MA 374 – Financial Engineering II

Lab 2
Report

Name: Vibhanshu
Roll No. : 120123049

Q1. The C++ Program

```
#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(k=0; k<j; k++)
```

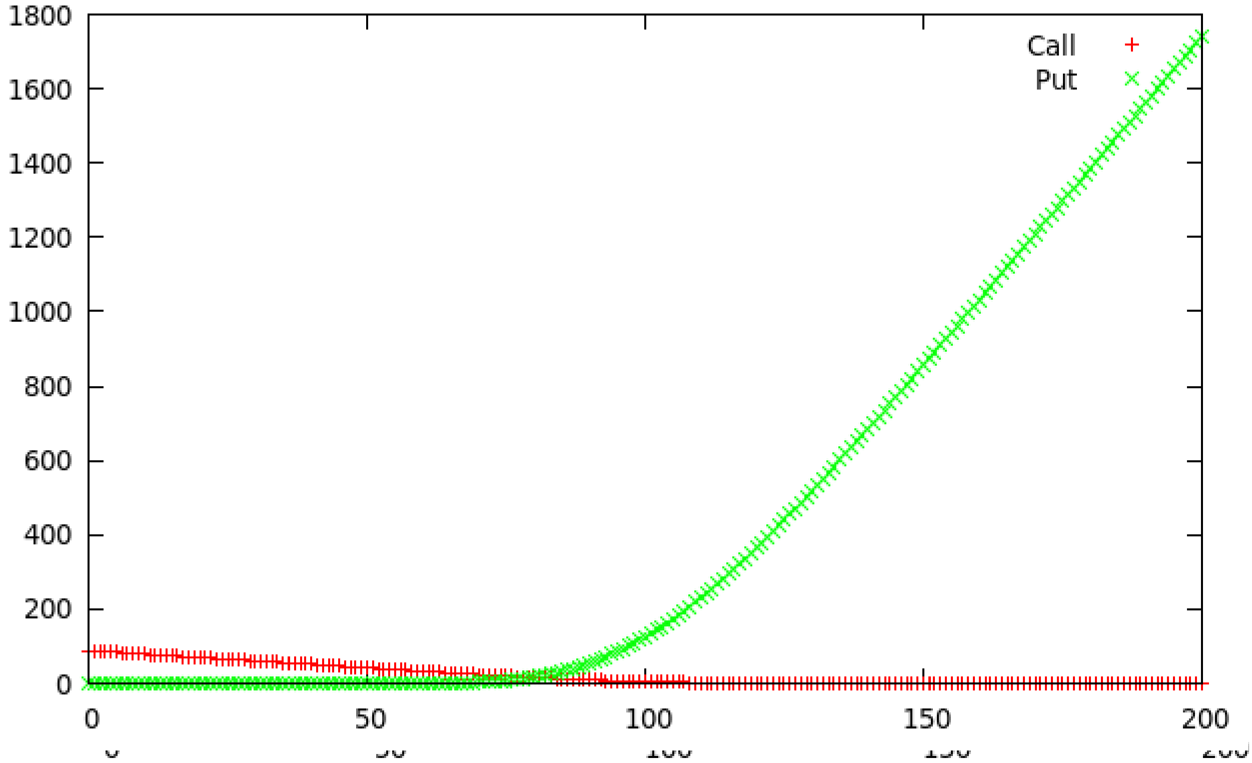
```
            value[k]= (P*value[k+1]+Q*value[k])/R;
    }
    return value[0];
}

// This is the fast version of the algorithm PUT
float get_P_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(k=0; k<j; k++)
            value[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_fast(0,S0,delta_t),get_P_fast(0,S0,delta_t)); }
        fclose(ptr);
        ptr=fopen("set2_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_fast(0,S0,delta_t),get_P_fast(0,S0,delta_t)); }
        fclose(ptr);
        ptr=fopen("set2_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_fast(0,S0,delta_t),get_P_fast(0,S0,delta_t)); }
        fclose(ptr);
        ptr=fopen("set2_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_fast(0,S0,delta_t),get_P_fast(0,S0,delta_t)); }
        fclose(ptr);
        ptr=fopen("set2_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_fast(0,S0,delta_t),get_P_fast(0,S0,delta_t));
        }
        fclose(ptr);
        ptr=fopen("set2_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_fast(0,S0,delta_t),get_P_fast(0,S0,delta_t));
        }
        fclose(ptr);
        ptr=fopen("set2_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_fast(0,S0,delta_t),get_P_fast(0,S0,delta_t));
        }
        fclose(ptr);
        ptr=fopen("set2_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;
}
```

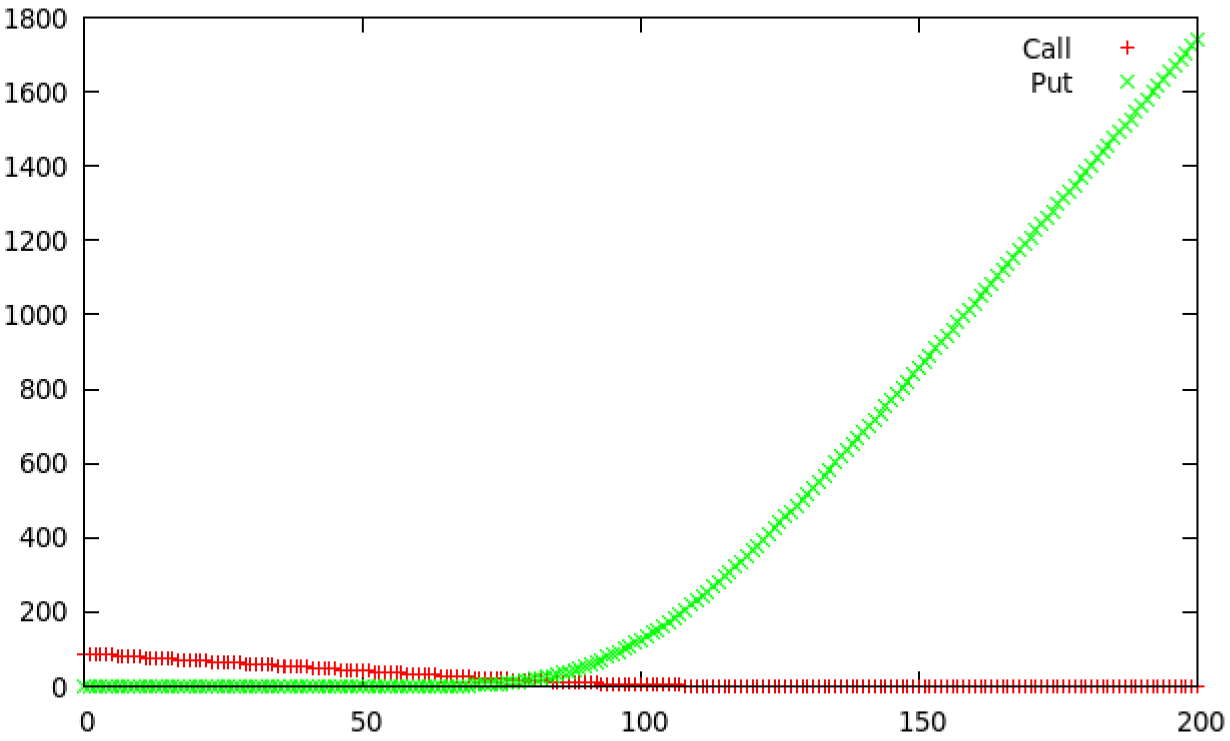
OUTPUT

SET 1:

