Greeks: option sensitivities, formula proofs and Python scripts Part A - 1^{st} order greeks

The Smile of Thales

April 11, 2015

Abstract

This documents is the first part of a general overview of vanilla options partial sensitivities (greeks). Here we provide 1^{st} generation greeks, their formula, mathematical proof, and suggest an implementation in Python.

⋆⋆⋆

Keywords: Options, Greeks, Python, Black Scholes

Contents

List of Figures

1 Delta

1.1 Definition

Delta is the option's sensitivity to small changes in the underlying price.

1.2 Shape

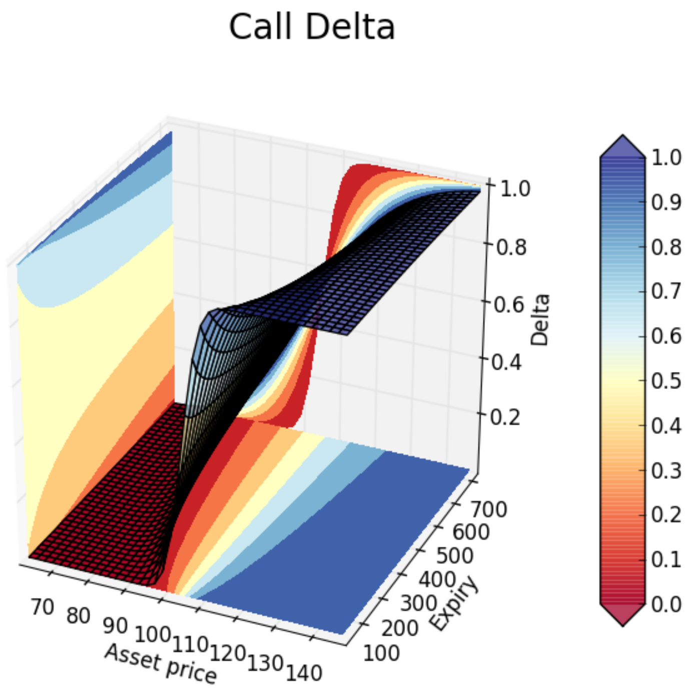

Figure 1: Delta

1.3 Formula

First let's remind the Black-Scholes-Merton formula for a vanilla Call option:

$$c = e^{-qT}SN(d_1) - Xe^{-rT}N(d_2) \tag{1.3.1}$$

$$p = Xe^{-rT}N(-d_2) - e^{-qT}SN(-d_1) \tag{1.3.2}$$

With:

$$d_1 = \frac{\ln(S/X) + (r - q + \frac{\sigma^2}{2})T}{\sigma\sqrt{T}} \tag{1.3.3}$$

And:

$$d_2 = d_1 - \sigma\sqrt{T} \tag{1.3.4}$$

The call option Delta will be:

$$\Delta_c = \frac{dc}{dS} = e^{-qT} N(d_1) \tag{1.3.5}$$

1.4 Proof

$$\begin{aligned}
\Delta_c = \frac{dc}{dS} = & \frac{d(e^{-qT} S N(d_1) - X e^{-rT} N(d_2))}{dS} \\
= & e^{-qT} N(d_1) + S e^{-qT} \frac{\partial N(d_1)}{\partial S} - X e^{-rT} \frac{\partial N(d_2)}{\partial S} \\
= & e^{-qT} N(d_1) + S e^{-qT} \frac{\partial d_1}{\partial S} \frac{\partial N(d_1)}{\partial d_1} - X e^{-rT} \frac{\partial d_2}{\partial S} \frac{\partial N(d_2)}{\partial d_2} \\
= & e^{-qT} N(d_1) + S e^{-qT} \frac{\partial d_1}{\partial S} N'(d_1) - X e^{-rT} \frac{\partial d_2}{\partial S} N'(d_2) \\
= & e^{-qT} N(d_1) + \underbrace{\frac{S e^{-qT} N'(d_1)}{S \sigma \sqrt{T}} - \frac{X e^{-rT} N'(d_2)}{S \sigma \sqrt{T}}}_{\text{I=0}}
\end{aligned} \tag{1.4.1}$$

