Student Id: 9910821

Introduction

This report is a two part study of:

I) Pricing a convertible bond contract in which, **at expiry** T the holder has the option to choose between receiving the principle F or alternatively receiving R underlying stocks with price S

II) An extension to the above contract where the holder is able to exercise the decision to convert the bond in stock at **any time before** the maturity of the contract. This is known as an American embedded option

through the use of advanced numerical methods such as Crank-Nicolson with PSOR.

Task 2.1

This task consisted in valuing a portfolio comprising of shorting a call option with strike price X_1, longing a call option with strike price X_2, longing $2X_2$ binary cash or nothing call options with strike price X_2 and unit payoff and longing a call option with strike price equal to zero with parameters:

Appendix

Portfolio Pricing Program Listing

```cpp
#include <iostream>
#include <fstream>
#include <cmath>
#include <vector>
#include <algorithm>
using namespace std;

/*
 * ON INPUT:
 * a, b and c -- are the diagonals of the matrix
 * rhs        -- is the right hand side
 * x          -- is the initial guess
 * iterMax    -- is maximum iterations
 * tol        -- is the tolerance level
 * omega      -- is the relaxation parameter
 * sor        -- not used
 * ON OUTPUT:
 * a, b, c, rhs        -- unchanged
 * x                   -- solution to Ax=b
 * iterMax, tol, omega -- unchanged
 * sor                 -- number of iterations to converge
 */
void sorSolve(const std::vector<double> &a, const std::vector<double> &b, const std::vector<double> &c, const std::vector<double> &rhs,
              std::vector<double> &x, int iterMax, double tol, double omega, int &sorCount)
{
  // assumes vectors a,b,c,d,rhs and x are same size (doesn't check)
  int n = a.size() - 1;
  // sor loop
  for (sorCount = 0; sorCount < iterMax; sorCount++)
  {
    double error = 0.;
    // implement sor in here
    {
      double y = (rhs[0] - c[0] * x[1]) / b[0];
      x[0] = x[0] + omega * (y - x[0]);
    }
    for (int j = 1; j < n; j++)
    {
      double y = (rhs[j] - a[j] * x[j - 1] - c[j] * x[j + 1]) / b[j];
      x[j] = x[j] + omega * (y - x[j]);
    }
    {
      double y = (rhs[n] - a[n] * x[n - 1]) / b[n];
      x[n] = x[n] + omega * (y - x[n]);
    }
    // calculate residual norm ||r|| as sum of absolute values
    error += std::fabs(rhs[0] - b[0] * x[0] - c[0] * x[1]);
    for (int j = 1; j < n; j++)
      error += std::fabs(rhs[j] - a[j] * x[j - 1] - b[j] * x[j] - c[j] * x[j + 1]);
    error += std::fabs(rhs[n] - a[n] * x[n - 1] - b[n] * x[n]);
    // make an exit condition when solution found
```

```
    if (error < tol)
      break;
  }
}
std::vector<double> thomasSolve(const std::vector<double> &a, const std::vector<double> &b_, const std::vector<double> &c, std::vector<double> &d)
{
  int n = a.size();
  std::vector<double> b(n), temp(n);
  // initial first value of b
  b[0] = b_[0];
  for (int j = 1; j < n; j++)
  {
    b[j] = b_[j] - c[j - 1] * a[j] / b[j - 1];
    d[j] = d[j] - d[j - 1] * a[j] / b[j - 1];
  }
  // calculate solution
  temp[n - 1] = d[n - 1] / b[n - 1];
