MA 374 – Financial Engineering II

Lab 2
Report

Name: Vibhanshu
Roll No. : 120123049

1).

```
#include<iostream>
#include<cstdio>
#include<cstdlib>
#include<cmath>
using namespace std;

float S0=100;
float K=100;
float T=1;
float r=0.08;
float sigma=0.2;
int M=100;

float max(float a,float b)
{
	if( a>b ) return a;
	else return b;
}

float u(float delta_t)
{
	return exp(sigma*sqrt(delta_t));
}

float u2(float delta_t)
{
	return exp( sigma*sqrt(delta_t) + (r-0.5*sigma*sigma )*delta_t );
}

float d(float delta_t)
{
	return exp(-sigma*sqrt(delta_t));
}

float d2(float delta_t)
{
	return exp( -sigma*sqrt(delta_t) + ( r-0.5*sigma*sigma )*delta_t );
}


float discount_rate(float t)
{
	return exp(r*t);
```

```
}

float p(float delta_t)
{
	return ( exp(r*delta_t) - d(delta_t) )/ ( u(delta_t) -d(delta_t) ) ;
}

float q(float delta_t)
{
	return ( u(delta_t) - exp(r*delta_t) )/ ( u(delta_t) - d(delta_t) );
}

/// Calculated using u2,d2
float p2(float delta_t)
{
	return ( exp(r*delta_t) - d2(delta_t) )/ ( u2(delta_t) -d2(delta_t) ) ;
}

float q2(float delta_t)
{
	return ( u2(delta_t) - exp(r*delta_t) )/ ( u2(delta_t) - d2(delta_t) );
}

// This is for the call option(recursive)
float get_C(float t,float S,float delta_t)    // returns the price of the option at time t if
the price os stock is S )
{
	if(t>=T)
	{
		if(S>=K) return (S-K);
		else return 0;
	}
	else
	{
		float t1=get_C(t+delta_t , S*u(delta_t) , delta_t);
		float t2=get_C(t+delta_t , S*d(delta_t) , delta_t);
		return  ( p(delta_t)*t1 +q(delta_t)*t2)/discount_rate(delta_t) ;
	}
}

// This is for the put option(recursive)
float get_P(float t,float S,float delta_t)    // returns the price of the option at time t if
the price os stock is S )
{
	if(t>=T)
```

```
        {
                if(S<=K) return (K-S);
                else return 0;
        }
        else
        {
                float t1=get_P(t+delta_t , S*u(delta_t) , delta_t);
                float t2=get_P(t+delta_t , S*d(delta_t) , delta_t);
                return  ( p(delta_t)*t1 +q(delta_t)*t2)/discount_rate(delta_t) ;
        }
}

// This is the fast version of the algorithm CALL
float get_C_fast(float t,float S,float delta_t)
{
        float U=u(delta_t);
        float D=d(delta_t);
        float P=p(delta_t);
        float Q=q(delta_t);
        float R=discount_rate(delta_t);
        float* SS=new float[M+2];
        float* value=new float[M+2];
        int i,j,k;
        for(i=0;i<=M;i++)
                SS[i]=S0*(pow(U,(float)(M-i)))*(pow(D,(float)i));
        for(i=0;i<=M;i++)
                value[i]=max(SS[i]-K,0);
        for(j=M; j>=0; j--)
        {
                for(k=0; k<j; k++)
                        value[k]= (P*value[k+1]+Q*value[k])/R;
        }
        return value[0];
}

// This is the fast version of the algorithm PUT
float get_P_fast(float t,float S,float delta_t)
{
        float U=u(delta_t);
        float D=d(delta_t);
        float P=p(delta_t);
        float Q=p(delta_t);
        float R=discount_rate(delta_t);
        float* SS=new float[M+2];
        float* value=new float[M+2];
```

