Empirical Industrial Organisations I: PSet 3

S M Sajid Al Sanai

December 4, 2018

Contents

1 Question I

1.1 Choice-Specific Value Functions

Define the maximisation problem:

$$\Big(\sum_{t=1}^{\infty}\beta^t[u(x_t,i_t,\theta_1)+\epsilon_t(i_t)]\Big) \tag{1}$$

Define the utility function:

$$u(x_t,i_t;\theta_1)+\epsilon_t(i_t)=\begin{cases}u(x_t,1;\theta_1)+\epsilon_t(1), & \text{if } i_t=1.\\ u(x_t,0;\theta_1)+\epsilon_t(0), & \text{if } i_t=0.\end{cases} \tag{2}$$

$$u(x_t,i_t;\theta_1)+\epsilon_t(i_t)=\begin{cases}-RC-c(0,\theta_1)+\epsilon_t(1), & \text{if } i_t=1.\\ -c(x_t,\theta_1)+\epsilon_t(0), & \text{if } i_t=0.\end{cases} \tag{3}$$

Define the shorthand utility function:

$$\equiv u(x_t,i_t;\theta_1)+\epsilon_t(i_t)=-RC*i_t-c([1-i_t]*x_t,\theta_1)+\epsilon_t(i_t) \tag{4}$$

Define the value function:

$$V(x_t,\epsilon_t)=max\{\tilde{V}(x_t,\epsilon_t(1),1),\tilde{V}(x_t,\epsilon_t(0),0)\} \tag{5}$$

Define the choice-specific value functions:

$$\tilde{V}(x_t,\epsilon_t(i_t),i_t)=\begin{cases}u(x_t,1;\theta_1)+\epsilon_t(1)+\beta E_tV(0,\epsilon_{t+1}), & \text{if } i_t=1.\\ u(x_t,0;\theta_1)+\epsilon_t(0)+\beta E_tV(x_{t+1},\epsilon_{t+1}), & \text{if } i_t=0.\end{cases} \tag{6}$$

$$\tilde{V}(x_t,\epsilon_t(i_t),i_t)=\begin{cases}-RC-c(0,\theta_1)+\epsilon_t(1)+\beta E_tV(0,\epsilon_{t+1}), & \text{if } i_t=1.\\ -c(x_t,\theta_1)+\epsilon_t(0)+\beta E_tV(x_{t+1},\epsilon_{t+1}), & \text{if } i_t=0.\end{cases} \tag{7}$$

Define the shorthand choice-specific value function:

$$\equiv \tilde{V}(x_t,\epsilon_t(i_t),i_t)=u(x_t,i_t;\theta_1)+\epsilon_t(i_t)+\beta E_tV([1-i_t]*x_{t+1},\epsilon_{t+1}) \tag{8}$$

$$\equiv \tilde{V}(x_t,\epsilon_t(i_t),i_t)=-RC*i_t-c([1-i_t]*x_t,\theta_1)+\epsilon_t(i_t)+\beta E_tV([1-i_t]*x_{t+1},\epsilon_{t+1}) \tag{9}$$

1.2 Optimal Stopping Rule

The optimal stopping rule is represented by $i_t=i_t^*(x_t,\epsilon_t;\theta_1)$ that is the solution to the bellman equation at each time period t, such that replacement of engine occurs when the mileage $x_t>x_t^*(\epsilon_t)$ for some optimal mileage cutoff $x_t^*(\epsilon_t)$ dependent on unobservable disturbance.

$$i_t^*(x_t,\epsilon_t)=argmax_{i_t\in\{0,1\}}[u(x_t,i_t;\theta_1)+\epsilon_t(i_t)+\beta E_tV(x_t,\epsilon_t,i_t)] \tag{10}$$

$$i_t^*(x_t,\epsilon_t)=argmax_{i_t\in\{0,1\}}\tilde{V}(x_t,\epsilon_t,i_t) \tag{11}$$

Define choice probability in terms of continuation value:

$$Pr(i_t|x_t;\theta_1)=\frac{exp(-RC*i_t-c([1-i_t]*x_t,\theta_1)+\beta E_tV([1-i_t]*x_{t+1},\epsilon_{t+1}))}{\sum_{i_t\in\{0,1\}}exp(-RC*i_t-c([1-i_t]*x_t,\theta_1)+\beta E_tV([1-i_t]*x_{t+1},\epsilon_{t+1}))} \tag{12}$$

1.3 Computation of Expected Value Function

Define the expected value function incorporating choice probabilities:

