FIN 503: Advanced derivatives **(Due: 22/09/20)**

Assignment #2

Professor: Elena Perazzi *Assistant:* Marc-Aurèle Divernois

Students: Hien Lê, Francesco Maizza, Anita Mezzetti

Problem 1

Using the Black Scholes formula, we find that the value of a call option on the firm's equity is 75.3816. We also obtain the numerical values of the options for different strikes and maturities (See Table 1). The Black Scholes implied volatility is shown in Table 2.

Table 1: Compounded Option values

	2 Y	5 Y	7 Y	9Y
60%E	39.09216585	49.53140729	54.77978727	59.27764682
80%E	30.64982974	43.69253387	50.00889411	55.32253533
100%E	23.98228379	38.75141817	45.86230031	51.80461377
120%E	18.77786021	34.53438814	42.22356557	48.65239087

Table 2: Implied Volatility

	2 Y	5 Y	7 Y	9Y
60%E	0.52804688	0.55023438	0.56992188	0.5971875
80%E	0.51757813	0.53742188	0.5546875	0.57765625
100%E	0.50976563	0.5278125	0.54328125	0.563125
120%E	0.50359375	0.52015625	0.53421875	0.55179688

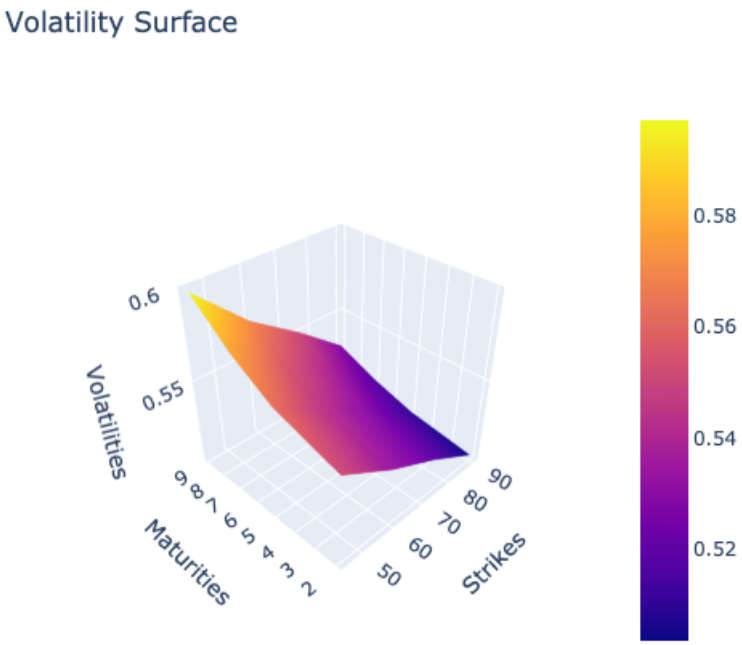


Figure 1: Volatility Surface

In Figure 1 we can see the Volatility Surface. We also plot the surface in another way (See Figure 2).

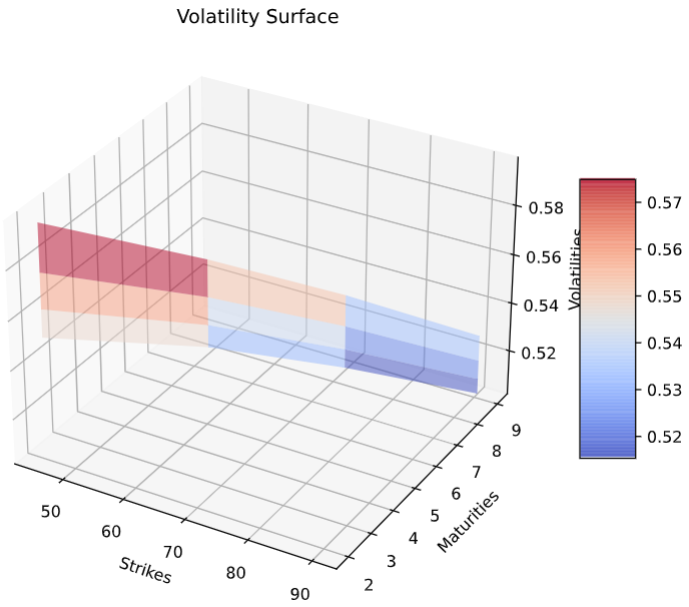


Figure 2: Volatility Surface

Problem 2

In this exercise, we were asked to compute the price of 4 x 4 call options with different maturities and strikes. The parameters are as follow:

$$\begin{aligned} S_0 &= 100 \\ \gamma &= -0.08 \\ \lambda &= 0.2 \\ \sigma &= 0.2 \\ r &= 0.04 \\ T(\text{maturities}) &\in \{0.02, 0.08, 0.25, 0.5\} \\ K(\text{strikes}) &\in \{0.8, 0.9, 1, 1.1\} \times S_0 \end{aligned} \tag{0.1}$$

