Jaime Frade
Computational Finance: Dr. Kopriva
Homework #5

0. **Executive Summary**

In this assignment I developed a program which evaluates definite integrals given the intregrand f, the closed interval $[a, b]$, and a relative tolerance for the error. I will use an adaptive quadrature method that uses Simpson's rule with an estimate for the errors involved. I will then use the equation to solve applications in the **Black-Scholes** equation.

To ensure accuracy, I tested several functions with known and calculatble results. (Advice from collegue, similiar to HW3). Each test case function was specifically chosen so that I could calculate the exact value and the error estimate in theory. The numerical results from the code were compared to the exact values and the error was given with a certian tolerance. Each test case was proven correct and behaved according to the error estimates.

I obtained the following for each question in this homework:

(b). $P(1) = 0.5 + 0.341342151 = 0.841342151$

(c). Value of Call: 98.5671380

(d). Implied volatility: 7.6089457%

1. **Statement of Problem**

In this homework assignment there were 4 assigned problems.

(a). Write an adaptive quadrature function that uses Simpson's rule as its quadrature rule. Test your routine on some integrals for which I know the exact values and that demonstrate how the method behaves.

(b). Find the value P at $x = 1$ in the function below, accurate to at least six significant digits. Explain the acute of accuracy in my result.

$$P(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{\frac{-x^2}{2}} dt \tag{1}$$

(c). Given the data below, compute the value of the call, C, to at least three significant digits using the equations below and (1). State the number of figures.

Given the following:

Time of expiry (T)	$\frac{54}{365}$
Current time (t)	0
Interest rate (r)	$6.75\% = 0.0675$
Volatility (σ)	$13.5\% = 0.135$
Exercise Price (E)	3425.0
Asset Price (S)	3441.0

Using the information above and (1), compute the **value for the call**, C, using the following equations:

$$C(S,t) = SP(d_1) - Ee^{-r(T-t)}P(d_2), \tag{2}$$

where d_1 and d_2 are determined by

$$d_1 = \frac{\log\left(\frac{S}{E}\right) + \left(r + \frac{\sigma^2}{2}\right)(T-t)}{\sigma\sqrt{T-t}} \tag{3}$$

$$d_2 = \frac{\log\left(\frac{S}{E}\right) + \left(r - \frac{\sigma^2}{2}\right)(T-t)}{\sigma\sqrt{T-t}} \tag{4}$$

(d). Given the data below, compute the **implied volatilty**, σ to at least four significant digits using the equations below and (1).

Given the following:

Value of call (C)	94.0
Time of expiry (T)	$\frac{117}{365}$
Current time (t)	0
Interest rate (r)	$6.75\% = 0.0675$
Exercise Price (E)	3475.0
Asset Price (S)	3461.0

In this problem, I will determine the implied volatility by using the methods from previous homework. Using the Newton-Rhapson method, I will calculate with a certain error.

2. **Description of the Mathematics**

There are several concepts of mathematics used in this homework assignment:

(a). Statistics. In a given Gaussian standard normal probability function, (1), to make calculations in **part(b)** less complicated and less prone to error. I will use the properties of integrals to split up into sum of two separate integrals where one is known. Let $x = 1$.

$$\begin{aligned} P(1) &= \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{1} e^{\frac{-x^2}{2}} dt \\ &= \underbrace{\frac{1}{\sqrt{2\pi}} \int_{-\infty}^{0} e^{\frac{-x^2}{2}} dt}_{=0.5} + \frac{1}{\sqrt{2\pi}} \int_{0}^{1} e^{\frac{-x^2}{2}} dt \\ &= \frac{1}{2} + \frac{1}{\sqrt{2\pi}} \int_{0}^{1} e^{\frac{-x^2}{2}} dt \end{aligned} \tag{5}$$

(b). Black-Scholes equation. In **part(c-i)**, I will use the given equations, (2) →(4), to solve for the value of the call option.

$$\begin{aligned} C(S,t) &= SP(d_1) - Ee^{-r(T-t)} P(d_2) \\ &= \frac{S}{\sqrt{2\pi}} \int_{-\infty}^{d_1} e^{\frac{-x^2}{2}} dt - \frac{Ee^{-r(T-t)}}{\sqrt{2\pi}} \int_{-\infty}^{d_2} e^{\frac{-x^2}{2}} dt \end{aligned} \tag{6}$$

