# <PRV Project>

**(A) 2018142180 Lee Gyu Geun**

**(B) 2018142031 Jeong Kun Mo**

**(C) 2018142023 Jo Sung Min**

# Introduction (A 33%, B 34%, C 33%)

This project solves the problem by applying a random variable in three different situations.

In Question 1, we will design a quantizer based on conditional probability model by event. As we deal with 2, 4, 10 bits of quantizer, we will see how the number of bits differs parameters. We will calculate the value of inputs and outputs. We also can calculate the mean square difference of values and SQNR by using them. We assume the signal follows Laplace random variable, Uniform random variable, and Gaussian random variable.

In Question 2, we will design a spam mail detector by using random variable estimation techniques that are MAP, ML, and CLT. In the method of Naive Bayes Classification, we will first implement a 'nbayes_learn' function by using 'spam_labels' text file as an input, and then use 'spam_features' text file to test the accuracy of the classifier that we have made. Next, we will make a difference in learning methods(MAP, ML, CLT) and the number of samples(1500, 500, 10), and then check the difference between them.

In Question 3, it contains Basic Monte Carlo integration and Monty Hall problem. For Basic Monte Carlo integration, we will solve simple integration by applying to matlab. Then, we will show why it can’t apply Gaussian random variable. And, we will apply more complicated integral and compare results from different number of samples. For Monty Hall problem, we will see the probability to choose a car is really 50:50 to switch or not at last step of show. And make a graph to see change of payoff.

# Q1: Quantizer Design based on Conditional Probability Model by Event (A 100%)

## Part A : 2-bits Quantizer Design

**(a)**

$E[X^2] = \int_{-\infty}^{\infty} x^2 f_X(x)dx = \int_{-\infty}^{\infty} x^2 * \frac{1}{2}e^{-|x|}dx = \int_0^{\infty} x^2 * e^{-x}dx = 2$

**(b)**

$f_{X|B_i} = \frac{f_X(x)}{P[B_i]} = \frac{f_X(x)}{\int_{x_{i-1}}^{x_i} f_X(x)dx} = \frac{\frac{1}{2}e^{-|x|}}{\int_{x_{i-1}}^{x_i} \frac{1}{2}e^{-|x|}dx}$

$$E[B_i] = E[x_{i-1} < X < x_i] = \frac{\int_{x_{i-1}}^{x_i} x * f_X(x)dx}{\int_{x_{i-1}}^{x_i} f_X(x)dx} = \frac{\int_{x_{i-1}}^{x_i} x * \frac{1}{2}e^{-|x|}dx}{\int_{x_{i-1}}^{x_i} \frac{1}{2}e^{-|x|}dx}$$

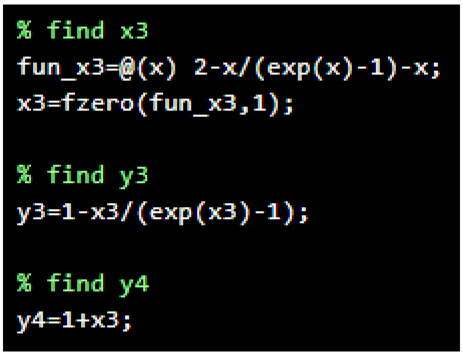
**(c)**

$y_3 = E[B_3] = E[x_2 < X < x_3] = \frac{\int_{x_2}^{x_3} x*f_X(x)dx}{\int_{x_2}^{x_3} f_X(x)dx} = \frac{\int_0^{x_3} x*\frac{1}{2}e^{-x}dx}{\int_0^{x_3}\frac{1}{2}e^{-x}dx} = \frac{1-e^{-x_3}(x_3+1)}{1-e^{-x_3}} = 1 - \frac{x_3}{e^{x_3}-1}$

$y_4 = E[B_4] = E[x_3 < X < x_4] = \frac{\int_{x_3}^{\infty} x*f_X(x)dx}{\int_{x_3}^{\infty} f_X(x)dx} = \frac{\int_{x_3}^{\infty} x*\frac{1}{2}e^{-x}dx}{\int_{x_3}^{\infty}\frac{1}{2}e^{-x}dx} = \frac{e^{-x_3}(x_3+1)}{e^{-x_3}} = x_3 + 1$

$x_3 = \frac{y_3+y_4}{2}, 2 - \frac{x_3}{e^{x_3}-1} = x_3$

**[Matlab code]**

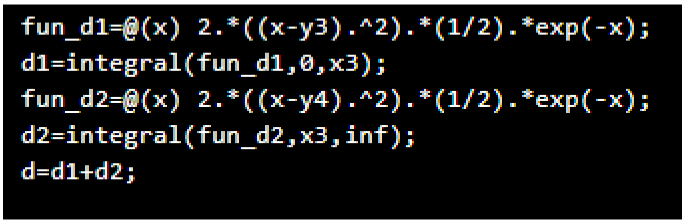


```
% find x3
fun_x3=@(x) 2-x/(exp(x)-1)-x;
x3=fzero(fun_x3,1);

% find y3
y3=1-x3/(exp(x3)-1);

% find y4
y4=1+x3;
```

The result is {x3, y3, y4} = {1.5936, 0.5936, 2.5936}

**(d)**

**[Matlab code]**



```
fun_d1=@(x) 2.*((x-y3).^2).*(1/2).*exp(-x);
d1=integral(fun_d1,0,x3);
fun_d2=@(x) 2.*((x-y4).^2).*(1/2).*exp(-x);
d2=integral(fun_d2,x3,inf);
d=d1+d2;
```

The result is d=0.3524

**(e)**

$SQNR = \frac{E[X^2]}{d} = \frac{2}{0.3524} = 5.6755$

## Part B : m-bits Quantizer Design

**(a)**

$$y_i = E[B_i] = E[x_{i-1} < X < x_i] = \frac{\int_{x_{i-1}}^{x_i} x * f_X(x)dx}{\int_{x_{i-1}}^{x_i} f_X(x)dx} = \frac{\int_{i-1}^{x_i} x * \frac{1}{2}e^{-x}dx}{\int_{i-1}^{x_i} \frac{1}{2}e^{-x}dx} = \frac{(x_{i-1}+1)exp exp\,(-x_{i-1}) - (x_i+1)exp\,(-x_i)}{exp exp\,(-x_{i-1}) - exp\,(-x_i)}$$

**(b)**

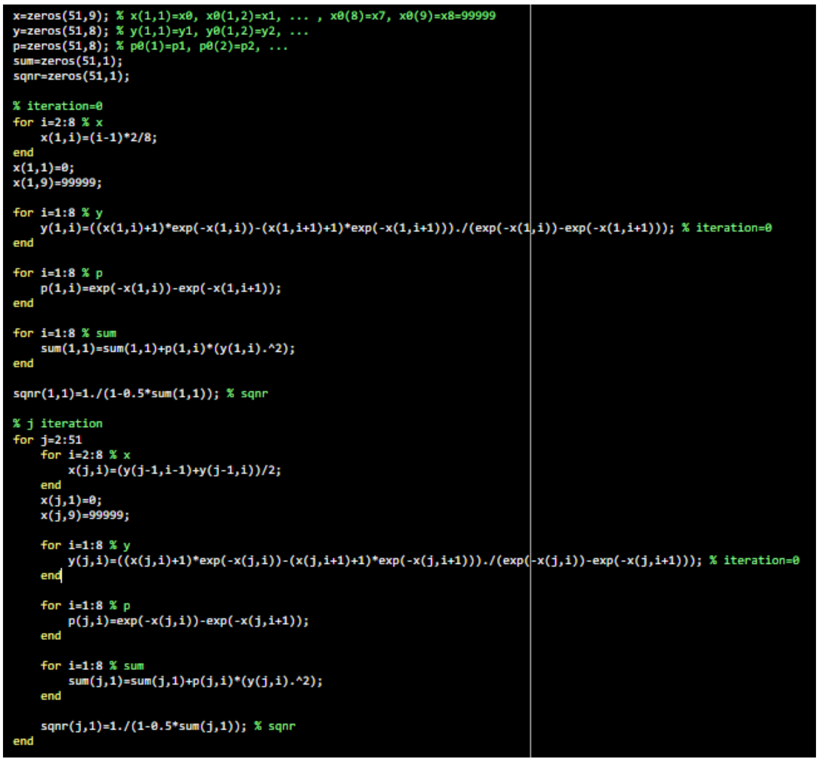
$$SQNR = \frac{E[X^2]}{\sum_{i=1}^{M} \int_{x_{i-1}}^{x_i} (x-y_i)^2 f_X(x)dx} = \frac{2}{\sum_{i=1}^{M} \int_{x_{i-1}}^{x_i} [x^2 f_X(x) - 2x f_X(x) y_i + y_i^2 f_X(x)]dx}$$

$$= \frac{2}{2+\sum_{i=1}^{M} \int_{x_{i-1}}^{x_i} -2x f_{X|B_i}(x)P[B_i]y_i + y_i^2 f_{X|B_i}(x)P[B_i]dx} = \frac{2}{2+\sum_{i=1}^{M} -2y_i^2 p_i + y_i^2 p_i}$$

$$= \frac{2}{2+\sum_{i=1}^{M} -y_i^2 p_i} = \frac{1}{1-\frac{1}{2}\sum_{i=1}^{M} y_i^2 p_i},\ p_i = exp\ exp\,(-x_{i-1})\ - exp\ exp\,(-x_i)$$

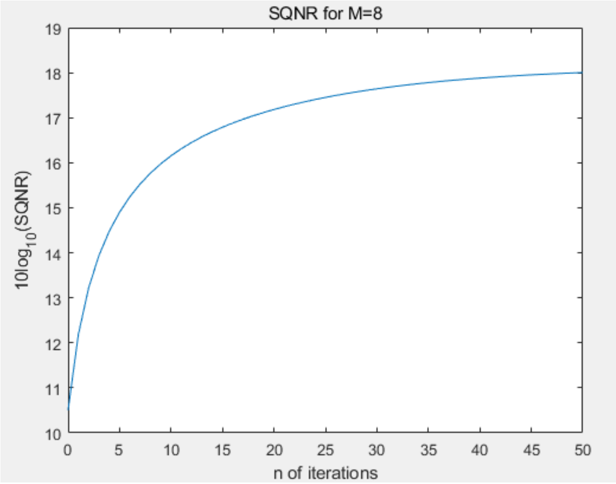
**(c)**

**[Matlab code]**



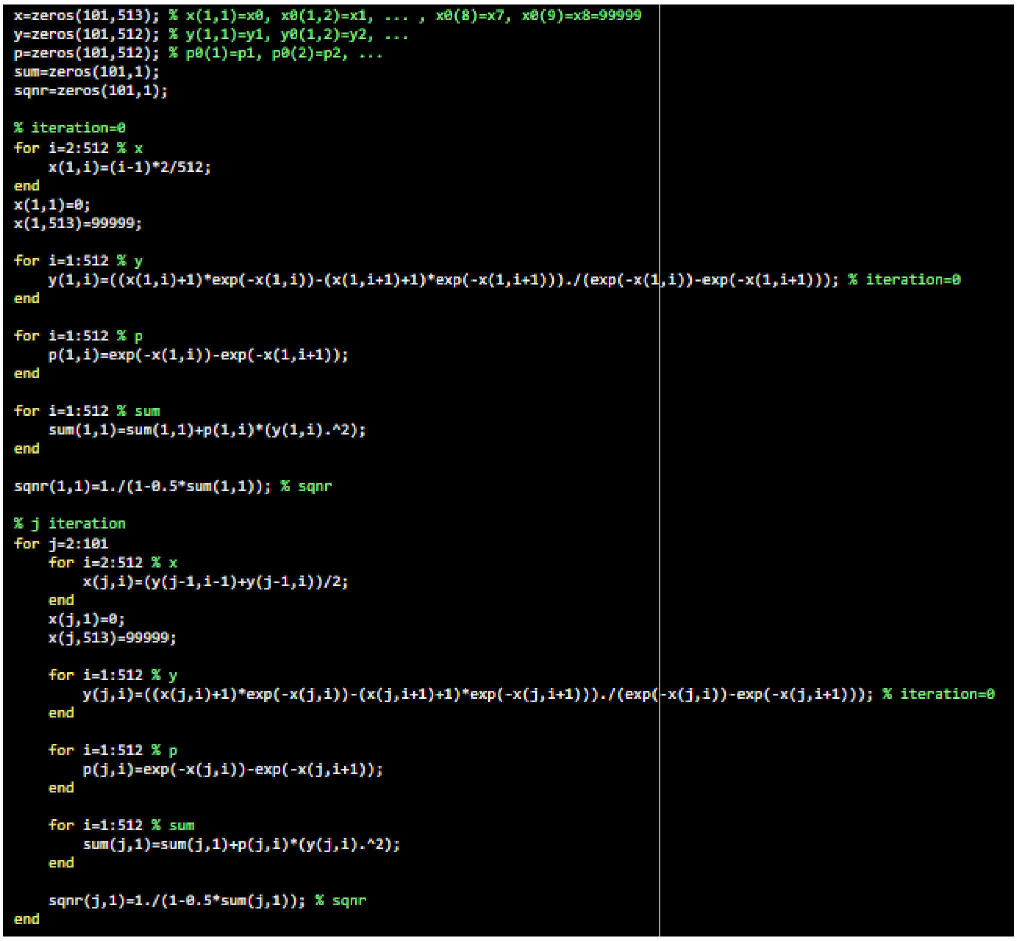
```
x=zeros(51,9); % x(1,1)=x0, x0(1,2)=x1, ... , x0(8)=x7, x0(9)=x8=99999
y=zeros(51,8); % y(1,1)=y1, y0(1,2)=y2, ...
p=zeros(51,8); % p0(1)=p1, p0(2)=p2, ...
sum=zeros(51,1);
sqnr=zeros(51,1);

% iteration=0
for i=2:8 % x
    x(1,i)=(i-1)*2/8;
end
x(1,1)=0;
x(1,9)=99999;

for i=1:8 % y
    y(1,i)=((x(1,i)+1)*exp(-x(1,i))-(x(1,i+1)+1)*exp(-x(1,i+1)))./(exp(-x(1,i))-exp(-x(1,i+1))); % iteration=0
end

for i=1:8 % p
    p(1,i)=exp(-x(1,i))-exp(-x(1,i+1));
end

for i=1:8 % sum
    sum(1,1)=sum(1,1)+p(1,i)*(y(1,i).^2);
end

sqnr(1,1)=1./(1-0.5*sum(1,1)); % sqnr

% j iteration
for j=2:51
    for i=2:8 % x
        x(j,i)=(y(j-1,i-1)+y(j-1,i))/2;
    end
    x(j,1)=0;
    x(j,9)=99999;

    for i=1:8 % y
        y(j,i)=((x(j,i)+1)*exp(-x(j,i))-(x(j,i+1)+1)*exp(-x(j,i+1)))./(exp(-x(j,i))-exp(-x(j,i+1))); % iteration=0
    end

    for i=1:8 % p
        p(j,i)=exp(-x(j,i))-exp(-x(j,i+1));
    end

    for i=1:8 % sum
        sum(j,1)=sum(j,1)+p(j,i)*(y(j,i).^2);
    end

    sqnr(j,1)=1./(1-0.5*sum(j,1)); % sqnr
end
```

**[Graph for 10log_10(SQNR) of using a 2-bits quantizer]**



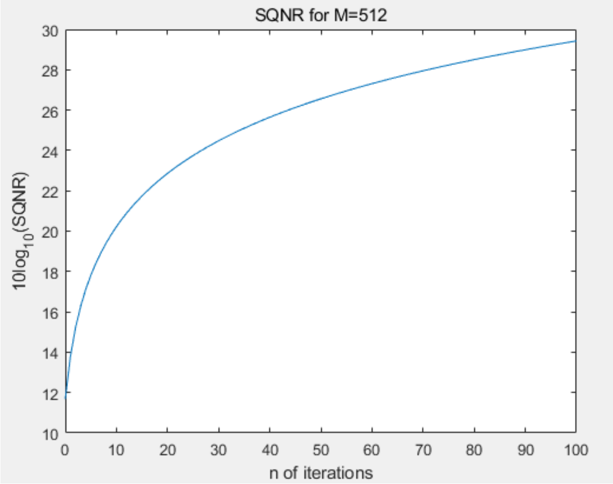
SQNR of 2-bits quantizer was 5.6755, but SQNR of 4-bits quantizer was 11.2319 at 0 iteration. It means that there is more power of the original signal when we use more bits. Because the mean-squared distortion(d) decreases, SQNR increases. Also, by increasing the number of iterations, SQNR increases, and it means the value of x, y becomes closer to the original value. When the number of iterations becomes large enough, SQNR doesn't change rapidly as 0~15 iterations.

