# Free energy after convergence ults

# Assignment 5

## Probabilistic and Unsupervised Learning

## Yuan Zhang

## SN: 17044633

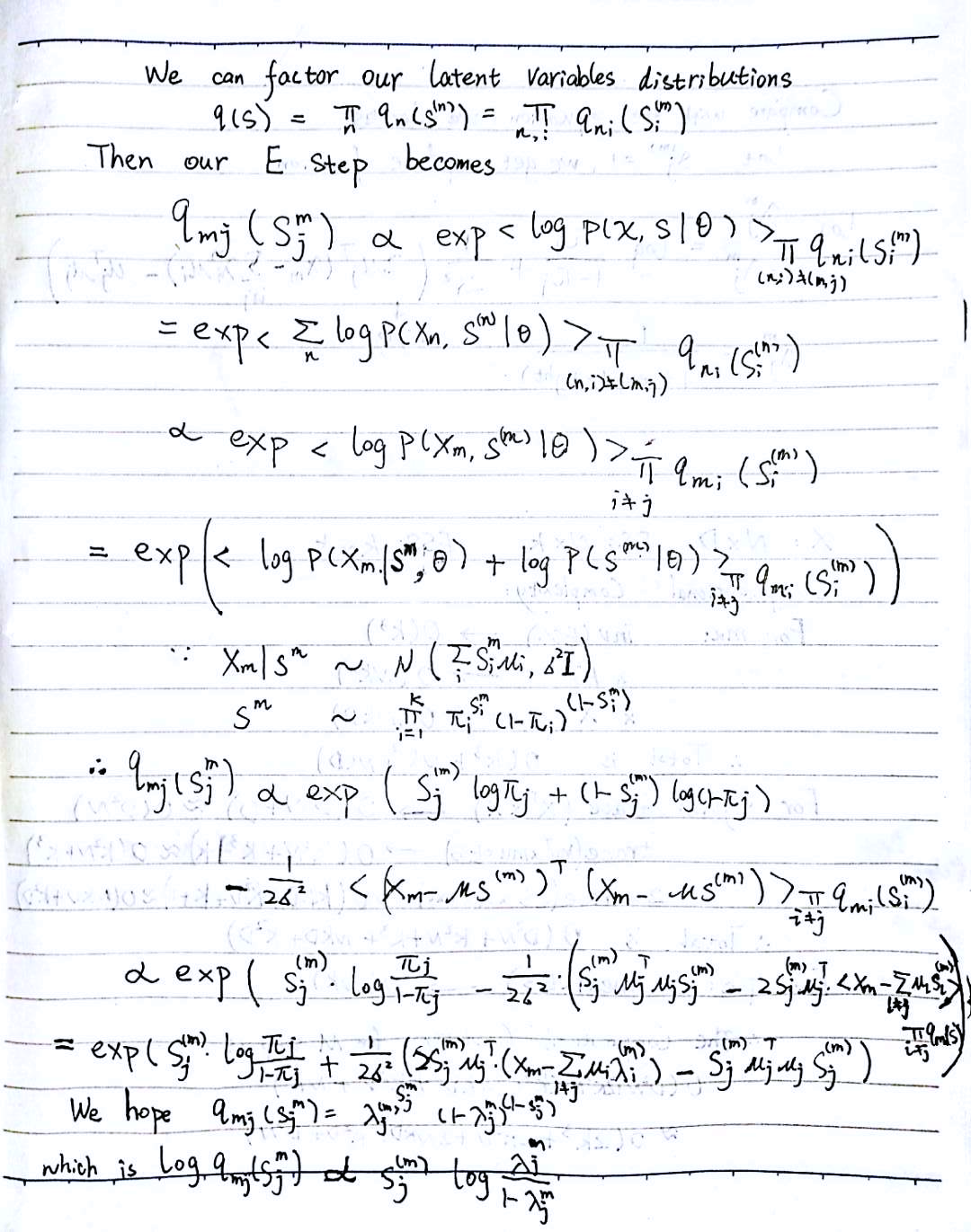
## [ucaby02@ucl.ac.uk](mailto:ucaby02@ucl.ac.uk)

