Valoración de opciones "Bermuda"

Diego Oleiarz, 14/11/22

Introducción

Las opciones bermudas se caracterizan por ser un punto medio entre las opciones europeas (las cuales solo pueden ser ejercidas al vencimiento de las mismas) y las opciones americanas (que pueden ser ejercidas en cualquier momento del plazo de vencimiento). En la práctica, las opciones bermudas son opciones americanas que solo pueden ser ejercidas en momentos determinados del plazo de vencimiento. Al ser una variación de las opciones americanas, las opciones bermudas tienen las mismas características que estas últimas, por lo que se puede decir que las opciones bermudas son opciones americanas con restricciones en el momento de ejercicio. Esto no significa que la forma de valorarlas sea la misma, ya que las opciones bermudas tienen un valor intrínseco distinto al de las opciones americanas, por lo que su valoración es distinta. Debido a esto, las opciones bermudas son más difíciles de valorar que las opciones americanas, ya que se debe tener en cuenta el valor intrínseco de la opción en cada momento de corte. En este informe se analizará la forma de valorar las opciones bermudas, utilizando una variación el modelo de Black-Scholes para la valoración de las opciones americanas.

Algoritmo

En sí el algorítmo conlleva pocos pasos:

1. Generar los valores del activo subyacente en cada momento de corte, a través de movimientos geométricos brownianos
2. Generar las cotas inferiores conocidas como valores de barrera (ejercicio mediante parametros de barrera) a partir de los datos obtenidos en el paso anterior
3. Descartar los valores del paso 1 y generar nuevos utilizando el mismo método, pero guardando las cotas inferiores obtenidas en el paso 2
4. Con los nuevos valores y las cotas inferiores, valorar la opción bermuda.

Análisis del Código

El código se encuentra en el archivo `bermuda.py` y se puede ejecutar con el comando `python bermuda.py`. Para poder ejecutar el código se debe tener instalado `numpy` (pip install numpy) y `pandas` (pip install pandas). El código se divide en 3 partes:

1. Generación de los valores del activo subyacente
2. Generación de las cotas inferiores
3. Valoración de la opción bermuda

Generación de los valores del activo subyacente

La generación de los valores del activo subyacente se realiza mediante el método `geo_brownian_motion` de la clase `Bermuda`. Este método recibe como parámetros:

- `self`: instancia de la clase `Bermuda`
- `delta_t`: paso de tiempo entre cada valor generado
- `s0`: valor inicial del activo subyacente

El método `brownian_motion` genera los valores del activo subyacente en cada momento de corte, a través de movimientos geométricos brownianos. Para esto, se utiliza la siguiente fórmula:

$$S_t = S_0 e^{(r - \frac{\sigma^2}{2})t + \sigma W_t}$$

Donde S_t es el valor del activo subyacente en el momento t, S_0 es el valor inicial del activo subyacente, r es la tasa de crecimiento del activo subyacente, σ es la volatilidad del activo subyacente, t es el plazo de vencimiento de la opción, W_t es el movimiento browniano en el momento t.

Generación de las cotas inferiores

La generación de las cotas inferiores se realiza mediante el método `gen_barrier` de la clase `Bermuda`. Este método recibe como parámetros:

- `self`: instancia de la clase `Bermuda`
- `actual_column`: listado de valores del activo subyacente en el momento de corte `t`
- `payoff_column`: listado de valores del activo subyacente en el momento de corte `t + 1`

El método `gen_barrier` genera las cotas inferiores conocidas como valores de barrera (enfoque de ejercicio mediante parametros de barrera) mediante los datos obtenidos en el paso anterior. Para esto, con cada valor del listado `actual_column` se compara individualmente (de ahora en mas `possible_value`) contra todos los valores del listado `payoff_column`.

```
posible_corte = []
for i in range(self.trayectorias):
```

Si `possible_value` es mayor que el valor de `actual_column[i]`, quiere decir que en ese momento se debería ejercer la opción utilizando el valor `actual_column[i]`, por lo que se guarda su payoff como un posible valor de corte en una lista.

```
if actual_column[i] <= possible_value:
    posible_corte.append(self.gen_payoff(actual_column[i]))
```

Si `possible_value` es menor que el valor de `actual_column[i]`, quiere decir que en ese momento no se debería ejercer la opción, por lo que se guarda el valor de `payoff_column[i]` deducido un período como un posible valor de corte en una lista.

```
    else:
        posible_corte.append(self.deduct_period(payoff_column[i]))
```