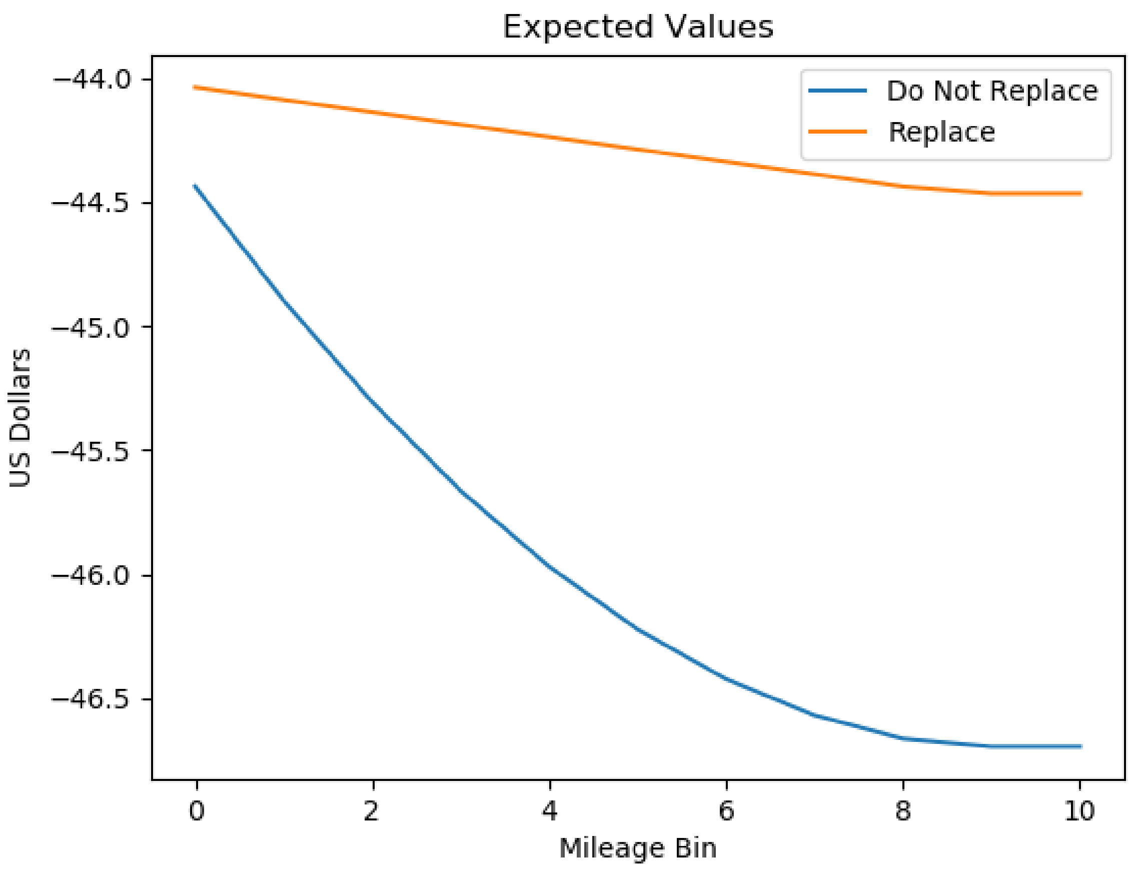
$$E_tV(x_t, i_t; \theta_1) = \int_{x_t} log\{\sum_{i_t \in \{0,1\}} exp[u(x_t, i_t; \theta_1) + \beta E_tV(x_t, i_t)]\}p(dx_t|x_t, i_t) \tag{13}$$

