Optional project: CCR model and American options

Name: Jinxin Xiong

UNI: jx2383

Github link for the project: https://github.com/JinxinXiong/Optional-Project-for-IEOR-4706

1 PERPETUAL AMERICAN OPTIONS

1.1

For the owner of the perpetual American put can exercise at any time. Therefore we can think that the optimal exercise region is not related to the time to maturity, and only depend on the price of the stock. Combined with the payoff of the put option, a reasonable way to exercise the perpetual American Option is when the stock price falls below to a level x^*, i.e., the exercise region is $(0, x^*]$

1.2

When $x > x^*$, according to the definition of x^*, we should have $v(x) > h(x)$. Then, combined with the Hamilton-Jacobi-Bellman equation, we need to have $0 = rv(x) - \mu x v'(x) - \frac{1}{2}\sigma^2 x^2 v''(x) \leq v(x) - h(x)$.

Noticed that the ODE is in the form of Cauchy-Euler equation, we can guess the solution is x^α and plug in the ODE to get the characteristic equation and then the general solution for this ODE.

$$\frac{1}{2}\sigma^2 x^2 v''(x) + \mu x v'(x) - rv(x) = 0,\ x > x^*$$

The characteristic for the ODE and two solutions:

$$\frac{1}{2}\sigma^2\alpha(\alpha-1) + \mu\alpha - r = 0$$

$$\alpha_1 = m = \frac{(\frac{1}{2}\sigma^2 - \mu) - \sqrt{(\frac{1}{2}\sigma^2 - \mu)^2 + 2r\sigma^2}}{\sigma^2}$$

$$\alpha_2 = n = \frac{(\frac{1}{2}\sigma^2 - \mu) + \sqrt{(\frac{1}{2}\sigma^2 - \mu)^2 + 2r\sigma^2}}{\sigma^2}$$

Therefore, the general solution of this ODE is:

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

where A, B are constant that remained to be specified.

1.3

For $\lim_{x\to\infty} h(x) = 0$, we can find that $\lim_{x\to\infty} v(x) = 0$. Also, considering the definition of the payoff of the put option, we can conclude that $v(x)$ is bounded and $B = 0$. And later on we consider the general solution to the original ODE is $v(x) = Ax^m$.

1.4

According to previous analysis, we have

$$v(x) = \begin{cases} Ax^m & x < x^* \\ K - x & x \geq x^* \end{cases}$$

and we have the derivative of $v(x)$ as:

$$v'(x) = \begin{cases} Amx^{m-1} & x < x^* \\ -1 & x \geq x^* \end{cases}$$

We want $v(x)$ and $v'(x)$ to be continuous at x^*, therefore we need

$$v(x^*) = A(x^*)^m = K - x^*$$

$$v'(x^*) = Am(x^*)^{m-1} = -1$$

Then we can get $x^* = \frac{mK}{m-1}, A = -\frac{1}{m}(\frac{mK}{m-1})^{1-m}$, where $m = \frac{(\frac{1}{2}\sigma^2 - \mu) - \sqrt{(\frac{1}{2}\sigma^2 - \mu)^2 + 2r\sigma^2}}{\sigma^2}$.

Therefore, when the stock price falls below x^*, it is optimal to exercise.

When r goes to zero, we can find that m goes to 0, and x^* goes to 0 as well, which means it will never be optimal for an early exercise.

2 PRICING AND EXERCISE FRONTIER FOR THE AMERICAN PUT IN THE CRR MODEL.

2.1

See Appendix A for the complete code.

2.2

See Appendix B for the complete code. In particular, when considering parameters $n = 20, S0 = 110, r = 0.05, K = 100, \sigma = 0.2, T = 1., \mu = r$, the price at time 0 for European call is 17.615 and the price for European put is 2.738.

2.3

Recursion formula:

$$P^n = h(S_{t_n}),$$

$$P^i = max(\mathbb{E}^{\mathbb{Q}}[e^{-r\frac{T}{n}}P^{i+1}|\mathbf{F}_{t_i}], h(S_{t_i})),\ i = 0, \cdots, n-1$$

See Appendix C for the complete code. When considering parameters $n = 20, S0 = 110, r = 0.05, K = 100, \sigma = 0.2, T = 1., \mu = r$, the price at time 0 for American call is 17.615, which is identical to the European call with the same parameters. And it is coincide with what we have covered in class, that it is never optimal to early exercise a American call with a positive interest rate and zero dividend. And the price for the American put is 2.947, which is greater than the European put.

In particular, when considering $r = \mu = 0$ and other parameters remain the same, the price for American Call is 14.234, and the price for American put is 4.234. We can see that in the case when $r = 0$, the price for European options and American options are identical, which shows that it is never optimal to early exercise an American option when the risk-free rate is zero.

2.4

The figure below shows the exercise frontier of the American put. Each points represents a node in the binomial tree, red represent should early exercise and blue represent the situations that should not early exercise.

