Jaime Frade
Computational Finance: Dr. Kopriva
Homework #6

0. Executive Summary

In this assignment, there was a need to setup a program that calculated the value of an option using the Black Scholes differential equation. The method will be based on the Crank-Nicolson scheme which will be demostrated is stable and coverengent, unlike other explict methods. This finite difference method is provides the stability of an implicit scheme, yet still is explicit in its values because it is average of the both explict and implicit methods.

Using the provided test case with a forcing term, the program solved the partial differential equation, where the exact solution was provided. A comparsion was made between the numerical and exact solutions which will be found to be the same.

The expected error behavior will be also shown. As each time step was divided by a factor of two, the error ratio increased by a factor of four. The method is based on second order and the error illusrated this coverengence behavior in space and in time.

After illustrating the program and the methods used are acceptable and expected error behavior, the program will calculate the value of a Euporean Put. From lecture, there valuation has an exact solution so a comparsion using similar methods as in previous task. From Willmott, the program will also calculate the Greeks, the variables measuring the equation's senstivity to stock price, Δ, and interest rate, ρ.

In conclusion, my numerical values over an closed interval of stock prices, [0,20], were the same values as calculated from the exact solution.

1. **Statement of Problem**

1. Problem one

Using a finite difference method solve the following partial differential equation with forcing term, where the exact solution is known. Before solving the problem numerically, I will verify that the adding the given forcing term to the right hand side and adding the intial conditions so that the exact solution is $e^x + e^{-\tau}$.

$$\begin{aligned} v(x,\tau) &= e^x + e^{-\tau} && (1)\\ \frac{\partial v}{\partial \tau} &= -e^{-\tau} && (2)\\ \frac{\partial v}{\partial x} &= e^x && (3)\\ \frac{\partial^2 v}{\partial x^2} &= e^x && (4) \end{aligned}$$

Suppose $\sigma = r = 1$, Subsitituting (1)-(4) into the following and simplifying will obtain:

$$\begin{aligned} \frac{\partial v}{\partial \tau} - rx\frac{\partial v}{\partial x} &= \frac{1}{R}\sigma^2 x^2 \frac{\partial^2 v}{\partial x^2} - rv && (5)\\ -e^{-\tau} - xe^x &= \frac{1}{R}x^2 e^x - e^x - e^{-\tau}\\ -xe^x &= \frac{1}{R}x^2 e^x - e^x\\ e^x\left(1 - x - \frac{1}{R}\right) &= 0 && (6) \end{aligned}$$

From above, (1) is not an exact soution to (5). However, by adding the following forcing term to the right hand side of (5), (1) will become an exact solution.

$$\begin{aligned} \frac{\partial v}{\partial \tau} - rx\frac{\partial v}{\partial x} &= \frac{1}{R}\sigma^2 x^2 \frac{\partial^2 v}{\partial x^2} - rv + \underbrace{e^x\left(1 - x - \frac{1}{R}\right)}_{\text{FORCING TERM}} && (7)\\ e^x\left(1 - x - \frac{1}{R}\right) &= e^x\left(1 - x - \frac{1}{R}\right)\\ 0 &= 0 \end{aligned}$$

Approximation.

To approximate equation (5), I will use the Crank-Nicolson approximation listed below:

$$\delta_\tau^+ u_j^n = \frac{u_j^{n+1} - u_j^n}{\Delta\tau} = \left(\frac{1}{R}\sigma_j^2\, x_j^2\, \delta_x^+\, \delta_x^- + r_j\, x_j \delta_x^0 - r_j\right)\frac{[u_j^{n+1} + u_j^n]}{2} \quad (8)$$

Error behavior

Verfiy that (8) has a local truncation error of $O(\Delta x^2, \Delta\tau^2)$

Numerical problem

Solve the equation on the interval $[0, 1]$ with teh exact solution as the intial condition. Integrate to time $\tau = 1$. Compare the exact solution for a sequence of space and time step sizes.

2. Valuation.

Compute the values of the European put for the following values. $E^* = \$10, r^* = 0.05$ per year, $\sigma^* = 0.20$ per year and with six months ($T^* = 0.6$) to expiry. The interest rate and the volatility are held constant.

(i) Report the values for S^* in $[0, 20]$. Compute and report the solution accurate to the nearest cent.

(ii) Report the value of the option and the $\Delta = \dfrac{\partial V^*}{\partial S^*}$ for $S^* = 8$

(iii) (Extra Credit) Report the value of $\rho = \dfrac{\partial V^*}{\partial r^*}$.

2. **Description of Mathematics**

1 . **Mathematics for Problem 2**

Valuation. Report the value of the option using the Black-Scholes Model.

