# **Workshop: Optionen, Black-Scholes-Merton-Modell und Monte-Carlo Simulation**

Dieser Workshop beschäftigt sich mit dem Black-Scholes-Merton-Modell (auch bekannt als Black-Scholes-Modell) zur Preisbestimmung sogenannter Optionen.   
Die Option ist ein Finanzprodukt, welches sich auf ein zugrunde liegendes Produkt am Finanzmarkt bezieht (Basis/Underlying). Der Inhaber der Option bezahlt eine Prämie, um zu einem gewissen Zeitpunkt das Kaufrecht auf das zugrunde liegende Finanzprodukt zu einem vorher festgelegten Preis zu erhalten. Die Prämie wird bei der Berechnung des Optionswertes nicht berücksichtigt, da das Modell das Payout und damit den Marktpreis bestimmt. Aufgrund der ökonomischen Bedeutung wurde 1997 der Nobelpreis für Wirtschaftswissenschaften für diese Errungenschaft an Scholes und Merton verliehen (Fischer †1995).

Der Workshop konzentriert sich im Folgenden auf den Wert der Europäischen Call- und Put-Option aus der Sicht des Käufers, d.h. der Long-Position. Er setzt keine ausgeprägten Programmier-/Fachkenntnisse voraus. Die Gruppen sind jedoch so verteilt, dass alle Gruppen eine ausgeglichene Mischung von Programmier- und Fachkenntnissen beinhalten.

Für Fragen, Anmerkungen und Kritik, wendet Euch sehr gerne an mich (s.u.).  
  
Viel Spaß!  
Paul

Hilfreiche Bonusinformationen zur Bearbeitung  
Die Struktur der Jupyter-Notebooks sieht vor, dass einzelne Code-Blöcke mit *str + enter*,  
bzw*. shift + enter* ausgeführt werden können. Stellt sicher, dass nur die Code-Blöcke/Zellen ausgeführt werden, welche Ihr bereits bearbeitet habt, da der gesamte Code erst nach gesamter Bearbeitung fehlerfrei funktioniert.

Beachtet außerdem, dass Änderungen im Code nur registriert werden, wenn Ihr die entsprechenden Zellen erneut ausführt. Dies gilt insbesondere, wenn Ihr die Parameter zu Beginn des Codes ändern wollt, z.B., um Plots mit veränderten Parametern zu generieren.

In Python gilt außerdem die internationale Standardschreibweise mit einem Punkt für Gleitkommazahlen. Um die Nähe zum Code beizubehalten und Verwechslungsgefahr mit Kommata im Text zu vermeiden, halten wir in dieser Anleitung diese Konvention ein.

Gemäß der konventionellen Behandlung des Black-Scholes-Merton Modells geht dieser Workshop außerdem von kontinuierlicher Verzinsung und Diskontierung aus (Siehe z.B.: *Optionen, Futures und andere Derivate, John C. Hull, Pearson (11. Aufl., 2022)*).

## Aufgabe 1 – Lösung der Black-Scholes-Merton Gleichung

|  |  |  |
| --- | --- | --- |
| Gruppe Black | Gruppe Scholes | Gruppe Merton |
| … | … | ... |

Ziele dieser Aufgabe sind, sich mit Funktionen in Python, mit elementaren algebraischen Operatoren, mit den Lösungen der Black-Scholes-Merton Gleichung zum Pricing von Optionen und zu graphischen Tools in Python vertraut zu machen.

Hierzu werden in dieser Aufgabe die Lösungen der Black-Scholes-Merton Gleichung in Python übersetzt, dargestellt, und untersucht.

### Formeln eingeben

Fügt an den entsprechenden Stellen in den dafür vorgesehenen Funktionen im Jupyter-Notebook die Formeln für die Lösung der Black-Scholes-Merton Gleichung ein.   
Die Formeln für den Wert von Call-Option und Put-Option lassen sich schreiben als:

und ,

wobei und .

Hierbei bezeichnet die kumulative Verteilungsfunktion (cumulative distribution function, CDF) der Standardnormalverteilung und lässt sich im Jupyter-Notebook als *norm.cdf(…)* aufrufen. Die Funktion bezeichnet den natürlichen Logarithmus und kann im Jupyter-Notebook als *np.log(...)* aufgerufen werden.

Die Konstanten bezeichnen:  
Spot-Price des Underlying bei Laufzeitbeginn , Strike-Price , risikolose jährliche kontinuierliche Zinsrate , Laufzeit in Jahren und Standardabweichung der Returns des Underlying .

