

FE Project
IEORE 4706 Foundations for Financial Engineering

Ahmad Shayaan¹
as5948

{ahmad.shayaan@columbia.edu}@columbia.edu
Columbia University
December 26, 2019

¹All authors contributed equally to this work.

Contents

Perpetual American Options	2
Exercise Region.	2
Solution of the differential equation	2
Boundedness of value of option.	3
Optimal Exercise	3
Pricing using CRR model	6
Building the tree	6
Pricing European and American options.	6
Exercise Frontier of an American Put.	8
Other pricing model	10
Jarrow-Rudd model	10
Titan model	11
Jarrow Rudd Risk neutral measure.	12
CRR model with drift	13
Finite difference method for American option	15
Generating the matrix	15
Discretization Scheme	15
Stability of explicit scheme	16
Implicit Euler Scheme	17

Perpetual American Options

Exercise region

For a put option the pay-off is only positive if the value of the underlying security is less than the strike price. Thus if x^* is the optimal price at which we exercise the option then, necessarily for a positive payoff x^* has to be less than K . This means that the range of values for x^* is $(0, K]$, 0 is not included in the range because if the value of the underlying security hit 0 it cannot be traded in the market. We also know that the yield of a put option increases with a decrease in the value of the underlying security. Thus if it is optimal to exercise the option at x^* it will always be optimal to exercise the option for all the prices of the security that are below x^* . Thus the range of the price of the security (S) is $(0, x^*]$. Again 0 is not included because if the value of the security becomes zero it cannot be traded on the market.

Solution of the differential equation

If $x > x^*$ this will mean that the exercise is not optimal, and the payoff of the option is less than the value i.e. $v(x) - h(x) > 0$. Thus from the Hamilton-Jacobi-Bellman equation v must satisfy the following differential equation.

$$\begin{aligned} rv(x) - \mu xv'(x) - \frac{1}{2}\sigma^2 x^2 v''(x) &= 0 \\ x^2 v''(x) + \frac{2\mu x}{\sigma^2} v'(x) - \frac{2r}{\sigma^2} v(x) &= 0 \end{aligned}$$

Let the solution to the above general equation be $v(x) = x^\lambda$. Then $v'(x) = \lambda x^{(\lambda-1)}$ and $v''(x) = \lambda(\lambda-1)x^{(\lambda-2)}$. Thus the above equation becomes.

$$\begin{aligned} \lambda(\lambda-1)x^\lambda + \frac{2\mu}{\sigma^2} \lambda x^\lambda - \frac{2r}{\sigma^2} x^\lambda &= 0 \\ \lambda(\lambda-1) + \frac{2\mu}{\sigma^2} \lambda - \frac{2r}{\sigma^2} &= 0 \\ \sigma^2 \lambda^2 + (2\mu - \sigma^2)\lambda - 2r &= 0 \\ \lambda &= \frac{-(2\mu - \sigma^2) \pm \sqrt{(2\mu - \sigma^2)^2 + 8\sigma^2 r}}{2\sigma^2} \end{aligned}$$

Let m and n be the solution to the equation. Then

$$m = \frac{-(2\mu - \sigma^2) - \sqrt{(2\mu - \sigma^2)^2 + 8r\sigma}}{2\sigma^2}$$

$$n = \frac{-(2\mu - \sigma^2) + \sqrt{(2\mu - \sigma^2)^2 + 8r\sigma}}{2\sigma^2}$$

Where $m \leq 0$ and since $8r\sigma^2 > 0$, $n \geq 0$. Thus $v(x)$ is given by.

$$v(x) = Ax^m + Bx^n$$

$$m = \frac{-(2\mu - \sigma^2) - \sqrt{(2\mu - \sigma^2)^2 + 8r\sigma}}{2\sigma^2}$$

$$n = \frac{-(2\mu - \sigma^2) + \sqrt{(2\mu - \sigma^2)^2 + 8r\sigma}}{2\sigma^2}$$

Boundedness of value of option

Let us assume that $v(x)$ is not bounded then there definitely exists a x such that $v(x) > K$. We sell the put whose value is K and put the money in the bank at time t . Then there are two cases.

Case 1. The put is exercised at time τ . If this happens then we take the money out of the bank to fund the exercise of the put. Thus in this case the profit is given by.

$$-(K - x^*)^+ + K/d(0, \tau) > 0$$

Case 2. The put is not exercised till maturity T . Then in that case the payoff is.

$$-(K - x^*)^+ + K/d(0, \tau) > 0$$

We have an arbitrage opportunity in both the cases. Since there are no arbitrage opportunities in the market thus are assumption that $v(x) > K$ is wrong. Thus we can conclude that $v(x) \leq K$ and bounded.