Now, in order to compute the call price of a jump-diffusion model, we need to use the equation given on slide 26 of the lecture, which is:

$$C(S_0) = \sum_{j=0}^{\infty} P(N_T = j) BS_{call}(S_0(1-\gamma)^j, K, T, \sigma, r, q - \lambda^Q \gamma)$$

Note here that we can now replace $P(N_T = j)$ by the density function of the Poisson distribution (slide 23), truncate the summation by using a finite number of jumps in stead of infinity (in our case, 50), take $q = 0$ (no dividends), and since the γ that was given to us was already negative, we can flip the sign of the γ in our equation. In short, the final equation for computing the call price will be:

$$C(S_0) = \sum_{j=0}^{50} \exp(-\lambda T) \frac{(\lambda T)^j}{j!} BS_{call}(S_0(1+\gamma)^j, K, T, \sigma, r, \lambda^Q \gamma)$$

Note also that we did not write again the full formula of the Black-Scholes formula for call, but this can be found in the first lecture of the course (or in other courses such as Derivatives in the second semester); the last argument $\lambda^Q \gamma$ will be one that replaces the dividend δ in the original BS formula.

The implementation was carried out accordingly, as can be seen in the provided Python code. We subsequently found the following results for the prices of the call options (Table 3):

T \ K	0.8	0.9	1	1.1
0.02	2.00639746e+01	1.00732624e+01	1.18014528e+00	3.34841923e-04
0.08	2.02557972e+01	2.02557972e+01	2.44829663e+00	1.33599938e-01
0.25	2.08332976e+01	1.15176577e+01	4.54565505e+00	1.17510269e+00
0.5	4.54565505e+00	1.32103380e+01	1.32103380e+01	1.32103380e+01

Table 3: Prices of European Calls

Next, to compute the implied volatilities, we optimise the squared errors between the results found in Table 3 with the prices computed by the "traditional" Black-Scholes formula (with dividend set to 0), for each given maturity and strike. The results can be found in Table 4,

T \ K	0.8	0.9	1	1.1
0.02	0.31398438	0.25039063	020210938	0.20054688
0.08	0.22140625	0.20742188	0.20289063	0.2015625
0.25	0.20601563	0.20414063	0.20304688	0.20242188
0.5	0.20421875	0.20351563	0.20296875	0.20257813

Table 4: Implied Volatilities when calibrating with Jump-Diffusion Model

and illustrated in Figure 3.

As expected, in the case of European call options, the higher the strike, the lower the moneyness and the lower the implied volatility. One can also have a look at the implied volatility surface, which can be found in Figure 4.