Al finalizar el cíclo, se obtiene el promedio de los valores guardados en la lista `posible_corte` y se guarda como posible valor de corte para el momento `t`.

```
    return sum(posible_corte) / len(posible_corte)
```

Se repite el proceso para todos los elementos de `actual_column`, al terminar se guarda el máximo `posible_corte` y se guarda el valor de `actual_column` correspondiente al mismo como el valor de corte para el momento `t`.

```
    average = [sum(elem) / len(possible_cuts) for elem in possible_cuts]
    max_average_elem_index = average.index(np.max(average))
    return actual_column[max_average_elem_index]
```

Valoración de la opción bermuda

La valoración de la opción bermuda se realiza mediante el método `valuate_bermuda_option` de la clase `Bermuda`. Este método recibe como parámetros:

- `self`: instancia de la clase `Bermuda`
- `s_1_s, s_2_s, s_3_s`: valores de corte al momento `t=1`, `t=2` y `t=3` respectivamente
- `n`: cantidad de pasos de tiempo a simular

El método `valuate_bermuda_option` utiliza las cotas inferiores obtenidas anteriormente y nuevos valores generados para el activo subyacente para obtener la prima para la opción bermuda. Para ello, pasa como argumentos la cota inferior `s_3_s`, la tercer columna de la tabla de datos generados (correspondiente al `t=3`) `table[3]` y `v_3` siendo los payoff correspondientes a `table[3]` a la función `get_cut_possible_values`. El resultado de esa función se guarda temporalmente en la variable `payoff_column_3`

```
    table = self.gen_table(n)

    v_3 = [self.gen_payoff(elem) for elem in table[3]]
    payoff_column_3 = self.get_cut_possible_values(s_3_s, table[3], v_3)
```

`payoff_column_3` es utilizado como argumento para la función `get_cut_possible_values` en lugar de una variable `v_2` (ya que los payoff ya están calculados en esa variable y fueron descontados al tiempo correspondiente de ser necesario), junto con la cota inferior `s_2_s`, la segunda columna de la tabla de datos generados (correspondiente al `t=2`) `table[2]` quedando guardado en la variable

`payoff_column_2`. Este proceso se repite para el momento `t=1` quedando guardado en la variable `payoff_column_1`.

```
payoff_column_2 = self.get_cut_possible_values(s_2_s, table[2],
                                               payoff_column_3)
payoff_column_1 = self.get_cut_possible_values(s_1_s, table[1],
                                               payoff_column_2)
```

Finalmente, se obtiene el valor de la opción bermuda como el maximo entre el promedio de los valores de `payoff_column_1` deducido un tiempo y el ejercicio temprano de la acción.

```
average = sum(payoff_column_1) / self.trayectorias
deducted_average = self.deduct_period(average)

early_exercise = max(self.K - self.s0, 0.0)
return max(early_exercise, deducted_average)
```

Resultados

A continuación, se demostrará paso a paso el funcionamiento del programa mediante la ejecución de un ejemplo con la siguiente tabla de datos (tomada de las páginas 606 a 609 del documento Hull - Options, Futures and Other Derivatives 7th Edition):

	t = 0	t = 1	t = 2	t = 3
0	1.0	1.09	1.08	1.34
1	1.0	1.16	1.26	1.54
2	1.0	1.22	1.07	1.03
3	1.0	0.93	0.97	0.92
4	1.0	1.11	1.56	1.52
5	1.0	0.76	0.77	0.90
6	1.0	0.92	0.84	1.01
7	1.0	0.88	1.22	1.34

Sabemos que s*(3)=1.1=K, por lo que partiremos calculando s*(2) y s*(1) para luego calcular el valor de la opción bermuda. Como es a modo de ejmplo, solo se calculará el valor de la opción bermuda para 3 de las trayectorias en la tabla por cada tiempo.