Above, I=0. Indeed, according to (1.3.3) we have:

$$\begin{aligned}
& \ln(S/X) + (r - q + \frac{\sigma^2}{2})T = d_1 \sigma \sqrt{T} \\
& \Rightarrow \ln(S) - \ln(X) + (r-q)T = d_1 \sigma \sqrt{T} - \frac{\sigma^2}{2} T = \frac{1}{2}\left[d_1^2 - (d_1 - \sigma\sqrt{T})^2\right] \\
& \Rightarrow \ln(S) + \ln(\frac{1}{\sqrt{2\pi}}) - \frac{d_1^2}{2} = \ln(X) - (r-q)T + \ln(\frac{1}{\sqrt{2\pi}}) - \frac{d_2^2}{2} \\
& \Rightarrow S \frac{1}{\sqrt{2\pi}} e^{-\frac{d_1^2}{2}} = X e^{-(r-q)T} \frac{1}{\sqrt{2\pi}} e^{-\frac{d_2^2}{2}} \\
& \Rightarrow S N'(d_1) = X e^{-(r-q)T} N'(d_2) \\
& \Rightarrow S e^{-qT} N'(d_1) = X e^{-rT} N'(d_2)
\end{aligned} \tag{1.4.2}$$

Finally from (1.4.1) we obtain:

$$\boldsymbol{\Delta_c = e^{-qT} N(d_1)} \tag{1.4.3}$$

For the Put, since by parity we have:

$$\begin{aligned}
& p + S e^{-qT} = c + X e^{-rT} \\
& \Rightarrow \frac{dp}{dS} = \frac{dc}{dS} - e^{-qT} \\
& \Rightarrow \boldsymbol{\Delta_p = N(d_1) - 1}
\end{aligned} \tag{1.4.4}$$

1.5 Python script

```
import numpy as np
from math import sqrt, pi,log, e
from enum import Enum
import scipy.stats as stat
from scipy.stats import norm
import time

class BSMerton:
    def __init__(self, args):
        self.Type = int(args[0])                # 1 for a Call, - 1 for a put
        self.S = float(args[1])                 # Underlying asset price
        self.K = float(args[2])                 # Option strike K
        self.r = float(args[3])                 # Continuous risk fee rate
```

```
        self.q = float(args[4])                  # Dividend continuous rate
        self.T = float(args[5]) / 365.0          # Compute time to expiry
        self.sigma = float(args[6])              # Underlying volatility
        self.sigmaT = self.sigma * self.T ** 0.5# sigma*T for reusability
        self.d1 = (log(self.S / self.K) + \
                   (self.r - self.q + 0.5 * (self.sigma ** 2)) \
                   * self.T) / self.sigmaT
        self.d2 = self.d1 - self.sigmaT
        [self.Delta] = self.delta()

    def delta(self):
        dfq = e ** (-self.q * self.T)
        if self.Type == 1:
            return [dfq * norm.cdf(self.d1)]
        else:
            return [dfq * (norm.cdf(self.d1) - 1)]
```

Now here is a piece of code that you can use to calculate and chart the Delta surface displayed above (the python file that contains the Delta calculation above is called ”OptionsAnalytics.py”).

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import math
from matplotlib import cm
import OptionsAnalytics
from OptionsAnalytics import BSMerton

# Option parameters
sigma = 0.12        # Flat volatility
strike = 105.0      # Fixed strike
epsilon = 0.4       # The % on the left/right of Strike.
                    # Asset prices are centered around Spot ("ATM Spot")
shortexpiry = 30    # Shortest expiry in days
longexpiry = 720    # Longest expiry in days
riskfree = 0.00     # Continuous risk free rate
divrate = 0.00      # Continuous div rate