Die Einheiten der Preise sind im Code in EUR. Die Theorie funktioniert jedoch für jede Währung.

### Grenzwerte untersuchen

Der nächste Code-Block beinhaltet Code zur Darstellung des Marktwertes der Call- und Put- Optionen gemäß der Black-Scholes-Merton Gleichung.   
Die Konstanten, welche zur Darstellung genutzt werden, findet Ihr im zweiten Code-Block im Notebook.

Für diesen Aufgabenteil spielt der Wert im Parameter-Block keine Rolle, da verschiedene gleichzeitig betrachtet und auf der horizontalen Achse aufgetragen werden. Dies befindet sich bereits an der entsprechenden Stelle im Code. Die betrachteten Werte liegen im Intervall EUR.

1. Stellt sicher, dass die Parameter zunächst auf die Standardwerte , , und , gesetzt sind.   
   Betrachtet die Lösung für diese Werte durch Ausführung des Code-Blocks zur Darstellung.

Warum steigt der Wert der Call-Option bereits, wenn ?

1. Setzt nun auf . Was beobachtet Ihr? Warum?
2. Setzt nun auf . Was beobachtet Ihr? Warum?
3. Setzt nun die Parameter auf die Standardwerte und . Was beobachtet Ihr? Warum?

## Aufgabe 2 – Monte-Carlo Simulation des Preises des Underlying

|  |  |  |
| --- | --- | --- |
| Gruppe Black | Gruppe Scholes | Gruppe Merton |
| … | … | … |

Das Ziel dieser Aufgabe ist es sich mit der elementaren Struktur der Monte-Carlo Simulation und der Implementation in Python vertraut zu machen. Der Wert der Option wird bestimmt, indem der Preis des Unterlyings, vielfach simuliert wird. Hieraus wird vielfach der Wert der Option bei Laufzeitende und der Erwartungswert über den Mittelwert bestimmt. Der Wert Der Option bei Laufzeitbeginn kann nun durch Diskontierung bestimmt werden.

### 

### Simulationsfunktion

Die zugrunde liegende Dynamik des Wertes des Underlying wird im Rahmen der Black-Scholes-Merton-Theorie mit Hilfe es Log-Normal Prozesses modelliert. Mit anderen Worten, der natürliche Logarithmus folgt in der Simulation der diskreten Dynamik

Hierbei bezeichnet die Diskretisierung, den Drift, und eine standardnormalverteilte Zufallsvariable, welche in jedem Zeitschritt neu gezogen wird.

1. Definiert eine Konstante, die den Logarithmus des Wertes des Unterlying

bezeichnet.

1. Definiert zwei Hilfskonstanten , welche in jedem Zeitschritt erneut genutzt werden, um die Rechenzeit zu minimieren.
2. Ergänzt die entsprechende algebraische Operation innerhalb des *for-loops*.

Eine standardnormalverteilte Zufallsvariable kann mit *np.random.normal()* gezogen werden.

1. Definiert den zurückgegebenen Wert der Funktion (return value) mit Hilfe der Exponentialfunktion *np.exp(…)*.

### Test

Überlegt Euch einige geeignete Test-Cases bzgl. der Inputs und diskutiert diese.

## Aufgabe 3 – Wert der Option

|  |  |  |
| --- | --- | --- |
| Gruppe Black | Gruppe Scholes | Gruppe Merton |
| … | … | … |

Jede Iteration des *for-loop* definiert einen möglichen Verlauf des Preises des Underlying.

Solltet Ihr den Code während der Programmierung zum Testen ausführen, verringert gerne den Wert der Variablen , um eine schnellere Rechenzeit zu gewährleisten. Am Ende sollte der Wert jedoch wieder ausreichend klein sein ().

1. Zu Beginn im *for-loop*: Definiert eine Variable, die den Wert des Underlying bei Maturität definiert.
2. Rechnet gemäß und , die Werte der Optionen nach Ende des möglichen Verlaufs aus.
3. Definiert im Code vor dem *for-loop* eine Variable, die den Mittelwert definiert und fügt Code im Code-Block hinzu, welcher diesen angemessen ermittelt.
4. Bestimmt den Diskontierungsfaktor mit Hilfe der Funktion *np.exp(…)*. Multipliziert den Wert der Option bei Laufzeitende mit und vergleicht das diskontierte Ergebnis der Simulation im Output mit der analytischen Lösung durch das Black-Scholes-Merton-Modell. Die Standardwerte , , und sollten, mit und einen relativen Fehler von weniger als ergeben.