**Problem 1)**

First, we have the differential equation for the perpetual American put

Now we want non-dimensionalize the problem with the relationships

Thus we have the first order and second order derivative to be

Plug the derivatives back into the Black-Scholes equation, then we have

Divide both side of the equation by K, then we can obtain

Then for the boundary conditions, since K is a non-zero positive constant, thus

**Problem 2)**

To obtain the analytic solution of the pricing problem, first we can divide both side of the Black-Scholes equation by to obtain the form of Cauchy-Euler equation

Then we have a trial solution and its derivatives

Substituting this trial solution into the original equation, then

Then divide both side by we can have an equation only with respect to m

Solve for the quadratic equation we can obtain two distinct roots

Thus we have the solution to be of the form

Since we know the boundary condition , and, so

Then we have the equation system with respect to and

Thus we have

Thus we have the analytic solution to be

**Problem 3)**

Applying the Riccati transformation, we have:

According to Trapezoidal Rule:

Let

Then

Here, we choose

Similarly, we have

The boundary condition is u(X) = 0, where X is a large enough number. Since u(X) = R(X)\*v(X) + w(X), we have R(X) = 0 and w(X) = u(X). Knowing the initial values of R and w, we can iteratively solve all values using the equations above.

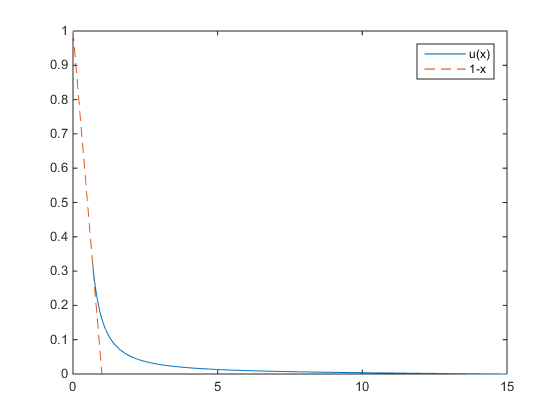
According to the free boundary condition, u(s0) = 1-s0 and u’(s0) = -1.

Let and . So the x subject to is s0. With X=15 and N = 106, the numerical solution of s0 is 0.6492.

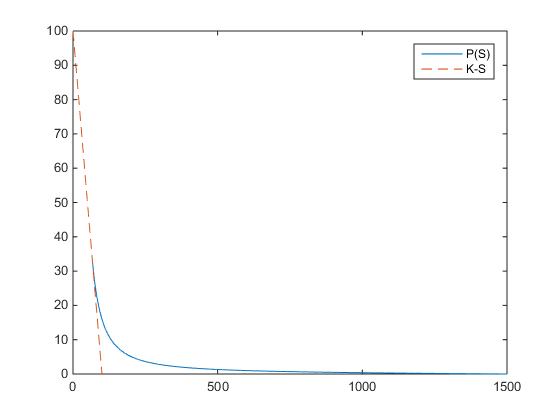
Similarly, for v, we have

So we can integrate the equation for v from s0 to X and find values of u(x) at the mesh points.

The plot of u(x) is shown as below.



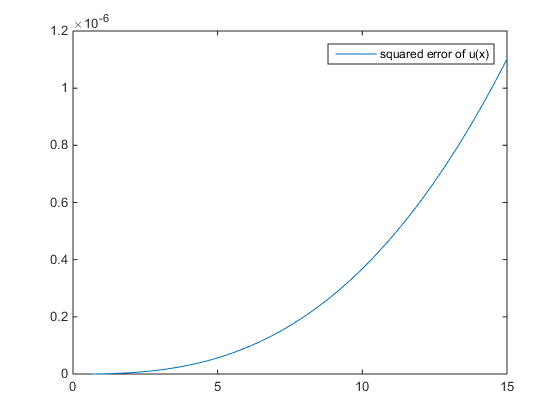
Since , the plot of P(s) is as below.



**Problem 4)**

With X=15 and N = 106, the numerical solution of s0 is 0.6492, which is the same as the analytic solution in Problem 2.

The squared error of u(x) is shown as below and the Mean Squared Error of u(x) is 3.11\*10-7.

****

**Problem 5)**

The error of u(x) is determined by the error of s0 and the error caused by Riccati transformation. The second one is inevitable so we only need to minimize the error of s0.

The number of effective decimal digits of s0 depends on mesh size . For example, in order to get 4 effective decimal digits of s0, has to be less than or equal to 10-4.

The number of calculations . Assuming no numerical truncation error, when X and N both approach infinity, the numerical solution s0 approaches the accurate analytic solution. So we can gradually increase the value of X until is zero. To reduce runtime, X(i+1) – X(i) is large at the beginning and then decreases.