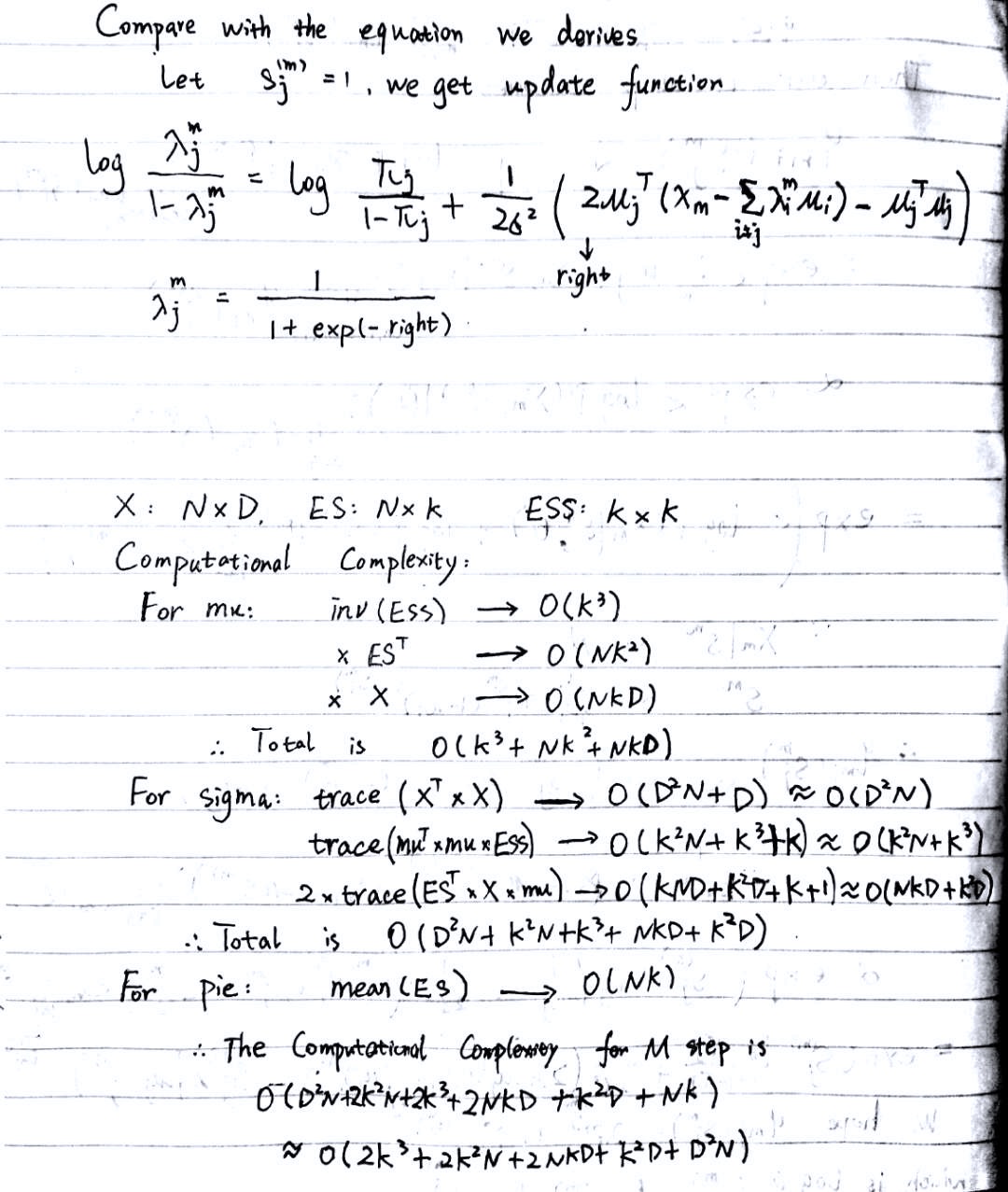
## Dec 14th, 2017

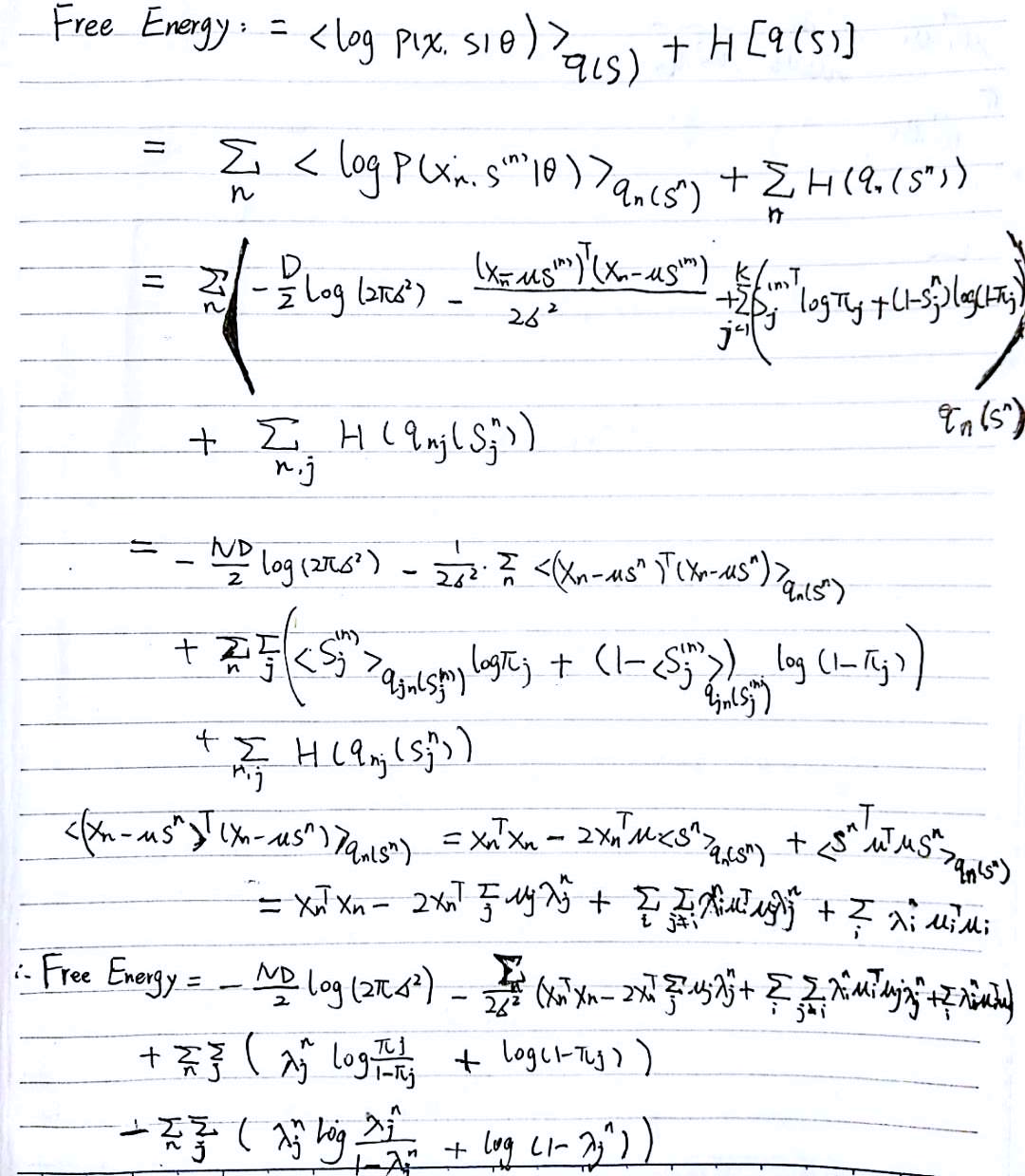
## Mean-field Learning

#### E step

*Derivation:*



**



*Code Parts:*

function [lambda,F] = MeanField(X,mu,sigma,pie,lambda0,maxsteps)

% Variational Estep for our models.

%

% Inputs:

% mu: D\*K matrix of means

% pie: 1\*K vector of priors on s

% lambda0: initial values for lambda

% maxsteps: maximum number of steps of the fixed point equations.

%

% Outputs:

% lambda: N\*K distributions on latent variables

% F: lower bound on the likelihood

% X: N\*D data matrix

[N,D] = size(X);

[~,K] = size(lambda0);

thres = 10^-9; % convergence criterion to stop iteration

self\_m = diag(mu'\*mu)'; % self product for mu

self\_X = sum(sum(X.\*X)); % self product for X

% Computer initial Free Energy without updates lambda

index = find(lambda0>0 & lambda0<1); % to avoid numerical problems in entropy computations

F = sum(log(pie./(1-pie))\*lambda0') ...

+ N\*sum(log(1-pie))...

-N\*D/2\*log(2\*pi\*sigma\*sigma)...

-(self\_X-2\*sum(sum(X.\*(lambda0\*mu')))...

+sum(sum((lambda0\*mu').^2))-sum(sum(lambda0.^2.\*repmat(self\_m,N,1)))...

+sum(sum(lambda0.\*repmat(self\_m,N,1))))...

/(2\*sigma\*sigma)...