**(e)**



```
x=zeros(101,513); % x(1,1)=x0, x0(1,2)=x1, ... , x0(8)=x7, x0(9)=x8=99999
y=zeros(101,512); % y(1,1)=y1, y0(1,2)=y2, ...
p=zeros(101,512); % p0(1)=p1, p0(2)=p2, ...
sum=zeros(101,1);
sqnr=zeros(101,1);

% iteration=0
for i=2:512 % x
    x(1,i)=(i-1)*2/512;
end
x(1,1)=0;
x(1,513)=99999;

for i=1:512 % y
    y(1,i)=((x(1,i)+1)*exp(-x(1,i))-(x(1,i+1)+1)*exp(-x(1,i+1)))./(exp(-x(1,i))-exp(-x(1,i+1))); % iteration=0
end

for i=1:512 % p
    p(1,i)=exp(-x(1,i))-exp(-x(1,i+1));
end

for i=1:512 % sum
    sum(1,1)=sum(1,1)+p(1,i)*(y(1,i).^2);
end

sqnr(1,1)=1./(1-0.5*sum(1,1)); % sqnr

% j iteration
for j=2:101
    for i=2:512 % x
        x(j,i)=(y(j-1,i-1)+y(j-1,i))/2;
    end
    x(j,1)=0;
    x(j,513)=99999;

    for i=1:512 % y
        y(j,i)=((x(j,i)+1)*exp(-x(j,i))-(x(j,i+1)+1)*exp(-x(j,i+1)))./(exp(-x(j,i))-exp(-x(j,i+1))); % iteration=0
    end

    for i=1:512 % p
        p(j,i)=exp(-x(j,i))-exp(-x(j,i+1));
    end

    for i=1:512 % sum
        sum(j,1)=sum(j,1)+p(j,i)*(y(j,i).^2);
    end

    sqnr(j,1)=1./(1-0.5*sum(j,1)); % sqnr
end
```

**[Graph for 10log_10(SQNR) of using a 10-bits quantizer]**



## Part C - (a) : Uniform random variable

**(a)**

$$E[X^2] = \int_{-1}^{1} x^2 f_X(x)dx = \int_{-1}^{1} x^2 * \frac{1}{2}dx = \frac{1}{3}$$

**(b)**

$$f_{X|B_i} = \frac{f_X(x)}{P[B_i]} = \frac{f_X(x)}{\int_{x_{i-1}}^{x_i} f_X(x)dx} = \frac{\frac{1}{2}}{\int_{x_{i-1}}^{x_i}\frac{1}{2}dx} = \frac{1}{x_i - x_{i-1}}$$

$$E[B_i] = E[x_{i-1} < X < x_i] = \frac{\int_{x_{i-1}}^{x_i} x * f_X(x)dx}{\int_{x_{i-1}}^{x_i} f_X(x)dx} = \frac{\int_{x_{i-1}}^{x_i} x * \frac{1}{2}dx}{\int_{x_{i-1}}^{x_i}\frac{1}{2}dx} = \frac{\frac{1}{2}(x_i^2 - x_{i-1}^2)}{x_i - x_{i-1}} = \frac{1}{2}(x_i + x_{i-1})$$

**(c)**

$$y_3 = E[B_3] = E[x_2 < X < x_3] = \frac{1}{2}x_3$$

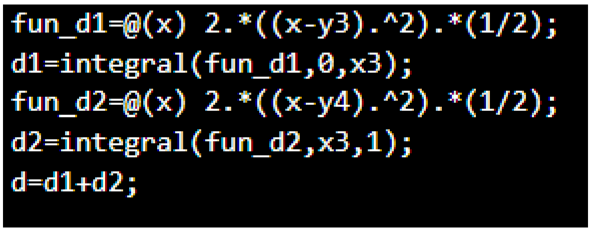
$$y_4 = E[B_4] = E[x_3 < X < x_4] = \frac{1}{2}(1 + x_3)$$

$$x_3 = \frac{y_3 + y_4}{2}, 2 * x_3 = \frac{1}{2} + x_3$$

The result is {x3, y3, y4} = {0.5, 0.25, 0.75}

**(d)**

**[Matlab code]**



```
fun_d1=@(x) 2.*((x-y3).^2).*(1/2);
d1=integral(fun_d1,0,x3);
fun_d2=@(x) 2.*((x-y4).^2).*(1/2);
d2=integral(fun_d2,x3,1);
d=d1+d2;
```

The result is d=0.0208

**(e)**

$$SQNR = \frac{E[X^2]}{d} = \frac{2}{0.0208} = 16.0256$$

## Part C - (b) : Gaussian random variable

**(a)**

$$E[X^2] = \int_{-\infty}^{\infty} x^2 f_X(x) dx = \int_{-\infty}^{\infty} x^2 * \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} dx = 1$$

**(b)**

$$f_{X|B_i} = \frac{f_X(x)}{P[B_i]} = \frac{f_X(x)}{\int_{x_{i-1}}^{x_i} f_X(x) dx} = \frac{\frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}}{\int_{x_{i-1}}^{x_i} \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} dx}$$

$$E[B_i] = E[x_{i-1} < X < x_i] = \frac{\int_{x_{i-1}}^{x_i} x * f_X(x) dx}{\int_{x_{i-1}}^{x_i} f_X(x) dx} = \frac{\int_{x_{i-1}}^{x_i} x * \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} dx}{\int_{x_{i-1}}^{x_i} \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} dx}$$

**(c)**

$$y_3 = E[B_3] = E[x_2 < X < x_3] = \frac{\int_{x_2}^{x_3} x * f_X(x) dx}{\int_{x_2}^{x_3} f_X(x) dx} = \frac{\int_0^{x_3} x * \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} dx}{\int_0^{x_3} \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} dx} = \frac{\int_0^{x_3} x * e^{-\frac{x^2}{2}} dx}{\int_0^{x_3} e^{-\frac{x^2}{2}} dx}$$

$$y_4 = E[B_4] = E[x_3 < X < x_4] = \frac{\int_{x_3}^{\infty} x * f_X(x) dx}{\int_{x_3}^{\infty} f_X(x) dx} = \frac{\int_{x_3}^{\infty} x * \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} dx}{\int_{x_3}^{\infty} \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} dx} = \frac{\int_{x_3}^{\infty} x * e^{-\frac{x^2}{2}} dx}{\int_{x_3}^{\infty} e^{-\frac{x^2}{2}} dx}$$

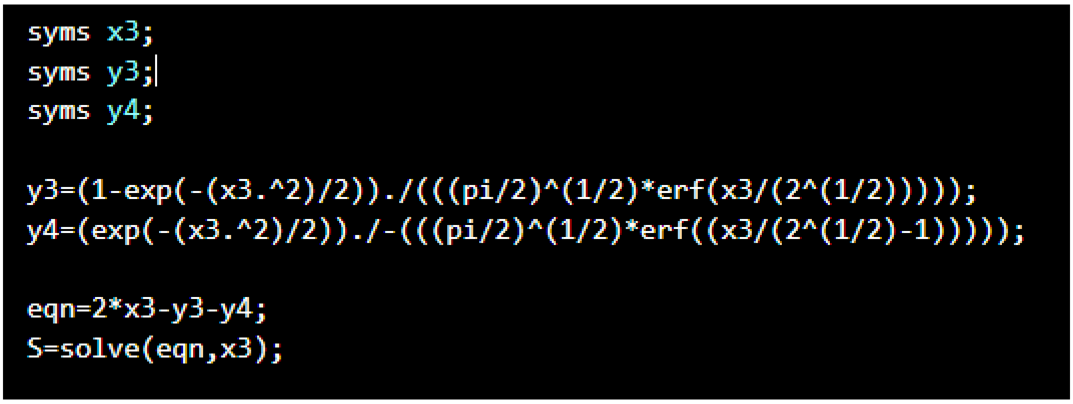
$$\int_0^{x_3} x * e^{-\frac{x^2}{2}} dx = 1 - e^{-\frac{x_3^2}{2}}$$

$$\int_0^{x_3} e^{-\frac{x^2}{2}} dx = \sqrt{\frac{\pi}{2}} erf\left(\frac{x_3}{\sqrt{2}}\right)$$

$$\int_{x_3}^{\infty} x * e^{-\frac{x^2}{2}} dx = -e^{-\frac{x_3^2}{2}}$$

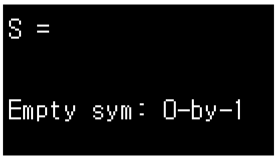
$$\int_{x_3}^{\infty} e^{-\frac{x^2}{2}} dx = -\sqrt{\frac{\pi}{2}} erf\left(\frac{x_3}{\sqrt{2}} - 1\right)$$

**[Matlab code]**



```
syms x3;
syms y3;
syms y4;

y3=(1-exp(-(x3.^2)/2))./(((pi/2)^(1/2)*erf(x3/(2^(1/2)))));
y4=(exp(-(x3.^2)/2))./-(((pi/2)^(1/2)*erf((x3/(2^(1/2)-1)))));

eqn=2*x3-y3-y4;
S=solve(eqn,x3);
```

The result is



```
S =

Empty sym: 0-by-1
```

It means there is no real value for x3, and it means that y3 and y4 also don't have real value.

**(d), (e)**

We can't calculate d or SQNR as we can't calculate x3, y3, y4. It means when the signal follows a Gaussian random variable, we can't calculate the SQNR or d. We can assume there is no signal that follows a Gaussian random variable.

## Discussion for Part A,B

When we use more bits, we can get a larger SQNR. We could increase SQNR by doing more iterations, too. We wondered if there is a new parameter such as SQNR that determines an accuracy for a bit design. Also, there was a large enough number of iterations that SQNR didn't increase rapidly. We wanted to know what the number is.

## Discussion for Part C

We used different types of random variables. We could see how SQNR or {x3, y3, y4} become different. We saw SQNR was larger when we used a Uniform random variable than Laplace random variable. We could find an invalid value using a Gaussian random variable. It means some random variable cannot be quantized.

# Q2: Random variable estimation - Spam Email Detector (C 100%)

## ◆ Basic Ideas for Question 1~3

**-params(1).classprobs(i) =** $P\left(C=c_i\right)$

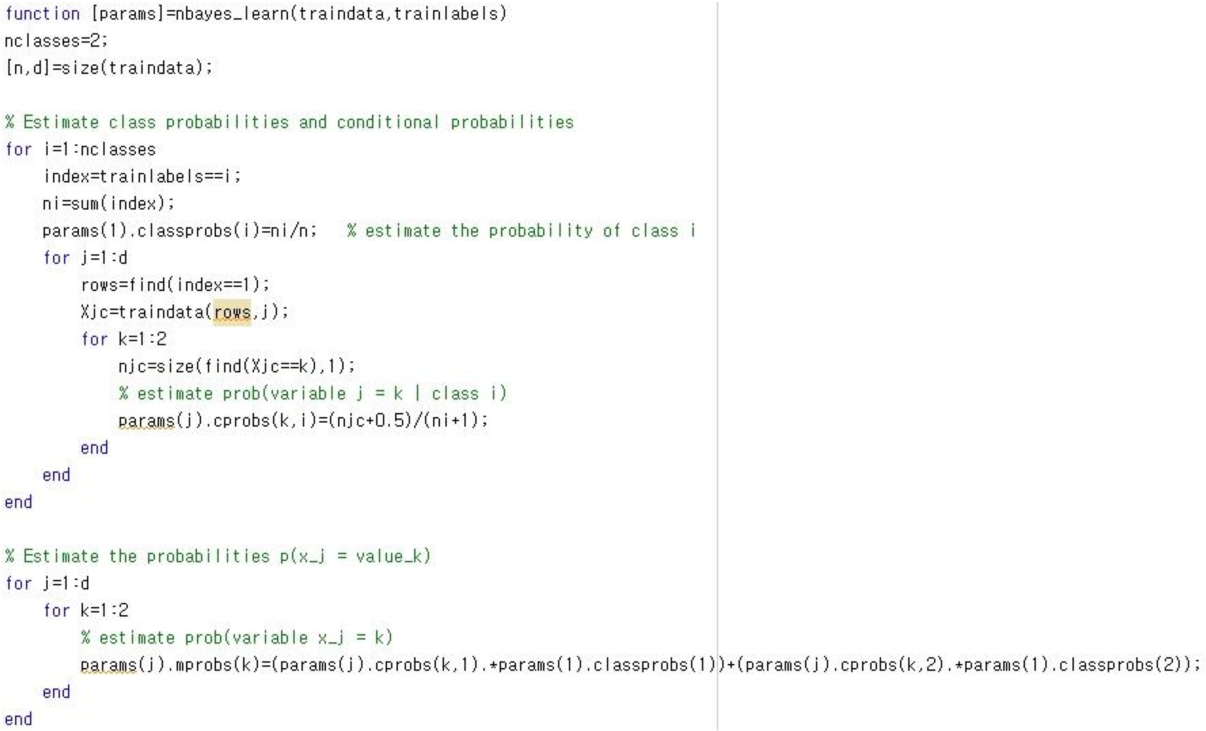
**-params(j).cprobs(k,i) =** $P\left(X_j=k \mid C=c_i\right)$ (+smoothing)

