

# Barriere Optionen

Preis und Sensitivität

von

Ananda Eraz Irfananto (547048)

Abschlussprojekt des Moduls Computergestützte Methoden der Finanzmathematik

Wintersemester 21/22

# Contents

<b>1</b>	<b>Einführung</b>	<b>3</b>
<b>2</b>	<b>Finanzinstrumente</b>	<b>4</b>
2.1	Optionen . . . . .	4
2.2	Auszahlung (Payoff) . . . . .	4
2.3	Option Sensitivität . . . . .	5
2.3.1	Delta ( $\Delta$ ) . . . . .	5
2.3.2	Gamma ( $\Gamma$ ) . . . . .	6
2.3.3	Theta ( $\Theta$ ) . . . . .	6
2.3.4	Vega ( $\nu$ ) . . . . .	6
2.3.5	Rho ( $\rho$ ) . . . . .	6
2.4	Barriere Optionen . . . . .	7
<b>3</b>	<b>Implementierung in Python</b>	<b>8</b>
3.1	Barriere Option Preisbewertung . . . . .	8
3.2	Sensitivität . . . . .	12
3.3	Delta ( $\Delta$ ) . . . . .	13
3.4	Gamma ( $\Gamma$ ) . . . . .	13
3.5	Vega ( $\nu$ ) . . . . .	14
3.6	Theta ( $\Theta$ ) . . . . .	14
3.7	Rho ( $\rho$ ) . . . . .	15
<b>4</b>	<b>Quelle</b>	<b>18</b>

# 1 Einführung

Barriere Optionen sind Derivat, die billiger als andere standard Vanilla Optionen sind. Sie sind unterschiedlich im Sinne dass der Wert von dem zugrundeliegenden Wertpapier eine Barriere  $H$ , während der Laufzeit von  $T$ , berühren oder nicht berühren muss. Es gibt zwei Arten von Barriere Optionen: *kick-out* Option und *kick-in* Option. Eine *kick-out* Option bedeutet eine Null-Auszahlung (*payoff*), wenn die Barriere berührt ist. Andererseits, eine *kick-in* Option resultiert in eine Null-Auszahlung, wenn die Barriere nicht berührt ist. Diese beide Situationen machen die Optionpreis zwar billiger als Vanilla Optionen, weil die Option jetzt keinen Wert haben, wo Vanilla Optionen ausgezahlt würde. In dieser Arbeit, wir konzentrieren uns auf *kick-out* Option und deren Sensitivität. Die Implementierung wird auf Python durchgeführt.

## 2 Finanzinstrumente

Ein *Derivat* ist ein Finanzinstrument, dessen Wert von anderen Finanzinstrumenten abgeleitet (derived) ist. Der Wert von dem Derivat ist von dem Wert der zugrunde liegenden Wertpapiere abhängig. Wie sich der Preis des Basiswerts entwickelt, profitiert der Anleger oder verzeichnet er Verluste.

### 2.1 Optionen

Eine Option gibt dem Käufer das Recht, Pflicht aber nicht, ein Finanzprodukt wie Aktie  $S$ , zu einem vereinbarten *Strike-Preis*  $K$  während der Laufzeit  $T$  (maturity), zu kaufen oder zu verkaufen. Eine Option  $X$  garantiert ihrem Besitzer eine nicht-negative, endliche Zahlung

$$X = f(S_T)$$

zur Zeit  $T$ , wobei  $f$  eine nicht-negative, endliche Funktion ist.

Eine *Call-Option* gibt dem Besitzer das Recht die Aktie zu kaufen. Eine *Put-Option* gibt dem Besitzer das Recht die Aktie zu verkaufen. Zwei allgemeine Arten von Optionen sind die *europäische* und *amerikanische* Optionen. Europäische Option hat feste Maturität  $T$  und kann nur an dieser Zeit ausgeübt werden. Amerikanische Option ist eher flexibler, dass die Option irgendwann während der Laufzeit oder Maturität  $T$  ausgeübt werden kann.