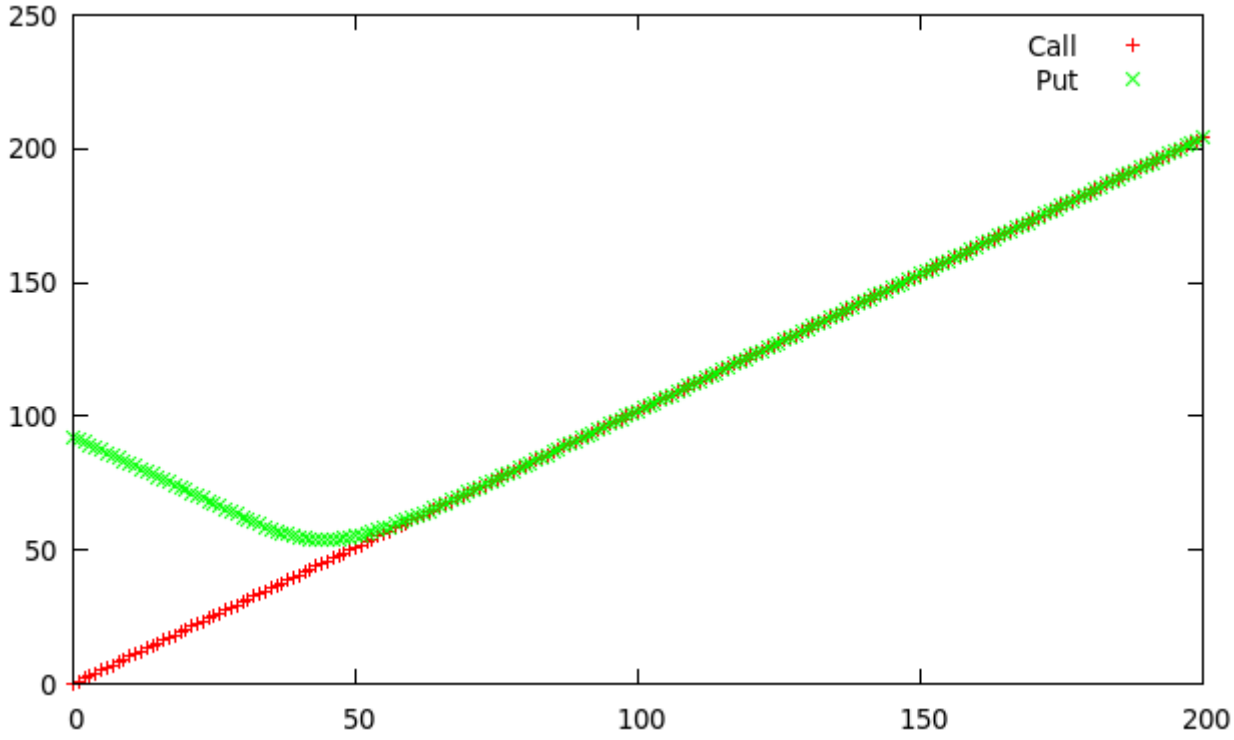
```
	int i,j,k;
	for(i=0;i<=M;i++)
		SS[i]=S0*(pow(U,(float)(M-i)))*(pow(D,(float)i));
	for(i=0;i<=M;i++)
		value[i]=max(K-SS[i],0);
	for(j=M; j>=0; j--)
	{
		for(k=0; k<j; k++)
			value[k]= (P*value[k+1]+Q*value[k])/R;
	}
	return value[0];
}

//# ================================
// This is the fast version of the algorithm CALL
float get_C_fast1(float t,float S,float delta_t)
{
	float U=u2(delta_t);
	float D=d2(delta_t);
	float P=p2(delta_t);
	float Q=q2(delta_t);
	float R=discount_rate(delta_t);
	float* SS=new float[M+2];
	float* value=new float[M+2];
	int i,j,k;
	for(i=0;i<=M;i++)
		SS[i]=S0*(pow(U,(float)(M-i)))*(pow(D,(float)i));
	for(i=0;i<=M;i++)
		value[i]=max(SS[i]-K,0);
	for(j=M; j>=0; j--)
	{
		for(i=0;i<j;i++)
			SS[i]=S0*(pow(U,(float)(j-i)))*(pow(D,(float)i));
		for(k=0; k<j; k++)
			value[k]= max(SS[k],(P*value[k+1]+Q*value[k])/R);
	}
	return value[0];
}

// This is the fast version of the algorithm PUT
float get_P_fast1(float t,float S,float delta_t)
{
	float U=u2(delta_t);
	float D=d2(delta_t);
	float P=p2(delta_t);
```

```
	float Q=p2(delta_t);
	float R=discount_rate(delta_t);
	float* SS=new float[M+2];
	float* value=new float[M+2];
	int i,j,k;
	for(i=0;i<=M;i++)
		SS[i]=S0*(pow(U,(float)(M-i)))*(pow(D,(float)i));
	for(i=0;i<=M;i++)
		value[i]=max(K-SS[i],0);
	for(j=M; j>=0; j--)
	{
		for(i=0;i<j;i++)
			SS[i]=S0*(pow(U,(float)(j-i)))*(pow(D,(float)i));
		for(k=0; k<j; k++)
			value[k]= max(SS[k],(P*value[k+1]+Q*value[k])/R);
	}
	return value[0];
}




FILE* ptr;


int main()
{
	float delta_t;

	// Part (a)
	S0=100;
	K=100;
	T=1;
	M=100;
	r=0.08;
	sigma=0.2;
	delta_t=T/M;
	ptr=fopen("set1_a.dat","w");
	for(S0 = 0; S0<=200; S0+=1 )
	{ fprintf(ptr,"%f\t%f\t
%f\n",S0,get_C_fast1(0,S0,delta_t),get_P_fast1(0,S0,delta_t)); }
	fclose(ptr);

	// Part(b)
```

