

Tarea 3. Métodos de Diferencias Finitas para valuación de opciones americanas

Gustavo De la Cruz Tovar

24 de Septiembre 2001

1 Ecuación en derivadas parciales de una opción americana

1.1 Ecuación de Black-Scholes para un put Americano

La ecuación en derivadas parciales debe cumplir una desigualdad de la forma

$$\frac{\partial P(S, t)}{\partial t} + rS \frac{\partial P(S, t)}{\partial S} + \frac{1}{2} \sigma^2 S^2 \frac{\partial^2 P(S, t)}{\partial S^2} - rP(S, t) \leq 0$$

Para esta problema se debe encontrar el valor de la opción y cuando es óptimo el ejercicio de la misma. Esto permite dividir los valores del activo en dos regiones, la primera cumple con:

$$P = E - S \quad \frac{\partial P(S, t)}{\partial t} + rS \frac{\partial P(S, t)}{\partial S} + \frac{1}{2} \sigma^2 S^2 \frac{\partial^2 P(S, t)}{\partial S^2} - rP(S, t) < 0$$

cuando el precio del put es igual al valor de ejercicio, dado un precio de ejercicio E

$$P > E - S \quad \frac{\partial P(S, t)}{\partial t} + rS \frac{\partial P(S, t)}{\partial S} + \frac{1}{2} \sigma^2 S^2 \frac{\partial^2 P(S, t)}{\partial S^2} - rP(S, t) = 0$$

cuando el precio del put es mayor al valor de ejercicio

De ahí, la frontera libre es aquel valor de S , en el tiempo t que cumple:

$$P(S_f(t), t) = \max(E - S_f(t), 0) \quad S_f \quad \text{frontera libre}$$

1.2 Problema del obstaculo

Para poder planterar la solución a la ecuacion de Black y Scholes mostrada, se puede resolver numericamente, pero esto lleva a problemas en la precision y convergencia de la solución. Como en la ecuación de una opción europea, se convierte a la ecuación de calor para simplicar el método numérico, es posible plantear la solución de una opción americana recurriendo al problema del obstáculo.

Se considera una cadena elástica que se encuentra sujeta a dos extremos y en su trayectoria puede toparse con un obstáculo de una altura variable y descrita por una función matemática. Si x describe la posición horizontal de la cadena, donde

$$x \in [-1; 1]$$

y si $u(x)$ describe el desplazamiento vertical de la cadena. El obstáculo queda descrito por una función $f(x)$. En este problema es necesario determinar la forma de la función $u(x)$, de tal manera que al conocer la descripción de $f(x)$, tambien es posible encontrar el punto de contacto entre la cadena y el obstáculo. Esto lleva a un conjunto de desigualdades en derivadas parciales, :

$$\frac{\partial^2 u(x)}{\partial x^2} (u(x) - f(x)) = 0$$

$$-\frac{\partial^2 u(x)}{\partial x^2} \geq 0$$

$$(u(x) - f(x)) \geq 0$$

El conjunto de desigualdades es la formulación del problema del obstáculo como complementaridad lineal. Los puntos de contacto entre la cadena y el obstáculo permiten determinar la frontar libre.

1.3 Opción americana como un problema de obstáculo

El put americano se puede reducir a un problema de frontera libre, y planteado con complementaridad lineal, dando un sistema de desigualdades variacionales parabólicas.

Para reducir la ecuación de Black y Scholes, se necesitan aplicar los siguientes cambios de variables:

$$S = Ee^x$$

$$t = T - (\tau/0.5\sigma^2)$$

$$k = \frac{r}{0.5\sigma^2}$$

$$P(S, t) = u(x, \tau) E e^{-0.5*(k-1)x - 0.25*(k+1)^2 \tau}$$

donde S es el valor del subyacente, T el tiempo de vencimiento, r la tasa libre de riesgo, y sigma la desviación estándar.