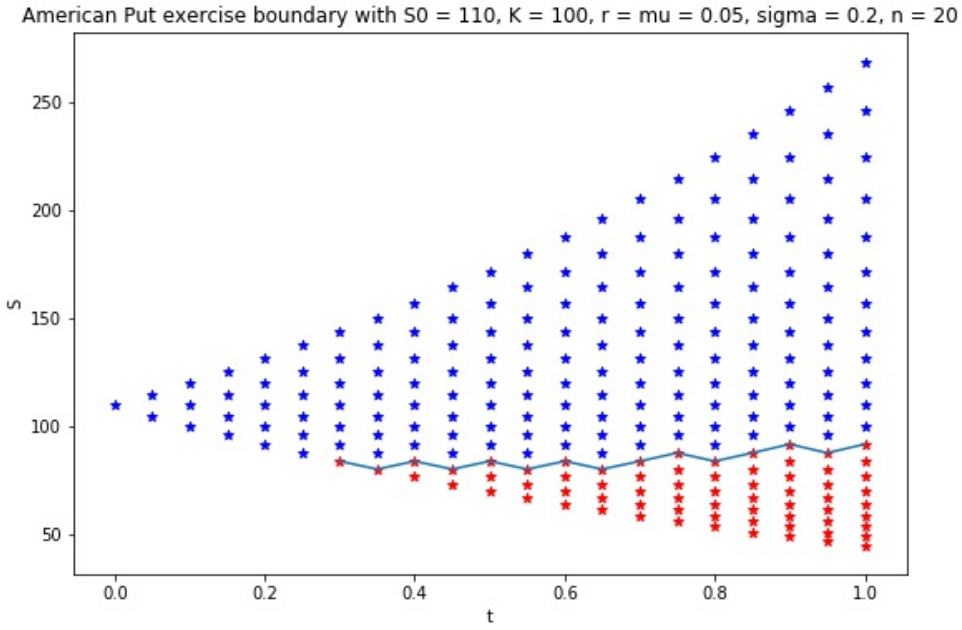

Figure 1

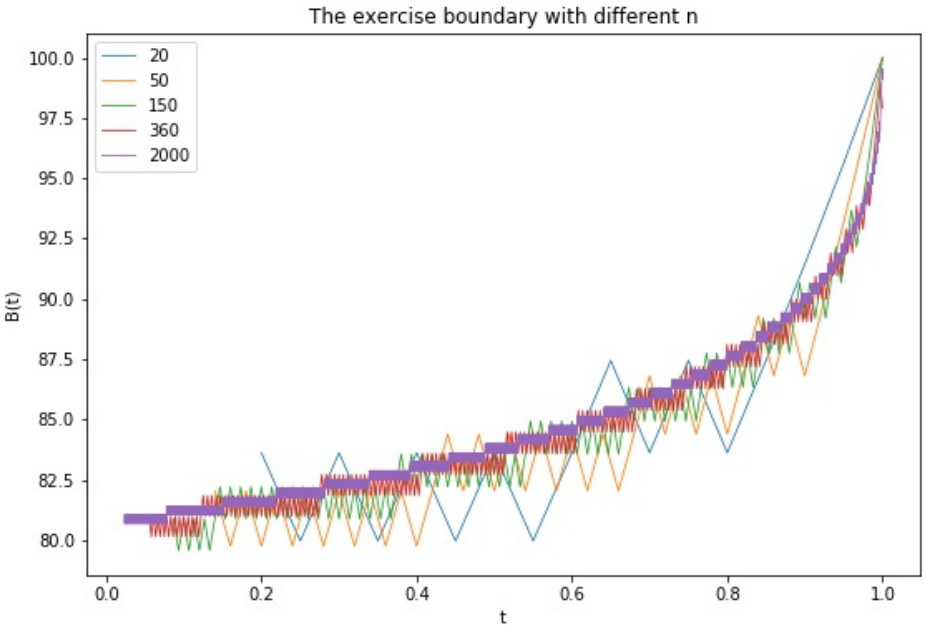

Figure 2

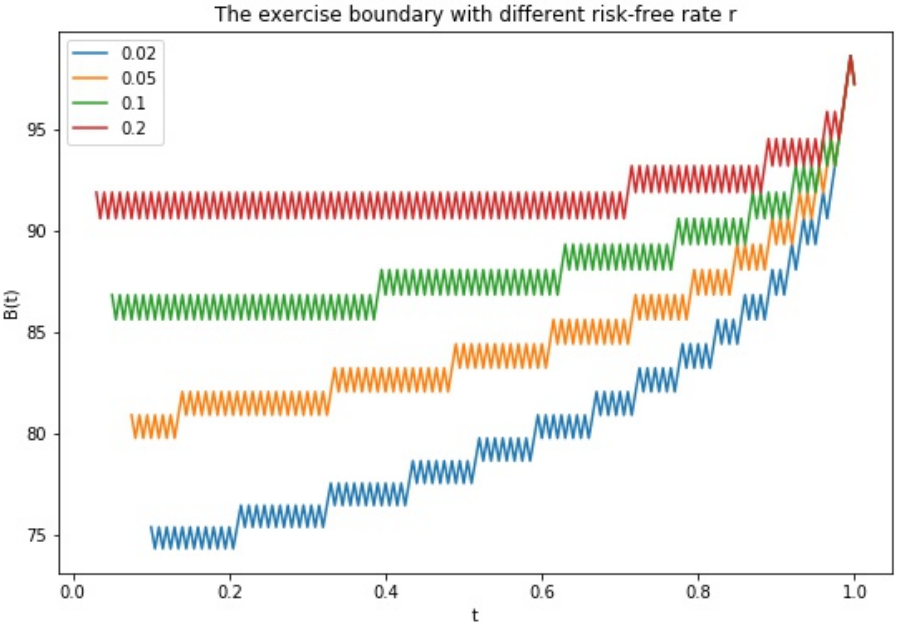

Figure 3

Figure 2 shows the curve of exercise boundary$B(t)$ with time t under the parameters $S0 = 100, r = 0.05, K = 100, \sigma = 0.2, T = 1., \mu = r$ and $n = 20, 50, 150, 360, 2000$. We can see that the boundary will not change with the choose of n, even though the curve will become smoother as n becomes larger.

Figure 3 shows the curve of exercise boundary$B(t)$ with time t under the parameters $n = 360, S0 = 100, r = 0.05, K =$

$100, \sigma = 0.2, T = 1., \mu = r = 0.02, 0.05, 0.1, 0.2$. We can see that the boundary will indeed change as r change, and the larger the r is, the higher the boundary and the flatter the curve is.

3 FINITE DIFFERENCES FOR AMERICAN OPTIONS

3.1 *Explicit Euler scheme*

3.1.1

Denote $a_j = -\frac{\sigma^2 x_j^2}{2h^2}, b_j = \frac{\sigma^2 x_j^2}{h^2} + \frac{r x_j}{h} + r, c_j = -\frac{\sigma^2 x_j^2}{2h^2} - \frac{r x_j}{h}$, then the matrix A can be represented as

$$\begin{pmatrix} b_0 & c_0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ a_1 & b_1 & c_1 & 0 & \cdots & 0 & 0 & 0 \\ 0 & a_2 & b_2 & c_2 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & a_{M-1} & b_{M-1} & c_{M-1} \\ 0 & 0 & 0 & 0 & 0 & 0 & a_M & b_M \end{pmatrix}$$

The codes to generate matrix A see Appendix F.

3.1.2

With the notation in previous question, the scheme becomes

$$min\left\{\frac{P^{n+1} - P^n}{\Delta t} + AP^n, P^{n+1} - \phi\right\} = 0$$

Then we have:

$$\begin{cases} \frac{P^{n+1} - P^n}{\Delta t} + AP^n \geq 0 \\ P^{n+1} - \phi \geq 0 \end{cases}$$

For $\Delta t > 0$, we further have:

$$\begin{cases} P^{n+1} \geq P^n - \Delta t A P^n \\ P^{n+1} \geq \phi \end{cases}$$

Therefore we can conclude that

$$P^{n+1} = max\left\{P^n - \Delta A P^n, \phi\right\}$$

3.1.3

For the full code see Appendix F.

According to Figure 4 and 5, we can see that the explicit schemes is indeed unstable.

