

Finite Difference Approach to Option Pricing II

27 February 1998

CS522 Lab Note

1.0 Nonlinear least square problem

Nonlinear least square problem is as follows :

Given $F : R^n \rightarrow R^m$ $m \geq n$
Find x for which is $\min \|F(x)\|_2^2$

Example - Suppose we wish to fit the data $(t_i, y_i), i = 1, \dots, 4$ with the model $m(x, t_i) = \exp(t_i x_1) - \exp(t_i x_2)$. This problem is apparently overdetermined since there are more 4 equations but 2 variables. We can get the least square solution using the Matlab command `leastsq`. We need to create an m-file describing the function F first.

```
function z = F(x);  
% This m-file describes the function which we wish to minimize.  
t = [1 3 5 7]'; y = [2 4 6 2]';  
z = exp(x(1)*t) - exp(x(2)*t) - y;
```

The syntax of `leastsq` can be found in the following on-line help.

```
LEASTSQ Solves non-linear least squares problems.  
LEASTSQ solves problems of the form:  
min_x sum {FUN(X).^2} where FUN and X may be vectors or matrices.  
X=LEASTSQ('FUN',X0) starts at the matrix X0 and finds a minimum to the  
sum of squares of the functions described in FUN. FUN is usually  
an M-file which returns a vector of objective functions: F=FUN(X).  
NOTE: FUN should return FUN(X) and not the sum-of-squares  
sum(FUN(X).^2) (FUN(X) is summed and squared implicitly in  
the algorithm).  
  
X=LEASTSQ('FUN',X0,OPTIONS) allows a vector of optional parameters to  
be defined. OPTIONS(2) is a measure of the precision required for the  
values of X at the solution. OPTIONS(3) is a measure of the precision  
required of the objective function at the solution. See HELP FOPTIONS.  
  
X=LEASTSQ('FUN',X0,OPTIONS,'GRADFUN') enables a function 'GRADFUN'  
to be entered which returns the partial derivatives of the functions,  
dF/dX, (stored in columns) at the point X: gf = GRADFUN(X).  
  
X=LEASTSQ('FUN',X,OPTIONS,'GRADFUN',P1,P2,...) passes the  
problem-dependent parameters P1,P2,... directly to the functions FUN
```

and GRADFUN: FUN(X,P1,P2,...) and GRADFUN(X,P1,P2,...). Pass empty matrices for OPTIONS and 'GRADFUN' to use the default values.

[X,OPTIONS,F,J]=LEASTSQ('FUN',X0,...) returns, F, the value of FUN(X) at the solution X, and J the Jacobian of the function FUN at the solution.

Now we are ready to solve the above least square problem.

```
>> [x, opt, f, j] = leastsq('F', [0 0]')
x =
    0.1992
   -13.2532
```

The variable opt above provides us with some information about the process. For example,

```
OPTIONS(2)-Termination tolerance for X.(Default: 1e-4).
OPTIONS(3)-Termination tolerance on F.(Default: 1e-4).
OPTIONS(10)-Number of Function and Constraint Evaluations.
OPTIONS(11)-Number of Function Gradient Evaluations.
OPTIONS(12)-Number of Constraint Evaluations.
OPTIONS(13)-Number of equality constraints.
OPTIONS(14)-Maximum number of function evaluations.
           (Default is 100*number of variables)
```

2.0 BLSPRICEFDAM - FD method for American option pricing

We introduce some codes you need for the problem set 2. You can get the on-line help for all the functions we present here.

BLSPRICEFDAM uses the Crank-Nicholson finite difference solution of Black-Scholes equation to price American style options.

The syntax of BLSPRICEFDAM is

```
[call, put, bdata, u] = blspricefdam(S, X, R, T, SIG, Q, M, N);
```

S : current underlying asset price

X : exercise price

R : risk-free interest rate

T : time to maturity in years

SIG : volatility

Q : dividend rate of the asset

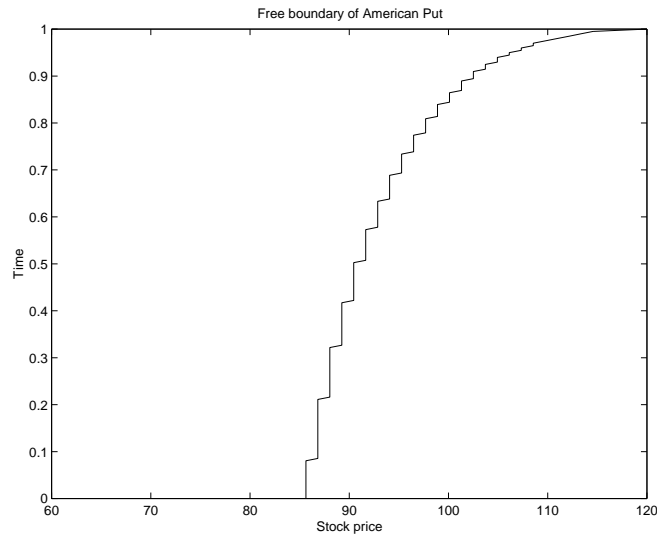
M : number of discretization points in time domain

