Lab11_sol

November 14, 2017

1 Sample solutions of exercises of lab 11

1.1 Exercise 1

Write a function that for given values of m, n, x_{min}, x_{max}, T and for given functions p,σ, ϕ_1 and ϕ_2 returns the values U_{i0}, $i = 0, \ldots, n$ of the approximate solution (option prices) obtained by solving the transformed BS equation by the implicit finite difference method. Use this method for computing approximate values of the option price in the case $r = 0.02$, $\sigma(s,t) = 0.5$, $D = 0.03$, $T = 0.5$, $E = 100$, $S0 = 98$, $p(s) = 2 \cdot |E - s|$, $\rho = 2$, $x_{min} = \ln \frac{S0}{\rho}$, $x_{max} = \ln(\rho\, S0)$, $n = 20$, $m = 100$. Use $\phi_1(t) = p(e^{x_{min}})$, $\phi_2(t) = p(e^{x_{max}})$

Solution

```
In [1]: import numpy as np
        from scipy import linalg
        def p(s): #assume that E will be defined outside
            return 2*abs(s-E)

        def phi1_const(xmin,t):
            return p(np.exp(xmin))

        def phi2_const(xmax,t):
            return p(np.exp(xmax))


        def implicit_solver(m,n,xmin,xmax,r,D,T,sigma,p,phi1,phi2):
            """sigma is assumed to be a function of s and t
            phi1,phi2 are functions of xmin,t and xmax,t
            """
            delta_x=(xmax-xmin)/n
            x=np.linspace(xmin,xmax,n+1)
            #define the function alpha
            #for transformed BS equation
            def alpha(x,t):
                return sigma(np.exp(x),t)**2/2
            delta_t=T/m
            #define values of x_i
            x=np.linspace(xmin,xmax,n+1)
            #define matrix U with dimension (n+1)x(m+1)
```

```
        U=np.zeros(shape=(n+1,m+1))
        #fill in the final condition
        U[:,m]=p(np.exp(x))
        #compute all other values
        i=np.arange(1,n)
        t=np.linspace(0,T,m+1)
        #define matrix M
        M=np.zeros(shape=(n+1,n+1))
        M[0,0]=1
        M[n,n]=1
        #define vector F
        F=np.zeros(n+1)
        for k in range(m-1,-1,-1):
            #compute the coefficients
            beta=r-D-alpha(x[i],t[k])
            a=delta_t/delta_x**2*(-alpha(x[i],t[k])+beta*delta_x/2)
            b=1+2*delta_t/delta_x**2*alpha(x[i],t[k])+r*delta_t
            c=-delta_t/delta_x**2*(alpha(x[i],t[k])+beta*delta_x/2)
            #Fill M with right values
            M[i,i-1]=a
            M[i,i]=b
            M[i,i+1]=c
            #Fill F
            F[0]=phi1(xmin,t[k])
            F[n]=phi2(xmax,t[k])
            F[i]=U[i,k+1]
            #solve the system
            U[:,k]=linalg.solve(M,F)
        return U[:,0] #option prices for t=0
    #test the function with given data
    r = 0.02;  D = 0.03;
    T = 0.5;  E=100;S0 = 98
    rho=2
    xmin=np.log(S0/rho)
    xmax=np.log(S0*rho)
    def sigma(s,t):
        return 0.5
    n=20
    m=100
    approx_sol=implicit_solver(m,n,xmin,xmax,r,D,T,
                               sigma,p,phi1_const,phi2_const)[n//2]
    print(approx_sol)
```

```
54.9213712474
```

Since in the case of the test problem volatility is actually constant and the pay-off function is continuous and piecewise linear, it is possible to express the exact solution in term of pricing

functions of the Put and Call options. For this we notice that the payoff function is just two times the sum of payoff functions with the Put and Call options with exercise price E, so the linearity of the problem implies that the solution is also two times the sum of Put and Call pricing functions for all time values and stock prices.

```
In [2]: import sys
        sys.path.append("c:/users/raul/Google Drive/compfin17/")
        import BSformulas as bs
        exact_price=2*(bs.Call(S0,E,T,r,0.5,D)+bs.Put(S0,E,T,r,0.5,D))
        print(exact_price)
        print("error:",np.abs(exact_price-approx_sol))
```

```
55.0144434066
error: 0.0930721591864
```

So we got an answer with a quite high accuracy.

1.2 Exercise 2

Repeat the previous exercise in the case of boundary conditions derived from special solutions.

Solution

```
In [3]: def phi1_spec(xmin,t):
            c1=-2
            c2=2*E
            return c1*np.exp(-D*(T-t))*np.exp(xmin)+c2*np.exp(-r*(T-t))

        def phi2_spec(xmax,t):
            c1=2
            c2=-2*E
            return c1*np.exp(-D*(T-t))*np.exp(xmax)+c2*np.exp(-r*(T-t))
        approx_sol2=implicit_solver(m,n,xmin,xmax,r,D,T,
                                    sigma,p,phi1_spec,phi2_spec)[n//2]
        print(approx_sol2)
        print("error:",np.abs(exact_price-approx_sol2))
```

```
54.8734570412
error: 0.140986365423
```

For this choice of parameters (ρ, m and n) the simple boundary conditions gave even more accurate result but the error casused by the choice of boundary conditions goes to zero much faster (when ρ increases) in the case of boundary conditions derived from special solutions.