### 2.2 Auszahlung (Payoff)

Der Preis, der für die ausgeübte europäische Option ausgezahlt ist, heißt *Streik-Preis*  $K$ . Der Gewinn am Fälligkeit  $T$  oder *payoff* kann man so formulieren:

$$Payoff = \max[0, (S_T - K)] = (S_T - K)^+$$

Für die europäische Call-Option. Analog für die europäische Put-Option:

$$Payoff = \max[0, (K - S_T)] = (K - S_T)^+$$

Figure 1<sup>1</sup> zeigt die Auszahlungsprofil einer europäischen Option.

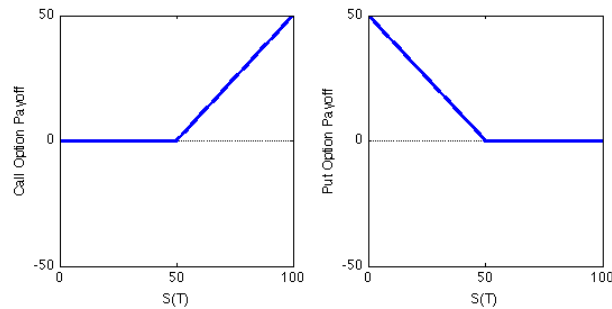


Figure 1: Payoff europäische Call bzw. Put-Option

## 2.3 Option Sensitivität

Sensitivität (oder *Greeks*) ist ein wichtiges Instrument in Finanzmathematik. Um europäische Optionen bewerten zu können, werden die Sensitivität des Optionspreises zu den Parameter berechnet. Sie ist eine partielle Ableitung des Optionspreises nach verschiedene Parameter und bezeichnet man mit griechischen Buchstaben. Sie beschreibt, wie verhält sich der Optionspreis unter kleine Änderung von den Parametern.

### 2.3.1 Delta ( $\Delta$ )

Delta ist die partielle Ableitung des Call-Preises nach dem Underlying. Delta misst, wie stark sich der Call-Preis ändert bei Änderung des Aktienpreises um eine Einheit.

$$\Delta = \frac{\partial C}{\partial S_0}$$

<sup>1</sup><https://www.researchgate.net/profile/Joshua-Knight/publication/272165209/figure/fig2/AS:669541793284097@1536642611104/Payoffs-for-a-European-call-option-left-and-put-option-right-with-strike-price-of.ppm>

### 2.3.2 Gamma ( $\Gamma$ )

Gamma ist die zweite partielle Ableitung des Call-Preises nach dem Preis der Underlyings.

$$\Gamma = \frac{\partial^2 C}{\partial^2 S_0}$$

### 2.3.3 Theta ( $\Theta$ )

Theta misst die Sensitivität des Calls zu einer negativen Laufzeit, also die partielle Ableitung nach negativen Laufzeit. Bei verkürzender Laufzeit sinkt der Call-Optionspreis, mittlerweile beim Put hängt dieses Verhalten sehr stark von dem aktuellen Kurs ab.

$$\Theta = \frac{\partial C}{\partial (-T)}$$

### 2.3.4 Vega ( $\nu$ )

Vega ist die partielle Ableitung nach der Volatilität  $\sigma$ . Bei steigender Volatilität steigt der Preis einer Call- bzw. Put-Option.

$$\nu = \frac{\partial C}{\partial \sigma}$$

### 2.3.5 Rho ( $\rho$ )

Rho misst die Sensitivität des Preises zum Zinssatz. Bei zunehmendem Zins steigt der Preis von Call und sinkt der Preis von Put.

$$\rho = \frac{\partial C}{\partial r}$$

## 2.4 Barriere Optionen

Barriere Optionen ändern sich sprungweise, wenn der Aktienpreis  $S_t$  eine gegebene Barriere  $H$  zum Laufzeitende  $T$  erreichen. Angenommen eine einfache europäische barriere Option