Formula and the $\Delta = \dfrac{\partial V^*}{\partial S^*}$ for $S^* = 8$ In order to calculate the Δ will use the following

$$\Delta = \frac{\partial V}{\partial S} = N(d_1) - 1 \tag{9}$$

from above, where

$$d_1 = \frac{\log\left(\frac{S}{E}\right) + \left(r + \frac{\sigma^2}{2}\right)(T-t)}{\sigma\sqrt{T-t}}$$

$$N(d_1) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{d_1} e^{\frac{-x^2}{2}}$$

(iii) (Extra Credit) Report the value of $\rho = \dfrac{\partial V^*}{\partial r^*}$. In order to calculate the Δ will use the following:

$$\rho = \frac{\partial V}{\partial r} = -ETe^{-rT} \int_{-\infty}^{d_2} e^{\frac{-x^2}{2}} \tag{10}$$

from above, where

$$d_2 = \frac{\log\left(\frac{S}{E}\right) + \left(r - \frac{\sigma^2}{2}\right)(T-t)}{\sigma\sqrt{T-t}}$$

3. **Description of Algorithm**

In the Black-Scholes equation for a European put will be solved by changing variables such that the resulting parameters will be dimensionless and the equation will become more like a diffusion equation. In the equation variables like r^* are units of time, (years^{-1}) and S^* in dollars. Making the equation dimensionless changes into a forward equation.

From lecture there is certian steps to solve the following problem.

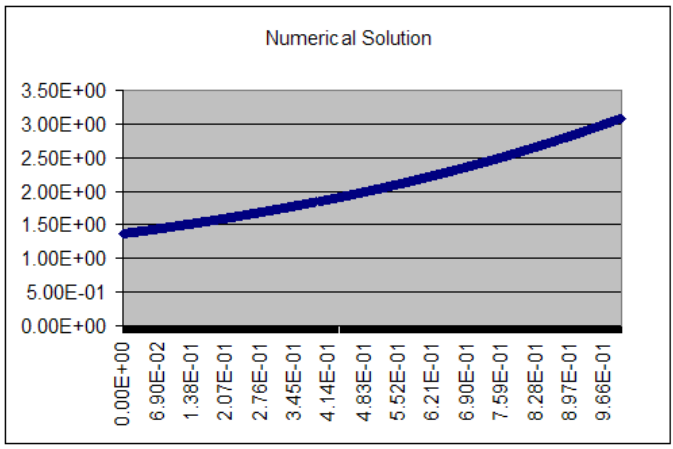
- Changing the variables such that dimensionless.
- Since valuing a put option, set the boundary conditions.
- Calculate the tridiagonals of the matrix, a, b, c
- Using above, calculate the matrix M.
- Calculate the right hand side, RHS.
- Solve the tridiagonal matrix using Thomas algorthim.
- Recurively update the new values
- Chaning the vairbale back to the given dimensions.

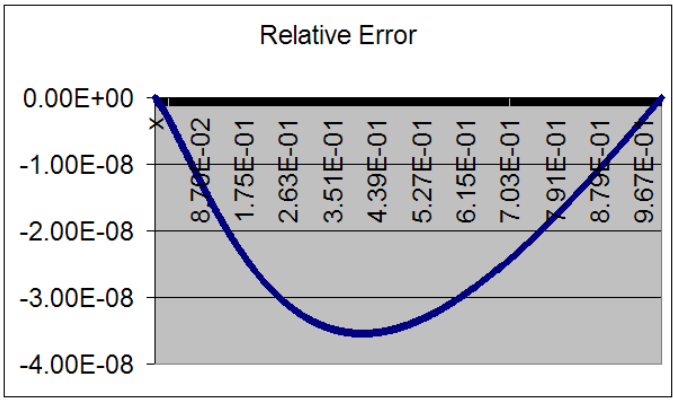
4. **Results**

1. Problem 1: A view on the solution

The numerical solutions over $[0, 1]$. The relative error between the exact solution the numerical solution.

x	Numerical soln	Relative error
$0.00E+00$	1.367879	0.00E+00
$5.00E-02$	1.419151	-6.99E-09
$1.00E-01$	1.47305	-1.45E-08
$1.50E-01$	1.529714	-2.11E-08
$2.00E-01$	1.589282	-2.64E-08
$2.50E-01$	1.651905	-3.04E-08
$3.00E-01$	1.717738	-3.32E-08
$3.50E-01$	1.786947	-3.48E-08
$4.00E-01$	1.859704	-3.54E-08
$4.50E-01$	1.936192	-3.51E-08
$5.00E-01$	2.016601	-3.41E-08
$5.50E-01$	2.101133	-3.24E-08
$6.00E-01$	2.189998	-3.02E-08
$6.50E-01$	2.28342	-2.74E-08
$7.00E-01$	2.381632	-2.43E-08
$7.50E-01$	2.48488	-2.08E-08
$8.00E-01$	2.59342	-1.70E-08
$8.50E-01$	2.707526	-1.30E-08
$9.00E-01$	2.827483	-8.78E-09
$9.50E-01$	2.953589	-4.44E-09
$1.00E+00$	3.086161	0.00E+00