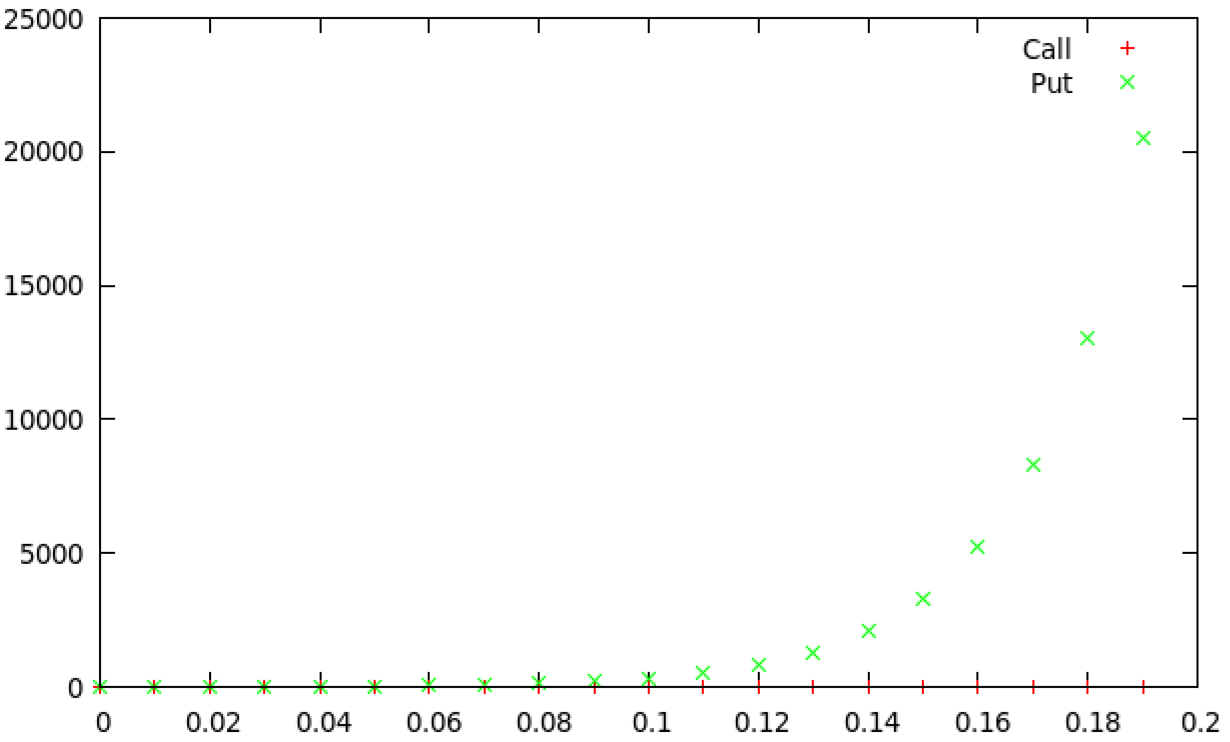
a). Varying S(0)