In **part(c-ii)**, I will use the given equations, (2)-(4), to solve for the value of the implied volatility. I will use (6), however, since the value for the call is known, C, I will set it equal to zero. The equation is rewritten as a function of σ. Refer to (3),(4) for equations of d_1 and d_2

$$\begin{aligned} C(S,t) &= \frac{S}{\sqrt{2\pi}} \int_{-\infty}^{d_1} e^{\frac{-x^2}{2}} dt - \frac{Ee^{-r(T-t)}}{\sqrt{2\pi}} \int_{-\infty}^{d_2} e^{\frac{-x^2}{2}} dt \\ F(\sigma) &= \frac{S}{\sqrt{2\pi}} \int_{-\infty}^{d_1} e^{\frac{-x^2}{2}} dt - \frac{Ee^{-r(T-t)}}{\sqrt{2\pi}} \int_{-\infty}^{d_2} e^{\frac{-x^2}{2}} dt - C(S,t) \end{aligned} \tag{7}$$

Let $F(\sigma) = 0$. Using the **Newton Method**, I will determine the solution and error.

(c). Calculus. In this program, I will need to approximate the value of an integral. I will use Simpson's Rule as my quadrature rule to evaluate. I will be exact for all polynomials of degree ≤ 3, which will be illustrated in the tests cases. In this quadrature rule, I will use three points in an interval to determine the approximate to my solution. For the values, $x_0 \leq x_1 \leq x_2$, can estimate the area, $\widetilde{I}$ and the error, E using the following:

$$\int_{x_0}^{x_2} f(x)dx \approx \underbrace{\frac{\Delta x}{3}[y_0 + 4y_1 + y_2]}_{\widetilde{I}} - \underbrace{\frac{(\Delta x)^5}{90} f^{(4)}(\xi)}_{E} \tag{8}$$

(d). Calculus. Using the (8), I will use the Simpson's Adapative Quadature method to obtain the value for the integral faster. This method is faster, because in the Simpson's rule, the idea was that on a given subinterval is not sufficently accurate to within an acceptable tolerance, the interval is subdivided into **two equal parts**. Then the process, Simpson's rule, is repeated on each half. In the adaptive approach, the subinterval are refined as in the Simpson's rule, however, only refined over the subintervals needed. For example, if the area for subinterval between $[x_0, x_1]$ is within a certain tolerance, and $[x_1, x_2]$ is not; $[x_0, x_1]$ is left alone and and will reuse Simpson's rule over $[x_1, x_2]$, recursively.

$$\int_{x_0}^{x_2} f(x)dx \approx \underbrace{\frac{\Delta x}{3}[y_0 + 4y_1 + y_2]}_{\widetilde{I}} - \underbrace{\frac{(\Delta x)^5}{90} f^{(4)}(\xi)}_{E} \tag{9}$$

3. **Description of the Algorithm**

I used the following algorithm to evaluate the intergals in this homework assignment. This method uses an Simpson's Adaptive Quadature method over the interval $[a, b]$. Let $AdaptiveQuad(F, a, b, Tol)$ represent the function used in my program where F is the intergrand, (a, b) is the interval over which integrated, and Tol is the absolute tolerance. E represents the error.

$$
\begin{aligned}
\widetilde{I} &= Q(F, a, b, Tol) \\
\widetilde{I_{sum}} &= Q(F, a, \frac{b+a}{2}, Tol) + Q(F, \frac{b+a}{2}, b, Tol) \\
E &= \frac{1}{2^r - 1}\left(\widetilde{I_{sum}} - \widetilde{I}\right)
\end{aligned}
$$

(where $r = 4$ because using Simpson's Rule)

If $|E| < Tol$

return $\widetilde{I_{sum}} + E$

Else

$\widetilde{I_{left}} = AdaptiveQuad(F, a, \frac{b+a}{2}, Tol)$

$\widetilde{I_{right}} = AdaptiveQuad(F, \frac{b+a}{2}, b, Tol)$

return $\widetilde{I_{left}} + \widetilde{I_{right}}$

End If

A test case is used to illustrate the method above. It will be illustrated in the results how the subintervals are chosen and adaptation is displayed.