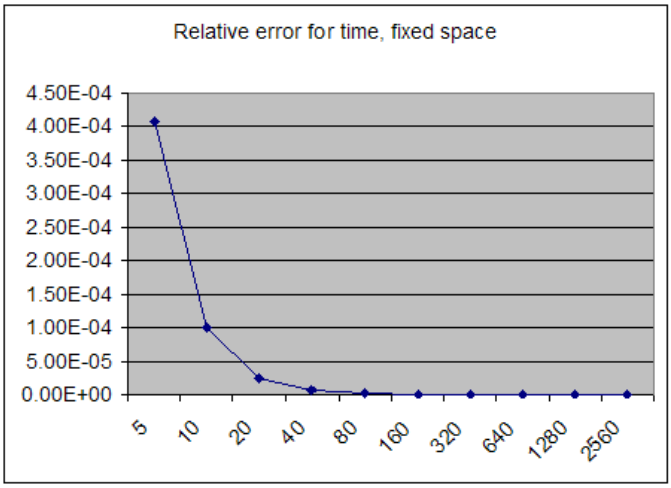
2. Problem 1: A view on the error charateristic in time. Space is held fixed.

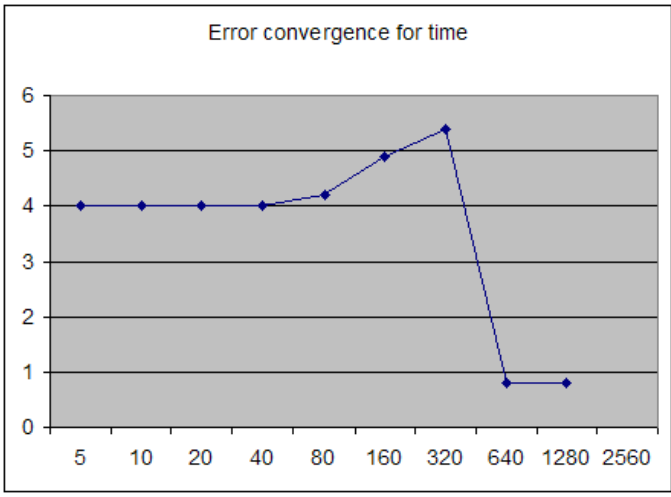
This part will illustrate that error behavior as expected, $O(\Delta t^2)$. As each time step is doubled, the relative error is divided by a factor of four. Therefore,

$$\frac{Error_{\frac{\Delta t}{2}}}{Error_{\Delta t}} \approx 4$$

However, due to roundoff error as discussed in class, as time increases eventually lose expected behavior.

time	Relative error	Error ratio	
5	4.07E-04	4.0297	
10	1.01E-04	3.992	
20	2.53E-05	4.0158	
40	6.30E-06	4.0384	
80	1.56E-06	4.193	
160	3.72E-07	4.881	
320	7.62E-08	5.366	
640	1.42E-08	8.06E-01	
1280	1.76E-08	8.34E-01	
2560	2.11E-08		





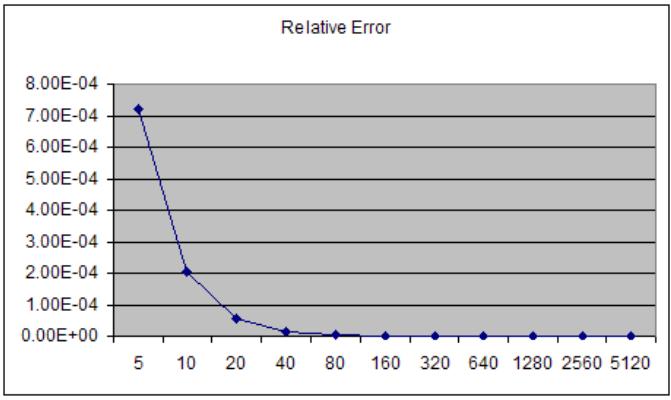
3. Problem 1: A view on the error charateristic in space, where time is held fixed

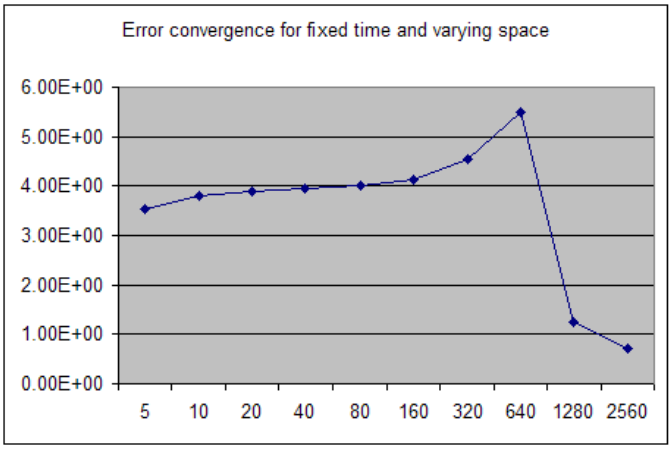
This part will illustrate that error behavior as expected, $O(\Delta x^2)$. As each space step is doubled, the relative error is divided by a factor of four. Therefore,

$$\frac{Error_{\frac{\Delta x}{2}}}{Error_{\Delta x}} \approx 4$$

However, due to roundoff error as discussed in class, as time increases eventually lose expected behavior.