# Grid definition
dx, dy = 40, 40     # Steps throughout asset price and expiries axis

# xx: Asset price axis, yy: expiry axis, zz: greek axis
xx, yy = np.meshgrid(np.linspace(strike*(1-epsilon), (1+epsilon)*strike, dx), \
    np.linspace(shortexpiry, longexpiry, dy))
print "Calculating greeks ..."
zz = np.array([BSMerton([1,x,strike,riskfree,divrate,y,sigma]).Delta for
               x,y in zip(np.ravel(xx), np.ravel(yy))])
zz = zz.reshape(xx.shape)

# Plot greek surface
print "Plotting surface ..."
fig = plt.figure()
fig.suptitle('Call Delta',fontsize=20)
ax = fig.gca(projection='3d')
surf = ax.plot_surface(xx, yy, zz,rstride=1, cstride=1,alpha=0.75,cmap=cm.RdYlBu)
ax.set_xlabel('Asset price')
ax.set_ylabel('Expiry')
ax.set_zlabel('Delta')

# Plot 3D contour
zzlevels = np.linspace(zz.min(),zz.max(),num=8,endpoint=True)
xxlevels = np.linspace(xx.min(),xx.max(),num=8,endpoint=True)
yylevels = np.linspace(yy.min(),yy.max(),num=8,endpoint=True)
```

```
cset = ax.contourf(xx, yy, zz, zzlevels, zdir='z',offset=zz.min(),
                   cmap=cm.RdYlBu,linestyles='dashed')
cset = ax.contourf(xx, yy, zz, xxlevels, zdir='x',offset=xx.min(),
                   cmap=cm.RdYlBu,linestyles='dashed')
cset = ax.contourf(xx, yy, zz, yylevels, zdir='y',offset=yy.max(),
                   cmap=cm.RdYlBu,linestyles='dashed')

for c in cset.collections:
    c.set_dashes([(0, (2.0, 2.0))]) # Dash contours

plt.clabel(cset,fontsize=10, inline=1)

ax.set_xlim(xx.min(),xx.max())
ax.set_ylim(yy.min(),yy.max())
ax.set_zlim(zz.min(),zz.max())

#ax.relim()
#ax.autoscale_view(True,True,True)

# Colorbar
colbar = plt.colorbar(surf, shrink=1.0, extend='both', aspect = 10)
l,b,w,h = plt.gca().get_position().bounds
ll,bb,ww,hh = colbar.ax.get_position().bounds
colbar.ax.set_position([ll, b+0.1*h, ww, h*0.8])

# Show chart
plt.show()
```

2 Gamma

2.1 Definition

Gamma is the Delta's sensitivity to small changes in the underlying price.

2.2 Shape

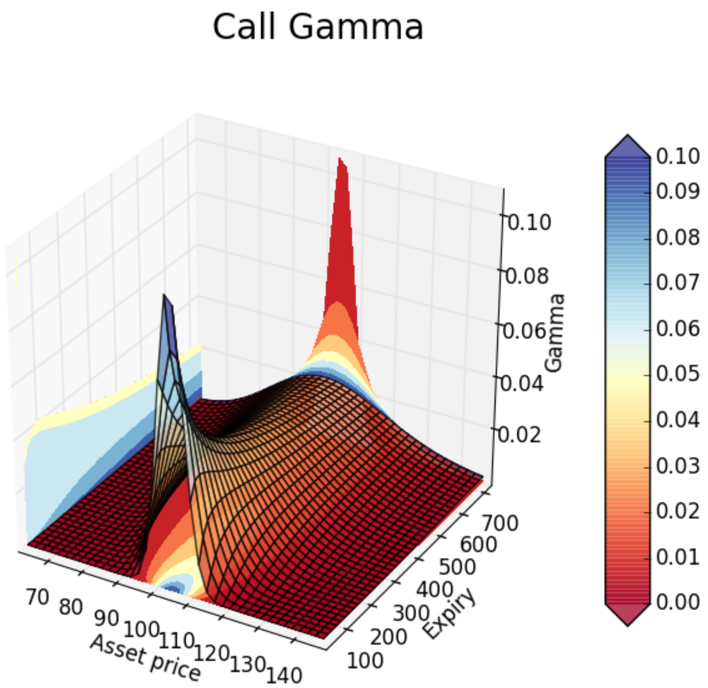

Figure 2: Gamma

2.3 Formula

The call/Put option Gamma will be:

$$\Gamma_c = \frac{\partial \Delta_c}{\partial S} = e^{-qT} \frac{N'(d_1)}{S\sigma\sqrt{T}} \tag{2.3.1}$$

2.4 Proof

$$\begin{aligned} \Gamma_c = \frac{\partial \Delta_c}{\partial S} &= e^{-qT} \frac{\partial d_1}{\partial S} \frac{\partial N(d_1)}{\partial d_1} \\ &= e^{-qT} \frac{N'(d_1)}{S\sigma\sqrt{T}} \end{aligned} \tag{2.4.1}$$

2.5 Python script

The following code simply adds the Gamma property to the BSMerton class.