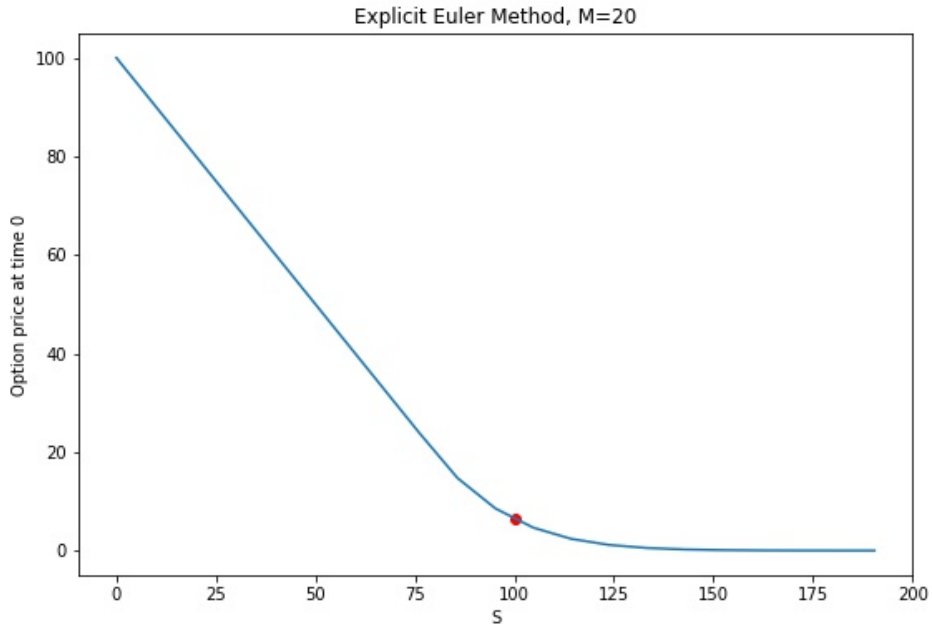

Figure 4

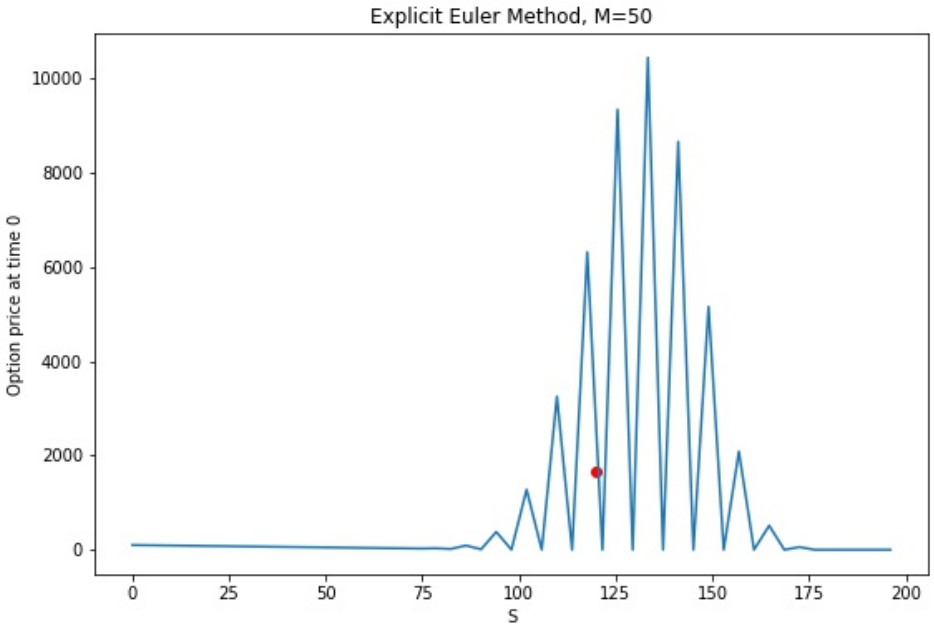

Figure 5

3.2 *Implicit Euler scheme*

3.2.1

Following the definition that $B = I_{M+1} + \Delta t A$ and $b = P^n$. $F(x) = min\{Bx - b, x - \phi\} = 0$. Denote the ith element of $F(x)$ as $F(x)_i$ and

$$F(x)_i = min\left\{\sum_{j=0}^{M} B_{i,j} * x_j - b, x_i - \phi_i\right\}$$

Therefore,

$$F(x)_i = \begin{cases} \sum_{j=0}^{M} B_{i,j} * x_j - b, & if (Bx-b)_i \leq (x_\phi)_i \\ x_i - \phi_i, & if (Bx-b)_i > (x_\phi)_i \end{cases}$$

Further, we denote the derivative of $F(x)_i$ respective to x_j is $F'(x)_{i,j}$:

$$F'(x)_{i,j} = \begin{cases} B_{i,j}, & if (Bx-b)_i \leq (x_\phi)_i \\ \delta_{i,j}, & if (Bx-b)_i > (x_\phi)_i \end{cases}$$

where $\delta_{i,j}$ equals 1 if $i = j$ and 0 otherwise.

3.2.2

First, considering the parameters identical to the parameters provided for the numerical experiments in the previous section for Explicit Euler Scheme, which are $K = 100, S_{max} = 200, T = 1, \sigma = 0.2, r = 0.1, S0 = 100, N = 20,$. And we take $M = 20$ and $M = 50$ respectively to get:

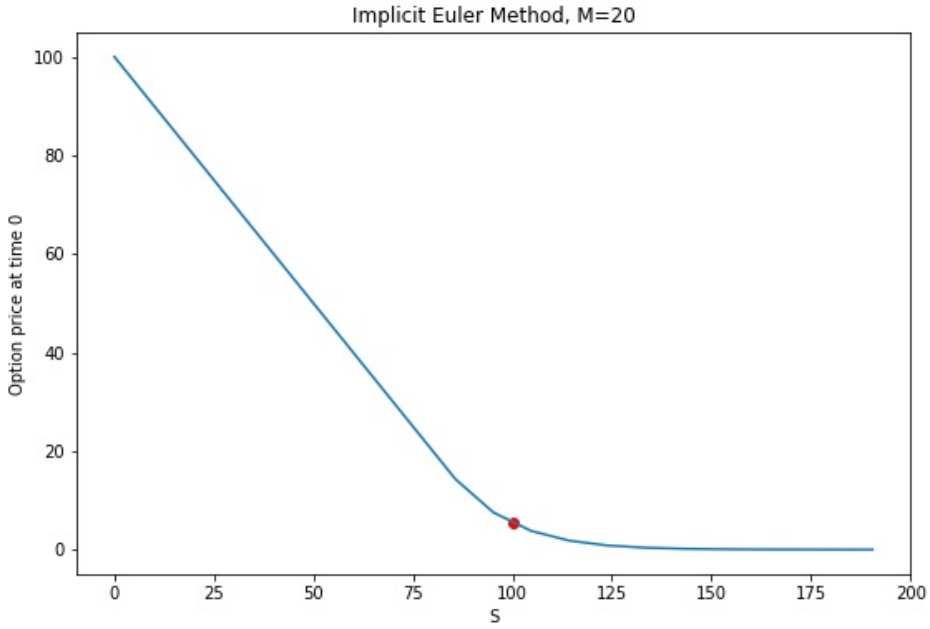

Figure 6

From Figure 6 and Figure 7, we can see that the Explicit Euler Scheme is more stable than the Implicit Euler Scheme.