Para expresar la condición $\max(E-S, 0)$, se puede poner como una funcion $g()$, tal que:

$$g(x, \tau) = e^{0.25\tau(k+1)^2} \max(e^{0.5(k-1)x} - e^{0.5(k+1)x}, 0)$$

Con la condición inicial :

$$u(x, 0) = g(x, 0)$$

y la restricción

$$P(S, t) \geq \max(E - S, 0)$$

se convierte

$$u(x, \tau) \geq g(x, \tau)$$

Y la condición de frontera :

$$\lim_{x \rightarrow -\infty} u(x, \tau) = 0$$

Este problema se puede plantear como un problema de complementaridad lineal, obteniendo un conjunto de desigualdades variacionales parabólicas:

$$\left(\frac{\partial u(x, \tau)}{\partial \tau} - \frac{\partial^2 u(x, \tau)}{\partial x^2} \right) (u(x, \tau) - g(x, \tau)) = 0$$

$$\left(\frac{\partial u(x, \tau)}{\partial \tau} - \frac{\partial^2 u(x, \tau)}{\partial x^2} \right) \geq 0$$

$$(u(x, \tau) - g(x, \tau)) \geq 0$$

con las condiciones de frontera

$$u(x, 0) = g(x, 0)$$

$$u(-x^-, \tau) = g(-x^-, \tau)$$

$$u(-x^+, \tau) = g(x^+, \tau) = 0$$

Con este planteamiento se muestra donde es óptimo ejercer ($u=g$) .

2 Solución numérica de un put americano

2.1 Solución numérica de un put europeo

Antes de plantear la valuación del put americano, se explicará la manera como se valua un put europeo, dado que la valuación del americano es una modificación.

El planteamiento es resolver la ecuación de calor :

$$\frac{\partial u(x, \tau)}{\partial \tau} = \frac{\partial^2 u(x, \tau)}{\partial x^2}$$

La solución consiste en discretizar la ecuación, de tal forma que se propone una rejilla de valores de x y tau.

Para resolver esta ecuación se puede aplicar diferentes planteamientos.

La solución explícita consiste en realizar un conjunto de iteraciones, dado el valor hacia adelante en tau de todos los x. El problema con este metodo es que puede llevar a condiciones de inestabilidad.

La solución implícita permite calcular los valores hacia adelante en tau, a partir de valores iniciales en tau igual a 0. Esto lleva a la resolución de un sistema de ecuaciones simultaneas, la cuales se pueden solucionar con dos métodos, Crout y SOR.

El método de Crout consiste en factorizar la matriz de coeficientes en dos matrices, L y U, de tal forma que es posible obtener las incógnitas, sin tener que calcular la matriz inversa. A continuación se muestra el código en Java.

```
public class Crout {
    public static boolean debug = true;
    public static double[] solve( double a[][] ) {
        int sz = a[0].length-1;
        //    System.out.println("n="+sz);
        double [] x = new double[sz];
        double [] z = new double[sz];
        double [][] l = new double[sz][sz];
        double [][] u = new double[sz][sz];

        //generar matrices l y u
        l[0][0] = a[0][0];
```

```

    u[0][1] = a[0][1]/l[0][0];
//    System.out.println("l[0][0] "+l[0][0]);
//    System.out.println("u[0][1] "+u[0][1]);

    for ( int i=1;i<sz-1;i++) {
//        System.out.println(i);
        l[i][i-1] = a[i][i-1];
//        System.out.println("l["+i+"] ["+(i-1)+"] "+l[i][i-1]);
        l[i][i] = a[i][i] - l[i][i-1]*u[i-1][i];
//        System.out.println("l["+i+"] ["+i+"] "+l[i][i]);
        u[i][i+1] = a[i][i+1]/l[i][i];
//        System.out.println("u["+i+"] ["+(i+1)+"] "+u[i][i+1]);
    } //for

    l[sz-1][sz-2] = a[sz-1][sz-2];
//    System.out.println("l["+ (sz-1)+"] ["+(sz-2)+"] "+l[sz-1][sz-2]);
    l[sz-1][sz-1] = a[sz-1][sz-1] - l[sz-1][sz-2]*u[sz-2][sz-1];
//    System.out.println("l["+ (sz-1)+"] ["+(sz-1)+"] "+l[sz-1][sz-1]);

    //Resolver Lz = b
    z[0] = a[0][sz] / l[0][0];
//    System.out.println("z[0] "+z[0]);
    for (int i=1;i<z.length;i++) {
        z[i] = (a[i][sz] - l[i][i-1]*z[i-1]) / l[i][i];
//        System.out.println("z["+i+"] "+z[i]);
    } //for

    //Resolver Ux = z
    x[sz-1] = z[sz-1];
    for (int i=sz-2;i>=0;i--) {
        x[i] = z[i] - u[i][i+1]*x[i+1];
    } //for
    return x;
} //solve
} //Crout