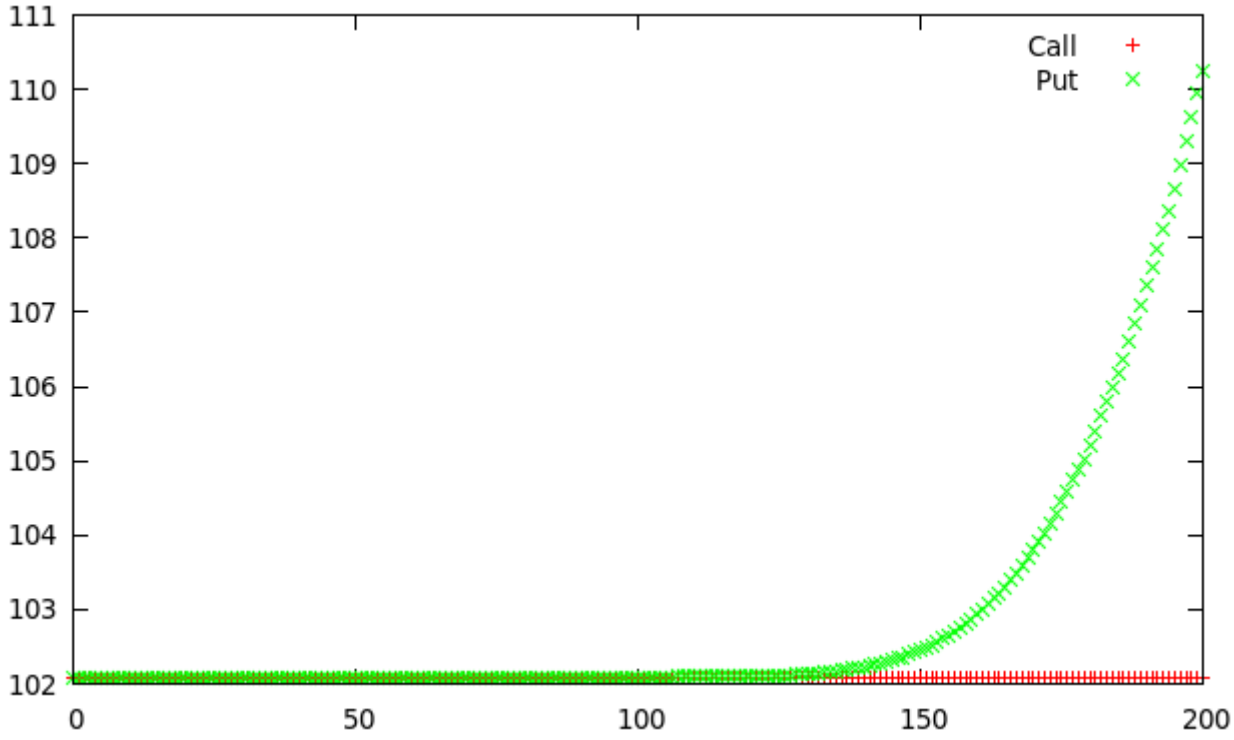
```
	S0=100;
	K=100; T=1;
	M=100;
	r=0.08;
	sigma=0.2;
	delta_t=T/M;
	ptr=fopen("set1_b.dat","w");
	for(K = 0; K<=200; K+=1 )
	{ fprintf(ptr,"%f\t%f\t
%f\n",K,get_C_fast1(0,S0,delta_t),get_P_fast1(0,S0,delta_t)); }
	fclose(ptr);


	// Part(c)
	S0=100;
	K=100; T=1;
	M=100;
	r=0.08;
	sigma=0.2;
	delta_t=T/M;
	ptr=fopen("set1_c.dat","w");
	for(r = 0; r<=0.2; r+=0.01 )
	{ fprintf(ptr,"%f\t%f\t
%f\n",r,get_C_fast1(0,S0,delta_t),get_P_fast1(0,S0,delta_t)); }
	fclose(ptr);


	// Part(d)
	S0=100;
	K=100; T=1;
	M=100;
	r=0.08;
	sigma=0.2;
	delta_t=T/M;
	ptr=fopen("set1_d.dat","w");
	for(sigma = 0; sigma<=0.5; sigma+=0.01 )
	{ fprintf(ptr,"%f\t%f\t
%f\n",sigma,get_C_fast1(0,S0,delta_t),get_P_fast1(0,S0,delta_t)); }
	fclose(ptr);


	// Part(e)
	S0=100;
	K=95; T=1;
	M=100;
```

```
	r=0.08;
	sigma=0.2;
	ptr=fopen("set1_e_k1.dat","w");
	for(M = 1; M<=200; M+=1 )
	{
		delta_t=T/M;
		fprintf(ptr,"%d\t%f\t
%f\n",M,get_C_fast1(0,S0,delta_t),get_P_fast1(0,S0,delta_t));
	}
	fclose(ptr);


	S0=100;
	K=100; T=1;
	M=100;
	r=0.08;
	sigma=0.2;
	ptr=fopen("set1_e_k2.dat","w");
	for(M = 1; M<=200; M+=1 )
	{
		delta_t=T/M;
		fprintf(ptr,"%d\t%f\t
%f\n",M,get_C_fast1(0,S0,delta_t),get_P_fast1(0,S0,delta_t));
	}
	fclose(ptr);

	S0=100;
	K=105; T=1;
	M=100;
	r=0.08;
	sigma=0.2;
	ptr=fopen("set1_e_k3.dat","w");
	for(M = 1; M<=200; M+=1 )
	{
		delta_t=T/M;
		fprintf(ptr,"%d\t%f\t
%f\n",M,get_C_fast1(0,S0,delta_t),get_P_fast1(0,S0,delta_t));
	}
	fclose(ptr);

	return 0;
}
```

