

A Fast Numerical Scheme for Black-Scholes Option Pricing Model

Rundong Jiang

1. Abstract

The exact solution of the Black-Scholes equation involves stochastic term, which made it time-consuming to calculate. Therefore, I try to find a way to solve the Black-Scholes equation numerically to avoid evaluating the stochastic term. In this paper, I use forward difference, backward difference, and Crank-Nicolson method to discretize the equation and Jacobi method, Gauss-Seidel method and Successive Over Relaxation (SOR) Method are used to speed up the matrix operation process.

2. Background

The Black-Scholes Equation is: $\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + rS \frac{\partial V}{\partial S} - rV = 0$. S represents the price of the stock, which will sometimes be a random variable and other times a constant (context should make this clear). $V(S, t)$ represents the price of a derivative as a function of time and stock price. $C(S, t)$ represents the price of a European call option and $P(S, t)$, the price of a European put option. S represents the strike price of the option. r represents the annualized risk-free interest rate, continuously compounded (the force of interest). σ represents the volatility of the stock's returns. This is the square root of the quadratic variation of the stock's log price process.

t , a time in years; we generally use: now=0, expiry=T.[1]

Since we know the price of option at expiry, we use a time inverse Black-Scholes equation to change the terminal condition to initial condition. By changing of variable $t=T-t'$, we get $\frac{\partial V}{\partial t} - \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} - rS \frac{\partial V}{\partial S} + rV = 0$. The initial condition for call option is $C(S, 0) = \max(S - E, 0)$ and for put option is $P(S, 0) = \max(E - S, 0)$ which are given by financial theories. The boundary condition for call option are $C(0, t) = 0$, $C(S, t)/S \rightarrow 1$, for $S \rightarrow \infty$. The boundary condition for put option are $P(0, t) = Ee^{-r(T-t)}$, $P(S, t) \rightarrow 0$, for $S \rightarrow \infty$.

But in real life, it is impossible for S to be infinity. So in our numerical scheme, We can set the maximum S to be from 3 to 5 times as large as the strike price of the option and let the boundary condition for call option be $C(S_{max},$

$t) = S_{\max} - Ke^{-r(T-t)}$. The boundary condition for put option does not need to be changed. Moreover, Discretize the equation by letting $t=j\Delta t$ where $j=1,2,\dots,M$ and $S=m\Delta S$ where $m=1,2,\dots,N$.

3. Forward difference method:

3.1 Derivation

We apply central difference for $\frac{\partial V}{\partial s}$ and $\frac{\partial^2 V}{\partial s^2}$

$$\frac{\partial V}{\partial s} = \frac{V_{j,m+1} - V_{j,m-1}}{2\Delta S}$$

$$\frac{\partial^2 V}{\partial s^2} = \frac{V_{j,m+1} - 2V_{j,m} + V_{j,m-1}}{\Delta S^2}$$

Forward difference for $\frac{\partial V}{\partial t}$ when develop explicit method

$$\frac{\partial V}{\partial t} = \frac{V_{j+1,m} - V_{j,m}}{\Delta t}$$

Then the Black-Scholes equation turns into

$$\frac{V_{j+1,m} - V_{j,m}}{\Delta t} - \frac{1}{2}\sigma^2 S^2 \frac{V_{j,m+1} - 2V_{j,m} + V_{j,m-1}}{\Delta S^2} - rS \frac{V_{j,m+1} - V_{j,m-1}}{2\Delta S} + rV_{j,m} = 0$$

Then we get

$$V_{j+1,m} = V_{j,m} + \frac{1}{2}\Delta t\sigma^2 S^2 \frac{V_{j,m+1} - 2V_{j,m} + V_{j,m-1}}{\Delta S^2} + \Delta t r S \frac{V_{j,m+1} - V_{j,m-1}}{2\Delta S} - rV_{j,m}\Delta t$$

Since $S/\Delta S=m$,we have

$$V_{j+1,m} = (1 - \sigma^2 m^2 \Delta t - r\Delta t) V_{j,m} + (\frac{1}{2}\sigma^2 m^2 \Delta t + \frac{1}{2}\Delta t r m) V_{j,m+1} + (\frac{1}{2}\sigma^2 m^2 \Delta t + \frac{1}{2}\Delta t r m) V_{j,m-1}$$

Let

$$\alpha(m) = 1 - \sigma^2 m^2 \Delta t - r\Delta t$$

$$\beta(m) = \frac{1}{2}\sigma^2 m^2 \Delta t + \frac{1}{2}\Delta t r m$$

$$\gamma(m) = \frac{1}{2}\sigma^2 m^2 \Delta t - \frac{1}{2}\Delta t r m$$

then,let

$$A = \begin{pmatrix} \alpha(1) & \beta(1) & 0 & & & & \\ \gamma(2) & \alpha(2) & \beta(2) & \cdots & & & 0 \\ 0 & \gamma(3) & \alpha(3) & & & & \\ \vdots & \vdots & \vdots & \ddots & & & \vdots \\ & & & & \alpha(N-3) & \beta(N-3) & 0 \\ & 0 & \cdots & \gamma(N-2) & \alpha(N-2) & \beta(N-2) & \\ & & & 0 & \gamma(N-1) & \alpha(N-1) & \end{pmatrix}$$

$$\mathbf{v}^j = \begin{pmatrix} V_{j,1} \\ \vdots \\ V_{j,N-1} \end{pmatrix}$$

$$\mathbf{b} = \begin{pmatrix} \gamma(1) * V_{j,0} \\ 0 \\ \vdots \\ 0 \\ \beta(N-1) * V_{j,N} \end{pmatrix}$$

