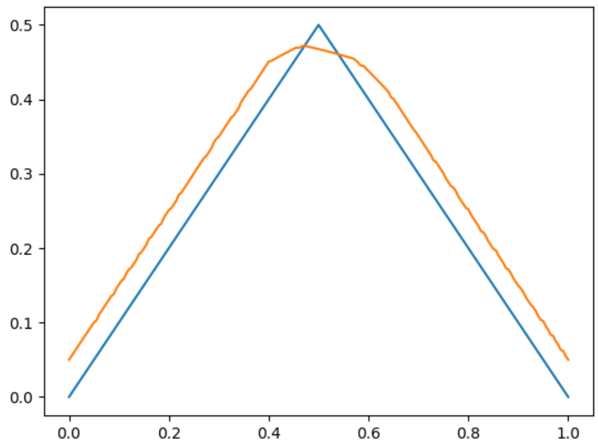
Problem 1.

(a).



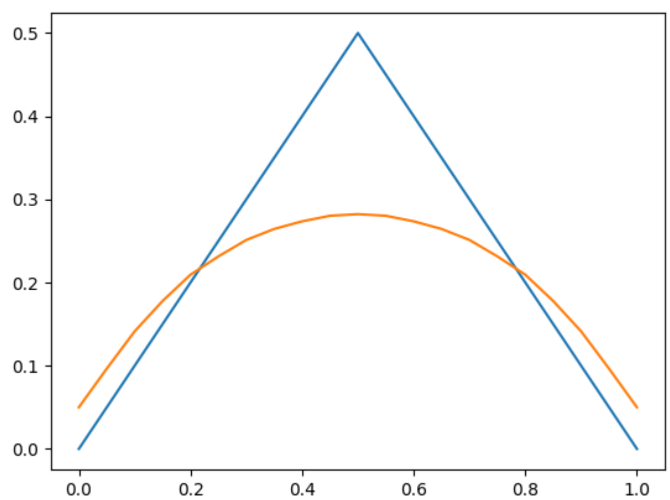
The blue curve is $Q(p, H, stop)$ and the orange curve is $Q(p, H, go)$.

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).



The blue curve is $Q(p, H, stop)$ and the orange curve is $Q(p, H, go)$.

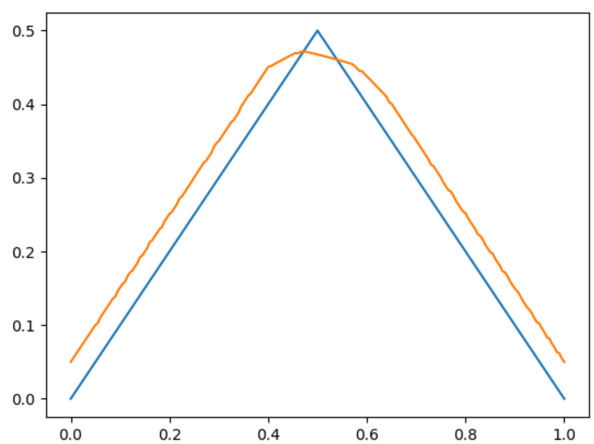
Python code:

```
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)
```