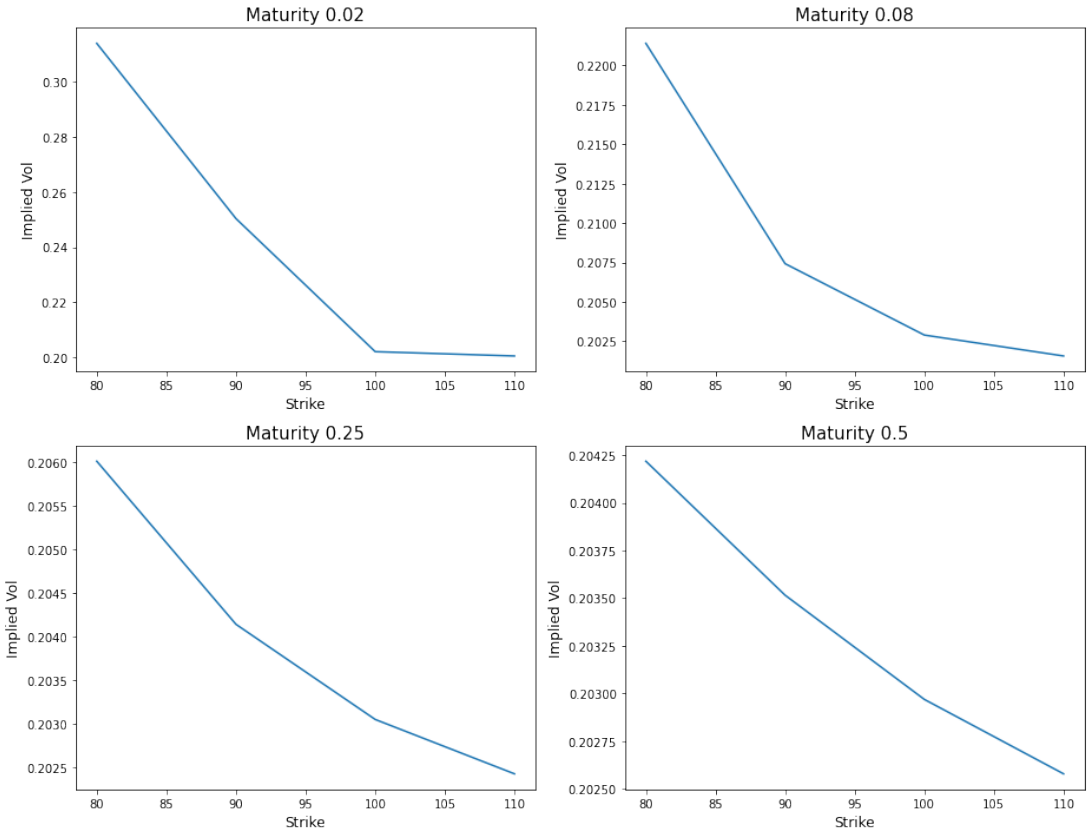


Figure 3: Implied Vols with Strikes

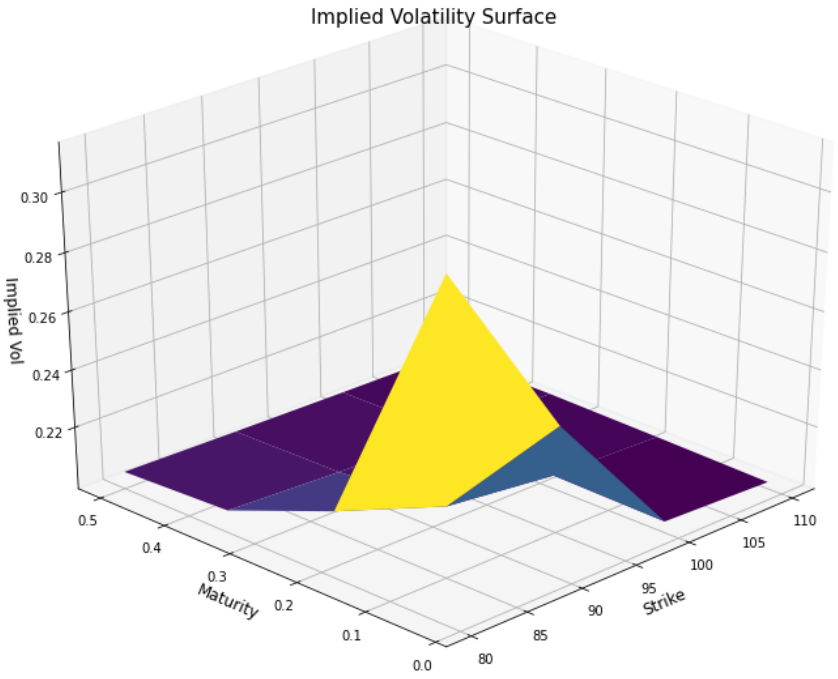


Figure 4: Implied Volatility Surface

Appendix: Python code

Exercise 1

Exercise 1

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy import stats
import scipy.optimize
from scipy.stats import multivariate_normal as mvn
from scipy.stats import norm
from numpy import matlib
import plotly.graph_objects as go
from matplotlib import cm
```

In [10]:
```
# constants:
T = 10
r = 0.05
sigma = 0.4
S = 100
D = 50
t = 0
```

Use the analytic formula for the compound option (call on call) to find the value of a call option:

In [11]:
```
def price_call(S, K, T, t, r, sigma):
    d1 = 1/(sigma*np.sqrt(T-t))*np.log(S*np.exp(r*(T-t))/K) + sig
ma*np.sqrt(T-t)/2
    d2 = 1/(sigma*np.sqrt(T-t))*np.log(S*np.exp(r*(T-t))/K) - sig
ma*np.sqrt(T-t)/2
    return S*stats.norm.cdf(d1) - np.exp(-r*(T-t))*K*stats.norm.c
df(d2)
```

In [12]:
```
C0 = price_call(S, D, T, 0, r, sigma) #current value of the firm'
s equity
C0
```

Out[12]:
```
75.38155847862096
```

Obtain the numerical value of the option for different strikes and maturities:

In [14]:
```
# strikes
k = [0.60, 0.80, 1, 1.2]
k = pd.Series(k)*C0