```

Otro método es SOR (Successive Over Relaxation), permite resolver un sistema de ecuaciones por medio de iteraciones. Dados los coeficientes de la ecuación y los términos del lado derecho:

$$A_{ij} \quad y \quad b_i$$

Se resuelve con un conjunto de iteraciones, dado un valor inicial propuesto:

$$x_i^{k+1} = x_i^k + \omega \left(\frac{1}{A_{ii}} \left(b_i - \sum_{j=1}^{i-1} A_{ij} x_j^k - \sum_{j=i+1}^n A_{ij} x_j^k \right) \right)$$

donde conforme se calcula el valor de x , se mide la distancia entre el valor calculado y al anterior. El algoritmo es:

```
public class SOR {

    public static double[] resta(double [] v1, double [] v2) {
        double [] v = new double[v1.length];
        for (int i=0;i<v.length;i++) {
            v[i] = v1[i] - v2[i];
        }
        return v;
    } //resta

    public static double norma(double [] v) {
        double acum =0;
        for (int i=0;i<v.length;i++) {
            acum += v[i]*v[i];
        }
        return Math.sqrt(acum);
    } //norma

    public static double [] solve(double [] [] a, double [] b, double [] x0,
        double w, double tol, int N)
    throws SORIteracionesExcedidas{
        int sz = x0.length;
        double [] x = new double[sz];
        for ( int k =1;k<=N;k++) {
            for (int i =0 ;i<sz;i++) {
                double ax =0;
                double ax0 = 0;

                for (int j =0;j<=(i-1);j++) {
                    ax += a[i][j]*x[j];
                }//for
                for (int j =i+1;j<sz;j++) {
                    ax0 += a[i][j]*x0[j];
                } //for
                x[i] = (1-w)*x0[i] + w*( -ax - ax0 + b[i] )/a[i][i];
                // System.out.println(k+" "+x[i]);
            }//for
            //si se excede la tolerancia, romper el ciclo
            if ( norma(resta(x,x0)) < tol ) { return x; }
            //copiar x a x0
            for (int i=0;i<sz;i++) {
                x0[i] = x[i];
            }
        }
    }
}
```

```

        }//for
    }//for
    throw new SORIteracionesExcedidas("Numero maximo de iteraciones excedido");
} //solve
} //SOR

```

A partir de estos algoritmos, es posible resolver la ecuación de calor, usando el método implícito. Se muestra para SOR.

```

public class ParabolicPDESOR {

    public static double[][] solve( double alpha, double T,int nt,
        double XT,int nx,
        CondicionInicial f,CondicionFrontera g1,CondicionFrontera g2,
        double omega, double tol,int maxIter) {
        //definir incrementos
        double dx = XT / nx;
        double dt = T / nt;
        double lambda = alpha*alpha*dt/(dx*dx);
        double [] w = new double[nx-1];
        double [][] u = new double[nt][nx+1];
        //tomar valores iniciales, en t=0 recorriendo toda el eje x
        for ( int i=0 ; i< nx-1;i++) {
            w[i] = f.valor((i+1)*dx);
        }//for

        double [][] A = new double[nx-1][nx-1]; //en la ultima columna tiene los
//valores independientes
        double [] b = new double[nx-1];
        //construir la matriz de coeficientes
        for (int i=0;i<nx-1;i++) {
            if ( (i-1) >=0 ) { A[i] [ i-1] = - lambda; }
            A[i][i] = 1 + 2*lambda;
            if ( (i+1) < (nx -1) ) {A[i][i+1] = -lambda;}
            b[i] = w[i]; //empieza con condiciones iniciales
        }//for
        double [] x0 = new double[b.length];
        for (int i=0;i<x0.length;i++) { x0[i] = 1; }
        //para todo el eje del tiempo
        for (int i=1;i<=nt; i++ ) {
            try {
                w=SOR.solve(A,b,x0,omega,tol,maxIter);
                u[i-1][0] = g1.valor(0,i*dt);
                u[i-1][nx] = g2.valor(XT,i*dt);
            }
        }
    }
}