-sum(sum(lambda0(index).\*log(lambda0(index))+(1-lambda0(index)).\*log(1-lambda0(index))));

% fprintf('iteration 0 : %f\r',F);

lambda = lambda0;

for j= 1:maxsteps

%% Update lambda

current\_lambda = lambda;

for k = 1:K

tmp = log(repmat(pie(k)/(1-pie(k)),N,1))...

+((2\*X-2\*current\_lambda\*mu')\*mu(:,k)+2\*current\_lambda(:,k).\*repmat(self\_m(k),N,1)-repmat(self\_m(k),N,1))/(2\*sigma\*sigma);

current\_lambda(:,k) = 1./(1+exp(-tmp));

end

%% Compute Free Energy

index = find( current\_lambda>0 & current\_lambda<1);

current\_F = sum(log(pie./(1-pie))\*current\_lambda') ...

+ N\*sum(log(1-pie))...

-N\*D/2\*log(2\*pi\*sigma\*sigma)...

-(self\_X-2\*sum(sum(X.\*(current\_lambda\*mu')))...

+sum(sum((current\_lambda\*mu').^2))-sum(sum(current\_lambda.^2.\*repmat(self\_m,N,1)))...

+sum(sum(current\_lambda.\*repmat(self\_m,N,1))))...

/(2\*sigma\*sigma)...

-sum(sum(current\_lambda(index).\*log(current\_lambda(index))))-sum(sum((1-current\_lambda(index)).\*log(1-current\_lambda(index))));

% fprintf('iteration %d : %f\r',j,current\_F);

%% check if F increases after update

if current\_F-F < thres;

break;

end

F = current\_F;

lambda = current\_lambda;

end

end

#### M step

For the free energy, we compute a expectation of a likelihood (output X given latent factors S is Gaussian distribution with mean value mu\*S and noise sigma). This likelihood is similar to the linear regression, output X given input S is Gaussian distribution with mean value mu\*S and noise sigma. So the parameters estimation has a close form.



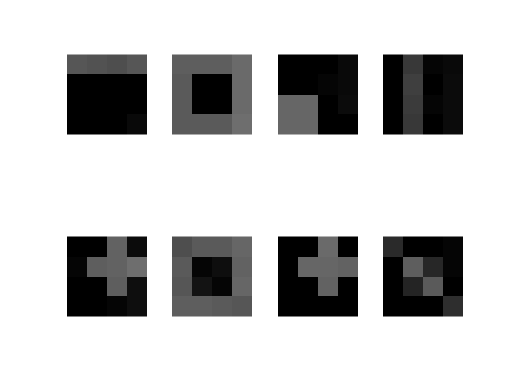
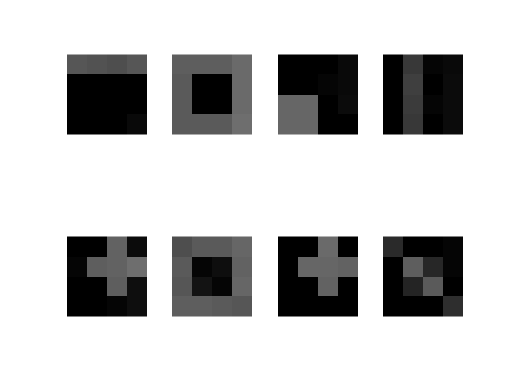
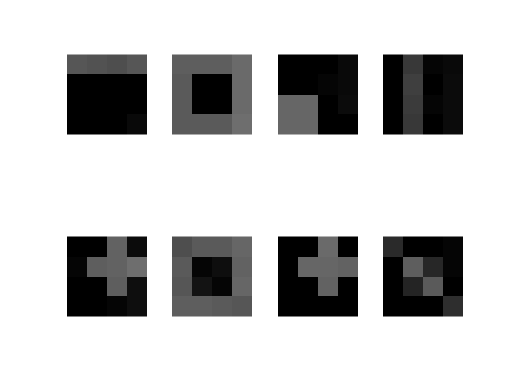
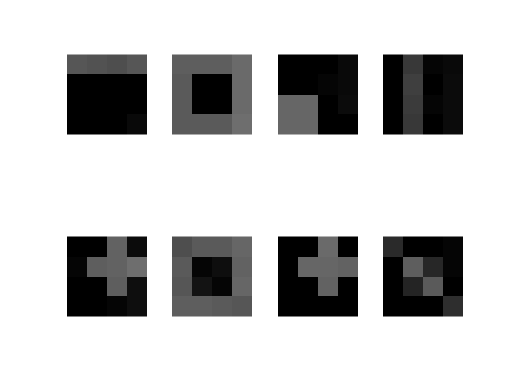
For our model, we need to compute the expectation of formal equation. For sigma, we also need to calculate the expectation of the similar form of linear regression.