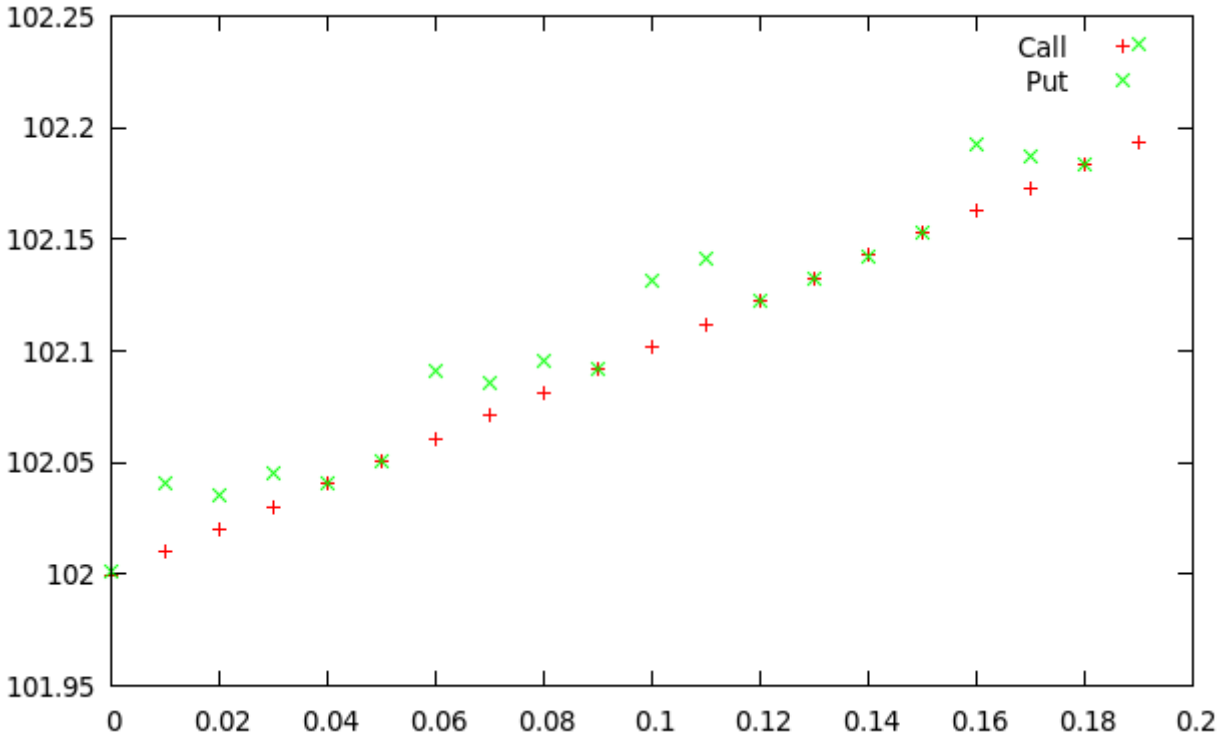
a).


Call
Put
250
200
150
100
50
0
0
50
100
150
200

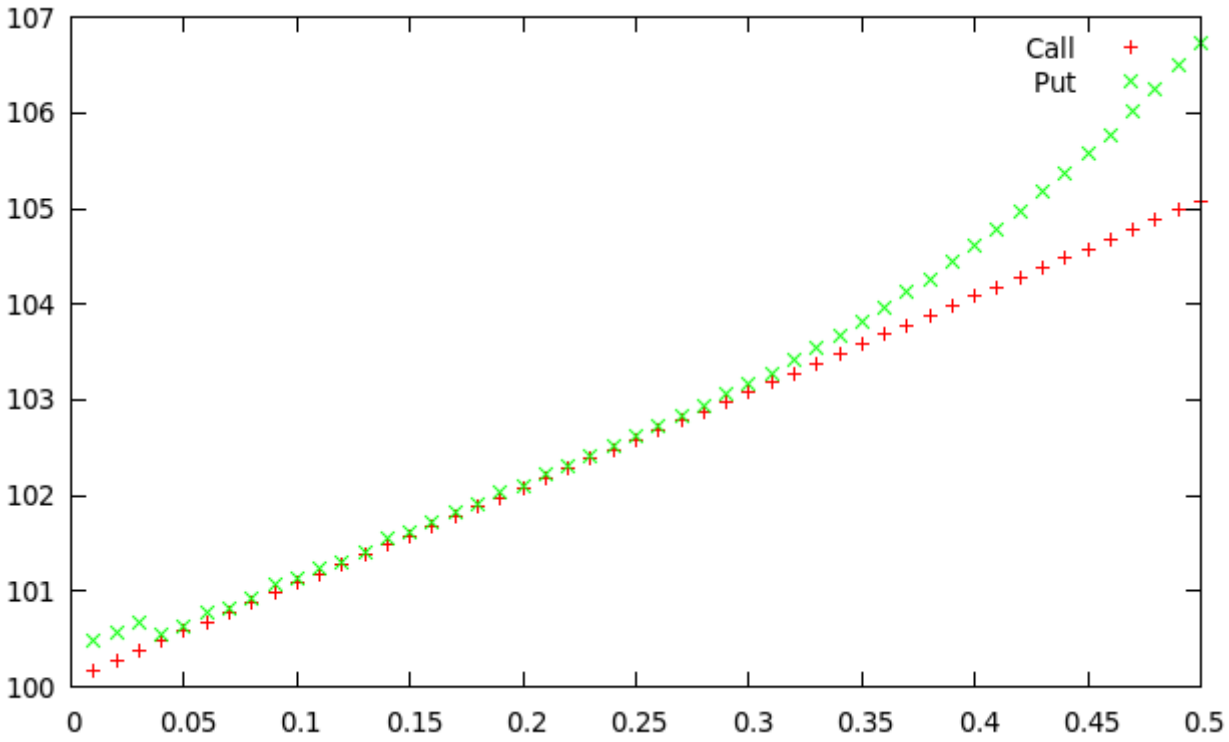
b).


Call
Put
111
110
109
108
107
106
105
104
103
102
0
50
100
150
200

c).

