HW4: Introduction to Financial Engineering

Miguel Angel Aguilo Gonzalez, 1699413
Judit de Paz Ramírez, 1570590
Laia Mòdol Rodríguez, 1565282
Elena Rubio Zabala, 1699049
Guillem Tutusaus Alcaraz, 1533701

Noviembre 2023

Ejercicio 1

Vamos a programar un algoritmo de árbol binomial para calcular los precios de las opciones.

Como todos sabemos, las variables necesarias para construir un árbol son el precio de las acciones S, el movimiento ascendente permitido u y el movimiento descendente permitido v, dónde $0 < v < 1 < u$. También necesitamos la variable N que es el número de pasos de tiempo que se utilizarán.

Para crear una función que nos devuelva los posibles precios de las acciones para intervalos de tiempo sucesivos $t \in (0, N)$, cuando suministramos el precio inicial de estas, y el número de intervalos de tiempo que queremos que se calculen, empezamos creando una matriz. Ésta tiene dimensiones $(N+1) \times (N+1)$, dónde las filas representan la profundidad del árbol y las columnas representan los nodos (escenarios para el precio de las acciones). Inicializamos la matriz con ceros, que es donde se encuentra el precio inicial de las acciones S.

Utilizamos dos bucles anidados (`for`) para recorrer las filas y las columnas de la matriz, llenándola con los precios de las acciones. En cada posición $M[i, j]$, se calcula el precio de la acción utilizando la fórmula

$$S \cdot u^{(j-1)} \cdot v^{(i-j)}$$

y se almacena en esa posición.

La función que obtenemos es

```
build_stock_tree = function(S, u, v, N) {
  tree=matrix(0,nrow=N+1,ncol=N+1)
  for(i in 1:(N+1)){
    for(j in 1:i){
      tree[i,j] = S*u^(j-1)*v^(i-j)
    }
  }
  return(tree)
}
```

Finalmente, podemos observar que para cualquier precio de acción en el momento $t-1$, supongamos $M[t, j]$, nuestra matriz aplica una caída de precio v en la siguiente fila manteniendo la columna, que corresponde a la posición $M[t+1, j]$, mientras que los aumentos se colocan en la posición $M[t+1, j+1]$. Por lo tanto, hemos creado una matriz que se caracteriza por ser una triangular superior la cual tendrá $t+1$ elementos distintos de 0 por fila.

Calculemos ahora la probabilidad neutral al riesgo de un momento ascendente q. Para ello necesitamos los valores u, v y los tipos de interés r con el paso de tiempo dt (recordemos que la probabilidad de un movimiento a la baja será $(1-q)$). Recordemos también que mediante un argumento sin arbitraje tenemos

$$S(1 + rdt) = quS + (1 - q)vS$$

Asumiendo $S \neq 0$ podemos aislar q de la ecuación que relaciona los precios futuros y la probabilidad neutral al riesgo en un modelo binomial. Tenemos

$$S(1 + rdt) = quS + (1 - q)vS \iff 1 + rdt - v = q(u - v) \iff q = \frac{1 + rdt - v}{u - v}$$

Así pues, podemos definir la función mediante el siguiente código:

```
q_prob=function(r, u, v, dt){
  q=(1+r*dt-v)/(u-v)
  return(q)
}
```

A continuación queremos escribir una función que calcule el valor de una opción de forma recursiva. El primer paso a dar es crear un árbol vacío que rellenaremos con el valor de la opción de forma recursiva. Por lo tanto, creamos una matriz de dimensión $(N+1) \times (N+1)$ dónde almacenaremos los valores de las opciones en cada nodo del árbol, así pues, debe tener las mismas dimensiones que la matriz triangular M creada anteriormente.

Luego llenaremos los últimos nodos del árbol con la función de pago (la función debe aceptar una llamada $(CALL)$ o una opción de venta (PUT)), así tendremos

$$P[N+1,j] = \texttt{payoff}(K, M[N+1,j])$$

donde $\texttt{payoff}(K,S)$ es la opción con pago de *strike* K, cuando el precio final de las acciones es S. Definimos pues las funciones:

```
call=function(K,s){
  (s-K)*(s-K>0)
}

pull=function(K,s){
  (K-s)*(K-s>0)
}
```

Ahora usamos un bucle para completar los nodos un paso hacia atrás usando la fórmula

$$V(t) = \frac{V(t+dt)^{+}q + V(t+dt)^{-}(1-q)}{(1+rdt)}$$

dónde $V(t+dt)^{+}$ significa el valor de la opción en $t+dt$ en el nodo ascendente, y de manera similar para $V(t+dt)^{-}$.

```
value_binomial_option=function(tree, u, v, r, dt, K, type){
  q=q_prob(r, u, v, dt)
  option_tree=matrix(0, nrow=nrow(tree), ncol=ncol(tree))

  if(type=="call"){
    for(i in 1:nrow(tree)){
      option_tree[nrow(tree),i]=call(K, tree[nrow(tree), i])
    }
  }

  if(type=="put"){
    for(i in 1:nrow(tree)){
      option_tree[nrow(tree),i] = put(K,tree[nrow(tree),i])
    }
  }

  for(i in (nrow(tree)-1):1){
    for(j in i:1){
      option_tree[i,j] = (option_tree[i+1,j+1]*q+option_tree[i+1,j]*(1-q))/(1+r*dt)
    }
  }

  return(option_tree)
}
```

El precio de la opción que buscamos es $V(0)$ y, por lo tanto, será la posición $P[1,1]$ de la matriz que imprime nuestra función creada.

En resumen, hemos creado un modelo binomial en una matriz llamada opción binomial de valor. Hemos observado que el elemento $[1,1]$ de la matriz corresponde al primer componente de $V(t)$ y, por lo tanto, al precio final de una opción. De esta manera, para obtener este valor podemos crear la siguiente función:

```
binomial_option=function(type,u,v,dt,r,K,S,N){
  tree=build_stock_tree(S,u,v,N)
  option_tree=value_binomial_option(tree,u,v,r,dt,K,type)
  price=option_tree[1,1]
  return(price)
}
```

Ejercicio 2

Usaremos las funciones que hemos definido anteriormente para calcular el precio de una opción *CALL* con *strike* ATM (*At The Money*, es decir, el precio de negociación subyacente actual), con una subyacente que cotiza a USD 100 y vence en un año, por lo tanto, tenemos valores de $S = K = 100$ y $N = 12$.

Nos fijaremos un paso de tiempo mensual ($dt = 1/12$) y una tasa de interés anual del 10%, así $r = 0, 1$. Finalmente, sabemos que la acción puede subir o bajar un 1% cada mes, dónde tenemos $u = 1, 01$ y $v = 0, 99$.

Ejecutando el siguiente comando

```
binomial_option(type="call",u=1.01,v=0.99,dt=1/12,r=0.1,K=100,S=100,N=12)
```

```
## [1] 9.478797
```

obtenemos un resultado de 9,478797, que es el precio de la opción *CALL* que queríamos obtener.