Laufzeit  $T$ , Strike-Preis  $K$ , Barriere  $H$

gibt der Besitzer das Recht, eine Aktie zur Zeit  $T$  mit Strike-Preis  $K$  zu kaufen, wenn gilt:

- *down-and-out*:  $S_t > H$  für alle  $0 \leq t \leq T$
- *up-and-out*:  $S_t < H$  für alle  $0 \leq t \leq T$

Der Fall von *Knock-out Option* resultiert in einer Null-Auszahlung, wenn der Preis  $S_t$  die Barriere erreichen. Figure 2<sup>2</sup> zeigt die Situation von *down-and-out* Option. Wenn der Preis die Barriere erreicht, die Option wird wertlos sein, obwohl ihr Preis in der Zukunft weiter steigt.

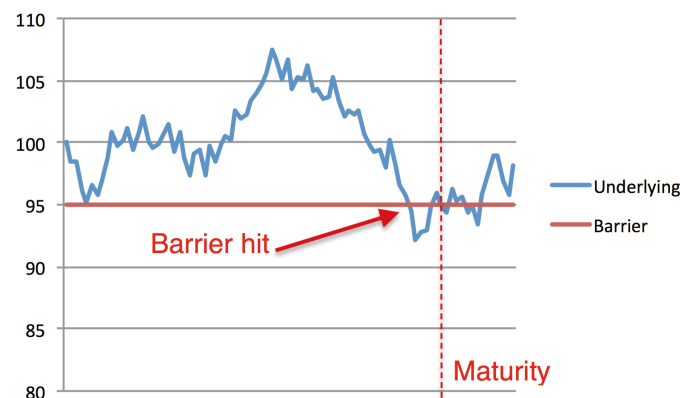


Figure 2: Situation von down-and-out Option

<sup>2</sup><https://www.quantstart.com/articles/Monte-Carlo-Simulations-In-CUDA-Barrier-Option-Pricing/>

Angenommen eine *down-and-out* Call-Option mit  $K > B$ . Solange  $S_t > H$  gilt, dann erfüllt  $V(S, t)$  die Black-Scholes Differentialgleichung mit Begrenzung:

$$V(S, T) = \max(S_T - K, 0)$$

Weiterhin gilt, wenn der Basiswert  $S_t$  die Barriere  $H$  erreicht, dann:

$$V(B, t) = 0, 0 \leq t \leq T$$

Die geschlossene Formel für den Preis ist:

$$V(S, t) = C(S, t) - \left(\frac{H}{S}\right)^\alpha C\left(\frac{H^2}{S}, t\right)$$

mit  $\alpha = \frac{2r}{\sigma^2} - 1$ .

## 3 Implementierung in Python

### 3.1 Barriere Option Preisbewertung

In diesem Kapitel, werden wir die vorhergezeigten Formeln zur Preisbewertung von Barriere Option sowie ihre Sensitivitäten in Python implementieren. Das Paket *numpy* und *py\_vollib* werden für die Implementierung notwendig sein. Das Binomialbaum wird als ein Netzwerk mit Knoten  $(i, j)$  betrachtet, mit  $i$  als einen Zeitschritt und  $j$  als die Outputnummer von gefragtem Preis.

$$S_{ij} = S_0 u^j d^{i-j}$$

$C_{ij}$  stellt den Preis für jede Knoten  $(i, j)$  dar, wo  $C_{Nj}$  den Payoff am Ende bezeichnet. Für *up-and-out* Put Barriere Option:



wenn  $T = t_N$ , dann gilt  $C_N^j = (K - S_N^j)^+ \mathbb{1}(S_N^j < H)$

für die restliche Knoten  $(i, j)$

- $t_n \in T$  und  $S_i^j \geq H \Rightarrow C_i^j = 0$
- $t_n \notin T$  oder  $S_i^j < H \Rightarrow C_i^j = \exp^{-r\Delta T} [q_i^j C_{i+1}^{j+1} + (1 - q_i^j) C_{i+1}^{j-1}]$