**-params(j).mprobs(k) =** $P\left(X_j=k\right)$

**-MAP:** $argmax_c = P\left(C=c_i \mid X_1,X_2,\cdots,X_d\right) = \dfrac{P\left(X_1,X_2,\cdots,X_d \mid C=c_i\right)P\left(C=c_i\right)}{P\left(X_1,X_2,\cdots,X_d\right)}$

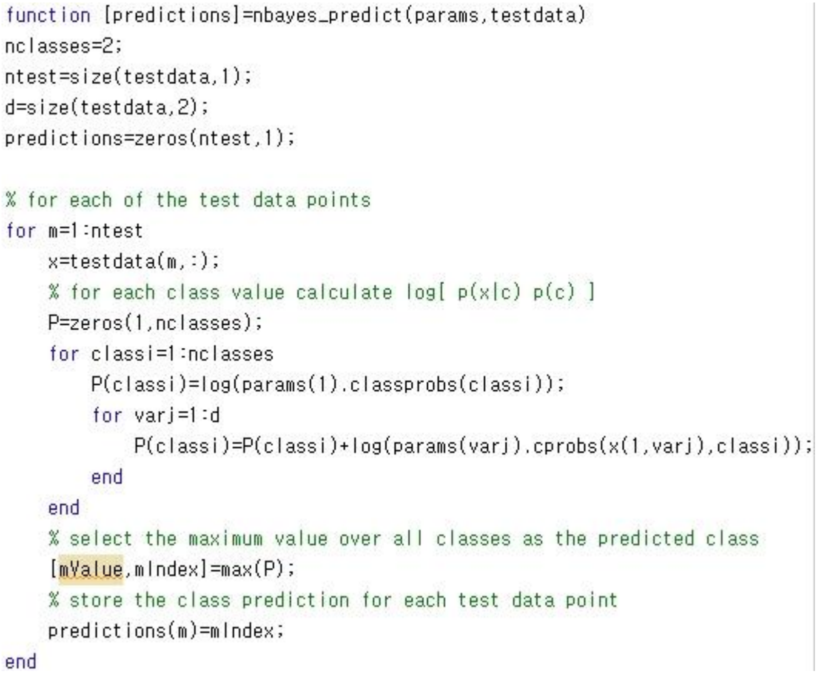
## ◆ Codes used for Question 1~3

**-nbayes_learn.m**



```
function [params]=nbayes_learn(traindata,trainlabels)
nclasses=2;
[n,d]=size(traindata);

% Estimate class probabilities and conditional probabilities
for i=1:nclasses
    index=trainlabels==i;
    ni=sum(index);
    params(1).classprobs(i)=ni/n;   % estimate the probability of class i
    for j=1:d
        rows=find(index==1);
        Xjc=traindata(rows,j);
        for k=1:2
            njc=size(find(Xjc==k),1);
            % estimate prob(variable j = k | class i)
            params(j).cprobs(k,i)=(njc+0.5)/(ni+1);
        end
    end
end

% Estimate the probabilities p(x_j = value_k)
for j=1:d
    for k=1:2
        % estimate prob(variable x_j = k)
        params(j).mprobs(k)=(params(j).cprobs(k,1).*params(1).classprobs(1))+(params(j).cprobs(k,2).*params(1).classprobs(2));
    end
end
```

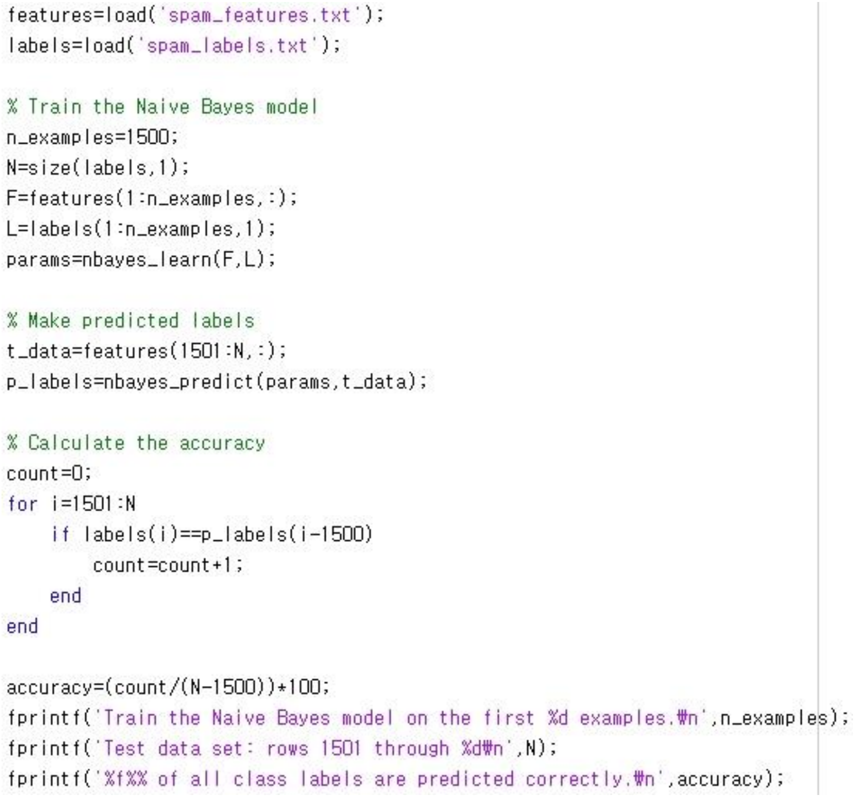
**-nbayes_predict.m**



```
function [predictions]=nbayes_predict(params,testdata)
nclasses=2;
ntest=size(testdata,1);
d=size(testdata,2);
predictions=zeros(ntest,1);

% for each of the test data points
for m=1:ntest
    x=testdata(m,:);
    % for each class value calculate log[ p(x|c) p(c) ]
    P=zeros(1,nclasses);
    for classi=1:nclasses
        P(classi)=log(params(1).classprobs(classi));
        for varj=1:d
            P(classi)=P(classi)+log(params(varj).cprobs(x(1,varj),classi));
        end
    end
    % select the maximum value over all classes as the predicted class
    [mValue,mIndex]=max(P);
    % store the class prediction for each test data point
    predictions(m)=mIndex;
end
```

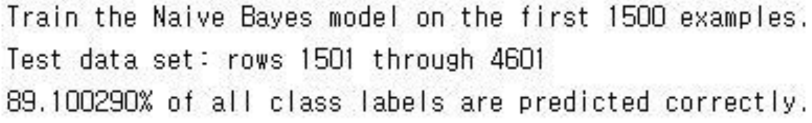
## ◆ Question 1

**-Code**



```
features=load('spam_features.txt');
labels=load('spam_labels.txt');

% Train the Naive Bayes model
n_examples=1500;
N=size(labels,1);
F=features(1:n_examples,:);
L=labels(1:n_examples,1);
params=nbayes_learn(F,L);

% Make predicted labels
t_data=features(1501:N,:);
p_labels=nbayes_predict(params,t_data);

% Calculate the accuracy
count=0;
for i=1501:N
    if labels(i)==p_labels(i-1500)
        count=count+1;
    end
end

accuracy=(count/(N-1500))*100;
fprintf('Train the Naive Bayes model on the first %d examples.₩n',n_examples);
fprintf('Test data set: rows 1501 through %d₩n',N);
fprintf('%f%% of all class labels are predicted correctly.₩n',accuracy);
```

**-Result**

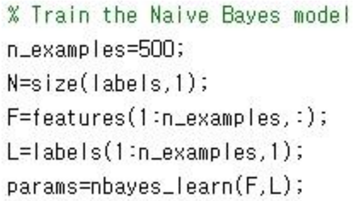


```
Train the Naive Bayes model on the first 1500 examples.
Test data set: rows 1501 through 4601
89.100290% of all class labels are predicted correctly.
```

**Accuracy: 89.100290%**

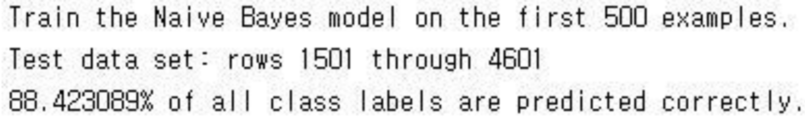
## ◆ Question 2

**-Code**



```
% Train the Naive Bayes model
n_examples=500;
N=size(labels,1);
F=features(1:n_examples,:);
L=labels(1:n_examples,1);
params=nbayes_learn(F,L);
```

**-Result**

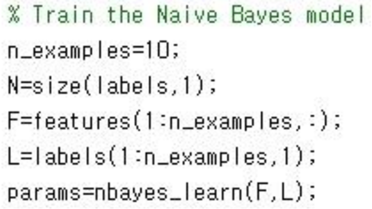


```
Train the Naive Bayes model on the first 500 examples.
Test data set: rows 1501 through 4601
88.423089% of all class labels are predicted correctly.
```

**Accuracy: 88.423089%**

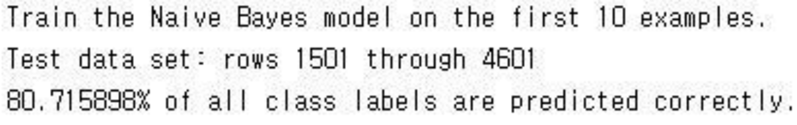
## ◆ Question 3

**-Code**



```
% Train the Naive Bayes model
n_examples=10;
N=size(labels,1);
F=features(1:n_examples,:);
L=labels(1:n_examples,1);
params=nbayes_learn(F,L);
```

**-Result**



```
Train the Naive Bayes model on the first 10 examples.
Test data set: rows 1501 through 4601
80.715898% of all class labels are predicted correctly.
```

**Accuracy: 80.715898%**

## ◆ Discussion 1~3

- As the number of the examples used for training the Naive Bayes model decreases, the accuracy decreases either.

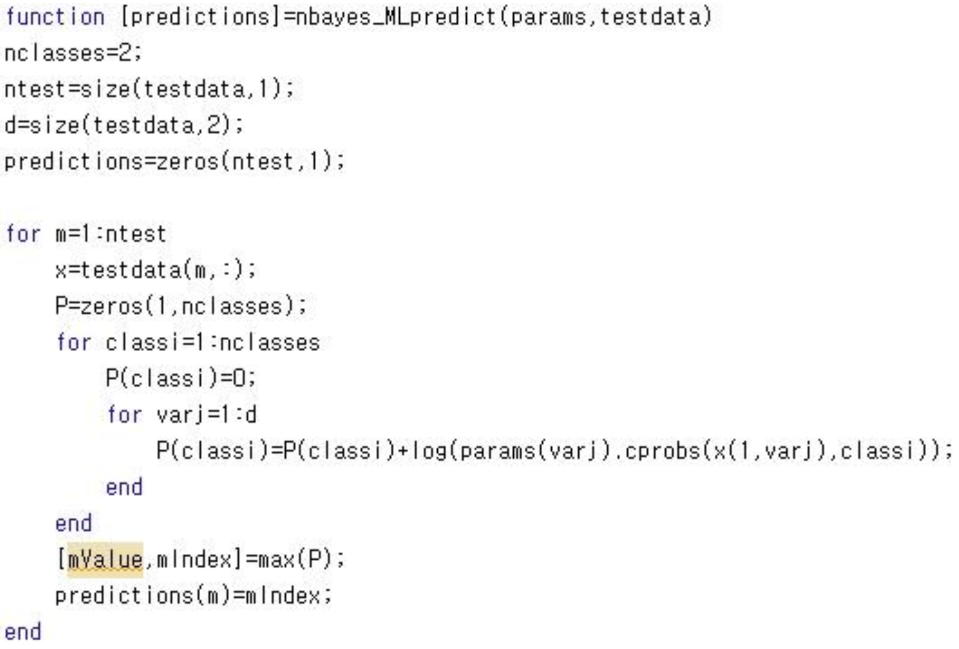
- Based on WLLN of relative frequency, because the sample mean gets close to $E[X_j]$ and the margin of error decreases as the number of samples increases, the accuracy is proportional to the number of samples.