$$x_t \in \{0, 1, 2, ...10\}; i_t \in \{0, 1\} \tag{14}$$

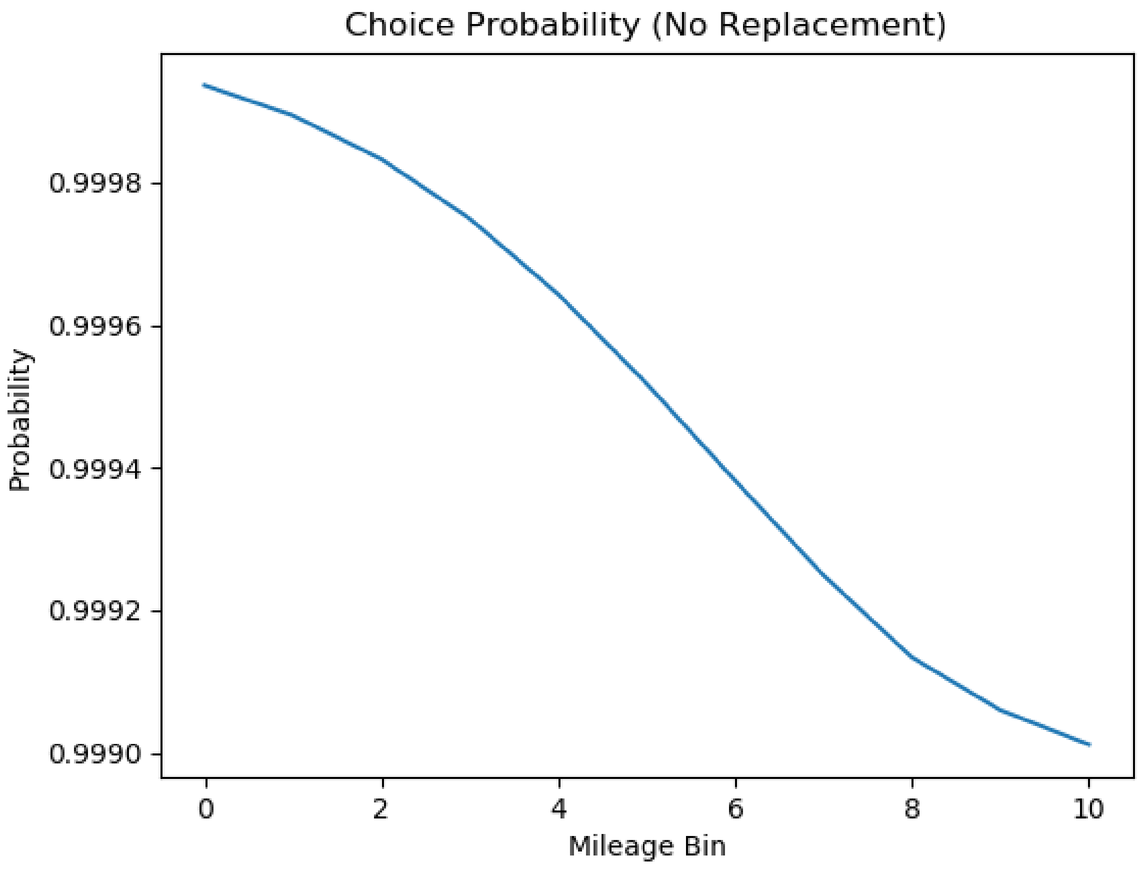
My method is to iterate values of EV based on an initial guess until the convergence of the sum of the flow and discounted continuation utilities to a small difference in comparison to prior iterations within a tolerance band. I use these expected values to calculate my choice probability.