  for (int j = n - 2; j >= 0; j--)
    temp[j] = (d[j] - c[j] * temp[j + 1]) / b[j];
  return temp;
}
/* Template code for the Crank Nicolson Finite Difference
 */
double crank_nicolson(double S0, double X, double F, double T, double r, double sigma,
                      double R, double kappa, double mu, double C, double alpha, double beta, int iMax, int jMax, int S_max, double tol, double omega, int iterMax, int &sorCount, std::vector<double> &gamma)
{
  // declare and initialise local variables (ds,dt)
  double dS = S_max / jMax;
  double dt = T / iMax;
  // create storage for the stock price and option price (old and new)
  vector<double> S(jMax + 1), vOld(jMax + 1), vNew(jMax + 1);
  // setup and initialise the stock price
  for (int j = 0; j <= jMax; j++)
  {
    S[j] = j * dS;
  }
  // setup and initialise the final conditions on the option price
  for (int j = 0; j <= jMax; j++)
  {
    vOld[j] = max(F, R * S[j]);
    vNew[j] = max(F, R * S[j]);
  }
  // start looping through time levels
  for (int i = iMax - 1; i >= 0; i--)
  {
    // declare vectors for matrix equations
    vector<double> a(jMax + 1), b(jMax + 1), c(jMax + 1), d(jMax + 1);
    // set up matrix equations a[j]=
    double theta = (1 + mu) * X * exp(mu * i * dt);
    a[0] = 0;
    b[0] = (-1 / dt) - (r / 2) - (kappa * theta / dS);
    c[0] = (kappa * theta / dS);
    d[0] = (-C * exp(-alpha * i * dt)) + (vOld[0] * (-(1 / dt) + (r / 2)));
```

```
    for (int j = 1; j <= jMax - 1; j++)
    {
      //
      a[j] = (pow(sigma, 2) * pow(j * dS, 2 * beta) / (4 * pow(dS, 2))) - (kappa * (theta - j * dS) / (4 * dS));
      b[j] = (-1 / dt) - ((pow(sigma, 2.) * pow(j * dS, 2. * beta)) / (2. * pow(dS, 2))) - (r / 2.);
      c[j] = ((pow(sigma, 2.) * pow(j * dS, 2. * beta)) / (4. * pow(dS, 2.))) + ((kappa * (theta - j * dS)) / (4. * dS));
      d[j] = (-vOld[j] / dt) - ((pow(sigma, 2.) * pow(j * dS, 2. * beta) / (4. * pow(dS, 2.))) * (vOld[j + 1] - 2. * vOld[j] + vOld[j - 1])) - (((kappa * (theta - j * dS)) / (4. * dS)) * (vOld[j + 1] - vOld[j - 1])) + ((r / 2.) * vOld[j]) - (C * exp(-alpha * dt * i));
    }
    double A = R * exp((kappa + r) * (i * dt - T));
    double B = -X * A + C * exp(-alpha * i * dt) / (alpha + r) + X * R * exp(r * (i * dt - T)) - C * exp(-(alpha + r) * T + r * i * dt) / (alpha + r);
    a[jMax] = 0;
    b[jMax] = 1;
    c[jMax] = 0;
    d[jMax] = jMax * dS * A + B;
    // solve matrix equations with SOR
    sorSolve(a, b, c, d, vNew, iterMax, tol, omega, sorCount);
    //vNew = thomasSolve(a, b, c, d);
    gamma[0] = 0;
    for (size_t j = 1; j < jMax; j++)
    {
      gamma[j] = (1 / (2 * pow(dS, 2))) * (vNew[j + 1] - 2 * vNew[j] + vNew[j - 1] + vOld[j + 1] - 2 * vOld[j] + vOld[j - 1]);
    }
    gamma[jMax] = 0;
    if (sorCount == iterMax)
      return -1;

    // set old=new
    vOld = vNew;
  }
  // finish looping through time levels

  // output the estimated option price
  double optionValue;

  int jStar = S0 / dS;
  double sum = 0.;
  sum += (S0 - S[jStar]) / (dS)*vNew[jStar + 1];
  sum += (S[jStar + 1] - S0) / (dS)*vNew[jStar];
  optionValue = sum;

  return optionValue;
}

int main()
{
  //