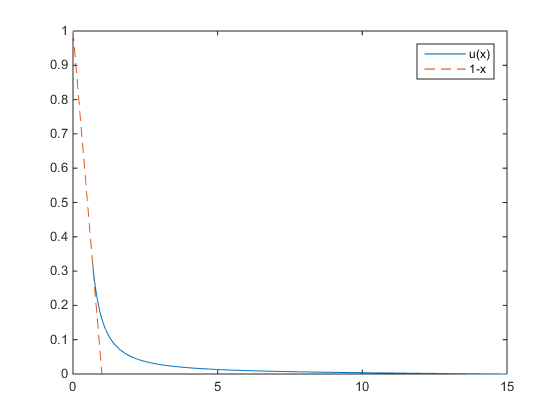
For example, the process of getting a 4-decimal-digits accurate s0 is shown as below.

|  |  |  |  |
| --- | --- | --- | --- |
| X | N | S0 |  |
| 2 | 2\*104 | 0.6605 |  |
| 10 | 105 | 0.6493 | -0.0112 |
| 15 | 1.5\*105 | 0.6492 | -0.0001 |
| 17 | 1.7\*105 | 0.6492 | 0 |

**Problem 6)**

Since we didn’t take derivative of σ, the σ in the equations can be simply replaced by σ\*x0.1.

The u(x) of CEV put is shown as below.

****

The value of s0 is 0.6540 now, which is slightly different.

**Appendix: Matlab code**

% boundary conditions

uX = 0;

RX = 0;

wX = uX;

% parameters

X = 15;

N = 1000000;

r = 0.05;

q = 0.02;

sigma = 0.2;

K = 100;

% compute R, w iteratively

R = zeros(1,N);

w = zeros(1,N);

v = zeros(1,N);

phi = zeros(1,N);

dx = X/N;

x = dx:dx:X;

sigma = sigma .\* x.^0.1;

% sigma = sigma \* ones(1, N);

% set boundary condition

R(N) = RX;

w(N) = wX;

for i = N-1:-1:1

% aR^2 + bR + c = 0

a = r./sigma(i).^2./x(i).^2;

b = -1./dx - (r-q)./sigma(i).^2./x(i);

c = R(i+1)./dx - (r-q).\*R(i+1)./sigma(i+1).^2./x(i+1) + r.\*R(i+1).^2./sigma(i+1).^2./x(i+1).^2 - 1;

R(i) = (-b - sqrt(b.^2 - 4.\*a.\*c)) ./ a ./ 2;

% a \* w(n) = b \* w(n+1)

a = 1./dx - r.\*R(i)./sigma(i).^2./x(i).^2;

b = 1./dx + r.\*R(i+1)./sigma(i+1).^2./x(i+1).^2;

w(i) = b .\* w(i+1) ./ a;

% compute phi(x)

phi(i) = -R(i) + w(i) - (1 - x(i));

% pause;

end

% find S0 s.t. phi(S) is almost 0

S0 = 0;

i0 = 0;

for i = 2:1:N

if (phi(i-1) \* phi(i) < 0)

if (abs(phi(i-1)) < abs(phi(i)))

S0 = x(i-1);

i0 = i-1;

else

S0 = x(i);

i0 = i;

end

end

end

% compute v from S0 to X

v(i0) = -1;

for i = i0:1:N-1

% a .\* v(n) = b .\* v(n+1) + c

a = 1./dx + (r .\* R(i) - (r-q).\*x(i))./sigma(i).^2./x(i).^2;

b = 1./dx - (r .\* R(i+1) - (r-q).\*x(i+1))./sigma(i+1).^2./x(i+1).^2;

c = -r.\*w(i+1)./sigma(i+1).^2./x(i+1).^2 - r.\*w(i)./sigma(i).^2./x(i).^2;

v(i+1) = (a .\* v(i) - c) ./ b;

end

u = R .\* v + w;

% plot u(x)

plot(x(i0:N), u(i0:N));

hold on;

plot(x(1:1/dx), 1-x(1:1/dx), '--');

legend('u(x)', '1-x');

% plot P(s)

figure;

plot(x(i0:N)\*K, u(i0:N)\*K);

hold on;

plot(x(1:1/dx)\*K, (1-x(1:1/dx))\*K, '--');

legend('P(S)', 'K-S');

% u\_star = 0.1576933168 \* x.^(-1.850781059);

% SE = (u(i0:N) - u\_star(i0:N)).^2;

% MSE = sum(SE) / (N-i0);

% figure;

% plot(x(i0:N), SE);

% legend('squared error of u(x)');