# maturities
tu = [2,5,7,9]
tau=pd.Series(tu)

s_star = np.zeros((4,4))
```

In [15]:
```
s_star = np.zeros((4,4))
#Find s* using a minimisation function
i=0
j=0
for i in range(len(k)):
    for j in range(len(tau)):
        error = lambda X: (price_call(X, D, T, tau[j], r, sigma)-
k[i])**2
        s_star[i,j] = scipy.optimize.fmin(func=error,x0=S)
```

```
Optimization terminated successfully.
         Current function value: 0.000000
         Iterations: 21
         Function evaluations: 42
Optimization terminated successfully.
         Current function value: 0.000000
         Iterations: 20
         Function evaluations: 40
Optimization terminated successfully.
         Current function value: 0.000000
         Iterations: 20
         Function evaluations: 40
Optimization terminated successfully.
         Current function value: 0.000000
         Iterations: 19
         Function evaluations: 38
Optimization terminated successfully.
         Current function value: 0.000000
         Iterations: 19
         Function evaluations: 38
Optimization terminated successfully.
         Current function value: 0.000000
         Iterations: 18
         Function evaluations: 36
Optimization terminated successfully.
         Current function value: 0.000000
         Iterations: 17
         Function evaluations: 34
Optimization terminated successfully.
         Current function value: 0.000000
         Iterations: 18
         Function evaluations: 36
Optimization terminated successfully.
```

```
         Current function value: 0.000000
         Iterations: 17
         Function evaluations: 34
Optimization terminated successfully.
         Current function value: 0.000000
         Iterations: 18
         Function evaluations: 36
Optimization terminated successfully.
         Current function value: 0.000000
         Iterations: 19
         Function evaluations: 38
Optimization terminated successfully.
         Current function value: 0.000000
         Iterations: 20
         Function evaluations: 40
Optimization terminated successfully.
         Current function value: 0.000000
         Iterations: 19
         Function evaluations: 38
Optimization terminated successfully.
         Current function value: 0.000000
         Iterations: 20
         Function evaluations: 40
Optimization terminated successfully.
         Current function value: 0.000000
         Iterations: 21
         Function evaluations: 42
Optimization terminated successfully.
         Current function value: 0.000000
         Iterations: 21
         Function evaluations: 42
```

In [16]:
```
s_star
```

Out[16]:
```
array([[ 71.07559204,  78.38737488,  84.766922  ,  92.25997925],
       [ 87.69226074,  94.96421814, 101.03637695, 107.65327454],
       [103.86741638, 111.05072021, 116.84333801, 122.85415649],
       [119.76531982, 126.84074402, 132.38945007, 137.98118591]])
```

In [17]:
```
tau = tau.to_numpy()
```

In [18]:
```
a_1 = (np.log(S/s_star) + (r+(sigma**2)/2)*(tau -t))/(sigma*np.sq
rt(tau-t))
a_2 = a_1 - sigma*np.sqrt(tau-t)
b_1 = (np.log(S/D) + (r+sigma**2/2)*(T - t))/(sigma*np.sqrt(T-t))
b_2 = b_1 - sigma*np.sqrt(T-t)
rho = np.sqrt((tau-t)/(T-t))

mu= [0, 0]
```

In [19]:
```
C = np.zeros((4,4)) # We compute the payoff for k = [0.60; 0.80; 1; 1.2]*payoff

i = 0
j = 0

for i in range(len(k)):
    for j in range(len(tau)):
        cov = [[1, rho[j]], [rho[j], 1]]
        x = [a_1[i,j],b_1]
        y = [a_2[i,j], b_2]
        C[i,j] = S * mvn.cdf(x, mean=mu, cov=cov) - D * np.exp(-r*(T-t)) * mvn.cdf(y, mean=mu, cov=cov)- np.exp(-r*(tau[j]-t)) * k[i] * norm.cdf(a_2[i,j])
```

In [20]:
```
C
```

Out[20]:
```
array([[39.09216585, 49.53140729, 54.77978727, 59.27764682],
       [30.64982974, 43.69253387, 50.00889411, 55.32253533],
       [23.98228379, 38.75141817, 45.86230031, 51.80461377],
       [18.77786021, 34.53438814, 42.22356557, 48.65239087]])
```

Derive the Black-Scholes implied volatility of the firm's equity corresponding to all these strikes and expirations:

In [14]:
```
IV = np.zeros((4,4))
i=0
j=0
for i in range(len(k)):
    for j in range(len(tau)):
        error2 = lambda X: (price_call(C0, k[i], tau[j], t, r, X)-C[i,j])**2
        IV[i,j]= scipy.optimize.fmin(func=error2,x0=0.4)
```