We have proved that $v(x) \leq K$ which implies that $Ax^m + B \leq K \forall x$.

$$\begin{aligned} \lim_{x \rightarrow \infty} v(x) &\leq K \\ \implies \lim_{x \rightarrow \infty} Ax^m + Bx^n &\leq K \\ \implies \lim_{x \rightarrow \infty} Ax^m + \lim_{x \rightarrow \infty} Bx^n &\leq K & \because m < 0 \\ \implies \lim_{x \rightarrow \infty} Bx^n &\leq K \\ \implies B &= 0 & \because n > 1 \end{aligned}$$

Optimal exercise

From the second section we can conclude that any $x \leq x^*$ we have $v(x) = h(x) \implies Ax^m = (K - x)$. Since value of a option are continuous we can differentiate wrt x .

$$\begin{aligned}
& \frac{dv(x)}{x} \Big|_{x=x^*} = \frac{dh(x)}{dx} \Big|_{x=x^*} \\
\Rightarrow & \frac{d(AX^m)}{dx} \Big|_{x=x^*} = \frac{d(K-x)}{dx} \Big|_{x=x^*} \\
\Rightarrow & mA x^{*(m-1)} = -1 \\
\Rightarrow & mA x^{*m} = -x^* \\
\Rightarrow & m(K-x^*) = -x^* \\
\Rightarrow & x^* = \frac{mK}{(m-1)}
\end{aligned}$$

Substituting the value of x^* in the equation $Ax^{*m} = (K-x^*)$.

$$\begin{aligned}
A &= \frac{(K-x^*)}{x^{*m}} \\
A &= \frac{(K - \frac{mK}{(m-1)})}{\left[\frac{mK}{(m-1)}\right]^m}
\end{aligned}$$

From the above results we can conclude that it is optimal to exercise the option when

$$x \leq \frac{mK}{(m-1)} \leq K.$$

From the previous section we know that m is given by $m = \frac{-(2\mu - \sigma^2) - \sqrt{(2\mu - \sigma^2)^2 + 8\sigma r}}{2\sigma^2}$. We now need to find the limit when $r \rightarrow 0$.

$$\begin{aligned}
& \lim_{r \rightarrow 0} m \\
&= \lim_{r \rightarrow 0} m = \frac{-(2\mu - \sigma^2) - \sqrt{(2\mu - \sigma^2)^2 + 8\sigma r}}{2\sigma^2} \\
&= 0
\end{aligned}$$

We now need to find the limit of x^* as r tends to zero.

$$\begin{aligned}
& \lim_{r \rightarrow 0} x^* \\
&= \lim_{r \rightarrow 0} \frac{mK}{m-1} \\
&= \lim_{m \rightarrow 0} \frac{mK}{m-1} \\
&= 0
\end{aligned}$$

Now we need to find A as r tends to zero.

$$\begin{aligned}
& \lim_{r \rightarrow 0} A \\
&= \lim_{m \rightarrow 0} A \\
&= \lim_{m \rightarrow 0} \frac{-K}{m-1} \left[\frac{m-1}{mK} \right]^m \\
\Rightarrow \lim_{m \rightarrow 0} \log A &= \lim_{m \rightarrow 0} \log \frac{-K}{m-1} + \lim_{m \rightarrow 0} m \log \frac{m-1}{mK} \\
\Rightarrow \lim_{m \rightarrow 0} \log A &= \log K + \lim_{m \rightarrow 0} \frac{\log \frac{m-1}{mK}}{\frac{1}{m}} \\
\Rightarrow \lim_{m \rightarrow 0} \log A &= \log K \\
\Rightarrow \lim_{m \rightarrow 0} A &= K \\
\Rightarrow \lim_{r \rightarrow 0} A &= K
\end{aligned}$$

Pricing using CRR model

In this question we had use the Cox-Ross-Rubinstein model valuation for valuation of options. The CRR model is a discrete time pricing model in which the evolution of the risky asset is given by.

$$S_{t_{i+1}} = S_{t_i} U^i$$

Where U^i is the up state (u) or the down state (d).

$$u := e^{\sigma \sqrt{\frac{T}{n}}}$$

$$d := e^{-\sigma \sqrt{\frac{T}{n}}}$$

Where σ is the volatility of the underlying security, T is the time at which the security matures and n is the depth of the tree.

Building the tree

We had to create the binomial price tree for the risky asset. The tree is generated by using a matrix of size $(n + 1) \times (n + 1)$, where n is the depth of the tree. The matrix is filled by working forward from the date of valuation to the date of maturity of the underlying security. At each step, it is assumed that the underlying security can move up by a factor of u or down by a factor of d. The code used to generate the price tree of the underlying security is shown below.

```
1 def buildTree(self):  
    for i in range(self.n+1):  
3         for j in range(i+1):  
            self.tree_security[i][j] = self.security_price * math.pow(  
                self.u, j) * math.pow(self.d, i-j)  
5
```

Pricing European and American options

We had to come up with a recursive formula for valuation of European and American options at every node of the tree. At the terminals nodes i.e. at the maturity of the option, the option value is simply the exercise value.