```

```

        for (int j=0;j<w.length;j++) {
            b[j] = w[j];
            u[i-1][j+1] = w[j];
        }
        } catch ( Exception ex ) { ex.printStackTrace(); }
    } //for
    return u;
} //solve
public static double[][] solve( double alpha, double T,int nt,
    double minXT,double maxXT,int nx,
    CondicionInicial f,CondicionFrontera g1,CondicionFrontera g2,
    double omega, double tol,int maxIter) throws SORIteracionesExcedidas {
    //definir incrementos
    double dx = (maxXT - minXT) / nx;
    System.out.println(dx);
    double dt = T / nt;
    double lambda = alpha*alpha*dt/(dx*dx);
    double [] w = new double[nx-1];
    double [][] u = new double[nt][nx+1];
    //tomar valores iniciales, en t=0 recorriendo toda el eje x
    for ( int i=0 ; i< nx-1;i++) {
        w[i] = f.valor(minXT+(i+1)*dx);
    } //for

    double [][] A = new double[nx-1][nx-1]; //en la ultima columna tiene los
    //valores independientes
    double [] b = new double[nx-1];
    //construir la matriz de coeficientes
    for (int i=0;i<nx-1;i++) {
        if ( (i-1) >=0 ) { A[i] [ i-1] = - lambda; }
        A[i][i] = 1 + 2*lambda;
        if ( (i+1) < (nx -1) ) {A[i][i+1] = -lambda;}
        b[i] = w[i]; //empieza con condiciones iniciales
    } //for
    double [] x0 = new double[b.length];
    for (int i=0;i<x0.length;i++) { x0[i] = 1; }
    //para todo el eje del tiempo
    for (int i=1;i<=nt; i++ ) {
        try {
            w=SOR.solve(A,b,x0,omega,tol,maxIter);
            u[i-1][0] = g1.valor(minXT,i*dt);
            u[i-1][nx] = g2.valor(maxXT,i*dt);
            for (int j=0;j<w.length;j++) {
                b[j] = w[j];
                u[i-1][j+1] = w[j];
            }
        }
    }
}

```

```

        } catch ( SORIteracionesExcedidas ex ) { ex.printStackTrace(); throw ex;}
    } //for
    return u;
} //solve
}

```

Para solucionar la ecuación de Black y Scholes en términos de ecuación de calor, es posible obtener el valor de un put:

```

public class PutEuropeanPDESOR {

    public static double value(double S0,double K,double r,double sigma, double T,
        int nT,double SMax,int nS,double omega,double tolerancia,int maxIter) throws SORIt

        //resolver la ecuacion de calor
        double Ttau = T*sigma*sigma/2;
        double XT = Math.log(SMax/K);
        int nX = nS;
        double k = r/(0.5*sigma*sigma);
        double [][]u=null;
        try {
            u=ParabolicPDESOR.solve(1,Ttau,nT, -XT,XT,nX,
                new PutCondicionInicial(k) ,
                new PutCondicionFrontera(k),
                new CondicionFronteraCero(),
                omega,tolerancia,maxIter
            );
        } catch (SORIteracionesExcedidas ex) { ex.printStackTrace(); throw ex; }

        int fin = u.length-1;
        double [] uT= u[fin];
        double dx = (XT+XT) / nX;
        double P=0,Sant = 0,pAnt=0;
        for (int i=0;i<uT.length;i++) {
            double x = -XT+i*dx;
            double S = K*Math.exp(x);
            double alpha = -0.5*(k-1);
            double beta = -0.25*(k+1)*(k+1);
            double v = Math.exp(alpha*x+beta*Ttau)*uT[i];
            double p = K * v;
            if ( S0 >= Sant && S0<=S ) {
                double p1,p2;
                P = pAnt + (S0-Sant)*(p-pAnt)/(S-Sant);
            }
            Sant = S;
        }
    }
}

