

C++ IN QUANTITATIVE FINANCE FINAL PROJECT

QUOC FREJTER

PROJECT DESCRIPTION

The aim of this project was to write a C++ program to price a down-and-in barrier put option using Monte Carlo simulations.

ASSUMPTIONS

To calculate path dependent option, first we need to assume underlying asset motion. We assume that underlying (stock) follows geometric Brownian motion which is described by $dS_t = \mu S_t dt + \sigma S_t dW_t$,

Using Ito's lemma to solve this equation we obtain $\log S_t = \log S_0 + \left(r - \frac{1}{2}\sigma^2\right)t + \sigma W_t$.

Where wiener process can be rewritten as $W_T = \sqrt{T}x$, where $x \sim N(0,1)$. This leads to the discretized form for small time steps Δt : $\log S_{t+\Delta t} = \log S_t + \left(r - \frac{1}{2}\sigma^2\right)\Delta t + \sigma\sqrt{\Delta t}x$. This discretized form is used to simulate the evolution of the underlying asset's price path over time in Monte Carlo simulations. For basic case of simulation number of timestep interval is set to 100 and time to maturity is 0,75. This means that each time step is approximately two days

OPTION CHARACTERISTICS

The assigned put option was activated by the knock-in barrier.

Parameters of options are following:

- Initial price of underlying 100
- Strike price 110
- Volatility 25%
- Risk free rate 5%
- Time to maturity 0.75

CODE

Project file structure is following:

```

.
├── ./includes
│   ├── ./includes/gen_rand.cpp
│   ├── ./includes/gen_rand.h
│   ├── ./includes/helpers.cpp
│   ├── ./includes/helpers.h
│   ├── ./includes/options.cpp
│   └── ./includes/options.h
└── ./main.cpp

```

Below is the description of each code file in the project:

Code file	Description
main.cpp	The main program file that executes the core functionality.
options.cpp	Contains the implementation of the class <code>Option</code> .
helpers.cpp	Includes additional utility functions used throughout the code.
gen_rand.cpp	Implements functions for generating random variables with a normal distribution.

File `options.cpp` contains class `Option` which has following data members:

Type	Datatype	Name	Description
private	double	strike	Strike price
	double	spot	Spot price
	double	vol	Annualized volatility
	double	r	Risk-free rate
	double	expiry	Time to expiry in years
	unsigned int	n_intervals	Number of intervals for simulation
	unsigned int	n_sim	Number of simulations
	vector<double>	this_path	Path of underlying from single simulation
	vector<double>	this_payout	Payout of the option for each simulation
	double	price	Price of the option (initial is -1 which indicates that none simulations was run)
Public	enum	OptionType	Group of constants <code>call</code> or <code>put</code> indicating option type
	enum	OptionStyle	Group of constants <code>european</code> or <code>barrier</code> indicating option style
	enum	BarrierType	Group of constants <code>down_in</code> , <code>down_out</code> , <code>up_in</code> , <code>up_out</code> indicating option type

Methods of class Option are following:

Return type	Name	Arguments	Description
	Option		Constructs an Option object with default values as defined in homework assignment
	Option	<ul style="list-style-type: none"> • <code>_strike</code> • <code>_spot</code> • <code>_vol</code> • <code>_r</code> • <code>_expiry</code> • <code>_n_intervals</code> 	Constructs an Option object by overwriting the default constructor with user-defined parameters.
void	print_details		Prints details of option object.
void	generate_path		Generates single path with n_intervals defined in the constructor.
void	generate_path	<ul style="list-style-type: none"> • <code>n</code> 	Generates a single path with n intervals, currently used for pricing European options where n = 1 (not path-dependent).
void	print_path		Prints last simulated path.
double	get_payout	<ul style="list-style-type: none"> • <code>type</code> • <code>end_price</code> 	Calculates payout given OptionType type and end price.
bool	is_barrier_touched	<ul style="list-style-type: none"> • <code>barrier_price</code> • <code>type</code> 	Check if barrier is touched given barrier price and BarrierType type .
double	get_price	<ul style="list-style-type: none"> • <code>n_sim</code> • <code>tyoe</code> • <code>style</code> 	Calculates price of European option given number of simulations, OptionType type and OptionStyle style .
double	get_price	<ul style="list-style-type: none"> • <code>n_sim</code> • <code>tyoe</code> • <code>style</code> • <code>b_tyoe</code> • <code>barrier_price</code> 	Calculates price of European /Barrier option given number of simulations, OptionType type , OptionStyle style , BarrierType b_type and barrier price.
double	get_price		Returns price of option if pricing simulations were made.
double	get_stddev		Returns standard deviation of the option price if pricing simulations were made.
void	print_barrier_sensitivity	<ul style="list-style-type: none"> • <code>base_barrier</code> • <code>range</code> • <code>type</code> • <code>style</code> • <code>b_type</code> 	Calculates and prints the option price sensitivity to changes in the barrier price.

