Joint Default Probability using the Gaussian Copula Method

Below is an implementation of the Gaussian Copula function in C++. The bivariate Gaussian Copula is used to model the dependence and joint behavior of two random variables. When assessing credit risk, the random variables in question, say $u$ and $v$, are the marginal probabilities of default of two members from some larger pool. The Gaussian Copula is given by,

$$C_\rho(u, v) = \Phi(\Phi^{-1}(u), \Phi^{-1}(v)) \quad u, v \in [0, 1]$$

where $\rho$ is the coefficient of correlation between $u$ and $v$.

The marginal probabilities $u$ and $v$ are computed using survival functions based on hazard rates, or default intensities. Default can be modeled as the first arrival time at some point of a Poisson process, where the default intensity is represented by $\lambda$. The probability of surviving between time 0 and time $t$ is therefore,

$$S(t) = e^{-\int_0^t \lambda(t)dt}.$$

The code for the implementation is below.

```
// Gaussian Copula
// Models dependencies between two members of a pool (u and v) based on time to default of each members, and the correlation of default between the two.

#include <iostream>
#include <math.h>
#include <assert.h>

double gaussCopula(double a, double b, double rho);
double survivalFunc_u(double time);
double survivalFunc_v(double time);
double intensity_u(double t);
double intensity_v(double t);
double trapezoidalMethod_u(double f, double low, double up, int n);
double trapezoidalMethod_v(double f, double low, double up, int n);
double N(double z);
double func(double x, double y, double aa, double bb, double rho);
int sign(double q);
double stdNormalCDFInv(double z);
double erfInv(double x);

using namespace std;

const double PI = 3.14159265358979323846264338327950288;
```
double lambda_u = 0.02; //hazard rate for u
double lambda_v = 0.04; //hazard rate for v

int main()
{
    //time (in years) until default
double _time_u = 3;
double _time_v = 3;
    //u and v are to be in the interval (0, 1)
    //default probabilities
double u = 1 - survivalFunc_u(_time_u);
double v = 1 - survivalFunc_v(_time_v);
    //rho is to be between (-1, 1); it is the correlation between u and v
double _rho = 0.5;

double u1 = stdNormalCDFInv(u);
double v1 = stdNormalCDFInv(v);

double PD = gaussCopula(u1, v1, _rho);

cout << "Gaussian Copula" << endl;
cout << "------------------" << endl;
cout << endl;
cout << "The joint probability of default between two members of " << endl;
cout << "the pool, with default rates of " << u << " and " << v << endl;
cout << "with correlation " << _rho << " is " << PD << endl;
cout << endl;
    system("pause");
    return 0;
}

double gaussCopula(double a, double b, double rho)
{
    double begin = a;
double end = b;
double _rho = rho;
double sum = 0;
double rho1 = ((_rho*begin-end)*sign(begin))/sqrt(begin*begin-2*_rho*begin*end+end*end);
double rho2 = ((_rho*end-begin)*sign(end))/sqrt(begin*begin-2*_rho*begin*end+end*end);
double delta = (1-sign(begin)*sign(end))/4;

double aa = begin/sqrt(2*(1-_rho*_rho));
double bb = end/sqrt(2*(1-_rho*_rho));
double A [4] = {0.3253030, 0.4211071, 0.1334425, 0.006374323};

double B [4] = {0.1337764, 0.6243247, 1.3425378, 2.2626645};

if ((begin<=0.0) && (end<=0.0) && (_rho<=0.0))
{
    for(int i=0; i<=3; i++)
    {
        for(int j=0; j<=3; j++)
            sum = sum + (A[i]*A[j]*func(B[i], B[j], aa, bb, _rho));
    }
    double M1_ = sum *(sqrt(1-_rho*_rho)/PI);
    return M1_;
}