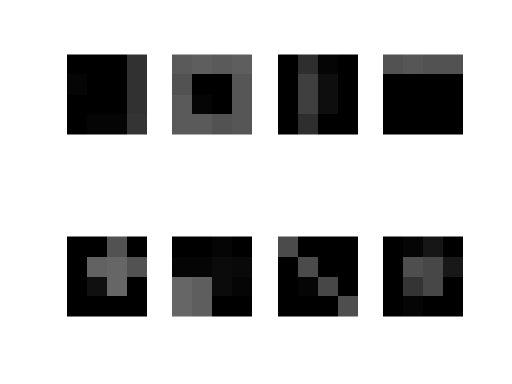
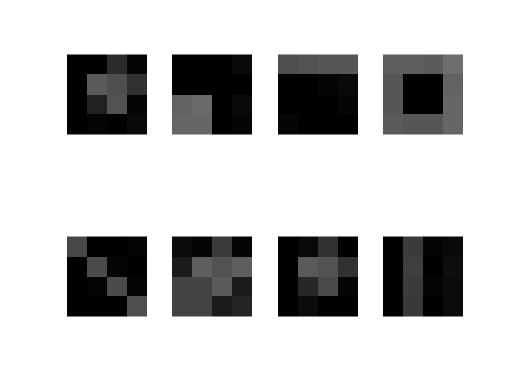
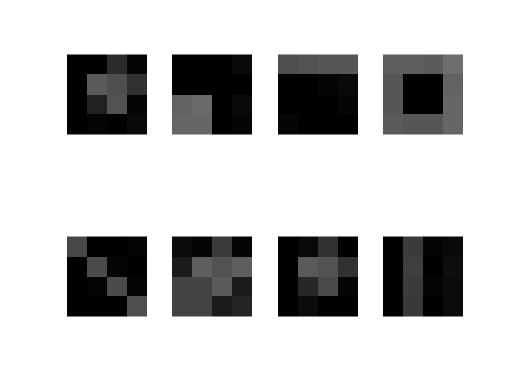
#### Computational complexity



#### Guess Features

We can see that there are 8 features and they are independent.





FA doesn’t fit our data. Latent variable here represents if certain feature shows. So it should be binary data instead of continuous Gaussian distributed data.

MOG also doesn’t fit. Latent variable here should be a binary vector instead of a discrete scalar. Output here is truly generated from the combination of different features. But MOG output belongs to one single point with noise.

ICA fits our data. Imagine different features as different sources. Latent variables S combine these features for a single output. Feature i relates to the latent variable si, which follows Bernoulli distribution and is independent.

**(e) Learn Bin Factors**

*Code Parts:*

function [mu, sigma, pie] = LearnBinFactors(X,K,iterations)

[N,D] = size(X);

[lambda0,mu,sigma,pie] = initial\_para(X,N,D,K);

F\_last = -inf;

thres = 10^-9; % convergence criterion to stop iteration

for j=1:iterations

% Implement E step

[lambda,F] = MeanField(X,mu,sigma,pie,lambda0,30);

% Implement M step

ES = lambda;

ESS = zeros(N,K,K);

for n=1:N

tmp = lambda(n,:)'\*lambda(n,:);

tmp(logical(eye(K)))=lambda(n,:);

ESS(n,:,:) = tmp;

end

[mu, sigma, pie] = MStep(X,ES,ESS);

[lambda,F] = MeanField(X,mu,sigma,pie,lambda,0);

fprintf('iteration %d : %f\r',j,F);

if F-F\_last < thres;

break;

end

F\_last = F;

lambda0 = lambda;

end

end

function [lambda0,mu,sigma,pie] = initial\_para(X,N,D,K)

lambda0 = rand(N,K);

% Use M step to initialize parameters

ES = lambda0;

ESS = zeros(N,K,K);

for n=1:N

tmp = lambda0(n,:)'\*lambda0(n,:);

tmp(logical(eye(K)))=lambda0(n,:);

ESS(n,:,:) = tmp;

end

[mu, sigma, pie] = MStep(X,ES,ESS);

end

**(f) Run algorithm**

*Code Parts:*

K = 8;

iterations = 100;

[mu, sigma, pie] = LearnBinFactors(Y,K,iterations);

% plot mu

mu = mu';

set(gcf,'Color',[0.2 0.4 0.6]); % Background color

colormap gray;

for k=1:K

subplot(2,4,k);