$\max[(S_T - K), 0]$ American and European call option

$\max[(K - S_T), 0]$ American and European put option

Once the valuation is computed at the terminal nodes, the valuation at every other node can be computed as the discounted expected value of the pay off under the risk neutral measure. In CRR model the risk-neutral measure is given by.

$$q := \mathbb{Q}[U^i = u] = \frac{e^{R_n^T} - d}{u - d}$$

And the value of the option at each node for a European options are given by.

$$S_{t_i} = e^{-rT/n} [qS_{t_{i+1}}\{U^i = u\} + (1 - q)S_{t_{i+1}}\{U^i = D\}]$$

S in the above recursion represents the price of the option. With America options we have the choice of exercising early and the value at every node is given by.

$$S_{t_i} = \max[(SP_{t_i} - K), e^{-rT/n} [qS_{t_{i+1}}\{U^i = u\} + (1 - q)S_{t_{i+1}}\{U^i = D\}]] \text{ American Call}$$

$$S_{t_i} = \max[(K - SP_{t_i}), e^{-rT/n} [qS_{t_{i+1}}\{U^i = u\} + (1 - q)S_{t_{i+1}}\{U^i = D\}]] \text{ American Put}$$

SP in the above recursion is the price of the underlying security which we get from the tree that was computed early. The code that was used to compute the valuation of the American and European options is shown below.

```

2  def computePrice(self):
3      for i in range(self.n-1, -1, -1):
4          for j in range(i+1):
5              if self.type == 'European Call':
6                  self.tree_option[i][j] = math.exp(-1*self.rate*self.deltaT)
7                      * ((1-self.p) * self.tree_option[i+1][j]
8                      + (self.p) * self.tree_option[i+1][j+1])
9
10             elif self.type == 'European Put':
11                 self.tree_option[i][j] = math.exp(-1*self.rate*self.deltaT)
12                     * ((1-self.p) * self.tree_option[i+1][j]
13                     + (self.p) * self.tree_option[i+1][j+1])
14
15             elif self.type == 'American Call':
16                 self.tree_option[i][j] = max(self.tree_security[i][j] -
17                     self.strike_price,
18                     math.exp(-1*self.rate*self.deltaT)
19                         * ((1-self.p) * self.tree_option[i+1][j]
20                         + (self.p) * self.tree_option[i+1][j+1]))
21
22             elif self.type == 'American Put':
23                 self.tree_option[i][j] = max(self.strike_price
24                     - self.tree_security[i][j],
25                     math.exp(-1*self.rate*self.deltaT)
26                         * ((1-self.p) * self.tree_option[i+1][j]
27                         + (self.p) * self.tree_option[i+1][j+1]))
28
29             #Finding the exercise frontier
30             if self.tree_option[i][j] == (self.strike_price - self.
tree_security[i][j]):

```


We compared the values of American call and put when the interest rate are taken to zero. The value of the American call option decreased when the interest rate is brought down to zero. Whereas the value of the American option increased when we brought the interest down to zero.

From the American call put party we know that.

$$S_t - K \leq C_t^A(T, K, S) - P_t^A(T, K, S) \leq S_t - Kd(t, T)$$

When the interest go down to zero we get that.

$$C_t^A(T, K, S) - P_t^A(T, K, S) = S_t - K$$

In the implementation of your model we have set the initial value of the security and strike to be the same, thus at time $t = 0$ we get the values of the call and put option to be the same.

$$C_0^A(T, K, S) = P_0^A(T, K, S)$$

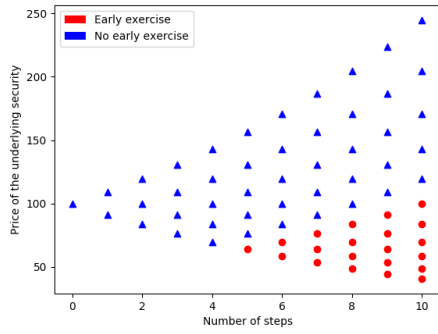
In the cases where the initial value of the security and strike are not same. The value of call and put options are related as follows.

$$C_0^A(T, K, S) = P_0^A(T, K, S) + S_0 - K$$

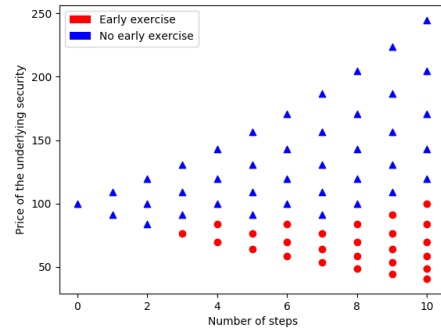
From the above relation we can conclude that if the strike is selected such that it is greater than the initial value of the security then the value of American call will be less than an American put when interest rates are zero and vice-versa.