```python
class BSMerton:
    def __init__(self, args):
        self.Type = int(args[0])                # 1 for a Call, - 1 for a put
        self.S = float(args[1])                 # Underlying asset price
        self.K = float(args[2])                 # Option strike K
        self.r = float(args[3])                 # Continuous risk fee rate
        self.q = float(args[4])                 # Dividend continuous rate
        self.T = float(args[5]) / 365.0         # Compute time to expiry
        self.sigma = float(args[6])             # Underlying volatility
        self.sigmaT = self.sigma * self.T ** 0.5# sigma*T for reusability
        self.d1 = (log(self.S / self.K) + \
                   (self.r - self.q + 0.5 * (self.sigma ** 2)) \
                   * self.T) / self.sigmaT
        self.d2 = self.d1 - self.sigmaT
        [self.Delta] = self.delta()
        [self.Gamma] = self.gamma()

    def gamma(self):
       return [e ** (-self.q * self.T) * norm.pdf(self.d1) / (self.S * self.sigmaT)]
```

3 Vega

3.1 Definition

Vega is the option's sensitivity to small changes in the underlying volatility.

3.2 Shape

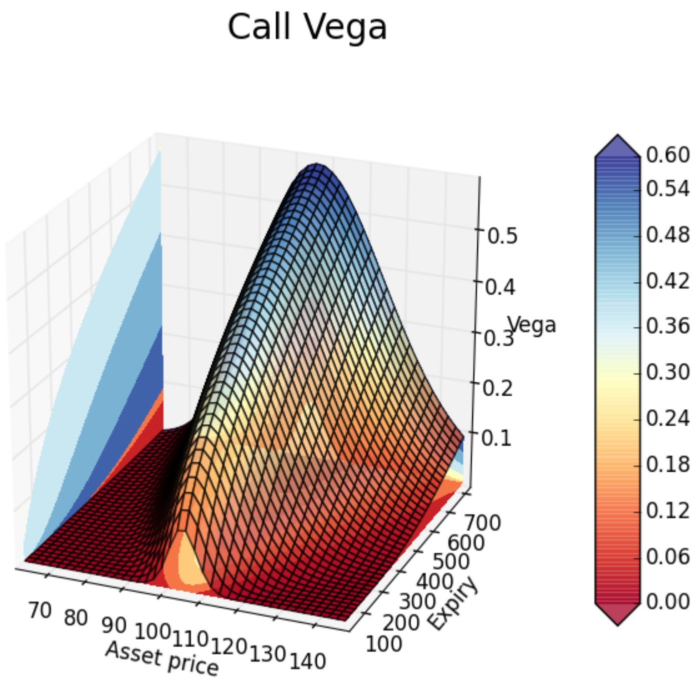

Figure 3: Vega

3.3 Formula

The call/Put option Vega will be:

$$\nu_c = \nu_p = \frac{\partial c}{\partial \sigma} = Se^{-qT}N'(d_1)\sqrt{T} \tag{3.3.1}$$

3.4 Proof

$$\begin{aligned}
\nu_c &= \frac{\partial c}{\partial \sigma} \\
&= \frac{\partial(e^{-qT}SN(d_1) - Xe^{-rT}N(d_2))}{\partial \sigma} \\
&= Se^{-qT}N'(d_1)\frac{\partial d_1}{\partial \sigma} - Xe^{-rT}N'(d_2)\frac{\partial d_2}{\partial \sigma} \\
&= Se^{-qT}N'(d_1)(\sqrt{T} - \frac{d_1}{\sigma}) - Xe^{-rT}N'(d_2)(\frac{\partial d_1}{\partial \sigma} - \sqrt{T}) \\
&= Se^{-qT}N'(d_1)(\sqrt{T} - \frac{d_1}{\sigma}) - Xe^{-rT}N'(d_2)(-\frac{d_1}{\sigma}) \\
&= \frac{-d_1}{\sigma}\big[\underbrace{Se^{-qT}N'(d_1) - Xe^{-rT}N'(d_2)}_{\mathrm{I}=0}\big] + Se^{-qT}N'(d_1)\sqrt{T} \\
&= Se^{-qT}N'(d_1)\sqrt{T}
\end{aligned} \tag{3.4.1}$$

3.5 Python script

The following code simply adds the Vega property to the BSMerton class.