We can rewrite the discretized time-inverse Black-Scholes equation as a tridiagonal system

$$\mathbf{v}^{j+1} = \mathbf{A} * \mathbf{v}^j + \mathbf{b}$$

3.2 Stability analysis

In order to make this numerical scheme stable, we want the spectral radius of our tridiagonal matrix to be less than one. We know the eigenvalues for tridiagonal matrix is

$$u_k = \alpha + 2\sqrt{\beta\gamma} * \cos \frac{k\pi}{N+1}$$

Plug in the entites we have in our tridiagonal matrix, we get

$$u_k = 1 - \sigma^2 m^2 \Delta t - r\Delta t + 2\sqrt{\left(\frac{1}{2}\sigma^2 m^2 \Delta t + \frac{1}{2}\Delta t r m\right)\left(\frac{1}{2}\sigma^2 m^2 \Delta t - \frac{1}{2}\Delta t r m\right)} * \cos \frac{k\pi}{N+1}$$

$$= 1 - \sigma^2 m^2 \Delta t - r\Delta t + \sqrt{\sigma^4 m^4 \Delta t^2 - \Delta t^2 m^2 r^2}$$

We want to make $|u_k| < 1$, then

$$-1 < 1 - \sigma^2 m^2 \Delta t - r\Delta t + \sqrt{\sigma^4 m^4 \Delta t^2 - \Delta t^2 m^2 r^2} * \cos \frac{k\pi}{N+1} < 1$$

$$-2 < -\sigma^2 m^2 \Delta t - r\Delta t + \sqrt{\sigma^4 m^4 \Delta t^2 - \Delta t^2 m^2 r^2} * \cos \frac{k\pi}{N+1} < 0$$

From which we get,

$$\begin{cases} -\sigma^2 m^2 \Delta t - r\Delta t - \Delta t \sqrt{\sigma^4 m^4 - m^2 r^2} > -2 \\ -\sigma^2 m^2 \Delta t - r\Delta t + \Delta t \sqrt{\sigma^4 m^4 - m^2 r^2} < 0 \end{cases}$$

$$\begin{cases} \Delta t < \frac{-2}{-\sigma^2 m^2 - r - m\sqrt{\sigma^4 m^2 - r^2}} \\ \Delta t(-\sigma^2 m^2 - r + m\sqrt{\sigma^4 m^2 - r^2}) < 0 \end{cases}$$

As $(-\sigma^2 m^2 - r + m\sqrt{\sigma^4 m^2 - r^2}) < 0$ always true

We only need to make

$$\Delta t < \frac{-2}{-\sigma^2 m^2 - r - m\sqrt{\sigma^4 m^2 - r^2}}$$

As $m \leq N$, we have

$$\begin{aligned} \frac{-2}{-\sigma^2 m^2 - r - m\sqrt{\sigma^4 m^2 - r^2}} &= \frac{2}{\sigma^2 m^2 + r + m\sqrt{\sigma^4 m^2 - r^2}} > \frac{2}{\sigma^2 m^2 + r + N\sqrt{\sigma^4 N^2 - r^2}} > \frac{2}{\sigma^2 N^2 + r + N\sqrt{\sigma^4 N^2}} \\ &> \frac{2}{2\sigma^2 N^2 + r} \end{aligned}$$

Hence, the forward difference method will be stable as long as $\Delta t < \frac{2}{2\sigma^2 N^2 + r}$.

4. Backward difference method

4.1 Derivation

We apply central difference for $\frac{\partial V}{\partial s}$ and $\frac{\partial V^2}{\partial^2 s}$

$$\begin{aligned} \frac{\partial V}{\partial s} &= \frac{V_{j,m+1} - V_{j,m-1}}{2\Delta S} \\ \frac{\partial V^2}{\partial^2 s} &= \frac{V_{j,m+1} - 2V_{j,m} + V_{j,m-1}}{\Delta S^2} \end{aligned}$$

Forward difference for $\frac{\partial V}{\partial t}$ when develop explicit method

$$\frac{\partial V}{\partial t} = \frac{V_{j,m} - V_{j-1,m}}{\Delta t}$$

Then the Black-Scholes equation turns into

$$\frac{V_{j,m} - V_{j-1,m}}{\Delta t} - \frac{1}{2}\sigma^2 S^2 \frac{V_{j,m+1} - 2V_{j,m} + V_{j,m-1}}{\Delta S^2} - rS \frac{V_{j,m+1} - V_{j,m-1}}{2\Delta S} + rV_{j,m} = 0$$

Then we get

$$V_{j-1,m} = V_{j,m} - \frac{1}{2}\Delta t \sigma^2 S^2 \frac{V_{j,m+1} - 2V_{j,m} + V_{j,m-1}}{\Delta S^2} - \Delta t r S \frac{V_{j,m+1} - V_{j,m-1}}{2\Delta S} + r\Delta t V_{j,m}$$

Since $S/\Delta S = m$, we have

$$V_{j-1,m} = (1 + \sigma^2 m^2 \Delta t + r\Delta t) V_{j,m} + (-\frac{1}{2}\sigma^2 m^2 \Delta t - \frac{1}{2}\Delta t r m) V_{j,m+1} + (-\frac{1}{2}\sigma^2 m^2 \Delta t + \frac{1}{2}\Delta t r m) V_{j,m-1}$$