Exercise frontier of an American put

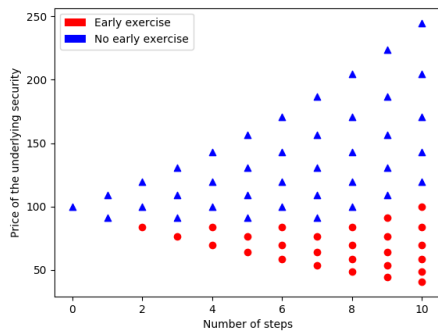
Once the valuation of the American put option had been done we had to find the optimal exercise frontier of the American Put and study its relation to the interest rate and the depth of the tree in CRR model. We plotted the exercise frontier for different values of depth and interest rate.



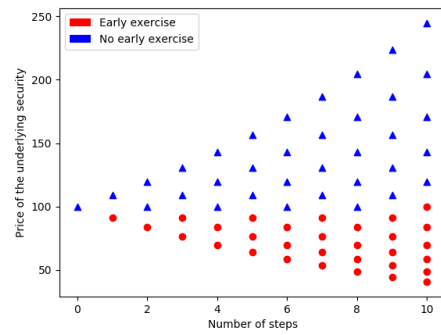
(a) 0.1% interest



(b) 5% interest



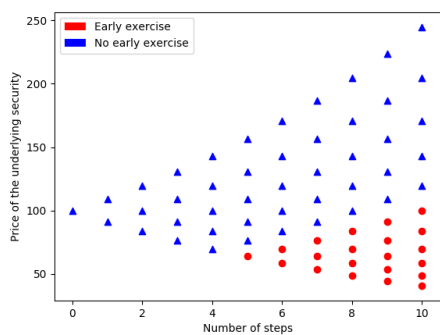
(c) 10% interest



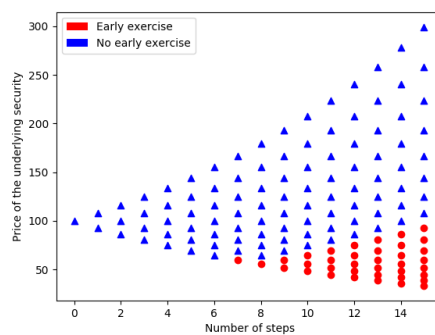
(d) 15% interest

Figure 1: Interest rate variation in a tree of depth 10

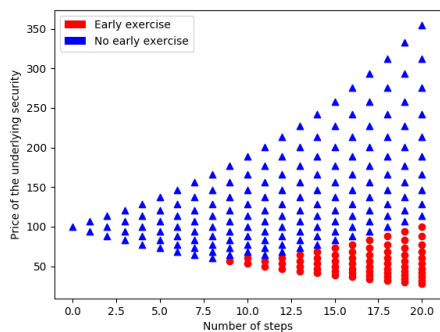
It can be observed from the figures that for the same depth the exercise frontier moves up in the tree i.e. when we increase the interest rate we can optimally exercise an American option earlier.



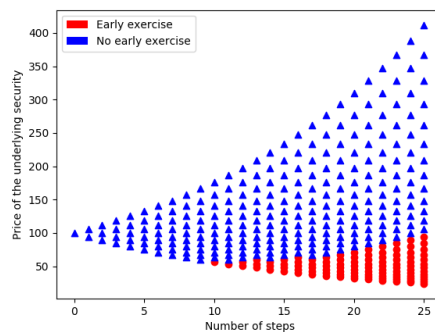
(a) Tree with depth 10



(b) Tree with depth 15



(c) Tree with depth 20



(d) Tree with depth 25

Figure 2: Depth variation with interest rate 0.1%

From the figures we can observe that with the variation of the depth of the tree for the same interest rate, the exercise frontier gets delayed. This means that in the depth of the tree is increased the points of early exercise move further down in the tree.

Other pricing model

We also went ahead and implemented other popular pricing models like the Jarrow-Rudd model, Titan model, Jarrow-Rudd Risk Neutral, and CRR model with drift.

Jarrow-Rudd model

The Jarrow-Rudd model(Y_t) is based on the log-transformation of the binomial model. The figure below shows the exercise frontier for an American option priced using a Jarrow-Rudd model.