  double T = 2., F = 95., R = 2., r = 0.0229, kappa = 0.125, altSigma = 0.416,
```

```cpp
        mu = 0.0213, X = 47.66, C = 1.09, alpha = 0.02, beta = 0.486, sigma = 3.03, tol = 1.e-7, omega = 1., S_max = 10 * X;
  //
  /*
  double T = 3., F = 56., R = 1., r = 0.0038, kappa = 0.083333333, altSigma = 0.369,
        mu = 0.0073, X = 56.47, C = 0.106, alpha = 0.01, beta = 1., sigma = 3.73, S_max = 10 * X, tol = 1.e-7, omega = 1.;
  */
  int iterMax = 10000, iMax = 100, jMax = 100;
  //Create graph of varying  S and optionvalue
  int length = 300;
  double S_range = 3 * X;
  int sor;
  std::ofstream outFile1("./data/varying_s_beta_1.csv");
  std::ofstream outFile2("./data/varying_s_beta_0_4.csv");
  for (int j = 1; j <= length - 1; j++)
  {
    vector<double> gamma(jMax + 1);
    outFile1 << j * S_range / length << " , " << crank_nicolson(j * S_range / length, X, F, T, r, altSigma, R, kappa, mu, C, alpha, 1, iMax, jMax, S_max, tol, omega, iterMax, sor, gamma) << "\n";
    outFile2 << j * S_range / length << " , " << crank_nicolson(j * S_range / length, X, F, T, r, sigma, R, kappa, mu, C, alpha, beta, iMax, jMax, S_max, tol, omega, iterMax, sor, gamma) << "\n";
  }
  outFile1.close();
  outFile2.close();

  std::ofstream outFile3("./data/varying_imax.csv");
  auto incFn = [](int val) { return val + 1; };
  jMax = 25;
  for (iMax = 1; iMax <= 75; iMax = incFn(iMax))
  {
    vector<double> gamma(jMax + 1);
    double S = X;
    outFile3 << S_max << "," << iMax << "," << jMax << "," << S << " , " << crank_nicolson(S, X, F, T, r, sigma, R, kappa, mu, C, alpha, beta, iMax, jMax, S_max, tol, omega, iterMax, sor, gamma) << "\n";
  }
  outFile3.close();

  std::ofstream outFile4("./data/varying_jmax.csv");
  iMax = 25;
  for (jMax = 1; jMax <= 100; jMax = incFn(jMax))
  {
    vector<double> gamma(jMax + 1);
    double S = X;
    outFile4 << S_max << "," << iMax << "," << jMax << "," << S << " , " << crank_nicolson(S, X, F, T, r, sigma, R, kappa, mu, C, alpha, beta, iMax, jMax, S_max, tol, omega, iterMax, sor, gamma) << "\n";
  }
  outFile4.close();

  std::ofstream outFile5("./data/varying_smax.csv");
  iMax = 10;
  tol = 1.e-7;
  for (int s_Mult = 10; s_Mult <= 100; s_Mult = incFn(s_Mult))
```

```
  {
    jMax = s_Mult * 5;
    double S = X;
    S_max = s_Mult * X;
    double result = 0, prevResult = 0;
    std::cout << s_Mult << std::endl;
    do
    {
      int sorCount;
      prevResult = result;
      jMax += 5;
      vector<double> gamma(jMax + 1);
      result = crank_nicolson(S, X, F, T, r, sigma, R, kappa, mu, C, alpha, beta, iMax, jMax, S_max, tol, omega, iterMax, sorCount, gamma);
    } while (trunc(1.e5 * prevResult) != trunc(1.e5 * result) && result != -1);
    outFile5 << S_max << "," << iMax << "," << jMax << "," << S << " , " << prevResult << "\n";
  }
  outFile5.close();

  S_max = 30 * X;
  std::ofstream outFile6("./data/analytic.csv");
  iMax = 10, jMax = 150;

  for (int j = 1; j <= length - 1; j++)
  {
    vector<double> gamma(jMax + 1);

    outFile6 << j * S_range / length << " , " << crank_nicolson(j * S_range / length, X, F, T, r, altSigma, R, 0, mu, C, alpha, 1, iMax, jMax, S_max, tol, omega, iterMax, sor, gamma) << "\n";
  }
  outFile6.close();
}
```