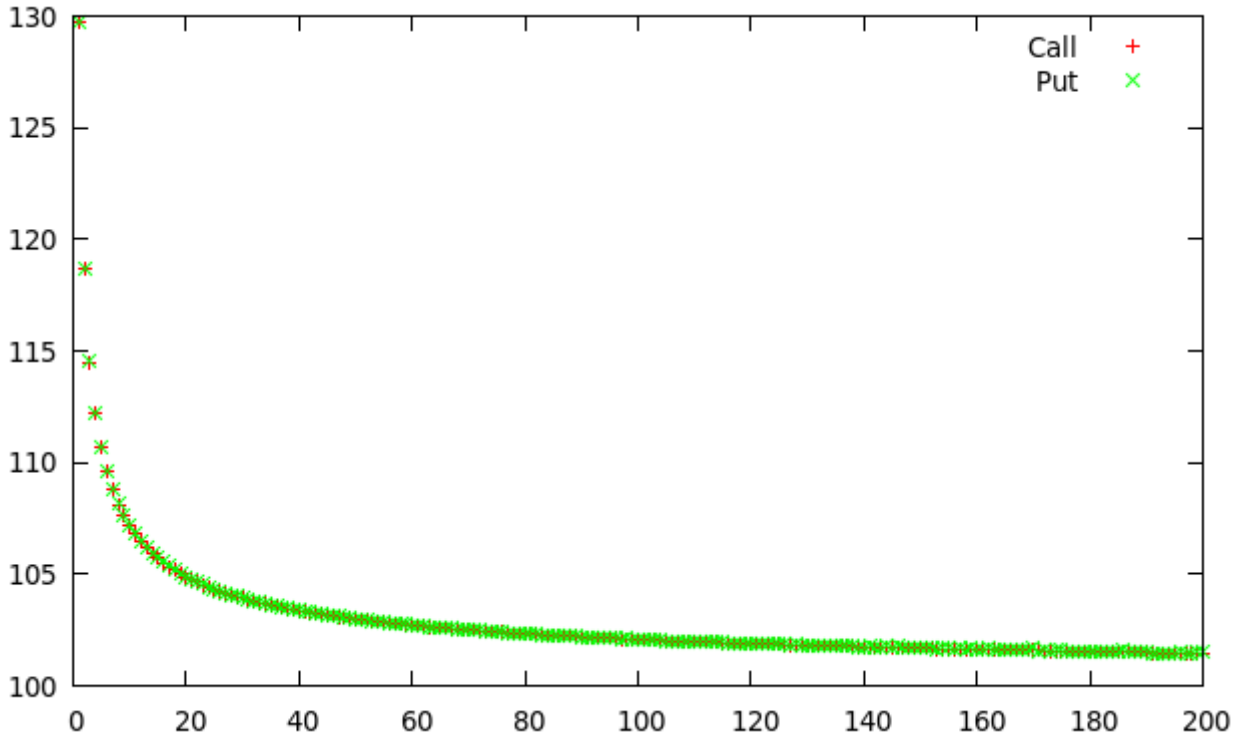

Call
Put
102.25
102.2
102.15
102.1
102.05
102
101.95
0
0.02
0.04
0.06
0.08
0.1
0.12
0.14
0.16
0.18
0.2

d).

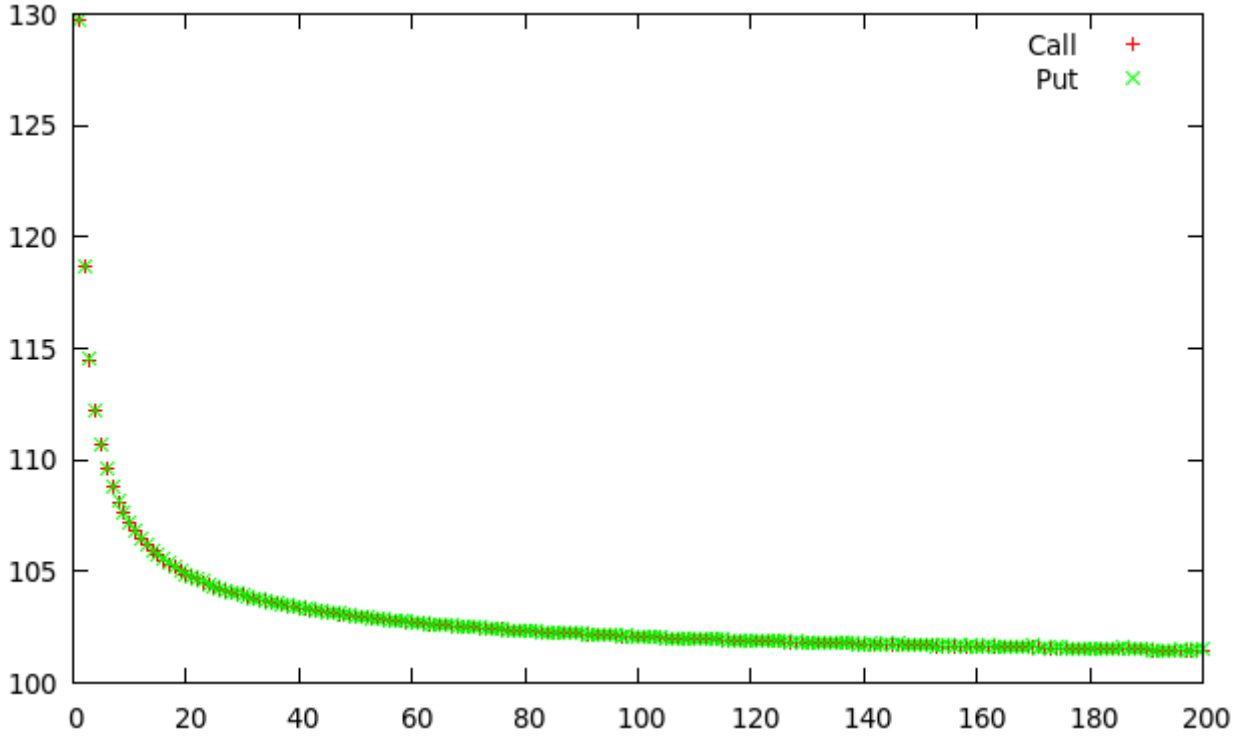

Call
Put
107
106
105
104
103
102
101
100
0
0.05
0.1
0.15
0.2
0.25
0.3
0.35
0.4
0.45
0.5

e).