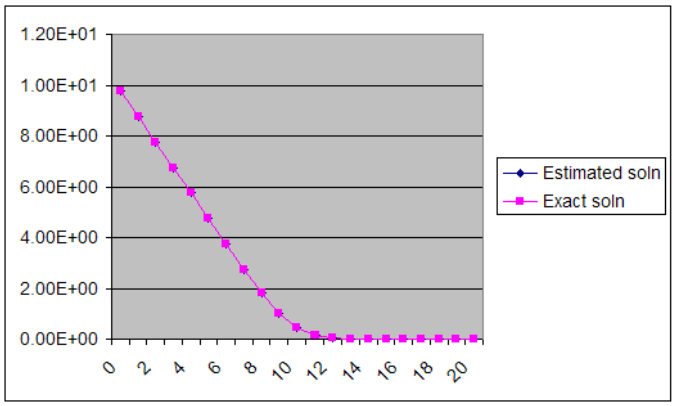
x	Relative error	Error ratio	
5	7.19E-04	3.54E+00	
10	2.03E-04	3.79E+00	
20	5.35E-05	3.91E+00	
40	1.37E-05	3.95E+00	
80	3.47E-06	4.01E+00	
160	8.65E-07	4.12E+00	
320	2.10E-07	4.54E+00	
640	4.63E-08	5.51E+00	
1280	8.41E-09	1.26E+00	
2560	6.70E-09	7.24E-01	
5120	9.20E-09		

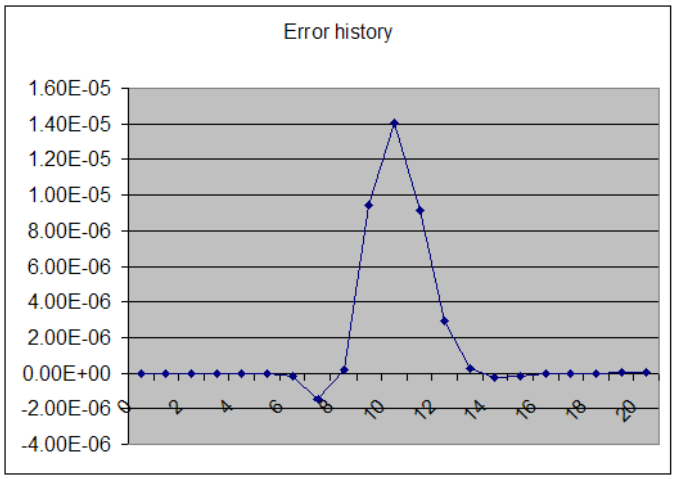




4. Problem 2: Valuation results over $[0, 20]$.

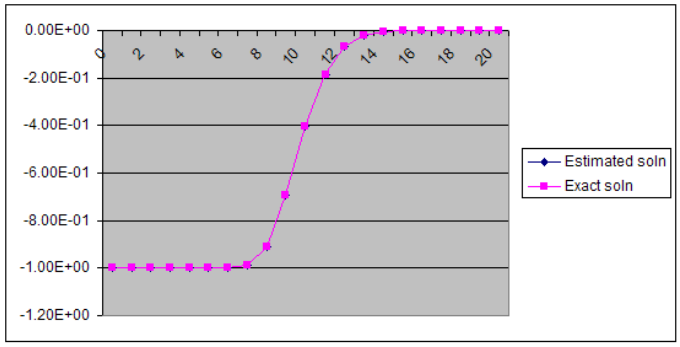
Stock price	Estimated price	Exact price	Error
$0.00E+00$	9.753099	9.753099	2.66E-13
$1.00E+00$	8.753099	8.753099	1.44E-12
$2.00E+00$	7.753099	7.753099	1.50E-12
$3.00E+00$	6.753099	6.753099	1.44E-10
$4.00E+00$	5.753099	5.753099	3.29E
$5.00E+00$	4.753099	4.753099	-2.24E
$6.00E+00$	3.753181	3.753181	-1.88E
$7.00E+00$	2.756837	2.756835	-1.46E
$8.00E+00$	1.798714	1.798715	1.59E
$9.00E+00$	0.9880325	0.9880419	9.46E-06
$1.00E+01$	0.4419579	0.441972	1.40E-05
$1.10E+01$	0.1679003	0.1679094	9.13E-06
$1.20E+01$	0.05088732	0.05089027	2.94E
$1.30E+01$	0.01311002	0.01311026	2.36E
$1.40E+01$	0.002952744	0.002952496	-2.47E
$1.50E+01$	0.0005964066	0.0005962517	-1.55E
$1.60E+01$	0.0001104181	0.0001103609	-5.71E
$1.70E+01$	0.00001908242	0.00001906611	-1.63E
$1.80E+01$	0.000003123969	0.000003120993	-2.98E-09
$1.90E+01$	0.0000004816872	0.0000004900686	8.38E-09
$2.00E+01$	0.00000001033597	0.00000007456492	6.42E-08