## ◆ Basic Idea for Question 4-1~3

**-ML:** $argmax_c = P\left(X_1, X_2, \cdots, X_d \mid C = c_i\right)$

## ◆ Code used for Question 4-1~3

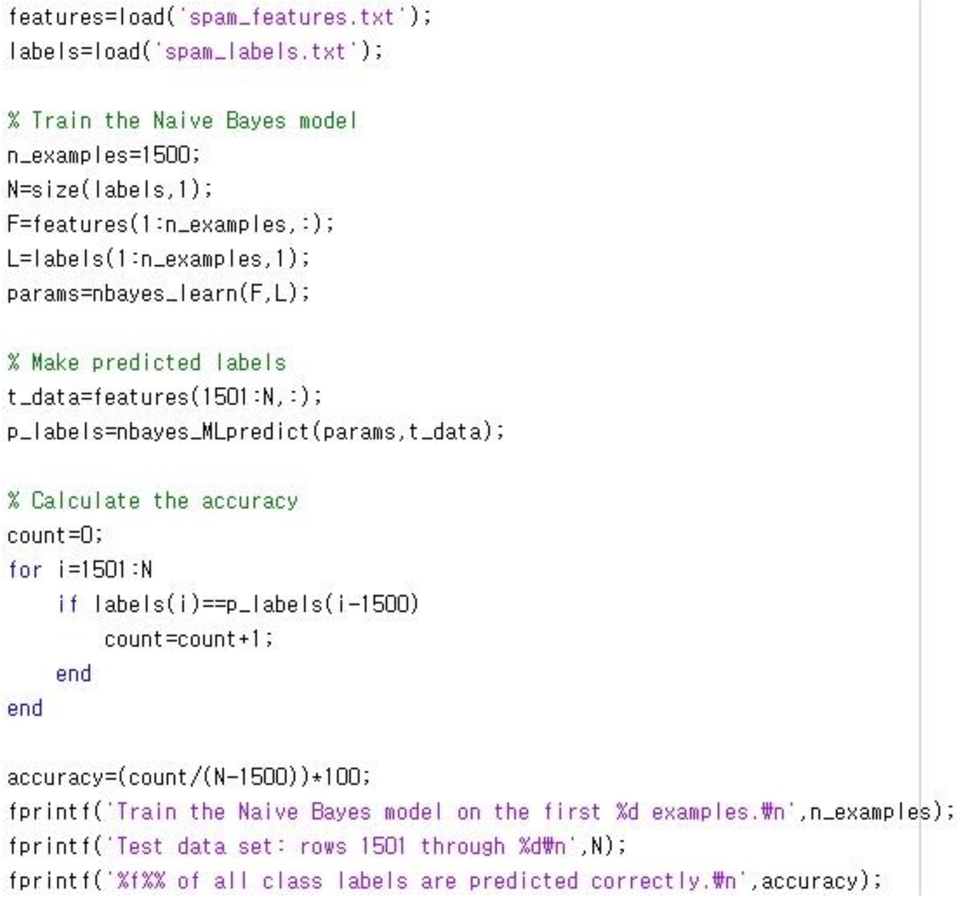
**-nbayes_MLpredict.m**



```
function [predictions]=nbayes_MLpredict(params,testdata)
nclasses=2;
ntest=size(testdata,1);
d=size(testdata,2);
predictions=zeros(ntest,1);

for m=1:ntest
    x=testdata(m,:);
    P=zeros(1,nclasses);
    for classi=1:nclasses
        P(classi)=0;
        for varj=1:d
            P(classi)=P(classi)+log(params(varj).cprobs(x(1,varj),classi));
        end
    end
    [mValue,mIndex]=max(P);
    predictions(m)=mIndex;
end
```

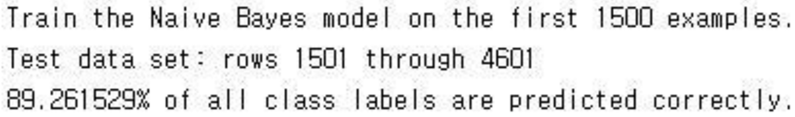
## ◆ Question 4-1

**-Code**



```
features=load('spam_features.txt');
labels=load('spam_labels.txt');

% Train the Naive Bayes model
n_examples=1500;
N=size(labels,1);
F=features(1:n_examples,:);
L=labels(1:n_examples,1);
params=nbayes_learn(F,L);

% Make predicted labels
t_data=features(1501:N,:);
p_labels=nbayes_MLpredict(params,t_data);

% Calculate the accuracy
count=0;
for i=1501:N
    if labels(i)==p_labels(i-1500)
        count=count+1;
    end
end

accuracy=(count/(N-1500))*100;
fprintf('Train the Naive Bayes model on the first %d examples.₩n',n_examples);
fprintf('Test data set: rows 1501 through %d₩n',N);
fprintf('%f%% of all class labels are predicted correctly.₩n',accuracy);
```

**-Result**

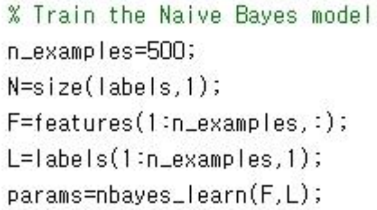


```
Train the Naive Bayes model on the first 1500 examples.
Test data set: rows 1501 through 4601
89.261529% of all class labels are predicted correctly.
```

**Accuracy: 89.261529%**

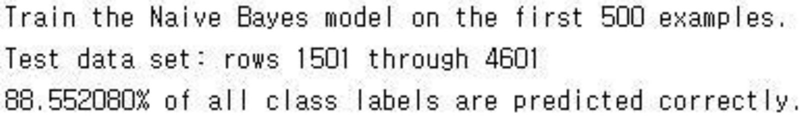
## ◆ Question 4-2

**-Code**



```
% Train the Naive Bayes model
n_examples=500;
N=size(labels,1);
F=features(1:n_examples,:);
L=labels(1:n_examples,1);
params=nbayes_learn(F,L);
```

**-Result**

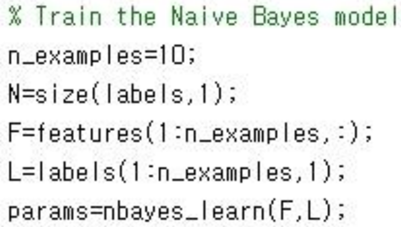


```
Train the Naive Bayes model on the first 500 examples.
Test data set: rows 1501 through 4601
88.552080% of all class labels are predicted correctly.
```

**Accuracy: 88.552080%**

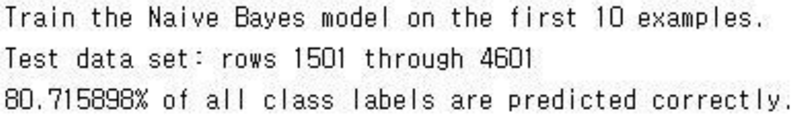
## ◆ Question 4-3

**-Code**



```
% Train the Naive Bayes model
n_examples=10;
N=size(labels,1);
F=features(1:n_examples,:);
L=labels(1:n_examples,1);
params=nbayes_learn(F,L);
```

**-Result**



```
Train the Naive Bayes model on the first 10 examples.
Test data set: rows 1501 through 4601
80.715898% of all class labels are predicted correctly.
```

**Accuracy: 80.715898%**

## ◆ Discussion 4-1~3

- There's almost no difference in accuracy between MAP(Maximum a Priori) approach and ML(Maximum Likelihood) approach.

- The reason is that in the MAP approach case, the $argmax_c$ is determined by $P\left( X_1, X_2, \cdots, X_d \mid C=c_i \right) P\left( C=c_i \right)$, and in the ML approach case, $argmax_c$ is determined by $P\left( X_1, X_2, \cdots, X_d \mid C=c_i \right)$

- Since the only difference between the two methods is $P\left( C=c_i \right)$, it is likely that the accuracy is almost the same. (We have gone through the same smoothing process in nbayes_learn.m as the MAP method.)

## ◆ Basic Ideas for Question 5-1,3

**-params(j).E(i)** = $E\left[X_j \mid C=c_i\right]$

**-params(j).V(i)** = $Var\left[X_j \mid C=c_i\right]$

**-params(j).S(i)** = $\sigma\left[X_j \mid C=c_i\right]$ (+0.00001 in case of value=0)

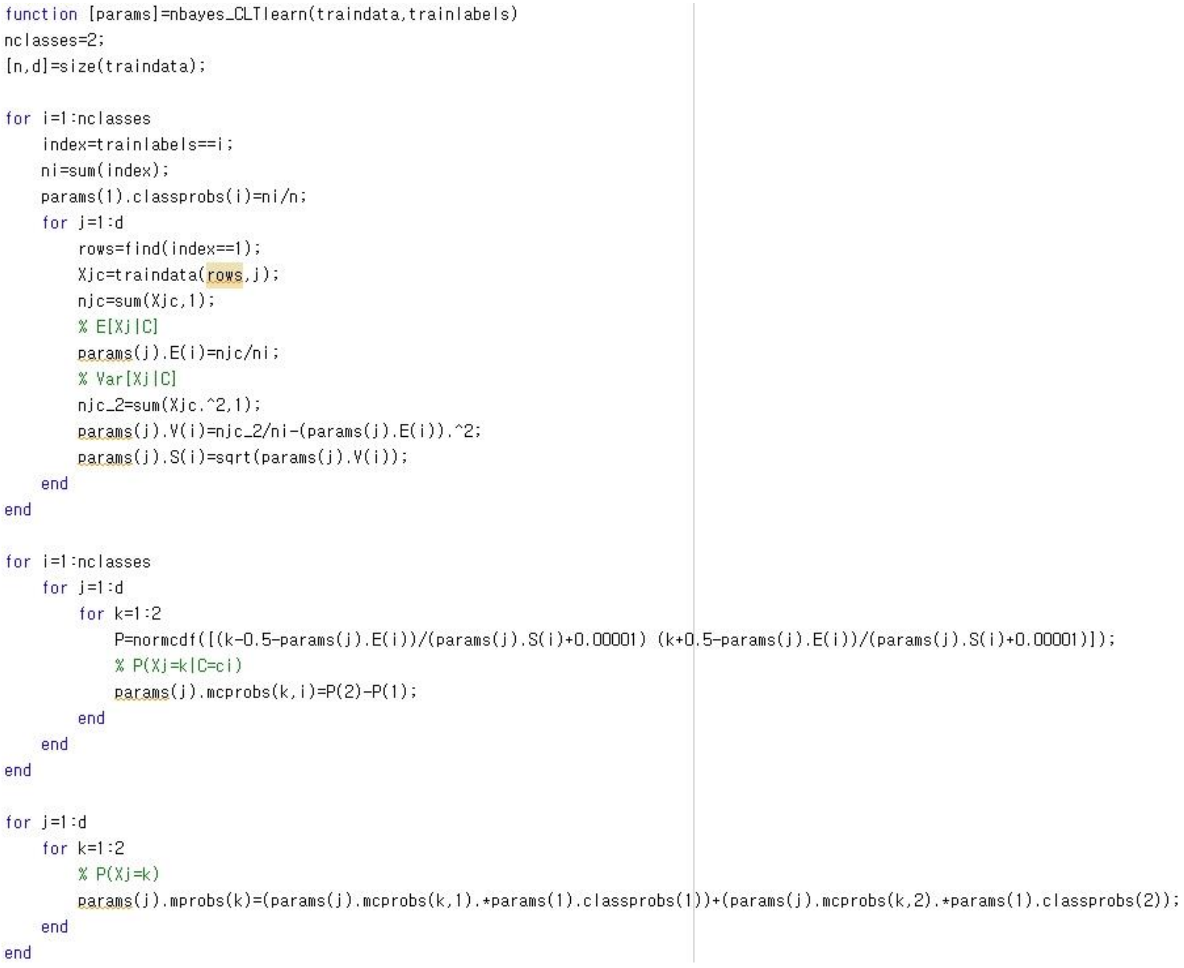
**-params(j).mcprobs(k,i)** = $P\left[k-0.5<X_j<k+0.5 \mid C=c_i\right]=\Phi\left(\frac{k+0.5-\mu_{x_i|c_i}}{\sigma_{x_i|c_i}+0.00001}\right)-\Phi\left(\frac{k-0.5-\mu_{x_i|c_i}}{\sigma_{x_i|c_i}+0.00001}\right)$

**-params(j).mprobs(k)** = $P\left[k-0.5<X_j<k+0.5\right]$

= $P\left[k-0.5<X_j<k+0.5 \mid C=c_1\right]P\left(C=c_1\right)+P\left[k-0.5<X_j<k+0.5 \mid C=c_2\right]P\left(C=c_2\right)$

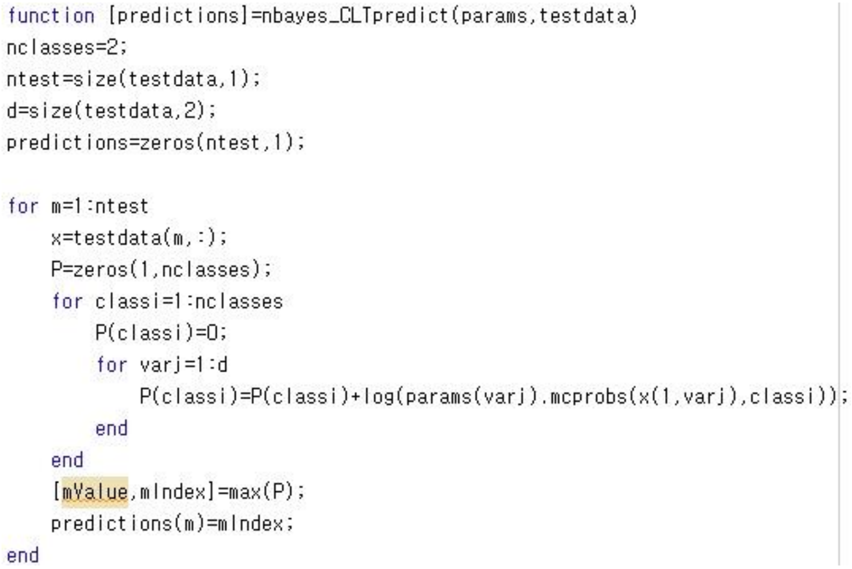
## ◆ Codes used for Question 5-1,3

**-nbayes_CLTlearn.m**



```
function [params]=nbayes_CLTlearn(traindata,trainlabels)
nclasses=2;
[n,d]=size(traindata);

for i=1:nclasses
    index=trainlabels==i;
    ni=sum(index);
    params(1).classprobs(i)=ni/n;
    for j=1:d
        rows=find(index==1);
        Xjc=traindata(rows,j);
        njc=sum(Xjc,1);
        % E[Xj|C]
        params(j).E(i)=njc/ni;
        % Var[Xj|C]
        njc_2=sum(Xjc.^2,1);
        params(j).V(i)=njc_2/ni-(params(j).E(i)).^2;
        params(j).S(i)=sqrt(params(j).V(i));
    end
end

for i=1:nclasses
    for j=1:d
        for k=1:2
            P=normcdf([(k-0.5-params(j).E(i))/(params(j).S(i)+0.00001) (k+0.5-params(j).E(i))/(params(j).S(i)+0.00001)]);
            % P(Xj=k|C=ci)
            params(j).mcprobs(k,i)=P(2)-P(1);
        end
    end
end

for j=1:d
    for k=1:2
        % P(Xj=k)
        params(j).mprobs(k)=(params(j).mcprobs(k,1).*params(1).classprobs(1))+(params(j).mcprobs(k,2).*params(1).classprobs(2));
    end
end
```