Para s*(2) partiremos desde el valor de la posición 5 en s(2) 0.77 ya que es menor a todos los elementos de s(2). Según el algoritmo, cuando el posible valor de corte es* menor* que el valor iterante de la columna actual, se debe tomar el payoff de la columna siguiente y descontarlo al tiempo correspondiente. Y cuando es `mayor o igual`, se debe tomar el payoff de la columna actual.

d(m(k-s(t)_i, 0)) = discount(max(k-s(t)_i, 0))

[d(m(1.1 - 1.34, 0)), d(m(1.1 - 1.54, 0)), d(m(1.1 - 1.03, 0)), d(m(1.1 - 0.92, 0)), d(m(1.1 - 1.52, 0)), max(1.1 - 0.77, 0), d(m(1.1 - 1.01, 0)), d(m(1.1 - 1.34, 0))]

[0.0, 0.0, 0.0659, 0.1695, 0.0, 0.33, 0.0848, 0.0]

Cuyo promedio es 0.0848.

Posición 3 en s(2) = 0.97

[d(m(1.1 - 1.34, 0)), d(m(1.1 - 1.54, 0)), d(m(1.1 - 1.03, 0)), max(1.1 - 0.97, 0), d(m(1.1 - 1.52, 0)), max(1.1 - 0.77, 0), max(1.1 - 0.84, 0), d(m(1.1 - 1.34, 0))]

[0.0, 0.0, 0.0659, 0.13, 0.0, 0.33, 0.26, 0.0]

Cuyo promedio es 0.0982

Posición 6 en s(2) = 0.84

[d(m(1.1 - 1.34, 0)), d(m(1.1 - 1.54, 0)), d(m(1.1 - 1.03, 0)), d(m(1.1 - 0.92, 0)), d(m(1.1 - 1.52, 0)), max(1.1 - 0.77, 0), max(1.1 - 0.84, 0), d(m(1.1 - 1.34, 0))]

[0.0, 0.0, 0.0659, 0.1695, 0.0, 0.33, 0.26, 0.0]

Cuyo promedio es 0.1032, que es el mayor de los promedios obtenidos para s(2), por lo que s*(2) = 0.84

Para calcular s*(1) utilizaremos los payoff obtenidos para s*(2) = 0.84 como columna de payoff, ya que es el mayor de los promedios obtenidos para s(2).

payoff_column = [0.0, 0.0, 0.0659, 0.1695, 0.0, 0.33, 0.26, 0.0]

Partiremos desde el valor de la posición 5 en s(1) 0.76 ya que es menor a todos los elementos de s(1).

[d(0), d(0), d(0.0659), d(0.1659), d(0), max(1.1 - 0.76, 0), d(1.1 - 0.84, 0), d(0)]

[0.0, 0.0, 0.0621, 0.1596, 0.0, 0.34, 0.2449, 0.0]

Cuyo promedio es 0.1008.

Posición 3 en s(1) = 0.93

[d(0), d(0), d(0.0659), max(1.1 - 0.93, 0), d(0), max(1.1 - 0.76, 0), max(1.1 - 0.92, 0), max(1.1 - 0.88, 0)]

[0.0, 0.0, 0.0621, 0.17, 0.0, 0.34, 0.18, 0.22]

Cuyo promedio es 0.1215

Posición 7 en s(1) = 0.88

[d(0), d(0), d(0.0659), d(0.1659), d(0), `max(1.1 - 0.76, 0)`, d(1.1 - 0.84, 0), `max(1.1 - 0.88, 0)`]

[0.0, 0.0, 0.0621, 0.1596, 0.0, 0.34, 0.2449, 0.22]

Cuyo promedio es 0.1283, que es el mayor de los promedios obtenidos para s(2), por lo que s*(1) = 0.88

Finalmente 0.1283 descontado un período es 0.1208, lo cuál será el valor de la opción bermuda.

Como veremos mas adelante, este resultado es simulado con éxito por el programa y se obtiene el mismo valor de la prima de la opción bermuda.

Siendo N y M cantidades de elementos de la tabla de datos generados para la obtención de las cotas inferiores y para la prima de la opción respectivamente, se obtienen los siguientes resultados para la opción bermuda:

N	M	Valor de la opción bermuda
0008	20000	0.1209
1000	20000	0.0101

Conclusiones

El programa desarrollado en Python es capaz de calcular el valor de la opción bermuda, para cualquier valor de K, T, r, sigma, S0, N y M.

Eso demuestra que, con esos simples datos, es posible obtener el valor de la opción bermuda, y que el método de Black-Scholes es capaz de calcular el valor de la opción bermuda.

Se puede concluir que las opciones Bermuda si bien no tienen una formula cerrada para el calculo de su prima, el mismo puede ser estimado y simulado con el método de Black-Scholes.

link al repositorio

https://github.com/Diegoolei/MMFC_bermuda-options-pricing