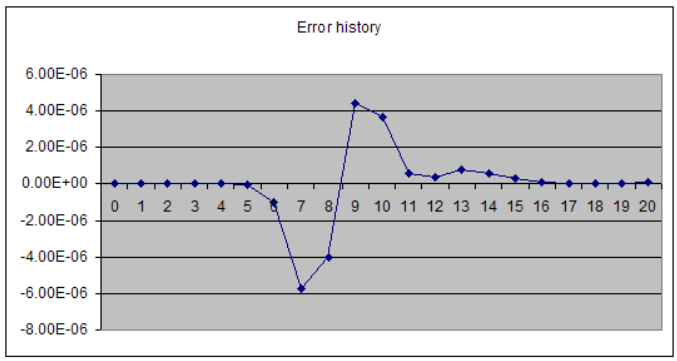




5. **Problem 2: Calcluation of Δ over $[0, 20]$.**

Stock price	Estimated Δ	Exact Δ	Error
$0.00E+00$	-1.00E+00	-1.00E+00	0.0000000003612624
$1.00E+00$	-1.00E+00	-1.00E+00	-0.00000000001025802
$2.00E+00$	-1.00E+00	-1.00E+00	0.000000000004528822
$3.00E+00$	-1.00E+00	-1.00E+00	-0.00000000005836753
$4.00E+00$	-1.00E+00	-1.00E+00	-0.00000000001157752
$5.00E+00$	-1.00E+00	-0.9999984	-0.0(
$6.00E+00$	-1.00E+00	-0.9996167	-0.(
$7.00E+00$	-0.9885288	-0.9885346	-0.(
$8.00E+00$	-0.6905946	-0.9083028	-0.(
$9.00E+00$	-0.4022692	-0.6905902	0.0
$1.00E+01$	-0.4022692	-0.4022655	0.000003698735
$1.10E+01$	-0.1852145	-0.1852139	0.0000005713374
$1.20E+01$	-0.06512931	-0.06512892	0.0
$1.30E+01$	-0.01872267	-0.01872189	0.0(
$1.40E+01$	0.002952744	-0.004573996	0.0(
$1.50E+01$	-0.0009821056	-0.0009818198	0.0(
$1.60E+01$	-0.000190377	-0.0001902766	0.0(
$1.70E+01$	-0.00003406324	-0.00003403421	0.00000002902262
$1.80E+01$	-0.000005728505	-0.000005718948	0.000000009556443
$1.90E+01$	-0.0000009363763	-0.0000009157599	0.00000002061637
$2.00E+01$	-0.0000002595296	-0.000000141356	0.0000001181737



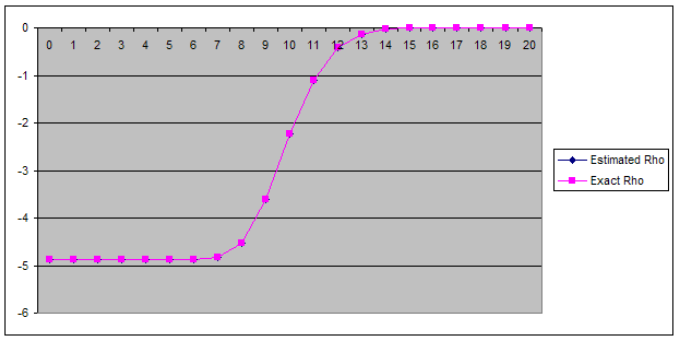


5. **Problem 3: Calcluation of ρ over $[0, 20]$.**

Below are the results for ρ over the given interval. The exact results were obtained by using the following formula, (9)

$$\begin{aligned} \rho &= \frac{\partial V}{\partial r} \\ &= -E(T-t)e^{-r(T-t)}N(-d_2) \end{aligned} \tag{11}$$