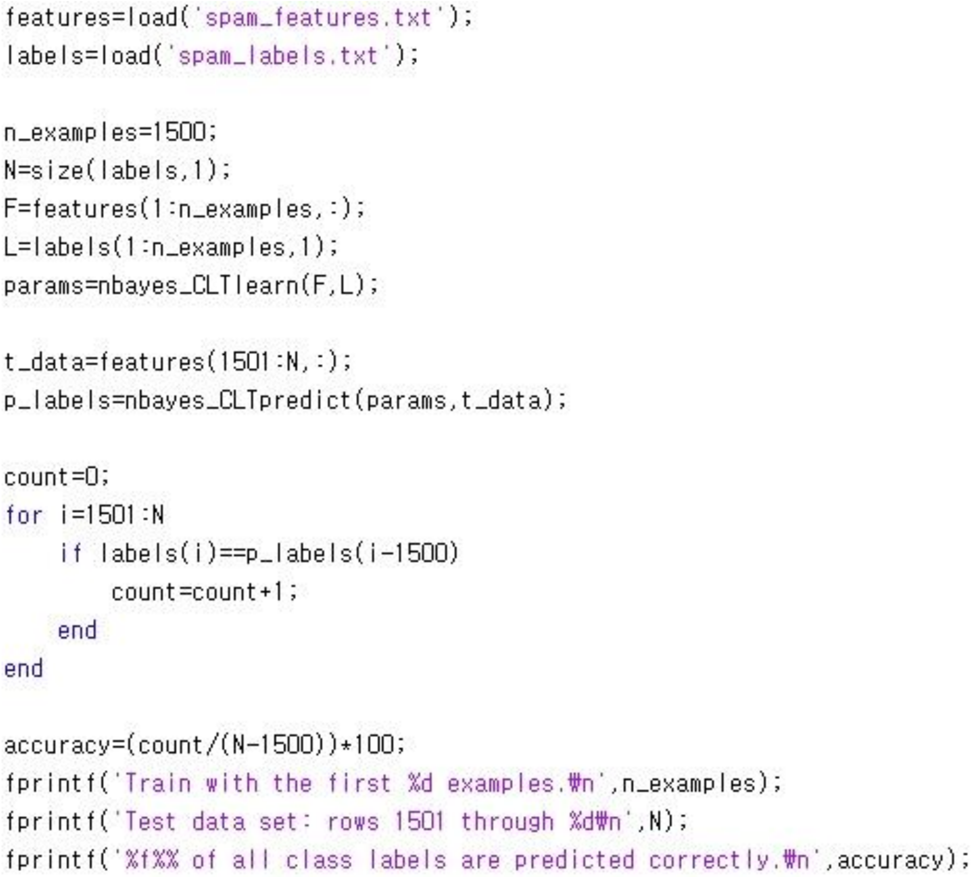
**-nbayes_CLTpredict.m**



```
function [predictions]=nbayes_CLTpredict(params,testdata)
nclasses=2;
ntest=size(testdata,1);
d=size(testdata,2);
predictions=zeros(ntest,1);

for m=1:ntest
    x=testdata(m,:);
    P=zeros(1,nclasses);
    for classi=1:nclasses
        P(classi)=0;
        for varj=1:d
            P(classi)=P(classi)+log(params(varj).mcprobs(x(1,varj),classi));
        end
    end
    [mValue,mIndex]=max(P);
    predictions(m)=mIndex;
end
```

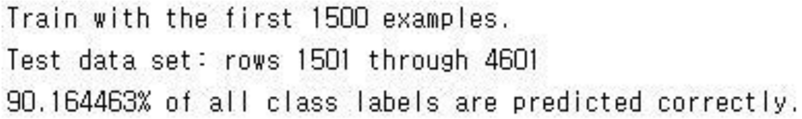
## ◆ Question 5-1

**-Code**



```
features=load('spam_features.txt');
labels=load('spam_labels.txt');

n_examples=1500;
N=size(labels,1);
F=features(1:n_examples,:);
L=labels(1:n_examples,1);
params=nbayes_CLTlearn(F,L);

t_data=features(1501:N,:);
p_labels=nbayes_CLTpredict(params,t_data);

count=0;
for i=1501:N
     if labels(i)==p_labels(i-1500)
         count=count+1;
     end
end

accuracy=(count/(N-1500))*100;
fprintf('Train with the first %d examples.₩n',n_examples);
fprintf('Test data set: rows 1501 through %d₩n',N);
fprintf('%f%% of all class labels are predicted correctly.₩n',accuracy);
```

**-Result**

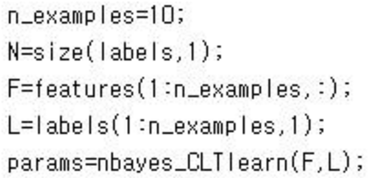


```
Train with the first 1500 examples.
Test data set: rows 1501 through 4601
90.164463% of all class labels are predicted correctly.
```

**Accuracy: 90.164463%**

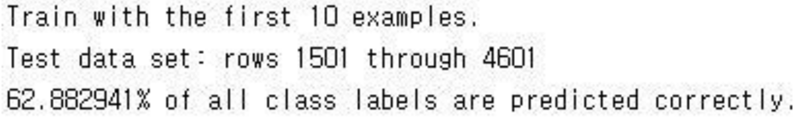
## ◆ Question 5-3

**-Code**



```
n_examples=10;
N=size(labels,1);
F=features(1:n_examples,:);
L=labels(1:n_examples,1);
params=nbayes_CLTlearn(F,L);
```

**-Result**



```
Train with the first 10 examples.
Test data set: rows 1501 through 4601
62.882941% of all class labels are predicted correctly.
```

**Accuracy: 62.882941%**

## ◆ Discussion 5-1,3

- In case of the model trained by 1500 examples, the accuracy is slightly higher than that driven by the method of the Naive Bayes model.

- However, if the number of examples is not enough, we can see that the accuracy becomes significantly low compared to the Naive Bayes model.

- The probabilities that we obtained from Gaussian pdf using CLT are not exactly accurate because there's a range of 1 and the number of samples is not enough.

- Since the probabilities are inaccurate, it is unlikely that the match between the predicted label and the original label would be good.

# Q3: Simulation using Random Variables (B 100%)

## Part A : Monte Carlo Integration

**(a)**

By WLLN, we know that

$$\widehat{\Theta}_n = \frac{1}{n}\sum_{i=1}^{n} w(X_i) \overset{p}{\rightarrow} E\big(w(X_1)\big)$$

So $\widehat{\Theta}_n$ is sample mean of w(X) which random variable $X \sim \mathrm{Uniform}(A)$

And we can say $\widehat{\Theta}_n$ is sample mean of $w(X_1)$ .

And by Theorem 10.1, we know that variance of sample mean is equal to variance of random variable divided by n.

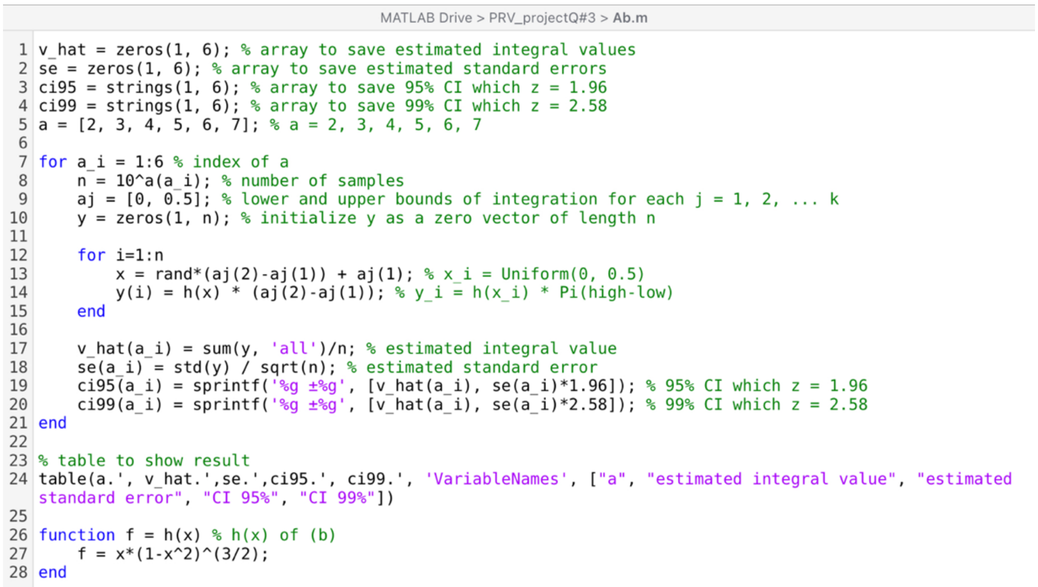
$$\mathrm{Var}[M_n(X)] = \frac{\mathrm{Var}[X]}{n}$$

Using these, we can say

$$se_n = \sqrt{Var\left(\widehat{\Theta}_n\right)} = \sqrt{\frac{Var\big(w(X)\big)}{n}} = \sqrt{\frac{Var\big(w(X_1)\big)}{n}} = \frac{1}{\sqrt{n}}\sqrt{Var\big(w(X_1)\big)}$$

## (b)

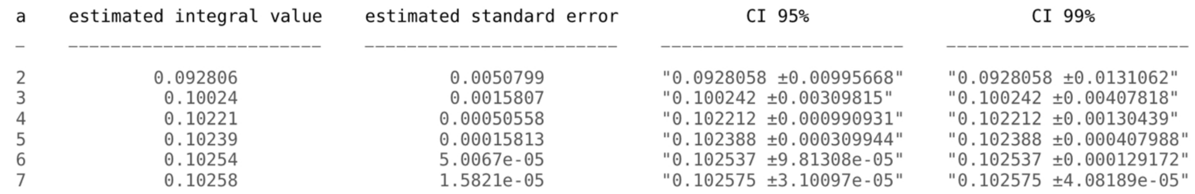
## -Code



MATLAB Drive > PRV_projectQ#3 > Ab.m

```
v_hat = zeros(1, 6); % array to save estimated integral values
se = zeros(1, 6); % array to save estimated standard errors
ci95 = strings(1, 6); % array to save 95% CI which z = 1.96
ci99 = strings(1, 6); % array to save 99% CI which z = 2.58
a = [2, 3, 4, 5, 6, 7]; % a = 2, 3, 4, 5, 6, 7

for a_i = 1:6 % index of a
    n = 10^a(a_i); % number of samples
    aj = [0, 0.5]; % lower and upper bounds of integration for each j = 1, 2, ... k
    y = zeros(1, n); % initialize y as a zero vector of length n

    for i=1:n
        x = rand*(aj(2)-aj(1)) + aj(1); % x_i = Uniform(0, 0.5)
        y(i) = h(x) * (aj(2)-aj(1)); % y_i = h(x_i) * Pi(high-low)
    end

    v_hat(a_i) = sum(y, 'all')/n; % estimated integral value
    se(a_i) = std(y) / sqrt(n); % estimated standard error
    ci95(a_i) = sprintf('%g ±%g', [v_hat(a_i), se(a_i)*1.96]); % 95% CI which z = 1.96
    ci99(a_i) = sprintf('%g ±%g', [v_hat(a_i), se(a_i)*2.58]); % 99% CI which z = 2.58
end

% table to show result
table(a.', v_hat.',se.',ci95.', ci99.', 'VariableNames', ["a", "estimated integral value", "estimated standard error", "CI 95%", "CI 99%"])

function f = h(x) % h(x) of (b)
    f = x*(1-x^2)^(3/2);
end
```

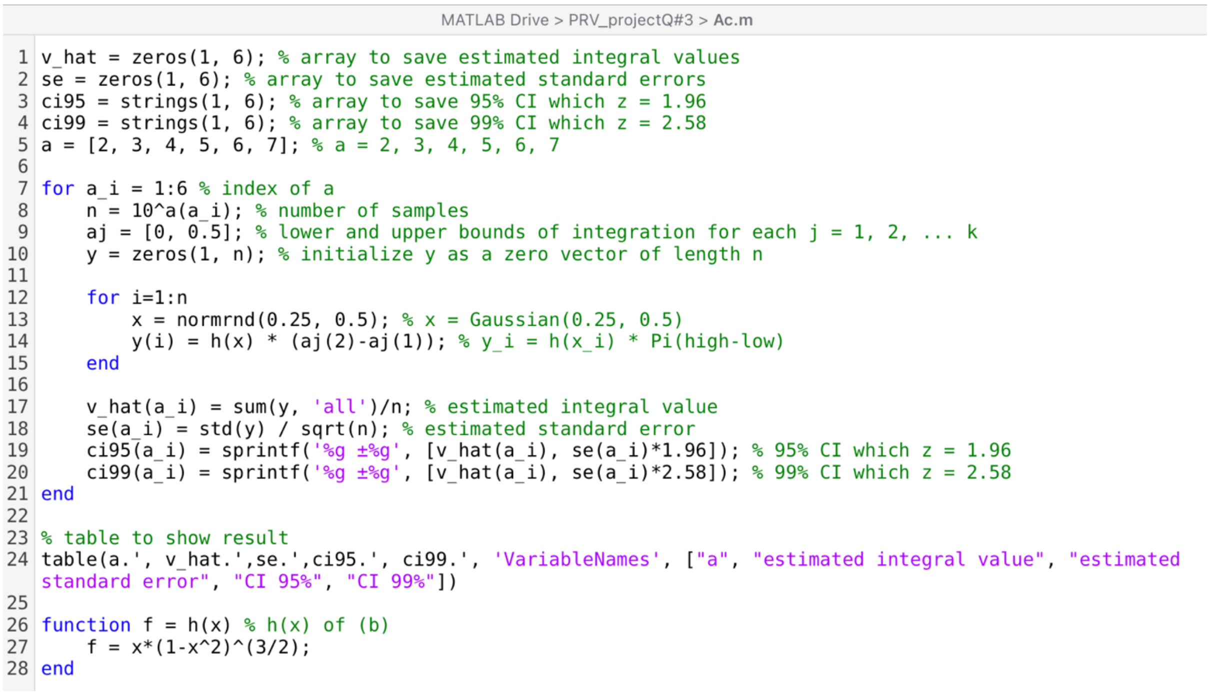
## -Result



```
a    estimated integral value    estimated standard error             CI 95%                          CI 99%
_    ________________________    ________________________    ________________________    ________________________

2            0.092806                   0.0050799            "0.0928058 ±0.00995668"     "0.0928058 ±0.0131062"
3             0.10024                   0.0015807            "0.100242 ±0.00309815"      "0.100242 ±0.00407818"
4             0.10221                  0.00050558            "0.102212 ±0.000990931"     "0.102212 ±0.00130439"
5             0.10239                  0.00015813            "0.102388 ±0.000309944"     "0.102388 ±0.000407988"
6             0.10254                  5.0067e-05            "0.102537 ±9.81308e-05"     "0.102537 ±0.000129172"
7             0.10258                  1.5821e-05            "0.102575 ±3.10097e-05"     "0.102575 ±4.08189e-05"
```