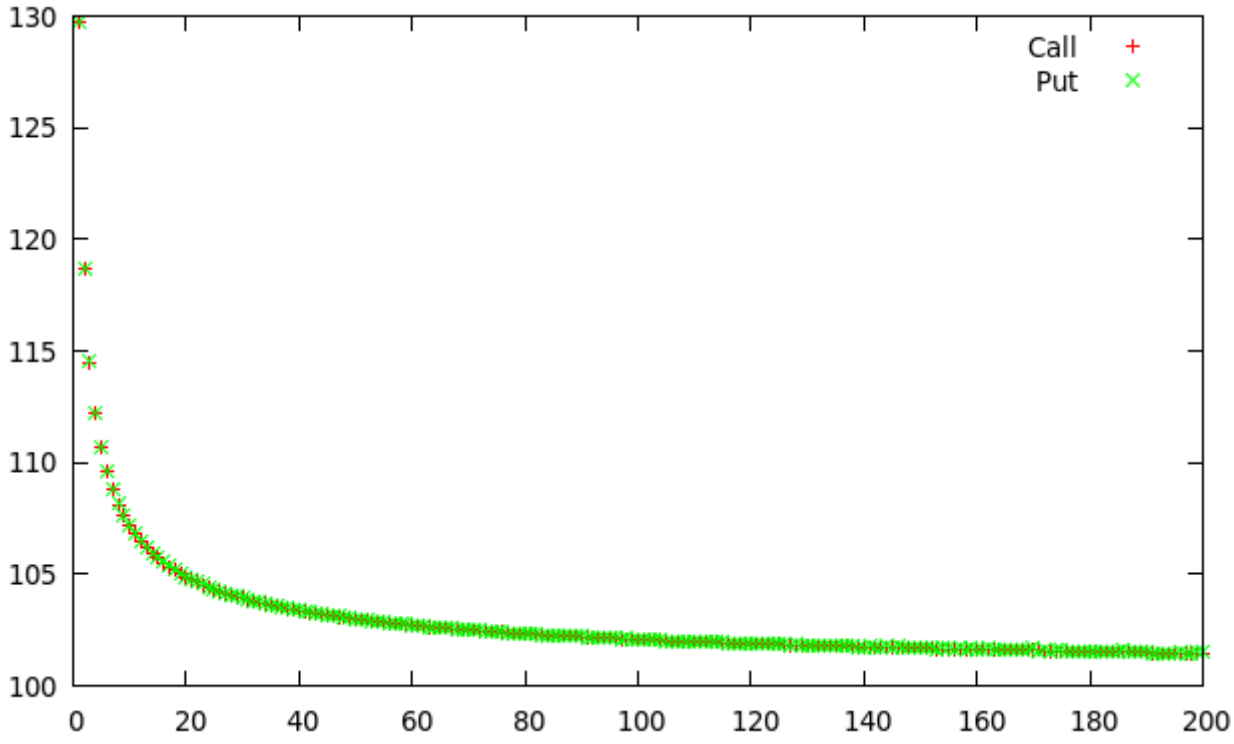
K= 95


Call
Put
130
125
120
115
110
105
100
0
20
40
60
80
100
120
140
160
180
200

K= 100


Call
Put
130
125
120
115
110
105
100
0
20
40
60
80
100
120
140
160
180
200

K = 105


Call
Put
130
125
120
115
110
105
100
0
20
40
60
80
100
120
140
160
180
200

2).

Part (a) and (b):

```
#include<iostream>
#include<cstdio>
#include<cstdlib>
#include<cmath>
using namespace std;

float S0=100;
float K=100;
float T=1;
float r=0.08;
float sigma=0.2;
int M=100;

float max(float a,float b)
{
	if( a>b ) return a;
	else return b;
}

float u(float delta_t)
{
	return exp(sigma*sqrt(delta_t));
}

float u2(float delta_t)
{
	return exp( sigma*sqrt(delta_t) + (r-0.5*sigma*sigma )*delta_t );
}

float d(float delta_t)
{
	return exp(-sigma*sqrt(delta_t));
}

float d2(float delta_t)
{
	return exp( -sigma*sqrt(delta_t) + ( r-0.5*sigma*sigma )*delta_t );
}


float discount_rate(float t)
```

```
{
	return exp(r*t);
}

float p(float delta_t)
{
	return ( exp(r*delta_t) - d(delta_t) )/ ( u(delta_t) -d(delta_t) ) ;
}

float q(float delta_t)
{
	return ( u(delta_t) - exp(r*delta_t) )/ ( u(delta_t) - d(delta_t) );
}

/// Calculated using u2,d2
float p2(float delta_t)
{
	return ( exp(r*delta_t) - d2(delta_t) )/ ( u2(delta_t) -d2(delta_t) ) ;
}

float q2(float delta_t)
{
	return ( u2(delta_t) - exp(r*delta_t) )/ ( u2(delta_t) - d2(delta_t) );
}

// This is for the call option(recursive)
// maxpath -> it is used to store the max value until now
float get_lookback(float t,float S,float delta_t,float mx_path)    // returns the price of
the option at time t if the price os stock is S )
{
	if(t>=T)
	{
		return (mx_path - S);
	}
	else
	{
		float t1=get_lookback(t+delta_t , S*u2(delta_t) , delta_t,
max( S*u2(delta_t), mx_path) );
		float t2=get_lookback(t+delta_t , S*d2(delta_t) , delta_t,
max( S*d2(delta_t), mx_path) );
		return  ( p2(delta_t)*t1 +q2(delta_t)*t2)/discount_rate(delta_t) ;
	}
}

FILE* ptr;
```

```
int main()
{
	float delta_t;

	// Part (a)
	S0=100;
	K=100;
	T=1;
	r=0.08;
	sigma=0.2;



	M=5;
	delta_t=T/M;
	cout<<"For M = 5 : "<< get_lookback(0,S0, delta_t, S0 )<<endl;


	M=10;
	delta_t=T/M;
	cout<<"For M = 10 : "<< get_lookback(0,S0, delta_t, S0 )<<endl;


	M=25;
	delta_t=T/M;
	cout<<"For M = 25 : "<< get_lookback(0,S0, delta_t, S0 )<<endl;


	M=50;
	delta_t=T/M;
	cout<<"For M = 50 : "<< get_lookback(0,S0, delta_t, S0 )<<endl;

	return 0;
}
```