To further test the stability, we plot the result under different choice of M and the American put price under parameters $K = 90, S_{max} = 200, T = 1, \sigma = 0.2, r = 0.1, S0 = 100$, with the interpolation method.

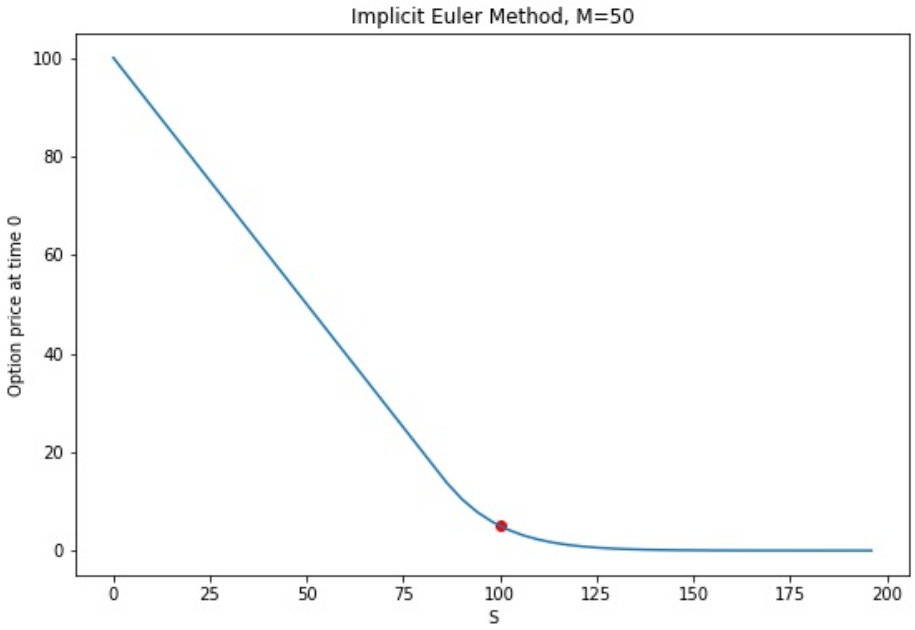

Figure 7

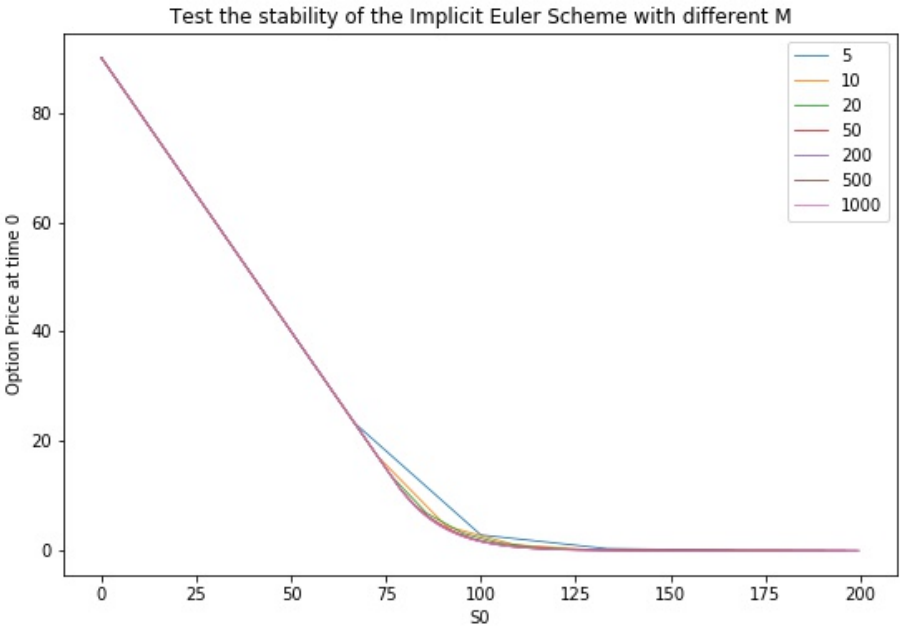

Figure 8

From Figure 8 we can see that the Implicit Euler Scheme is indeed stable with different choice of M. And from Figure 9 we can see that this scheme is converge as M and N become larger.

The codes for the Newton algorithm and the implicit Euler scheme are in Appendix H and Appendix I.