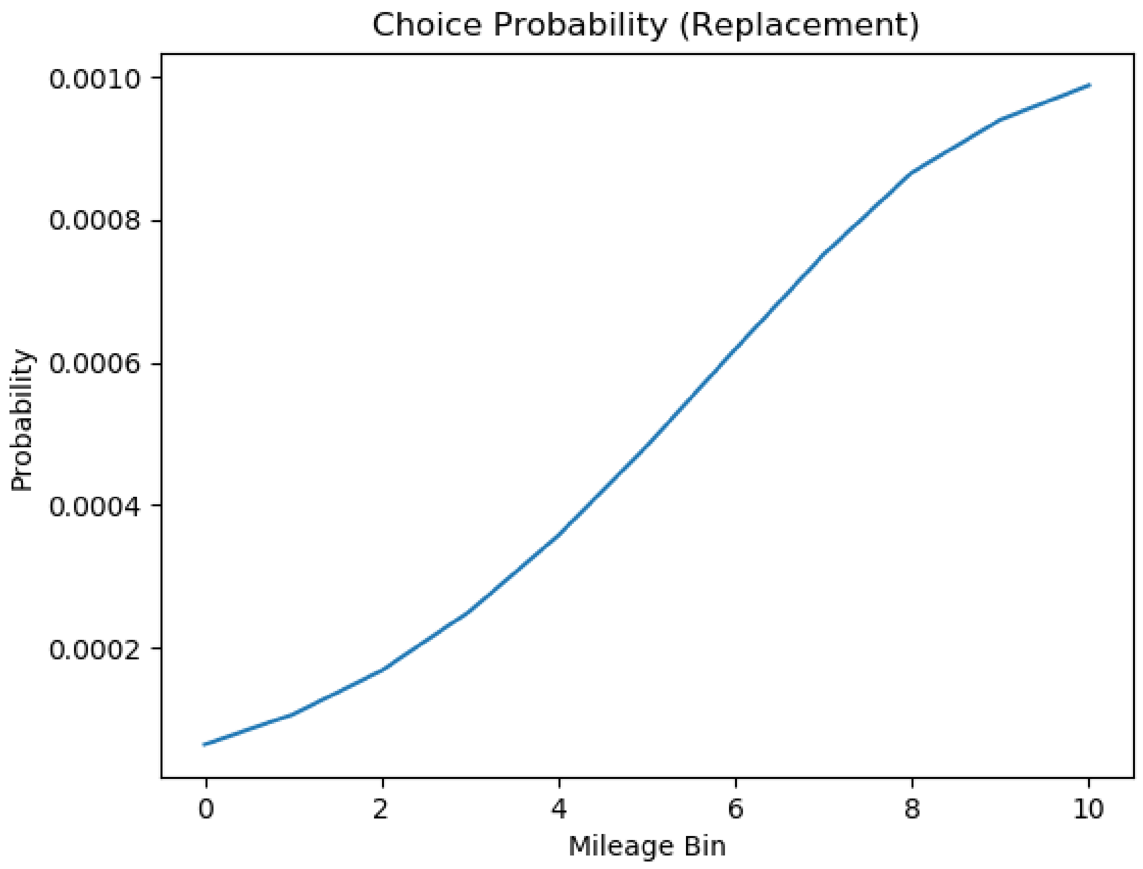
1.4 Graph of Expected Value Function

It appears that due to an error in my methodology (perhaps), an intersection at a single crossing between the expected values of replacing and not replacing does not seem to exist. However, this is not in fact the true case. An optimal stopping occurs when the expected value of not replacing falls below the expected value of replacing, instigating replacement of the bus engine.



1.5 Graphs of Choice Probability


Choice Probability (No Replacement)
Probability
0.9998
0.9996
0.9994
0.9992
0.9990
Mileage Bin
0
2
4
6
8
10


Choice Probability (Replacement)
0.0010