4. **Results**

Rationale for test cases

- A majority of the tests were setup to evaluate to one.
- Each function was chosen to illustrate errors
- Tests 1-3, because the degree is ≤ 3, the derivative become zero and will be exact.
- Test 4, the error will be constant. The error term depends on the fourth derivative, See (9) for forumla for error.
- Test case 5 is the same as inclass lecture, just by a constant factor.
- Test case 7 will ilustrate the adaptation method discussed in the previous section.
- Last case in the part of the assignment, **part(b)**.

Table 1

Functions	Exact value	Adapative Estimate	Simpson's Estimate
$f(x) = 2x$	$1.00E+00$	$1.00E+00$	$1.00E+00$
$f(x) = 6x^2 - 2x$	$1.00E+00$	$1.00E+00$	$1.00E+00$
$f(x) = 8(x - 2x^3) + 2(3x - 1)x$	$1.00E+00$	$1.00E+00$	$1.00E+00$
$f(x) = 5(x^4 + 2x^3 - 3x^2 + x)$	$1.00E+00$	$1.00E+00$	1.04167
$f(x) = \dfrac{1}{\ln(2)(1+x)}$	$1.00E+00$	$1.00E+00$	$1.00E+00$
$f(x) = \dfrac{2e^{2x}+1}{e^2+1}$	$1.00E+00$	$1.00E+00$	$1.00E+00$
$f(x) = \begin{cases} 1 & 0 \leq x \leq 1 \\ e^{x-1} & 1 < x \leq 2 \end{cases}$	e^1	$2.71828175e+000$	$2.57276e+000$
part(b)$\rightarrow$ (5)	0.341342151	0.64525	

Table 2

Functions	Adapative Estimate Error	Simpson's Estimate Error	Iterations
$f(x) = 2x$	$0.00E+00$	$0.00E+00$	1
$f(x) = 6x^2 - 2x$	$0.00E+00$	$0.00E+00$	1
$f(x) = 8(x - 2x^3) + 2(3x - 1)x$	$0.00E+00$	$0.00E+00$	1
$f(x) = 5(x^4 + 2x^3 - 3x^2 + x)$	$0.00E+00$	-0.0416666	31
$f(x) = f(x) = \dfrac{1}{\ln(2)(1+x)}$	$0.00E+00$	-0.00187159	17
$f(x) = \dfrac{2e^{2x}+1}{e^2+1}$	$0.00E+00$	-0.00377536	17
$f(x) = \begin{cases} 1 & 0 \leq x \leq 1 \\ e^{x-1} & 1 < x \leq 2 \end{cases}$	$-1.71828e+000$	$-1.57276e+000$	17
part(b)$\rightarrow$ (5)			5

From the Simpson's Rule, there exists a formula to derive the error for each approximation for the integral. I will calculate the error for each test case over a subinterval $[x_{i-1}, x_i]$. In each test case the fourth derivative exists.

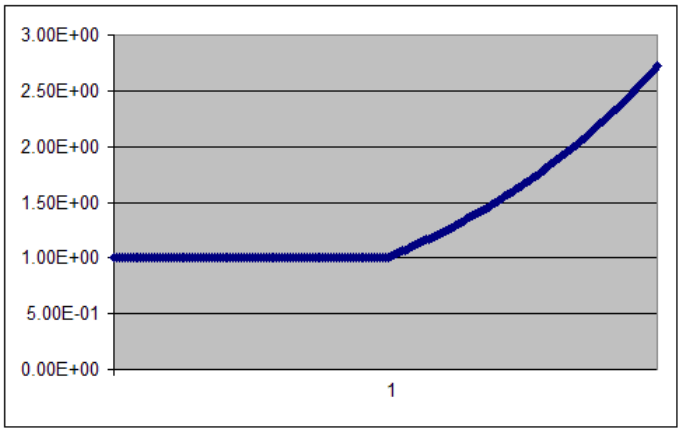
$$E = -\frac{(\Delta x)^5}{90} f^{(4)}(\xi) \tag{10}$$

Table 3

Functions	Minimum Error	Maximum Error
$f(x) = 2x$	0	0
$f(x) = 6x^2 - 2x$	0	0
$f(x) = 8(x - 2x^3) + 2(3x - 1)x$	0	0
$f(x) = 5(x^4 + 2x^3 - 3x^2 + x)$	$-\frac{1}{18}$	$-\frac{1}{18}$
$f(x) = \frac{1}{\ln(2)(1+x)}$	$-2.02e-4$	$-5.49e-4$

Graph: Test Function 7: Ilustrates the adaptation.