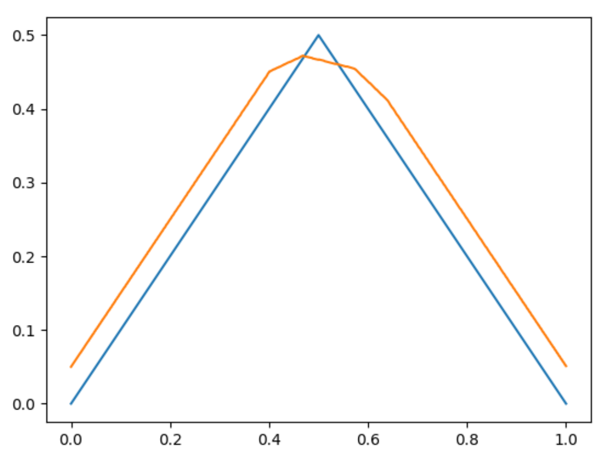
```
    H_arr = np.arange(0,H+1)
  V = np.zeros((p_arr.size,H+1))
    Qvalue = np.zeros((p_arr.size,H+1))
    for h in H_arr:
            for i in range(M+1):
                    p = p_arr[i]
                    if h == 0:
                            V[i,h] = min(p,1-p)
                    else:
                            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 $V(p,0)$ and the orange curve is $V(p,100)$. $V(0.5,100) = 0.4708$.

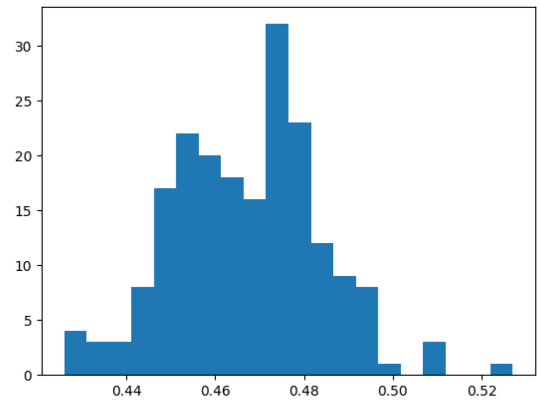


For infinite horizon problem, we use Value Iteration to solve $V(p)$. The curve of $V(p)$ is shown as follows:



Thus, we can determine $a = 0.472, b = 0.539$

Then we conduct the simulation and consider the loss function $1\{\theta \neq \hat{\theta}\}$ and the sample cost c. We have done 200 epochs with 1000 simulations for each. The distribution of average cost is shown as the following histogram:

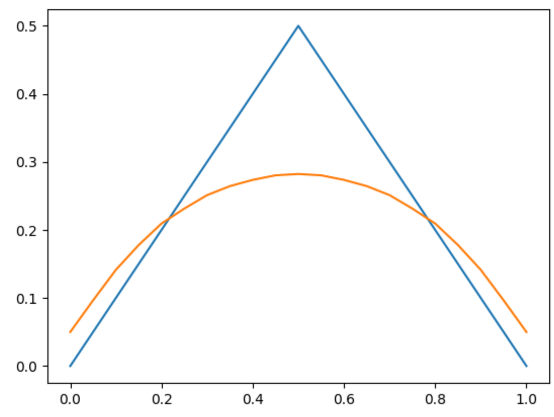


As shown by the histogram above, the average cost is concentrated around 0.47, which is pretty closed to the value computed from the finite horizon approach: $V(0.5,100) = 0.4708$.

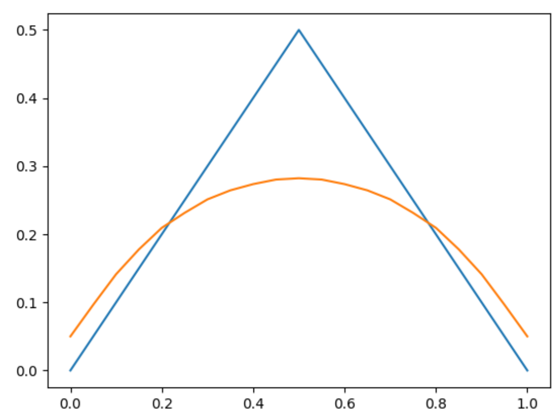
Python code:
```
import numpy as np
import numpy as np
from scipy.stats import norm
import matplotlib.pyplot as plt
from scipy.stats import bernoulli


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 $V(p, 0)$ and the orange curve is $V(p, 100)$. $V(0.5, 100) = 0.278$.



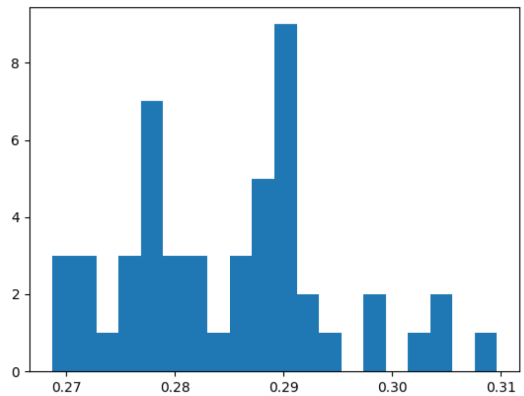
For infinite horizon problem, we use Value Iteration to solve $V(p)$. The curve of $V(p)$ is shown as follows:



Thus, we can determine $a = 0.2, b = 0.8$.

Then we conduct the simulation and consider the loss function $1\{\theta \neq \hat{\theta}\}$ and the sample cost c. 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: $V(0.5,100) = 0.278$.