```
class BSMerton:
    def __init__(self, args):
        self.Type = int(args[0])                # 1 for a Call, - 1 for a put
        self.S = float(args[1])                 # Underlying asset price
        self.K = float(args[2])                 # Option strike K
        self.r = float(args[3])                 # Continuous risk fee rate
        self.q = float(args[4])                 # Dividend continuous rate
        self.T = float(args[5]) / 365.0         # Compute time to expiry
        self.sigma = float(args[6])             # Underlying volatility
        self.sigmaT = self.sigma * self.T ** 0.5# sigma*T for reusability
        self.d1 = (log(self.S / self.K) + \
                   (self.r - self.q + 0.5 * (self.sigma ** 2)) \
                   * self.T) / self.sigmaT
        self.d2 = self.d1 - self.sigmaT
        [self.Vega] = self.vega()

    # Vega for 1% change in vol
    def vega(self):
      return [0.01 * self.S * e ** (-self.q * self.T) * \
          norm.pdf(self.d1) * self.T ** 0.5]
```

4 Theta

4.1 Definition

Theta is the option's sensitivity to small changes in time to expiry.

4.2 Shape

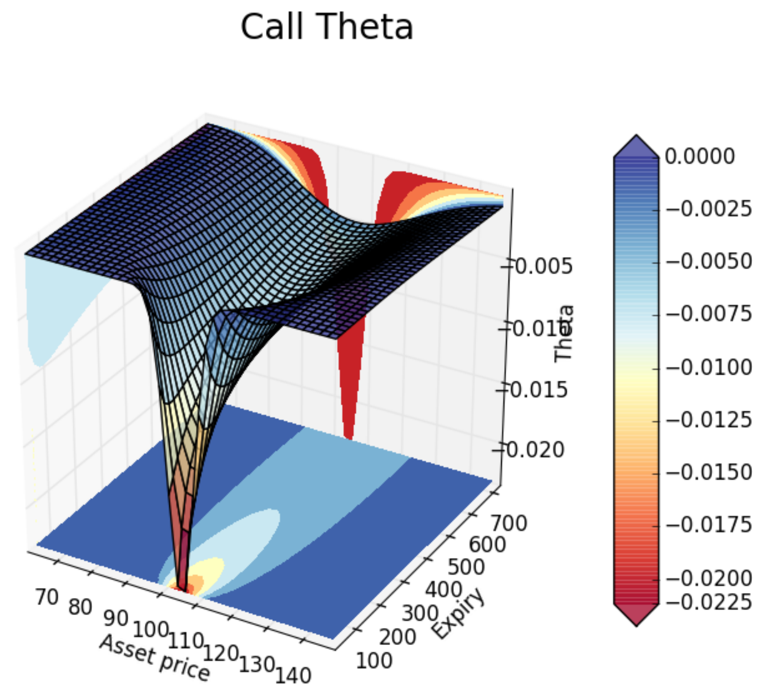

Figure 4: Theta

4.3 Formula

The call option Theta will be:

$$\Theta_c = \frac{\partial c}{\partial T} = \frac{\sigma S e^{-qT} N'(d_1)}{2\sqrt{T}} - qSe^{-qT}N(d_1) + rXe^{-rT}N(d_2) \tag{4.3.1}$$

And

$$\Theta_p = \frac{\partial c}{\partial T} = \frac{\sigma S e^{-qT} N'(d_1)}{2\sqrt{T}} + qSe^{-qT}N(-d_1) - rXe^{-rT}N(-d_2) \tag{4.3.2}$$

4.4 Proof

$$\begin{aligned}
\Theta_c &= \frac{\partial c}{\partial T} \\