Let

$$\alpha(m) = 1 + \sigma^2 m^2 \Delta t + r\Delta t$$

$$\beta(m) = -\frac{1}{2}\sigma^2 m^2 \Delta t - \frac{1}{2}\Delta t r m$$

$$\gamma(m) = -\frac{1}{2}\sigma^2 m^2 \Delta t + \frac{1}{2}\Delta t r m$$

then, let

$$u'_k = 1 + \frac{1}{2}\sigma^2 m^2 \Delta t + r\Delta t + 2\sqrt{\frac{1}{16}\sigma^4 m^4 \Delta t^2 - \frac{1}{16}\Delta t^2 m^2 r^2} * \cos \frac{k\pi}{N+1}$$

$$u'_k = 1 + \frac{1}{2}(\sigma^2 m^2 \Delta t + r\Delta t + 2\sqrt{\frac{1}{4}\sigma^4 m^4 \Delta t^2 - \frac{1}{4}\Delta t^2 m^2 r^2} * \cos \frac{k\pi}{N+1})$$

From backward difference method we have

$$u_k = 1 + \sigma^2 m^2 \Delta t + r\Delta t + 2\sqrt{\frac{1}{4}\sigma^4 m^4 \Delta t^2 - \frac{1}{4}\Delta t^2 m^2 r^2} * \cos \frac{k\pi}{N+1} > 1$$

So it is always true that

$$u'_k > 1$$

Therefore,

$$1 < T_k = -1 + 2 * \frac{1}{u'_k} < 1$$

We can conclude that the Crank-Nicolson method is unconditionally stable.

6. Exact Solution

By financial theory[2], the price of call option is

$$C(S, t) = N(d_1)S - N(d_2)Ke^{-r(T-t)}$$

$$d_1 = \frac{1}{\sigma\sqrt{T-t}} \left[\ln \left(\frac{S}{K} \right) + \left(r + \frac{\sigma^2}{2} \right) (T-t) \right]$$

$$d_2 = \frac{1}{\sigma\sqrt{T-t}} \left[\ln \left(\frac{S}{K} \right) + \left(r - \frac{\sigma^2}{2} \right) (T-t) \right]$$

$$= d_1 - \sigma\sqrt{T-t}$$

By put-call parity we have the price of put option

$$P(S, t) = Ke^{-r(T-t)} - S + C(S, t)$$

$$= N(-d_2)Ke^{-r(T-t)} - N(-d_1)S$$

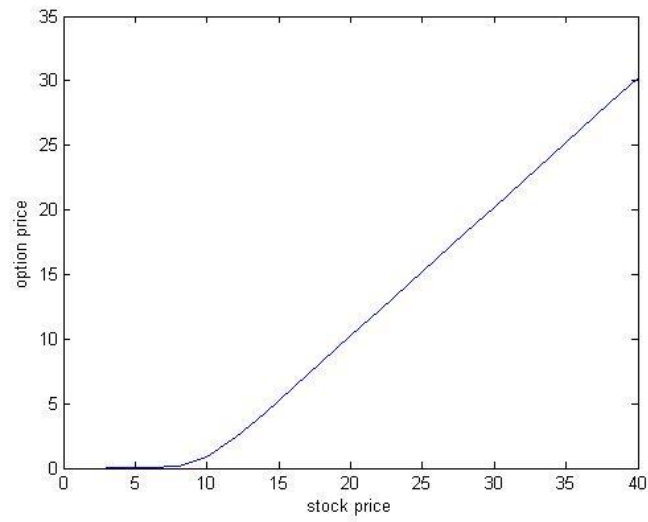
7. Analysis of error and computation speed

We use the following option and grid to test our scheme.

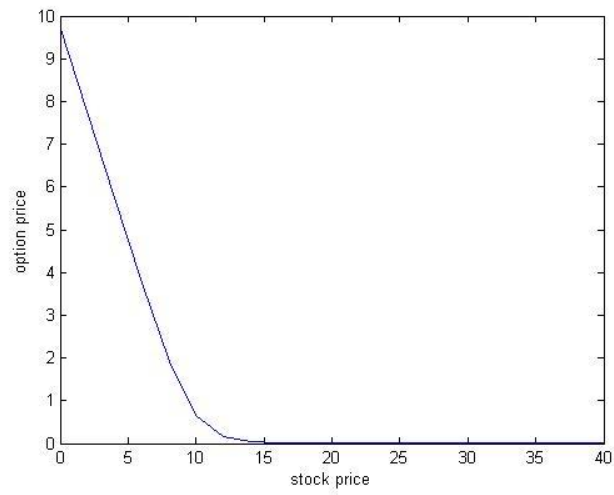
Time to maturity T=0.25. Strike price for the option K=10; Risk-free interest rate r=0.1; Volatility of the market

$\sigma = 0.4$. Max price for call option Smax= 40; Step size for price h=40/20; Step size for time dt=0.25/200;