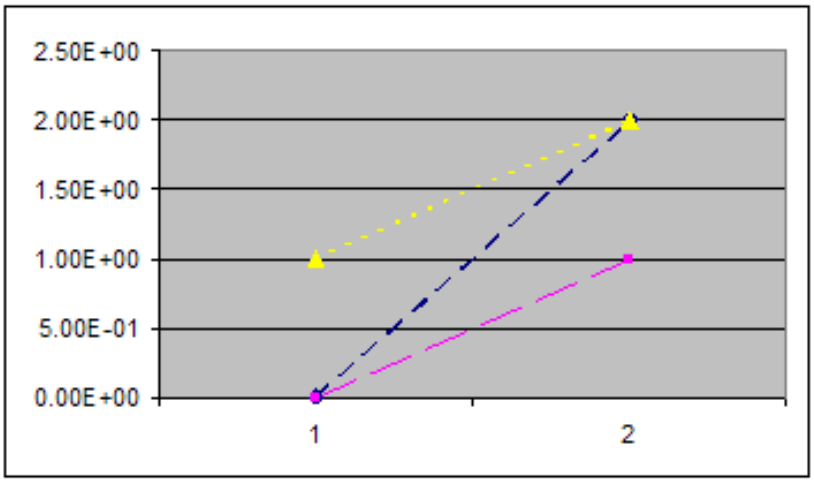
$$f(x) = \begin{cases} 1 & 0 \le x \le 1 \\ e^{x-1} & 1 < x \le 2 \end{cases}$$



The next few graphs illustrate how the program adapts to the function. It will divide into subintervals and check the error term for each subinterval. If one side satisfies the error, the program will divide the other subinterval. From the table and graphs below, one can see that the program will spend time in the interval [1,2], because between [0,1] I set up the function to be constant.

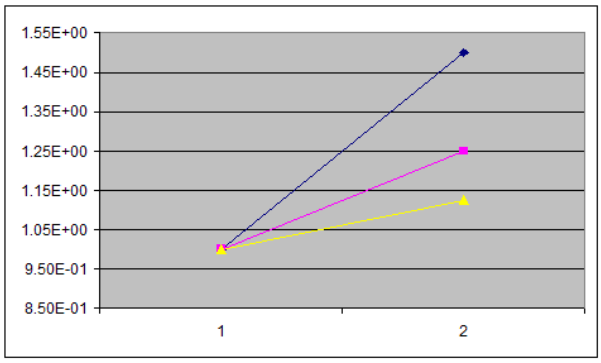
Point a	Point b	
0.00E+00	2.00E+00	→ Interval [0,2]
0.00E+00	1.00E+00	→ Interval [0,1]
1.00E+00	2.00E+00	→ Interval [1,2]
1.00E+00	1.50E+00	→ Interval [1,1.5]
1.00E+00	1.25E+00	→ Interval [1,1.25]
1.00E+00	1.13E+00	→ Interval [1,1.13]
1.13E+00	1.25E+00	etc..
1.25E+00	1.50E+00	
1.25E+00	1.38E+00	
1.38E+00	1.50E+00	
1.50E+00	2.00E+00	
1.50E+00	1.75E+00	
1.50E+00	1.63E+00	
1.63E+00	1.75E+00	
1.75E+00	2.00E+00	
1.75E+00	1.88E+00	
1.88E+00	2.00E+00	

