Constructing Volatility Smile and Heston Model Calibration with Quantlib

Financial Algorithm- Midterm Project

台科大 M10808050 呂偉丞

CODE 說明

```
In [3]: day_count = ql.Actual365Fixed()
        calendar = ql.Taiwan() #using Taiwan Calender

        calculation_date = ql.Date(21, 11, 2018) #from 2018/11/21 to 2020/11/18

        spot = df.loc['2018-11-21']['Close']
        ql.Settings.instance().evaluationDate = calculation_date

        dividend_yield = ql.QuoteHandle(ql.SimpleQuote(0.0))
        risk_free_rate = 0.01
        dividend_rate = 0.0
        flat_ts = ql.YieldTermStructureHandle(
            ql.FlatForward(calculation_date, risk_free_rate, day_count)) #construct risk-free rate termstructure
        dividend_ts = ql.YieldTermStructureHandle(
            ql.FlatForward(calculation_date, dividend_rate, day_count)) #construct dividend rate termstructure
```

CODE 說明

Implied Volatility Surface

In [5]:
```
implied_vols = ql.Matrix(len(strikes), len(expiration_dates))
for i in range(implied_vols.rows()):
    for j in range(implied_vols.columns()):
        implied_vols[i][j] = data[j][i]
```

In [6]:
```
black_var_surface = ql.BlackVarianceSurface(
    calculation_date, calendar,
    expiration_dates, strikes,
    implied_vols, day_count)
```

In [7]:
```
strike = 9100.0
expiry = 1.0 #years
black_var_surface.blackVol(expiry, strike)
```

Out[7]:
```
0.34330230156325797
```

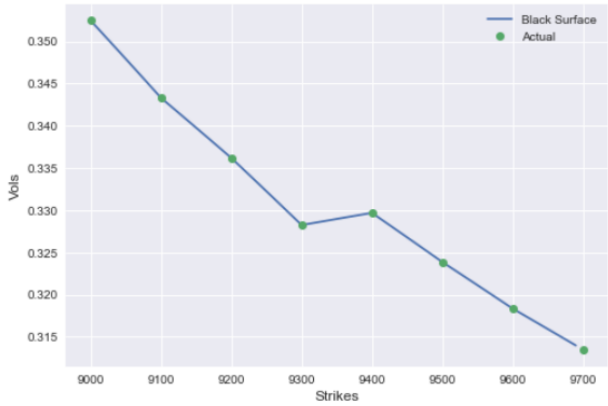
將所有data導入Quantlib Matrix

每個row代表不同的到期時間，每個column代表不同的strike price

最後只要輸入strike price與到期時間就可以得知implied volatility

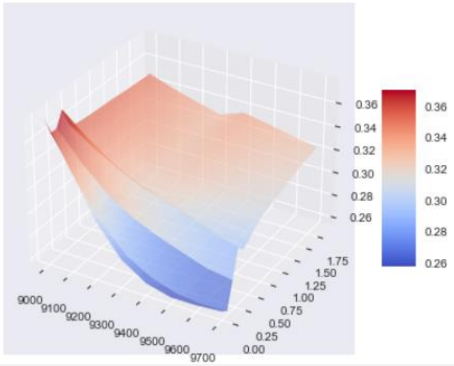
CODE 說明

In [8]:
```
strikes_grid = np.arange(strikes[0], strikes[-1],10)
expiry = 1.0 #years
implied_vols = [black_var_surface.blackVol(expiry, s)
                for s in strikes_grid]
actual_data = data[11]

fig, ax = plt.subplots()
ax.plot(strikes_grid, implied_vols, label="Black Surface")
ax.plot(strikes, actual_data, "o", label="Actual")
ax.set_xlabel("Strikes", size=12)
ax.set_ylabel("Vols", size=12)
legend = ax.legend(loc="upper right")
```