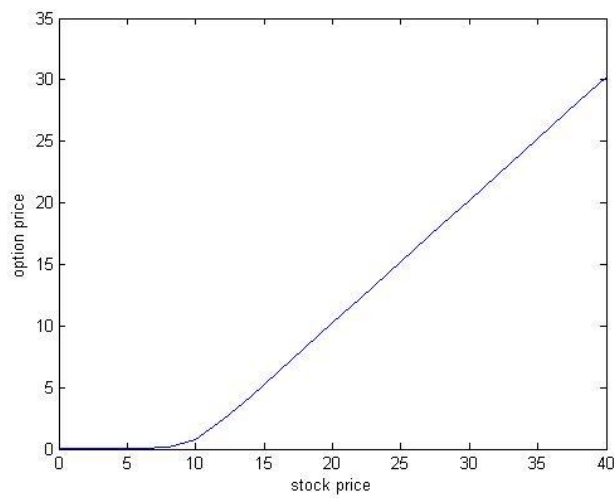
The exact solution for call option is



The exact solution for put option is

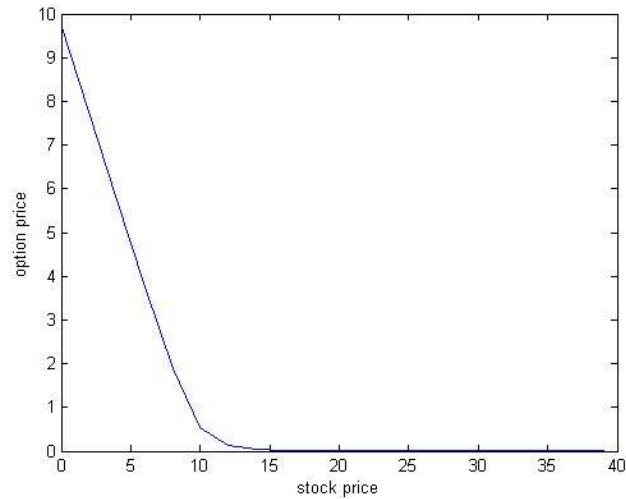


Price for call option from forward difference



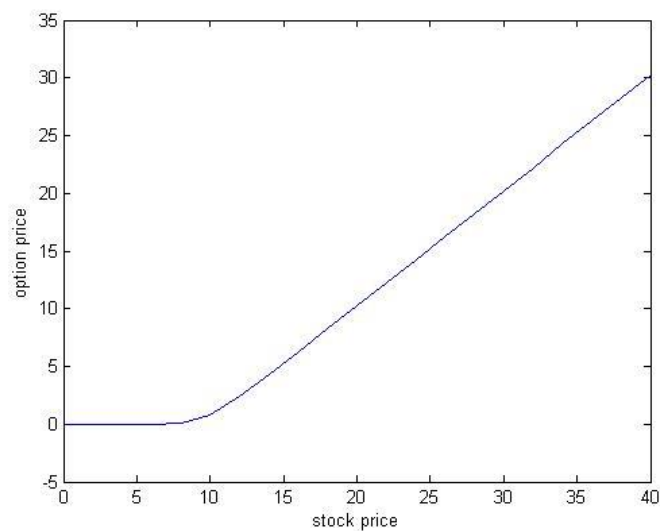
With error=0.1213

Price for put option from forward difference



With error=0.1213

Price for call option from backward difference



With error = 0.1227

CPU time for Jacobi method: 35.6563

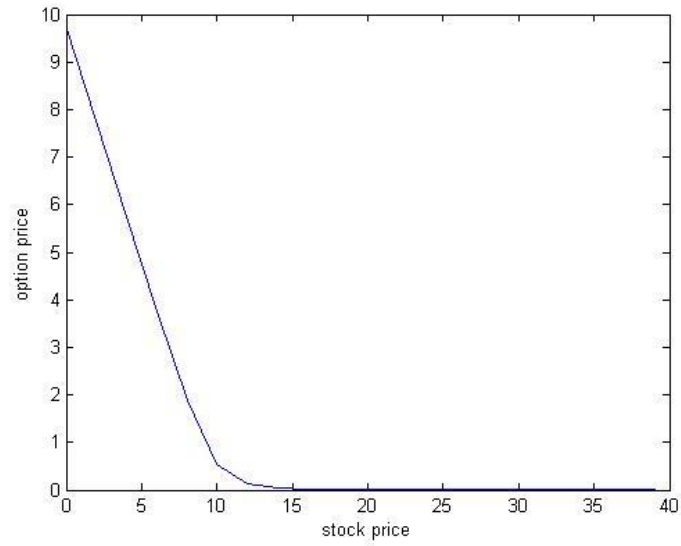
CPU time for Gauss-Seidel method: 19.8438

CPU time for SOR method with relaxation parameter 0.9: 24.7188

We are not going to try to find the optimal relaxation parameter for SOR method as it is too slow and we will not use

this method anyway.