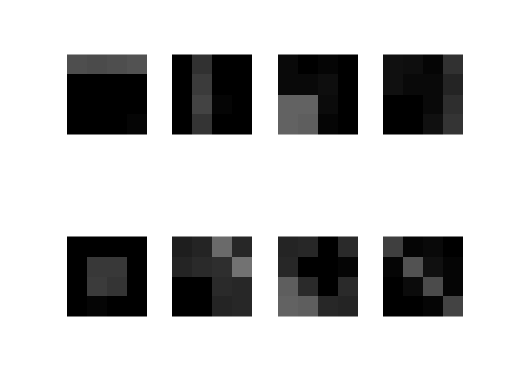
imagesc(reshape(mu(k,:),4,4),[0 2]);

axis off;

axis equal;

end

*Results:* Plot the features we learn below.



*Improve Algorithm:*

The result after convergence relies a lot on initial values since our initial values are random. We can run our algorithm for several times and choose some mutual features among them.

*Improve the features:*

To enhance features we learn, we can remap our features to binary data. A simple way is to set threshold=0.5 and let pixel over threshold equal to 1, pixel below 0.5 equal to 0. The codes shown below.

function mu\_eh = enhance(mu)

mu\_eh = zeros(size(mu));

mu\_eh(find(mu>0.3))=1;

end

Then we can plot our mu after enhancement.

**

We can see that 8 features are quite clear.

*What to set K:*

From d, we have guessed 8 features, so K should at least be 8. We can set K over 8 incase there are features we didn’t find. But K should be too large, otherwise there will be overfit.

*How to initialize:*

From our initial\_para.m function we can see that we initialize lambda, and use M-step to calculate other parameters. This gives us less randomness. We use uniform random to initialize lambda.

**(g) Convergence and sigma**

We test on sigma = 1 and 2 and 4 separately. Only use Mean Field updates lambda and calculate Free Energy. Take 100 iterations, results as below.





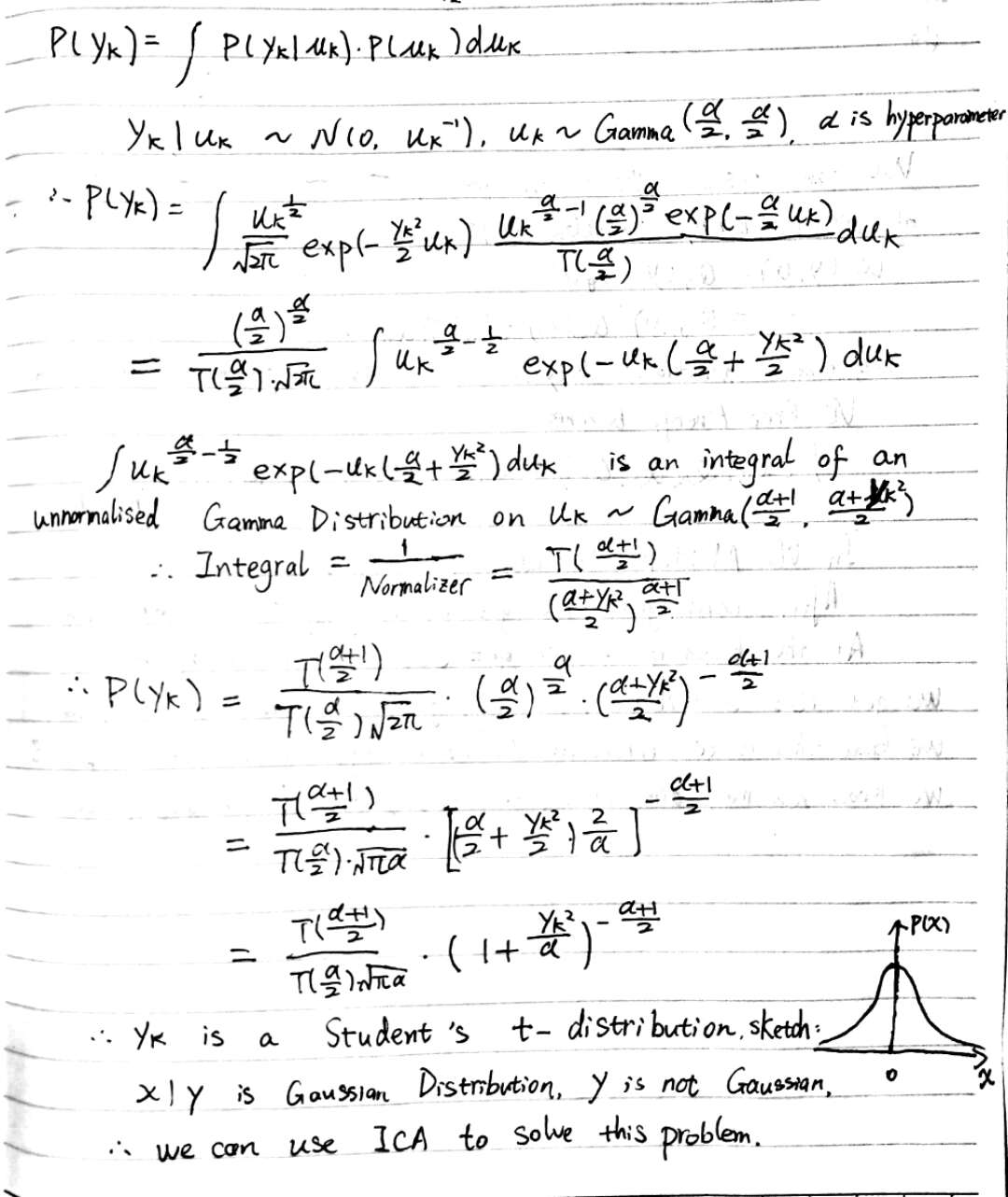
We can see that smaller sigma has larger Free energy after convergence, but it takes more time to converge for smaller sigma.