### (c)

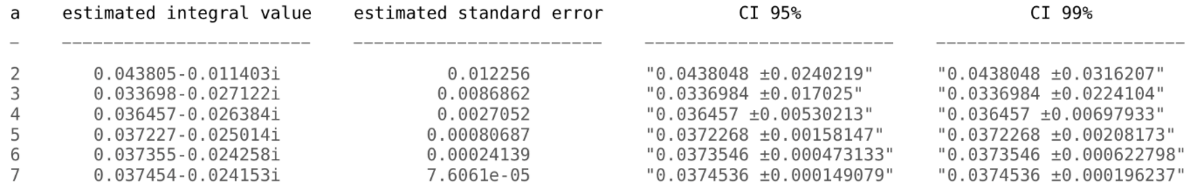
### -Code



MATLAB Drive > PRV_projectQ#3 > Ac.m

```
v_hat = zeros(1, 6); % array to save estimated integral values
se = zeros(1, 6); % array to save estimated standard errors
ci95 = strings(1, 6); % array to save 95% CI which z = 1.96
ci99 = strings(1, 6); % array to save 99% CI which z = 2.58
a = [2, 3, 4, 5, 6, 7]; % a = 2, 3, 4, 5, 6, 7

for a_i = 1:6 % index of a
    n = 10^a(a_i); % number of samples
    aj = [0, 0.5]; % lower and upper bounds of integration for each j = 1, 2, ... k
    y = zeros(1, n); % initialize y as a zero vector of length n

    for i=1:n
        x = normrnd(0.25, 0.5); % x = Gaussian(0.25, 0.5)
        y(i) = h(x) * (aj(2)-aj(1)); % y_i = h(x_i) * Pi(high-low)
    end

    v_hat(a_i) = sum(y, 'all')/n; % estimated integral value
    se(a_i) = std(y) / sqrt(n); % estimated standard error
    ci95(a_i) = sprintf('%g ±%g', [v_hat(a_i), se(a_i)*1.96]); % 95% CI which z = 1.96
    ci99(a_i) = sprintf('%g ±%g', [v_hat(a_i), se(a_i)*2.58]); % 99% CI which z = 2.58
end

% table to show result
table(a.', v_hat.',se.',ci95.', ci99.', 'VariableNames', ["a", "estimated integral value", "estimated
standard error", "CI 95%", "CI 99%"])

function f = h(x) % h(x) of (b)
    f = x*(1-x^2)^(3/2);
end
```

### -Result



```
a    estimated integral value    estimated standard error            CI 95%                         CI 99%
_    ________________________    ________________________    ________________________    ________________________

2       0.043805-0.011403i               0.012256            "0.0438048 ±0.0240219"      "0.0438048 ±0.0316207"
3       0.033698-0.027122i              0.0086862            "0.0336984 ±0.017025"       "0.0336984 ±0.0224104"
4       0.036457-0.026384i              0.0027052            "0.036457 ±0.00530213"      "0.036457 ±0.00697933"
5       0.037227-0.025014i             0.00080687            "0.0372268 ±0.00158147"     "0.0372268 ±0.00208173"
6       0.037355-0.024258i             0.00024139            "0.0373546 ±0.000473133"    "0.0373546 ±0.000622798"
7       0.037454-0.024153i             7.6061e-05            "0.0374536 ±0.000149079"    "0.0374536 ±0.000196237"
```

BMC with Gaussian estimates totally different values.

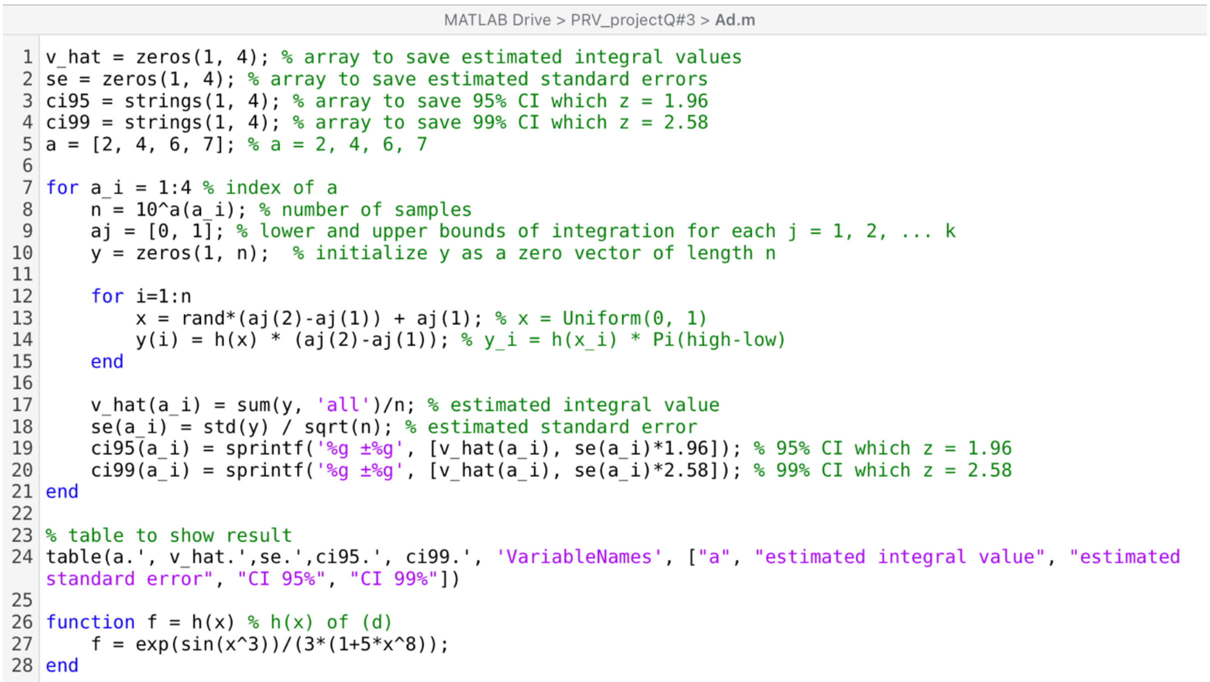
There are two reasons why Gaussian is not applicable in BMC.

First, Gaussian has no boundary. So if we apply Gaussian in BMC, it calculates values which is out of range. Then BMC get wrong values.

Second, Gaussian is concentrated at the center, but integral must be performed with a uniform density to get an accurate value. So Gaussian is not applicable in BMC.

## (d)

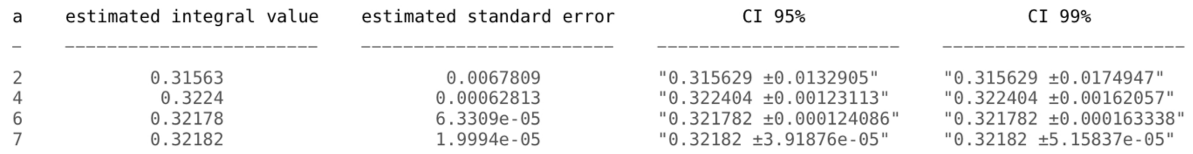
## -Code



MATLAB Drive > PRV_projectQ#3 > Ad.m

```matlab
v_hat = zeros(1, 4); % array to save estimated integral values
se = zeros(1, 4); % array to save estimated standard errors
ci95 = strings(1, 4); % array to save 95% CI which z = 1.96
ci99 = strings(1, 4); % array to save 99% CI which z = 2.58
a = [2, 4, 6, 7]; % a = 2, 4, 6, 7

for a_i = 1:4 % index of a
    n = 10^a(a_i); % number of samples
    aj = [0, 1]; % lower and upper bounds of integration for each j = 1, 2, ... k
    y = zeros(1, n);  % initialize y as a zero vector of length n

    for i=1:n
        x = rand*(aj(2)-aj(1)) + aj(1); % x = Uniform(0, 1)
        y(i) = h(x) * (aj(2)-aj(1)); % y_i = h(x_i) * Pi(high-low)
    end

    v_hat(a_i) = sum(y, 'all')/n; % estimated integral value
    se(a_i) = std(y) / sqrt(n); % estimated standard error
    ci95(a_i) = sprintf('%g ±%g', [v_hat(a_i), se(a_i)*1.96]); % 95% CI which z = 1.96
    ci99(a_i) = sprintf('%g ±%g', [v_hat(a_i), se(a_i)*2.58]); % 99% CI which z = 2.58
end

% table to show result
table(a.', v_hat.',se.',ci95.', ci99.', 'VariableNames', ["a", "estimated integral value", "estimated standard error", "CI 95%", "CI 99%"])

function f = h(x) % h(x) of (d)
    f = exp(sin(x^3))/(3*(1+5*x^8));
end
```

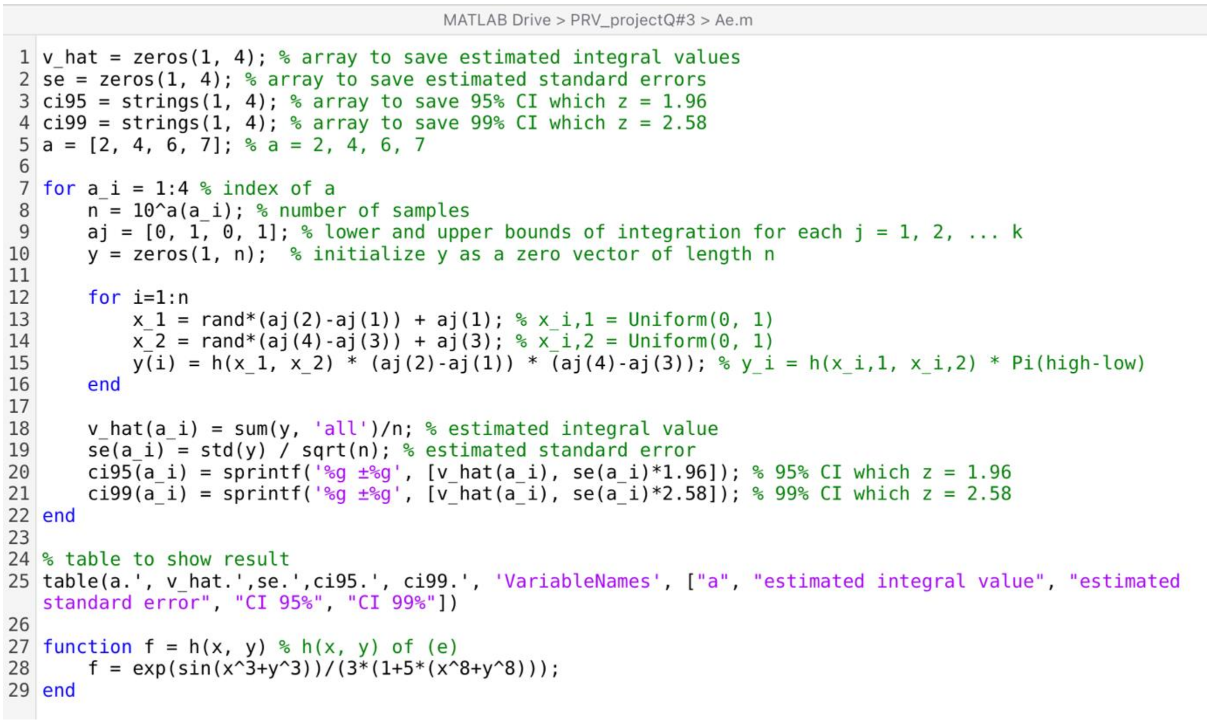
## -Result



```
a    estimated integral value    estimated standard error            CI 95%                      CI 99%
_    ________________________    ________________________    ________________________    ________________________

2            0.31563                     0.0067809           "0.315629 ±0.0132905"       "0.315629 ±0.0174947"
4             0.3224                    0.00062813           "0.322404 ±0.00123113"      "0.322404 ±0.00162057"
6            0.32178                    6.3309e-05           "0.321782 ±0.000124086"     "0.321782 ±0.000163338"
7            0.32182                    1.9994e-05           "0.32182 ±3.91876e-05"      "0.32182 ±5.15837e-05"
```

**(e)**

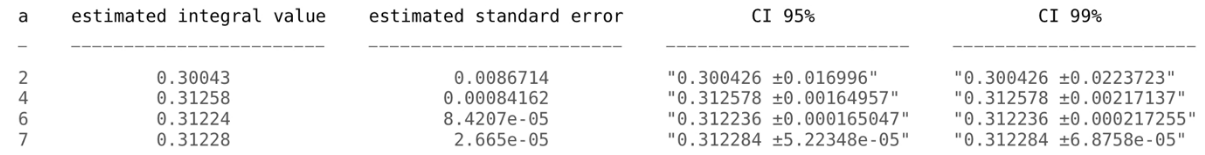
## -Code



MATLAB Drive > PRV_projectQ#3 > Ae.m

```
v_hat = zeros(1, 4); % array to save estimated integral values
se = zeros(1, 4); % array to save estimated standard errors
ci95 = strings(1, 4); % array to save 95% CI which z = 1.96
ci99 = strings(1, 4); % array to save 99% CI which z = 2.58
a = [2, 4, 6, 7]; % a = 2, 4, 6, 7

for a_i = 1:4 % index of a
    n = 10^a(a_i); % number of samples
    aj = [0, 1, 0, 1]; % lower and upper bounds of integration for each j = 1, 2, ... k
    y = zeros(1, n);  % initialize y as a zero vector of length n

    for i=1:n
        x_1 = rand*(aj(2)-aj(1)) + aj(1); % x_i,1 = Uniform(0, 1)
        x_2 = rand*(aj(4)-aj(3)) + aj(3); % x_i,2 = Uniform(0, 1)
        y(i) = h(x_1, x_2) * (aj(2)-aj(1)) * (aj(4)-aj(3)); % y_i = h(x_i,1, x_i,2) * Pi(high-low)
    end

    v_hat(a_i) = sum(y, 'all')/n; % estimated integral value
    se(a_i) = std(y) / sqrt(n); % estimated standard error
    ci95(a_i) = sprintf('%g ±%g', [v_hat(a_i), se(a_i)*1.96]); % 95% CI which z = 1.96
    ci99(a_i) = sprintf('%g ±%g', [v_hat(a_i), se(a_i)*2.58]); % 99% CI which z = 2.58
end

% table to show result
table(a.', v_hat.',se.',ci95.', ci99.', 'VariableNames', ["a", "estimated integral value", "estimated standard error", "CI 95%", "CI 99%"])

function f = h(x, y) % h(x, y) of (e)
    f = exp(sin(x^3+y^3))/(3*(1+5*(x^8+y^8)));
end
```

## -Result



```
a    estimated integral value    estimated standard error            CI 95%                        CI 99%
_    ________________________    ________________________    ________________________    ________________________

2             0.30043                    0.0086714           "0.300426 ±0.016996"        "0.300426 ±0.0223723"
4             0.31258                   0.00084162           "0.312578 ±0.00164957"      "0.312578 ±0.00217137"
6             0.31224                   8.4207e-05           "0.312236 ±0.000165047"     "0.312236 ±0.000217255"
7             0.31228                    2.665e-05           "0.312284 ±5.22348e-05"     "0.312284 ±6.8758e-05"
```

## Part B : The Monte Hall Problem

**(a)**

Let's derive expected payoff for the policy with $P = a$.