Price for put option from backward difference

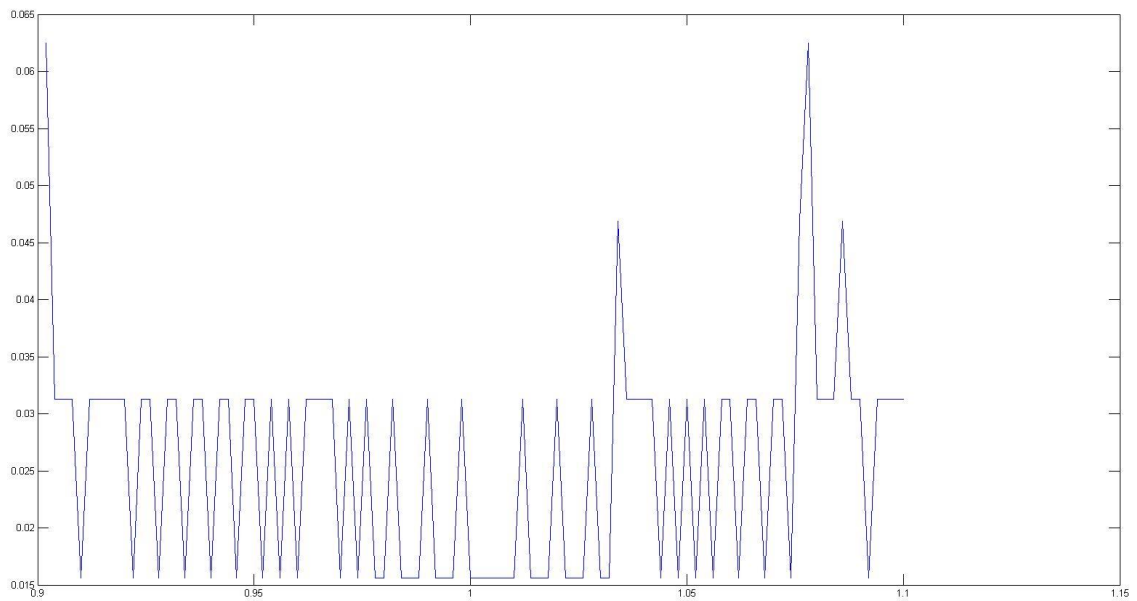


With error =0.1226

CPU time for Jacobi method: 0.1719

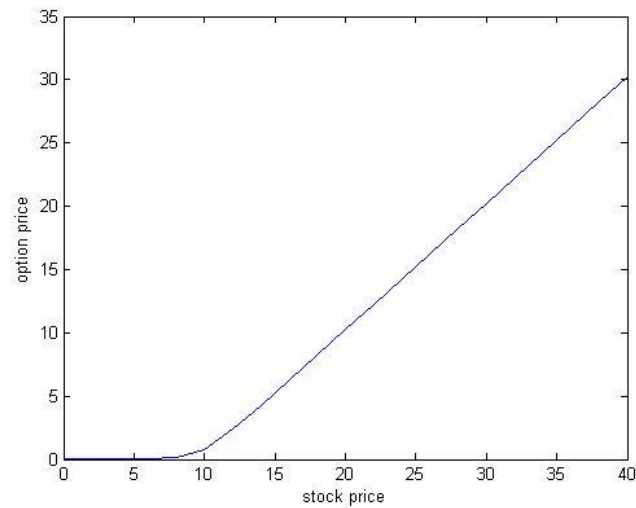
CPU time for Gauss-Seidel method: 0.1719

CPU time for SOR method with relaxation parameter 0.9: 0.2500



From the above graph, we can see that there are multiple relaxation parameters that can be optimal. 1 is one of the optimal relaxation parameters.

Price for call from Crank-Nicolson Method

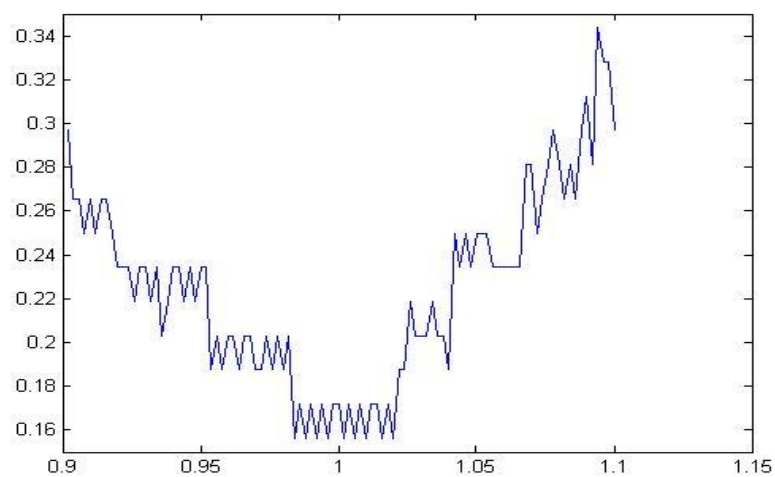


With error =0.1220

CPU time for Jacobi method: 0. 0.1875

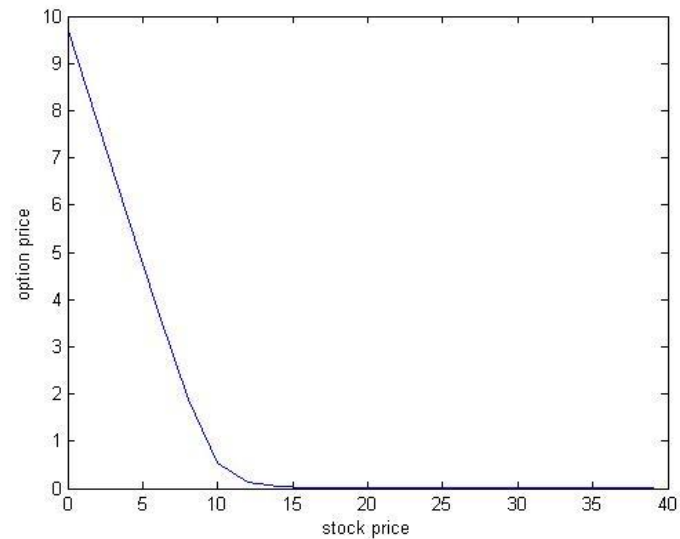
CPU time for Gauss-Seidel method: 0.2188

CPU time for SOR method with relaxation parameter 0.9: 0.3281



From the above graph, we can see that there are multiple relaxation parameter that can be optimal. 1 is not optimal.

