Homework 5 Advanced Derivatives

Albin Henriksson

October 2022

Exercise 1

In order to obtain a standard error, we calculated the option price $M = 100$ times using a monte carlo least squares approach. At each iteration, we generate $N = 10000$ stock paths, which are used in the pricing process. To decide whether or not to exercise early, we fit a regression model where the y-vector are the nonzero elements of the payoffs at the current step. We then compare the regression prices to the payoff if we were to exercise at the current step. If the regression price is larger, we do not exercise. The reason we only fit the nonzero elements of the payoff is because it's clear not to exercise if the exercise value is 0. The option price obtained was 7.84 with a standard error of 0.012. The code we used to obtain these values can be seen below.

Code

```
import pandas as pd
from scipy.stats import norm
import matplotlib.pyplot as plt
import numpy as np
import random

"""Parameters"""

S_0=100
K=98
r=0
q=.02
N=10000
sigma=.23
dt=.25
t=np.linspace(0,1,5).reshape(-1,1)

M = 100
price = np.zeros((M, 1))

"""Loop where option price is calculated"""

for j in range(M):  # Iterate M times to estimate a standard error
```

```python
"""Generate stock prices"""

dW = np.sqrt(dt) * np.random.normal(0, 1, (N, len(t) - 1))
BM_path = np.cumsum(dW, axis=1)  # Simulate brownian motion
S = S_0 * np.exp((r - q + 0.5 * sigma ** 2) * np.ones((N, 1)) @ t
    [1:].T + sigma * BM_path)  # Stock process

""" Calculate A"""

A = np.cumsum(S, axis=1)
for i in range(len(t) - 1):
    A[:, i] = A[:, i] / (i + 1)
    payoff = np.maximum(A - K, 0)

"""Create 3d matrix with regressor variables, where the
    dimensions are time,
number of stock simulations and number of regressors including a
    constant term."""

regressors = np.array([S, S ** 2, S ** 3, A, A ** 2, A ** 3])
regressors = np.transpose(regressors, (2, 1, 0))
X = np.concatenate((np.ones((len(t) - 1, N, 1)), regressors),
    axis=2)

"""Initiate cashflow vector. Since the risk-free rate is 0,
we do not need to keep track of time, so a matrix is unnceccesary
    """

cashflows = payoff[:, -1]

"""Loop to update the cashflow vector. Decision to exercise or
    not is
determined by comparing regression value and exercise value"""

for i in range(len(t) - 2):

    x = X[len(t) - 2 - 1 - i, ::]  # Regressor variables at each
        timestep.

    if i == len(t) - 2 - 1:  # At the first time step, A=S, A^2=S
        ^2 and A^3=S^3, so we exclude the powers of A.
        x = x[:, 0:4]

    payofftemp = payoff[:, len(t) - 2 - 1 - i]
    boolean1 = payofftemp > 0
    cashflowtemp = boolean1 * cashflows  # Setting elements to 0
        where the current payoff is 0

    for k in range(len(x[0, :])):
```

```
            x[:, k] = x[:, k] * boolean1  # Setting rows to 0 where
                the current payoff is 0

        x_nonzero = x

        beta = np.linalg.inv(x_nonzero.T @ x_nonzero) @ x_nonzero.T @
             cashflowtemp  # Fitted parameters
        regression_prices = x_nonzero @ beta  # Fitted prices
        boolean = regression_prices > payofftemp
        cashflows = boolean * cashflowtemp + (
                    1 - boolean) * payofftemp  # Updating cashflow
                        vector based on boolean rule

    price[j] = np.mean(cashflows)

mean_price = np.mean(price)
error_price = np.std(price) / np.sqrt(M)
print('The estimated price is {0} with a standard error of {1}'.
    format(mean_price, error_price))
```