0.0008
0.0006
0.0004
0.0002
Probability
0
2
4
6
8
10
Mileage Bin

1.6 Summary Statistics of Simulated Dataset

Using a seed of 23061994, I obtain my simulated dataset of state variables $\{x_t, i_t, \epsilon_t\}$.

```
Summary Statistics of State Variable x[t] across [100] Buses
Mean:  4.73287
Std:   3.2951709459601637
Max:   10.0
Min:   0.0

[Epsilon(0)]
Mean:  −0.0009730448585669649
Max:  8.06320490171678
Min:  −2.4312376822875166

[Epsilon(1)]
Mean:  0.08335043407806184
Max:  7.379166140483501
Min:  −2.6733304185053752
```

2 Question 2

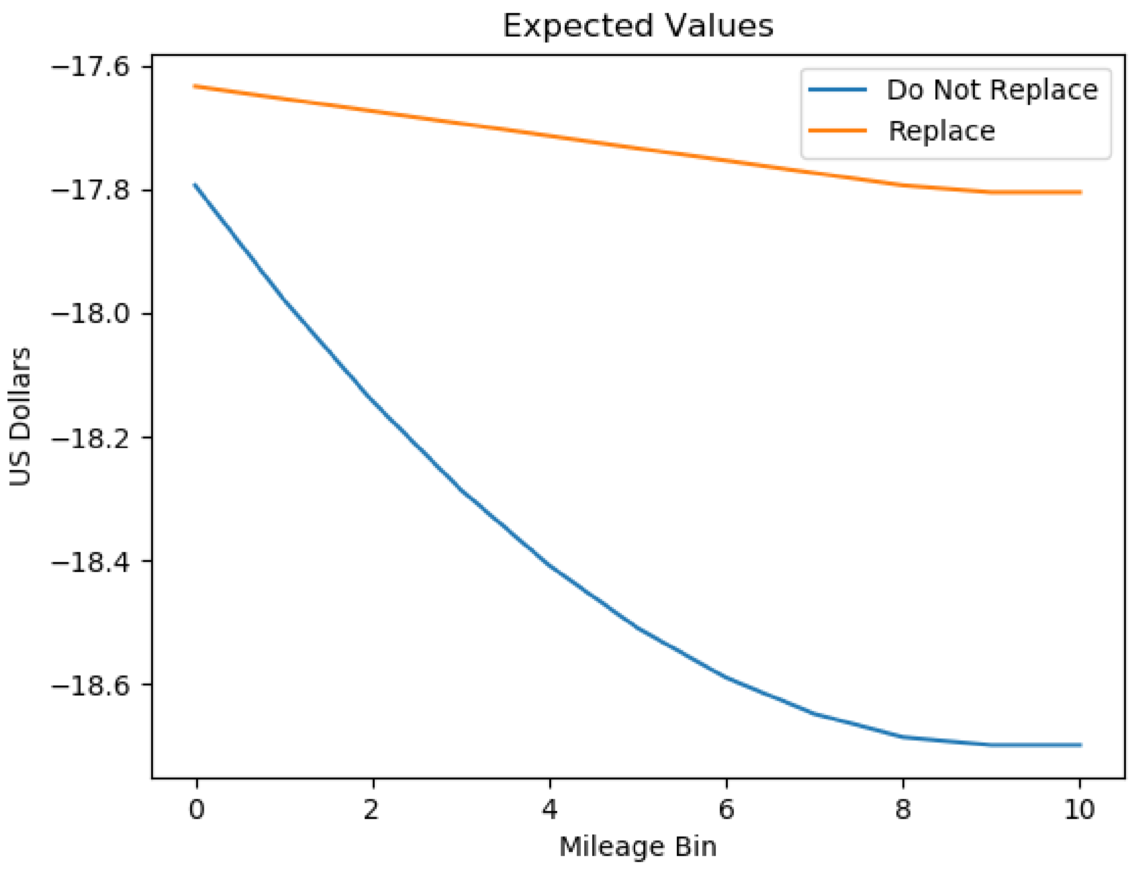
2.1 Estimation of Parameters using Nested Fixed Point Algorithm