Part(c):

```
#include<iostream>
#include<cstdio>
#include<cstdlib>
#include<cmath>
using namespace std;
```

```
float S0=100;
float K=100;
float T=1;
float r=0.08;
float sigma=0.2;
int M=100;

float max(float a,float b)
{
	if( a>b ) return a;
	else return b;
}

float u(float delta_t)
{
	return exp(sigma*sqrt(delta_t));
}

float u2(float delta_t)
{
	return exp( sigma*sqrt(delta_t) + (r-0.5*sigma*sigma )*delta_t );
}

float d(float delta_t)
{
	return exp(-sigma*sqrt(delta_t));
}

float d2(float delta_t)
{
	return exp( -sigma*sqrt(delta_t) + ( r-0.5*sigma*sigma )*delta_t );
}


float discount_rate(float t)
{
	return exp(r*t);
}

float p(float delta_t)
{
	return ( exp(r*delta_t) - d(delta_t) )/ ( u(delta_t) -d(delta_t) ) ;
}
```

```
float q(float delta_t)
{
	return ( u(delta_t) - exp(r*delta_t) )/ ( u(delta_t) - d(delta_t) );
}

/// Calculated using u2,d2
float p2(float delta_t)
{
	return ( exp(r*delta_t) - d2(delta_t) )/ ( u2(delta_t) -d2(delta_t) ) ;
}

float q2(float delta_t)
{
	return ( u2(delta_t) - exp(r*delta_t) )/ ( u2(delta_t) - d2(delta_t) );
}

// This is for the call option(recursive)
// maxpath -> it is used to store the max value until now
float get_lookback(float t,float S,float delta_t,float mx_path=-1)    // returns the price
of the option at time t if the price os stock is S )
{
	if( mx_path==-1 )  mx_path=S ;    // i.e. the starting point of the loop
	if(t>=T)
	{
		return (mx_path - S);
	}
	else
	{
		float t1=get_lookback(t+delta_t , S*u2(delta_t) , delta_t,
max( S*u2(delta_t), mx_path) );
		float t2=get_lookback(t+delta_t , S*d2(delta_t) , delta_t,
max( S*d2(delta_t), mx_path) );
		return  ( p2(delta_t)*t1 +q2(delta_t)*t2)/discount_rate(delta_t) ;
	}
}

FILE* ptr;


int main()
{
	float delta_t;

	// Part (a)
	S0=100;
```

```
	K=100;
	T=1;
	r=0.08;
	sigma=0.2;

	cout<<"All the intermediate values for M = 5 "<<endl;
	M=5;
	delta_t=T/M;
	cout<<"At i=0 "<< get_lookback(0,S0, delta_t, S0 )<<endl;
	cout<<"At i=1 "<< get_lookback(delta_t , S0*u2(delta_t), delta_t,
S0*u2(delta_t) )<<"   "<< get_lookback(delta_t , S0*d2(delta_t) , delta_t,
S0*d2(delta_t) )<<endl;
	cout<<"At i=2 "<< get_lookback(delta_t*2 , S0*pow(u2(delta_t),2) ,delta_t )
<<"   "<<get_lookback(delta_t*2, S0*pow(u2(delta_t),1)*pow(d2(delta_t),1) , delta_t
) << "  "<< get_lookback( delta_t*2 , S0*pow(d2(delta_t),2) ,delta_t ) <<endl;
	cout<<"At i=3 "<< get_lookback(delta_t*3 , S0*pow(u2(delta_t),3) ,delta_t )
<<"   "<<get_lookback(delta_t*3, S0*pow(u2(delta_t),2)*pow(d2(delta_t),1) , delta_t
) <<"   "<< get_lookback(delta_t*3 , S0*pow(u2(delta_t),1)*pow(d2(delta_t),2) ,
delta_t )<<"   "<<get_lookback( delta_t*4, S0*pow(d2(delta_t),3) , delta_t )<<endl;

	cout<<"At i=4 "<< get_lookback(delta_t*4 , S0*pow(u2(delta_t),4) ,delta_t )
<<"   "<<get_lookback(delta_t*4, S0*pow(u2(delta_t),3)*pow(d2(delta_t),1),
delta_t ) <<"   "<<  get_lookback(delta_t*4 ,
S0*pow(u2(delta_t),2)*pow(d2(delta_t),2) , delta_t )<<"   "<<
get_lookback(delta_t*4 , S0*pow(u2(delta_t),1)*pow(d2(delta_t),3 ) , delta_t ) <<"
"<< get_lookback(delta_t*4, S0*pow(d2(delta_t),4) , delta_t ) <<endl;

	return 0;
}
```