It's important to note that the barrier price is not defined as a separate member of the **Option** class, but rather as an argument in the **get_price** method. This design choice differs from what was suggested in the

assignment description. I found it more efficient to analyze the option's price sensitivity to different barrier prices by running the method with varying barrier price values, rather than creating a new `Option` object for each experiment or modifying the option's data members.

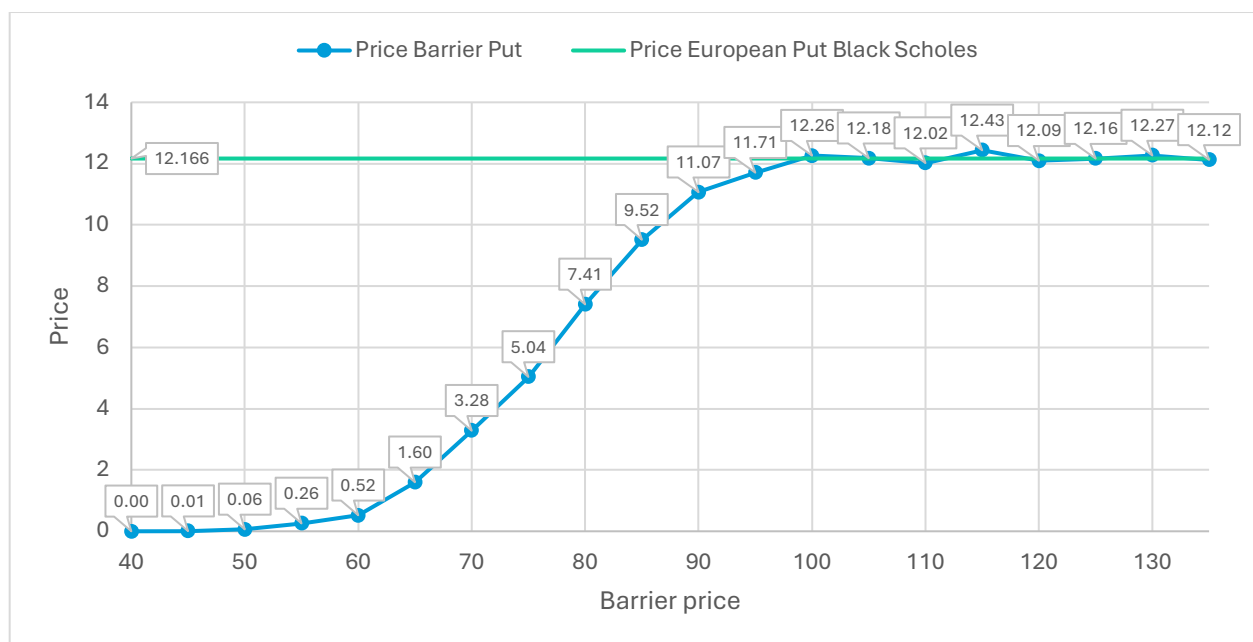
Additional functions from `helpers.cpp` and `gen_rand.cpp` are:

Return type	Name	Arguments	Description
double	<code>generate_rand_norm</code>		Generates a random number from the standard normal distribution using the modified Box-Muller method.
double	<code>get_max</code>	<ul style="list-style-type: none"> • <code>a</code> • <code>b</code> 	Calculates the maximum value between two given numbers.
double	<code>get_arr_mean</code>	<ul style="list-style-type: none"> • <code>arr</code> 	Calculates the mean of the elements in the given vector.
double	<code>get_arr_stddev</code>	<ul style="list-style-type: none"> • <code>arr</code> 	Calculates the standard deviation of the elements in the given vector.
double	<code>get_arr_stddev</code>	<ul style="list-style-type: none"> • <code>arr</code> • <code>mean</code> 	Calculates the standard deviation of the elements in the given vector, with the mean provided.