Initial step is to estimate θ_3 as was done below.

```
theta3=[[ 0.00000000e+00 -3.57913930e+08  3.57913931e+08]
 [ 0.00000000e+00 -3.61493080e+08  3.61493081e+08]
 [ 0.00000000e+00 -3.61493080e+08  3.61493081e+08]]
```

2.2 Counterfactual

Counterfactually, a higher replacement cost and lower maintenance cost would prolong HZ's decision to remain in the state where he avoids replacement contingent on past states of mileage before optimal stopping is reached.



3 Code

3.1 Output

```
 RESTART: C:\Users\Dell\Documents\Graduate - Economics\Empirical Industrial Organisation I\
   pset3_working\pset3_SMSajidAlSanai.py

Question 1:

cost_replacement=10
cost_maintenance=0.05
beta=0.99
theta3=[0.3 0.5 0.2]

Transition Probability Matrix (n x n)
[[0.3 0.5 0.2 0.  0.  0.  0.  0.  0.  0.  0. ]
 [0.  0.3 0.5 0.2 0.  0.  0.  0.  0.  0.  0. ]
 [0.  0.  0.3 0.5 0.2 0.  0.  0.  0.  0.  0. ]
 [0.  0.  0.  0.3 0.5 0.2 0.  0.  0.  0.  0. ]
 [0.  0.  0.  0.  0.3 0.5 0.2 0.  0.  0.  0. ]
 [0.  0.  0.  0.  0.  0.3 0.5 0.2 0.  0.  0. ]
 [0.  0.  0.  0.  0.  0.  0.3 0.5 0.2 0.  0. ]
 [0.  0.  0.  0.  0.  0.  0.  0.3 0.5 0.2 0. ]
 [0.  0.  0.  0.  0.  0.  0.  0.  0.3 0.5 0.2]
 [0.  0.  0.  0.  0.  0.  0.  0.  0.  0.3 0.7]
 [0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  1. ]]

Iteration [843]:
[[-44.43672519 -44.03718872]
 [-44.89646106 -44.0871851 ]
 [-45.30542188 -44.1371813 ]
 [-45.66305082 -44.1871773 ]
 [-45.96880391 -44.23717309]
 [-46.22205426 -44.28716867]
 [-46.4224129  -44.33716403]
 [-46.56858692 -44.38715914]
 [-46.66237856 -44.43715401]
 [-46.69465285 -44.46465107]
 [-46.69465285 -44.46465107]]

Choice Probability (No Replacement)
[0.99993583 0.99989367 0.99983243 0.99974902 0.99964292 0.99951771
 0.99938183 0.99924904 0.99913383 0.99905992 0.99901177]

Choice Probability (Replacement)
[6.41671297e-05 1.06334248e-04 1.67569366e-04 2.50978334e-04
 3.57078639e-04 4.82291458e-04 6.18172606e-04 7.50956047e-04
 8.66173759e-04 9.40078244e-04 9.88229455e-04]

Summary Statistics of State Variable x[t] across [100] Buses
Mean: 4.73287
Std:  3.2951709459601637
Max:  10.0
Min:  0.0

[Epsilon(0)]
Mean: -0.0009730448585669649
Max: 8.06320490171678
Min: -2.4312376822875166
```

```
[Epsilon(1)]
Mean: 0.08335043407806184
Max: 7.379166140483501
Min: -2.6733304185053752


Question 2:

theta3=[[ 0.00000000e+00 -3.57913930e+08  3.57913931e+08]
 [ 0.00000000e+00 -3.61493080e+08  3.61493081e+08]
 [ 0.00000000e+00 -3.61493080e+08  3.61493081e+08]]

cost_replacement=20
cost_maintenance=0.02
beta=0.99
theta3=[0.3 0.5 0.2]

Transition Probability Matrix (n x n)
[[0.3 0.5 0.2 0.  0.  0.  0.  0.  0.  0.  0. ]
 [0.  0.3 0.5 0.2 0.  0.  0.  0.  0.  0.  0. ]
 [0.  0.  0.3 0.5 0.2 0.  0.  0.  0.  0.  0. ]
 [0.  0.  0.  0.3 0.5 0.2 0.  0.  0.  0.  0. ]
 [0.  0.  0.  0.  0.3 0.5 0.2 0.  0.  0.  0. ]
 [0.  0.  0.  0.  0.  0.3 0.5 0.2 0.  0.  0. ]
 [0.  0.  0.  0.  0.  0.  0.3 0.5 0.2 0.  0. ]
 [0.  0.  0.  0.  0.  0.  0.  0.3 0.5 0.2 0. ]
 [0.  0.  0.  0.  0.  0.  0.  0.  0.3 0.5 0.2]
 [0.  0.  0.  0.  0.  0.  0.  0.  0.  0.3 0.7]
 [0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  1. ]]

Iteration [752]:
[[-17.79399838 -17.63395976]
 [-17.97813281 -17.65395976]
 [-18.14193814 -17.67395976]
 [-18.28518495 -17.69395976]
 [-18.40764947 -17.71395976]
 [-18.50907658 -17.73395976]
 [-18.58930961 -17.75395976]
 [-18.64783442 -17.77395976]
 [-18.68538008 -17.79395976]
 [-18.69829846 -17.80495976]
 [-18.69829846 -17.80495976]]

***
[EOF] Output Terminates
***
```