We choose one door out of three.

There are three doors and olny one for the car.

So, there are two case.

Case 1. Choose a door that hides the car = 1/3

Case 2. Choose a door that hides the goat = 2/3

Then, host opens the door that hides a goat, which you didn't choose.

For case 1, you'll get car if you don't change you choice.

So payoff for case 1 is, $\frac{1}{3} * (1 - a)$

For case 2, you'll get car if you change your choice

So payoff for case 2 is, $\frac{2}{3} * a$

Now, we can derive total payoff by adding payoff of case 1 and case 2.

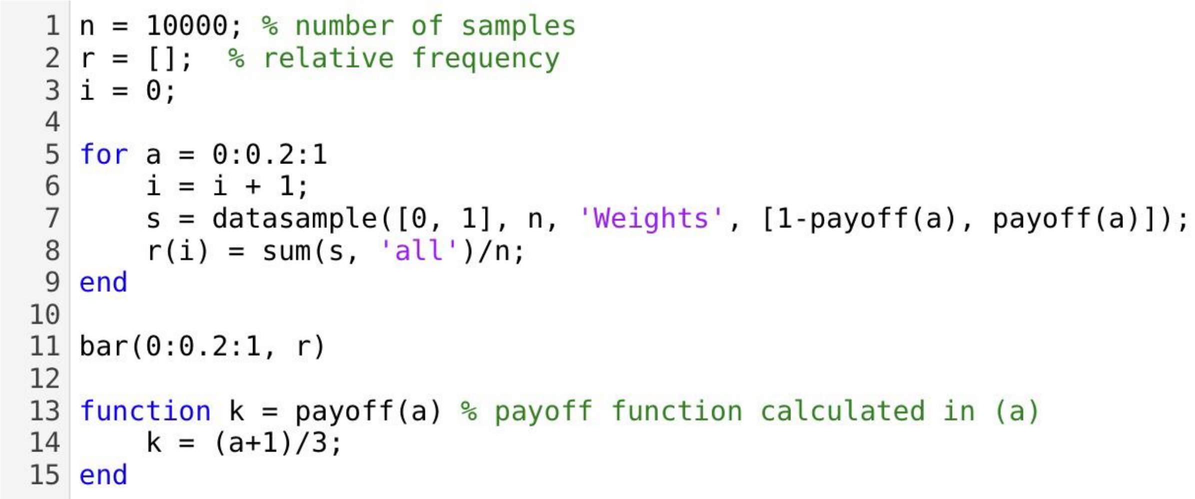
$$\text{payoff} = \frac{1}{3} * (1 - a) + \frac{2}{3} * a = \frac{1}{3} * (a + 1)$$

Where a is $0 \leq a \leq 1$.

So we can maximize payoff when $a = 1$

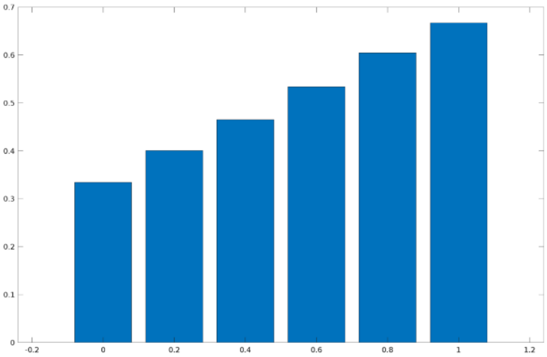
**(b)**

**-Code**



```
n = 10000; % number of samples
r = [];  % relative frequency
i = 0;

for a = 0:0.2:1
    i = i + 1;
    s = datasample([0, 1], n, 'Weights', [1-payoff(a), payoff(a)]);
    r(i) = sum(s, 'all')/n;
end

bar(0:0.2:1, r)

function k = payoff(a) % payoff function calculated in (a)
    k = (a+1)/3;
end
```

**-Result**



## Discussion for Part A

Basic Monte Carlo integration gives us easier way to calculate integral. In (a) we showed equation is true by using theorem 10.1. In (b), (d), (e), we show that the confidence interval decreases as the number of samples increases. In (c), we showed why we can’t use Gaussian random variable by two reasons.

## Discussion for Part B

Monte Hall problem seems totally random. In (a) we showed what value to a maximize payoff. In (b), we showed change of payoff according to a.

# Conclusion (A 34%, B 33%, C 33%)

In Question 1, we used 3 different random variables to design a m-bits of quantizer based on conditional probability model by event. We first used a Laplace random variable. We could see how bits or iterations affect the value of SQNR. We calculated SQNR with a Uniform random variable and Gaussian random variable, too. But in the case of Gaussian, there was an invalid value for SQNR. We can calculate many parameters such as distortion or SQNR with different random variables at last.

In Question 2, we can see that no matter what methods we use for designing a spam mail detector, the accuracy of the classifier increases as the number of samples used for learning increases. Next, the results show that there's almost no difference between the MAP and ML learning methods, and that the CLT method is a little bit better than these two if there are lots of samples. Overall, we can conclude that the spam mail classifier has been made successfully by use of Naive Bayes Programming.

In Question 3, we applied Basic Monte Carlo integration and Monty Hall problem to matlab. In Basic Monte Carlo integration, confident interval became smaller as number of samples increases. So we need a sufficient number of samples to get more accurate result. In Monty Hall problem, we showed relation between payoff and a(probability to switch). If a is increases, payoff increases up to 2/3. So we could maximize payoff by a=1(switch always).

| **Contribution** | |
|---|---|
| **Introduction** | **A 33%, B 34%, C 33%** |
| **Question 1** | **A 100%** |
| **Question 2** | **C 100%** |
| **Question 3** | **B 100%** |
| **Discussion** | **A 33%, B 33%, C 34%** |
| **Conclusion** | **A 34%, B 33%, C 33%** |

# <Appendix>

## Q1

### <Part A>

```
clc
clear
close all;

% problem (c)

% find x3
fun_x3=@(x) 2-x/(exp(x)-1)-x;
x3=fzero(fun_x3,1);

% find y3
y3=1-x3/(exp(x3)-1);

% find y4
y4=1+x3;

% problem (d)
fun_d1=@(x) 2.*((x-y3).^2).*(1/2).*exp(-x);
d1=integral(fun_d1,0,x3);
fun_d2=@(x) 2.*((x-y4).^2).*(1/2).*exp(-x);
d2=integral(fun_d2,x3,inf);
d=d1+d2;

% problem (e)
sqnr=2/d;
```

## <Part B_(c)>

```
clc
clear
close all;

x=zeros(51,9); % x(1,1)=x0, x0(1,2)=x1, ... , x0(8)=x7, x0(9)=x8=99999
y=zeros(51,8); % y(1,1)=y1, y0(1,2)=y2, ...
p=zeros(51,8); % p0(1)=p1, p0(2)=p2, ...
sum=zeros(51,1);
sqnr=zeros(51,1);

% iteration=0
for i=2:8 % x
    x(1,i)=(i-1)*2/8;
end

x(1,1)=0;
x(1,9)=99999;

for i=1:8 % y     y(1,i)=((x(1,i)+1)*exp(-x(1,i))-(x(1,i+1)+1)*exp(-x(1,i+1)))./(exp(-x(1,i))-exp(-
x(1,i+1))); % iteration=0
end

for i=1:8 % p
    p(1,i)=exp(-x(1,i))-exp(-x(1,i+1));
end

for i=1:8 % sum
    sum(1,1)=sum(1,1)+p(1,i)*(y(1,i).^2);
end

sqnr(1,1)=1./(1-0.5*sum(1,1)); % sqnr

% j iteration
for j=2:51
    for i=2:8 % x
        x(j,i)=(y(j-1,i-1)+y(j-1,i))/2;
    end
    x(j,1)=0;
    x(j,9)=99999;
    for i=1:8 % y
        y(j,i)=((x(j,i)+1)*exp(-x(j,i))-(x(j,i+1)+1)*exp(-x(j,i+1)))./(exp(-x(j,i))-exp(-x(j,i+1))); % iteration=0
    end
    for i=1:8 % p
        p(j,i)=exp(-x(j,i))-exp(-x(j,i+1));
    end
    for i=1:8 % sum
        sum(j,1)=sum(j,1)+p(j,i)*(y(j,i).^2);
    end
    sqnr(j,1)=1./(1-0.5*sum(j,1)); % sqnr
end

n=0:50;
figure
plot(n,10*(log10(sqnr)))
title('SQNR for M=8')
xlabel('n of iterations')
ylabel('10log_1_0(SQNR)')
```

## <Part B_(d)>

```
clc
clear
close all;

x=zeros(101,513); % x(1,1)=x0, x0(1,2)=x1, ... , x0(8)=x7, x0(9)=x8=99999
y=zeros(101,512); % y(1,1)=y1, y0(1,2)=y2, ...
p=zeros(101,512); % p0(1)=p1, p0(2)=p2, ...
sum=zeros(101,1);
sqnr=zeros(101,1);

% iteration=0
for i=2:512 % x
    x(1,i)=(i-1)*2/512;
end

x(1,1)=0;
x(1,513)=99999;

for i=1:512 % y
y(1,i)=((x(1,i)+1)*exp(-x(1,i))-(x(1,i+1)+1)*exp(-x(1,i+1)))./(exp(-x(1,i))-exp(-x(1,i+1))); % iteration=0
end

for i=1:512 % p
    p(1,i)=exp(-x(1,i))-exp(-x(1,i+1));
end

for i=1:512 % sum
    sum(1,1)=sum(1,1)+p(1,i)*(y(1,i).^2);
end

sqnr(1,1)=1./(1-0.5*sum(1,1)); % sqnr

% j iteration
for j=2:101
    for i=2:512 % x
        x(j,i)=(y(j-1,i-1)+y(j-1,i))/2;
    end
    x(j,1)=0;
    x(j,513)=99999;

    for i=1:512 % y
        y(j,i)=((x(j,i)+1)*exp(-x(j,i))-(x(j,i+1)+1)*exp(-x(j,i+1)))./(exp(-x(j,i))-exp(-x(j,i+1))); % iteration=0
    end

    for i=1:512 % p
        p(j,i)=exp(-x(j,i))-exp(-x(j,i+1));
    end

    for i=1:512 % sum
        sum(j,1)=sum(j,1)+p(j,i)*(y(j,i).^2);
    end

    sqnr(j,1)=1./(1-0.5*sum(j,1)); % sqnr

end

n=0:100;
figure
plot(n,10*(log10(sqnr)))
title('SQNR for M=512')
xlabel('n of iterations')
ylabel('10log_1_0(SQNR)')
```

## <Part C_(a)>

```
clc
clear
close all;

% problem (d)

x3=0.5;
y3=0.25;
y4=0.75;

% problem (d)
fun_d1=@(x) 2.*((x-y3).^2).*(1/2);
d1=integral(fun_d1,0,x3);
fun_d2=@(x) 2.*((x-y4).^2).*(1/2);
d2=integral(fun_d2,x3,1);
d=d1+d2;

% problem (e)
sqnr=(1/3)/0.0208;
```

## <Part C_(b)>

```
clc
clear
close all;

% problem (c)

syms x3;
syms y3;
syms y4;

y3=(1-exp(-(x3.^2)/2))./(((pi/2)^(1/2)*erf(x3/(2^(1/2)))));
y4=(exp(-(x3.^2)/2))./-(((pi/2)^(1/2)*erf((x3/(2^(1/2)-1)))));

eqn=2*x3-y3-y4;
S=solve(eqn,x3);
```

# Q2

## <nbayes_learn.m>

```
function [params]=nbayes_learn(traindata,trainlabels)
nclasses=2;
[n,d]=size(traindata);
% Estimate class probabilities and conditional probabilities
for i=1:nclasses
    index=trainlabels==i;
    ni=sum(index);
    params(1).classprobs(i)=ni/n;    % estimate the probability of class i
    for j=1:d
        rows=find(index==1);
        Xjc=traindata(rows,j);
        for k=1:2
            njc=size(find(Xjc==k),1);
            % estimate prob(variable j = k | class i)
            params(j).cprobs(k,i)=(njc+0.5)/(ni+1);
        end
    end
end
% Estimate the probabilities p(x_j = value_k)
for j=1:d
    for k=1:2
        % estimate prob(variable x_j = k)

params(j).mprobs(k)=(params(j).cprobs(k,1).*params(1).classprobs(1))+(params(j).cprobs(k,2).*params(1).classprobs(2));
    end
end
```

## <nbayes_predict.m>

```
function [predictions]=nbayes_predict(params,testdata)
nclasses=2;
ntest=size(testdata,1);
d=size(testdata,2);
predictions=zeros(ntest,1);
% for each of the test data points
for m=1:ntest
    x=testdata(m,:);
    % for each class value calculate log[ p(x|c) p(c) ]
    P=zeros(1,nclasses);
    for classi=1:nclasses
        P(classi)=log(params(1).classprobs(classi));
        for varj=1:d
            P(classi)=P(classi)+log(params(varj).cprobs(x(1,varj),classi));
        end
    end
    % select the maximum value over all classes as the predicted class
    [mValue,mIndex]=max(P);
    % store the class prediction for each test data point
    predictions(m)=mIndex;
end
```

### <question 1>

```
clc;
clear;
features=load('spam_features.txt');
labels=load('spam_labels.txt');
% Train the Naive Bayes model
n_examples=1500;
N=size(labels,1);
F=features(1:n_examples,:);
L=labels(1:n_examples,1);
params=nbayes_learn(F,L);
% Make predicted labels
t_data=features(1501:N,:);
p_labels=nbayes_predict(params,t_data);
% Calculate the accuracy
count=0;
for i=1501:N
    if labels(i)==p_labels(i-1500)
        count=count+1;
    end
end
accuracy=(count/(N-1500))*100;
fprintf('Train the Naive Bayes model on the first %d examples.₩n',n_examples);
fprintf('Test data set: rows 1501 through %d₩n',N);
fprintf('%f%% of all class labels are predicted correctly.₩n',accuracy);
```

## <question 2>

```
clc;
clear;
features=load('spam_features.txt');
labels=load('spam_labels.txt');
% Train the Naive Bayes model
n_examples=500;
N=size(labels,1);
F=features(1:n_examples,:);
L=labels(1:n_examples,1);
params=nbayes_learn(F,L);
% Make predicted labels
t_data=features(1501:N,:);
p_labels=nbayes_predict(params,t_data);
% Calculate the accuracy
count=0;
for i=1501:N
    if labels(i)==p_labels(i-1500)
        count=count+1;
    end
end
accuracy=(count/(N-1500))*100;
fprintf('Train the Naive Bayes model on the first %d examples.₩n',n_examples);
fprintf('Test data set: rows 1501 through %d₩n',N);
fprintf('%f%% of all class labels are predicted correctly.₩n',accuracy);
```