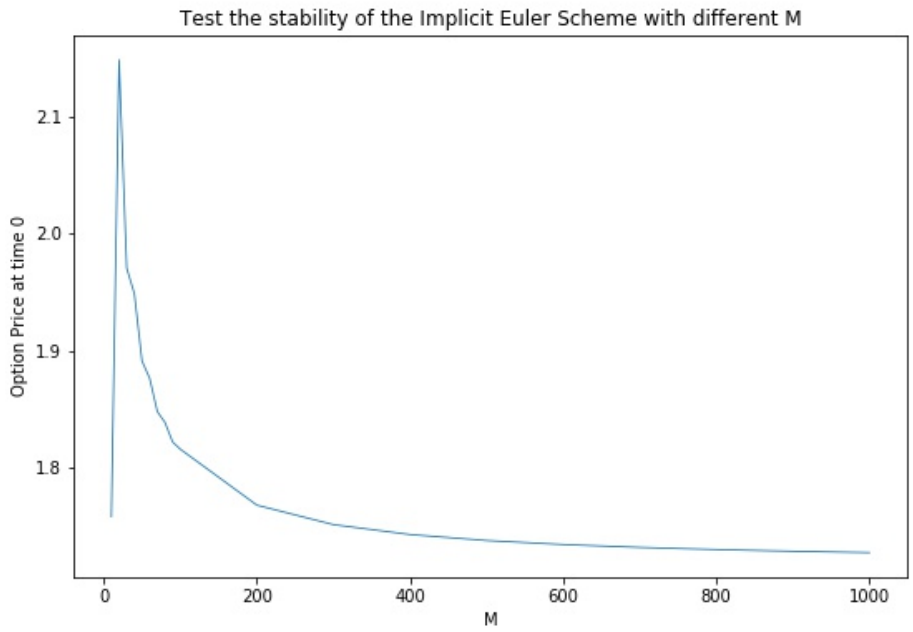

Figure 9

Appendices

A

Code to create the tree of the possible values of the risky asset

```
def Tree(S0, n, sigma):
    u = np.exp(sigma/np.sqrt(n))
    d = np.exp(-sigma/np.sqrt(n))
    tree = [[S0*u**j*d**(i-j) if j <=i else 0 for j in range(n+1)] for i in range(n+1)]
    return np.array(tree)
```

B

Code for European Call and Put in the CRR model

```
def EuropeanOption(T, r, K, n, S0, sigma, SS, t):
    u = np.exp(sigma/np.sqrt(n));
    d = np.exp(-sigma/np.sqrt(n));
    #risk neutral probability
    q = (np.exp(r*T/n)-d)/(u-d);
    #length of the time periods
    h = T/n;
    if t == 'c':
        EuroCall = np.zeros((n+1, n+1))
        for j in range(n+1):
            EuroCall[-1, j] = max(SS[-1, j] - K, 0)
        for i in range(n-1,-1, -1):
            for j in range(i+1):
                EuroCall[i, j] = np.exp(-r*h)*((1-q) * EuroCall[i+1, j] + q * EuroCall[i+1, j+1]);
        return EuroCall
```

```
    elif t == 'p':
        EuroPut = np.zeros((n+1, n+1))

        for j in range(n+1):
            EuroPut[-1, j] = max(K - SS[-1, j], 0)

        for i in range(n-1,-1, -1):
            for j in range(i+1):
                EuroPut[i, j] = np.exp(-r*h)*((1-q) * EuroPut[i+1, j] + q * EuroPut[i+1, j+1]);
        return EuroPut
    else:
        print("Input Error")
        return
```

C

Code for American Call and Put in the CRR model

```
def AmericanOption(T, r, K, n, S0, sigma, SS, t):
    u = np.exp(sigma/np.sqrt(n));
    d = np.exp(-sigma/np.sqrt(n));
    #risk neutral probability
    q = (np.exp(r*T/n)-d)/(u-d);
    if q > 1:
        raise Exception('Error')
    #length of the time periods
    h = T/n;
    if t == 'c':
        frontier_call = dict()
        AmericanCall = np.zeros((1+n, 1+n))
        EPA_call = np.zeros((1+n, 1+n))
        for i in range(n+1):
            for j in range(i+1):
                AmericanCall[i, j] = max(SS[i, j] - K, 0);
        for j in range(1+n):
            if AmericanCall[n, j] > 0:
                EPA_call[n, j] = 1
                frontier_call[n*h] = SS[n, j]
        for i in range(n-1, -1, -1):
            for j in range(i+1):
                temp_call = np.exp(-r*h)*((1-q) * AmericanCall[i+1, j] + q * AmericanCall[i+1, j+1]
                                                                        );
                if AmericanCall[i,j] <= temp_call:
                    AmericanCall[i,j] = temp_call
                else:
                    EPA_call[i,j] = 1
                    frontier_call[i*h] = SS[i,j]
        return AmericanCall, EPA_call, frontier_call
    elif t == 'p':
        frontier_put = dict()
        AmericanPut = np.zeros((1+n, 1+n))
        EPA_put = np.zeros((1+n, 1+n))
        for i in range(n+1):
            for j in range(i+1):
                AmericanPut[i, j] = max(K - SS[i, j], 0);
        for j in range(1+n):
            if AmericanPut[n, j] > 0:
                EPA_put[n, j] = 1
```

```
                frontier_put[n*h] = SS[n, j]
        for i in range(n-1, -1, -1):
            for j in range(i+1):
                temp_put = np.exp(-r*h)*((1-q) * AmericanPut[i+1, j] + q * AmericanPut[i+1, j+1]);
                if AmericanPut[i,j] <= temp_put:
                    AmericanPut[i,j] = temp_put
                else:
                    EPA_put[i,j] = 1
                    frontier_put[i*h] = SS[i, j]

        return AmericanPut, EPA_put, frontier_put
```