Jetzt definieren wir die Parameter für die Funktion, nämlich:

- $S_0$  Basiswert
- $K$  Strike-Preis
- $T$  Laufzeit oder Maturität in Jahre
- $H$  Barrierpreis
- $r$  Zinssatz
- $N$  Anzahl des Zeitschrittes
- $u$  Up-Faktor
- $opt$  Optiontyp Call oder Put

Dann wir werden *for-loops* zur Iteration der Knoten  $j$  in jedem Zeitschritt  $i$ .

```
import numpy as np
def barrieropt(S0,K,T,H,r,N,u,opt):
    #Konstante definieren
    dt = T/N # Länge
    d = 1/u # down-faktor
    q = (np.exp(r * dt) - d)/(u - d) # Erfolgswahrscheinlichkeit
    disk = np.exp(-r * dt) # Diskontierung

    #Wert von Underlyings am Maturität
    S = np.zeros(N+1)
    for j in range(0, N+1):
        S[j] = S0 * u**j * d**(N - j)
```

```

#Option payoff
C = np.zeros(N+1)
for j in range(0, N+1):
    if opt == 'C':
        C[j] = max(0, S[j] - K)
    elif opt == 'P':
        C[j] = max(0, K - S[j])

#Barrier prüfen
for j in range(0, N+1):
    S = S0 * u**j * d**(N - j)
    if S >= H:
        C[j] = 0

#Rückwärts durch den Binomialbaum
for i in np.arange(N-1, -1, -1):
    for j in range(0, i+1):
        S = S0 * u**j * d**(i - j)
        if S >= H:
            C[j] = 0
        else:
            C[j] = disk * (q * C[j+1] + (1 - q) * C[j])
return C[0]

```

Hier setzen wir erstmal  $S$  und  $C$  als Nullvektoren, die später durch *for-loops* ausgefüllt werden. Danach wir können die Barriere prüfen ob sie schon erreicht wurde. Das Vorgehen ist, wir berechnen  $S$  und dann setezen wir eine Bedingung, wenn der Preis größer als Barriere  $H$ , dann wird der Preis  $C[j] = 0$ . Folgende Code zeigen die Ergebnisse von verschiedenen Parametern.

```

# S0 = 100  Basiswert
# K = 100   Strike-Preis
# T = 1     Maturität im Jahr (betrachtende Laufzeit)
# H = 125   up-and-out Barrierpreis
# r = 0.06  Zins
# N = 3     Anzahl von Schritt
# u = 1.1   up-faktor
barrieropt(100,100,1,125,0.06,3,1.1,opt='C') # Call-Option

```

4.00026736854323

```

# S0 = 200  Basiswert
# K = 250   Strike-Preis
# T = 1     Maturität im Jahr (betrachtende Laufzeit)
# H = 80    down-and-out Barrierpreis
# r = 0.06  Zins
# N = 3     Anzahl von Schritt
# u = 1.1   up-faktor
barrieropt(200,250,1,80,0.06,3,1.1,opt='C') # Call-Option

```

0.0

```

# S0 = 100  Basiswert
# K = 100   Strike-Preis
# T = 1     Maturität im Jahr (betrachtende Laufzeit)
# H = 125   up-and-out Barrierpreis
# r = 0.06  Zins
# N = 3     Anzahl von Schritt
# u = 1.1   up-faktor
barrieropt(100,100,1,125,0.06,3,1.1,opt='P') # Put-Option

```

4.322189158353709

```

# S0 = 200  Basiswert
# K = 250   Strike-Preis
# T = 1     Maturität im Jahr (betrachtende Laufzeit)
# H = 80    down-and-out Barrierpreis
# r = 0.06  Zins
# N = 3     Anzahl von Schritt
# u = 1.1   up-faktor
barrieropt(200,250,1,80,0.06,3,1.1,opt='P')  # Put-Option

0.0