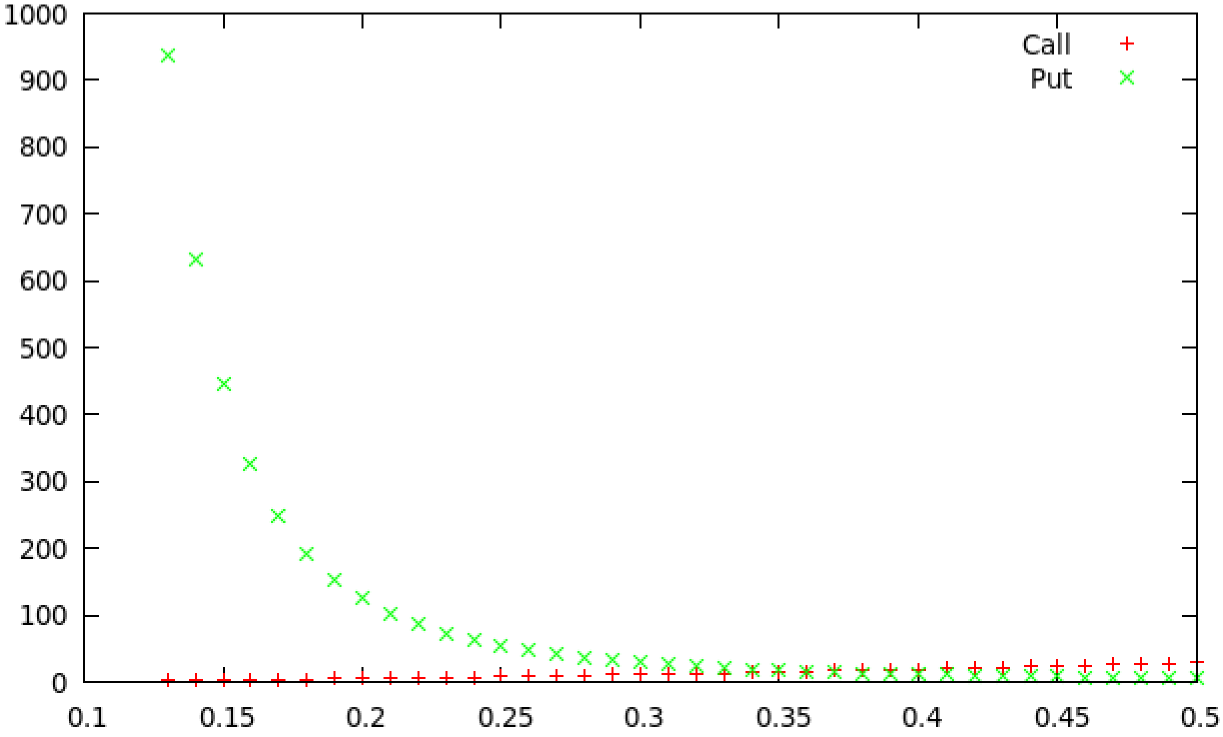
b). Varying K



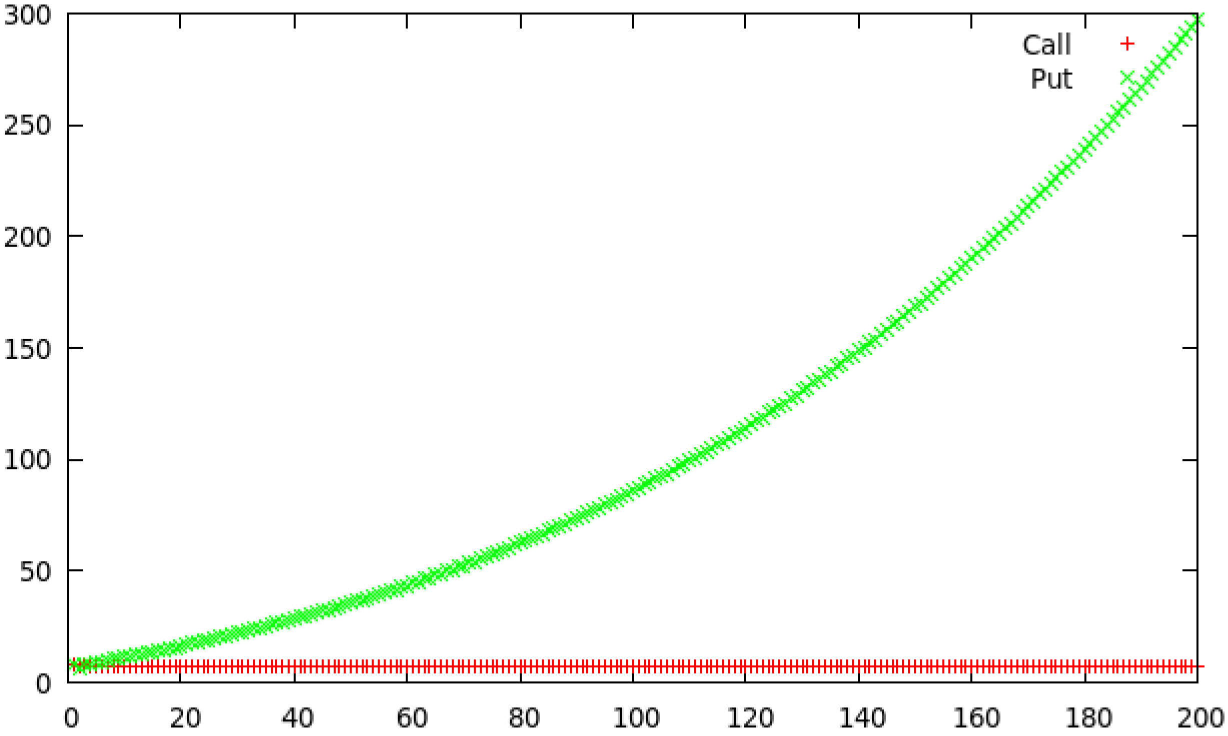
c). Varying r



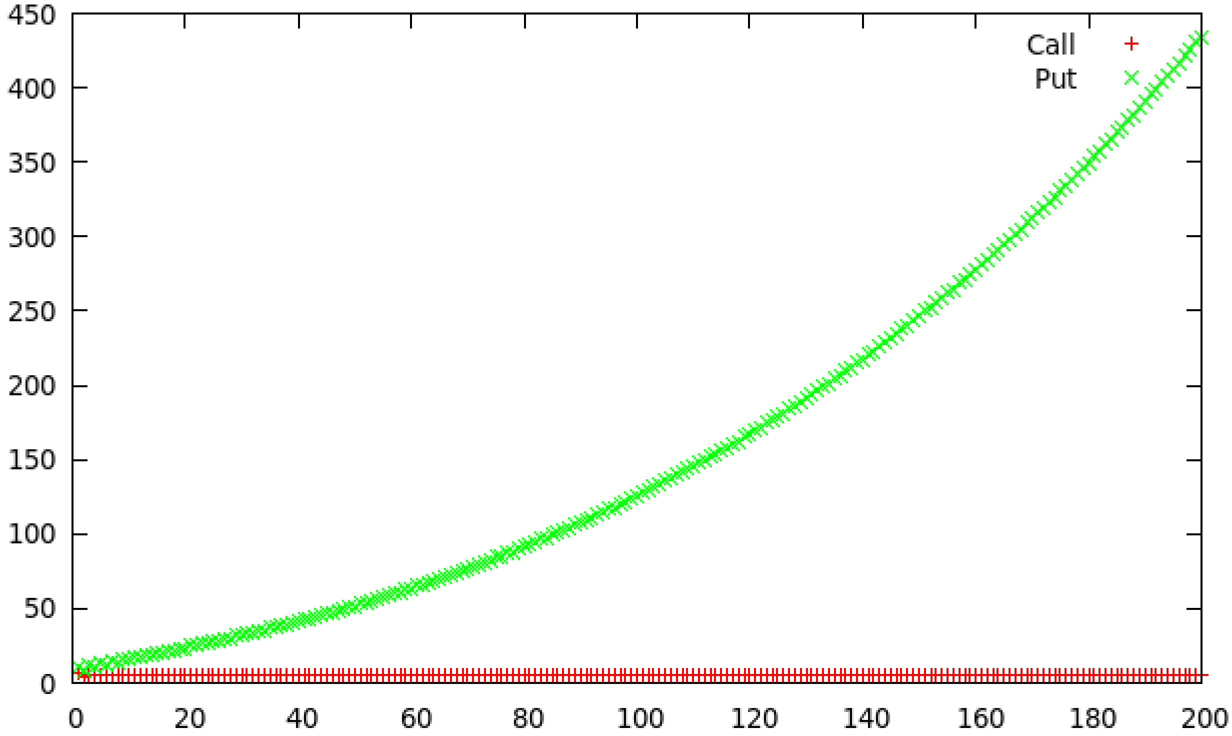
d). Varying (sigma)