&= \frac{\partial(e^{-qT}SN(d_1) - Xe^{-rT}N(d_2))}{\partial T} \\
&= -qSe^{-qT}N(d_1) + Se^{-qT}N'(d_1)\frac{\partial d_1}{\partial T} + rXe^{-rT}N(d_2) - Xe^{-rT}N'(d_2)\frac{\partial d_2}{\partial T} \\
&= rXe^{-rT}N(d_2) - qSe^{-qT}N(d_1) + Se^{-qT}N'(d_1)\frac{\partial d_1}{\partial T} - Xe^{-rT}N'(d_2)\frac{\partial(d_1 - \sigma\sqrt{T})}{\partial T} \\
&= rXe^{-rT}N(d_2) - qSe^{-qT}N(d_1) + \frac{\partial d_1}{\partial T}\underbrace{\left[Se^{-qT}N'(d_1) - Xe^{-rT}N'(d_2)\right]}_{=0, see (1.4.2)} + \frac{Xe^{-rT}N'(d_2)\sigma}{2\sqrt{T}} \\
&= rXe^{-rT}N(d_2) - qSe^{-qT}N(d_1) + \frac{Se^{-qT}N'(d_1)\sigma}{2\sqrt{T}}
\end{aligned} \tag{4.4.1}$$

4.5 Python script

The following code simply adds the Theta property to the BSMerton class.

```
class BSMerton:
    def __init__(self, args):
        self.Type = int(args[0])                # 1 for a Call, - 1 for a put
        self.S = float(args[1])                 # Underlying asset price
        self.K = float(args[2])                 # Option strike K
        self.r = float(args[3])                 # Continuous risk fee rate
        self.q = float(args[4])                 # Dividend continuous rate
        self.T = float(args[5]) / 365.0         # Compute time to expiry
        self.sigma = float(args[6])             # Underlying volatility
        self.sigmaT = self.sigma * self.T ** 0.5# sigma*T for reusability
        self.d1 = (log(self.S / self.K) + \
                   (self.r - self.q + 0.5 * (self.sigma ** 2)) \
                   * self.T) / self.sigmaT
        self.d2 = self.d1 - self.sigmaT
        [self.Theta] = self.theta()

