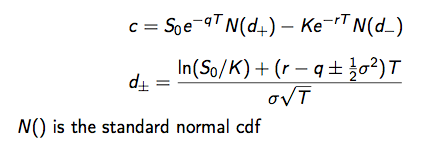
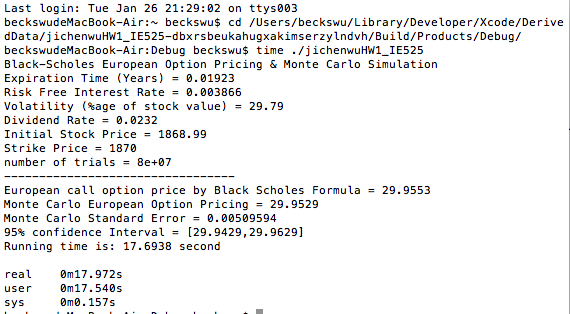
Jichen Wu

1. According to Black Scholes formula:

 where S0 = 1868.99, r = 0.3866%, k = 1870, = 29.79%, q = 2.32%, the call option is 29.9553

2.



3.

obtain a confidence interval that is 2 cents wide, we need 80,000,000 sample size

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Sample size | price | Std error | 95% CI | Time |
| 100 | 34.6291 | 4.9801 | [24.8681,44.3901] | real 0m0.012s  user 0m0.002s  sys 0m0.003s |
| 1000 | 31.6661 | 1.53392 | [28.6596,34.6725] | real 0m0.013s  user 0m0.002s  sys 0m0.003s |
| 10000 | 30.826 | 0.468045 | [29.9087,31.7434] | real 0m0.015s  user 0m0.005s  sys 0m0.003s |
| 100000 | 29.8631 | 0.144129 | [29.5806,30.1456] | real 0m0.028s  user 0m0.025s  sys 0m0.003s |
| 500000 | 30.0575 | 0.0645231 | [29.931,30.184] | real 0m0.103s  user 0m0.100s  sys 0m0.003s |
| 1000000 | 29.9495 | 0.0455774 | [29.8602,30.0389] | real 0m0.217s  user 0m0.212s  sys 0m0.004s |
| 10,000,000 | 29.9472 | 0.0144117 | [29.9189,29.9754] | real 0m2.022s  user 0m2.002s  sys 0m0.011s |
| 80,000,000 | 29.9529 | 0.00509594 | [29.9429,29.9629] | real 0m18.374s  user 0m18.114s  sys 0m0.119s |

//

// main.cpp

// jichenwuHW1\_IE525

//

// Created by beckswu on 16/1/24.

// Copyright © 2016年 beckswu. All rights reserved.

//

//

// main.cpp

// hw1 E525

//

// Created by beckswu on 16/1/23.

// Copyright © 2016年 beckswu. All rights reserved.

//

#include <iostream>

#include <stdlib.h>

#include <cmath>

#include "normdist.h"

#include <vector>

#include<time.h>

using namespace std;

#define max(a, b) (((a) > (b)) ? (a) : (b))

class call\_option

{

double risk\_free\_rate,strike\_price,initial\_stock\_price, expiration\_time,volatility,dividend\_yield;;

double no\_of\_trials;

double simulated\_stock\_price,sim\_option\_price\_sum, option\_price\_square, black\_scholes\_call\_price,sim\_option\_price\_square\_sum;

double option\_price,standard\_error,lower\_bound,upper\_bound;

double no\_of\_trials\_to\_get\_two\_cent\_CI; // this variable is compute the number of trials to get the confidence interval that is 2 cent wide

float get\_uniform()

{

return (((float) random())/(pow(2.0, 31.0)-1.0));

}

float get\_gaussian()

{

return (sqrt(-2.0\*log(get\_uniform()))\*cos(6.283185307999998\*get\_uniform()));

}

double call\_black\_scholes(const double& S, const double& K, const double& r, const double& sigma, const double& time, const double& q)// this function is used to compute option price by Black Scholes fomula

{

double time\_sqrt = sqrt(time);

double d1 = (log(S/K)+(r-q)\*time)/(sigma\*time\_sqrt)+0.5\*sigma\*time\_sqrt;

double d2 = d1-(sigma\*time\_sqrt);

return S\*exp(-q\*time)\*N(d1) - K\*exp(-r\*time)\*N(d2);

}

double N(const double& z) {

if (z > 6.0) { return 1.0; }; // this guards against overflow

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 c2 = 0.3989423;

double a=fabs(z);

double t = 1.0/(1.0+a\*p);

double b = c2\*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;

}

void monte\_carlo\_simulation(double no\_of\_trials)// this function is built by the calculate the mean of option price and the mean of option price square. This method is to avoid calculating error by updating mean in no\_of\_trials times

{

for(int i =0;i<no\_of\_trials;i++)