## <question 3>

```
clc;
clear;
features=load('spam_features.txt');
labels=load('spam_labels.txt');
% Train the Naive Bayes model
n_examples=10;
N=size(labels,1);
F=features(1:n_examples,:);
L=labels(1:n_examples,1);
params=nbayes_learn(F,L);
% Make predicted labels
t_data=features(1501:N,:);
p_labels=nbayes_predict(params,t_data);
% Calculate the accuracy
count=0;
for i=1501:N
    if labels(i)==p_labels(i-1500)
        count=count+1;
    end
end
accuracy=(count/(N-1500))*100;
fprintf('Train the Naive Bayes model on the first %d examples.₩n',n_examples);
fprintf('Test data set: rows 1501 through %d₩n',N);
fprintf('%f%% of all class labels are predicted correctly.₩n',accuracy);
```

### <nbayes_MLpredict.m>

```
function [predictions]=nbayes_MLpredict(params,testdata)
nclasses=2;
ntest=size(testdata,1);
d=size(testdata,2);
predictions=zeros(ntest,1);
for m=1:ntest
    x=testdata(m,:);
    P=zeros(1,nclasses);
    for classi=1:nclasses
        P(classi)=0;
        for varj=1:d
            P(classi)=P(classi)+log(params(varj).cprobs(x(1,varj),classi));
        end
    end
    [mValue,mIndex]=max(P);
    predictions(m)=mIndex;
end
```

## <question 4-1>

```
clc;
clear;
features=load('spam_features.txt');
labels=load('spam_labels.txt');
% Train the Naive Bayes model
n_examples=1500;
N=size(labels,1);
F=features(1:n_examples,:);
L=labels(1:n_examples,1);
params=nbayes_learn(F,L);
% Make predicted labels
t_data=features(1501:N,:);
p_labels=nbayes_MLpredict(params,t_data);
% Calculate the accuracy
count=0;
for i=1501:N
    if labels(i)==p_labels(i-1500)
        count=count+1;
    end
end
accuracy=(count/(N-1500))*100;
fprintf('Train the Naive Bayes model on the first %d examples.₩n',n_examples);
fprintf('Test data set: rows 1501 through %d₩n',N);
fprintf('%f%% of all class labels are predicted correctly.₩n',accuracy);
```

## <question 4-2>

```
clc;
clear;
features=load('spam_features.txt');
labels=load('spam_labels.txt');
% Train the Naive Bayes model
n_examples=500;
N=size(labels,1);
F=features(1:n_examples,:);
L=labels(1:n_examples,1);
params=nbayes_learn(F,L);
% Make predicted labels
t_data=features(1501:N,:);
p_labels=nbayes_MLpredict(params,t_data);
% Calculate the accuracy
count=0;
for i=1501:N
    if labels(i)==p_labels(i-1500)
        count=count+1;
    end
end
accuracy=(count/(N-1500))*100;
fprintf('Train the Naive Bayes model on the first %d examples.₩n',n_examples);
fprintf('Test data set: rows 1501 through %d₩n',N);
fprintf('%f%% of all class labels are predicted correctly.₩n',accuracy);
```

## <question 4-3>

```
clc;
clear;
features=load('spam_features.txt');
labels=load('spam_labels.txt');
% Train the Naive Bayes model
n_examples=10;
N=size(labels,1);
F=features(1:n_examples,:);
L=labels(1:n_examples,1);
params=nbayes_learn(F,L);
% Make predicted labels
t_data=features(1501:N,:);
p_labels=nbayes_MLpredict(params,t_data);
% Calculate the accuracy
count=0;
for i=1501:N
    if labels(i)==p_labels(i-1500)
        count=count+1;
    end
end
accuracy=(count/(N-1500))*100;
fprintf('Train the Naive Bayes model on the first %d examples.₩n',n_examples);
fprintf('Test data set: rows 1501 through %d₩n',N);
fprintf('%f%% of all class labels are predicted correctly.₩n',accuracy);
```

### <nbayes_CLTlearn.m>

```
function [params]=nbayes_CLTlearn(traindata,trainlabels)
nclasses=2;
[n,d]=size(traindata);
for i=1:nclasses
    index=trainlabels==i;
    ni=sum(index);
    params(1).classprobs(i)=ni/n;
    for j=1:d
        rows=find(index==1);
        Xjc=traindata(rows,j);
        njc=sum(Xjc,1);
        % E[Xj|C]
        params(j).E(i)=njc/ni;
        % Var[Xj|C]
        njc_2=sum(Xjc.^2,1);
        params(j).V(i)=njc_2/ni-(params(j).E(i)).^2;
        params(j).S(i)=sqrt(params(j).V(i));
    end
end
for i=1:nclasses
    for j=1:d
        for k=1:2
            P=normcdf([(k-0.5-params(j).E(i))/(params(j).S(i)+0.00001)                    (k+0.5-
params(j).E(i))/(params(j).S(i)+0.00001)]);
            % P(Xj=k|C=ci)
            params(j).mcprobs(k,i)=P(2)-P(1);
        end
    end
end
for j=1:d
    for k=1:2
        % P(Xj=k)

params(j).mprobs(k)=(params(j).mcprobs(k,1).*params(1).classprobs(1))+(params(j).mcprobs(k,2).*par
ams(1).classprobs(2));
    end
end
```

### <nbayes_CLTpredict.m>

```
function [predictions]=nbayes_CLTpredict(params,testdata)
nclasses=2;
ntest=size(testdata,1);
d=size(testdata,2);
predictions=zeros(ntest,1);
for m=1:ntest
    x=testdata(m,:);
    P=zeros(1,nclasses);
    for classi=1:nclasses
        P(classi)=0;
        for varj=1:d
            P(classi)=P(classi)+log(params(varj).mcprobs(x(1,varj),classi));
        end
    end
    [mValue,mIndex]=max(P);
    predictions(m)=mIndex;
end
```

## <question 5-1>

```
clc;
clear;
features=load('spam_features.txt');
labels=load('spam_labels.txt');
n_examples=1500;
N=size(labels,1);
F=features(1:n_examples,:);
L=labels(1:n_examples,1);
params=nbayes_CLTlearn(F,L);
t_data=features(1501:N,:);
p_labels=nbayes_CLTpredict(params,t_data);
count=0;
for i=1501:N
    if labels(i)==p_labels(i-1500)
        count=count+1;
    end
end
accuracy=(count/(N-1500))*100;
fprintf('Train with the first %d examples.₩n',n_examples);
fprintf('Test data set: rows 1501 through %d₩n',N);
fprintf('%f%% of all class labels are predicted correctly.₩n',accuracy);
```

## <question 5-3>

```
clc;
clear;
features=load('spam_features.txt');
labels=load('spam_labels.txt');
n_examples=10;
N=size(labels,1);
F=features(1:n_examples,:);
L=labels(1:n_examples,1);
params=nbayes_CLTlearn(F,L);
t_data=features(1501:N,:);
p_labels=nbayes_CLTpredict(params,t_data);
count=0;
for i=1501:N
    if labels(i)==p_labels(i-1500)
        count=count+1;
    end
end
accuracy=(count/(N-1500))*100;
fprintf('Train with the first %d examples.₩n',n_examples);
fprintf('Test data set: rows 1501 through %d₩n',N);
fprintf('%f%% of all class labels are predicted correctly.₩n',accuracy);
```

# Q3

## <Part A_(b)>

```
v_hat = zeros(1, 6); % array to save estimated integral values
se = zeros(1, 6); % array to save estimated standard errors
ci95 = strings(1, 6); % array to save 95% CI which z = 1.96
ci99 = strings(1, 6); % array to save 99% CI which z = 2.58
a = [2, 3, 4, 5, 6, 7]; % a = 2, 3, 4, 5, 6, 7

for a_i = 1:6 % index of a
    n = 10^a(a_i); % number of samples
    aj = [0, 0.5]; % lower and upper bounds of integration for each j = 1, 2, ... k
    y = zeros(1, n); % initialize y as a zero vector of length n

    for i=1:n
        x = rand*(aj(2)-aj(1)) + aj(1); % x_i = Uniform(0, 0.5)
        y(i) = h(x) * (aj(2)-aj(1)); % y_i = h(x_i) * Pi(high-low)
    end

    v_hat(a_i) = sum(y, 'all')/n; % estimated integral value
    se(a_i) = std(y) / sqrt(n); % estimated standard error
    ci95(a_i) = sprintf('%g ±%g', [v_hat(a_i), se(a_i)*1.96]); % 95% CI which z = 1.96
    ci99(a_i) = sprintf('%g ±%g', [v_hat(a_i), se(a_i)*2.58]); % 99% CI which z = 2.58
end

% table to show result
table(a.', v_hat.',se.',ci95.', ci99.', 'VariableNames', ["a", "estimated integral value", "estimated standard error", "CI 95%", "CI 99%"])

function f = h(x) % h(x) of (b)
    f = x*(1-x^2)^(3/2);
end
```

## <Part A_(c)>

```
v_hat = zeros(1, 6); % array to save estimated integral values
se = zeros(1, 6); % array to save estimated standard errors
ci95 = strings(1, 6); % array to save 95% CI which z = 1.96
ci99 = strings(1, 6); % array to save 99% CI which z = 2.58
a = [2, 3, 4, 5, 6, 7]; % a = 2, 3, 4, 5, 6, 7

for a_i = 1:6 % index of a
    n = 10^a(a_i); % number of samples
    aj = [0, 0.5]; % lower and upper bounds of integration for each j = 1, 2, ... k
    y = zeros(1, n); % initialize y as a zero vector of length n

    for i=1:n
        x = normrnd(0.25, 0.5); % x = Gaussian(0.25, 0.5)
        y(i) = h(x) * (aj(2)-aj(1)); % y_i = h(x_i) * Pi(high-low)
    end

    v_hat(a_i) = sum(y, 'all')/n; % estimated integral value
    se(a_i) = std(y) / sqrt(n); % estimated standard error
    ci95(a_i) = sprintf('%g ±%g', [v_hat(a_i), se(a_i)*1.96]); % 95% CI which z = 1.96
    ci99(a_i) = sprintf('%g ±%g', [v_hat(a_i), se(a_i)*2.58]); % 99% CI which z = 2.58
end

% table to show result
table(a.', v_hat.',se.',ci95.', ci99.', 'VariableNames', ["a", "estimated integral value", "estimated standard error", "CI 95%", "CI 99%"])

function f = h(x) % h(x) of (b)
    f = x*(1-x^2)^(3/2);
end
```

### <Part A_(d)>

```
v_hat = zeros(1, 4); % array to save estimated integral values
se = zeros(1, 4); % array to save estimated standard errors
ci95 = strings(1, 4); % array to save 95% CI which z = 1.96
ci99 = strings(1, 4); % array to save 99% CI which z = 2.58
a = [2, 4, 6, 7]; % a = 2, 4, 6, 7

for a_i = 1:4 % index of a
    n = 10^a(a_i); % number of samples
    aj = [0, 1]; % lower and upper bounds of integration for each j = 1, 2, ... k
    y = zeros(1, n);  % initialize y as a zero vector of length n

    for i=1:n
        x = rand*(aj(2)-aj(1)) + aj(1); % x = Uniform(0, 1)
        y(i) = h(x) * (aj(2)-aj(1)); % y_i = h(x_i) * Pi(high-low)
    end

    v_hat(a_i) = sum(y, 'all')/n; % estimated integral value
    se(a_i) = std(y) / sqrt(n); % estimated standard error
    ci95(a_i) = sprintf('%g ±%g', [v_hat(a_i), se(a_i)*1.96]); % 95% CI which z = 1.96
    ci99(a_i) = sprintf('%g ±%g', [v_hat(a_i), se(a_i)*2.58]); % 99% CI which z = 2.58
end

% table to show result
table(a.', v_hat.',se.',ci95.', ci99.', 'VariableNames', ["a", "estimated integral value", "estimated standard error", "CI 95%", "CI 99%"])

function f = h(x) % h(x) of (d)
    f = exp(sin(x^3))/(3*(1+5*x^8));
end
```

### <Part A_(e)>

```
v_hat = zeros(1, 4); % array to save estimated integral values
se = zeros(1, 4); % array to save estimated standard errors
ci95 = strings(1, 4); % array to save 95% CI which z = 1.96
ci99 = strings(1, 4); % array to save 99% CI which z = 2.58
a = [2, 4, 6, 7]; % a = 2, 4, 6, 7

for a_i = 1:4 % index of a
    n = 10^a(a_i); % number of samples
    aj = [0, 1, 0, 1]; % lower and upper bounds of integration for each j = 1, 2, ... k
    y = zeros(1, n);  % initialize y as a zero vector of length n

    for i=1:n
        x_1 = rand*(aj(2)-aj(1)) + aj(1); % x_i,1 = Uniform(0, 1)
        x_2 = rand*(aj(4)-aj(3)) + aj(3); % x_i,2 = Uniform(0, 1)
        y(i) = h(x_1, x_2) * (aj(2)-aj(1)) * (aj(4)-aj(3)); % y_i = h(x_i,1, x_i,2) * Pi(high-low)
    end

    v_hat(a_i) = sum(y, 'all')/n; % estimated integral value
    se(a_i) = std(y) / sqrt(n); % estimated standard error
    ci95(a_i) = sprintf('%g ±%g', [v_hat(a_i), se(a_i)*1.96]); % 95% CI which z = 1.96
    ci99(a_i) = sprintf('%g ±%g', [v_hat(a_i), se(a_i)*2.58]); % 99% CI which z = 2.58
end

% table to show result
table(a.', v_hat.',se.',ci95.', ci99.', 'VariableNames', ["a", "estimated integral value", "estimated standard error", "CI 95%", "CI 99%"])

function f = h(x, y) % h(x, y) of (e)
    f = exp(sin(x^3+y^3))/(3*(1+5*(x^8+y^8)));
end
```

## <Part B_(b)>

```
n = 10000; % number of samples
r = [];  % relative frequency
i = 0;

for a = 0:0.2:1
    i = i + 1;
    s = datasample([0, 1], n, 'Weights', [1-payoff(a), payoff(a)]);
    r(i) = sum(s, 'all')/n;
end

bar(0:0.2:1, r)

function k = payoff(a) % payoff function calculated in (a)
    k = (a+1)/3;
end
```