SCILAB *à l'École nationale des ponts et chaussées*

http://cermics.enpc.fr/scilab

Halmstad 2006

Dynamic programming and Markov chains

Jean-Philippe CHANCELIER

23 novembre 2006 (dernière date de mise à jour)

Table des matières

1 Finite horizon problems with the Cox-Ross Model

The Cox–Ross random walk, widely used in Finance, is defined as follows. Let $(U_n, n \geq 0)$ be a sequence of independent random variables with common law $\mathbb{P}(U_n = 1 + b = d) = p$, $\mathbb{P}(U_n = 1 + a = u) = 1 - p$ where $0 < p < 1$, u and d being two fixed real numbers. Let $X_0 = x$ and

$$X_{n+1} = X_n U_{n+1}.$$

Thus, $X_n = x \prod_{i=1}^{n} U_i$. It is easy to check that (X_n) is an homogeneous Markov.

Let $(X_n)_{n\in\mathbb{N}}$ be a finite Markov chain following a Cox-Ross model, we want to recursively compute for $n \in [0, T]$:

$$v_n(x) = \mathbb{E}\left[\frac{1}{(1+r)^{T+1-n}} f(T, X_T)|X_n = x\right],$$

We will use the following numerical values :

```
r=log(1 + 0.05);    // non risky interest rate
mu=0.05;   // unused
sigma=0.2; // volatility of risky asset
S0=40;    // price at time 0 of the risky asset
T=4.0/12.0;   // horizon
K=40;     //  strike value
N = 10;    // number of time steps
Dt = T/N;  // step size
R = r* Dt; // interest rate on time step
down = (1+R) * exp(- sigma * sqrt(Dt)); // state "up"
up = (1+R) * exp(+ sigma * sqrt(Dt)); // state "down"
p = (up-(1+R))/(up-down); // probability to go down !

a=up-1;
b=down-1;
X0=S0;
```

Question 1 *For a given discrete time t compute in a vector the possible states of X_t.*

```
// The state at time t=3
t=3;
I=0:t;
X=X0*(1+a).^(I).*(1+b).^(t-I);
```

Question 2 *Draw on a figure all the possible states that can be reached from* `X0` *when discrete time evolves from* 0 *to* N

```
// draw the points
// that can be reached from X0
// when the state evolves.

t=N;
xbasc();
for i=0:t
  I=0:i;
  X=X0*(1+a).^(I).*(1+b).^(i-I);
  plot2d(i*ones(I),X,style=-1);
end
```

Question 3 *The number of possible states at time* t *are* $X_0(1+a)^i*(1+b)^(t-i)$ *for* $i \in [0,t]$. *Thus a given state at time* t *can be labeled by its indice* i. *Suppose that we are at time* t *in a state labeled* i *what are the possible labels for the state at time* $t+1$. *What are the probabilities of the possible states.*

Question 4 *We want now to recursively compute the value function* `V(x,n)` *and store all the results in a matrix* `V`. *The number of rows of the matrix is given by the number of possible states at the last discrete time* $t = N$. *At position* `V(i,t)` *we will store the value function at time* `t` *for the state labelled* `i`. *Choosing* $f(x) = (x-K)_+$, *give a function which recursively computes the value function.*

Question 5 *Compute also for each discrete time, the values of the state variables and plot the value function according to the state value.*

```
// The cost function at time T
function y=f(x,K) .... endfunction

// at each time we compute in a column of V the function value associated to states
// X_0(1+a)^i*(1+b)^(t-i)

T=10;
I=0:T;
XT=...
n=.. // maximum number of states for t in [0,T]
V=zeros(n,T); // initialize V
X=zeros(n,T); // initialize X


V(:,T)=...
X(:,T)=...

for t=T-1:-1:1
  ...
  X(I+1,t)=... // I is the vector of possible indices at time t
  V(I+1,t) =...
end

// plot the value function at each step

for t=1:T
  xbasc();
  I=0:t;
  plot2d(X(I+1,t),V(I+1,t))
  halt();
end
```

Question 6 *Use the previous code to implement a function which computes the value function at time t for a given grid of state values. Draw these functions for time evolving from* 0 *to* N.

```
// fix a time and compute the V value on a grid
function v=compute_V(T,X0)
  ...
  v=V(1,1);
endfunction

X=linspace(K-10,K+10,100);
V=zeros(X);
for T=0:10
  // Compute V(i) value at 0 for initial state X(i).
  ...
  plot2d(X,V,style=T)
  halt();
end
```

We want now to verify that given the price V at time 0 of a call, we can manage a portfolio with initial value V which contains the non risky asset and the risky asset in order to bring the portfolio value to `f(X_T,K)` at time t.

Using the previously programmed function `compute_V(N,S0)` we can write

```
// price at time 0 of the call with horizon N
function[Y]= price(n,N,S0)
  Y= compute_V(N-n,S0);
endfunction

// the quantity of risky asset that we must have at time $n$
function[Y]= hedging(n,N,x)
  Y = (price(n+1,N,x*up)-price(n+1,N,x*down)) / (x*(up-down))
endfunction
```

Now using the policy given by `hedging` and a possible sample of X_n in a vector `cours` we can compute the evolution of the portfolio :

```
function [res]=check_hedging(N,cours)
  S = cours(1)
  V = price(0,N,S);
  H = hedging(0,N,S);
  SoHo = V - H * S;
  for n=1:N-1
     S      = cours(n+1);
```

```
    V      = H * S + SoHo * (1+R);
    H      = hedging(n,N,S);
    SoHo   = V - H * S;
  end;
  V = H * cours(N+1) + SoHo * (1+R);
  res = V - f(cours(N+1),K);
endfunction
```

Question 7 *Use random generator to generate samples of a Cox-Ross model for X_n and a given probability. Then use* `check_hedging` *to check that the portfolio value at T is equalt to the value function* `f`*. You can modify the previous function to get also a trajectory for the portfolio value and make drawings of its time evolution. Note that the result does not depend*