Computational Finance - Mini Task 2

10134621

March 8th 2021

1 Calculate h

$R_{r,t,T}$ is normally distributed with mean $f(r,t,T)$ and variance $v^2(t,T)$,

$$R_{r,t,T} \sim N(f(r,t,T), v^2(t,T)). \tag{1}$$

$N(h)$ is the Cumulative Normal Distribution, or the probability that $x < h$ if $x \sim N(0,1)$, i.e

$$P(x < h). \tag{2}$$

Require $N(h)$ to be equivalent to $P(R_{r,t,T} < X_r)$ which we transform to the standard Normal Distribution and thus

$$P(z < \frac{X_r - f(r,t,T)}{\sqrt{v(r,T)}}), \tag{3}$$

where $z \sim N(0,1)$.

Comparing equation (2) and equation (3) we obtain an expression for h,

$$h = \frac{X_r - f(r,t,T)}{\sqrt{v^2(t,T)}} \tag{4}$$

2 Option value for $\mathbf{r}_0$

Defining the following parameters,

$$\begin{aligned} r_0 &= 0.0263, \\ T &= 3, \\ X_r &= 0.05, \\ \kappa &= 0.0957, \\ \theta &= 0.051, \\ \sigma &= 0.0221, \end{aligned}$$

we calculate the value of the financial contract at time $t = 0$ to be

$$V(r_0, t = 0, T) = 0.819304. \tag{5}$$

3 Option value for multiple r

Taking approximately 100 different values of r in the range $r \in [0, 0.2]$ we make plots of $P(r, t = 0, T)$ and $V(r, t = 0, T)$ as shown in Figure 1.

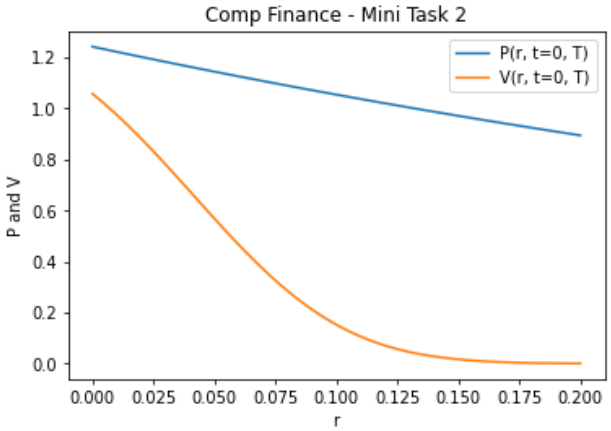

Figure 1: Plots of $P(r, t = 0, T)$ and $V(r, t = 0, T)$ with r for our interest rate derivative contract using a non-standard model.