3.2 Source

```
# Import Libraries
import numpy as np
import scipy as sp
from scipy import optimize
from scipy import io
import matplotlib.pyplot as plt
from matplotlib import colors
```

```
# Question 1:
print('Question 1:')
print('')

# Estimation of Theta3
def MaximumLogLikelihoodTheta3( probability, setting ):
    out = 0
    outcome  = np.array( setting[0:3] )
    binomial = np.array( [ np.sum( outcome ), setting[3] ] )
    binomial_coefficient = sp.special.binom( binomial[0], binomial[1] )
    for i in range( np.size( probability ) ):
        out += np.log( probability[i] ) * outcome[i]
    out += np.log( 1 - ( probability[0] + probability[1] ) ) * outcome[2]
    out += np.log( binomial_coefficient )
    return out

def MaximumLikelihoodTheta3( probability, setting ):
    outcome  = np.array( setting[0:3] )
    binomial = np.array( [ np.sum( outcome ), setting[3] ] )
    binomial_coefficient = sp.special.binom( binomial[0], binomial[1] )
    out = binomial_coefficient
    for i in range( np.size( probability ) ):
        out *= probability[i] ** outcome[i]
    out *= ( 1 - ( probability[0] + probability[1] ) ) ** outcome[2]
    out *= binomial_coefficient
    return out

# Flow Payoff
def FlowPayoff( parameter, state, x ):
    out = -parameter[0] * state - parameter[1] * ( 1 - state ) * x
    return out

# Inner Loop
def InnerLoop( parameter, setting ):
    n              = setting[0]
    tolerance      = setting[1]
    max_iterations = setting[2]
    sup_difference = 1
    expected_value = np.zeros( (n, 2) )
    curr_iteration = 1

    while sup_difference > tolerance and curr_iteration < max_iterations:
        x = np.arange( n )
        t_expected_value = np.zeros( (n, 2) )
        # Do not replace
        t_expected_value[:, 0] = np.log( np.exp( FlowPayoff( parameter, 0, x ) + parameter[2] * ex
        # Replace
        t_expected_value[:, 1] = np.log( np.exp( FlowPayoff( parameter, 0, x ) + parameter[2] * ex
        # Expected Value
        t_expected_value = np.matmul( transition_probability, t_expected_value )
        t_expected_value[n-2:, :] = np.array( [ 0.5 * (t_expected_value[n-3,0] + t_expected_value[
        sup_difference = np.max( abs( t_expected_value - expected_value ) )
        expected_value = t_expected_value
        # Increment Tau
        curr_iteration += 1
    out = expected_value
    print( 'Iteration [' + str(curr_iteration) + ']:' )
    print( out )
    # Returns (Expected Value)
    return out

# Outer Loop
def OuterLoop( parameter, setting ):
    max_iterations = int( setting[0] )
    tolerance      = setting[1]
    n              = int( setting[2] )
    # [ Replacement Cost, Maintenance Cost ]: Initial Guess Vector (Theta)
    # [ i_t, x_t ]: Vector of State Variables at Time t
    out = InnerLoop( parameter, [n, tolerance, max_iterations] )
    '''
    for i in range( n ):
        print( 'Iteration [' + str(i) + ']' )
        out[i, :] = InnerLoop( parameter, [i, tolerance, max_iterations, n] )
       #out[t, :] = sp.optimize.minimize( InnerLoop, [1, 1], [state_i[t], state_x[t], tolerance,
    '''
    # Returns (Expected Values)
    # Returns (Theta)
    return out

# Choice Probability
def ChoiceProbability( parameter, x ):
    #out = 1 / ( 1 + np.exp( parameter[0] + parameter[1] * x - parameter[2] * expected_values[:, 0
    #out = 1 / (1 + np.exp(-parameter[0] + parameter[2] * expected_values[0, 1] + parameter[1] * x
    if parameter[6] == 0:
        out1 =  np.exp( -parameter[1] * x + parameter[2] * expected_values[:, 0] )
        out2 =  np.exp( -parameter[0] + parameter[2] * expected_values[1, 1] )
        out2 += out1
        out = out1 / out2
    if parameter[6] == 1:
        out1 =  np.exp( -parameter[0] + parameter[2] * expected_values[1, 1] )
        out2 =  np.exp( -parameter[1] * x + parameter[2] * expected_values[:, 0] )
        out2 += out1
        out = out1 / out2
    return out

# Main Program Entry Point
n = 11

#state_i = np.zeros( (S,) ) # state_i[:]: i_t
#state_x = np.zeros( (S,) ) # state_x[:]: x_t | i_t = 0
#for s in range( S ):
#    out = np.random.multinomial( 1, [0.3, 0.5, 0.2] )
#    state_i[s] = 0
#    state_x[s] = np.matmul( np.array( [0, 1, 2] ).T, out )

cost_maintenance = 0.05
cost_replacement = 10
beta   = 0.99
theta3 = np.array( [0.3, 0.5, 0.2] )

print( 'cost_replacement=' + str(cost_replacement) )
print( 'cost_maintenance=' + str(cost_maintenance) )
print( 'beta=' + str(beta) )
print( 'theta3=' + str(theta3) )
print('')

# Construct Transition Probability Matrix
transition_probability = np.zeros( (n, n) )
```

```
p = np.array( (theta3[0], theta3[1], theta3[2], 0, 0, 0, 0, 0, 0, 0, 0) )
transition_probability = np.tile( p, (1, n) )
