EXCEL VBA in Finance — Final Project

Project 13: Bitcoin prices, options on Bitcoin

ESILV — Practical Session (Excel VBA in Finance, 2025)

1. Objective

The objective of this project is to load approximately one year of historical Bitcoin prices, estimate the historical volatility from log-returns, and price European Call/Put options on Bitcoin using the Black–Scholes model. The algorithm is implemented both in Excel formulas and in VBA , and the results are compared in order to validate.

2. Data source and data loading (STOCKHISTORY)

Historical BTC prices are retrieved from an online market data feed using the Excel function STOCKHISTORY. The workbook downloads daily prices over a year, producing a Date/Price table that is used as input for all calculations. Excel uses this formula to compute.

$$C = S\,N(d_1) - Ke^{-rT}N(d_2)$$

3. Methodology

3.1 Log-returns

From the daily price series P_t, we compute daily log-returns:

$$r_t = \ln\left(\frac{P_t}{P_{t-1}}\right)$$

We use log-returns rather than simple returns because they are additive over time and they provide a more stable input for statistical estimation. For daily data, log-returns are also close to simple returns , which makes them convenient for volatility estimation.

3.2 Historical volatility (daily and annualized)

We estimate daily volatility as the sample standard deviation of log-returns:

$$\sigma_{\text{daily}} = \text{stdev}(r_t)$$

We annualize volatility using the crypto-market convention (trading 7 days/week):

$$\sigma_{\text{annual}} = \sigma_{\text{daily}}\sqrt{N}$$

Here we have bitcoin so N the number of trading days is 365

In the workbook, volatility is computed with a VBA function

Historical_Volatility, which internally computes log-returns and returns σ_annual.

3.3 Black–Scholes European option pricing

Black–Scholes assumes a lognormal diffusion for the underlying price, a constant risk-free rate r, constant volatility σ. While these assumptions are simplified for cryptocurrencies, the model provides a standard baseline and allows a clean comparison between the Excel and VBA implementations.

Using inputs Spot S, Strike K, Maturity T (in years), risk-free rate r, and volatility σ, we compute:

$$d_1 = \frac{\ln(S/K) + (r + \frac{1}{2}\sigma^2)T}{\sigma\sqrt{T}}$$

$$d_2 = d_1 - \sigma\sqrt{T}$$

$$P = Ke^{-rT}N(-d_2) - SN(-d_1)$$

4. Implementation (Excel + VBA)

This project follows the guideline to implement the algorithm in both Excel and VBA. Excel provides transparency and easy interface (returns, σ, d1, d2), while VBA provides reusable functions that can be called with simple inputs. This dual implementation is useful for validation: if both approaches give the same prices, the risk of formula mistakes is significantly reduced

4.1 VBA functions

We used these VBA functions for our project:

• **Historical_Volatility(PriceRange As Range) As Double**

```
Function Historical_Volatility(PriceRange As Range) As Double
    Dim i As Long
    Dim RowCount As Long
    Dim PriceData As Variant
    Dim ReturnsArray() As Double
    Dim DailyVol As Double

    ' Load range data into an array for faster processing
    PriceData = PriceRange.Value
    RowCount = UBound(PriceData, 1)

    ' Redimension the returns array (N-1 returns for N prices)
    ReDim ReturnsArray(1 To RowCount - 1)

    ' Calculate Logarithmic Returns (Ln)
    For i = 2 To RowCount
        ' Validate that values are numeric and positive to avoid errors
        If IsNumeric(PriceData(i, 1)) And IsNumeric(PriceData(i - 1, 1)) Then
            If PriceData(i, 1) > 0 And PriceData(i - 1, 1) > 0 Then
                ReturnsArray(i - 1) = Log(PriceData(i, 1) / PriceData(i - 1, 1))
            Else
                ReturnsArray(i - 1) = 0
            End If
        End If
    Next i

    ' Daily Volatility
    DailyVol = Manual_StdDev(ReturnsArray)
```

• **BlackScholes(OptionType As String, S As Double, K As Double, T As Double, r As Double, sigma As Double) As Double**

```
' BlackScholes function to price a Call or a Put
Function BlackScholes(OptionType As String, S As Double, K As Double, T As Double, r As Double, sigma As Double) As Double
    Dim d1 As Double
    Dim d2 As Double

    ' Calculate d1 and d2
    d1 = (Log(S / K) + (r + (sigma ^ 2) / 2) * T) / (sigma * Sqr(T))
    d2 = d1 - sigma * Sqr(T)

    ' Pricing based on Option Type
    If UCase(OptionType) = "CALL" Then
        BlackScholes = S * Application.WorksheetFunction.Norm_S_Dist(d1, True) - _
                       K * Exp(-r * T) * Application.WorksheetFunction.Norm_S_Dist(d2, True)
    ElseIf UCase(OptionType) = "PUT" Then
        BlackScholes = K * Exp(-r * T) * Application.WorksheetFunction.Norm_S_Dist(-d2, True) - _
                       S * Application.WorksheetFunction.Norm_S_Dist(-d1, True)
    End If
End Function
```

5. Results and validation (Excel vs VBA)

We validate the implementation by comparing Excel and VBA outputs using the same input parameters. In the workbook, the Excel formulas and the VBA return identical Call/Put prices , which confirms correctness of our VBA functions.

Input parameters	Value
Spot price S	87,100.59
Strike K	100,000
Maturity T	0.082192 (≈ 30/365)
Risk-free rate r	4.1082%
Annual volatility σ	43.7694%

d1	-1.010949
d2	-1.136432

Output	Excel	VBA UDF
Call price C	843.790276	843.790276
Put price P	13,406.109433	13,406.109433

6. Conclusion

This project loads BTC daily prices over a one-year window using STOCKHISTORY, computes log-returns and historical volatility, and prices European Bitcoin options with the Black–Scholes model. The Excel and VBA implementations are consistent, which validates the VBA and the overall workflow.

The volatility used here is historical and may differ from implied volatility observed in option markets. Black–Scholes also assumes constant $\sigma$and r, which is restrictive for BTC.