D

Code for exercise frontier of the American Put

```
plt.figure(figsize=(9,6))
n = 20
S0 = 110
r = 0.05
K = 100
sigma = 0.2
T = 1.
mu = r

SS = Tree(S0, n, sigma)
AmericanPut, EPA_put, frontier_put = AmericanOption(T, r, K, n, S0, sigma, SS, 'p')
plt.plot(list(frontier_put.keys()), list(frontier_put.values()))
h = T/n
for i in range(n+1):
    for j in range(i+1):
        if EPA_put[i,j] == 1:
            plt.scatter(i*h, SS[i,j], c = 'r', marker = '*')
        else:
            plt.scatter(i*h, SS[i,j], c = 'b', marker = '*')
plt.title("American Put exercise boundary with S0 = {}, K = {}, r = mu = {}, sigma = {}, n = {}".
                                                format(S0, K,r, sigma, n))
plt.xlabel('t')
plt.ylabel('S')
plt.savefig('Bin_boundary.jpg')
plt.show()
```

E

Code for exercise frontier with different r and n

```
test_n = [20, 50, 150, 360, 2000]
S0 = 100
r = 0.05
K = 100
sigma = 0.2
T = 1.
mu = r

plt.figure(figsize=(9,6))
for ni in test_n:
    SS = Tree(S0, ni, sigma)
    AmericanPut, EPA_put, frontier_put = AmericanOption(T, r, K, ni, S0, sigma, SS, 'p')
    plt.plot(list(frontier_put.keys()), list(frontier_put.values()), linewidth = '.8')
```

```
plt.title('The exercise boundary with different n')
plt.xlabel('t')
plt.ylabel('B(t)')
plt.legend(test_n)
plt.savefig('relation_n.jpg')
plt.show()
```

```
test_r = [0.02, 0.05, 0.1, 0.2]
n = 360
S0 = 100
K = 100
sigma = 0.2
T = 1.
mu = r

plt.figure(figsize=(9,6))
for ri in test_r:
    SS = Tree(S0, 200, sigma)
    AmericanPut, EPA_put, frontier_put = AmericanOption(T, ri, K, 200, S0, sigma, SS, 'p')
    plt.plot(list(frontier_put.keys()), list(frontier_put.values()))
plt.title('The exercise boundary with different risk-free rate r')
plt.xlabel('t')
plt.ylabel('B(t)')
plt.legend(test_r)
plt.savefig('relation_r.jpg')
plt.show()
```

F

Code to compute and generate matrix A in the Explicit Euler scheme and the function for the Explicit Euler scheme

```
def Explicit(K, S_max, S0, T, sigma, r, N, M):
    def payoff(K, x):
        temp = np.zeros(x.shape)
        temp = np.stack((K-x, temp), axis = 1)
        return np.max(temp, axis=1)

    h = S_max/(M+1)
    delta_t = T/N
    x = np.arange(M+2) * h
    x = x[:-1]
    t = np.arange(N+1) * delta_t

    x2 = x ** 2
    sigma2 = sigma * sigma
    h2 = h * h

    a = -sigma2 * x2 /(2 * h2)
    b = sigma2 * x2 / h2 + r*x/h+r
    c = -sigma2*x2/(2*h2)-r*x/h

    A = np.diag(b)
    for i in range(1, M+1):
        A[i, i-1] = a[i]
    for i in range(M):
        A[i, i+1] = c[i]

    phy = payoff(K, x)
    P = phy
```

```
    for i in range(1, N):
        P_temp = P - delta_t * A @ P
        temp = np.stack((P_temp, phy), axis = 1)
        P = np.max(temp, axis=1)

    f=interpolate.interp1d(x,P,kind="cubic")
    print(f(S0))
    return x, P
```