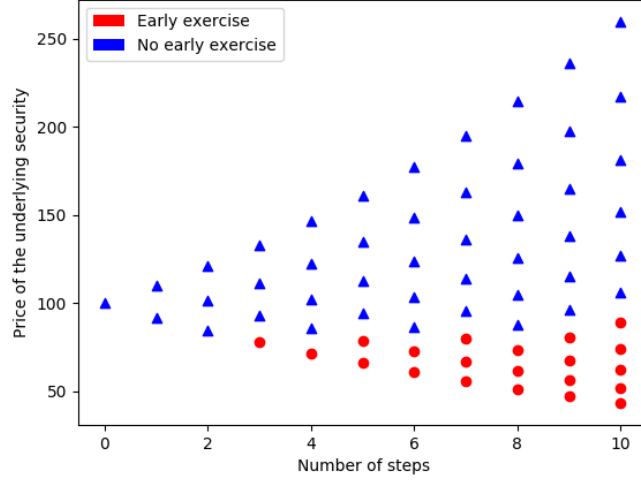


Figure 3: Exercise frontier obtained using Jarrow Rudd Model with $r=5\%$

$$Y_t = \log(S_t)$$

From Black-Scholes model we can conclude that.

$$Y_t \sim \mathcal{N}(Y_0 + (r - \sigma^2/2)\Delta t, \sigma^2\Delta t)$$

The probability for the risk neutral measure in the Jarrow-Rudd model is set to $\frac{1}{2}$. Using this probability we get the value of u and d as.

$$u = e^{(r - \sigma^2/2)\Delta t + \sigma\Delta t}$$

$$d = e^{(r - \sigma^2/2)\Delta t - \sigma\Delta t}$$

Titan model

The Titan model matches the first three moments of the binomial model to the first three moments of a lognormal distribution.

$$pu + (1 - p) = e^{r\Delta t}$$

$$pu^2 + (1 - p)d^2 = (e^{r\Delta t})^2 e^{\sigma^2\Delta t}$$

$$pu^3 + (1 - p)d^3 = (e^{r\Delta t})^3 (e^{\sigma^2\Delta t})^3$$

These equation can be solved to get the following parameters values.

$$v = e^{\sigma^2 \Delta t}$$

$$u = 0.5e^{r\Delta t}v(v + 1 + \sqrt{v^2 + 2v - 3})$$

$$u = 0.5e^{r\Delta t}v(v + 1 - \sqrt{v^2 + 2v - 3})$$

$$p = \frac{e^{r\Delta t} - d}{u - d}$$

The implementation for Titan model can be found in the code attached. The figure below shows the exercise frontier for an American option obtained using a Titan model.

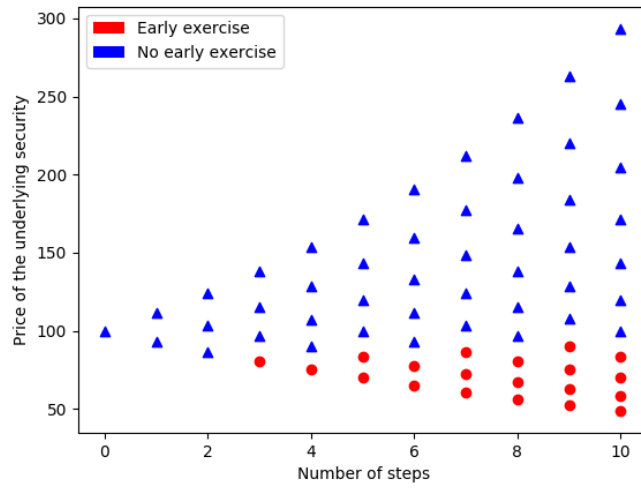


Figure 4: Exercise frontier obtained using Titan model with $r=5\%$

Jarrow-Rudd risk neutral model

Jarrow-Rudd risk neutral model is a slight modification on the standard Jarrow-Rudd model. We use the same values for u and d , however instead of $p = \frac{1}{2}$ we use the standard risk neutral definition for p .

$$u = e^{(r-\sigma^2/2)\Delta t + \sigma\Delta t}$$

$$d = e^{(r-\sigma^2/2)\Delta t - \sigma\Delta t}$$

$$p = \frac{e^{r\Delta t} - d}{u - d}$$

The implementation for Jarrow-Rudd risk neutral model can be found in the code attached. The figure below shows the exercise frontier for an American option obtained using a Jarrow-Rudd risk neutral model.

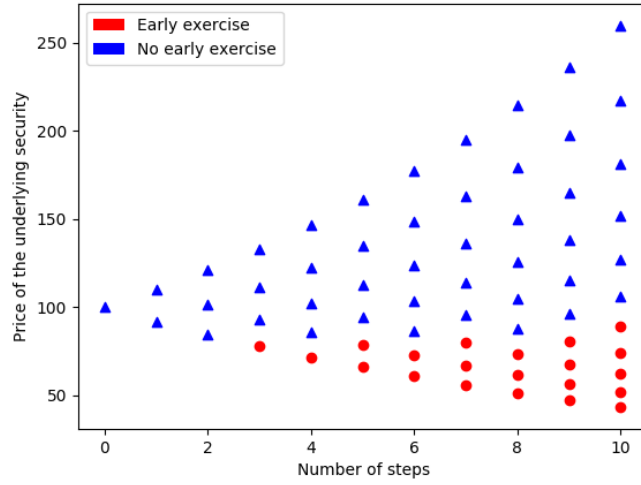


Figure 5: Exercise frontier obtained using Jarrow-Rudd risk neutral model with $r=5\%$

