Seminario Finanzas
Valoración Cap

Alumno
Manuel de la Llave*

Profesor
Carlos Catalán

5 de marzo de 2021

*manudela@ucm.es

Como en la práctica 1, los scripts de Python y los datos empleados se pueden encontrar en este repositorio

1 Valoración Cap

Voy a obtener el NPV (valor) de un Cap teniendo empleando un modelo Shifted Log-Normal con un shift de $\theta = 3\%$. La forma de cargar los datos en Python y el uso de los fixings es igual que en la práctica 1, lo que cambia en esta práctica (se puede ver el Cap como la pata flotante de un IRS, pero con opcionalidad) es que el valor del *forward* implícito lo calculo de la siguiente forma:

$$f(t) = \theta + (f(0) - \theta) exp\left(\sigma_{DD} W(t) - \frac{1}{2}\sigma_{DD}^2 t\right)$$

donde $W(t)$ es un Browniano y σ_{DD} es la volatilidad implícita del modelo Shift Log-normal que he obtenido interpolando linealmente las volatilidades proporcionadas en el fichero Excel *new_vols.xlsx*. En el código, simulo el browniano conjuntamente con el forward y hago uso de que $t = \Delta_i$, el número de días entre pago y pago dividido entre la convención (360 en este caso):

```
for j in range(0, aux2):
    index[j] = fixings_date[fixings_date == initial_date.values[j]].index[0
        ]
    forward[j] = fixings_rates[index[j]]
nsim = 1000000  # 1e6
for i in range(aux2 + 1, niter):
    forward[i] = np.mean(shift + (forward[aux2] - shift) * np.exp(
        volatility_strike[i] * np.random.normal(0, delta[i], nsim) - 0.5 *
        volatility_strike[i] ** 2 * delta[i]))
```

Y a continuación calculo el valor del Cap teniendo en cuenta que el modelo Shifted Log-Normal se puede expresar como el Black-76 pero ajustando las variables según el shift empleado:

$$C_{DD}(T, K, f(t)) = C_{B76}(T, K - \theta, f(t) - \theta, \sigma_{DD}(T, K - \theta))$$

de esta forma, podemos calcular el NPV de la siguiente manera:

```
d1 = (np.log((forward - shift) / (self.strike - shift)) + 0.5 *
    volatility_strike[2:] ** 2 *      (delta)) / (volatility_strike[2:] * np
    .sqrt(delta))
d2 = (np.log((forward - shift) / (self.strike - shift)) - 0.5 *
    volatility_strike[2:] ** 2 *      (delta)) / (volatility_strike[2:] * np
    .sqrt(delta))
N1_cdf = norm.cdf(d1, 0, 1)
N2_cdf = norm.cdf(d2, 0, 1)
caplets = ((forward - shift) * N1_cdf - (self.strike - shift) * N2_cdf) *
    self.notional * df_interp_libor[1:]
```

Cada caplet se obtiene como $c_i = DF_i \cdot N \cdot [(f_i - \theta)N(d_1^i) - (K - \theta)N(d_2^i)]$, donde DF_i es el factor de descuento empleando la curva del LIBOR a 3 meses, N es el nocional del contrato y tanto d_1 como d_2 han sido ajustados también con el shift. Este valor no es exacto, pues hemos empleado simulación para calcular el Browniano, aún así arroja consistentemente resultados de entorno a -15.5\$ millones para un strike del 1.1% fijado en el contrato. El valor del forward se mantiene por encima del strike a lo largo de todo el periodo, algo que sucedía de igual manera al calcular la pata flotante del IRS en la práctica 1, cuyo NPV era de unos -12\$ millones, por lo que este valor no nos ha de extrañar: esperaríamos que se parecieran, pues estamos valorando instrumentos similares de una forma muy parecida.

El Cap se valora desde el punto de vista del comprador, por lo que el valor negativo indica un pago negativo (beneficio positivo); es decir, como estamos vendiendo este producto, nosotros tendremos que pagar 15.5$ millones para cancelarlo.

