional accuracy of the divided differences method with the chosen step size.

Testing with different step sizes (h) shows that an optimal h exists. If h is too small, round-off errors dominate; if h is too large, truncation errors dominate.

Our analysis demonstrates that:

- For Delta calculations, h $\approx$ 0.001 provides optimal results
- For Gamma calculations, which involve h^2, optimal h is around 0.01-0.001
- Mean absolute error (MAE) is minimized at these values

The table below shows how MAE varies with step size h:

Step Size (h)	MAE Δ	MAE Γ
1e-16	0.0004647	0.0001224
1e-15	0.0004647	88.07
1e-14	0.0007536	4.85e+11
1e-13	0.0001059	3.279e+09
1e-12	1.023e-05	5.933e+07
1e-11	1.017e-06	5.367e+05
1e-10	1.427e-07	5492
1e-09	1.587e-08	32.18
1e-08	7.762e-10	0.7228
1e-07	9.936e-11	0.003609
1e-06	1.473e-11	2.743e-05
1e-05	7.059e-13	3.404e-07
1e-04	1.495e-13	3.129e-09
1e-03	4.566e-12	3.252e-11
1e-02	4.565e-10	4.12e-11
1e-01	4.566e-08	4.091e-09

This demonstrates the trade-off between truncation error and round-off error in numerical differentiation.

Part B: Perpetual American Options

a. Incorporate procing formulae in to classes

Perpetual American options (options with no expiry date) were implemented according to the provided formulas.

b. Call/Put Option Pricing for given Data Set

Testing with the provided parameters (K=100, σ=0.1, r=0.1, b=0.02, S=110) yielded:

- Call price: 18.50 (matching expected: 18.50)
- Put price: 3.031 (matching expected: 3.031)

c. Call/Put Option Prices for monotonically increasing Spot values

The `MatrixPricer` class was used to calculate perpetual American option prices across a range of underlying prices $S \in [10, 50]$ and across a grid of S and K values. The results demonstrate how these prices vary with the parameters.

Spot Price	Call Price	Put Price
10	8.262e-03	9.035e+06
11	1.123e-02	4.996e+06
12	1.485e-02	2.908e+06
13	1.922e-02	1.768e+06
14	2.439e-02	1.115e+06
15	3.045e-02	7.264e+05
16	3.748e-02	4.863e+05
17	4.555e-02	3.336e+05
18	5.474e-02	2.338e+05
19	6.514e-02	1.671e+05
20	7.683e-02	1.215e+05
21	8.988e-02	8.968e+04
22	1.044e-01	6.716e+04
23	1.204e-01	5.094e+04
24	1.381e-01	3.910e+04
25	1.575e-01	3.033e+04
26	1.787e-01	2.377e+04
27	2.017e-01	1.880e+04
28	2.268e-01	1.499e+04
29	2.539e-01	1.206e+04
30	2.831e-01	9.765e+03
31	3.146e-01	7.964e+03
32	3.485e-01	6.537e+03
33	3.847e-01	5.399e+03
34	4.235e-01	4.485e+03
35	4.649e-01	3.745e+03
36	5.090e-01	3.143e+03
37	5.559e-01	2.651e+03
38	6.057e-01	2.246e+03
39	6.585e-01	1.911e+03
40	7.144e-01	1.633e+03
41	7.734e-01	1.400e+03
42	8.358e-01	1.206e+03
43	9.015e-01	1.041e+03
44	9.707e-01	9.028e+02
45	1.043e+00	7.851e+02
46	1.120e+00	6.848e+02
47	1.200e+00	5.991e+02
48	1.284e+00	5.256e+02
49	1.372e+00	4.624e+02
50	1.464e+00	4.078e+02

Table 8: Perpetual Option Prices

An interesting observation is that perpetual American put prices are significantly higher for lower underlying prices, as expected, due to the ability to exercise at any time. The put prices show a dramatic decrease as the spot price increases, reflecting the decreasing probability of profitable exercise.

d. Compute Option Prices as a function of Option Parameters (S & K)

Comprehensive pricing matrices were generated for both perpetual puts and calls across spot prices 1-10 and strike prices 1-10:

Perpetual Put Price Matrix

S\K	1	2	3	4	5	6	7	8	9	10
1	5.48e-02	8.16e+00	1.52e+02	1.21e+03	6.07e+03	2.26e+04	6.89e+04	1.81e+05	4.22e+05	9.04e+05
2	7.37e-04	1.10e-01	2.05e+00	1.63e+01	8.16e+01	3.04e+02	9.26e+02	2.43e+03	5.68e+03	1.22e+04
3	5.93e-05	8.82e-03	1.65e-01	1.31e+00	6.56e+00	2.45e+01	7.44e+01	1.95e+02	4.57e+02	9.77e+02
4	9.91e-06	1.47e-03	2.75e-02	2.19e-01	1.10e+00	4.09e+00	1.25e+01	3.26e+01	7.63e+01	1.63e+02
5	2.47e-06	3.68e-04	6.87e-03	5.48e-02	2.74e-01	1.02e+00	3.11e+00	8.15e+00	1.91e+01	4.08e+01
6	7.97e-07	1.19e-04	2.21e-03	1.76e-02	8.82e-02	3.29e-01	1.00e+00	2.62e+00	6.14e+00	1.31e+01
7	3.06e-07	4.55e-05	8.48e-04	6.76e-03	3.38e-02	1.26e-01	3.84e-01	1.01e+00	2.35e+00	5.04e+00
8	1.33e-07	1.98e-05	3.70e-04	2.95e-03	1.48e-02	5.50e-02	1.67e-01	4.39e-01	1.03e+00	2.20e+00
9	6.40e-08	9.53e-06	1.78e-04	1.42e-03	7.09e-03	2.64e-02	8.04e-02	2.11e-01	4.93e-01	1.06e+00
10	3.33e-08	4.95e-06	9.23e-05	7.36e-04	3.69e-03	1.37e-02	4.18e-02	1.10e-01	2.56e-01	5.48e-01

Table 9: Perpetual American put price matrix (spot prices 1-10, strike prices 1-10)

Perpetual Call Price Matrix

S\K	1	2	3	4	5	6	7	8	9	10
1	1.36e-01	2.93e-02	1.19e-02	6.30e-03	3.84e-03	2.56e-03	1.82e-03	1.36e-03	1.04e-03	8.26e-04
2	1.27e+00	2.72e-01	1.11e-01	5.86e-02	3.57e-02	2.38e-02	1.69e-02	1.26e-02	9.70e-03	7.68e-03
3	4.67e+00	1.00e+00	4.09e-01	2.16e-01	1.32e-01	8.79e-02	6.24e-02	4.64e-02	3.58e-02	2.83e-02
4	1.18e+01	2.53e+00	1.03e+00	5.45e-01	3.32e-01	2.22e-01	1.58e-01	1.17e-01	9.02e-02	7.14e-02
5	2.41e+01	5.19e+00	2.11e+00	1.12e+00	6.81e-01	4.55e-01	3.23e-01	2.40e-01	1.85e-01	1.46e-01
6	4.34e+01	9.33e+00	3.80e+00	2.01e+00	1.22e+00	8.17e-01	5.81e-01	4.32e-01	3.33e-01	2.63e-01
7	7.13e+01	1.53e+01	6.24e+00	3.30e+00	2.01e+00	1.34e+00	9.53e-01	7.09e-01	5.46e-01	4.32e-01
8	1.10e+02	2.36e+01	9.58e+00	5.07e+00	3.09e+00	2.06e+00	1.47e+00	1.09e+00	8.39e-01	6.64e-01
9	1.60e+02	3.44e+01	1.40e+01	7.40e+00	4.51e+00	3.01e+00	2.14e+00	1.59e+00	1.23e+00	9.70e-01
10	2.24e+02	4.83e+01	1.97e+01	1.04e+01	6.33e+00	4.23e+00	3.00e+00	2.23e+00	1.72e+00	1.36e+00

Table 10: Perpetual American call price matrix (spot prices 1-10, strike prices 1-10)

The matrices demonstrate the expected pricing behaviour:

- Put prices increase dramatically with higher strike prices and decrease with higher spot prices
- Call prices increase with higher spot prices and decrease with higher strike prices
- The extreme values in the put matrix for low spot/high strike combinations reflect the high intrinsic value and immediate exercise opportunity