OBSERVATION:

Part (a) and (b):

For M = 5 : 9.11931
For M = 10 : 10.0806
For M = 25 : 11.0034

Part(c):

All the intermediate values for M = 5

At i=0	9.11931				
At i=1	9.02796	7.5492			
At i=2	8.54809	7.14792	5.97711		
At i=3	7.41678	6.20192	5.18606	2.9065	
At i=4	5.50165	4.60048	3.84693	3.21681	2.6899

3).

```
#include<iostream>
#include<cstdio>
#include<cstdlib>
#include<set>
#include<vector>
#include<map>
#include<cmath>
#include<climits>
using namespace std;


float S0=100;
float K=100;
float T=1;
float r=0.08;
float sigma=0.2;
int M=100;

float max(float a,float b)
{
	if( a>b ) return a;
	else return b;
}

double max( double a, double b)
{
	if( a> b) return a;
	else return b;
}

float u(float delta_t)
{
	return exp(sigma*sqrt(delta_t));
}

float u2(float delta_t)
{
	return exp( sigma*sqrt(delta_t) + (r-0.5*sigma*sigma )*delta_t );
}

float d(float delta_t)
{
	return exp(-sigma*sqrt(delta_t));
}
```

```
float d2(float delta_t)
{
	return exp( -sigma*sqrt(delta_t) + ( r-0.5*sigma*sigma )*delta_t );
}


double discount_rate(double t)
{
	return exp(r*t);
}

float p(float delta_t)
{
	return ( exp(r*delta_t) - d(delta_t) )/ ( u(delta_t) -d(delta_t) ) ;
}

float q(float delta_t)
{
	return ( u(delta_t) - exp(r*delta_t) )/ ( u(delta_t) - d(delta_t) );
}

/// Calculated using u2,d2
double p2(double delta_t)
{
	return ( exp(r*delta_t) - d2(delta_t) )/ ( u2(delta_t) -d2(delta_t) ) ;
}

double q2(double delta_t)
{
	return ( u2(delta_t) - exp(r*delta_t) )/ ( u2(delta_t) - d2(delta_t) );
}

// This is for the call option(recursive)
// maxpath -> it is used to store the max value until now
double get_lookback(double t,double S,double delta_t,double mx_path)    // returns
the price of the option at time t if the price os stock is S )
{
	if(t>=T)
	{
		return (mx_path - S);
	}
	else
	{
		double t1=get_lookback(t+delta_t , S*u2(delta_t) , delta_t,
```

```
max( S*u2(delta_t), mx_path) );
                double t2=get_lookback(t+delta_t , S*d2(delta_t) , delta_t,
max( S*d2(delta_t), mx_path) );
                return  ( p2(delta_t)*t1 +q2(delta_t)*t2)/discount_rate(delta_t) ;
        }
}


// This function returns a Brownian path
vector<double> get_path(double delta_t,double S0)
{
        double U= u2(delta_t);
        double D= d2(delta_t);
        double P = p2(delta_t);
        vector<double> out;  // the output to be appended;
        double i=0;
        double curr;
        curr=S0;
        for(i=0; i<=1 ; i+=delta_t )
        {
                if( ((double)rand()/RAND_MAX ) <  P )  // i.e. stock going up
                {
                        curr = curr*U;
                        out.push_back( curr );
                }
                else
                {
                        curr = curr*D;
                        out.push_back( curr );
                }
        }
        return out;
}

double get_price( vector<double> path )   // return the price of the option according
to the vector
{
        double mx = 0;  //
        vector<double>::iterator it;
        for(it= path.begin() ; it!=path.end() ; it++)
        {
                mx = max( mx, *it );
        }
        return (mx - path[ path.size()-1 ] );
}
```

```
double get_lookback_price(double delta_t, double S0 )  // Generates thousands of path and then take their average
{
	double price =0;
	int count =0;
	for(count =0 ;count < 1000000; count++)
		price += get_price( get_path(delta_t, S0 ));
	return (price/count)/discount_rate(1.0);
}

FILE* ptr;

int main()
{
	double delta_t;

	// Part (a)
	S0=100;
	K=100;
	T=1;
	r=0.08;
	sigma=0.2;



	M=5;
	delta_t=T/M;
	cout<<"For M = 5 : "<< get_lookback_price(delta_t, S0 )<<endl;


	M=10;
	delta_t=T/M;
	cout<<"For M = 10 : "<< get_lookback_price(delta_t, S0 )<<endl;


	M=25;
	delta_t=T/M;
	cout<<"For M = 25 : "<< get_lookback_price( delta_t, S0 )<<endl;


	M=50;
	delta_t=T/M;
	cout<<"For M = 50 : "<< get_lookback_price( delta_t, S0 )<<endl;
```

```
        return 0;
}
```

OBSERVATION:

For M = 5 : 8.16982
For M = 10 : 9.57383
For M = 25 : 11.0504
For M = 50 : 11.533