```

```

        pAnt = p;
    }
    return P;
}
}

```

2.2 Solución numérica del problema libre de frontera

El problema de la frontera libre, se puede resolver numéricamente por medio de una aproximación discreta:

$$Ax \geq b \quad x \geq c \quad (x - c)(Ax - b) = 0$$

Para solucionar este problema, se debe modificar el método SOR, es el SOR proyectado. Este consiste en aplicar el algoritmo SOR

$$x_i^{k+1} = x_i^k + \omega \left(\frac{1}{A_{ii}} \left(b_i - \sum_{j=1}^{i-1} A_{ij} x_j^k - \sum_{j=i+1}^n A_{ij} x_j^k \right) \right)$$

pero se incluye una nueva condición, que es la de la frontera libre.

$$x_i^{k+1} = \max(c_i, x_i^{k+1})$$

donde c es una función que define la condición de frontera libre.

El código de SOR proyectado es una modificación al SOR, donde se debe indicar la condición de frontera libre y la evaluación del maximo entre esta y la solución del sistema:

```

public class ProjectedSOR {

    public static double[] resta(double [] v1, double [] v2) {
        double [] v = new double[v1.length];
        for (int i=0;i<v.length;i++) {
            v[i] = v1[i] - v2[i];
        }
        return v;
    } //resta

    public static double norma(double [] v) {

```

```

        double acum =0;
        for (int i=0;i<v.length;i++) {
            acum += v[i]*v[i];
        }
        return Math.sqrt(acum);
    } //norma

    public static double [] solve(double [] [] a, double [] b, double [] x0,double[] frontera,
                                double w, double tol, int N)
    throws SORIteracionesExcedidas{
        int sz = x0.length;
        double [] x = new double[sz];
        for ( int k =1;k<=N;k++) {
            for (int i =0 ;i<sz;i++) {
                double ax =0;
                double ax0 = 0;

                for (int j =0;j<=(i-1);j++) {
                    ax += a[i][j]*x[j];
                }//for
                for (int j =i+1;j<sz;j++) {
                    ax0 += a[i][j]*x0[j];
                } //for
                //Comparar contra la frontera
                x[i] = Math.max((1-w)*x0[i] + w*( -ax - ax0 + b[i] )/a[i][i],frontera[i]);
            }//for
            //si se excede la tolerancia, romper el ciclo
            if ( norma(resta(x,x0)) < tol ) { return x; }
            //copiar x a x0
            for (int i=0;i<sz;i++) {
                x0[i] = x[i];
            }//for
        }//for
        throw new SORIteracionesExcedidas("Numero maximo de iteraciones excedido");
    }//solve
} //ProjectedSOR

```

Teniendo este algoritmo, el sistema de desigualdades variacionales parabólicas, es:

- Comenzar con un valor inicial propuestos en el eje de x.
- Proponer un parámetro de relajación
- Construir la matriz de coeficientes

- Aplicar la condición inicial para tau en 0
- Para todo tiempo tau, resolver con el SOR proyectado y obtener los valores de la funcion
- Evaluar cuando los valores son iguales a la condición de frontera libre, y así identificar los puntos que pertenecen a la frontera libre
- Aplicar las condiciones de frontera

```

public class FronteraLibreSOR {
    public static double[] fronteraLibre = null;

    public static double[][] solve( double alpha, double T,int nt,
        double minXT,double maxXT,int nx,
        CondicionInicial f,CondicionFrontera g1,CondicionFrontera g2, FronteraLibre fr
        double omega, double tol,int maxIter) throws SORIteracionesExcedidas {
        //definir incrementos
        double dx = (maxXT - minXT) / nx;
        double dt = T / nt;
        double lambda = alpha*alpha*dt/(dx*dx);
        double [] w = new double[nx-1];
        double [][] u = new double[nt][nx+1];
        double [] xf = new double[nt];
        //tomar valores iniciales, en t=0 recorriendo toda el eje x
        for ( int i=0 ; i< nx-1;i++) {
            w[i] = f.valor(minXT+(i+1)*dx);
        }//for

        double [][] A = new double[nx-1][nx-1]; //en la ultima columna tiene los
        //valores independientes
        double [] b = new double[nx-1];
        //construir la matriz de coeficientes
        for (int i=0;i<nx-1;i++) {
            if ( (i-1) >=0 ) { A[i] [ i-1] = - lambda; }
            A[i][i] = 1 + 2*lambda;
            if ( (i+1) < (nx -1) ) {A[i][i+1] = -lambda;}
            b[i] = w[i]; //empieza con condiciones iniciales
        }//for
        double [] x0 = new double[nx-1];
        for (int i=0;i<x0.length;i++) { x0[i] = 1; }
        //para todo el eje del tiempo
        for (int i=1;i<=nt; i++ ) {
            try {
                double [] fronteraLibre = new double[nx-1];
            }
        }
    }
}