Graphing Program Listing

```
import scipy.stats as si
import numpy as np
import csv
import matplotlib.pyplot as plt

C = 1.09
alpha = 0.02
r = 0.0229
T = 2.
F = 95.
R = 2.
sigma = 0.416
K = 47.66
'''
T=3
C=0.106
alpha=0.01
r=0.0038
R=1
F=56
sigma = 0.369
```

```
K=56.47
'''
#Calulate value of coupon
#Through integrating Cexp(-(alpha+r)t)dt from 0 to T
COUPON = C/(alpha+r) * (1- np.exp((-(alpha+r)*T)))

BOND = F*np.exp(-r*T)

variationData=[]
with open('data/analytic.csv', newline='\n') as csvfile:
    reader = csv.DictReader(csvfile,fieldnames=['S','V'],quoting=csv.
   QUOTE_NONNUMERIC)
    currentData={'x':[],'y':[]}
    for row in reader:
        currentData['x'].append(row['S'])
        currentData['y'].append(row['V'])
    variationData.append(currentData)

def euro_vanilla_call(S, K, T, r, sigma):

    #S: spot price
    #K: strike price
    #T: time to maturity
    #r: interest rate
    #sigma: volatility of underlying asset

    d1 = (np.log(S / K) + (r + 0.5 * sigma ** 2) * T) / (sigma * np.sqrt(T)
   )
    d2 = (np.log(S / K) + (r - 0.5 * sigma ** 2) * T) / (sigma * np.sqrt(T)
   )

    call = (S * si.norm.cdf(d1, 0.0, 1.0) - K * np.exp(-r * T) * si.norm.
   cdf(d2, 0.0, 1.0))

    return call

ANALYTIC_PRICE = []
STOCK_PRICE = []
BOND_FLOOR = []
CONV_BOND = []
for s in range(1,140):
    STOCK_PRICE.append(s)
    ANALYTIC_PRICE.append(R*euro_vanilla_call(s, K, T, r, sigma) + BOND +
   COUPON)

#plt.plot(S,V1, label = " beta = 1")
#plt.plot(STOCK_PRICE,BOND_FLOOR, label = "Bond")
plt.plot(STOCK_PRICE,ANALYTIC_PRICE, label = "Analytic")
plt.plot(variationData[0]['x'],variationData[0]['y'], label = "Crank")
plt.xlabel('Stock price')
plt.ylabel('Eurocall Option')
plt.legend()
plt.savefig('images/analytic.png',bbox_inches='tight', pad_inches=0.2)
```

```
import matplotlib.pyplot as plt
import numpy as np
import csv