else if (begin*end*_rho<=0.0)
{
    if ((begin<=0.0) && (end>=0.0) && (_rho>=0.0))
        return N(begin) - gaussCopula(begin, -end, -_rho);
    else if ((begin>=0.0) && (end<=0.0) && (_rho>=0.0))
        return N(end) - gaussCopula(-begin, end, -_rho);
    else if ((begin>=0.0) && (end>=0.0) && (_rho<=0.0))
        return N(begin) + N(end) - 1.0 + gaussCopula(-begin, -end, _rho);
}
else if (begin*end*_rho>=0.0)
    return gaussCopula(begin, 0, rho1) + gaussCopula(end, 0, rho2) - delta;

double survivalFunc_u(double time)
{//survival function for u
    double t_;  
    double lowBound = 0;  
    double upBound = time;  
    int _n = 100000;  
    double integrand = intensity_u(upBound);  
    return exp(trapezoidalMethod_u(integrand, lowBound, upBound, _n));
}

double survivalFunc_v(double time)
{//survival function for v
    double t_;  
    double lowBound = 0;  
    double upBound = time;  
}
int _n = 100000;
double integrand = intensity_v(upBound);
return exp(trapezoidalMethod_v(integrand, lowBound, upBound, _n));
}

double trapezoidalMethod_u(double f, double low, double up, int n)
{ //integration of the continual variation in intensity for u
    int steps = n;
    double begin = low;
    double end = up;
    double y = 0;
    double answer;

    for (int i = 1; i <= n-1; i++)
        y = y + intensity_u(begin + i*((end-begin)/n));

    answer = ((end-begin)/steps)*(((intensity_u(begin)+intensity_u(end))/2)+ y);

    return answer;
}

double trapezoidalMethod_v(double f, double low, double up, int n)
{ //integration of the continual variation in intensity for v
    int steps = n;
    double begin = low;
    double end = up;
    double y = 0;
    double answer;

    for (int i = 1; i <= n-1; i++)
        y = y + intensity_v(begin + i*((end-begin)/n));

    answer = ((end-begin)/steps)*(((intensity_v(begin)+intensity_v(end))/2)+ y);

    return answer;
}

double intensity_u(double t)
{ //continual variation in intensity for u
    return -lambda_u*t;
}

double intensity_v(double t)
{ //continual variation in intensity for v

return -lambda_v*t;
}

double N(double z)
{//univariate cdf of standard normal distribution
    if (z > 6.0) {return 1.0;}
    if (z < -6.0) {return 0.0;}
    double b1 = 0.31938153;
    double b2 = -0.356563782;
    double b3 = 1.781477937;
    double b4 = -1.821255978;
    double b5 = 1.330274429;
    double p = 0.2316419;
    double c_ = 0.3989423;
    double __a = fabs(z);
    double t = 1.0/(1.0+__a*p);
    double __b = c_*exp((-z)*(z/2.0));
    double n = (((b5*t+b4)*t+b3)*t+b2)*t+b1)*t;
    n = 1.0-__b*n;
    if (z < 0.0)
        n = 1.0 - n;
    return n;
}

double func(double x, double y, double aa, double bb, double rho)
{//density function
    double _rho = rho;
    double f1 = exp(aa*(2*x-aa)+bb*(2*y-bb)+2*__rho*(x-aa)*(y-bb));
    return f1;
}

int sign(double q)
{
    if (q >= 0)
        return 1;
    else
        return -1;
}

double stdNormalCDFInv(double z)
{//inverse standard normal cdf with 0 < z < 1
if (z == 0.5)
    return -1.0100667546808495e-7;
else
    return sqrt(2)*erfInv(2*z-1);
}

double erfInv(double x)
{//inverse error function with -1 < x < 1
  double a = (8/(3*PI))*((PI-3)/(4-PI));

  double f = sqrt(-2/(PI*a))-(log(1-x*x)/2)+sqrt(((2/(PI*a)+(log(1-x*x)/2))/*
      ((2/(PI*a)+(log(1-x*x)/2)))-1/a)*log(1-x*x)));

  if (x == 0)
    return -2.9802322387695313e-8;
else if (x<0)
    return -1*f;

  return f;
}