```

```

        //generar la condicion de frontera libre
        for (int j=0;j<nx-1;j++) {
            fronteraLibre[j]=frontera.valor(minXT+(j+1)*dx,i*dt);
        }//for
        w=ProjectedSOR.solve(A,b,x0,fronteraLibre,omega,tol,maxIter);
        //determinar cuando tomo los valores de la condicion de
        //frontera libre
        for (int j=0;j<w.length;j++) {
            if (w[j] == fronteraLibre[j]) {
                xf[i-1] = minXT+(j+1)*dx;
            } else { break; }
        }
        u[i-1][0] = g1.valor(minXT,i*dt);
        u[i-1][nx] = g2.valor(maxXT,i*dt);
        for (int j=0;j<w.length;j++) {
            b[j] = w[j];
            u[i-1][j+1] = w[j];
        }//for
    } catch ( SORIteracionesExcedidas ex ) { ex.printStackTrace(); throw ex;}
} //for
    fronteraLibre=xf;
    return u;
} //solve
}

```

2.3 Valuación de un put americano

La valuación consiste en aplicar el algoritmo para el problema de la frontera libre y la conversión de variables para obtener los valores en términos del model de Black y Scholes.

Para un put,la condicion de frontera libre es:

```

public class PutAmericanoFronteraLibre implements FronteraLibre {
    private double k;
    public PutAmericanoFronteraLibre(double k) {
        this.k = k;
    }

    public double valor(double x,double t) {
        double alpha = 0.5*(k-1);
        double beta = 0.5*(k+1);
        double a=Math.max( Math.exp(alpha*x) - Math.exp(beta*x) , 0 );
    }
}

```

```

        double b=Math.exp(0.25*(k+1)*(k+1)*t);
        return a*b;
    }
}

```

Y la condición inicial es:

```

public class PutCondicionInicial implements CondicionInicial{
    private double k ;
    public PutCondicionInicial(double k) {
this.k = k;
    }
    public double valor(double x){
return Math.max( Math.exp(0.5*(k-1)*x) - Math.exp(0.5*(k+1)*x) , 0 );
    }

}

```

Y la condición de frontera es:

```

public class PutCondicionFrontera implements CondicionFrontera {
private double k;
public PutCondicionFrontera (double k) {
this.k = k;
}

public double valor(double x,double t){
double alpha = 0.5*(k-1);
double beta = 0.25*(k-1)*(k-1);

return Math.exp(alpha*x+beta*t);
}

}

```

y cero para la otra condición de frontera.

Con este algoritmo es posible valuar un put americano. Dado que la rejilla retornada no toca exactamente el valor del subyacente, se necesita hacer una interpolación lineal.

Como ejemplo, se valuó una opción americana con valor de subyacente 8, precio de ejercicio 10, vencimiento a 3 meses, tasa libre de riesgo 0.1 y una desviación estándar de 0.4

Los parámetros del algoritmo son: 80 intervalos en el tiempo Valor máximo del subyacente es 200 100 intervalos para el valor del subyacente Se probó el parámetro de relajación con un intervalo de 0 a 2, con un incremento de 0.1. A partir de 0.9, el algoritmo converge al precio del put. El precio determinado es de 2.0204849039418784

2.4 Determinación de la frontera libre

La frontera libre se determina evaluando cuando el valor de las desigualdades variacionales parabólicas es igual a la condición de frontera libre. El último valor que cumpla esto, es el extremo de la frontera libre. La gráfica que se muestra es la frontera libre, y a partir de ésta, se puede determinar si la opción se ejerce o no.

3 Bibliografía

- Option Pricing. Wilmott, P.
- Análisis Numérico. Burden.
- Options, futures and other derivatives. Hull, J.