RESULTS

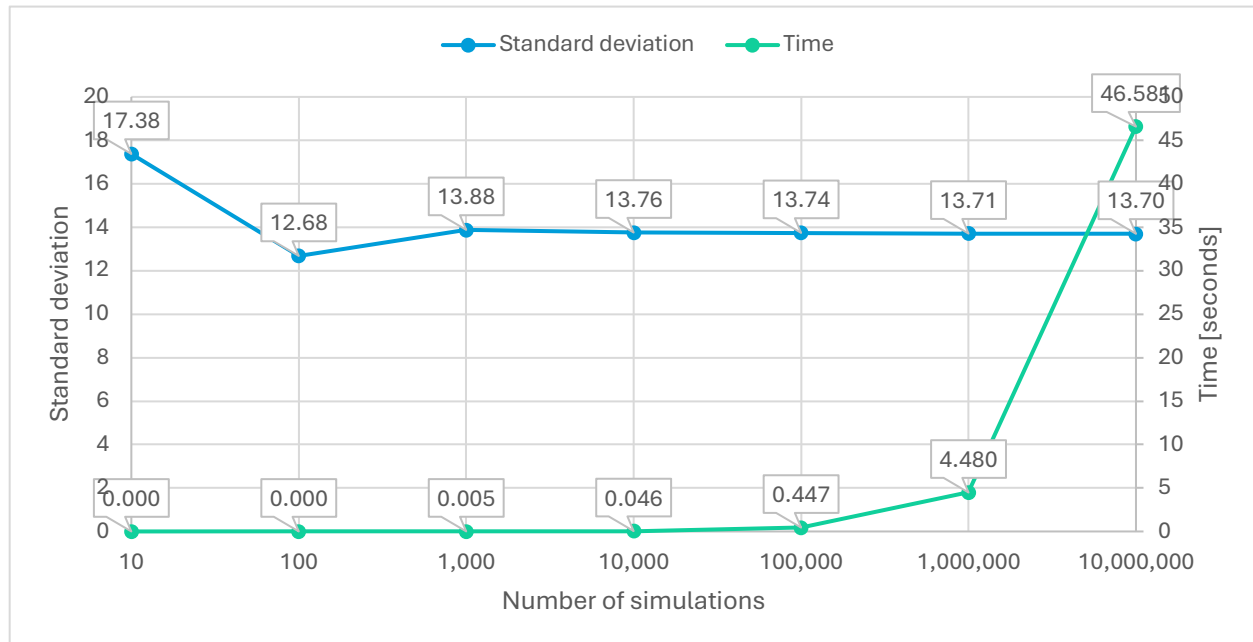
The price of a down-and-in put option with a barrier price of 90 is 11.0687. When the barrier price is set equal to the strike price, the option effectively becomes a regular European put option, as it will only generate profit if the underlying asset reaches the strike price. Similarly, setting the barrier price above the initial price transforms the option into a standard European put, as the barrier is immediately activated.

The graph below illustrates the sensitivity of the option price to changes in the barrier price.



As shown, setting the barrier price above the initial price of 100 has no impact on the option price since the barrier is already activated. However, as the barrier price decreases, the option becomes less likely to be activated. When it is activated, the profit potential increases because the option is only triggered when the underlying price drops significantly, aligning with the put option's purpose. Once the barrier price is set below 40, the expected profit becomes zero, as the option is unlikely to be triggered.

The standard deviation of option prices in the baseline scenario (10,000 simulations) was 13.76, which is unsatisfactory. Such a high standard deviation results in a wide confidence interval for the valuation, making the results less reliable and almost insignificant. While increasing the number of simulations reduces the standard deviation slightly, even with 10,000,000 simulations, the standard deviation remains unsatisfactory.



SUMMARY

Barrier Price Impact:

- Setting the barrier above the initial price converts the option to a European put.
- Lowering the barrier reduces activation likelihood but increases profit potential when triggered.

Simulation Results:

- Initial standard deviation was 13.76 for 10,000 simulations, leading to wide confidence intervals.
- Increasing simulations to 10,000,000 slightly reduced the standard deviation but remained unsatisfactory.

The next steps for this project could involve expanding the analysis to cover all types of barrier options, including down-and-up knock-in and knock-out options. Additionally, to improve the accuracy of valuations, variance reduction techniques such as antithetic variates should be tested and implemented.