```
Optimization terminated successfully.
         Current function value: 0.000001
         Iterations: 13
         Function evaluations: 26
Optimization terminated successfully.
         Current function value: 0.000000
         Iterations: 13
         Function evaluations: 26
Optimization terminated successfully.
         Current function value: 0.000002
         Iterations: 13
         Function evaluations: 26
Optimization terminated successfully.
         Current function value: 0.000001
         Iterations: 14
         Function evaluations: 28
Optimization terminated successfully.
         Current function value: 0.000001
```

```
         Iterations: 12
         Function evaluations: 24
Optimization terminated successfully.
         Current function value: 0.000001
         Iterations: 13
         Function evaluations: 26
Optimization terminated successfully.
         Current function value: 0.000002
         Iterations: 13
         Function evaluations: 26
Optimization terminated successfully.
         Current function value: 0.000000
         Iterations: 13
         Function evaluations: 26
Optimization terminated successfully.
         Current function value: 0.000000
         Iterations: 12
         Function evaluations: 24
Optimization terminated successfully.
         Current function value: 0.000003
         Iterations: 13
         Function evaluations: 26
Optimization terminated successfully.
         Current function value: 0.000000
         Iterations: 13
         Function evaluations: 26
Optimization terminated successfully.
         Current function value: 0.000001
         Iterations: 13
         Function evaluations: 26
Optimization terminated successfully.
         Current function value: 0.000001
         Iterations: 12
         Function evaluations: 24
Optimization terminated successfully.
         Current function value: 0.000000
         Iterations: 13
         Function evaluations: 26
Optimization terminated successfully.
         Current function value: 0.000001
         Iterations: 13
         Function evaluations: 26
Optimization terminated successfully.
         Current function value: 0.000001
         Iterations: 13
         Function evaluations: 26
```

In [15]:
```
IV
```

Out[15]:
```
array([[0.52804688, 0.55023438, 0.56992188, 0.5971875 ],
       [0.51757813, 0.53742188, 0.5546875 , 0.57765625],
       [0.50976563, 0.5278125 , 0.54328125, 0.563125  ],
       [0.50359375, 0.52015625, 0.53421875, 0.55179688]])
```

Volatility Surface:

In [22]:
```
# the function go.Figure does not show anything if we run it on J
upyter Lab. However, everything is fine if we use Visual Studio C
ode

x = k[::-1]
y = tau[::-1]
fig = plt.figure()
fig = go.Figure(data=[go.Surface(z=IV, x=x, y=y)])
fig.update_layout(title='Volatility Surface',scene = dict(
                    xaxis_title='Strikes',
                    yaxis_title='Maturities',
                    zaxis_title='Volatilities'), autosize=False,
                  width=500, height=500,
                  margin=dict(l=65, r=50, b=65, t=90))
fig.update_scenes(xaxis_autorange="reversed",yaxis_autorange="rev
ersed")
fig.show()
```

