

### 2.12 Programming

Write a C/C++ function `double crr(...)` that implements a CRR model for a European call or put with starting price  $S$ , strike price  $K$ , continuous risk-free rate  $rc$ , time to maturity [in years]  $t$ , volatility parameter  $\sigma$  and number of steps  $n$ . Assume no dividends. (Hint: Use the example in 2.14 as a guide.)

SOLUTION:

```
#include <iostream>
#include <math.h>
double crr (double myS, double myK,
            double myt, double myrc,
            double mysigma, int mynumsteps, int myisPut){
    /* declare the return value */
    double retval;
    /* declare variables for supporting calculations */
    double deltat, up, down, pu, pd, discfac;
    double price[mynumsteps+5], optionvalue[mynumsteps+5];
    int ctr, ctr1;
    /* calculate delta t */
    deltat = myt / ( (double) mynumsteps);
    /* calculate discount factor */
    discfac = exp (-myrc*myt);
    /* we run the model without discounting till
     * the last step when we multiply by
     *  $(e^{-r \text{ deltat}})^{\text{mynumsteps}}$  or
     *  $e^{-r \text{ myt}}$  */
    /* calculate up and down steps */
    up = exp(sqrt(deltat)*mysigma);
    down = 1.0/up;
    /* compute risk-neutral probabilities */
    pu = (exp(myrc*deltat)-down)/(up-down);
    pd = 1.0-pu;
    /* compute prices at the expiry date */
    /* top price */
    price[0]=myS*(double)pow(up, mynumsteps);
    /* rest of prices (lower price is
     * given by: upper price / up*down) */
    for (ctr=1; ctr <= mynumsteps+1; ctr++){
        price[ctr]=(price[ctr-1]/up)*down;
    }
}
```

```

/* compute option values at last node */
if (myisPut == 1){
    for(ctr=0; ctr<=mynumsteps+1; ctr++){
        optionvalue[ctr]=(myK-price[ctr]);
        if (optionvalue[ctr]<0){
            optionvalue[ctr]=0;
        } /* put option */
    }
} else {
    for (ctr=0; ctr<=mynumsteps+1; ctr++){
        optionvalue[ctr]=price[ctr]-myK;
        if (optionvalue[ctr]<0){
            optionvalue[ctr]=0;
        }
    } /* call option */
}
/* go backwards,
 * solving prior option values
 * using risk-neutral probabilities */
for (ctr=mynumsteps-1;ctr>=0; ctr--){
    for (ctr1=0; ctr1<=ctr; ctr1++){
        optionvalue[ctr1]=
            (optionvalue[ctr1]*pu+optionvalue[ctr1+1]*pd);
    }
}
retval=optionvalue[0]*discfac;
return retval;
}

int main(){
    double retval;
    double myS, myK, mydays, myt,
        myrc, mysigma;
    int mynumsteps, myisPut;
    std::cout << "Input starting price: ";
    std::cin >> myS;
    std::cout << "Input strike price: ";
    std::cin >> myK;
    std::cout << "Input days to maturity: ";
    std::cin >> mydays;
    std::cout << "Input risk-free rate (rc) as decimal: ";
    std::cin >> myrc;
    std::cout << "Input volatility as decimal: ";
    std::cin >> mysigma;
    std::cout << "Input number of steps: ";
    std::cin >> mynumsteps;

```

```
std::cout << "If the option is a put, type 1: ";
std::cin >> myisPut;
myt = (mydays /365.0);
retval = crr(myS, myK, myt, myrc,
             mysigma, mynumsteps, myisPut);
std::cout << "The option value is: " << retval << std::endl;
}
```