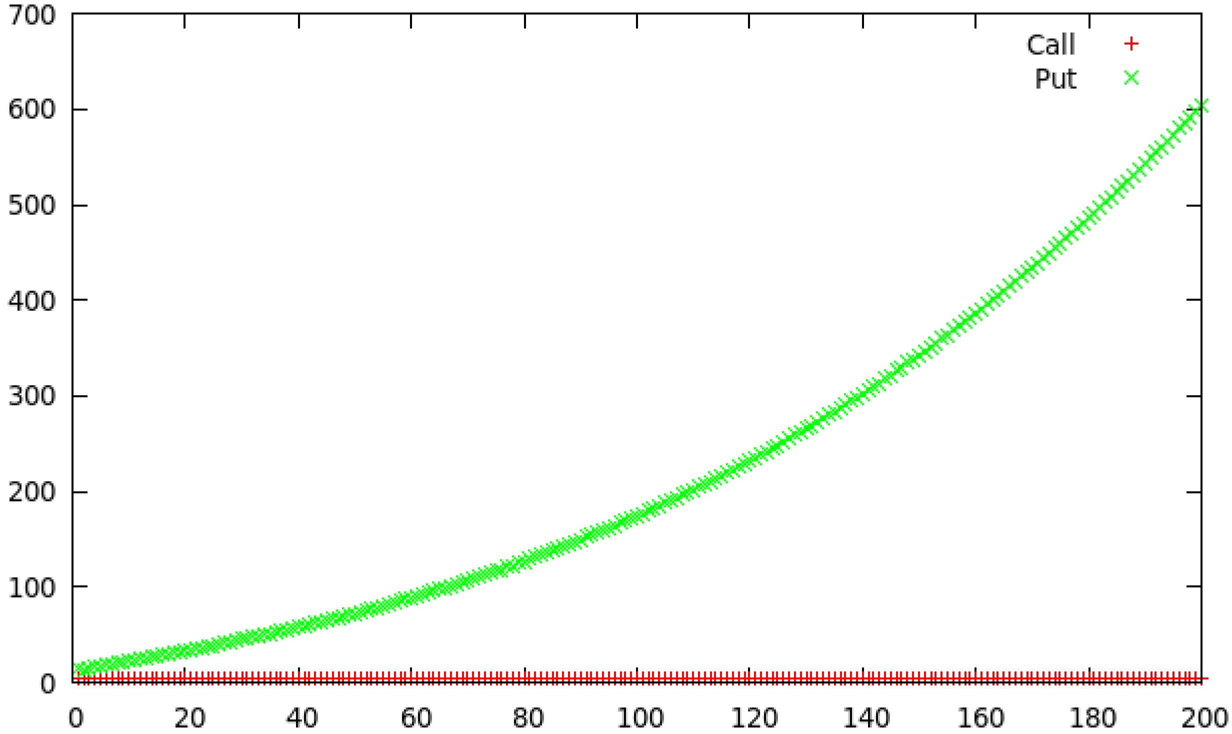
e). Varying M
K=95



K = 100

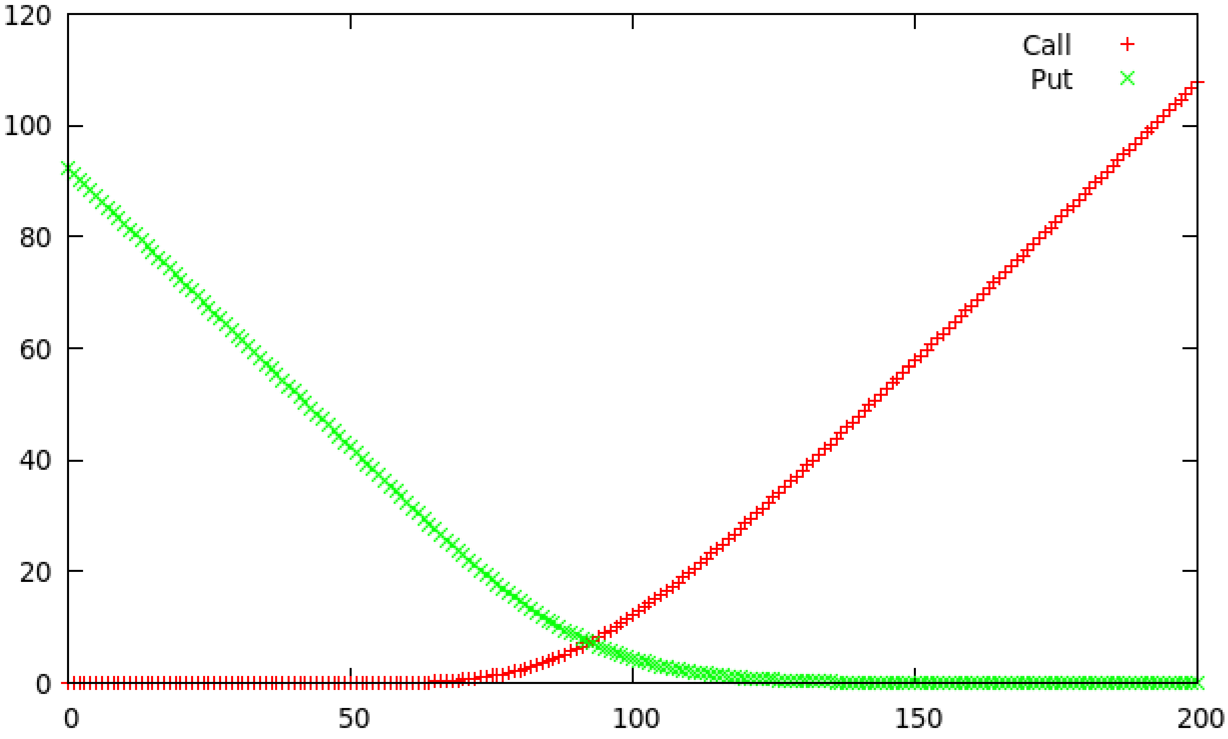

450
400
350
300
250
200
150
100
50
0
0
20
40
60
80
100
120
140
160
180
200
Call +
Put ×

K = 105

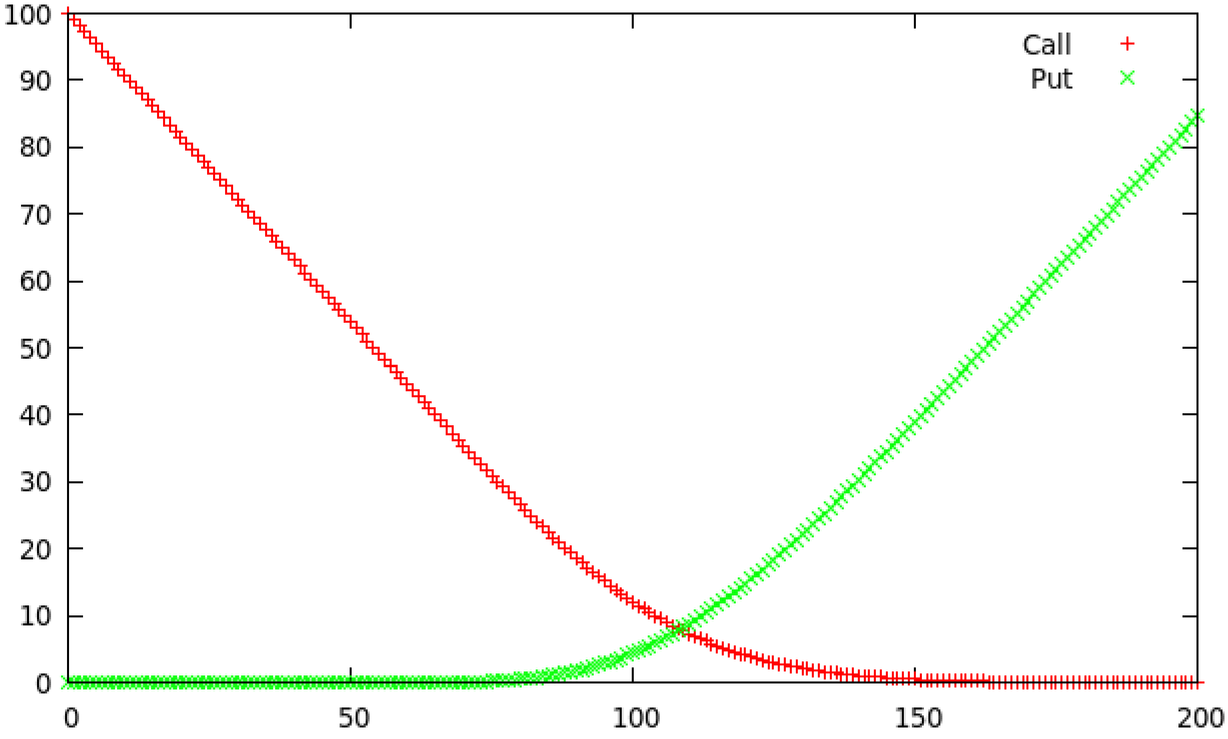

700
600
500
400
300
200
100
0
0
20
40
60
80
100
120
140
160
180
200
Call +
Put ×

SET 2 :

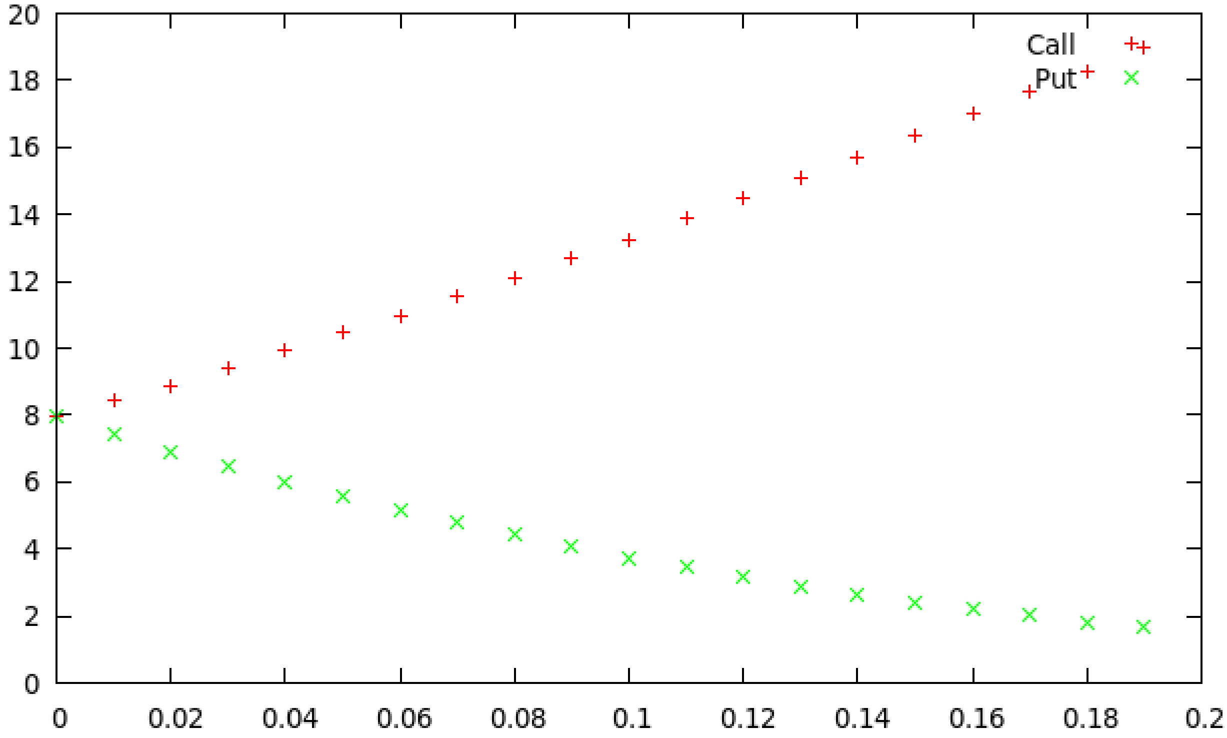
a). Varying S(0)