```
<Figure size 432x288 with 0 Axes>
```

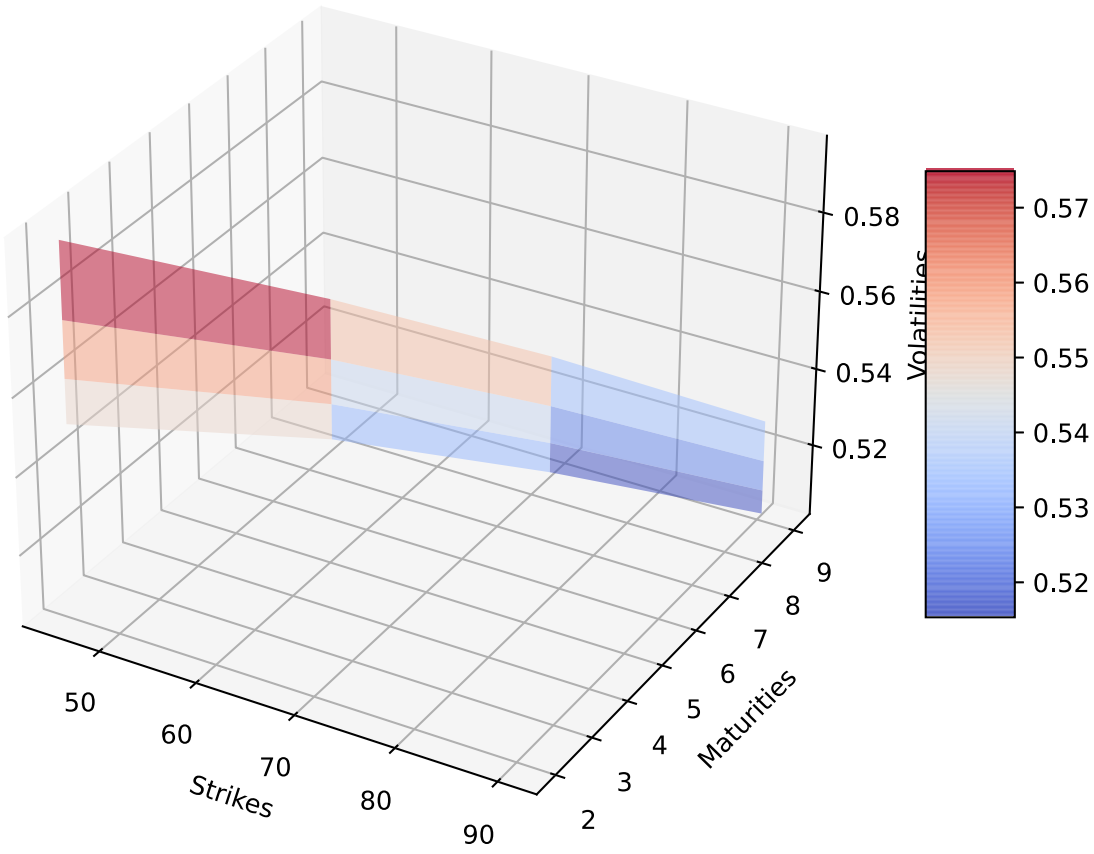
In [34]:
```
# in case we decide to use Jupyter lab and the previous cell does
not work, here there is another possible way to plot the volatlit
y surface (less fancy)
#label axes
fig = plt.figure(figsize=(8,8))
ax = fig.add_subplot(111, projection='3d')

ax.set_xlabel('Strikes')
ax.set_ylabel('Maturities')
ax.set_zlabel('Volatilities')

#plot figure
surf = ax.plot_surface(x,y,IV,linewidth=0, antialiased=False, sha
de = True, alpha = 0.5, cmap=cm.coolwarm)
fig.colorbar(surf, shrink=0.4, aspect=5)

plt.title("Volatility Surface")

plt.show()
```


Volatility Surface
0.58
0.56
0.54
0.52
50
60
70
80
90
Strikes
2
3
4
5
6
7
8
9
Maturities
Volatilities
0.57
0.56
0.55
0.54
0.53
0.52

In [ ]:

Exercise 2

In [15]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy import stats, optimize
```

Consider a non-dividend-paying stock with current value S = 100. In the risk-neutral measure the stock follows a jump-diffusion process

$$dS/S = (r - \lambda^Q \gamma)dt + \sigma dW_t + \gamma dN_t$$

Take γ = −0.08 (i.e. the stock price is subject to downward 8% jumps), λ^Q = 0.2 (i.e. on average the stock experiences a jump every 5 years), σ = 0.2, r = 0.04. Compute the value of a call option with expirations T = 0.02, 0.08, 0.25, 0.5 (i.e. approximately 1 week, 1 month, 3 months, 6 months). and strikes K/S = 0.8, 0.9, 1, 1.1. For every expiration, plot the Black-Scholes implied volatility of the option as a function of strike.

In [50]:

```
S0 = 100.
gamma = -.08
lam = .2
sigma = .2
r = .04

Ts = np.array([.02, .08, .25, .5]) # 1 week, 1 month, 3 month, 6 months
Ks = np.array([.8, .9, 1., 1.1])*S0
```

In [53]:

```
def bs_call(S, K, T, t, r, sigma, div):
    d1 = 1/(sigma*np.sqrt(T-t))*np.log(S*np.exp(r*(T-t))/K) + sigma*np.sqrt(T-t)/2
    d2 = 1/(sigma*np.sqrt(T-t))*np.log(S*np.exp(r*(T-t))/K) - sigma*np.sqrt(T-t)/2
    return S*np.exp(-div*(T-t))*stats.norm.cdf(d1) - K*np.exp(-r*(T-t))*stats.norm.cdf(d2)

def price_call(S, K, T, t, r, sigma, gamma, lam):

    # this function prices the call based on the jump-diffusion model in the lecture
    n_jumps = 50 # the summation in the formula slide 26
    price = 0
    for n in range(n_jumps):
        price += np.exp(-lam*T)*((lam*T)**n)/np.math.factorial(n)* \
        bs_call(S*(1+gamma)**n, K, T, t, r, sigma, lam*gamma)
    return price
```