CRR model with drift

The CRR model with drift is a slight modification on the standard CRR model. In this model an arbitrary drift is applied to the parameters of the model and new parameters are given by.

$$\begin{aligned}
 u &= e^{\eta\Delta t + \sigma\sqrt{\frac{T}{n}}} \\
 d &= e^{\eta\Delta t - \sigma\sqrt{\frac{T}{n}}} \\
 p &= \frac{e^{r\Delta t} - d}{u - d}
 \end{aligned}$$

The drift term is added so that the lattice of security prices can be moved up or down around the current asset price S_0 so that more prices can expire in the money. The implementation for CRR model with drift can be found in the code attached. The figure below shows the exercise frontier for an American option obtained using a CRR model with drift.

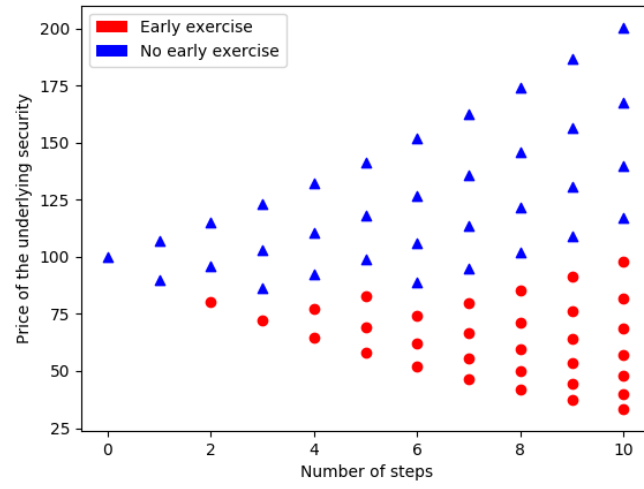


Figure 6: Exercise frontier obtained using CRR model with $\eta = -0.1$ and $r = 0.5\%$

Finite difference method for American options

Generating the matrix

We know that the matrix is A is defined by.

$$(AP)_j = \frac{\sigma^2 x_j^2}{2} \left(\frac{-P_{j-1}^n + 2P_j^n - P_{j+1}^n}{h^2} \right) - rx_j \left(\frac{P_{j+1}^n - P_j^n}{h} \right) + rP_j^n$$

$(AP)_j$ is a linear function of the components of the components of P. The code to generate the matrix is shown below.

```
x = np.array([h*j for j in range(M+1)])
2 t = np.array([DELTA_T*j for j in range(N)])
A = [[0 for i in range(M+1)] for j in range(M+1)]
4 for i in range(M):
    if(i>0):
6     A[i][i-1] = - (((SIGMA**2)*(x[i]**2))/(2*(h**2)))
    A[i][i] = (((SIGMA**2)*(x[i]**2))/(h**2)) + (r*x[i]/h) + r
8     A[i][i+1] = - (((SIGMA**2)*(x[i]**2))/(2*(h**2))) - (r*x[i]/h
    )
A = np.array(A)
```

Discretization scheme

The explicit Euler scheme given to us is as follows.

$$\min \left\{ \frac{P_j^{n+1} - P_j^n}{\Delta t} + \frac{\sigma^2 x_j^2}{2} \left(\frac{-P_{j-1}^n + 2P_j^n - P_{j+1}^n}{h^2} \right) - rx_j \left(\frac{P_{j+1}^n - P_j^n}{h} \right) + rP_j^n, P_j^{n+1} - g(x_j) \right\} = 0$$

$$P_{M+1}^{n+1} = 0$$

Not considering the stopping condition we can see that the explicit scheme is given by.

$$\begin{aligned}
& \min \left\{ \frac{P_j^{n+1} - P_j^n}{\Delta t} + \frac{\sigma^2 x_j^2}{2} \left(\frac{-P_{j-1}^n + 2P_j^n - P_{j+1}^n}{h^2} \right) - r x_j \left(\frac{P_{j+1}^n - P_j^n}{h} \right) + r P_j^n, P_j^{n+1} - g(x_j) \right\} = 0 \\
& \Rightarrow \min \left\{ \frac{P_j^{n+1} - P_j^n}{\Delta t} + A P_j^n, P_j^{n+1} - g(x_j) \right\} = 0 \\
& \Rightarrow P_j^{n+1} + \min \left\{ -P_j^n + \Delta t A P_j^n, -g(x_j) \right\} = 0 \\
& \Rightarrow P_j^{n+1} = -\min \left\{ -P_j^n + \Delta t A P_j^n, -g(x_j) \right\} \\
& \Rightarrow P_j^{n+1} = \max \left\{ P_j^n - \Delta t A P_j^n, g(x_j) \right\} \\
& \Rightarrow P^{n+1} = \max \left\{ P^n + \Delta t A P^n, \phi \right\}
\end{aligned}$$

Stability of explicit scheme

The solution to the explicit scheme is unstable and blows up as the value of M is increased.