2 Obtener la volatilidad implícita

En este apartado vamos a obtener la volatilidad implícita de este Cap pero empleando el modelo normal, es decir, usando el precio anterior podemos aplicar un *Solver* en el modelo normal para que nos de la volatilidad implícita según ese modelo, que es el siguiente:

$$C_N(T,K) = e^{-rT}\left[(f(t)-K)N(d) + \sigma_N\sqrt{t}N'(d)\right]$$

donde $f(t) = f(0) + \sigma_N W(t)$ y $d = \frac{f(t)-K}{\sigma_N\sqrt{t}}$

A pesar de todos mis esfuerzos por tratar de utilizar una función *Solver* en Python, no he conseguido hallar la volatilidad implícita de esta forma (se ve que no se puede emplear si la variable de interés está dentro de la función erf, que se encuentra en la función de distribución acumulada de la normal). Por lo que he optado por hacerlo "a mano" y aplicar el método Newton-Raphson, calculando la derivada (la Vega del Cap) de manera numérica. Así pues, primero calculo el valor del Cap de manera equivalente al del apartado anterior, para luego calcular el valor del Cap bajo el modelo normal, su vega y finalmente aplicar N-R, tal que así:

```
TOLABS = 1e-6
MAXITER = 100
h = 1e-6
for f in range(0, niter):
    caplets[f] = ((forward[f] - shift) * N1_cdf[f] - (self.strike - shift)
        * N2_cdf[f]) * self.notional * df_interp_libor[f]
    # Newton-Raphson Method
    imp_vol[f] = 0.005  # Initial guess
    dSigma = 10 * TOLABS  # Enter loop for the first time
    nIter = 0
    W = np.mean(np.random.normal(0, delta[aux2 + f], nsim))
    while nIter < MAXITER and np.abs(dSigma) > TOLABS:
        nIter = nIter + 1
        d = (forward[f] - self.strike) / (imp_vol[f] * np.sqrt(delta[aux2 +
            f]))
        N_cdf = norm.cdf(d, 0, 1)
        N_pdf = norm.pdf(d, 0, 1)
        normal_caplet = ((forward[f] - self.strike) * N_cdf + imp_vol[f] *
            np.sqrt(delta[aux2 + f]) * N_pdf) * df_interp_libor[f] * self.
            notional
        # Numerical derivative (vega)
        forward2 = shift + (forward[aux2 - 1] - shift) * np.exp((imp_vol[f]
            + h) * W - 0.5 *            (imp_vol[f] + h) ** 2 * delta[aux2
            + f])
        d2 = (forward2 - self.strike) / ((imp_vol[f] + h) * np.sqrt(delta[
            aux2 + f]))
        N_cdf2 = norm.cdf(d2, 0, 1)
        N_pdf2 = norm.pdf(d2, 0, 1)
        vega_caplet = (((forward2 - self.strike) * N_cdf2 + (imp_vol[f] + h
            ) * np.sqrt(
                    delta[aux2 + f]) * N_pdf2) * df_interp_libor[f] *
                        self.notional - normal_caplet) / h
        dSigma = (normal_caplet - caplets[f]) / vega_caplet
        imp_vol[f] = imp_vol[f] - dSigma
```

Para calcular la derivada numérica he empleado un aumento de $1e-6$, pues en torno a ese valor se consigue minimizar el error global, y en el caso de N-R, para una tolerancia al error absoluto he empleado también $1e-6$, aunque de esta forma no convergían los valores (el bucle llega al tope de iteraciones impuesto), el error seguía siendo del orden de $1e-5$. En este sentido, los valores de σ_N rondaban en torno al 1%, llegando a ser negativos y de un orden menos (e.g. -0.03%) en caso de que el método N-R convergiera. Podemos ver pues, que las volatilidades entre distintos modelos no son comparables, ya que en nuestro modelo inicial empleábamos volatilidades del orden del 20%