N : number of discretization points in the stock price domain

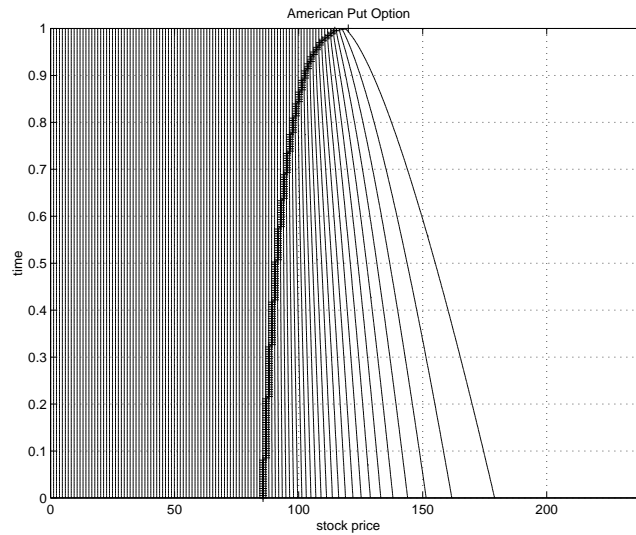
bdata gives us the free boundary data for American put option. `plot(bdata(:,1), bdata(:, 2))` plots the boundary. `u` is the surface of American put option price over the discretized space of S and T . `amcontour(bdata, u)` shows the boundary with put prices.

Example

```
>> S = 100; X = 120; r = .06; T = 1; sig = .3; q = 0; M = 200; N = 200;
>> [call, put, bdata, u] = blspricfdam(S, X, r, T, sig, q, M, N);
>> plot(bdata(:, 1), bdata(:, 2));
>> xlabel('Stock price'); ylabel('Time');
>> title('Free boundary of American Put');
```



```
>> amcontour(bdata, u);
```



3.0 BLSPRICEFDAMSIG - σ as functions of S and T

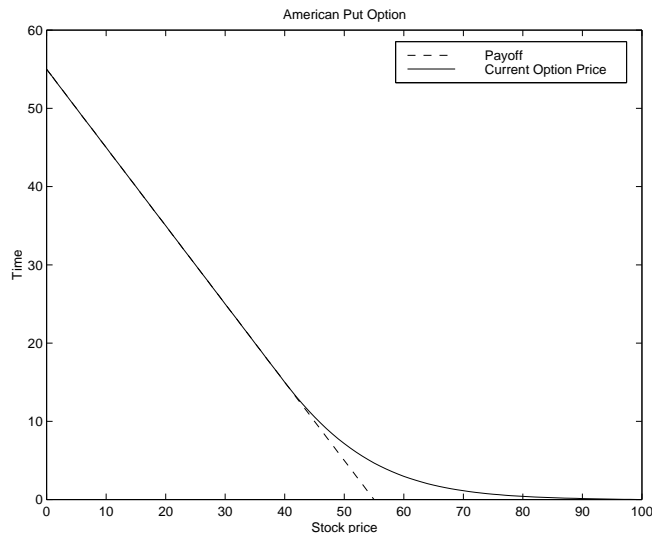
So far σ is considered to be constant. We can generalize this assuming that σ is a function of the underlying stock price and time. `blsfdamsig` allows σ to be treated as a function of S and T. The syntax is

```
[call,put,Svec,Tvec,sigval] = blspricefdamsig(S,X,R,T,SIG,Q,M,N);
```

Note that X, R, and Q must be scalar. `blspricefdamsig` returns matrices of call and put option prices even if S and SIG are vectors of the same size and T is a scalar. The size of the matrix is `length(S) x M`. If S is a vector, S should be equally spaced with `S(1) = 0`.

Example

```
>> S = 0:1:100; X = 55; R = .07; T = .5; SIG = .3:.001:.4; Q = 0;
>> M = 200; N = 200;
>> [call,put,Svec,Tvec,sigval] = blspricefdamsig(S,X,R,T,SIG,Q,M,N);
>> size(call)
ans =
    101    200
>> plot(S, put(:, 1), '--'); % payoff
>> hold on
>> plot(S, put(:, 200))      % current option price
>> xlabel('Stock price'); ylabel('Time'); title('American Put Option');
>> legend('Payoff', 'Current Option Price');
```



Try `mesh(Tvec, Svec, put);`