ORIE6750 HW2 Yucheng Chen

**Problem 1.**

(a).

A close up of a map

Description automatically generated

The blue curve is and the orange curve is .

Python code:

import numpy as np

import matplotlib.pyplot as plt

def computeQFactor1(prob,horizon,V,M):#problem a

#T(p,y)

#likelihood

Y = np.array([0,1])

Q = 0

for y in Y:

prob\_next = prob\*f1\_prob\_mass\_bern(y)/(prob\*f1\_prob\_mass\_bern(y) + (1.0-prob)\*f0\_prob\_mass\_bern(y))

i = round(prob\_next / (1.0/M))

Q = Q + V[int(i),int(horizon-1)]\*(prob\*f1\_prob\_mass\_bern(y) + (1.0-prob)\*f0\_prob\_mass\_bern(y))

return Q

def f0\_prob\_mass\_bern(y):

if y == 0:

return 1.0/2

elif y == 1:

return 1.0/2

else:

print('wrong input')

return -1

def f1\_prob\_mass\_bern(y):

if y == 0:

return 1.0/3

elif y == 1:

return 2.0/3

else:

print('wrong input')

return -1

if \_\_name\_\_== "\_\_main\_\_":

c = 0.05

H = 20

M = 200

delta = 1.0/M

p\_arr = np.arange(0.,1. + delta/2.0, delta)

H\_arr = np.arange(0,H+1)

V = np.zeros((p\_arr.size,H+1))

Qvalue = np.zeros((p\_arr.size,H+1))

for h in H\_arr:

for i in range(M+1):

p = p\_arr[i]

if h == 0:

V[i,h] = min(p,1-p)

else:

Qvalue[i,h] = c + computeQFactor1(p,h,V,M)

V[i,h] = min(p,1-p, Qvalue[i,h])

plt.figure('')

ax = plt.plot(p\_arr,V[:,0])

ax = plt.plot(p\_arr,Qvalue[:,H])

plt.show()

(b).

A screenshot of a cell phone

Description automatically generated

The blue curve is and the orange curve is .

Python code:

import numpy as np

from scipy.stats import norm

import matplotlib.pyplot as plt

def computeQFactor2(prob,horizon,V,M):#problem a

#T(p,y)

#likelihood

delta = 0.1

Y = np.arange(-7,7,delta)

Q = 0

for y in Y:

prob\_next = prob\*f1\_prob\_mass\_Gauss(y,delta)/(prob\*f1\_prob\_mass\_Gauss(y,delta) + (1.0-prob)\*f0\_prob\_mass\_Gauss(y,delta))

i = round(prob\_next / (1.0/M))

Q = Q + V[int(i),int(horizon-1)]\*(prob\*f1\_prob\_mass\_Gauss(y,delta) + (1.0-prob)\*f0\_prob\_mass\_Gauss(y,delta))

return Q

def f0\_prob\_mass\_Gauss(y,delta):

return norm(0,1).cdf(y) - norm(0,1).cdf(y-delta)

def f1\_prob\_mass\_Gauss(y,delta):

return norm(1,1).cdf(y) - norm(1,1).cdf(y-delta)

if \_\_name\_\_== "\_\_main\_\_":

c = 0.05

H = 20

M = 20

delta = 1.0/M

p\_arr = np.arange(0.,1. + delta/2.0, delta)

H\_arr = np.arange(0,H+1)

V = np.zeros((p\_arr.size,H+1))

Qvalue = np.zeros((p\_arr.size,H+1))

for h in H\_arr:

for i in range(M+1):

p = p\_arr[i]

if h == 0:

V[i,h] = min(p,1-p)

else:

Qvalue[i,h] = c + computeQFactor2(p,h,V,M)

V[i,h] = min(p,1-p, Qvalue[i,h])

plt.figure('')

ax = plt.plot(p\_arr,V[:,0])

ax = plt.plot(p\_arr,Qvalue[:,H])

plt.show()

**Problem 2.**

(a). For Bernoulli likelihoods, as shown by the following plot, where the blue curve is and the orange curve is . .