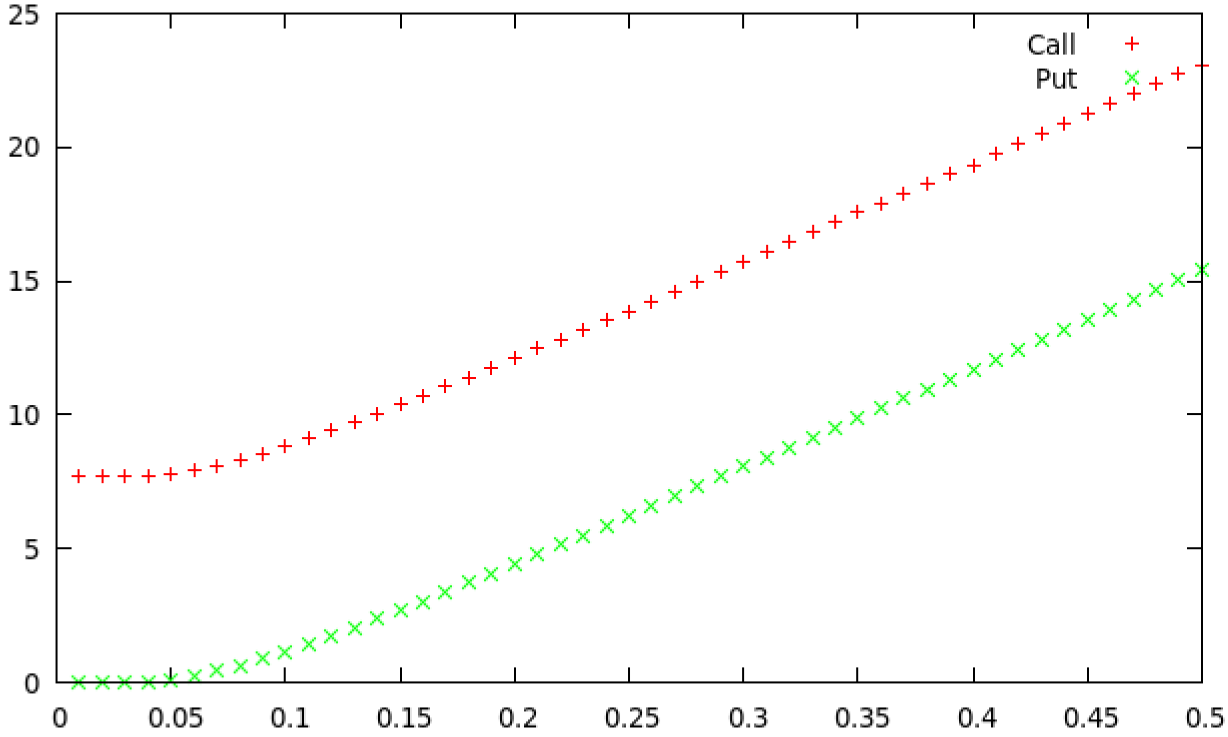
b). Varying K



c). Varying r

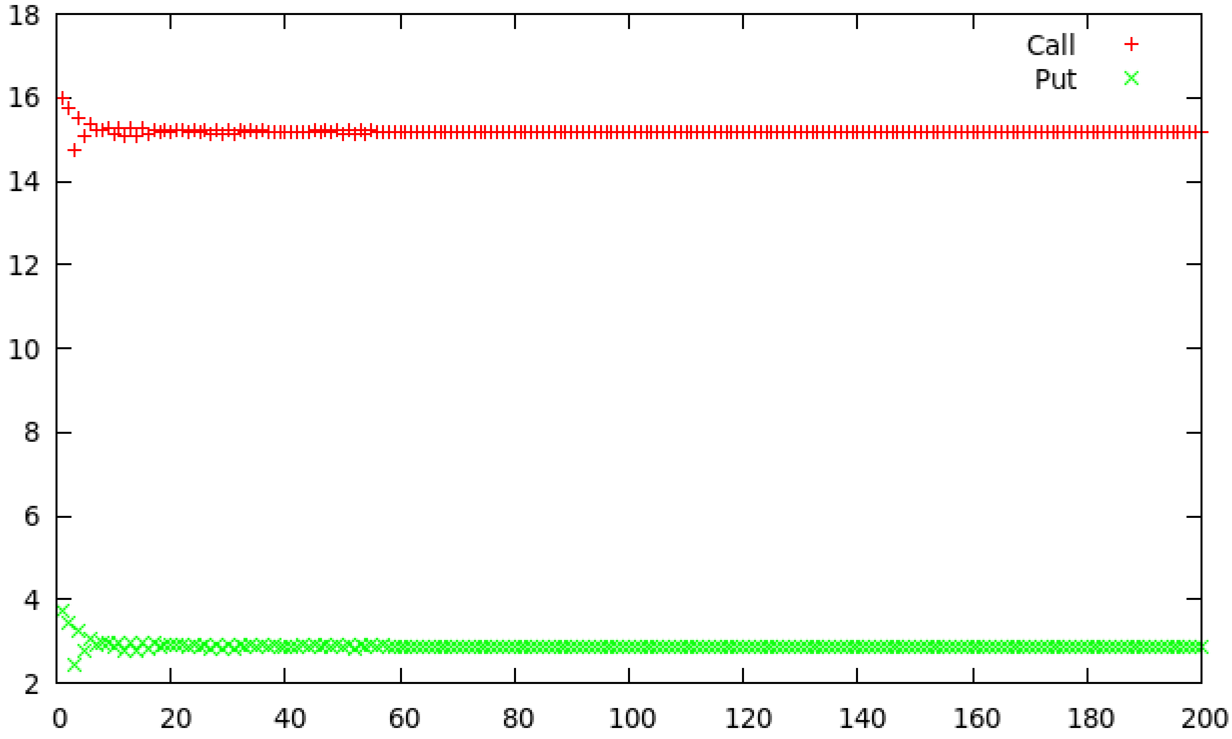


d). Varying (sigma)