Price for put option from Crank-Nicolson Method

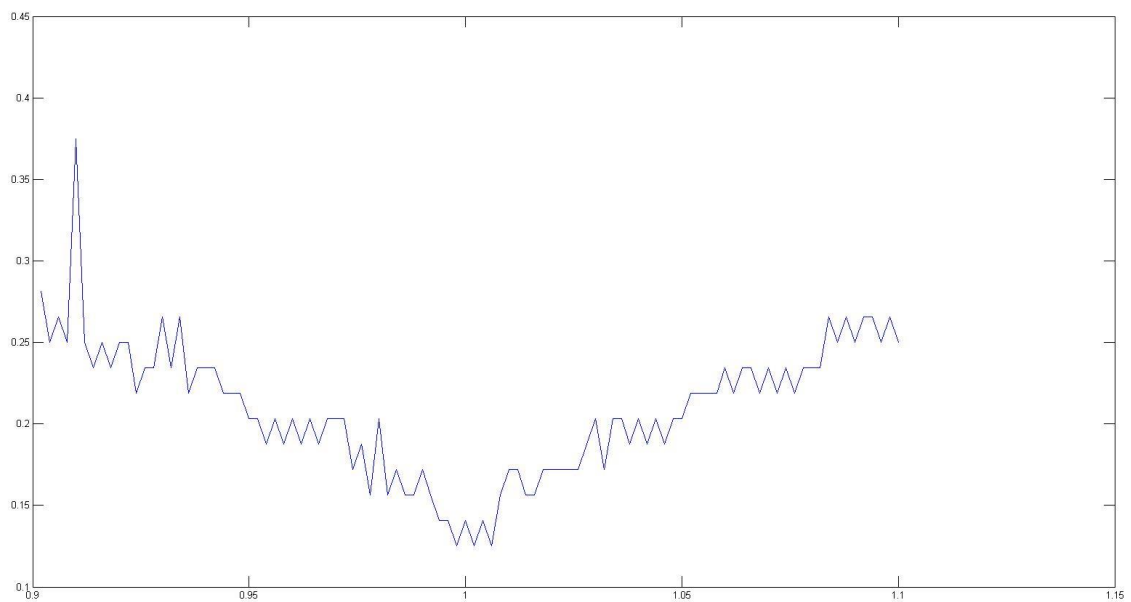


With error =0.1220

CPU time for Jacobi method: 0.1719

CPU time for Gauss-Seidel method: 0.1719

CPU time for SOR method with relaxation parameter 0.9: 0.2813



From the above graph, we can see that there are multiple relaxation parameter that can be optimal. 1 is not optimal.

Reference

1. Black, Fischer, and Myron Scholes. "The Pricing of Options and Corporate Liabilities." *Journal of Political Economy*. Vol. 81, No. 3. May - Jun. (1973): 637-654. Print.
2. Durrett, Richard. *Essentials of Stochastic Processes*. 2nd ed. 2011. New York: Springer, 2011. 198-203. Print.
3. Bradie, Brian. *A Friendly Introduction to Numerical Analysis*. 1st ed. Upper Saddle River, New Jersey: Pearson, 2005. 438-445. Print.

Appendix.

1. Code for exact solution:

```
x=linspace(0,Smax,N+1);
for t=0:Tn
    for m=0:N
        d1(m+1,t+1)=1/(sigma*sqrt(T-t*dt))*(log(m*h/K)+(r+sigma^2/2)*(T-t*dt));
        d2(m+1,t+1)=d1(m+1,t+1)-sigma*sqrt(T-t*dt);

        callex(m+1,t+1)=normcdf(d1(m+1,t+1))*m*h-normcdf(d2(m+1,t+1))*K*exp(-r*(T-t*dt));%exact
        value for call option
        putex(m+1,t+1)=K*exp(-r*(T-t*dt))-m*h+callex(m+1,t+1); %from put-call
        parity %exact value for put option
    end
end
```

2. Code for call option

2.1 Explicit method

```
for m=1:N-1
    alpha(m)=1-sigma^2*m^2*dt-dt*r;
    beta(m)=0.5*(sigma^2*m^2*dt+dt*r*m);
    gamma(m)=0.5*(sigma^2*m^2*dt-r*dt*m);
end
A(1,1:2)=[alpha(1) beta(1)];
A(N-1,N-2:N-1)=[gamma(N-1) alpha(N-1)];
for i=2:N-2
    A(i,i-1:i+1)=[gamma(i) alpha(i) beta(i)];
end

b(1,1:Tn-1)=gamma(1)*0;
for i=1:Tn+1
    VN(i)=Smax-K*exp(-r*(i-1)*dt);
    b(N-1,i)=beta(N-1)*VN(i);
end

%From terminal condition
for m=1:N-1
    if h*m>K
        V(m)=h*m-K;
    else
        V(m)=0;
    end
end
Vt(:,1)=V;
for i=1:Tn
    Vt(:,i+1)=A*Vt(:,i)+b(:,i);
end
```

```
Vfinal(1)=0;
Vfinal(2:N)=Vt(:,Tn+1);
Vfinal(N+1)=VN(Tn+1);
```