```

## 3.2 Sensitivität

In diesem Teil, wir werden die Sensitivitäten oder Greeks, die wir selbst berechnen, mit die Module von *py\_vollib*. Zunächst implementieren wir die Black-Scholes Formel in Python. Für die benötigte Variablen, benutzen wir dieselbe Parameter wie oben.

```

from scipy.stats import norm
from py_vollib.black_scholes import black_scholes as bs
from py_vollib.black_scholes.greeks.analytical import delta, gamma,
vega, theta, rho

# Variablen definieren
r = 0.06 # Zins
S = 100  # Optionswert
K = 100  # Strike-Preis
T = 240/365 # Laufzeit
sigma = 0.30 # Volatilität

def blackScholes(r, S, K, T, sigma, type="c"):
    # Black-Scholes Preis berechnen
    d1 = (np.log(S/K) + (r + sigma**2/2)*T)/(sigma*np.sqrt(T))

```

```

d2 = d1 - sigma*np.sqrt(T)
try:
    if type == "c":
        preis = S*norm.cdf(d1, 0, 1) - K*np.exp(-r*T)*norm.cdf(d2, 0, 1)
    elif type == "p":
        preis = K*np.exp(-r*T)*norm.cdf(-d2, 0, 1) - S*norm.cdf(-d1, 0, 1)
    return preis, bs(type, S, K, T, r, sigma)
except:
    print("Bitte geben Sie 'type' ein")

```

Jetzt wir können die Greeks definieren, damit wir am Ende die Werte vergleichen können.

### 3.3 Delta ( $\Delta$ )

```

def delta_calc(r, S, K, T, sigma, type="c"):
    # Delta berechnen
    d1 = (np.log(S/K) + (r + sigma**2/2)*T)/(sigma*np.sqrt(T))
    try:
        if type == "c":
            delta_calc = norm.cdf(d1, 0, 1)
        elif type == "p":
            delta_calc = -norm.cdf(-d1, 0, 1)
        return delta_calc, delta(type, S, K, T, r, sigma)
    except:
        print("Bitte geben Sie 'type' ein")

```

### 3.4 Gamma ( $\Gamma$ )

```

def gamma_calc(r, S, K, T, sigma, type="c"):
    # Gamma berechnen

```

```

d1 = (np.log(S/K) + (r + sigma**2/2)*T)/(sigma*np.sqrt(T))
d2 = d1 - sigma*np.sqrt(T)
try:
    gamma_calc = norm.pdf(d1, 0, 1)/(S*sigma*np.sqrt(T))
    return gamma_calc, gamma(type, S, K, T, r, sigma)
except:
    print("Bitte geben Sie 'type' ein")

```

### 3.5 Vega ( $\nu$ )

```

def vega_calc(r, S, K, T, sigma, type="c"):
    # Vega berechnen
    d1 = (np.log(S/K) + (r + sigma**2/2)*T)/(sigma*np.sqrt(T))
    d2 = d1 - sigma*np.sqrt(T)
    try:
        vega_calc = S*norm.pdf(d1, 0, 1)*np.sqrt(T)
        return vega_calc*0.01, vega(type, S, K, T, r, sigma)
    except:
        print("Bitte geben Sie 'type' ein")

```

### 3.6 Theta ( $\Theta$ )

```

def theta_calc(r, S, K, T, sigma, type="c"):
    # Theta berechnen
    d1 = (np.log(S/K) + (r + sigma**2/2)*T)/(sigma*np.sqrt(T))
    d2 = d1 - sigma*np.sqrt(T)
    try:
        if type == "c":
            theta_calc = -S*norm.pdf(d1, 0, 1)*sigma/(2*np.sqrt(T)) - r*K*np.exp(-
        elif type == "p":

```

```

        theta_calc = -S*norm.pdf(d1, 0, 1)*sigma/(2*np.sqrt(T)) + r*K*np.exp(-
    return theta_calc/365, theta(type, S, K, T, r, sigma)
except:
    print("Bitte geben Sie 'type' ein")