We can have an intuitive idea that if our noise (sigma) is small, then our training model or latent distribution must be very close to the “truth” then our free energy can converge. On the other hand, if our noise is large, a rough model can make free energy converge. So larger sigma converges faster. But this model is not precise, so free energy converges at a low level. So noise is a tradeoff between convergence speed and precision.

#### Macintosh HD:Users:yuanzhang:Desktop:WechatIMG182.jpeg(h) Variational Bayesian method for selecting K

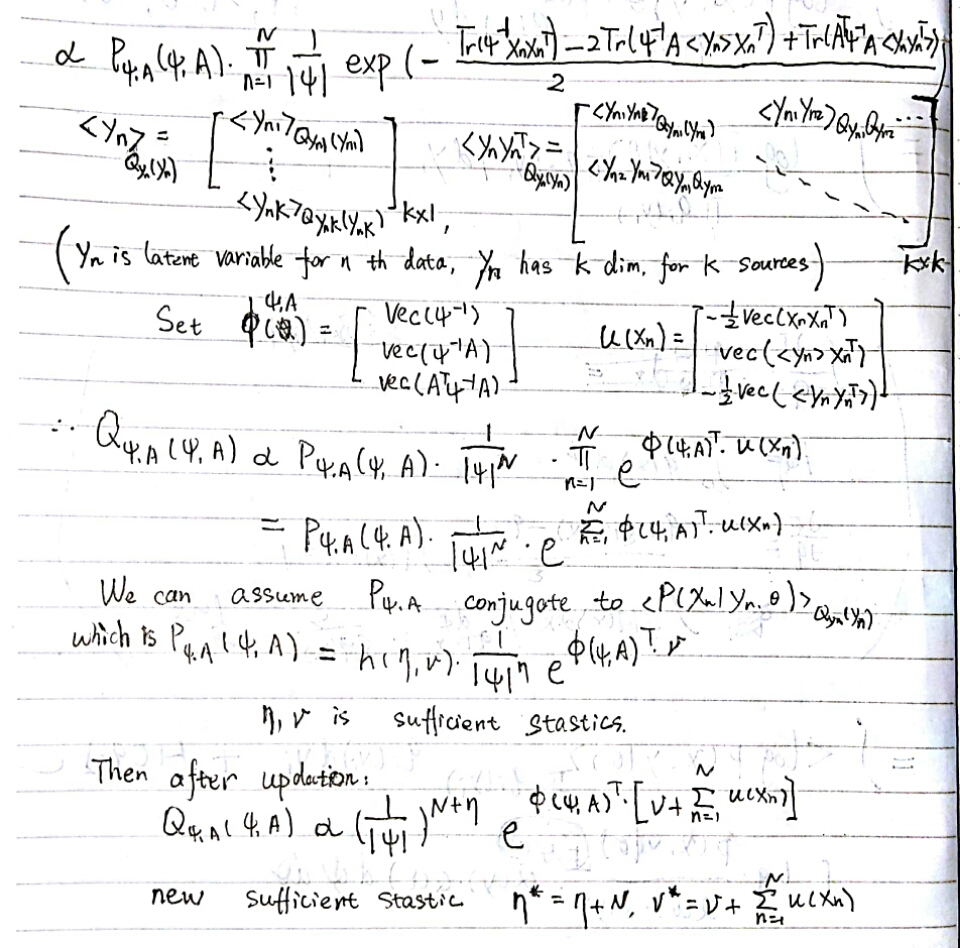
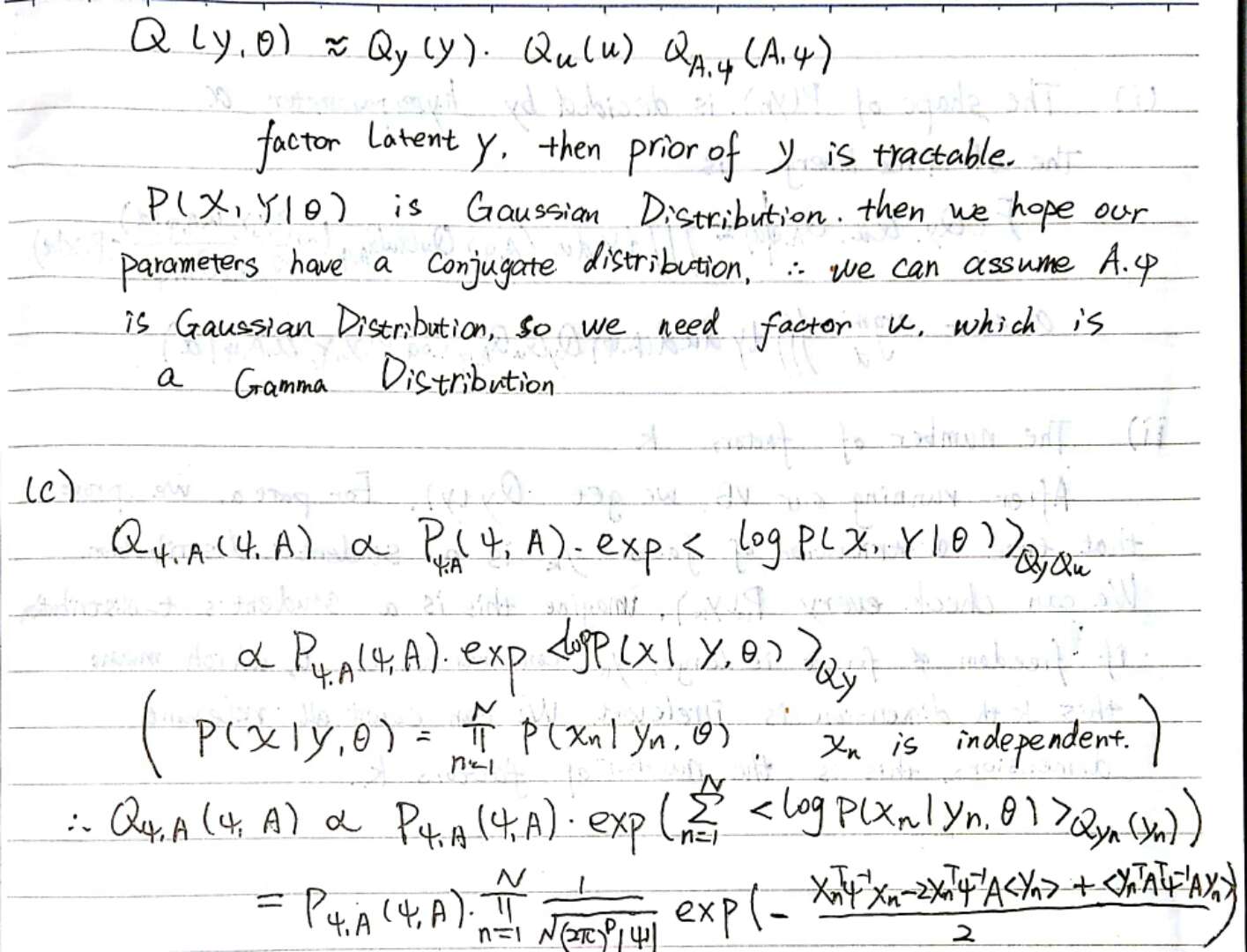
## Noisy ICA

#### (a) Derive the marginal distribution



#### (b) VB Factorization

#### Macintosh HD:Users:yuanzhang:Desktop:WechatIMG184.jpeg(c) Update Distribution



#### Macintosh HD:Users:yuanzhang:Desktop:WechatIMG186.jpeg(d) Hyperparameter optimization

