**1. [b]**

**2. [d]**

**Vmin is close to 0.036, close to [b] 0.01; while Vfirst and Vrand are both 0.499, close to 0.5.**

**Q1 and Q2 are based on the following codes:**

import numpy as np

class EachCoin:

def \_\_init\_\_(self, nflips):

self.flips = np.random.randint(0, 2, nflips)

self.heads = np.average(self.flips) #suppose head = 1

class CoinFlips:

def \_\_init\_\_(self, ncoins):

self.results = [EachCoin(10) for x in range(ncoins)]

self.heads = np.array([x.heads for x in self.results])

self.first = self.heads[0]

self.rand = self.heads[np.random.randint(0, ncoins)]

self.min = self.heads.min()

def Hoeffding(coins,times):

results = [CoinFlips(coins) for x in range(times)]

mins = np.array([x.min for x in results])

firsts = np.array([x.first for x in results])

rands = np.array([x.rand for x in results])

minimum = np.average(mins)

first = np.average(firsts)

rand = np.average(rands)

print(f"first coin: {first}")

print(f"min freq: {minimum}")

print(f"rand coin: {rand}")

def main():

Hoeffding(1000,100000)

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

main()

**3. [e]**

**There exists two cases: 1) h makes an error in approximating f, and y = f(x); 2) h correctly approximates f, and y ≠ f(x). So in total Perr = 𝜇\*𝜆 + (1 − 𝜇)\*(1 − 𝜆 ).**

**4. [b]**

**𝑃 = 1 − 𝜆 − 𝜇 + 2𝜆 𝜇. When 𝜆 = 0.5, P = 1− 𝜆 is independent of 𝜇.**

**5. [c]**

**6. [c]**

**7. [a]**

**Q5, Q6 and Q7 are based on the following codes:**

clear ; clc; close all;

d = 2; % dimension of x vector

a = -1; b = 1; % range [a,b]

N\_in = 100; % number of in-sample points

m = 1000; % number of runs

N\_out = 1000; % number of out-of-sample points

Ein\_total = 0;

Eout\_total = 0;

% data set

data = unifrnd(a,b,d,N\_in);

% scatter(data(1,:),data(2,:)); hold on;

for iter = 1:m

clear x y;

% choose a randome line by taking two random points

p1 = unifrnd (a,b,d,1);

p2 = unifrnd (a,b,d,1);

k = (p1(2)-p2(2))/(p1(1)-p2(1));

c = p1(2) - k \* p1(1);

% fplot(@(x) k\*x+c,'LineWidth',1.5); hold on;

% axis([-1 1 -1 1]);

y = sign(data(2,:)-data(1,:)\*k-c);

y = y';

x = [ones(N\_in,1) data'];

% generate N\_out fresh points for Eout estimation

data\_out = unifrnd(a,b,d,N\_out);

y\_out = sign(data\_out(2,:)-data\_out(1,:)\*k-c);

y\_out = y\_out';

x\_out = [ones(N\_out,1) data\_out'];

% linear regression

w = (x'\*x)\x'\*y;

% in-sample error

Ein = length(find(sign(x\*w)-y))/N\_in;

Ein\_total = Ein\_total+Ein;

% fplot(@(x) -w(2)/w(3)\*x-w(1)/w(3),'LineWidth',1.5);

% legend('dataset','original f','h from linear regression');

% out-of-sample error

Eout = length(find(sign(x\_out\*w)-y\_out))/N\_out;

Eout\_total = Eout\_total+Eout;

end

Ein\_avg = Ein\_total/m;

Eout\_avg = Eout\_total/m;

**8. [d]**

**Q8 is based on the following codes:**

clear ; clc; close all;

d = 2; % dimension of x vector

a = -1; b = 1; % range [a,b]

N\_in = 1000; % number of in-sample points

m = 1000; % number of runs

Ein\_total = 0;

% data set

data = unifrnd(a,b,d,N\_in);

for iter = 1:m

clear x y;

y = sign(data(2,:).^2+data(1,:).^2-0.6);

% y\_nonoise = y'; % y without noise

rand\_index = ceil(unifrnd(0,N\_in,1,N\_in\*0.1));

y(rand\_index) = -y(rand\_index); % random selection to add noise

y\_noise = y';

x = [ones(N\_in,1) data'];

% linear regression

w = (x'\*x)\x'\*y\_noise;

% in-sample error

Ein = length(find(sign(x\*w)-y\_noise))/N\_in;

Ein\_total = Ein\_total+Ein;

end

Ein\_avg = Ein\_total/m;

**9. [a]**

**Q9 is based on the following codes:**

clear ; clc; close all;

d = 2; % dimension of x vector

a = -1; b = 1; % range [a,b]

N\_in = 1000; % number of in-sample points

m = 1000; % number of runs

N\_out = 1000; % number of out-of-sample points

% data set

data = unifrnd(a,b,d,N\_in);

y = sign(data(2,:).^2+data(1,:).^2-0.6);

% y without noise

y\_nonoise = y';

% random selection to add noise

rand\_index = ceil(unifrnd(0,N\_in,1,N\_in\*0.1));

y(rand\_index) = -y(rand\_index);

y\_noise = y';

x = [ones(N\_in,1) data' (data(1,:).\*data(2,:))' ...

(data(1,:).^2)' (data(2,:).^2)'];

w = (x'\*x)\x'\*y\_noise;

**10. [b]**

**Q10 is based on the following codes:**

clear ; clc; close all;

d = 2; % dimension of x vector

a = -1; b = 1; % range [a,b]

N\_in = 1000; % number of in-sample points

m = 1000; % number of runs

N\_out = 1000; % number of out-of-sample points

Eout\_total = 0;

% data set

data = unifrnd(a,b,d,N\_in);

for iter = 1:m

clear x y;

y = sign(data(2,:).^2+data(1,:).^2-0.6);

% y\_nonoise = y'; % y without noise

% random selection to add noise

rand\_index = ceil(unifrnd(0,N\_in,1,N\_in\*0.1));

y(rand\_index) = -y(rand\_index);

y\_noise = y';

x = [ones(N\_in,1) data' (data(1,:).\*data(2,:))' ...

(data(1,:).^2)' (data(2,:).^2)'];

% generate N\_out fresh points for Eout estimation

data\_out = unifrnd(a,b,d,N\_out);

y\_out = sign(data\_out(2,:).^2+data\_out(1,:).^2-0.6);

% random selection to add noise

rand\_index\_out = ceil(unifrnd(0,N\_out,1,N\_out\*0.1));

y(rand\_index\_out) = -y(rand\_index\_out);

y\_noise\_out = y\_out';

x\_out = [ones(N\_out,1) data\_out' (data\_out(1,:).\*data\_out(2,:))' ...

(data\_out(1,:).^2)' (data\_out(2,:).^2)'];

% linear regression

w = (x'\*x)\x'\*y\_noise;

% out-of-sample error

Eout = length(find(sign(x\_out\*w)-y\_noise\_out))/N\_out;

Eout\_total = Eout\_total+Eout;

end

Eout\_avg = Eout\_total/m;