{

simulated\_stock\_price=initial\_stock\_price\*exp((risk\_free\_rate-dividend\_yield-0.5\*pow(volatility,2))\*expiration\_time+volatility\*get\_gaussian()\*sqrt(expiration\_time));

double simulated\_option\_price= exp(-risk\_free\_rate\*expiration\_time)\*(max(simulated\_stock\_price-strike\_price,0));

sim\_option\_price\_sum += simulated\_option\_price;

sim\_option\_price\_square\_sum += simulated\_option\_price\*simulated\_option\_price;

}

option\_price = sim\_option\_price\_sum/no\_of\_trials;

option\_price\_square = sim\_option\_price\_square\_sum/no\_of\_trials;

standard\_error=sqrt((1/(no\_of\_trials-1))\*(option\_price\_square-option\_price\*option\_price));

upper\_bound = option\_price+1.96\*standard\_error;

lower\_bound = option\_price-1.96\*standard\_error;

}

void monte\_carlo\_simulation\_2(double no\_of\_trials)// this function is built by the update the mean of option square and the mean of option price square in the process of simulation

{

for(int i =0;i<no\_of\_trials;i++)

{

simulated\_stock\_price=initial\_stock\_price\*exp((risk\_free\_rate-dividend\_yield-0.5\*pow(volatility,2))\*expiration\_time+volatility\*get\_gaussian()\*sqrt(expiration\_time));

double simulated\_option\_price= exp(-risk\_free\_rate\*expiration\_time)\*(max(simulated\_stock\_price-strike\_price,0));

option\_price = (option\_price \*i + simulated\_option\_price)/(i+1);

option\_price\_square = (option\_price\_square \*i + simulated\_option\_price\*simulated\_option\_price)/(i+1);

}

standard\_error=sqrt((1/(no\_of\_trials-1))\*(option\_price\_square-option\_price\*option\_price));

upper\_bound = option\_price+1.96\*standard\_error;

lower\_bound = option\_price-1.96\*standard\_error;

}

void print\_result()

{

cout << "Black-Scholes European Option Pricing & Monte Carlo Simulation" << endl;

cout << "Expiration Time (Years) = " << expiration\_time << endl;

cout << "Risk Free Interest Rate = " << risk\_free\_rate << endl;

cout << "Volatility (%age of stock value) = " << volatility\*100 << endl;

cout << "Dividend Rate = "<<dividend\_yield<<endl;

cout << "Initial Stock Price = " << initial\_stock\_price << endl;

cout << "Strike Price = " << strike\_price << endl;

cout << "number of trials = " << no\_of\_trials << endl;

cout << "---------------------------------"<<endl;

cout << "European call option price by Black Scholes Formula = "<<black\_scholes\_call\_price<<endl;

cout <<"Monte Carlo European Option Pricing = " <<option\_price<<endl;

cout <<"Monte Carlo Standard Error = "<<standard\_error<<endl;

cout <<"95% confidence Interval = "<< "["<<lower\_bound<<","<<upper\_bound<<"]"<<endl;

}

public:

void main(int argc, char \* argv[])

{

//sscanf(argv[1], "%lf",&no\_of\_trials);

expiration\_time=0.01923;

risk\_free\_rate=0.003866;

initial\_stock\_price=1868.99;

volatility=0.2979;

strike\_price=1870;

dividend\_yield=0.0232;

no\_of\_trials=80000000;

sim\_option\_price\_sum = 0.0;

sim\_option\_price\_square\_sum=0.0;

option\_price = 0.0;

option\_price\_square = 0.0;

monte\_carlo\_simulation\_2(no\_of\_trials);

black\_scholes\_call\_price = call\_black\_scholes(initial\_stock\_price, strike\_price, risk\_free\_rate, volatility, expiration\_time, dividend\_yield);

print\_result();

}

};

int main(int argc, char\* argv[])

{

clock\_t start\_time=clock();

call\_option x;

x.main(argc, argv);

clock\_t end\_time=clock();

cout<< "Running time is: "<<static\_cast<double>(end\_time-start\_time)/CLOCKS\_PER\_SEC<<" second"<<endl;

return 0;

}

//

// normdist.h

// jichenwuHW1\_IE525

//

// Created by beckswu on 16/1/24.

// Copyright © 2016年 beckswu. All rights reserved.

//

#ifndef normdist\_h

#define normdist\_h

double n(const double& z); // normal distribution function

double N(const double& z); // cumulative probability of univariate normal

double N(const double& a, const double& b, const double& rho);// cumulative probability of bivariate normal

double N3(const double& h, const double& k, const double& j,

const double& rho12, const double& rho13, const double& rho23); // trivariate

double random\_normal(); // random numbers with mean zero and variance one

double random\_uniform\_0\_1();

#endif /\* normdist\_h \*/