A close up of a map

Description automatically generated

For infinite horizon problem, we use Value Iteration to solve . The curve of is shown as follows:

A close up of a map

Description automatically generated

Thus, we can determine

Then we conduct the simulation and consider the loss function and the sample cost . We have done 200 epochs with 1000 simulations for each. The distribution of average cost is shown as the following histogram:

A picture containing screenshot

Description automatically generated

As shown by the histogram above, the average cost is concentrated around 0.47, which is pretty closed to the value computed from the finite horizon approach: .

Python code:

import numpy as np

import numpy as np

from scipy.stats import norm

import matplotlib.pyplot as plt

from scipy.stats import bernoulli

def f0\_prob\_mass\_bern(y):

if y == 0:

return 1.0/2

elif y == 1:

return 1.0/2

else:

print('wrong input')

return -1

def f1\_prob\_mass\_bern(y):

if y == 0:

return 1.0/3

elif y == 1:

return 2.0/3

else:

print('wrong input')

return -1

def computeQFactor1(prob,V,M):#problem a

Y = np.array([0,1])

Q = 0

for y in Y:

prob\_next = prob\*f1\_prob\_mass\_bern(y)/(prob\*f1\_prob\_mass\_bern(y) + (1.0-prob)\*f0\_prob\_mass\_bern(y))

i = round(prob\_next / (1.0/M))

Q = Q + V[int(i)]\*(prob\*f1\_prob\_mass\_bern(y) + (1.0-prob)\*f0\_prob\_mass\_bern(y))

return Q

def ValueIter(V,N,M,c):

delta = 1.0/M

for i in range(N):

for j in range(M):

p = j\*delta

#print(c+computeQFactor1(p,V,M))

V[j] = min(p, 1-p, c + computeQFactor1(p,V,M))

Q = np.zeros(M)

for i in range(M):

p = i\*delta

Q[i] = c + computeQFactor1(p,V,M)

return V, Q

if \_\_name\_\_== "\_\_main\_\_":

M = 1000

N = 1000

c = 0.05

delta = 1.0/M

p\_arr = np.arange(0.,1.0 + delta/2, delta)

V0 = np.minimum(p\_arr,1 - p\_arr)

V = np.random.rand(M+1)

V,Q = ValueIter(V,N,M+1,c)

a = -1

b = -1

flag = 0

for i in range(M+1):

if Q[i] < V0[i]:

if i\*delta < 0.5 and flag == 0:

a = i\*delta

flag = 1

else:

b = i\*delta

sampNum = 1000

epochs = 2000

epochCost = np.zeros(epochs)

for e in range(epochs):

sampleCosts = -np.ones(sampNum)

p0 = 0.5

for i in range(sampNum):

u = np.random.uniform(0,1,1)#sample from p(theta)

if u >= p0:

theta = 0

else:

theta = 1

p = p0

goCost = 0

while (p > a and p < b):

if u > p:#sample f0

y = bernoulli.rvs(1.0/2.0, size = 1)

else:

y = bernoulli.rvs(2.0/3.0, size = 1)

p = p\*f1\_prob\_mass\_bern(y)/(p\*f1\_prob\_mass\_bern(y) + (1-p)\*f0\_prob\_mass\_bern(y))

goCost = goCost + c

#print(p)

if p < a:

theta\_est = 0

else:

theta\_est = 1

if theta == theta\_est:

sampleCosts[i] = goCost

else:

sampleCosts[i] = 1 + goCost

epochCost[e] = np.average(sampleCosts)

plt.figure()

ax = plt.hist(epochCost, bins = 50)

plt.show()

(b). For Gaussian likelihoods, as shown by the following plot, where the blue curve is and the orange curve is . .

A screenshot of a cell phone

Description automatically generated

For infinite horizon problem, we use Value Iteration to solve . The curve of is shown as follows:

A screenshot of a cell phone

Description automatically generated

Thus, we can determine

Then we conduct the simulation and consider the loss function and the sample cost . We have done 50 epochs with 1000 simulations for each. The distribution of average cost is shown as the following histogram:

As shown by the histogram above, the average cost is concentrated around 0.28, which is pretty closed to the value computed from the finite horizon approach: .