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 $L(\hat{\theta}, \theta) = 1\{\hat{\theta} \neq \theta\}$.

$$V(p, 0) = \min\{p, 1-p\}$$

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

$$V(p, h) = \min\{p, 1-p, c + \int V(T(p, y), h-1)[pf_1(y) + (1-p)f_0(y)]dy\}$$

If $V(p, h) = \min\{p, 1-p\}$, then we must have $V(p, h-1) = \min\{p, 1-p\}$ because the decision has been made when there are h steps left. Then we have $V(p, h) \leq V(p, h-1)$

If $V(p, h) \neq \min\{p, 1-p\}$ but $V(p, h-1) = \min\{p, 1-p\}$. We also have $V(p, h) \geq V(p, h-1)$ because $V(p, h) \neq \min\{p, 1-p\}$ implies $V(p, h) \leq \min\{p, 1-p\} = V(p, h-1)$.

If $V(p, h) \neq \min\{p, 1-p\}$ and $V(p, h-1) \neq \min\{p, 1-p\}$.

Given the fact that the value function $V(p, h)$ is concave, then we have.

$$\int V(T(p, y), h-1)[pf_1(y) + (1-p)f_0(y)]dy$$
$$\leq V(\int T(p, y)[pf_1(y) + (1-p)f_0(y)]dy, h-1)$$

$$\int T(p, y)[pf_1(y) + (1-p)f_0(y)]dy$$
$$= \int \frac{pf_1(y)}{[pf_1(y) + (1-p)f_0(y)]}[pf_1(y) + (1-p)f_0(y)]dy = p$$

Therefore,

$$\int V(T(p, y), h-1)[pf_1(y) + (1-p)f_0(y)]dy \leq V(p, h-1)$$

Thus,

$$V(p, h) = c + \int V(T(p, y), h-1)[pf_1(y) + (1-p)f_0(y)]dy < V(p, h-1)$$

In summary,

$$V(p, h) \leq V(p, h-1)$$

(b). Suppose a_n as the intersection of $f(p) = p \wedge (1-p)$ with $V(p, H-n)$ in the interval $p \in \left[0, \frac{1}{2}\right]$.

Consider that $V(p, H-n) \leq V(p, H-n-1)$ from (a) and For $\forall h, V(p,h)$ is concave in $p \in [0,1]$. Then the intersection of $V(p, H-n-1)$ with $f(p) = p \wedge (1-p)$ $a_{n+1} \in \left[0, \frac{1}{2}\right]$ should follow $a_n \geq a_{n+1}$.

Consider the loss function is symmetric, $b_n = 1 - a_n$. Therefore, $b_n \leq b_{n+1}$

Problem 4.

In the Value Iteration algorithm,
For each discretization of interval $[0,1]$,

$$V(i) = \min\{\delta i, 1 - \delta i, c + \sum_{k=1}^{H} V(k) q(i,k)\}$$

where H is the number of discretization, $\delta = \frac{1}{M}$ and

$$q(i) = \sum_{y} 1\{T(\delta i, y) \in [\delta k, \delta(k+1)]\}\{\delta i f_1(y) + (1 - \delta i) f_0(y)\}$$

Consider two initializations

$$\forall i, \bar{V}^0(i) > V^0(i)$$

Then in the first iteration i.e. $m = 1$

$$\bar{V}^1(i) = \min\{\delta i, 1 - \delta i, c + \sum_{k=1}^{H} \bar{V}^0(k) q(i,k)\}$$

$$V^1(i) = \min\{\delta i, 1 - \delta i, c + \sum_{k=1}^{H} V^0(k) q(i,k)\}$$

Different initializations of the $V(p)$ only the third term in the minimization operator.
Then for $\forall i = 1, \dots H$,

$$c + \sum_{k=1}^{H} \bar{V}^0(k) q(i,k) - \left(c + \sum_{k=1}^{H} V^0(k) q(i,k)\right)$$

$$= \sum_{k=1}^{H} \{\bar{V}^0(k) - V^0(k)\} q(i,k) > 0$$

Therefore, $\bar{V}^1(i) \geq V^1(i), \forall i = 1, \dots H$
Reasoning by reduction, for $m = 1,2 \dots M$

$$\bar{V}^m(i) \geq V^m(i), \forall i = 1, \dots H$$

Also,

$$V^m(i) \geq \underline{V}^m(i), \forall i = 1, \dots H$$