```

```python

X=47.66
R=2
F=95

variationData=[]
with open('data/varying_imax.csv', newline='\n') as csvfile:
    reader = csv.DictReader(csvfile,fieldnames=['sMax','iMax','jMax','S','V'],quoting=csv.QUOTE_NONNUMERIC)
    currentData={'x':[],'y':[],'jMax':0,'sMax':0}
    for row in reader:
        currentData['x'].append(row['iMax'])
        currentData['y'].append(row['V'])
        currentData['jMax']=row['jMax']
        currentData['sMax']=int(row['sMax']/X)
    variationData.append(currentData)

plt.figure()
plt.grid()
plt.plot(variationData[0]['x'],variationData[0]['y'],label=r'$\beta=0.486,\sigma=3.03,jMax=%i,sMax=%i$'%(variationData[0]['jMax'],variationData[0]['sMax']),linewidth=2)
plt.xlabel('iMax')
plt.ylabel(r'$V(S=X,t=0)$')
plt.legend(loc='upper center',fancybox=False, framealpha=0.0)
plt.savefig('images/european_varying_imax.png',bbox_inches='tight',pad_inches=0.2)

with open('data/varying_jmax.csv', newline='\n') as csvfile:
    reader = csv.DictReader(csvfile,fieldnames=['sMax','iMax','jMax','S','V'],quoting=csv.QUOTE_NONNUMERIC)
    currentData={'x':[],'y':[],'iMax':0,'sMax':0}
    for row in reader:
        currentData['x'].append(row['jMax'])
        currentData['y'].append(row['V'])
        currentData['iMax']=row['iMax']
        currentData['sMax']=int(row['sMax']/X)
    variationData.append(currentData)

plt.figure()
plt.plot(variationData[1]['x'],variationData[1]['y'],label=r'$\beta=0.486,\sigma=3.03,iMax=%i,sMax=%i$'%(variationData[1]['iMax'],variationData[1]['sMax']),linewidth=2)
plt.xlabel('jMax')
plt.ylabel(r'$V(S=X,t=0)$')
plt.legend(loc='upper center',fancybox=False, framealpha=0.0)
plt.grid()
plt.savefig('images/european_varying_jmax.png',bbox_inches='tight',pad_inches=0.2)

with open('data/varying_smax.csv', newline='\n') as csvfile:
    reader = csv.DictReader(csvfile,fieldnames=['sMax','iMax','jMax','S','V'],quoting=csv.QUOTE_NONNUMERIC)
    currentData={'x':[],'y':[],'jMax':[],'iMax':0}
    for row in reader:
        currentData['x'].append(int(row['sMax']/X))
        currentData['y'].append(row['V'])
        currentData['jMax'].append(row['jMax'])
```

```
        currentData['iMax']=row['iMax']
    variationData.append(currentData)

fig, ax1 = plt.subplots()
ax1.set_xlabel(r'sMax (multiples of X)')
ax1.set_ylabel(r'$V(S=X,t=0)$')
ax1.grid()
ax1.scatter(np.asarray(variationData[2]['x'][:20]),variationData[2]['y'
    ][:20],label=r'$V(S=X,t=0)$ for $\beta=0.486,\sigma=3.03,iMax=%i$'%(
    variationData[2]['iMax']))
ax2 = ax1.twinx()
ax2.set_ylabel(r'$jMax$')
fig.tight_layout()
ax2.plot(np.asarray(variationData[2]['x'][:20]),variationData[2]['jMax'
    ][:20],label=r'$jMax$',color="orange")
lines, labels = ax1.get_legend_handles_labels()
lines2, labels2 = ax2.get_legend_handles_labels()
ax2.legend(lines + lines2, labels + labels2,loc='lower right',fancybox=
    False, framealpha=0.0)
plt.savefig('images/european_varying_smax_zoomed.png',bbox_inches='tight',
    pad_inches=0.2)

fig, ax1 = plt.subplots()
ax1.set_xlabel(r'sMax (multiples of X)')
ax1.set_ylabel(r'$V(S=X,t=0)$')
ax1.grid()
ax1.scatter(np.asarray(variationData[2]['x']),variationData[2]['y'],label=r
    '$V(S=X,t=0)$ for $\beta=0.486,\sigma=3.03,iMax=%i$'%(variationData[2]['
    iMax']))
ax2 = ax1.twinx()
ax2.set_ylabel(r'$jMax$')
fig.tight_layout()
ax2.plot(np.asarray(variationData[2]['x']),variationData[2]['jMax'],label=r
    '$jMax$',color="orange")
lines, labels = ax1.get_legend_handles_labels()
lines2, labels2 = ax2.get_legend_handles_labels()
ax2.legend(lines + lines2, labels + labels2,loc='lower right',fancybox=
    False, framealpha=0.0)
plt.savefig('images/european_varying_smax.png',bbox_inches='tight',
    pad_inches=0.2)

allData=[]
with open('data/varying_s_beta_1.csv', newline='\n') as csvfile:
    reader = csv.DictReader(csvfile,fieldnames=['S','V'],quoting=csv.
    QUOTE_NONNUMERIC)
    currentData={'S':[], 'V':[]}
    for row in reader:
        currentData['S'].append(row['S'])
        currentData['V'].append(row['V'])
    allData.append(currentData)

with open('data/varying_s_beta_0_4.csv', newline='\n') as csvfile:
    reader = csv.DictReader(csvfile,fieldnames=['S','V'],quoting=csv.
    QUOTE_NONNUMERIC)
    currentData={'S':[], 'V':[]}
    for row in reader:
        currentData['S'].append(row['S'])
        currentData['V'].append(row['V'])
```

```
    allData.append(currentData)

plt.figure()
plt.grid()
plt.plot(allData[0]['S'],allData[0]['V'],label=r'$\beta=1,\sigma=0.416$',linewidth=2)
plt.plot(allData[1]['S'],allData[1]['V'],label=r'$\beta=0.486,\sigma=3.03$',linewidth=2)
plt.plot(allData[1]['S'],np.ones(len(allData[1]['S'])) * F,label=r'Bond Only',linewidth=2)
plt.plot(allData[1]['S'],np.asarray(allData[1]['S'])*R,label=r'Options Only',linewidth=2)

plt.xlabel('S')
plt.ylabel(r'$V(S,t=0)$')
plt.legend(loc='upper center',fancybox=False, framealpha=0.0)
plt.savefig('images/european_varying_s.png',bbox_inches='tight', pad_inches=0.2)
```