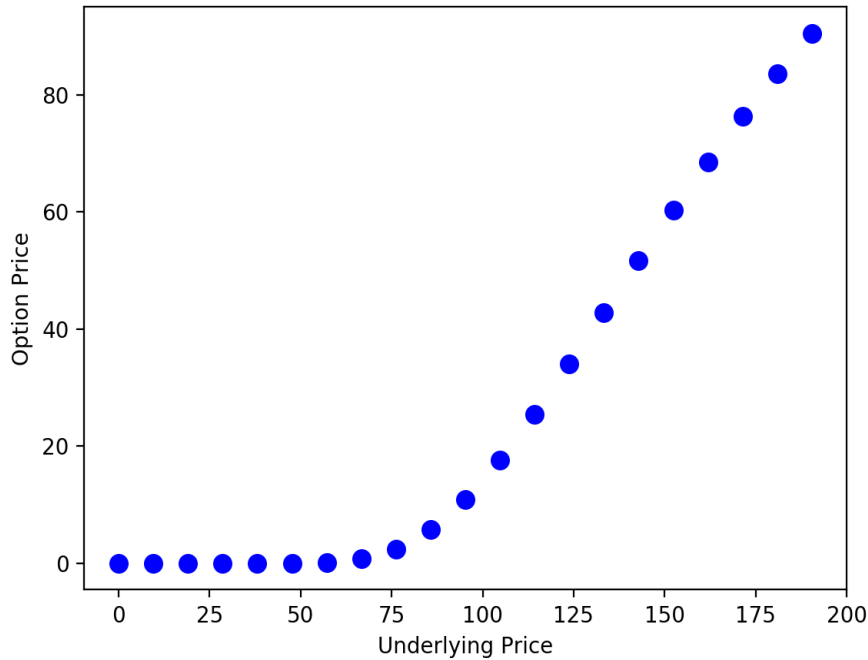


Figure 7: Explicit Scheme Solution for Different Values of S at $t = 0$

The figure shows the solution to the explicit solution scheme for $M = 20$. The code used to generate the solution is shown below.

```

1 def phi_call(x):
    payoff = max(x-K,0)
3 return payoff

5 def phi_put(x):
    payoff = max(K-x,0)
7 return payoff

9 P = np.array([[0.0 for i in range(M+1)] for j in range(N+1)])

11 for i in range(M+1):
    P[0][i] = phi_call(x[i])
13

15 for i in range(1,N+1):
    P[i] = np.maximum(P[i-1]-DELTA_T*np.matmul(A,P[i-1]),np.array([
        phi_call(x[j]) for j in range(M+1)]))
17
print("Explicit Euler Pricing Solution:")
19 print(P[N])
explicit_sol = P[N]

```

Implicit Euler scheme

We know that.

$$\left[\frac{\partial p}{\partial q} \right]_{ij} = \frac{\partial p_i}{\partial q_j} \quad \text{for any vector } p \text{ and } q$$

$$(Bx - b)_i = \sum_{a=1}^{M+1} (B_{ia}x_a) - b_i \quad \text{for } i = 0, 1 \dots M+1$$

From the above results we can conclude that. If $(Bx - b)_i \leq (x - \phi)$.

$$\begin{aligned}
\left[\frac{\partial F(x)}{\partial x} \right]_{ij} &= \left[\frac{\partial (Bx - b)}{\partial x} \right]_{ij} \\
&= \frac{\partial (Bx - b)_i}{\partial x_j} \\
&= \frac{\partial (\sum_{a=1}^{M+1} (B_{ia} x_a) - b_i)}{\partial x_j} \\
&= B_{ij}
\end{aligned}$$

Now considering the case when $(Bx - b)_i > (x - \phi)_i$.

$$\begin{aligned}
\left[\frac{\partial F(x)}{\partial x} \right]_{ij} &= \left[\frac{\partial (x - \phi)}{\partial x} \right]_{ij} \\
&= \frac{\partial (x - \phi)_i}{\partial x_j} \\
&= \frac{\partial (x_i - \phi_i)}{\partial x_j} \\
&= \delta_{ij}
\end{aligned}$$

The above is the required result.