2.2 Implicit method

```
for m=1:N-1
    beta(m)=1+sigma^2*m^2*dt+dt*r;
    alpha(m)=0.5*(-sigma^2*m^2*dt+dt*r*m);
    gamma(m)=-0.5*(sigma^2*m^2*dt+r*dt*m);
end
A(1,1:2)=[beta(1) gamma(1)];
A(N-1,N-2:N-1)=[alpha(N-1) beta(N-1)];
for i=2:N-2
    A(i,i-1:i+1)=[alpha(i) beta(i) gamma(i)];
end
b(1,1:Tn-1)=alpha(1)*0;
for i=1:Tn+1
    VN(i)=Smax-K*exp(-r*(i-1)*dt);
    b(N-1,i)=gamma(N-1)*VN(i);
end
%From terminal conditionc
for m=1:N-1
    if h*m>K
        V(m)=h*m-K;
    else
        V(m)=0;
    end
end
Vt(:,1)=V;
%build-in inverse
% t = cputime;
% for i=1:Tn
%     Vt(:,i+1)=A^-1*(Vt(:,i)-b(:,i));
% end
% e = cputime-t

% Jacobi
% t = cputime;
% for i=1:Tn
%     Vt(:,i+1)=jacobi ( A, Vt(:,i)'\-b(:,i)', Vt(:,i)', 10^-9, 10000 );
% end
% e = cputime-t
%
% Gauss-Seidel
% t = cputime;
% for i=1:Tn
%     Vt(:,i+1)=gauss_seidel ( A, Vt(:,i)'\-b(:,i)', Vt(:,i)', 10^-9, 10000 );
```

```

% end
% e = cputime-t
%
% SOR
% for j=1:90
% t = cputime;
% for i=1:Tn
%     Vt(:,i+1)=sor ( A, Vt(:,i) '-b(:,i)', Vt(:,i)',0.1+0.01*j, 10^-9, 10000 );
% end
% e(j) = cputime-t;
% end

Vfinal(1)=0;
Vfinal(2:N)=Vt(:,Tn+1);
Vfinal(N+1)=VN(Tn+1);

```

2.3 Crank-Nicolson Method

```

%construct matrix
for m=1:N
    u(m)=dt/4*(sigma^2*m^2-r*m);
    v(m)=dt/2*(sigma^2*m^2+r);
    w(m)=dt/4*(sigma^2*m^2+r*m);
end
A(1,1)=1-v(1);
A(1,2)=w(1);
A(N-1,N-2)=u(N-1);
A(N-1,N-1)=1-v(N-1);
B(1,1)=1+v(1);
B(1,2)=-w(1);
B(N-1,N-2)=-u(N-1);
B(N-1,N-1)=1+v(N-1);
for m=2:N-2
    A(m,m-1)=u(m);
    A(m,m)=1-v(m);
    A(m,m+1)=w(m);
    B(m,m-1)=-u(m);
    B(m,m)=1+v(m);
    B(m,m+1)=-w(m);
end
%From terminal conditionc
for m=1:N-1
    if h*m>K
        V(m,Tn+1)=h*m-K;
    else
        V(m,Tn+1)=0;
    end
end
end

```

```

%From boundary condition of put
% S goes to inf
b(1,1:Tn+1)=0;
c(1,1:Tn+1)=0;
% S is 0
for i=1:Tn+1
    VN(i)=Smax-K*exp(-r*(Tn-(i-1))*dt);
    b(N-1,i)=w(N-1)*VN(i);
    c(N-1,i)=-w(N-1)*VN(i);
end
%build-in inverse
% for i=1:Tn
%     V(1:N-1,Tn+1-i)=B^-1*(A*V(1:N-1,Tn+2-i)+b(:,Tn+2-i)-c(:,Tn+1-i));
% end
% Jacobi
% t = cputime;
% for i=1:Tn
%     V(1:N-1,Tn+1-i)=jacobi ( B, (A*V(1:N-1,Tn+2-i)+b(:,Tn+2-i)-c(:,Tn+1-i))',
V(1:N-1,Tn+2-i)', 10^-9, 10000 );
% end
% e = cputime-t
% Gauss-Seidel
% t = cputime;
% for i=1:Tn
%     V(1:N-1,Tn+1-i)=gauss_seidel ( B, (A*V(1:N-1,Tn+2-i)+b(:,Tn+2-i)-c(:,Tn+1-i))',
V(1:N-1,Tn+2-i)', 10^-9, 10000 );
% end
% e = cputime-t
%
% SOR
% for j=1:90
% t = cputime;
% for i=1:Tn
%     V(1:N-1,Tn+1-i)=sor ( B, (A*V(1:N-1,Tn+2-i)+b(:,Tn+2-i)-c(:,Tn+1-i))',
V(1:N-1,Tn+2-i)',0.1+0.01*j, 10^-9, 10000 );
% end
% e(j) = cputime-t;
% end

Vfinal(1)=0;
Vfinal(2:N)=V(1:N-1,1);
Vfinal(N+1)=VN(1);

```

3. Code for put option

3.1 Explicit Method

```
for m=1:N-1
    alpha(m)=1-sigma^2*m^2*dt-dt*r;
    beta(m)=0.5*(sigma^2*m^2*dt+dt*r*m);
    gamma(m)=0.5*(sigma^2*m^2*dt-r*dt*m);
end
A(1,1:2)=[alpha(1) beta(1)];
A(N-1,N-2:N-1)=[gamma(N-1) alpha(N-1)];
for i=2:N-2
    A(i,i-1:i+1)=[gamma(i) alpha(i) beta(i)];
end
% for i=2:N-2
% A(i,:)=A(i,:)./sum(A(i,:));
% end
%From boundary condition of put
b(N-1,:)=beta(N-1)*0;
for i=1:Tn+1
    V0(i)=K*exp(-r*(i-1)*dt);
    b(1,i)=gamma(1)*V0(i);
end
%From terminal conditionc
for m=1:N-1
    if h*m<K
        V(m)=K-h*m;
    else
        V(m)=0;
    end
end
Vt(:,1)=V;
for i=1:Tn
    Vt(:,i+1)=A*Vt(:,i)+b(:,i);
end
Vfinal(1)=V0(Tn+1);
Vfinal(2:N)=Vt(:,Tn+1);
Vfinal(N+1)=0;
```