transition_probability = np.reshape( transition_probability , (n, n) )
for i in range(n):
    transition_probability[i, :] = np.roll( transition_probability[i, :], i)
transition_probability[ 9, 0], transition_probability[10, 0:2] = 0, 0
transition_probability[ 9, n-1] = 1 - transition_probability[9, n-2]
transition_probability[10, 10] = 1

print( 'Transition Probability Matrix (n x n)' )
print( transition_probability )
print('')

# Run Outer Loop
expected_values = OuterLoop( np.array( [cost_replacement, cost_maintenance, beta, theta3[0], theta
print('')

# Plot Expected Values
plt.plot( expected_values[:, 0] )
plt.plot( expected_values[:, 1] )
plt.title('Expected Values')
plt.xlabel('Mileage Bin')
plt.ylabel('US Dollars')
plt.legend(('Do Not Replace', 'Replace'))
plt.show()

# Calculate Choice Probabilities
choice_probability0 = ChoiceProbability( np.array( [cost_replacement, cost_maintenance, beta, thet
choice_probability1 = ChoiceProbability( np.array( [cost_replacement, cost_maintenance, beta, thet

print( 'Choice Probability (No Replacement)' )
print( choice_probability0)
plt.plot( choice_probability0 )
plt.title( 'Choice Probability (No Replacement)' )
plt.xlabel('Mileage Bin')
plt.ylabel('Probability')
plt.show()
print('')

print( 'Choice Probability (Replacement)' )
print( choice_probability1)
plt.plot( choice_probability1 )
plt.title( 'Choice Probability (Replacement)' )
plt.xlabel('Mileage Bin')
plt.ylabel('Probability')
plt.show()
print('')

# Simulation of Dataset
N = 100
S = 1000

# Set Seed Value
np.random.seed(23061994)

state_x = np.zeros( (S, N) )

for k in range(N):
    #state_x[0, k] = int(np.random.uniform(0, 10))
    state_x[0, k] = np.random.randint(0, 7)

for j in range(N):
    for l in range(S-1):
        uniform_draw = np.random.uniform(0, 1)
        #print( uniform_draw )
        num1 = 0
        num2 = 0
        depen_x = int(state_x[l, j])
        #state_x[ l+1, j ] = 0
        for k in range(n):
            num1 = np.sum( transition_probability[ depen_x, 0:max(0, k-1) ] )
            num2 = np.sum( transition_probability[ depen_x, 0:k ] )
            #print( str(num1) + '<' + str(uniform_draw) + '<=' + str(num2) )
            if uniform_draw > num1 and uniform_draw <= num2:
                state_x[l+1, j] = int(k)
        #print( state_x[l+1, j] )

print( 'Summary Statistics of State Variable x[t] across [100] Buses' )
print( 'Mean: ' + str( np.mean( state_x ) ) )
print( 'Std:  ' + str( np.std( state_x ) ) )
print( 'Max:  ' + str( np.max( state_x ) ) )
print( 'Min:  ' + str( np.min( state_x ) ) )
print('')

state_epsilon = np.zeros( (S, 2) )

for t in range( S ):
    state_epsilon[t, 0] = np.random.gumbel( 0, 1 ) - 0.57721
    state_epsilon[t, 1] = np.random.gumbel( 0, 1 ) - 0.57721

for i in range(2):
    print( '[Epsilon(' + str(i) + ')]' )
    print( 'Mean: ' + str( np.mean(state_epsilon[:, i]) ) )
    print( 'Max:  ' + str( np.max(state_epsilon[:, i]) ) )
    print( 'Min:  ' + str( np.min(state_epsilon[:, i]) ) )
    print('')

state_i = np.zeros( (S, N) )


# Question 2
print('Question 2:')
print('')

# Estimation of Theta3
theta3 = np.zeros( (3, 3) )
for k in range(3):
    out = sp.optimize.minimize( MaximumLikelihoodTheta3, [0, 0], [0, 1, 2, k] )
    theta3[k, 0:2] = out.x
    theta3[k,   2] = 1 - np.sum( theta3[k, 0:2] )
print( 'theta3=' + str(theta3) )
print('')

cost_maintenance = 0.02
cost_replacement = 20
beta   = 0.99
theta3 = np.array( [0.3, 0.5, 0.2] )
```

```
print( 'cost_replacement=' + str(cost_replacement) )
print( 'cost_maintenance=' + str(cost_maintenance) )
print( 'beta=' + str(beta) )
print( 'theta3=' + str(theta3) )
print('')

# Construct Transition Probability Matrix
transition_probability = np.zeros( (n, n) )
p = np.array( (theta3[0], theta3[1], theta3[2], 0, 0, 0, 0, 0, 0, 0, 0) )
transition_probability = np.tile( p, (1, n) )
transition_probability = np.reshape( transition_probability, (n, n) )
for i in range(n):
    transition_probability[i, :] = np.roll( transition_probability[i, :], i)
transition_probability[ 9, 0], transition_probability[10, 0:2] = 0, 0
transition_probability[ 9, n-1] = 1 - transition_probability[9, n-2]
transition_probability[10, 10] = 1

print( 'Transition Probability Matrix (n x n)' )
print( transition_probability )
print('')

# Run Outer Loop
expected_values = OuterLoop( np.array( [cost_replacement, cost_maintenance, beta, theta3[0], theta
print('')

# Plot Expected Values
plt.plot( expected_values[:, 0] )
plt.plot( expected_values[:, 1] )
plt.title('Expected Values')
plt.xlabel('Mileage Bin')
plt.ylabel('US Dollars')
plt.legend(('Do Not Replace', 'Replace'))
plt.show()

print('')
print('***')
print('[EOF] Output Terminates')
print('***')
# EOF
```