Newton's method for implicit scheme

The implementation of Newton-Raphson algorithm for the implicit scheme is shown below.

```

B = np.identity(M+1) + DELTA_T*A
2 P = [np.array([np.random.uniform() for i in range(M+1)]) for i in
      range(N+1)]
for i in range(M+1):
4 P[0][i] = phi_call(x[i])

6 def F(a,b):
    return np.minimum(np.matmul(B,a)-b,a-P[0])

8
def deriv(a,b,i,j):
10 p = np.matmul(B,a)-b
    q = a - P[0]
12 if(p[i]<=q[i]):

```

```

    return B[i][j]
14 else:
    if(i==j):
16     return 1
    else:
18     return 0

20 def F_dash(a,b):
    K = [[0 for i in range(M+1)] for i in range(M+1)]
22     for i in range(M+1):
        for j in range(M+1):
24         K[i][j] = deriv(a,b,i,j)
    return np.array(K)

26
28 for j in range(1,N+1):
    for i in range(num_iterations):
        P[j] = P[j] - np.matmul(np.linalg.pinv(F_dash(P[j],P[j-1])),F(P
        [j],P[j-1]))

```

We run the code for 1000 iterations and find that solution remains stable. Shown below are the results for the option price for $M=20$ and $M=50$.

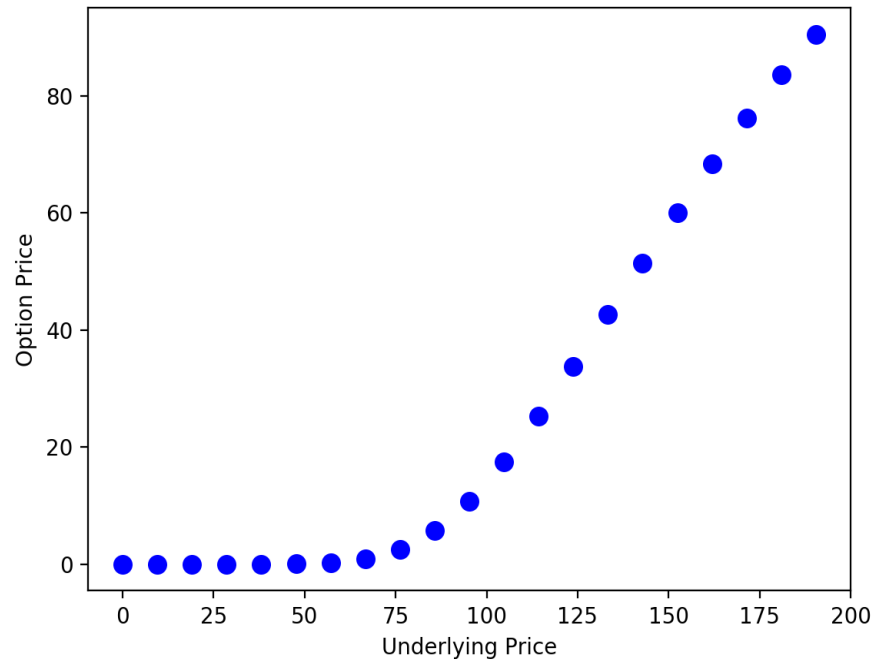


Figure 8: Implicit Scheme Solution for Different Values of S at $t = 0$ and $M = 20$

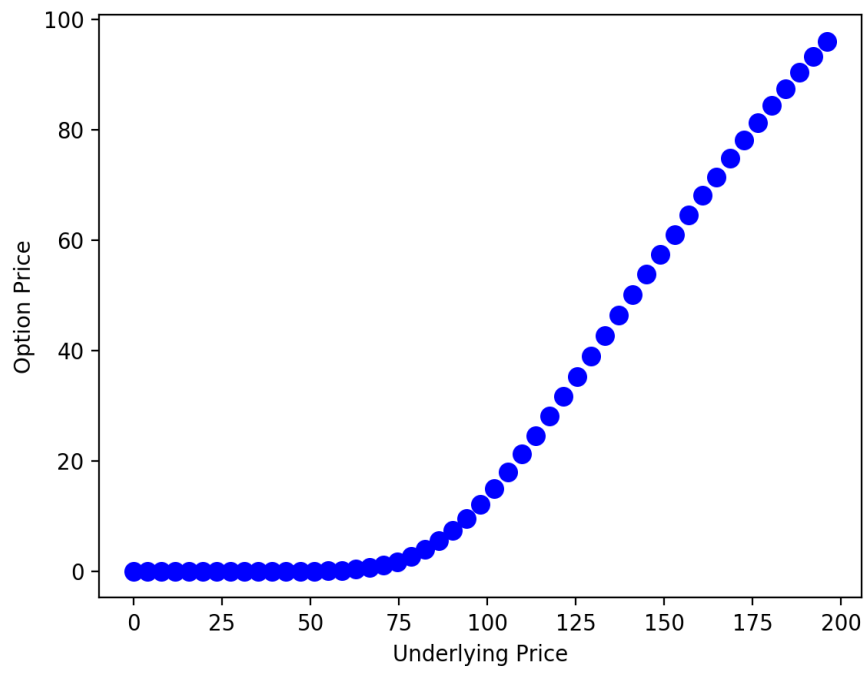


Figure 9: Implicit Scheme Solution for Different Values of S at $t = 0$ and $M = 50$