In [54]:

```
prices = np.zeros((Ts.size, Ks.size))
imp_vols = np.zeros((Ts.size, Ks.size))
```

In [55]:

```
for i in range(Ts.size):
    for j in range(Ks.size):
        prices[i, j] = price_call(S0, Ks[j], Ts[i], 0, r, sigma, gamma, lam)
```

In [56]:

```
prices
```

Out[56]:

```
array([[2.00639746e+01, 1.00732624e+01, 1.18014528e+00, 3.34841923e-04],
       [2.02557972e+01, 1.03572479e+01, 2.44829663e+00, 1.33599938e-01],
       [2.08332976e+01, 1.15176577e+01, 4.54565505e+00, 1.17510269e+00],
       [2.18267557e+01, 1.32103380e+01, 6.70957152e+00, 2.82278465e+00]])
```

In [57]:

```
for i in range(Ts.size):
    for j in range(Ks.size):
        error = lambda sig: (bs_call(S0, Ks[j], Ts[i], 0, r, sig, 0) - prices[i,j])**2
        imp_vols[i,j]= optimize.fmin(func=error, x0=0.4)
```

```
Optimization terminated successfully.
         Current function value: 0.000000
         Iterations: 12
         Function evaluations: 24
Optimization terminated successfully.
         Current function value: 0.000000
         Iterations: 15
         Function evaluations: 30
Optimization terminated successfully.
         Current function value: 0.000000
         Iterations: 14
         Function evaluations: 28
Optimization terminated successfully.
         Current function value: 0.000000
         Iterations: 14
         Function evaluations: 28
Optimization terminated successfully.
         Current function value: 0.000000
         Iterations: 15
         Function evaluations: 30
Optimization terminated successfully.
         Current function value: 0.000000
         Iterations: 14
         Function evaluations: 28
Optimization terminated successfully.
         Current function value: 0.000000
         Iterations: 14
         Function evaluations: 28
Optimization terminated successfully.
         Current function value: 0.000000
         Iterations: 14
         Function evaluations: 28
Optimization terminated successfully.
         Current function value: 0.000000
         Iterations: 14
         Function evaluations: 28
Optimization terminated successfully.
         Current function value: 0.000000
         Iterations: 14
         Function evaluations: 28
Optimization terminated successfully.
         Current function value: 0.000000
         Iterations: 14
         Function evaluations: 28
Optimization terminated successfully.
         Current function value: 0.000000
         Iterations: 14
         Function evaluations: 28
Optimization terminated successfully.
         Current function value: 0.000000
         Iterations: 14
         Function evaluations: 28
Optimization terminated successfully.
         Current function value: 0.000000
         Iterations: 14
         Function evaluations: 28
Optimization terminated successfully.
         Current function value: 0.000000
         Iterations: 14
         Function evaluations: 28
Optimization terminated successfully.
         Current function value: 0.000001
         Iterations: 14
         Function evaluations: 28
```