Graph 2: 1st 3iterations

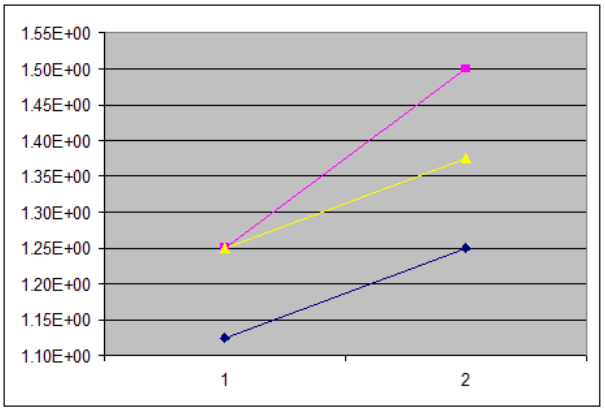


From the graph above, the first interval is from [0,2], then from [0,1], and finally [1,2]. From graph 3 below, the series is broken down into [1,1.5], then from [1,1.25], and finally [1,1.13], etc. From above, can conclude that the program illustrates the adaptative quadrature.

Graph 3: 2nd of the 3iterations



Graph 4: 3rd of the 3iterations



5. **Conclusion**

Tables Results

- Table 1: Compares the numerical results with the exact results. The adapative quadrature give the integration for each test case.
- The relative error was set to $5\,X\,10^{-6}$. This gives six significant digits for each case.
- Table 2: illustrates the error in each calculation using adapative and Simpson's rule.
- Table 2: Due to the Error formula depending on the fourth derivative, all polynomials with degree less than must be integrated once. Counting the number of iterations in the program is illustrated
- Table 3: Since we have a formula for error in Simpson's rule, I checked if my error lied within the theoretical error which can be calculated using the formula or upper and lower bounds. My errors lied within the bounds for each case.
- Table 3: One test function was setup to be a constant because it was of fourth degree. Here, we can calculate the error exactly because $f^{(4)}(\xi)$ is constant. Both the numerical and error in theory were $-\frac{1}{18}$.
- Graph: Test function 7 was setup to test how the function adapted on each subintervals. I setup the function such that it was constant on one piece and increasing on the piece. I expected that the piece that was not constant needed to be refined. As shown in the graph, this interval was refined into 4 subintervals.

Part(b)

From Table 1, $P(1)$, can be determined using the program on (5)

$$\begin{aligned} P(1) &= \frac{1}{2} + \frac{1}{\sqrt{2\pi}} \int_0^1 e^{\frac{-x^2}{2}} dt \\ &= \frac{1}{2} + \underbrace{0.341342151}_{table1} \\ &= 0.841342151 \end{aligned}$$

- I set my relative error to $5\,X\,10^{-6}$, therefore I can obtain at least 6 sigfigs.

Part(c-i)

- Using the adaptive quadrature, I obtained the following the value of the call: 98.567138.
- To get at least three significant digits, I would have to set the relative error tolerance to $5\,X\,10^{-3}$. However, I wrote my adapative quadrature to use absolute tolerance. Therefore, need to convert the relative tolerance. Because the call price is of the order 100, the absolute tolerance $5\,X\,10^{-1}$ will achieve the required 3 sigfigs.
- I set the absolute error tolerance to $5\,X\,10^{-7}$ to obtain at least 6 sigfigs.

Part(c-ii)

- Using methods of finding zeros, I obtained the following the value of the implied volatilty:
  Newton Method: 7.6089457%
  Bisection Method: 7.6089583%
- Using Newton's method required finding the derivative of the call price with respect to volatility.

$$\frac{\partial C(S,t)}{\partial \sigma} = S(\sqrt{T-t})P(d_1) \tag{11}$$

- I found how sensitive the problem was to rounding errors. Tried to see if a small change in the input caused a large change in the solution. I derived the condition number, using $F(\sigma)$, from (7). Let $C = F(\sigma)$ and $\sigma = f^{-1}(C)$

$$K = \left|\frac{C}{\sigma}\right| \left|\frac{dF^{-1}(C)}{dC}\right| \tag{12}$$

$$= \left|\frac{C}{\sigma}\right| \left|\frac{1}{F'(C)}\right| \tag{13}$$

Using the above equation, I was able to determine that the condition number is 63.45. The result for the implied volatilty will lost 2 sigfigs becuase of roundoff error in the call price. In IEEE single precision, the implied volatilty has 4 sigfigs.