    # Theta for 1 day change
    def theta(self):
        df = e ** -(self.r * self.T)
        dfq = e ** (-self.q * self.T)
        tmptheta = (1.0 / 365.0) \
            * (-0.5 * self.S * dfq * norm.pdf(self.d1) * \
               self.sigma / (self.T ** 0.5) + \
           self.Type * (self.q * self.S * dfq * norm.cdf(self.Type * self.d1) \
            - self.r * self.K * df * norm.cdf(self.Type * self.d2)))
        return [tmptheta]
```

5 Rho

5.1 Definition

Rho is the option's sensitivity to small changes in the risk-free interest rate.

5.2 Shape

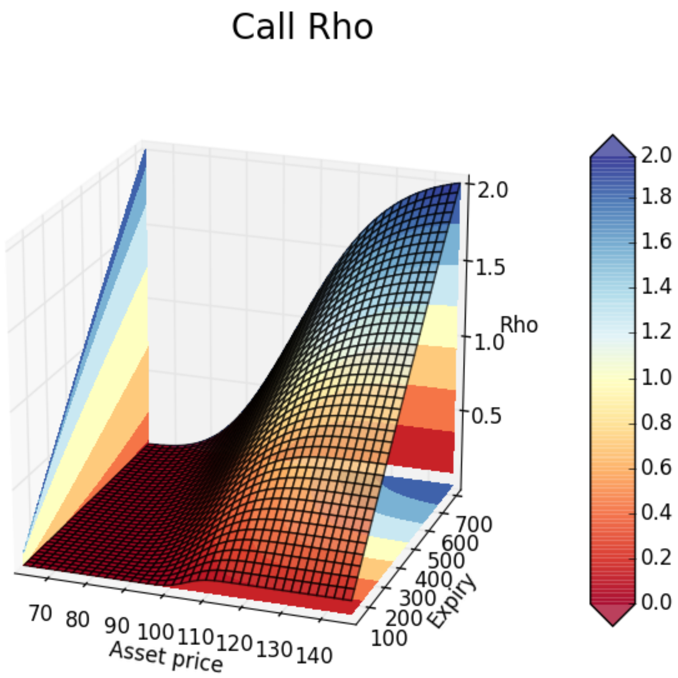

Figure 5: Rho

5.3 Formula

The call option Rho will be:

$$\rho_c = \frac{\partial c}{\partial r} = TXe^{-rT}N(d_2) \tag{5.3.1}$$

And

$$\rho_p = \frac{\partial p}{\partial r} = -TXe^{-rT}N(-d_2) \tag{5.3.2}$$

5.4 Proof

$$
\begin{aligned}
\rho_c &= \frac{\partial c}{\partial r} \\
&= \frac{\partial(e^{-qT}SN(d_1) - Xe^{-rT}N(d_2))}{\partial r} \\
&= Se^{-qT}N'(d_1)\frac{\partial d_1}{\partial r} + TXe^{-rT}N(d_2) - Xe^{-rT}N'(d_2)\frac{\partial d_2}{\partial r} \\
&= Se^{-qT}N'(d_1)\frac{\partial d_1}{\partial r} + TXe^{-rT}N(d_2) - Xe^{-rT}N'(d_2)\frac{\partial(d_1 - \sigma\sqrt{T})}{\partial r} \\
&= \frac{\partial d_1}{\partial r}\,[\underbrace{Se^{-qT}N'(d_1) - Xe^{-rT}N'(d_2)}_{=0, see (1.4.2)}] + Te^{-rT}N(d_2) \\
&= Te^{-rT}N(d_2)
\end{aligned}
\tag{5.4.1}
$$

5.5 Python script

The following code simply adds the Rho property to the BSMerton class.

```
class BSMerton:
    def __init__(self, args):
        self.Type = int(args[0])                # 1 for a Call, - 1 for a put
        self.S = float(args[1])                 # Underlying asset price
        self.K = float(args[2])                 # Option strike K
        self.r = float(args[3])                 # Continuous risk fee rate
        self.q = float(args[4])                 # Dividend continuous rate
        self.T = float(args[5]) / 365.0         # Compute time to expiry
        self.sigma = float(args[6])             # Underlying volatility
        self.sigmaT = self.sigma * self.T ** 0.5# sigma*T for reusability
        self.d1 = (log(self.S / self.K) + \
                   (self.r - self.q + 0.5 * (self.sigma ** 2)) \
                   * self.T) / self.sigmaT
        self.d2 = self.d1 - self.sigmaT
        [self.Rho] = self.rho()