Stock price	Estimated ρ	Exact ρ	Error
$0.00E+00$	-4.87649999999995	-4.87655	-0.0000500000000496215
$1.00E+00$	-4.87649999999995	-4.87655	-0.0000500000000496215
$2.00E+00$	-4.87649999999995	-4.87655	-0.0000500000000496215
$3.00E+00$	-4.87649999999995	-4.87655	-0.0000500000000496215
$4.00E+00$	-4.87649999999995	-4.87655	-0.0000500000000496215
$5.00E+00$	-4.87499999999996	-4.876546	-0.0000460000000499505
$6.00E+00$	-4.87499999999996	-4.87544	-0.000440000000040186
$7.00E+00$	-4.83800000000012	-4.838289	-0.
$8.00E+00$	-4.53250000000005	-4.532568	-0.
$9.00E+00$	-3.60169999999999	-3.601677	0.
$1.00E+01$	-2.23229999999999	-2.232314	-0.
$1.10E+01$	-1.09894999999999	-1.098927	0.
$1.20E+01$	-0.414919999999999	-0.4149161	3.89999999900192E-06
$1.30E+01$	-0.12788	-0.127873	0.000007000000000007
$1.40E+01$	-0.0334090000000001	-0.03340274	6.26000000009813E-06
$1.50E+01$	-0.00764500000000003	-0.007642138	2.86200000003053E-06
$1.60E+01$	-0.0015746	-0.001573588	0.0000010120000000001
$1.70E+01$	-0.00029844	-0.0002981432	0.0000002968
$1.80E+01$	-0.0000529745	-0.00005291665	5.78500000000015E-08
$1.90E+01$	-0.0000081399	-0.000008305731	-1.65831000000001E-07
$2.00E+01$	0	-0.000001345503	-0.000001345503



5. **Conclusion**

- From the test case, the obtained numerical results were the same as the exact solution, up to seven sigfigs.
- For number of steps in time was more than the number of steps in space, as needed for stabiltiy.
- The error convergence across space holding time fixed decreased by a factor of 4 as each space in time was halved.
- The above is also shown for holding the number of steps in space fixed and varying time.
- For task two, the graphs illustrate the numerical results for stock prices in [0,20]