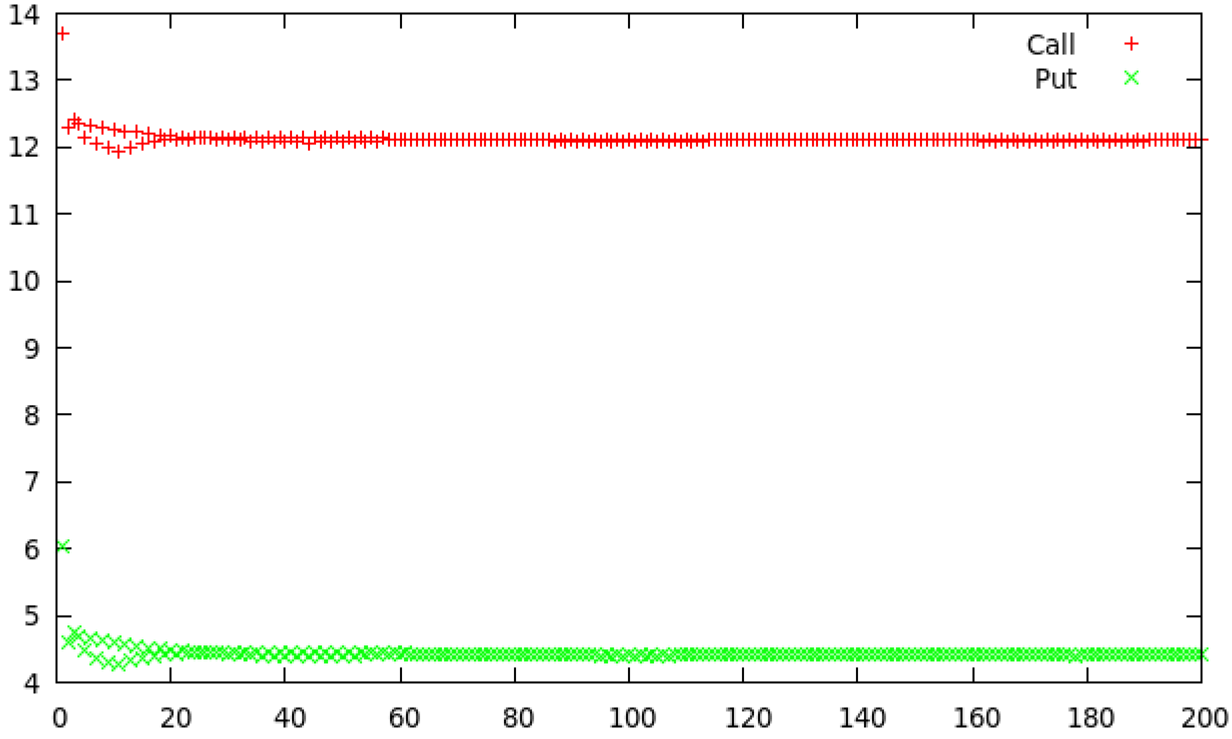


e). Varying M

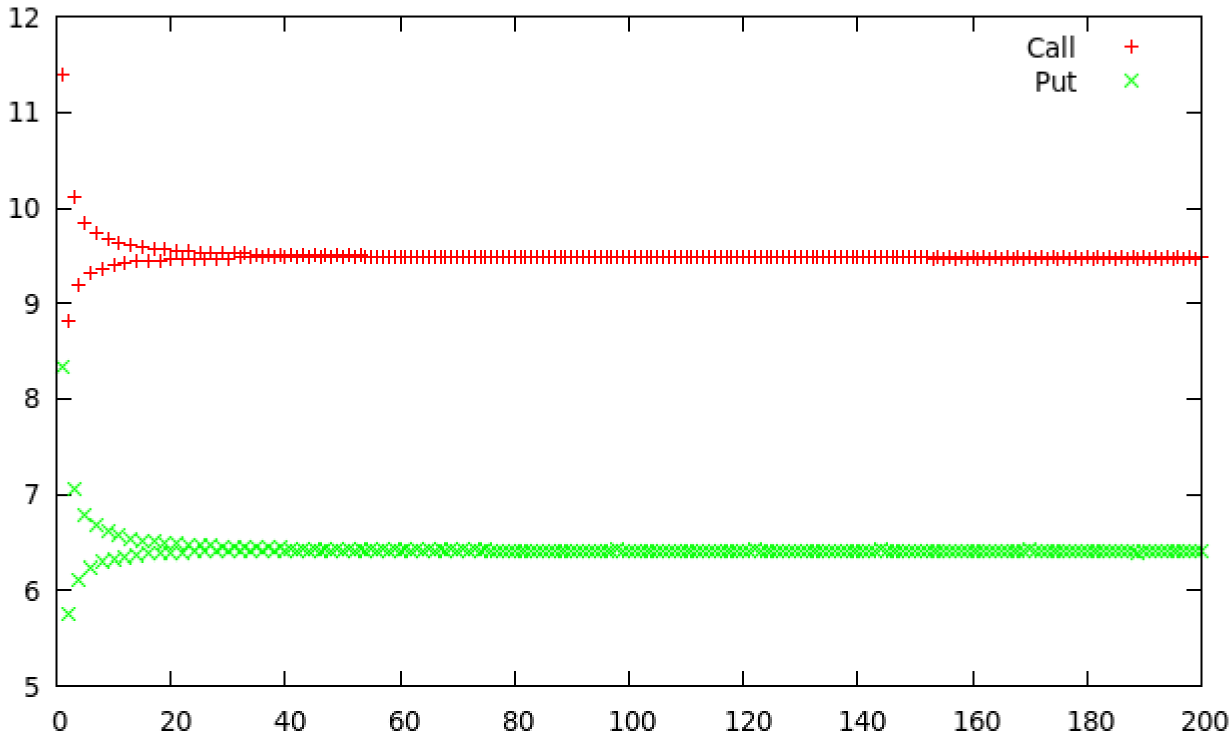
K = 95



K=100



K = 105



Q2. The path-dependent option is American Call-Put Option

The C++ program

```
#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 );
}
```

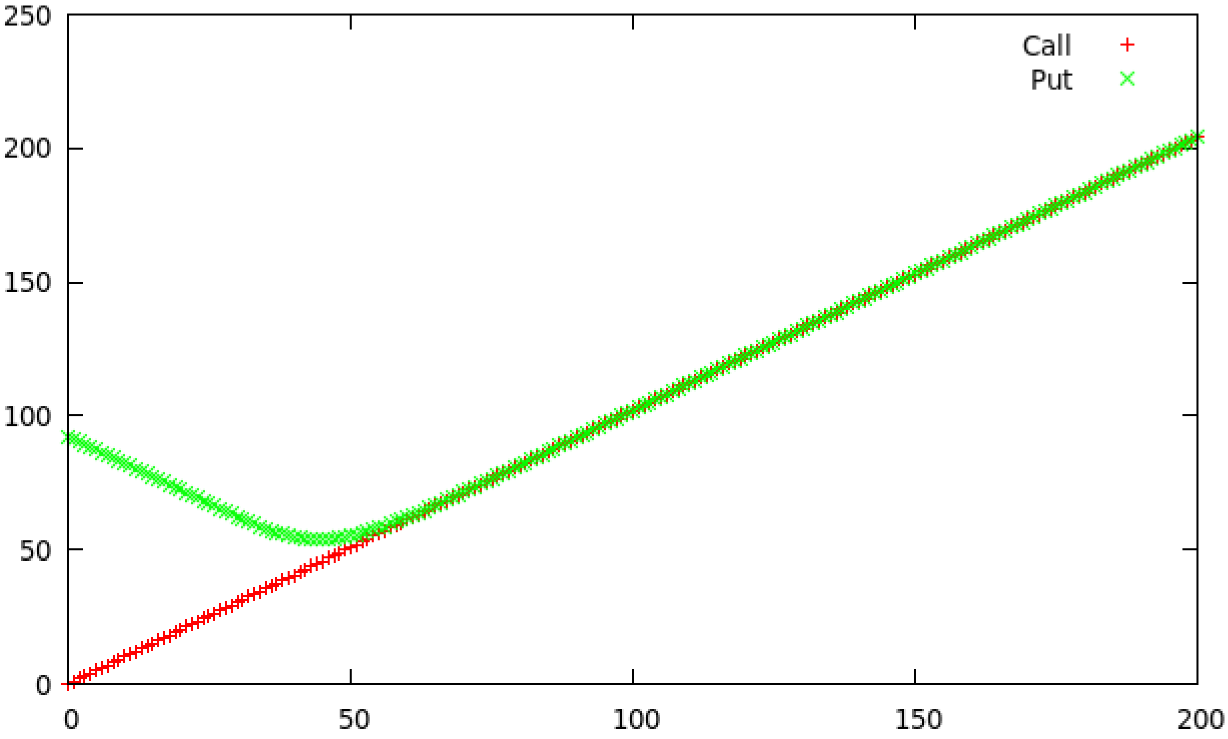
```
float discount_rate(float t)
{
        return exp(r*t);
}

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;
}
```

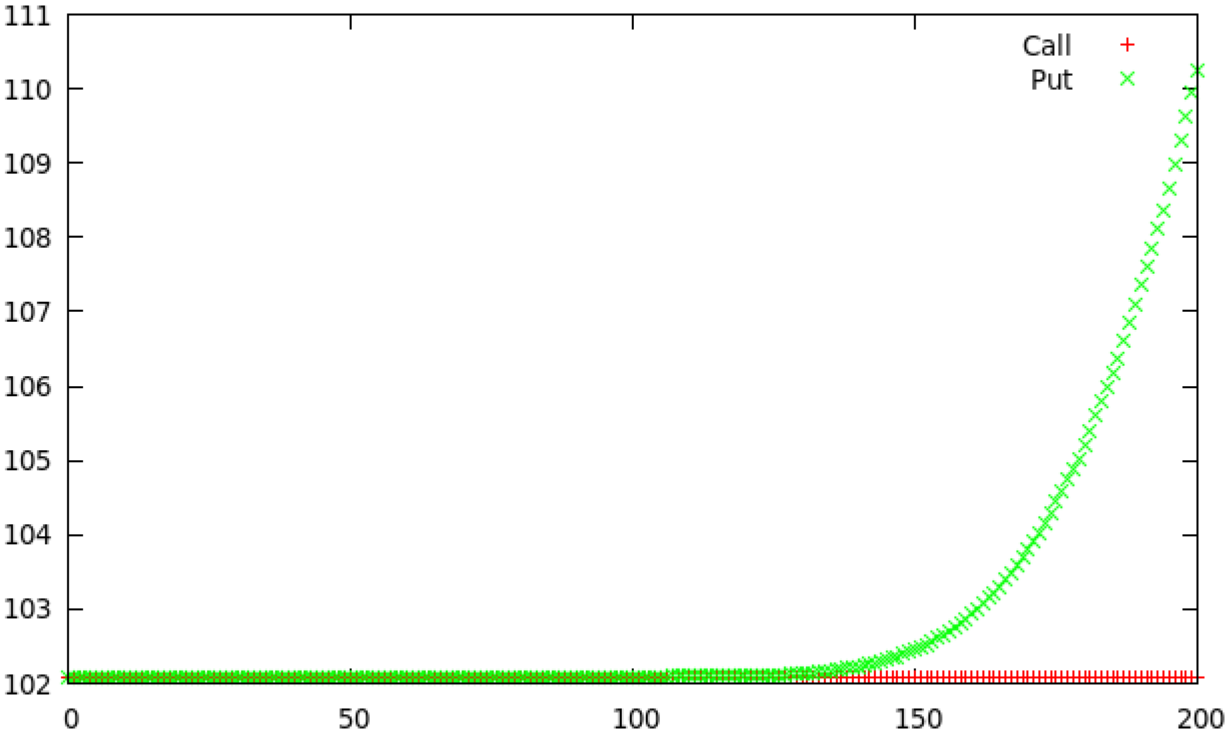
OUTPUT:

SET 2:

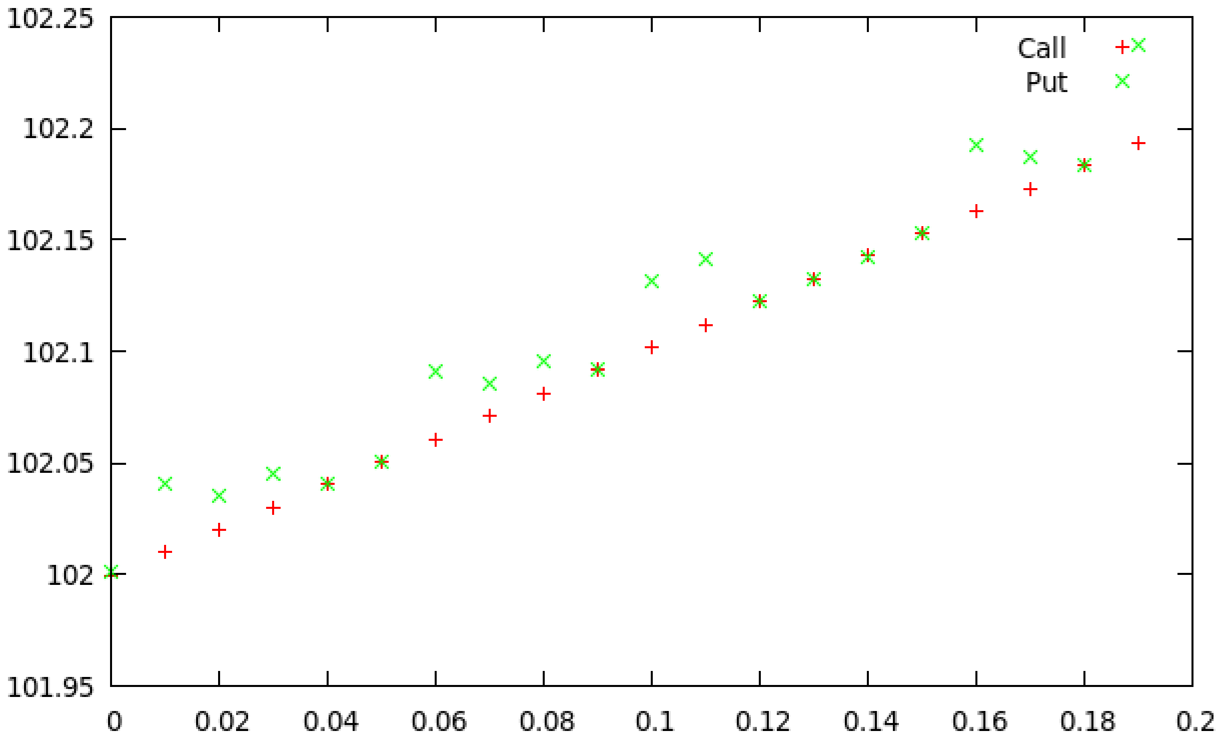
a). Varying S(0)