In [58]:

```
imp_vols
```

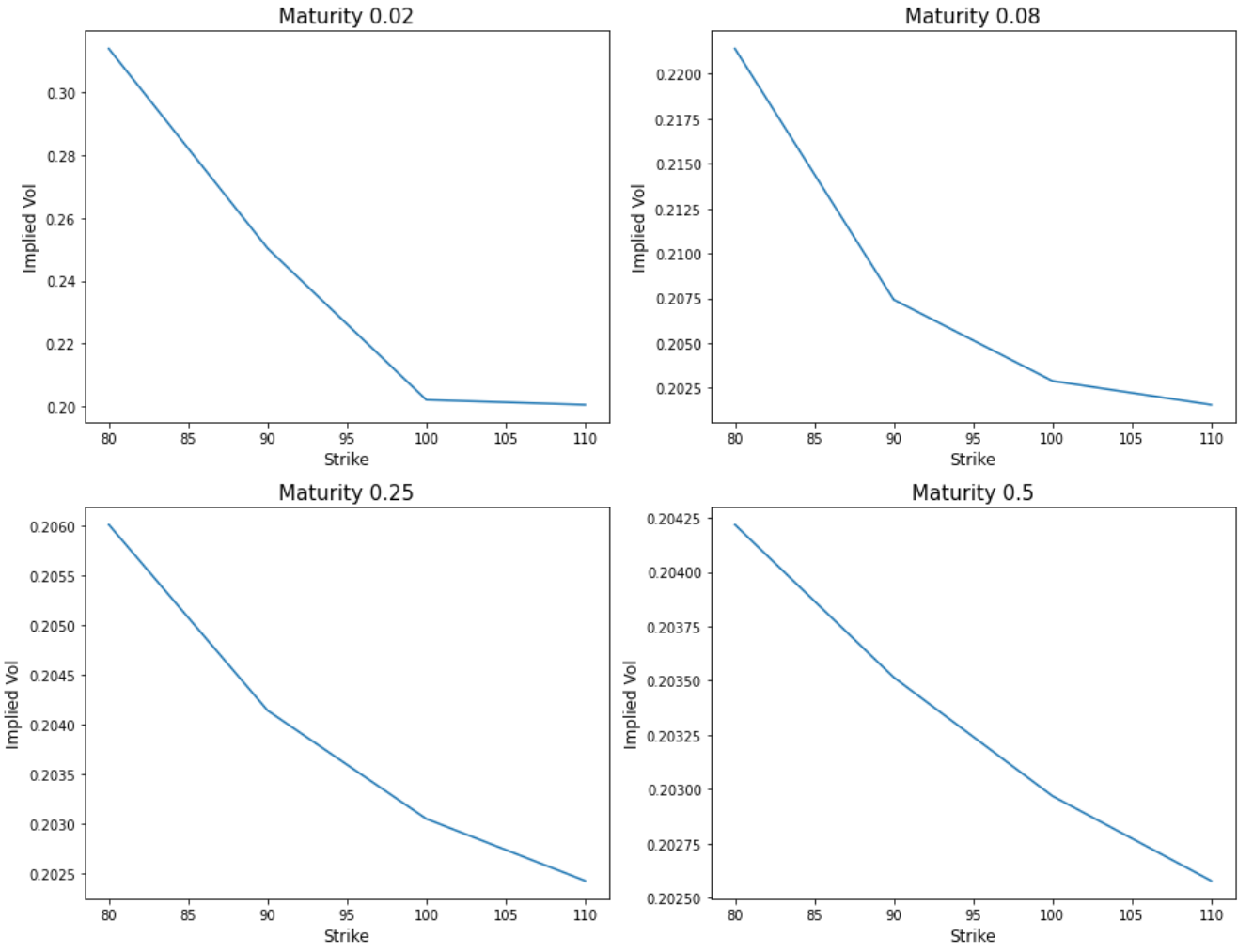
Out[58]:

```
array([[0.31398438, 0.25039063, 0.20210938, 0.20054688],
       [0.22140625, 0.20742188, 0.20289063, 0.2015625 ],
       [0.20601563, 0.20414063, 0.20304688, 0.20242188],
       [0.20421875, 0.20351563, 0.20296875, 0.20257813]])
```

In [66]:

```
fig, axes = plt.subplots(figsize=(13,10), nrows=2, ncols=2)
axes = axes.flatten()
for i in range(Ts.size):
    ax = axes[i]
    ax.plot(Ks, imp_vols[i,:])
    ax.set_title(f"Maturity {Ts[i]}", size=15)
    ax.set_xlabel("Strike", size=12)
    ax.set_ylabel("Implied Vol", size=12)
fig.tight_layout()
fig.show()
```

```
/home/hienle/.virtualenvs/notebook/lib/python3.6/site-packages/ipykernel_launcher.py:10:
UserWarning: Matplotlib is currently using module://ipykernel.pylab.backend_inline, which is a
non-GUI backend, so cannot show the figure.
  # Remove the CWD from sys.path while we load stuff.
```



In [101]:

```
# imp vol surface
from matplotlib import cm
```

In [123]:

```
plot_strikes = np.linspace(np.min(Ks), np.max(Ks), 4)
```

```
plot_ttm = np.linspace(np.min(Ts), np.max(Ts), 4)
X, Y = np.meshgrid(plot_strikes, plot_ttm)
```

In [125]:

```
fig = plt.figure(figsize=(13,10))
ax = fig.gca(projection='3d')
#ax.view_init(0, 45)
ax.plot_surface(X, Y, imp_vols, rstride=1, cstride=1, cmap=cm.viridis, linewidth=0.1)
ax.set_xlabel("Strike", size=12)
ax.set_ylabel("Maturity", size=12)
ax.set_zlabel("Implied Vol", size=12)
ax.view_init(30,225)
ax.set_title("Implied Volatility Surface", size=15)
plt.show()
```