APPENDIX: 1Main file

```
// HW5.cpp : Defines the entry point for the console application.
//


#include "ostream.h"
#include "math.h"
#include "stdio.h"
#include "funcs.h"
#include "adapsimps.h"


int main()
{
float a,b;
float Tol, Tol_simp;
int iter, iter_simp;
float integral,integ_simp;

FILE *file1, *file6;
file1= fopen ("integral.txt", "w");
file6= fopen ("graph.txt", "w");

Tol=1.e-7;
Tol_simp=1.e10;

a=0.0;
b=1.e0;

iter=0;
iter_simp=0;
integral = ADAP_QUADSIMP(f1,a,b,Tol,iter);
integ_simp = ADAP_QUADSIMP(f1,a,b,Tol_simp,iter_simp);
fprintf(file1,"%12.5e %14.8e %12.5e %12.5e %12.5e %4u \n",
    1.0,integral,integ_simp,1-integral,1-integ_simp,iter);
iter=0;
iter_simp=0;
integral = ADAP_QUADSIMP(f2,a,b,Tol,iter);
integ_simp = ADAP_QUADSIMP(f2,a,b,Tol_simp,iter_simp);
fprintf(file1,"%12.5e %14.8e %12.5e %12.5e %12.5e %4u \n",
    1.0,integral,integ_simp,1-integral,1-integ_simp,iter);

iter=0;
iter_simp=0;
integral = ADAP_QUADSIMP(f3,a,b,Tol,iter);
integ_simp = ADAP_QUADSIMP(f3,a,b,Tol_simp,iter_simp);
fprintf(file1,"%12.5e %14.8e %12.5e %12.5e %12.5e %4u \n",
    1.0,integral,integ_simp,1-integral,1-integ_simp,iter);

iter=0;
iter_simp=0;
integral = ADAP_QUADSIMP(f4,a,b,Tol,iter);
```

```
integ_simp = ADAP_QUADSIMP(f4,a,b,Tol_simp,iter_simp);
fprintf(file1,"%12.5e %14.8e %12.5e %12.5e %12.5e %4u \n",
    1.0,integral,integ_simp,1-integral,1-integ_simp,iter);

iter=0;
iter_simp=0;
integral = ADAP_QUADSIMP(f5,a,b,Tol,iter);
integ_simp = ADAP_QUADSIMP(f5,a,b,Tol_simp,iter_simp);
fprintf(file1,"%12.5e %14.8e %12.5e %12.5e %12.5e %4u \n",
    1.0,integral,integ_simp,1-integral,1-integ_simp,iter);

iter=0;
iter_simp=0;
integral = ADAP_QUADSIMP(f6,a,b,Tol,iter);
integ_simp = ADAP_QUADSIMP(f6,a,b,Tol_simp,iter_simp);
fprintf(file1,"%12.5e %14.8e %12.5e %12.5e %12.5e %4u \n",
    1.0,integral,integ_simp,1-integral,1-integ_simp,iter);

iter=0;
iter_simp=0;
integral = ADAP_QUADSIMP_2(f7,a,2,Tol,iter, file6);
integ_simp = ADAP_QUADSIMP(f7,a,2,Tol_simp,iter_simp);
fprintf(file1,"%12.5e %14.8e %12.5e %12.5e %12.5e %4u \n",
    1.0,integral,integ_simp,1-integral,1-integ_simp,iter);
iter=0;
iter_simp=0;
integral=ADAP_QUADSIMP(fstdnorm,0.0,1.0,Tol,iter);
integ_simp=ADAP_QUADSIMP(fstdnorm,-15.0,1.0,Tol_simp,iter_simp);
fprintf(file1,"%12.5e %14.8e %12.5e %12.5e %12.5e %4u \n",
    1.0,integral,integ_simp,1-integral,1-integ_simp,iter);
return 0;
}
```