    def rho(self):
       df = e ** -(self.r * self.T)
       return [self.Type * self.K * self.T * df * 0.01 * norm.cdf(self.Type * self.d2)]
```

6 Phi

6.1 Definition

Phi is the option's sensitivity to small changes in the dividend yield. In the chart below, $\Phi_(S, T)$ is calculated for a 1% change in the dividend yield.

6.2 Shape

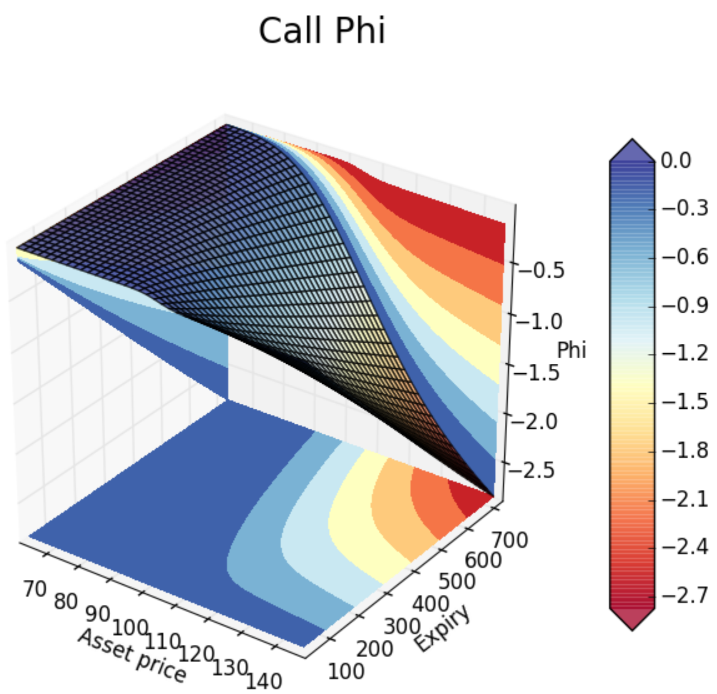

Figure 6: Phi

6.3 Formula

The call option Phi will be:

$$\Phi c = \frac{\partial c}{\partial q} = -TSe^{-qT}N(d_1) > 0 \tag{6.3.1}$$

And

$$\Phi p = \frac{\partial p}{\partial q} = TSe^{-qT}N(-d_1) < 0 \tag{6.3.2}$$

6.4 Proof

$$\begin{aligned}
\Phi c &= \frac{\partial c}{\partial q} \\
&= \frac{\partial(e^{-qT}SN(d_1) - Xe^{-rT}N(d_2))}{\partial q} \\
&= -TSe^{-qT}N(d_1) + Se^{-qT}N(d_1)\frac{\partial d_1}{\partial q} - Xe^{-rT}N'(d_2)\frac{\partial d_2}{\partial q} \\
&= \frac{\partial d_1}{\partial q}\big[\underbrace{Se^{-qT}N'(d_1) - Xe^{-rT}N'(d_2)}_{=0, see (1.4.2)}\big] - TSe^{-qT}N(d_1) \\
&= -TSe^{-qT}N(d_1)
\end{aligned} \tag{6.4.1}$$

6.5 Python script

The following code simply adds the Phi property to the BSMerton class.

```
class BSMerton:
    def __init__(self, args):
        self.Type = int(args[0])                # 1 for a Call, - 1 for a put
        self.S = float(args[1])                 # Underlying asset price
        self.K = float(args[2])                 # Option strike K
        self.r = float(args[3])                 # Continuous risk fee rate
        self.q = float(args[4])                 # Dividend continuous rate
        self.T = float(args[5]) / 365.0         # Compute time to expiry
        self.sigma = float(args[6])             # Underlying volatility
        self.sigmaT = self.sigma * self.T ** 0.5# sigma*T for reusability
        self.d1 = (log(self.S / self.K) + \
                   (self.r - self.q + 0.5 * (self.sigma ** 2)) \
                   * self.T) / self.sigmaT
        self.d2 = self.d1 - self.sigmaT
        [self.Phi] = self.phi()

    def phi(self):
       return [0.01* -self.Type * self.T * self.S * \
             e ** (-self.q * self.T) * norm.cdf(self.Type * self.d1)]
```

References

[1] Espen Gaarder Haug, *The Complete Guide to Option Pricing Formulas*. McGraw-Hill, 2nd Edition, 2007.