A picture containing screenshot

Description automatically generated

Python code:

import numpy as np

import numpy as np

from scipy.stats import norm

import matplotlib.pyplot as plt

from scipy.stats import bernoulli

import math

def f0\_prob\_mass\_Gauss(y,delta):

return norm(0,1).cdf(y) - norm(0,1).cdf(y-delta)

def f1\_prob\_mass\_Gauss(y,delta):

return norm(1,1).cdf(y) - norm(1,1).cdf(y-delta)

def computeQFactor2(prob,V,M):#problem a

delta = 0.1

Y = np.arange(-7,7,delta)

Q = 0

for y in Y:

prob\_next = prob\*f1\_prob\_mass\_Gauss(y,delta)/(prob\*f1\_prob\_mass\_Gauss(y,delta) + (1.0-prob)\*f0\_prob\_mass\_Gauss(y,delta))

#print(prob\_next)

i = round(prob\_next / (1.0/(M-1)))

Q = Q + V[int(i)]\*(prob\*f1\_prob\_mass\_Gauss(y,delta) + (1.0-prob)\*f0\_prob\_mass\_Gauss(y,delta))

return Q

def ValueIter(V,N,M,c):

delta = 1.0/(M-1)

V\_prev = V/2.0

while (np.sum(abs(V-V\_prev)) > 1e-2):

V\_prev = np.copy(V)

for j in range(M):

p = j\*delta

#print(c+computeQFactor1(p,V,M))

V[j] = min(p, 1-p, c + computeQFactor2(p,V,M))

print(np.sum(abs(V-V\_prev)))

Q = np.zeros(M)

for i in range(M):

p = i\*delta

Q[i] = c + computeQFactor2(p,V,M)

return V, Q

if \_\_name\_\_ == "\_\_main\_\_":

M = 1000

N = 1000

c = 0.05

delta = 1.0/M

p\_arr = np.arange(0.,1.0+delta/2.0, delta)

V0 = np.minimum(p\_arr,1 - p\_arr)

V = np.random.rand(p\_arr.size)

V,Q = ValueIter(V,N,p\_arr.size,c)

a = -1

b = -1

flag = 0

for i in range(M+1):

if Q[i] < V0[i]:

if i\*delta < 0.5 and flag == 0:

a = i\*delta

flag = 1

else:

b = i\*delta

sampNum = 1000

epochs = 200

epochCost = np.zeros(epochs)

for e in range(epochs):

sampleCosts = -np.ones(sampNum)

p0 = 0.5

for i in range(sampNum):

u = np.random.uniform(0,1,1)#sample from p(theta)

if u >= p0:

theta = 0

else:

theta = 1

p = p0

goCost = 0

while (p > a and p < b):

if u > p0:#sample f0

y = np.random.normal(0,1,1)

else:

y = np.random.normal(1,1,1)

p = p\*f1\_prob\_mass\_Gauss(y,delta)/(p\*f1\_prob\_mass\_Gauss(y,delta) + (1-p)\*f0\_prob\_mass\_Gauss(y,delta))

goCost = goCost + c

if p < a:

theta\_est = 0

else:

theta\_est = 1

if theta == theta\_est:

sampleCosts[i] = goCost

else:

sampleCosts[i] = 1 + goCost

epochCost[e] = np.average(sampleCosts)

plt.figure()

ax = plt.hist(epochCost, bins = 5)

plt.show()

**Problem 3.**

(a). Given the loss function .

Then, the recursive equation of value function is as follows:

If , then we must have because the decision has been made when there are steps left. Then we have

If  but . We also have because implies =.

If and .

Given the fact that the value function is concave, then we have.

Therefore,

Thus,

In summary,

(b). Suppose as the intersection of with in the interval

Consider that from (a) and For is concave in . Then the intersection of with should

follow .

Consider the loss function is symmetric, . Therefore,

**Problem 4.**

In the Value Iteration algorithm,

For each discretization of interval ,

where is the number of discretization, and

Consider two initializations

Then in the first iteration i.e.

Different initializations of the only the third term in the minimization operator.

Then for ,

Therefore,

Reasoning by reduction, for

Also,