Listing 1: C++ code for calculating put option value for an interest rate derivative contract using a non-standard model. The data is written to a file called "data.csv" in the working directory.

```
1  // Header
2  // Title: Comp finance - Mini task 2
3  // Student ID: 10134621
4  // Date Created: 03/03/21
5  // Last Edited: 03/03/21
6
7
8  #define _USE_MATH_DEFINES_
9
10
11 // Includes
12 #include <iostream>
13 #include <iomanip>
14 #include <cmath>
15 #include <math.h>
16 #include <fstream>
17 #include <vector>
18 #include <constants.h>  // header file for constants
19
20
```

```
21 // Decalre Functions
22
23 // calculate cummulative normal distribution
24 double norm_cum(const double& x);
25
26 // calculate f
27 double f(const double& r, const double& t, const double&
       T);
28
29 // calculate m
30 double m(const double& r, const double& t, const double&
       T);
31
32 // calculate q
33 double q(const double& t, const double& T);
34
35 // calculate v^2
36 double v2(const double& t, const double& T);
37
38 // calculate P
39 double P(const double& r, const double& t, const double&
       T);
40
41 // calculate n
42 double n(const double& r, const double& t, const double&
       T);
43
44 // calculate k
45 double k2(const double& t, const double& T);
46
47 // calculate V for put
48 double V_put(const double& r, const double& t, const
       double& T, const double& h);
49
50
51
52 // Begin main program
53 int main()
54 {
55     // define variables
56     const double t{ 0 };
57     const double T{ 3 };
58     double b = 0.2;  // lower r limit
59     double a = 0;  // upper r limit
60     double n = 100;  // number of calculations
61
```

```
62      // open a file stream for writing
63      std::ofstream output;
64
65      // open the csv file
66      output.open("data.csv");
67
68      // if the file is open
69      if (output.is_open()) {
70
71          // for loop over r values
72          for (double r{ 0 }; r <= b + 0.002; r += (b - a)
                / n) {
73
74              // calculate h
75              double h_val = (constants::X_r - f(r, t, T))
                    / pow(v2(t, T), 0.5);
76
77              // calculate P
78              double P_val = P(r, t, T);
79
80              // calculate V(r, t=0, T) for a put option
81              double V_val = V_put(r, t, T, h_val);
82
83              // write data to file
84              output << r << "," << P_val << "," << V_val
                    << std::endl;
85
86          }
87
88          // close the file
89          std::cout << "File write successful" << std::endl
                ;
90          output.close();
91
92      }
93      // if file could not be opened
94      else {
95          std::cout << "Error: could not open file" << std
                ::endl;
96          return 1;
97      }
98
99      return 0;
100 }  // End main program
101
102
```

```
103 // Define functions
104
105 // calculate V for put
106 double V_put(const double& r, const double& t, const
        double& T, const double& h)
107 {
108     return P(r, t, T) * norm_cum(h);
109 }
110
111 // calculate cummulative normal distribution
112 double norm_cum(const double& x)
113 {
114     return 0.5 * erfc(-x / pow(2, 0.5));
115 }
116
117 // calculate f
118 double f(const double& r, const double& t, const double&
        T)
119 {
120     return m(r, t, T) - 0.5 * q(t, T);
121 }
122
123 // calculate m
124 double m(const double& r, const double& t, const double&
        T)
125 {
126     return exp(-constants::kappa*(T-t))*r+(1-exp(-
            constants::kappa*(T-t)))*constants::theta;
127 }
128
129 // calculate q
130 double q(const double& t, const double& T)
131 {
132     return (pow(constants::sigma, 2) / (3 * pow(constants
            ::kappa, 2)))* pow(1 - exp(-constants::kappa * (T
            - t)), 5);
133 }
134
135 // calculate v^2
136 double v2(const double& t, const double& T)
137 {
138     return (pow(constants::sigma, 2) / constants::kappa)
            * (1 - exp(-constants::kappa * (T - t)));
139 }
140
141 // calculate P
```

```
142 double P(const double& r, const double& t, const double&
        T)
143 {
144     return exp((2. / 3.) * k2(t, T) - (1. / 4.) * n(r, t,
            T));
145 }
146
147 // calculate n
148 double n(const double& r, const double& t, const double&
        T)
149 {
150     return r * (T - t) - ((constants::theta - r) / (2 *
            constants::kappa)) * (1 - exp(-4 * constants::
            kappa * (T - t)));
151 }
152
153 // calculate k
154 double k2(const double& t, const double& T)
155 {
156     return ((pow(constants::sigma, 2)) / (2 * pow(
            constants::kappa, 3))) * (5 * exp(-constants::
            kappa * (T - t)) - 3 * exp(-2 * constants::kappa *
             (T - t))
157         + 3 * constants::kappa * (T - t) - 2);
158 }
```

Listing 2: Header file for constants.

```
1  #pragma once
2  // Header file for constants
3
4  namespace constants
5  {
6      // define variables
7      const double r_0{ 0.0263 };
8      const double X_r{ 0.05 };
9      const double kappa{ 0.0957 };
10     const double theta{ 0.051 };
11     const double sigma{ 0.0221 };
12 }
```