

## Numerical Finance Reading Course

### Sheet 5 - Sample Solution

#### Exercise 1: Risk-neutral measures

Consider the binomial model. Let  $S_t$ ,  $B_t$  and  $V_t$  be the price processes of an asset, a bond and a derivative, respectively, and let  $T = \Delta t = 1$ . Prove that  $p = \frac{e^r - d}{u - d}$  defines a risk-neutral measure  $P$  by showing that the initial value  $V_\psi(0)$  of a replicating portfolio  $\psi = (a, b)$  is given by

$$V_\psi(0) = V_0 = e^{-r} \mathbb{E}_P[V_1] = e^{-r} (p \cdot V_1(\{u\}) + (1 - p)V_1(\{d\})).$$

**Hint:** For a replicating portfolio  $(a, b)$  it holds

$$V_T = a \cdot B_T + b \cdot S_T.$$

#### Solution:

Since  $\psi$  is a replicating portfolio, it must hold

$$\begin{aligned} V_u &:= V_1(\{u\}) = a \cdot B_1 + b \cdot S_1(\{u\}) = a \cdot e^r B_0 + b \cdot u S_0 \\ V_d &:= V_1(\{d\}) = a \cdot B_1 + b \cdot S_1(\{d\}) = a \cdot e^r B_0 + b \cdot d S_0 \end{aligned}$$

Hence we can solve for  $a$  and  $b$  to get

$$\begin{aligned} b &= \frac{V_u - V_d}{S_0(u - d)}, \\ a &= \frac{1}{e^r B_0} (V_d - b d S_0) = \frac{1}{e^r B_0} \cdot \frac{uV_d - dV_u}{u - d}. \end{aligned}$$

Then the initial value of the portfolio is given by

$$\begin{aligned} V_0 &= aB_0 + bS_0 \\ &= \frac{B_0}{e^r B_0} \cdot \frac{uV_d - dV_u}{u - d} + \frac{V_u - V_d}{S_0(u - d)} \cdot S_0 \\ &= \frac{1}{e^r} \left( \left( \frac{e^r}{u - d} - \frac{d}{u - d} \right) V_u + \left( \frac{u}{u - d} - \frac{e^r}{u - d} \right) V_d \right) \\ &= e^{-r} \left( \frac{e^r - d}{u - d} V_u + \frac{u - e^r}{u - d} V_d \right) \\ &= e^{-r} (pV_u + (1 - p)V_d) \\ &= e^{-r} \mathbb{E}_P[V_1]. \end{aligned}$$

## Exercise 2: Binomial Tree Algorithm

Implement Algorithm 4.2.2 to find the fair price of a European put with  $S_0 = 10$ ,  $K = 12$ ,  $r = 0.04$ ,  $\sigma = 0.4$  and maturity  $T = 2$ .

**Solution:** For  $M = 10000$ , the Binomial method obtains as value of the put  $V_0 = 2.92294$ .

```
#include <iostream>
#include <cmath>
#include <cstdlib>

using namespace std;

int main(){
    // Input:
    int S = 10;
    int K = 12;
    double r = 0.04;
    double sigma = 0.4;
    double T = 2;

    // Initialization:
    int M = 10000;
    double deltaT = T/(double)M;
    double alpha = exp(r*deltaT);
    double alpha_inv = exp(-r*deltaT);
    double beta = 0.5*(alpha_inv + alpha*exp(sigma*sigma*deltaT));
    double u = beta + sqrt(beta*beta - 1);
    double d = 1./u;
    double p = (alpha - d)/(u - d);

    double* V[M+1];
    for(int i = 0; i < M+1; i++){
        V[i] = (double*)malloc((i+1)*sizeof(double));
    }

    // Forward Phase: we have a closed form solution for S-T,
    // so that we do not have to calculate the values of S_t
    // for t < T.
    V[0][0] = S;
    for(int j = 0; j <= M; j++){
        V[M][j] = S*pow(u, j)*pow(d, M-j);
    }

    // Evaluation
    for(int j = 0; j <= M; j++){
        V[M][j] = (K - V[M][j]) > 0 ? K - V[M][j] : 0;
    }

    // Backward Phase:
    for(int i = M-1; i >= 0; i--){
        for(int j = 0; j <= i; j++){
            V[i][j] = alpha_inv*(p*V[i+1][j+1] + (1-p)*V[i+1][j]);
        }
    }

    cout << "V(0) = " << V[0][0] << endl;
}
return 0;
}
```