```

### 3.7 Rho ( $\rho$ )

```

def rho_calc(r, S, K, T, sigma, type="c"):
    # Rho berechnen
    d1 = (np.log(S/K) + (r + sigma**2/2)*T)/(sigma*np.sqrt(T))
    d2 = d1 - sigma*np.sqrt(T)
    try:
        if type == "c":
            rho_calc = K*T*np.exp(-r*T)*norm.cdf(d2, 0, 1)
        elif type == "p":
            rho_calc = -K*T*np.exp(-r*T)*norm.cdf(-d2, 0, 1)
        return rho_calc*0.01, rho(type, S, K, T, r, sigma)
    except:
        print("Bitte geben Sie 'type' ein")

```

```

option_type='p'

print("Option Preis: ", [round(x,3) for x in blackScholes(r, S, K, T,
                                                         sigma, option_type)])
print("      Delta: ", [round(x,3) for x in delta_calc(r, S, K, T,
                                                         sigma, option_type)])
print("      Gamma: ", [round(x,3) for x in gamma_calc(r, S, K, T,
                                                         sigma, option_type)])
print("      Vega : ", [round(x,3) for x in vega_calc(r, S, K, T,
                                                         sigma, option_type)])
print("      Theta: ", [round(x,3) for x in theta_calc(r, S, K, T,
                                                         sigma, option_type)])
print("      Rho  : ", [round(x,3) for x in rho_calc(r, S, K, T,
                                                         sigma, option_type)])

```

```

Option Preis:  [7.684, 7.684]
      Delta:  [-0.388, -0.388]
      Gamma:  [0.016, 0.016]
      Vega :  [0.311, 0.311]
      Theta:  [-0.012, -0.012]
      Rho   :  [-0.306, -0.306]

```

```

option_type='c'

print("Option Preis: ", [round(x,3) for x in blackScholes(r, S, K, T,
                                                         sigma, option_type)])
print("      Delta: ", [round(x,3) for x in delta_calc(r, S, K, T,
                                                         sigma, option_type)])
print("      Gamma: ", [round(x,3) for x in gamma_calc(r, S, K, T,
                                                         sigma, option_type)])
print("      Vega : ", [round(x,3) for x in vega_calc(r, S, K, T,
                                                         sigma, option_type)])

```



```

sigma, option_type)])
print("      Theta: ", [round(x,3) for x in theta_calc(r, S, K, T,
sigma, option_type)])
print("      Rho  : ", [round(x,3) for x in rho_calc(r, S, K, T,
sigma, option_type)])

```

```

Option Preis: [11.552, 11.552]
Delta: [0.612, 0.612]
Gamma: [0.016, 0.016]
Vega : [0.311, 0.311]
Theta: [-0.028, -0.028]
Rho  : [0.326, 0.326]

```

Hier sehen wir dass die Werte von unserer Berechnung die gleiche Werte von Module haben.

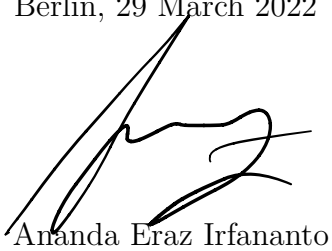
## 4 Quelle

- Jürgen Franke, Wolfgang Karl Härdle and Christian Matthias Hafner Statistics of Financial Markets – 5th ed. Springer, 2019.
- Jakub Stoklosa Studies of Barrier Options and their Sensitivities available on <http://www.ms.unimelb.edu.au>, 2007.

## Declaration of Authorship

We hereby confirm that we have authored this Seminar paper independently and without use of others than the indicated sources. All passages (and codes) which are literally or in general matter taken out of publications or other sources are marked as such.

Berlin, 29 March 2022

A handwritten signature in black ink, consisting of a series of loops and strokes, positioned above the printed name.

Ananda Eraz Irfananto