APPENDIX: 2 Problem 5(c) code

```
//HW5 part c.  Compute the vale of the call

#include "ostream.h"
#include "math.h"
#include "stdio.h"
#include "funcs.h"
#include "adapsimps.h"

//function
float call(float T, float r, float sigma, float E, float S);


int main()
{

float valcall;
float T, t,r,sigma,E,S;

FILE *file2;
file2 = fopen ("call.txt", "w");


T=54.0/365.0; //Time of expiry
t=0;
r=0.0675; //interest rate
sigma = 0.135; //volatility
E=3425.0; //Exercise price
S = 3441.0; //Asset Price

valcall = call(T,r,sigma,E,S);
fprintf(file2, "Value of Call: %10.7e \n",valcall);

cout << valcall << endl;

return 0;
}

float call(float T, float r, float sigma, float E, float S)
{
float I, d1,d2,d1_int, d2_int;
float lower;
float Tol;
int iter;
int t=0;
Tol=1.e-7;
lower=-10;

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));

d1_int = ADAP_QUADSIMP(fstdnorm,lower,d1,Tol,iter);
d2_int = ADAP_QUADSIMP(fstdnorm,lower,d2,Tol,iter);


I = S*d1_int-E*exp(-1.0*r*T)*d2_int;

return I;
}
```

APPENDIX: 3 Problem 5(d) code

```
//HW5 part d.  Compute the implied volatility

//include rootprog.h


#include "ostream.h"
#include "math.h"
#include "stdio.h"
#include "funcs.h"
#include "adapsimps.h"
#include "rootprog.h"

//functions
float call(float T, float r, float sigma, float E, float S);
float func(float sigma);
float funcd(float sigma);


int main()
{
float Vol_bisect;
float Vol_newton;
float startval;
float Vol_combo;
float imp_vol_n;
float sense;
float Tol;
int iter;

float condition;


Tol=1.e-7;
iter=0;

FILE *file3, *file4;


file3=fopen("vol.txt","w");
file4=fopen("vol2.txt","w");

float a = 1.e-5;
float b = 1.0;
float AbsTol = 0.0;
Vol_bisect = bisect(func,a,b,AbsTol, Tol, iter);

startval= 0.1;
Vol_newton = newt(func,funcd,startval,AbsTol, Tol, iter);
cout << Vol_bisect << endl;
cout << Vol_newton << endl;
```

```
fprintf(file3, "Bisection: %10.7e \nNewton: %10.7e \nRel Err:%10.7e \n",Vol_bisect,Vol_newton,fabs(



//compute the sensitivity
//look for small change in data
//causes a large change in soln

condition = (Vol_newton/3441.0)*funcd(Vol_newton);
condition = 1.0/condition;

cout << condition << endl;

fprintf(file4, "Condition: %10.7e \nSPcond: %10.7e \nDPcond: %10.7e \n",condition, 1.19e-7*conditio


return 0;
}

float call(float T, float r, float sigma, float E, float S)
{
float I, d1,d2,d1_int, d2_int;
float lower;
float Tol;
int iter;
int t=0;
Tol=1.e-7;
lower=-10;

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));

d1_int = ADAP_QUADSIMP(fstdnorm,lower,d1,Tol,iter);
d2_int = ADAP_QUADSIMP(fstdnorm,lower,d2,Tol,iter);


I = S*d1_int-E*exp(-1.0*r*T)*d2_int;

return I;
}

float func(float sigma)
{
float I, C, T, r, E, S;
C=94.0; //Value of call
T=117.0/365.0; //time of expiry
r=0.0675; //interst rate
E=3475.0; //Exercise price
S=3461.0; //Asset price
```

```
I = call(T,r,sigma,E,S) - C;

return I;

}

float funcd(float sigma)
{
float I,d1, C, T, r, E, S;

C=94.0; //Value of call
T=117.0/365.0; //time of expiry
r=0.0675; //interst rate
E=3475.0; //Exercise price
S=3461.0; //Asset price
float t= 0.0;


d1= log(S/E) + ((r+pow(sigma,2)/2.0)*(T-t));
d1= d1/(sigma*sqrt(T-t));

I = S*sqrt(T-t)*fstdnorm(d1);

return I;
}
```