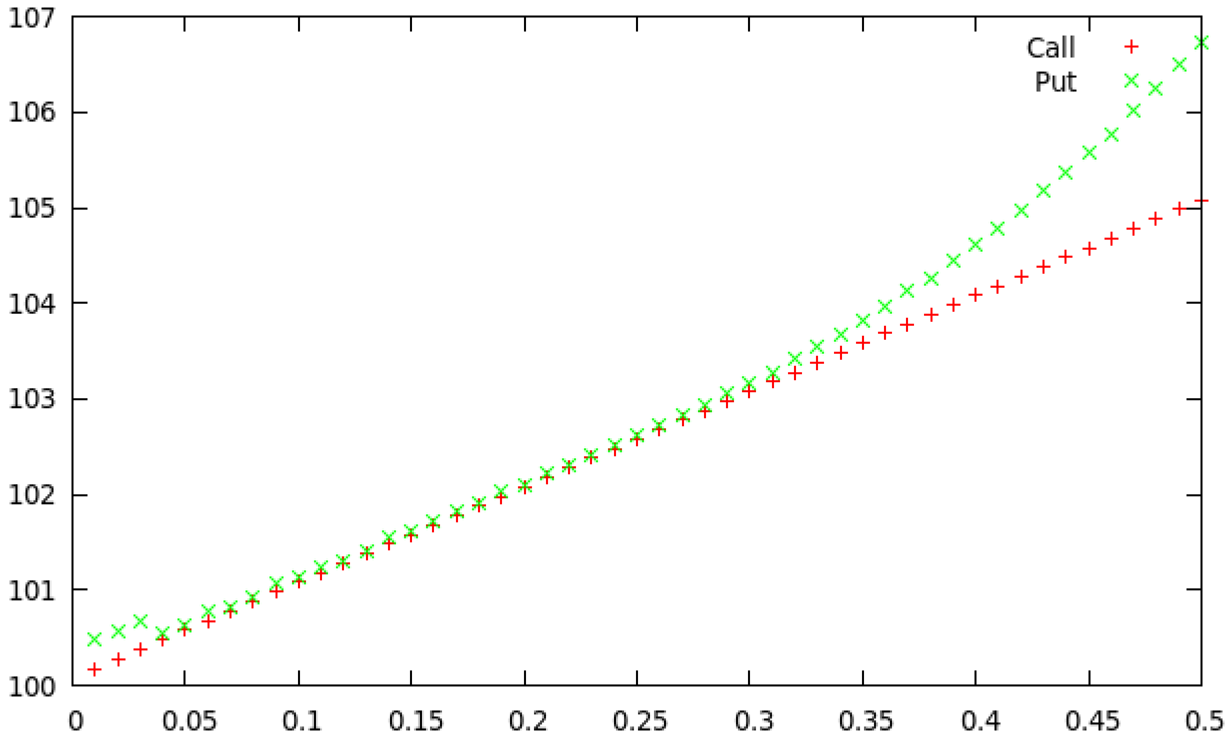
b). Varying K



c). Varying r

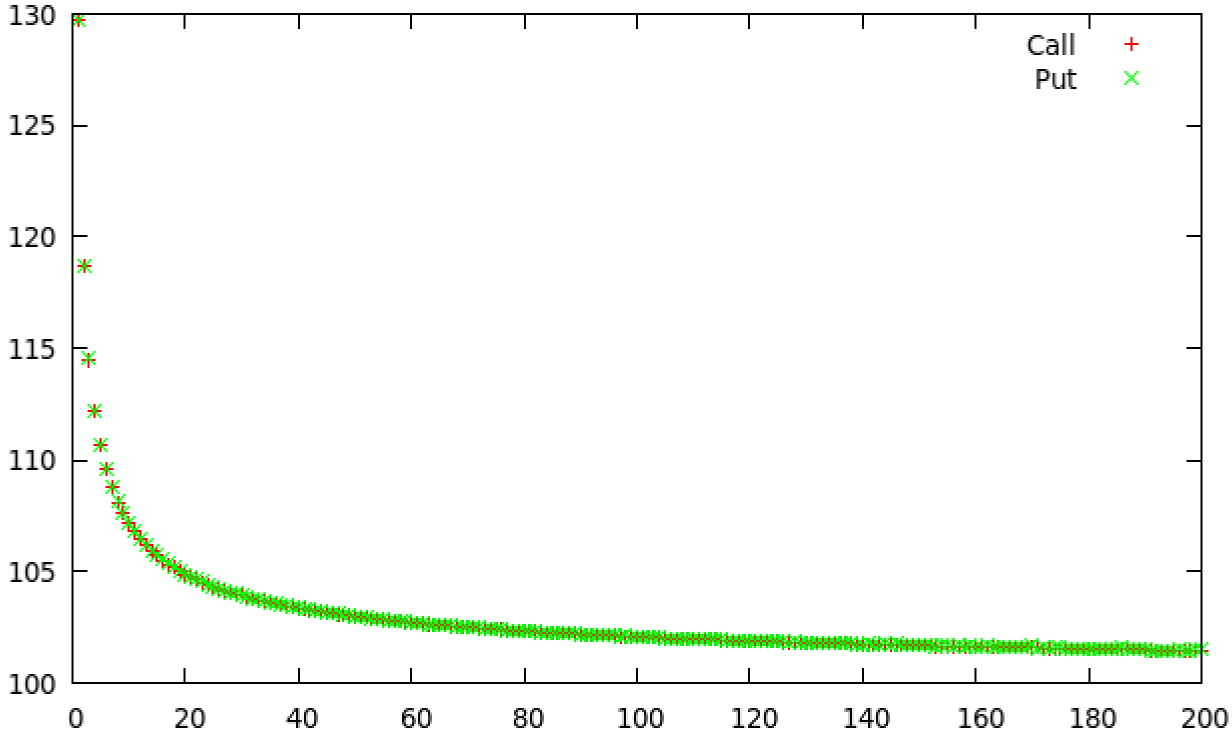


d). Varying (sigma)

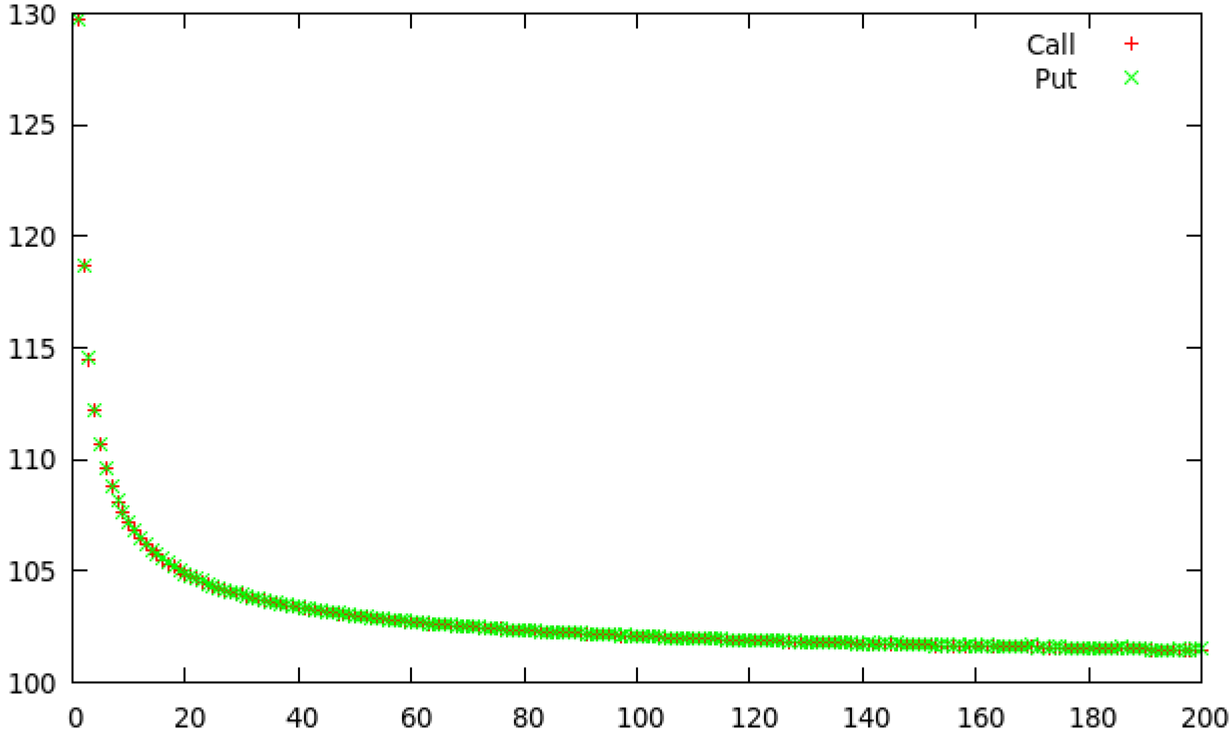


e). Varying M

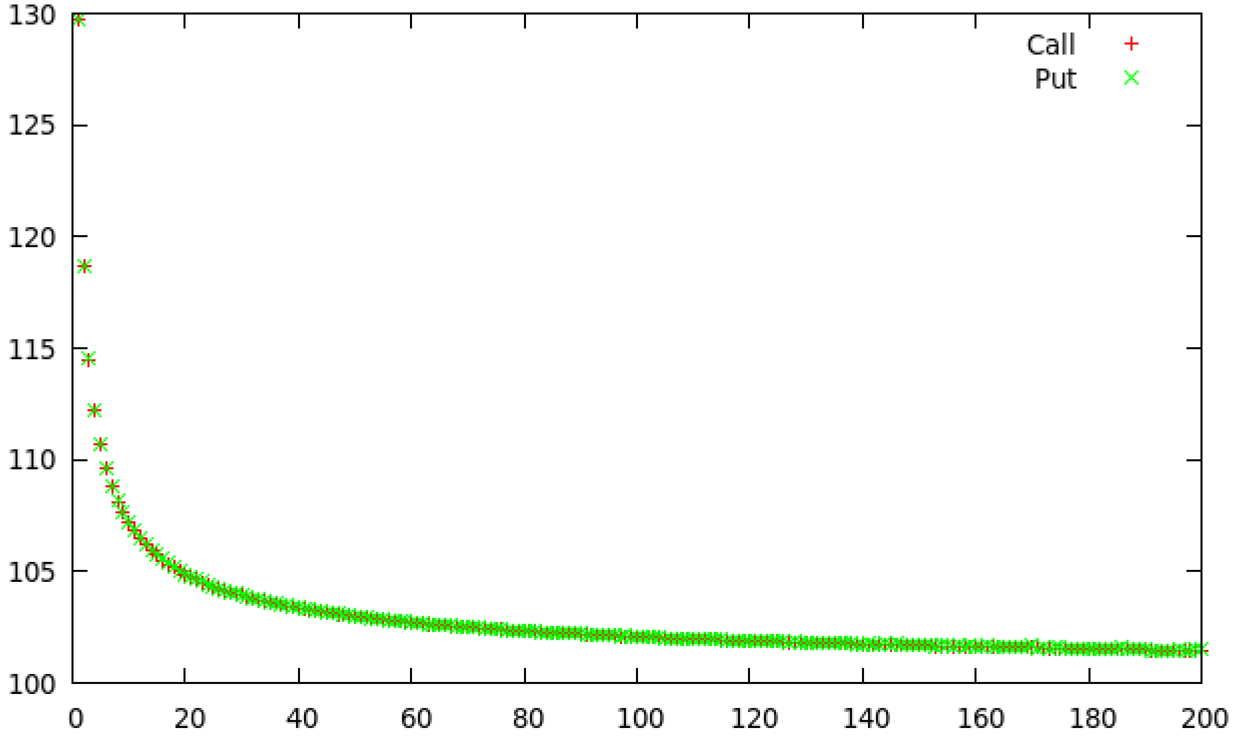
K=95



K=100



K=105



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