APPENDIX: 1 Main program

```
#include <iostream>
using namespace std;
#include "stdio.h"
#include "math.h"
#include "adapsimps.h"
#include "funcs.h"


double PUT(double S, double E, double T, double sigma,
   double r, double *delta, double *rho);


//x-space
//t-time

void main()
{
FILE *pdesoln;
FILE *pdeERROR;

pdesoln = fopen("pdesoln.txt", "w");
pdeERROR = fopen("pdeerror.txt", "w");

const int xspace =1000;
const int tspace =10000;
const double Stock_max = 20.0;
const double Stock_min = 0.0;
const int prob = 1;

double Ma[xspace+1], Mb[xspace+1], Mc[xspace+1],
rhs[xspace+1];
double Un[xspace+1], Unext1[xspace+1];
double S_nonD[xspace+1], x_nonD[xspace+1];
double a[xspace+1], b[xspace+1], c[xspace+1];
double sigma_nonD[xspace+1], r_nonD[xspace+1];
double force[xspace+1];
double t[tspace+1];
double soln[xspace+1], gDELTA[xspace+1], gRHO[xspace+1],
Edelta[xspace+1];
double sigma_0, r_0, R;
double sSCALE, xSCALE, tSCALE;
double ERROR;
double E_star, r_star, sigma_star, t_star;

if (prob ==1)
{
E_star=1.0;
r_star=1.0;
sigma_star=1.0;
t_star=1.0;
sigma_0=sigma_star;
```

```
r_0= r_star;
R = (2.0*r_star)*(1/pow(sigma_star,2));
}
else if (prob ==2)
{
E_star = 10.0;
r_star = 0.05;
sigma_star=0.20;
t_star = 0.5;
sigma_0 = sigma_star;
r_0 = r_star;
R = (2.0*r_star)*(1/pow(sigma_star,2));
}




sSCALE = (Stock_max- Stock_min)*(1/((double) xspace));
xSCALE = sSCALE/ E_star;



for (int i=0; i<=xspace; i++)
{
S_nonD[i] = Stock_min + sSCALE*i;
x_nonD[i] = S_nonD[i]/E_star;
sigma_nonD[i] = sigma_star/sigma_0;
r_nonD[i] = r_star/r_0;
force[i] = (1-x_nonD[i] - pow(x_nonD[i],2)/R)*exp(x_nonD[i]);

}

tSCALE = (r_0*t_star)/((double) tspace);

for (int j=0; j<=tspace; j++)
{
t[j] = tSCALE*j;
}

if (prob ==1)
{
for (int k=1; k<=xspace-1; k++)
{
Un[k] = exp(x_nonD[k]) + 1.0; //tau=0
}
}

else if (prob ==2)
{
for (int k2=1; k2<=xspace-1; k2++)
{
if ((1.0-x_nonD[k2]) >= 0.0)
```

```
{
Un[k2] = 1.0 - x_nonD[k2];
}
else
Un[k2]=0;
}
}


boundary1(&Un[0], &Un[xspace], r_nonD[0], t[0], prob);
calculateDIAGs(a,b,c,r_nonD,x_nonD, sigma_nonD, xSCALE, R, xspace);
calculate_M1(Ma, Mb, Mc, a,b,c,tSCALE,xspace);

double t1[xspace+1];

for(int m=0; m<=tspace-1;m++)
{
boundary1(&Unext1[0], &Unext1[xspace], r_nonD[0], t[m+1], prob);
calculate_RHS1(rhs, Un, Unext1, a,b,c,force, tSCALE, xspace, prob);
triDIAG_solver(Ma,Mb,Mc,t1, rhs, Unext1, xspace);

for(int n=0; n<=xspace; n++)
{
Un[n] = Unext1[n];
}
}

for(int p=0; p<=xspace; p++)
{
Un[p] = Un[p]*E_star;
}

Edelta[0]=(Un[1]-Un[0])/(S_nonD[1]-S_nonD[0]);
Edelta[xspace]=(Un[xspace]-Un[xspace-1])/(S_nonD[xspace]-S_nonD[xspace-1]);

for(int q=1; q<=xspace-1; q++)
{
Edelta[q] = (Un[q+1]-Un[q-1])/(S_nonD[q+1]-S_nonD[q-1]);
}

for(int u=0; u<=xspace; u++)
{
if (prob ==1)
soln[u] = exp(x_nonD[u])+exp(-1.0*t[tspace]);
else if (prob ==2)
soln[u] = PUT(S_nonD[u],E_star,t_star,sigma_0, r_0,&gDELTA[u],&gRHO[u]);
}

ERROR = 0.0;  //global error

for(int i3=0; i3<=xspace; i3++)
{
ERROR += fabs((soln[i3]-Un[i3])/Un[i3]);
```

```
}

ERROR = ERROR/(double (xspace +1));

cout << ERROR << endl;

// for(int i4=0; i4<=xspace; i4++)
// {
// cout << soln[i4] << "\t";
// }



for(int i2=0; i2<=xspace; i2++)
{
fprintf(pdesoln, "%14.6e %14.6e %14.6e %14.6e %14.6e %14.6e %14.6e %14.6e \n",
S_nonD[i2], Un[i2],soln[i2], soln[i2] -Un[i2],
Edelta[i2], gDELTA[i2],gDELTA[i2]-Edelta[i2],
gRHO[i2]);
}
fprintf(pdeERROR, "%6u  %12.6e", tspace, ERROR);
}




double PUT(double S, double E, double T, double sigma,
   double r, double *DELTA, double *RHO)
{
int t=0;
double PUT_price;
double Tol = 1.e-10;
double d1, d2, pde1, pde2;
double alpha_d1, alpha_d2;
int iter =0;

if (S<=0)
{
d1=-100.0;
d2=d1;
}
else
{
int t=0;
d1= log(S/E) + ((r+pow(sigma,2)/2.0)*(T-t));
d1= d1/(sigma*sqrt(T-t));
d2 = log(S/E)+((r-pow(sigma,2)/2.0)*(T-t));
d2=  d2/(sigma*sqrt(T-t));
}

alpha_d1 = -1.0*d1;
alpha_d2 = -1.0*d2;
```

```
if (alpha_d1>0.0)
{
pde1 = ADAP_QUADSIMP(fstdnorm,0.0,alpha_d1,Tol,iter);
pde1 = 0.5 + pde1;
}
else
{
pde1=  ADAP_QUADSIMP(fstdnorm,alpha_d1,0.0,Tol,iter);
pde1 = 0.5 - pde1;
}

if (alpha_d2>0.0)
{
pde2 =  ADAP_QUADSIMP(fstdnorm,0.0,alpha_d2,Tol,iter);
pde2 = 0.5 + pde2;
}
else
{
pde2 = ADAP_QUADSIMP(fstdnorm,alpha_d2,0.0,Tol,iter);
pde2 = 0.5 - pde2;
}

PUT_price= E*exp(-1.0*r*(T-t))*pde2-S*pde1;

if(d1>0.0)
{
*DELTA = 0.5+ ADAP_QUADSIMP(fstdnorm,0.0,d1,Tol,iter);
*DELTA = *DELTA-1.0;
}
else
{
*DELTA = 0.5- ADAP_QUADSIMP(fstdnorm,d1,0.0, Tol,iter);
*DELTA = *DELTA-1.0;
}

*RHO = -1.0*E*(T-t)*exp(-1.0*r*(T-t))*pde2;

return PUT_price;

}
```