APPENDIX: 4 Graph code

```
// graph to check nodes
//

//#include "stdafx.h"
#include "ostream.h"
#include "math.h"
#include "stdio.h"
#include "funcs.h"
#include "adapsimps.h"



void main()
{
float a,b;
int iter;
float integral,Tol;

Tol=1.e-7;
FILE *file5, *file6;

file5= fopen("graph.txt", "w");
file6= fopen ("graph2.txt", "w");

a=0.0;
b=1.e0;
float x,y;

for (int i=0; i<=200; i++)
{
x=i/100.0;
y=f7(x);
fprintf(file5,"%10.7e %10.7e \n",x,y);
}

iter=0;
integral = ADAP_QUADSIMP_2(f7,a,2.0,Tol,iter, file6);
}
```

APPENDIX: 5 Simpsons and Adapative Quadrature.h

```
# 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;
}
}
```

APPENDIX: 6 Functions

```
#include "math.h"


// Test Case 1: First-order Function
float f1(float x)
{
float f;
f=2.0*x;
return f;

}


// Test Case 2: Second-order Function

float f2(float x)
{
float f;
f=2.0*(3.0*x-1.0)*x;
return f;

}


// Test Case 3: Third-order Function

float f3(float x)
{
float f;
f=8.0*(x-2.0*pow(x,3))+2.0*(3.0*x-1.0)*x;
return f;

}


// Test Case 4: Fourth-order Function

float f4(float x)
{
float f;
f=5.0*(pow(x,4)+2.0*pow(x,3)-3.0*pow(x,2)+x);
return f;

}

// Test Case 5: inclass ln(2)/(1+x) Function

float f5(float x)
{
float f;
f=1.0/(log(2)*(1.0+x));
```

```
return f;

}

// Test Case 6: Exponential Function

float f6(float x)
{
float f;
f=2.0*(exp(2*x)+1)/(exp(2)+1.0);
return f;

}


// Test Case 7: Test Function for Adaptive Quadrature

float f7(float x)
{
float f;
if (x>=0.0 && x<=1.0)
f=1.0;
else if (x>1.0 && x<=2.0)
f=exp(x-1);
// else
// f=0;


return f;

}


// Gaussian probabilty function N(0,1)

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;

}
```

APPENDIX: 7 Bisection and Newton method.h

```
# include "stdio.h"


//Need to derive a function that returns
//the sign of a number,
// number is postive = 1
// number is negative = -1
//number is zero = 0
//Not needed in FORTRAN


int sign (float x)
{
int sign;
if (x > 0.0)
sign=1;
else if (x<0.0)
sign=-1;
else
sign=0;
return sign;
}

//Bisection algothrim (as noted in lecture)



float bisect(float (*f)(float), float a, float b,
float AbsTol, float RelTol, int & iteriations)

{
int N; //iteriations stop here
int i;
int beta=20;
float p;
float fa;
float fp;


N = fabs(b-a)/(AbsTol+RelTol*(a+b)/2.0);
N = log10(N)/log10(2)*beta;

for (i=1;i<=N;i++)
{

p=a+(b-a)/2.0;

if (((b-a)/2.0 <= AbsTol+fabs(p)*RelTol) )
{
iteriations=i-1;
break;
```

```
}

fa=f(a);
fp=f(p);

if (sign(fa)*sign(fp)>0.0)
a=p;
else
b=p;

}

return p;

}


//Newton-Rapshon Method.  algothrim from lecture


float newt (float (*f)(float), float (*fp)(float), float startvalue,
  float AbsTol, float RelTol, int & iteriation)

{
int N;
int i;
float p;
float dp;

N = 1e5;  //iteriations stop here
p=startvalue;

for (i=1;i<=N;i++)
{

dp=-f(p)/fp(p);
if ( fabs(dp) < AbsTol+fabs(p)*RelTol )
{
iteriation=i-1;
break;
}
p=p+dp;

}

return p;

}

//Combo: (besection and Netwon)  Within solution of (alpha) = .01

float combo (float (*f)(float), float (*fp)(float), float a, float b,
  float AbsTol, float RelTol, int &iter)
```

```
{
float first_RelTol=0.01;
float first_sol;
int first_iter;

first_sol=bisect(f,a,b,AbsTol,first_RelTol,first_iter);
first_sol=newt(f,fp,first_sol,AbsTol,RelTol,iter);
iter=iter+first_iter;

return first_sol;

}
```