G

Code for implementing the Explicit Euler scheme with M=20 and M=50 respectively

```
def Explicit_plot_help(x, P, M, filename):
    plt.figure(figsize=(9,6))
    plt.title("Explicit Euler Method, M={}".format(M))
    plt.plot(x, P)
    plt.xlabel('S')
    plt.ylabel('Option price at time 0')
    f=interpolate.interp1d(x,P,kind="cubic")
    plt.scatter(S0, f(S0), c = 'r')
    plt.savefig(filename + '.jpg')
    plt.show()
```

```
K = 100
S_max = 200
T = 1
sigma = 0.2
r = 0.05
S0 = 100
N = 20
M = 20
xx, Price = Explicit(K, S_max, S0, T, sigma, r, N, M)
Explicit_plot_help(xx, Price, M, 'Explicit_M20')

M = 50
xx, Price = Explicit(K, S_max, S0, T, sigma, r, N, M)
Explicit_plot_help(xx, Price, M, 'Explicit_M50')
```

H

Code for the Newton algorithm in the Implicit Euler scheme

```
def F_prime(B, b, xi, phy):
    m = B.shape[0]
    Bx = B @ xi
    temp1 = Bx - b
    temp2 = xi - phy
    temp = np.stack((temp1, temp2), axis = 1)
    F = np.min(temp, axis = 1)
    F_p= np.eye(m)
    index = temp1 < temp2
    F_p[index, :] = B[index, :]
    return F, F_p
```

```
def Newton_Raphson(B, b, phy):
    max_iter = 1000
```

```
    tol = 1e-6
    xi = np.random.rand(b.shape[0])
    for i in range(max_iter):
        F, F_p = F_prime(B, b, xi, phy)
        x_old = xi
        xi = xi - np.linalg.inv(F_p) @ F
        if np.linalg.norm(xi - x_old) < tol:
            break
    return xi
```

I

Code for the Implicit Euler scheme

```
def Implicit(K, S_max, S0, T, sigma, r, N, M):
    def payoff(K, x):
        temp = np.zeros(x.shape)
        temp = np.stack((K-x, temp), axis = 1)
        return np.max(temp, axis=1)

    h = S_max/(M+1)
    delta_t = T/N
    x = np.arange(M+1) * h
    t = np.arange(N+1) * delta_t

    x2 = x ** 2
    sigma2 = sigma * sigma
    h2 = h * h

    a = -sigma2 * x2 /(2 * h2)
    b = sigma2 * x2 / h2 + r * x / h + r
    c = -sigma2 * x2/(2 * h2)-r * x / h

    A = np.diag(b)
    for i in range(1, M+1):
        A[i, i-1] = a[i]
    for i in range(M):
        A[i, i+1] = c[i]

    phy = payoff(K, x)
    P = phy
    B = np.eye(M+1) + delta_t * A

    for i in range(1, N):
        b = P
        P = Newton_Raphson(B, b, phy)
        F, _ = F_prime(B, b, P, phy)

    f=interpolate.interp1d(x,P,kind="cubic")
    print(f(S0))
    return x, P
```

J

Code for implementing the Implicit Euler scheme with M=20 and M=50 respectively

```
def Implicit_plot_help(x, P, M, filename):
    plt.figure(figsize=(9,6))
    plt.title("Implicit Euler Method, M={}".format(M))
```

```
    plt.plot(x, P)
    plt.xlabel('S')
    plt.ylabel('Option price at time 0')
    f=interpolate.interp1d(x,P,kind="cubic")
    plt.scatter(S0, f(S0), c = 'r')
    plt.savefig(filename + '.jpg')
    plt.show()
```

```
K = 100
S_max = 200
T = 1
sigma = 0.2
r = 0.1
S0 = 100
N = 20
M = 20
xx, Price = Implicit(K, S_max, S0, T, sigma, r, N, M)
Implicit_plot_help(xx, Price, M, 'Implicit_M20')

M = 50
xx, Price = Implicit(K, S_max, S0, T, sigma, r, N, M)
Implicit_plot_help(xx, Price, M, 'Implicit_M50')
```

K

Code to test the stability of the Implicit Euler scheme

```
K = 90
S_max = 200
T = 1
sigma = 0.2
r = 0.1
S0 = 100
N = [5,10, 20, 50, 200, 500, 1000]
M = [5,10, 20, 50, 200, 500, 1000]
plt.figure(figsize=(9,6))
plt.title('Test the stability of the Implicit Euler Scheme with different M')
for i in range(len(M)):
    xx, Price = Implicit(K, S_max, S0, T, sigma, r, N[i], M[i])
    plt.plot(xx, Price, linewidth = '.8')
plt.legend(M)
plt.xlabel('S0')
plt.ylabel('Option Price at time 0')
plt.savefig('Explicit_stable_NM.jpg')
plt.show()
```

L

Code to test the convergence of the Implicit Euler scheme with respective to M

```
K = 90
S_max = 200
T = 1
sigma = 0.2
r = 0.1
S0 = 100
N = 20
M = [5,10, 20, 50, 200, 500, 1000]
```

```
plt.figure(figsize=(9,6))
plt.title('Test the stability of the Implicit Euler Scheme with different M')
for m in M:
    xx, Price = Implicit(K, S_max, S0, T, sigma, r, N, m)
    plt.plot(xx, Price, linewidth = '.8')
plt.legend(M)
plt.xlabel('S0')
plt.ylabel('Option Price at time 0')
plt.savefig('Explicit_stable.jpg')
plt.show()
```