APPENDIX: 2 Functions.h //all functions used in main

```
#include "math.h"



void boundary1(double *u_intial, double *u_final,double r, double t, long prob)
{
if (prob==1)
{
*u_intial = exp(-1.0*t) + 1.0;
*u_final = exp(-1.0*t) + exp(1.0);
}
else if (prob==2)
{
*u_intial = exp(-1.0*r*t);
*u_final = 0.0;
}
}



void calculateDIAGs(double *a, double *b, double *c,
   double *r, double *x, double *sigma,
  double delta_x, double R, long xspace)

{
double a1, b1;

for(long i=0; i<=xspace; i++)
{
a1 = (*(r+i))*(*(x+i))/(2.0*delta_x);
b1 = pow(*(sigma+i),2)*pow(*(x+i),2)*(1/R)*(1/pow(delta_x,2));
*(a+i) = -1.0*a1+b1;
*(b+i) = -2.0*b1-*(r+i);
*(c+i) = a1+b1;
}
}

void calculate_M1(double *Ma, double *Mb, double *Mc,
 double *a, double *b, double *c,
 double delta_t, long xspace)

{
double delta_t2;
delta_t2 = ((delta_t)/2);

for (long j=0; j<=xspace; j++)
{
*(Ma+j) = -delta_t2*(*(a+j));
*(Mb+j) = 1.0-delta_t2*(*(b+j));
*(Mc+j) = -delta_t2*(*(c+j));
}
```

```
}

void calculate_RHS1(double *rhs, double *Un, double *Unext1,
double *a, double *b, double *c,double *force,
double delta_t, long xspace, long prob)
{
long p;
double delta_t2;
delta_t2 = ((delta_t)/2);

for(long i=1; i<=xspace-1; i++)
{
*(rhs+i) = delta_t2*(*(a+i))*(*(a+i))*(*(Un+i-1))
+ (1.0+ delta_t2*(*(b+i)))*(*(Un+i))
+ delta_t2*(*(c+i))*(*(Un+i+1));
}

p =1;
*(rhs+p) = *(rhs+p)+delta_t2*(*(a+p))*(*(Unext1+p-1));

p=xspace-1;
*(rhs+p) = *(rhs+p)+delta_t2*(*(c+p))*(*(Unext1+p+1));

if (prob ==1)
{
for (long k=1; k<=xspace-1; k++)
{
*(rhs+k) = *(rhs+k) + delta_t*(*(force+k));
}
}
}



void triDIAG_solver(double *a, double *b, double *c, double *d, double *rhs, double *y,long xspace)
{
for(long i=0; i<=xspace; i++)
{
*(d+i)=*(b+i);
}

for(long j=2; j<=xspace-1; j++)
{
*(d+j) = *(d+j)-(*(a+j))*(*(c+j-1))*(1/(*(d+j-1)));
*(rhs+j) = *(rhs+j)- (*(a+j))*(*(rhs+j-1))*(1/(*(d+j-1)));

}

*(y+xspace-1) = *(rhs+xspace-1)*(1/(*(d+xspace-1)));

for (long k=xspace-2; k>=1; k--)
{
*(y+j) = *(rhs+j)-(*(c+j))*(*(y+j+1))*(1/(*(d+j)));
```

```
}
}

float fstdnorm(float x)
{
float f;
float pi; //= 3.141592654;
pi = 3+sqrt(5);
pi = pi*3;
pi = pi/5;
f=exp(-1.0*pow(x,2)*0.5)/(sqrt(2.0*pi));
return f;

}
```

```
# include "stdio.h"

//Need to derive aan adaptive
//Simpson's rule to approx the
//value of a definite integral


//AbsTol: absolute error tolerance


float SIMPSON(float (*f)(float), float a, float b)
{
float f0, f1, f2;
float I;

f0=f(a);
f1=f((a+b)/2);
f2=f(b);
I = (f0+4*f1+f2)*((b-a)/6.0);
return I;

}

float ADAP_QUADSIMP(float (*f)(float),float a, float b, float Tol, int & iter)
{
float mid;
float Icenter,Isum,Ileft,Iright;
float error;
float r=4.0; //power for error term


iter=iter+1;

mid=(a+b)/2.0;

Icenter=SIMPSON(f,a,b);
Ileft=SIMPSON(f,a,mid);
Iright=SIMPSON(f,mid,b);

Isum = Ileft + Iright;

error = (1/(pow(2,r)-1.0))*(Isum-Icenter);

if (fabs(error)<Tol)
{
if (Tol>1.e5)
return Icenter;
else
return Isum + error;
}
else
```

```
{
Ileft = ADAP_QUADSIMP(f,a,mid,Tol*0.5,iter);
Iright = ADAP_QUADSIMP(f,mid,b,Tol*0.5,iter);
return Ileft + Iright;
}
}

float ADAP_QUADSIMP_2(float (*f)(float),float a, float b, float Tol, int & iter, FILE *file)
{
float mid;
float Icenter,Isum,Ileft,Iright;
float error;
float r;

r=4.0; //power for error term

// cout a, b value
fprintf(file,"%7.4e %7.4e \n",a,b);

iter=iter+1;

mid=(a+b)/2.0;

Icenter=SIMPSON(f,a,b);
Ileft=SIMPSON(f,a,mid);
Iright=SIMPSON(f,mid,b);

Isum = Ileft + Iright;

error = (Isum-Icenter)/(pow(2,r)-1.0);

if (fabs(error)<Tol)
return Isum + error;
else
{
Ileft = ADAP_QUADSIMP_2(f,a,mid,Tol*0.5,iter, file);
Iright = ADAP_QUADSIMP_2(f,mid,b,Tol*0.5,iter, file);
return Ileft + Iright;
}
}
```