In [9]:
```
plot_years = np.arange(0, 2, 0.1)
plot_strikes = np.arange(9001, 9700, 1)
fig = plt.figure()
ax = fig.gca(projection='3d')
X, Y = np.meshgrid(plot_strikes, plot_years)
Z = np.array([black_var_surface.blackVol(float(y), float(x))
              for xr, yr in zip(X, Y)
                  for x, y in zip(xr,yr) ]
             ).reshape(len(X), len(X[0]))

surf = ax.plot_surface(X,Y,Z, rstride=1, cstride=1, cmap=cm.coolwarm,
                linewidth=0.1)
fig.colorbar(surf, shrink=0.5, aspect=5)
```

Out[9]: <matplotlib.colorbar.Colorbar at 0x18670d9c6d0>



透過Quantlib可以畫出2D/3D volatility smile

CODE 說明

```
In [11]: v0 = 0.01; kappa = 0.2; theta = 0.02; rho = -0.75; sigma = 0.5;

         process = ql.HestonProcess(flat_ts, dividend_ts,
                                    ql.QuoteHandle(ql.SimpleQuote(spot)),
                                    v0, kappa, theta, sigma, rho)
         model = ql.HestonModel(process)
         engine = ql.AnalyticHestonEngine(model)
```

```
In [12]: heston_helpers = []
         black_var_surface.setInterpolation("bicubic")
         one_year_idx = 11
         date = expiration_dates[one_year_idx]
         for j, s in enumerate(strikes):
             t = (date - calculation_date )
             p = ql.Period(t, ql.Days)
             sigma = data[one_year_idx][j]

             helper = ql.HestonModelHelper(p, calendar, spot, s,
                                           ql.QuoteHandle(ql.SimpleQuote(sigma)),
                                           flat_ts,
                                           dividend_ts)
             helper.setPricingEngine(engine)
             heston_helpers.append(helper)
```

透過Heston Model校準市場報價，假設我們想知道一年期的option，我們需要calibrate the Heston Model，在此之前我們還需要建構Pricing engine

CODE 說明

```
In [13]: lm = ql.LevenbergMarquardt(1e-8, 1e-8, 1e-8)
         model.calibrate(heston_helpers, lm,
                          ql.EndCriteria(500, 50, 1.0e-8,1.0e-8, 1.0e-8))
         theta, kappa, sigma, rho, v0 = model.params()

In [14]: print("theta = {}, kappa = {}, sigma = {}, rho = {}, v0 = {}".format(theta, kappa, sigma, rho, v0))

         theta = 0.1435086943220949, kappa = 2.149809280806962, sigma = 0.10000619553604377, rho = -0.9999995896777616, v0 = 0.030767452
         780416215
```

已經有建構完Heston Model & pricing engine後，選擇所有strike price和1年maturity，即可算出所需參數

CODE 說明

```
avg = 0.0

print ("%15s %15s %15s %20s" % (
    "Strikes", "Market Value",
     "Model Value", "Relative Error (%)"))
print("="*70)
for i, opt in enumerate(heston_helpers):
    err = (opt.modelValue()/opt.marketValue() - 1.0)
    print("%15.2f %14.5f %15.5f %20.7f " % (
        strikes[i], opt.marketValue(),
        opt.modelValue(),
        100.0*(opt.modelValue()/opt.marketValue() - 1.0)))
    avg += abs(err)
avg = avg*100.0/len(heston_helpers)
print("-" * 70)
print ("Average Abs Error (%%) : %5.3f" % (avg))
```

```
        Strikes    Market Value     Model Value   Relative Error (%)
======================================================================
        9000.00      1175.74319      1082.47993           -7.9322816
        9100.00      1185.55000      1129.51708           -4.7263224
        9200.00      1204.43866      1177.61387           -2.2271617
        9300.00      1220.10195      1226.76171            0.5458363
        9400.00      1278.08406      1276.95148           -0.0886156
        9500.00      1304.53643      1328.17351            1.8119144
        9600.00      1332.96946      1380.41761            3.5595825
        9700.00      1365.05515      1433.67313            5.0267558
----------------------------------------------------------------------
Average Abs Error (%) : 3.240
```

使用模型得到calibration後的option price 還有市場價值的誤差