3.2 Implicit Method

```
for m=1:N-1
    alpha(m)=0.5*(-sigma^2*m^2*dt+dt*r*m);
    beta(m)=1+sigma^2*m^2*dt+dt*r;
    gamma(m)=-0.5*(sigma^2*m^2*dt+r*dt*m);
end
A(1,1:2)=[beta(1) gamma(1)];
A(N-1,N-2:N-1)=[alpha(N-1) beta(N-1)];
for i=2:N-2
    A(i,i-1:i+1)=[alpha(i) beta(i) gamma(i)];
```

```

end
% for i=2:N-2
%   A(i,:)=A(i,:)./sum(A(i,:));
% end
%From boundary condition of put
b(N-1,i)=gamma(N-1)*0;
for i=1:Tn+1
    V0(i)=K*exp(-r*(i-1)*dt);
    b(1,i)=alpha(1)*V0(i);
end
%From terminal conditionc
for m=1:N-1
    if h*m<K
        V(m)=K-h*m;
    else
        V(m)=0;
    end
end
Vt(:,1)=V;
% build-in inverse

% for i=1:Tn
%   Vt(:,i+1)=A^-1*(Vt(:,i)-b(:,i));
% end
% Jacobi
% t = cputime;
% for i=1:Tn
%   Vt(:,i+1)=jacobi ( A, Vt(:,i)'-b(:,i)', Vt(:,i)', 10^-9, 10000 );
% end
% e = cputime-t
% Gauss-Seidel
% t = cputime;
% for i=1:Tn
%   Vt(:,i+1)=gauss_seidel ( A, Vt(:,i)'-b(:,i)', Vt(:,i)', 10^-9, 10000 );
% end
% e = cputime-t
% SOR
% for j=1:90
% t = cputime;
% for i=1:Tn
%   Vt(:,i+1)=sor ( A, Vt(:,i)'-b(:,i)', Vt(:,i)',0.1+0.01*j, 10^-9, 10000 );
% end
% e(j) = cputime-t;
% end

Vfinal(1)=V0(Tn+1);
Vfinal(2:N)=Vt(:,Tn+1);
Vfinal(N+1)=0;

```

3.3 Crank-Nicolson Method

```
%construct matrix
for m=1:N
    u(m)=dt/4*(sigma^2*m^2-r*m);
    v(m)=dt/2*(sigma^2*m^2+r);
    w(m)=dt/4*(sigma^2*m^2+r*m);
end
A(1,1)=1-v(1);
A(1,2)=w(1);
A(N-1,N-2)=u(N-1);
A(N-1,N-1)=1-v(N-1);
B(1,1)=1+v(1);
B(1,2)=-w(1);
B(N-1,N-2)=-u(N-1);
B(N-1,N-1)=1+v(N-1);
for m=2:N-2
    A(m,m-1)=u(m);
    A(m,m)=1-v(m);
    A(m,m+1)=w(m);
    B(m,m-1)=-u(m);
    B(m,m)=1+v(m);
    B(m,m+1)=-w(m);
end
%From terminal conditionc
for m=1:N-1
    if h*m<K
        V(m,Tn+1)=K-h*m;
    else
        V(m,Tn+1)=0;
    end
end
%From boundary condition of put
% S goes to inf
b(N-1,1:Tn+1)=w(N-1)*0;
c(N-1,1:Tn+1)=-w(N-1)*0;
% S is 0
% for j=1:90
% t = cputime;
% for i=1:Tn
%     V(1:N-1,Tn+1-i)=sor ( B, (A*V(1:N-1,Tn+2-i)+b(:,Tn+2-i)-c(:,Tn+1-i))',
V(1:N-1,Tn+2-i)',0.1+0.01*j, 10^-9, 10000 );
% end
% e(j) = cputime-t;
% end
```

```

%build-in inverse
% for i=1:Tn
%     V(1:N-1,Tn+1-i)=B^-1*(A*V(1:N-1,Tn+2-i)+b(:,Tn+2-i)-c(:,Tn+1-i));
% end
% Jacobi
% t = cputime;
% for i=1:Tn
%     V(1:N-1,Tn+1-i)=jacobi ( B, (A*V(1:N-1,Tn+2-i)+b(:,Tn+2-i)-c(:,Tn+1-i))',
V(1:N-1,Tn+2-i)', 10^-9, 10000 );
% end
% e = cputime-t
% Gauss-Seidel
% t = cputime;
% for i=1:Tn
%     V(1:N-1,Tn+1-i)=gauss_seidel ( B, (A*V(1:N-1,Tn+2-i)+b(:,Tn+2-i)-c(:,Tn+1-i))',
V(1:N-1,Tn+2-i)', 10^-9, 10000 );
% end
% e = cputime-t
% SOR
% for j=1:90
% t = cputime;
% for i=1:Tn
%     V(1:N-1,Tn+1-i)=sor ( B, (A*V(1:N-1,Tn+2-i)+b(:,Tn+2-i)-c(:,Tn+1-i))',
V(1:N-1,Tn+2-i)',0.1+0.01*j, 10^-9, 10000 );
% end
% e(j) = cputime-t;
% end

Vfinal(1)=V0(1);
Vfinal(2:N)=V(1:N-1,1);
Vfinal(N+1)=0;

```